MPC5534RM
Rev. 2.1, 04/2012

MPC5534 Microcontroller Reference Manual

This MPC5534 Reference Manual set consists of the following files:

- MPC5534 Reference Manual Addendum, Rev 3
- MPC5534 Microcontroller Reference Manual, Rev 2

Freescale Semiconductor
Reference Manual Addendum

MPC5534RMAD
Rev. 3, 04/2012

MPC5534 Reference Manual Addendum

This errata document describes corrections to the *MPC5534 Microcontroller Reference Manual*, order number MPC5534RM. For convenience, the addenda items are grouped by revision. Please check our website at http://www.freescale.com/powerarchitecture for the latest updates.

The current version available of the *MPC5534 Microcontroller Reference Manual* is Revision 2.0.

Table of Contents

1 Addendum for Revision 2.0

Table 1. MPC5534RM Rev 2.0 addendum

<table>
<tr><th>Location</th><th>Description</th></tr>
<tr><td>Table 9-23, “DMA Request Summary for eDMA”/Page 9-39</td><td>Change one row in the table to correct information about eSCI COMBTX DMA request. Only the Transmit Data Register Empty and LIN Transmit Data Ready flags drive the DMA request. The Transmit Complete flag is not used.
<table>
<tr><th>DMA Request</th><th>Channel</th><th>Source</th><th>Description</th></tr>
<tr><td>eSCIA_COMBTX</td><td>18</td><td>ESCIA.SR[TDRE] || ESCIA.SR[TC] || ESCIA.SR[TXRDY]</td><td>eSCIA combined DMA request of the Transmit Data Register Empty and LIN Transmit Data Ready DMA requests</td></tr>
</table></td></tr>
<tr><td>Figure 5-2, “Master Privilege Control Registers”/Page 5-3</td><td>Change read status for bits 16–31 from zero to reserved.</td></tr>
<tr><td>Table A-2, “MPC5534 Detailed Register Map”/Page A-2</td><td>Correct name of peripheral bridge A control register by adding underscore (PBRIDGEA_x becomes PBRIDGE_A_x).
<table>
<tr><th>Register Description</th><th>Register Name</th><th>Used Size</th><th>Address</th></tr>
<tr><td>Peripheral bridge A master privilege control register</td><td>PBRIDGE_A_MPCR</td><td>32-bit</td><td>Base + 0x0000</td></tr>
</table></td></tr>
<tr><td>Table A-2, “MPC5534 Detailed Register Map”/Pages A-27</td><td>Correct names of peripheral bridge B control registers by adding underscore (PBRIDGEB_x becomes PBRIDGE_B_x).
<table>
<tr><th>Register Description</th><th>Register Name</th><th>Used Size</th><th>Address</th></tr>
<tr><td>Peripheral bridge B master privilege control register</td><td>PBRIDGE_B_MPCR</td><td>32-bit</td><td>Base + 0x0000</td></tr>
</table></td></tr>
<tr><td>Chapter 1, “Overview”/ Page 1-1</td><td>Delete the word “dual” from eTPU in the sentence “The complex I/O timer functions of MPC5534 are performed by a dual Enhanced Time Processor Unit engine (eTPU)”.</td></tr>
<tr><td>Section 6.4.1.9: “IRQ Rising-Edge Event Enable Register (SIU_IREER)”/Page 6-20</td><td>Correct offset address of SIU_IERRR by changing it from “Address: Base + 0x0002” to “Address: Base + 0x0028.”</td></tr>
<tr><td>Table 6-12, “SIU_DIRER Field Descriptions”/Page 6-18</td><td>Correct the table heading by replacing SIU_DIRER with SIU_DIRSR.</td></tr>
</table>

Table 1. MPC5534RM Rev 2.0 addendum (continued)

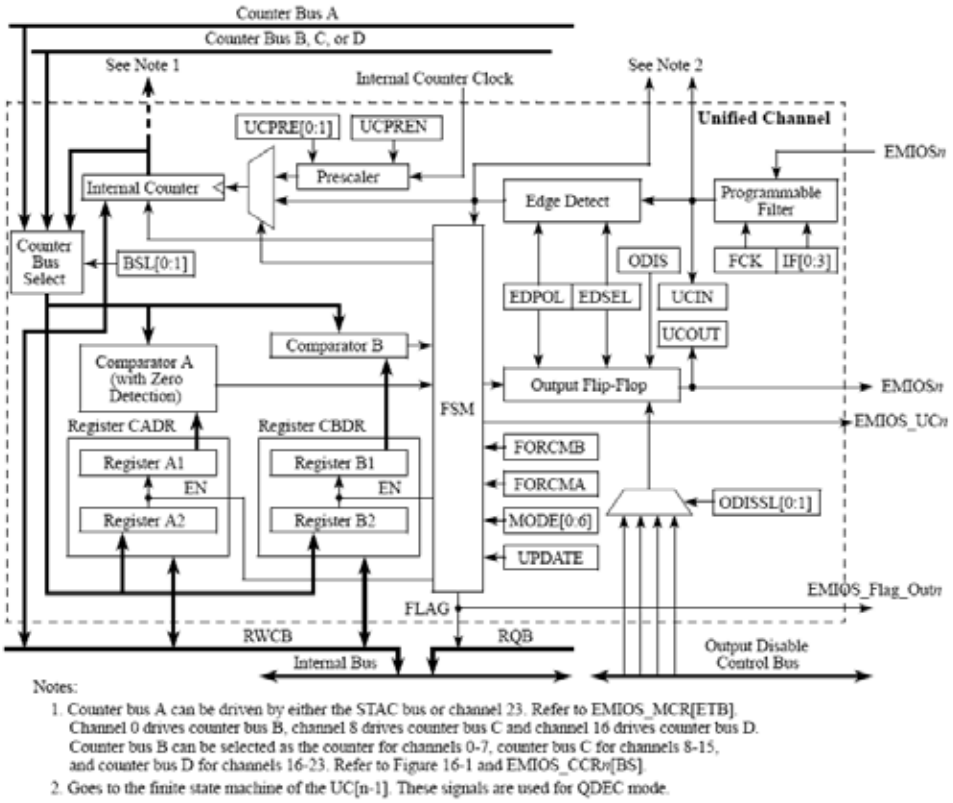
Location	Description
Figure 16-12," Unified Channel Block Diagram"/Page 16-25	Reverse the arrow between the "Programmable Filter" and "Edge Detect". Align the junction point on the signal path from "Edge Detect" in the direction of "Internal Counter".  **Figure 17-12. Unified Channel Block Diagram**
Section 9.3.1: eDMA Microarchitecture/ Page 9-29	In the Memory controller sub-bullet, delete the line "The hooks to a BIST controller for the local TCD memory are included in this module".
Section 9.2.2.13: eDMA Interrupt Request Register (EDMA_IRQRL)/ Page 9-17	In the third paragraph, remove the last line "without the need to perform a read-modify-write sequence to the EDMA_IRQRL".
Section 10.5.5.1: Elevating Priority/ Page 10-31	• From: After they release the resource, they must lower the PRI value in INTC_CPR to prevent further priority inversion • To: After they release the resource, the PRI value in INTC_CPR can be lowered. • From: Using the PCP instead of disabling processor recognition of all interrupts reduces the priority inversion time when accessing a shared resource. • To: Using the PCP instead of disabling processor recognition of all interrupts eliminates the time when accessing a shared resource that all higher priority interrupts are blocked."

Table 1. MPC5534RM Rev 2.0 addendum (continued)

Location	Description
Section 10.5.6: Selecting Priorities According to Request Rates and Deadlines/ Page 10-32	• From: Reducing the number of priorities does cause some priority inversion which reduces the processor's ability to meet its deadlines. It also allows easier management of ISRs with similar deadlines that share a resource. They can be placed at the same priority without any further priority inversion, and they do not need to use the PCP to access the shared resource” • To: Reducing the number of priorities does reduce the processor's ability to meet its deadlines. However, it also allows easier management of ISRs with similar deadlines that share a resource. They do not need to use the PCP to access the shared resource.
Section 10.5.7.1, “Scheduling a Lower Priority Portion of an ISR/ Page 10-33	• From: Therefore, executing this later portion which does not need to be executed at this higher priority can block the execution of ISRs which do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This priority inversion reduces the processor's ability to meet its deadlines. • To: Therefore, executing this later portion which does not need to be executed at this higher priority can prevent the execution of ISRs which do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This preemptive scheduling inefficiency reduces the processor's ability to meet its deadlines. • From: This software settable interrupt request, which usually will have a lower PRIn value in the INTC_PSRn, therefore will not cause priority inversion. • To: This software settable interrupt request, which usually will have a lower PRIn value in the INTC_PSRn, therefore will not cause preemptive scheduling inefficiencies.
Section 10.5.8, “Lowering Priority Within an ISR/ Page 10-33	• From: the only way (besides scheduling a task through an RTOS) to prevent priority inversion with an ISR whose work spans multiple priorities” • To: a way (besides scheduling a task through an RTOS) to prevent preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities • From: Therefore, the INTC does not support lowering the current priority within an ISR as a way to avoid priority inversion. • To: Therefore, through its use of the LIFO the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.
Table 6-29, “SIU_DISR Field Descriptions”/ Page 6-71	• Bit 14-15–TRIGSELB bit: Correct the input select description as follows: 00: Replace the term “Invalid value” with “No Trigger” 01: Replace the term “Invalid value” with “No Trigger” • Bit 22-23–TRIGSELC bit: Correct the input select description as follows: 00: Replace the term “Invalid value” with “No Trigger” 01: Replace the term “Invalid value” with “No Trigger” • Bit 30-31–TRIGSELD bit: Correct the input select description as follows: 00: Replace the term “Invalid value” with “No Trigger” 01: Replace the term “Invalid value” with “No Trigger”

Table 1. MPC5534RM Rev 2.0 addendum (continued)

<table>
<tr><th>Location</th><th>Description</th></tr>
<tr><td>Table 7-2, “XBAR Register Memory Map”/ Page 7-3</td><td>Add the following rows in the memory map table to include XBAR_MGPCRn registers.
<table>
<tr><th>Address</th><th>Register Name</th><th>Register Description</th><th>Bits</th></tr>
<tr><td>Base + 0x0800</td><td>XBAR_MGPCR0</td><td>Master General-purpose control register for master port 0</td><td>32</td></tr>
<tr><td>Base + 0x0804–0x08FF</td><td>—</td><td>Reserved</td><td>—</td></tr>
<tr><td>Base + 0x0900</td><td>XBAR_MGPCR1</td><td>Master General-purpose control register for master port 1</td><td>32</td></tr>
<tr><td>Base + 0x0904–0x09FF</td><td>—</td><td>Reserved</td><td>—</td></tr>
<tr><td>Base + 0x0A00</td><td>XBAR_MGPCR2</td><td>Master General-purpose control register for master port 2</td><td>32</td></tr>
<tr><td>Base + 0x0A04–0x0AFF</td><td>—</td><td>Reserved</td><td>—</td></tr>
<tr><td>Base + 0x0C00</td><td>XBAR_MGPCR4</td><td>Master General-purpose control register for master port 4</td><td>32</td></tr>
<tr><td>Base0x0B04–0x0003_FFFF</td><td>—</td><td>Reserved</td><td>—</td></tr>
</table></td></tr>
</table>

Table 1. MPC5534RM Rev 2.0 addendum (continued)

Location	Description
Section 7.2.1, “Register Descriptions”	Add the new section to include the XBAR_MGPCRn register description. **Master General Purpose Control Registers (XBAR_MGPCR*n*)** The Master General Purpose Control Register (XBAR_MGPCR) controls the arbitration policy during undefined length burst accesses. The AULB (Arbitrate on Undefined Length Bursts) field determines whether or not arbitration occurs for the slave port the master owns when the master is performing undefined length burst accesses. The MGPCR can only be accessed in supervisor mode with 32-bit access. Address : Base + 0x0800 (XBAR_MGPCR0) Base + 0x0900 (XBAR_MGPCR1) Base + 0x0A00 (XBAR_MGPCR2) Base + 0x0C00 (XBAR_MGPCR4) Access: R/W 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 R 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 W Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 R 0 0 0 0 0 0 0 0 0 0 0 0 0 AULB W Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **Master General-Purpose Control Registers (XBAR_MGPCR*n*)**
Section 10.4.1, “Interrupt Request Sources”/ Page 10-22	Correct the content of the note. • From: The INTC has no spurious vector support. If an asserted peripheral or software settable interrupt request: - Has a PRIn value (INTC_PSR0–INTC_PSR203) higher than the PRI value in INTC_CPR; and - Negates before the processor for that interrupt request acknowledges IRQ The IRQ to the processor can assert or remain asserted for that peripheral or software configurable interrupt request. In this case, the interrupt vector for the peripheral or software configurable IRQ remains, and the PRI value in the INTC_CPR is updated to the PRIn value in INTC_PSRn • To: The INTC does not have spurious vector support. The peripheral or software settable interrupt request asserts when the PRIn value in the interrupt priority select register (INTC_PSRn) is greater than the PRIn value in the interrupt current priority register (INTC_CPR). If an asserted peripheral or software settable interrupt request negates before the processor acknowledges its request, the interrupt request can reassert and remain asserted. If this occurs, the processor uses the INTC_PSRn value to locate the IRQ vector, and updates the PRIn value in the INTC_CPR with the PRIn value in INTC_PSRn.

Table 1. MPC5534RM Rev 2.0 addendum (continued)

Location	Description
Section 10.3.1.3, “INTC Interrupt Acknowledge Register (INTC_IACKR)”/Page 10-10	Remove the first paragraph from the “Note”: “The INTC_IACKR must not be read speculatively while in software vector mode. Therefore, for future compatibility, the TLB entry covering the INTC_IACKR must be configured to be guarded.”
Table 10-3. INTC Memory Map/Page 10-8	Add the following note at the end of this table: **Note:** To ensure compatibility with all PowerPC processors, the TLB entry covering the INTC memory map must be configured as guarded, both in software and hardware vector modes. • In software vector mode, the INTC_IACKR must not be read speculatively. • In hardware vector mode, guarded writes to the INTC_CPR or INTC_EOIR complete before the interrupt acknowledge signal from the processor asserts.

Table 1. MPC5534RM Rev 2.0 addendum (continued)

<table>
<tr><th>Location</th><th>Description</th></tr>
<tr><td>Table 16-9/ Page 16-15</td><td>Bit 7—DMA: Replace the table that shows the eMIOS channels that don’t support DMA with the following table.
<table>
<tr><th>eMIOS Channel</th><th>DMA = 0</th><th>DMA = 1</th></tr>
<tr><td>0</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>1</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>2</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>3</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>4</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>5</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>6</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>7</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>8</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>9</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>10</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>11</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>12</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>13</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>14</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>15</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>16</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>17</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>18</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>19</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>20</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>21</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>22</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>23</td><td>Interrupt</td><td>Reserved</td></tr>
</table></td></tr>
</table>

Table 1. MPC5534RM Rev 2.0 addendum (continued)

Location	Description
Table 10-9. INTC: Interrupt Request Sources/ Page 10-22	Update the note at the end of this table as follows: **Note:** The INTC has no spurious vector support. Therefore, if an asserted peripheral or software settable interrupt request (whose PRI value in INTC_PSRn is higher than the PRI value in INTC_CPR) negates before the interrupt request to the processor for that peripheral or software settable interrupt request is acknowledged, the interrupt request to the processor still can assert or remain asserted for that peripheral or software settable interrupt request. If the interrupt request to the processor does assert or does remain asserted: • The interrupt vector will correspond to that peripheral or software settable interrupt request. • The PRI value in the INTC_CPR will be updated with the corresponding PRI value in INTC_PSRn. Furthermore, clearing the peripheral interrupt request's enable bit in the peripheral or, alternatively, setting its mask bit has the same consequences as clearing its flag bit.Setting its enable bit or clearing its mask bit while its flag bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.
Section 10.4.2.1.4, “Priority Comparator Submodule”/ Page 10-23	Add the following paragraph to this section: One consequence of the priority comparator design is that once a higher priority interrupt is captured, it must be acknowledged by the CPU before a subsequent interrupt request of even higher priority can be captured. For example, if the CPU is executing a priority level 1 interrupt, and a priority level 2 interrupt request is captured by the INTC, followed shortly by a priority level 3 interrupt request to the INTC, the level 2 interrupt must be acknowledged by the CPU before a new level 3 interrupt will be generated.

Table 1. MPC5534RM Rev 2.0 addendum (continued)

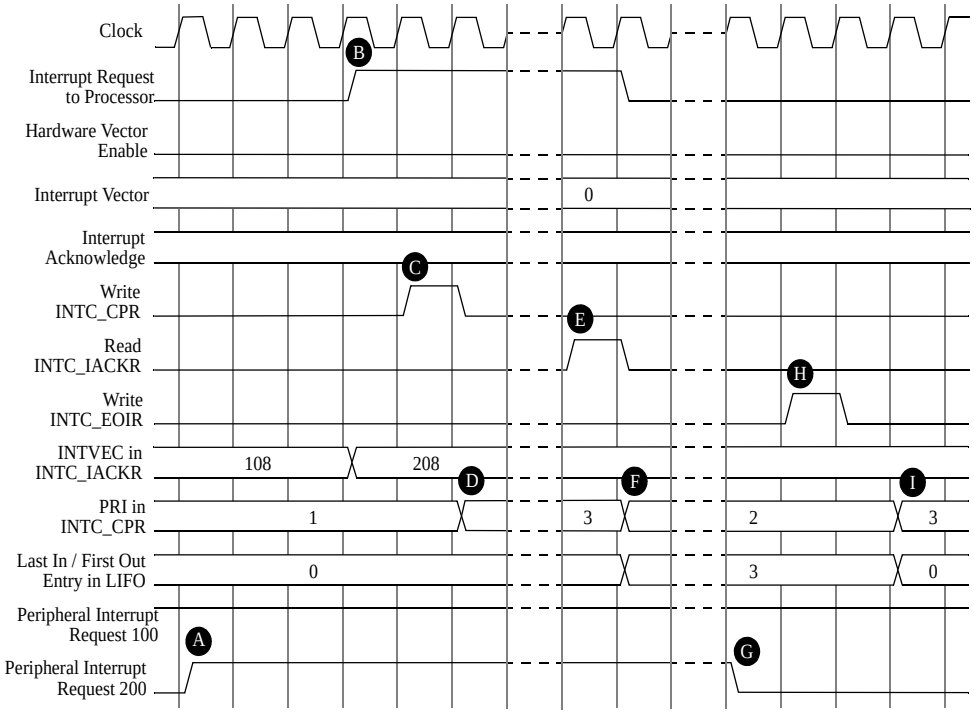
Location	Description
Section 10.5.5.2, “Ensuring Coherency”/ Page 10-31	Move the content of this section under a new heading Section 10.5.5.2.1, “Interrupt with Blocked Priority”. Add the following paragraph to this section: **Section 10.5.5.2.2: Raised Priority Preserved** Before the instruction after the GetResource system service executes, all pending transactions have completed. These pending transactions can include an ISR for a peripheral or software settable interrupt request whose priority was equal to or lower than the raised priority. Also, during the epilog of the interrupt exception handler for this preempting ISR, the raised priority has been restored from the LIFO to PRI in INTC_CPR. The shared coherent data block now can be accessed coherently. Following figure shows the timing diagram for this scenario, and the table explains the events. The example is for software vector mode, but except for the method of retrieving the vector and acknowledging the interrupt request to the processor, hardware vector mode is identical.  **Raised Priority Preserved Timing Diagram**

Table 1. MPC5534RM Rev 2.0 addendum (continued)

<table>
<tr><th>Location</th><th>Description</th></tr>
<tr><td></td><td>
Raised Priority Preserved Events
<table>
<tr><th>Event</th><th>Description</th></tr>
<tr><td>A</td><td>Peripheral interrupt request 200 asserts during execution of ISR108 running at priority 1.</td></tr>
<tr><td>B</td><td>Interrupt request to processor asserts. INTVEC in INTC_IACKR updates with vector for that peripheral interrupt request.</td></tr>
<tr><td>C</td><td>ISR108 writes to INTC_CPR to raise priority to 3 before accessing shared coherent data block.</td></tr>
<tr><td>D</td><td>PRI in INTC_CPR now at 3, reflecting the write. This write, just before accessing data block, is the last instruction the processor executes before being interrupted.</td></tr>
<tr><td>E</td><td>Interrupt exception handler prolog acknowledges interrupt by reading INTC_IACKR.</td></tr>
<tr><td>F</td><td>PRI of 3 pushed onto LIFO. PRI in INTC_CPR updates to 2, the priority of ISR208.</td></tr>
<tr><td>G</td><td>ISR208 clears its flag bit, deasserting its peripheral interrupt request.</td></tr>
<tr><td>H</td><td>Interrupt exception handler epilog writes to INTC_EOIR.</td></tr>
<tr><td>I</td><td>LIFO pops 3, restoring the raised priority onto PRI in INTC_CPR. Next value to pop from LIFO is the priority from before peripheral interrupt request 100 interrupted. ISR108 now can access data block coherently after interrupt exception handler executes rfi instruction.</td></tr>
</table>
</td></tr>
<tr><td>Section 18.4.5.4.1
“Calibration Overview”</td><td>Remove the braces from the equation “CAL_RES = GCC x (RAW_RES + OCC + 2)”.</td></tr>
<tr><td>Section 21.4.6.1
“Freeze mode”</td><td>Remove the text “CANx_RXIMRn registers can be programmed only if the MBFEN bit is asserted”.</td></tr>
<tr><td>Section 21.1.3
“Features”</td><td>• Remove the “Includes 256 bytes of RAM used for filtering individual RX mask registers” statement.
• Change “Programmable for global (compatible with previous versions) or individual receive ID masking” to “Programmable Global (compatible with previous versions) receive ID masking”.</td></tr>
<tr><td>Section 21.3.1
“Memory Map”</td><td>Change last three rows (Base+0x0880 - Base+0x097F) in "Module Memory Map" table to reserved</td></tr>
</table>

Table 1. MPC5534RM Rev 2.0 addendum (continued)

Location	Description
Section 21.3.3.1 “Module Configuration Register (CANx_MCR)”	In “CANx_MCR Field Descriptions” table, changed the MBFEN bit description. Change the whole text to remove all references to Individual masking. Left only Reception Queue function. The updated text is as follows: This bit provides the capability of enabling/disabling reception queue. By negating MBFEN, global masking is enabled and FlexCAN uses the Rx ID masking scheme of RXGMASK, RX14MASK and RX15MASK. MB14 and MB15 have individual masks and the others share the global mask. This configuration does not provide a reception queue; that is, a received message always fills the first matching buffer, setting the CODE field to overrun if the buffer contained an unread message. See Section 21.3.3.4, “RX Mask Registers” for more information. Keep MBFEN negated for compatibility with previous FlexCAN versions, which negates MBFEN at reset to retain compatibility with existing software. By asserting MBFEN, the reception queue features are enabled, while the global masking is used for message filtering as if MBFEN was negated. See Section 21.4.3.2, “Reception Queue” for more information. 0 = Reception queue features are disabled (thus the device is compatible with previous FlexCAN versions). 1 = Reception queue features are enabled.
Section 21.3.3.5 “RX Individual Mask Registers (CANx_RXIMR0 through CANx_RXIMR63)”	Remove this complete section.
Section 21.4.3.2 “Reception Queue”	Change “By programming more than one MB with the same ID, received messages are queued into the MBs. Matching to a range of IDs is possible by using ID acceptance masks that mask individual MBs” to “By programming more than one MB with the same ID, received messages are queued into the MBs. Matching to a range of IDs is possible by using ID acceptance masks”.
Section 21.5.1 “FlexCAN2 Initialization Sequence”	Change “The initialization of FlexCAN registers for either global or individual acceptance masking depends on the configuration of MBFEN: - If MBFEN is negated, initialize CANx_RXGMASK, CANx_RX14MASK, and CANx_RX15MASK registers for acceptance mask. - If MBFEN is asserted, initialize CANx_RXIMR[0-63] for individual acceptance masking”. to “Initialize FlexCAN registers for reception queue along with global acceptance masking. For either MBFEN state, initialize CANx_RXGMASK, CANx_RX14MASK, and CANx_RX15MASK registers for acceptance mask. - If MBFEN is negated, the reception queue feature is disabled. - If MBFEN is asserted, the reception queue feature is enabled”.
Section 11.3.1.1 Synthesizer Control Register (FMPLL_SYNCR)	Delete the last note in PREDIV field description that states: “To use the 8–20 MHz OSC, the PLL predivider must be configured for divide-by-two operation by tying PLLCFG[2] low (set PREDIV to 0b000).”

2 Revision history

Table 2 provides a revision history for this document.

Table 2. Revision history Table

Revision	Substantive changes	Date of release
1.0	• Initial release. Correct error in chapter 9, “Enhanced Direct Memory Access (eDMA).”	11/2009
2.0	• Correct errors in chapter 5, “Peripheral Bridge.” • Correct peripheral bridge name errors in appendix A, “MPC5534 Register Map.” • Correct eTPU number in the Chapter 1, “Overview” section. • Correct address offset in SIU_IREER, chapter 6, “SIU.” • Correct the table heading in Table 6-12, “SIU_DIRER Field Descriptions”. • Correct the arrow position in Figure 16-12, ”Unified Channel Block Diagram”. • Clarify the description in Section 9.4.1, “eDMA Microarchitecture”. • Clarify the description in Section 9.3.1.13, “eDMA Interrupt Request Registers (EDMA_IRQRH, EDMA_IRQRL). • Clarify Section 10.5.5.1, “Elevating Priority”, Section 10.5.6, “Selecting Priorities According to Request Rates and Deadlines”, Section 10.5.7.1, “Scheduling a Lower Priority Portion of an ISR”, Section 10.5.8, “Lowering Priority Within an ISR”. • Clarify bit description in Table 6-29, “SIU_DISR Field Descriptions”. • Clarify note in the INTC Interrupt Acknowledge Register. • Add a note in the INTC Memory Map table. • Clarify note at the end of the INTC: Interrupt Request Sources table. • Add a paragraph to the Section 10.4.2.1.4, “Priority Comparator Submodule”. • Update Section 10.5.5.2, “Ensuring Coherency”. • Update Section 21.4.6.1 “Freeze mode”. • In section in 21.1.3, remove the following statement: “Includes 256 bytes of RAM used for filtering individual RX mask registers”. • In section 21.1.3, changed from “Programmable for global (compatible with previous versions) or individual receive ID masking” to “Programmable Global (compatible with previous versions) receive ID masking”. • In in Table 21-2, made the last three rows reserved (Base+0x0880 -- Base+0x097F Reserved). • In Table 21-7, changed the MBFEN bit description - changed the whole text to remove all references to Individual masking. Leave only Reception Queue function. • Removed the section 21.3.3.5, section 21.4.3.2, and section 21.5.1	12/2011
3.0	• Deleted the note in PREDIV field description of Synthesizer Control Register (FMPLL_SYNCR) that states “To use the 8–20 MHz OSC, the PLL predivider must be configured for divide-by-two operation by tying PLLCFG[2] low (set PREDIV to 0b000).”	04/2012

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

MPC5534RMAD
Rev. 3
04/2012

MPC5534 Microcontroller Reference Manual

Devices Supported:

MPC5534

MPC5534RM
Rev. 2
5 Oct 2008

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 26668334
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

MPC5534RM
Rev. 2
5 Oct 2008

Table of Contents

Chapter 1
Overview

Chapter 2
Signals

Chapter 3
Core Complex (e200z3)

Chapter 4
Reset

Chapter 5
Peripheral Bridge

Chapter 6
System Integration Unit (SIU)

Chapter 7
Crossbar Switch (XBAR)

Chapter 8
Error Correction Status Module (ECSM)

Chapter 9
Enhanced Direct Memory Access (eDMA)

Chapter 10
Interrupt Controller (INTC)

Chapter 11
Frequency Modulated Phase Locked Loop and System Clocks (FMPLL)

Chapter 12
External Bus Interface (EBI)

Chapter 13
Flash Memory

Chapter 14
Internal Static RAM (SRAM)

Chapter 15
Boot Assist Module (BAM)

Chapter 16
Enhanced Modular Input/Output Subsystem (eMIOS)

Chapter 17
Enhanced Time Processing Unit (eTPU)

Chapter 18
Enhanced Queued Analog-to-Digital Converter (eQADC)

Chapter 19
Deserial Serial Peripheral Interface (DSPI)

Chapter 20
Enhanced Serial Communication Interface (eSCI)

Chapter 21
FlexCAN2 Controller Area Network

Chapter 22
Voltage Regulator Controller (VRC) and POR Module

Chapter 23
IEEE 1149.1 Test Access Port Controller (JTAGC)

Chapter 24
Nexus Development Interface

Appendix A
MPC5534 Register Map

Appendix B
Calibration

Appendix C
MPC5534RM Revision History

Chapter 1
Overview

The MPC5534 microcontroller (MCU) is a member of the MPC5500 family of next generation powertrain microcontrollers built on Power Architecture™ technology. The MPC5500 family contains a host processor core that complies with the Power Architecture embedded category, which is 100 percent user mode compatible with the original Power PC™ user instruction set architecture (UISA). This family of parts contains many new features coupled with high-performance CMOS technology to provide significant performance improvement over the MPC565.

The e200z3 CPU of the MPC5500 family is part of the family of CPU cores that implement versions built on the Power Architecture embedded category. This core also has additional instructions, including digital signal processing (DSP) instructions, beyond the classic PowerPC instruction set.

The e200z3 of the MPC5534 is compatible with the PowerPC Book E architecture. It is 100% user mode compatible (with floating point library) with the classic PowerPC instruction set. The Book E architecture has enhancements that improve the PowerPC architecture's fit in embedded applications. This core also has additional instructions, including digital signal processing (DSP) instructions, beyond the classic PowerPC instruction set.

The host processor core of the MPC5534 also includes an instruction set enhancement allowing variable length encoding (VLE). This allows optional encoding of mixed 16- and 32-bit instructions. With this enhancement, it is possible to achieve significant code size footprint reduction.

The MPC5534 has a single-level of memory hierarchy consisting of 64-KB on-chip SRAM and 1 MB of internal flash memory. Both the SRAM and the flash memory can hold instructions and data.

The External Bus Interface (EBI) supports most standard memories used with the MPC5xx family. This device does not support arbitration between itself and other masters on the external bus. It must be the only master on the EBI or be a slave-only device.

The complex I/O timer functions of MPC5534 are performed by a dual Enhanced Time Processor Unit engine (eTPU). The eTPU engine controls 32 hardware channels and has been enhanced over the MPC500 family's TPU by providing 24-bit timers, double-action hardware channels, a variable number of parameters per channel, angle clock hardware, and additional control and arithmetic instructions. The eTPU is programmed using a high-level programming language.

The less complex timer functions of MPC5534 are performed by the enhanced Modular Timer System (eMIOS). The eMIOS 24 hardware channels are capable of single action, double action, pulse width modulation (PWM) and modulus counter operation. Motor control capabilities include edge-aligned and center-aligned PWM.

Off-chip communication is performed by a suite of serial protocols including flexible controller area networks (FlexCANs), enhanced SPIs (Deserialize/Serialize Peripheral Interface) and SCIs. The DSPIs support pin reduction through hardware serialization and deserialization of timer channels and GPIO signals.

The MPC5534 MCU has an on-chip 40-channel Enhanced Queued Dual Analog-to-Digital Converter (eQADC), with 5 V conversion range.

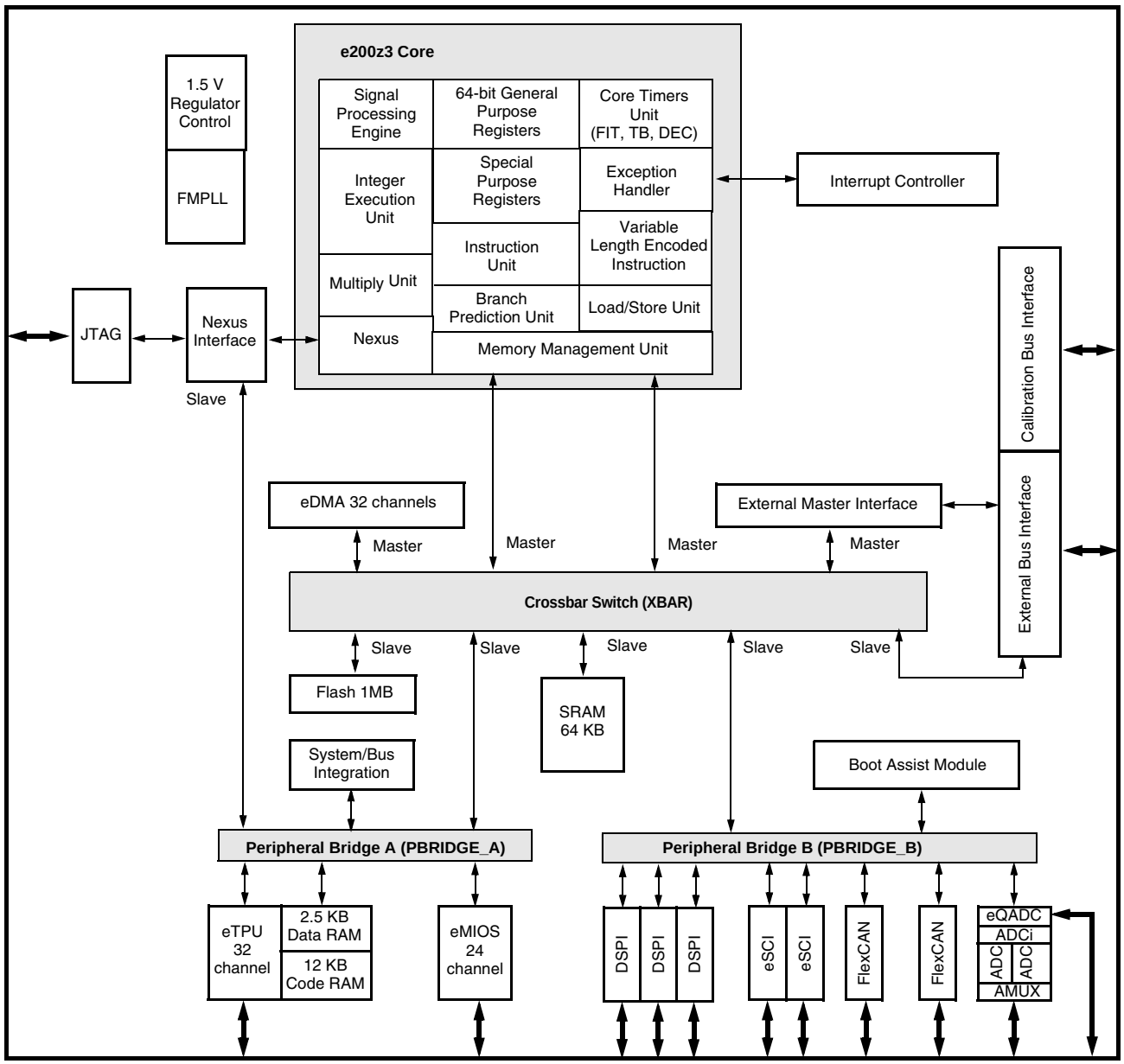
The System Integration Unit (SIU) performs several chip-wide configuration functions. Pad configuration and General-Purpose Input and Output (GPIO) are controlled from the SIU. External interrupts and reset control are configured in the SIU. The Internal Multiplexer sub-block (IMUX) provides multiplexing of eQADC trigger sources, daisy chaining the DSPIs, and external interrupt signal multiplexing.

On-chip modules include:

- Single issue, 32-bit PowerPC Book E compatible CPU core complex; includes VLE enhancements for code size footprint reduction
- 32-channel enhanced direct memory access controller (eDMA)
- Interrupt controller (INTC) capable of handling 210 selectable-priority interrupt sources
- Frequency Modulated Phase-locked loop (FMPLL)
- External bus interface (EBI) with error correction status module (ECSM)
- System integration unit (SIU)
- 1 MB on-chip flash with flash control unit (FCU)
- 64 KBs on-chip static RAM (SRAM)
- Boot assist module (BAM)
- 32-channel enhanced time processor unit (eTPU)
- 24-channels enhanced modular Input Output System (eMIOS)
- Two enhanced 5 V Queued Analog-to-Digital Converters (eQADC)
- Three deserial serial peripheral interface (DSPI) modules
- Two enhanced serial communication interface (eSCI) modules (with LIN support)
- Two controller area network (FlexCAN) modules
- Nexus development interface (NDI) per IEEE-ISTO 5001-2003 standard
- Device/board test support per Joint Test Action Group (JTAG) of IEEE (IEEE 1149.1)
- On-chip voltage regulator controller for regulating 3.3 V down to 1.5 V for core logic

1.1 Block Diagram

Figure 1-1 shows a top-level block diagram of the MPC5534.



LEGEND

MPC5500 Device Module Acronyms

CAN – Controller area network (FlexCAN)
DSPI – Deserial/serial peripheral interface
eDMA – Enhanced direct memory access
eMIOS – Enhanced modular I/O system
eQADC – Enhanced queued analog/digital converter
eSCI – Enhanced serial communications interface
eTPU – Enhanced time processing units
FMPLL – Frequency modulated phase-locked loop
SRAM – Static RAM

e200z3 Core Component Acronyms

DEC – Decrementer
FIT – Fixed interval timer
TB – Time base
WDT – Watchdog timer

Figure 1-1. MPC5534 Block Diagram

1.1.1 MPC5500 Family Comparison

The following table compares the features of the MPC553X and MPC555X products:

Table 1-1. MPC5500 Family Members

MPC5500 Device		MPC5533	MPC5534	MPC5553	MPC5554
Power PC core		e200z3	e200z3	e200z6	e200z6
Variable Length Instruction support		Y	Y	—	—
Unified cache	(KB)	—	—	8 [1]	32 [2]
Memory management unit (MMU)		16 entry	16 entry	32 entry	32 entry
Crossbar	(Master x Slave)	3 x 5	4 x 5	4 x 5	3 x 5
Core Nexus		Class 3+ (NZ3C3)	Class 3+ (NZ3C3)	Class 3+ (NZ6C3)	Class 3+ (NZ6C3)
Static RAM (SRAM)	(KB)	48 KB	64 KB	64 KB	64 KB
Flash memory	Main Array (MB)	768 KB [3]	1	1.5[4]	2[4]
	Shadow Block (KB)	1	1	1	1
External bus interface (EBI)	Data Bus	16 bit	16 bit	32 bit [6]	32 bit
	Address Bus	24 bit	24 bit	24 bit	24 bit
Calibration bus interface (CBI)		—	Y	Partial	—
Enhanced direct memory access (eDMA)		32 channel	32 channel	32 channel	64 channel
DMA Nexus		—	—	Class 3	Class 3
Enhanced serial communications interface (eSCI)		1	2	2	2
	eSCI A	Y	Y	Y	Y
	eSCI B	—	Y	Y	Y
Flexible controller area network (FlexCAN)		2	2	2	3
	CAN A	64 buffers	64 buffers	64 buffers	64 buffers
	CAN B	—	—	—	64 buffers
	CAN C	64 buffers	64 buffers	64 buffers	64 buffers
	CAN D	—	—	—	—
	CAN E	—	—	—	—
Deserial/serial peripheral interface (DSPI)		2	3	3	4
	DSPI A	—	—	—	Y
	DSPI B	—	Y	Y	Y
	DSPI C	Y	Y	Y	Y
	DSPI D	Y	Y	Y	Y
Enhanced management input/output system (eMIOS)		—	24 channel	24 channel	24 channel
Enhanced time processing unit (eTPU)		32 channel	32 channel	32 channel	64 channel
	eTPU A	Y	Y	Y	Y
	eTPU B	—	—	—	Y
	Code memory (KB)	12	12	12	16
	Parameter RAM (KB)	2.5	2.5	2.5	3
Interrupt controller		178	210	210	300

Table 1-1. MPC5500 Family Members (continued)

MPC5500 Device		MPC5533	MPC5534	MPC5553	MPC5554
Enhanced Analog to Digital Converter (eQADC)		40 channels [5]	40 channels [5]	40 channels	40 channels
	ADC 0	Y	Y	Y	Y
	ADC 1	—	Y	Y	Y
Fast Ethernet Controller (FEC)		—	—	Y [6]	—
Frequency Modulated (FM) Phase Lock Loop (PLL)		Y	Y	Y	Y
Maximum system frequency [7]	(MHz)	82 [8]	82 [8]	132 [9]	132 [9]
Crystal frequency range	(MHz)	8–20	8–20	8–20	8–20
Voltage Regulator Controller (VRC)		Y	Y	Y	Y

1 2-way associative
2 8-way associative
3 16-byte flash page size for programming
4 32-byte flash page size for programming
5 eQADC has 34 channels on the 208 package
6 The FEC signals are shared with the data bus pins DATA[16:31]
7 Initial automotive temperature range qualification
8 82 MHz parts allow for 80 MHz system clock + 2% FM
9 132 MHz parts allow for 128 MHz system clock + 2% FM

1.2 MPC5534 Features List

This section provides a high-level description of the features found on the MPC5534.

1.2.1 Operating Parameters

- Fully static operation, 0–80 MHz
- -40° to 150° C junction temperature (125° C ambient temperature)
- Low power design
  - Less than 0.8 Watts power dissipation
  - Designed for dynamic power management of core and peripherals
  - Software controlled clock gating of peripherals
  - Separate power supply for stand-by operation for a portion of SRAM
- Fabricated in 0.13 µm process
- 1.5 V internal logic
- Input and output pins with 3.0–5.5 V range
  - 35% or 65% V_{DDE} CMOS switch levels (with hysteresis)
  - Selectable hysteresis
  - Selectable slew rate control
- External bus and Nexus pins support 1.6–3.6 V operation
  - Selectable drive strength control

  - Unused pins configurable as GPIO
- Designed with EMI reduction techniques
  - Frequency Modulated Phase-Locked Loop (FMPLL)
  - On-chip bypass capacitance
  - Selectable slew rate and drive strength

1.2.2 e200z3 Core Processor

- Single issue, 32-bit PowerPC Architecture compatible CPU
  - In-order execution and retirement
  - Precise exception handling
  - User-mode binary compatible with MPC5xx except floating point instructions
- Variable Length Encoding Enhancements
  - e200z3 core supports both PowerPC Architecture Book E and VLE instruction sets
  - Allows optional encoding of mixed 16- and 32-bit instructions
  - Results in smaller code size footprint
  - Regions of the memory map are designated as PowerPC Architecture Book E or VLE based on configuration of the Memory Management Unit
- Branch processing unit
  - Dedicated branch address calculation adder
  - Branch acceleration using Branch Lookahead Instruction Buffer
- Load and store unit
  - Fully pipelined
  - Big and Little endian support
  - Misaligned access support
  - Zero load-to-use pipeline bubbles
  - Supports throughput of one load or store operation per cycle
  - Memory interface support for saving and restoring two registers per cycle
- Thirty-two 64-bit general purpose registers (GPRs)
- Memory management unit (MMU) with 16-entry fully-associative translation look-aside buffer (TLB)
- Separate instruction bus and load/store bus
- Vectored interrupt support
- Interrupt latency < 116 ns at 80 MHz (measured from interrupt request to execution of first instruction of interrupt exception handler)
- Reservation instructions for implementing read-modify-write constructs (CPU master only)

- Numerics and DSP
  - Saturated, unsaturated, and fractional arithmetic
  - Support for DSP addressing modes
  - Pipelined dual 32x32 MAC with one clock throughput
- Signal processing extension APU
  - Operating on all 32 GPRs that are all extended to 64 bits wide
  - Single Instruction Multiple Data (SIMD) provides a full compliment of vector and scalar integer and floating point arithmetic operations (integer vector MAC and MUL operations)
  - Provides rich array of extended 64-bit loads and stores to and from the extended GPRs
  - Fully code compatible with e200z6 core
- Floating point
  - IEEE 754 compatible with software wrapper
  - Scalar single precision in hardware, double precision with software library
  - Conversion instructions between single precision floating point and fixed point
  - Fully code compatible with e200z6 core
- Long cycle time instructions, except for guarded loads, do not increase interrupt latency
- Extensive system development support through Nexus debug port

1.2.3 Crossbar Switch (XBAR)

- Four master ports, and five slave ports
  - Masters: CPU Instruction bus, CPU Load/store bus, eDMA, EBI
  - Slave: Flash, SRAM, Peripheral Bridge A, Peripheral Bridge B; EBI
- 32-bit internal address bus, 64-bit internal data bus

1.2.4 Enhanced Direct Memory Access (eDMA) Controller

- 32 channels support independent 8-, 16- or 32-bit single value or block transfers
- Supports variable sized queues and circular queues
- Source and destination address registers are independently configured to post-increment or remain constant
- Each transfer is initiated by a peripheral, CPU, or eDMA channel request
- Each eDMA channel can optionally send an interrupt request to the CPU on completion of a single value or block transfer
- All data movement via dual-address transfers: read from source, write to destination
- Programmable source and destination addresses, transfer size, plus support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations

- An inner data transfer loop defined by a ‘minor’ byte transfer count
- An outer data transfer loop defined by a ‘major’ iteration count
- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
  - Peripheral-paced hardware requests (one per channel)
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
- One interrupt per channel, optionally asserted at completion of major iteration count
- Error termination interrupts are optionally enabled
- Support for scatter/gather DMA processing
- Channel transfers can be suspended by a higher priority channel

1.2.5 Interrupt Controller (INTC)

- Unique 9-bit vector per interrupt source for 210 total interrupt sources:
  - 190 peripheral interrupt sources
  - Eight software settable interrupt sources
  - 12 reserved interrupt sources
- 16 priority levels with fixed hardware arbitration within priority levels for each interrupt source
- Priority elevation for shared resources

1.2.6 Frequency Modulated Phase-Locked Loop (FMPLL)

- Input clock frequency from 8–20 MHz
- Current controlled oscillator (ICO) range from 48 MHz to maximum device frequency
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the FMPLL to re-lock
- Four modes of operation
  - Bypass mode
  - PLL normal mode with crystal reference (default)
  - PLL normal mode with external oscillator reference
  - PLL dual controller mode for EXTAL to CLKOUT skew minimization
- Programmable frequency modulation
  - Modulation enabled/disabled through software
  - Triangle wave modulation
  - Programmable modulation depth (1% or 2% deviation from center frequency)
  - Programmable modulation frequency dependent on reference frequency

- Lock detect circuitry reports when the FMPLL has achieved frequency lock and continuously monitors lock status to report loss of lock conditions
- Programmable interrupt request on system reset or loss of lock
- Loss-of-clock (LOC) detection for reference and feedback clocks
- Programmable interrupt request on system reset or loss of clock
- Self-clocked mode (SCM) operation

1.2.7 External Bus Interface (EBI)

NOTE

EBI features apply to devices using the 324 package. The EBI is not available in the 208 package.

- 1.8–3.3 V I/O
- Up to 24-bit address bus
  - Four most significant signals multiplexed with four chip selects
- 16-bit data bus
- Memory controller with support for various memory types:
  - Standard SRAM
  - Synchronous burst SDR (flash and SRAM)
  - Asynchronous/legacy (flash and SRAM)
- Most standard memories used with the MPC5xx family
- Single-master only or slave only operation
- Bus monitor
  - User selectable
  - Programmable time-out period (with 8 external bus clock resolution)
- Configurable wait states (via chip selects)
- Four chip-select ($\overline{CS}$[0:3]) signals (multiplexed with four most significant address signals)
- Two write/byte enable ($\overline{WE}$/$\overline{BE}$[0:1]) signals
- Configurable bus speed modes
  - system frequency
  - ½ of system frequency
  - ¼ of system frequency
- Configurable wait states
- Optional automatic CLKOUT gating to save power and reduce EMI
- Compatible with MPC5xx external bus (with some limitations)
- Selectable drive strengths through pad control in SIU; 10pF, 20pF, 30pF, 50pF

1.2.8 Calibration Bus Interface (CBI)

NOTE

The CBI is not available in the 208 package. CBI features apply to devices using the 324 package with the 496 assembly.

- 21-bit address bus
  - Two most significant signals multiplexed with two chip selects
  - Least significant address bit is not supported (CAL_ADDR31)
- 16-bit data bus
- Memory controller with support for various memory types:
  - non-burst SDR flash and SRAM
  - Asynchronous/legacy flash and SRAM
  - Most standard memories used with the MPC5xx family
- Bus monitor (shared with EBI)
  - User selectable
  - Programmable timeout period (with 8 external bus clock resolution)
- Configurable wait states (via chip selects)
- Three chip-select (CAL_$\overline{CS}$[0], CAL_$\overline{CS}$[2:3]) signals:
  (CAL_$\overline{CS}$[2:3] are multiplexed with the two most significant address signals CAL_ADDR[10:11])
- Two write/byte enable ($\overline{WE}$/$\overline{BE}$[0:1]) signals
- Configurable bus speed modes (shared with EBI)
  - system frequency
  - ½ of system frequency
  - ¼ of system frequency
- Optional automatic CLKOUT gating to save power and reduce EMI
- Compatible with MPC5xx external bus (with some limitations)
- Selectable drive strengths; 10pF, 20pF, 30pF, 50pF

1.2.9 System Integration Unit (SIU)

- System configuration
  - MCU reset configuration via external pins
  - Pad configuration control
- System reset monitoring and generation
  - Power-on reset support
  - Reset status register provides last reset source to software
  - Glitch detection on reset input

  - Software controlled reset assertion
- External interrupt
  - Sixteen interrupt requests
  - Rising- or falling-edge event detection
  - Programmable digital filter for glitch rejection
- GPIO
  - GPIO function on 155 I/O pins
  - Dedicated input and output registers for setting each GPIO
- Internal Multiplexing
  - Allows serial and parallel chaining of DSPIs
  - Allows flexible selection of eQADC trigger inputs
  - Allows selection of interrupt requests between external pins and DSPI

1.2.10 Error Correction Status Module (ECSM)

- Configurable error-correcting codes (ECC) reporting

1.2.11 On-chip Flash

- One MB burst flash memory
  - Configured as 64K x 128 bits
  - 12 blocks (2×16 KB + 2×48 KB + 2×64 KB + 6×128 KBs) to support features such as boot block, operating system block and EEPROM emulation
  - 1-KB shadow block compatible with all other parts in the family for storing censorship and configuration information (censorship protection scheme to prevent flash content visibility)
  - Accessed via a 64-bit wide bus interface
- Quadruple 128-bit wide prefetch/burst buffers to provide single cycle in-line accesses (prefetch buffers can be configured to prefetch code or data or both)
- Hardware read-while-write feature that allows blocks to be erased/programmed while other blocks are being read (used for EEPROM emulation and data calibration)
- Page mode (128-bits) programming for rapid end-of-line programming
- Hardware programming state machine
- Supports a 64-bit data bus. Byte, halfword, word and doubleword reads are supported. Only aligned word and doubleword writes are supported.
- Hardware and software configurable read and write access protections on a per-master basis.
- Interface to the flash array controller is pipelined with a depth of 1, allowing overlapped accesses to proceed in parallel for interleaved or pipelined flash array designs.
- Configurable access timing allowing use in a wide range of system frequencies.

- Multiple-mapping support and mapping-based block access timing (0–31 additional cycles) allowing use for emulation of other memory types.
- Software programmable block program/erase restriction control.
- Erase of selected block(s)
- Read page size of 128 bits (4 words)
- ECC with single-bit correction, double-bit detection.
- Embedded hardware program and erase algorithm
- Erase suspend, program suspend and erase-suspended program
- Shadow information stored in non-volatile shadow block
- Independent program/erase of the shadow block

1.2.12 On-chip Static RAM (SRAM)

- Supports read/write accesses mapped to the SRAM memory from any master
- 64-KB general purpose RAM of which 32 KBs are on standby power
- 32-KB block powered by separate supply for standby operation
- Byte, halfword, word and doubleword addressable
- ECC performs single bit correction, double bit detection on 32-bit data element

1.2.13 Boot Assist Module (BAM)

- Enables and manages the transition of MCU from reset to user code execution in the following configurations:
  - Execution from internal or external flash
  - Download and execution of code via FlexCAN or eSCI
- Sets up MMU to cover all resources and mapping all physical address to logical addresses with minimum address translation
- Sets up the MMU to allow user boot code to execute as either PowerPC Architecture Book E code (default) or as VLE code
- Location and detection of user boot code
- Automatic switch to serial boot mode if internal or external flash is blank or invalid
- Supports user programmable 64-bit password protection for serial boot mode
- Supports serial bootloading via FlexCAN bus and eSCI
- Supports serial bootloading of either PowerPC Architecture Book E code (default) or VLE code
- Supports censorship protection for internal flash memory
- Provides an option to enable the core watchdog timer

1.2.14 Enhanced Modular I/O System (eMIOS)

- 24 unified channels with these features:
  - 24-bit registers for captured/match values
  - 24-bit internal counter
  - Global prescaler
  - Separate input and output pins
  - Dedicated output pin for buffer direction control
  - Selectable time base
  - Can generate its own time base
- Four 24-bit wide counter buses
  - Counter Bus A can be shared among all unified channels and can be driven by either UC23 or by the STAC bus
  - Counter Bus B is driven by UC0, Counter Bus C is driven by UC8, Counter Bus D is driven by UC16 (then, UC0-UC7 can share Counter Bus B, UC8-UC15 can share Counter Bus C and UC16-UC23 can share Counter Bus D)
- Synchronization among internal and external time bases
- Shadow FLAG register
- State of block can be frozen for debug purposes
- 24 orthogonal channels with double-action, PWM and modulus counter functionality
- Supports all DASM and PWM modes of MIOS14 (MPC5xx)
- DMA and interrupt request support
- Motor control capability

1.2.15 Enhanced Time Processor Unit (eTPU)

- eTPU engine is an event triggered VLIW processor timer subsystem
- High level assembler/compiler
- 32 channels
- 24-bit timer resolution
- Code memory—12 KB; Data memory—2.5 KB
- Variable number of parameters allocatable per channel
- Double match/capture channels
- Angle clock hardware support
- Shared time or angle counter bus (STAC) for all eTPU and eMIOS channels
- DMA and interrupt request support
- Nexus Class 3 Debug support (with some Class 4 support)

1.2.16 Enhanced Queued A/D Converter (eQADC)

- Two independent ADCs
  - 12 Bit AD Resolution
  - Up to 10 bit accuracy at 400 ksample/s and 8 bit accuracy at 800 ksample/s
  - Differential conversions
  - Single-ended signal range from 0 to 5V
  - Sample times of 2 (default), 8, 64 or 128 ADC clock cycles
  - Provides time stamp information when requested
  - Parallel interface to eQADC CFIFOs and RFIFOs
  - Supports both right-justified unsigned and signed formats for conversion results
- 0–5 V common mode conversion range
- 40 single-ended inputs channels (40 in 324 BGA; 34 in 208 BGA), with support for up to 25 additional channels using external multiplexers
- Eight channels can be used as 4 pairs of differential analog input channels
- 10 bit accuracy at 400 ksample/s, 8-bit accuracy at 800 ksample/s
- Supports six FIFO queues with fixed priority.
- Queue modes with priority-based preemption, initiated by software command or internal (eTPU and eMIOS) or external triggers
- DMA and interrupt request support
- Supports all functional modes from QADC (MPC5xx family)
- Four pairs of differential analog input channels
- Full duplex synchronous serial interface (SSI) to an external device
  - Free-running clock for use by an external device
  - Supports a 26-bit message length
- Priority based CFIFOs
  - Supports six CFIFOs with fixed priority. The lower the CFIFO number, the higher its priority. When commands of distinct CFIFOs are bound for the same CBuffer, the higher priority CFIFO is always served first
  - Supports software and hardware trigger modes to arm a particular CFIFO
  - Generates interrupt when command coherency is not achieved
- External hardware triggers
  - Supports rising-edge, falling-edge, high-level and low-level triggers
  - Supports configurable digital filter
- Supports 4 external 8-to-1 muxes which can expand the input channels

1.2.17 Deserial Serial Peripheral Interface (DSPI) Module

- Three DSPI modules
- SPI
  - Full duplex communication ports with interrupt and DMA request support
  - Supports all functional modes from QSPI sub-block of QSMCM (MPC5xx family)
  - Support for queues in RAM
  - Six chip selects, expandable to 64 with external demultiplexers
  - Programmable frame size, baud rate, clock delay and clock phase on a per frame basis
  - Modified SPI mode for interfacing to peripherals with longer setup time requirements
- Deserial serial interface (DSI)
  - Pin reduction by hardware serialization and deserialization of eTPU and eMIOS channels
  - Chaining of DSI sub-blocks
  - Triggered transfer control and change in data transfer control (for reduced EMI)

1.2.18 Enhanced Serial Communication Interface (eSCI) Module

- Two eSCI modules
- UART mode provides NRZ format and half or full duplex interface
- eSCI bit rate up to 1 Mbps
- Advanced error detection, and optional parity generation and detection
- Separately enabled transmitter and receiver
- eDMA support (on one of the eSCI modules only)
- 13-bit baud rate selection
- Programmable 8- or 9-bit data format
- LIN support
  - Autonomous transmission of entire frames
  - Configurable to support all revisions of the LIN standard
  - Automatic parity bit generation
  - Double stop bit after bit error
  - 10- or 13-bit break support
- Separately enabled transmitter and receiver
- Programmable transmitter output parity
- Two receiver wake-up methods, idle line and address mark
- Interrupt-driven operation with flags
- Receiver framing error detection
- Hardware parity checking

- 1/16 bit-time noise detection
- Two channel eDMA interface

1.2.19 FlexCAN

- Two FlexCAN modules
- 64 message buffers each (0–8 bytes data length)
- Based on and including all existing features of the Freescale TouCAN module
- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Zero to eight bytes data length
  - Programmable bit rate up to 1 Mb/sec
- Programmable acceptance filters
- Short latency time for high priority transmit messages
- Arbitration scheme according to message ID or message buffer number
- Listen only mode capabilities
- Programmable clock source: system clock or oscillator clock
- Content-related addressing
- Each message buffer is configurable as receive (Rx) or transmit (Tx) buffers that support standard and extended messages
- Includes 1056 bytes of embedded memory for message buffer storage
- Programmable loop-back mode supporting self-test operation
- Three programmable Mask Registers
- Programmable transmit-first scheme: lowest ID or lowest buffer number
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Multi master concept
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages
- Low power mode, with programmable wake-up on bus activity

1.2.20 Nexus Development Interface (NDI)

- Per IEEE-ISTO 5001-2003
- Real time development support for e200z3 core and eTPU engine through Nexus Class 3 (selected Class 4 support)
- Read and write access
  - Run-time access of entire memory map
  - Calibration (table constants calibrated using MMU and internal and external RAM; scalar constants calibrated using cache line locking)
- Configured via the IEEE 1149.1 (JTAG) port
- High bandwidth mode for fast message transmission
- Reduced bandwidth mode for reduced pin usage

1.2.21 IEEE 1149.1 JTAG controller (JTAGC)

- IEEE 1149.1-2001 Test Access Port (TAP) interface 4 pins (TDI, TMS, TCK, and TDO)
- JCOMP input that provides the ability to share the TAP
  (selectable modes of operation include JTAGC/debug or normal system operation)
- Selectable modes of operation include JTAGC/debug or normal system operation.
- Five-bit instruction register that supports the following IEEE 1149.1-2001 defined instructions:
  - BYPASS, IDCODE, EXTEST, SAMPLE, SAMPLE/PRELOAD, HIGHZ, CLAMP
- Five-bit instruction register that supports the additional following public instructions:
  - ACCESS_AUX_TAP_NPC, ACCESS_AUX_TAP_ONCE, ACCESS_AUX_TAP_eTPU
- Three test data registers: a bypass register, a boundary scan register, and a device identification register
- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry

1.2.22 On-chip Voltage Regulator Controller

- Uses external NPN bipolar transistor
- Regulates 3.3 V down to 1.5 V for the core logic

1.3 MPC5534 Memory Map

This section describes the MPC5534 memory map. All addresses in the device, including those that are reserved, are identified in the tables. The addresses represent the physical addresses assigned to each IP block. Logical addresses are translated by the MMU into physical addresses.

Under software control of the Memory Management Unit (MMU), the logical addresses allocated to IP blocks can be changed on a minimum of a 4-KB boundary. Table 1-2 shows the MPC5534 memory map. Peripheral blocks can be redundantly mapped. You must use the MMU to prevent corruption.

Table 1-2. MPC5534 Memory Map

Address Range[1]	Bytes		Module
	Allocated	Used	
0x0000_0000–0x000F_FFFF	1 MB	1 MB	Flash Array
0x0010_0000–0x00FF_FBFF	15 MB	—	Reserved
0x00FF_FC00–0x00FF_FFFF	1 KB	1 KB	Flash Shadow Block
0x0100_0000–0x1FFF_FFFF	496 MB	1 MB	Emulation mapping of Flash Array
0x2000_0000–0x3FFF_FFFF	256 MB	—	External Memory [2]
0x4000_0000–0x4000_7FFF	32 KB	32 KB	SRAM Array, Standby Powered
0x4000_8000–0x4000_FFFF	32 KB	32 KB	SRAM Array
0x4001_0000–0xBFFF_FFFF	2048 MB–64 KB	—	Reserved
Bridge A Peripherals			
0xC000_0000–0xC3EF_FFFF	63 MB	—	Reserved
0xC3F0_0000–0xC3F0_3FFF	16 KB	4	Bridge A Registers
0xC3F0_4000–0xC3F7_FFFF	496 KB	—	Reserved
0xC3F8_0000–0xC3F8_3FFF	16 KB	20	Frequency Modulated Phase-Locked Loop (FMPLL)
0xC3F8_4000–0xC3F8_7FFF	16 KB	48	External Bus Interface (EBI) Configuration
0xC3F8_8000–0xC3F8_BFFF	16 KB	28	Flash Configuration
0xC3F8_C000–0xC3F8_FFFF	16 KB	—	Reserved
0xC3F9_0000–0xC3F9_3FFF	16 KB	2.5 KB	System Integration Unit (SIU)
0xC3F9_4000–0xC3F9_FFFF	48 KB	—	Reserved
0xC3FA_0000–0xC3FA_3FFF	16 KB	1056	Enhanced Modular Input/Output System (eMIOS)
0xC3FA_4000–0xC3FB_FFFF	112 KB	—	Reserved
0xC3FC_0000–0xC3FC_3FFF	16 KB	3 KB	Enhanced Time Processing Unit (eTPU) Registers
0xC3FC_4000–0xC3FC_7FFF	16 KB	—	Reserved
0xC3FC_8000–0xC3FC_BFFF	16 KB	2.5 KB	Enhanced Time Processing Unit (eTPU) Parameter RAM
0xC3FC_C000–0xC3FC_FFFF	16 KB	2.5 KB	Enhanced Time Processing Unit (eTPU) Parameter RAM mirror

Table 1-2. MPC5534 Memory Map (continued)

Address Range[1]	Bytes		Module
	Allocated	Used	
0xC3FD_0000–0xC3FD_3FFF	16 KB	12 KB	Enhanced Time Processing Unit (eTPU) Code RAM
0xC3FD_4000–0xC3FF_FFFF	176 KB	—	Reserved
0xC400_0000–0xDFFF_FFFF	(512–64 MB)	—	Reserved
Bridge B Peripherals			
0xE000_0000–0xFBFF_FFFF	(512–64 MB)	—	Reserved
0xFC00_0000–0xFFEF_FFFF	63 MB	—	Reserved
0xFFF0_0000–0xFFF0_3FFF	16 KB	—	Bridge B Registers
0xFFF0_4000–0xFFF0_7FFF	16 KB	—	Crossbar (XBAR)
0xFFF0_8000–0xFFF0_FFFF	32 KB	—	Reserved
0xFFF1_0000–0xFFF3_FFFF	192 KB	—	Reserved
0xFFF4_0000–0xFFF4_3FFF	16 KB	—	Memory Control Management
0xFFF4_4000–0xFFF4_7FFF	16 KB	—	Enhanced Direct Memory Access Controller 2 (eDMA)
0xFFF4_8000–0xFFF4_BFFF	16 KB	—	Interrupt Controller (INTC)
0xFFF4_C000–0xFFF4_FFFF	16 KB	—	Reserved
0xFFF5_0000–0xFFF7_FFFF	192 KB	—	Reserved
0xFFF8_0000–0xFFF8_3FFF	16 KB	164	Enhanced Queued Analog to Digital Converter (eQADC)
0xFFF8_4000–0xFFF9_3FFF	64 KB	—	Reserved
0xFFF9_4000–0xFFF9_7FFF	16 KB	200	Deserial Serial Peripheral Interface (DSPI B)
0xFFF9_8000–0xFFF9_BFFF	16 KB	200	Deserial Serial Peripheral Interface (DSPI C)
0xFFF9_C000–0xFFF9_FFFF	16 KB	200	Deserial Serial Peripheral Interface (DSPI D)
0xFFFA_0000–0xFFFA_FFFF	64 KB	—	Reserved
0xFFFB_0000–0xFFFB_3FFF	16 KB	44	Serial Communications Interface (eSCI A)
0xFFFB_4000–0xFFFB_7FFF	16 KB	44	Serial Communications Interface (eSCI B)
0xFFFB_8000–0xFFFB_FFFF	32 KB	—	Reserved
0xFFFC_0000–0xFFFC_3FFF	16 KB	1152	Flexible Controller Area Network (FlexCAN A)
0xFFFC_4000–0xFFFC_7FFF	16 KB	1152	Reserved
0xFFFC_8000–0xFFFC_BFFF	16 KB	1152	Flexible Controller Area Network (FlexCAN C)
0xFFFC_C000–0xFFFF_BFFF	192 KB	—	Reserved
0xFFFF_C000–0xFFFF_FFFF	16 KB	4 K	Boot Assist Module (BAM)

[1] If allocated size > used size, then the base address for the block is the lowest address of the listed address range, unless noted otherwise.

[2] It is recommended that the address rage of 0x3000_0000 through 0x3FFF_FFFF be reserved for use by the calibration bus.

1.3.1 External Master Mode Operation Memory Map

When the MPC5534 MCU acts as a slave from an external master device, the External Bus Interface (EBI) translates the 24-bit external address to a 32-bit internal address. Table 1-3 lists the translation parameters.

Table 1-3. External to Internal Memory Map Translation Table for Slave Mode

Ext addr[8:11][1]	Internal addr[0:11]	Size (bytes)	Internal Slave	Internal Address Range
0b0xxx	—	8 MB	—	Reserved for off-chip access
0b10xx	0b0000_0000_00xx	4 MB	Internal Flash	0x0000_0000–0x003F_FFFF
0b1100	0b0100_0000_0000	1 MB	SRAM	0x4000_0000–0x400F_FFFF
0b1101	0b0110_0000_0000	1 MB	Reserved[2]	0x6000_0000–0x600F_FFFF
0b1110	0b1100_0011_1111	1 MB	Bridge A Peripherals	0xC3F0_0000–0xC3FF_FFFF
0b1111	0b1111_1111_1111	1 MB	Bridge B Peripherals	0xFFF0_0000–0xFFFF_FFFF

[1] Only the lower 24 address signals (ADDR[8:31]) are available off-chip.
[2] Reserved for a future block that requires its own crossbar slave port.

Table 1-4 shows the memory map for a MPC5534 MCU acting as a slave from the point of view of the external master.

Table 1-4. MPC5534 Slave Memory Map as seen from an External Master

External Address Range[1]	Bytes	Use
0x00_0000[2]–0x7F_FFFF	8 MB	Used for off-chip memory accesses
0x80_0000–0x8F_FFFF	1 MB	Slave flash[3]
0x90_0000–0xBF_FFFF	3 MB	Reserved
0xC0_0000–0xC0_FFFF	64 KB	Slave SRAM
0xC1_0000–0xCF_FFFF	1 MB–64 KB	Reserved
0xD0_0000–0xCF_FFFF	1 MB	Reserved
0xE0_0000–0xEF_FFFF	1 MB	Slave Bridge A Peripherals
0xF0_0000–0xFF_FFFF	1 MB	Slave Bridge B Peripherals

[1] Only the lower 24 address signals (ADDR[8:31]) are available off-chip.
[2] This address range is not part of the MPC5534 slave memory map. It is shown to illustrate the addressing scheme for off-chip accesses when the MPC5534 is operating in slave mode.
[3] The shadow block of the slave flash is not accessible by an external master

1.4 Detailed Features

The following sections provided detailed information about each of the on-chip modules.

1.4.1 e200z3 Core Overview

The e200z3 processor uses a four-stage pipeline for instruction execution:

- Instruction Fetch (stage 1)
- Instruction Decode/Register file Read/Effective Address Calculation (stage 2)
- Execute/Memory Access (stage 3)
- Register Writeback (stage 4)

The operation of the stages overlap, allowing single-clock instruction execution for most instructions.

The integer execution unit consists of a 32-bit Arithmetic Unit (AU), a Logic Unit (LU), a 32-bit Barrel shifter (Shifter), a Mask-Insertion Unit (MIU), a Condition Register manipulation Unit (CRU), a Count-Leading-Zeros unit (CLZ), a 32x32 Hardware Multiplier array, result feed-forward hardware, and support hardware for division.

Most arithmetic and logical operations are executed in a single cycle with the exception of the divide instructions. A Count-Leading-Zeros unit operates in a single clock cycle. The Instruction Unit contains a PC incrementer and a dedicated Branch Address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching is performed to accelerate taken branches. Prefetched instructions are placed into an instruction buffer capable of holding six instructions.

Branches can also be decoded at the instruction buffer and branch target addresses calculated prior to the branch reaching the instruction decode stage, allowing the branch target to be prefetched early. When a branch is detected at the instruction buffer, a prediction can be made on whether the branch is taken or not. If the branch is predicted to be taken, a target fetch is initiated and its target instructions are placed in the instruction buffer following the branch instruction. Many branches take zero cycles to execute by using branch folding. Branches are folded out from the instruction execution pipe whenever possible. These include unconditional branches and conditional branches with condition codes that can be resolved early.

Conditional branches that are not taken and not folded execute in a single clock. Branches with successful target prefetching that are not folded have an effective execution time of one clock. All other taken branches have an execution time of two clocks. Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The Condition register unit supports the condition register (CR) and condition register operations defined by the PowerPC architecture. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching. Vectored and autovectored interrupts are supported

by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

The hardware floating-point unit uses the IEEE-754 single-precision floating-point format and supports single-precision floating-point operations in a pipelined fashion. The general purpose register file is used for source and destination operands, thus there is a unified storage model for single-precision floating-point data types of 32-bits and the normal integer type. Single-cycle floating-point add, subtract, multiply, compare, and conversion operations are provided. Divide instructions are multi-cycle and are not pipelined.

The Signal Processing Extension (SPE) Auxiliary Processing Unit (APU) provides hardware SIMD operations and supports a full compliment of dual integer arithmetic operation including Multiply Accumulate (MAC) and dual integer multiply (MUL) in a pipelined fashion. The general purpose register file is enhanced such that all 32 of the GPRs are extended to 64 bits wide and are used for source and destination operands, thus there is a unified storage model for 32x32 MAC operations which generate greater than 32 bit results.

The majority of both scalar and vector operations (including MAC and MUL) are executed in a single clock cycle. Both Scalar and Vector divides take multiple clocks. The SPE APU also provides extended load and store operations to support the transfer of data to and from the extended 64 bit GPRs. This SPE APU is fully binary compatible with the e200z6 SPE APU used in MPC5554 and MPC5553.

The CPU includes support for variable length encoding (VLE) instruction enhancements. This allows the optional execution of an alternate instruction set consisting of a mixture of 16-bit and 32-bit instructions. This results in a significantly smaller code size footprint without affecting performance noticeably. The e200z3 core supports both the PowerPC Architecture Book E and VLE instruction sets.

1.4.2 Crossbar Switch (XBAR)

The XBAR multi-port crossbar switch supports simultaneous connections between four master ports and five slave ports.

- Four master ports:
  - Core CPU - Instruction
  - Core CPU - Data
  - eDMA
  - EBI
- Five slave ports
  - Flash (64-bit data access)
  - EBI (16-bit data access)
  - SRAM (32-bit data access)
  - Peripheral Bridge A (32-bit data access)
  - Peripheral Bridge B (32-bit data access)

The crossbar allows for concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port stall until the higher priority master completes its transactions. By default, requesting masters have equal priority and are granted access to a slave port in round-robin fashion, based upon the ID of the last master to be granted access.

1.4.3 Enhanced Direct Memory Access (eDMA) Controller

The enhanced direct memory access (eDMA) controller is a second-generation module capable of performing complex data movements via 32 programmable channels, with minimal intervention from the host processor. The hardware micro architecture includes a DMA engine which performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels. This implementation is utilized to minimize the overall block size.

1.4.4 Interrupt Controller (INTC)

The INTC (interrupt controller) provides priority-based preemptive scheduling of interrupt requests, suitable for statically scheduled hard real-time systems. The INTC allows interrupt request servicing from up to 210 interrupt sources.

For high priority interrupt requests, the time from the assertion of the interrupt request from the peripheral to when the processor is executing the interrupt service routine (ISR) has been minimized. The INTC provides a unique vector for each interrupt request source for quick determination of which ISR needs to be executed. It also provides an ample number of priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. To allow the appropriate priorities for each source of interrupt request, the priority of each interrupt request is software configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks which share the resource can not preempt each other.

Multiple processors can assert interrupt requests to each other through software settable interrupt requests. These same software settable interrupt requests also can be used to break the work involved in servicing an interrupt request into a high priority portion and a low priority portion. The high priority portion is initiated by a peripheral interrupt request, but then the ISR asserts a software settable interrupt request to finish the servicing in a lower priority ISR. Therefore these software settable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

1.4.5 Frequency Modulated Phase-Locked Loop (FMPLL)

The frequency modulated PLL allows you to generate high speed system clocks from an 8–20 MHz crystal oscillator or external clock generator. Further, the FMPLL supports programmable frequency modulation of the system clock. The FMPLL multiplication factor, output clock divider ratio, modulation depth, and modulation rate are all software configurable.

1.4.6 External Bus Interface (EBI)

NOTE

The EBI is available on the 324 package only—the EBI is not available on the 208 package due to pin limitations.

The EBI controls data transfer across the crossbar switch to/from memories or peripherals in the external address space. The EBI includes a memory controller that generates interface signals to support a variety of external memories. The EBI memory controller supports single data rate (SDR) burst mode flash, SRAM, and asynchronous memories. In addition, the EBI supports up to four regions via chip selects (multiplexed with four address bits), along with programmed region-specific attributes.

1.4.7 Calibration Bus Interface (CBI)

NOTE

The 208 package does not have a CBI. The CBI is available only in the 324 package using the 496 assembly.

The CBI controls data transfer across the crossbar switch to/from memories or peripherals connected to the *VertiCal* connector. The CBI is only available when the silicon is packaged in the *VertiCal* calibration system. Although the CBI shares the memory controller and most of the control logic with the EBI, these buses come out on two completely independent sets of pads. The CBI memory controller supports single data rate (SDR) non-burst mode flash, SRAM, and asynchronous memories. In addition, the CBI supports up to three regions via dedicated calibration chip selects (two chip selects multiplexed with two address bits), along with programmed region-specific attributes.

1.4.8 System Integration Unit (SIU)

The SIU controls MCU reset configuration, pad configuration, external interrupt, general purpose I/O (GPIO), internal peripheral multiplexing, and the system reset operation. The reset configuration block contains the external pin boot configuration logic. The pad configuration block controls the static electrical characteristics of I/O pins. The GPIO block provides uniform and discrete input/output control of the I/O pins of the MCU. The reset controller performs reset monitoring of internal and external reset sources, and drives the $\overline{\text{RSTOUT}}$ pin. The SIU is accessed by the e200z3 core through the crossbar switch.

For more information on configuring the MPC5534 at reset see Section 1.5, “Chip Configuration.

1.4.9 On-chip Flash

The MPC5534 provides 1 MB of programmable, non-volatile, flash memory. The non-volatile memory (NVM) can be used for instruction and/or data storage. The flash module interfaces the system bus to a dedicated flash memory array controller. It supports a 64-bit data bus width at the system bus port, and a 128-bit read data interface to flash memory. The module contains a four-entry, 128-bit prefetch buffer and a prefetch controller which prefetches sequential lines of data from the flash array into the buffer. Prefetch buffer hits allow no-wait responses. Normal flash array accesses are registered and are forwarded to the system bus on the following cycle, incurring three wait-states. Prefetch operations can be automatically

controlled, and can be restricted to servicing a single bus master. Prefetches can also be restricted to being triggered for instruction or data accesses.

1.4.10 Static Random Access Memory (SRAM)

The MPC5534 SRAM module provides a general-purpose 64-KB memory block. The first 32 KB block of the SRAM is powered by its own power supply pin, called V_{STBY}. This allows the contents of this memory region to be preserved when the rest of the MCU is powered down.

ECC handling is done on a 32-bit boundary and is completely software compatible with MPC5500 family devices with an e200z6 core and 64-bit wide ECC syndrome. Because the e200z3 core in MPC5534 is a cacheless processor, the platform RAM is organized on a 32-bit boundary versus the 64-bit organization used on other MPC5500 family MCUs based on the e200z6 core.

1.4.11 Boot Assist Module (BAM)

The BAM is a block of read-only memory that is hard-coded by Freescale and is identical for all MPC5500 family MCU's with an e200z*n* core. The BAM program executes when the MCU is powered-on or reset in normal mode. The BAM supports the following modes of booting:

- Booting from internal flash memory
- Single master booting from external memory (for parts in 324 BGA that have an external bus)
- Serial boot loading (a program is downloaded into RAM via eSCI or the FlexCAN and then executed)

The BAM reads the reset configuration half word (RCHW) from flash memory and configures the device. Flash memory can be either internal (208 and 324 packages) or external (324 package only).

For more information on configuring the MPC5534 at reset, see Section 1.5, "Chip Configuration.

1.4.12 Enhanced Module Input/Output System (eMIOS)

The eMIOS module provides the functionality to generate or measure time events. A unified channel (UC) module uses a superset of the functionality of all the eMIOS channels, while providing a consistent user interface. This allows you more flexibility to program each unified channel for different functions to be used in various applications. To identify up to two timed events, configure the UC with two comparators, a time base selector and registers. This can produce match events which can measure or generate a waveform. Alternatively, input events can be used to capture the time base, allowing measurement of an input signal.

1.4.13 Enhanced Time Processing Unit (eTPU)

The eTPU is an enhanced co-processor designed for timing control. Operating in parallel with the host CPU, the eTPU processes instructions and real-time input events, performs output waveform generation, and accesses shared data without host intervention. Consequently, for each timer event, the host CPU setup and service times are minimized or eliminated. A powerful timer subsystem is formed by combining the

eTPU with its own instruction and data RAM. High-level assembler/compiler and documentation allows customers to develop their own functions on the eTPU.

The eTPU is an enhanced version of the TPU module implemented on the MC68332 and MPC500 products. Enhancements of the eTPU include a more powerful processor which handles high-level C code efficiently and allows for more functionality and increased performance. Although there is no compatibility at microcode level, the eTPU maintains several features of older TPU versions and is conceptually almost identical. The eTPU library is a superset of the standard TPU library functions modified to take advantage of enhancements in the eTPU. These, along with a C compiler, make it relatively easy to port previous applications. By providing source code for the Freescale library, the eTPU can support your own function development.

The eTPU includes these distinctive features:

- 32 channels, each channel is associated with one input and one output signal.
  - Enhanced input digital filters on the input pins for improved noise immunity.
  - Identical, orthogonal channels: each channel can perform any time function. Each time function can be assigned to more than one channel as a given time, so each signal can have any functionality.
  - Each channel has an event mechanism which supports single and double action functionality in various combinations. It includes two 24-bit capture registers, two 24-bit match registers, 24-bit greater-equal and equal-only comparators
  - Input and output signal states visible from the host
- Two independent 24-bit time bases for channel synchronization:
  - First time base clocked by system clock with programmable prescale division from 2 to 512 (in steps of 2), or by output of second time base prescaler
  - Both time bases can be exported to the eMIOS timer module
  - Second time base counter can work as a continuous angle counter, enabling angle based applications to match angle instead of time.
  - Both timebases visible from the host
- Event-triggered microengine:
  - Fixed-length instruction execution in two-system-clock microcycle
  - 12 KB of code memory (SCM)
  - 2.5 KB of parameter (data) RAM (SPRAM)
  - Parallel execution of data memory, ALU, channel control and flow control sub-instructions in selected combinations
  - 32-bit microengine registers and 24-bit wide ALU, with 1 microcycle addition and subtraction, absolute value, bitwise logical operations on 24-bit, 16-bit, or byte operands, single bit manipulation, shift operations, sign extension and conditional execution
  - Additional 24 bit Multiply/MAC/Divide unit which supports all signed/unsigned Multiply/MAC combinations, and unsigned 24-bit divide. The MAC/Divide unit works in parallel with the regular microcode commands

- Resource sharing features support channel use of common channel registers, memory and microengine time:
  - Hardware scheduler works as a “task management” unit, dispatching event service routines by pre-defined, host-configured priority.
  - Automatic channel context switch when a "task switch" occurs, i.e., one function thread ends and another begins to service a request from other channel: channel-specific registers, flags and parameter base address are automatically loaded for the next serviced channel
  - SPRAM shared between host CPU and eTPU, supporting communication either between channels and host or inter-channel
  - Dual-parameter coherency hardware support allows atomic access to two parameters by host
- Test and development support features:
  - Nexus Class 3 debug, supporting single-step execution, arbitrary microinstruction execution, hardware breakpoints and watchpoints on several conditions (see Section 1.4.18, “Nexus,” for more details on the Nexus module)
  - Software breakpoints
  - SCM continuous signature-check built-in self test (MISC—multiple input signature calculator), runs concurrently with eTPU normal operation

1.4.14 Enhanced Queued Analog/Digital Converter (eQADC)

The enhanced queued analog to digital converter (eQADC) block provides accurate and fast conversions for a wide range of applications. The eQADC provides a parallel interface to two on-chip analog to digital converters (ADCs), and a single master to single slave serial interface to an off-chip external device. The two on-chip ADCs are designed to allow access to all the analog channels.

The eQADC transfers commands from multiple command FIFOs (CFIFOs) to the on-chip ADCs or to the external device. The block can also receive data from the on-chip ADCs or from an off-chip external device into multiple result FIFOs (RFIFOs) in parallel, independently of the CFIFOs. The eQADC supports software and external hardware triggers from other blocks to initiate transfers of commands from the CFIFOs to the on-chip ADCs or to the external device. It also monitors the fullness of CFIFOs and RFIFOs, and accordingly generates DMA or interrupt requests to control data movement between the FIFOs and the system memory, which is external to the eQADC.

1.4.15 Deserial/Serial Peripheral Interface (DSPI)

The deserial serial peripheral interface (DSPI) block provides a synchronous serial interface for communication between the MPC5534 MCU and external devices. The DSPI supports pin count reduction through serialization and deserialization of eTPU channels and memory-mapped registers. The channels and register content are transmitted using a SPI-like protocol. There are two identical DSPI blocks on the MPC5534 MCU.

The DSPIs have three configurations:

- Serial peripheral interface (SPI) configuration where the DSPI operates as a SPI with support for queues

- Deserial serial interface (DSI) configuration where the DSPI serializes eTPU and eMIOS output channels and deserializes the received data by placing it on the eTPU and eMIOS input channels
- Combined serial interface (CSI) configuration where the DSPI operates in both SPI and DSI configurations interleaving DSI frames with SPI frames, giving priority to SPI frames

For queued operations, the SPI queues reside in system memory external to the DSPI. Data transfers between the memory and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software.

1.4.16 Enhanced System Communications Interface (eSCI)

The enhanced serial communications interface (eSCI) allows asynchronous serial communications with peripheral devices and other MCUs. It includes special support to interface to local interconnect network (LIN) slave devices.

1.4.17 Flexible Controller Area Network (FlexCAN)

The MPC5534 MCU contains two controller area network (FlexCAN) blocks. Each FlexCAN module is a communication controller implementing the CAN protocol according to Bosch Specification version 2.0B. The CAN protocol was designed to be used primarily as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. Each FlexCAN module has 64 message buffers (MB).

1.4.18 Nexus

The NDI (Nexus Debug Interface) block provides real-time development support capabilities for the MPC5534 MCU in compliance with the IEEE-ISTO 5001-2003 standard. This development support is supplied for MCUs without requiring external address and data pins for internal visibility. The NDI block is an integration of several individual Nexus blocks that are selected to provide the development support interface for the MPC5534. The NDI block interfaces to the host processor, eTPU, and internal buses to provide development support as per the IEEE-ISTO 5001-2003 standard. The development support provided includes program trace, data trace, watchpoint trace, ownership trace, run-time access to the MCUs internal memory map and access to the core and eTPU internal registers during halt. The Nexus interface also supports a JTAG only mode using only the JTAG pins. The following features are implemented:

- 15 or 23 full duplex pin interface for medium and high visibility throughput:
  - One of two modes selected by register configuration: full port mode (FPM) and reduced port mode (RPM): FPM comprises 12 MDO pins (324 package only); RPM comprises four MDO pins (208 and 324 packages).
  - Auxiliary output port
- One MCKO (message clock out) pin
- Four or 12 MDO (message data out) pins
- Two $\overline{\text{MSEO}}$ (message start/end out) pins
- One $\overline{\text{RDY}}$ (ready) pin

- One $\overline{\text{EVTO}}$ (event out) pin
  - Auxiliary input port
- One $\overline{\text{EVTI}}$ (event in) pin
- Five pin JTAG port (JCOMP, TDI, TDO, TMS, and TCK)
  - Supports JTAG mode
- Host processor (e200) development support features
  - IEEE-ISTO 5001-2003 standard Class 3 compliant
  - Data trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tool to trace reads and/or writes to selected internal memory resources.
  - Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing development tools to trace ownership flow.
  - Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct branches, indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code can be traced.
  - Watchpoint messaging (WPM) via the auxiliary port
  - Watchpoint trigger enable of program and/or data trace messaging
  - Data tracing of instruction fetches via private opcodes
  - Subset of PowerPC Architecture Book E software debug facilities with OnCE block (Nexus Class 1 features implemented by OnCE)
- eTPU development support features
  - IEEE-ISTO 5001-2003 standard Class 3 compliant for the eTPU
  - Data trace via data write messaging and data read messaging. This allows the development tool to trace reads and writes to selected shared parameter RAM (SPRAM) address ranges. Four data trace windows are available.
  - Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which channel is being serviced. An ownership trace message is transmitted to indicate when a new channel service request is scheduled, allowing the development tools to trace task flow. A special OTM is sent when the engine enters in idle state, meaning that all requests were serviced and no new requests are yet scheduled.
  - Program trace via branch trace messaging. BTM displays program flow discontinuities (start, repeat, jump, return, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code can be traced. The branch trace messaging method uses the branch/predicate method to reduce the number of generated messages.
  - Watchpoint messaging via the auxiliary port. WPM provides visibility of the occurrence of the eTPUs' watchpoints and breakpoints.
  - Nexus based breakpoint/watchpoint configuration and single step support.

- Run-time access to the on-chip memory map via the Nexus read/write access protocol. This feature supports accesses for run-time internal visibility, calibration variable acquisition, calibration constant tuning, and external rapid prototyping for powertrain automotive development systems.
- All features are independently configurable and controllable via the IEEE 1149.1 I/O port.
- The NDI block reset is controlled with JCOMP, power-on reset, and the TAP state machine. All these sources are independent of system reset.
- Power-on-reset status indication during reset via MDO[0] in disabled and reset modes

1.4.19 JTAG

The JTAGC (JTAG Controller) block provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. All data input to and output from the JTAGC block is communicated in serial format. The JTAGC block is compliant with the IEEE 1149.1-2001 standard.

1.5 Chip Configuration

Various functions of the MPC5534 can be implemented at reset, and the following operations can be configured:

- Boot mode
  - Internal memory boot (default)
  - External memory boot (single master—324 package only)
  - Boot from serial port (FlexCAN or eSCI)
- PLL mode
  - Normal mode with crystal reference (default)
  - Normal mode with external reference
  - Bypass mode
- Watchdog timer enable

1.6 Related Documentation

Table 1-5 lists other documents that provide information related to the MPC5534 and its development support tools. Documentation is available from a local Freescale distributor, a Freescale semiconductor sales office, the Freescale Literature Distribution Center, or through the Freescale world-wide web address at http://www.freescale.com.

Table 1-5. MPC5534 and Related Documentation

Freescale Document Number	Title
MPC5534RM	MPC5534 Reference Manual
AN2127/D	EMC Guidelines for MPC500-Based Automotive Powertrain Systems
AN1259/D	System Design and Layout Techniques for Noise Reduction in MCU-Based Systems
AN2613	MPC5554 Minimum Board Configuration
AN2614	Nexus Interface Options for the MPC5500 Family
AN2706	EMC Guidelines for MPC5500-based Systems

Chapter 2
Signals

This chapter describes the external device signals, including a table of signal properties, detailed descriptions of the available signals, and the I/O pin power/ground segmentation.

2.1 Block Diagram

This chapter describes the signals of the MPC5534 that connect off chip. It includes a table of signal properties, detailed descriptions of signals, and connections and serialization for the eTPU and eMIOS.

Figure 2-1 shows the signals that are available on the device in the 324 package. Signals designated in **red** are not available due to pin limitations on the 208 package. Signals shown in **blue** are primary functions that are not designed into this device.

Read the Package columns in Table 2-1 for the alpha/numeric code that identifies each ball for one set of muxed signals. You can cross reference this column to the BGA package figures in the Data Sheet to identify the ball location on the device. The VertiCal assembly assignments are also listed in the table.

NOTE

The VertiCal assembly has ball connections for all the *available* signals on the device.

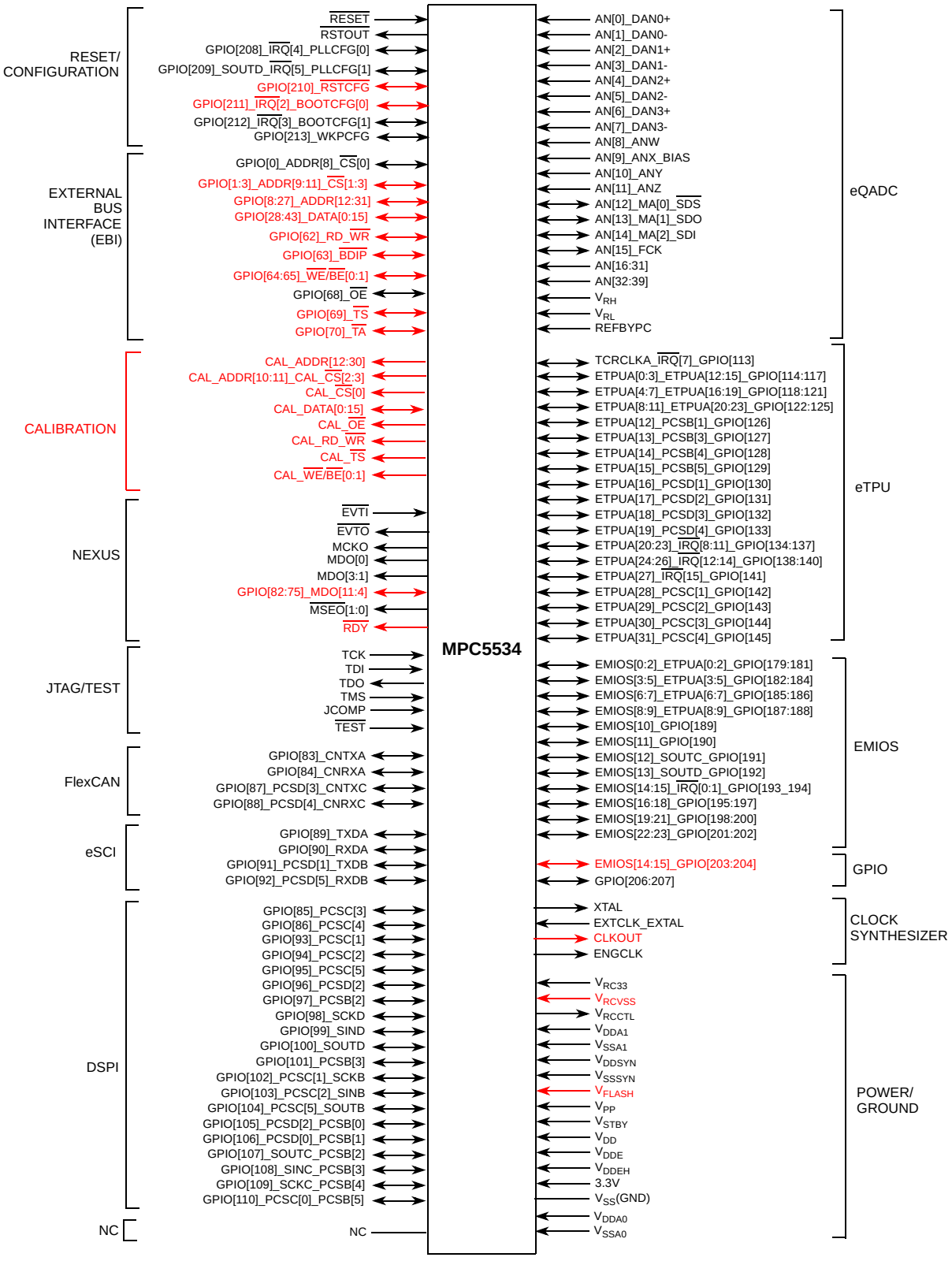


Figure 2-1. MPC5534 Signal Diagram

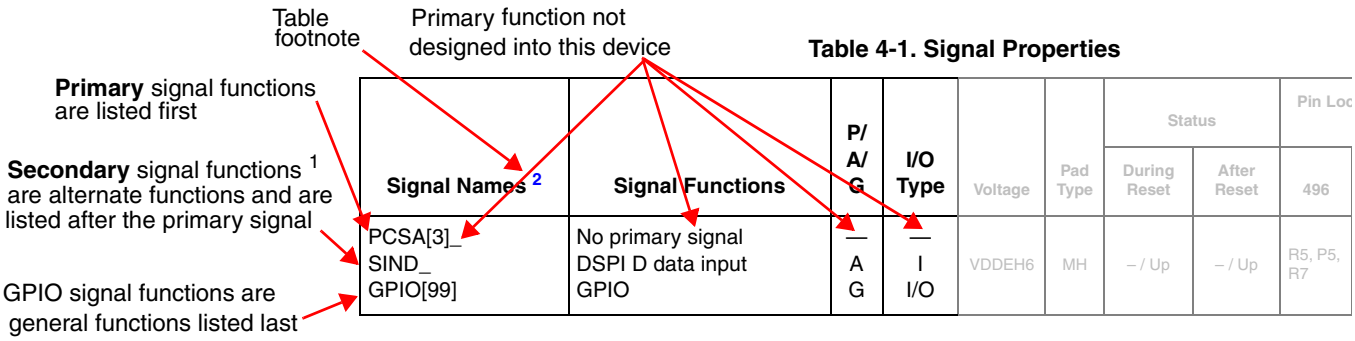
2.2 External Signal Descriptions

This section summarizes the external signal functions, the electrical characteristics, and pad configuration settings for this device. The signal properties and electrical characteristics are set in the System Integration Unit (SIU) Pad Configuration (PCR) registers.

2.2.1 Multiplexed Signals

Signal functions are multiplexed to each ball on the BGA in a function hierarchy: Primary, Main Primary, Alternate, Second Alternate, and General Purpose Input/Output (I/O). For example, in the signal PCSA[3]_SIND_GPIO[99], the primary signal function is PCSA[3], the first alternate signal function is SIND, and the GPIO function is a generic General Purpose I/O signal. Multiplexing signal functions allows for more flexibility when configuring the device, as well as providing compatibility with other devices in the MPC5500 product family.

The primary signal function name is used in the Ball Grid Array (BGA) map to identify the location of the ball, however, the primary signal function is not always valid for all devices. As shown in Figure 2-2, when the primary signal function is not available on the device, the Signal Name column contains the primary signal function name, ‘No primary signal’ is shown in the Signal Function column, and a dash appears in the P/A/G, and I/O Type columns.

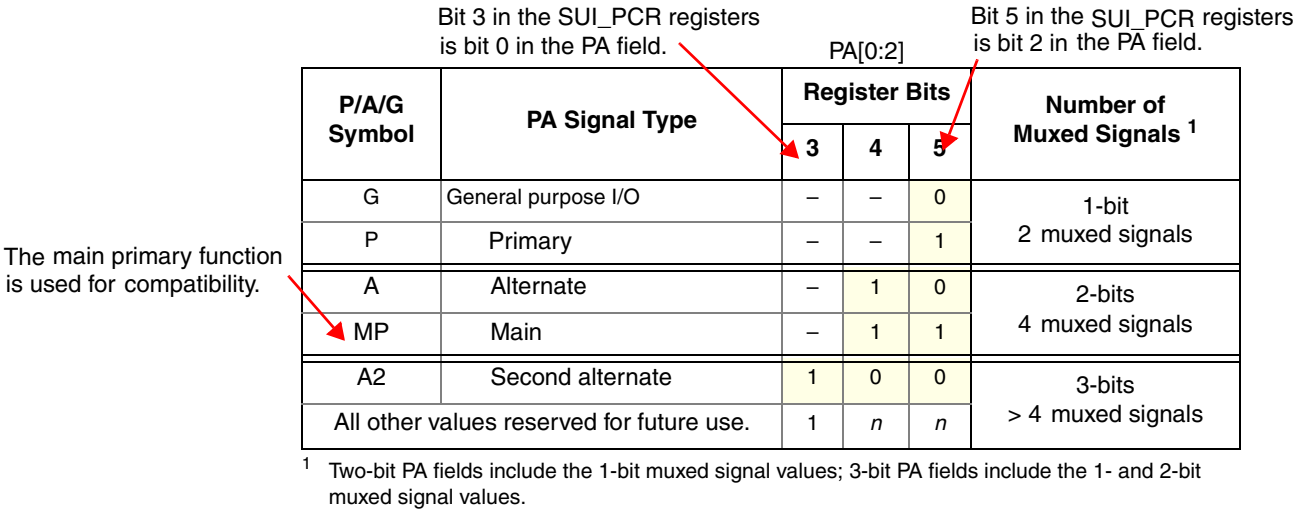


[1] Some devices have two alternate signals muxed to the same ball: first alternate (A); second alternate (A2).

Figure 2-2. Primary Function Not Available on Device

The entries in the P/A/G column designate the position in the signal function hierarchy for multiplexed functions. These symbols correspond to binary values for the Pin Assignment (PA) field in the SIU_PCR registers that determine the active signal function. The PA field length varies from 1- to 3-bits, depending on the PCR register.

Figure 2-3 explains the symbol definitions used in the P/A/G column for Table 2-1.



P/A/G Symbol	PA Signal Type	Register Bits 3	4	5	Number of Muxed Signals [1]
G	General purpose I/O	–	–	0	1-bit 2 muxed signals
P	Primary	–	–	1	
A	Alternate	–	1	0	2-bits 4 muxed signals
MP	Main	–	1	1	
A2	Second alternate	1	0	0	3-bits > 4 muxed signals
All other values reserved for future use.		1	*n*	*n*	

[1] Two-bit PA fields include the 1-bit muxed signal values; 3-bit PA fields include the 1- and 2-bit muxed signal values.

Figure 2-3. Understanding the P/A/G Column Entries

2.2.2 Device Signals Summary

Table 2-1 gives a summary of the device's external signals and properties. See Section 2.3, "Detailed Signal Description," for detailed descriptions of each signal. The signals shown in **red** are not available on the 208 package; signals shown in **blue** are primary functions that are not available in this device.

Table 2-1. MPC5534 Signal Properties

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status During Reset[5]	Status After Reset[6]	Package 208	Package 324	496[4] VertiCal
Reset / Configuration										
$\overline{\text{RESET}}$	External reset input	P	I	V_{DDEH6}	SH	$\overline{\text{RESET}}$ / Up	$\overline{\text{RESET}}$ / Up	L16	R22	AA27
$\overline{\text{RSTOUT}}$	External reset output	P	O	V_{DDEH6}	SH	$\overline{\text{RSTOUT}}$ / Low	$\overline{\text{RSTOUT}}$ / High	K15	P21	W26
PLLCFG[0]_ $\overline{\text{IRQ}}$[4]_ GPIO[208]	PLLMRFM mode selection External interrupt request GPIO	P A G	I I I/O	V_{DDEH6}	MH	PLLCFG / Up	– / Up	V21	V21	AB27
PLLCFG[1]_ $\overline{\text{IRQ}}$[5]_ SOUTD_ GPIO[209]	PLLMRFM reference selection External interrupt request DSPI D data output GPIO	P A A2 G	I I O I/O	V_{DDEH6}	MH	PLLCFG / Up	– / Up	N15	U20	AA26
$\overline{\text{RSTCFG}}$_[7] GPIO[210]	Reset configuration input GPIO	P G	I I/O	V_{DDEH6}	SH	$\overline{\text{RSTCFG}}$ / Up	– / Up	—	P22	Y28
BOOTCFG[0]_[7] $\overline{\text{IRQ}}$[2]_ GPIO[211]	Boot configuration input External interrupt request GPIO	P A G	I I I/O	V_{DDEH6}	SH	BOOTCFG / Down	– / Down	—	U21	AB26

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
BOOTCFG[1]_ IRQ[3]_ GPIO[212]	Boot configuration input External interrupt request GPIO	P A G	I I I/O	V_{DDEH6}	SH	BOOTCFG / Down	– / Down	M15	T20	AB24
WKPCFG_ GPIO[213]	Weak pull configuration input GPIO	P G	I I/O	V_{DDEH6}	SH	WKPCFG / Up	– / Up	L15	R19	AA24
External Bus Interface (EBI) [8]										
CS[0]_ ADDR[8]_ [9] GPIO[0]	External chip selects External address bus GPIO	P A G	O I/O I/O	V_{DDE2}	F	– / Up	– / Up[10]	R1	M4	T7
CS[1:3]_ [11] ADDR[9:11]_[9] GPIO[1:3]	External chip selects External address bus GPIO	P A G	O I/O I/O	V_{DDE2}	F	– / Up	– / Up[10]	—	M3, N2, N1	R5, P5, R7
ADDR[12:26]_[9, 11] GPIO[8:22]	External address bus GPIO	P G	I/O I/O	V_{DDE2}	F	– / Up	– / Up[10]	—	T3, U3, U4, V3, P1, P2, R1, R2, T1, T2, U1, U2, V1, V2, W1	Y7, AC3, AC5, AB5, T3, T2, T1, V2, W1, W2, Y1, Y2, AA2, AB2, AC2
ADDR[27:29]_[9, 11] GPIO[23:25]	External address bus GPIO	P G	I/O I/O	V_{DDE2}	F	– / Up	– / Up[10]	—	Y2, Y1, AA1	AD2, AD3, AD1
ADDR[30:31]_[9, 11] GPIO[26:27]	External address bus GPIO	P G	I/O I/O	V_{DDE2}	F	– / Up	– / Up[10]	—	W3, V4	AF2, AE3
DATA[0:15]_[9, 11] GPIO[28:43]	External data bus GPIO	P G	I/O I/O	V_{DDE3}	F	– / Up	– / Up[10]	—	AB4, AA5, AB5, AB6, AB7, AA8, AB8, AA9, Y6, Y7, Y8, W9, W10, Y10, W11, Y11	AG11, AF12, AG13, AH13, AG14, AH15, AG15, AH16, AB12, AF10, AD13, AF11, AB15, AD12, AD15, AF13
RD_WR_ [11] GPIO[62]	External read/write GPIO	P G	I/O I/O	V_{DDE2}	F	– / Up	– / Up[10]	—	P3	U3
BDIP_ [11] GPIO[63]	External burst data in progress GPIO	P G	O I/O	V_{DDE2}	F	– / Up	– / Up[10]	—	M1	N1
WE/BE[0:1]_ [11, 12] GPIO[64:65]	External write/byte enable GPIO	P G	O I/O	V_{DDE2}	F	– / Up	– / Up[10]	—	N4, N3	U5, T5

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
$\overline{OE}$_ GPIO[68]	External output enable GPIO	P G	O I/O	V_{DDE3}	F	– / Up	– / Up[10]	T3	AB9	AF16
$\overline{TS}$_ [11] GPIO[69]	External transfer start GPIO	P G	I/O I/O	V_{DDE2}	F	– / Up	– / Up[10]	—	T4	W3
$\overline{TA}$_ [11] GPIO[70]	External transfer acknowledge GPIO	P G	I/O I/O	V_{DDE2}	F	– / Up	– / Up[10]	—	R4	V3
Calibration Bus										
CAL_$\overline{CS}$[0] [13]	Calibration chip select	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	AB11
CAL_$\overline{CS}$[2]_ [13] CAL_ADDR[10]	Calibration chip select Calibration address bus	P A	O	V_{DDE12}	F	– / Up	– / Up	—	—	AA10
CAL_$\overline{CS}$[3]_ [13] CAL_ADDR[11]	Calibration chip select Calibration address bus	P A	O	V_{DDE12}	F	– / Up	– / Up	—	—	AB10
CAL_ADDR[12] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	AA14
CAL_ADDR[13] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	R8
CAL_ADDR[14] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	AA15
CAL_ADDR[15] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	W7
CAL_ADDR[16] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	P7
CAL_ADDR[17] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	P8
CAL_ADDR[18] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	U7
CAL_ADDR[19] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	N7
CAL_ADDR[20] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	M8
CAL_ADDR[21] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	M7
CAL_ADDR[22] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	V7
CAL_ADDR[23] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	L8
CAL_ADDR[24] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	T8
CAL_ADDR[25] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	K8
CAL_ADDR[26] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	L7
CAL_ADDR[27] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	U8
CAL_ADDR[28] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	V8
CAL_ADDR[29] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	AB13
CAL_ADDR[30] [13]	Calibration address bus	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	AB14
CAL_DATA[0:7] [13]	Calibration data bus	P	I/O	V_{DDE12}	F	– / Up	– / Up	—	—	W21,Y22, V21, W22, U21, U22, T21, T22

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
CAL_DATA[8:15] [13]	Calibration data bus	P	I/O	V_{DDE12}	F	– / Up	– / Up	—	—	AA17, AB16, AA18, AB17, AA19, AB19, AA20, AB20
CAL_RD_$\overline{WR}$ [13]	Calibration read/write	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	W8
CAL_$\overline{WE}$/$\overline{BE}$[0:1] [13]	Calibration write/byte enable	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	Y8, AA7
CAL_$\overline{OE}$ [13]	Calibration output enable	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	AD16
CAL_$\overline{TS}$ [13]	Calibration transfer start	P	O	V_{DDE12}	F	– / Up	– / Up	—	—	AA11
NEXUS										
$\overline{EVTI}$	Nexus event in	P	I	V_{DDE7}	F	I / Up	$\overline{EVTI}$ / Up	E15	F21	G26
$\overline{EVTO}$	Nexus event out	P	O	V_{DDE7}	F	O / Low	$\overline{EVTO}$ / High	D15	F22	G27
MCKO	Nexus message clock out	P	O	V_{DDE7}	F	O / Low	MCKO / Enabled[14]	F15	G20	H26
MDO[0][15]	Nexus message data out	P	O	V_{DDE7}	F	O / High	MDO / Low	A14	B20	C25
MDO[3:1]	Nexus message data out	P	O	V_{DDE7}	F	O / Low	MDO / Low	B13, A13, B14	D18, C18, C19	C23, B21, C24
MDO[11:4]_ GPIO[82:75][16]	Nexus message data out GPIO	P G	O I/O	V_{DDE7}	F	O / Low	– / Down	—	A17:18, B17, A19, B18, D17, C17, B19	A23, C22, A20, A24, B23, B20, C20, B24
$\overline{MSEO}$[1:0]	Nexus message start/end out	P	O	V_{DDE7}	F	O / High	$\overline{MSEO}$ / High	E16, C15	G22, G21	G24, H24
$\overline{RDY}$	Nexus ready output	P	O	V_{DDE7}	F	O / High	$\overline{RDY}$ / High	—	G19	J24
JTAG / TEST										
TCK	JTAG test clock input	P	I	V_{DDE7}	F	TCK / Down	TCK / Down	C16	D21	E27
TDI	JTAG test data input	P	I	V_{DDE7}	F	TDI / Up	TDI / Up	E14	D22	E28
TDO	JTAG test data output	P	O	V_{DDE7}	F	TDO / Up	TDO / Up	F14	E21	F27
TMS	JTAG test mode select input	P	I	V_{DDE7}	F	TMS / Up	TMS / Up	D14	E20	E26
JCOMP	JTAG TAP controller enable	P	I	V_{DDE7}	F	JCOMP / Down	JCOMP / Down	F16	F20	F26
$\overline{TEST}$	Test mode select	P	I	V_{DDE7}	F	$\overline{TEST}$ / Up	$\overline{TEST}$ / Up	D16	E22	F28

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
FlexCAN										
CNTXA_ [17] GPIO[83]	FlexCAN A transmit GPIO	P G	O I/O	V_{DDEH4}	SH	I / Up	– / Up	P12	Y17	AF22
CNRXA_ [17] GPIO[84]	FlexCAN A receive GPIO	P G	I I/O	V_{DDEH4}	SH	– / Up	– / Up	R12	AA18	AG22
CNTXC_ PCSD[3]_ GPIO[87]	FlexCAN C transmit DSPI D peripheral chip select GPIO	P A G	O O I/O	V_{DDEH6}	MH	– / Up	– / Up	K13	P19	W24
CNRXC_ PCSD[4]_ GPIO[88]	FlexCAN C receive DSPI D peripheral chip select GPIO	P A G	I O I/O	V_{DDEH6}	MH	– / Up	– / Up	L14	R20	Y26
eSCI										
TXDA_ GPIO[89]	eSCI A transmit GPIO	P G	O I/O	V_{DDEH6}	SH	– / Up	– / Up	J14	N20	V24
RXDA_ GPIO[90]	eSCI A receive GPIO	P G	I I/O	V_{DDEH6}	SH	– / Up	– / Up	K14	P20	U26
TXDB_ PCSD[1]_ GPIO[91]	eSCI B transmit DSPI D peripheral chip select GPIO	P A G	O O I/O	V_{DDEH6}	MH	– / Up	– / Up	L13	R21	Y27
RXDB_ PCSD[5]_ GPIO[92]	eSCI B receive DSPI D peripheral chip select GPIO	P A G	I O I/O	V_{DDEH6}	MH	– / Up	– / Up	M13	T19	Y24
DSPI										
CNTXB [18] PCSC[3]_ GPIO[85]	No primary signal DSPI C peripheral chip select GPIO	— A G	— O I/O	V_{DDEH4}	MH	– / Up	– / Up	T12	AB18	AG23
CNRXB [18] PCSC[4]_ GPIO[86]	No primary signal DSPI C peripheral chip select GPIO	— A G	— O I/O	V_{DDEH4}	MH	– / Up	– / Up	R13	AB19	AH23
SCKA [18] PCSC[1]_ GPIO[93]	No primary signal DSPI C peripheral chip select GPIO	— A G	— O I/O	V_{DDEH6}	MH	– / Up	– / Up	—	L22	U27
SINA [18] PCSC[2]_ GPIO[94]	No primary signal DSPI C peripheral chip select GPIO	— A G	— O I/O	V_{DDEH6}	MH	– / Up	– / Up	—	L21	P27
SOUTA[18] PCSC[5]_ GPIO[95]	No primary signal DSPI C peripheral chip select GPIO	— A G	— O I/O	V_{DDEH6}	MH	– / Up	– / Up	—	L20	P24
PCSA[0] [18] PCSD[2]_ GPIO[96]	No primary signal DSPI D peripheral chip select GPIO	— A G	— O I/O	V_{DDEH6}	MH	– / Up	– / Up	—	M20	R24

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
PCSA[1][18] PCSB[2]_ GPIO[97]	No primary signal DSPI B peripheral chip select GPIO	— A G	— O I/O	V_{DDEH6}	MH	– / Up	– / Up	—	M19	T24
PCSA[2]_ [18] SCKD_ GPIO[98]	No primary signal DSPI D clock GPIO	— A G	— I/O I/O	V_{DDEH6}	MH	– / Up	– / Up	J15	M21	N26
PCSA[3] _[18] SIND_ GPIO[99]	No primary signal DSPI D data input GPIO	— A G	— I I/O	V_{DDEH6}	MH	– / Up	– / Up	H13	K19	N24
PCSA[4]_[18] SOUTD_ GPIO[100]	No primary signal DSPI D data output GPIO	— A G	— O I/O	V_{DDEH6}	MH	– / Up	– / Up	—	N19	U24
PCSA[5]_ [18] PCSB[3]_ GPIO[101]	No primary signal DSPI B peripheral chip select GPIO	— A G	— O I/O	V_{DDEH6}	MH	– / Up	– / Up	—	N21	T26
SCKB_[19] PCSC[1]_ GPIO[102]	DSPI B clock DSPI C peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH10}	MH	– / Up	– / Up	J16	K21	T27
SINB_[19] PCSC[2]_ GPIO[103]	DSPI B data input DSPI C peripheral chip select GPIO	P A G	I O I/O	V_{DDEH10}	MH	– / Up	– / Up	G15	H22	P28
SOUTB_[19] PCSC[5]_ GPIO[104]	DSPI B data output DSPI C peripheral chip select GPIO	P A G	O O I/O	V_{DDEH10}	MH	– / Up	– / Up	G13	J19	N28
PCSB[0]_[19] PCSD[2]_ GPIO[105]	DSPI B peripheral chip select DSPI D peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH10}	MH	– / Up	– / Up	G16	J21	R27
PCSB[1]_[19] PCSD[0]_ GPIO[106]	DSPI B peripheral chip select DSPI D peripheral chip select GPIO	P A G	O I/O I/O	V_{DDEH10}	MH	– / Up	– / Up	H16	J22	R28
PCSB[2]_[19] SOUTC_ GPIO[107]	DSPI B peripheral chip select DSPI C data output GPIO	P A G	O O I/O	V_{DDEH10}	MH	– / Up	– / Up	H15	K22	T28
PCSB[3]_[19] SINC_ GPIO[108]	DSPI B peripheral chip select DSPI C data input GPIO	P A G	O I I/O	V_{DDEH6}	MH	– / Up	– / Up	G14	J20	M27
PCSB[4]_[19] SCKC_ GPIO[109]	DSPI B peripheral chip select DSPI C clock GPIO	P A G	O I/O I/O	V_{DDEH6}	MH	– / Up	– / Up	H14	K20	N27
PCSB[5]_[19] PCSC[0]_ GPIO[110]	DSPI B peripheral chip select DSPI C peripheral chip select GPIO	P A G	O I/O I/O	V_{DDEH6}	MH	– / Up	– / Up	J13	L19	M26

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
eQADC										
AN[0]_[20] DAN0+	Single-ended analog input Positive terminal differential input	P	I	V_{DDA1}	AE	I / –	AN[0] / –	B5	B8	C9
AN[1]_[20] DAN0–	Single-ended analog input Negative terminal differential input	P	I	V_{DDA1}	AE	I / –	AN[1] / –	A6	A8	B8
AN[2]_[20] DAN1+	Single-ended analog input Positive terminal differential input	P	I	V_{DDA1}	AE	I / –	AN[2] / –	D6	D10	G12
AN[3]_[20] DAN1–	Single-ended analog input Negative terminal differential input	P	I	V_{DDA1}	AE	I / –	AN[3] / –	C7	C9	E10
AN[4]_[20] DAN2+	Single-ended analog input Positive terminal differential input	P	I	V_{DDA1}	AE	I / –	AN[4] / –	B6	B9	C10
AN[5]_[20] DAN2–	Single-ended analog input Negative terminal differential input	P	I	V_{DDA1}	AE	I / –	AN[5] / –	A7	A9	B9
AN[6]_[20] DAN3+	Single-ended analog input Positive terminal differential input	P	I	V_{DDA1}	AE	I / –	AN[6] / –	D7	D11	G13
AN[7]_[20] DAN3–	Single-ended analog input Negative terminal differential input	P	I	V_{DDA1}	AE	I / –	AN[7] / –	C8	C10	E11
AN[8]_[20] ANW	Single-ended analog input External multiplexed analog input	P	I	V_{DDA1}	AE	I / –	AN[8] / –	—	C5	E7
AN[9]_[20] ANX	Single-ended analog input External multiplexed analog input	P	I	V_{DDA1}	AE	I / –	AN[9] / –	A2	D7	C4
AN[10]_[20] ANY	Single-ended analog input External multiplexed analog input	P	I	V_{DDA1}	AE	I / –	AN[10] / –	—	D8	E6
AN[11]_[20] ANZ	Single-ended analog input External multiplexed analog input	P	I	V_{DDA1}	AE	I / –	AN[11] / –	A3	A5	B6
AN[12]_[21] MA[0]_ $\overline{SDS}$ [22]	Single-ended analog input Mux address eQADC serial data strobe	MP A G	I O O	V_{DDEH9}	MH, A[23]	I / –	AN[12] / –	A12	A16	H15
AN[13]_[21] MA[1]_ SDO [22]	Single-ended analog input Mux address eQADC serial data out	MP A G	I O O	V_{DDEH9}	MH, A[23]	I / –	AN[13] / –	B12	B16	G15
AN[14]_ [21] MA[2]_ SDI [22]	Single-ended analog input Mux address eQADC serial data in	MP A G	I O I	V_{DDEH9}	MH, A[23]	I / –	AN[14] / –	C12	C16	E16
AN[15]_[21] FCK [22]	Single-ended analog input eQADC free running clock	MP G	I O	V_{DDEH9}	MH, A[23]	I / –	AN[15] / –	C13	D16	C16
AN[16:18]	Single-ended analog input	P	I	V_{DDA1}[30]	AE	I / –	AN[16:18] / –	C6, C4, D5	B7, C6, D9	B7, E8, H12
AN[19:20]	Single-ended analog input	P	I	V_{DDA1}[30]	AE	I / –	AN[19:20] / –	—	B6, C7	C7, C8
AN[21]	Single-ended analog input	P	I	V_{DDA1}[30]	AE	I / –	AN[21] / –	B4	C8	E9

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
AN[22:25]	Single-ended analog input	P	I	V_{DDA0}[30]	AE	I / –	AN[22:25] / –	B8, C9, D8, B9	C11, B11, D12, C12	C11, B11, H13, E12
AN[26]	Single-ended analog input	P	I	V_{DDA0}[30]	AE	I / –	AN[26] / –	—	B12	C12
AN[27:28]	Single-ended analog input	P	I	V_{DDA0}[30]	AE	I / –	AN[27:28] / –	A10, B10	A12, A13	B12, A13
AN[29]	Single-ended analog input	P	I	V_{DDA0}[30]	AE	I / –	AN[29] / –	—	D13	E13
AN[30:35]	Single-ended analog input	P	I	V_{DDA0}[30]	AE	I / –	AN[30:35] / –	D9, D10, C10, C11, C5, D11	C13, B13, B14, C14, D14, A14	C13, B13, B14, E14, G14, A14
AN[36:39]	Single-ended analog input	P	I	V_{DDA1}[30]	AE	I / –	AN[36:39] / –	F4, E3, B3, D2	B4, A4, D6, B5	C5, B5, B4, C6
V_{RH}	Voltage reference high	P	I	–	V_{DDINT}	– / –	V_{RH}	A8	A10	A9
V_{RL}	Voltage reference low	P	I	–	V_{SSINT}	– / –	V_{RL}	A9	A11	A10
REFBYPC	Reference bypass capacitor input	P	I	–	V_{DDINT}	– / –	REFBYPC	B7	B10	B10
eTPU										
TCRCLKA_ $\overline{IRQ}$[7]_ GPIO[113]	eTPU A TCR clock External interrupt request GPIO	P A G	I I I/O	V_{DDEH1}	SH	– / Up	– / Up	L4	M2	N5
ETPUA[0:3]_ ETPUA[12:15]_ GPIO[114:117]	eTPU A channel eTPU A channels (output only) GPIO	P A G	I/O O I/O	V_{DDEH1}	SH	– / WKPCFG	– / WKPCFG	N3, M3, P2, P1	L3, L4, K3, L2	M5, G8, M3, L3
ETPUA[4]_ ETPUA[16]_ GPIO[118]	eTPU A channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	N2	L1	L2
ETPUA[5]_ ETPUA[17]_ GPIO[119]	eTPU A channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	M4	K4	H9
ETPUA[6]_ ETPUA[18]_ GPIO[120]	eTPU A channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH1}	SH	– / WKPCFG	– / WKPCFG	L3	J3	M2
ETPUA[7]_ ETPUA[19]_ GPIO[121]	eTPU A channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH1}	SH	– / WKPCFG	– / WKPCFG	K3	K2	K3
ETPUA[8]_ ETPUA[20]_ GPIO[122]	eTPU A channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH1}	SH	– / WKPCFG	– / WKPCFG	N1	K1	K2
ETPUA[9]_ ETPUA[21]_ GPIO[123]	eTPU A channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH1}	SH	– / WKPCFG	– / WKPCFG	M2	J4	G9

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
ETPUA[10]_ ETPUA[22]_ GPIO[124]	eTPU A channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH1}	SH	– / WKPCFG	– / WKPCFG	M1	H3	L5
ETPUA[11]_ ETPUA[23]_ GPIO[125]	eTPU A channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH1}	SH	– / WKPCFG	– / WKPCFG	L2	J2	J3
ETPUA[12]_ PCSB[1]_ GPIO[126]	eTPU A channel DSPI B peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	L1	J1	J2
ETPUA[13]_ PCSB[3]_ GPIO[127]	eTPU A channel DSPI B peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	J4	G4	G10
ETPUA[14]_ PCSB[4]_ GPIO[128]	eTPU A channel DSPI B peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	J3	G3	K5
ETPUA[15]_ PCSB[5]_ GPIO[129]	eTPU A channel DSPI B peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	K2	H2	H3
ETPUA[16]_ PCSD[1]_ GPIO[130]	eTPU A channel DSPI D peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	K1	H1	K1
ETPUA[17]_ PCSD[2]_ GPIO[131]	eTPU A channel DSPI D peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	H3	F3	H10
ETPUA[18]_ PCSD[3]_ GPIO[132]	eTPU A channel DSPI D peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	H4	F4	J5
ETPUA[19]_ PCSD[4]_ GPIO[133]	eTPU A channel DSPI D peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	J2	G2	G3
ETPUA[20] $\overline{IRQ}$[8] GPIO[134]	eTPU A channel External interrupt request GPIO	P A G	I/O I I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	J1	G1	J1
ETPUA[21] $\overline{IRQ}$[9] GPIO[135]	eTPU A channel External interrupt request GPIO	P A G	I/O I I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	G4	E4	H11
ETPUA[22] $\overline{IRQ}$[10] GPIO[136]	eTPU A channel External interrupt request GPIO	P A G	I/O I I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	H2	F2	F3
ETPUA[23] $\overline{IRQ}$[11] GPIO[137]	eTPU A channel External interrupt request GPIO	P A G	I/O I I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	H1	F1	H2

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
ETPUA[24:26]_ IRQ[12:14]_ GPIO[138:140]	eTPU A channel (output only) External Interrupt Request GPIO	P A G	O I I/O	V_{DDEH1}	SH	– / WKPCFG	– / WKPCFG	G1, G3, F3	E1, E3, D3	G2, H5, G5
ETPUA[27]_ IRQ[15]_ GPIO[141]	eTPU A channel (output only) External Interrupt Request GPIO	P A G	O I I/O	V_{DDEH1}	SH	– / WKPCFG	– / WKPCFG	G2	E2	E3
ETPUA[28]_ PCSC[1]_ GPIO[142]	eTPU A channel (output only) DSPI C peripheral chip select GPIO	P A G	O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	F1	D1	F1
ETPUA[29]_ PCSC[2]_ GPIO[143]	eTPU A channel (output only) DSPI C peripheral chip select GPIO	P A G	O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	F2	D2	F2
ETPUA[30]_ PCSC[3]_ GPIO[144]	eTPU A channel DSPI C peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	E1	C1	E1
ETPUA[31]_ PCSC[4]_ GPIO[145]	eTPU A channel DSPI C peripheral chip select GPIO	P A G	I/O O I/O	V_{DDEH1}	MH	– / WKPCFG	– / WKPCFG	E2	C2	E2
eMIOS										
EMIOS[0:2]_ ETPUA[0:2]_ GPIO[179:181]	eMIOS channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	T4, T5, N7	AB10, AB11, W12	AD16, AD21, P21
EMIOS[3:5]_ ETPUA[3:5]_ GPIO[182:184]	eMIOS channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	R6, R5, T6	AA11, AB12, AA12	R22, AD18, AD22
EMIOS[6:7]_ ETPUA[6:7]_ GPIO[185:186]	eMIOS channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	P7, T7	Y12, AB13	P22, AD19
EMIOS[8:9]_ ETPUA[8:9]_ GPIO[187:188]	eMIOS channel eTPU A channel (output only) GPIO	P A G	I/O O I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	P8, R7	W13, AA13	N21, AD23
EMIOS[10:11]_ PCSD[3:4]_ GPIO[189:190]	eMIOS channel DSPI D peripheral chip select GPIO	P A G	I/O I/O I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	N8, R8	Y13, AB14	N22, AG18
EMIOS[12]_ SOUTC_ GPIO[191]	eMIOS channel (output only) DSPI C data output GPIO	P A G	O O I/O	V_{DDEH4}	MH	– / WKPCFG	– / WKPCFG	N10	W15	M21
EMIOS[13]_ SOUTD_ GPIO[192]	eMIOS channel (output only) DSPI D data output GPIO	P A G	O O I/O	V_{DDEH4}	MH	– / WKPCFG	– / WKPCFG	T8	AA14	AF18
EMIOS[14]_ IRQ[0]_ GPIO[193]	eMIOS channel (output only) External interrupt request GPIO	P A G	O I I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	R9	AB15	AH19

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
EMIOS[15]_ IRQ[1]_ GPIO[194]	eMIOS channel (output only) External interrupt request GPIO	P A G	O I I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	T9	Y14	M22
EMIOS[16]_ GPIO[195]	eMIOS channel GPIO	P G	I/O O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	P9	AA15	AG19
EMIOS[17]_ GPIO[196]	eMIOS channel GPIO	P G	I/O O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	P10	Y15	AF19
EMIOS[18]_ GPIO[197]	eMIOS channel GPIO	P G	I/O O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	T10	AB16	AH20
EMIOS[19]_ GPIO[198]	eMIOS channel GPIO	P G	I/O O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	R10	AA16	AG20
EMIOS[20]_ GPIO[199]	eMIOS channel GPIO	P G	I/O I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	T11	AB17	AG21
EMIOS[21]_ GPIO[200]	eMIOS channel GPIO	P G	I/O I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	N11	W16	L21
EMIOS[22]_ GPIO[201]	eMIOS channel GPIO	P G	I/O I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	P11	Y16	AF20
EMIOS[23]_ GPIO[202]	eMIOS channel GPIO	P G	I/O I/O	V_{DDEH4}	SH	– / WKPCFG	– / WKPCFG	R11	AA17	AF21
GPIO										
EMIOS[14:15]_ GPIO[203:204][24]	eMIOS channel (output only) GPIO	P G	O I/O	V_{DDEH6}	SH	– / Up	– / Up	—	H20, H21	J26, H27
GPIO[206:207][25]	GPIO	G	I/O	V_{DDE3}	F	– / Up	– / Up	R4, P5	AA7, Y9	AH10, AG10
Clock Synthesizer										
XTAL	Crystal oscillator output	P	O	V_{DDSYN}	AE	O / –	XTAL[26] / –	P16	V22	AD28
EXTAL_ EXTCLK[27]	Crystal oscillator input External clock input	P A	I I	V_{DDSYN}	AE	I / –	EXTAL[28] / –	N16	U22	AC28
CLKOUT	System clock output	P	O	V_{DDE5}	F	CLKOUT / Enabled	CLKOUT / Enabled	—	AA20	AF25
ENGCLK	Engineering clock output	P	O	V_{DDE5}	F	ENGCLK/ Enabled	ENGCLK / Enabled	T14	AB21	AG26
Power / Ground										
V_{RC33}[29]	Voltage regulator control supply	P	I	3.3 V	V_{DDINT}	I / –	V_{RC33}	P15	W21	AD26
V_{RCCTL}	Voltage regulator control output	P	O	3.3 V	V_{DDINT}	O / –	V_{RCCTL}	N14	V20	AC26
V_{RCVSS}	Voltage regulator control ground	P	I	—	V_{SSE}	I / –	V_{RCVSS}	—	T21	V27
V_{DDA0}[30]	Analog power input ADC[0]	P	I	5.0 V	V_{DDINT}	I / –	V_{DDA0}	B11	C15	E15
V_{SSA0}[30]	Analog ground input ADC[0]	P	I	—	V_{SSINT}	I / –	V_{SSA0}	A11	A15, B15	A15, B15
V_{DDA1}[30]	Analog power input ADC[1]	P	I	5.0 V	V_{DDINT}	I / –	V_{DDA1}	A4	A6	A5
V_{SSA1}[30]	Analog ground input ADC[1]	P	I	—	V_{SSINT}	I / –	V_{SSA1}	A5	A7	A6

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
V_{DDSYN}	Clock synthesizer power input	P	I	3.3 V	V_{DDE}	I / –	V_{DDSYN}	R16	W22	AD27
V_{SSSYN}	Clock synthesizer ground input	P	I	—	V_{SSE}	I / –	V_{SSSYN}	M16	T22	AC27
V_{FLASH}	Flash read supply input	P	I	3.3 V	V_{DDINT}	I / –	V_{FLASH}	—	N22	W27
V_{PP}[31]	Flash program/erase supply input	P	I	5.0 V	V_{DDINT}	I / –	V_{PP}	K16	M22	W28
V_{STBY}[32]	SRAM standby power input	P	I	0.8–1.2 V	V_{STBY}	I / –	V_{STBY}	C1	A3	B3
V_{DD}	Internal logic supply input	P	I	1.5 V	V_{DD}	I / –	V_{DD}	B1, C2, D3, E4, B16, P13, R14, T15, N5, P4, R3, T2	A2, A20, B3, C4, C22, D5, V19, W5, W20, Y4, Y21, AA3, AA22, AB2	B25, C2, D3, D27, F5, H7, J8, Y21, AA9, AA22, AB8, AC24, AD6, AE26, AF4, AF27, AG3
V_{DDE2}[33]	External I/O supply input	P	I	1.8–3.3 V	—	I / –	V_{DDE2}	P6	M9:10, N11, P11, R3, W2, W6, W8, Y5, AA4, AA6, AA10, AB3	M11, N11:13, P11:13, R1, V5, AA5, AC1
V_{DDE3}[33]	External I/O supply input	P	I	1.8–3.3 V	—	I / –	V_{DDE3}	—	—	T14, U13:14, V12:14, AD9, AD14, AH6, AH14
V_{DDE5}	External I/O supply input	P	I	1.8–3.3 V	—	I / –	V_{DDE5}	T13	W17, Y18, AA19, AB20,	AF23, AG24, AH24
V_{DDE7}	External I/O supply input	P	I	1.8–3.3 V	—	I / –	V_{DDE7}	E13	B22, C21, D20, E19, F19, J14	C27, D26, F24, H22, J21, L15:18, M11, M18, N11:13, N18

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
V_{DDE12}	External I/O supply input – calibration	P	I	1.8–3.3 V	—	I / –	V_{DDE12}	—	—	K7, N8, R11:13, R17:18, R21, T11:12, T15,T18, U2, U11, U15:16, V15:17, V22, AA13, AA16, AB18, AB21, AE2, AG4, AG12
V_{DDEH1}	External I/O supply input	P	I	3.3–5.0 V	—	I / –	V_{DDEH}	K4	H4	G11, J7
V_{DDEH4}	External I/O supply input	P	I	3.3–5.0 V	—	I / –	V_{DDEH}	N9	W14	AD20
V_{DDEH6}	External I/O supply input	P	I	3.3–5.0 V	—	I / –	V_{DDEH}	F13	U19	V26
V_{DDEH8}	External I/O supply input	P	I	3.3–5.0 V	—	I / –	V_{DDEH}	—	—	C21
V_{DDEH9}[34]	External I/O supply input	P	I	3.3–5.0 V	—	I / –	V_{DDEH}	D12	D15	H14
V_{DDEH10}	External I/O supply input	P	I	3.3–5.0 V	—	I / –	V_{DDEH}	—	H19	K24
V_{DD33}[35]	I/O pad pre-driver and level shifter reference voltage input	P	I	3.3 V	—	—	3.3 V	A15, D1, N6, N12	B1, A21, P4, Y22, W7	B26, D2, W5, AE27, AF9

Table 2-1. MPC5534 Signal Properties (continued)

Signal Name [1]	Signal Function	P/A/G	I/O Type	Voltage[2]	Pad Type[3]	Status		Package		496[4] VertiCal
						During Reset[5]	After Reset[6]	208	324	
V_{SS}	MCU ground	P	—	—	—	—	V_{SS}	A1, B2, C3, D4, D13, C14, B15, A16, N13, P14, R15, T16, N4, P3, R2, T1, G7:10, H7:10, J7:10, K7:10	A1, A22, B2, B21, C3, C20, D4, D19, J9:13, K9:14, L9:14, M11:14 N9:10, N12:14, P9:10, P12:14, W4, W19, Y3, Y20, AA2, AA21, AB1, AB22	A1, A2, A27, A28, B1, B2, B27, B28, C3, C26, E5, E24, G7, G22, H8, H21, L11:14, M12:17, N14:17, P14:17, R14:16, T13, T16:17, U12, U17:18, V7, V18, AA8, AA21, AB7, AB22, AD5, AF3, AF26, AG1:2, AG27:28 AH1:2, AH27:28
No Connect										
NC[36]	No connect	—	—	—	—	—	—	—	W18, Y19	A19, B17:19, C17:19, E17:23, G16:21, H16:20, J22, J27:28, K21:22, K26:28, L22, L24, L26:27, M24, P2, R2, AA12, AG17

1 Because more than one signal is often multiplexed to one pin, each row in the signal name column is a separate function. For all device I/O pins, the primary, alternate, or GPIO signal functions are designated in the PA field of the system integration unit (SIU) PCR registers except where explicitly noted.

[2] V_{DDE} (fast I/O) and V_{DDEH} (slow I/O) power supply inputs are grouped into segments. Each segment of V_{DDEH} pins connects to a separate 3.3–5.0 V (+5% and –10%) power supply input. Each segment of V_{DDE} pins connects to a separate 1.8–3.3 V (±10%) power supply, with the exception of the V_{DDE2} and V_{DDE3} segments that are shorted together and must use the same power supply input. This segment is labeled V_{DDE2} in the BGA map.

[3] The pad type is indicated by one of the abbreviations; F for fast, MH for medium (high voltage), SH for slow (high voltage), A for analog, AE for analog with ESD protection circuitry. Some pads have two types, depending on which pad function is selected.

[4] The 496 assembly contains the VertiCal base that includes all pins in the 324 and 208 packages.

[5] The Status During Reset pin is sampled after the internal POR is deasserted. Prior to exiting POR, the signal has a high impedance. Terminology is O - output, I - input, Up - weak pullup enabled, Down - weak pulldown enabled, Low - output driven low, High - output driven high. A dash on the left side of the slash denotes that both the input and output buffers for the pin are off. A dash on the right side of the slash denotes that there is no weak pullup/down enabled on the pin. The signal name to the left or right of the slash indicates the pin is enabled.

[6] Function after reset of GPI is general purpose input. A dash on the left side of the slash denotes that both the input and output buffers for the pin are off. A dash on the right side of the slash denotes that there is no weak pullup/down enabled on the pin.

[7] The BOOTCFG[0] and $\overline{\text{RSTCFG}}$ pin are not available in the 208 package and are internally asserted (driven to 0) in the package.

[8] The EBI is specified and tested at 1.8–3.3 V.

[9] When using the EBI functions, select the function in the SIU_PCR register, and then enable the EBI functions in the EBI registers for these pins. Both the SIU and EBI configurations must match to operate correctly.

[10] The function and state of this pin(s) after execution of the BAM program is determined by the BOOTCFG[0:1] pins. See for detail on the External Bus Interface (EBI) configuration after execution of the BAM program. BOOTCFG[0] is not available on the 208 package and is internally asserted (driven to 0).

[11] $\overline{\text{CS}}$[1:3], ADDR[12:31], RD_$\overline{\text{WR}}$, $\overline{\text{BDIP}}$, $\overline{\text{TS}}$, $\overline{\text{TA}}$, $\overline{\text{BR}}$, and $\overline{\text{BG}}$ signals are not available on the 208 package due to pin limitations.

[12] The functions for the $\overline{\text{WE}}$/$\overline{\text{BE}}$[0:1]_GPIO[64:65] and CAL_$\overline{\text{WE}}$/$\overline{\text{BE}}$[0:1] pins are specified in the SIU. To configure the EBI, the write enable or byte enable operation is specified in the EBI_BR0 through EBI_BR3 registers.To configure the calibration bus, the write enable or byte enable operation is specified in the EBI_CAL_BR0 through EBI_CAL_BR3 registers for each chip select region. $\overline{\text{WE}}$/$\overline{\text{BE}}$[0:1] are not available on the 208 package due to pin limitations.

[13] These signals are available on the VertiCal assembly only.

[14] $\overline{\text{MCKO}}$ is only enabled if debug mode is enabled. Debug mode can be enabled before or after exiting System Reset ($\overline{\text{RSTOUT}}$ deasserted).

[15] MDO[0] is driven high following a power-on reset until the system clock achieves lock, at which time it is then deasserted. There is an internal pullup on MDO[0].

[16] The function of the MDO[11:4]_GPIO[82:75] pins is selected during a debug port reset by the $\overline{\text{EVTI}}$ pin or by selecting FPM in the NPC_PCR. When functioning as MDO[11:4] the pad configuration specified by the SIU does not apply. See 2.3.4.4 for more detail on MDO[11:4] pin operation.

[17] The function and state of the FlexCAN A pins after execution of the BAM program is determined by the BOOTCFG[0:1] pins. See Table 15-9 for details on the FlexCAN pin configurations after the BAM executes. BOOTCFG[0] is not available on the 208 package and is internally asserted (driven to 0).

[18] The primary signal is not available on this device and is listed only for reference to the pin label in the BGA Map.

[19] For compatibility to the MPC5554, always power V_{DDEH6} and V_{DDEH10} from the same power supply 3.0–5.25 V. To allow one DSPI to operate at a different operating voltage, connect V_{DDEH6} and V_{DDEH10} to separate power supplies, but this configuration is not compatible with the MPC5554,

[20] All analog input channels are connected to both ADC blocks. The supply designation for this pin(s) specifies only the ESD rail used.

[21] Because the primary signal function designations for the analog functions AN[12] through AN[15] are internally reserved, the PA field of the corresponding SIU_PCR registers must be set to the main primary function value of 0b011 to use analog functions AN[12] through AN[15].

[22] To use the serial data strobe functions, the PA field in the SIU_PCR registers must be set to 0b00. Because $\overline{\text{SDS}}$, SDO, SDI, and FCK *use* the GPIO setting, a G is shown in the P/A/G column. However, these signals do not support GPIO functionality.

[23] If analog features are used, tie V_{DDEH9} to V_{DDA1}.

[24] Because other balls on this device are labeled EMIOS[14:15], the balls for these signals are referred to as GPIO[203:204]. These pins are not available on the 208 package.

[25] The GPIO[206:207] pins are protect-for-pins for double data rate (DDR) memory data strobes. These pins can be selected as the source for the eQADC trigger in the eQADC Trigger Input Select Register (SIU_ETISR). These pins are not available on the 208 package.

[26] The Function After Reset of the XTAL pin is determined by the value of the signal on the PLLCFG[1] pin. Ground the XTAL pin when using bypass mode.

[27] When the FMPLL is configured for external reference mode, the V_{DDE5} supply affects the acceptable signal levels for the external reference. See Section 11.1.4.2, “External Reference Mode.”

[28] The function after reset of the EXTAL_EXTCLK pin is determined by the value of the signal on the PLLCFG[1] pin. The operating voltage for the EXTAL function is 3.3 V; the operating voltage for the EXTCLK function is 1.62–3.6 V.

[29] V_{RC33} is the 3.3 V input for the voltage regulator control.

[30] The V_{DDAn} and V_{SSAn} supply inputs are split into separate traces in the package substrate. Each trace is bonded to a separate pad location, which provides isolation between the analog and digital sections within each ADC.

[31] Can be tied to 5.0 V for both read operation and program / erase.

[32] Tie the V_{STBY} pin to V_{SSA0} if the battery backed SRAM is not used.

[33] Both V_{DDE2} and V_{DDE3} pins are labeled as V_{DDE2} pins on the BGA maps. V_{DDE3} are connected internally V_{DDE2}.

[34] The V_{DDEH9} segment can be powered by 3.0–5.0 V for mux addresses or SSI functions, however the V_{DDEH9} segment must comply with the V_{DDA1} specifications (4.5–5.25 V) for analog input functions.

[35] All pins with pad type F (pad_fc) are driven to the high state if their V_{DDE} segment is powered before V_{DD33}.

[36] The pins are reserved for the clock and inverted clock outputs for the DDR memory interface.

2.3 Detailed Signal Description

Descriptions of the signals for the device are provided in the following sections. See Table 2-1 for signal properties.

2.3.1 Reset and Configuration Signals

2.3.1.1 External Reset Input RESET

Assert the RESET signal as an active low input from an external device during a power-on reset or external reset to reset all device modules. See Section 4.2.1, “Reset Input (RESET).”

2.3.1.2 External Reset Output RSTOUT

The RSTOUT output is a push/pull output that is asserted during an internal device reset. You can assert RSTOUT via software without causing an internal reset to the device MCU. See Section 4.2.2, “Reset Output (RSTOUT).”

NOTE

During a power-on-reset (POR), RSTOUT is tri-stated.

2.3.1.3 FMPLL Mode Selection / External Interrupt Request / GPIO PLLCFG[0]_IRQ[4]_GPIO[208]

PLLCFG[0]_IRQ[4]_GPIO[208] are sampled on the deassertion of the RESET input pin, if the RSTCFG pin is asserted at that time. The values are used to configure the FMPLL mode of operation. The alternate function is external interrupt request input.

2.3.1.4 FMPLL Mode Selection / External Interrupt Request / DSPI D / GPIO PLLCFG[1]_IRQ[5]_SOUTD_GPIO[209]

PLLCFG[1]_IRQ[5]_SOUTD_GPIO[209] are sampled on the deassertion of the RESET input pin, if the RSTCFG pin is asserted at that time. The values are used to configure the FMPLL mode of operation. The alternate functions are external interrupt request input, and data output for the DSPI module D.

2.3.1.5 Reset Configuration Input / GPIO RSTCFG_GPIO[210]

The RSTCFG input is used to enable the BOOTCFG[0:1] and PLLCFG[0:1] pins during reset. If RSTCFG is deasserted during reset, the BOOTCFG and PLLCFG pins are not sampled at the deassertion of RSTOUT. In that case, the default values for BOOTCFG and PLLCFG are used. If RSTCFG is asserted during reset, the values on the BOOTCFG and PLLCFG pins are sampled and configure the boot and FMPLL modes.

208 Package: $\overline{\text{RSTCFG}}$_GPIO[210] is not available due to pin limitations. The BOOTCFG[0] and the $\overline{\text{RSTCFG}}$ signals are internally asserted (driven to 0).

2.3.1.6 Reset Configuration / External Interrupt Request / GPIO BOOTCFG[0:1]_$\overline{\text{IRQ}}$[2:3]_GPIO[211:212]

The BOOTCFG[0:1]_$\overline{\text{IRQ}}$[2:3]_GPIO[211:212] are sampled when $\overline{\text{RSTOUT}}$ deasserts, if the $\overline{\text{RSTCFG}}$ pin is asserted at that time. The BOOTCFG[0:1] values are used by the BAM program to determine the boot configuration of the device. Use the alternate function for external interrupt request inputs.

208 Package: BOOTCFG[0]_$\overline{\text{IRQ}}$[2]_GPIO[211] and $\overline{\text{RSTCFG}}$_GPIO[210] are not available due to pin limitations, and are internally asserted.

2.3.1.7 Weak Pull Configuration / GPIO WKPCFG_GPIO[213]

WKPCFG_GPIO[213] determines whether specific eTPU and eMIOS pins are connected to a weak pullup or weak pulldown during and immediately after reset.

2.3.2 External Bus Interface (EBI)

This package has a 16-pin data bus [0:15] on the EBI.

208 Package: This package does not have EBI pins, therefore the external bus signals are not available except for the chip select 0 ($\overline{\text{CS}}$[0]) and the output enable ($\overline{\text{OE}}$) pins.

2.3.2.1 External Chip Selects / External Address / GPIO $\overline{\text{CS}}$[0]_ADDR[8]_GPIO[0]

$\overline{\text{CS}}$[0]_ADDR[8]_GPIO[0] is the external bus interface (EBI) chip select output signal. The alternate function is an EBI address signal.

2.3.2.2 External Chip Selects / External Address / GPIO $\overline{\text{CS}}$[1:3]_ADDR[9:11]_GPIO[1:3]

$\overline{\text{CS}}$[1:3]_ADDR[9:11]_GPIO[1:3] are the external bus interface (EBI) chip select output signals. The alternate functions are EBI address signals. They can be individually configured as chip selects, address signals or GPIO.

208 Package: The $\overline{\text{CS}}$[1:3]_ADDR[9:11]_GPIO[1:3] are not available due to pin limitations.

2.3.2.3 External Address / GPIO ADDR[12:31]_GPIO[8:27]

ADDR[12:31]_GPIO[8:27] are the EBI address signals.

208 Package: The ADDR[12:31]_GPIO[8:27] are not available due to pin limitations.

2.3.2.4 External Data / GPIO DATA[0:15]_GPIO[28:43]

DATA[0:15]_GPIO[28:43] are the EBI data signals.

208 Package: DATA[0:15]_GPIO[28:43] are not available due to pin limitations.

2.3.2.5 External Read/Write / GPIO RD_$\overline{WR}$_GPIO[62]

RD_$\overline{WR}$_GPIO[62] indicates whether an external bus transfer is a read or write operation.

208 Package: The RD_$\overline{WR}$_GPIO[62] is not available due to pin limitations.

2.3.2.6 External Burst Data In Progress / GPIO $\overline{BDIP}$_GPIO[63]

$\overline{BDIP}$_GPIO[63] indicates that an EBI burst transfer is in progress.

208 Package: The $\overline{BDIP}$_GPIO[63] signal is not available due to pin limitations.

2.3.2.7 External Write/Byte Enable / GPIO $\overline{WE}$/$\overline{BE}$[0:1]_GPIO[64:65]

$\overline{WE}$/$\overline{BE}$[0:1]_GPIO[64:65] specify which data pins contain valid data for an external bus transfer.

208 Package: The $\overline{WE}$/$\overline{BE}$[0:1]_GPIO[64:65] are not available due to pin limitations.

2.3.2.8 External Output Enable / GPIO $\overline{OE}$_GPIO[68]

$\overline{OE}$_GPIO[68] indicates that the EBI is ready to accept read data.

2.3.2.9 External Transfer Start / GPIO $\overline{TS}$_GPIO[69]

$\overline{TS}$_GPIO[69] is asserted by the EBI owner to indicate the start of a transfer.

208 Package: The $\overline{TS}$_GPIO[69] is not available due to pin limitations.

2.3.2.10 External Transfer Acknowledge $\overline{TA}$_GPIO[70]

$\overline{TA}$_GPIO[70] is asserted by the EBI owner to acknowledge that the slave has completed the current transfer.

208 Package: The $\overline{TA}$_GPIO[70] is not available due to pin limitations.

2.3.3 Calibration Bus Interface (CBI)

NOTE

Calibration bus signals for the 324 package require the 496 VertiCal assembly. In the 208 package, the calibration signals are not available due to pin limitations.

2.3.3.1 Calibration Chip Select CAL_$\overline{CS}$[0]

CAL_$\overline{CS}$[0] is the calibration chip select output signal. The 496 VertiCal assembly is required to use the calibration bus signals.

208 Package: The CAL_$\overline{CS}$[0] signal is not available due to pin limitations.

2.3.3.2 Calibration Chip Selects / Calibration Address CAL_$\overline{CS}$[2:3]_CAL_ADDR[10:11]

CAL_$\overline{CS}$[2:3]_CAL_ADDR[10:11] are the calibration chip select output signals. The alternate functions are calibration address signals.

208 Package: The CAL_$\overline{CS}$[2:3] signals are not available due to pin limitations.

2.3.3.3 Calibration Address CAL_ADDR[12:30]

CAL_ADDR[12:30] are the calibration address signals. The 496 VertiCal assembly is required to use the calibration bus signals.

208 Package: The CAL_ADDR[12:30] signals are not available due to pin limitations.

2.3.3.4 Calibration Data CAL_DATA[0:15]

The CAL_DATA[0:15] are the calibration data signals. The 496 VertiCal assembly is required to use the calibration bus signals.

208 Package: CAL_DATA[0:15] are not available due to pin limitations.

2.3.3.5 Calibration Read/Write CAL_RD_$\overline{WR}$

CAL_RD_$\overline{WR}$ indicates whether a calibration bus transfer is a read or write operation. The 496 VertiCal assembly is required to use the calibration bus signals.

208 Package: The CAL_RD_$\overline{WR}$ is not available due to pin limitations.

2.3.3.6 Calibration Write / Byte Enable CAL_WE/BE[0:1]

CAL_WE/BE[0:1] specify which data pins contain valid data for a calibration bus transfer. The 496 VertiCal assembly is required to use the calibration bus signals.

208 Package: The CAL_WE/BE[0:1] signals are not available due to pin limitations.

2.3.3.7 Calibration Output Enable CAL_OE

CAL_OE indicates that the calibration interface is ready to accept read data. The 496 VertiCal assembly is required to use the calibration bus signals.

208 Package: The CAL_OE signal is not available due to pin limitations.

2.3.3.8 Calibration Transfer Start CAL_TS

CAL_TS is asserted by the device to indicate the start of a transfer. The 496 VertiCal assembly is required to use the calibration bus signals.

208 Package: The CAL_TS signal is not available due to pin limitations.

2.3.4 Nexus Controller

2.3.4.1 Nexus Event In EVTI

EVTI is an input that is read when TRST asserts to enable or disable the Nexus debug port. After reset, the EVTI pin is used to initiate program and data trace synchronization messages or generate a breakpoint.

2.3.4.2 Nexus Event Out EVTO

EVTO is an output that provides timing to a development tool for a single watchpoint or breakpoint occurrence.

2.3.4.3 Nexus Message Clock Out MCKO

MCKO is a free running clock output to the development tools which is used for timing of the MDO and MSEO signals.

2.3.4.4 Nexus Message Data Out MDO[3:0]

MDO[3:0] are the trace message outputs to the development tools.

In addition to being a trace output, MDO[0] indicates the lock status of the system clock following a power-on reset. MDO[0] is driven high following a power-on reset until the system clock achieves lock, at which time it is then deasserted. There is an internal pullup on MDO[0].

2.3.4.5 Nexus Message Data Out / GPIO MDO[4:11]_GPIO[82:75]

MDO[11:4]_GPIO[82:75] are the trace message outputs to the development tools for full port mode. These pins function as GPIO when the Nexus port controller (NPC) operates in reduced port mode.

208 Package: MDO[11:4]_GPIO[82:75] signals are not available due to pin limitations.

2.3.4.6 Nexus Message Start/End Out MSEO[1:0]

$\overline{\text{MSEO}}$[1:0] are outputs that indicate when messages start and end on the MDO pins.

2.3.4.7 Nexus Ready Output RDY

$\overline{\text{RDY}}$ is an output that indicates to the development tools the data is ready to be read from or written to the Nexus read/write access registers.

208 Package: The $\overline{\text{RDY}}$ is not available due to pin limitations.

2.3.5 JTAG

2.3.5.1 JTAG Test Clock Input TCK

TCK provides the clock input for the on-chip test logic.

2.3.5.2 JTAG Test Data Input TDI

TDI provides the serial test instruction and data input for the on-chip test logic.

2.3.5.3 JTAG Test Data Output TDO

TDO provides the serial test data output for the on-chip test logic.

2.3.5.4 JTAG Test Mode Select Input TMS

TMS controls test mode operations for the on-chip test logic.

2.3.5.5 JTAG Compliance Input JCOMP

The JCOMP pin is used to enable the JTAG TAP controller.

2.3.5.6 Test Mode Enable Input TEST

The $\overline{\text{TEST}}$ pin is used to place the chip in test mode. Deassert this signal for normal operation.

2.3.6 Flexible Controller Area Network (FlexCAN)

2.3.6.1 FlexCAN A Transmit / GPIO CNTXA_GPIO[83]

CNTXA_GPIO[83] is the transmit pin for the FlexCAN A module.

2.3.6.2 FlexCAN A Receive / GPIO CNRXA_GPIO[84]

CNRXA_GPIO[84] is the receive pin for the FlexCAN A module.

2.3.6.3 FlexCAN B Transmit / DSPI C Chip Select / GPIO CNTXB_PCSC[3]_GPIO[85]

The primary function, CNTXB, is not available on this device. PCSC[3]_GPIO[85] is the alternate function and is a peripheral chip select output for the DSPI C module.

2.3.6.4 FlexCAN B Receive / DSPI C Chip Select / GPIO CNRXB_PCSC[4]_GPIO[86]

The primary function, CNRXB, is not available on this device. PCSC[4]_GPIO[86] is the alternate function and is a peripheral chip select output for the DSPI C module.

2.3.6.5 FlexCAN C Transmit / DSPI D Chip Select / GPIO CNTXC_PCSD[3]_GPIO[87]

CNTXC_PCSD[3]_GPIO[87] is the transmit pin for the FlexCAN C module. The alternate function is a peripheral chip select for the DSPI D module.

2.3.6.6 FlexCAN C Receive / DSPI D Chip Select / GPIO CNRXC_PCSD[4]_GPIO[88]

CNRXC_PCSD[4]_GPIO[88] is the receive pin for the FlexCAN C module. The alternate function is a peripheral chip select for the DSPI D module.

2.3.7 Enhanced Serial Communications Interface (eSCI)

2.3.7.1 eSCI A Transmit / GPIO TXDA_GPIO[89]

TXDA_GPIO[89] is the transmit pin for the eSCI A module.

2.3.7.2 eSCI A Receive / GPIO RXDA_GPIO[90]

RXDA_GPIO[90] is the receive pin for the eSCI A module. The pin is an input only for the RXD function, but as GPIO the pin is input or output based on the SIU PCR configuration.

2.3.7.3 eSCI B Transmit / DSPI D Chip Select / GPIO TXDB_PCSD[1]_GPIO[91]

TXDB_PCSD[1]_GPIO[91] is the transmit pin for the eSCI B module. The alternate function is a peripheral chip select output for the DSPI D module.

2.3.7.4 eSCI B Receive / DSPI D Chip Select / GPIO RXDB_PCSD[5]_GPIO[92]

RXDB_PCSD[5]_GPIO[92] is the transmit pin for the eSCI B module. The alternate function is a peripheral chip select for the DSPI D module.

2.3.8 Deserial/Serial Peripheral Interface (DSPI)

2.3.8.1 DSPI A Clock / DSPI C / GPIO SCKA_PCSC[1]_GPIO[93]

The primary function, SCKA, is not available on this device. PCSC[1]_GPIO[93] is the peripheral chip select output pin for the DSPI C module.

2.3.8.2 DSPI A Input / DSPI C / GPIO SINA_PCSC[2]_GPIO[94]

The primary function, SINA, is not available on this device. PCSC[2]_GPIO[94] is the peripheral chip select output pin for the DSPI C module.

2.3.8.3 DSPI A Output / DSPI C / GPIO SOUTA_PCSC[5]_GPIO[95]

The primary function, SOUTA, is not available on this device. PCSC[5]_GPIO[95] is the peripheral chip select output pin for the DSPI C module.

2.3.8.4 DSPI A / DSPI D / GPIO PCSA[0]_PCSD[2]_GPIO[96]

The primary function, PCSA[0], is not available on this device. PCSD[2]_GPIO[96] are peripheral chip select output pins for the peripheral chip select output pin for the DSPI D module.

208 Package: The PCSA[0]_PCSD[2]_GPIO[96] pin is not available due to pin limitations.

2.3.8.5 DSPI A / DSPI B / GPIO PCSA[1]_PCSB[2]_GPIO[97]

The primary function, PCSA[1], is not available on this device. PCSB[2]_GPIO[97] are peripheral chip select output pins for the DSPI B module.

208 Package: The PCSA[1]_PCSB[2]_GPIO[97] pin is not available due to pin limitations.

2.3.8.6 DSPI A / DSPI D Clock / GPIO PCSA[2]_SCKD_GPIO[98]

The primary function, PCSA[2], is not available on this device. SCKD_GPIO[98] is a peripheral chip select output pin for the DSPI D module.

2.3.8.7 DSPI A / DSPI D Data Input / GPIO PCSA[3]_SIND_GPIO[99]

The primary function, PCSA[3], is not available on this device. SIND_GPIO[99] is a peripheral chip select output pin for the DSPI D module.

2.3.8.8 DSPI A / DSPI D Data Output / GPIO PCSA[4]_SOUTD_GPIO[100]

The primary function, PCSA[4], is not available on this device. SOUTD_GPIO[100] is the alternate function is the data output for the DSPI D module.

208 Package: The PCSA[4]_SOUTD_GPIO[100] pin is not available due to pin limitations.

2.3.8.9 DSPI A / DSPI B / GPIO PCSA[5]_PCSB[3]_GPIO[101]

The primary function, PCSA[5], is not available on this device. PCSB[3] is the alternate function and is a peripheral chip select output pin for the DSPI B module. The GPIO[101] signal is the general purpose input/output function

208 Package: The PCSA[5]_PCSB[3]_GPIO[101] pin is not available due to pin limitations.

2.3.8.10 DSPI B Clock / DSPI C Chip Select / GPIO SCKB_PCSC[1]_GPIO[102]

SCKB_PCSC[1]_GPIO[102] is the SPI clock pin for the DSPI B module. The alternate function is a chip select output for the DSPI C module.

2.3.8.11 DSPI B Data Input / DSPI C Chip Select / GPIO SINB_PCSC[2]_GPIO[103]

SINB_PCSC[2]_GPIO[103] is the data input pin for the DSPI B module. The alternate function is a chip select output for the DSPI C module.

2.3.8.12 DSPI B Data Output / DSPI C Chip Select / GPIO SOUTB_PCSC[5]_GPIO[104]

SOUTB_PCSC[5]_GPIO[104] is the data output pin for the DSPI B module. The alternate function is a chip select output for the DSPI C module.

2.3.8.13 DSPI B Chip Select / DSPI D Chip Select / GPIO PCSB[0]_PCSD[2]_GPIO[105]

PCSB[0]_PCSD[2]_GPIO[105] is a peripheral chip select output pin (slave select input pin for slave operation) for the DSPI B module. The alternate function is a chip select output for the DSPI D module.

2.3.8.14 DSPI B Chip Select / DSPI D Chip Select / GPIO PCSB[1]_PCSD[0]_GPIO[106]

PCSB[1]_PCSD[0]_GPIO[106] is a peripheral chip select output pin for the DSPI B module. The alternate function is a chip select output (slave select input pin for slave operation) for the DSPI D module.

2.3.8.15 DSPI B Chip Select / DSPI C Data Output / GPIO PCSB[2]_SOUTC_GPIO[107]

PCSB[2]_SOUTC_GPIO[107] is a peripheral chip select output pin for the DSPI B module. The alternate function is the data output for the DSPI C module.

2.3.8.16 DSPI B Chip Select / DSPI C Data Input / GPIO PCSB[3]_SINC_GPIO[108]

PCSB[3]_SINC_GPIO[108] is a peripheral chip select output pin for the DSPI B module. The alternate function is the data input for the DSPI C module.

2.3.8.17 DSPI B Chip Select / DSPI C Clock / GPIO PCSB[4]_SCKC_GPIO[109]

PCSB[4]_SCKC_GPIO[109] is a peripheral chip select output pin for the DSPI B module. The alternate function is the SPI clock for the DSPI C module.

2.3.8.18 DSPI B Chip Select / DSPI C Chip Select / GPIO PCSB[5]_PCSC[0]_GPIO[110]

PCSB[5]_PCSC[0]_GPIO[110] is a peripheral chip select output pin for the DSPI B module. The alternate function is a chip select output (slave select input in slave mode) for the DSPI C module.

2.3.9 Enhanced Queued Analog/Digital Converter (eQADC)

NOTE

The eQADC has 40 channels in the 324 package; the 208 packages in limited to 34 channels due to pin limitations.

2.3.9.1 Analog Input / Differential Analog Input AN[0]_DAN0+

AN[0] is a single-ended analog input to the two on-chip ADCs. DAN0+ is the positive terminal of the differential analog input DAN0 (DAN0+ to DAN0–).

2.3.9.2 Analog Input / Differential Analog Input AN[1]_DAN0–

AN[1] is a single-ended analog input to the two on-chip ADCs. DAN0– is the negative terminal of the differential analog input DAN0 (DAN0+ to DAN0–).

2.3.9.3 Analog Input / Differential Analog Input AN[2]_DAN1+

AN[2] is a single-ended analog input to the two on-chip ADCs. DAN1+ is the positive terminal of the differential analog input DAN1 (DAN1+ to DAN1–).

2.3.9.4 Analog Input / Differential Analog Input AN[3]_DAN1–

AN[3] is a single-ended analog input to the two on-chip ADCs. DAN1– is the negative terminal of the differential analog input DAN1 (DAN1+ to DAN1–).

2.3.9.5 Analog Input / Differential Analog Input AN[4]_DAN2+

AN[4] is a single-ended analog input to the two on-chip ADCs. DAN2+ is the positive terminal of the differential analog input DAN2 (DAN2+ to DAN2–).

2.3.9.6 Analog Input / Differential Analog Input AN[5]_DAN2–

AN[5] is a single-ended analog input to the two on-chip ADCs. DAN2– is the negative terminal of the differential analog input DAN2 (DAN2+ to DAN2–).

2.3.9.7 Analog Input / Differential Analog Input AN[6]_DAN3+

AN[6] is a single-ended analog input to the two on-chip ADCs. DAN3+ is the positive terminal of the differential analog input DAN3 (DAN3+ to DAN3–).

2.3.9.8 Analog Input / Differential Analog Input AN[7]_DAN3–

AN[7] is a single-ended analog input to the two on-chip ADCs. DAN3– is the negative terminal of the differential analog input DAN3 (DAN3+ to DAN3–).

2.3.9.9 Analog Input / Multiplexed Analog Input AN[8]_ANW

AN[8] is an analog input pin. The alternate function, ANW, is an analog input in external multiplexed mode. This pin has reduced analog to digital conversion accuracy as compared to the AN[0:7] and AN[16:39] analog input pins.

208 Package: The AN[8]_ANW pin is not available due to pin limitations.

2.3.9.10 Analog Input / Multiplexed Analog Input AN[9]_ANX

AN[9] is an analog input pin. The alternate function, ANX, is an analog input in external multiplexed mode. This pin has reduced analog to digital conversion accuracy as compared to the AN[0:7] and AN[16:39] analog input pins.

2.3.9.11 Analog Input / Multiplexed Analog Input AN[10]_ANY

AN[10] is an analog input pin. The alternate function, ANY, is an analog input in external multiplexed mode. This pin has reduced analog to digital conversion accuracy as compared to the AN[0:7] and AN[16:39] analog input pins.

208 Package: The AN[10]_ANY pin is not available due to pin limitations.

2.3.9.12 Analog Input / Multiplexed Analog Input AN[11]_ANZ

AN[11] is an analog input pin. The alternate function, ANZ, is an analog input in external multiplexed mode. This pin has reduced analog to digital conversion accuracy as compared to the AN[0:7] and AN[16:39] analog input pins.

2.3.9.13 Analog Input / Mux Address 0 / eQADC Serial Data Strobe AN[12]_MA[0]_$\overline{SDS}$

AN[12]_MA[0]_$\overline{SDS}$ is an analog input pin. The alternate function, MA[0], is a MUX address pin. The second alternate function is the serial data strobe for the eQADC SSI. This pin has reduced analog to digital conversion accuracy as compared to the AN[0:7] and AN[16:39] analog input pins.

2.3.9.14 Analog Input / Mux Address 1 / eQADC Serial Data Out AN[13]_MA[1]_SDO

AN[13]_MA[1]_SDO is an analog input pin. The alternate function, MA[1], is a MUX address pin. The second alternate function is the serial data output for the eQADC SSI. This pin has reduced analog to digital conversion accuracy as compared to the AN[0:7] and AN[16:39] analog input pins.

2.3.9.15 Analog Input / Mux Address 2 / eQADC Serial Data In AN[14]_MA[2]_SDI

AN[14]_MA[2]_SDI is an analog input pin. The alternate function, MA[2], is a MUX address pin. The second alternate function is the serial data input for the eQADC SSI. This pin has reduced analog to digital conversion accuracy as compared to the AN[0:7] and AN[16:39] analog input pins.

2.3.9.16 Analog Input / eQADC Free Running Clock AN[15]_FCK

AN[15]_FCK is an analog input pin. The alternate function is the free running clock for the eQADC SSI. This pin has reduced analog to digital conversion accuracy as compared to the AN[0:7] and AN[16:39] analog input pins.

2.3.9.17 Analog Input AN[16:39]

AN[16:39] are analog input pins.

208 Package: The AN[19:20, 26, 29] pins are not available due to pin limitations.

2.3.9.18 Voltage Reference High V_{RH}

V_{RH} is the voltage reference high input pin for the eQADC.

208 Package: The V_{RH} pin is not available due to pin limitations.

2.3.9.19 Voltage Reference Low V_{RL}

V_{RL} is the voltage reference low input pin for the eQADC.

208 Package: The V_{RL} pin is not available due to pin limitations.

2.3.9.20 Reference Bypass Capacitor REFBYPC

REFBYPC is a bypass capacitor input for the eQADC. Use a 100nF external bias capacitor to connect the REFBYPC pin to the V_{RL}.

2.3.10 Enhanced Time Processing Unit (eTPU)

2.3.10.1 eTPU A TCR Clock / External Interrupt Request / GPIO TCRCLKA_IRQ[7]_GPIO[113]

TCRCLKA_IRQ[7]_GPIO[113] is the TCR A clock input for the eTPU module. The alternate function is an external interrupt request input for the SIU module.

2.3.10.2 eTPU A Channel / eTPU A Channel (Output Only) / GPIO ETPUA[0:11]_ETPUA[12:23]_GPIO[114:125]

ETPUA[0:11]_ETPUA[12:23]_GPIO[114:125] are input/output channel pins for the eTPU A module. The alternate function is for output channels of the eTPU A module; that is, when configured as ETPUA[12:23], the pins function as outputs only.

2.3.10.3 eTPU A Channel / DSPI / GPIO ETPUA[12:19]_PCS*Xn*_GPIO[126:133]

ETPUA[12:19]_PCS*Xn*_GPIO[126:133] are input/output channel pins for the eTPU A module muxed with DSPI B and D pins.

2.3.10.4 eTPU A Channel / External Interrupt Request / GPIO ETPUA[20:27]_IRQ[8:15]_GPIO[134:141]

ETPUA[20:27]_IRQ[8:15]_GPIO[134:141] are input/output channel pins for the eTPU A module muxed with interrupt request pins.

2.3.10.5 eTPU A Channels / DSPI C / GPIO ETPUA[28:31]_PCSC[1:4]_GPIO[142:145]

ETPUA[28:31]_PCSC[1:4]_GPIO[142:145] are input/output channel pins for the eTPU A module multiplexed with DSPI C pins.

2.3.11 Enhanced Modular Input/Output System (eMIOS)

2.3.11.1 eMIOS Channels / eTPU A Channels (Output Only) / GPIO EMIOS[0:9]_ETPUA[0:9]_GPIO[179:188]

EMIOS[0:9]_ETPUA[0:9]_GPIO[179:188] are the primary functions for input/output channel pins for the eMIOS module. The alternate functions are output channels for the eTPU A module; that is, when configured as ETPUA[0:9], the pins function as outputs only.

2.3.11.2 eMIOS Channels / GPIO EMIOS[10:11]_GPIO[189:190]

EMIOS[10:11]_GPIO[189:190] are input/output channel pins for the eMIOS module.

2.3.11.3 eMIOS Channel (Output Only) / DSPI C Data Output / GPIO EMIOS[12]_SOUTC_GPIO[191]

EMIOS[12]_SOUTC_GPIO[191] is an output channel pin for the eMIOS module. The alternate function is the data output for the DSPI C module.

2.3.11.4 eMIOS Channel (Output Only) / DSPI D Data Output / GPIO EMIOS[13]_SOUTD_GPIO192

EMIOS[13]_SOUTD_GPIO[192] is an output channel pin for the eMIOS module. The alternate function is the data output for the DSPI D module.

2.3.11.5 eMIOS Channel (Output Only) / External Interrupt Request / GPIO EMIOS[14:15]_$\overline{\text{IRQ}}$[0:1]_GPIO[193:194]

EMIOS[14:15]_$\overline{\text{IRQ}}$[0:1]_GPIO[193:194] are output channel pins for the eMIOS module. The alternate function is for external interrupt request inputs.

2.3.11.6 eMIOS Channel (Output Only) / GPIO EMIOS[16:23]_GPIO[195:202]

EMIOS[16:23]_GPIO[195:202] are input/output channel pins for the eMIOS module.

2.3.12 GPIO

2.3.12.1 GPIO EMIOS[14:15]_GPIO[203:204]

EMIOS[14:15]_GPIO[203:204] are input or output pins. When configured as EMIOS[14:15], the pins function as output channels for the eMIOS module.

208 Package: The EMIOS[14:15]_GPIO[203:204] signals are not available due to pin limitations.

2.3.12.2 GPIO GPIO[206:207]

The GPIO[206:207] pins only have GPIO functionality. These pins are reserved for double data rate memory interface support.The pad type for GPIO[206:207] is fast driver and CMOS input buffer (1.62–1.98 V).

2.3.13 Clock Synthesizer

2.3.13.1 Crystal Oscillator Output XTAL

XTAL is the output pin for an external crystal oscillator.

2.3.13.2 Crystal Oscillator Input / External Clock Input EXTAL_EXTCLK

EXTAL is the input pin for an external crystal oscillator or an external clock source. The alternate function is the external clock input. The function of this pin is determined by the PLLCFG configuration pins.

2.3.13.3 System Clock Output CLKOUT

CLKOUT is the system clock output.

208 Package: The CLKOUT signal is not available due to pin limitations.

2.3.13.4 Engineering Clock Output ENGCLK

ENGCLK is a 50% duty cycle output clock with a maximum frequency of the device’s system clock divided by two. ENGCLK is not synchronous to CLKOUT.

2.3.14 Power/Ground

2.3.14.1 Voltage Regulator Control Supply Input V_{RC33}

V_{RC33} is the 3.3 V supply input pin for the on-chip 1.5 V regulator control circuit.

2.3.14.2 Voltage Regulator Control Ground Input V_{RCVSS}

V_{RCVSS} is the ground reference for the on-chip 1.5 V regulator control circuit.

208 Package: The V_{RCVSS} signal is not available due to pin limitations.

2.3.14.3 Voltage Regulator Control Output V_{RCCTL}

V_{RCCTL} is the output pin for the on-chip 1.5 V regulator control circuit.

2.3.14.4 eQADC Analog Supply $V_{DDA}n$

$V_{DDA}n$ is the analog supply input pin for the eQADC.

2.3.14.5 eQADC Analog Ground Reference $V_{SSA}n$

$V_{SSA}n$ is the analog ground reference input pin for the eQADC.

2.3.14.6 Clock Synthesizer Power Input V_{DDSYN}

V_{DDSYN} is the power supply input for the FMPLL.

2.3.14.7 Clock Synthesizer Ground Input V_{SSSYN}

V_{SSSYN} is the ground reference input for the FMPLL.

2.3.14.8 Flash Read Supply Input V_{FLASH}

V_{FLASH} is the on-chip Flash read supply input.

208 Package: The V_{FLASH} signal is not available due to pin limitations.

2.3.14.9 Flash Program/Erase Supply Input V_{PP}

V_{PP} is the on-chip Flash program/erase supply input.

2.3.14.10 SRAM Standby Power Input V_{STBY}

V_{STBY} is the power supply input that is used to maintain a portion of the contents of internal SRAM during power down. If not used, tie V_{STBY} to V_{SS}.

2.3.14.11 Internal Logic Supply Input V_{DD}

V_{DD} is the 1.5 V logic supply input.

2.3.14.12 External I/O Supply Input V_{DDEn}

V_{DDEn} is the 1.8–3.3 V ±10% external I/O supply input.

2.3.14.13 External I/O Supply Input V_{DDEHn}

V_{DDEHn} is the 3.3–5.0 V -10% to +5% external I/O supply input.

2.3.14.14 Fixed 3.3 V Internal Supply Input V_{DD33}

V_{DD33} is the 3.3 V internal supply input.

2.3.14.15 Ground V_{SS}

V_{SS} is the ground reference input.

2.3.15 I/O Power/Ground Segmentation

Table 2-2 gives the power/ground segmentation of the device MCU. Each segment provides the power and ground for the given set of I/O pins. Each segment can be powered by either of the allowed voltages regardless of the power on the other segments. The power/ground segmentation applies regardless of whether a particular pin is configured for its primary function or GPIO.

See Table 2-1, as not all signals are available on the 324 and 208 packages. The primary signals shown in **blue** are not available in this device, but are shown to locate the pin on the ball grid array (BGA). The signals shown in **red** are not available on the 208 package.

Table 2-2. MPC5534 Power/Ground Segmentation

Power Segment V_{DDE}	Voltage Range[1]	I/O Pins Powered by Segment
V_{DDEH1}	3.3–5.0 V	TCRCLKA_$\overline{IRQ}$[7]_GPIO[113], ETPUA[0:11]_ETPUA[12:23]_GPIO[114:125], ETPUA[12]_PCSB[1]_GPIO[126], ETPUA[13:15]_PCSB[3:5]_GPIO[127:129], ETPUA[16:19]_PCSD[1:4]_GPIO[130:133], ETPUA[20:27]_$\overline{IRQ}$[8:15]_GPIO[134:141], ETPUA[28:31]_PCSC[1:4]_GPIO[141:145]
V_{DDE2}[2]	1.8–3.3 V	$\overline{CS}$[0]_ADDR[8]_GPIO[0], $\overline{CS}$[1:3]_ADDR[9:11]_GPIO[1:3], ADDR[12:31]_GPIO[8:27], DATA[0:15]_GPIO[28:43], RD_$\overline{WR}$_GPIO[62], $\overline{BDIP}$_GPIO[63], $\overline{WE}$/$\overline{BE}$[0:1]_GPIO[64:65], $\overline{OE}$_GPIO[68], $\overline{TS}$_GPIO[69], $\overline{TA}$_GPIO[70], GPIO[206:207]
V_{DDEH4}	3.3–5.0 V	EMIOS[0:9]_ETPUA[0:9]_GPIO[179:188], EMIOS[10:11]_PCSD[3:4]_GPIO[189:190], EMIOS[12]_SOUTC_GPIO[191], EMIOS[13]_SOUTD_GPIO[192], EMIOS[14:15]_$\overline{IRQ}$[0:1]_GPIO[193:194], EMOIS[16:23]_GPIO[195:202], CNTXA_GPIO[83], CNRXA_GPIO[84], CNTXB_PCSC[3]_GPIO[85], CNRXB_PCSC[4]_GPIO[86]

Table 2-2. MPC5534 Power/Ground Segmentation

Power Segment V_{DDE}	Voltage Range[1]	I/O Pins Powered by Segment
V_{DDE5}	1.8–3.3 V	CLKOUT, ENGCLK
V_{DDEH6}	3.3–5.0 V	RESET, RSTOUT, RSTCFG_GPIO[210], WKPCFG_GPIO[213], BOOTCFG[0]_IRQ[2]_GPIO[211], BOOTCFG[1]_IRQ[3]_GPIO[212], PLLCFG[0]_IRQ[4]_GPIO[208], PLLCFG[1]_IRQ[5]_SOUTD_GPIO[209], CNTXC_PCSD[3]_GPIO[87], CNRXC_PCSD[4]_GPIO[88], TXDA_GPIO[89], RXDA_GPIO[90], TXDB_PCSD[1]_GPIO[91], RXDB_PCSD[5]_GPIO[92], SCKA_PCSC[1]_GPIO[93], SINA_PCSC[2]_GPIO[94], SOUTA_PCSC[5]_GPIO[95], PSCA[0]_PCSD[2]_GPIO[96], PSCA[1]_PCSB[2]_GPIO[97], PSCA[2]_SCKD_GPIO[98], PSCA[3]_SIND_GPIO[99], PSCA[4]_SOUTD_GPIO[100], PSCA[5]_PCSB[3]_GPIO[101], PCSB[3]_SINC_GPIO[108], PCSB[4]_SCKC_GPIO[109], PCSB[5]_PCSC[0]_GPIO[110], EMIOS[14:15]_GPIO[203:204]
V_{DDE7}	1.8–3.3 V	EVTI, EVTO, MCKO, MDO[3:0], MDO[11:4]_GPIO[82:75], MSEO[1:0], RDY, TCK, TDI, TDO, TMS, JCOMP, TEST
V_{DDEH9}	3.3–5.0 V	AN[12]_MA[0]_SDS, AN[13]_MA[1]_SDO, AN[14]_MA[2]_SDI, AN[15]_FCK
V_{DDEH10}	3.3–5.0 V	SCKB_PCSC[1]_GPIO[102], SINB_PCSC[2]_GPIO[103], SOUTB_PCSC[5]_GPIO[104], PCSB[0]_PCSD[2]_GPIO[105], PCSB[1]_PCSD[0]_GPIO[106], PCSB[2]_SOUTC_GPIO[107]
V_{DDE12}	1.8–3.3 V	CAL_ADDR[12:30], CAL_DATA[0:15], CAL_CS[0], CAL_CS[2:3], CAL_RD_WR, CAL_WE/BE[0:1], CAL_OE, CAL_TS
V_{DDA0}	5.0 V	AN[22:35], V_{RH}
V_{DDA1}	5.0 V	AN[0:11,16:21, 36:39]
V_{DDSYN}	3.3 V	XTAL, EXTAL_EXTCLK
V_{RC33}	3.3 V	V_{RCCTL}

1 These are nominal voltages. All V_{DDE} and V_{DDEH} voltages are ±10% (V_{DDE} 1.62–3.6 V; V_{DDEH} 3.0–5.25 V). V_{RC33} is ±10%. V_{DDSYN} is ±10%. V_{DDA} is +5%, -10%.

2 V_{DDE2} and V_{DDE3} are separate segments in the device pad ring. These segments are shorted together in the package substrate. The following pins are part of the V_{DDE3} segment: DATA[0:15], GPIO[206:207], OE.

2.4 eTPU Pin Connections and Serialization

2.4.1 ETPUA[0:15]

The ETPUA[0:15] module channels connect to external pins or can be serialized out through the DSPI C module. A diagram for the ETPUA[0:15] and DSPI C connections is given in Figure 2-4. The full list of connections is given in Table 2-3. Although not shown in Figure 2-4, the output channels of ETPUA[12:15] are connected to the ETPUA[0:3]_ETPUA[12:15]_GPIO[114:117] pins.

The eTPU TCRCLKA clock input is connected to an external pin only.

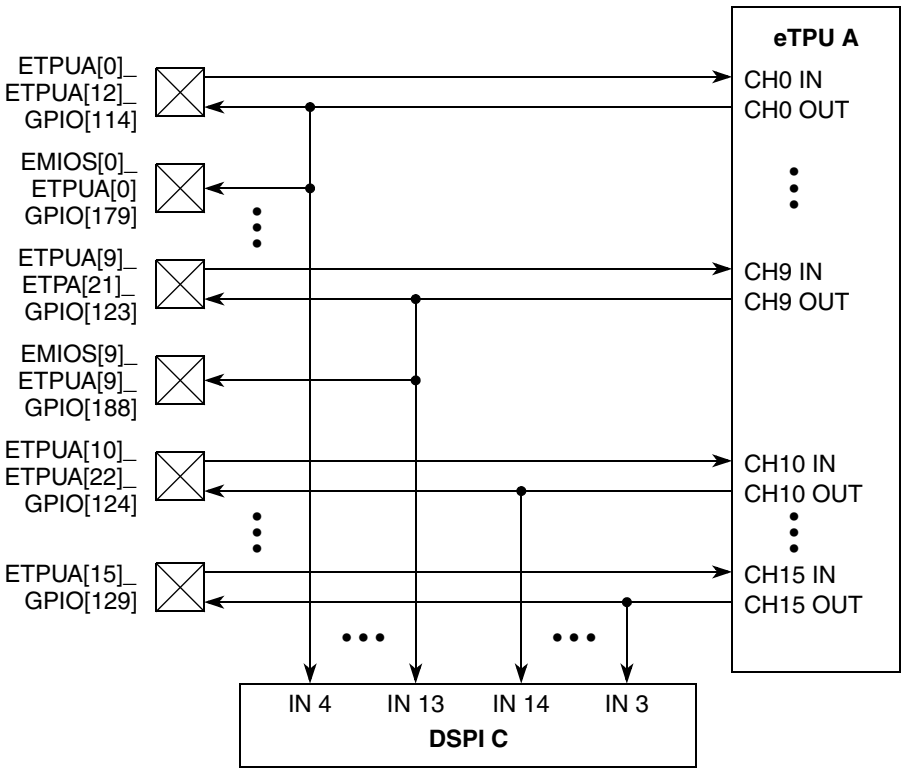


Figure 2-4. ETPUA[0:15]—DSPI C I/O Connections

Table 2-3. ETPUA[0:15]—DSPI C I/O Mapping

DSPI C Serialized Input	eTPU A Channel Output
15	11
14	10
13	9
12	8
11	7
10	6
9	5
8	4
7	3
6	2
5	1
4	0
3	15

Table 2-3. ETPUA[0:15]—DSPI C I/O Mapping (continued)

DSPI C Serialized Input	eTPU A Channel Output
2	14
1	13
0	12

2.4.2 ETPUA[16:31]

ETPUA[16:23,30:31] connect to external pins for both the input and output function. ETPUA[16:21,24:29] are serialized out on the DSPI B and DSPI D modules and ETPUA[22:23,30:31] are not serialized out. ETPUA[24:29] connect to external pins for only the output function. Figure 2-5 shows the connections for ETPUA[16] and applies to ETPUA[16:21]. Figure 2-6 shows the connections for ETPUA[24] and applies to ETPUA[24:29]. The full ETPUA to DSPI B connections are given in Table 2-4, and ETPUA to DSPI D in Table 2-5. Although not shown in Figure 2-5, the output channels of ETPUA[16:23] are also connected to the ETPUA[4:11]_ETPUA[16:23]_GPIO[118:125] pins.

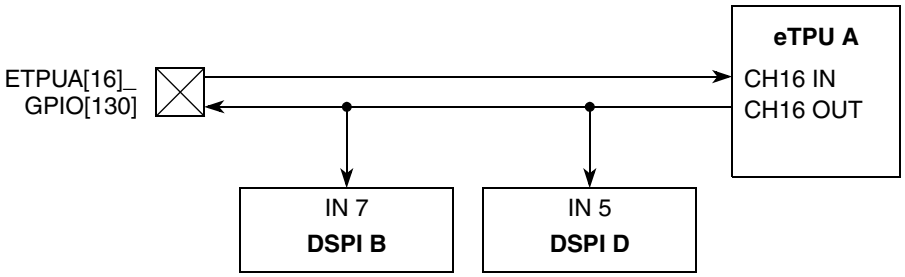


Figure 2-5. ETPUA[16:21]—DSPI B–DSPI D I/O Connections

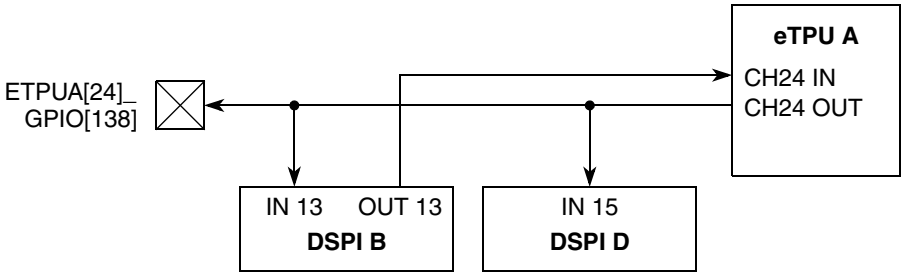


Figure 2-6. ETPU A[24:29]—DSPI B and DSPI D I/O Connections

Table 2-4. ETPU A[16:31]—DSPI B I/O Mapping

DSPI B Serialized Inputs / Outputs[1]	eTPU A Channel Output	eTPU A Channel Input
13	24	24
12	25	25
11	26	26

Table 2-4. ETPU A[16:31]—DSPI B I/O Mapping (continued)

DSPI B Serialized Inputs / Outputs[1]	eTPU A Channel Output	eTPU A Channel Input
10	27	27
9	28	28
8	29	29
7	16	—
6	17	—
5	18	—
4	19	—
3	20	—
2	21	—

1 DSPI B serialized input channels 0, 1, 14, and 15 are connected to EMIOS channels. DSPI B serialized output channels 14, 15 are connected to EMIOS channels. DSPI B serialized output channels 0–7 are not connected.

Table 2-5. ETPUA[16:31]—DSPI D I/O Mapping

DSPI D Serialized Inputs[1]	eTPU A Channel Output
15	24
14	25
13	26
12	27
11	28
10	29
5	16
4	17
3	18
2	19
1	20
0	21

1 DSPI D serialized input channels 6–9 are connected to EMIOS channels.

2.5 eMIOS Pin Connections and Serialization

The eMIOS channels connect to external pins or can be serialized in and out of the device. The input and output channels of EMIOS[0:11, 16:23] connect to pins. Only the output channels of EMIOS[12:15] connect to pins. The output channels of EMIOS[10:13] can be serialized out, and the inputs of EMIOS[12:15] can be serialized in. The DSPI connections for EMIOS[10:11] are given in Figure 2-7, Figure 2-8 for EMIOS[12:13], and Figure 2-9 for EMIOS[14:15].

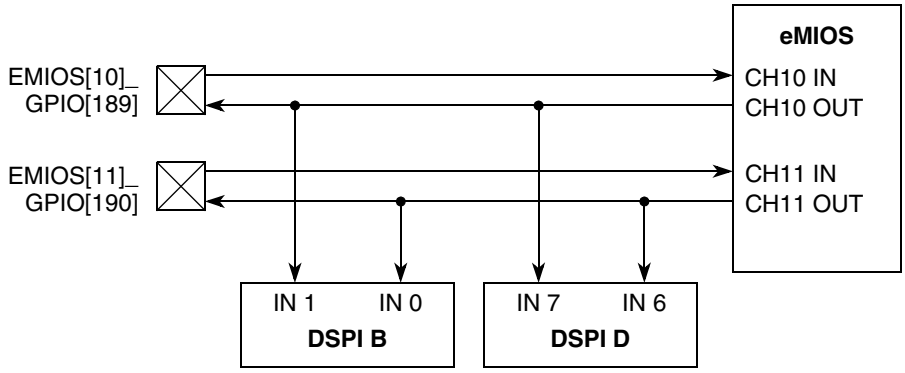


Figure 2-7. EMIOS[10:11]—DSPI B–DSPI D I/O Connections

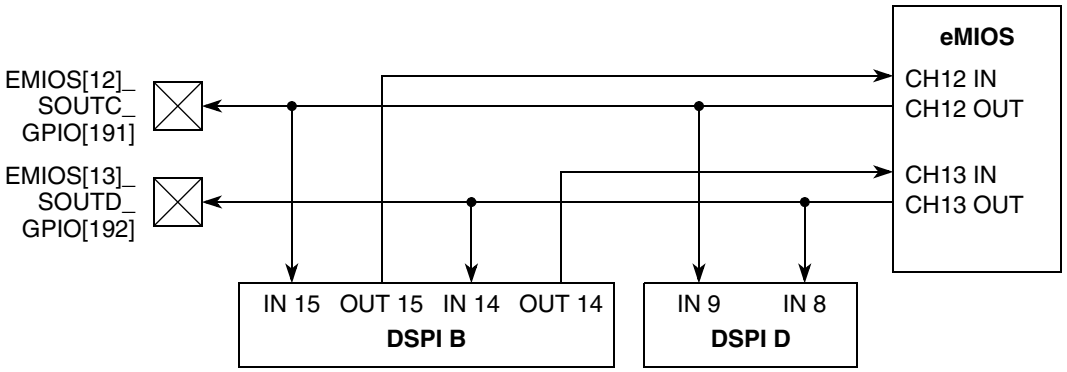


Figure 2-8. EMIOS[12:13]—DSPI B–DSPI D I/O Connections

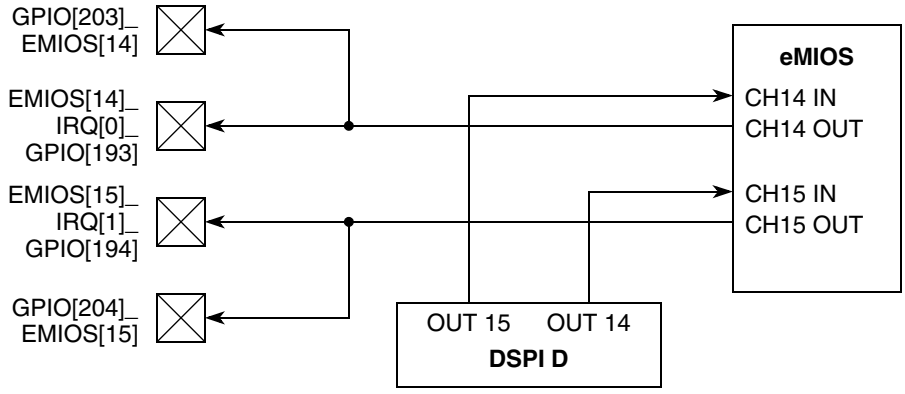


Figure 2-9. EMIOS[14:15]—DSPI D I/O Connections

Chapter 3
Core Complex (e200z3)

The e200z3 integrates a Z3 CPU core, a Memory Management Unit (MMU), a Signal Processing Extension (SPE) Auxiliary Processing Unit (APU), and a Nexus Class 3 real-time Debug unit. Separate Instruction and Data AHB 2.v6 system interfaces are provided. Overviews of the major components are described in this chapter.

Additional information:

- *e200z3 PowerPC Core Reference Manual*
- *EREF: A Programmer's Reference Manual for Freescale Book E Processors*
- *Variable-Length Encoding (VLE) Extension Programming Interface Manual*

3.1 Overview

The e200 processor family are a set of core devices that implement low-cost versions of the PowerPC Book E architecture. These processors are designed for deeply embedded control applications that require low cost solutions over maximum performance.

The initial e200z3 processor integrates an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable of sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by the branch unit to allow single-cycle branches in many cases.

The e200z3 core is a single-issue, 32-bit PowerPC Book E compliant design with 32 general purpose registers (GPRs). PowerPC Book E floating-point instructions are not supported by e200 in hardware, but are trapped and can be emulated by software. All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs).

A Signal Processing Extension (SPE) APU is provided to support real-time fixed point and single precision, embedded numerics operations using the general-purpose registers. All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs). The GPRs have been extended to 64-bits to support vector instructions defined by the SPE APU. These instructions operate on a vector pair of 16-bit or 32-bit data types, and deliver vector and scalar results.

In addition to the base PowerPC Book E instruction set, the e200z3 core also implements the VLE (Variable Length Encoding) APU, providing improved code density.

In the remainder of this chapter, the e200z3 core is also referred to as “the core.”

3.2 Features

The following is a list of some of the key features of the e200z3 core:

- 32-bit PowerPC Book E programmer's model
- Single issue, 32-bit PowerPC Book E compliant CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
  - Dedicated branch address calculation adder
  - Branch acceleration using Branch Lookahead Instruction Buffer
- Load/store unit
  - 1 cycle load latency
  - Fully pipelined
  - Big and Little endian support
  - Misaligned access support
  - Zero load-to-use pipeline bubbles
- Power management
  - Low power design
  - Dynamic power management of execution units
- Testability
  - Synthesizeable, full MuxD scan design
  - ABIST/MBIST for optional memory arrays

3.2.1 e200z3 Core Features Not Supported in the Device

This device implements a subset of the e200z3 core complex features. The e200z3 core complex features that are *not* supported in the device are described in Table 4-2.

Table 3-1. e200z3 Features Not Supported in the Device Core

Function / Category	Description
Disabled events	The external debug event (DEVT2) and unconditional debug event (UDE) are *not* supported
Power management	e200z3 core halted state and stopped state are *not* supported
Power management	The following low-power modes are *not* supported: • Doze mode • Nap mode • Sleep mode • Time-base interrupt wake-up from low-power mode is not supported
Power management	Core wake up is *not* supported MSR[WE] bit in the machine state register is *not* supported The OCR[WKUP] bit in the e200z3 OnCE control register (OCR) has no effect

Table 3-1. e200z3 Features Not Supported in the Device Core (continued)

Function / Category	Description
Machine check	The machine check input pin is *not* supported. HID0 [EMCP] has no effect, and MCSR[MCP] always reads a negated value.
PVR value	Least significant halfword of processor version register (PVR) is 0x0000, that contains the following bitfields: • MBG Use = 0x00 • MBG Rev = 0x0 • MBG ID = 0x0 The PVR register has two bitfields in the device.
Reservation management	Reservation management logic external to the e200z3 is *not* implemented.
Verification	The system version register (SVR) of the e200z3 is 0x 0000_0000.
Time Base	The decrement counters are always enabled in the e200z3.
	The timer external clock is *not* connected to a clock; Do *not* select the timer external clock.
Context control	The CTXCR and ALTCXTCR registers are *not* supported.

3.3 Microarchitecture Summary

The e200 processor utilizes a four stage pipeline for instruction execution. The Instruction Fetch (stage 1), Instruction Decode/Register file Read/Effective Address Calculation (stage 2), Execute/Memory Access (stage 3), and Register Writeback (stage 4) stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions.

The integer execution unit consists of a 32-bit Arithmetic Unit (AU), a Logic Unit (LU), a 32-bit Barrel shifter (Shifter), a Mask-Insertion Unit (MIU), a Condition Register manipulation Unit (CRU), a Count-Leading-Zeros unit (CLZ), a 32x32 Hardware Multiplier array, result feed-forward hardware, and support hardware for division.

Most arithmetic and logical operations are executed in a single cycle with the exception of the divide instructions. A Count-Leading-Zeros unit operates in a single clock cycle.

The Instruction Unit contains a PC incrementer and a dedicated Branch Address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching is performed to accelerate taken branches. Prefetched instructions are placed into an instruction buffer capable of holding six instructions.

Branches can also be decoded at the instruction buffer and branch target addresses calculated prior to the branch reaching the instruction decode stage, allowing the branch target to be prefetched early. When a branch is detected at the instruction buffer, a prediction can be made on whether the branch is taken or not. If the branch is predicted to be taken, a target fetch is initiated and its target instructions are placed in the instruction buffer following the branch instruction.

Conditional branches which are not taken and not folded execute in a single clock. Branches with successful target prefetching which are not folded have an effective execution time of one clock. All other taken branches have an execution time of two clocks.

Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The Condition Register unit supports the condition register (CR) and condition register operations defined by the PowerPC architecture. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

The SPE APU supports vector instructions operating on 16- and 32-bit fixed-point data types, as well as 32-bit IEEE-754 single-precision floating-point formats, and supports single-precision floating-point operations. The 64-bit general purpose register file is used for source and destination operands, and there is a unified storage model for single-precision floating-point data types of 32-bits and the normal integer type. Low latency fixed-point and floating-point add, subtract, multiply, multiply-add, multiply-subtract, divide, compare, and conversion operations are provided.

3.3.1 Instruction Unit Features

The features of the e200 Instruction unit are:

- 64-bit instruction fetch path supports fetching of two 32-bit instruction per clock, or up to four 16-bit VLE APU instructions per clock
- Instruction buffer holds up to six sequential instructions and two prefetched branch target instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch unit with dedicated branch address adder, and small branch target buffer logic supporting single cycle of execution of certain branches, two cycles for all others

3.3.2 Integer Unit Features

The e200 integer unit supports single cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count leading zero’s function
- 32-bit single cycle barrel shifter for static shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in ≤16 clocks with minimized execution timing
- 32x32 hardware multiplier array supports single-cycle 32x32→32 multiply

3.3.3 Load/Store Unit Features

The e200 load/store unit supports load, store, and the load multiple / store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- Dedicated 64-bit interface to memory supports saving and restoring of up to two registers per cycle for load multiple and store multiple word instructions

3.3.4 e200 System Bus Features

The features of the e200 System Bus interface are as follows:

- Independent Instruction and Data Buses
- AMBA AHB2.v6 protocol
- 32-bit address bus plus attributes and control on each bus
- 64-bit read data bus for Instruction Interface
- Separate unidirectional 64-bit read data bus and 64-bit write data bus for Data Interface
- Overlapped, in-order accesses

3.3.5 MMU Features

The features of the MMU are as follows:

- Virtual Memory support
- 32-bit Virtual and Physical Addresses
- 8-bit Process Identifier
- 16-entry Fully associative TLB
- Support for multiple page sizes from 4 KB to 256 MB
- Entry Flush Protection

3.3.6 Nexus 3 Features

The Nexus 3 module is compliant with Class 3 of the IEEE-ISTO 5001-2003 standard. The following features are implemented:

- Program Trace via Branch Trace Messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code can be traced.
- Data Trace via Data Write Messaging (DWM) and Data Read Messaging (DRM). This provides the capability for the development tool to trace reads and/or writes to selected internal memory resources.
- Ownership Trace via Ownership Trace Messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An Ownership Trace

Message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.

- Run-time access to embedded processor registers and memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint Messaging via the auxiliary pins
- Watchpoint Trigger enable of Program and/or Data Trace Messaging
- Auxiliary interface for higher data input/output
  - Configurable (min./max) Message Data Out pins (MDO[11:0])
  - One or two Message Start/End Out pins ($\overline{\text{MSEO}}$[1:0])
  - One Read/Write Ready pin ($\overline{\text{RDY}}$) pin
  - One Watchpoint Event pin ($\overline{\text{EVTO}}$)
  - One Event In pin ($\overline{\text{EVTI}}$)
  - One Message Clock Out (MCKO) pin
- Registers for Program Trace, Data Trace, Ownership Trace and Watchpoint Trigger.
- All features controllable and configurable via the JTAG port

3.4 Block Diagram

Figure 4-1 shows a block diagram of the e200z3 core complex.

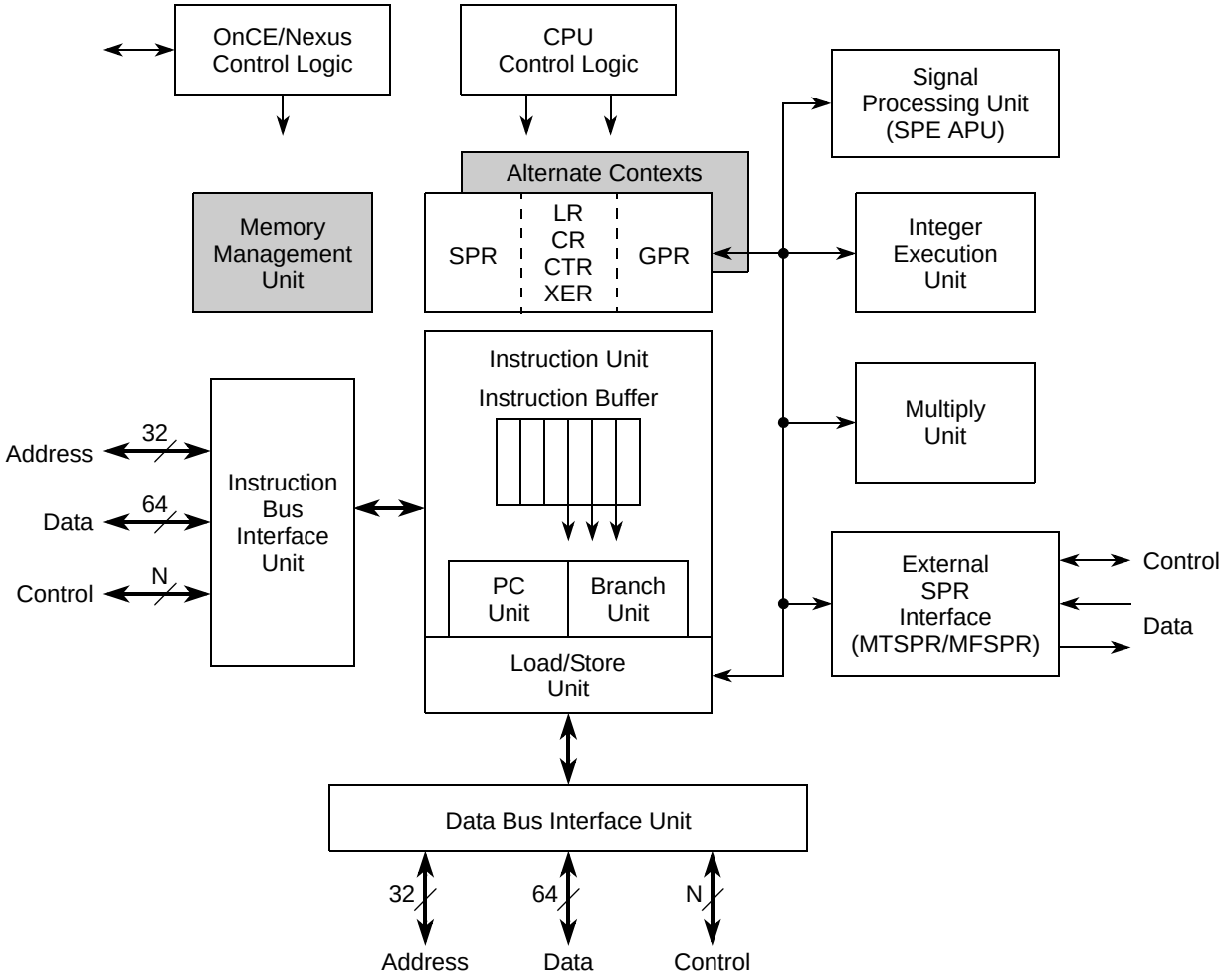


Figure 3-1. e200z3 Block Diagram

3.5 Memory Management Unit (MMU)

3.5.1 Overview

The e200z3 Memory Management Unit is a 32-bit PowerPC Book E compliant implementation, with the following feature set:

- Freescale Book E MMU architecture compliant
- Translates from 32-bit effective to 32-bit real addresses
- 16-entry fully associative TLB with support for nine page sizes (4 K, 16 K, 64 K, 256 K, 1 M, 4 M, 16 M, 64 M, 256 M)
- Hardware assist for TLB miss exceptions
- Software managed by **tlbre**, **tlbwe**, **tlbsx**, **tlbsync**, and **tlbivax** instructions

3.5.2 Translation Lookaside Buffer (TLB)

The Freescale Book E architecture defines support for zero or more TLBs in an implementation, each with its own characteristics, and provides configuration information for software to query the existence and structure of the TLB(s) through a set of special purpose registers: MMUCFG, TLB0CFG, TLB1CFG, etc. By convention, TLB0 is used for a set associative TLB with fixed page sizes, TLB1 is used for a fully associative TLB with variable page sizes, and TLB2 is arbitrarily defined by an implementation. The e200z3 MMU supports a single TLB which is fully associative and supports variable page sizes, thus it corresponds to TLB1. For the rest of this document, TLB, TLBCAM, and TLB1 are used interchangeably.

The TLB consists of a 16-entry, fully associative CAM array with support for nine page sizes. To perform a lookup, the CAM is searched in parallel for a matching TLB entry. The contents of this TLB entry are then concatenated with the page offset of the original effective address. The result is the physical address of the access.

A hit to multiple TLB entries is considered to be a programming error. If this occurs, the TLB generates an invalid address and TLB entries can be corrupted (an exception is not reported).

Table 4-3 shows the TLB entry bit definitions.

Table 3-2. TLB Entry Bit Definitions

Field	Comments
V	Valid bit for entry
TS	Translation address space (compared against AS bit)
TID[0:7]	Translation ID (compared against PID0 or ‘0’)
EPN[0:19]	Effective page number (compared against effective address)
RPN[0:19]	Real page number (translated address)
SIZE[0-3]	Page size (4 KB, 16 KB, 64 KB, 256 KB, 1 MB, 4 MB, 16 MB, 64 MB, 256 MBs)
SX, SW, SR	Supervisor execute, write, and read permission bits
UX, UW, UR	User execute, write, and read permission bits
WIMGE	Translation attributes (Write-through required, cache-Inhibited, Memory coherence required, Guarded, Endian)
U0–U3	Use bits for software
IPROT	Invalidation protect
VLE	VLE page indicator

The TLB is accessed indirectly through several MMU Assist (MAS) registers. Software can write and read the MMU Assist registers with **mtspr** and **mfspr** instructions. These registers contain information related to reading and writing a given entry within the TLB. Data is read from the TLB into the MAS registers with a **tlbre** (TLB read entry) instruction. Data is written to the TLB from the MAS registers with a **tlbwe** (TLB write entry) instruction.

Certain fields of the MAS registers are also written by hardware when an Instruction TLB Error, Data TLB Error, DSI, or ISI interrupt occurs.

On a TLB Error interrupt, the MAS registers are written by hardware with the proper EA, default attributes (TID, WIMGE, permissions, etc.), and TLB selection information, and an entry in the TLB to replace. Software manages this entry selection information by updating a replacement entry value during TLB miss handling. Software must provide the correct RPN and permission information in one of the MAS registers before executing a **tlbwe** instruction.

On taking a DSI or ISI interrupt, the hardware updates only the search PID (SPID) and search address space (SAS) fields in the MAS registers using PID0, and appropriate MSR[IS] or MSR[DS] values which were used when the DSI or ISI exception was recognized. During the interrupt handler, software can issue a TLB search instruction (**tlbsx**), which uses the SPID field along with the SAS field, to determine the entry related to the DSI or ISI exception. (It is possible that the entry which caused the DSI or ISI interrupt no longer exists in the TLB by the time the search occurs if a TLB invalidate or replacement removes the entry between the time the exception is recognized and when the **tlbsx** is executed.)

The **tlbre**, **tlbwe**, **tlbsx**, **tlbivax**, and **tlbsync** instructions are privileged.

3.5.3 Translation Flow

The effective address, concatenated with the address space value of the corresponding MSR bit (MSR[IS] or MSR[DS], is compared to the appropriate number of bits of the EPN field (depending on the page size) and the TS field of TLB entries. If the contents of the effective address plus the address space bit matches the EPN field and TS bit of the TLB entry, that TLB entry is a candidate for a possible translation match. In addition to a match in the EPN field and TS, a matching TLB entry must match with the current Process ID of the access (in PID0), or have a TID value of 0, indicating the entry is globally shared among all processes.

Figure 4-2 shows the translation match logic for the effective address plus its attributes, collectively called the virtual address, and how it is compared with the corresponding fields in the TLB entries.

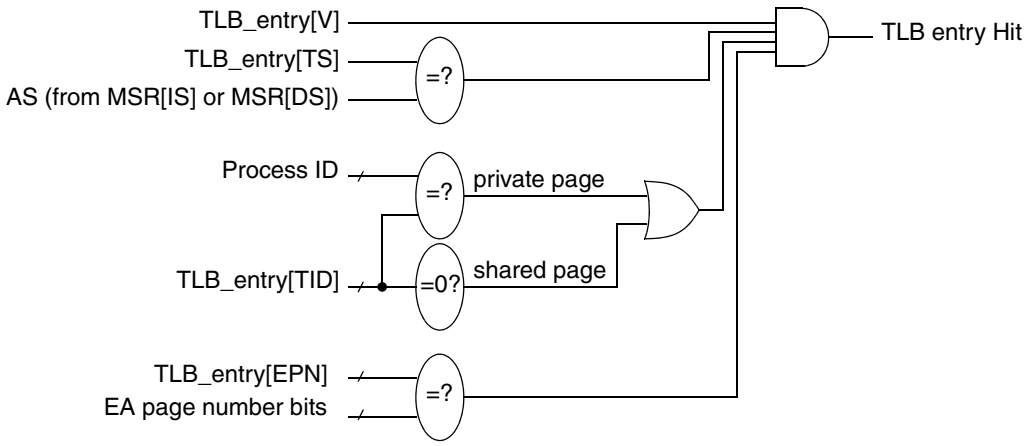


Figure 3-2. Virtual Address and TLB-Entry Compare Process

The page size for a TLB entry determines how many bits of the effective address are compared with the corresponding EPN field in the TLB entry as shown in Table 3-3. On a TLB hit, the corresponding bits of the Real Page Number (RPN) field are used to form the real address.

Table 3-3. Page Size and EPN Field Comparison

SIZE Field	Page Size (4^{SIZE} KBs)	EA to EPN Comparison
0001	4 KB	EA[0:19] =? EPN[0:19]
0010	16 KB	EA[0:17] =? EPN[0:17]
0011	64 KB	EA[0:15] =? EPN[0:15]
0100	256 KB	EA[0:13] =? EPN[0:13]
0101	1 MB	EA[0:11] =? EPN[0:11]
0110	4 MB	EA[0:9] =? EPN[0:9]
0111	16 MB	EA[0:7] =? EPN[0:7]
1000	64 MB	EA[0:5] =? EPN[0:5]
1001	256 MB	EA[0:3] =? EPN[0:3]

On a TLB hit, the generation of the physical address occurs as shown in Figure 3-3.

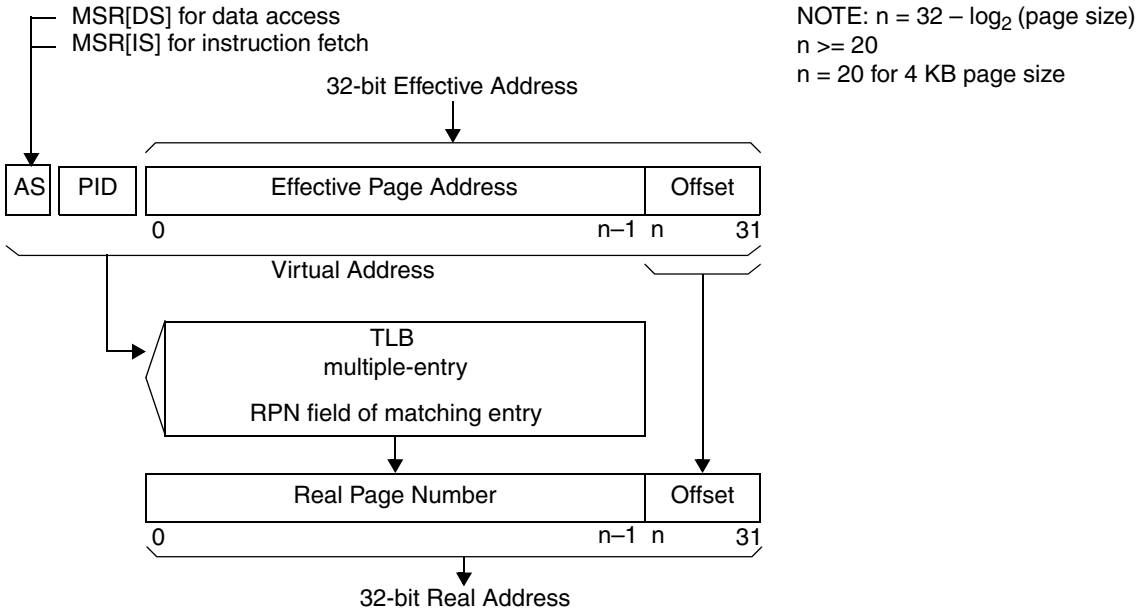


Figure 3-3. Effective to Real Address Translation Flow

3.5.4 Permissions

An operating system can restrict access to virtual pages by selectively granting permissions for user mode read, write, and execute, and supervisor mode read, write, and execute on a per page basis. These permissions can be set up for a particular system (for example, program code might be execute-only, data structures can be mapped as read/write/no-execute) and can also be changed by the operating system based on application requests and operating system policies.

The UX, SX, UW, SW, UR, and SR access control bits are provided to support selective permissions (access control):

- SR—Supervisor read permission. Allows loads and load-type cache management instructions to access the page while in supervisor mode (MSR[PR=0]).
- SW—Supervisor write permission. Allows stores and store-type cache management instructions to access the page while in supervisor mode (MSR[PR=0]).
- SX—Supervisor execute permission. Allows instruction fetches to access the page and instructions to be executed from the page while in supervisor mode (MSR[PR=0]).
- UR—User read permission. Allows loads and load-type cache management instructions to access the page while in user mode (MSR[PR=1]).
- UW—User write permission. Allows stores and store-type cache management instructions to access the page while in user mode (MSR[PR=1]).
- UX—User execute permission. Allows instruction fetches to access the page and instructions to be executed from the page while in user mode (MSR[PR=1]).

If the translation match was successful, the permission bits are checked as shown in Figure 3-4. If the access is not allowed by the access permission mechanism, the processor generates an Instruction or Data Storage interrupt (ISI or DSI).

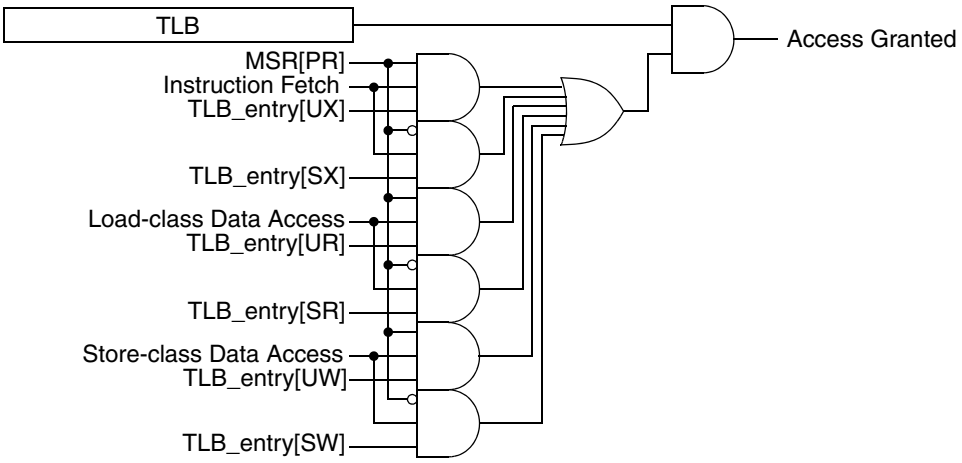


Figure 3-4. Granting of Access Permission

3.6 Bus Interface Unit (BIU)

The BIU encompasses control and data signals supporting instruction and data transfers. The memory interface supported by the BIU is based on the AMBA AHB-Lite subset of the AMBA 2.0 AHB, with V6 AMBA Extensions. (Ref. documents ARM IHI 0011A, ARM DVI 0044A, and ARM PR022-GENC-001011 0.7). Additional sideband signals have been added to support additional control functions.

NOTE

The AMBA AHB bit and byte ordering reflect a natural little-endian ordering, as used by the AMBA documentation. The e200z3 BIU automatically performs byte lane conversions for big-endian transfers.

Single-beat and misaligned transfers are supported for read and write cycles, and write-buffer writes.

3.7 Core Registers and Programmer's Models

This section describes the registers implemented in the e200z3 core. It includes an overview of registers defined by the PowerPC Book E architecture, highlighting differences in how these registers are implemented in the e200 core, and provides a detailed description of e200-specific registers. Full descriptions of the architecture-defined register set are provided in *Book E: Enhanced PowerPC*^{tm} *Architecture*.

The PowerPC Book E architecture defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions. Data is transferred between memory and registers with explicit load and store instructions only.

e200z3 extends the General Purpose Registers to 64-bits for supporting SPE APU operations. PowerPC Book E instructions operate on the lower 32 bits of the GPRs only, and the upper 32 bits are unaffected by these instructions. SPE vector instructions operate on the entire 64-bit register. The SPE APU defines load and store instructions for transferring 64-bit values to/from memory.

NOTE

e200z3 is a 32-bit implementation of the PowerPC Book E architecture. In this document, register bits are sometimes numbered from bit 31 (Most Significant Bit) to 0 (Least Significant Bit), rather than the Book E numbering scheme of 32:63, thus register bit numbers for some registers in Book E are 32 higher. Where appropriate, the Book E defined bit numbers are shown in parentheses.

Figure 3-5 and Figure 3-6 show the complete e200 register set. Figure 3-5 shows the registers which are accessible while in supervisor mode.

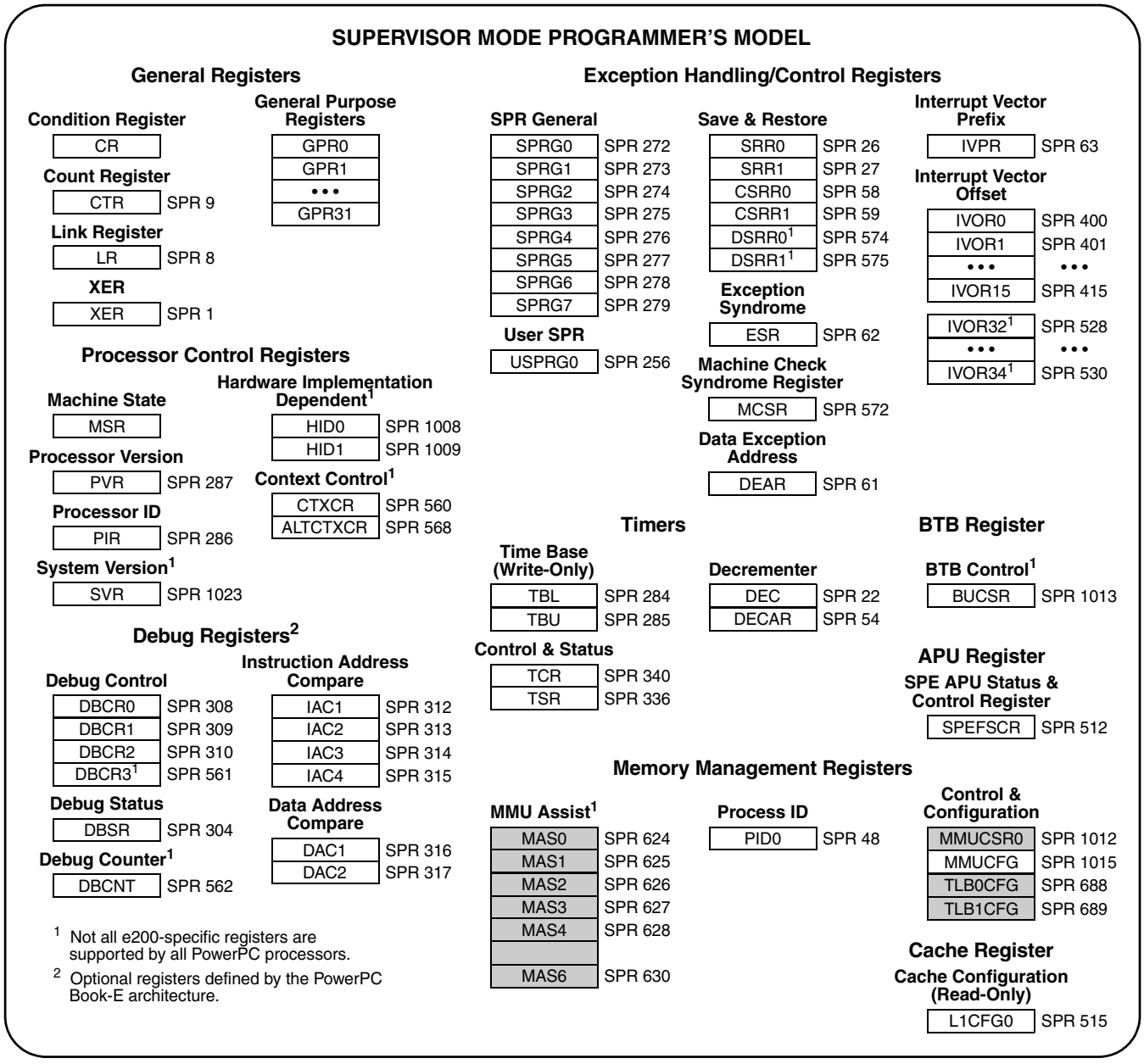


Figure 3-5. e200z3 Supervisor Mode Programmer's Model

Figure 3-6 shows the set of registers which are accessible while in user mode. The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register (for example, the integer exception register (XER) is SPR 1).

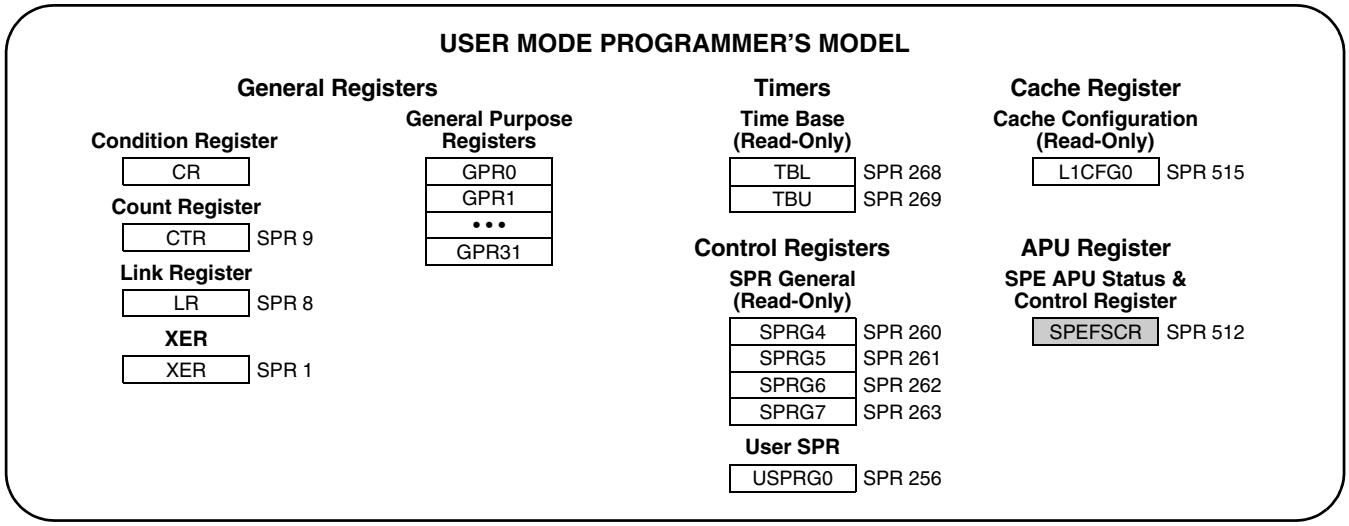


Figure 3-6. e200 User Mode Programmer's Model

General purpose registers (GPRs) are accessed through instruction operands. Access to other registers can be explicit (by using instructions for that purpose such as Move to Special Purpose Register (**mtspr**) and Move from Special Purpose Register (**mfspr**) instructions) or implicit as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

3.7.1 PowerPC Book E Registers

e200 supports most of the registers defined by *Book E: Enhanced PowerPC™ Architecture*. Notable exceptions are the Floating Point registers FPR0-FPR31 and FPSCR. e200 does not support the Book E Floating Point Architecture in hardware. The General Purpose registers have been extended to 64-bits. The e200 supported PowerPC BookE registers are described as follows:

3.7.1.1 User-level Registers

The user-level registers can be accessed by all software with either user or supervisor privileges, and are grouped as follows:

- *General-purpose registers (GPRs)*—The thirty-two 64-bit GPRs (GPR0–GPR31) are data source or destination registers for integer instructions, and provide data to generate addresses.
- *Condition register (CR)*—The 32-bit CR consists of eight 4-bit fields, (CR0–CR7), that reflect the results of arithmetic operations and are used for testing and branching.

  The remaining user-level registers are SPRs. The PowerPC architecture has the **mtspr** and **mfspr** instructions for accessing SPRs.
- *Integer exception register (XER)*—The XER indicates overflow and carries for integer operations.

- *Link register (LR)*—The LR provides the branch target address for the Branch Conditional to Link Register (**bclr**, **bclrl**) instructions, and is used to hold the address of the instruction that follows a branch and link instruction, typically used for linking to subroutines.
- *Count register (CTR)*—The CTR holds a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR also provides the branch target address for the Branch Conditional to Count Register (**bcctr**, **bcctrl**) instructions.
- *Time Base facility (TB)*— consists of two 32-bit registers—Time Base Upper (TBU) and Time Base Lower (TBL). These two registers are accessible in a read-only fashion to user-level software.
- *Software-use Special Purpose Registers (SPRGs)*—The PowerPC Book E architecture defines how software uses the SPRG4 through SPRG7 (SPRG4–SPRG7). These registers are read-only and are accessed by user-level software. The e200z3 does not allow user mode access to the SPRG3 register (defined as implementation-dependent by Book E).
- *User Software-Use Special Purpose Register (USPRG0)*—The PowerPC Book E architecture defines USPRG0 as a read-write register that is accessible by user-level software.

3.7.1.2 Supervisor-level Registers

In addition to the registers accessible in user mode, Supervisor-level software has access to additional control and status registers used for configuration, exception handling, and other operating system functions. The PowerPC Book E architecture defines the following supervisor-level registers:

- **Processor Control registers**
  - Machine State Register (MSR). The MSR defines the state of the processor. The MSR can be modified by the Move to Machine State Register (**mtmsr**), System Call (**sc)**, and Return from Exception (**rfi, rfci, rfdi)** instructions. It can be read by the Move from Machine State Register (**mfmsr)** instruction. When an interrupt occurs, the contents of the MSR are saved to one of the machine state save/restore registers (SRR1, CSRR1, DSRR1).
  - Processor version register (PVR). This register is a read-only register that identifies the version (model) and revision level of the PowerPC processor.
  - Processor Identification Register (PIR). This read-only register is provided to distinguish the processor from other processors in the system.
- **Storage Control register**
  - Process ID Register (PID, also referred to as PID0). This register is provided to indicate the current process or task identifier. It is used by the MMU as an extension to the effective address, and by Nexus 2 module for Ownership Trace message generation. PowerPC Book E allows for multiple PIDs; e200z3 implements only one.
- **Interrupt Registers**
  - Data Exception Address Register (DEAR). After most Data Storage Interrupt (DSI), or on an Alignment Interrupt, or Data TLB Miss Interrupt, the DEAR is set to the effective address (EA) generated by the faulting instruction.
  - SPRG0–SPRG7, USPRG0. The SPRG0–SPRG7 and USPRG0 registers are used by thee operating system. e200 does not allow user mode access to the SPRG3 register (defined as implementation dependent by Book E).

  - Exception Syndrome Register (ESR). The ESR register provides a syndrome to differentiate between the different kinds of exceptions which can generate the same interrupt.
  - Interrupt Vector Prefix Register (IVPR) and the Interrupt Vector Offset Registers (IVOR0–IVOR15, IVOR32–IVORxx). These registers together provide the address of the interrupt handler for different classes of interrupts.
  - Save/Restore Register 0 (SRR0). The SRR0 register is used to save machine state on a non-critical interrupt, and contains the address of the instruction at which execution resumes when an **rfi** instruction is executed at the end of a non-critical class interrupt handler routine.
  - Critical Save/Restore register 0 (CSRR0). The CSRR0 register is used to save machine state on a critical interrupt, and contains the address of the instruction at which execution resumes when an **rfci** instruction is executed at the end of a critical class interrupt handler routine.
  - Save/Restore register 1 (SRR1). The SRR1 register is used to save machine state from the MSR on non-critical interrupts, and to restore machine state when **rfi** executes.
  - Critical Save/Restore register 1 (CSRR1). The CSRR1 register is used to save machine state from the MSR on critical interrupts, and to restore machine state when **rfci** executes.
- Debug facility registers
  - Debug Control Registers (DBCR0–DBCR2). These registers provide control for enabling and configuring debug events.
  - Debug Status Register (DBSR). This register contains debug event status.
  - Instruction Address Compare registers (IAC1–IAC4). These registers contain addresses and/or masks which are used to specify Instruction Address Compare debug events.
  - Data address compare registers (DAC1–2). These registers contain addresses and/or masks which are used to specify Data Address Compare debug events.
  - e200 does **not** implement the Data Value Compare registers (DVC1 and DVC2).
- Timer Registers
  - The clock inputs for the timers are connected to the internal system clock.
  - Time base (TB). The TB is a 64-bit structure provided for maintaining the time of day and operating interval timers. The TB consists of two 32-bit registers, Time Base Upper (TBU) and Time Base Lower (TBL). The Time Base registers can be written to only by supervisor-level software, but can be read by both user and supervisor-level software.
  - Decrementer register (DEC). This register is a 32-bit decrementing counter that provides a mechanism for causing a decrementer exception after a programmable delay.
  - Decrementer Auto-Reload (DECAR). This register is provided to support the auto-reload feature of the Decrementer.
  - Timer Control Register (TCR). This register controls Decrementer, Fixed-Interval Timer, and Watchdog Timer options.
  - Timer Status Register (TSR). This register contains status on timer events and the most recent Watchdog Timer-initiated processor reset.

More details about these registers can be found in the PowerPC Book E architecture specifications.

3.7.2 e200-specific Registers

The PowerPC Book E architecture allows implementation-specific registers. Those incorporated in the e200 core are described in the following sections.

3.7.2.1 User-level Registers

The user-level registers can be accessed by all software with either user or supervisor privileges. They include the following:

- Signal Processing Extension APU status and control register (SPEFSCR). The SPEFSCR contains all fixed-point and floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE 754 standard.
- The L1 Cache Configuration register (L1CFG0). This read-only register allows software to query the configuration of the L1 cache.

3.7.2.2 Supervisor-level Registers

The following supervisor-level registers are defined in e200 in addition to the PowerPC Book E registers described above:

- Configuration Registers
  - Hardware implementation-dependent register 0 (HID0). This register controls various processor and system functions.
  - Hardware implementation-dependent register 1 (HID1). This register controls various processor and system functions.
- Exception Handling and Control Registers
  - Machine Check Syndrome register (MCSR). This register provides a syndrome to differentiate between the different kinds of conditions which can generate a Machine Check.
  - Debug Save/Restore register 0 (DSRR0). When enabled, the DSRR0 register is used to save the address of the instruction at which execution continues when **rfdi** executes at the end of a debug interrupt handler routine.
  - Debug Save/Restore register 1 (DSRR1). When enabled, the DSRR1 register is used to save machine status on debug interrupts and to restore machine status when **rfdi** executes.
- Debug Facility Registers
  - Debug Control Register 3 (DBCR3)—This register provides control for debug functions not described in PowerPC Book E architecture.
  - Debug Counter Register (DBCNT)—This register provides counter capability for debug functions.
- Branch Unit Control and Status Register (BUCSR) controls operation of the optional BTB. If an optional BTB is not present, this register returns all zeros.
- L1 Cache Configuration Register (L1CFG0) is a read-only register that allows software to query the configuration of the L1 Cache. This register returns all zeros.
- MMU Configuration Register (MMUCFG) is a read-only register that allows software to query the configuration of the MMU.

- Memory Management Unit Registers
  - MMU Assist (MAS0-MAS4, MAS6) registers. These registers provide the interface to the core from the Memory Management Unit.
  - MMU Control and Status Register (MMUCSR0) controls invalidation of the MMU.
  - TLB Configuration Registers (TLB0CFG, TLB1CFG) are read-only registers that allow software to query the configuration of the TLBs.
- System version register (SVR). This register is a read-only register that identifies the version (model) and revision level of the System which includes an e200 PowerPC processor.

It is not guaranteed that the implementation of e200 core-specific registers is consistent among PowerPC processors, although other processors can implement similar or identical registers. More details about these registers are in the *e200z3 PowerPC Core Reference Manual*.

3.8 Signal Processing Extension APU (SPE APU)

3.8.1 Overview

The e200z3 core provides a register file with thirty-two 64-bit registers. The PowerPC 32-bit Book E instructions operate on the lower (least significant) 32 bits of the 64-bit register. New SPE instructions are defined that view the 64-bit register as being composed of a vector of two 32-bit elements, and some of the instructions also read or write 16-bit elements. These new instructions can also be used to perform scalar operations by ignoring the results of the upper 32-bit half of the register file. Some instructions are defined that produce a 64-bit scalar result. Vector fixed-point instructions operate on a vector of two 32-bit or four 16-bit fixed-point numbers resident in the 64-bit GPRs. Vector floating-point instructions operate on a vector of two 32-bit single-precision floating-point numbers resident in the 64-bit GPRs. Scalar floating-point instructions operate on the lower half of GPRs. These single-precision floating-point instructions do not have a separate register file; there is a single shared register file for all instructions. The SPE and Book E instructions issue from a single instruction stream. Figure 3-7 shows two different representations of the 64-bit GPRs. The shaded half is the only region operated on by the 32-bit PowerPC instructions.

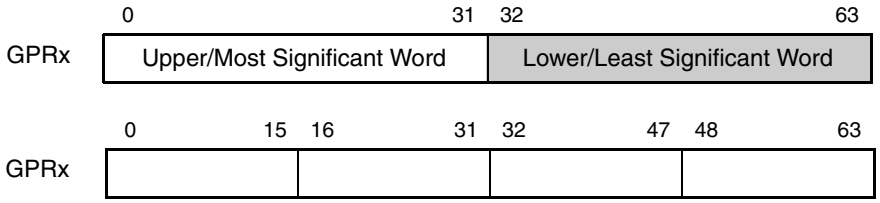


Figure 3-7. 64-bit General Purpose Registers

3.8.2 SPE Programming Model

Not all SPE instructions record events such as overflow, saturation and negative/positive result. See the description of the individual SPE instruction in the *e200z3 PowerPC Core Reference Manual* for information on which conditions are recorded and where they are recorded. Most SPE instructions record

conditions to the SPEFSCR. Vector compare instructions store the result of the comparison into the condition register (CR).

The e200z3 core has a 64-bit architectural accumulator register that holds the results of the SPE multiply accumulate (MAC) fixed-point instructions. The accumulator allows back-to-back execution of dependent fixed-point MAC instructions, something that is found in the inner loops of DSP code such as filters. The accumulator is partially visible to the programmer in that its results do not have to be explicitly read to use them. Instead, they are always copied into a 64-bit destination GPR specified as part of the instruction. The accumulator however, has to be explicitly cleared when starting a new MAC loop. Based upon the type of instruction, an accumulator can hold either a single 64-bit value or a vector of two 32-bit elements.

3.9 Instruction Summary

In addition to the PowerPC Book E instructions, the MPC5534 supports e200 core specific instructions, SPE APU instructions and VLE instructions.

See the *PowerPC Microprocessor Family: The Programming Environment for 32-bit Microprocessors*, the *e200z3 PowerPC Core Reference Manual* and the *Variable-Length Encoding (VLE) Extension Programming Interface Manual* documents.

3.9.1 SPE APU Simple and Complex Integer Instructions

The SPE APU supports both scalar and vector integer instructions. The instructions are grouped into two categories according to the latency and throughput; Simple Integer Instructions and Complex Integer Instructions.

The SPE APU Simple Integer Instructions perform operations such as addition, subtraction, logical operations, rotate, shift, compare, round, merge and swap, and sign or zero-extend. Table 3-4 briefly describes the SPE Simple Integer Instructions.

Table 3-4. SPE Simple Integer Instructions

Instruction	Description
brinc	Bit Reversed Increment
evabs	Vector Absolute Value
evaddiw	Vector Add Immediate Word
evaddw	Vector Add Word
evand	Vector AND
evandc	Vector AND with Complement
evcmpeq	Vector Compare Equal
evcmpgts	Vector Compare Greater Than Signed
evcmpgtu	Vector Compare Greater Than Unsigned
evcmplts	Vector Compare Less Than Signed
evcmpltu	Vector Compare Less Than Unsigned

Table 3-4. SPE Simple Integer Instructions (continued)

Instruction	Description
evcntlsw	Vector Count Leading Sign Bits Word
evcntlzw	Vector Count Leading Zeros Word
evdivws	Vector Divide Word Signed
evdivwu	Vector Divide Word Unsigned
eveqv	Vector Equivalent
evextsb	Vector Extend Sign Byte
evextsh	Vector Extend Sign Half Word
evmergehi	Vector Merge High
evmergehilo	Vector Merge High/Low
evmergelo	Vector Merge Low
evmergelohi	Vector Merge Low/High
evnand	Vector NAND
evneg	Vector Negate
evnor	Vector NOR
evor	Vector OR
evorc	Vector OR with Complement
evrlw	Vector Rotate Left Word
evrlwi	Vector Rotate Left Word Immediate
evrndw	Vector Round Word
evsel	Vector Select
evslw	Vector Shift Left Word
evslwi	Vector Shift Left Word Immediate
evsplatfi	Vector Splat Fractional Immediate
evsplati	Vector Splat Immediate
evsrwis	Vector Shift Right Word Immediate Signed
evsrwiu	Vector Shift Right Word Immediate Unsigned
evsrws	Vector Shift Right Word Signed
evsrwu	Vector Shift Right Word Unsigned
evsubfw	Vector Subtract from Word
evsubifw	Vector Subtract Immediate from Word
evxor	Vector XOR

The SPE APU Complex Integer Instructions perform operations such as multiplication, division, and multiply and accumulate. Table 3-5 briefly describes the SPE Complex Integer Instructions.

Table 3-5. SPE Complex Integer Instructions

Instruction	Description
evaddsmiaaw	Vector Add Signed, Modulo, Integer to Accumulator Word
evaddssiaaw	Vector Add Signed, Saturate, Integer to Accumulator Word
evaddumiaaw	Vector Add Unsigned, Modulo, Integer to Accumulator Word
evaddusiaaw	Vector Add Unsigned, Saturate, Integer to Accumulator Word
evdivws	Vector Divide Word Signed
evdivwu	Vector Divide Word Unsigned
evmhegsmfaa	Vector Multiply Half Words, Even, Guarded, Signed, Modulo, Fractional and Accumulate
evmhegsmfan	Vector Multiply Half Words, Even, Guarded, Signed, Modulo, Fractional and Accumulate Negative
evmhegsmiaa	Vector Multiply Half Words, Even, Guarded, Signed, Modulo, Integer and Accumulate
evmhegsmian	Vector Multiply Half Words, Even, Guarded, Signed, Modulo, Integer and Accumulate Negative
evmhegumiaa	Vector Multiply Half Words, Even, Guarded, Unsigned, Modulo, Integer and Accumulate
evmhegumian	Vector Multiply Half Words, Even, Guarded, Unsigned, Modulo, Integer and Accumulate Negative
evmhesmf	Vector Multiply Half Words, Even, Signed, Modulo, Fractional
evmhesmfa	Vector Multiply Half Words, Even, Signed, Modulo, Fractional, to Accumulator
evmhesmfaaw	Vector Multiply Half Words, Even, Signed, Modulo, Fractional and Accumulate into Words
evmhesmfanw	Vector Multiply Half Words, Even, Signed, Modulo, Fractional and Accumulate Negative into Words
evmhesmi	Vector Multiply Half Words, Even, Signed, Modulo, Integer
evmhesmia	Vector Multiply Half Words, Even, Signed, Modulo, Integer, to Accumulator
evmhesmiaaw	Vector Multiply Half Words, Even, Signed, Modulo, Integer and Accumulate into Words
evmhesmianw	Vector Multiply Half Words, Even, Signed, Modulo, Integer and Accumulate Negative into Words
evmhessf	Vector Multiply Half Words, Even, Signed, Saturate, Fractional
evmhessfa	Vector Multiply Half Words, Even, Signed, Saturate, Fractional, to Accumulator
evmhessfaaw	Vector Multiply Half Words, Even, Signed, Saturate, Fractional and Accumulate into Words
evmhessfanw	Vector Multiply Half Words, Even, Signed, Saturate, Fractional and Accumulate Negative into Words
evmhessiaaw	Vector Multiply Half Words, Even, Signed, Saturate, Integer and Accumulate into Words
evmhessianw	Vector Multiply Half Words, Even, Signed, Saturate, Integer and Accumulate Negative into Words
evmheumi	Vector Multiply Half Words, Even, Unsigned, Modulo, Integer
evmheumia	Vector Multiply Half Words, Even, Unsigned, Modulo, Integer, to Accumulator
evmheumiaaw	Vector Multiply Half Words, Even, Unsigned, Modulo, Integer and Accumulate into Words
evmheumianw	Vector Multiply Half Words, Even, Unsigned, Modulo, Integer and Accumulate Negative into Words
evmheusiaaw	Vector Multiply Half Words, Even, Unsigned, Saturate, Integer and Accumulate into Words

Table 3-5. SPE Complex Integer Instructions (continued)

Instruction	Description
evmheusianw	Vector Multiply Half Words, Even, Unsigned, Saturate, Integer and Accumulate Negative into Words
evmhogsmfaa	Multiply Half Words, Odd, Guarded, Signed, Modulo, Fractional and Accumulate
evmhogsmfan	Multiply Half Words, Odd, Guarded, Signed, Modulo, Fractional and Accumulate Negative
evmhogsmiaa	Multiply Half Words, Odd, Guarded, Signed, Modulo, Integer and Accumulate
evmhogsmian	Multiply Half Words, Odd, Guarded, Signed, Modulo, Integer and Accumulate Negative
evmhogumiaa	Multiply Half Words, Odd, Guarded, Unsigned, Modulo, Integer and Accumulate
evmhogumian	Multiply Half Words, Odd, Guarded, Unsigned, Modulo, Integer and Accumulate Negative
evmhosmf	Vector Multiply Half Words, Odd, Signed, Modulo, Fractional
evmhosmfa	Vector Multiply Half Words, Odd, Signed, Modulo, Fractional, to Accumulator
evmhosmfaaw	Vector Multiply Half Words, Odd, Signed, Saturate, Fractional and Accumulate into Words
evmhosmfanw	Vector Multiply Half Words, Odd, Signed, Saturate, Fractional and Accumulate Negative into Words
evmhosmi	Vector Multiply Half Words, Odd, Signed, Modulo, Integer
evmhosmia	Vector Multiply Half Words, Odd, Signed, Modulo, Integer, to Accumulator
evmhosmiaaw	Vector Multiply Half Words, Odd, Signed, Modulo, Integer and Accumulate into Words
evmhosmianw	Vector Multiply Half Words, Odd, Signed, Modulo, Integer and Accumulate Negative into Words
evmhossf	Vector Multiply Half Words, Odd, Signed, Saturate, Fractional
evmhossfa	Vector Multiply Half Words, Odd, Signed, Saturate, Fractional, to Accumulator
evmhossfaaw	Vector Multiply Half Words, Odd, Signed, Saturate, Fractional and Accumulate into Words
evmhossfanw	Vector Multiply Half Words, Odd, Signed, Saturate, Fractional and Accumulate Negative into Words
evmhossiaaw	Vector Multiply Half Words, Odd, Signed, Saturate, Integer and Accumulate into Words
evmhossianw	Vector Multiply Half Words, Odd, Signed, Saturate, Integer and Accumulate Negative into Words
evmhoumi	Vector Multiply Half Words, Odd, Unsigned, Modulo, Integer
evmhoumia	Vector Multiply Half Words, Odd, Unsigned, Modulo, Integer, to Accumulator
evmhoumiaaw	Vector Multiply Half Words, Odd, Unsigned, Modulo, Integer and Accumulate into Words
evmhoumianw	Vector Multiply Half Words, Odd, Unsigned, Modulo, Integer and Accumulate Negative into Words
evmhousiaaw	Vector Multiply Half Words, Odd, Unsigned, Saturate, Integer and Accumulate into Words
evmhousianw	Vector Multiply Half Words, Odd, Unsigned, Saturate, Integer and Accumulate Negative into Words
evmra	Move Register to Accumulator
evmwhsmf	Vector Multiply Word High Signed, Modulo, Fractional
evmwhsmfa	Vector Multiply Word High Signed, Modulo, Fractional, to Accumulator
evmwhsmi	Vector Multiply Word High Signed, Modulo, Integer
evmwhsmia	Vector Multiply Word High Signed, Modulo, Integer, to Accumulator
evmwhssf	Vector Multiply Word High Signed, Saturate, Fractional

Table 3-5. SPE Complex Integer Instructions (continued)

Instruction	Description
evmwhssfa	Vector Multiply Word High Signed, Saturate, Fractional, to Accumulator
evmwhumi	Vector Multiply Word High Unsigned, Modulo, Integer
evmwhumia	Vector Multiply Word High Unsigned, Modulo, Integer, to Accumulator
evmwlsmf	Vector Multiply Word Low Signed, Modulo, Fractional
evmwlsmfa	Vector Multiply Word Low Signed, Modulo, Fractional, to Accumulator
evmwlsmfaaw	Vector Multiply Word Low Signed, Modulo, Fractional and Accumulate in Words
evmwlsmfanw	Vector Multiply Word Low Signed, Modulo, Fractional and Accumulate Negative in Words
evmwlsmiaaw	Vector Multiply Word Low Signed, Modulo, Integer and Accumulate in Words
evmwlsmianw	Vector Multiply Word Low Signed, Modulo, Integer and Accumulate Negative in Word
evmwlssf	Vector Multiply Word Low Signed, Saturate, Fractional
evmwlssfa	Vector Multiply Word Low Signed, Saturate, Fractional, to Accumulator
evmwlssfaaw	Vector Multiply Word Low Signed, Saturate, Fractional and Accumulate in Words
evmwlssfanw	Vector Multiply Word Low Signed, Saturate, Fractional and Accumulate Negative in Words
evmwlssiaaw	Vector Multiply Word Low Signed, Saturate, Integer and Accumulate in Words
evmwlssianw	Vector Multiply Word Low Signed, Saturate, Integer and Accumulate Negative in Words
evmwlumi	Vector Multiply Word Low Unsigned, Modulo, Integer
evmwlumia	Vector Multiply Word Low Unsigned, Modulo, Integer, to Accumulator
evmwlumiaaw	Vector Multiply Word Low Unsigned, Modulo, Integer and Accumulate in Words
evmwlumianw	Vector Multiply Word Low Unsigned, Modulo, Integer and Accumulate Negative in Words
evmwlusiaaw	Vector Multiply Word Low Unsigned, Saturate, Integer and Accumulate in Words
evmwlusianw	Vector Multiply Word Low Unsigned, Saturate, Integer and Accumulate Negative in Words
evmwsmf	Vector Multiply Word Signed, Modulo, Fractional
evmwsmfa	Vector Multiply Word Signed, Modulo, Fractional, to Accumulator
evmwsmfaa	Vector Multiply Word Signed, Modulo, Fractional and Accumulate
evmwsmfan	Vector Multiply Word Signed, Modulo, Fractional and Accumulate Negative
evmwsmi	Vector Multiply Word Signed, Modulo, Integer
evmwsmia	Vector Multiply Word Signed, Modulo, Integer, to Accumulator
evmwsmiaa	Vector Multiply Word Signed, Modulo, Integer and Accumulate
evmwsmian	Vector Multiply Word Signed, Modulo, Integer and Accumulate Negative
evmwssf	Vector Multiply Word Signed, Saturate, Fractional
evmwssfa	Vector Multiply Word Signed, Saturate, Fractional, to Accumulator
evmwssfaa	Vector Multiply Word Signed, Saturate, Fractional and Accumulate
evmwssfan	Vector Multiply Word Signed, Saturate, Fractional and Accumulate Negative

Table 3-5. SPE Complex Integer Instructions (continued)

Instruction	Description
evmwumi	Vector Multiply Word Unsigned, Modulo, Integer
evmwumia	Vector Multiply Word Unsigned, Modulo, Integer, to Accumulator
evmwumiaa	Vector Multiply Word Unsigned, Modulo, Integer and Accumulate
evmwumian	Vector Multiply Word Unsigned, Modulo, Integer and Accumulate Negative
evsubfsmiaaw	Vector Subtract Signed, Modulo, Integer to Accumulator Word
evsubfssiaaw	Vector Subtract Signed, Saturate, Integer to Accumulator Word
evsubfumiaaw	Vector Subtract Unsigned, Modulo, Integer to Accumulator Word
evsubfusiaaw	Vector Subtract Unsigned, Saturate, Integer to Accumulator Word

3.9.2 SPE APU Scalar and Vector Floating Point Instructions

Support for a single precision floating point format is implemented in the SPE APU. The single precision format consists of a sign bit, an 8-bit exponent, and a 23-bit fraction. The floating point instructions can operate on both of the 32-bit fields in the 64-bit GPRs e.g. two pairs of single precision floating point numbers can be multiplied simultaneously. The supported floating point operations include: add, subtract, compare/test, multiply, and divide.

The SPE unit in the e200 implements several instructions to facilitate converting between floating point and fixed point formats. Various fixed point formats are supported including signed and unsigned, integer and fractional formats. Table 3-6 briefly describes the SPE Scalar Floating Point and Conversion Instructions.

Table 3-6. SPE Scalar Floating Point and Conversion Instructions

Instruction	Description
efsabs	Floating-Point Absolute Value
efsadd	Floating-Point Add
efscfsf	Convert Floating-Point from Signed Fraction
efscfsi	Convert Floating-Point from Signed Integer
efscfuf	Convert Floating-Point from Unsigned Fraction
efscfui	Convert Floating-Point from Unsigned Integer
efscmpeq	Floating-Point Compare Equal
efscmpgt	Floating-Point Compare Greater Than
efscmplt	Floating-Point Compare Less Than
efsctsf	Convert Floating-Point to Signed Fraction
efsctsi	Convert Floating-Point to Signed Integer
efsctsiz	Convert Floating-Point to Signed Integer with Round toward Zero
efsctuf	Convert Floating-Point to Unsigned Fraction

Table 3-6. SPE Scalar Floating Point and Conversion Instructions (continued)

Instruction	Description
efsctui	Convert Floating-Point to Unsigned Integer
efsctuiz	Convert Floating-Point to Unsigned Integer with Round toward Zero
efsdiv	Floating-Point Divide
efsmul	Floating-Point Multiply
efsnabs	Floating-Point Negative Absolute Value
efsneg	Floating-Point Negate
efssub	Floating-Point Subtract
efststeq	Floating-Point Test Equal
efststgt	Floating-Point Test Greater Than
efststlt	Floating-Point Test Less Than
efsabs	Floating-Point Absolute Value
efsadd	Floating-Point Add
efscfsf	Convert Floating-Point from Signed Fraction
efscfsi	Convert Floating-Point from Signed Integer
efscfuf	Convert Floating-Point from Unsigned Fraction
efscfui	Convert Floating-Point from Unsigned Integer
efscmpeq	Floating-Point Compare Equal

Table 3-7 briefly describes the SPE Vector Floating Point and Conversion Instructions.

Table 3-7. SPE Vector Floating Point and Conversion Instructions

Instruction	Description
evfsabs	Vector Floating-Point Absolute Value
evfsadd	Vector Floating-Point Add
evfscfsf	Vector Convert Floating-Point from Signed Fraction
evfscfsi	Vector Convert Floating-Point from Signed Integer
evfscfuf	Vector Convert Floating-Point from Unsigned Fraction
evfscfui	Vector Convert Floating-Point from Unsigned Integer
evfscmpeq	Vector Floating-Point Compare Equal
evfscmpgt	Vector Floating-Point Compare Greater Than
evfscmplt	Vector Floating-Point Compare Less Than
evfsctsf	Vector Convert Floating-Point to Signed Fraction
evfsctsi	Vector Convert Floating-Point to Signed Integer
evfsctsiz	Vector Convert Floating-Point to Signed Integer with Round toward Zero

Table 3-7. SPE Vector Floating Point and Conversion Instructions (continued)

Instruction	Description
evfsctuf	Vector Convert Floating-Point to Unsigned Fraction
evfsctui	Vector Convert Floating-Point to Unsigned Integer
evfsctuiz	Vector Convert Floating-Point to Unsigned Integer with Round toward Zero
evfsdiv	Vector Floating-Point Divide
evfsmul	Vector Floating-Point Multiply
evfsnabs	Vector Floating-Point Negative Absolute Value
evfsneg	Vector Floating-Point Negate
evfssub	Vector Floating-Point Subtract
evfststeq	Vector Floating-Point Test Equal
evfststgt	Vector Floating-Point Test Greater Than
evfststlt	Vector Floating-Point Test Less Than

3.9.3 SPE APU Load and Store Instructions

To effectively operate on the 64-bit register file, the SPE APU supports load and store instructions that handle up to 64 bits at the time. These 64 bits can be interpreted as two words or four half-words depending on the instruction as shown in Figure 3-7.

Every Vector Load/Store instruction has an indexed and a non-indexed version. The mnemonic for the indexed version is appended with an ’x’ to indicate an indexed instruction. For example the indexed version of the *evldd* instruction is *evlddx*. Table 3-8 briefly describes the SPE Load and Store Instructions.

Table 3-8. SPE Load and Store Instructions

Instruction	Description
evldd	Vector Load Double into Double
evldh	Vector Load Double into Halfwords
evldw	Vector Load Double into Words
evlhhesplat	Vector Load Halfword into Halfword Even and Splat
evlhhossplat	Vector Load Halfword into Halfword Odd Signed and Splat
evlhhousplat	Vector Load Halfword into Halfword Odd Unsigned and Splat
evlwhe	Vector Load Word into Halfwords Even
evlwhos	Vector Load Word into Halfwords Odd Signed (with sign extension)
evlwhou	Vector Load Word into Halfwords Odd Unsigned (zero-extended)
evlwhsplat	Vector Load Word into Halfwords and Splat
evlwwsplat	Vector Load Word into Word and Splat
evstdd	Vector Store Double of Double

Table 3-8. SPE Load and Store Instructions (continued)

Instruction	Description
evstdh	Vector Store Double of Four Halfwords
evstdw	Vector Store Double of Two Words
evstwhe	Vector Store Word of Two Halfwords from Even
evstwho	Vector Store Word of Two Halfwords from Odd
evstwwe	Vector Store Word of Word from Even
evstwwo	Vector Store Word of Word from Odd

3.10 Book E Instruction Extensions—VLE

The variable length encoding (VLE) provides an extension to 32-bit PowerPC Book E. There are additional operations defined using an alternate instruction encoding to enable reduced code footprint. This alternate encoding set is selected on an instruction page basis. A single page attribute bit selects between standard PowerPC Book E instruction encodings and VLE instructions for that page of memory. This page attribute is an extension to the PowerPC Book E page attributes. Pages can be freely intermixed, allowing for a mixture of code using both types of encodings.

Instruction encodings in pages marked as using the VLE extension are either 16 or 32 bits, and are aligned on 16-bit boundaries. Therefore, all instruction pages marked as VLE are required to use big-endian byte ordering.

This section describes the various extensions to Book E instructions to support the VLE extension.

rfci, **rfdi**, **rfi—**no longer mask bit 62 of CSRR0, DSRR0, or SRR0 respectively. The destination address is [D,C]SRR0[32:62] || 0b0.

bclr, **bclrl**, **bcctr**, **bcctrl**—no longer mask bit 62 of the LR or CTR respectively. The destination address is [LR,CTR][32:62] || 0b0.

Chapter 4
Reset

4.1 Introduction

The following reset sources are supported in this device:

- Power-on reset
- External reset (324 package only)
- Loss-of-lock reset
- Loss-of-clock reset
- Watchdog timer/debug reset
- JTAG reset
- Checkstop reset
- Software system reset
- Software external reset (324 package only)

The reset status register (SIU_RSR) gives the source of the last reset and indicates whether a glitch occurred on the $\overline{\text{RESET}}$ pin. The SIU_RSR is updated for all reset sources.

All reset sources are processed by the reset controller, which is located in the SIU module. The reset controller monitors the reset input sources and when a reset event is detected, resets the internal logic and controls the assertion of the $\overline{\text{RSTOUT}}$ pin. All reset sources invoke the boot assist module (BAM) program, except for the software external reset.

You can assert the $\overline{\text{RSTOUT}}$ signal by setting the SER bit in the SIU_SRCR to 1. The PLL configuration determines the number of system clocks[1] the $\overline{\text{RSTOUT}}$ signal is asserted. This does not reset the MCU. All other reset sources initiate an internal reset of the MCU.

For more information, see Section 4.2.2, “Reset Output (RSTOUT)”.

For all reset sources, use the BOOTCFG[0:1] signals to determine the boot mode and the PLLCFG[0:1] signals to determine the FMPLL configuration. If the $\overline{\text{RSTCFG}}$ pin is asserted during reset, the values on the BOOTCFG[0:1] pins are latched in the SIU_RSR four clock cycles before the $\overline{\text{RSTOUT}}$ pin deasserts, determining the boot mode. The values on the PLLCFG[0:1] pins are latched when the $\overline{\text{RSTOUT}}$ pin deasserts, determining the configuration of the FMPLL. If the $\overline{\text{RSTCFG}}$ pin deasserts during reset, the FMPLL defaults to normal operation (PLL enabled) with a crystal reference, and the boot mode (latched in the SIU_RSR) defaults to internal boot from flash.

1. Unless noted otherwise, the use of ‘clock’ or ‘clocks’ in this section is a reference to the system clock.

208 Package: BOOTCFG[0] and $\overline{\text{RSTCFG}}$ signals are not available due to pin limitations and are internally asserted (driven to 0). Therefore, the BOOTCFG[1] and PLLCFG[0:1] pins are always sampled.

The state of the BOOTCFG[0:1] pins specifies the location of the RCHW (internal flash or external memory), or that the MCU is configured to boot from a serial (eSCI) or FlexCAN port. See Chapter 2, “Signals” a complete description of the BOOTCFG[0:1].

The BAM program reads the values of the BOOTCFG[0:1] pins from the SIU_RSR, then reads the RCHW from the specified location and uses the RCHW value to determine and execute the specified boot procedure. See Section 4.4.3, “Reset Configuration and Configuration Pins,” for a complete description.

The reset status register (SIU_RSR) gives the source of the last reset and indicates whether a glitch occurred on the $\overline{\text{RESET}}$ pin. The SIU_RSR is updated for all reset sources.

The reset configuration half word (RCHW) provides several basic functions at reset:

- Locates the boot code
- Configures flash memory to either program or erase
- Enables or disables the watchdog timer
- Configures the MMU to boot: classic PowerPC Book E code or Freescale VLE code
- Sets the bus size (external boot only)

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

4.2 External Signal Description

4.2.1 Reset Input ($\overline{\text{RESET}}$)

Assert the $\overline{\text{RESET}}$ pin as an active low input by an external device during a power-on or external reset. The $\overline{\text{RESET}}$ pin must assert for at least 10 clock cycles to assert the internal reset signal. Assertion of the $\overline{\text{RESET}}$ pin while the device is in reset restarts the reset cycle. The $\overline{\text{RESET}}$ pin has a glitch detector to detect spikes more than two clocks in duration that fall below the switch point of the input buffer logic.

4.2.2 Reset Output ($\overline{\text{RSTOUT}}$)

The $\overline{\text{RSTOUT}}$ pin is an active low output that uses a push/pull configuration. The $\overline{\text{RSTOUT}}$ pin is driven to the low state by the MCU for all internal and external reset sources.

After the $\overline{\text{RESET}}$ input deasserts, if the PLL is configured for 1:1 (dual controller) mode or bypass mode, the $\overline{\text{RSTOUT}}$ signal asserts for 16000 clocks, plus four clocks for sampling the configuration pins. If the PLL is configured for another operating mode, the $\overline{\text{RSTOUT}}$ signal asserts for 2400 clocks, plus four clocks for sampling of the configuration pins. See Section 11.1.4, “FMPLL Modes of Operation” for details of PLL configuration.

Writing a one to the SER bit of the system reset control register (SIU_SRCR) asserts the $\overline{\text{RSTOUT}}$ pin.

NOTE

During a power on reset, RSTOUT is tri-stated.

4.2.3 Reset Configuration (RSTCFG)

The RSTCFG input is used to enable the BOOTCFG[0:1] and PLLCFG[0:1] pins during reset. If RSTCFG deasserts during reset, the BOOTCFG and PLLCFG pins are not sampled when RSTOUT deasserts. In that case, the default values for BOOTCFG and PLLCFG are used. If RSTCFG asserts during reset, the values on the BOOTCFG and PLLCFG pins are sampled and used to configure the boot and FMPLL modes.

208 Package: BOOTCFG[0] and RSTCFG signals are not available due to pin limitations and are internally asserted (driven to 0). Therefore, the BOOTCFG[1] and PLLCFG[0:1] pins are always sampled.

4.2.4 Weak Pull Configuration (WKPCFG)

WKPCFG determines whether specified eTPU and eMIOS pins are connected to a weak pullup or weak pulldown during and immediately after reset.

4.2.5 Boot Configuration (BOOTCFG[0:1])

BOOTCFG determines the function and state of the following pins after execution of the BAM reset: CS[0, 2:3], ADDR[8:31], DATA[0:15], RD_WR, BDIP, WE/BE[0:1], OE, TS, TA.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0). CS[1:3] are not available due to pin limitations in the 208 package.

4.3 Memory Map/Register Definition

Table 4-1 summarizes the reset controller registers. The base address of the system integration unit is 0xC3F9_0000.

Table 4-1. Reset Controller Memory Map

Address	Register Name	Register Description	Bits
Base (0xC3F9_0000) + 0x000C	SIU_RSR	Reset status register	32
Base (0xC3F9_0000) + 0x0010	SIU_SRCR	System reset control register	32

4.3.1 Register Descriptions

This section describes all the reset controller registers. It includes details about the fields in each register, the number of bits per field, the reset value of the register, and the function of the register.

4.3.1.1 Reset Status Register (SIU_RSR)

The reset status register (SIU_RSR) can be read at all times and contains a bit flag for each reset source, as well as the source of the last reset. A reset source bit flag set to 1 indicates that type of reset occurred. Simultaneous reset requests set more than one bit at the same time. Once set, the reset source bits in the SIU_RSR remain set until another reset occurs, except for a software external reset.

A software external reset sets the SERF bit, but does not clear any previously set bits in the SIU_RSR.

For additional information about the SIU_RSR, see Section 6.4.1.2, “Reset Status Register (SIU_RSR).”

The SIU_RSR also contains the values latched at the last reset on the WKPCFG and BOOTCFG[0:1] pins and a $\overline{\text{RESET}}$ input pin glitch flag. The reset glitch flag (RGF) is cleared to 0 by writing a 1 to the bit. A write of 0 has no effect on the bit value.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

Address: Base + 0x000C Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PORS	ERS	LLRS	LCRS	WDRS	CRS	0	0	0	0	0	0	0	0	SSRS	SERF
W																w1c
Reset[1]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WKP CFG	0	0	0	0	0	0	0	0	0	0	0	0	BOOTCFG		RGF
W																w1c
Reset	—[2]	0	0	0	0	0	0	0	0	0	0	0	0	—[3, 4]	—[3]	0

1 The RESET values for this register are defined for power-on reset only.

2 The RESET value of this bit or field is determined by the value latched on the associated pin or pins at the deassertion of the last reset.

3 The RESET value of this bit or field is determined by the value latched on the associated pin or pins at the deassertion of the last reset. On the 324 package, when $\overline{\text{RSTCFG}}$ is not asserted, a default value of 0b10 is loaded into BOOTCFG.

4 208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted ((driven to 0).

Figure 4-1. Reset Status Register (SIU_RSR)

Table 4-2. SIU_RSR Field Descriptions

Field	Description
0 PORS	Power-on reset status 0 No power-on reset has occurred 1 A power-on reset has occurred
1 ERS	External reset status 0 No external reset has occurred 1 An external reset has occurred The ERS bit is also set during a POR event
2 LLRS	Loss-of-lock reset status 0 No loss-of-lock reset has occurred 1 A loss-of-lock reset has occurred
3 LCRS	Loss-of-clock reset status 0 No loss-of-clock reset has occurred 1 A loss-of-clock reset has occurred due to a loss of the reference or failure of the FMPLL
4 WDRS	Watchdog timer/debug reset status 0 No watchdog timer or debug reset has occurred 1 A watchdog timer or debug reset has occurred
5 CRS	Checkstop reset status 0 No enabled checkstop reset has occurred 1 An enabled checkstop reset has occurred
6–13	Reserved
14 SSRS	Software system reset status 0 No software system reset has occurred. 1 A software system reset has occurred.
15 SERF	Software external reset flag. Write a 1 to clear this bit. 0 No software external reset has occurred 1 A software external reset has occurred
16 WKPCFG	Weak pull configuration pin status 0 WKPCFG pin latched during the last reset was logic 0 and weak pull down is the default setting 1 WKPCFG pin latched during the last reset was logic 1 and weak pull up is the default setting
17–28	Reserved
29–30 BOOTCFG	Reset configuration pin status. Holds the value of the BOOTCFG[0:1] pins that was latched 4 clocks before the last deassertion of the $\overline{\text{RSTOUT}}$ pin, if the $\overline{\text{RSTCFG}}$ pin was asserted. If the $\overline{\text{RSTCFG}}$ pin was deasserted at the last deassertion of $\overline{\text{RSTOUT}}$, the BOOTCFG field is set to the value 0b00. The BOOTCFG field is used by the BAM program to determine the location of the reset configuration half word. See Section 4.4.3.5, "Reset Configuration Half Word (RCHW)," for a translation of the reset configuration half word location from the BOOTCFG field value. 208Package: BOOTCFG[0] and $\overline{\text{RSTCFG}}$ are not available due to pin limitations and are internally asserted (driven to 0) in the 208 package. Therefore, BOOTCFG[1] and PLLCFG[0:1] are always sampled.
31 RGF	$\overline{\text{RESET}}$ glitch flag. Set by the MCU when the $\overline{\text{RESET}}$ pin is asserted for more than 2 clocks clock cycles, but less than the minimum $\overline{\text{RESET}}$ assertion time of 10 consecutive clocks to cause a reset. This bit is cleared by the reset controller for a valid assertion of the $\overline{\text{RESET}}$ pin or a power-on reset or a write of 1 to the bit. 0 No glitch was detected on the $\overline{\text{RESET}}$ pin 1 A glitch was detected on the $\overline{\text{RESET}}$ pin

4.3.1.2 System Reset Control Register (SIU_SRCR)

The system reset control register (SIU_SRCR) allows software to generate either a software system reset or software external reset. The software system reset causes an internal reset sequence, while the software external reset only causes the external $\overline{\text{RSTOUT}}$ pin to be asserted. When written to 1, the SER bit automatically clears after a predetermined number of clock cycles (see Section 4.2.2, "Reset Output ($\overline{\text{RSTOUT}}$)"). If the value of the SER bit is 1 and a 0 is written to the bit, the bit is cleared and the $\overline{\text{RSTOUT}}$ pin is deasserted regardless of whether the relevant number of clocks has expired.

The CRE bit in the SIU_SRCR allows software to enable a checkstop reset. If enabled, a checkstop reset occurs if the checkstop reset input to the reset controller asserts. The checkstop reset is enabled by default.

Address: Base + 0x0010 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SSR	SER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CRE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] The CRE bit is reset to 1 by POR. Other resets sources do not reset the bit value.

Figure 4-2. System Reset Control Register (SIU_SRCR)

Table 4-3. SIU_SRCR Field Descriptions

Field	Description
0 SSR	Software system reset. Writing a 1 to this bit causes an internal reset and assertion of the $\overline{\text{RSTOUT}}$ pin. The bit is automatically cleared by all reset sources except the software external reset. 0 Do not generate a software system reset 1 Generate a software system reset
1 SER	Software external reset. Writing a 1 to this bit causes an software external reset. The $\overline{\text{RSTOUT}}$ pin is asserted for a predetermined number of clock cycles (see Section 4.2.2, "Reset Output ($\overline{\text{RSTOUT}}$)"), but the MCU is not reset. The bit is automatically cleared when the software external reset completes. 0 Do not generate an software external reset 1 Generate an software external reset
2–15	Reserved
16 CRE	Checkstop reset enable Writing a 1 to this bit enables a checkstop reset when the e200z3 core enters a checkstop state. The CRE bit defaults to checkstop reset enabled. This bit is reset at POR. 0 No reset occurs when the e200z3 core enters a checkstop state 1 A reset occurs when the e200z3 core enters a checkstop state
17–31	Reserved

4.4 Functional Description

4.4.1 Reset Vector Locations

The reset vector contains a pointer to the instruction where code execution begins after BAM execution. The location of the reset vector is determined by boot mode, as illustrated in Table 4-4.

Table 4-4. Reset Vector Locations

Boot Mode	Reset Vector Location
External Boot	0x0000_0004 (0x0000_0000 must have a valid RCHW)
Internal Boot	Next word address after the first valid RCHW found. The BAM searches the lowest address of each of the six low address space blocks in flash memory for a valid RCHW. Hence, the possible reset vector locations are: 0x0000_0004 0x0000_4004 0x0001_0004 0x0001_C004 0x0002_0004 0x0003_0004
Serial Boot	Specified over serial download

4.4.2 Reset Sources

4.4.2.1 FMPLL Lock

A loss-of-lock of the FMPLL can cause a reset (provided the reset source is enabled by the FMPLL_SYNCR[LOLRE] bit). Regardless of the reset source, $\overline{\text{RESET}}$ remains asserted until the FMPLL locks.

4.4.2.2 Flash High Voltage

There is no flash access gating signal implemented in this device. This device remains in $\overline{\text{RESET}}$ to guarantee that high voltage circuits are reset and stabilized and that flash memory is accessible.

4.4.2.3 Reset Source Descriptions

For the following reset source descriptions see the reset flow diagrams in Figure 4-5 and Figure 4-6. Figure 4-5 shows the reset flow for assertion of the $\overline{\text{RESET}}$ pin. Figure 4-6 shows the internal processing of reset for all reset sources.

4.4.2.3.1 Power-on Reset

The power-on reset (POR) circuit is designed to detect a POR event and ensure that the $\overline{\text{RESET}}$ signal is correctly sensed. Do not use the POR to detect falling power supply voltages. Ensure that the external power supply is monitored. The output signals from the power-on reset circuits are active low signals. All power-on reset output signals are combined into one POR signal at the V_{DD} level and input to the reset controller.

NOTE

Even though asserting the power-on reset (POR) signal causes a reset, you must also assert the RESET pin at the same time to guarantee the MCU operates correctly.

The PLLCFG[0:1] and RSTCFG pins determine the configuration of the FMPLL.

- If RSTCFG is asserted when RSTOUT deasserts, the PLLCFG[0:1] pins set the operating mode of the FMPLL.
- If RSTCFG is asserted anytime that RSTOUT is asserted, the FMPLL switches to the mode specified by the PLLCFG[0:1] pins.

The values on the RSTCFG and the PLLCFG[0:1] pins must be kept constant once RSTCFG asserts to avoid transient mode changes in the FMPLL. If the reset configuration RSTCFG pin is in the deasserted state when RSTOUT deasserts, the FMPLL defaults to enabled with a crystal reference. See Chapter 11, "Frequency Modulated Phase Locked Loop and System Clocks (FMPLL)," for more details on the operation of the FMPLL and the PLLCFG[0:1] pins.

The signal on the WKPCFG pin determines whether weak pullup or pulldown devices are enabled after reset on the eTPU and eMIOS pins. The WKPCFG pin is applied when the internal reset signal asserts, as indicated by the assertion of RSTOUT. See Figure 4-4 and Chapter 2, "Signals," for more information on WKPCFG and RSTOUT.

Once the RESET input pin deasserts, the reset controller checks if the FMPLL is locked. The internal reset signal and RSTOUT remain asserted until the FMPLL is locked. After the FMPLL is locked, the reset controller waits a predetermined number of clock cycles (See Section 4.2.2, "Reset Output (RSTOUT)") before deasserting the RSTOUT pin. The WKPCFG and BOOTCFG[0:1] pins (the BOOTCFG[0:1] pins are sampled only if RSTCFG asserts) are sampled four clock cycles before RSTOUT deasserts, and the reset status register (SIU_RSR) fields are updated. The PORS and ERS bits are set, and all other reset status bits are cleared.

208 Package: BOOTCFG[0] and RSTCFG are not available due to pin limitations, and are internally asserted (driven to 0). Therefore, BOOTCFG[1] and PLLCFG[0:1] are always sampled.

4.4.2.3.2 External Reset

The external reset feature is available on this device in the 324 package only, which has a 16-bit external bus interface. The 208 package does not have EBI pins, therefore the external reset feature is not supported.

When the reset controller detects assertion of the RESET pin, the internal reset signal and RSTOUT are asserted. Starting at the assertion of the internal reset signal (as indicated by assertion of RSTOUT), the value on the WKPCFG pin is applied; at the same time the PLLCFG[0:1] values are applied if RSTCFG is asserted. Once the RESET pin is deasserted and the FMPLL loss-of-lock request signal is deasserted, the reset controller waits the predetermined number of clock cycles (see Section 4.2.2, "Reset Output (RSTOUT)").

Once the clock count completes, the WKPCFG and BOOTCFG[0:1] pins are sampled (BOOTCFG[0:1] are sampled only if RSTCFG is asserted). The reset controller then waits four clock cycles before the deasserting RSTOUT, and updates the fields in the SIU_RSR. In addition, the ERS bit is set, and all other reset status bits in the SIU_RSR are cleared.

208 Package: There are no external bus interface (EBI) pins on this package.

4.4.2.3.3 Loss-of-Lock Reset

A loss-of-lock reset occurs when the FMPLL loses lock and the loss-of-lock reset enable (LOLRE) bit in the FMPLL synthesizer control register (FMPLL_SYNCR) is set. Beginning when the internal reset asserts, as indicated by the device reset signal (RSTOUT) asserting, the weak pull value is applied on the WKPCFG pin. At the same time, the PLLCFG[0:1] values are applied if RSTCFG is asserted.

When the FMPLL locks, the reset controller waits until the predetermined clock count finishes and then the WKPCFG and BOOTCFG[0:1] pins are sampled (BOOTCFG[0:1] pins are only sampled if RSTCFG is asserted). The reset controller then waits four clock cycles before deasserting RSTOUT, and updating the Reset Status Register (SIU_RSR) fields. In addition, the LLRS bit is set, and all other reset status bits in the SIU_RSR are cleared.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

For more information, see:

- Section 4.2.2, “Reset Output (RSTOUT)”
- Chapter 11, “Frequency Modulated Phase Locked Loop and System Clocks (FMPLL)”

4.4.2.3.4 Loss-of-Clock Reset

A loss-of-clock reset occurs when the FMPLL detects a failure in either the reference signal or FMPLL output, and the loss-of-clock reset enable (LOCRE) bit in the FMPLL_SYNCR is set. The internal reset signal is asserted (as indicated by assertion of RSTOUT). Starting at the assertion of the internal reset signal (as indicated by assertion of RSTOUT), the value on the WKPCFG pin is applied; at the same time the PLLCFG[0:1] values are applied if RSTCFG is asserted. Once the FMPLL has a clock and is locked, the reset controller waits the predetermined clock cycles (See Section 4.2.2, “Reset Output (RSTOUT)”) before deasserting RSTOUT. When the clock count finishes the WKPCFG and BOOTCFG[0:1] pins are sampled (BOOTCFG[0:1] pins are only sampled if RSTCFG is asserted). The reset controller then waits 4 clock cycles before the deasserting RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, the LCRS bit is set, and all other reset status bits in the SIU_RSR are cleared. See Section 11.4.2.6, “Loss-of-Clock Detection,” for more information on loss-of-clock.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

4.4.2.3.5 Watchdog Timer/Debug Reset

The WDRS bit in the reset status register (SIU_RSR) is set when the watchdog timer or a debug request reset occurs.

A watchdog timer reset occurs and the WDRS bit is set when all the following conditions occur:

- e200z3 core watchdog timer is enabled with the enable next watchdog timer (EWT)
- Watchdog timer interrupt status (WIS) bits are set in the timer status register (TSR)
- Watchdog reset control (WRC) field in the timer control register (TCR) is configured to reset
- Time-out occurs

The debug tool can issue a debug reset command by writing 2'b10 to the RST bit {DBCR0[2:3]} register in the e200z3 core, which sets the WDRS bit in the reset status register of the systems integration unit (SIU_RSR).

To determine if WDRS was set by a watchdog timer or debug reset, check the WRS field in the e200z3 core TSR.

The effect of a watchdog timer or debug reset request is the same on the reset controller.

The debug tool can also reset the device using one of the following methods:

- Debug tool asserts the $\overline{\text{RESET}}$ signal on the RESET_b pin
- Debug tool sets the software system reset (SSR) bit in the system reset control register (SIU_SRCR)

The debug tool writes a one to the software external reset (SER) bit in the system reset control register (SIU_SRCR) to generate an external software reset.

The device comes out of reset using the following sequence:

1. Starting when the internal reset signal asserts, as indicated by $\overline{\text{RSTOUT}}$ asserting, the value on the WKPCFG pin is applied. At the same time, the PLLCFG[0:1] values are applied only if $\overline{\text{RSTCFG}}$ is asserted.
2. After the FMPLL is locked, the reset controller waits the predetermined number of clock cycles before negating $\overline{\text{RSTOUT}}$. When the clock count finishes, WKPCFG and BOOTCFG[0:1] are sampled. BOOTCFG[0:1] is only sampled if $\overline{\text{RSTCFG}}$ asserts.
3. The reset controller then waits four clock cycles before the deasserting $\overline{\text{RSTOUT}}$, and then updates the SIU_RSR. The WTRS bit is set and all other reset status bits in the SIU_RSR are cleared.

See the e200z3 Core Guide for more information on the watchdog timer and debug operation.

See Section 4.2.2, "Reset Output (RSTOUT)."

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

4.4.2.3.6 Checkstop Reset

When the e200z3 core enters a checkstop state, and the checkstop reset is enabled (the CRE bit in the system reset control register (SIU_SRCR) is set), a checkstop reset occurs. Starting at the assertion of the internal reset signal (as indicated by assertion of RSTOUT), the value on the WKPCFG pin is applied; at the same time the PLLCFG[0:1] values are applied if RSTCFG is asserted. Once the FMPLL is locked, the reset controller waits a predetermined number of clock cycles (see Section 4.2.2, “Reset Output (RSTOUT)”) before deasserting RSTOUT.When the clock count finishes the WKPCFG and BOOTCFG[0:1] pins are sampled (the BOOTCFG[0:1] pins are only sampled if RSTCFG is asserted). The reset controller then waits four clock cycles before the deasserting RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, the CRS bit is set, and all other reset status bits in the SIU_RSR are cleared. See e200z3 Core Guide for more information.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

4.4.2.3.7 JTAG Reset

A system reset occurs when JTAG is enabled and either the EXTEST, CLAMP, or HIGHZ instructions are executed by the JTAG controller. When the internal reset signal asserts (indicated by RSTOUT asserting), the value on the WKPCFG pin is applied, and at the same time, the PLLCFG[0:1] values are applied (as long as RSTCFG is asserted).

When the JTAG reset request deasserts and the FMPLL is locked, the reset controller waits a predetermined number of clock cycles before deasserting RSTOUT. When the clock count completes, the WKPCFG and BOOTCFG[0:1] pins are sampled (BOOTCFG[0:1] is sampled only if RSTCFG is asserted), and the fields in the SIU_RSR are updated. The reset source status bits in the SIU_RSR are unaffected.

See Section 4.2.2, “Reset Output (RSTOUT) and Chapter 23, “IEEE 1149.1 Test Access Port Controller (JTAGC),” for more information.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

4.4.2.3.8 Software System Reset

A software system reset is caused by a write to the SSR bit in the system reset control register (SIU_SRCR). A write of 1 to the SSR bit causes an internal reset of the MCU. The value on the WKPCFG pin is applied when the internal reset signal asserts (indicated by RSTOUT asserting), and at the same time the PLLCFG[0:1] values are applied (as long as RSTCFG is asserted).

Once the FMPLL locks, the reset controller waits a predetermined number of clock cycles before deasserting RSTOUT. When the clock count completes, the WKPCFG and BOOTCFG[0:1] pins are sampled (BOOTCFG[0:1] is sampled only if RSTCFG is asserted). The reset controller then waits four clock cycles before deasserting RSTOUT, and updates the fields in the SIU_RSR. In addition, the SSRS bit is set, and all other reset status bits in the SIU_RSR are cleared.

See Section 4.2.2, “Reset Output (RSTOUT) for more information.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

4.4.2.3.9 Software External Reset

A write of 1 to the SER bit in the SIU_SRCR causes the external $\overline{\text{RSTOUT}}$ pin to be asserted for a predetermined number of clocks (See Section 4.2.2, “Reset Output (RSTOUT)”). The SER bit automatically clears after the clock cycle expires. A software external reset does not cause a reset of the MCU, the BAM program is not executed, the PLLCFG[0:1], BOOTCFG[0:1], and WKPCFG pins are not sampled. The SERF bit in the SIU_RSR is set, but no other status bits are affected. The SERF bit in the SIU_RSR is not automatically cleared after the clock count expires, and remains set until cleared by software or another reset besides the software external reset occurs.

For a software external reset, the e200z3 core continues to execute instructions, timers that are enabled continue to operate, and interrupt requests continue to be processed. It is the responsibility of the application to ensure devices connected to $\overline{\text{RSTOUT}}$ are not accessed during a software external reset, and to determine how to manage MCU resources.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

4.4.3 Reset Configuration and Configuration Pins

The microcontroller and the BAM perform a reset configuration that allows certain functions of the MCU to be controlled and configured at reset. This reset configuration is defined by:

- Configuration pins
- A reset configuration half word (RCHW), if present
- Serial port, if a serial boot is used

The following sections describe these configuration pins and the RCHW.

4.4.3.1 $\overline{\text{RSTCFG}}$ Pin

Table 4-5 shows the $\overline{\text{RSTCFG}}$ pin settings for configuring the MCU to use a default or a custom configuration. See Chapter 2, “Signals” for more information about the $\overline{\text{RSTCFG}}$ pin.

Table 4-5. $\overline{\text{RSTCFG}}$ Settings

$\overline{\text{RSTCFG}}$	Description
1	Use default configuration of: – booting from internal flash – clock source is a crystal on FMPLL
0	Get configuration information from: – BOOTCFG[0:1] – PLLCFG[0:1]

208 Package: BOOTCFG[0] and $\overline{\text{RSTCFG}}$ are not available due to pin limitations, and are internally asserted (driven to 0). Therefore, BOOTCFG[1] and PLLCFG[0:1] are always sampled.

4.4.3.2 WKPCFG Pin (Reset Weak Pullup/Pulldown Configuration)

As shown in Table 4-6, the signal on the WKPCFG pin determines whether specific eTPU and eMIOS pins are connected to weak pullup or weak pulldown devices during and after reset (see Chapter 2, "Signals," for the eTPU and eMIOS pins that are affected by WKPCFG). For all reset sources except the software external reset, the WKPCFG pin is applied starting at the assertion of the internal reset signal (as indicated by the assertion of $\overline{\text{RSTOUT}}$). If the WKPCFG signal is logic high at this time, pullup devices are enabled on the eTPU and eMIOS pins. If the WKPCFG signal is logic low at the assertion of the internal reset signal, pulldown devices are enabled on those pins. The value on WKPCFG must be held constant during reset to avoid oscillations on the eTPU and eMIOS pins caused by switching pullup and pulldown states. The final value of WKPCFG is latched four clock cycles before the deassertion of $\overline{\text{RSTOUT}}$. After reset, software can modify the weak pullup and pulldown selections for all I/O pins through the PCRs in the SIU.

Table 4-6. WKPCFG Settings

WKPCFG	Description
0	Weak pull down applied to eTPU and eMIOS pins at reset
1	Weak pull up applied to eTPU and eMIOS pins at reset

Also see Chapter 2, "Signals" for information about the WKPCFG pin.

4.4.3.3 BOOTCFG[0:1] Pins (MCU Configuration)

In addition to specifying the RCHW location, the values latched on the BOOTCFG[0:1] pins at reset are used to initialize the internal flash memory enabled/disabled state, and whether no arbitration or external arbitration of the external bus interface is selected. Additionally, the RCHW can determine either directly or indirectly how the MMU is configured, how the external bus is configured, CAN or eSCI module and pin configuration, Nexus enabling, and password selection.

Also see Chapter 2, "Signals" for information about the BOOTCFG pins.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

4.4.3.3.1 BOOTCFG[0:1] Configuration in the 208 Package

In the 208 BGA package, the BOOTCFG[0] pin is unavailable and BOOTCFG[1] has a constant value based on PLLCFG[0]. The device configuration is mapped based on Table 4-7.

Table 4-7. Boot Configuration in the 208 BGA

PLLCFG0	BOOTCFG1	Boot Identifier Field (RCHW)	Boot Mode	Configuration Word Source
0	0	Valid	Internal	The lowest address of one of the six low address spaces (LAS) in internal flash memory.
		Invalid	Serial	Not applicable
	1	—	Serial	Not applicable
1	0	Valid	Reserved	The lowest address (0x2000_0000) of an external memory device, enabled by chip select $\overline{CS}$[0] using 16-bit data bus.
		Invalid	Reserved	Not applicable
	1	Valid	Reserved	The lowest address (0x2000_0000) of an external memory device, enabled by chip select $\overline{CS}$[0] using 16-bit data bus.
		Invalid	Serial boot	Not applicable

4.4.3.4 PLLCFG[0:1] Pins

The role of PLLCFG pins in PLL configuration is explained in Section 11.1.4, “FMPLL Modes of Operation.” Also see Chapter 2, “Signals” for information about the PLLCFG pins.

Table 4-8. PLLCFG[0:1] and RSTCFG in Configuration

$\overline{RSTCFG}$	PLLCFG[0]	PLLCFG[1]	Clock Mode	MODE	PLLSEL	PLLREF
1	PLLCFG pins ignored		Crystal reference (default)	1	1	1
0	0	0	Bypass mode	0	0	0
0	0	1	External reference	1	1	0
0	1	0	Crystal reference	1	1	1
0	1	1	Dual controller (1:1) mode	1	0	0

4.4.3.5 Reset Configuration Half Word (RCHW)

4.4.3.5.1 Reset Configuration Half Word (RCHW) Definition

The RCHW is read from either external memory or internal flash memory. If a valid RCHW is not found, a CAN/SCI boot is initiated. The RCHW is a collection of control bits that specify a minimum MCU configuration after reset and define the boot mode for the BAM program. At reset the RCHW provides a means to locate the boot code, determines if flash memory is programmed or erased, enables or disables the watchdog timer, configures the MMU to boot as either classic PowerPC Book E code or as Freescale VLE code, and if booting externally, sets the bus size. See the register indicated by the RCHW bit descriptions for a detailed description of each control bit.

NOTE

Do not configure the RCHW to a 32-bit bus size because the 324 package has a 16-bit data bus. The 208 package does not have an external bus.

If booting from internal flash or external memory, you must ensure that the RCHW is configured for the correct value and memory address. The boot ID of the RCHW must be read as 0x005A. BOOT_BLOCK_ADDRESS is explained in Section 15.3.2.3.4, “Read the Reset Configuration Halfword.”

The fields of the RCHW are shown in Figure 4-3.

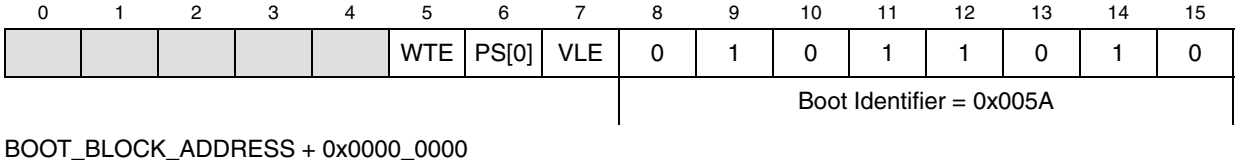


Figure 4-3. RCHW Fields

Table 4-9. Internal Boot RCHW Field Descriptions

Field	Description
0–4	Reserved: These bit values are ignored when the halfword is read. Write to 0 for future compatibility.
5 WTE	Watchdog timer enable. This is used to enable or disable the e200z3 watchdog timer through the BAM program. The configuration of the watchdog timer function is managed through the timer control register (TCR). 0 BAM does not write the e200z3 timebase registers (TBU and TBL) nor enable the e200z3 core watchdog timer. 1 BAM writes the e200z3 timebase registers (TBU and TBL) to 0x0000_0000_0000_0000 and enables the e200z3 core watchdog timer with a time-out period of 3 x 2^{17} system clock cycles. (Example: For 8 MHz crystal –> 12MHz system clock–> 32.7mS time-out. For 20 MHz crystal –> 30 MHz system clock –> 13.1mS time-out)
6 PS [0]	Port size. Defines the width of the data bus connected to the memory on $\overline{CS}$[0]. After system reset, $\overline{CS}$[0] is changed to a 16-bit port by the BAM which fetches the RCHW from either 16- or 32-bit external memories. Then the BAM reconfigures the EBI either as a 16-bit bus or a 32-bit bus, according to the settings of this bit. 0 Invalid value 1 16-bit $\overline{CS}$[0] port size **Note:** Used only in external boot mode. Do not set the port to 32-bits because the 324 package only has a 16-bit data bus.
7 VLE	VLE Code Indicator. This bit is used to configure the MMU for the boot block to execute as either Classic PowerPC Book E code or as Freescale VLE code. 0 = Boot code executes as Classic PowerPC Book E code 1 = Boot code executes as Freescale VLE code
8–15 BOOTID [0:7]	Boot identifier. This field indicates which block in flash memory contains the boot program and identifies whether the flash memory is programmed or invalid. A valid boot identifier is 0x005A (0b01011010). The BAM program checks the first half word of each flash memory block starting at block 0 until a valid boot identifier is found. If all blocks in the low address space of the internal flash are checked and no valid boot identifier is found, boot code is initiated from a FlexCAN or eSCI port. 324 Package Only: For an external boot, only block 0 is checked for a valid boot identifier, and if not found, boot code is initiated from a FlexCAN or eSCI port.

4.4.3.5.2 Invalid Reset Configuration Half Word (RCHW)

If the device is configured to boot from internal flash, a valid boot ID must be read at the lowest address of one of the six LAS blocks in internal flash memory. If the device is configured to boot from external

memory, a valid boot ID must be read at 0x00_0000 of $\overline{CS}$[0]. See Chapter 15, “Boot Assist Module (BAM)” for more information.

If a valid RCHW is not found, a serial boot is initiated which does not use a RCHW. The watchdog timer is enabled. For serial boot entered from a failed external boot, the port size remains configured as 16 bits wide. For serial boot entered from a failed internal boot, the external bus is never configured and remains in the reset state of GPIO inputs.

4.4.3.5.3 Reset Configuration Half Word (RCHW) Source

The reset configuration half word (RCHW) specifies a minimal MCU configuration after reset. The RCHW also contains bits that control the BAM program flow. See Section 15.3.2.2.1, “Finding the Reset Configuration Halfword” for information on the BAM using the RCHW. The RCHW is read and applied each time the BAM program executes, which is for every power-on, external, or internal reset event. The only exception to this is the software external reset. See Section 4.4.3.5, “Reset Configuration Half Word (RCHW),” for detailed descriptions of the bits in the RCHW. The RCHW is read from one of the following locations:

- The lowest address (0x0000_0000) of an external memory device, enabled by chip select $\overline{CS}$[0] using a 16-bit data bus
- The lowest address of one of the six low address space (LAS) blocks in the internal flash memory. (2 x 16 KB; 2 x 48 KB; 2 x 64 KB)

At the deassertion of the $\overline{RSTOUT}$ pin, the BOOTCFG field in the RSR has been updated. If BOOTCFG[0] is asserted, then the BAM program reads the RCHW from address 0x0000_0000 in the external memory connected to $\overline{CS}$[0] (the BAM first configures the MMU and $\overline{CS}$[0] such that address 0x0000_0000 is translated to 0x2000_0000 and then directed to $\overline{CS}$[0]). When BOOTCFG[0] is asserted, BOOTCFG[1] determines whether external arbitration must be enabled to fetch the RCHW.

If BOOTCFG[0] and BOOTCFG[1] are deasserted at the deassertion of the $\overline{RSTOUT}$ pin, then the BAM program attempts to read the RCHW from the first address of each of the six blocks in the low address space (LAS) of internal flash. Table 4-10 shows the LAS addresses.

Table 4-10. LAS Block Memory Addresses

Block	Address
0	0x0000_0000
1	0x0000_4000
2	0x0001_0000
3	0x0001_C000
4	0x0002_0000
5	0x0003_0000

If the RCHW stored in either internal or external flash is invalid (boot identifier field of RCHW is *not* 0x005A), or if BOOTCFG[0] is deasserted and BOOTCFG[1] is asserted at the deassertion of the $\overline{RSTOUT}$ pin, then RCHW is not applicable, and serial boot mode is performed. Table 4-11 summarizes the RCHW location options.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).

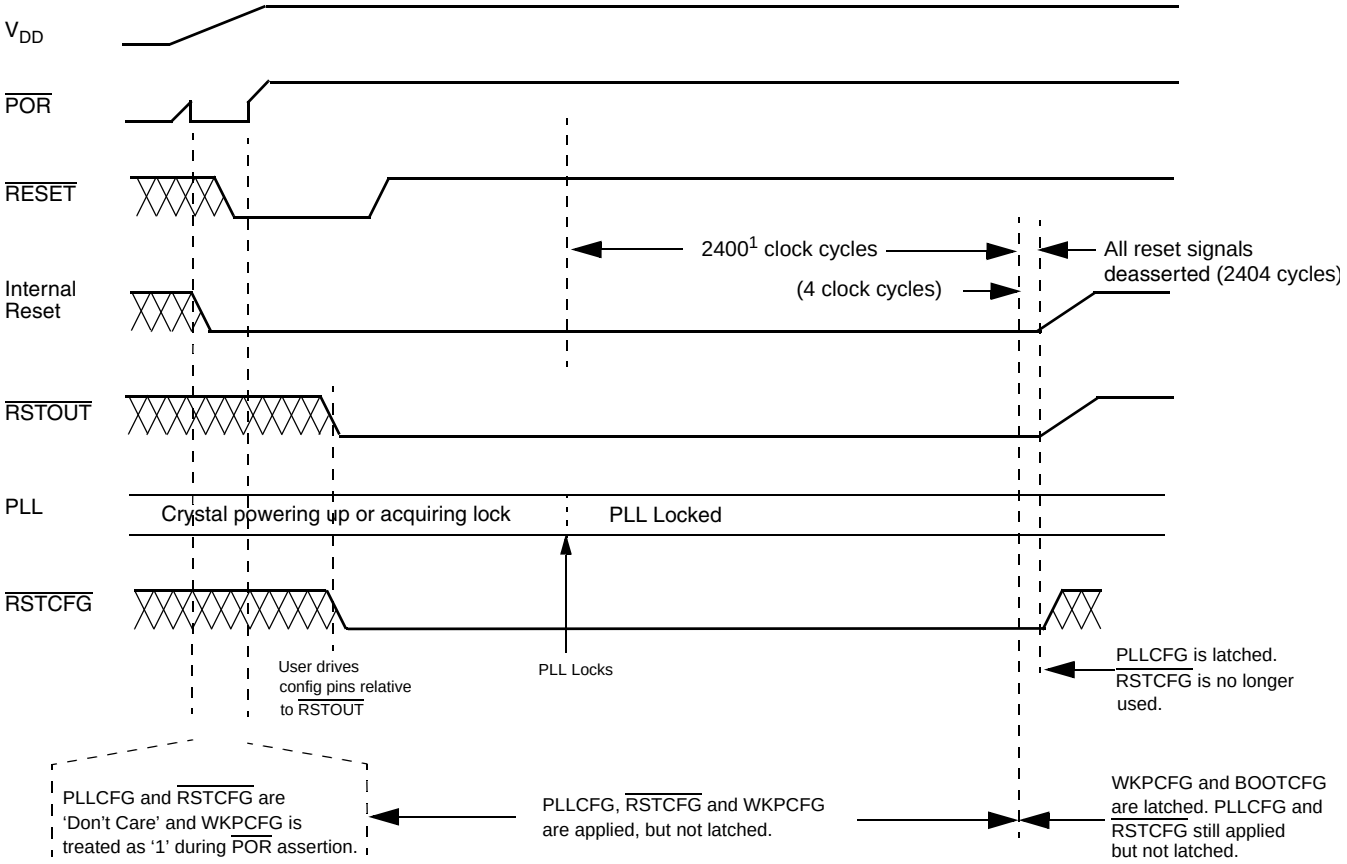
Table 4-11. Reset Configuration Half Word Sources

$\overline{\text{RSTCFG}}$	BOOTCFG[0]	BOOTCFG[1]	Boot Identifier Field (RCHW)	Boot Mode	Configuration Word Source
1	—	—	Valid	Internal	The lowest address of one of the six low address spaces (LAS) in internal flash memory.
			Invalid	Serial	Not applicable
0	0	0	Valid	Internal	The lowest address of one of the six low address spaces (LAS) in internal flash memory.
			Invalid	Serial	Not applicable
0	0	1	—	Serial	Not applicable
0	1	0	Valid	External Boot, No Arbitration	The lowest address (0x00_0000) of an external memory device, enabled by chip select $\overline{\text{CS}}$[0] using either 16- or 32-bit data bus
			Invalid	Serial	Not applicable
0	1	1	External Arbitration Not Supported		

4.4.4 Reset Configuration Timing

The timing diagram in Figure 4-4 shows the sampling of the BOOTCFG[0:1], WKPCFG, and PLLCFG[0:1] pins for a power-on reset. The timing diagram is also valid for internal/external resets assuming that V_{DD}, V_{DDSYN}, and V_{DDEH6} are within valid operating ranges. The values of the PLLCFG[0:1] pins are latched at the deassertion of the $\overline{\text{RSTOUT}}$ pin, if the $\overline{\text{RSTCFG}}$ pin is asserted at the deassertion of $\overline{\text{RSTOUT}}$. The value of the WKPCFG signal is applied at the assertion of the internal reset signal (as indicated by the assertion of $\overline{\text{RSTOUT}}$). The values of the WKPCFG and BOOTCFG[0:1] pins are latched four clock cycles before the deassertion of $\overline{\text{RSTOUT}}$ and stored in the reset status register (SIU_RSR). BOOTCFG[0:1] are latched only if $\overline{\text{RSTCFG}}$ is asserted. WKPCFG is not dependent on $\overline{\text{RSTCFG}}$.

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).



[1] This clock count is dependent on the configuration of the FMPLL (See Section 4.2.2, "RSTOUT"). If the FMPLL is configured for 1:1 (dual controller) operation or for bypass mode, this clock count is 16000.

Figure 4-4. Reset Configuration Timing

4.4.5 Reset Flow

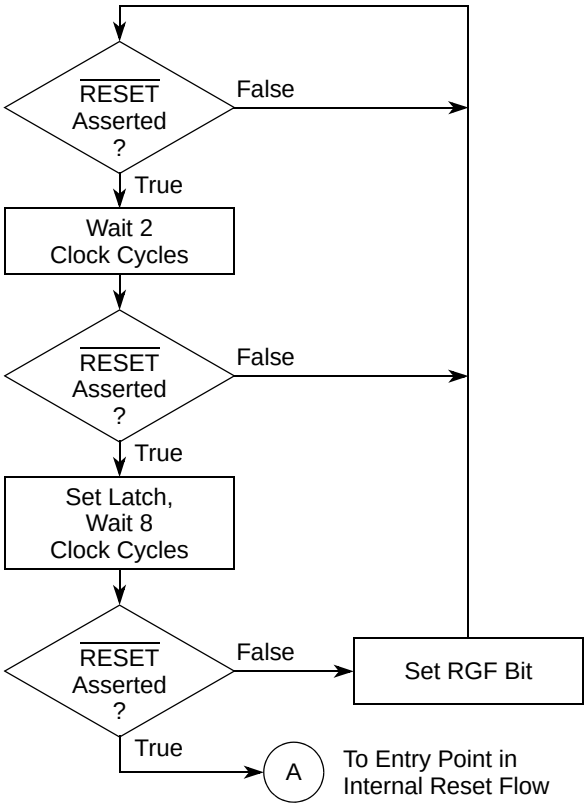
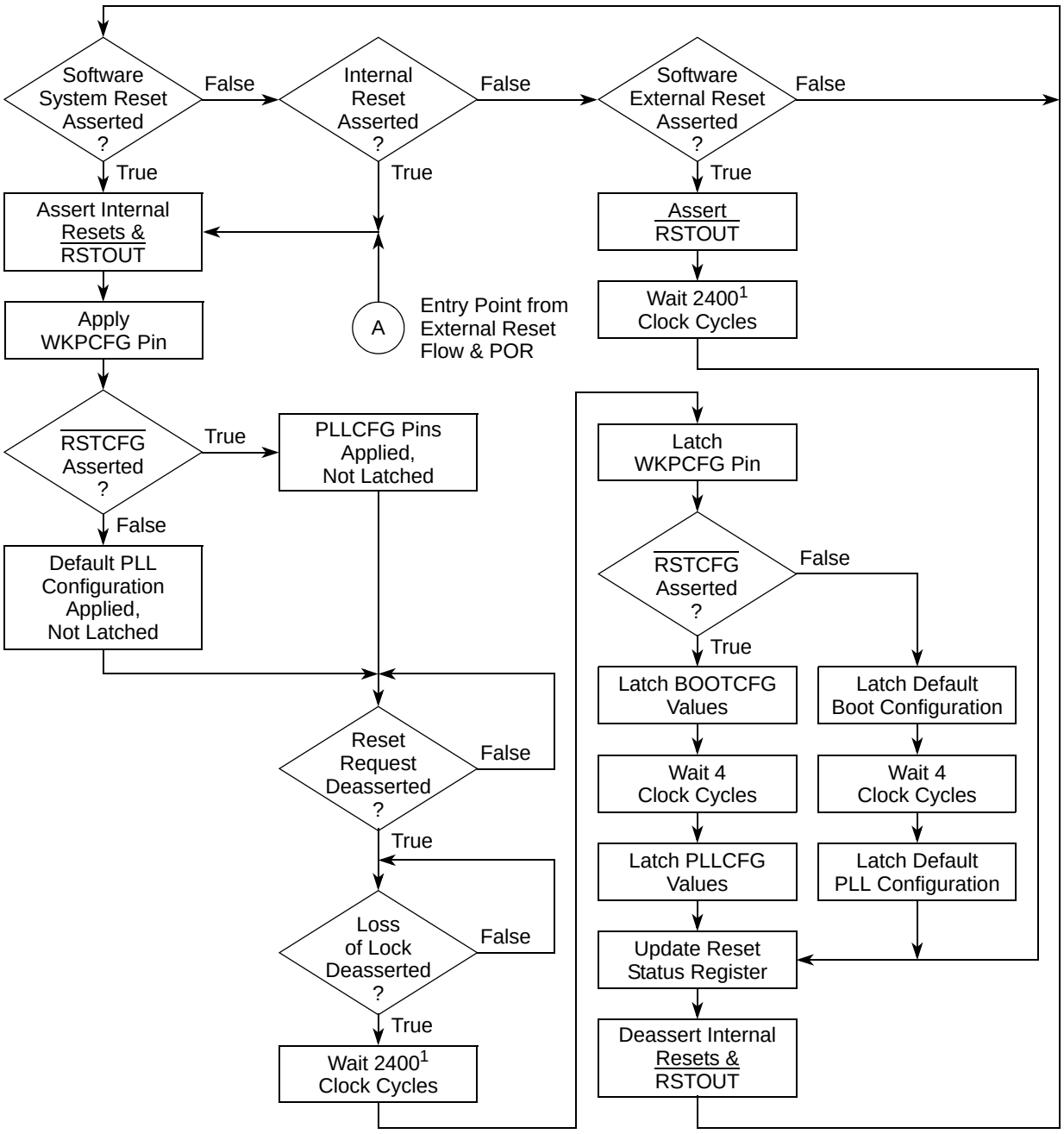


Figure 4-5. External Reset Flow Diagram



NOTES:
[1] The clock count is dependent on the configuration of the FMPLL (see Section 5.3.1.2, 'RSTOUT'). If the FMPLL is configured in 1:1 (dual controller) or bypass mode, this clock count is 16000.

Figure 4-6. Internal Reset Flow Diagram

Chapter 5
Peripheral Bridge

5.1 Introduction

5.1.1 Block Diagram

The PBRIDGE is the interface between the system bus and on-chip peripherals as shown in Figure 5-1.

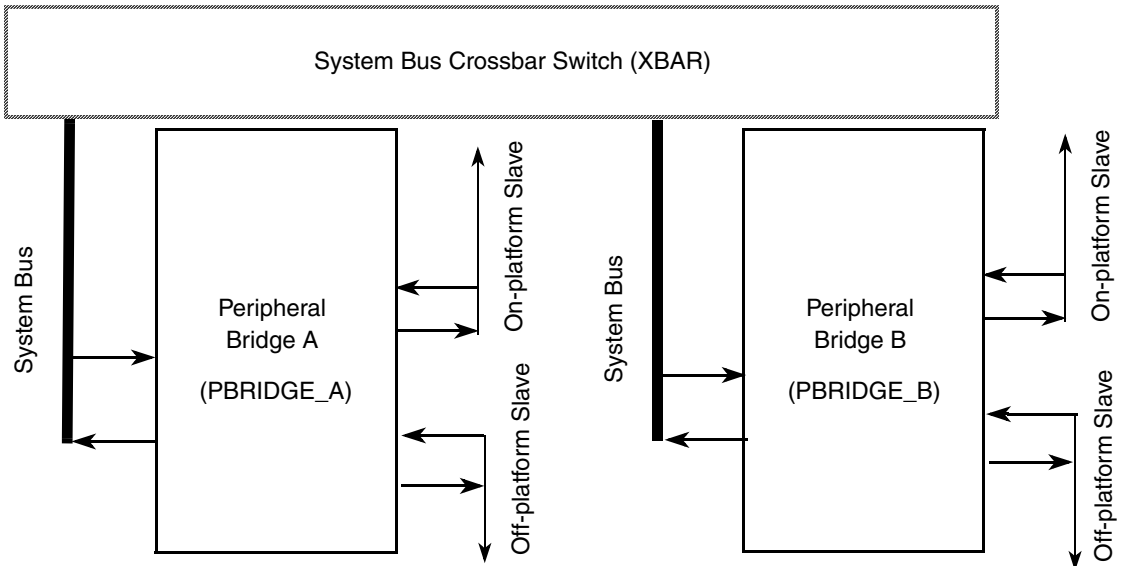


Figure 5-1. PBRIDGE Interface

5.1.2 Overview

There are two peripheral bridges, PBRIDGE_A and PBRIDGE_B, which act as interfaces between the system bus and lower bandwidth peripherals. In this manual, PBRIDGE refers to either of these bridges, as their functionality is identical. The only difference is the peripherals to which they connect. Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the slave bus interface.

5.1.2.1 Access Protections

The PBRIDGE provides programmable access protections for masters. It allows the privilege level of a master to be overridden, forcing it to user mode privilege, and allows masters to be designated as trusted or untrusted. More information on access protection can be found in Section 13.3.2.9, “Flash Bus Interface Unit Access Protection Register FLASH_BIUAPR.”

5.1.3 Features

The following list summarizes the key features of the PBRIDGE:

- Supports the slave interface signals, which is meant for slave peripherals only
- Supports 32-bit slave peripherals
  (byte, halfword, and word reads and writes are supported to each)
- Supports a pair of slave accesses for 64-bit instruction fetches
- Provides configurable per-master access protections

5.1.4 Modes of Operation

The PBRIDGE has only one operating mode.

5.2 External Signal Description

The PBRIDGE has no external signals.

5.3 Memory Map and Register Definition

The memory map for the program-visible PBRIDGE A and PBRIDGE B registers is shown in Table 5-1.

Table 5-1. PBRIDGE A and B Memory Map

Address	Register Name	Register Description	Bits
Base[1] + 0x0000	PBRIDGE_*x*_MPCR	Master privilege control register	32

[1] PBRIDGE_A base is 0xC3F0_0000. PBRIDGE_B base is 0xFFF0_0000.

5.3.1 Register Descriptions

All registers are 32-bit registers and can only be accessed in supervisor mode by trusted bus masters. Additionally, these registers must only be read from or written to by a 32-bit aligned access. PBRIDGE registers are mapped into the PBRIDGE_A and PBRIDGE_B address spaces. The protection and access fields of the MPCR are 4 bits in width.

5.3.1.1 Master Privilege Control Register (PBRIDGE_*x*_MPCR)

Each master privilege control register (PBRIDGE_*x*_MPCR) specifies 4-bit access fields defining the access privilege level associated with a bus master in the platform, as well as specifying whether the write accesses from this master are buffered. The registers provide one field per bus master.

Address: Base + 0x0000 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Access Field 0				Access Field 1				Access Field 2				Access Field 3			
R	MBW	MTR	MTW	MPL	MBW	MTR	MTW	MPL	MBW	MTR	MTW	MPL	MBW	MTR	MTW	MPL
W	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
Reset	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	Access Field 4				Access Field 5				Access Field 6				Access Field 7			
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-2. Master Privilege Control Registers (PBRIDGE_*x*_MPCR)

Table 5-2. PBRIDGE_*x*_MPCR Field Descriptions

Field	Description
0 MBW0	Master buffer writes. Determines whether the PBRIDGE is enabled to buffer writes from the CPU. Writes not able to be buffered by default. 0 Write accesses from the CPU cannot be buffered 1 Write accesses from the CPU can be buffered
1 MTR0	Master trusted for reads. Determines whether the CPU is trusted for read accesses. Trusted by default. 0 The CPU is not trusted for read accesses. 1 The CPU is trusted for read accesses.
2 MTW0	Master trusted for writes. Determines whether the master is trusted for write accesses. Trusted by default. 0 The CPU is not trusted for write accesses. 1 The CPU is trusted for write accesses.
3 MPL0	Master privilege level. Determines how the privilege level of the CPU is determined. Accesses not forced to user mode by default. 0 Accesses from the CPU are forced to user mode. 1 Accesses from the CPU are not forced to user mode.
4 MBW1	Master buffer writes. Determines whether the PBRIDGE is enabled to buffer writes from the Nexus. Writes not able to be buffered by default. 0 Write accesses from the Nexus cannot be buffered 1 Write accesses from the Nexus can be buffered
5 MTR1	Master trusted for reads. Determines whether the Nexus is trusted for read accesses. Trusted by default. 0 The Nexus is not trusted for read accesses. 1 The Nexus is trusted for read accesses.

Table 5-2. PBRIDGE_*x*_MPCR Field Descriptions (continued)

Field	Description
6 MTW1	Master trusted for writes. Determines whether the master is trusted for write accesses. Trusted by default. 0 The Nexus is not trusted for write accesses. 1 The Nexus is trusted for write accesses.
7 MPL1	Master privilege level. Determines how the privilege level of the Nexus is determined. Accesses not forced to user mode by default. 0 Accesses from the Nexus are forced to user mode. 1 Accesses from the Nexus are not forced to user mode.
8 MBW2	Master buffer writes. Determines whether the PBRIDGE is enabled to buffer writes from the eDMA. Writes not able to be buffered by default. 0 Write accesses from the eDMA cannot be buffered 1 Write accesses from the eDMA can be buffered
9 MTR2	Master trusted for reads. Determines whether the eDMA is trusted for read accesses. Trusted by default. 0 The eDMA is not trusted for read accesses. 1 The eDMA is trusted for read accesses.
10 MTW2	Master trusted for writes. Determines whether the master is trusted for write accesses. Trusted by default. 0 The eDMA is not trusted for write accesses. 1 The eDMA is trusted for write accesses.
11 MPL2	Master privilege level. Determines how the privilege level of the eDMA is determined. Accesses not forced to user mode by default. 0 Accesses from the eDMA are forced to user mode. 1 Accesses from the eDMA are not forced to user mode.
12 MBW3	Master buffer writes. Determines whether the PBRIDGE is enabled to buffer writes from the EBI. Writes not able to be buffered by default. 0 Write accesses from the EBI cannot be buffered 1 Write accesses from the EBI can be buffered
13 MTR3	Master trusted for reads. Determines whether the EBI is trusted for read accesses. Trusted by default. 0 The EBI is not trusted for read accesses. 1 The EBI is trusted for read accesses.
14 MTW3	Master trusted for writes. Determines whether the master is trusted for write accesses. Trusted by default. 0 The EBI is not trusted for write accesses. 1 The EBI is trusted for write accesses.
15 MPL3	Master privilege level. Determines how the privilege level of the EBI is determined. Accesses not forced to user mode by default. 0 Accesses from the EBI are forced to user mode. 1 Accesses from the EBI are not forced to user mode.
16–31	Reserved

5.4 Functional Description

The PBRIDGE serves as an interface between a system bus and the peripheral (slave) bus. It functions as a protocol translator. Support is provided for generating a pair of 32-bit peripheral instruction accesses (not data accesses) when targeted by a 64-bit system bus access. No other bus-sizing access support is provided.

Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the slave bus interface.

5.4.1 Access Support

Aligned 64-bit accesses, aligned word and halfword accesses, as well as byte accesses are supported for 32-bit peripherals. Peripheral registers must not be misaligned, although no explicit checking is performed by the PBRIDGE.

NOTE

Data accesses that cross a 32-bit boundary are not supported.

5.4.2 Peripheral Write Buffering

NOTE

MPC5534 only supports write buffering on a per-master basis, not on a per-peripheral basis.

The PBRIDGE provides programmable write buffering capability to allow certain write accesses to be buffered in the PBRIDGE for later completion, while terminating the system bus access early. This provides improved performance in systems where frequent writes to a slow peripheral are performed. Enable write buffering for masters only if:

- the slave bus does not generate termination errors
- it is safe to ignore termination errors

The PBRIDGE controller ignores the error signal on the termination of the buffered writes.

When write buffering is enabled, all accesses through the PBRIDGE occur in-order; no bypassing of buffered writes is supported.

Write buffering is controllable on a per-master basis.

5.4.2.1 Read Cycles

Two-clock read accesses are possible with the PBRIDGE when the requested access size is 32-bits or smaller, and is not misaligned across a 32-bit boundary. If the requested instruction access size is 64-bits, or it is misaligned across a 32-bit boundary (not supported), then a minimum of three clocks are required to complete the access. Misaligned read accesses are not supported. 64-bit data reads (not instruction) are not supported.

5.4.2.2 Write Cycles

Three clock write accesses are possible with the PBRIDGE when the requested access size is 32-bits or smaller. Misaligned writes that cross a 32-bit boundary are not supported. 64-bit data writes (not instruction) are not supported.

5.4.2.3 Buffered Write Cycles

Single clock write responses to the system bus are possible with the PBRIDGE when the requested write access is bufferable. If the requested access does not violate the permissions check, and if the master is enabled for buffering writes, the PBRIDGE internally buffers the write cycle. The write cycle is terminated early with zero system bus wait states. The access proceeds normally on the slave interface, but error responses are ignored.

All accesses are initiated and completed in order on the slave interface, regardless of buffering. If the buffer is full, the following write cycle waits until it can either be buffered (if bufferable) or can be initiated. If the buffer has valid entries, the following read cycle waits until the buffer is emptied and the read cycle completes.

5.4.3 General Operation

NOTE

This device supports write buffering on a per-master basis only—not on a per-peripheral basis.

Slave peripherals are modules that contain readable/writable control and status registers. The system bus master reads and writes these registers through the PBRIDGE. The PBRIDGE generates module enables, the module address, transfer attributes, byte enables, and write data as inputs to the slave peripherals. The PBRIDGE captures read data from the slave interface and drives it on the system bus.

Separate interface ports are provided for on-platform and off-platform peripherals. The distinction between on-platform and off-platform is made to allow platform-based designs incorporating the PBRIDGE to separate the interface ports to allow for ease of timing closure. In addition, module selects and control register storage for on-platform peripherals are allocated at synthesis time, allowing only needed resources to be implemented. Off-platform module selects and control register storage do not have the same degree of configurability.

The modules that are on-platform and those that are off-platform are detailed in Table 5-3.

Table 5-3. On-Platform and Off-Platform Peripherals

On-Platform	Off-Platform
Enhanced Direct Memory Access (eDMA)	Deserial Serial Peripheral Interface (DSPI)
PBridge A and B	Enhanced Queued Analog-to-Digital Converter (eQADC)
Interrupt Controller (INTC)	Enhanced Serial Communication Interface (eSCI)
Error Correction Status Module (ECSM)	FlexCAN Controller Area Network
System Bus Crossbar Switch (XBAR)	Boot Assist Module (BAM)

Table 5-3. On-Platform and Off-Platform Peripherals (continued)

On-Platform	Off-Platform
	System Integration Unit (SIU)
	Enhanced Modular Input/Output Subsystem (eMIOS)
	Frequency Modulated Phase Locked Loop (FMPLL)
	Enhanced Time Processing Unit (eTPU)
	External Bus Interface (EBI)
	Flash Bus Interface Unit (FBIU)

The PBRIDGE occupies a 64 MB portion of the address space. A 0.5 MB portion of this space is allocated to on-platform peripherals. The remaining 63.5 MBs are available for off-platform devices. The register maps of the slave peripherals are located on 16-KB boundaries. Each slave peripheral is allocated one 16-KB block of the memory map, and is activated by one of the module enables from the PBRIDGE. Up to thirty-two 16-KB external slave peripherals can be implemented, occupying contiguous blocks of 16 KBs. Two global external slave module enables are available for the remaining 63 MBs of address space to allow for customization and expansion of addressed peripheral devices. In addition, a single non-global module enable is also asserted whenever any of the 32 non-global module enables is asserted.

The PBRIDGE is responsible for indicating to slave peripherals if an access is in supervisor or user mode. The PBRIDGE also supports the notion of trusted masters for security purposes. Masters can be individually designated as trusted for reads, trusted for writes, or trusted for both reads and writes, as well as being forced to look as though all accesses from a master are in user mode privilege level.

The PBRIDGE also supports buffered writes, allowing write accesses to be terminated on the system bus in a single clock cycle, and then subsequently performed on the slave interface. Write buffering is controllable on a per-peripheral basis. The PBRIDGE implements a two-entry 32-bit write buffer.

Chapter 6
System Integration Unit (SIU)

6.1 Introduction

This chapter describes the device system integration unit (SIU) that configures and initializes the following controls:

- MCU reset configuration
- System reset operation
- Pad configuration
- External interrupts (324 package only)
- General-purpose I/O (GPIO)
- Internal peripheral multiplexing

6.2 Block Diagram

Figure 6-1 is a block diagram of the SIU. The signals shown on the right side of the diagram are external pins on the device. The SIU registers are accessed through the crossbar switch.

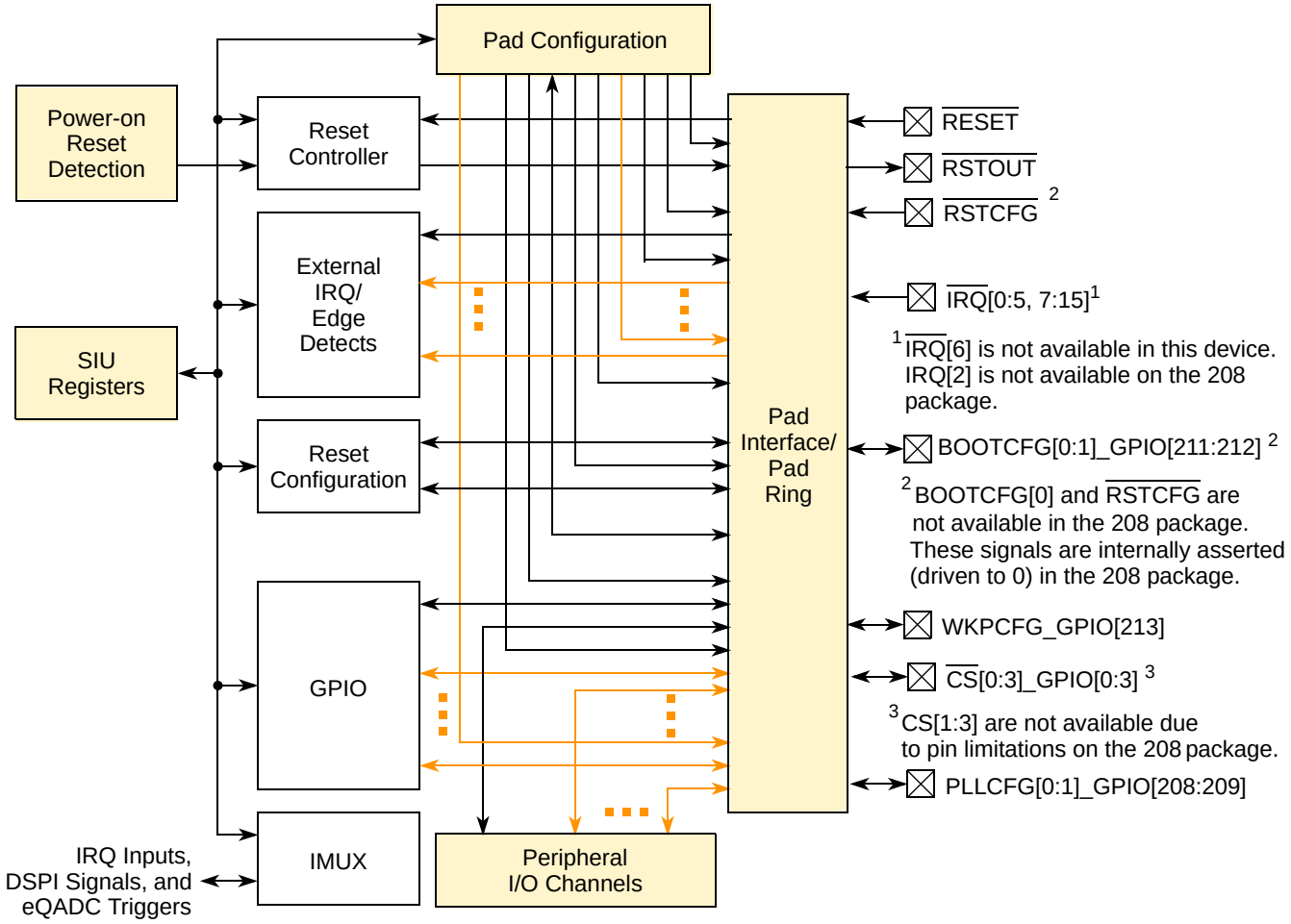


Figure 6-1. SIU Block Diagram

NOTE

The power-on reset detection module, pad interface/pad ring module, and peripheral I/O channels shown shaded in Figure 6-1 are external to the SIU.

6.2.1 Overview

The system integration unit (SIU) is accessed by the e200z3 core through the system bus crossbar switch (XBAR) and the peripheral bridge A (PBRIDGE_A). Table 6-1 lists the features the SIU configures:

Table 6-1. SIU Features

Feature	Description
MCU reset operations	Controls the external pin boot logic
System reset operations	Monitors internal and external reset sources, and drives the $\overline{\text{RSTOUT}}$ signal • Power-on reset support • Reset status register providing last reset source to software • Glitch detection on reset input • Software controlled reset assertion
Pad configuration registers	Enables the configuration and initialization of the I/O pin electrical characteristics using software to select the following: • Active function from the set of multiplexed functions • Pullup and pulldown characteristics of the pin • Slew rate for slow and medium pads • Open drain mode for output pins • Hysteresis for input pins • Drive strength of bus signals for fast pads
External interrupt operations	• 15 interrupt requests • Rising- or falling-edge event detection • Programmable digital filter for glitch rejection
General-purpose I/O (GPIO)	Provides uniform and discrete I/O control of 150 MCU general-purpose I/O pins, where each GPIO signal has an input register and an output register. The number of GPIO pins varies depending on the package.
Internal peripheral multiplexing	Provides flexibility to customize signal/pin assignments for application development that allows you to assign IRQs between external pins and the DSPI.

6.2.2 Modes of Operation

The MPC5500 family of devices has several operating modes for configuring and testing the device:

Table 6-2. SIU Operating Modes

Operating Mode	Description
Normal	In normal mode, the SIU provides the register interface and logic that controls the device and system configuration, the reset controller, and GPIO. The SIU continues operation with no changes in stop mode.
Debug	SIU operation in debug mode is identical to operation in normal mode.

6.3 External Signal Description

Table 6-3 lists the external pins used by the SIU.

Table 6-3. SIU Signal Properties

Name	Function	I/O Type	Pad Type	Pull Up/Down[1]
Resets				
$\overline{\text{RESET}}$	Reset Input	Input	—	Up
$\overline{\text{RSTCFG}}$ [2]	Reset Configuration	Input	—	Up
$\overline{\text{RSTOUT}}$	Reset Output	Output	Slow	—
System Configuration				
GPIO[0:210]	General-Purpose I/O	I/O	Slow	Up/Down
BOOTCFG[0]_ [2] GPIO[211]	Boot Configuration Input General-Purpose I/O	Input I/O	Slow	Down Up/Down
BOOTCFG[1]_ GPIO[212]	Boot Configuration Input General-Purpose I/O	Input I/O	Slow	Down Up/Down
WKPCFG GPIO[213]	Weak Pull Configuration Pin General-Purpose I/O	Input I/O	Slow	Up Up/Down
PLLCFG[0]_ GPIO[208]	Boot Configuration Input General-Purpose I/O	Input I/O	Slow	Down Up/Down
PLLCFG[1]_ GPIO[209]	Boot Configuration Input General-Purpose I/O	Input I/O	Slow	Down Up/Down
External Interrupt (324 package only)				
$\overline{\text{IRQ}}$[0:5, 7:15][3]	External Interrupt Request Input	Input	Slow	—[4]

1 Internal weak pull up/down. The reset weak pull up/down state is given by the pull up/down state for the primary pin function. For example, the reset weak pull up/down state of the BOOTCFG[1]_GPIO[212] pin is weak pullup enabled.

2 $\overline{\text{RSTCFG}}$ and BOOTCFG[0] pins are not available on the 208 package. These signals are internally asserted (driven to 0) in the 208 package.

3 The GPIO and IRQ pins are multiplexed with other functions on the chip. The $\overline{\text{IRQ}}$[6] function is not available in this device. Not all GPIO pins are available on all packages. See Chapter 2, “Signals” for a list of available IRQ and GPIO signals.

4 The weak pull up/down state at reset for the $\overline{\text{IRQ}}$[0:5, 7:15] depends on the primary signals with which they are muxed. The weak pull up/down state for these pins is as follows: $\overline{\text{IRQ}}$[0, 1, 4, 5, 7, 12, 13, 14]: Up, $\overline{\text{IRQ}}$[2, 3, 15]: Down, IRQ[8:11]: WKPCFG. $\overline{\text{IRQ}}$[2] is not available on the 208 package.

6.3.1 Detailed Signal Descriptions

6.3.1.1 Reset Input ($\overline{\text{RESET}}$)

$\overline{\text{RESET}}$ is an active-low input signal asserted by an external device during a power-on reset (POR) or external reset. If $\overline{\text{RESET}}$ asserts for ten clock cycles only, the internal reset signal asserts. Asserting the $\overline{\text{RESET}}$ signal while the device is processing a reset restarts the reset process at the beginning.

$\overline{\text{RESET}}$ has a glitch detector logic that senses electrical fluctuations on the V_{DDEH} input pins that drop below the switch point value for more than two clock cycles. The switch point value is between the maximum V_{IL} and minimum V_{IH} specifications for the V_{DDEH} input pins.

6.3.1.2 Reset Output ($\overline{\text{RSTOUT}}$)

$\overline{\text{RSTOUT}}$ is an active-low output signal that uses a push/pull configuration. It is driven to the low state by the MCU for all internal and external reset sources. After the $\overline{\text{RESET}}$ input signal deasserts, $\overline{\text{RSTOUT}}$ asserts for:

- 16000 clock cycles for devices configured in bypass mode
- 16004 clock cycles for devices configured for FMPLL dual-controller mode (1:1)
- 2400 clock cycles for all other FMPLL modes

To invoke an external software reset, write a 1 to the system external reset (SER) bit in the system reset control register (SIU_SRCR). This asserts $\overline{\text{RSTOUT}}$ for 2400 clock cycles. An external software reset does not execute the BAM module or sample BOOTCFG[0:1].

208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0) in the 208 package.

6.3.1.3 General-Purpose I/O Pins (GPIO[0:213])

208 Package: Not all GPIO pins are available on all packages. See Chapter 2, “Signals” for more information on the GPIO pins available on this device.

The GPIO pins provide general-purpose input and output function. The GPIO pins are generally multiplexed with other I/O pin functions. Each GPIO input and output is separately controlled by an eight-bit input (SIU_GPDI) or output (SIU_GPDO) register.

NOTE

Not all GPIO pins are available on all packages. See Chapter 2, “Signals” for a listing of available GPIO pins.

For more information, see the following sections:

Section 6.4.1.13, “GPIO Pin Data Output Registers 0–213 (SIU_GPDOn)”

Section 6.4.1.14, “GPIO Pin Data Input Registers 0–213 (SIU_GPDIn)”

6.3.1.4 Boot Configuration Pins (BOOTCFG[0:1])

208 Package: BOOTCFG[0] and $\overline{\text{RSTCFG}}$ are not available due to pin limitations and are internally asserted (driven to 0) in the 208 package.

The BOOTCFG value specifies the location and boot mode used by the boot assist module (BAM). All reset sources can read the boot configuration field, BOOTCFG[0:1], except a debug port reset and a software external reset.

The BOOTCFG values are read only if $\overline{\text{RSTCFG}}$ asserts while $\overline{\text{RSTOUT}}$ is asserted. The BOOTCFG signal asserts after $\overline{\text{RSTCFG}}$ to get the boot input information. BOOTCFG[0:1] is sampled four clock cycles before $\overline{\text{RSTOUT}}$ negates, and the latched boot values are stored in the reset status register (SIU_RSR).

If $\overline{\text{RSTCFG}}$ asserts while processing a reset, BOOTCFG[0:1] is sampled. Otherwise, if $\overline{\text{RSTCFG}}$ negates while processing a reset, the following occurs:

1. BOOTCFG[0:1] is not sampled
2. BAM module boots from internal flash (default = 0b00)
3. Boot value from internal flash is written to BOOTCFG[0:1] field in the reset status register (SIU_RSR)
4. BOOTCFG[0:1] values are latched and driven as output signals from the SIU

The BOOTCFG values are used only if $\overline{\text{RSTCFG}}$ asserts while $\overline{\text{RSTOUT}}$ is asserted. Otherwise, the default values for BOOTCFG (0b00) in the reset status register (SIU_RSR) is used, as shown in Table 6-4.

Table 6-4. BOOTCFG[0:1] Configuration

Value	Meaning
0b00	Boot from internal flash memory (default)
0b01	FlexCAN / eSCI boot
0b10	Boot from external memory (324 package only)
0b11	Invalid value

6.3.1.5 I/O Pin Weak Pull Up Reset Configuration Pin (WKPCFG)

The WKPCFG signal is applied when the internal reset signal asserts (indicated by $\overline{\text{RSTOUT}}$ asserting), and is sampled four clock cycles before $\overline{\text{RSTOUT}}$ negates. The WKPCFG value configures the internal weak pullup or weak pulldown pin characteristics after a reset occurs in the eMIOS or eTPU modules.

The value of WKPCFG is latched at reset, stored in the reset status register (SIU_RSR), and updated for all reset sources except the debug port reset and software external reset. The WKPCFG value must be valid and not change until $\overline{\text{RSTOUT}}$ negates.

6.3.1.6 External Interrupt Request Input Pins ($\overline{\text{IRQ}}$[0:5, 7:15])

$\overline{\text{IRQ}}$[0:5, 7:15] are the external interrupt request (IRQ) inputs connect to the SIU IRQ inputs. The external trigger IRQ select register 1 (SIU_ETISR) specifies the $\overline{\text{IRQ}}$[0:5, 7:15] signals that are input to the SIU IRQs.

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

External interrupt requests are triggered by rising- and/or falling-edge events that are enabled by setting a bit in:

- IRQ rising-edge event enable register (SIU_IREER)
- IRQ falling-edge event enable register (SIU_IFEER)

If the bit is set in both registers, both rising- and falling-edge events trigger an interrupt request. Each IRQ has a counter that tracks the number of system clock cycles that occur between the rising- and falling-edge events. An IRQ counter exists for each IRQ rising- or falling-edge event enable bit.

The digital filter length field in the IRQ digital filter register (SIU_IDFR) specifies the minimum number of system clocks that the IRQ signal must hold a logic value to qualify the edge-triggered event as a valid state change. When the number of system clocks in the IRQ counter equals the value in the digital filter length field, the IRQ state latches and the IRQ counter is cleared.

If the previous filtered state of the IRQ does not match the current state, and the rising- or falling-edge event is enabled, the IRQ flag bit is set to 1. For example, the IRQ flag bit is set if a rising-edge event occurs under the following conditions:

- Previous filtered IRQ state was a logic 0
- Current latched IRQ state is a logic 1
- Rising-edge event is enabled for the IRQ

When the counter for an IRQ is not enabled, the state of the IRQ is held in the current and previous state latches. The IRQ counter operates independently of the IRQ or overrun flag bit. Clearing the IRQ flag or overrun flag bits does not clear or reload the counter.

See the following sections for more information:

Section 6.4.1.4, “External Interrupt Status Register (SIU_EISR)”
Section 6.4.1.9, “IRQ Rising-Edge Event Enable Register (SIU_IREER)”
Section 6.4.1.10, “IRQ Falling-Edge Event Enable Register (SIU_IFEER)”
Section 6.4.1.11, “IRQ Digital Filter Register (SIU_IDFR)”

Rising- or falling-edge events are enabled by setting the bits in SIU_IREER or SIU_IFEER. If the same bit location is set in both registers, both rising- and falling-edge events set the IRQ FLAG bit in Section 6.4.1.4, “External Interrupt Status Register (SIU_EISR).”

6.3.1.6.1 External Interrupts

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

The IRQ signals map to 15 independent interrupt requests output from the SIU. The IRQ flag bit is set when a rising-edge and/or falling-edge event occurs for the IRQ. An external IRQ signal is asserted when all of the following occur:

- Enable bit is set in the IRQ rising- and/or falling-edge event registers (SIU_IREER, SIU_IFEER)
- IRQ flag bit is set in the external interrupt status register (SIU_EISR)
- Enable bit is cleared in the DMA/Interrupt request enable register (SIU_DIRER)
- Select bit is cleared in the DMA/Interrupt select register (SIU_DIRSR)

See the following sections for more information:

Section 6.4.1.5, “DMA/Interrupt Request Enable Register (SIU_DIRER)”
Section 6.4.1.6, “DMA/Interrupt Request Select Register (SIU_DIRSR)”

6.3.1.6.2 DMA Transfers

DMA IRQ signals ($\overline{\text{IRQ}}$[0] through $\overline{\text{IRQ}}$[3]) map to four independent DMA transfer *or* interrupt request outputs configured in the SIU. A DMA transfer or interrupt request asserts when all of the following occur:

- IRQ flag bit is set in the external interrupt status register (SIU_EISR)
- Enable bit is set in the DMA transfer or interrupt request enable register (SIU_DIRER)
- Select bit is set in the DMA transfer or interrupt request select register (SIU_DIRSR)

The SIU receives a ‘DMA transfer done’ signal for each DMA or interrupt request transmitted. When the ‘DMA done’ signal asserts, the IRQ flag bit is cleared.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

See the following sections for more information:

Section 6.4.1.5, “DMA/Interrupt Request Enable Register (SIU_DIRER)”
Section 6.4.1.6, “DMA/Interrupt Request Select Register (SIU_DIRSR)”

6.3.1.6.3 Overruns

An overrun IRQ exists for each overrun flag bit in the overrun status register (SIU_OSR). An overrun IRQ asserts when all of the following occur:

- Enable bit is set in the IRQ rising- and/or falling-edge event registers (SIU_IREER, SIU_IFEER)
- IRQ flag bit is set in the external interrupt status register (SIU_EISR)
- Bit is set in the overrun request enable and overrun status registers (SIU_ORER, SIU_OSR)
- Rising- or falling-edge event triggers an interrupt request

The SIU outputs one overrun IRQ bit that is the logical OR of all of the IRQ overrun bits.

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

See the following sections for more information:

Section 6.4.1.4, “External Interrupt Status Register (SIU_EISR)”

Section 6.4.1.7, “Overrun Status Register (SIU_OSR)”

Section 6.4.1.8, “Overrun Request Enable Register (SIU_ORER)”

6.3.1.6.4 Edge Detects

An IRQ asserts when an:

- Edge-detect event is enabled
- Edge-detect event occurs

To assert an IRQ when an edge-detect event occurs:

1. Set the enable bit in the IRQ rising- and falling-edge event enable registers (SIU_IREER, SIU_IFEER)
2. Clear the enable bits for the DMA/Interrupt request enable register (SIU_DIRER)

The IRQ bit is set in the external IRQ status register (SIU_EISR) when an edge-detect event occurs for that IRQ.

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

See the following sections for more information:

Section 6.4.1.4, “External Interrupt Status Register (SIU_EISR)”

Section 6.4.1.9, “IRQ Rising-Edge Event Enable Register (SIU_IREER)”

Section 6.4.1.10, “IRQ Falling-Edge Event Enable Register (SIU_IFEER)”

6.4 Memory Map and Register Definition

Table 6-5 is the address map for the SIU registers. All register addresses are given as an offset of the SIU base address.

Table 6-5. SIU Address Map

Address	Register Name	Register Description	Bits
Base (0xC3F9_0000)	—	Reserved	—
Base + 0x0004	SIU_MIDR	MCU ID register	32
Base + 0x0008	—	Reserved	—
Base + 0x000C	SIU_RSR	Reset status register	32

Table 6-5. SIU Address Map (continued)

Address	Register Name	Register Description	Bits
Base + 0x0010	SIU_SRCR	System reset control register	32
Base + 0x0014	SIU_EISR	SIU external interrupt status register	32
Base + 0x0018	SIU_DIRER	DMA/interrupt request enable register	32
Base + 0x001C	SIU_DIRSR	DMA/interrupt request select register	32
Base + 0x0020	SIU_OSR	Overrun status register	32
Base + 0x0024	SIU_ORER	Overrun request enable register	32
Base + 0x0028	SIU_IREER	IRQ rising-edge event enable register	32
Base + 0x002C	SIU_IFEER	IRQ falling-edge event enable register	32
Base + 0x0030	SIU_IDFR	IRQ digital filter register	32
Base + (0x0034–0x003F)	—	Reserved	—
Base + (0x0040–0x02EC)	SIU_PCR0–SIU_PCR342	Pad configuration registers 0–342 [1]	16
Base + (0x02F0–0x005F)	—	Reserved	—
Base + (0x0600–0x06D4)	SIU_GPDO0–SIU_GPDO213	GPIO pin data output registers 0–213[2]	8
Base + (0x06D8–0x07FF)	—	Reserved	—
Base + (0x0800–0x08D4)	SIU_GPDI0–SIU_GPDI213	GPIO pin data input registers 0–213[2]	8
Base + (0x08D8–0x08FF)	—	Reserved	—
Base + 0x0900	SIU_ETISR	eQADC trigger input select register	32
Base + 0x0904	SIU_EISR	External IRQ input select register	32
Base + 0x0908	SIU_DISR	DSPI input select register	32
Base + (0x090C–0x097)	—	Reserved	—
Base + 0x0980	SIU_CCR	Chip configuration register	32
Base + 0x0984	SIU_ECCR	External clock control register	32
Base + 0x0988	SIU_CARH	Compare A high register	32
Base + 0x098C	SIU_CARL	Compare A low register	32
Base + 0x0990	SIU_CBRH	Compare B high register	32
Base + 0x0994	SIU_CBRL	Compare B low register	32
Base + (0x0998–0x09FF)	—	Reserved	—

[1] Gaps exist in the pad configuration where I/O pins are not available in this package.

[2] Gaps exist in this memory space where I/O pins are not available in this package.

6.4.1 Register Descriptions

The register figures use the following notational conventions in this section:

w1c	Write 1 to clear the bit to 0.
—	Not applicable.
	Reserved or unimplemented bit.
U	Bit value is uninitialized upon reset.
u	Bit value is unchanged upon reset.

6.4.1.1 MCU ID Register (SIU_MIDR)

The SIU_MIDR contains the part identification number and mask revision number specific to the device. The part number is a read-only field that is mask programmed with the part number of the device. The part number is changed if a new module is added to the device or a memory size is changed, for example. It is not changed for bug fixes or process changes.

The mask number is a read-only field that is mask programmed with the specific mask revision level of the device.

The MCU ID register is 32-bits. Figure 6-2 shows the MCU ID register values.

Address: Base + 0x0004 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PARTNUM															
W																
Reset	0	1	0	1	0	1	0	1	0	0	1	1	0	1	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	MASKNUM_MAJOR				MASKNUM_MINOR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-2. MCU ID Register (SIU_MIDR)

Table 6-6. SIU_MIDR Field Descriptions

Field	Description
0–15 PARTNUM [0:15]	MCU part number. Read-only, mask programmed part identification number of the MCU. Reads 0x5534 for the MPC5534.
16–23	Reserved

Table 6-6. SIU_MIDR Field Descriptions (continued)

Field	Description
24–27 MASKNUM_MAJOR [0:3]	Major revision number of MCU mask. Read-only, mask programmed mask number of the MCU. Reads 0x0000 for the initial mask set, and changes sequentially for each mask set.
28–31 MASKNUM_MINOR [0:3]	Minor revision number of MCU mask. Read-only, mask programmed mask number of the MCU. Reads 0x0000 for the initial mask set, and changes sequentially for each mask set.

6.4.1.2 Reset Status Register (SIU_RSR)

The SIU_RSR reflects the most recent source, or sources of reset, and the state of configuration pins at reset. This register contains one bit for each reset source, indicating that the last reset was power-on reset (POR), external, software system, software external reset, watchdog, loss of PLL lock, loss of clock or checkstop reset. A reset status bit set to logic one indicates the type of reset that occurred. Once set, the reset source status bits in the SIU_RSR remain set until another reset occurs. In the following cases more than one reset bit is set:

- *If a power-on reset request has negated* and the device is still in the resulting reset, and then an external reset is requested, both the power-on and external reset status bits are set. In this case, the device started the reset sequence due to a power-on reset request, but it ended the reset sequence after an external reset request.
- *If a software external reset is requested*, the SERF flag bit is set, but no previously set bits in the SIU_RSR are cleared. The SERF bit is cleared by writing a 1 to the bit location or when another reset source is asserted.
- *If any of the loss of clock, loss of lock, watchdog or checkstop reset requests occur on the same clock cycle*, and no other higher priority reset source is requesting reset (see Table 6-7), the reset status bits for all of the requesting resets are set.

Simultaneous reset requests are prioritized. When reset requests of different priorities occur on the same clock cycle, the lower priority reset request is ignored. Only the highest priority reset request's status bit is set. Except for a power-on reset request and condition above, all reset requests of any priority are ignored until the device exits reset.

Table 6-7. Reset Source Priorities

Reset Source	Priority
Power on reset (POR) and external reset (Group 0)	Highest
Software system reset (Group1)	Lowest-high
Loss of clock, loss of lock, watchdog, checkstop (Group2)	Highest-low
Software external reset (Group 3)	Lowest

The WKPCFG bit contains the value of the signal on the WKPCFG pin at the last reset. The BOOTCFG field contains the values on the BOOTCFG[0:1] pins at the last reset.

Address: Base + 0x000C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PORS	ERS	LLRS	LCRS	WDRS	CRS	0	0	0	0	0	0	0	0	SSRS	SERF
W																W1c
Reset[1]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WKP CFG[2]	0	0	0	0	0	0	0	0	0	0	0	0	BOOTCFG		RGF
W																W1c
Reset[1]	U[2]														U[3]	0

1 The reset status register receives its reset values during power-on reset.

2 The reset value of the WKPCFG bit is determined by the value on the WKPCFG pin at reset.

3 The reset value of the BOOTCFG field is determined by the values on the BOOTCFG[0:1] pins at reset. BOOTCFG[0] is not available due to pin limitations, and is internally asserted (driven to 0) in the 208 package.

Figure 6-3. Reset Status Register (SIU_RSR)

Table 6-8. SIU_RSR Field Descriptions

Field	Description
0 PORS	Power-on reset status. 0 Another reset source has been acknowledged by the reset controller since the last assertion of the power-on reset input. 1 The power-on reset input to the reset controller has been asserted and no other reset source has been acknowledged since that assertion of the power-on reset input except an external reset.
1 ERS	External reset status. 0 The last reset source acknowledged by the reset controller was not a valid assertion of the $\overline{\text{RESET}}$ pin. 1 The last reset source acknowledged by the reset controller was a valid assertion of the $\overline{\text{RESET}}$ pin.
2 LLRS	Loss of lock reset status. 0 The last reset source acknowledged by the reset controller was not a loss of PLL lock reset. 1 The last reset source acknowledged by the reset controller was a loss of PLL lock reset.
3 LCRS	Loss of clock reset status. 0 The last reset source acknowledged by the reset controller was not a loss of clock reset. 1 The last reset source acknowledged by the reset controller was a loss of clock reset.
4 WDRS	Watchdog timer/debug reset status. 0 The last reset source acknowledged by the reset controller was not a watchdog timer or debug reset. 1 The last reset source acknowledged by the reset controller was a watchdog timer or debug reset.
5 CRS	Checkstop reset status. 0 The last reset source acknowledged by the reset controller was not an enabled checkstop reset. 1 The last reset source acknowledged by the reset controller was an enabled checkstop reset.
6–13	Reserved
14 SSRS	Software system reset status. 0 The last reset source acknowledged by the reset controller was not a software system reset. 1 The last reset source acknowledged by the reset controller was a software system reset.

Table 6-8. SIU_RSR Field Descriptions (continued)

Field	Description
15 SERF	Software external reset flag. 0 This bit has been cleared from a 1 to a 0 by a write of 1 to it when it was a 1 or the software external reset input to the reset controller has not been asserted. 1 The software external reset input to the reset controller has been asserted while this bit was a 0.
16 WKPCFG	Weak pull configuration pin status 0 The WKPCFG pin latched during the last reset was a logical 0 and weak pull down is the default setting 1 The WKPCFG pin latched during the last reset was a logical 1 and weak pullup is the default setting
17–28	Reserved
29–30 BOOTCFG	Reset configuration pin status. Holds the value of the BOOTCFG pins that were latched on the last negation of the RSTOUT pin, if the RSTCFG pin was asserted. If the RSTCFG pin was not asserted at the last negation of RSTOUT, and the lower half or least significant half word of the censorship control word equals 0xFFFF or 0x0000, the BOOTCFG field is set to the value 0b10. Otherwise, if the RSTCFG pin was negated at the last negation of RSTOUT and the lower half of the censorship control word does not equal 0xFFFF or 0x0000, then the BOOTCFG field is set to the value 0b00. The BOOTCFG field is used by the BAM program to determine the location of the reset configuration half word. See Table 4-11 for a translation of the reset configuration half word location from the BOOTCFG field value. NOTE: BOOTCFG[0] is not available due to pin limitations and is internally asserted (drive to 0) in the 208 package.
31 RGF	Reset glitch flag. Set by the reset controller when a glitch is detected on the RESET pin. This bit is cleared by the assertion of the power-on reset input to the reset controller, or a write of 1 to the RGF bit. See Section 6.5.2.1, “RESET Pin Glitch Detect,” for more information on glitch detection. 0 No glitch has been detected on the RESET pin. 1 A glitch has been detected on the RESET pin.

6.4.1.3 System Reset Control Register (SIU_SRCR)

The system reset control register allows software to generate either a system or external reset. The software system reset causes an internal reset, while the software external reset only causes the external RSTOUT pin to be asserted. When written to 1, the SER bit automatically clears.

Address: Base + 0x0010 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SER [1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	SSR [2]	w1c														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CRE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1[3]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-4. System Reset Control Register (SIU_SRCR)

1 Write 1 to the SER bit to generate a software external reset. A write of 0 to this bit has no effect. When the reset completes, the SER bit is cleared to 0.

2 The SSR bit always reads 0. A write of 0 to this bit has no effect.

[3] The CRE bit is set to 1 by POR. Other reset sources cannot set the CRE bit.

Table 6-9. SIU_SRCR Field Descriptions

Field	Description
0 SSR	Software system reset. Used to generate a software system reset. Writing a 1 to this bit causes an internal reset. The software system reset is processed as a synchronous reset. The bit is automatically cleared on the assertion of any other reset source except a software external reset. 0 Do not generate a software system reset. 1 Generate a software system reset.
1 SER	Software external reset. Used to generate a software external reset. Writing a 1 to this bit causes the $\overline{\text{RSTOUT}}$ pin to be asserted for 2400 clocks, but the internal reset is not asserted. The bit is automatically cleared when the software external reset completes or any other reset source is asserted. Once a software external reset has been initiated, the $\overline{\text{RSTOUT}}$ pin is negated if this bit is cleared before the 2400 clock period expires. 0 Do not generate a software external reset. 1 Generate a software external reset. **Note:** If the PLL is configured for dual controller mode writing a 1 to SER causes the $\overline{\text{RSTOUT}}$ pin to be asserted for 16000 clocks. See Section 4.2.2, "Reset Output (RSTOUT)."
2–15	Reserved
16 CRE	Checkstop reset enable. Writing a 1 to this bit enables a reset when the checkstop reset request input is asserted. The checkstop reset request input is a synchronous internal reset source. The CRE bit defaults to checkstop reset enabled at POR. If this bit is cleared, it remains cleared until the next POR. 0 No reset occurs when the checkstop reset input to the reset controller is asserted. 1 A reset occurs when the checkstop reset input to the reset controller is asserted.
17–31	Reserved

6.4.1.4 External Interrupt Status Register (SIU_EISR)

The external interrupt status register is used to record edge triggered events on the $\overline{\text{IRQ}}$[0:5, 7:15] inputs to the SIU. When an edge triggered event is enabled in the SIU_IREER or SIU_IFEER for an $\overline{\text{IRQ}}$[*n*] input and then sensed, the corresponding SIU_EISR flag bit is set. The IRQ flag bit is set regardless of the state of the corresponding DMA/interrupt request enable bit in SIU_DIRER. The IRQ flag bit remains set until cleared by software or through the servicing of a DMA request. The IRQ flag bits are cleared by writing a 1 to the bits. A write of 0 has no effect.

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

Address: Base + 0x0014 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF15	EIF14	EIF13	EIF12	EIF11	EIF10	EIF9	EIF8	EIF7	EIF6	EIF5	EIF4	EIF3	EIF2	EIF1	EIF0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-5. SIU External Interrupt Status Register (SIU_EISR)

Table 6-10. SIU_EISR Field Descriptions

Field	Description
0–15	Reserved
16–31 EIF*n*	External interrupt request flag *n*. This bit is set when an edge triggered event occurs on the corresponding IRQ*n* input. 0 No edge triggered event has occurred on the corresponding $\overline{\text{IRQ}}$[*n*] input. 1 An edge triggered event has occurred on the corresponding $\overline{\text{IRQ}}$[*n*] input.

6.4.1.5 DMA/Interrupt Request Enable Register (SIU_DIRER)

The SIU_DIRER allows the assertion of a DMA or interrupt request if the corresponding flag bit is set in the SIU_EISR. The external interrupt request enable bits enable the interrupt or DMA request. There is only one interrupt request from the SIU to the interrupt controller. The EIRE bits determine which external interrupt request flag bits assert the interrupt request signal.

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

Address: Base + 0x0018 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIRE 15	EIRE 14	EIRE 13	EIRE 12	EIRE 11	EIRE 10	EIRE 9	EIRE 8	EIRE 7	EIRE 6	EIRE 5	EIRE 4	EIRE 3	EIRE 2	EIRE 1	EIRE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-6. SIU DMA/Interrupt Request Enable Register (SIU_DIRER)

Table 6-11. SIU_DIRER Field Descriptions

Field	Description
0–15	Reserved
16–31 EIRE*n*	External interrupt request enable *n*. Enables the assertion of the interrupt request from the SIU to the interrupt controller when an edge triggered event occurs on the IRQ*n* pin. 0 External interrupt request is disabled. 1 External interrupt request is enabled.

6.4.1.6 DMA/Interrupt Request Select Register (SIU_DIRSR)

The SIU_DIRSR allows selection between a DMA or interrupt request for events on the $\overline{\text{IRQ}}$[0:3] inputs. The SIU_DIRSR selects between DMA and interrupt requests. If the corresponding bits are set in SIU_EISR and the SIU_DIRER, then the DMA/interrupt request select bit determines whether a DMA or interrupt request is asserted.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

Address: Base + 0x001C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DIRS 3	DIRS 2	DIRS 1	DIRS 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-7. DMA/Interrupt Request Select Register (SIU_DIRSR)

Table 6-12. SIU_DIRER Field Descriptions

Field	Description
0–27	Reserved
28–31 DIRS*n*	DMA/interrupt request select *n*. Selects between a DMA or interrupt request when an edge triggered event occurs on the corresponding IRQ*n* pin. 0 Interrupt request is selected. 1 DMA request is selected.

6.4.1.7 Overrun Status Register (SIU_OSR)

The SIU_OSR contains flag bits that record an overrun.

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

Address: Base + 0x0020 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	OVF 15	OVF 14	OVF 13	OVF 12	OVF 11	OVF 10	OVF 9	OVF 8	OVF 7	OVF 6	OVF 5	OVF 4	OVF 3	OVF 2	OVF 1	OVF 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-8. Overrun Status Register (SIU_OSR)

Table 6-13. SIU_OSR Field Descriptions

Field	Function
0–15	Reserved
16–31 OVF*n*	Overrun flag *n*. This bit is set when an overrun occurs on the corresponding IRQ*n* pin. 0 No overrun has occurred on the corresponding IRQ*n* pin. 1 An overrun has occurred on the corresponding IRQ*n* pin.

6.4.1.8 Overrun Request Enable Register (SIU_ORER)

The SIU_ORER contains bits to enable an overrun if the corresponding flag bit is set in the SIU_OSR. If any overrun request enable bit and the corresponding flag bit are set, the single combined overrun request from the SIU to the interrupt controller is asserted.

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

Address: Base + 0x0024 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ORE 15	ORE 14	ORE 13	ORE 12	ORE 11	ORE 10	ORE 9	ORE 8	ORE 7	ORE 6	ORE 5	ORE 4	ORE 3	ORE 2	ORE 1	ORE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-9. Overrun Request Enable Register (SIU_ORER)

Table 6-14. SIU_ORER Field Descriptions

Field	Function
0–15	Reserved
16–31 ORE*n*	Overrun request enable *n*. Enables the overrun request when an overrun occurs on the $\overline{\text{IRQ}}$[*n*] pin. Bit 31 (ORE0) is the enable overrun flag for $\overline{\text{IRQ}}$[0]; bit 16 (ORE15) is overrun flag for $\overline{\text{IRQ}}$[15]. 0 Overrun request is disabled. 1 Overrun request is enabled.

6.4.1.9 IRQ Rising-Edge Event Enable Register (SIU_IREER)

The SIU_IREER allows rising edge triggered events to be enabled on the corresponding $\overline{\text{IRQ}}$[*n*] pins. Rising and falling edge events can be enabled by setting the corresponding bits in both the SIU_IREER and SIU_IFEER.

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

Address: Base + 0x0002 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IREE 15	IREE 14	IREE 13	IREE 12	IREE 11	IREE 10	IREE 9	IREE 8	IREE 7	IREE 6	IREE 5	IREE 4	IREE 3	IREE 2	IREE 1	IREE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-10. IRQ Rising-Edge Event Enable Register (SIU_IREER)

Table 6-15. SIU_IREER Field Descriptions

Field	Function
0–15	Reserved
16–31 IREE*n*	IRQ rising-edge event enable *n*. Enables rising-edge triggered events on the corresponding IRQ*n* pin. 0 Rising edge event is disabled. 1 Rising edge event is enabled.

6.4.1.10 IRQ Falling-Edge Event Enable Register (SIU_IFEER)

The SIU_IFEER allows falling edge triggered events to be enabled on the corresponding $\overline{\text{IRQ}}$[*n*] pins. Rising and falling edge events can be enabled by setting the corresponding bits in both the SIU_IREER and SIU_IFEER.

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

Address: Base + 0x002C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IFEE 15	IFEE 14	IFEE 13	IFEE 12	IFEE 11	IFEE 10	IFEE 9	IFEE 8	IFEE 7	IFEE 6	IFEE 5	IFEE 4	IFEE 3	IFEE 2	IFEE 1	IFEE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-11. IRQ Falling-Edge Event Enable Register (SIU_IFEER)

The following table describes the fields in the IRQ falling-edge event enable register:

Table 6-16. SIU_IFEER Field Descriptions

Field	Function
0–15	Reserved
16–31 IFEE*n*	IRQ falling-edge event enable *n*. Enables falling-edge triggered events on the corresponding $\overline{\text{IRQ}}$[*n*] pin. 0 Falling edge event is disabled. 1 Falling edge event is enabled.

6.4.1.11 IRQ Digital Filter Register (SIU_IDFR)

The SIU_IDFR specifies the amount of digital filtering on the $\overline{\text{IRQ}}$[0:5, 7:15] pins. The digital filter length field specifies the number of system clocks that define the period of the digital filter and the minimum time a signal must be held in the active state on the IRQ pins to be recognized as an edge triggered event.

NOTE

$\overline{\text{IRQ}}$[6] is not available in this device.

208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

Address: Base + 0x0030 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DFL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-12. External IRQ Digital Filter Register (SIU_IDFR)

Table 6-17. SIU_IDFR Field Descriptions

Field	Function
0–27	Reserved
28–31 DFL	Digital filter length. Defines the digital filter period on the $\overline{\text{IRQ}}n$ inputs according to the following equation: $\text{Filter Period} = (\text{SystemClockPeriod} \times 2^{\text{DFL}}) + 1(\text{SystemClockPeriod})$ For a 82-MHz system clock, this gives a range of 24ns to 400µs. The minimum time of three clocks accounts for synchronization of the IRQ input pins with the system clock.

6.4.1.12 Pad Configuration Registers (SIU_PCR)

The following subsections define the SIU_PCRs for all device pins that allow configuration of the pin function, direction, and static electrical attributes. The information presented pertains to which bits and fields are active for a given pin or group of pins, and the reset state of the register. The reset state of SIU_PCRs given in the following sections is that prior to execution of the BAM program. The BAM program can change certain SIU_PCRs based on the reset configuration. See the BAM section of the manual for more detail.

For all PCRs, if the pin is configured as an input, the ODE, SRC, and DSC bits do not apply. If the pin is configured as an output, the HYS bit does not apply. When a pin is configured as an output, the weak internal pull up/down is disabled regardless of the WPE or WPS settings in the PCR.

The IBE and OBE bit definitions are specific for each PCR. In cases where an I/O function is input or output only the IBE and OBE bits do not need to be set to enable the input or output. In cases where an I/O function can be either an input and output, the IBE and OBE bits must be set accordingly (IBE = 1 for input, and OBE = 1 for output). For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits have no effect.

For all PCRs where a GPIO function is available on the pin, if the pin is configured as an output and the IBE bit is set, the value of the pin is shown in its GPDI*x_x* register. Negating the IBE bit when the pin is configured as an output reduces noise and power consumption.

The SIU_PCRs are 16-bit registers that can be read or written as 16-bit values aligned on 16-bit boundaries, or as 32-bit values aligned on 32-bit address boundaries. Table 6-18 describes the SIU_PCR fields.

NOTE

Not all of the fields occur in all SIU_PCRs, depending on the type of pad it controls. See the specific SIU_PCR definition.

All pin names begin with the primary function, followed by the alternate function, and then GPIO. The primary function is not available on all MPC5500 devices.

In some cases, the third function can be a secondary alternate, which supersedes the GPIO. Those exceptions are noted in the documentation. For example, SIU_PCR85 configures the CNTXB_PCSC[3]_GPIO[85] muxed signal, where CNTXB is the primary function, PCSC[3] is the alternate function.

Figure 6-13 shows a sample PCR register with all bit fields displayed:

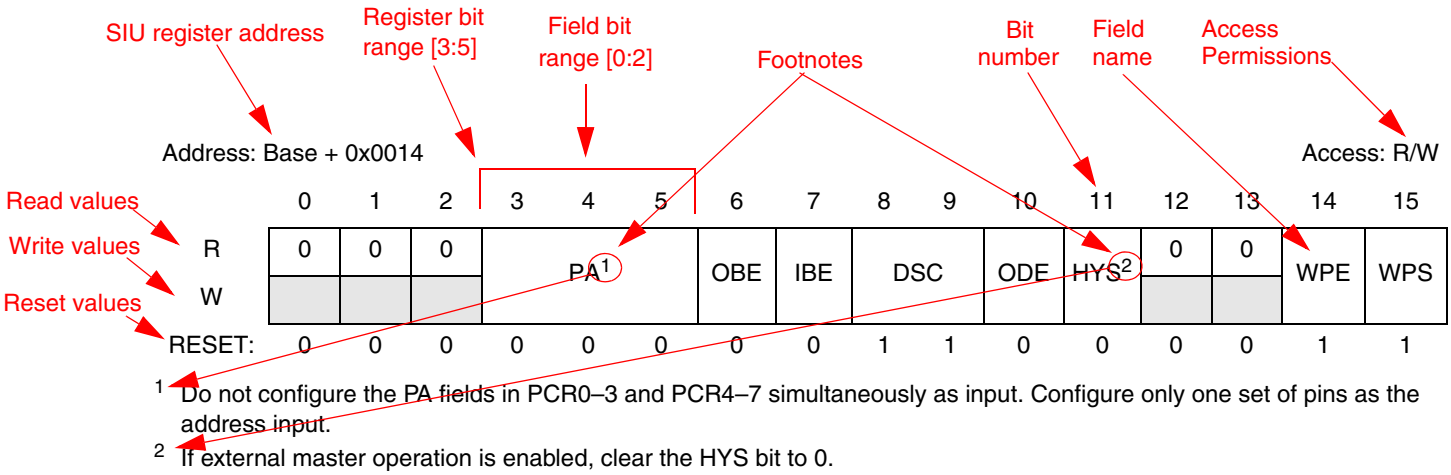


Figure 6-13. Sample PCR Register Description

For identification of the source module for primary and alternate functions, and the description of these signals, refer to Chapter 2, “Signals” of this manual. Refer to the chapter for the module that uses the signal for an additional signal description.

Table 6-18. SIU_PCR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0–2</td><td>Reserved</td></tr>
<tr><td>3–5
PA[0:2]</td><td>Pin assignment. Selects the function of a multiplexed pad. A separate port enable output signal from the SIU is asserted for each value of this register. The size of the field can be from 1 to 3 bits, depending on the amount of multiplexing on the pad.
<table>
<tr><th colspan="9">PA Bit Field</th><th rowspan="2">Pin Function[1]</th></tr>
<tr><th colspan="3">1-bit[2]
(2 Functions)</th><th colspan="3">2-bit[2]
(4 Functions)</th><th colspan="3">3-bit
(5 Functions)</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>GPIO</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>Primary function</td></tr>
<tr><td></td><td></td><td></td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>Alternate function 1</td></tr>
<tr><td></td><td></td><td></td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>Main primary function [3]</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td>0</td><td>0</td><td>Alternate function 2</td></tr>
<tr><td colspan="3" rowspan="4">The shaded columns indicate invalid bits in 1- and 2-bit PA fields; the shaded rows indicate invalid values for 3-bit PA fields.</td><td></td><td></td><td></td><td>1</td><td>0</td><td>1</td><td>Invalid value</td></tr>
<tr><td></td><td></td><td></td><td>1</td><td>1</td><td>0</td><td>Invalid value</td></tr>
<tr><td></td><td></td><td></td><td>1</td><td>1</td><td>1</td><td>Invalid value</td></tr>
</table>
[1] For all SIU_PCRs that do not comply with these rules, the PA definition is given explicitly with the SIU_PCR definition.
[2] For future software compatibility, it is recommended that all PA fields be treated as 3-bit fields, with the unused bits written as 0.
[3] The main primary function is used when the primary function is not available on the package or is used for a different purpose.</td></tr>
<tr><td>6
OBE</td><td>Output buffer enable. Enables the pad as an output and drives the output buffer enable signal.
0 Output buffer for the pad is disabled.
1 Output buffer for the pad is enabled.</td></tr>
<tr><td>7
IBE</td><td>Input buffer enable. Enables the pad as an input and drives the input buffer enable signal.
0 Input buffer for the pad is disabled.
1 Input buffer for the pad is enabled.</td></tr>
<tr><td>8–9
DSC[0:1]</td><td>Drive strength control. Controls the pad drive strength. Drive strength control pertains to pins with the fast I/O pad type.
00 10 pF Drive Strength
01 20 pF Drive Strength
10 30 pF Drive Strength
11 50 pF Drive Strength</td></tr>
<tr><td>10
ODE</td><td>Open drain output enable. Controls output driver configuration for the pads. Either open drain or push/pull driver configurations can be selected. This feature applies to output pins only.
0 Open drain is disabled for the pad (push/pull driver enabled).
1 Open drain is enabled for the pad.</td></tr>
<tr><td>11
HYS</td><td>Input hysteresis. Controls whether hysteresis is enabled for the pad.
0 Hysteresis is disabled for the pad.
1 Hysteresis is enabled for the pad.</td></tr>
</table>

Table 6-18. SIU_PCR Field Descriptions (continued)

Field	Description
12–13 SRC[0:1]	Slew rate control. Controls slew rate for the pad. Slew rate control pertains to pins with slow or medium I/O pad types, and the output signals are driven according to the value of this field. Actual slew rate is dependent on the pad type and load. See the electrical specification for this information 00 Minimum slew rate (slowest) 01 Medium slew rate 10 Invalid value 11 Maximum slew rate (fastest)
14 WPE	Weak pull up/down enable. Controls whether the weak pull up/down devices are enabled/disabled for the pad. Pull up/down devices are enabled by default. 0 Weak pull device is disabled for the pad. 1 Weak pull device is enabled for the pad.
15 WPS	Weak pull up/down select. Controls whether weak pull up or weak pull down devices are used for the pad when weak pull up/down devices are enabled. The WKPCFG pin determines whether pull up or pull down devices are enabled at reset. The WPS bit determines whether weak pull up or pull down devices are used after reset, or for pads in which the WKPCFG pin does not determine the reset weak pull up/down state. 0 The pull down value is enabled for the pad. 1 The pull up value is enabled for the pad.

6.4.1.12.1 Pad Configuration Registers 0–3 (SIU_PCR0–SIU_PCR3)

The SIU_PCR0–SIU_PCR3 registers control the pin function, direction, and static electrical attributes of the $\overline{CS}$[0:3]_ADDR[8:11]_GPIO[0:3] pins.

208 Package: $\overline{CS}$[1:3]_ADDR[9:11]_GPIO[1:3] are not available due to pin limitations in the 208 package.

Address: SIU_BASE + (0x0040–0x0046) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	DSC		ODE[3]	HYS	0	0	WPE[4]	WPS[4]
W																
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

1 When configured as $\overline{CS}$[0:3], the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as $\overline{CS}$[0:3] or GPI, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

3 When configured as $\overline{CS}$[0:3], set the ODE bit to zero.

4 See the EBI section for weak pull up settings when configured as $\overline{CS}$[0:3].

Figure 6-14. $\overline{CS}$[0:3]_ADDR[8:11]_GPIO[0:3] Pad Configuration Registers (SIU_PCR0–SIU_PCR3)

See Table 6-18 for bit field definitions. Table 6-19 lists the PA values for $\overline{CS}$[1:3]_ADDR[9:11]_GPIO[1:3].

Table 6-19. PCR0–PCR3 PA Field Descriptions

PA Field	Pin Function
0b00	GPIO[0:3]
0b01	$\overline{CS}$[0:3]
0b10	ADDR[8:11]
0b11	$\overline{CS}$[0:3]

6.4.1.12.2 Pad Configuration Registers 8–27 (SIU_PCR8–SIU_PCR27)

The SIU_PCR8–SIU_PCR27 registers control the pin function, direction, and static electrical attributes of the ADDR[12:31]_GPIO[8:27] pins.

208 Package: ADDR[12:31]_GPIO[8:27] pins are not available due to pin limitations.

Address: SIU_BASE + (0x0048–0x0076) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	DSC		ODE[3]	HYS	0	0	WPE[4]	WPS[4]
W																
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

1 When configured as ADDR[12:31], the OBE bit has no effect. When configured as GPO, set the OBE bit to one.
2 When configured as ADDR[12:31] or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.
3 When configured as ADDR[12:31], set the ODE bit to zero.
4 See the EBI section for weak pull up settings when configured as ADDR[12:31]

Figure 6-15. Pad Configuration Registers 8–27 (SIU_PCR8–SIU_PCR27)

See Table 6-18 for bit field definitions.

6.4.1.12.3 Pad Configuration Registers 28–43 (SIU_PCR28–SIU_PCR43)

The SIU_PCR28–SIU_PCR43 registers control the pin function, direction, and static electrical attributes of the DATA[0:15]_GPIO[28:43] pins.

208 Package: DATA[1:15] pins are not available in the 208 package due to pin limitations.

Address: SIU_BASE + (0x0078–0x0096) (16) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	DSC		ODE[3]	HYS	0	0	WPE[4]	WPS[4]
W																
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

1 When configured as DATA[0:15], the OBE bit has no effect. When configured as GPO, set the OBE bit to one.
2 When configured as DATA[0:15] or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.
3 When configured as DATA[0:15], set the ODE bit to zero.
4 See the EBI section for weak pull up settings when configured as DATA[0:15].

Figure 6-16. DATA[0:15]_GPIO[28:43] Pad Configuration Registers (SIU_PCR28–SIU_PCR43)

See Table 6-18 for bit field definitions.

6.4.1.12.4 Pad Configuration Register 62 (SIU_PCR62)

The SIU_PCR62 register controls the pin function, direction, and static electrical attributes of the RD_$\overline{\text{WR}}$_GPIO[62] pin.

208 Package: RD_$\overline{\text{WR}}$_GPIO[62] pin is not available in the 208 package due to pin limitations.

Address: SIU_BASE + 0x00BC Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	DSC		ODE[3]	HYS	0	0	WPE[4]	WPS[4]
W																
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

1 When configured as RD_$\overline{\text{WR}}$, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.
2 When configured as RD_$\overline{\text{WR}}$ or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.
3 When configured as RD_$\overline{\text{WR}}$, set the ODE bit to zero.
4 See the EBI section for weak pull up settings when configured as RD_$\overline{\text{WR}}$.

Figure 6-17. RD_$\overline{\text{WR}}$_GPIO[62] Pad Configuration Register (SIU_PCR62)

See Table 6-18 for bit field definitions.

6.4.1.12.5 Pad Configuration Register 63 (SIU_PCR63)

The SIU PCR63 register controls the pin function, direction, and static electrical attributes of the $\overline{\text{BDIP}}$_GPIO[63] pin.

208 Package: $\overline{\text{BDIP}}$_GPIO[63] pin is not available in the 208 package due to pin limitations.

Address: SIU_BASE+0x00BE Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	DSC		ODE[3]	HYS	0	0	WPE[4]	WPS[4]
W																
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

1 When configured as $\overline{\text{BDIP}}$, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.
2 When configured as $\overline{\text{BDIP}}$ or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.
3 When configured as $\overline{\text{BDIP}}$, set the ODE bit to zero.
4 See the EBI section for weak pull up settings when configured as $\overline{\text{BDIP}}$.

Figure 6-18. $\overline{\text{BDIP}}$_GPIO[63] Pad Configuration Register (SIU_PCR63)

See Table 6-18 for bit field definitions.

6.4.1.12.6 Pad Configuration Registers 64–65 (SIU_PCR64–SIU_PCR65)

The SIU_PCR64–SIU_PCR65 registers control the pin function, direction, and static electrical attributes of the $\overline{\text{WE}}$/$\overline{\text{BE}}$[0:1]_GPIO[64:65] pins. The PA bit in the PCR64–65 registers selects between the write enable/byte enable and GPIO functions. The WEBS bit in the EBI base registers selects between the write enable and byte enable function.

208 Package: $\overline{\text{WE}}$/$\overline{\text{BE}}$[0:1]_GPIO[64:65] pins are not available in the 208 package due to pin limitations.

Address: SIU_BASE + (0x00C0–0x00C6) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	DSC		ODE[3]	HYS	0	0	WPE[4]	WPS[4]
W																
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

1 When configured as $\overline{\text{WE}}$[0:1] or $\overline{\text{BE}}$[0:1], the OBE bit has no effect. When configured as GPO, set the OBE bit to one.
2 When configured as $\overline{\text{WE}}$[0:1] or $\overline{\text{BE}}$[0:1] or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.
3 When configured as $\overline{\text{WE}}$[0:1] or $\overline{\text{BE}}$[0:1], set the ODE bit to zero.
4 See the EBI section for weak pull up settings when configured as $\overline{\text{WE}}$[0:1] or $\overline{\text{BE}}$[0:1].

Figure 6-19. $\overline{\text{WE}}$/$\overline{\text{BE}}$[0:1]_GPIO[64:65] Pad Configuration Registers (SIU_PCR64–SIU_PCR65)

See Table 6-18 for bit field definitions.

6.4.1.12.7 Pad Configuration Register 68 (SIU_PCR68)

The SIU_PCR68 register controls the pin function, direction, and static electrical attributes of the $\overline{OE}$_GPIO[68] pin.

Address: SIU_BASE + 0x00C8 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	DSC		ODE[3]	HYS	0	0	WPE[4]	WPS[4]
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

1 When configured as $\overline{OE}$, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as $\overline{OE}$ or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

3 When configured as $\overline{OE}$, set the ODE bit to zero.

4 See the EBI section for weak pull up settings when configured as $\overline{OE}$.

Figure 6-20. $\overline{OE}$_GPIO[68] Pad Configuration Register (SIU_PCR68)

See Table 6-18 for bit field definitions.

6.4.1.12.8 Pad Configuration Register 69 (SIU_PCR69)

The SIU_PCR69 register controls the pin function, direction, and static electrical attributes of the $\overline{TS}$_GPIO[69] pin.

208 Package: $\overline{TS}$_GPIO[69] pin is not available in the 208 package due to pin limitations.

Address: SIU_BASE + 0x00CA Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	DSC		ODE[3]	HYS	0	0	WPE[4]	WPS[4]
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

1 When configured as $\overline{TS}$, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as $\overline{TS}$ or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

3 When configured as $\overline{TS}$, set the ODE bit to zero.

4 See the EBI section for weak pull up settings when configured as $\overline{TS}$.

Figure 6-21. $\overline{TS}$_GPIO[69] Pad Configuration Register (SIU_PCR69)

See Table 6-18 for bit field definitions.

6.4.1.12.9 Pad Configuration Register 70 (SIU_PCR70)

The SIU_PCR70 register controls the pin function, direction, and static electrical attributes of the $\overline{\text{TA}}$_GPIO[70] pin.

208 Package: $\overline{\text{TA}}$_GPIO[70] pin is not available in the 208 package due to pin limitations.

Address: SIU_BASE + 0x00CC Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	DSC		ODE[3]	HYS	0	0	WPE[4]	WPS[4]
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

1 When configured as $\overline{\text{TA}}$, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.
2 When configured as $\overline{\text{TA}}$, or GPIO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.
3 When configured as $\overline{\text{TA}}$ and external master operation is enabled, set the ODE bit to zero.
4 See the EBI section for weak pull up settings when configured as $\overline{\text{TA}}$.

Figure 6-22. $\overline{\text{TA}}$_GPIO[70] Pad Configuration Register (SIU_PCR70)

See Table 6-18 for bit field definitions.

6.4.1.12.10 Pad Configuration Register 82–75 (SIU_PCR82–SIU_PCR75)

The SIU_PCR82–SIU_PCR75 registers control the pin function, direction, and static electrical attributes of the MDO[11:4]_GPIO[82:75] pins. GPIO is the default function at reset for these pins. The full port mode (FPM) bit in the Nexus port controller (NPC) port configuration register controls whether the pins function as MDO[11:4]_GPIO[82:75]. The pad interface port enable for these pins is driven by the NPC block. When the FPM bit is set, the NPC enables the MDO port enable, and disables GPIO. When the FPM bit is cleared, the NPC disables the MDO port enable, and enables GPIO.

208 Package: MDO[11:4]_GPIO[82:75] pins are not available in the 208 package.

Address: SIU_BASE + (0x00E4–0x00D6) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	OBE[1]	IBE[1]	DSC		ODE[2]	HYS[3]	0	0	WPE[4]	WPS
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

1 This bit applies only to GPIO operation.
2 Set the ODE bit to zero for MDO operation.
3 The HYS bit has no effect on MDO operation.
4 Set the WPE bit to zero for MDO operation.

Figure 6-23. MDO[11:4]_GPIO[82:75] Pad Configuration Register (SIU_PCR82–SIU_PCR75)

See Table 6-18 for bit field definitions.

6.4.1.12.11 Pad Configuration Register 83 (SIU_PCR83)

The SIU_PCR83 register controls the pin function, direction, and static electrical attributes of the CNTXA_GPIO[83] pin.

Address: SIU_BASE + 0x00E6 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as CNTXA, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as CNTXA or GPO, set the IBE bit to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-24. CNTXA_GPIO[83] Pad Configuration Register (SIU_PCR83)

See Table 6-18 for bit field definitions.

6.4.1.12.12 Pad Configuration Register 84 (SIU_PCR84)

The SIU_PCR84 register controls the pin function, direction, and static electrical attributes of the CNRXA_GPIO[84] pin.

Address: SIU_BASE + 0x00E8 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as CNRXA, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as CNRXA or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-25. CNRXA_GPIO[84] Pad Configuration Register (SIU_PCR84)

See Table 6-18 for bit field definitions.

6.4.1.12.13 Pad Configuration Register 85 (SIU_PCR85)

The SIU_PCR85 register controls the pin function, direction, and static electrical attributes of the CNTXB_PCSC[3]_GPIO[85] pin. CNTXB is the primary function and is not available in this device. This register allows you to select the PCSC[3] or GPIO[85] function.

Address: SIU_BASE + 0x00EA Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b01 or 0b11 for the PA field. Valid values are 0b00 for GPIO[85] and 0b10 for PCSC[3].

2 When configured as PCSC, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as PCSC or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-26. PCSC[3]_GPIO[85] Pad Configuration Register (SIU_PCR85)

See Table 6-18 for bit field definitions.

6.4.1.12.14 Pad Configuration Register 86 (SIU_PCR86)

The SIU_PCR86 register controls the pin function, direction, and static electrical attributes of the CNRXB_PCSC[4]_GPIO[86] pin. CNRXB is the primary function and is not available in this device. This register allows you to select the PCSC[4] or GPIO[86] function.

Address: SIU_BASE + 0x00EC Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b01 or 0b11 for the PA field. Valid values are 0b00 for GPIO[86] and 0b10 for PCSC[4].

2 When configured as PCSC, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as PCSC or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-27. PCSC[4]_GPIO[86] Pad Configuration Register (SIU_PCR86)

See Table 6-18 for bit field definitions.

6.4.1.12.15 Pad Configuration Register 87 (SIU_PCR87)

The SIU_PCR87 register controls the pin function, direction, and static electrical attributes of the CNTXC_PCSD[3]_GPIO[87] pin.

Address: SIU_BASE + 0x00EE Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as CNTXC or PCSD, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as CNTXC or PCSD or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-28. CNTXC_PCSD[3]_GPIO[87] Pad Configuration Register (SIU_PCR87)

See Table 6-18 for bit field definitions.

6.4.1.12.16 Pad Configuration Register 88 (SIU_PCR88)

The SIU_PCR88 register controls the pin function, direction, and static electrical attributes of the CNRXC_PCSD[4]_GPIO[88] pin.

Address: SIU_BASE + 0x00F0 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as CNRXC or PCSD, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as CNRXC or PCSD or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-29. CNRXC_PCSD[4]_GPIO[88] Pad Configuration Register (SIU_PCR88)

See Table 6-18 for bit field definitions.

6.4.1.12.17 Pad Configuration Register 89 (SIU_PCR89)

The SIU_PCR89 register controls the pin function, direction, and static electrical attributes of the TXDA_GPIO[89] pin.

Address: SIU_BASE + 0x00F2 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as TXDA, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as TXDA or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. For SCI loop back operation, set the IBE bit to one. When configured as GPI, set the IBE bit to one.

Figure 6-30. TXDA_GPIO[89] Pad Configuration Register (SIU_PCR89)

See Table 6-18 for bit field definitions.

6.4.1.12.18 Pad Configuration Register 90 (SIU_PCR90)

The SIU_PCR90 register controls the pin function, direction, and static electrical attributes of the RXDA_GPIO[90] pin.

Address: SIU_BASE + 0x00F4 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as RXDA, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as RXDA or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-31. RXDA_GPIO[90] Pad Configuration Register (SIU_PCR90)

See Table 6-18 for bit field definitions.

6.4.1.12.19 Pad Configuration Register 91 (SIU_PCR91)

The SIU_PCR91 register controls the pin function, direction, and static electrical attributes of the TXDB_PCSD[1]_GPIO[91] pin.

Address: SIU_BASE + 0x00F6 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1

2 When configured as TXDB or PCSD, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as TXDB or PCSD or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. For SCI loop back operation, set the IBE bit to one. When configured as GPI, set the IBE bit to one.

Figure 6-32. TXDB_PCSD[1]_GPIO[91] Pad Configuration Register (SIU_PCR91)

See Table 6-18 for bit field definitions.

6.4.1.12.20 Pad Configuration Register 92 (SIU_PCR92)

The SIU_PCR92 register controls the pin function, direction, and static electrical attributes of the RXDB_PCSD[5]_GPIO[92] pin.

Address: SIU_BASE + 0x00F8 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as RXDB or PCSD, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as RXDB or PCSD or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-33. RXDB_PCSD[5]_GPIO[92] Pad Configuration Register (SIU_PCR92)

See Table 6-18 for bit field definitions.

6.4.1.12.21 Pad Configuration Register 93 (SIU_PCR93)

The SIU_PCR93 register controls the pin function, direction, and static electrical attributes of the SCKA_PCSC[1]_GPIO[93] pin. The SCKA signal is the primary function and is not available in this device. This register allows you to select the PCSC[1] or GPIO[93] function.

Address: SIU_BASE + 0x00FA Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b01 or 0b11 for the PA field. Valid values are 0b10 for PCSC[1] and 0b00 for GPIO[93].

2 When configured as PCSC, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as PCSC or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-34. PCSC[1]_GPIO[93] Pad Configuration Register (SIU_PCR93)

See Table 6-18 for bit field definitions.

6.4.1.12.22 Pad Configuration Register 94 (SIU_PCR94)

The SIU_PCR94 register controls the pin function, direction, and static electrical attributes of the SINA_PCSC[2]_GPIO[94] pin. SINA is the primary function is not available in this device. This register allows you to select of the PCSC[2] or GPIO[94] function.

Address: SIU_BASE + 0x00FC Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b01 or 0b11 for the PA field. Valid values are 0b10 for PCSC[2] and 0b00 for GPIO[94].

2 When configured as PCSC, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as PCSC or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-35. PCSC[2]_GPIO[94] Pad Configuration Register (SIU_PCR94)

See Table 6-18 for bit field definitions.

6.4.1.12.23 Pad Configuration Register 95 (SIU_PCR95)

The SIU_PCR95 register controls the pin function, direction, and static electrical attributes of the SOUTA_PCSC[5]_GPIO[95] pin. SOUTA is the primary function and is not available in the device. This register allows you to select the PCSC[5] or GPIO[95] function.

Address: SIU_BASE + 0x00FE Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b01 or 0b11 for the PA field. Valid values are 0b10 for PCSC[5] and 0b00 for GPIO[95].

2 When configured as PCSC, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as PCSC or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-36. PCSC[5]_GPIO[95] Pad Configuration Register (SIU_PCR95)

See Table 6-18 for bit field definitions.

6.4.1.12.24 Pad Configuration Registers 96 (SIU_PCR96)

The SIU_PCR96 registers control the pin function, direction, and static electrical attributes of the PCSA[0]_PCSD[2]_GPIO[96] pin. PCSA[0] is the primary function and is not available in this device. This register allows you to select the PCSD[2] or GPIO[96] function.

Address: SIU_BASE + 0x0100 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b01 or 0b11 for the PA field. Valid values are 0b10 for PCSD[2] and 0b00 for GPIO[96].

2 When configured as PCSD[2], the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as PCSD[2] or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-37. PCSD[2]_GPIO[96] Pad Configuration Register (SIU_PCR96)

See Table 6-18 for bit field definitions.

6.4.1.12.25 Pad Configuration Registers 97 (SIU_PCR97)

The SIU_PCR97 registers control the pin function, direction, and static electrical attributes of the PCSA[1]_PCSB[2]_GPIO[97] pin. PCSA[1] is the primary function and is not available in this device. This register allows you to select the PCSB[2] or GPIO[97] function.

Address: SIU_BASE + 0x0102 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b01 or 0b11 for the PA field. Valid values are 0b10 for PCSB[2] and 0b00 for GPIO[97].

2 When configured as PCSB[2], the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as PCSB[2] or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-38. PCSB[2]_GPIO[97] Pad Configuration Register (SIU_PCR97)

See Table 6-18 for bit field definitions.

6.4.1.12.26 Pad Configuration Register 98 (SIU_PCR98)

The SIU_PCR98 register controls the pin function, direction, and static electrical attributes of the PCSA[2]_SCKD_GPIO[98] pin. PCSA[2] is the primary function and is not available in this device. This register allows you to select the SCKD or GPIO[98] function.

Address: SIU_BASE + 0x0104 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b001 or 0b011 for the PA field. Valid values are 0b10 for SCKD and 0b00 for GPIO[98].

2 When configured as SCKD, set the OBE bit to one for master operation, or set to zero for slave operation. When configured as GPO, set the OBE bit to one.

3 When configured as SCKD in slave operation, set the IBE bit to one. When configured as SCKD in master operation or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-39. SCKD_GPIO[98] Pad Configuration Register (SIU_PCR98)

See Table 6-18 for bit field definitions.

6.4.1.12.27 Pad Configuration Register 99 (SIU_PCR99)

The SIU_PCR99 register controls the pin function, direction, and static electrical attributes of the PCSA[3]_SIND_GPIO[99] pin. PCSA[3] is the primary function and is not available in this device. This register allows you to select the SIND or GPIO[99] function.

Address: SIU_BASE + 0x0106 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b01 or 0b11 for the PA field. Valid values are 0b10 for SIND and 0b00 for GPIO[99].

2 When configured as GPO, set the OBE bit to one.

3 When configured as SIND or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-40. SIND_GPIO[99] Pad Configuration Register (SIU_PCR99)

See Table 6-18 for bit field definitions.

6.4.1.12.28 Pad Configuration Register 100 (SIU_PCR100)

The SIU_PCR100 register controls the pin function, direction, and static electrical attributes of the PCSA[4]_SOUTD_GPIO[100] pin. PCSA[4] is the primary function is not available in this device. This register allows you to select the SOUTD or GPIO[100] function.

Address: SIU_BASE + 0x0108 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b01 or 0b11 for the PA field. Valid values are 0b10 for SOUTD and 0b00 for GPIO[100].

2 When configured as SOUTD, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as SOUTD or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-41. SOUTD_GPIO[100] Pad Configuration Register (SIU_PCR100)

See Table 6-18 for bit field definitions.

6.4.1.12.29 Pad Configuration Registers 101 (SIU_PCR101)

The SIU_PCR101 register controls the pin function, direction, and static electrical attributes of the PCSA[5]_PCSB[3]_GPIO[101] pin. PCSA[5] is the primary function is not available in this device. PCSB[3] is not available in this device. This register allows you to select the PCSB[3] or GPIO[101] function.

Address: SIU_BASE + 0x010A Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 The primary function is not available on this device. Do not select 0b01 or 0b11 for the PA field. Valid values are 0b10 for PCSB[3] and 0b00 for GPIO[101].

2 When configured as PCSB[3], the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as PCSB[3] or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-42. PCSB[3]_GPIO[101] Pad Configuration Register (SIU_PCR101)

See Table 6-18 for bit field definitions.

6.4.1.12.30 Pad Configuration Register 102 (SIU_PCR102)

The SIU_PCR102 register controls the pin function, direction, and static electrical attributes of the SCKB_PCSC[1]_GPIO[102] pin.

Address: SIU_BASE + 0x010C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as SCKB, set the OBE bit to one for master operation, and set to zero for slave operation. When configured as GPO, set the OBE bit to one.

2 When configured as SCKB in slave operation, set the IBE bit to one. When configured as SCKB in master operation or PCSC or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-43. SCKB_PCSC[1]_GPIO[102] Pad Configuration Register (SIU_PCR102)

See Table 6-18 for bit field definitions.

6.4.1.12.31 Pad Configuration Register 103 (SIU_PCR103)

The SIU_PCR103 register controls the pin function, direction, and static electrical attributes of the SINB_PCSC[2]_GPIO[103] pin.

Address: SIU_BASE + 0x010E Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as SINB, set the OBE bit to zero. When configured as PCSC, set the OBE bit to one.

2 When configured as SINB or PCSC, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-44. SINB_PCSC[2]_GPIO[103] Pad Configuration Register (SIU_PCR103)

See Table 6-18 for bit field definitions.

6.4.1.12.32 Pad Configuration Register 104 (SIU_PCR104)

The SIU_PCR104 register controls the pin function, direction, and static electrical attributes of the SOUTB_PCSC[5]_GPIO[104] pin.

Address: SIU_BASE + 0x0110 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as SOUTB or PCSC, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as SOUTB or PCSC or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-45. SOUTB_PCSC[5]_GPIO[104] Pad Configuration Register (SIU_PCR104)

See Table 6-18 for bit field definitions.

6.4.1.12.33 Pad Configuration Register 105 (SIU_PCR105)

The SIU_PCR105 register controls the pin function, direction, and static electrical attributes of the PCSB[0]_PCSD[2]_GPIO[105] pin.

Address: SIU_BASE + 0x0112 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as PCSB[0], the OBE bit has no effect. When configured as PCSD[2], set the OBE bit to one for master operation, and set to zero for slave operation. When configured as GPO, set the OBE bit to one.

2 When configured as PCS or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-46. PCSB[0]_PCSD[2]_GPIO[105] Pad Configuration Register (SIU_PCR105)

See Table 6-18 for bit field definitions.

6.4.1.12.34 Pad Configuration Register 106 (SIU_PCR106)

The SIU_PCR106 register controls the pin function, direction, and static electrical attributes of the PCSB[1]_PCSD[0]_GPIO[106] pin.

Address: SIU_BASE + 0x0114 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as PCSB[1], the OBE bit has no effect. When configured as PCSD[0], set the OBE bit to one for master operation, and set to zero for slave operation. When configured as GPO, set the OBE bit to one.

2 When configured as PCSD[0] in slave operation, set the IBE bit to one. When configured as PCS in master operation or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-47. PCSB[1]_PCSD[0]_GPIO[106] Pad Configuration Register (SIU_PCR106)

See Table 6-18 for bit field definitions.

6.4.1.12.35 Pad Configuration Register 107 (SIU_PCR107)

The SIU_PCR107 register controls the pin function, direction, and static electrical attributes of the PCSB[2]_SOUTC_GPIO[107] pin.

Address: SIU_BASE + 0x0116 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as PCSB or SOUTC, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as PCSB or SOUTC or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-48. PCSB[2]_SOUTC_GPIO[107] Pad Configuration Register (SIU_PCR107)

See Table 6-18 for bit field definitions.

6.4.1.12.36 Pad Configuration Register 108 (SIU_PCR108)

The SIU_PCR108 register controls the pin function, direction, and static electrical attributes of the PCSB[3]_SINC_GPIO[108] pin.

Address: SIU_BASE + 0x0118 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as PCSB or SINC, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as PCSB or SINC or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-49. PCSB[3]_SINC_GPIO[108] Pad Configuration Register (SIU_PCR108)

See Table 6-18 for bit field definitions.

6.4.1.12.37 Pad Configuration Register 109 (SIU_PCR109)

The SIU_PCR109 register controls the pin function, direction, and static electrical attributes of the PCSB[4]_SCKC_GPIO[109] pin.

Address: SIU_BASE + 0x011A Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as PCSB, the OBE bit has no effect. When configured as SCKC, set the OBE bit to one for master operation, and set to zero for slave operation. When configured as GPO, set the OBE bit to one.

2 When configured as SCKC in slave operation, set the IBE bit to one. When configured as PCSB or SCKC in master operation or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-50. PCSB[4]_SCKC_GPIO[109] Pad Configuration Register (SIU_PCR109)

See Table 6-18 for bit field definitions.

6.4.1.12.38 Pad Configuration Register 110 (SIU_PCR110)

The SIU_PCR110 register controls the pin function, direction, and static electrical attributes of the PCSB[5]_PCSC[0]_GPIO[110] pin.

Address: SIU_BASE + 0x011C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as PCSB[5], the OBE bit has no effect. When configured as PCSC[0], set the OBE bit to one for master operation, and set to zero for slave operation. When configured as GPO, set the OBE bit to one.

2 When configured as PCSC[0] in slave operation, set the IBE bit to one. When configured as PCSB[5] or PCSC in master operation or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-51. PCSB[5]_PCSC[0]_GPIO[110] Pad Configuration Register (SIU_PCR110)

See Table 6-18 for bit field definitions.

6.4.1.12.39 Pad Configuration Register 113 (SIU_PCR113)

The SIU_PCR113 register controls the pin function, direction, and static electrical attributes of the TCRCLKA_$\overline{\text{IRQ}}$[7]_GPIO[113] pin.

Address: SIU_BASE + 0x0122 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as TCRCLKA or IRQ, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as TCRCLKA or IRQ or GPO, set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-52. TCRCLKA_$\overline{\text{IRQ}}$[7]_GPIO[113] Pad Configuration Register (SIU_PCR113)

See Table 6-18 for bit field definitions.

6.4.1.12.40 Pad Configuration Register 114–125 (SIU_PCR114–SIU_PCR125)

The SIU_PCR114–SIU_PCR125 registers control the pin function, direction, and static electrical attributes of the ETPUA[0:11]_ETPUA[12:23]_GPIO[114:125] pins. Only the output channels of ETPUA[12:23] are connected to pins. Both the input and output channels of ETPUA[0:11] are connected to pins.

Address: SIU_BASE + (0x0124–0x013A) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 The OBE bit must be set to one for both ETPUA[0:11] and GPIO[114:125] when configured as outputs. When configured as ETPUA[12:23], the OBE bit has no effect.

2 The IBE bit must be set to one for both ETPUA[0:11] and GPIO[114:125] when configured as inputs. When configured as ETPUA[12:23] or when ETPUA[0:11] or GPIO[114:125] are configured as outputs, you can set the IBE bit to one to reflect the pin state in the GPDI register.

3 The weak pull up/down selection at reset for the ETPUA[0:11] pins is determined by the WKPCFG pin.

Figure 6-53. ETPUA[0:11]_ETPUA[12:23]_GPIO[114:125] Pad Configuration Register (SIU_PCR114–SIU_PCR125)

See Table 6-18 for bit field definitions.

6.4.1.12.41 Pad Configuration Register 126 (SIU_PCR126)

The SIU_PCR126 register controls the pin function, direction, and static electrical attributes of the ETPUA[12]_PCSB[1]_GPIO[126] pin.

Address: SIU_BASE + 0x013C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as PCSB, the OBE bit has no effect. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.

2 The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCSB, or ETPUA or GPO outputs, set the IBE bit to one to reflect the pin state in the GPDI register.

3 The weak pull up/down selection at reset for the ETPUA[12] pin is determined by the WKPCFG pin.

Figure 6-54. ETPUA[12]_PCSB[1]_GPIO[126] Pad Configuration Register (SIU_PCR126)

See Table 6-18 for bit field definitions.

6.4.1.12.42 Pad Configuration Register 127 (SIU_PCR127)

The SIU_PCR127 register controls the pin function, direction, and static electrical attributes of the ETPUA[13]_PCSB[3]_GPIO[127] pin.

Address: SIU_BASE + 0x013E Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as PCSB, the OBE bit has no effect. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.

2 The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCSB, or ETPUA or GPO outputs, set the IBE bit to one to reflect the pin state in the GPDI register.

3 The weak pull up/down selection at reset for the ETPUA[13] pin is determined by the WKPCFG pin.

Figure 6-55. ETPUA[13]_PCSB[3]_GPIO[127] Pad Configuration Register (SIU_PCR127)

See Table 6-18 for bit field definitions.

6.4.1.12.43 Pad Configuration Register 128 (SIU_PCR128)

The SIU_PCR128 register controls the pin function, direction, and static electrical attributes of the ETPUA[14]_PCSB[4]_GPIO[128] pin.

Address: SIU_BASE + 0x0140 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as PCSB, the OBE bit has no effect. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.

2 The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCSB, or ETPUA or GPO outputs, set the IBE bit to one to reflect the pin state in the GPDI register.

3 The weak pull up/down selection at reset for the ETPUA[14] pin is determined by the WKPCFG pin.

Figure 6-56. ETPUA[14]_PCSB[4]_GPIO[128] Pad Configuration Register (SIU_PCR128)

See Table 6-18 for bit field definitions.

6.4.1.12.44 Pad Configuration Register 129 (SIU_PCR129)

The SIU_PCR129 register controls the pin function, direction, and static electrical attributes of the ETPUA[15]_PCSB[5]_GPIO[129] pin.

Address: SIU_BASE + 0x0142 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as PCSB, the OBE bit has no effect. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.

2 The IBE bit must be set to one for ETPUA or GPIO when configured as inputs. When configured as PCSB, or ETPUA or GPO outputs, set the IBE bit to one to reflect the pin state in the GPDI register.

3 The weak pull up/down selection at reset for the ETPUA[15] pin is determined by the WKPCFG pin.

Figure 6-57. ETPUA[15]_PCSB[5]_GPIO[129] Pad Configuration Register (SIU_PCR129)

See Table 6-18 for bit field definitions.

6.4.1.12.45 Pad Configuration Register 130 (SIU_PCR130)

The SIU_PCR130 register controls the pin function, direction, and static electrical attributes of the ETPUA[16]_PCSD[1]_GPIO[130] pin.

Address: SIU_BASE + 0x0144 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as PCSD, the OBE bit has no effect. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.

2 The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCSD, or ETPUA or GPO outputs, set the IBE bit to one to reflect the pin state in the GPDI register.

3 The weak pull up/down selection at reset for the ETPUA[16] pin is determined by the WKPCFG pin.

Figure 6-58. ETPUA[16]_PCSD[1]_GPIO[130] Pad Configuration Register (SIU_PCR130)

See Table 6-18 for bit field definitions.

6.4.1.12.46 Pad Configuration Register 131 (SIU_PCR131)

The SIU_PCR131 register controls the pin function, direction, and static electrical attributes of the ETPUA[17]_PCSD[2]_GPIO[131] pin.

Address: SIU_BASE + 0x0146 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as PCSD, the OBE bit has no effect. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.

2 The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCSD, or ETPUA or GPO outputs, set the IBE bit to one to reflect the pin state in the GPDI register.

3 The weak pull up/down selection at reset for the ETPUA[17] pin is determined by the WKPCFG pin.

Figure 6-59. ETPUA[17]_PCSD[2]_GPIO[131] Pad Configuration Register (SIU_PCR131)

See Table 6-18 for bit field definitions.

6.4.1.12.47 Pad Configuration Register 132 (SIU_PCR132)

The SIU_PCR132 register controls the pin function, direction, and static electrical attributes of the ETPUA[18]_PCSD[3]_GPIO[132] pin.

Address: SIU_BASE + 0x0148 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as PCSD, the OBE bit has no effect. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.

2 The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCSD, or ETPUA or GPO outputs, you can set the IBE bit to one to reflect the pin state in the GPDI register.

3 The weak pull up/down selection at reset for the ETPUA[18] pin is determined by the WKPCFG pin.

Figure 6-60. ETPUA[18]_PCSD[3]_GPIO[132] Pad Configuration Register (SIU_PCR132)

See Table 6-18 for bit field definitions.

6.4.1.12.48 Pad Configuration Register 133 (SIU_PCR133)

The SIU_PCR133 register controls the pin function, direction, and static electrical attributes of the ETPUA[19]_PCSD[4]_GPIO[133] pin.

Address: SIU_BASE + 0x014A Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as PCSD, the OBE bit has no effect. The OBE bit must be set to one for both ETPUA and GPIO when configured as outputs.

2 The IBE bit must be set to one for both ETPUA and GPIO when configured as inputs. When configured as PCSD, or ETPUA or GPO outputs, you can set the IBE bit to one to reflect the pin state in the GPDI register.

3 The weak pull up/down selection at reset for the ETPUA[19] pin is determined by the WKPCFG pin.

Figure 6-61. ETPUA[19]_PCSD[4]_GPIO[133] Pad Configuration Register (SIU_PCR133)

See Table 6-18 for bit field definitions.

6.4.1.12.49 Pad Configuration Register 134–141 (SIU_PCR134–SIU_PCR141)

The SIU_PCR134–SIU_PCR141 registers control the pin function, direction, and static electrical attributes of the ETPUA[20:27]_IRQ[8:15]_GPIO[134:141] pins. Only the output channels of ETPUA[24:27] are connected to pins. Both the input and output channels of ETPUA[20:23] are connected to pins.

Address: SIU_BASE + (0x014C–0x015A) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as ETPUA[24:27] or IRQ, the OBE bit has no effect. The OBE bit must be set to one for both ETPUA[20:23] and GPIO[134:141] when configured as outputs.

2 When configured as ETPUA[24:27] or IRQ or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for ETPUA[20:23] or GPIO[134:141] when configured as inputs.

3 The weak pull up/down selection at reset for the ETPUA[20:27] pins is determined by the WKPCFG pin.

Figure 6-62. ETPUA[20:27]_IRQ[8:15]_GPIO[134:141] Pad Configuration Register (SIU_PCR134–SIU_PCR141)

See Table 6-18 for bit field definitions.

6.4.1.12.50 Pad Configuration Register 142 (SIU_PCR142)

The SIU_PCR142 register controls the pin function, direction, and static electrical attributes of the ETPUA[28]_PCSC[1]_GPIO[142] pin. Only the output channel of ETPUA[28] is connected to the pin.

Address: SIU_BASE + 0x015C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as ETPUA or PCSC, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as ETPUA, PCSC, or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for GPIO when configured as input.

3 The weak pull up/down selection at reset for the ETPUA[28] pin is determined by the WKPCFG pin

Figure 6-63. ETPUA[28]_PCSC[1]_GPIO[142] Pad Configuration Register (SIU_PCR142)

See Table 6-18 for bit field definitions.

6.4.1.12.51 Pad Configuration Register 143 (SIU_PCR143)

The SIU_PCR143 register controls the pin function, direction, and static electrical attributes of the ETPUA[29]_PCSC[2]_GPIO[143] pin. For ETPUA[29], only the output channel is connected to the pin.

Address: SIU_BASE + 0x015E Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as ETPUA or PCSC, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

2 When configured as ETPUA, PCSC, or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for GPIO when configured as input.

3 The weak pull up/down selection at reset for the ETPUA[29] pin is determined by the WKPCFG pin

Figure 6-64. ETPUA[29]_PCSC[2]_GPIO[143] Pad Configuration Register (SIU_PCR143)

See Table 6-18 for bit field definitions.

6.4.1.12.52 Pad Configuration Register 144 (SIU_PCR144)

The SIU_PCR144 register controls the pin function, direction, and static electrical attributes of the ETPUA[30]_PCSC[3]_GPIO[144] pin.

Address: SIU_BASE + 0x0160 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as PCSC, the OBE bit has no effect. When configured as ETPUA output or GPO, set the OBE bit to one.

2 When configured as ETPUA output, PCSC, or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for ETPUA or GPIO when configured as input.

3 The weak pull up/down selection at reset for the ETPUA[30] pin is determined by the WKPCFG pin

Figure 6-65. ETPUA[30]_PCSC[3]_GPIO[144] Pad Configuration Register (SIU_PCR144)

See Table 6-18 for bit field definitions.

6.4.1.12.53 Pad Configuration Register 145 (SIU_PCR145)

The SIU_PCR145 register controls the pin function, direction, and static electrical attributes of the ETPUA[31]_PCSC[4]_GPIO[145] pin.

Address: SIU_BASE + 0x0162 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 When configured as PCSC, the OBE bit has no effect. When configured as ETPUA output or GPO, set the OBE bit to one.

2 When configured as ETPUA output, PCSC, or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for ETPUA or GPIO when configured as input.

3 The weak pull up/down selection at reset for the ETPUA[31] pin is determined by the WKPCFG pin

Figure 6-66. ETPUA[31]_PCSC[4]_GPIO[145] Pad Configuration Register (SIU_PCR145)

See Table 6-18 for bit field definitions.

6.4.1.12.54 Pad Configuration Register 179–188 (SIU_PCR179–SIU_PCR188)

The SIU_PCR179–SIU_PCR188 registers control the pin function, direction, and static electrical attributes of the EMIOS[0:9]_ETPUA[0:9]_GPIO[179:188] pins. Both the input and output functions of EMIOS[0:9] are connected to pins. For ETPUA[0:9], only the output channels are connected to pins.

Address: SIU_BASE + (0x01A6–0x01B8) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 The OBE bit must be set to one for EMIOS[0:9] or GPIO[179:188] when configured as outputs.

2 When configured as EMIOS, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for EMIOS[0:9] or GPIO[179:188] when configured as inputs.

3 The weak pull up/down selection at reset for the EMIOS[0:9] pins is determined by the WKPCFG pin.

Figure 6-67. EMIOS[0:9]_ETPUA[0:9]_GPIO[179:188] Pad Configuration Register (SIU_PCR179–SIU_PCR188)

See Table 6-18 for bit field definitions.

6.4.1.12.55 Pad Configuration Register 189–190 (SIU_PCR189–SIU_PCR190)

The SIU_PCR189–SIU_PCR190 registers control the pin function, direction, and static electrical attributes of the EMIOS[10:11]_GPIO[189:190] pins. Both the input and output functions of EMIOS[10:11] are connected to pins.

Address: SIU_BASE + (0x01BA–0x01BC) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 The OBE bit must be set to one for EMIOS[10:11] or GPIO[189:190] when configured as outputs.

2 When configured as EMIOS or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for EMIOS[10:11] or GPIO[189:190] when configured as inputs.

3 The weak pull up/down selection at reset for the EMIOS[10:11] pins is determined by the WKPCFG pin.

Figure 6-68. EMIOS[10:11]_GPIO[189:190] Pad Configuration Register (SIU_PCR189–SIU_PCR190)

See Table 6-18 for bit field definitions.

6.4.1.12.56 Pad Configuration Register 191 (SIU_PCR191)

The SIU_PCR191 register controls the pin function, direction, and static electrical attributes of the EMIOS[12]_SOUTC_GPIO[191] pin. Only the output of EMIOS[12] is connected to the pin.

Address: SIU_BASE + 0x01BE Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 The OBE bit must be set to one for GPIO[191] when configured as an output.

2 When configured as EMIOS or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for GPIO[191] when configured as an input.

3 The weak pull up/down selection at reset for the EMIOS[12] pin is determined by the WKPCFG pin.

Figure 6-69. EMIOS[12]_SOUTC_GPIO[191] Pad Configuration Register (SIU_PCR191)

See Table 6-18 for bit field definitions.

6.4.1.12.57 Pad Configuration Register 192 (SIU_PCR192)

The SIU_PCR191 register controls the pin function, direction, and static electrical attributes of the EMIOS[13]_SOUTD_GPIO[192] pin. Only the output of EMIOS[13] is connected to the pin.

Address: SIU_BASE + 0x01C0 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 The OBE bit must be set to one for GPIO[192] when configured as an output.

2 When configured as EMIOS or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for GPIO[192] when configured as an input.

3 The weak pull up/down selection at reset for the EMIOS[13] pin is determined by the WKPCFG pin.

Figure 6-70. EMIOS[13]_SOUTD_GPIO[192] Pad Configuration Register (SIU_PCR192)

See Table 6-18 for bit field definitions.

6.4.1.12.58 Pad Configuration Register 193–194 (SIU_PCR193–SIU_PCR194)

The SIU_PCR193–SIU_PCR194 registers control the pin function, direction, and static electrical attributes of the EMIOS[14:15]_$\overline{\text{IRQ}}$[0:1]_GPIO[193:194] pins. Only the output functions of EMIOS[14:15] are connected to pins.

Address: SIU_BASE + (0x01C2–0x01C4) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE[1]	IBE[2]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[3]

1 The OBE bit must be set to one for GPIO[193:194] when configured as outputs.

2 When configured as EMIOS or IRQ or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for GPIO[193:194] when configured as inputs.

3 The weak pull up/down selection at reset for the EMIOS[14:15] pins is determined by the WKPCFG pin.

Figure 6-71. EMIOS[14:15]_$\overline{\text{IRQ}}$[0:1]_GPIO[193:194] Pad Configuration Register (SIU_PCR193–SIU_PCR194)

See Table 6-18 for bit field definitions.

6.4.1.12.59 Pad Configuration Register 195–202 (SIU_PCR195–SIU_PCR202)

The SIU_PCR195–SIU_PCR202 registers control the pin function, direction, and static electrical attributes of the EMIOS[16:23]_GPIO[195:202] pins. Both the input and output functions of EMIOS[16:23] are connected to pins. The alternate function is not available on this device.

Address: SIU_BASE + (0x01C6–0x01D4) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U[4]

1 The alternate function is not available on this device. Do not select 0b10. Valid values are 0b01 or 0b11 for EMIOS[16:23] and 0b00 for GPIO[195:202].

2 The OBE bit must be set to one for EMIOS[16:23] or GPIO[195:202] when configured as outputs.

3 When configured as EMIOS, you can set the IBE bit to one to reflect the pin state in the GPDI register. The IBE bit must be set to one for EMIOS[16:23] or GPIO[195:202] when configured as inputs.

4 The weak pull up/down selection at reset for the EMIOS[0:9] pins is determined by the WKPCFG pin.

Figure 6-72. EMIOS[16:23]_GPIO[195:202]
Pad Configuration Register (SIU_PCR195–SIU_PCR202)

See Table 6-18 for bit field definitions.

6.4.1.12.60 Pad Configuration Register 203–204 (SIU_PCR203–SIU_PCR204)

The SIU_PCR203–SIU_PCR204 registers control the pin function, direction, and static electrical attributes of the EMIOS[14:15]_GPIO[203:204] pins. For EMIOS[14:15], only the output functions are connected to the pins. The BGA labels for these pins are GPIO[203:204] because other pins are already labeled EMIOS[14:15].

208 Package: EMIOS[14:15]_GPIO[203:204] are not available due to pin limitations on the 208 package.

Address: SIU_BASE + (0x01D6–0x01D8) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA[1]	OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 Set the PA bit to one for EMIOS. Clear the PA bit to zero when used as GPIO.

2 When configured as EMIOS the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as EMIOS or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-73. EMIOS[14:15]_GPIO[203:204] Pad Configuration Register (SIU_PCR203–SIU_PCR204)

See Table 6-18 for bit field definitions.

6.4.1.12.61 Pad Configuration Registers 206–207 (SIU_PCR206–SIU_PCR207)

The SIU_PCR206–SIU_PCR207 registers control the pin function, direction, and static electrical attributes of the GPIO[206:207] pins. The PA bit is not implemented for these PCRs since GPIO is the only pin function.

Address: SIU_BASE + (0x01DC–0x01DE) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	OBE[1]	IBE[2]	DSC		ODE	HYS	0	0	WPE	WPS
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

1 When configured as GPO, set the OBE bit to one.

2 When configured as GPO, you can set the IBE to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-74. GPIO[206:207] Pad Configuration Registers (SIU_PCR206–SIU_PCR207)

See Table 6-18 for bit field definitions.

6.4.1.12.62 Pad Configuration Register 208 (SIU_PCR208)

The SIU_PCR208 register controls the pin function, direction, and static electrical attributes of the PLLCFG[0]_$\overline{\text{IRQ}}$[4]_GPIO[208] pin.

Address: SIU_BASE + 0x01E0 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS[4]	SRC		WPE	WPS
W																
RESET:	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	1

1 The PLLCFG function applies only during reset when the $\overline{\text{RSTCFG}}$ pin is asserted during reset. Set the PA field to 0b10 for $\overline{\text{IRQ}}$[4] and set to 0b00 for GPIO[208].

2 When configured as IRQ, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as IRQ or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

4 When configured as IRQ, set the HYS bit to one.

Figure 6-75. PLLCFG[0]_$\overline{\text{IRQ}}$[4]_GPIO[208] Pad Configuration Register (SIU_PCR208)

See Table 6-18 for bit field definitions.

6.4.1.12.63 Pad Configuration Register 209 (SIU_PCR209)

The SIU_PCR209 register controls the pin function, direction, and static electrical attributes of the PLLCFG[1]_$\overline{\text{IRQ}}$[5]_SOUTD_GPIO[209] pins.

Address: SIU_BASE + 0x01E2 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[1]			OBE[2]	IBE[3]	0	0	ODE	HYS[4]	SRC		WPE	WPS
W																
RESET:	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	1

1 The PLLCFG function applies only during reset when the $\overline{\text{RSTCFG}}$ pin is asserted during reset. Set the PA field to 0b010 for $\overline{\text{IRQ}}$[5], 0b100 for SOUTD, or 0b000 for GPIO[209].

2 When configured as IRQ, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as IRQ or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

4 When configured as IRQ, set the HYS bit to one.

Figure 6-76. PLLCFG[1]_$\overline{\text{IRQ}}$[5]_SOUTD_GPIO[209] Pad Configuration Register (SIU_PCR209)

See Table 6-18 for bit field definitions.

6.4.1.12.64 Pad Configuration Register 210 (SIU_PCR210)

The SIU_PCR210 register controls the pin function, direction, and static electrical attributes of the $\overline{\text{RSTCFG}}$_GPIO[210] pin.

208 Package: $\overline{\text{RSTCFG}}$_GPIO[210] is not available due to pin limitations on the 208 package.

Address: SIU_BASE + 0x01E4 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA[1]	OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1

1 $\overline{\text{RSTCFG}}$ function is only applicable during reset. Set the PA bit to zero for GPIO operation

2 When configured as GPO, set the OBE bit to one.

3 When configured as GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-77. $\overline{\text{RSTCFG}}$_GPIO[210] Pad Configuration Register (SIU_PCR210)

See Table 6-18 for bit field definitions.

6.4.1.12.65 Pad Configuration Register 211–212 (SIU_PCR211–SIU_PCR212)

The SIU_PCR211–SIU_PCR212 registers control the pin function, direction, and static electrical attributes of the BOOTCFG[0:1]_$\overline{\text{IRQ}}$[2:3]_GPIO[211:212] pins.

208 Package: BOOTCFG[0]_$\overline{\text{IRQ}}$[2]_GPIO[211] is not available due to pin limitations on the 208 package.

Address: SIU_BASE + (0x01E6–0x01E8) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		OBE[2]	IBE[3]	0	0	ODE	HYS[4]	SRC		WPE	WPS
W																
RESET:	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0

1 The BOOTCFG function applies only during reset when the $\overline{\text{RSTCFG}}$ pin is asserted during reset. Set the PA field to 0b10 for $\overline{\text{IRQ}}$[2:3] and set to 0b00 for GPIO[211:212].

2 When configured as IRQ, the OBE bit has no effect. When configured as GPO, set the OBE bit to one.

3 When configured as IRQ or GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

4 When configured as IRQ, set the HYS bit to one.

Figure 6-78. BOOTCFG[0:1]_$\overline{\text{IRQ}}$[2:3]_GPIO[211:212] Pad Configuration Register (SIU_PCR211–SIU_PCR212)

See Table 6-18 for bit field definitions.

6.4.1.12.66 Pad Configuration Register 213 (SIU_PCR213)

The SIU_PCR213 register controls the pin function, direction, and static electrical attributes of the WKPCFG_GPIO[213] pin.

Address: SIU_BASE + 0x01EA Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA[1]	OBE[2]	IBE[3]	0	0	ODE	HYS	SRC		WPE	WPS
W																
RESET:	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1

1 WKPCFG function is only applicable during reset. The PA bit must be set to zero for GPIO operation

2 When configured as GPO, set the OBE bit to one.

3 When configured as GPO, you can set the IBE bit to one to reflect the pin state in the GPDI register. When configured as GPI, set the IBE bit to one.

Figure 6-79. WKPCFG_GPIO[213] Pad Configuration Register (SIU_PCR213)

See Table 6-18 for bit field definitions.

6.4.1.12.67 Pad Configuration Register 214 (SIU_PCR214)

The SIU_PCR214 register controls the enabling/disabling and drive strength of the ENGCLK pin. The ENGCLK pin is enabled and disabled by setting and clearing the OBE bit. The ENGCLK pin is enabled during reset.

Address: SIU_BASE + 0x01EC Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	OBE	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0

Figure 6-80. ENGLCK Pad Configuration Register (SIU_PCR214)

See Table 6-18 for bit field definitions.

6.4.1.12.68 Pad Configuration Register 215 (SIU_PCR215)

The SIU_PCR215 register controls the pin function, direction, and static electrical attributes of the AN[12]_MA[0]_$\overline{SDS}$ pin.

Address: SIU_BASE + 0x01EE Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		0	0	0	0	ODE	0	SRC		0	0
W																
RESET:	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

[1] The input and output buffers are enabled/disabled based on the PA selection. Both the input and output buffers are disabled for the AN[12] function. The output buffer only is enabled for the MA[0] and $\overline{SDS}$ functions.

Figure 6-81. AN[12]_MA[0]_$\overline{SDS}$ Pad Configuration Register (SIU_PCR215)

See Table 6-18 for bit field definitions. The PA field for PCR215 is given in Table 6-20.

Table 6-20. PCR215 PA Field Definition

PA Field	Pin Function
0b00	$\overline{SDS}$
0b01	Reserved
0b10	MA[0]
0b11	AN[12]

6.4.1.12.69 Pad Configuration Register 216 (SIU_PCR216)

The SIU_PCR216 register controls the pin function, direction, and static electrical attributes of the AN[13]_MA[1]_SDO pin.

Address: SIU_BASE + 0x01F0 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		0	0	0	0	ODE	0	SRC		0	0
W																
RESET:	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

1 The input and output buffers are enabled/disabled based on the PA selection. Both the input and output buffers are disabled for the AN[13] function. The output buffer only is enabled for the MA[1] and SDO functions.

Figure 6-82. AN[13]_MA[1]_SDO Pad Configuration Register (SIU_PCR216)

See Table 6-18 for bit field definitions. The PA field for PCR216 is given in Table 6-21.

Table 6-21. PCR216 PA Field Definition

PA Field	Pin Function
0b00	SDO
0b01	Reserved
0b10	MA[1]
0b11	AN[13]

6.4.1.12.70 Pad Configuration Register 217 (SIU_PCR217)

The SIU_PCR217 register controls the pin function, direction, and static electrical attributes of the AN[14]_MA[2]_SDI pin.

Address: SIU_BASE + 0x01F2 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[1]		0	0	0	0	ODE	HYS	SRC		WPE[2]	WPS[3]
W																
RESET:	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

1 The input and output buffers are enabled/disabled based on the PA selection. Both input and output buffers are disabled for the AN[14] function. The output buffer only is enabled for the MA[2] function; the input buffer only is enabled for the SDI function.

2 Set the WPE bit to zero when configured as an analog input or MA[2], and set the WPE bit to one when configured as SDI.

3 Set the WPS bit to one when configured as SDI.

Figure 6-83. AN[14]_MA[2]_SDI Pad Configuration Register (SIU_PCR217)

See Table 6-18 for bit field definitions. The PA field for PCR217 is given in Table 6-22.

Table 6-22. PCR217 PA Field Definition

PA Field	Pin Function
0b00	SDI
0b01	Reserved
0b10	MA[2]
0b11	AN[14]

6.4.1.12.71 Pad Configuration Register 218 (SIU_PCR218)

The SIU_PCR218 register controls the pin function, direction, and static electrical attributes of the AN[15]_FCK pin.

Address: SIU_BASE + 0x01F4 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA[1]	0	0	0	0	ODE	0	SRC		0	0
W																
RESET:	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

[1] The input and output buffers are enabled/disabled based on the PA selection. Both the input and output buffers are disabled for the AN[15] function. The output buffer only is enabled for the FCK function.

Figure 6-84. AN[15]_FCK Pad Configuration Register (SIU_PCR218)

See Table 6-18 for bit field definitions. The PA field for PCR218 is given in Table 6-23.

Table 6-23. PCR218 PA Field Definition

PA Field	Pin Function
0b0	FCK
0b1	AN[15]

6.4.1.12.72 Pad Configuration Register 219 (SIU_PCR219)

The SIU_PCR219 register controls the drive strength of the MCKO pin.

Address: SIU_BASE + 0x01F6 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 6-85. MCKO Pad Configuration Register (SIU_PCR219)

See Table 6-18 for bit field definitions.

6.4.1.12.73 Pad Configuration Register 223–220 (SIU_PCR223–SIU_PCR220)

The SIU_PCR223–SIU_PCR220 registers control the drive strength of the MDO[3:0] pins.

Address: SIU_BASE + (0x01FE–0x01F8) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 6-86. MDO[3:0] Pad Configuration Register (SIU_PCR223–SIU_PCR220)

See Table 6-18 for bit field definitions.

6.4.1.12.74 Pad Configuration Register 225–224 (SIU_PCR225–SIU_PCR224)

The SIU_PCR225–SIU_PCR224 registers control the drive strength of the $\overline{\text{MSEO}}$[1:0] pins.

Address: SIU_BASE + (0x0202–0x0200) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 6-87. $\overline{\text{MSEO}}$[1:0] Pad Configuration Register (SIU_PCR225–SIU_PCR224)

See Table 6-18 for bit field definitions.

6.4.1.12.75 Pad Configuration Register 226 (SIU_PCR226)

The SIU_PCR226 register controls the drive strength of the $\overline{\text{RDY}}$ pin.

208 Package: $\overline{\text{RDY}}$ is not available due to pin limitations on the 208 package.

Address: SIU_BASE + 0x0204 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 6-88. $\overline{\text{RDY}}$ Pad Configuration Register (SIU_PCR226)

See Table 6-18 for bit field definitions.

6.4.1.12.76 Pad Configuration Register 227 (SIU_PCR227)

The SIU_PCR227 register controls the drive strength of the $\overline{\text{EVTO}}$ pin.

Address: SIU_BASE + 0x0206 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 6-89. $\overline{\text{EVTO}}$ Pad Configuration Register (SIU_PCR227)

See Table 6-18 for bit field definitions.

6.4.1.12.77 Pad Configuration Register 228 (SIU_PCR228)

The SIU_PCR228 register controls the drive strength of the TDO pin.

Address: SIU_BASE + 0x0208 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 6-90. TDO Pad Configuration Register (SIU_PCR228)

See Table 6-18 for bit field definitions.

6.4.1.12.78 Pad Configuration Register 229 (SIU_PCR229)

The SIU_PCR229 register controls the enabling/disabling and drive strength of the CLKOUT pin. The CLKOUT pin is enabled and disabled by setting and clearing the OBE bit. The CLKOUT pin is enabled during reset.

208 Package: CLKOUT is not available due to pin limitations in the 208 package.

Address: SIU_BASE + 0x020A Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	OBE	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0

Figure 6-91. CLKOUT Pad Configuration Register (SIU_PCR229)

See Table 6-18 for bit field definitions.

6.4.1.12.79 Pad Configuration Register 230 (SIU_PCR230)

The SIU_PCR230 register controls the slew rate of the $\overline{\text{RSTOUT}}$ pin.

Address: SIU_BASE + 0x020C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	SRC		0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

Figure 6-92. $\overline{\text{RSTOUT}}$ Pad Configuration Register (SIU_PCR230)

See Table 6-18 for bit field definitions.

6.4.1.12.80 Pad Configuration Register 336 (SIU_PCR336)

The SIU_PCR336 register controls the drive strength of the CAL_$\overline{\text{CS}}$[0] pin.

208 Package: Calibration signals are not available due to pin limitations.

Address: SIU_BASE + 0x02E0 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 6-93. CAL_$\overline{\text{CS}}$[0] Pad Configuration Register (SIU_PCR336)

6.4.1.12.81 Pad Configuration Registers 338–339 (SIU_PCR338–SIU_PCR339)

The SIU_PCR338–SIU_PCR339 registers control the pin function and drive strength of the CAL_$\overline{\text{CS}}$[2:3]_CAL_ADDR[10:11] pins.

208 Package: Calibration signals are not available due to pin limitations.

Address: SIU_BASE + (0x02E4–0x02E6) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PA	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0

Figure 6-94. CAL_$\overline{\text{CS}}$[2:3]_CAL_ADDR[10:11] Pad Configuration Registers (SIU_PCR338–SIU_PCR339)

6.4.1.12.82 Pad Configuration Register 340 (SIU_PCR340)

The SIU_PCR340 register controls the drive strength of the CAL_ADDR[12:30] pins. Multiple pins are controlled by this one PCR.

208 Package: Calibration signals are not available due to pin limitations.

Address: SIU_BASE + 0x02E8 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 6-95. CAL_ADDR[12:30] Pad Configuration Register (SIU_PCR340)

6.4.1.12.83 Pad Configuration Register 341 (SIU_PCR341)

The SIU_PCR341 register controls the drive strength of the CAL_DATA[0:15] pins. Multiple pins are controlled by this one PCR.

208 Package: Calibration signals are not available due to pin limitations.

Address: SIU_BASE + 0x02EA Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 6-96. CAL_DATA[0:15] Pad Configuration Register (SIU_PCR341)

6.4.1.12.84 Pad Configuration Register 342 (SIU_PCR342)

The SIU_PCR342 register controls the drive strength of the CAL_RD_$\overline{WR}$, CAL_$\overline{WE}$/$\overline{BE}$[0:1], CAL_$\overline{OE}$, and CAL_$\overline{TS}$ pins. Multiple pins are controlled by this one PCR. The WEBS bit in the EBI Base Registers selects between the write enable and byte enable functions.

208 Package: Calibration signals are not available due to pin limitations.

Address: SIU_BASE + 0x02EC Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	DSC		0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 6-97. CAL_RD_$\overline{WR}$, CAL_$\overline{WE}$/$\overline{BE}$[0:1], CAL_$\overline{OE}$, and CAL_$\overline{TS}$ Pad Configuration Register (SIU_PCR342)

6.4.1.13 GPIO Pin Data Output Registers 0–213 (SIU_GPDO*n*)

The definition of the 8-bit SIU_GPDO*n* registers, with each register specifying the drive data for a single GPIO pin, is given in Figure 6-98. The *n* notation in the name of the SIU_GPDO*n* registers corresponds to the pins with the same GPIO pin numbers. For example, PDO[213] is the pin data output bit for the WKPCFG_GPIO[213] pin, and you select it in SIU_GPDO213. The GPDO address for a pin is the SIU_BASE + 0x0600 plus the GPIO pin number.

The SIU_GPDO*n* registers are written to by software to drive data out on the external GPIO pin. Each register drives a single external GPIO pin, which allows the state of the pin to be controlled independently from other GPIO pins. Writes to the SIU_GPDO*n* registers have no effect on pin states if the pins are configured as inputs by the associated Pad Configuration Registers. The SIU_GPDO*n* register values are automatically driven to the GPIO pins without software update if the direction of the GPIO pins is changed from input to output.

When the pins are configured for the primary function, writes to the SIU_GPDO*n* registers have no effect on the state of these pins.

Address: SIU_BASE + (0x0600 + *n*) Access: R/W

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W								PDO*n*
Reset	0	0	0	0	0	0	0	0

Figure 6-98. GPIO Pin Data Output Register 0–213 (SIU_GPDO*n*)

Table 6-24. SIU_GPDO*n* Field Descriptions

Name	Description
PDO*n*	Pin data out. Stores the data to be driven out on the external GPIO pin associated with the register. If the register is read, it returns the value written. 0 V_{OL} is driven on the external GPIO pin when the pin is configured as an output. 1 V_{OH} is driven on the external GPIO pin when the pin is configured as an output.

6.4.1.14 GPIO Pin Data Input Registers 0–213 (SIU_GPDI*n*)

The definition of the 8-bit SIU_GPDI*n* registers, with each register specifying the drive data for a single GPIO pin, is given in Figure 6-99. The *n* notation in the name of the 155 SIU_GPDI*n* registers corresponds to the pins with the same GPIO pin numbers. For example, PDI0 is the pin data input bit for the $\overline{CS}$[0]_GPIO[0] pin and is found in SIU_GPDI0, and PDI213 is the pin data input bit for the WKPCFG_GPIO213 pin and is found in SIU_GPDI213. The GPDI address for a pin is the SIU_BASE + 0x0800 plus the GPIO pin number. Gaps exist in the memory addresses for pins that are not available in the 208 or 324 packages.

The SIU_GPDI*n* registers are read-only registers that allow software to read the input state of an external GPIO pin. Each register represents the input state of a single external GPIO pin. If the GPIO pin is configured as an output, and the input buffer enable (IBE) bit is set in the pad configuration register (PCR), the SIU_GPDI*n* register reflects the actual state of the output pin.

Address: SIU_BASE + (0x0800 + *n*) Access: R/O

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	PDI*n*
W								
Reset	0	0	0	0	0	0	0	0

Figure 6-99. GPIO Pin Data Input Register 0–213 (SIU_GPDI*n*)

Table 6-25. SIU_GPDI*n* Field Description

Name	Description
PDI*n*	Pin data in. This bit reflects the input state on the external GPIO pin associated with the register. If PCR*n*[IBE] = 1, then: 0 Signal on pin is less than or equal to V_{IL}. 1 Signal on pin is greater than or equal to V_{IH}.

6.4.1.15 eQADC Trigger Input Select Register (SIU_ETISR)

The SIU_ETISR selects the source for the eQADC trigger inputs. The eQADC trigger numbers 0–5 specified by TSEL(0–5) correspond to CFIFO numbers 0–5. To calculate the CFIFO number that each trigger is connected to, divide the DMA channel number by 2. So, for example, eQADC CFIFO 1 (connected to DMA channel 2) can be triggered by eTPUA[31] or eMIOS[11]. To select a trigger, the TSEL must be initialized.

When an eQADC trigger is connected, the timer output is connected to the eQADC CFIFO trigger input. To trigger the eQADC, the eTPU output must change to the state that the eQADC recognizes as a trigger. There are rising- or falling-edges, and low- or high-gated trigger types, so it is possible to trigger the eQADC immediately if desired.

Table 6-26. Trigger Interconnections

TSEL Field (Trigger Number)	eQADC CFIFO	EQADC DMA Channel	eTPUA Channel	eMIOS Channel
0	0	0	eTPUA30	eMIOS10
1	1	2	eTPUA31	eMIOS11
2	2	4	eTPUA29	eMIOS15
3	3	6	eTPUA28	eMIOS14
4	4	8	eTPUA27	eMIOS13
5	5	10	eTPUA26	eMIOS12

Address: SIU_BASE + 0x0900 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TSEL5		TSEL4		TSEL3		TSEL2		TSEL1		TSEL0		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-100. eQADC Trigger Input Select Register (SIU_ETISR)

Table 6-27. SIU_ETISR Field Descriptions

Bits	Name	Description
0–1	TSEL5 [0:1]	eQADC trigger input select 5. Specifies the input for eQADC trigger 5. 00 GPIO[207] 01 ETPUA[26] channel 10 EMIOS[12] channel 11 Invalid value
2–3	TSEL4 [0:1]	eQADC trigger input select 4. Specifies the input for eQADC trigger 4. 00 GPIO[206] 01 ETPUA[27] channel 10 EMIOS[13] channel 11 Invalid value
4–5	TSEL3 [0:1]	eQADC trigger input select 3. Specifies the input for eQADC trigger 3. 00 GPIO[207] 01 ETPUA[28] channel 10 EMIOS[14] channel 11 Invalid value
6–7	TSEL2 [0:1]	eQADC trigger input select 2. Specifies the input for eQADC trigger 2 00 GPIO[206] 01 ETPUA[29] channel 10 EMIOS[15] channel 11 Invalid value
8–9	TSEL1 [0:1]	eQADC trigger input select 1. Specifies the input for eQADC trigger 1 00 GPIO[207] 01 ETPUA[31] channel 10 EMIOS[11] channel 11 Invalid value
10–11	TSEL0 [0:1]	eQADC trigger input select 0. Specifies the input for eQADC trigger 0 00 GPIO[206] 01 ETPUA[30] channel 10 EMIOS[10] channel 11 Invalid value
12–31	—	Reserved

6.4.1.16 External IRQ Input Select Register (SIU_EIISR)

The SIU_EIISR selects the source for the external interrupt/DMA inputs.

Address: SIU_BASE + 0x0904 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ESEL15		ESEL14		ESEL13		ESEL12		ESEL11		ESEL10		ESEL9		ESEL8	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ESEL7		0	0	ESEL5		ESEL4		ESEL3		ESEL2		ESEL1		ESEL0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-101. External IRQ Input Select Register 1 (SIU_EIISR)

Table 6-28. SIU_EIISR Field Descriptions

Bits	Name	Description
0–1	ESEL15 [0:1]	External IRQ input select 15. Specifies the input for $\overline{\text{IRQ}}$ [15]. 00 $\overline{\text{IRQ}}$[15] pin 01 PCSB[15] serialized input (EMIOS[12] pin) 10 PCSC[0] serialized input (ETPUA[12] pin) 11 PCSD[1] serialized input (ETPUA[20] pin)
2–3	ESEL14 [0:1]	External IRQ input select 14. Specifies the input for $\overline{\text{IRQ}}$[14]. 00 $\overline{\text{IRQ}}$[14] pin 01 PCSB[14] serialized input (EMIOS[13] pin) 10 PCSC[15] serialized input (ETPUA[11] pin) 11 PCSD[0] serialized input (ETPUA[21] pin)
4–5	ESEL13 [0:1]	External IRQ input select 13. Specifies the input for $\overline{\text{IRQ}}$[13]. 00 $\overline{\text{IRQ}}$[13] pin 01 PCSB[13] serialized input (ETPUA[24] pin) 10 PCSC[14] serialized input (ETPUA[10] pin) 11 PCSD[15] serialized input (ETPUA[24] pin)
6–7	ESEL12 [0:1]	External IRQ input select 12. Specifies the input for $\overline{\text{IRQ}}$12]. 00 $\overline{\text{IRQ}}$[12] pin 01 PCSB[12] serialized input (ETPUA[25] pin) 10 PCSC[13] serialized input (ETPUA[9] pin) 11 PCSD[14] serialized input (ETPUA[25] pin)
8–9	ESEL11 [0:1]	External IRQ input select 11. Specifies the input for $\overline{\text{IRQ}}$[11]. 00 $\overline{\text{IRQ}}$[11] pin 01 PCSB[11] serialized input (ETPUA[26] pin) 10 PCSC[12] serialized input (ETPUA[8] pin) 11 PCSD[13] serialized input (ETPUA[26] pin)

Table 6-28. SIU_EIISR Field Descriptions (continued)

Bits	Name	Description
10–11	ESEL10 [0:1]	External IRQ input select 10. Specifies the input for $\overline{\text{IRQ}}$[10]. 00 $\overline{\text{IRQ}}$[10] pin 01 PCSB[10] serialized input (ETPUA[27] pin) 10 PCSC[11] serialized input (ETPUA[7] pin) 11 PCSD[12] serialized input (ETPUA[27] pin)
12–13	ESEL9 [0:1]	External IRQ input select 9. Specifies the input for $\overline{\text{IRQ}}$[9]. 00 $\overline{\text{IRQ}}$[9] pin 01 PCSB[9] serialized input (ETPUA[28] pin) 10 PCSC[10] serialized input (ETPUA[6] pin) 11 PCSD[11] serialized input (ETPUA[28] pin)
14–15	ESEL8 [0:1]	External IRQ input select 8. Specifies the input for $\overline{\text{IRQ}}$[8]. 00 $\overline{\text{IRQ}}$[8] pin 01 PCSB[8] serialized input (ETPUA[29] pin) 10 PCSC[9] serialized input (ETPUA[5] pin) 11 PCSD[10] serialized input (ETPUA[29] pin)
16–17	ESEL7 [0:1]	External IRQ input select 7. Specifies the input for $\overline{\text{IRQ}}$[7]. 00 $\overline{\text{IRQ}}$[7] pin 01 PCSB[7] serialized input (ETPUA[16] pin) 10 PCSC[8] serialized input (ETPUA[4] pin) 11 PCSD[9] serialized input (EMIOS[12] pin)
18–19	ESEL6 [0:1]	Although the $\overline{\text{IRQ}}$[6] pin is not available, ESEL6 selects other inputs for the internal $\overline{\text{IRQ}}$[6] signal. 00 Invalid value 01 PCSB[6] serialized input (ETPUA[17] pin) 10 PCSC[7] serialized input (ETPUA[3] pin) 11 PCSD[8] serialized input (EMIOS[13] pin)
20–21	ESEL5 [0:1]	External IRQ input select 5. Specifies the input for $\overline{\text{IRQ}}$[5]. 00 $\overline{\text{IRQ}}$[5] 01 PCSB[5] serialized input (ETPUA[18] pin) 10 PCSC[6] serialized input (ETPUA[2] pin) 11 PCSD[7] serialized input (EMIOS[10] pin)
22–23	ESEL4 [0:1]	External IRQ input select 4. Specifies the input for $\overline{\text{IRQ}}$[4]. 00 $\overline{\text{IRQ}}$[4] pin 01 PCSB[4] serialized input (ETPUA[19] pin) 10 PCSC[5] serialized input (ETPUA[1] pin) 11 PCSD[6] serialized input (EMIOS[11] pin)
24–25	ESEL3 [0:1]	External IRQ input select 3. Specifies the input for $\overline{\text{IRQ}}$[3]. 00 $\overline{\text{IRQ}}$[3] pin 01 PCSB[3] serialized input (ETPUA[20] pin) 10 PCSC[4] serialized input (ETPUA[0] pin) 11 PCSD[5] serialized input (ETPUA[16] pin)
26–27	ESEL2 [0:1]	External IRQ input select 2. Specifies the input for $\overline{\text{IRQ}}$[2]. 00 $\overline{\text{IRQ}}$[2] pin 01 PCSB[2] serialized input (ETPUA[21] pin) 10 PCSC[3] serialized input (ETPUA[15] pin) 11 PCSD[4] serialized input (ETPUA[17] pin)

Table 6-28. SIU_EIISR Field Descriptions (continued)

Bits	Name	Description
28–29	ESEL1 [0:1]	External IRQ input select 1. Specifies the input for $\overline{IRQ}$[1]. 00 $\overline{IRQ}$[1] pin 01 PCSB[1] serialized input (EMIOS[10] pin) 10 PCSC[2] serialized input (ETPUA[14] pin) 11 EMIOS[15] pin
30–31	ESEL0 [0:1]	External IRQ input select 0. Specifies the input for $\overline{IRQ}$[0]. 00 $\overline{IRQ}$[0] pin 01 PCSB[0] serialized input (EMIOS[11] pin) 10 PCSC[1] serialized input (ETPUA[5] pin) 11 EMIOS[14] pin

6.4.1.17 DSPI Input Select Register (SIU_DISR)

The SIU_DISR specifies the source of each DSPI data input, slave select, clock input, and trigger input to allow serial and parallel chaining of the DSPI modules.

Address: SIU_BASE + 0x0908 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	SINSELB		SSSELB		SCKSELB		TRIGSELB	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SINSELC		SSSELC		SCKSELC		TRIGSELC		SINSELD		SSSELD		SCKSELD		TRIGSELD	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-102. DSPI Input Select Register (SIU_DISR)

Table 6-29. SIU_DISR Field Descriptions

Bits	Name	Description
0–7	0	Reserved
8–9	SINSELB [0:1]	PCSB data input select. Specifies the source of PCSB data input. 00 SINB_PCSC[2]_GPIO[103] pin 01 Invalid value 10 SOUTC 11 SOUTD
10–11	SSSELB [0:1]	PCSB slave select input select. Specifies the source of the PCSB slave select input. 00 PCSB[0]_PCSD[2]_GPIO[105] pin 01 Invalid value 10 PCSC[0] (Master) 11 PCSD[0] (Master)

Table 6-29. SIU_DISR Field Descriptions (continued)

Bits	Name	Description
12–13	SCKSELB [0:1]	PCSB clock input select. Specifies the source of the PCSB clock input. 00 SCKB_PCSC[1]_GPIO[102] pin 01 Invalid value 10 SCKC (Master) 11 SCKD (Master)
14–15	TRIGSELB [0:1]	PCSB trigger input select. Specifies the source of the PCSB trigger input for master or slave mode. 00 Invalid value 01 Invalid value 10 PCSC[4] 11 PCSD[4]
16–17	SINSELC [0:1]	SINC data input select. Specifies the source of the SINC data input. 00 PCSB[2]_SINC_GPIO[108] pin01Invalid value 10 SOUTB 11 SOUTD
18–19	SSSELC [0:1]	PCSC slave select input select. Specifies the source of the PCSC slave select input. 00 PCSB[5]_PCSC[0]_GPIO[110] pin 01 Invalid value 10 PCSB[0] (Master) 11 PCSD[0] (Master)
20–21	SCKSELC [0:1]	PCSC clock input select. Specifies the source of the PCSC clock input when in slave mode. 00 PCSB[4]_SCKC_GPIO[109] pin 01 Invalid value 10 SCKB (Master) 11 SCKD (Master)
22–23	TRIGSELC [0:1]	PCSC trigger input select. Specifies the source of the PCSC trigger input for master or slave mode. 00 Invalid value 01 Invalid value 10 PCSB[4] 11 PCSD[4]
24–25	SINSELD [0:1]	SIND data input select. Specifies the source of the SIND data input. 00 SIND_GPIO[99] pin01Invalid value 10 SOUTB 11 SOUTC
26–27	SSSELD [0:1]	PCSD slave select input select. Specifies the source of the PCSD slave select input. 00 PCSB[1]_PCSD[0]_GPIO[106] pin 01 Invalid value 10 PCSB[0] (Master) 11 PCSC[0] (Master)

Table 6-29. SIU_DISR Field Descriptions (continued)

Bits	Name	Description
28–29	SCKSELD [0:1]	PCSD clock input select. Specifies the source of the PCSD clock input in slave mode. 00 SCKD_GPIO[98] pin 01 Invalid value 10 SCKB (Master) 11 SCKC (Master)
30–31	TRIGSELD [0:1]	PCSD trigger input select. Specifies the source of the PCSD trigger input for master or slave mode. 00 Invalid value 01 Invalid value 10 PCSB[4] 11 PCSC[4]

6.4.1.18 Chip Configuration Register (SIU_CCR)

Address: SIU_BASE + 0x0980 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MATCH	DISNEX[1]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U	X[1]

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CRSE[2]	TEST[2]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 When system reset negates, the value in this bit depends on the censorship control word and the boot configuration bits. In the 208 package, BOOTCFG[0] is not available due to pin limitation and internally asserted (driven to 0).

2 This bit is reset with a power on reset.

Figure 6-103. Chip Configuration Register (SIU_CCR)

Table 6-30. SIU_CCR Field Descriptions

Bits	Name	Description
0–13	—	Reserved
14	MATCH	Compare register match. Holds the value of the match input signal to the SIU. The match input is asserted if the values in the SIU_CARH and SIU_CARL, and SIU_CBRH and SIU_CBRL are equal. The MATCH bit is reset by the synchronous reset signal. 0 The content of SIU_CARH and SIU_CARL does not match the content of SIU_CBRH and SIU_CBRL 1 The content of SIU_CARH and SIU_CARL matches the content of SIU_CBRH and SIU_CBRL

Table 6-30. SIU_CCR Field Descriptions

Bits	Name	Description
15	DISNEX	Disable Nexus. Holds the value of the Nexus disable input signal to the SIU. When system reset negates, the value in this bit depends on the censorship control word and the boot configuration bits. 0 Nexus disable input signal is negated. 1 Nexus disable input signal is asserted.
16–29	—	Reserved
30	CRSE	Calibration Reflection Suppression Enable. Enables the suppression of reflections from the EBI calibration bus onto the non-calibration bus. The EBI drives some outputs to both the calibration and non-calibration busses. When CRSE is asserted, the values driven on the calibration bus pins are not shown on the non-calibration bus pins. When CRSE is negated, the values driven on the calibration bus pins are shown on the non-calibration bus pins. CRSE only enables reflection suppression for non-calibration bus pins which do not have a negated state to which the pins return at the end of the access. CRSE does not enable reflection suppression for the non-calibration bus pins which have a negated state to which the pins return at the end of an access. Those reflections always are suppressed. Furthermore, the suppression of reflections from the non-calibration bus onto the calibration bus is not enabled by CRSE. Those reflections also always are suppressed. 0 Calibration reflection suppression is disabled. 1 Calibration reflection suppression is enabled.
31	TEST	Test mode enable. Allows reads or writes to undocumented registers used only for production tests. Since these production test registers are undocumented, estimating the impact of errant accesses to them is impossible. The application must not change this bit from its negated state at reset. 0 Undocumented production test registers can not be read or written. 1 Undocumented production test registers can be read or written.

6.4.1.19 External Clock Control Register (SIU_ECCR)

The SIU_ECCR controls the timing relationship between the system clock and the external clocks ENGCLK and CLKOUT. All bits and fields in the SIU_ECCR are read/write and are reset by the synchronous reset signal.

208 Package: CLKOUT is not available due to pin limitations.

Address: SIU_BASE + 0x0984 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	ENGDIV						0	0	0	0	EBTS	0	EBDF	
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

Figure 6-104. External Clock Control Register (SIU_ECCR)

Table 6-31. SIU_ECCR Field Descriptions

Bits	Name	Description
0–17	—	Reserved
18–23	ENGDIV [0:5]	Engineering clock division factor. Specifies the frequency ratio between the system clock and ENGCLK. The ENGCLK frequency is divided from the system clock frequency according to the following equation: $\text{Engineering clock frequency} = \frac{\text{System clock frequency}}{\text{ENGDIV} \times 2}$ **Note:** Clearing ENGDIV to 0 is reserved. Synchronization between ENGCLK and CLKOUT cannot be guaranteed.
24–27	—	Reserved
28	EBTS	External bus tap select. Changes the phase relationship between the system clock and CLKOUT. Changing the phase relationship so that CLKOUT is advanced in relation to the system clock increases the output hold time of the external bus signals to a non-zero value. It also increases the output delay times, increases the input hold times to non-zero values, and decreases the input setup times. See the Electrical Specifications for how the EBTS bit affects the external bus timing. 0 External bus signals have zero output hold times. 1 External bus signals have non-zero output hold times. **Note:** Do not modify the EBTS bit while an external bus transaction is in progress.

Table 6-31. SIU_ECCR Field Descriptions (continued)

Bits	Name	Description
29	—	Reserved
30–31	EBDF [0:1]	External bus division factor. Specifies the frequency ratio between the system clock and the external clock, CLKOUT. The EBDF field must not be changed during an external bus access or while an access is pending. The CLKOUT frequency is divided from the system clock frequency according to the descriptions below. This divider must be kept as divide-by-2 when operating in dual controller mode. 00 Divide by 1 01 Divide by 2 10 Invalid value 11 Divide by 4 **Note:** The reset value of the EBDF field is divide-by-2. After reset, if EBDF is changed to divided-by-1, no glitches occur on the CLKOUT signal. If EBDF is changed back to divide-by-2 or divide-by-4, glitches can occur during the switch. **Note:** CLKOUT is not available in the 208 package due to pin limitations.

6.4.1.20 Compare A High Register (SIU_CARH)

The compare registers are not intended for general application use, but are used temporarily by the BAM during boot and intended optionally for communication with calibration tools. After reset, calibration tools can immediately write a non-zero value to these registers. The application code, using the registers then as read only, can read them to determine if a calibration tool is attached and operate appropriately.

The compare registers can be used just like 128 bits of memory mapped RAM that is always zero out of reset, or they can perform a 64 bit to 64 bit compare. The compare function is continuous (combinational logic - not requiring a start or stop). The compare result appears in the MATCH bit in the SIU_CCR register.

The SIU_CARH holds the 32-bit value that is compared against the value in the SIU_CBRH register. The CMPAH field is read/write and is reset by the synchronous reset signal.

Address: SIU_BASE + 0x0988 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPAH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMPAH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-105. Compare A High Register (SIU_CARH)

6.4.1.21 Compare A Low Register (SIU_CARL)

The SIU_CARL register holds the 32-bit value that is compared against the value in the SIU_CBRL register. The CMPAL field is read/write and is reset by the synchronous reset signal.

Address: SIU_BASE + 0x098C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPAL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMPAL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-106. Compare A Low Register (SIU_CARL)

6.4.1.22 Compare B High Register (SIU_CBRH)

The SIU_CBRH holds the 32-bit value that is compared against the value in the SIU_CARH. The CMPBH field is read/write and is reset by the synchronous reset signal.

Address: SIU_BASE + 0x0990 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPBH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMPBH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-107. Compare B High Register (SIU_CBRH)

6.4.1.23 Compare B Low Register (SIU_CBRL)

The SIU_CBRL holds the 32-bit value that is compared against the value in the SIU_CARL. The CMPBL field is read/write and is reset by the synchronous reset signal.

Address: SIU_BASE + 0x0994 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPBL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMPBL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-108. Compare B Low Register (SIU_CBRL)

6.5 Functional Description

The following sections provide an overview of the SIU operation.

6.5.1 System Configuration

6.5.1.1 Boot Configuration

The BOOTCFG[0:1] pins are used to determine the boot mode initiated by the BAM program, and whether external arbitration is selected for external booting. The BAM program uses the BOOTCFG field to determine where to read the reset configuration word, and whether to initiate a FlexCAN or eSCI boot. See Section 15.3.2.3.4, “Read the Reset Configuration Halfword” of the BAM chapter for detail on the RCHW. Table 6-32 defines the boot modes specified by the BOOTCFG[0:1] pins. If the $\overline{\text{RSTCFG}}$ pin is asserted during the assertion of $\overline{\text{RSTOUT}}$, except in the case of a software external reset, the BOOTCFG pins are latched 4 clock cycles prior to the negation of the $\overline{\text{RSTOUT}}$ pin and are used to update the SIU_RSR and the BAM boot mode. Otherwise, if $\overline{\text{RSTCFG}}$ is negated during the assertion of $\overline{\text{RSTOUT}}$, the BOOTCFG pins are ignored and the device defaults to ‘boot from internal flash memory’ mode.

208 Package: BOOTCFG[0] and $\overline{\text{RSTCFG}}$ are not available due to pin limitations and are internally asserted (driven to 0) in the 208 package.

Table 6-32. BOOTCFG[0:1] Configuration

Value	Meaning
0b00	Boot from internal flash memory
0b01	FlexCAN or eSCI boot

Table 6-32. BOOTCFG[0:1] Configuration

Value	Meaning
0b10	Boot from external memory (no arbitration)
0b11	Invalid value

6.5.1.2 Pad Configuration

The pad configuration registers (SIU_PCR) in the SIU allow software control of the static electrical characteristics of external pins. The pad configuration registers allow control over the following external pin characteristics:

- Weak pull up/down enable/disable
- Weak pull up/down selection
- Slew-rate selection for outputs
- Drive strength selection for outputs
- Input buffer enable (when direction is configured for output)
- Input hysteresis enable/disable
- Open drain/push-pull output selection
- Multiplexed function selection
- Data direction selection

The pad configuration registers are provided to allow centralized control over external pins that are shared by more than one module. Each pad configuration register controls a single pin.

6.5.2 Reset Control

The reset controller logic is located in the SIU. See Chapter 4, “Reset” for detail on reset operation.

6.5.2.1 $\overline{\text{RESET}}$ Pin Glitch Detect

The reset controller provides a glitch detect feature on the $\overline{\text{RESET}}$ pin. If the reset controller detects that the $\overline{\text{RESET}}$ pin is asserted for more than two clock cycles, the event is latched. Once the latch is set, if the $\overline{\text{RESET}}$ pin is negated before 10 clock cycles completes the reset controller sets the RGF bit without affecting any of the other bits in the reset status register. The latch is cleared when the RGF bit is set or a valid reset is recognized. The RGF bit remains set until cleared by software or the $\overline{\text{RESET}}$ pin is asserted for 10 clock cycles. The reset controller does not respond to assertions of the $\overline{\text{RESET}}$ pin if a reset cycle is already being processed.

6.5.3 External Interrupt

There are sixteen external interrupt inputs $\overline{\text{IRQ}}$[0:15] to the SIU. The $\overline{\text{IRQ}}$[*n*] inputs can be configured for rising or falling edge events or both. Each $\overline{\text{IRQ}}$[*n*] input has a corresponding flag bit in the external interrupt status register (SIU_EISR). The flag bits for the $\overline{\text{IRQ}}$[4:15] inputs are ORed together to form one interrupt request to the interrupt controller (OR function performed in the integration glue logic). The flag

bits for the $\overline{\text{IRQ}}$[0:3] inputs can generate either an interrupt request to the interrupt controller or a DMA transfer request to the DMA controller. Table 6-109 shows the DMA and interrupt request connections to the interrupt and DMA controllers.

The SIU contains an overrun request for each IRQ and one combined overrun request which is the logical OR of the individual overrun requests. Only the combined overrun request is used in the device, and the individual overrun requests are not connected.

Each IRQ pin has a programmable filter for rejecting glitches on the IRQ signals. The filter length for the IRQ pins is specified in the external IRQ digital filter register (SIU_IDFR).

NOTE

$\overline{\text{IRQ}}$[2] signal is not available due to pin limitations and is internally asserted (driven to 0) on the 208 package. $\overline{\text{IRQ}}$[6] signal is not available in this device, however the $\overline{\text{IRQ}}$[6] signal is available internally.

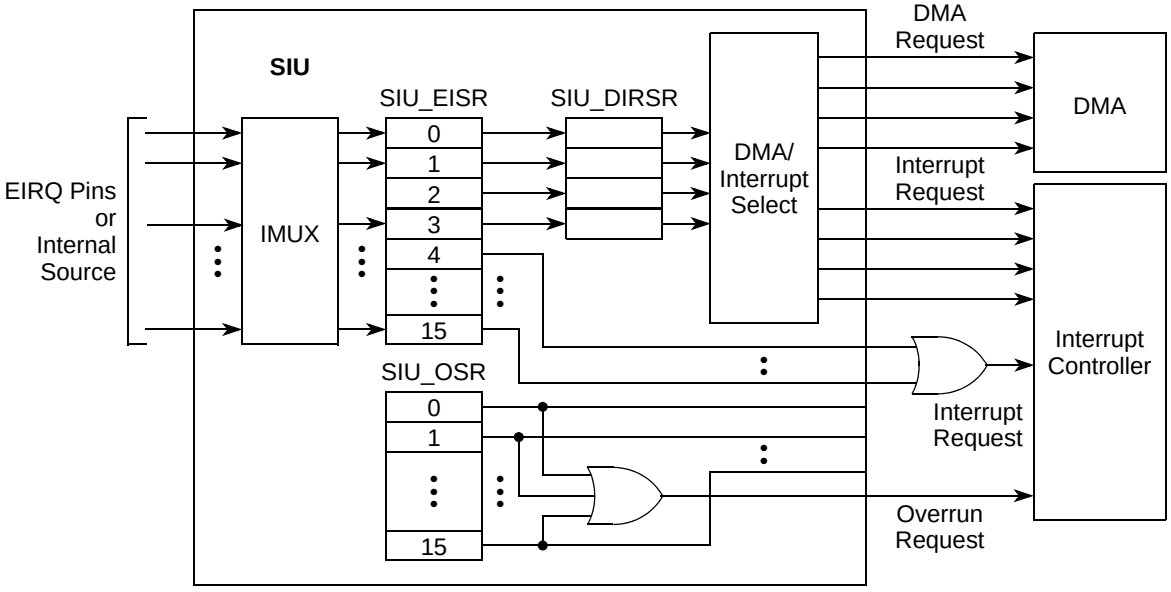


Figure 6-109. SIU DMA/Interrupt Request Diagram

6.5.4 GPIO Operation

All GPIO functionality is provided by the SIU for the device. Each device pin that has GPIO functionality has an associated pin configuration register in the SIU where the GPIO function is selected for the pin. In addition, each device pin with GPIO functionality has an input data register (SIU_GPDI*n*) and an output data register (SIU_GPDO*n*).

6.5.5 Internal Multiplexing

The internal multiplexing select registers SIU_ETISR, SIU_EIISR, and SIU_DISR provide selection of the source of the input for the eQADC external trigger inputs, the SIU external interrupts, and the DSPI signals that are used in serial and parallel chaining of the DSPI modules.

Internal multiplexing allows you to select the input for multiplexed external signals. For each field of each of the select registers, a multiplexor exists in the SIU. The inputs and outputs of the multiplexors are external signals to and from the SIU.

A block diagram of the internal multiplexing feature is given in Figure 6-110. The figure shows the multiplexing of four external signals to an output from the SIU. A two bit SEL field from an SIU select register is used to select the input of the multiplexor.

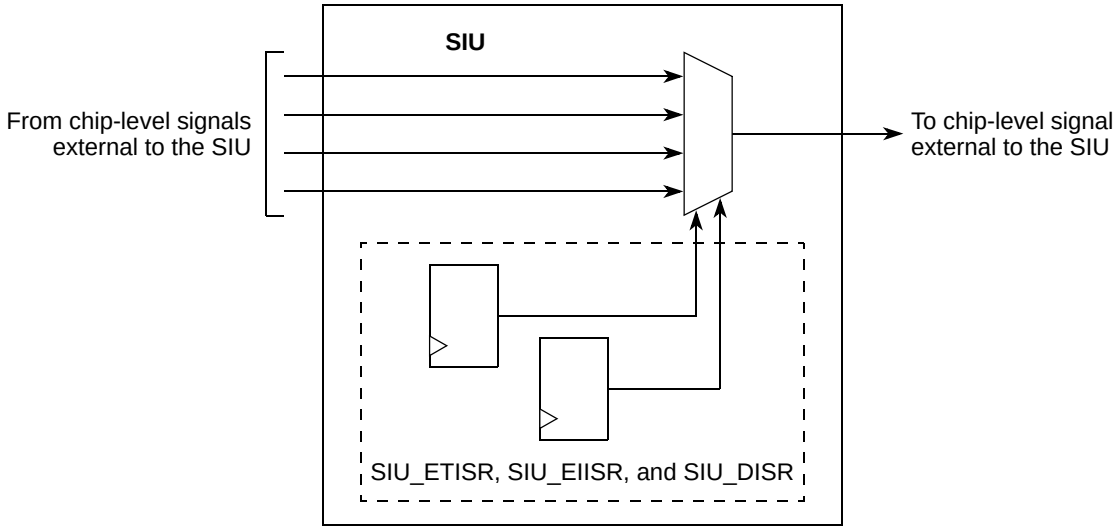


Figure 6-110. Four-to-One Internal Multiplexing Block Diagram

6.5.5.1 eQADC External Trigger Input Multiplexing

The eQADC external trigger inputs can be connected to an external pin, eTPU channel, or eMIOS channel. The input source for each eQADC external trigger is individually specified in the eQADC trigger input select register (SIU_ETISR). An example of the multiplexing of an eQADC external trigger input is given in Figure 6-111. As shown in the figure, the GPIO[206] input of the eQADC can be connected to the ETPUA[30] channel or the EMIOS[10] channel. The remaining trigger inputs are multiplexed in the same manner (see Section 6.4.1.15, “eQADC Trigger Input Select Register (SIU_ETISR)” for the SIU_ETISR[TSEL0]–SIU_ETISR[TSEL5] bit definitions). If an external input trigger is connected to an eTPU or eMIOS channel, the external pin used by that channel can be used by the alternate function on that pin.

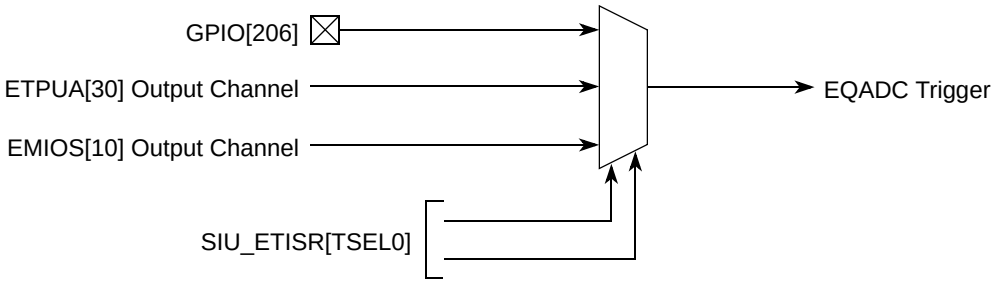


Figure 6-111. eQADC External Trigger Input Multiplexing

6.5.5.2 SIU External Interrupt Input Multiplexing

The sixteen SIU external interrupt inputs can be connected to either an external pin or to serialized output signals from a DSPI module. The input source for each SIU external interrupt is individually specified in the external IRQ input select register (SIU_EIISR). An example of the multiplexing of an SIU external interrupt input is given in Figure 6-112. As shown in the figure, the $\overline{\text{IRQ}}$[0] input of the SIU can be connected to either the EMIOS[14]_$\overline{\text{IRQ}}$[0]_GPIO[193] pin, the PCSB[0] serial input signal, the PCSC[1] deserialized output signal, or the PCSD[2] deserialized output signal. The remaining IRQ inputs are multiplexed in the same manner. The inputs to the IRQ from each DSPI module are offset by one so that if more than one DSPI module is connected to the same external device type, a separate interrupt can be generated for each device. This also applies to DSPI modules connected to external devices of different type that have status bits in the same bit location of the deserialized information.

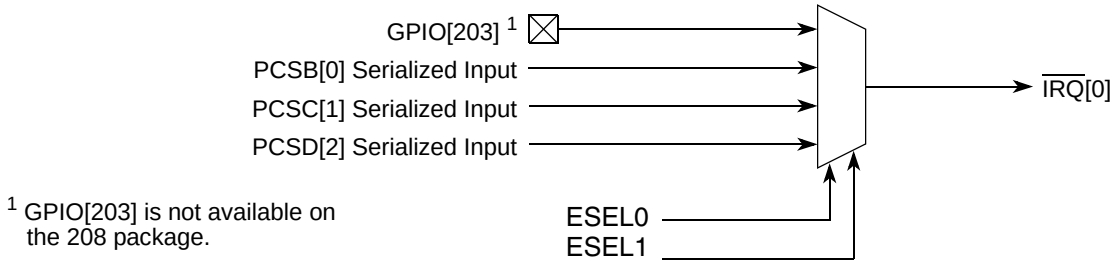


Figure 6-112. DSPI Serialized Input Multiplexing

6.5.5.3 Multiplexed Inputs for DSPI Multiple Transfer Operation

Each DSPI module can be combined in a serial or parallel chain (multiple transfer operation). Serial chaining allows SPI operation with an external device that has more bits than one DSPI module. An example of a serial chain is shown in Figure 6-113. In a serial chain, one DSPI module operates as a master, the second, third, or fourth DSPI modules operate as slaves. The data output (SOUT) of the master is connected to the data input (SIN) of the slave. The SOUT of a slave is connected to the SIN of subsequent slaves until the last module in the chain, where the SOUT is connected to an external pin, which connects to the input of an external SPI device. The slave DSPI and external SPI device use the master peripheral chip select (PCS) and clock (SCK). The trigger input of the master allows a slave DSPI to trigger a transfer when a data change occurs in the slave DSPI and the slave DSPI is operating in change in data mode. The trigger input of the master is connected to $\overline{\text{MTRIG}}$ output of the slave. If more than two DSPIs are chained

in change in data mode, a chain must be connected of $\overline{\text{MTRIG}}$ outputs to trigger inputs through the slaves with the last slave $\overline{\text{MTRIG}}$ output connected to the master trigger input.

Parallel chaining allows the PCS and SCK from one DSPI to be used by more than one external SPI device, thus reducing pin utilization of the MCU. An example of a parallel chain is shown in Figure 6-114. In this example, the SOUT and SIN of the two DSPIs connect to separate external SPI devices, which share a common PCS and SCK.

To support multiple transfer operation of the DSPIs, an input multiplexor is required for the SIN, $\overline{\text{SS}}$, SCK IN, and trigger signals of each DSPI. The input source for the SIN input of a DSPI can be a pin or the SOUT of any of the other three DSPIs. The input source for the $\overline{\text{SS}}$ input of a DSPI can be a pin or the PCS0 of any of the other three DSPIs. The input source for the SCK input of a DSPI can be a pin or the SCK output of any of the other three DSPIs. The input source for the trigger input can be the $\overline{\text{PCSS}}$ output of any of the other three DSPIs. The input source for each DSPI SIN, $\overline{\text{SS}}$, SCK, and trigger signal is individually specified in the DSPI input select register (SIU_DISR).

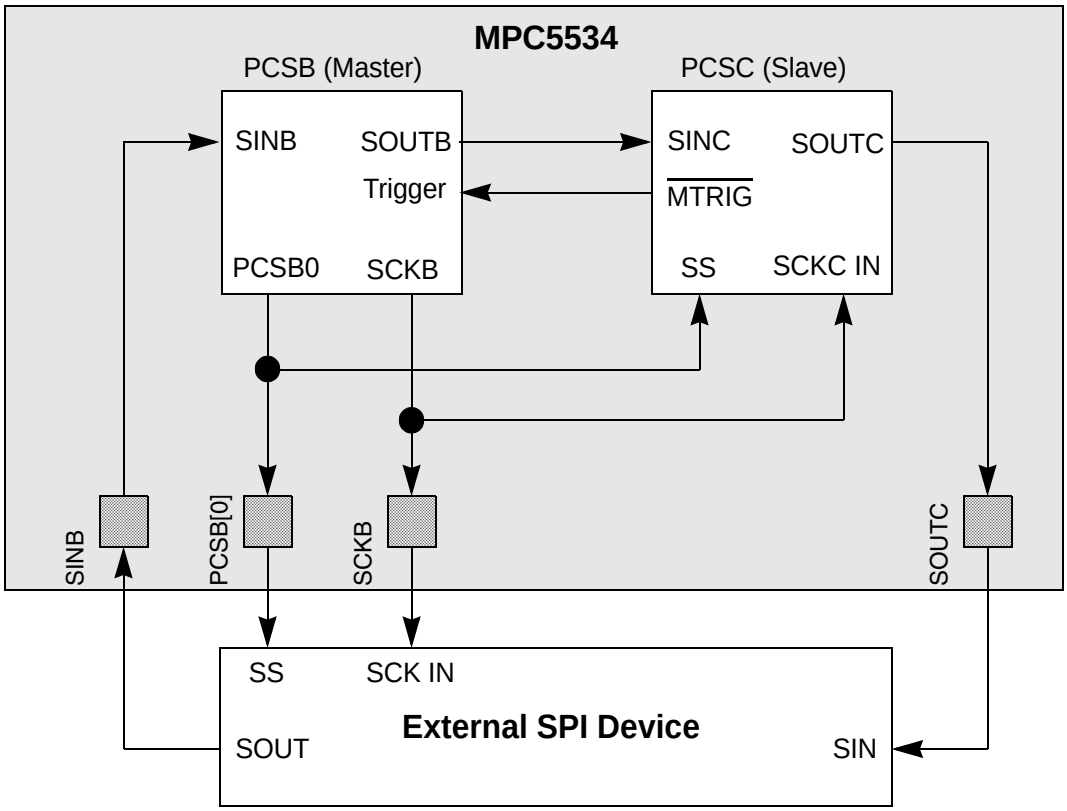


Figure 6-113. DSPI Serial Chaining

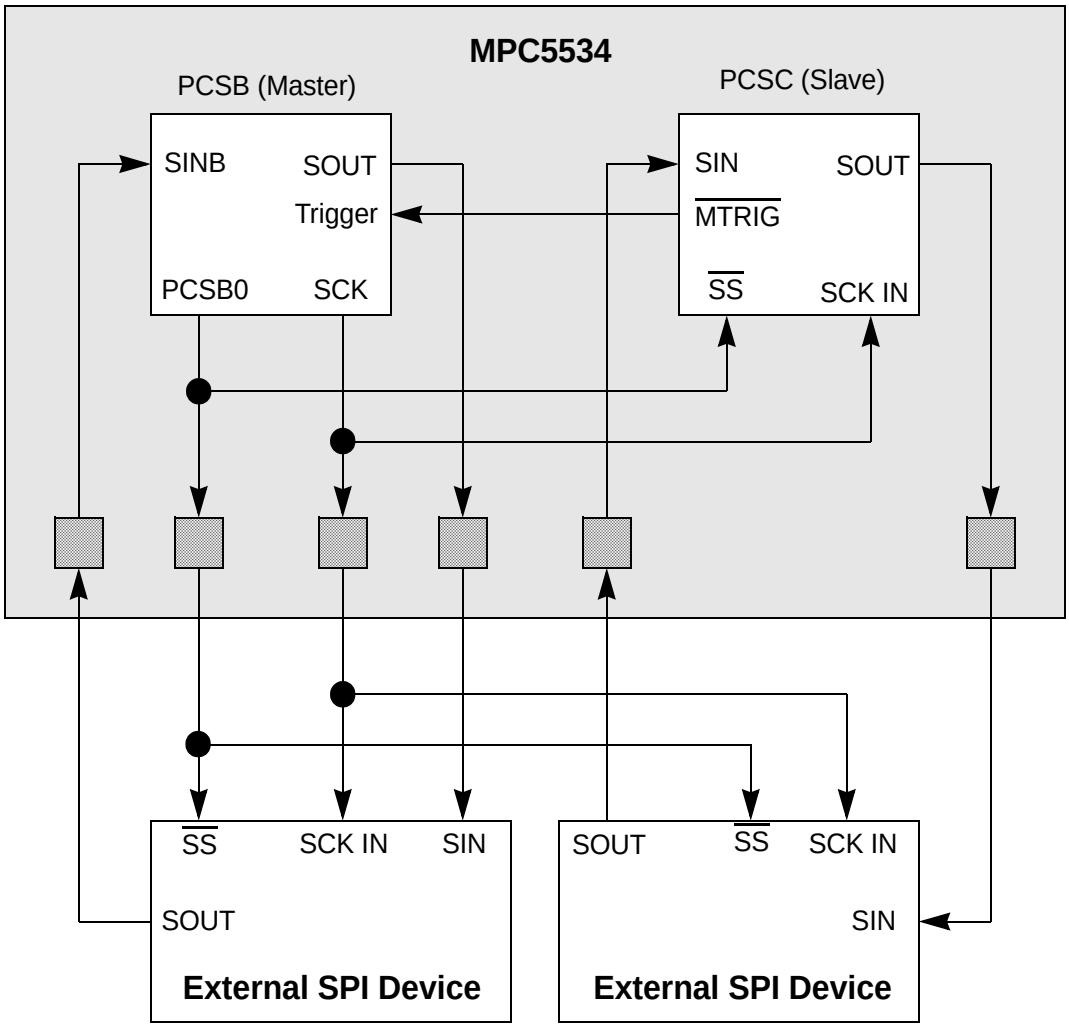


Figure 6-114. DSPI Parallel Chaining

Chapter 7
Crossbar Switch (XBAR)

7.1 Introduction

This chapter describes the multi-port crossbar switch (XBAR), which supports simultaneous connections between four master ports and five slave ports. XBAR supports a 32-bit address bus width and a 64-bit data bus width at all master and slave ports.

7.1.1 Block Diagram

Figure 7-1 shows a block diagram of the crossbar switch.

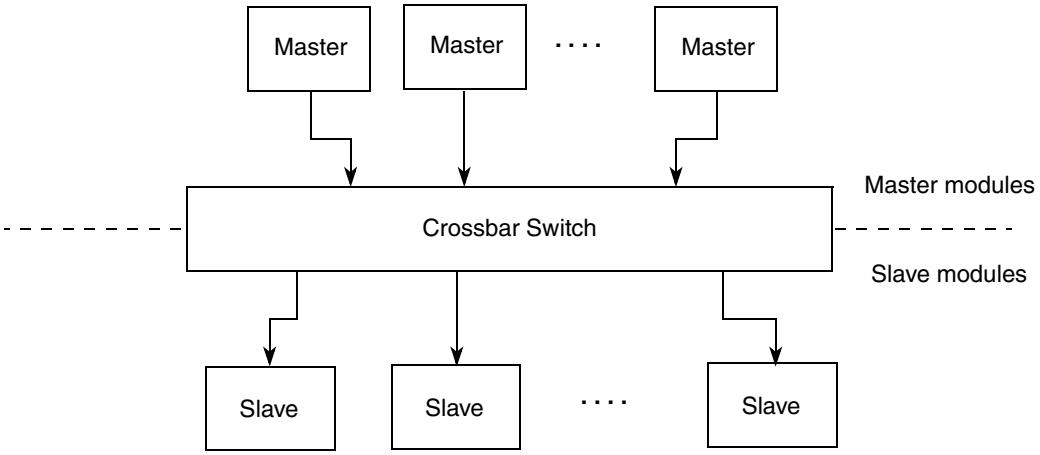


Figure 7-1. XBAR Block Diagram

7.1.2 Overview

The XBAR allows for concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions.

By default, requesting masters are granted access based on a fixed priority. A round-robin priority mode also is available. In this mode, requesting masters are treated with equal priority and are granted access to a slave port in round-robin fashion, based upon the ID of the last master to be granted access. A block diagram of the XBAR is shown in Figure 7-1.

The XBAR can place a slave port in a low-power park mode to avoid dissipating power, transitional address, control, or data signals when the master port is not actively accessing the slave port. There is a one-cycle arbitration overhead for exiting low-power park mode.

7.1.3 Features

- Four master ports:
  - core: e200z3 core–CPU data
    - e200z3 core–CPU instruction
  - eDMA
  - EBI
- Five slave ports
  - Flash (see Chapter 13, “Flash Memory” for information on accessing flash memory)
  - EBI
  - Internal SRAM
  - Peripheral bridge A
  - Peripheral bridge B
- 32-bit address, 64-bit data paths
- Fully concurrent transfers between independent master and slave ports

7.1.4 Modes of Operation

The following table lists the operating modes for the crossbar switch module.

Table 7-1. XBAR Operating Modes

Operating Mode	Description
Normal	XBAR provides the register interface and logic that controls crossbar switch configuration.
Debug	XBAR operation in debug mode is identical to operation in normal mode.

7.2 Memory Map and Register Definition

The memory map for the XBAR registers that are visible to the application is shown in Table 7-2.

NOTE

Do not read or write to the reserved memory locations in the XBAR register memory map; accessing these areas can cause unpredictable results.

Table 7-2. XBAR Register Memory Map

Address	Register Name	Register Description	Bits
Base = 0xFFF0_4000	XBAR_MPR0	Master priority register for slave port 0	32
Base + (0x0004–0x000F)	—	Reserved	—
Base + 0x0010	XBAR_SGPCR0	General-purpose control register for slave port 0	32
Base + (0x0014–0x00FF)	—	Reserved	—
Base + 0x0100	XBAR_MPR1	Master priority register for slave port 1	32
Base +(0x0104–0x010F)	—	Reserved	—
Base + 0x0110	XBAR_SGPCR1	General-purpose control register for slave port 1	32
Base + (0x0114–0x02FF)	—	Reserved	—
Base + 0x0300	XBAR_MPR3	Master priority register for slave port 3	32
Base + (0x0304–0x030F)	—	Reserved	—
Base + 0x0310	XBAR_SGPCR3	General-purpose control register for slave port 3	32
Base + (0x0314–0x05FF)		Reserved	
Base + 0x0600	XBAR_MPR6	Master priority register for slave port 6	32
Base + (0x0604–0x060F)	—	Reserved	—
Base + 0x0610	XBAR_SGPCR6	General-purpose control register for slave port 6	32
Base + (0x0614–0x06FF)	—	Reserved	—
Base + 0x0700	XBAR_MPR7	Master priority register for slave port 7	32
Base + (0x0704–0x070F)	—	Reserved	—
Base + 0x0710	XBAR_SGPCR7	General-purpose control register for slave port 7	32
(Base + 0x0714)–0x0003_FFFF	—	Reserved	—

7.2.1 Register Descriptions

There are two registers for each slave port of the XBAR. The registers can only be accessed in Supervisor Mode using 32-bit accesses.

The slave SGPCR also features a bit (RO), which when written with a 1, prevents all slave registers for that port from being written to again until a reset occurs. The registers remain readable, but future write attempts have no effect on the registers and are terminated with an error response.

Table 7-3 lists the crossbar switch master and slave identifiers for each module in the device:

- Fixed internal master ID numbers for each master port
- XBAR master and slave port ID numbers

Shown are the internal master ID numbers as they relate to the crossbar master port ID numbers:

Table 7-3. XBAR Switch Ports

Module	Internal Master ID	XBAR Port	
		Type	Number
e200z3 core–CPU instruction	0	Master	0
Enhanced direct memory access (eDMA)	2	Master	1
External bus interface (EBI)	3	Master	2
e200z3 core–CPU data	0	Master	4
e200z3–Nexus	1	Master	4
Flash memory	—	Slave	0
External bus interface (EBI)	—	Slave	1
Internal SRAM	—	Slave	3
Peripheral bridge A (PBRIDGE_A)	—	Slave	6
Peripheral bridge B (PBRIDGE_B)	—	Slave	7

7.2.1.1 Master Priority Registers (XBAR_MPR*n*)

The XBAR_MPR for a slave port sets the priority of each master port when operating in fixed priority mode. These registers are not used in round-robin priority mode unless more than one master is assigned as high priority by a slave.

IMPORTANT

Master ports must be assigned unique priority levels.

The master priority registers are accessible in Supervisor Mode only using 32-bit accesses. After the read only (RO) bit is set in the slave general-purpose control register, no writes to the master priority register are permitted; only read instructions are allowed. Attempts to write to master priority registers (MPR) have no effect and result in an error.

NOTE

XBAR_MPR must be written with a read/modify/write for code compatibility.

Address: Base + 0x0000 (XBAR_MPR0)
Base + 0x0100 (XBAR_MPR1)
Base + 0x0300 (XBAR_MPR3)
Base + 0x0700 (XBAR_MPR7)

Access: Supervisor R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	MSTR4		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	MSTR2			0	MSTR1			0	MSTR0		
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0

Table 7-4. XBAR_MPR*n* Descriptions

Field	Description
0–12	Reserved, must be cleared to 0
13–15 MSTR4	Master 4 priority. Sets the arbitration priority for the e200z3 core data and Nexus support for the core to master port 4 on the associated slave port. The reset value of 0b011 is the lowest priority. 000This master has the highest priority when accessing the slave port. ⋮ 011This master has the lowest priority when accessing the slave port. 100–111Invalid values
16–20	Reserved, must be cleared to 0
21–23 MSTR2	Master 2 priority. Sets the arbitration priority for the external bus interface (EBI) for master port 2 on the slave port. The MSTR2 reset value of 0b010 is the third highest priority. 000 This master has the highest priority when accessing the slave port. ⋮ 011 This master has the lowest priority when accessing the slave port. 100–111 Invalid values
24	Reserved, must be cleared to 0
25–27 MSTR1	Master 1 priority. Sets the arbitration priority for direct memory access (eDMA) for master port 1 on the slave port. The MSTR1 reset value of 0b001 is the second highest priority. 000 This master has the highest priority when accessing the slave port. ⋮ 011 This master has the lowest priority when accessing the slave port. 100–111 Invalid values

Table 7-4. XBAR_MPR*n* Descriptions (continued)

Field	Description
28	Reserved, must be cleared.
29–31 MSTR0	Master 0 priority. Sets the arbitration priority for the e200z3 core instruction for master port 0 on the slave port. The MSTR0 reset value of 0b000 is the highest priority. 000 This master has the highest priority when accessing the slave port. ⋮ 011 This master has the lowest priority when accessing the slave port. 100–111 Invalid values

7.2.1.2 Slave General-Purpose Control Registers (XBAR_SGPCR*n*)

The XBAR_SGPCR*n* of a slave port controls several features of the slave port, including the following:

- Round-robin or fixed arbitration policy for a particular slave port
- Write protection of any slave port registers
- Parking algorithm used for a slave port

The PARK field indicates which master port this slave port parks on when no active access attempts are being made to the slave and the parking control field is set to park on a specific master. XBAR_SGPCR*n*[PARK] must only be programmed to select master ports that are actually available on the device, otherwise undefined behavior results. The low-power park feature can result in an overall power savings if the slave port is not saturated; however, an extra clock cycle of latency results whenever any master tries to access a slave (not being accessed by another master) because it is not parked on any master.

The XBAR_SGPCR can only be accessed in supervisor mode with 32-bit accesses. After the RO (read only) bit is set in the XBAR_SGPCR, the XBAR_SGPCR and the SBAR_MPR can only be read. Attempts to write to them have no effect and results in an error.

NOTE

Some of the unused bits in the SGPCR*n* registers are writeable and readable, but they serve no function. Setting any of these bits has no effect on the operation of this module.

Addess: Base + 0x0010 (XBAR_SGPCR0)
Base + 0x0110 (XBAR_SGPCR1)
Base + 0x0310 (XBAR_SGPCR3)
Base + 0x0610 (XBAR_SGPCR6)
Base + 0x0710 (XBAR_SGPCR7)

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RO[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	ARB		0	0	PCTL		0	PARK		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] After this bit is set, only a hardware reset clears it.

Figure 7-2. Slave General-Purpose Control Registers (XBAR_SGPCR*n*)

Table 7-5. XBAR_SGPCR*n* Field Descriptions

Field	Description
0 RO	Read only. Forces all slave port registers to read only. To clear the read only bit requires a hardware reset. 0 All this slave port's registers can be written. 1 All this slave port's registers are read only and cannot be written (attempted writes have no effect and result in an error response).
1–21	Reserved, must be cleared.
22–23 ARB	Arbitration mode. Used to select the arbitration policy for the slave port. This field is initialized by hardware reset. 00 Fixed priority using MPR 01 Round-robin priority 10 Invalid value 11 Invalid value
24–25	Reserved, must be cleared.
26–27 PCTL	Parking control. Used to select the parking algorithm used by the slave port. This field is initialized by hardware reset. 00 When no master is making a request, the arbiter parks the slave port on the master port defined by the PARK control field. 01 POL—Park on last. When no master is making a request, the arbiter parks the slave port on the last master to own the slave port. 10 LPP—Low-power park. When no master is making a request, the arbiter parks the slave port on no master and drives all slave port outputs to a safe state. 11 Invalid value

Table 7-5. XBAR_SGPCR*n* Field Descriptions (continued)

Field	Description
28	Reserved, must be cleared.
29–31 PARK	Park. Used to determine which master port this slave port parks on when no masters are actively making requests. PCTL must be set to 00. 000 Park on master port 0 001 Park on master port 1 010 Park on master port 2 011 Invalid value 100 Park on master port 4 101 Invalid value 110 Invalid value 111 Invalid value

7.3 Functional Description

This section describes the functionality of the XBAR in more detail.

7.3.1 Overview

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate concurrently with multiple slaves. To maximize data throughput, it is essential to keep arbitration delays to a minimum.

This section examines data throughput from the point of view of masters and slaves, detailing when the XBAR stalls masters, or inserts bubbles on the slave side.

7.3.2 General Operation

When a master accesses the XBAR from an idle master state, the access is taken immediately by the XBAR. If the targeted slave port of the access is available (that is, the requesting master is currently granted ownership of the slave port), the access is immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR by a granted master. If the targeted slave port of the access is busy or parked on a different master port, the requesting master receives wait states until the targeted slave port can service the master request. The latency in servicing the request depends on each master's priority level and the responding slave's access time.

Because the XBAR appears to be simply another slave to the master device, the master device has no indication that it owns the slave port it is targeting. While the master does not have control of the slave port it is targeting, it is wait-stated.

A master is given control of a targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has the following conditions:

- Outstanding request to slave port A that has a long response time
- Pending access to a different slave port B
- Lower priority master also makes a request to the different slave port B.

In this case, the lower priority master is granted bus ownership of slave port B after a cycle of arbitration, assuming the higher priority master slave port A access is not terminated.

After a master has control of the slave port it is targeting, the master remains in control of that slave port until it gives up the slave port by running an IDLE cycle, leaves that slave port for its next access, or loses control of the slave port to a higher priority master with a request to the same slave port. However, because all masters run a fixed-length burst transfer to a slave port, it retains control of the slave port until that transfer sequence is completed. In round-robin arbitration mode, the current master is forced to hand off bus ownership to an alternately requesting master at the end of its current transfer sequence.

When a slave bus is idled by the XBAR, it can be parked on the master port using the PARK bits in the XBAR_SGPCR (slave general-purpose control register), or on the last master to have control of the slave port. This can avoid the initial clock of the arbitration delay if the master must arbitrate to gain control of the slave port. The slave port can also be put into low-power park mode to save power.

7.3.3 Master Ports

The XBAR terminates an access and it is not allowed to pass through the XBAR unless the master currently is granted access to the slave port to which the access is targeted. A master access is taken if the slave port to which the access decodes is either currently servicing the master or is parked on the master. In this case, the XBAR is completely transparent and the master access is immediately transmitted on the slave bus and no arbitration delays are incurred. A master access stalls if the access decodes to a slave port that is busy serving another master, parked on another master or is in low-power park mode.

If the slave port is currently parked on another master or is in low-power park mode, and no other master is requesting access to the slave port, then only one clock of arbitration is incurred. If the slave port is currently serving another master of a lower priority and the master has a higher priority than all other requesting masters, then the master gains control over the slave port as soon as the data phase of the current access is completed. If the slave port is currently servicing another master of a higher priority, then the master gains control of the slave port after the other master releases control of the slave port if no other higher priority master is also waiting for the slave port.

A master access is responded to with an error if the access decodes to a location not occupied by a slave port. This is the only time the XBAR directly responds with an error response. All other error responses received by the master are the result of error responses on the slave ports being passed through the XBAR.

7.3.4 Slave Ports

The goal of the XBAR with respect to the slave ports is to keep them 100% saturated when masters are actively making requests. To do this the XBAR must not insert any bubbles onto the slave bus unless absolutely necessary.

There is only one instance when the XBAR forces a bubble onto the slave bus when a master is actively making a request. This occurs when a handoff of bus ownership occurs and there are no wait states from the slave port. A requesting master which does not own the slave port is granted access after a one clock delay.

The only other time the XBAR has control of the slave port is when no masters are making access requests to the slave port and the XBAR is forced to either park the slave port on a specific master, or place the slave port into low-power park mode. In these cases, the XBAR forces IDLE for the transfer type.

7.3.5 Priority Assignment

Each master port must be assigned a unique 2-bit priority level in fixed priority mode. If multiple master ports are assigned the same priority level within a register (XBAR_MPR) undefined behavior results.

7.3.6 Arbitration

XBAR supports two arbitration schemes; a simple fixed-priority comparison algorithm, and a round-robin fairness algorithm. The arbitration scheme is independently programmable for each slave port.

7.3.6.1 Fixed Priority Operation

When operating in fixed-priority arbitration mode, each master is assigned a unique priority level in the XBAR_MPR. If two masters both request access to a slave port, the master with the highest priority in the selected priority register gains control over the slave port.

Any time a master makes a request to a slave port, the slave port checks to see if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (if any). The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

If the new requesting master's priority level is higher than that of the master that currently has control of the slave port, the higher priority master is granted control at the termination of any currently pending access, assuming the pending transfer is not part of a burst transfer.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. But if the new requesting master's priority level is lower than that of the master that currently has control of the slave port, the new requesting master is forced to wait until the master that currently has control of the slave port is finished accessing the current slave port.

7.3.6.2 Round-Robin Priority Operation

When operating in round-robin mode, each master is assigned a relative priority based on the master port number. This relative priority is compared to the port number of the last master to perform a transfer on the slave bus. The highest priority requesting master becomes the owner of the slave bus at the next transfer boundary (accounting for fixed-length burst transfers). Priority is based on how far ahead the port number of the requesting master is to the port number of the last master.

After granted access to a slave port, a master can perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line is granted access to the slave port when the current transfer is completed, or possibly on the next clock cycle if the current master has no pending access request.

As an example of arbitration in round-robin mode, assume the three masters have ID's 0, 1, and 2. If the last master of the slave port was master 1, and masters 0 and 2 make simultaneous requests, they are serviced in the order 2 and then 0 assuming no further requests are made.

As another example, if master 1 is waiting on a response from a slow slave and has no further pending access to that slave, no other masters are requesting, and master 0 then makes a request, master 0's request is granted on the next clock (assuming that master 1's transfer is not a burst transfer), and the request information for master 0 is driven to the slave as a pending access. If master 2 were to make a request after master 0 has been granted access, but prior to master 0's access being accepted by the slave, master 0 maintains the grant on the slave port, and master 2 is delayed until the next arbitration boundary, which occurs after the transfer is complete. The round-robin pointer is reset to 0, so if master 1 has another request that occurs before master 0's transfer completes, master 1 is the granted the bus. This implies a worst case latency of *N* transfers for a system with *N* masters.

Parking can continue to be used in round-robin mode, but affects the round-robin pointer unless the parked master actually performs a transfer. Handoff to the next master in line occurs after one cycle of arbitration.

The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. If the new requesting master's priority level is lower than that of the master that currently has control of the slave port, the new requesting master is forced to wait until the master that currently has control of the slave port completes its access.

7.3.6.2.1 Parking

If no master is currently requesting the slave port, the slave port is parked. The slave port parks in one of three places, indicated by the value of the PCTL field in the XBAR_SGPCR.

- *If park-on-specific master mode is selected,* the slave port parks on the master designated by the PARK field. When the master accesses the slave port again, a one clock arbitration penalty is incurred only for an access request made by another master port to the slave port. No other arbitration penalties are incurred. All other masters pay a one clock penalty.
- *If park-on-last (POL) mode is selected,* then the slave port parks on the last master to access it, passing that master's signals through to the slave bus. When the master accesses the slave port again, no other arbitration penalties are incurred except that a one clock arbitration penalty is incurred for each access request to the slave port made by another master port. All other masters pay a one clock penalty.
- *If the low-power-park (LPP) mode is selected,* then the slave port enters low-power park mode. It is not under control by any master and does not transmit any master signals to the slave bus. All slave bus activity halts because all slave bus signals are not toggling. This saves power if the slave port is not used for some time. However, when a master does make a request to a slave port parked in low-power-park, a one clock arbitration delay is incurred to get ownership of the slave port.

Chapter 8
Error Correction Status Module (ECSM)

This device includes error-correcting code (ECC) implementations to improve the quality and reliability of internal SRAM and internal flash memories. The error correction status module (ECSM) allows the application to collect data on memory errors reported by the ECC or generic access error information.

8.1 Overview

The ECSM provides a set of registers that configure and report ECC errors for the device, including accesses to SRAM and flash memory. The types of memory are:

- SRAM—32 data bits plus seven check bits for every 32-bit word
- Flash—64 data bits plus eight check bits for every 64-bit doubleword

The application must:

1. Configure the ECC for the types of memory errors to report.
2. Initialize internal SRAM by performing write operations to the entire SRAM memory before enabling the ECC. Flash memory does not require this step. See Section 8.3, “Initialization and Application Information.”
3. Query a set of read-only status and information registers to identify ECC errors, in response to an enabled ECC error interrupt.

8.1.1 Types of ECC Errors

The ECSM is configurable for reporting non-correctable errors, and has registers for capturing ECC information for internal SRAM and flash access errors. The types of ECC errors are:

- Correctable error—A correctable ECC error is generated when only one bit is incorrect in the data and ECC check bits. In this case, the bit in error is corrected automatically by hardware, and no flags or other indicators are set by the error that occurred.
- Non-correctable error—An ECC non-correctable error is generated when two or more bits in the data and ECC check bits are incorrect. The bus transaction which caused the memory access to produce the non-correctable error terminates with a bus error. If correctly enabled in the ECSM module, non-correctable ECC errors can generate an interrupt and capture additional error details about the access in ECSM registers.

8.1.2 ECC Operations

ECCs are calculated across the entire width of the data field. The memory controller (flash or SRAM) checks for ECC errors on read accesses, and calculates the ECC check bits on write accesses. ECC operations for system RAM differ depending on the operation and memory:

- Read operations are comparable for SRAM and flash memory
- Write operations for SRAM and flash memory have major differences

Flash memory writes are program events and are managed entirely by the flash memory module. See Chapter 13, “Flash Memory” for information on flash memory write operations. The following write operations apply to SRAM only.

Read operation—SRAM and flash memory:

1. Read the data bits that contain the desired byte, halfword, word or doubleword from memory.
2. Calculate the syndrome from the read data and the check bits to determine if a correctable or non-correctable error is present.
3. Return the data bits, error-free or corrected data, to the requesting bus master; or respond with an error termination, and assert an interrupt if necessary.

32-bit write operation—SRAM only:

1. Generate the check bits based on the 32 data bits to be written.
2. Write the 32 data bits plus the check bits to memory.

Eight- or 16-bit write operations—SRAM only:

1. Read the data bits that contain the desired byte or halfword from memory.
2. Perform a read ECC check.
   a) If the read operation is error-free, go to Step 3.
   b) If a correctable single-bit error is detected, forward the corrected data to Step 3.
   c) If a non-correctable error is detected, the write operation is not performed and the bus cycle terminates with an error.
3. Merge the write data with the 32-bit read data (error-free or corrected data).
4. Generate the check bits for the destination write data.
5. Write the 32 bits of destination write data plus the check bits to memory.

8.2 Memory Map and Register Definition

Table 8-1 is the memory map for the ECSM registers.

Table 8-1. ECSM Memory Map

Address	Register Name	Register Description	Bits
Base (0xFFF4_0000) + 0x0016	ECSM_SWTCR	Software watchdog timer control register [1]	16
Base + (0x0018–0x001A)	—	Reserved	—
Base + 0x001B	ECSM_SWTSR	Software watchdog timer service register [1]	8
Base + (0x001C–0x001E)	—	Reserved	—
Base + 0x001F	ECSM_SWTIR	Software watchdog timer interrupt register [1]	8
Base + (0x0020–0x0042)	—	Reserved	—
Base + 0x0043	ECSM_ECR	ECC configuration register	8
Base + (0x0044–0x0046)	—	Reserved	—
Base + 0x0047	ECSM_ESR	ECC status register	8
Base + (0x0048–0x0049)	—	Reserved	—
Base + 0x004A	ECSM_EEGR	ECC error generation register	16
Base + (0x004B–0x004F)	—	Reserved	—
Base + 0x0050	ECSM_FEAR	Flash ECC address register	32
Base + (0x0054–0x0055)	—	Reserved	—
Base + 0x0056	ECSM_FEMR	Flash ECC master register	8
Base + 0x0057	ECSM_FEAT	Flash ECC attribute register	8
Base + 0x0058	ECSM_FEDRH	Flash ECC data high register	32
Base + 0x005C	ECSM_FEDRL	Flash ECC data low register	32
Base + 0x0060	ECSM_REAR	SRAM ECC address register	32
Base + (0x0064–0x0065)	—	Reserved	—
Base + 0x0066	ECSM_REMR	SRAM ECC master register	8
Base + 0x0067	ECSM_REAT	SRAM ECC attributes register	8
Base + 0x0068	ECSM_REDRH	SRAM ECC data high register	32
Base + 0x006C	ECSM_REDRL	SRAM ECC data low register	32
Base + (0x0070–0x007F)	—	Reserved	—

[1] These registers control and configure the software watchdog timer, and are included as part of a standard Freescale ECSM module. Use the core watchdog functions to implement watchdog capabilities rather than these registers.
See Section 8.2.1.1, “Software Watchdog Timer Registers: Control, Service, and Interrupt (ECSM_SWTCR, ECSM_SWTSR, and ECSM_SWTIR).”

8.2.1 Register Descriptions

The following limitations apply to ECC accesses:

- Attempted accesses to reserved addresses result in an error termination.
- Attempted writes to read-only registers are ignored and do not terminate with an error.
- Writes to the programming model must match the size of the register unless noted otherwise; for example, an *n*-bit register only supports *n*-bit writes. Attempted writes greater or less than the register width produce an error termination of the bus cycle and no change to the targeted register.

8.2.1.1 Software Watchdog Timer Registers: Control, Service, and Interrupt (ECSM_SWTCR, ECSM_SWTSR, and ECSM_SWTIR)

The core provides watchdog functions for flexible watchdog implementation. Use the core watchdog functions to optimize code portability to other Power Architecture-based products in the MPC5500 family. See the core reference manual for information on the core watchdog timer functions.

These ECSM read-only registers control and configure the ESCM software watchdog timer that is included as part of the Freescale standard for this device:

Table 8-2. ECSM Register Domains

Domain	ECSM Register Name	Register Purpose
Software Watchdog	ECSM_SWTCR	ECSM Watchdog Timer Control
	ECSM_SWTSR	ECSM Watchdog Timer Service
	ECSM_SWTIR	ECSM Watchdog Timer Interrupt

NOTE

DO NOT change the reset values in the ECSM software watchdog registers. Any change to the reset values can cause an ECSM_SWTIR_SWTIC interrupt.

8.2.1.2 ECC Registers

The ECSM registers in Table 8-3 are visible to application software, and allow you to configure the data reported and log ECC memory failures.

Table 8-3. ECSM Register Domains

Domain	ECSM Register Name	Register Purpose
Global Reporting	ECSM_ECR	ECC configuration
	ECSM_ESR	ECC status
	ECSM_EEGR	ECC error generation

Table 8-3. ECSM Register Domains (continued)

Domain	ECSM Register Name	Register Purpose
Flash Reporting	ECSM_FEAR	Flash ECC address
	ECSM_FEMR	Flash ECC master number
	ECSM_FEAT	Flash ECC attributes
	ECSM_FEDR	Flash ECC data
SRAM Reporting	ECSM_REAR	SRAM ECC address
	ECSM_REMR	SRAM ECC master number
	ECSM_REAT	SRAM ECC attributes
	ECSM_REDR	SRAM ECC data

The details of each ECC register are described in the following sections.

8.2.1.3 ECC Configuration Register (ECSM_ECR)

ECSM_ECR is an 8-bit control register that enables or disables ECC error reporting during internal SRAM and flash accesses. In addition to the interrupt generation, the ECSM captures specific information (memory address, attributes and data, bus master number, etc.) that is useful for failure analysis.

The ECC reporting logic can detect non-correctable memory errors. When a non-correctable error terminates the current access to the memory (flash or SRAM), an error condition is generated. In many cases, the error termination is reported directly by the initiating bus master.

Base (0xFFF4_0000) + 0x0043 Access: R/W

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	ERNCR	EFNCR
W								
Reset	0	0	0	0	0	0	0	0

Figure 8-1. ECC Configuration Register (ECSM_ECR)

The following table describes the fields in the error configuration register:

Table 8-4. ECSM_ECR Field Definitions

Field	Description
0–5	Reserved
6 ERNCR	Enable internal SRAM non-correctable reporting. When this bit is set (enabled), a non-correctable multi-bit internal SRAM error sets the ECSM_ESR[RNCE] bit in the ECC status register, which generates an ECSM ECC internal SRAM interrupt. The faulting address, attributes and data are also captured in the REAR, REMR, REAT and REDR registers. 0 Reporting of non-correctable internal SRAM errors is disabled. 1 Reporting of non-correctable internal SRAM errors is enabled.
7 EFNCR	Enable flash non-correctable reporting. When this bit is set (enabled), a non-correctable multi-bit flash error sets the ECSM_ESR[FNCE] bit in the ECC status register, which generates an ECSM ECC flash interrupt. The faulting address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. 0 Reporting of non-correctable flash errors is disabled. 1 Reporting of non-correctable flash errors is enabled.

8.2.1.4 ECC Status Register (ECSM_ESR)

The ECC status register (ECSM_ESR) is an 8-bit control register that defines the types of ECC events detected. The ESR indicates the last, correctly-enabled memory event detected. The ECSM ECC interrupt request is generated as defined by the boolean equation:

```
ECSM_ECC_IRQ
    = ECSM_ECR[ERNCR] & ECSM_ESR[RNCE]     // ram,   noncorrectable error
    | ECSM_ECR[EFNCR] & ECSM_ESR[FNCE]     // flash, noncorrectable error
```

where the combination of the following criteria generates the interrupt request:

- Correctly enabled category in the ECSM_ECR; and
- Condition in the ECSM_ESR detected.

The ECSM allows a maximum of one bit of the ECSM_ESR to assert at any given time. This preserves the relationship of the ECSM_ESR to the address and attribute registers, which are loaded for each enabled ECC event. If an ECC interrupt is pending and another enabled ECC event occurs, the ECSM hardware automatically performs ECSM_ESR reporting by clearing the previous data, and then loading the new status, which ensures that only a single flag is asserted.

To maintain a coherent software view of the reported event, use the following sequence in the ECSM error interrupt service routine:

1. Read the ECSM_ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ECSM_ESR and verify the current contents matches the original contents. If the two values differ, return to step 1 and repeat this sequence.
4. When the values are identical, write a 1 to the asserted ECSM_ESR flag to negate the interrupt request.

If multiple status flags are detected simultaneously, the ECSM records the higher priority SRAM non-correctable error (RNCE) events before flash non-correctable error (FNCE) events.

Base + 0x0047 Access: R/W1c

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	RNCE	FNCE
W							w1c	w1c
Reset	0	0	0	0	0	0	0	0

Figure 8-2. ECC Status Register (ECSM_ESR)

Table 8-5. ECSM_ESR Field Definitions

Field	Description
0–5	Reserved
6 RNCE	SRAM non-correctable error. A non-correctable SRAM error occurs, generates an ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the REAR, REMR, REAT and REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable non-correctable SRAM error has been detected. 1 A reportable non-correctable SRAM error has been detected.
7 FNCE	Flash non-correctable error. The occurrence of a correctly-enabled non-correctable flash error generates an ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable non-correctable flash error has been detected. 1 A reportable non-correctable flash error has been detected.

8.2.1.5 ECC Error Generation Register (ECSM_EEGR)

The ECSM_EEGR is a 16-bit control register used to generate double-bit data errors in internal SRAM. This allows you to test the software service routines for memory error logging. By generating errors during data write cycles, subsequent reads of the corrupt address locations generate ECC events, such as double-bit noncorrectable errors that are terminated with an error response.

If an attempt to force a non-correctable error (by asserting ECSM_EEGR[FRCNCI] or ECSM_EEGR[FRC1NCI]) and the ECSM_EEGR[ERRBIT] equals 64, then no data error is generated.

NOTE

Only values {0,0}, {1,0} and {0,1} are allowed for the two control bit enables {FRCNCI, FR1NCI}. The value {1,1} causes undefined results.

Base + 0x004A Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FRC NCI	FR1 NCI	0	ERRBIT[0:6]						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-3. ECC Error Generation (ECSM_EEGR) Register

Table 8-6. ECSM_EEGR Field Definitions

Field	Description
0–5	Reserved
6 FRCNCI	Force internal SRAM continuous noncorrectable data errors. 0 No internal SRAM continuous 2-bit data errors are generated. 1 2-bit data errors in the internal SRAM are continuously generated. When this bit is cleared: 1. The RAM controller generates a normal ECC 2. The polarity of the bit position specified in ERRBIT plus the overall odd parity bit are inverted to introduce a 2-bit ECC error in internal SRAM. When this bit is set: 1. The internal SRAM controller generates 2-bit data errors, as defined by the bit position specified in ERRBIT[0:6] and the overall odd parity bit, on every write operation.
7 FR1NCI	Force internal SRAM one noncorrectable data errors. 0 No internal SRAM single 2-bit data errors are generated. 1 One 2-bit data error in internal SRAM is generated. When this bit is cleared, the RAM controller generates a normal ECC, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the internal SRAM. When this bit is set, the internal SRAM controller generates 2-bit data errors, as defined by the bit position specified in ERRBIT[0:6] and the overall odd parity bit, on every write operation. The assertion of this bit forces the internal SRAM controller to create one 2-bit data error, as defined by the bit position specified in ERRBIT[0:6] and the overall odd parity bit, on the first write operation after this bit is set. The normal ECC generation takes place in the internal SRAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the internal SRAM. After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to correctly re-enable the error generation logic.
8	Reserved
9–15 ERRBIT	Error bit position. Defines the bit position which is complemented to create the data error on the write operation. The bit specified by this field plus the odd parity bit of the ECC code are inverted. The internal SRAM controller follows a vector bit ordering scheme where LSB=0. Errors in the ECC check bits can be generated by setting this field to a value greater than the internal SRAM width. The following association between the ERRBIT field and the corrupted memory bit is defined: if ERRBIT = 0, then internal SRAM[0] is inverted if ERRBIT = 1, then internal SRAM[1] is inverted ... if ERRBIT = 31, then internal SRAM[31] is inverted if ERRBIT = 32, then ECC Parity[0] is inverted if ERRBIT = 33, then ECC Parity[1] is inverted ... if ERRBIT = 39, then ECC Parity[7] is inverted For ERRBIT values greater than 39, no bit position is inverted.

8.2.1.6 Flash ECC Address Register (ECSM_FEAR)

The ECSM_FEAR is a 32-bit register for capturing the address of the last, correctly-enabled ECC event in the flash memory. Depending on the state of the ECSM_ECR, an ECC event in the flash loads the address, attributes and data of the access into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT, and ECSM_FEDR registers, and asserts the F1BC or FNCE flag in ECSM_ESR.

The address that is captured in ECSM_FEAR is the flash page address as seen on the system bus. See Section 13.3.2.7, “Address Register FLASH_AR” to retrieve the doubleword address.

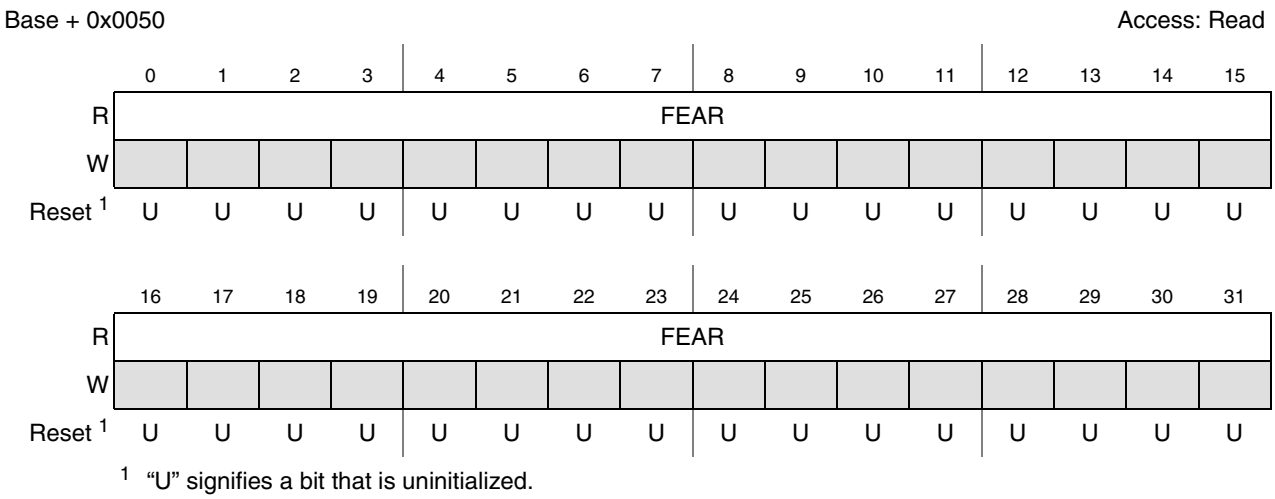


Figure 8-4. Flash ECC Address Register (ECSM_FEAR)

Table 8-7. ECSM_FEAR Field Descriptions

Field	Description
0–31 FEAR [0:31]	Flash ECC address. Contains the faulting access address of the last, correctly-enabled flash ECC event.

8.2.1.7 Flash ECC Master Number Register (ECSM_FEMR)

The FEMR is an 8-bit register for capturing the XBAR bus master number of the last, correctly-enabled ECC event in the flash memory. Depending on the state of the ECSM_ECR, an ECC event in the flash loads the address, attributes and data of the access into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers, and asserts the FNCE flag in the ECSM_ESR.

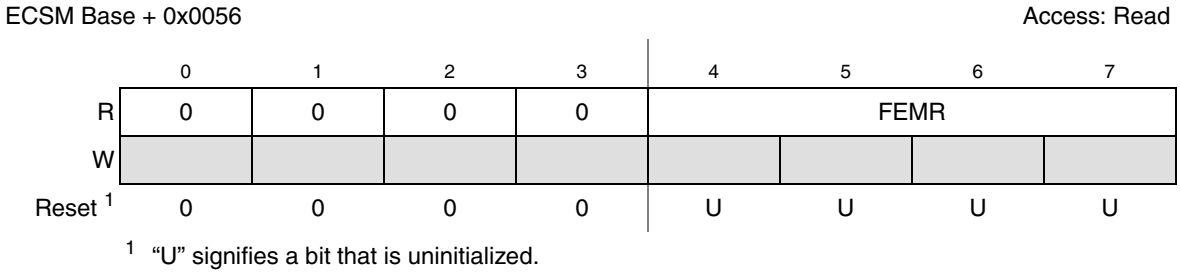


Figure 8-5. Flash ECC Master Number Register (ECSM_FEMR)

Table 8-8. ECSM_FEMR Field Descriptions

Field	Description
0–3	Reserved
4–7 FEMR [0:3]	Flash ECC master number. Contains the XBAR bus master number of the faulting access of the last, correctly-enabled flash ECC event. The reset value of this field is undefined.

8.2.1.8 Flash ECC Attributes Register (ECSM_FEAT)

The ECSM_FEAT is an 8-bit register for capturing the XBAR bus master attributes of the last, correctly-enabled ECC event in the flash memory. Depending on the state of the ECSM_ECR register, an ECC event in the flash loads the address, attributes and data of the access into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT, and ECSM_FEDR registers, and asserts the FNCE flag in ECSM_ESR.

Base + 0x0057 Access: Read

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROT0	PROT1	PROT2	PROT3
W								
Reset [1]	U	U	U	U	U	U	U	U

[1] "U" signifies a bit that is uninitialized.

Figure 8-6. Flash ECC Attributes Register (ECSM_FEAT)

Table 8-9. ECSM_FEAT Field Descriptions

Field	Description
0 WRITE	Write. The reset value of this field is undefined. 0 System bus read access 1 System bus write access
1–3 SIZE [0:2]	Size. The reset value of this field is undefined. 000 8-bit System bus access 001 16-bit System bus access 010 32-bit System bus access 011 Reserved 1xx Reserved
4 PROT0	Protection: cache. The reset value of this field is undefined. 0 Non-cacheable 1 Cacheable
5 PROT1	Protection: buffer. The reset value of this field is undefined. 0 Non-bufferable 1 Bufferable

Table 8-9. ECSM_FEAT Field Descriptions (continued)

Field	Description
6 PROT2	Protection: mode. The reset value of this field is undefined. 0 User mode 1 Supervisor mode
7 PROT3	Protection: type. The reset value of this field is undefined. 0 I-Fetch 1 Data

8.2.1.9 Flash ECC Data High Register (ECSM_FEDRH)

The ECSM_FEDRH and ECSM_FEDRL are 32-bit registers for capturing the data of the last, correctly-enabled ECC event in flash memory. Depending on the state of the ECSM_ECR, an ECC event in the flash loads the address, attributes and data of the access into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers, and asserts the FNCE flag in ECSM_ESR.

The data captured on a multi-bit non-correctable ECC error is undefined.

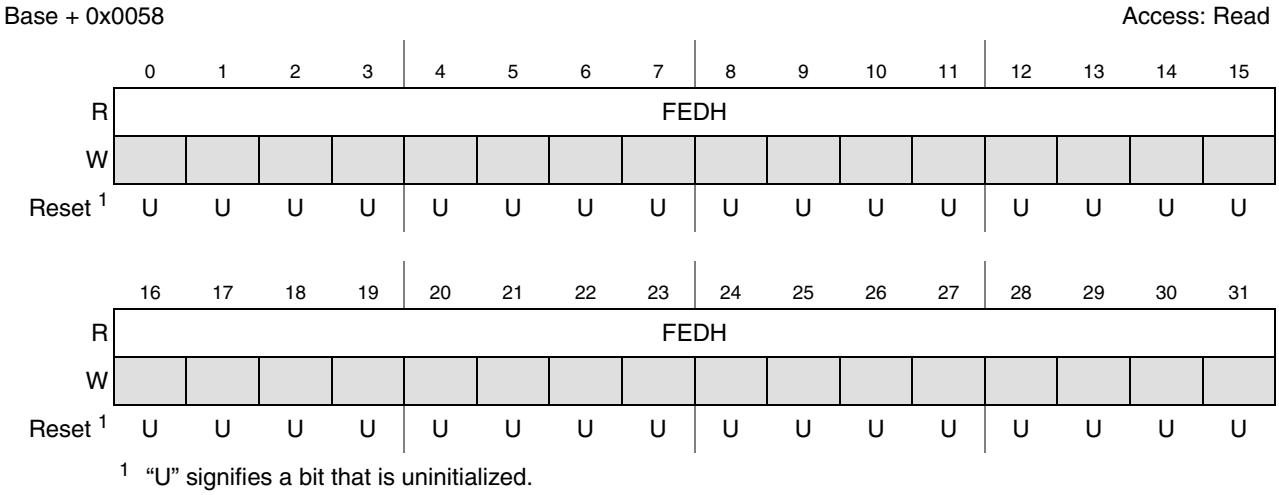


Figure 8-7. Flash ECC Data High Register (ECSM_FEDRH)

Table 8-10. ECSM_FEDRH Field Descriptions

Field	Description
0–31 FEDH [0:31]	Flash ECC data. Contains the data associated with the faulting access of the last, correctly-enabled flash ECC event. The register contains the data value taken directly from the data bus. The reset value of this field is undefined.

8.2.1.10 Flash ECC Data Low Registers (ECSM_FEDRL)

The ECSM_FEDRH and ECSM_FEDRL are 32-bit registers for capturing the data of the last, correctly-enabled ECC event in the flash memory. Depending on the state of the ECSM_ECR, an ECC event in the flash loads the address, attributes and data of the access into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers, and asserts the FNCE flag in ECSM_ESR.

The data captured on a multi-bit non-correctable ECC error is undefined.

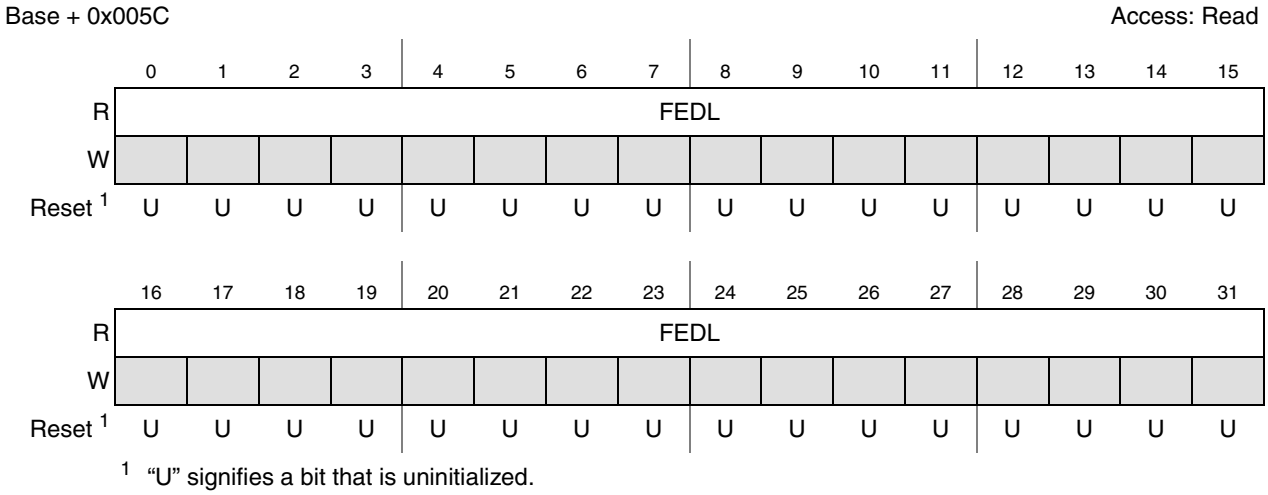


Figure 8-8. Flash ECC Data Low Register (ECSM_FEDRL)

Table 8-11. ECSM_FEDRL Field Descriptions

Field	Description
0–31 FEDL [0:31]	Flash ECC data. Contains the data associated with the faulting access of the last, correctly-enabled flash ECC event. The register contains the data value taken directly from the data bus. The reset value of this field is undefined.

8.2.1.11 SRAM ECC Address Register (ECSM_REAR)

The ECSM_REAR is a 32-bit register for capturing the address of the last, correctly-enabled ECC event in the RAM memory. Depending on the state of the ECSM_ECR, an ECC event in the RAM loads the address, attributes and data of the access into the ECSM_REAR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and asserts the RNCE flag in ECSM_ESR.

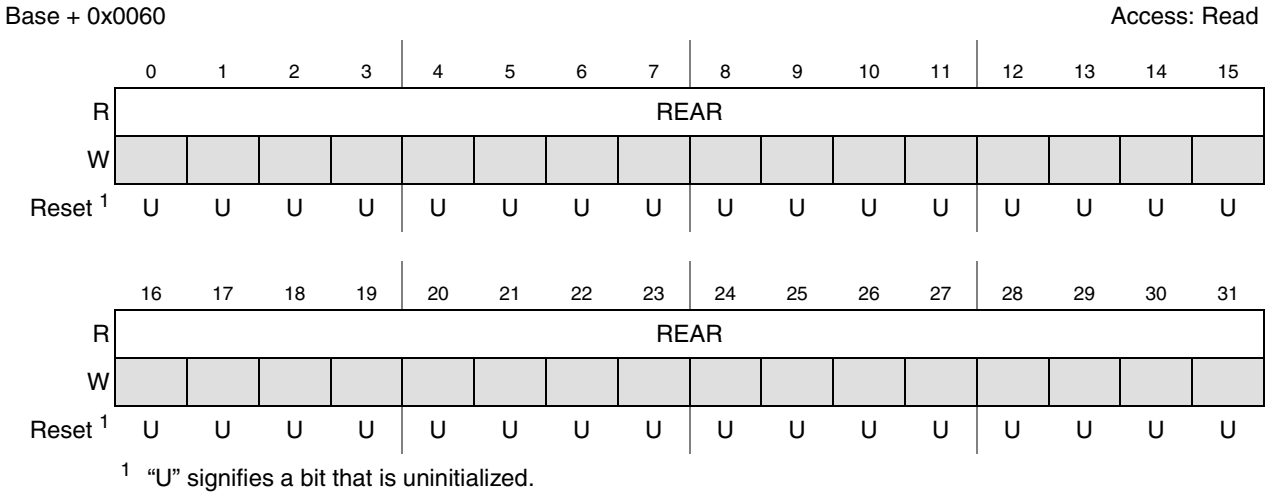


Figure 8-9. RAM ECC Address Register (ECSM_REAR)

Table 8-12. ECSM_REAR Field Descriptions

Field	Description
0–31 REAR [0:31]	SRAM ECC address. Contains the faulting access address of the last, correctly-enabled RAM ECC event. The reset value of this field is undefined.

8.2.1.12 SRAM ECC Master Number Register (ECSM_REMR)

The REMR is an 8-bit register for capturing the XBAR bus master number of the last, correctly-enabled ECC event in the RAM memory. Depending on the state of the ECSM_ECR, an ECC event in the SRAM loads the address, attributes and data of the access into the ECSM_REAR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and asserts the RNCE flag in ECSM_ESR.

Base + 0x0066 Access: Read

	0	1	2	3	4	5	6	7
R	0	0	0	0	REMR			
W								
Reset [1]	0	0	0	0	U	U	U	U

[1] "U" signifies a bit that is uninitialized.

Figure 8-10. RAM ECC Master Number Register (ECSM_REMR)

Table 8-13. ECSM_REMR Field Descriptions

Field	Description
0–3	Reserved
4–7 REMR [0:3]	SRAM ECC master number. Contains the XBAR bus master number of the faulting access of the last, correctly-enabled RAM ECC event. The reset value of this field is undefined.

8.2.1.13 SRAM ECC Attributes Register (ECSM_REAT)

The ECSM_REAT is an 8-bit register for capturing the XBAR bus master attributes of the last, correctly-enabled ECC event in the RAM memory. Depending on the state of the ECSM_ECR, an ECC event in the RAM loads the address, attributes and data of the access into the ECSM_REAR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and asserts the RNCE flag in ECSM_ESR.

Base + 0x0067 Access: Read

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROT0	PROT1	PROT2	PROT3
W								
Reset [1]	U	U	U	U	U	U	U	U

[1] "U" signifies a bit that is uninitialized.

Figure 8-11. SRAM ECC Attributes Register (ECSM_REAT)

Table 8-14. ECSM_REAT Field Descriptions

Field	Description
0 WRITE	Write. The reset value of this field is undefined. 0 System bus read access 1 System bus write access
1–3 SIZE [0:2]	Size. The reset value of this field is undefined. 000 8-bit system bus access 001 16-bit system bus access 010 32-bit system bus access 011 Reserved 1xx Reserved
4 PROT0	Protection: cache. The reset value of this field is undefined. 0 Non-cacheable 1 Cacheable
5 PROT1	Protection: buffer. The reset value of this field is undefined. 0 Non-bufferable 1 Bufferable
6 PROT2	Protection: mode. The reset value of this field is undefined. 0 User mode 1 Supervisor mode
7 PROT3	Protection: type. The reset value of this field is undefined. 0 Fetch 1 Data

8.2.1.14 SRAM ECC Data High Register (ECSM_REDRH)

The ECSM_REDRH and ECSM_REDRL are 32-bit registers for capturing the data of the last, correctly-enabled ECC event in the RAM memory. Depending on the state of the ECSM_ECR, an ECC event in the RAM loads the address, attributes and data of the access into the ECSM_REAR, ECSM_REMR, ECSM_REAT, ECSM_REDRH and ECSM_REDRL registers, and asserts the RFNCE flag in ECSM_ESR.

The data captured on a multi-bit non-correctable ECC error is undefined.

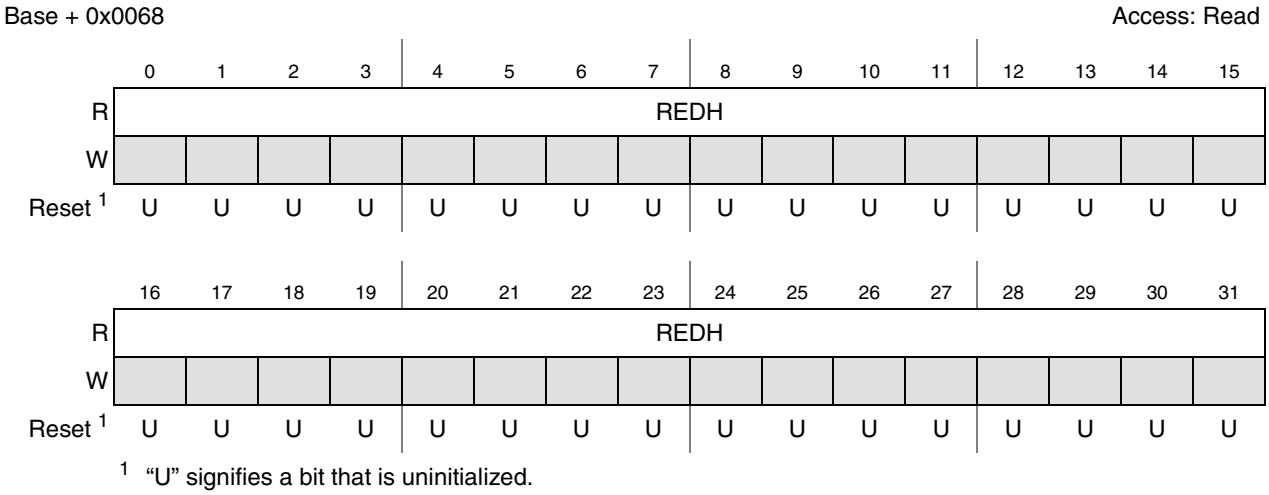


Figure 8-12. SRAM ECC Data High Register (ECSM_REDRH)

Table 8-15. ECSM_REDRH Field Descriptions

Field	Description
0–31 REDH [0:31]	RAM ECC data. Contains the data of the faulting access of the last, correctly-enabled RAM ECC event. The register contains the data value taken directly from the data bus. The reset value of this field is undefined.

8.2.1.15 SRAM ECC Data Low Registers (ECSM_REDRL)

The ECSM_REDRH and ECSM_REDRL are 32-bit registers for capturing the data for the last, correctly-enabled ECC event in RAM. Depending on the state of the ECSM_ECR, an ECC event in the RAM loads the address, attributes and data of the access to the following registers:

- ECSM_REAR
- ECSM_REMR
- ECSM_REAT
- ECSM_REDRH
- ECSM_REDRL

and asserts the RFNCE flag in ECSM_ESR. The data captured on a multi-bit non-correctable ECC error is undefined.

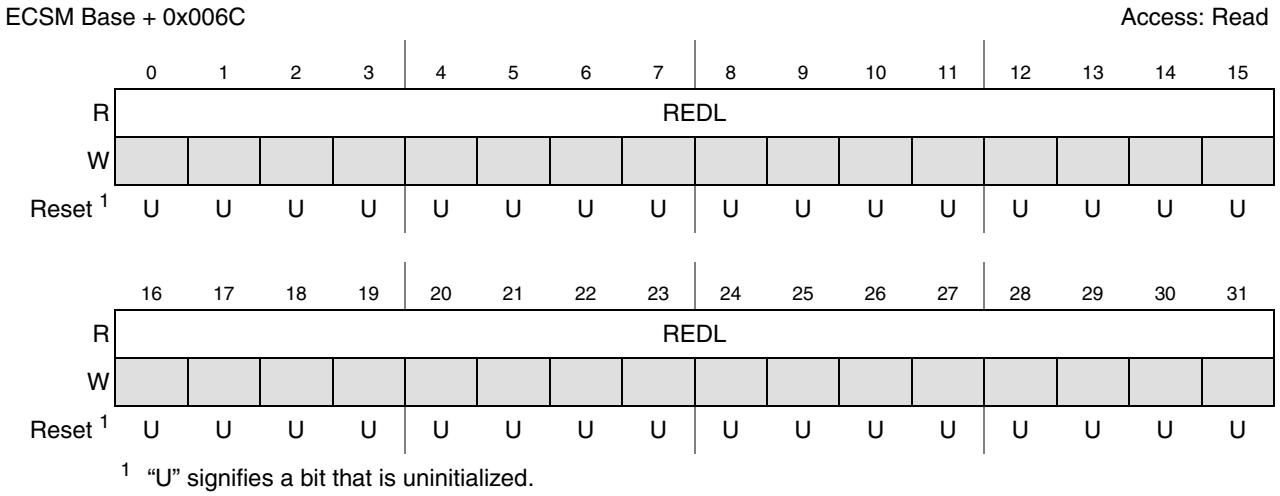


Figure 8-13. SRAM ECC Data Low Register (ECSM_REDRL)

The following table describes the RAM ECC data field in the

Table 8-16. ECSM_REDRL Field Descriptions

Field	Description
0–31 REDL [0:31]	RAM ECC data. Contains the data associated with the faulting access of the last, correctly-enabled RAM ECC event. The register contains the data value taken directly from the data bus. The reset value of this field is undefined.

8.3 Initialization and Application Information

The Error Correction Code (ECC) is used to verify the contents of the internal SRAM and flash memories. This is done by generating ECC check bits. Typically ECC check bits are calculated on writes and then used on reads to detect and correct errors.

- SRAM—Seven ECC check bits for each 32-bit SRAM data word.
- Flash—Eight ECC check bits for each 32-bit flash data word.

After Power on Reset (POR), the contents of internal SRAM is random and the corresponding ECC check bits are unknown. To prevent generating ECC errors during reads, an initialization routine must perform 32-bit writes to all SRAM locations. Because the flash module is non-volatile, the ECC check bits are calculated and stored when the flash is programmed.

Transparent to the application, the ECC uses the check bits to automatically correct single-bit memory errors. Multi-bit memory errors are not correctable. If the ECC detects a multi-bit error, an exception is generated. The type of exception generated by a multi-bit error depends on the settings of the EE and ME in the Machine State Register (MSR), as shown in Table 8-17. When error reporting is enabled, as long as its priority is 0, an interrupt request is generated to the interrupt controller (INTC) even though the INTC request is not serviced.

Table 8-17. MSR[EE] and MSR[ME] Bit Settings

Field	Description
EE	External interrupt enable. 0 External input interrupts disabled. 1 External interrupts enabled.
ME	Machine check enable. 0 Machine check interrupts disabled. Enters machine check. 1 Machine interrupts enabled.

A non-correctable data ECC error executes one of the following actions, regardless of whether non-correctable reporting is enabled:

Table 8-18. Non-correctable Data ECC States

MSR[EE]	MSR[ME]	Access Type	Result
0	0	Instruction or data	Enters checkstop state. A reset is required to resume processing.
0	1	Instruction or data	Machine check interrupt (IVOR1).
1	X	Data	Data storage interrupt (IVOR2). External interrupt must be enabled. Machine check can be enabled or disabled.
1	X	Instruction	Instruction storage interrupt (IVOR3).

When the device is in the checkstop state, processing is suspended and cannot resume without a reset. When a debug request is presented to the core while it is in the checkstop state, the core temporarily exits the checkstop state and enters debug mode. When debug mode exits, the core re-enters the checkstop state.

If the external interrupt bit in the MSR is enabled, data or instruction stage interrupts are reported when the ECC errors are a result of CPU accesses, regardless of whether non-correctable reporting is enabled. ECC errors generated by other masters (eDMA, etc.) do not generate data or instruction storage exceptions, and the ECSM is used to report these errors. You must initialize the ECSM to enable non-correctable reporting with interrupt generation to detect and report ECC interrupts from the ECSM.

Error reporting details can be independently enabled for flash memory and SRAM. To enable non-correctable error reporting and save the error details for:

- SRAM—set the ERNCR bit in the ECSM Error Configuration Register (ECSM_ECR).
- Flash—set the EFNCR bit in ECSM_ECR.

When these bits are set and a non-correctable ECC error occurs, error information is recorded in other ECSM registers and an interrupt request is generated on vector 9 of the interrupt controller (INTC).

- CPU data access error—Generates data storage exception (IVOR2).
- CPU instruction access error—Generates instruction storage exception (IVOR3).
- Vector 9 of INTC enabled—Generates an external exception (IVOR4).

To prevent generating an ECSM interrupt in response to a non-correctable error:

- Enable non-correctable reporting in the ECSM.
- Ensure the external interrupt is disabled.
- Ensure that the INTC_PSR[PRI] value for the ECC error interrupt request is 0.

To use the detailed data or instruction storage exception information, design an exception handler that can determine:

- The destination that asserted the error, indicated by the value of the ESR[XTE] bit.
- The address of the corrupted instruction for an instruction storage exception (SRR0).
- The address where the error occurred for a data storage exception, indicated in the data exception address register (DEAR).

Chapter 9
Enhanced Direct Memory Access (eDMA)

9.1 Introduction

This chapter describes the enhanced Direct Memory Access (eDMA) controller, a second-generation module capable of performing complex data transfers with minimal intervention from a host processor.

The enhanced direct memory access (eDMA) controller hardware microarchitecture includes a DMA engine which performs source and destination address calculations, and the actual data movement operations, along with SRAM-based local memory containing the transfer control descriptors (TCD) for the channels.

Figure 9-1 is a block diagram of the eDMA module.

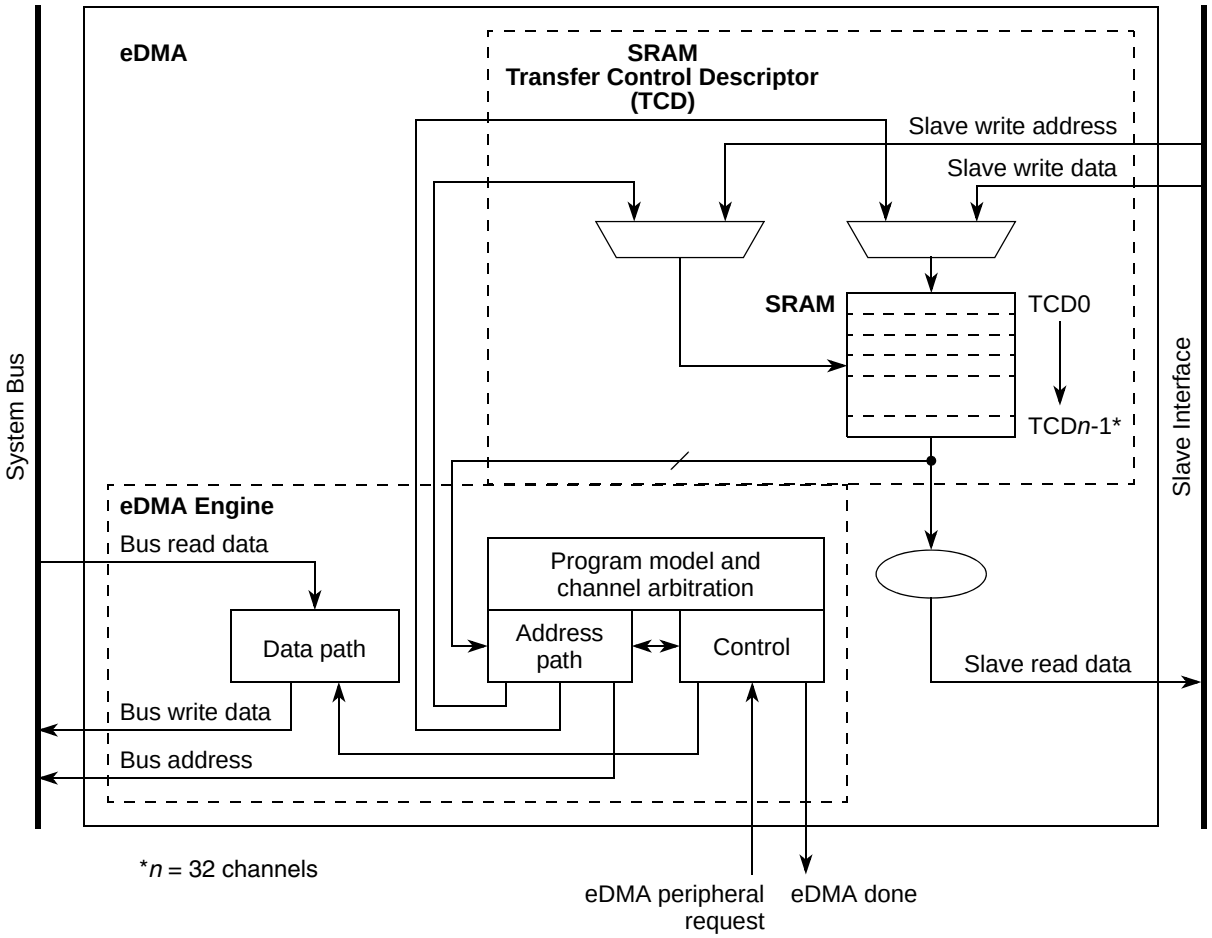


Figure 9-1. eDMA Block Diagram

9.1.1 Features

The eDMA is a highly-programmable data transfer engine, which has been optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known, and is *not* defined within the data packet itself. The eDMA module features:

- All data movement via dual-address transfers: read from source, write to destination

  Programmable source, destination addresses, transfer size, plus support for enhanced addressing modes
- 32-channel implementation performs complex data transfers with minimal intervention from a host processor
  - 32 bytes of data registers, used as temporary storage to support burst transfers (see SSIZE bit)
  - Connections to the crossbar switch for bus mastering the data movement
- Transfer control descriptor (TCD) organized to support two-deep, nested transfer operations
  - 32-byte TCD per channel stored in local memory
  - An inner data transfer loop defined by a minor byte transfer count
  - An outer data transfer loop defined by a major iteration count
- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continual transfers
  - Peripheral-paced hardware requests (one per channel)

NOTE

For all three methods, one activation per execution of the minor loop is required.

- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are enabled per channel, and logically summed together to form a single error interrupt
- Support for scatter/gather DMA processing
- Any channel can be programmed so that it can be suspended by a higher priority channel's activation, before completion of a minor loop

Throughout this chapter, *n* is used to reference the channel number. Additionally, data sizes are defined as byte (8-bit), halfword (16-bit), word (32-bit) and doubleword (64-bit).

9.1.2 Modes of Operation

9.1.2.1 Normal Mode

In normal mode, the eDMA is used to transfer data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

9.1.2.2 Debug Mode

If enabled by EDMA_CR[EDBG] and the CPU enters debug mode, the eDMA does not grant a service request when the debug input signal is asserted. If the signal asserts during a data block transfer as described by a minor loop in the current active channel's TCD, the eDMA continues the operation until the minor loop completes.

9.2 Memory Map and Register Definition

9.2.1 Memory Map

The eDMA programming model is partitioned into two regions:

Region 1 defines control registers; region 2 defines the local transfer control for the descriptor memory.

Table 9-1 is a 32-bit view of the eDMA memory map.

Table 9-1. eDMA 32-bit Memory Map

Address	Register Name	Register Description	Bits
Base (0xFFF4_4000)	EDMA_CR	eDMA control register	32
Base + 0x0004	EDMA_ESR	eDMA error status register	32
Base + 0x0008	—	Reserved	—
Base + 0x000C	EDMA_ERQRL	eDMA enable request low register	32
Base + 0x0010	—	Reserved	—
Base + 0x0014	EDMA_EEIRL	eDMA enable error interrupt low register	32
Base + 0x0018	EDMA_SERQR	eDMA set enable request register	8
Base + 0x0019	EDMA_CERQR	eDMA clear enable request register	8
Base + 0x001A	EDMA_SEEIR	eDMA set enable error interrupt register	8
Base + 0x001B	EDMA_CEEIR	eDMA clear enable error interrupt register	8
Base + 0x001C	EDMA_CIRQR	eDMA clear interrupt request register	8
Base + 0x001D	EDMA_CER	eDMA clear error register	8
Base + 0x001E	EDMA_SSBR	eDMA set start bit register	8
Base + 0x001F	EDMA_CDSBR	eDMA clear done status bit register	8
Base + 0x0020	—	Reserved	—

Table 9-1. eDMA 32-bit Memory Map (continued)

Address	Register Name	Register Description	Bits
Base + 0x0024	EDMA_IRQRL	eDMA interrupt request low register	32
Base + 0x0028	—	Reserved	—
Base + 0x002C	EDMA_ERL	eDMA error low register	32
Base + 0x0030– Base + 0x00FF	—	Reserved	—
Base + 0x0100	EDMA_CPR0	eDMA channel 0 priority register	8
Base + 0x0101	EDMA_CPR1	eDMA channel 1 priority register	8
Base + 0x0102	EDMA_CPR2	eDMA channel 2 priority register	8
Base + 0x0103	EDMA_CPR3	eDMA channel 3 priority register	8
Base + 0x0104	EDMA_CPR4	eDMA channel 4 priority register	8
Base + 0x0105	EDMA_CPR5	eDMA channel 5 priority register	8
Base + 0x0106	EDMA_CPR6	eDMA channel 6 priority register	8
Base + 0x0107	EDMA_CPR7	eDMA channel 7 priority register	8
Base + 0x0108	EDMA_CPR8	eDMA channel 8 priority register	8
Base + 0x0109	EDMA_CPR9	eDMA channel 9 priority register	8
Base + 0x010A	EDMA_CPR10	eDMA channel 10 priority register	8
Base + 0x010B	EDMA_CPR11	eDMA channel 11 priority register	8
Base + 0x010C	EDMA_CPR12	eDMA channel 12 priority register	8
Base + 0x010D	EDMA_CPR13	eDMA channel 13 priority register	8
Base + 0x010E	EDMA_CPR14	eDMA channel 14 priority register	8
Base + 0x010F	EDMA_CPR15	eDMA channel 15 priority register	8
Base + 0x0110	EDMA_CPR16	eDMA channel 16 priority register	8
Base + 0x0111	EDMA_CPR17	eDMA channel 17 priority register	8
Base + 0x0112	EDMA_CPR18	eDMA channel 18 priority register	8
Base + 0x0113	EDMA_CPR19	eDMA channel 19 priority register	8
Base + 0x0114	EDMA_CPR20	eDMA channel 20 priority register	8
Base + 0x0115	EDMA_CPR21	eDMA channel 21 priority register	8
Base + 0x0116	EDMA_CPR22	eDMA channel 22 priority register	8
Base + 0x0117	EDMA_CPR23	eDMA channel 23 priority register	8
Base + 0x0118	EDMA_CPR24	eDMA channel 24 priority register	8
Base + 0x0119	EDMA_CPR25	eDMA channel 25 priority register	8
Base + 0x011A	EDMA_CPR26	eDMA channel 26 priority register	8
Base + 0x011B	EDMA_CPR27	eDMA channel 27 priority register	8

Table 9-1. eDMA 32-bit Memory Map (continued)

Address	Register Name	Register Description	Bits
Base + 0x011C	EDMA_CPR28	eDMA channel 28 priority register	8
Base + 0x011D	EDMA_CPR29	eDMA channel 29 priority register	8
Base + 0x011E	EDMA_CPR30	eDMA channel 30 priority register	8
Base + 0x011F	EDMA_CPR31	eDMA channel 31 priority register	8
Base + 0x0120–0x0FFF	—	Reserved	—
Base + 0x1000	TCD00	eDMA transfer control descriptor 00	256
Base + 0x1020	TCD01	eDMA transfer control descriptor 01	256
Base + 0x1040	TCD02	eDMA transfer control descriptor 02	256
Base + 0x1060	TCD03	eDMA transfer control descriptor 03	256
Base + 0x1080	TCD04	eDMA transfer control descriptor 04	256
Base + 0x10A0	TCD05	eDMA transfer control descriptor 05	256
Base + 0x10C0	TCD06	eDMA transfer control descriptor 06	256
Base + 0x10E0	TCD07	eDMA transfer control descriptor 07	256
Base + 0x1100	TCD08	eDMA transfer control descriptor 08	256
Base + 0x1120	TCD09	eDMA transfer control descriptor 09	256
Base + 0x1140	TCD10	eDMA transfer control descriptor 10	256
Base + 0x1160	TCD11	eDMA transfer control descriptor 11	256
Base + 0x1180	TCD12	eDMA transfer control descriptor 12	256
Base + 0x11A0	TCD13	eDMA transfer control descriptor 13	256
Base + 0x11C0	TCD14	eDMA transfer control descriptor 14	256
Base + 0x11E0	TCD15	eDMA transfer control descriptor 15	256
Base + 0x1200	TCD16	eDMA transfer control descriptor 16	256
Base + 0x1220	TCD17	eDMA transfer control descriptor 17	256
Base + 0x1240	TCD18	eDMA transfer control descriptor 18	256
Base + 0x1260	TCD19	eDMA transfer control descriptor 19	256
Base + 0x1280	TCD20	eDMA transfer control descriptor 20	256
Base + 0x12A0	TCD21	eDMA transfer control descriptor 21	256
Base + 0x12C0	TCD22	eDMA transfer control descriptor 22	256
Base + 0x12E0	TCD23	eDMA transfer control descriptor 23	256
Base + 0x1300	TCD24	eDMA transfer control descriptor 24	256
Base + 0x1320	TCD25	eDMA transfer control descriptor 25	256
Base + 0x1340	TCD26	eDMA transfer control descriptor 26	256
Base + 0x1360	TCD27	eDMA transfer control descriptor 27	256

Table 9-1. eDMA 32-bit Memory Map (continued)

Address	Register Name	Register Description	Bits
Base + 0x1380	TCD28	eDMA transfer control descriptor 28	256
Base + 0x13A0	TCD29	eDMA transfer control descriptor 29	256
Base + 0x13C0	TCD30	eDMA transfer control descriptor 30	256
Base + 0x13E0	TCD31	eDMA transfer control descriptor 31	256
Base + 0x1400–0x17FC	—	Reserved	—

9.2.2 Register Descriptions

Read operations on reserved bits in a register return undefined data. Do not write operations to reserved bits. Writing to reserved bits in a register can generate errors. The maximum register bit-width for this device is 32-bits wide.

9.2.2.1 eDMA Control Register (EDMA_CR)

The 32-bit EDMA_CR defines the basic operating configuration of the eDMA.

The eDMA arbitrates channel service requests in two groups (0, 1) of 16 channels each:

- Group 0 contains channels 0–15
- Group 1 contains channels 16–31

Arbitration within a group can be configured to use either fixed-priority or round-robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers. In round-robin arbitration mode, the channel priorities are ignored and the channels within each group are cycled through, from channel 15 down to channel 0, without regard to priority.

See Section 9.2.2.16, “eDMA Channel n Priority Registers (EDMA_CPRn).”

The group priorities operate in a similar fashion. In group fixed-priority arbitration mode, channel service requests in the highest priority group are executed first where priority level 1 is the highest and priority level 0 is the lowest. The group priorities are assigned in the GRP*n*PRI fields of the eDMA control register (EDMA_CR). All group priorities must have unique values prior to any channel service requests occur, otherwise a configuration error is reported. In group round-robin mode, the group priorities are ignored and the groups are cycled through, from group 1 down to group 0, without regard to priority.

Address: Base + 0x0000 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	GRP1 PRI	0	GRP0 PRI	0	0	0	0	ERGA	ERCA	EDBG	0
W																
Reset	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 9-2. eDMA Control Register (EDMA_CR)

Table 9-2. EDMA_CR Field Descriptions

Field	Description
0–20	Reserved
21 GRP1PRI	Channel group 1 priority. Group 1 priority level when fixed-priority group arbitration is enabled.
22	Reserved
23 GRP0PRI	Channel group 0 priority. Group 0 priority level when fixed-priority group arbitration is enabled.
24–27	Reserved
28 ERGA	Enable round-robin group arbitration. 0 Fixed-priority arbitration is used for selection among the groups. 1 Round-robin arbitration is used for selection among the groups.
29 ERCA	Enable round-robin channel arbitration. 0 Fixed-priority arbitration is used for channel selection within each group. 1 Round-robin arbitration is used for channel selection within each group.
30 EDBG	Enable debug. 0 The assertion of the system debug control input is ignored. 1 The assertion of the system debug control input causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the system debug control input is negated or the EDBG bit is cleared.
31	Reserved

9.2.2.2 eDMA Error Status Register (EDMA_ESR)

The EDMA_ESR provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count, and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

In fixed arbitration mode, a configuration error is caused by any two channel priorities being equal within a group, or any group priority levels being equal among the groups. For either type of priority configuration error, the ERRCHN field is undefined. All channel priority levels within a group must be unique and all group priority levels among the groups must be unique when fixed arbitration mode is enabled.

If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (DLAST_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD.CITER.E_LINK bit does not equal the TCD.BITER.E_LINK bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and assert an error interrupt request if enabled. When correctly enabled, a scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is immediately stopped and the appropriate bus error flag is set. In this case, the state of the channel's transfer control descriptor is updated by the eDMA engine with the current source address, destination address, and minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write executes using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence executes before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the eDMA engine to stop the active channel, and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA_ESR. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are *not* affected when an error is detected. After the error status has been updated, the eDMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel executes and terminates with the same error condition.

Address: Base + 0x0004 Access: User R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPE	CPE	ERRCHN						SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-3. eDMA Error Status Register (EDMA_ESR)

Table 9-3. EDMA_ESR Field Descriptions

Field	Description
0 VLD	Logical OR of all EDMA_ERH and EDMA_ERL status bits. 0 No EDMA_ER bits are set. 1 At least one EDMA_ER bit is set indicating a valid error exists that has not been cleared.
1–15	Reserved
16 GPE	Group priority error. 0 No group priority error. 1 The last recorded error was a configuration error among the group priorities indicating not all group priorities are unique.
17 CPE	Channel priority error. 0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities within a group, indicating not all channel priorities within a group are unique.
18–23 ERRCHN[0:5]	Error channel number. Channel number of the last recorded error (excluding GPE and CPE errors). **Note:** Do not rely on the number in the ERRCHN field for group and channel priority errors. Group and channel priority errors need to be resolved by inspection. The application code must interrogate the priority registers to find groups or channels with duplicate priority level.
24 SAE	Source address error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD.SADDR field, indicating TCD.SADDR is inconsistent with TCD.SSIZE.
25 SOE	Source offset error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.SOFF field, indicating TCD.SOFF is inconsistent with TCD.SSIZE.
26 DAE	Destination address error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD.DADDR field, indicating TCD.DADDR is inconsistent with TCD.DSIZE.

Table 9-3. EDMA_ESR Field Descriptions (continued)

Field	Description
27 DOE	Destination offset error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.DOFF field, indicating TCD.DOFF is inconsistent with TCD.DSIZE.
28 NCE	NBYTES/CITER configuration error. 0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the TCD.NBYTES or TCD.CITER fields, indicating the following conditions exist: • TCD.NBYTES is not a multiple of TCD.SSIZE and TCD.DSIZE, or • TCD.CITER is equal to zero, or • TCD.CITER.E_LINK is not equal to TCD.BITER.E_LINK.
29 SGE	Scatter/gather configuration error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD.DLAST_SGA field, indicating TCD.DLAST_SGA is not on a 32-byte boundary. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD.E_SG is enabled.
30 SBE	Source bus error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
31 DBE	Destination bus error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

9.2.2.3 eDMA Enable Request Register (EDMA_ERQRL)

The EDMA_ERQRL provides a bit map for the 32 implemented channels to enable the request signal for each channel. EDMA_ERQRL maps to channels 31–0.

The state of any given channel enable is directly affected by writes to this register; the state is also affected by writes to the EDMA_SERQR and EDMA_CERQR. The EDMA_CERQR and EDMA_SERQR are provided so that the request enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the EDMA_ERQRL.

Both the DMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does *not* affect a channel service request made explicitly through software or a linked channel request.

Address: Base + 0x000C Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERQ 31	ERQ 30	ERQ 29	ERQ 28	ERQ 27	ERQ 26	ERQ 25	ERQ 24	ERQ 23	ERQ 22	ERQ 21	ERQ 20	ERQ 19	ERQ 18	ERQ 17	ERQ 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERQ 15	ERQ 14	ERQ 13	ERQ 12	ERQ 11	ERQ 10	ERQ 09	ERQ 08	ERQ 07	ERQ 06	ERQ 05	ERQ 04	ERQ 03	ERQ 02	ERQ 01	ERQ 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-4. eDMA Enable Request Low Register (EDMA_ERQRL)

Table 9-4. EDMA_ERQRL Field Descriptions

Field	Description
0–31 ERQ*n*	Enable DMA hardware service request *n.* 0 The DMA request signal for channel n is disabled. 1 The DMA request signal for channel n is enabled.

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that can affect the ending state of the EDMA_ERQR bit for that channel. If the TCD.D_REQ bit is set, then the corresponding EDMA_ERQR bit is cleared after the major loop is complete, disabling the DMA hardware request. Otherwise if the D_REQ bit is cleared, the state of the EDMA_ERQR bit is unaffected.

9.2.2.4 eDMA Enable Error Interrupt Register (EDMA_EEIRL)

The EDMA_EEIRL provides a bit map for the 32 channels to enable the error interrupt signal for each channel. EDMA_EEIRL maps to channels 31-0.

The state of any given channel's error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA_SEEIR and EDMA_CEEIR. The EDMA_SEEIR and EDMA_CEEIR are provided so that the error interrupt enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the EDMA_EEIRL.

Both the DMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted.

Address: Base + 0x0014 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EEI 31	EEI 30	EEI 29	EEI 28	EEI 27	EEI 26	EEI 25	EEI 24	EEI 23	EEI 22	EEI 21	EEI 20	EEI 19	EEI 18	EEI 17	EEI 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EEI 15	EEI 14	EEI 13	EEI 12	EEI 11	EEI 10	EEI 09	EEI 08	EEI 07	EEI 06	EEI 05	EEI 04	EEI 03	EEI 02	EEI 01	EEI 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-5. eDMA Enable Error Interrupt Low Register (EDMA_EEIRL)

Table 9-5. EDMA_EEIRL Field Descriptions

Field	Description
0–31 EEI*n*	Enable error interrupt *n*. 0 The error signal for channel *n* does not generate an error interrupt. 1 The assertion of the error signal for channel *n* generate an error interrupt request.

9.2.2.5 eDMA Set Enable Request Register (EDMA_SERQR)

The EDMA_SERQR is a simple memory-mapped mechanism used to enable the DMA request for a given channel by setting a bit in the EDMA_ERQRL. The data value on a register write sets the bit in the EDMA_ERQRL. Bit 1 (SERQ*n*) is a global set function that asserts the entire contents of EDMA_ERQRL. Reads of this register return all zeroes.

Address: Base + 0x0018 Access: User W/O

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 9-6. eDMA Set Enable Request Register (EDMA_SERQR)

Table 9-6. EDMA_SERQR Field Descriptions

Field	Descriptions
0	Reserved
1–7 SERQ [0:6]	Set enable request. 0–31 Set corresponding bit in EDMA_ERQRL 32–63 Reserved 64–127 Set all bits in EDMA_ERQRH and EDMA_ERQRL Bit 2 (SERQ1) is not used.

9.2.2.6 eDMA Clear Enable Request Register (EDMA_CERQR)

The EDMA_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERQRL to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be cleared. Setting bit 1 (CERQ*n*) provides a global clear function, forcing the entire contents of the EDMA_ERQRL to be zeroed, disabling all DMA request inputs. Reads of this register return all zeroes.

Address: Base + 0x0019 Access: User W/O

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 9-7. eDMA Clear Enable Request Register (EDMA_CERQR)

Table 9-7. EDMA_CERQR Field Descriptions

Field	Description
0	Reserved
1–7 CERQ[0:6]	Clear enable request. 0–31 Clear corresponding bit in EDMA_ERQRL 32–63 Reserved 64–127 Clear all bits in EDMA_ERQRH and EDMA_ERQRL Bit 2 (CERQ1) is not used.

9.2.2.7 eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

The EDMA_SEEIR provides a simple memory-mapped mechanism to set a given bit in the EDMA_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be set. Setting bit 1 (SEEI*n*) provides a global set function, forcing the entire contents of EDMA_EEIRL to be asserted. Reads of this register return all zeroes.

Address: Base + 0x001A Access: User W/O

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 9-8. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

Table 9-8. EDMA_SEEIR Field Descriptions

Field	Description
0	Reserved
1–7 SEEI[0:6]	Set enable error interrupt. 0–31 Set corresponding bit in EDMA_EIRRL 32–63 Reserved 64–127 Set all bits in EDMA_EEIRH or EDMA_EEIRL Bit 2 (SEEI1) is not used.

9.2.2.8 eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

The EDMA_CEEIR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_EEIRL which disables the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be cleared. Setting bit 1 (CEEI*n*) provides a global clear function, forcing the entire contents of the EDMA_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register return all zeroes.

Address: Base + 0x001B Access: User W/O

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 9-9. eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

Table 9-9. EDMA_CEEIR Field Descriptions

Field	Description
0	Reserved
1–7 CEEI[0:6]	Clear enable error interrupt. 0–31 Clear corresponding bit in EDMA_EEIRL 32–63 Reserved 64–127 Clear all bits in EDMA_EEIRH or EDMA_EEIRL Bit 2 (CEEI1) is not used.

9.2.2.9 eDMA Clear Interrupt Request Register (EDMA_CIRQR)

The EDMA_CIRQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_IRQRL to be cleared. Setting bit 1 (CINT*n*) provides a global clear function, forcing the entire contents of the EDMA_IRQRL to be zeroed, disabling all DMA interrupt requests. Reads of this register return all zeroes.

Address: Base + 0x001C Access: User W/O

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CINT[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 9-10. eDMA Clear Interrupt Request (EDMA_CIRQR)

Table 9-10. EDMA_CIRQR Field Descriptions

Field	Description
0	Reserved
1–7 CINT[0:6]	Clear interrupt request. 0–31 Clear corresponding bit in EDMA_IRQRL 32–63 Reserved 64–127 Clear all bits in EDMA_IRQRH or EDMA_IRQRL Bit 2 (CINT1) is not used.

9.2.2.10 eDMA Clear Error Register (EDMA_CER)

The EDMA_CER provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_ERL to be cleared. Setting bit 1 (CERR*n*) provides a global clear function, forcing the entire contents of the EDMA_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes.

Address: Base + 0x001D Access:User W/O

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CERR[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 9-11. eDMA Clear Error Register (EDMA_CER)

Table 9-11. EDMA_CER Field Descriptions

Field	Description
0	Reserved
1–7 CERR[0:6]	Clear error indicator. 0–31 Clear corresponding bit in EDMA_ERL 32–63 Reserved 64–127 Clear all bits in EDMA_ERH or EDMA_ERL Bit 2 (CERR1) is not used.

9.2.2.11 eDMA Set START Bit Register (EDMA_SSBR)

The EDMA_SSBR provides a memory-mapped mechanism to set the START bit in the TCD for a channel. The data value on a register write sets the START bit in the transfer control descriptor. SSB*n* is a global set function that sets all START bits for a channel. Reads of this register return all zeroes.

Address: Base + 0x001E Access: User W/O

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 9-12. eDMA Set START Bit Register (EDMA_SSBR)

Table 9-12. EDMA_SSBR Field Descriptions

Field	Description
0	Reserved
1–7 SSB[0:6]	Set START bit (channel service request). 0–31 Set the corresponding channel's TCD START bit 32–63 Reserved 64–127 Set all TCD START bits Bit 2 (SSB1) is not used.

9.2.2.12 eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

The EDMA_CDSBR provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting bit 1 (CDSB*n*) provides a global clear function, forcing all DONE bits to be cleared.

Address: Base + 0x001F Access: User W/O

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CDSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 9-13. eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

Table 9-13. EDMA_CDSBR Field Descriptions

Field	Description
0	Reserved
1–7 CDSB[0:6]	Clear DONE status bit. 0–31 Clear the corresponding channel's DONE bit 32–63 Reserved 64–127 Clear all TCD DONE bits Bit 2 (CDSB1) is not used.

9.2.2.13 eDMA Interrupt Request Register (EDMA_IRQRL)

The EDMA_IRQRL provide a bit map for the 32 channels signaling the presence of an interrupt request for each channel. EDMA_IRQRL maps to channels 31–0.

The eDMA engine signals the occurrence of a programmed interrupt upon the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, it is software's responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA_CIRQR in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA_CIRQR. On writes to the EDMA_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA_CIRQR is provided so the interrupt request for a *single* channel can easily be cleared without the need to perform a read-modify-write sequence to the EDMA_IRQRL.

Address: Base + 0x0024 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INT 31	INT 30	INT 29	INT 28	INT 27	INT 26	INT 25	INT 24	INT 23	INT 22	INT 21	INT 20	INT 19	INT 18	INT 17	INT 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INT 15	INT 14	INT 13	INT 12	INT 11	INT 10	INT 09	INT 08	INT 07	INT 06	INT 05	INT 04	INT 03	INT 02	INT 01	INT 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-14. eDMA Interrupt Request Low Register (EDMA_IRQRL)

Table 9-14. EDMA_IRQRL Field Descriptions

Field	Description
0–31 INT*n*	eDMA interrupt request *n*. 0 The interrupt request for channel *n* is cleared. 1 The interrupt request for channel *n* is active.

9.2.2.14 eDMA Error Register (EDMA_ERL)

The EDMA_ERL provides a bit map for the 32 channels signaling the presence of an error for each channel. EDMA_ERL maps to channels 31-0.

The eDMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA_EEIR, then logically summed across groups of 16 and 32 channels to form several group error interrupt requests which is then routed to the interrupt controller. During the execution of the interrupt service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the EDMA_CER in the interrupt service routine is used for this purpose. Recall the normal DMA channel completion indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are *not* affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the EDMA_EEIR. The EDMA_ESR[VLD] bit is a logical OR of all bits in this register and it provides a single bit indication of any errors. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the EDMA_CER. On writes to EDMA_ERL, a 1 in any bit position clears the corresponding channel's error status. A 0 in any bit position has no affect on the corresponding channel's current error status. The EDMA_CER is provided so the error indicator for a *single* channel can easily be cleared.

Address: Base + 0x002C Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERR 31	ERR 30	ERR 29	ERR 28	ERR 27	ERR 26	ERR 25	ERR 24	ERR 23	ERR 22	ERR 21	ERR 20	ERR 19	ERR 18	ERR 17	ERR 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERR 15	ERR 14	ERR 13	ERR 12	ERR 11	ERR 10	ERR 09	ERR 08	ERR 07	ERR 06	ERR 05	ERR 04	ERR 03	ERR 02	ERR 01	ERR 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-15. eDMA Error Low Register (EDMA_ERL)

Table 9-15. EDMA_ERL Field Descriptions

Field	Description
0–31 ERR*n*	eDMA Error *n*. 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.

9.2.2.15 DMA Hardware Request Status (EDMA_HRSL)

The EDMA_HRSL registers provide a bit map for the implemented channels (16and32) to show the current hardware request status for each channel. EDMA_HRSL supports channels 31–0.See Table 9-16 for the EDMA_HRS definition.

Address: Base + 0x0034 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HRS 31	HRS 30	HRS 29	HRS 28	HRS 27	HRS 26	HRS 25	HRS 24	HRS 23	HRS 22	HRS 21	HRS 20	HRS 19	HRS 18	HRS 17	HRS 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HRS 15	HRS 14	HRS 13	HRS 12	HRS 11	HRS 10	HRS 09	HRS 08	HRS 07	HRS 06	HRS 05	HRS 04	HRS 03	HRS 02	HRS 01	HRS 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-16. EDMA Hardware Request Status Register Low (EDMA_HRSL)

Table 9-16. EDMA_HRSL Field Descriptions

Field	Description
0–31 HRSn	DMA Hardware Request Status 0 A hardware service request for channel n is not present. 1 A hardware service request for channel n is present. **Note:** The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[ERQn] bit.

9.2.2.16 eDMA Channel *n* Priority Registers (EDMA_CPR*n*)

When the fixed-priority channel arbitration mode is enabled (EDMA_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel within a group. The channel priorities are evaluated by numeric value; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software chooses to modify channel priority values, then the software must ensure that the channel priorities contain unique values, otherwise a configuration error is reported. The range of the priority value is limited to the values of 0 through 15. When read, the GRPPRI bits of the EDMA_CPR*n* register reflect the current priority level of the group of channels in which the corresponding channel resides. GRPPRI bits are not affected by writes to the EDMA_CPR*n* registers. The group priority is assigned in the EDMA_CR.

See Figure 9-2 and Table 9-2 for the EDMA_CR definition.

Channel preemption is enabled on a per-channel basis by setting the ECP bit in the EDMA_CPR*n* register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all of its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is only available when fixed arbitration is selected for both group and channel arbitration modes.

Address: Base + (0x100 + n) Access: User R/W

	0	1	2	3	4	5	6	7
R	ECP	0	GRPPRI		CHPRI			
W								
Reset	0	0			— [1]			

[1] The reset value for the group and channel priority fields, GRPPRI[0–1] and CHPRI[0–3] is the channel number for the priority register; EDMA_CPR31[GRPPRI] = 0b01 and EDMA_CPR31[CHPRI] = 0b1111.

Figure 9-17. eDMA Channel *n* Priority Register (EDMA_CPR*n*)

The following table describes the fields in the eDMA channel *n* priority register:

Table 9-17. EDMA_CPR*n* Field Descriptions

Field	Description
0 ECP	Enable channel preemption. 0 Channel *n* cannot be suspended by a higher priority channel's service request. 1 Channel *n* can be temporarily suspended by the service request of a higher priority channel.
1	Reserved
2–3 GRPPRI [0:1]	Channel *n* current group priority. Group priority assigned to this channel group when fixed-priority arbitration is enabled. These two bits are read only; writes are ignored. The reset value for the group priority fields, is equal to the corresponding channel number for each priority register; that is, EDMA_CPR31[GRPPRI] = 0b01.
4–7 CHPRI [0:3]	Channel *n* arbitration priority. Channel priority when fixed-priority arbitration is enabled. The reset value for the channel priority fields CHPRI[0–3], is equal to the corresponding channel number for each priority register; that is, EDMA_CPR31[CHPRI] = 0b1111.

9.2.2.17 Transfer Control Descriptor (TCD)

Each channel requires a 256-bit transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 31. The definitions of the TCD are presented as 23 variable-length fields.

Table 9-18 defines the fields of the basic TCD structure.

Table 9-18. TCD*n* 32-bit Memory Structure

eDMA Bit Offset	Bit Length	TCD*n* Field Name	TCD*n* Abbreviation	Word #
0x1000 + (32 x n) + 0	32	Source address	SADDR	Word 0
0x1000 + (32 x n) + 32	5	Source address modulo	SMOD	Word 1
0x1000 + (32 x n) + 37	3	Source data transfer size	SSIZE	
0x1000 + (32 x n) + 40	5	Destination address modulo	DMOD	
0x1000 + (32 x n) + 45	3	Destination data transfer size	DSIZE	
0x1000 + (32 x n) + 48	16	Signed source address offset	SOFF	
0x1000 + (32 x n) + 64	32	Inner minor byte count	NBYTES	Word 2
0x1000 + (32 x n) + 96	32	Last source address adjustment	SLAST	Word 3
0x1000 + (32 x n) + 128	32	Destination address	DADDR	Word 4
0x1000 + (32 x n) + 160	1	Channel-to-channel linking on minor loop complete	CITER.E_LINK	Word 5
0x1000 + (32 x n) + 161	6	Current major iteration count or Link channel number	CITER or CITER.LINKCH	
0x1000 + (32 x n) + 167	9	Current major iteration count	CITER	
0x1000 + (32 x n) + 176	16	Destination address offset (signed)	DOFF	

Table 9-18. TCD*n* 32-bit Memory Structure (continued)

eDMA Bit Offset	Bit Length	TCD*n* Field Name	TCD*n* Abbreviation	Word #
0x1000 + (32 x n) + 192	32	Last destination address adjustment / scatter gather address	DLAST_SGA	Word 6
0x1000 + (32 x n) + 224	1	Channel-to-channel Linking on Minor Loop Complete	BITER.E_LINK	Word 7
0x1000 + (32 x n) + 225	6	Starting major iteration count or link channel number	BITER or BITER.LINKCH	
0x1000 + (32 x n) + 231	9	Starting major iteration count	BITER	
0x1000 + (32 x n) + 240	2	Bandwidth control	BWC	
0x1000 + (32 x n) + 242	6	Link channel number	MAJOR.LINKCH	
0x1000 + (32 x n) + 248	1	Channel done	DONE	
0x1000 + (32 x n) + 249	1	Channel active	ACTIVE	
0x1000 + (32 x n) + 250	1	Channel-to-channel linking on major loop complete	MAJOR.E_LINK	
0x1000 + (32 x n) + 251	1	Enable scatter/gather processing	E_SG	
0x1000 + (32 x n) + 252	1	Disable request	D_REQ	
0x1000 + (32 x n) + 253	1	Channel interrupt enable when current major iteration count is half complete	INT_HALF	
0x1000 + (32 x n) + 254	1	Channel interrupt enable when current major iteration count complete	INT_MAJ	
0x1000 + (32 x n) + 255	1	Channel start	START	

Figure 9-18 defines the fields of the TCD*n* structure.

Word Offset	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31
0x0000	SADDR							
0x0004	SMOD	SSIZE	DMOD	DSIZE	SOFF			
0x0008	NBYTES							
0x000C	SLAST							
0x0010	DADDR							
0x0014	CITER.E_ LINK	CITER[1] or CITER.LINKCH	CITER[1]		DOFF			
0x0018	DLAST_SGA							
0x001C	BITER.E_ LINK	BITER[2] or BITER.LINKCH	BITER[2]	BWC	MAJOR LINKCH	DONE	ACTIVE	MAJOR.E_LINK E_SG D_REQ INT_HALF INT_MAJ START

Figure 9-18. TCD Structure

1 If channel linking on minor link completion is disabled, TCD bits [161:175] are used to form a 15-bit CITER field; if channel-to-channel linking is enabled, CITER becomes a 9-bit field.

2 If channel linking on minor link completion is disabled, TCD bits [225:239] are used to form a 15-bit BITER field; if channel-to-channel linking is enabled, BITER becomes a 9-bit field.

NOTE

The TCD structures for the eDMA channels shown in Figure 9-18 are implemented in internal SRAM. These structures are not initialized at reset. Therefore, all channel TCD parameters must be initialized by the application code before activating that channel.

The following table gives a detailed description of the TCD*n* fields:

Table 9-19. TCD*n* Field Descriptions

Bits Word Offset [n:n]	Field Name	Description
0–31 0x0 [0:31]	SADDR [0:31]	Source address. Memory address pointing to the source data. Word 0x0, bits 0–31.
32–36 0x4 [0:4]	SMOD [0:4]	Source address modulo. 0 Source address modulo feature is disabled. not 0 This value defines a specific address range which is specified to be either the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 “size” bytes, start the queue at a 0-modulo-size address and set the SMOD field to the value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range.
37–39 0x4 [5:7]	SSIZE [0:2]	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 32-bit 101 32-byte burst (64-bit x 4) 110 Reserved 111 Reserved The attempted specification of a ‘reserved’ encoding causes a configuration error.
40–44 0x4 [8:12]	DMOD [0:4]	Destination address modulo. See the SMOD[0:5] definition.
45–47 0x4 [13:15]	DSIZE [0:2]	Destination data transfer size. See the SSIZE[0:2] definition.
48–63 0x4 [16:31]	SOFF [0:15]	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.
64–95 0x8 [0:31]	NBYTES [0:31]	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. **Note:** The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a four GB transfer.

Table 9-19. TCD*n* Field Descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
96–127 0xC [0:31]	SLAST [0:31]	Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.
128–159 0x10 [0:31]	DADDR [0:31]	Destination address. Memory address pointing to the destination data.
160 0x14 [0]	CITER.E_LINK	Enable channel-to-channel linking on minor loop completion. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** This bit must be equal to the BITER.E_LINK bit otherwise a configuration error is reported.
161–166 0x14 [1:6]	CITER [0:5] or CITER.LINKCH [0:5]	Current “major” iteration count or link channel number. If channel-to-channel linking is disabled (TCD.CITER.E_LINK = 0), then • No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] are used to form a 15-bit CITER field. otherwise • After the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel’s TCD.START bit.
167–175 0x14 [7:15]	CITER [6:14]	Current “major” iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field. **Note:** When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER must be 0x0001.
176–191 0x14 [16:31]	DOFF [0:15]	Destination address signed offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

Table 9-19. TCD*n* Field Descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
192–223 0x18 [0:31]	DLAST_SGA [0:31]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather). If scatter/gather processing for the channel is disabled (TCD.E_SG = 0) then • Adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to “restore” the destination address to the initial value, or adjust the address to reference the next data structure. Otherwise • This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32 byte, otherwise a configuration error is reported.
224 0x1C [0]	BITER.E_LINK	Enables channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
225–230 0x1C [1:6]	BITER [0:5] or BITER.LINKCH [0:5]	Beginning or starting “major” iteration count or link channel number. If channel-to-channel linking is disabled (TCD.BITER.E_LINK = 0), then • No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] are used to form a 15-bit BITER field. Otherwise • After the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel’s TCD.START bit. **Note:** When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
231–239 0x1C [7:15]	BITER [6:14]	Beginning or starting major iteration count. As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER must be 0x0001.

Table 9-19. TCD*n* Field Descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
240–241 0x1C [16:17]	BWC [0:1]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR). To minimize start-up latency, bandwidth control stalls are suppressed for the first two system bus cycles and after the last write of each minor loop. 00 No eDMA engine stalls 01 Reserved 10 eDMA engine stalls for four cycles after each r/w 11 eDMA engine stalls for eight cycles after each r/w
242–247 0x1C [18:23]	MAJOR.LINKCH [0:5]	Link channel number. If channel-to-channel linking on major loop complete is disabled (TCD.MAJOR.E_LINK = 0) then: • No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise • After the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's TCD.START bit.
248 0x1C [24]	DONE	Channel done. This flag indicates the eDMA has completed the outer major loop. It is set by the eDMA engine as the CITER count reaches zero; it is cleared by software or hardware when the channel is activated (when the channel has begun to be processed by the eDMA engine, not when the first data transfer occurs). **Note:** This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.
249 0x1C [25]	ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the eDMA engine as the inner minor loop completes or if any error condition is detected.
250 0x1C [26]	MAJOR.E_LINK	Enable channel-to-channel linking on major loop completion. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. NOTE: To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
251 0x1C [27]	E_SG	Enable scatter/gather processing. As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If enabled, the eDMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. NOTE: To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The current channel's TCD is "normal" format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.

Table 9-19. TCD*n* Field Descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
252 0x1C [28]	D_REQ	Disable hardware request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQL bit when the current major iteration count reaches zero. 0 The channel's EDMA_ERQL bit is not affected. 1 The channel's EDMA_ERQL bit is cleared when the outer major loop is complete.
253 0x1C [29]	INT_HALF	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the bit in the EDMA_ERQL when the current major iteration count reaches the halfway point. The eDMA engine performs the compare (CITER == (BITER >> 1)). This halfway point interrupt request supports double-buffered schemes, or where the processor needs an early indication of the data transfer's progress during data movement. CITER = BITER = 1 with INT_HALF enabled generates an interrupt as it satisfies the equation (CITER == (BITER >> 1)) after a single activation. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
254 0x1C [30]	INT_MAJ	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQL when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
255 0x1C [31]	START	Channel start. If this flag is set, the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

9.3 Functional Description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

9.3.1 eDMA Microarchitecture

The eDMA module is partitioned into two major modules: the eDMA engine and the transfer control descriptor local memory. Additionally, the eDMA engine is further partitioned into four submodules, which are detailed below.

- eDMA engine
  - *Address path*: This module implements registered versions of two channel transfer control descriptors: channel 'x' and channel 'y,' and is responsible for all the master bus address calculations. All the implemented channels provide the exact same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by EDMA_CPR*n*[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.

When any other channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other address path channel{x,y}. After the inner minor loop completes execution, the address path hardware writes the new values for the TCDn.{SADDR, DADDR, CITER} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCDn.CITER field, and a possible fetch of the next TCD*n* from memory as part of a scatter/gather operation.

— *Data path*: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The system read data bus is the primary input, and the system write data bus is the primary output.

The address and data path modules directly support the 2-stage pipelined system bus. The address path module represents the 1st stage of the bus pipeline (the address phase), while the data path module implements the 2nd stage of the pipeline (the data phase).

— *Program model/channel arbitration*: This module implements the first section of eDMA's programming model as well as the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request outputs are also connected to this module (via the Control logic).

— *Control*: This module provides all the control functions for the eDMA engine. For data transfers where the source and destination sizes are equal, the eDMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner 'minor loop' byte count has been moved.

A minor loop interaction is defined as the number of bytes to transfer (*n*bytes) divided by the transfer size. Transfer size is defined as the following:

if (SSIZE < DSIZE)

transfer size = destination transfer size (# of bytes)

else

transfer size = source transfer size (# of bytes)

Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, AND START. Major loop TCD variables are DLAST, SLAST, CITER, BITER, DONE, D_REQ, INT_MAJ, MAJOR_LNKCH, and INT_HALF.

For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. As an example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- TCD local memory
  — *Memory controller*: This logic implements the required dual-ported controller, handling accesses from both the eDMA engine as well as references from the slave bus. As noted earlier, in the event of simultaneous accesses, the eDMA engine is given priority and the slave transaction is stalled. The hooks to a BIST controller for the local TCD memory are included in this module.
  — *Memory array*: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

9.3.2 eDMA Basic Data Flow

The basic flow of a data transfer can be partitioned into three segments. As shown in Figure 9-19, the first segment involves the channel service request. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel *n*. Channel service request via software and the TCDn.START bit follows the same basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed to through the eDMA engine, first through the control module, then into the program model/channel arbitration module. In the next cycle, the channel arbitration is performed, either using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the eDMA engine address path channel{x,y} registers. The TCD memory is organized 64-bits in width to minimize the time needed to fetch the activated channel's descriptor and load it into the eDMA engine address path channel{x,y} registers.

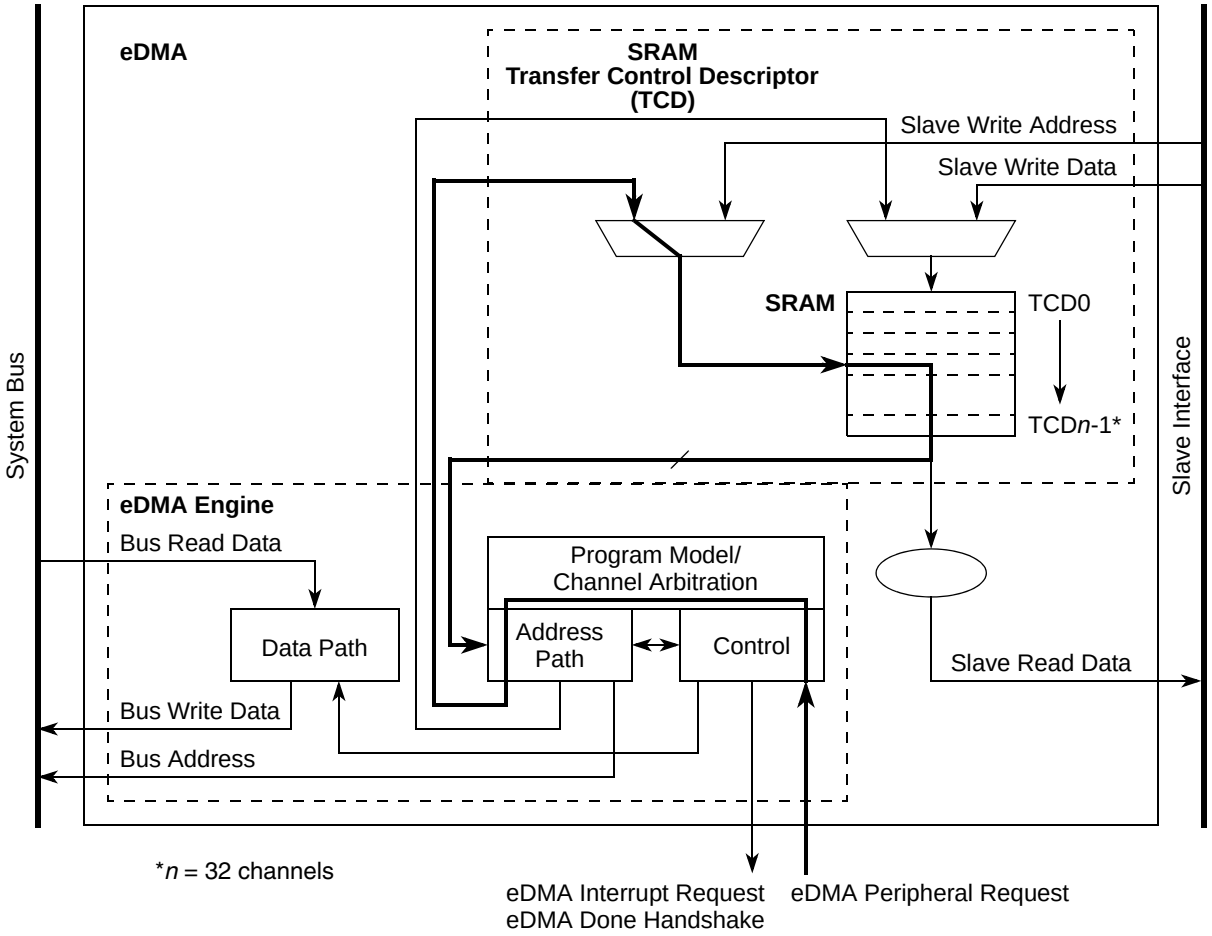


Figure 9-19. eDMA Operation, Part 1

In the second part of the basic data flow as shown in Figure 9-20, the modules associated with the data transfer (address path, data path and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the system bus during the destination write.

This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA Done Handshake signal is asserted at the end of the minor byte count transfer.

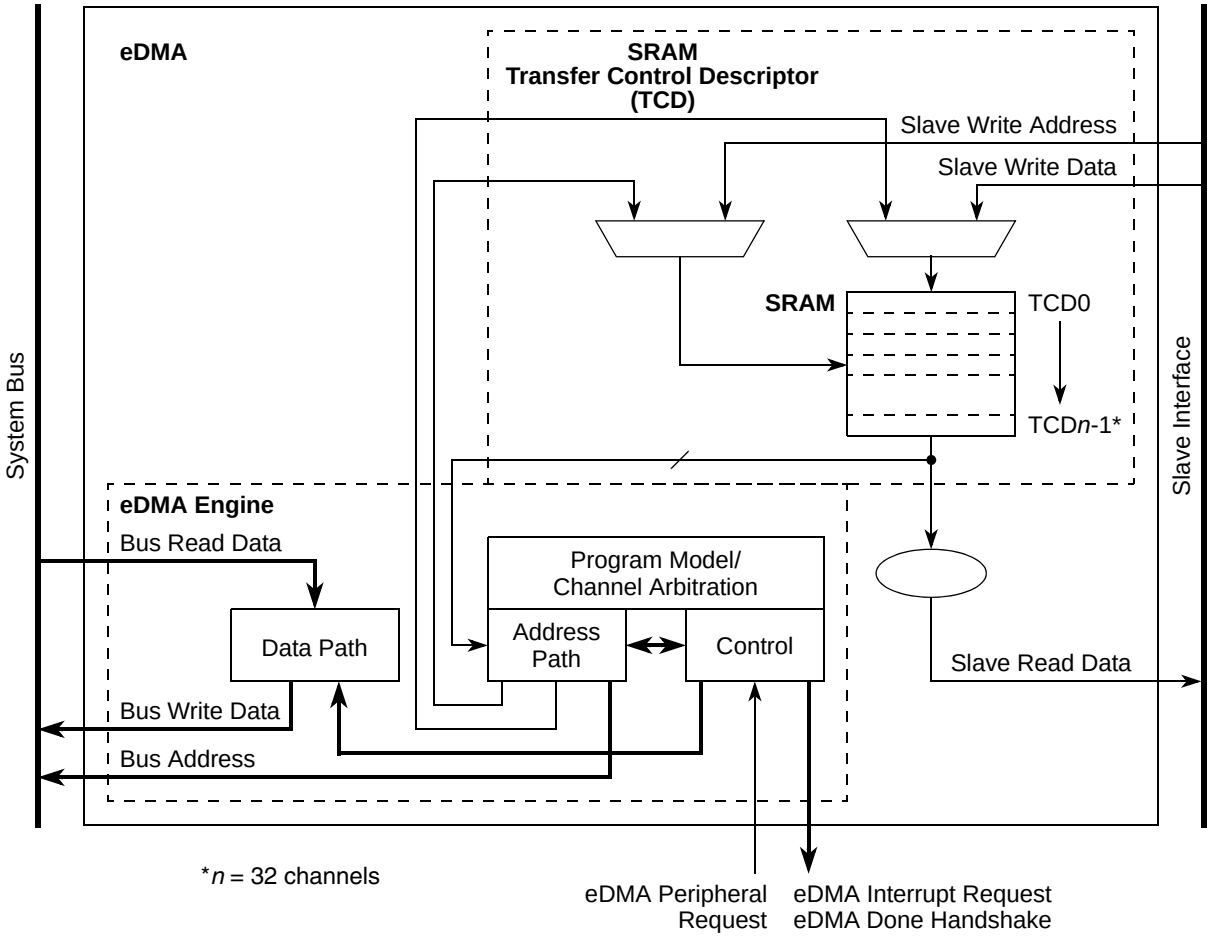


Figure 9-20. eDMA Operation, Part 2

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel's TCD: for example., SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then there are additional operations which are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in Figure 9-21.

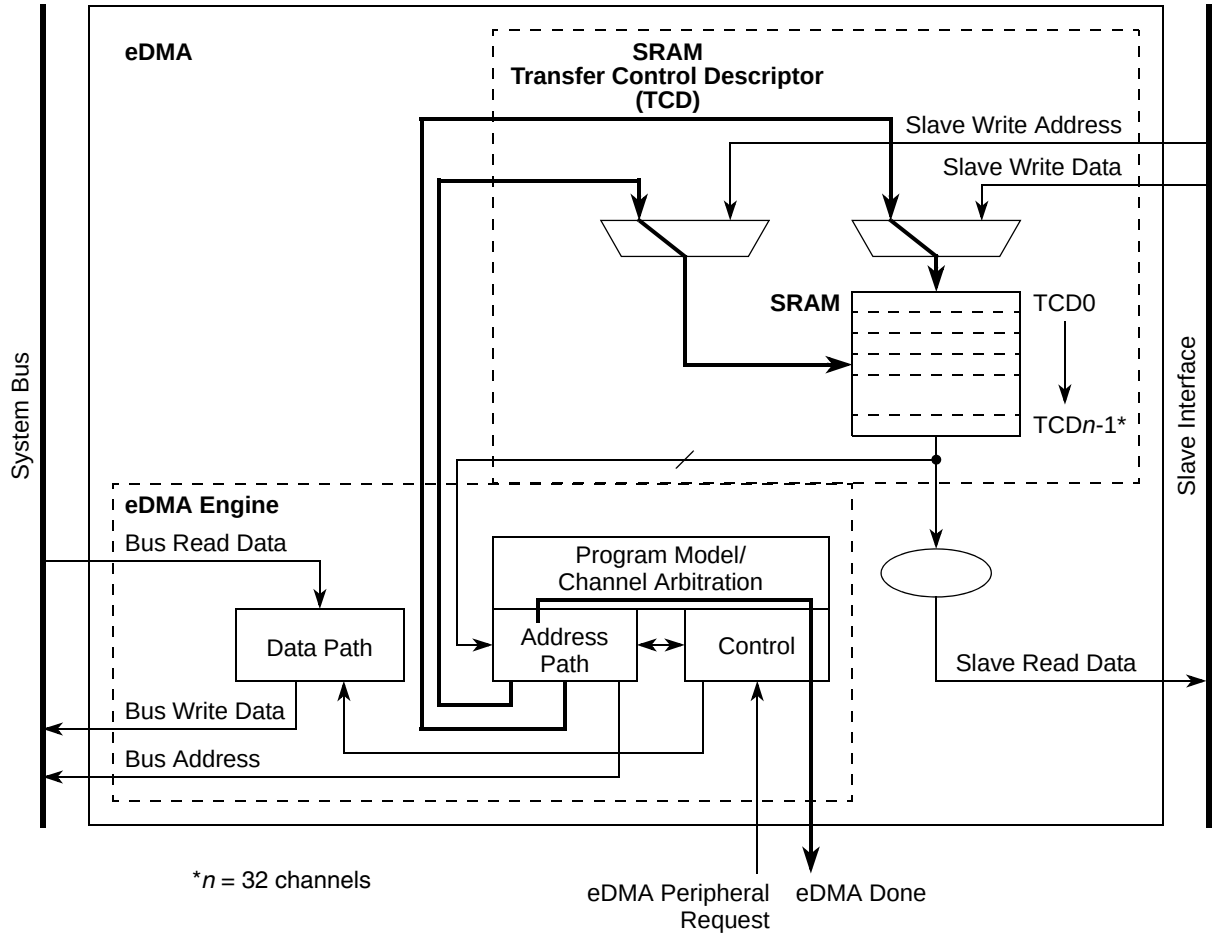


Figure 9-21. eDMA Operation, Part 3

9.3.3 eDMA Performance

This section addresses the performance of the eDMA module, focusing on two separate metrics. In the traditional data movement context, performance is best expressed as the peak data transfer rates achieved using the eDMA. In most implementations, this transfer rate is limited by the speed of the source and destination address spaces. In a second context where device-paced movement of single data values to/from peripherals is dominant, a measure of the requests that can be serviced in a fixed time is a more useful metric. In this environment, the speed of the source and destination address spaces remains important, but the microarchitecture of the eDMA also factors significantly into the resulting metric.

The peak transfer rates for several different source and destination transfers are shown in Table 9-20. The following assumptions apply to Table 9-20 and Table 9-21:

- Internal SRAM can be accessed with zero wait-states when viewed from the system bus data phase.
- All slave reads require two wait-states, and slave writes three wait-states, again viewed from the system bus data phase.
- All slave accesses are 32-bits in size.

Table 9-20. eDMA Peak Transfer Rates (MB/Sec)

System Speed, Transfer Size	Internal SRAM-to-Internal SRAM	32-Bit Slave-to-Internal SRAM	Internal SRAM-to-32-Bit Slave (buffering disabled)	Internal SRAM-to-32-Bit Slave (buffering enabled)
66.7 MHz, 32 bit	66.7	66.7	53.3	88.7
66.7 MHz, 64 bit	133.3	66.7	53.3	88.7
66.7 MHz, 256 bit[1]	213.4	—[2]	—[2]	—[2]
83.3 MHz, 32 bit	83.3	83.3	66.7	110.8
83.3 MHz, 64 bit	166.7	83.3	66.7	110.8
83.3 MHz, 256 bit[1]	266.6	—[2]	—[2]	—[2]
100.0 MHz, 32 bit	100.0	100.0	80.0	133.0
100.0 MHz, 64 bit	200.0	100.0	80.0	133.0
100.0 MHz, 256 bit[1]	320.0	—[2]	—[2]	—[2]
132.0 MHz, 32 bit	132.0	132.0	105.6	175.6
132.0 MHz, 64 bit	264.0	132.0	105.6	175.6
132.0 MHz, 256 bit[1]	422.4	—[2]	—[2]	—[2]

1 A 256-bit transfer occurs as a burst of four 64-bit beats.

2 Not applicable: burst access to a slave port is not supported.

Table 9-20 presents a peak transfer rate comparison, measured in MBs per second where the internal-SRAM-to-internal-SRAM transfers occur at the core's datapath width; that is, either 32- or 64-bits per access. For all transfers involving the slave bus, 32-bit transfer sizes are used. In all cases, the transfer rate includes the time to read the source plus the time to write the destination.

The second performance metric is a measure of the number of DMA requests that can be serviced in a given amount of time. For this metric, it is assumed the peripheral request causes the channel to move a single slave-mapped operand to/from internal SRAM. The same timing assumptions used in the previous example apply to this calculation. In particular, this metric also reflects the time required to activate the channel. The eDMA design supports the following hardware service request sequence:

- Cycle 1: eDMA peripheral request is asserted.
- Cycle 2: The eDMA peripheral request is registered locally in the eDMA module and qualified. (TCD.START bit initiated requests start at this point with the registering of the slave write to TCD bit 255).
- Cycle 3: Channel arbitration begins.
- Cycle 4: Channel arbitration completes. The transfer control descriptor local memory read is initiated.
- Cycle 5–6: The first two parts of the activated channel's TCD is read from the local memory. The memory width to the eDMA engine is 64 bits, so the entire descriptor can be accessed in four cycles.

- Cycle 7: The first system bus read cycle is initiated, as the third part of the channel's TCD is read from the local memory. Depending on the state of the crossbar switch, arbitration at the system bus can insert an additional cycle of delay here.
- Cycle 8 – *n*: The last part of the TCD is read in. This cycle represents the first data phase for the read, and the address phase for the destination write.

  The exact timing from this point is a function of the response times for the channel's read and write accesses. In this case of an slave read and internal SRAM write, the combined data phase time is 4 cycles. For an SRAM read and slave write, it is five cycles.
- Cycle *n* + 1: This cycle represents the data phase of the last destination write.
- Cycle *n* + 2: The eDMA engine completes the execution of the inner minor loop and prepares to write back the required TCD*n* fields into the local memory. The control and status fields at word offset 0x1C in TCD*n* are read. If the major loop is complete, the MAJOR.E_LINK and E_SG bits are checked and processed if enabled.
- Cycle *n* + 3: The appropriate fields in the first part of the TCD*n* are written back into the local memory.
- Cycle *n* + 4: The fields in the second part of the TCD*n* are written back into the local memory. This cycle coincides with the next channel arbitration cycle start.
- Cycle *n* + 5: The next channel to be activated performs the read of the first part of its TCD from the local memory. This is equivalent to Cycle 4 for the first channel's service request.

Assuming zero wait states on the system bus, DMA requests can be processed every 9 cycles. Assuming an average of the access times associated with slave-to-SRAM (4 cycles) and SRAM-to-slave (5 cycles), DMA requests can be processed every 11.5 cycles (4 + (4+5)/2 + 3). This is the time from Cycle 4 to Cycle "*n* + 5." The resulting peak request rate, as a function of the system frequency, is shown in Table 9-21. This metric represents millions of requests per second.

Table 9-21. eDMA Peak Request Rate (MReq/Sec)

System Frequency (MHz)	Request Rate (Zero Wait States)	Request Rate (with Wait States)
66.6	7.4	5.8
83.3	9.2	7.2
100.0	11.1	8.7
133.3	14.8	11.6
150.0	16.6	13.0

A general formula to compute the peak request rate (with overlapping requests) is:

PEAKreq = freq / [entry + (1 + read_ws) + (1 + write_ws) + exit]

where:

PEAKreq is the peak request rate

freq is the system frequency

entry is the channel startup (four cycles)

read_ws is the wait states seen during the system bus read data phase

write_ws is the wait states seen during the system bus write data phase

exit is the channel shutdown (three cycles)

For example, consider a system with the following characteristics:

- Internal SRAM can be accessed with one wait-state when viewed from the system bus data phase.
- All slave reads require two wait-states, and slave writes three wait-states, again viewed from the system bus data phase.
- System operates at 150 MHz.

For an SRAM to slave transfer,

PEAKreq = 150 MHz / [4 + (1 + 1) + (1 + 3) + 3] cycles = 11.5 Mreq/sec

For a slave to SRAM transfer,

PEAKreq = 150 MHz / [4 + (1 + 2) + (1 + 1) + 3] cycles = 12.5 Mreq/sec

Assuming an even distribution of the two transfer types, the average peak request rate is:

PEAKreq = (11.5 Mreq/sec + 12.5 Mreq/sec) / 2 = 12.0 Mreq/sec

The minimum number of cycles to perform a single read/write, zero wait states on the system bus, from a cold start (no channel is executing, eDMA is idle) are the following:

- 11 cycles for a software (TCD.START bit) request
- 12 cycles for a hardware (eDMA peripheral request signal) request

Two cycles account for the arbitration pipeline and one extra cycle on the hardware request resulting from the internal registering of the eDMA peripheral request signals. For the peak request rate calculations above, the arbitration and request registering is absorbed in or overlap the previous executing channel.

NOTE

When channel linking or scatter/gather is enabled, a two-cycle delay is imposed on the next channel selection and startup. This allows the link channel or the scatter/gather channel to be eligible and considered in the arbitration pool for next channel selection.

9.4 Initialization and Application Information

9.4.1 eDMA Initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA_CPR*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_EEIRL and/or EDMA_EEIRH registers (optional).
4. Write the 32-byte TCD for each channel that can request service.
5. Enable any hardware service requests via the EDMA_ERQRH and/or EDMA_ERQRL registers.
6. Request channel service by either software (setting the TCD.START bit) or by hardware (slave device asserting its eDMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The eDMA engine reads the entire TCD, including the primary transfer control parameter shown in Table 9-22, for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the system bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.SADDR) to the destination (as defined by the destination address, TCD.DADDR) continue until the specified number of bytes (TCD.NBYTES) have been transferred. When the transfer is complete, the eDMA engine's local TCD.SADDR, TCD.DADDR, and TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed: for example, interrupts, major loop channel linking, and scatter/gather operations, if enabled.

Table 9-22. TCD Primary Control and Status Fields

TCD Field Name	Description
START	Control bit to explicitly start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (Cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service
BWC	Control bits for “throttling” bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

Figure 9-22 shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).

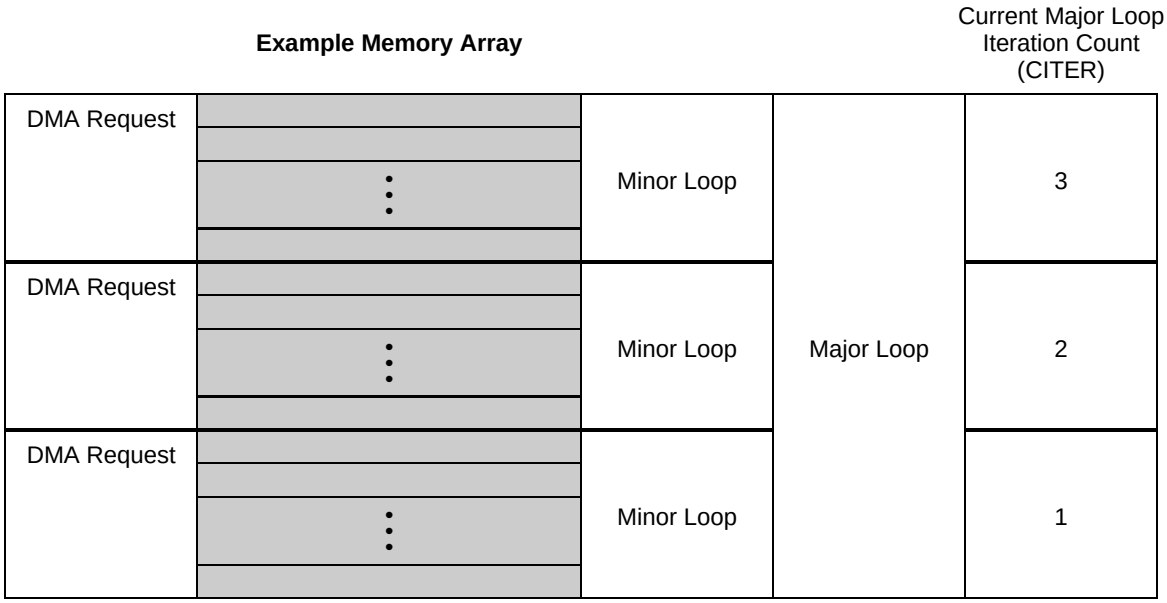


Figure 9-22. Example of Multiple Loop Iterations

Figure 9-23 lists the memory array terms and how the TCD settings interrelate.

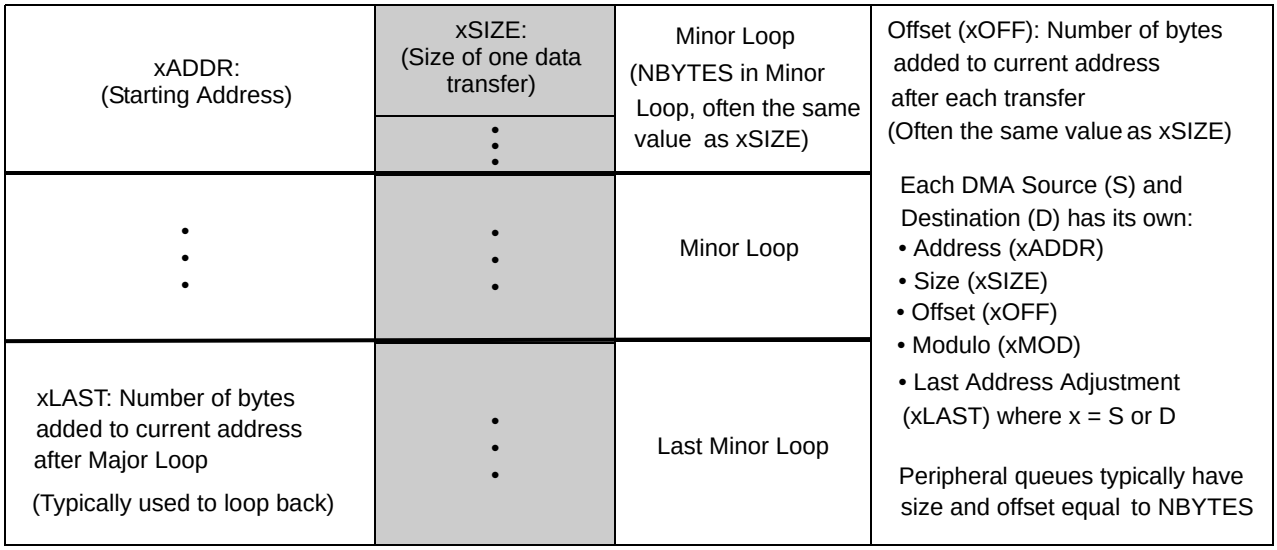


Figure 9-23. Memory Array Terms

9.4.2 DMA Programming Errors

The eDMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of two errors: group priority error and channel priority error, or EDMA_ESR[GPE] and EDMA_ESR[CPE], respectively.

For all error types other than group or channel priority errors, the channel number causing the error is recorded in the EDMA_ESR. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

Channel priority errors are identified within a group after that group has been selected as the active group. For the example that follows, all of the channel priorities in Group 1 are unique, but some of the channel priorities in Group 0 are the same:

1. Configure the eDMA for fixed-group and fixed-channel arbitration modes so that:
   — Group 1 is the highest priority and all channels are unique in that group.
   — Group 0 is the next highest priority and two channels have the same priority level.
2. If Group 1 has service requests pending, those requests are executed.
3. After all Group 1 requests have completed, Group 0 becomes the active group.
4. If Group 0 has a service request, the eDMA selects the undefined channel in Group 0 and generates a channel priority error.
5. Repeat Step 4 until the all Group 0 requests are serviced or a higher-priority Group 1 request is received.

In step 2, the eDMA acknowledge lines assert only if the selected channel is requesting service via the eDMA peripheral request signal. If interrupts are enabled for all channels, an error interrupt is generated. However, the channel number for the EDMA_ER and the error interrupt request line contain undefined data because the channel is ‘undefined’. A group priority error is global and any request in any group causes a group priority error.

If priority levels are not unique, the highest (channel/group) priority that has an active request is selected, but the lowest numbered (channel/group) with that priority is selected by arbitration and executed by the eDMA engine. The hardware service request handshake signals, error interrupts and error reporting are associated with the selected channel.

9.4.3 DMA Request Assignments

The assignments between the DMA requests from the modules to the channels of the eDMA are shown in Table 9-23. The source column is written in C language syntax. The syntax is module_instance.register[bit].

Table 9-23. DMA Request Summary for eDMA

DMA Request	Channel	Source	Description
eQADC_FISR0_CFFF0	0	EQADC.FISR0[CFFF0]	eQADC Command FIFO 0 Fill Flag
eQADC_FISR0_RFDF0	1	EQADC.FISR0[RFDF0]	eQADC Receive FIFO 0 Drain Flag
eQADC_FISR1_CFFF1	2	EQADC.FISR1[CFFF1]	eQADC Command FIFO 1 Fill Flag

Table 9-23. DMA Request Summary for eDMA (continued)

DMA Request	Channel	Source	Description
eQADC_FISR1_RFDF1	3	EQADC.FISR1[RFDF1]	eQADC Receive FIFO 1 Drain Flag
eQADC_FISR2_CFFF2	4	EQADC.FISR2[CFFF2]	eQADC Command FIFO 2 Fill Flag
eQADC_FISR2_RFDF2	5	EQADC.FISR2[RFDF2]	eQADC Receive FIFO 2 Drain Flag
eQADC_FISR3_CFFF3	6	EQADC.FISR3[CFFF3]	eQADC Command FIFO 3 Fill Flag
eQADC_FISR3_RFDF3	7	EQADC.FISR3[RFDF3]	eQADC Receive FIFO 3 Drain Flag
eQADC_FISR4_CFFF4	8	EQADC.FISR4[CFFF4]	eQADC Command FIFO 4 Fill Flag
eQADC_FISR4_RFDF4	9	EQADC.FISR4[RFDF4]	eQADC Receive FIFO 4 Drain Flag
eQADC_FISR5_CFFF5	10	EQADC.FISR5[CFFF5]	eQADC Command FIFO 5 Fill Flag
eQADC_FISR5_RFDF5	11	EQADC.FISR5[RFDF5]	eQADC Receive FIFO 5 Drain Flag
DSPIB_SR_TFFF	12	DSPIB.SR[TFFF]	DSPIB Transmit FIFO Fill Flag
DSPIB_SR_RFDF	13	DSPIB.SR[RFDF]	DSPIB Receive FIFO Drain Flag
DSPIC_SR_TFFF	14	DSPIC.SR[TFFF]	DSPIC Transmit FIFO Fill Flag
DSPIC_SR_RFDF	15	DSPIC.SR[RFDF]	DSPIC Receive FIFO Drain Flag
DSPID_SR_TFFF	16	DSPID.SR[TFFF]	DSPID Transmit FIFO Fill Flag
DSPID_SR_RFDF	17	DSPID.SR[RFDF]	DSPID Receive FIFO Drain Flag
eSCIA_COMBTX	18	ESCIA.SR[TDRE] || ESCIA.SR[TC] || ESCIA.SR[TXRDY]	eSCIA combined DMA request of the Transmit Data Register Empty, Transmit Complete, and LIN Transmit Data Ready DMA requests
eSCIA_COMBRX	19	ESCIA.SR[RDRF] || ESCIA.SR[RXRDY]	eSCIA combined DMA request of the Receive Data Register Full and LIN Receive Data Ready DMA requests
eMIOS_GFR_F0	20	EMIOS.GFR[F0]	eMIOS channel 0 Flag
eMIOS_GFR_F1	21	EMIOS.GFR[F1]	eMIOS channel 1 Flag
eMIOS_GFR_F2	22	EMIOS.GFR[F2]	eMIOS channel 2 Flag
eMIOS_GFR_F3	23	EMIOS.GFR[F3]	eMIOS channel 3 Flag
eMIOS_GFR_F4	24	EMIOS.GFR[F4]	eMIOS channel 4 Flag
eMIOS_GFR_F8	25	EMIOS.GFR[F8]	eMIOS channel 8 Flag
eMIOS_GFR_F9	26	EMIOS.GFR[F9]	eMIOS channel 9 Flag
eTPU_CDTRSR_A_DTRS0	27	ETPU.CDTRSR_A[DTRS0]	eTPUA Channel 0 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS1	28	ETPU.CDTRSR_A[DTRS1]	eTPUA Channel 1 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS2	29	ETPU.CDTRSR_A[DTRS2]	eTPUA Channel 2 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS14	30	ETPU.CDTRSR_A[DTRS14]	eTPUA Channel 14 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS15	31	ETPU.CDTRSR_A[DTRS15]	eTPUA Channel 15 Data Transfer Request Status

9.4.4 DMA Arbitration Mode Considerations

9.4.4.1 Fixed-Group Arbitration and Fixed-Channel Arbitration

In this mode, the channel service request from the highest priority channel in the highest priority group is selected to execute. If the eDMA is programmed so the channels within one group use ‘fixed’ priorities, and that group is assigned the highest ‘fixed’ priority of all groups, it is possible for that group to take all the bandwidth of the eDMA controller; that is, no other groups is serviced if there is always at least one DMA request pending on a channel in the highest priority group when the controller arbitrates the next DMA request. The advantage of this scenario is that latency can be small for channels that need to be serviced quickly. Preemption is available in this scenario only.

9.4.4.2 Round-Robin Group Arbitration, Fixed-Channel Arbitration

The occurrence of one or more DMA requests from one or more groups, the channel with the highest priority from a specific group is serviced first. Groups are serviced starting with the highest group number with a service request and rotating through to the lowest group number containing a service request.

After the channel request is serviced, the group round-robin algorithm selects the highest pending request from the next group in the round-robin sequence. Servicing continues using the round-robin method, always servicing the highest priority channel in the next group in the sequence, or just skipping a group if it has no pending requests.

If a channel requests service at a rate that equals or exceeds the round-robin service rate, then that channel is always serviced before lower priority channels in the same group, and thus the lower priority channels never are serviced. The advantage of this scenario is that no one group uses all the eDMA bandwidth. The highest priority channel selection latency is potentially greater than fixed/fixed arbitration. Excessive request rates on high priority channels can prevent the servicing of lower priority channels in the same group.

9.4.4.3 Round-Robin Group Arbitration, Round-Robin Channel Arbitration

Groups are serviced as described in Section 9.4.4.2, “Round-Robin Group Arbitration, Fixed-Channel Arbitration, but this time channels are serviced in channel number order. Only one channel is serviced from each requesting group for each round-robin pass through the groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to channel priority levels.

Because channels are serviced using a round-robin method, any channel that generates DMA requests faster than a combination of the group round-robin service rate and the channel service rate for its group does not prevent the servicing of other channels in its group.

This scenario ensures that all channels are guaranteed service at some point, regardless of the request rates. However, the potential latency can be quite high. All channels are treated equally. Priority levels are not used in round-robin/round-robin mode.

9.4.4.4 Fixed-Group Arbitration, Round-Robin Channel Arbitration

The highest priority group with a request is serviced. Lower priority groups are serviced if no pending requests exist in the higher priority groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned within the group.

This can cause the same bandwidth problem indicated in Section 9.4.4.1, but all the channels in the highest priority group are serviced. Service latency is short on the highest-priority group, but increases as the group priority decreases.

9.4.5 DMA Transfer

9.4.5.1 Single Request

To perform a simple transfer of '*n*' bytes of data with one activation, set the major loop to 1 (TCD.CITER = TCD.BITER = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer completes, the TCD.DONE bit is set and an interrupt is generated if correctly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte-wide memory port located at 0x1000. The destination memory has a word-wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

TCD.CITER = TCD.BITER = 1
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = –16
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA= –16
TCD.INT_MAJ = 1
TCD.START = 1 (Initialize all other fields before writing to this bit)
All other TCD fields = 0

This generates the following sequence of events:

1. Slave write to the TCD.START bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a) read_byte (0x1000), read_byte (0x1001), read_byte (0x1002), read_byte (0x1003)
   b) write_word (0x2000) –> first iteration of the minor loop
   c) read_byte (0x1004), read_byte (0x1005), read_byte (0x1006), read_byte (0x1007)
   d) write_word (0x2004) –> second iteration of the minor loop
   e) read_byte (0x1008), read_byte (0x1009), read_byte (0x100A), read_byte (0x100B)
   f) write_word (0x2008) –> third iteration of the minor loop
   g) read_byte (0x100C), read_byte (0x100D), read_byte (0x100E), read_byte (0x100F)
   h) write_word (0x200C) –> last iteration of the minor loop –> major loop complete
6. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 1 (TCD.BITER).
7. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR*n* = 1.
8. The channel retires.

The eDMA goes idle or services the next channel.

9.4.5.2 Multiple Requests

The next example is similar except it transfers 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the EDMA_ERQR, channel service requests are initiated by the slave device (set ERQR after TCD; TCD.START = 0).

TCD.CITER = TCD.BITER = 2
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = –32
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA= –32
TCD.INT_MAJ = 1
TCD.START = 0 (Initialize all other fields before writing this bit.)
All other TCD fields = 0

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers execute as follows:
   a) read_byte (0x1000), read_byte (0x1001), read_byte (0x1002), read_byte (0x1003)
   b) write_word (0x2000) –> first iteration of the minor loop
   c) read_byte (0x1004), read_byte (0x1005), read_byte (0x1006), read_byte (0x1007)
   d) write_word (0x2004) –> second iteration of the minor loop
   e) read_byte (0x1008), read_byte (0x1009), read_byte (0x100A), read_byte (0x100B)
   f) write_word (0x2008) –> third iteration of the minor loop
   g) read_byte (0x100C), read_byte (0x100D), read_byte (0x100E), read_byte (0x100F)
   h) write_word (0x200C) –> last iteration of the minor loop
6. eDMA engine writes: TCD.SADDR = 0x1010, TCD.DADDR = 0x2010, TCD.CITER = 1.
7. eDMA engine writes: TCD.ACTIVE = 0.
8. The channel retires –> one iteration of the major loop.

The eDMA goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers execute as follows:
    a) read_byte (0x1010), read_byte (0x1011), read_byte (0x1012), read_byte (0x1013)
    b) write_word (0x2010) –> first iteration of the minor loop
    c) read_byte (0x1014), read_byte (0x1015), read_byte (0x1016), read_byte (0x1017)
    d) write_word (0x2014) –> second iteration of the minor loop
    e) read_byte (0x1018), read_byte (0x1019), read_byte (0x101A), read_byte (0x101B)
    f) write_word (0x2018) –> third iteration of the minor loop
    g) read_byte (0x101C), read_byte (0x101D), read_byte (0x101E), read_byte (0x101F)
    h) write_word (0x201C) –> last iteration of the minor loop –> major loop complete
14. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 2 (TCD.BITER).
15. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR*n* = 1.
16. The channel retires –> major loop complete.

The eDMA goes idle or services the next channel.

9.4.5.3 Modulo Feature

The modulo feature of the eDMA provides the ability to easily implement a circular data queue in which the size of the queue is a power of two. MOD is a 5-bit field for the source and destination in the TCD, and specifies which lower address bits increment from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. Clearing this field to zero disables the modulo feature.

Table 9-24 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the mod field is set to 4, allowing for a 2^4 byte (16-byte) size queue.

Table 9-24. Modulo Feature Example

Transfer Number	Address
1	0x1234_5670
2	0x1234_5674
3	0x1234_5678
4	0x1234_567C
5	0x1234_5670
6	0x1234_5674

9.4.6 TCD Status

9.4.6.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.CITER field and test for a change. Another method can be extracted from the following sequence. The second method is to test the TCD.START bit AND the TCD.ACTIVE bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.START was written to a one. Polling the TCD.ACTIVE bit can be inconclusive because the active status can be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.START = 1, TCD.ACTIVE = 0, TCD.DONE = 0 (issued service request via software)
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (executing)
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (completed minor loop and is idle) or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (completed major loop and is idle)

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer’s model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. eDMA peripheral request asserts (issued service request via hardware)
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (executing)
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (completed minor loop and is idle) or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (completed major loop and is idle)

For both activation types, the major loop complete status is explicitly indicated via the TCD.DONE bit.

The TCD.START bit is cleared automatically when the channel begins execution regardless of how the channel was activated.

9.4.6.2 Active Channel TCD Reads

The eDMA reads the true TCD.SADDR, TCD.DADDR, and TCD.NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

9.4.6.3 Preemption Status

Preemption is only available when fixed arbitration is selected for both group and channel arbitration modes. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed group, fixed channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel and/or group priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCD.ACTIVE bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.ACTIVE bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

9.4.7 Channel Linking

Channel linking (or chaining) is a mechanism where one channel sets the TCD.START bit of another channel (or itself) thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when correctly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.CITER.E_LINK field are used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last.

When the major loop is exhausted, only the major loop channel link fields are used to determine whether to make a channel link. For example, with the initial fields of:

TCD.CITER.E_LINK = 1

TCD.CITER.LINKCH = 0xC

TCD.CITER value = 0x4

TCD.MAJOR.E_LINK = 1

TCD.MAJOR.LINKCH = 0x7

channel linking executes as:

1. Minor loop done –> set channel 12 TCD.START bit
2. Minor loop done –> set channel 12 TCD.START bit
3. Minor loop done –> set channel 12 TCD.START bit
4. Minor loop done, major loop done –> set channel 7 TCD.START bit

When minor loop linking is enabled (TCD.CITER.E_LINK = 1), the TCD.CITER field uses a nine bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.CITER.E_LINK = 0), the TCD.CITER field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD.CITER.LINKCH field are concatenated onto the CITER value to increase the range of the CITER.

NOTE

After configuration, the TCD.CITER.E_LINK bit and the TCD.BITER.E_LINK bit must be equal or a configuration error is reported. The CITER and BITER vector widths must be equal to calculate the major loop, half-way done interrupt point.

Table 9-25 summarizes how a DMA channel can “link” to another DMA channel, i.e, use another channel’s TCD, at the end of a loop.

Table 9-25. Channel Linking Parameters

Desired Link Behavior	TCD Control Field Name	Description
Link at end of Minor Loop	citer.e_link	Enable channel-to-channel linking on minor loop completion (current iteration)
	citer.linkch	Link channel number when linking at end of minor loop (current iteration)
Link at end of Major Loop	major.e_link	Enable channel-to-channel linking on major loop completion
	major.linkch	Link channel number when linking at end of major loop

9.4.8 Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

9.4.8.1 Dynamic Channel Linking and Dynamic Scatter/Gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCD.MAJOR.E_LINK or TCD.E_SG bits during channel execution. These bits are read from the TCD local memory at the *end* of channel execution thus allowing you to enable either feature during channel execution.

Because you are allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where you try to execute a dynamic channel link by enabling the TCD.MAJOR.E_LINK bit at the same time the eDMA engine is retiring the channel. The TCD.MAJOR.E_LINK is set in the programmer's model, but it is unclear whether the link completed before the channel retired.

Use the following coherency model when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the TCD.MAJOR.E_LINK bit
2. Read the TCD.MAJOR.E_LINK bit
3. Test the TCD.MAJOR.E_LINK request status:
   a) If the bit is set, the dynamic link attempt was successful.
   b) If the bit is cleared, the channel had already retired before the dynamic link completed.

This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the TCD.MAJOR.E_LINK and TCD.E_SG bits to zero on any writes to a channel's TCD after that channel's TCD.DONE bit is set indicating the major loop is complete.

NOTE

You must clear the TCD.DONE bit before writing the TCD.MAJOR.E_LINK or TCD.E_SG bits. The TCD.DONE bit is cleared automatically by the eDMA engine after a channel begins execution.

Chapter 10
Interrupt Controller (INTC)

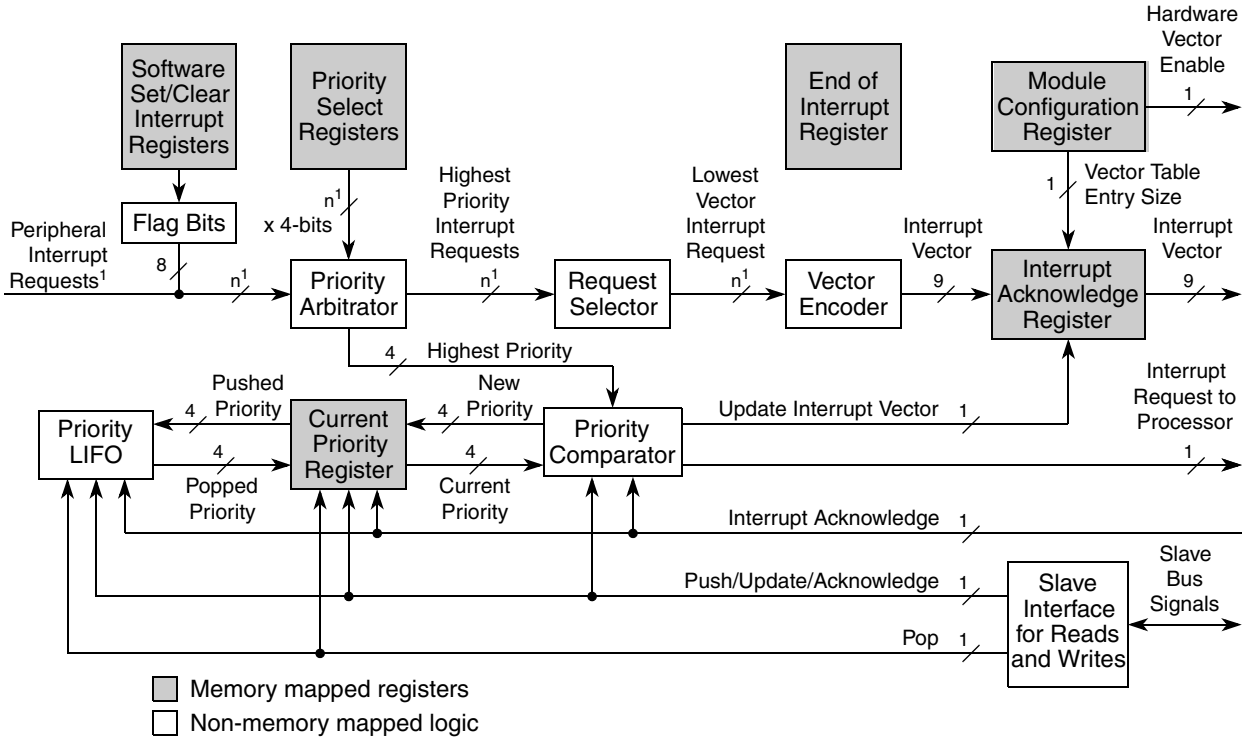
10.1 Introduction

This chapter describes the interrupt controller (INTC), which schedules interrupt requests (IRQs) from software and internal peripherals to the e200z3 core. The INTC provides interrupt prioritization and preemption, interrupt masking, interrupt priority elevation, and protocol support.

Interrupts implemented by the MCU are defined in the *e200z3 PowerPC™ Core Reference Manual*.

10.1.1 Block Diagram

Figure 4-1 shows details of the interrupt controller.



1 The total number of interrupt sources is 210, which includes 12 reserved sources, and 8 software sources.

Figure 10-1. INTC Block Diagram

10.1.2 Overview

Interrupt functionality for the device is handled between the e200z3 core and the interrupt controller. The CPU core has 19 exception sources, each of which can interrupt the core. One exception source is from the interrupt controller (INTC). The INTC provides priority-based preemptive scheduling of interrupt requests. This scheduling scheme is suitable for statically scheduled hard real-time systems. The INTC is optimized for a large number of interrupt requests. It is targeted to work with a PowerPC book E processor and automotive powertrain applications where the ISRs nest to multiple levels.

Table 10-1 displays the interrupt sources and the number available for each module; Figure 10-2 shows a general diagram of INTC software vector mode. See Table 10-9 for interrupt source vector details.

Table 10-1. Interrupt Sources Available

Interrupt Source (IRQs)	Number Available
Software	8
Watchdog	1
Memory	1
eDMA	33
FMPLL	2
External IRQ Input Pins	6
eMIOS	24
eTPU Engine A	33
eQADC	31
DSPI	15
eSCI	2
FlexCAN	40

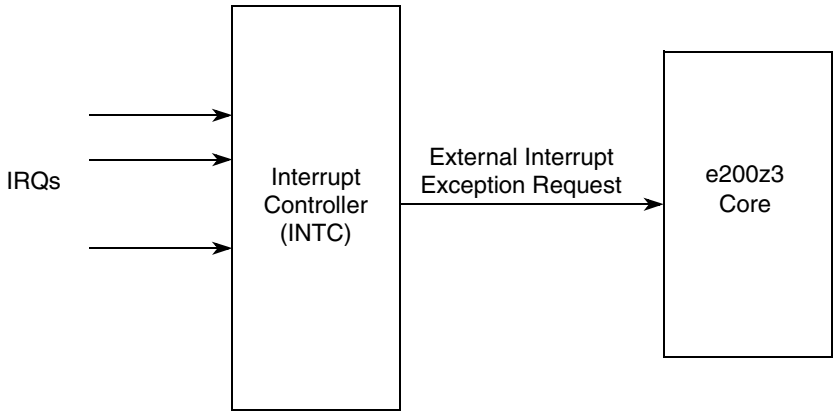


Figure 10-2. INTC Software Vector Mode

Two modes are available to determine the vector for the interrupt request source: software vector mode and hardware vector mode. In software vector mode, as shown in Figure 10-2, the e200z3 branches to a common interrupt exception handler whose location is determined by an address derived from special purpose registers IVPR and IVOR4. The interrupt exception handler reads the INTC_IACKR to determine the vector of the interrupt request source. Typical program flow for software vector mode is shown in Figure 10-3.

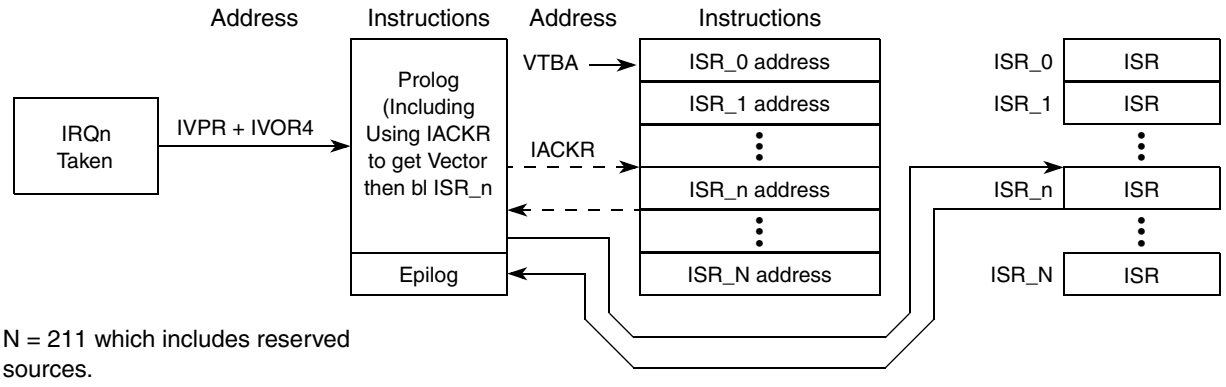


Figure 10-3. Program Flow–Software Vector Mode

In hardware vector mode, the core branches to a unique interrupt exception handler whose location is unique for each interrupt request source. Typical program flow for hardware vector mode is shown in Figure 10-4.

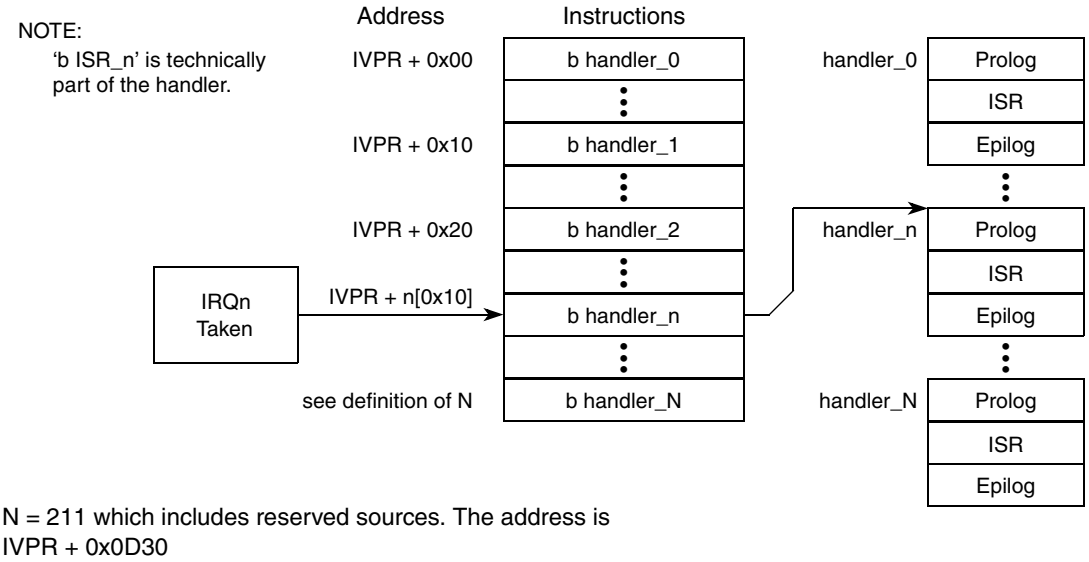


Figure 10-4. Program Flow–Hardware Vector Mode

For high priority interrupt requests in these target applications, the time from when the interrupt request from the peripheral asserts to the time when the processor begins to service the interrupt request must be minimized. The INTC can be optimized to minimize the time-to-service an interrupt request using hardware vector mode, where a unique vector is provided for each interrupt request source. It also provides

16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. Since applications have different priorities for each interrupt request source, the priority of each interrupt request is configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority level can be raised temporarily so that no task can preempt another task that shares the same resource.

Multiple processors can assert interrupt requests to each other through software settable interrupt requests, i.e., by using application software to assert an interrupt request. These same software settable interrupt requests also can be used to break the work involved in servicing an interrupt request into a high priority portion and a low priority portion. The high priority portion is initiated by a peripheral interrupt request, but then the ISR can assert a software settable interrupt request to finish the servicing in a lower priority ISR.

10.1.3 Features

Features include the following:

- Total number of interrupt vectors is 210 of which
  - 8 are software settable sources, and
  - 12 are reserved sources.
- 9-bit unique vector for each interrupt request source in hardware vector mode.
- Each interrupt source can be programmed to one of 16 priorities.
- Preemption:
  - Preemptive prioritized interrupt requests to processor.
  - ISR at a higher priority preempts ISRs or tasks at lower priorities.
  - Automatic pushing or popping of preempted priority to or from a LIFO.
  - Ability to modify the ISR or task priority. Modifying the priority can be used to implement the priority ceiling protocol for accessing shared resources.
- Low latency: three clocks from receipt of interrupt request from peripheral to interrupt request to processor.

10.1.4 Modes of Operation

The interrupt controller has two handshaking modes with the processor: software vector mode and hardware vector mode. The state of the hardware vector enable bit, INTC_MCR[HVEN], determines which mode is used.

In debug mode the interrupt controller operation is identical to its normal operation of software vector mode or hardware vector mode.

10.1.4.1 Software Vector Mode

In software vector mode, there is a common interrupt exception handler address which is calculated by hardware as shown in Figure 10-5. The upper half of the interrupt vector prefix register (IVPR) is added to the offset contained in the external input interrupt vector offset register (IVOR4). Because bits IVOR4[28:31] are not part of the offset value, the vector offset must be located on a quad-word (16-byte) aligned location in memory.

In software vector mode, the interrupt exception handler software must read the INTC interrupt acknowledge register (INTC_IACKR) to obtain the vector associated with the corresponding peripheral or software interrupt request. The INTC_IACKR contains a 32-bit address composed of a vector table base address (VTBA) plus an offset which is the interrupt vector (INTVEC). The address is then used to branch to the corresponding routine for that peripheral or software interrupt source.

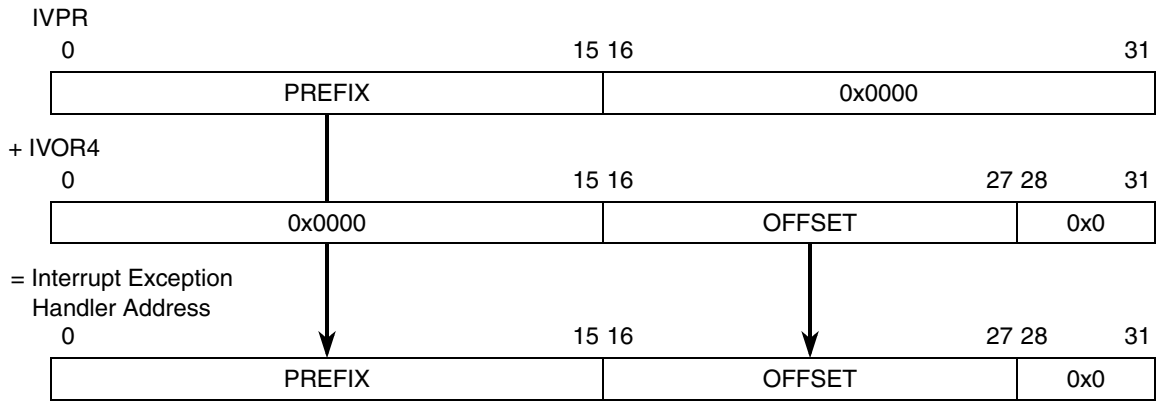


Figure 10-5. Software Vector Mode: Interrupt Exception Handler Address Calculation

Reading the INTC_IACKR acknowledges the INTC interrupt request and negates the interrupt request to the processor. The interrupt request to the processor does not clear if a higher priority interrupt request arrives. Even in this case, INTVEC does not update to the higher priority request until the lower priority interrupt request is acknowledged by reading the INTC_IACKR. Reading INTC_IACKR pushes the PRI value in the INTC current priority register (INTC_CPR) onto the LIFO and updates PRI in the INTC_CPR with the priority of the interrupt request. The INTC_CPR masks any peripheral or software settable interrupt request at the same or lower priority of the current value of the PRI field in INTC_CPR from generating an interrupt request to the processor.

The last actions of the interrupt exception handler must be the write to the end-of-interrupt register (INTC_EOIR). Writing to the INTC_EOIR signals the end of the servicing of the interrupt request. The INTC LIFO is popped into the INTC_CPR's PRI field by writing to the INTC_EOIR, and the size of a write does not affect the operation of the write. Those values and sizes written to this register neither update the INTC_EOIR contents nor affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR. The timing relationship between popping the LIFO and disabling recognition of external input has no restriction. The writes can happen in either order.

However, disabling recognition of the external input before popping the LIFO eases the calculation of the maximum pipe depth at the cost of postponing the servicing of the next interrupt request.

10.1.4.2 Hardware Vector Mode

In hardware vector mode, the interrupt exception handler address is specific to the peripheral or software settable interrupt source rather than being common to all of them. No IVOR is used. The interrupt exception handler address is calculated by hardware as shown in Figure 10-6. The upper half of the interrupt vector prefix register (IVPR) is added to an offset which corresponds to the peripheral or software interrupt source which caused the interrupt request. The offset matches the value in the Interrupt Vector field, INTC_IACKR[INTVEC]. Each interrupt exception handler address is aligned on a quad word (16-byte) boundary. IVOR4 is unused in this mode, and software does not need to read INTC_IACKR to get the interrupt vector number.

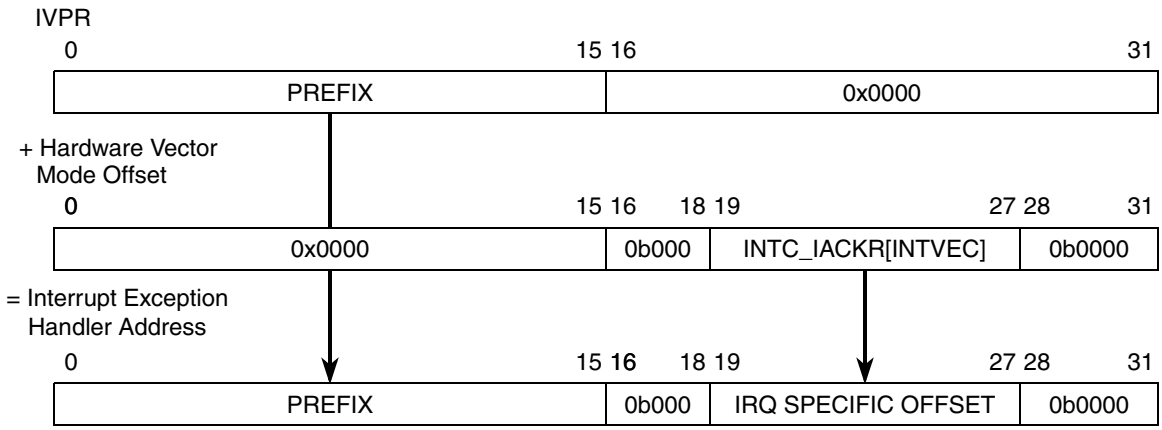


Figure 10-6. Hardware Vector Mode: Interrupt Exception Handler Address Calculation

The processor negates INTC interrupt request when automatically acknowledging the interrupt request. However, the interrupt request to the processor does not negate if a higher priority interrupt request arrives. Even in this case, the interrupt vector number does not update to the higher priority request until the lower priority request is acknowledged by the processor.

The assertion of the interrupt acknowledge signal pushes the PRI value in the INTC_CPR onto the LIFO and updates PRI in the INTC_CPR with the new priority.

10.2 External Signal Description

The INTC does not have any direct external MCU signals. However, there are fifteen external pins which can be configured in the SIU as external interrupt request input pins. When configured in this function, an interrupt on the pin sets a corresponding SIU external interrupt flag. These flags can cause one of five peripheral interrupt requests to the interrupt controller. See Table 10-2 for a list of the external interrupt pins. See the SIU chapter for more information on these pins.

Table 10-2. External Interrupt Signals

Function[1]	Description	P/A/G[2]	I/O Type	Reset Function/ State[3]	Post Reset Function/ State[4]	Pin
EMIOS[14:15]	eMIOS channel (output only)	P	O	— / WKPCFG	— / WKPCFG	AF19: AD18
IRQ[0:1]	External interrupt request	A	I			
GPIO[193:194]	GPIO	G	I/O			
BOOTCFG[0][5]	Boot configuration input	P	I	BOOTCFG / Down	— / Down	AA25: Y24
$\overline{\text{IRQ}}$[2]	External interrupt request	A	I			
GPIO[211]	GPIO	G	I/O			
BOOTCFG[1]	Boot configuration input	P	I	BOOTCFG / Down	— / Down	AA25: Y24
$\overline{\text{IRQ}}$[3]	External interrupt request	A	I			
GPIO[212]	GPIO	G	I/O			
PLLCFG[0]	FMPLL mode selection	P	I	PLLCFG / Up	— / Up	AB25
$\overline{\text{IRQ}}$[4]	External Interrupt Request	A	I			
GPIO[208]	GPIO	G	I/O			
PLLCFG[1]	FMPLL mode selection	P	I	PLLCFG / Up	— / Up	AA24
$\overline{\text{IRQ}}$[5]	External Interrupt Request	A	I			
SOUTD	DSPI D Data Output	A2	O			
GPIO[209]	GPIO	G	I/O			
TCRCLKA	eTPU A TCR clock	P	I	— / Up	— / Up	N4
$\overline{\text{IRQ}}$[7]	External interrupt request	A	I			
GPIO[113]	GPIO	G	I/O			
ETPUA[20:23]	eTPU A channel	P	I/O	— / WKPCFG	— / WKPCFG	H1:G4 G2:G1
$\overline{\text{IRQ}}$[8:11]	External interrupt request	A	I			
GPIO[134:137]	GPIO	G	I/O			
ETPUA[24:26]	eTPU A channel (output only)	P	O	— /WKPCFG	— /WKPCFG	F1:G3: F3
$\overline{\text{IRQ}}$[12:14]	External interrupt request	A	I			
GPIO[138:140]	GPIO	G	I/O			
ETPUA[27]	eTPU A channel (output only)	P	O	— /WKPCFG	— /WKPCFG	F2
$\overline{\text{IRQ}}$[15]	External interrupt request	A	I			
GPIO[141]	GPIO	G	I/O			

1 For each pin in the table, each line in the function column is a separate function of the pin. For all device I/O pins the selection of primary, secondary or tertiary function is done in the SIU module except where explicitly noted.

2 Primary, alternate, or GPIO function.

3 Terminology is O - output, I - input, Up - weak pull up enabled, Down - weak pull down enabled, Low - output driven low, High - output driven high.

4 Function after reset of GPI is general-purpose input.

5 This signal is not available on the 208 package due to pin limitations.

10.3 Memory Map/Register Definition

Table 10-3 is the INTC memory map.

10.3.1 Register Descriptions

Table 10-3. INTC Memory Map

Address	Register Name	Register Description	Bits
Base (0xFFF4_8000)	INTC_MCR	INTC module configuration register	32
Base + 0x0004	—	Reserved	—
Base + 0x0008	INTC_CPR	INTC current priority register	32
Base + 0x000C	—	Reserved	—
Base + 0x0010	INTC_IACKR	INTC interrupt acknowledge register [1]	32
Base + 0x0014	—	Reserved	—
Base + 0x0018	INTC_EOIR	INTC end-of-interrupt register	32
Base + 0x001C	—	Reserved	—
Base + 0x0020	INTC_SSCIR0	INTC software set/clear interrupt register 0	8
Base + 0x0021	INTC_SSCIR1	INTC software set/clear interrupt register 1	8
Base + 0x0022	INTC_SSCIR2	INTC software set/clear interrupt register 2	8
Base + 0x0023	INTC_SSCIR3	INTC software set/clear interrupt register 3	8
Base + 0x0024	INTC_SSCIR4	INTC software set/clear interrupt register 4	8
Base + 0x0025	INTC_SSCIR5	INTC software set/clear interrupt register 5	8
Base + 0x0026	INTC_SSCIR6	INTC software set/clear interrupt register 6	8
Base + 0x0027	INTC_SSCIR7	INTC software set/clear interrupt register 7	8
Base + (0x0028–0x003C)	—	Reserved	—
Base + (0x0040–0x0110)	INTC_PSR*n*	INTC priority select register[2] 0–211	8

[1] When the HVEN bit in the INTC_MCR is asserted, a read of the INTC_IACKR has no side effects.

[2] The PRI fields are “Reserved” for peripheral interrupt requests whose vectors are labeled as Reserved in Table 10-9.

With the exception of the INTC_SSCI*n* and INTC_PSR*n* registers, all of the registers are 32 bits in width. Any combination of accessing the 4 bytes of a register with a single access is supported, provided that the access does not cross a register boundary. These supported accesses include types and sizes of 8 bits, aligned 16 bits, and aligned 32 bits.

Although INTC_SSCI*n* and INTC_PSR*n* and 8 bits wide, they can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

In software vector mode, the side effects of a read of the INTC interrupt acknowledge register (INTC_IACKR) are the same regardless of the size of the read. In either software or hardware vector

mode, the size of a write to the INTC end-of-interrupt register (INTC_EOIR) does not affect the operation of the write.

10.3.1.1 INTC Module Configuration Register (INTC_MCR)

The INTC_MCR is used to configure options of the INTC.

Address: Base + 0x0000 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	VTES	0	0	0	0	HVEN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-7. INTC Module Configuration Register (INTC_MCR)

Table 10-4. INTC_MCR Field Descriptions

Field	Description
0–25	Reserved, must be cleared.
26 VTES	Vector table entry size. Controls the number of ‘0’s to the right of INTVEC in Section 10.3.1.3, “INTC Interrupt Acknowledge Register (INTC_IACKR). If the contents of INTC_IACKR are used as an address of an entry in a vector table as in software vector mode, then the number of rightmost ‘0’s determines the size of each vector table entry. VTES impacts software vector mode operation but also affects INTC_IACKR[INTVEC] position in both hardware vector mode and software vector mode. 0 4 bytes (Normal expected use) 1 8 bytes
27–30	Reserved, must be cleared.
31 HVEN	Hardware vector enable. Controls whether the INTC is in hardware vector mode or software vector mode. See Section 10.1.4, “Modes of Operation”, for the details of the handshaking with the processor in each mode. 0 Software vector mode 1 Hardware vector mode

10.3.1.2 INTC Current Priority Register (INTC_CPR)

The INTC_CPR masks any peripheral or software settable interrupt request set at the same or lower priority as the current value of the INTC_CPR[PRI] field from generating an interrupt request to the processor. When the INTC interrupt acknowledge register (INTC_IACKR) is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When the INTC end-of-interrupt register (INTC_EOIR) is written, the LIFO is popped into the INTC_CPR’s PRI field.

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. See Section 10.5.5, “Priority Ceiling Protocol.”

NOTE

On some eSys MCUs, a store to raise the PRI field which closely precedes an access to a shared resource can result in a non-coherent access to that resource unless an MBAR or MSYNC followed by an ISYNC sequence of instructions is executed between the accesses. An MBAR or MSYNC instruction is also necessary after accessing the resource but before lowering the PRI field. See Section 10.5.5.2, "Ensuring Coherency."

Address: Base + 0x0008 (INTC_CPR) Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PRI			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 10-8. INTC Current Priority Register (INTC_CPR)

Table 10-5. INTC_CPR Field Descriptions

Field	Description
0–27	Reserved, must be cleared.
28–31 PRI	Priority. PRI is the priority of the currently executing ISR according to the field values defined below. 1111 Priority 15 (highest) 1110 Priority 14 ... 0001 Priority 1 0000 Priority 0 (lowest)

10.3.1.3 INTC Interrupt Acknowledge Register (INTC_IACKR)

The INTC_IACKR provides a value that can be used to load the address of an ISR from a vector table. The vector table can be composed of addresses of the ISRs specific to their respective interrupt vectors.

Also, in software vector mode, the INTC_IACKR has side effects from reads. The side effects are the same regardless of the size of the read. Reading the INTC_IACKR does not have side effects in hardware vector mode.

NOTE

The INTC_IACKR must not be read speculatively while in software vector mode. Therefore, for future compatibility, the TLB entry covering the INTC_IACKR must be configured to be guarded.

In software vector mode, the INTC_IACKR must be read before setting MSR[EE]. No synchronization instruction is needed after reading the INTC_IACKR and before setting MSR[EE].

However, the time for the processor to recognize the assertion or negation of the external input to it is not defined by the book E architecture and can be greater than 0. Therefore, insert instructions between the reading of the INTC_IACKR and the setting of MSR[EE] that consume at least two processor clock cycles. This length of time allows the negation of the interrupt request to propagate through the processor before MSR[EE] is set.

Address: Base + 0x0010 (INTC_IACKR) Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA																				INTVEC										0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-9. INTC Interrupt Acknowledge Register (INTC_IACKR)—INTC_MCR[VTES] = 0

Address: Base + 0x0010 (INTC_IACKR) Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA																				INTVEC									0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-10. INTC Interrupt Acknowledge Register (INTC_IACKR)—INTC_MCR[VTES] = 1

Table 10-6. INTC_IACKR Field Descriptions

Field	Description
0–20 *or* 0–19 VTBA	Vector table base address. Can be the base address of a vector table of addresses of ISRs. The VTBA only uses the leftmost 20 bits when the VTES bit in INTC_MCR is asserted.
21–29 *or* 20–28 INTVEC	Interrupt vector. Vector of the peripheral or software-settable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC is updated, whether the INTC is in software or hardware vector mode. **Note:** If INTC_MCR[VTES] = 1, then the INTVEC field is shifted left one position to bits 20–28. VTBA is then shortened by one bit to bits 0–19.
30–31 *or* 29–31	Reserved, must be cleared.

10.3.1.4 INTC End-of-Interrupt Register (INTC_EOIR)

Writing to the INTC_EOIR signals the end of the servicing of the interrupt request. When the INTC_EOIR is written, the priority last pushed on the LIFO is popped into INTC_CPR. The values and size of data written to the INTC_EOIR are ignored. Those values and sizes written to this register neither update the INTC_EOIR contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0's to the INTC_EOIR.

Reading the INTC_EOIR has no effect on the LIFO.

Address: Base + 0x0018 (INTC_EOIR) Access: User W/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	EOIR																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-11. INTC End-of-Interrupt Register (INTC_EOIR)

10.3.1.5 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR[0–7])

The INTC_SSCIR*n* support the setting or clearing of software settable interrupt requests. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. With the exception of being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC just like a peripheral interrupt request. Writing a 1 to SET*n* leaves SET*n* unchanged at 0, but sets CLR*n*. Writing a 0 to SET*n* has no effect. CLR*n* is the flag bit. Writing a 1 to CLR*n* clears it. Writing a 0 to CLR*n* has no effect. If a 1 is written to a pair SET*n* and CLR*n* bits at the same time, CLR*n* is asserted, regardless of whether CLR*n* was asserted before the write.

Although INTC_SSCI*n* is 8-bits wide, it can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

Address: Base + 0x0020 + *n* (INTC_SSCIR*n*); *n* = 0–7 Access: User R/W

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	CLR*n*
W							SET*n*	
Reset	0	0	0	0	0	0	0	0

Figure 10-12. INTC Software Set/Clear Interrupt Register (INTC_SSCIR*n*)

Table 10-7. INTC_SSCIR*n* Field Descriptions

Field	Description
0–5	Reserved, must be cleared.
6 SET*n*	Set flag bits. Writing a 1 sets the CLR*n* bit. Writing a 0 has no effect. Each SET*n* always is read as a 0.
7 CLR*n*	Clear flag bits. CLR*n* is the flag bit. Writing a 1 to CLR*n* clear it provided that a 1 is not written simultaneously to its corresponding SET*n* bit. Writing a 0 to CLR*n* has no effect. 0 Interrupt request not pending within INTC. 1 Interrupt request pending within INTC.

10.3.1.6 INTC Priority Select Registers (INTC_PSR[0–211])

The INTC_PSR*n* support the selection of an individual priority for each source of interrupt request. The unique vector of each peripheral or software settable interrupt request determines which INTC_PSR*n* is assigned to that interrupt request. The software settable interrupt requests 0–7 are assigned vectors 0–7, and their priorities are configured in INTC_PSR0–INTC_PSR7, respectively. The peripheral interrupt

requests are assigned vectors 8–211 and their priorities are configured in INTC_PSR8 through INTC_PSR211, respectively.

Although INTC_PSR*n* is 8 bits wide, it can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

NOTE

Do not modify the PRI*n* field of an INTC_PSR*n* while its corresponding peripheral or software settable interrupt request is asserted.

Address: Base + 0x0040 + *n* (INTC_PSR*n*); *n* = 0–211 Access: User R/W

	0	1	2	3	4	5	6	7
R	0	0	0	0	PRI*n*			
W								
Reset	0	0	0	0	0	0	0	0

Figure 10-13. INTC Priority Select Registers (INTC_PSR*n*)

Table 10-8. INTC_SSCIR*n* Field Descriptions

Field	Description
0–3	Reserved, must be cleared.
4–7 PRI*n*	Priority select. Selects the priority for corresponding interrupt request. 1111 Priority 15 (highest) 1110 Priority 14 ... 0001 Priority 1 0000 Priority 0 (lowest)

10.4 Functional Description

10.4.1 Interrupt Request Sources

The INTC has two types of interrupt requests, peripheral and software settable. The assignments between the interrupt requests from the modules to the vectors for input to the e200z3 are shown in Table 10-9. The Offset column lists the IRQ specific offsets when using hardware vector mode. The Source column is written in C language syntax. The syntax is ‘module_register[bit].’ Interrupt requests from the same module location or ORed together. The individual interrupt priorities are selected in INTC_PSR*n*, where the specific select register is assigned according to the vector.

Table 10-9. INTC: Interrupt Request Sources

Hardware Vector Mode Offset	Vector	Source[1]	Description
Software			
0x0000	0	INTC_SSCIR0[CLR0]	INTC software settable Clear flag 0
0x0010	1	INTC_SSCIR1[CLR1]	INTC software settable Clear flag 1
0x0020	2	INTC_SSCIR2[CLR2]	INTC software settable Clear flag 2
0x0030	3	INTC_SSCIR3[CLR3]	INTC software settable Clear flag 3
0x0040	4	INTC_SSCIR4[CLR4]	INTC software settable Clear flag 4
0x0050	5	INTC_SSCIR5[CLR5]	INTC software settable Clear flag 5
0x0060	6	INTC_SSCIR6[CLR6]	INTC software settable Clear flag 6
0x0070	7	INTC_SSCIR7[CLR7]	INTC software settable Clear flag 7
Watchdog / ECC			
0x0080	8	ECSM_SWTIR[SWTIC]	ECSM Software Watchdog Interrupt flag
0x0090	9	ECSM_ESR[RNCE] ECSM_ESR[FNCE]	ECSM combined interrupt requests: Internal SRAM Non-Correctable Error and flash Non-Correctable Error
eDMA			
0x00A0	10	EDMA_ERL[ERR31:ERR0]	eDMA channel Error flags 31–0
0x00B0	11	EDMA_IRQRL[INT00]	eDMA channel Interrupt 0
0x00C0	12	EDMA_IRQRL[INT01]	eDMA channel Interrupt 1
0x00D0	13	EDMA_IRQRL[INT02]	eDMA channel Interrupt 2
0x00E0	14	EDMA_IRQRL[INT03]	eDMA channel Interrupt 3
0x00F0	15	EDMA_IRQRL[INT04]	eDMA channel Interrupt 4
0x0100	16	EDMA_IRQRL[INT05]	eDMA channel Interrupt 5
0x0110	17	EDMA_IRQRL[INT06]	eDMA channel Interrupt 6
0x0120	18	EDMA_IRQRL[INT07]	eDMA channel Interrupt 7
0x0130	19	EDMA_IRQRL[INT08]	eDMA channel Interrupt 8
0x0140	20	EDMA_IRQRL[INT09]	eDMA channel Interrupt 9
0x0150	21	EDMA_IRQRL[INT10]	eDMA channel Interrupt 10
0x0160	22	EDMA_IRQRL[INT11]	eDMA channel Interrupt 11
0x0170	23	EDMA_IRQRL[INT12]	eDMA channel Interrupt 12
0x0180	24	EDMA_IRQRL[INT13]	eDMA channel Interrupt 13
0x0190	25	EDMA_IRQRL[INT14]	eDMA channel Interrupt 14
0x01A0	26	EDMA_IRQRL[INT15]	eDMA channel Interrupt 15
0x01B0	27	EDMA_IRQRL[INT16]	eDMA channel Interrupt 16

Table 10-9. INTC: Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector	Source[1]	Description
0x01C0	28	EDMA_IRQRL[INT17]	eDMA channel Interrupt 17
0x01D0	29	EDMA_IRQRL[INT18]	eDMA channel Interrupt 18
0x01E0	30	EDMA_IRQRL[INT19]	eDMA channel Interrupt 19
0x01F0	31	EDMA_IRQRL[INT20]	eDMA channel Interrupt 20
0x0200	32	EDMA_IRQRL[INT21]	eDMA channel Interrupt 21
0x0210	33	EDMA_IRQRL[INT22]	eDMA channel Interrupt 22
0x0220	34	EDMA_IRQRL[INT23]	eDMA channel Interrupt 23
0x0230	35	EDMA_IRQRL[INT24]	eDMA channel Interrupt 24
0x0240	36	EDMA_IRQRL[INT25]	eDMA channel Interrupt 25
0x0250	37	EDMA_IRQRL[INT26]	eDMA channel Interrupt 26
0x0260	38	EDMA_IRQRL[INT27]	eDMA channel Interrupt 27
0x0270	39	EDMA_IRQRL[INT28]	eDMA channel Interrupt 28
0x0280	40	EDMA_IRQRL[INT29]	eDMA channel Interrupt 29
0x0290	41	EDMA_IRQRL[INT30]	eDMA channel Interrupt 30
0x02A0	42	EDMA_IRQRL[INT31]	eDMA channel Interrupt 31
PLL			
0x02B0	43	FMPLL_SYNSR[LOCF]	FMPLL Loss of Clock Flag
0x02C0	44	FMPLL_SYNSR[LOLF]	FMPLL Loss of Lock Flag
SIU			
0x02D0	45	SIU_OSR[OVF15:OVF0]	SIU combined overrun interrupt requests of the external interrupt Overrun Flags
0x02E0	46	SIU_EISR[EIF0]	SIU External Interrupt Flag 0
0x02F0	47	SIU_EISR[EIF1]	SIU External Interrupt Flag 1
0x0300	48	SIU_EISR[EIF2]	SIU External Interrupt Flag 2
0x0310	49	SIU_EISR[EIF3]	SIU External Interrupt Flag 3
0x0320	50	SIU_EISR[EIF15:EIF4]	SIU External Interrupt Flags 15–4
eMIOS			
0x0330	51	EMIOS_GFR[F0]	eMIOS channel 0 Flag
0x0340	52	EMIOS_GFR[F1]	eMIOS channel 1 Flag
0x0350	53	EMIOS_GFR[F2]	eMIOS channel 2 Flag
0x0360	54	EMIOS_GFR[F3]	eMIOS channel 3 Flag
0x0370	55	EMIOS_GFR[F4]	eMIOS channel 4 Flag

Table 10-9. INTC: Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector	Source[1]	Description
0x0380	56	EMIOS_GFR[F5]	eMIOS channel 5 Flag
0x0390	57	EMIOS_GFR[F6]	eMIOS channel 6 Flag
0x03A0	58	EMIOS_GFR[F7]	eMIOS channel 7 Flag
0x03B0	59	EMIOS_GFR[F8]	eMIOS channel 8 Flag
0x03C0	60	EMIOS_GFR[F9]	eMIOS channel 9 Flag
0x03D0	61	EMIOS_GFR[F10]	eMIOS channel 10 Flag
0x03E0	62	EMIOS_GFR[F11]	eMIOS channel 11 Flag
0x03F0	63	EMIOS_GFR[F12]	eMIOS channel 12 Flag
0x0400	64	EMIOS_GFR[F13]	eMIOS channel 13 Flag
0x0410	65	EMIOS_GFR[F14]	eMIOS channel 14 Flag
0x0420	66	EMIOS_GFR[F15]	eMIOS channel 15 Flag
eTPU A			
0x0430	67	ETPU_MCR[MGEA] ETPU_MCR[MGEB] ETPU_MCR[ILFA] ETPU_MCR[ILFB] ETPU_MCR[SCMMISF]	eTPU Global Exception
0x0440	68	ETPU_CISR_A[CIS0]	eTPU Engine A Channel 0 Interrupt Status
0x0450	69	ETPU_CISR_A[CIS1]	eTPU Engine A Channel 1 Interrupt Status
0x0460	70	ETPU_CISR_A[CIS2]	eTPU Engine A Channel 2 Interrupt Status
0x0470	71	ETPU_CISR_A[CIS3]	eTPU Engine A Channel 3 Interrupt Status
0x0480	72	ETPU_CISR_A[CIS4]	eTPU Engine A Channel 4 Interrupt Status
0x0490	73	ETPU_CISR_A[CIS5]	eTPU Engine A Channel 5 Interrupt Status
0x04A0	74	ETPU_CISR_A[CIS6]	eTPU Engine A Channel 6 Interrupt Status
0x04B0	75	ETPU_CISR_A[CIS7]	eTPU Engine A Channel 7 Interrupt Status
0x04C0	76	ETPU_CISR_A[CIS8]	eTPU Engine A Channel 8 Interrupt Status
0x04D0	77	ETPU_CISR_A[CIS9]	eTPU Engine A Channel 9 Interrupt Status
0x04E0	78	ETPU_CISR_A[CIS10]	eTPU Engine A Channel 10 Interrupt Status
0x04F0	79	ETPU_CISR_A[CIS11]	eTPU Engine A Channel 11 Interrupt Status
0x0500	80	ETPU_CISR_A[CIS12]	eTPU Engine A Channel 12 Interrupt Status
0x0510	81	ETPU_CISR_A[CIS13]	eTPU Engine A Channel 13 Interrupt Status
0x0520	82	ETPU_CISR_A[CIS14]	eTPU Engine A Channel 14 Interrupt Status
0x0530	83	ETPU_CISR_A[CIS15]	eTPU Engine A Channel 15 Interrupt Status
0x0540	84	ETPU_CISR_A[CIS16]	eTPU Engine A Channel 16 Interrupt Status

Table 10-9. INTC: Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector	Source[1]	Description
0x0550	85	ETPU_CISR_A[CIS17]	eTPU Engine A Channel 17 Interrupt Status
0x0560	86	ETPU_CISR_A[CIS18]	eTPU Engine A Channel 18 Interrupt Status
0x0570	87	ETPU_CISR_A[CIS19]	eTPU Engine A Channel 19 Interrupt Status
0x0580	88	ETPU_CISR_A[CIS20]	eTPU Engine A Channel 20 Interrupt Status
0x0590	89	ETPU_CISR_A[CIS21]	eTPU Engine A Channel 21 Interrupt Status
0x05A0	90	ETPU_CISR_A[CIS22]	eTPU Engine A Channel 22 Interrupt Status
0x05B0	91	ETPU_CISR_A[CIS23]	eTPU Engine A Channel 23 Interrupt Status
0x05C0	92	ETPU_CISR_A[CIS24]	eTPU Engine A Channel 24 Interrupt Status
0x05D0	93	ETPU_CISR_A[CIS25]	eTPU Engine A Channel 25 Interrupt Status
0x05E0	94	ETPU_CISR_A[CIS26]	eTPU Engine A Channel 26 Interrupt Status
0x05F0	95	ETPU_CISR_A[CIS27]	eTPU Engine A Channel 27 Interrupt Status
0x0600	96	ETPU_CISR_A[CIS28]	eTPU Engine A Channel 28 Interrupt Status
0x0610	97	ETPU_CISR_A[CIS29]	eTPU Engine A Channel 29 Interrupt Status
0x0620	98	ETPU_CISR_A[CIS30]	eTPU Engine A Channel 30 Interrupt Status
0x0630	99	ETPU_CISR_A[CIS31]	eTPU Engine A Channel 31 Interrupt Status
eQADC			
0x0640	100	EQADC_FISR*x*[TORF] EQADC_FISR*x*[RFOF] EQADC_FISR*x*[CFUF]	eQADC combined overrun interrupt request s from all of the FIFOs: Trigger Overrun, Receive FIFO Overflow, and command FIFO Underflow
0x0650	101	EQADC_FISR0[NCF]	eQADC command FIFO 0 Non-Coherency Flag
0x0660	102	EQADC_FISR0[PF]	eQADC command FIFO 0 Pause Flag
0x0670	103	EQADC_FISR0[EOQF]	eQADC command FIFO 0 command queue End of Queue Flag
0x0680	104	EQADC_FISR0[CFFF]	eQADC Command FIFO 0 Fill Flag
0x0690	105	EQADC_FISR0[RFDF]	eQADC Receive FIFO 0 Drain Flag
0x06A0	106	EQADC_FISR1[NCF]	eQADC command FIFO 1 Non-Coherency Flag
0x06B0	107	EQADC_FISR1[PF]	eQADC command FIFO 1 Pause Flag
0x06C0	108	EQADC_FISR1[EOQF]	eQADC command FIFO 1 command queue End of Queue Flag
0x06D0	109	EQADC_FISR1[CFFF]	eQADC Command FIFO 1 Fill Flag
0x06E0	110	EQADC_FISR1[RFDF]	eQADC Receive FIFO 1 Drain Flag
0x06F0	111	EQADC_FISR2[NCF]	eQADC command FIFO 2 Non-Coherency Flag
0x0700	112	EQADC_FISR2[PF]	eQADC command FIFO 2 Pause Flag

Table 10-9. INTC: Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector	Source[1]	Description
0x0710	113	EQADC_FISR2[EOQF]	eQADC command FIFO 2 command queue End of Queue Flag
0x0720	114	EQADC_FISR2[CFFF]	eQADC Command FIFO 2 Fill Flag
0x0730	115	EQADC_FISR2[RFDF]	eQADC Receive FIFO 2 Drain Flag
0x0740	116	EQADC_FISR3[NCF]	eQADC command FIFO 3 Non-Coherency Flag
0x0750	117	EQADC_FISR3[PF]	eQADC command FIFO 3 Pause Flag
0x0760	118	EQADC_FISR3[EOQF]	eQADC command FIFO 3 command queue End of Queue Flag
0x0770	119	EQADC_FISR3[CFFF]	eQADC Command FIFO 3 Fill Flag
0x0780	120	EQADC_FISR3[RFDF]	eQADC Receive FIFO 3 Drain Flag
0x0790	121	EQADC_FISR4[NCF]	eQADC command FIFO 4 Non-Coherency Flag
0x07A0	122	EQADC_FISR4[PF]	eQADC command FIFO 4 Pause Flag
0x07B0	123	EQADC_FISR4[EOQF]	eQADC command FIFO 4 command queue End of Queue Flag
0x07C0	124	EQADC_FISR4[CFFF]	eQADC Command FIFO 4 Fill Flag
0x07D0	125	EQADC_FISR4[RFDF]	eQADC Receive FIFO 4 Drain Flag
0x07E0	126	EQADC_FISR5[NCF]	eQADC command FIFO 5 Non-Coherency Flag
0x07F0	127	EQADC_FISR5[PF]	eQADC command FIFO 5 Pause Flag
0x0800	128	EQADC_FISR5[EOQF]	eQADC command FIFO 5 command queue End of Queue Flag
0x0810	129	EQADC_FISR5[CFFF]	eQADC Command FIFO 5 Fill Flag
0x0820	130	EQADC_FISR5[RFDF]	eQADC Receive FIFO 5 Drain Flag
DSPI B, DSPI C, DSPI D			
0x0830	131	DSPI_BSR[TFUF] DSPI_BSR[RFOF]	DSPI B combined overrun interrupt requests: Transmit FIFO Underflow and Receive FIFO Overflow
0x0840	132	DSPI_BSR[EOQF]	DSPI B transmit FIFO End of Queue Flag
0x0850	133	DSPI_BSR[TFFF]	DSPI B Transmit FIFO Fill Flag
0x0860	134	DSPI_BSR[TCF]	DSPI B Transfer Complete Flag
0x0870	135	DSPI_BSR[RFDF]	DSPI B Receive FIFO Drain Flag
0x0880	136	DSPI_CSR[TFUF] DSPI_CSR[RFOF]	DSPI C combined overrun interrupt requests: Transmit FIFO Underflow and Receive FIFO Overflow
0x0890	137	DSPI_CSR[EOQF]	DSPI C transmit FIFO End of Queue Flag
0x08A0	138	DSPI_CSR[TFFF]	DSPI C Transmit FIFO Fill Flag

Table 10-9. INTC: Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector	Source[1]	Description
0x08B0	139	DSPI_CSR[TCF]	DSPI C Transfer Complete Flag
0x08C0	140	DSPI_CSR[RFDF]	DSPI C Receive FIFO Drain Flag
0x08D0	141	DSPI_DSR[TFUF] DSPI_DSR[RFOF]	DSPI D combined overrun interrupt requests: Transmit FIFO Underflow and Receive FIFO Overflow
0x08E0	142	DSPI_DSR[EOQF]	DSPI D transmit FIFO End of Queue Flag
0x08F0	143	DSPI_DSR[TFFF]	DSPI D Transmit FIFO Fill Flag
0x0900	144	DSPI_DSR[TCF]	DSPI D Transfer Complete Flag
0x0910	145	DSPI_DSR[RFDF]	DSPI D Receive FIFO Drain Flag
eSCI			
0x0920	146	ESCIA_SR[TDRE] ESCIA_SR[TC] ESCIA_SR[RDRF] ESCIA_SR[IDLE] ESCIA_SR[OR] ESCIA_SR[NF] ESCIA_SR[FE] ESCIA_SR[PF] ESCIA_SR[BERR] ESCIA_SR[RXRDY] ESCIA_SR[TXRDY] ESCIA_SR[LWAKE] ESCIA_SR[STO] ESCIA_SR[PBERR] ESCIA_SR[CERR] ESCIA_SR[CKERR] ESCIA_SR[FRC] ESCIA_SR[OVFL]	Combined Interrupt Requests of ESCI Module A: Transmit Data Register Empty, Transmit Complete, Receive Data Register Full, Idle line, Overrun, Noise Flag, Framing Error Flag, and Parity Error Flag interrupt requests, SCI Status Register 2 Bit Error interrupt request, LIN Status Register 1 Receive Data Ready, Transmit Data Ready, Received LIN Wakeup Signal, Slave TimeOut, Physical Bus Error, CRC Error, Checksum Error, Frame Complete interrupts requests, and LIN Status Register 2 Receive Register Overflow
0x0930	147	Reserved	Reserved
0x0940	148	Reserved	Reserved

Table 10-9. INTC: Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector	Source[1]	Description
0x0950	149	ESCIB_SR[TDRE] ESCIB_SR[TC] ESCIB_SR[RDRF] ESCIB_SR[IDLE] ESCIB_SR[OR] ESCIB_SR[NF] ESCIB_SR[FE] ESCIB_SR[PF] ESCIB_SR[BERR] ESCIB_SR[RXRDY] ESCIB_SR[TXRDY] ESCIB_SR[LWAKE] ESCIB_SR[STO] ESCIB_SR[PBERR] ESCIB_SR[CERR] ESCIB_SR[CKERR] ESCIB_SR[FRC] ESCIB_SR[OVFL]	Combined Interrupt Requests of ESCI Module B: Transmit Data Register Empty, Transmit Complete, Receive Data Register Full, Idle line, Overrun, Noise Flag, Framing Error Flag, and Parity Error Flag interrupt requests, SCI Status Register 2 Bit Error interrupt request, LIN Status Register 1 Receive Data Ready, Transmit Data Ready, Received LIN Wakeup Signal, Slave TimeOut, Physical Bus Error, CRC Error, Checksum Error, Frame Complete interrupts requests, and LIN Status Register 2 Receive Register Overflow
0x0960	150	Reserved	Reserved
0x0970	151	Reserved	Reserved
FlexCAN A and FlexCAN C			
0x0980	152	CANA_ESR[BOFF_INT]	FLEXCAN A Bus Off Interrupt
0x0990	153	CANA_ESR[ERR_INT]	FLEXCAN A Error Interrupt
0x09A0	154	Reserved	Reserved
0x09B0	155	CANA_IFRL[BUF0]	FLEXCAN A Buffer 0 Interrupt
0x09C0	156	CANA_IFRL[BUF1]	FLEXCAN A Buffer 1 Interrupt
0x09D0	157	CANA_IFRL[BUF2]	FLEXCAN A Buffer 2 Interrupt
0x09E0	158	CANA_IFRL[BUF3]	FLEXCAN A Buffer 3 Interrupt
0x09F0	159	CANA_IFRL[BUF4]	FLEXCAN A Buffer 4 Interrupt
0x0A00	160	CANA_IFRL[BUF5]	FLEXCAN A Buffer 5 Interrupt
0x0A10	161	CANA_IFRL[BUF6]	FLEXCAN A Buffer 6 Interrupt
0x0A20	162	CANA_IFRL[BUF7]	FLEXCAN A Buffer 7 Interrupt
0x0A30	163	CANA_IFRL[BUF8]	FLEXCAN A Buffer 8 Interrupt
0x0A40	164	CANA_IFRL[BUF9]	FLEXCAN A Buffer 9 Interrupt
0x0A50	165	CANA_IFRL[BUF10]	FLEXCAN A Buffer 10 Interrupt
0x0A60	166	CANA_IFRL[BUF11]	FLEXCAN A Buffer 11 Interrupt
0x0A70	167	CANA_IFRL[BUF12]	FLEXCAN A Buffer 12 Interrupt
0x0A80	168	CANA_IFRL[BUF13]	FLEXCAN A Buffer 13 Interrupt
0x0A90	169	CANA_IFRL[BUF14]	FLEXCAN A Buffer 14 Interrupt

Table 10-9. INTC: Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector	Source[1]	Description
0x0AA0	170	CANA_IFRL[BUF15]	FLEXCAN A Buffer 15 Interrupt
0x0AB0	171	CANA_IFRL[BUF31I:BUF16]	FLEXCAN A Buffers 31–16 Interrupts
0x0AC0	172	CANA_IFRH[BUF63I:BUF32]	FLEXCAN A Buffers 63–32 Interrupts
0x0AD0	173	CANC_ESR[BOFF_INT]	FLEXCAN C Bus Off Interrupt
0x0AE0	174	CANC_ESR[ERR_INT]	FLEXCAN C Error Interrupt
0x0AF0	175	Reserved	Reserved
0x0B00	176	CANC_IFRL[BUF0]	FLEXCAN C Buffer 0 Interrupt
0x0B10	177	CANC_IFRL[BUF1]	FLEXCAN C Buffer 1 Interrupt
0x0B20	178	CANC_IFRL[BUF2]	FLEXCAN C Buffer 2 Interrupt
0x0B30	179	CANC_IFRL[BUF3]	FLEXCAN C Buffer 3 Interrupt
0x0B40	180	CANC_IFRL[BUF4]	FLEXCAN C Buffer 4 Interrupt
0x0B50	181	CANC_IFRL[BUF5]	FLEXCAN C Buffer 5 Interrupt
0x0B60	182	CANC_IFRL[BUF6]	FLEXCAN C Buffer 6 Interrupt
0x0B70	183	CANC_IFRL[BUF7]	FLEXCAN C Buffer 7 Interrupt
0x0B80	184	CANC_IFRL[BUF8]	FLEXCAN C Buffer 8 Interrupt
0x0B90	185	CANC_IFRL[BUF9]	FLEXCAN C Buffer 9 Interrupt
0x0BA0	186	CANC_IFRL[BUF10]	FLEXCAN C Buffer 10 Interrupt
0x0BB0	187	CANC_IFRL[BUF11]	FLEXCAN C Buffer 11 Interrupt
0x0BC0	188	CANC_IFRL[BUF12]	FLEXCAN C Buffer 12 Interrupt
0x0BD0	189	CANC_IFRL[BUF13]	FLEXCAN C Buffer 13 Interrupt
0x0BE0	190	CANC_IFRL[BUF14]	FLEXCAN C Buffer 14 Interrupt
0x0BF0	191	CANC_IFRL[BUF15]	FLEXCAN C Buffer 15 Interrupt
0x0C00	192	CANC_IFRL[BUF31:BUF16]	FLEXCAN C Buffers 31–16 Interrupts
0x0C10	193	CANC_IFRH[BUF63:BUF32]	FLEXCAN C Buffers 63–32 Interrupts
eMIOS			
0x0CA0	202	EMIOS_GFR[F16]	eMIOS channel 16 Flag
0x0CB0	203	EMIOS_GFR[F17]	eMIOS channel 17 Flag
0x0CC0	204	EMIOS_GFR[F18]	eMIOS channel 18 Flag
0x0CD0	205	EMIOS_GFR[F19]	eMIOS channel 19 Flag
0x0CE0	206	EMIOS_GFR[F20]	eMIOS channel 20 Flag
0x0CF0	207	EMIOS_GFR[F21]	eMIOS channel 21 Flag
0x0D00	208	EMIOS_GFR[F22]	eMIOS channel 22 Flag

Table 10-9. INTC: Interrupt Request Sources (continued)

Hardware Vector Mode Offset	Vector	Source[1]	Description
0x0D10	209	EMIOS_GFR[F23]	eMIOS channel 23 Flag
0x0D20	210	Reserved	Reserved
0x0D30	211	Reserved	Reserved

1 Interrupt requests from the same module location are ORed together.

NOTE

The INTC has no spurious vector support. If an asserted peripheral or software settable interrupt request:

- Has a PRI*n* value (INTC_PSR0–INTC_PSR211) higher than the PRI value in INTC_CPR; and
- Negates before the processor for that interrupt request acknowledges it

the IRQ to the processor can assert or remain asserted for that peripheral or software settable interrupt request. In this case, the interrupt vector for the peripheral or software settable IRQ remains, and the PRI value in the INTC_CPR is updated to the PRI*n* value in INTC_PSR*n*.

Clearing the peripheral interrupt request enable bit, or setting its mask bit has the same consequences as clearing its flag bit. Setting its enable bit or clearing its mask bit while its FLAG bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.

10.4.1.1 Peripheral Interrupt Requests

An interrupt event in a peripheral's hardware sets a flag bit which resides in that peripheral. The interrupt request from the peripheral is driven by that flag bit.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

10.4.1.2 Software Settable Interrupt Requests

The software set and clear interrupt registers (INTC_SSCIR*x*_*x*) support the setting or clearing of software-settable interrupt requests. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. With the exception of being set by software, this flag bit operates the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC just like a peripheral interrupt request.

An interrupt request is triggered by software writing a 1 to the SET*n* bit in INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7). This write sets the corresponding CLR*n* bit, which is a flag bit, resulting in the interrupt request. The interrupt request is cleared by writing a 1 to the CLR*n* bit. Specific operations includes the following:

- Writing a 1 to SET*n* leaves SET*n* unchanged at '0' but sets the flag bit (which is the CLR*n* bit).

- Writing a 0 to SET*n* has no effect.
- Writing a 1 to CLR*n* clears the flag (CLRx) bit.
- Writing a 0 to CLR*n* has no effect.
- If a 1 is written to a pair of SET*n* and CLR*n* bits at the same time, the flag (CLRx) is set, regardless of whether CLR*n* was asserted before the write.

The time from the write to the SET*n* bit to the time that the INTC starts to drive the interrupt request to the processor is four clocks.

10.4.1.3 Unique Vector for Each Interrupt Request Source

Each peripheral and software settable interrupt request is assigned a hardwired unique 9-bit vector. Software settable interrupts 0–7 are assigned vectors 0–7, respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to cover all of the peripheral interrupt requests.

10.4.2 Priority Management

The asserted interrupt requests are compared to each other based on their PRI*n* values in INTC priority select registers (INTC_PSR0–INTC_PSR211). The result of that comparison also is compared to PRI in INTC current priority register (INTC_CPR). The results of those comparisons are used to manage the priority of the ISR being executed by the processor. The LIFO also assists in managing that priority.

10.4.2.1 Current Priority and Preemption

The priority arbitrator, selector, encoder, and comparator submodules shown in Figure 10-1 are used to compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software settable interrupt request is higher than the current priority, then the interrupt request to the processor is asserted. Also, a unique vector for the preempting peripheral or software settable interrupt request is generated for INTC interrupt acknowledge register (INTC_IACKR), and if in hardware vector mode, for the interrupt vector provided to the processor.

10.4.2.1.1 Priority Arbitrator Submodule

The priority arbitrator submodule compares all the priorities of all of the asserted interrupt requests, both peripheral and software settable. The output of the priority arbitrator submodule is the highest of those priorities. Also, any interrupt requests which have this highest priority are output as asserted interrupt requests to the request selector submodule.

10.4.2.1.2 Request Selector Submodule

If only one interrupt request from the priority arbitrator submodule is asserted, then it is passed as asserted to the vector encoder submodule. If multiple interrupt requests from the priority arbitrator submodule are asserted, then only the one with the lowest vector is passed as asserted to the vector encoder submodule. The lower vector is chosen regardless of the time order of the assertions of the peripheral or software settable interrupt requests.

10.4.2.1.3 Vector Encoder Submodule

The vector encoder submodule generates the unique 9-bit vector for the asserted interrupt request from the request selector submodule.

10.4.2.1.4 Priority Comparator Submodule

The priority comparator submodule compares the highest priority output from the priority arbitrator submodule with PRI in INTC_CPR. If the priority comparator submodule detects that this highest priority is higher than the current priority, then it asserts the interrupt request to the processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in INTC_CPR or the PRI value in INTC_CPR is lowered below this highest priority. This highest priority then becomes the new priority, which is written to PRI in INTC_CPR when the interrupt request to the processor is acknowledged. Interrupt requests with the PRI*n* in INTC_PSR*n* set to zero do not cause a preemption because their PRI*n* are not higher than PRI in INTC_CPR.

10.4.2.2 LIFO

The LIFO stores the preempted PRI values from the INTC_CPR. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the INTC_CPR does not need to be loaded from the INTC_CPR and stored onto the context stack. Likewise at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the INTC_CPR.

The PRI value in the INTC_CPR is pushed onto the LIFO when the INTC_IACKR is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode. The priority is popped into PRI in the INTC_CPR whenever the INTC_EOIR is written.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR equal to 15 is not preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority 0 is not needed because of how pushing onto a full LIFO and popping an empty LIFO operate.

- If the LIFO is pushed 15 or more times than it is popped, the priorities first pushed are overwritten (priority 0 is overwritten).
- If the LIFO pops more times than it is pushed, the popped priorities are 0.

Therefore, although a priority 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory mapped.

10.4.3 Details on Handshaking with Processor

10.4.3.1 Software Vector Mode Handshaking

10.4.3.1.1 Acknowledging Interrupt Request to Processor

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in Figure 10-14. The INTC examines the peripheral and software settable interrupt requests. When it finds an asserted peripheral or software settable interrupt request with a higher priority than PRI in INTC current priority register (INTC_CPR), it asserts the interrupt request to the processor. The INTVEC field in INTC interrupt acknowledge register (INTC_IACKR) is updated with the preempting interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The rest of the handshaking is described in Section 10.1.4.1, "Software Vector Mode."

10.4.3.1.2 End-of-Interrupt Exception Handler

Before the interrupt exception handling completes, INTC end-of-interrupt register (INTC_EOIR) must be written. When it is written, the LIFO is popped so that the preempted priority is restored into PRI of the INTC_CPR. Before it is written, the peripheral or software settable flag bit must be cleared so that the peripheral or software settable interrupt request is negated.

NOTE

To ensure proper operation across all devices, execute an MBAR or MSYNC instruction between the access to clear the flag bit and the write to the INTC_EOIR.

When returning from the preemption, the INTC does not search for the peripheral or software settable interrupt request whose ISR was preempted. Depending on how much the ISR progressed, that interrupt request can be asserted. When PRI in INTC_CPR is decreased to the priority of the preempted ISR, the interrupt request for the preempted ISR or any other asserted peripheral or software settable interrupt request at or less than that priority does not cause a preemption. Instead, after the restoration of the preempted context, the processor returns to the instruction address for the next ISR to execute before it is preempted. This next instruction is part of the preempted ISR or the interrupt exception handler's prolog or epilog.

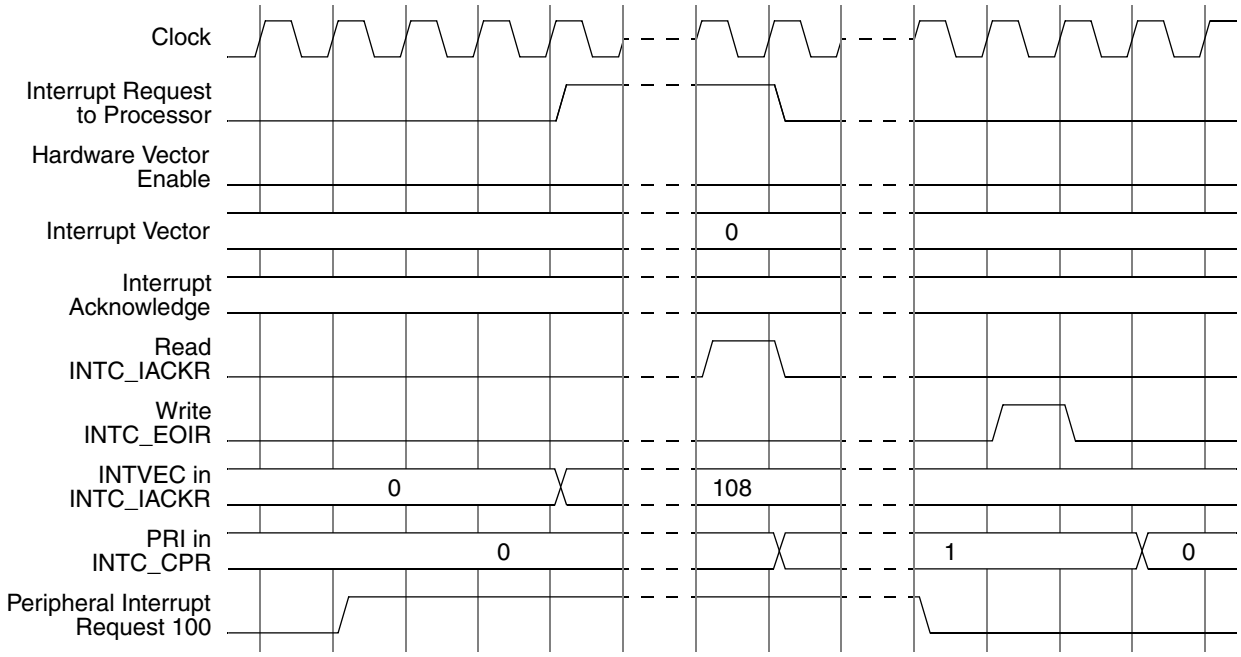


Figure 10-14. Software Vector Mode Handshaking Timing Diagram

10.4.3.2 Hardware Vector Mode Handshaking

A timing diagram of the interrupt request and acknowledge handshaking in hardware vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in Figure 10-15. As in software vector mode, the INTC examines the peripheral and software settable interrupt requests, and when it finds an asserted one with a higher priority than PRI in INTC_CPR, it asserts the interrupt request to the processor. The INTVEC field in the INTC_IACKR is updated with the preempting peripheral or software settable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. In addition, the value of the interrupt vector to the processor matches the value of the INTVEC field in the INTC_IACKR. The rest of the handshaking is described in Section 10.1.4.2, "Hardware Vector Mode."

The handshaking near the end of the interrupt exception handler, that is the writing to the INTC_EOIR, is the same as in software vector mode. See Section 10.4.3.1.2, "End-of-Interrupt Exception Handler."

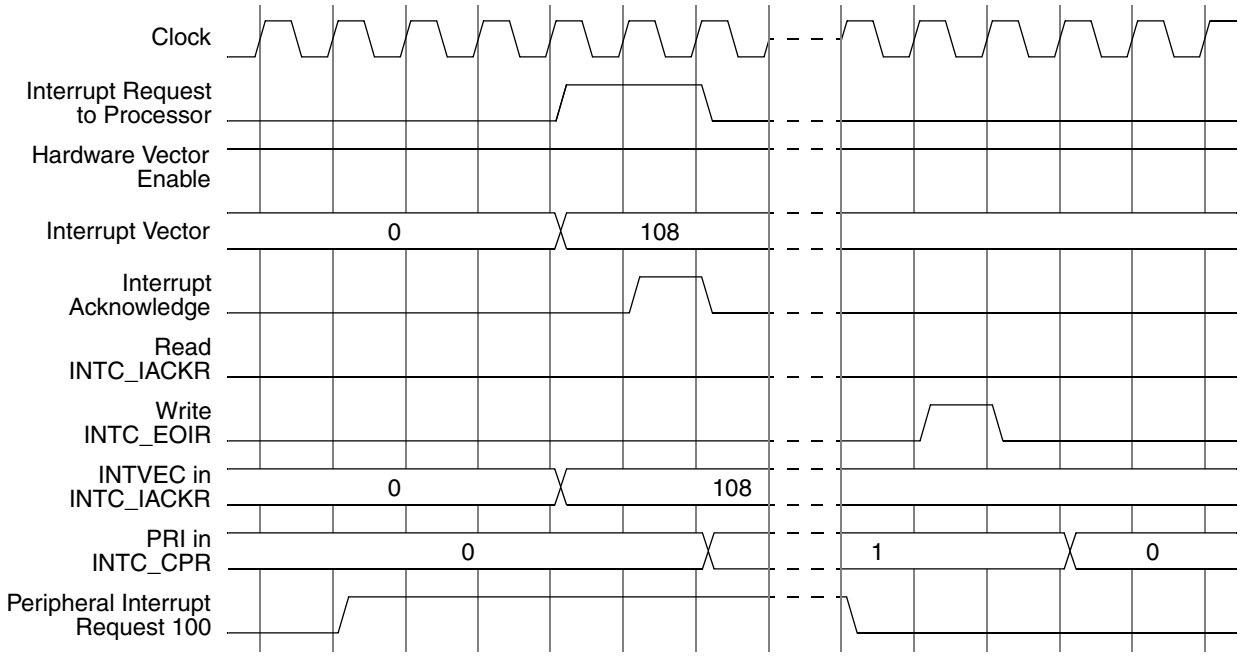


Figure 10-15. Hardware Vector Mode Handshaking Timing Diagram

10.5 Initialization/Application Information

10.5.1 Initialization Flow

After exiting reset, all of the PRI*n* fields in INTC priority select registers (INTC_PSR0–INTC_PSR211) are zero, and PRI in INTC current priority register (INTC_CPR) is 15. These reset values prevent INTC from asserting the interrupt request to the processor. The enable or mask bits in the peripherals are reset such that the peripheral interrupt requests are negated. An initialization sequence that allows peripheral and software settable interrupt requests to generate an interrupt request to the processor follows:

Interrupt request initialization

1. Configure the VTES and HVEN fields in the master control register INTC_MCR
2. Configure the VTBA field in INTC_IACKR
3. Raise the PRI*n* fields in INTC_PSR*n*
4. Set the enable bits or clear the mask bits for the peripheral interrupt requests
5. Clear the PRI field in INTC_CPR to zero
6. Enable processor recognition of interrupts

10.5.2 Interrupt Exception Handler

These example interrupt exception handlers use PowerPC Book E assembly code.

10.5.2.1 Software Vector Mode

```
interrupt_exception_handler:
code to create stack frame, save working register, and save SRR0 and SRR1

lis      r3,INTC_IACKR@ha           # form adjusted upper half of INTC_IACKR address
lwz      r3,INTC_IACKR@l(r3)        # load INTC_IACKR, which clears request to processor
lwz      r3,0x0(r3)                 # load address of ISR from vector table
wrteei   1                          # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

mtlr     r3                         # move address of ISR into link register
blrl                                # branch to ISR; link register updated with epilog
                                    # address

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth at the cost of
# postponing the servicing of the next interrupt request.
mbar                                # ensure store to clear flag bit has completed
lis      r3,INTC_EOIR@ha            # form adjusted upper half of INTC_EOIR address
li       r4,0x0                     # form 0 to write to INTC_EOIR
wrteei   0                          # disable processor recognition of interrupts
stw      r4,INTC_EOIR@l(r3)         # store to INTC_EOIR, informing INTC to lower priority

code to restore SRR0 and SRR1, restore working registers, and delete stack frame

rfi


vector_table_base_address:
address of ISR for interrupt with vector 0
address of ISR for interrupt with vector 1
         .
         .
         .
address of ISR for interrupt with vector 510
address of ISR for interrupt with vector 511


ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                                 # return to epilog
```

10.5.2.2 Hardware Vector Mode

This interrupt exception handler is useful with processor and system bus implementations that support a hardware vector. This example assumes that each `interrupt_exception_handlerx` only has space for four instructions, and therefore a branch to `interrupt_exception_handler_continuedx` is needed.

```
interrupt_exception_handlerx:
b       interrupt_exception_handler_continuedx# 4 instructions available, branch to continue


interrupt_exception_handler_continuedx:
code to create stack frame, save working register, and save SRR0 and SRR1

wrteei   1                          # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

bl       ISRx                       # branch to ISR for interrupt with vector x


epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth at the cost of
# postponing the servicing of the next interrupt request.
mbar                                # ensure store to clear flag bit has completed
lis      r3,INTC_EOIR@ha            # form adjusted upper half of INTC_EOIR address
li       r4,0x0                     # form 0 to write to INTC_EOIR
wrteei   0                          # disable processor recognition of interrupts
stw      r4,INTC_EOIR@l(r3)         # store to INTC_EOIR, informing INTC to lower priority

code to restore SRR0 and SRR1, restore working registers, and delete stack frame

rfi


ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                                 # branch to epilog
```

10.5.3 ISR, RTOS, and Task Hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC current priority register (INTC_CPR) having a value of 0. The RTOS executes the tasks according to the its current priority scheme, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC_CPR priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task’s priority can be elevated in the INTC_CPR while the shared resource is being accessed.

An ISR whose PRI*n* in INTC priority select registers (INTC_PSR0–INTC_PSR211) has a value of 0 does not cause an interrupt request to the processor, even if its peripheral or software settable interrupt request is asserted. For a peripheral interrupt request, not setting its enable bit or disabling the mask bit causes it to remain deasserted, which does not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR is not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

10.5.4 Order of Execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software settable interrupt requests. However, if multiple peripheral or software settable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, the INTC selects the one with the lowest unique vector regardless of the order in time that they asserted. However, the ability to meet deadlines with this scheduling scheme is no less than if the ISRs execute in the time order that their peripheral or software settable interrupt requests asserted.

The example in Table 10-10 shows the order of execution of both ISRs with different priorities and the same priority.

Table 10-10. Order of ISR Execution Example

Step	Step Description	Code Executing At End of Step						PRI in INTC_CPR at End of Step
		RTOS	ISR108[1]	ISR208	ISR308	ISR408	Interrupt Exception Handler	
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 is asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 is asserts.					X		4
5	Peripheral interrupt request 200 at priority 3 is asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3

Table 10-10. Order of ISR Execution Example (continued)

Step	Step Description	Code Executing At End of Step						PRI in INTC_CPR at End of Step
		RTOS	ISR108[1]	ISR208	ISR308	ISR408	Interrupt Exception Handler	
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR.						X	0
12	RTOS continues execution.	X						0

1 ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software settable interrupt requests.

10.5.5 Priority Ceiling Protocol

10.5.5.1 Elevating Priority

The PRI field in INTC current priority register (INTC_CPR) is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol therefore allows coherent accesses of the ISRs to that shared resource.

For example, ISR1 has a priority of 1, ISR2 has a priority of 2, and ISR3 has a priority of 3. They all share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC_CPR to 3, the ceiling of all of the ISR priorities. After they release the resource, they must lower the PRI value in INTC_CPR to prevent further scheduling inefficiencies. If they do not raise their priority, then ISR2 can preempt ISR1, and ISR3 can preempt ISR1 or ISR2, possibly corrupting the shared resource. Another possible failure mechanism is deadlock if the higher priority ISR needs the lower priority ISR to release the resource before it can continue, but the lower priority ISR can not release the resource until the higher priority ISR completes and execution returns to the lower priority ISR.

Using the PCP instead of disabling processor recognition of all interrupts reduces the time used by scheduling inefficiencies when accessing a shared resource. For example, while ISR3 can not preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

10.5.5.2 Ensuring Coherency

A scenario can exist that can cause non-coherent accesses to the shared resource. As an example, ISR1 and ISR2 both share a resource. ISR1 has a lower priority than ISR2. ISR1 is executing, and it writes to the INTC_CPR. The instruction following this store is a store to a value in a shared coherent data block. Either just before or at the same time as the first store, the INTC asserts the interrupt request to the processor because the peripheral interrupt request for ISR2 has asserted. As the processor is responding to the interrupt request from the INTC, and as it is terminating transactions and flushing its pipeline, it is possible

for both of these stores to execute. ISR2 attempts to access the data block coherently, but the data block has been corrupted.

OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent corrupting a coherent data block, use these same system services with the following code to modify the PRI in INTC_CPR. Interrupts must be enabled before executing the following the GetResource code sequence.

```
GetResource:
raise PRI
mbar
isync

ReleaseResource:
mbar
lower PRI
```

10.5.6 Selecting Priorities According to Request Rates and Deadlines

The selection of the priorities for the ISRs can be made using rate monotonic scheduling (RMS) or a superset of it, deadline monotonic scheduling (DMS). In RMS, the ISRs with the higher request rates have higher priorities.

In DMS, if the ISR deadline is set to occur before the next request for the ISR, then the ISR priority is assigned according to the time from the request for the IRS to the deadline, not from the time of the ISR request to the next ISR request for it. For example, ISR1 executes every 100 µs, ISR2 executes every 200 µs, and ISR3 executes every 300 µs. ISR1 has a higher priority than ISR2, which has a higher priority than ISR3. However, if ISR3 has a deadline of 150 µs, then its priority is higher than ISR2.

The INTC has 16 priorities, which can be less than the number of ISRs. In this case, group the priority ISRs with other ISRs that have similar deadlines. For example, a priority can be allocated every time the request rate doubles. ISRs with the same approximate request rates can share a priority:

- ISRs with request rates of approximately 1 ms
- ISRs with request rates of approximately 500 µs
- ISRs with request rates of approximately 250 µs

Using this approach, a 2^{16} range of ISR request rates can be prioritized, regardless of the number of ISRs.

Reducing the number of priorities can cause scheduling inefficiencies which reduces the processor's ability to meet its deadlines. It also allows easier management of ISRs with similar deadlines that share a resource. They can be placed at the same priority without any further scheduling inefficiencies, and they do not need to use the PCP to access the shared resource.

10.5.7 Software Settable Interrupt Requests

The software settable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and for processors to interrupt other processors in a multiple processor system.

10.5.7.1 Scheduling a Lower Priority Portion of an ISR

A portion of an ISR needs to be executed at the PRI*n* value in INTC priority select registers (INTC_PSR0–INTC_PSR211), which becomes the PRI value in INTC current priority register (INTC_CPR) with the interrupt acknowledgement. The ISR, however, can have a portion of it which does not need to be executed at this higher priority. Therefore, executing this later portion which does not need to be executed at this higher priority can block the execution of ISRs which do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. These scheduling inefficiencies reduce the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a SET*n* bit in INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7). Writing a 1 to SET*n* causes a software settable interrupt request. This software settable interrupt request, which usually has a lower PRI*n* value in the INTC_PSR*n*, does not cause scheduling inefficiencies.

10.5.7.2 Scheduling an ISR on Another Processor

Since the SET*n* bits in the INTC_SSCIR*n* are memory mapped, processors in multiple processor systems can schedule ISRs on the other processors. One application is that one processor simply wants to command another processor to perform a piece of work, and the initiating processor does not need to use the results of that work. If the initiating processor is concerned that processor executing the software settable ISR has not completed the work before asking it to again execute that ISR, it can check if the corresponding CLR*n* bit in INTC_SSCIR*n* is asserted before again writing a 1 to the SET*n* bit.

Another application is the sharing of a block of data. For example, a first processor has completed accessing a block of data and wants a second processor to then access it. Furthermore, after the second processor has completed accessing the block of data, the first processor again wants to access it. The accesses to the block of data must be done coherently. The procedure is that the first processor writes a 1 to a SET*n* bit on the second processor. The second processor, after accessing the block of data, clears the corresponding CLR*n* bit and then writes 1 to a SET*n* bit on the first processor, informing it that it now can access the block of data.

10.5.8 Lowering Priority Within an ISR

In implementations without the software-settable interrupt requests in the INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7), the only way—besides scheduling a task through an RTOS—to prevent scheduling inefficiencies with an ISR whose work spans multiple priorities (as described in Section 10.5.7.1, "Scheduling a Lower Priority Portion of an ISR,") is to lower the current priority. However, the INTC has a LIFO whose depth is determined by the number of priorities.

NOTE

Lowering the PRI value in INTC current priority register (INTC_CPR) within an ISR to less than the ISR corresponding PRI value in INTC priority select registers (INTC_PSR0–INTC_PSR211) allows more preemptions than the depth of the LIFO can support.

Therefore, the INTC does not support lowering the current priority within an ISR as a way to avoid scheduling inefficiencies.

10.5.9 Negating an Interrupt Request Outside of its ISR

10.5.9.1 Negating an Interrupt Request as a Side Effect of an ISR

Some peripherals have flag bits which can be cleared as a side effect of servicing a peripheral interrupt request. For example, reading a specific register can clear the flag bits, and consequently their corresponding interrupt requests too. This clearing as a side effect of servicing a peripheral interrupt request can cause the negation of other peripheral interrupt requests besides the peripheral interrupt request whose ISR presently is executing. This negating of a peripheral interrupt request outside of its ISR can be a desired effect.

10.5.9.2 Negating Multiple Interrupt Requests in One ISR

An ISR can clear other flag bits besides its own flag bit. One reason that an ISR clears multiple flag bits is because it serviced those other flag bits, and therefore the ISRs for these other flag bits do not need to be executed.

10.5.9.3 Proper Setting of Interrupt Request Priority

Whether an interrupt request negates outside of its own ISR due to the side effect of an ISR execution or the intentional clearing a flag bit, the priorities of the peripheral or software settable interrupt requests for these other flag bits must be selected correctly. Their PRI*n* values in INTC priority select registers (INTC_PSR0–INTC_PSR211) must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits still can cause the interrupt request to the processor to assert. Furthermore, the clearing of these other flag bits also has the same timing relationship to the writing to INTC end-of-interrupt register (INTC_EOIR) as the clearing of the flag bit that caused the present ISR to be executed. See Section 10.4.3.1.2, “End-of-Interrupt Exception Handler,” for more information.

A flag bit whose enable bit or mask bit is negating its peripheral interrupt request can be cleared at any time, regardless of the peripheral interrupt request’s PRI*n* value in INTC_PSR*n*.

10.5.10 Examining LIFO Contents

Normally you do not need to know the contents of the LIFO, or even how deep the LIFO is nested. Although the LIFO contents are not memory mapped, you can read the contents by popping the LIFO and reading the PRI field in the INTC current priority register (INTC_CPR). Disabling processor recognition of interrupts while examining the LIFO contents provides a coherent view of the preempted priorities. The code sequence is:

```
pop_lifo:
store to INTC_EOIR
load INTC_CPR, examine PRI, and store onto stack
if PRI is not zero or value when interrupts were enabled, branch to pop_lifo
```

When you are finished examining the LIFO contents, you can restore it in software vector mode using the following code sequence. In hardware vector mode, reading the INTC_IACKR does not push the INTC_CPR[PRI] onto the LIFO, therefore the LIFO contents cannot be restored in hardware vector mode.

```
push_lifo:
load stacked PRI value and store to INTC_CPR
load INTC_IACKR
if stacked PRI values are not depleted, branch to push_lifo
```

NOTE

Reading the INTC_IACKR acknowledges the interrupt request to the processor and updates the INTC_CPR[PRI] with the priority of the preempting interrupt request. If the processor recognition of interrupts is disabled during the LIFO restoration, interrupt requests to the processor can go undetected. However, since the peripheral or software settable interrupt requests are not cleared, the peripheral interrupt request to the processor re-asserts when INTC_CPR[PRI] is lower than the priorities of those peripheral or software settable interrupt requests.

Chapter 11
Frequency Modulated Phase Locked Loop and System Clocks (FMPLL)

11.1 Introduction

This section describes the features and function of the FMPLL module.

11.1.1 Block Diagrams

This section contains block diagrams that illustrate the FMPLL, the clock architecture, and the various FMPLL and clock configurations that are available. The following diagrams are provided:

- Figure 11-1, “FMPLL and Clock Architecture”
- Figure 11-2, “FMPLL Bypass Mode”
- Figure 11-3, “FMPLL External Reference Mode”
- Figure 11-4, “FMPLL Crystal Reference Mode Without FM”
- Figure 11-5, “FMPLL Crystal Reference Mode With FM”
- Figure 11-6, “FMPLL Dual-Controller (1:1) Mode”

11.1.1.1 FMPLL and Clock Architecture

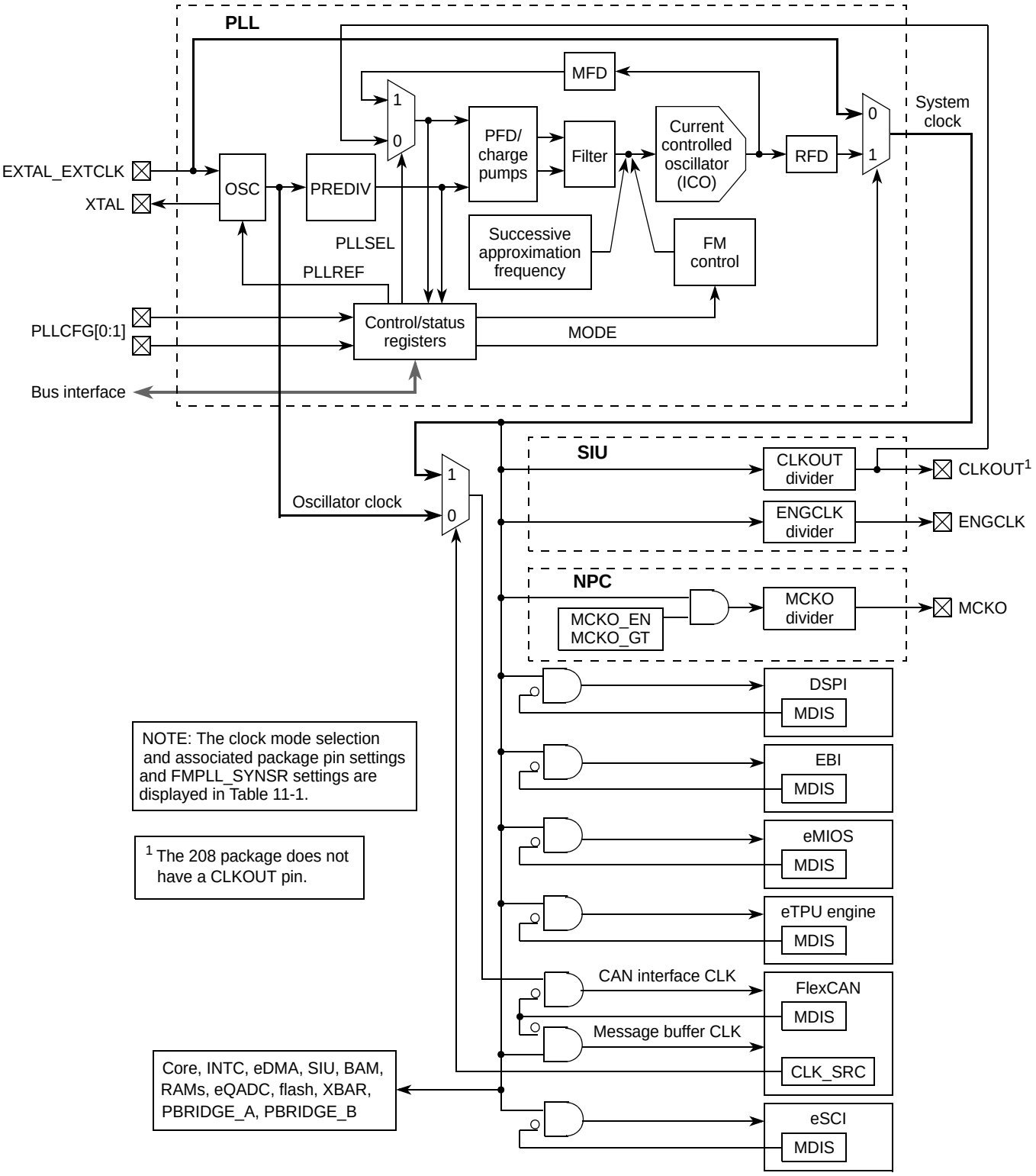


Figure 11-1. FMPLL Block and Clock Architecture

11.1.1.2 FMPLL Bypass Mode

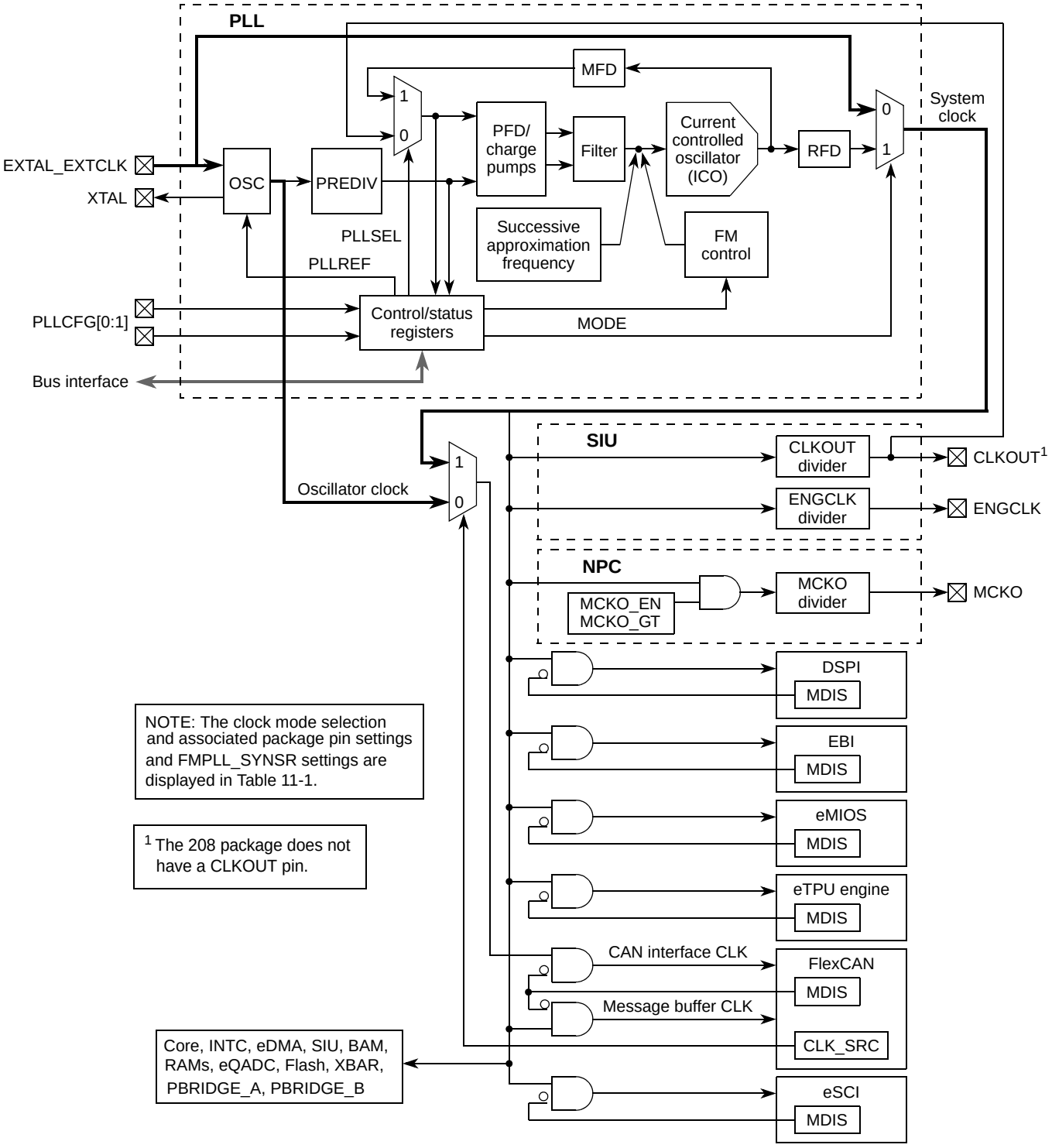


Figure 11-2. FMPLL Bypass Mode

11.1.1.3 FMPLL External Reference Mode

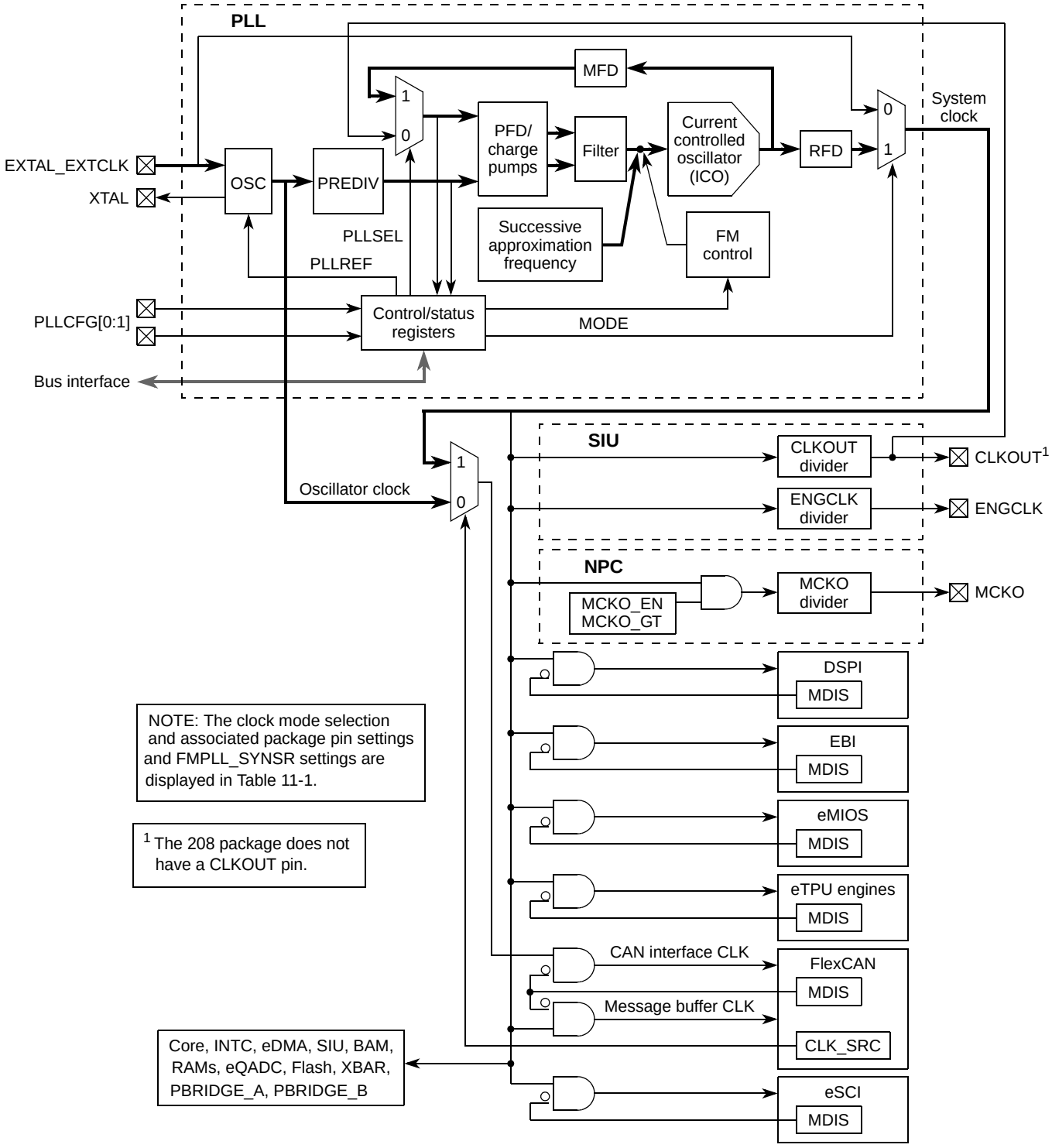


Figure 11-3. FMPLL External Reference Mode

11.1.1.4 FMPLL Crystal Reference Mode Without FM

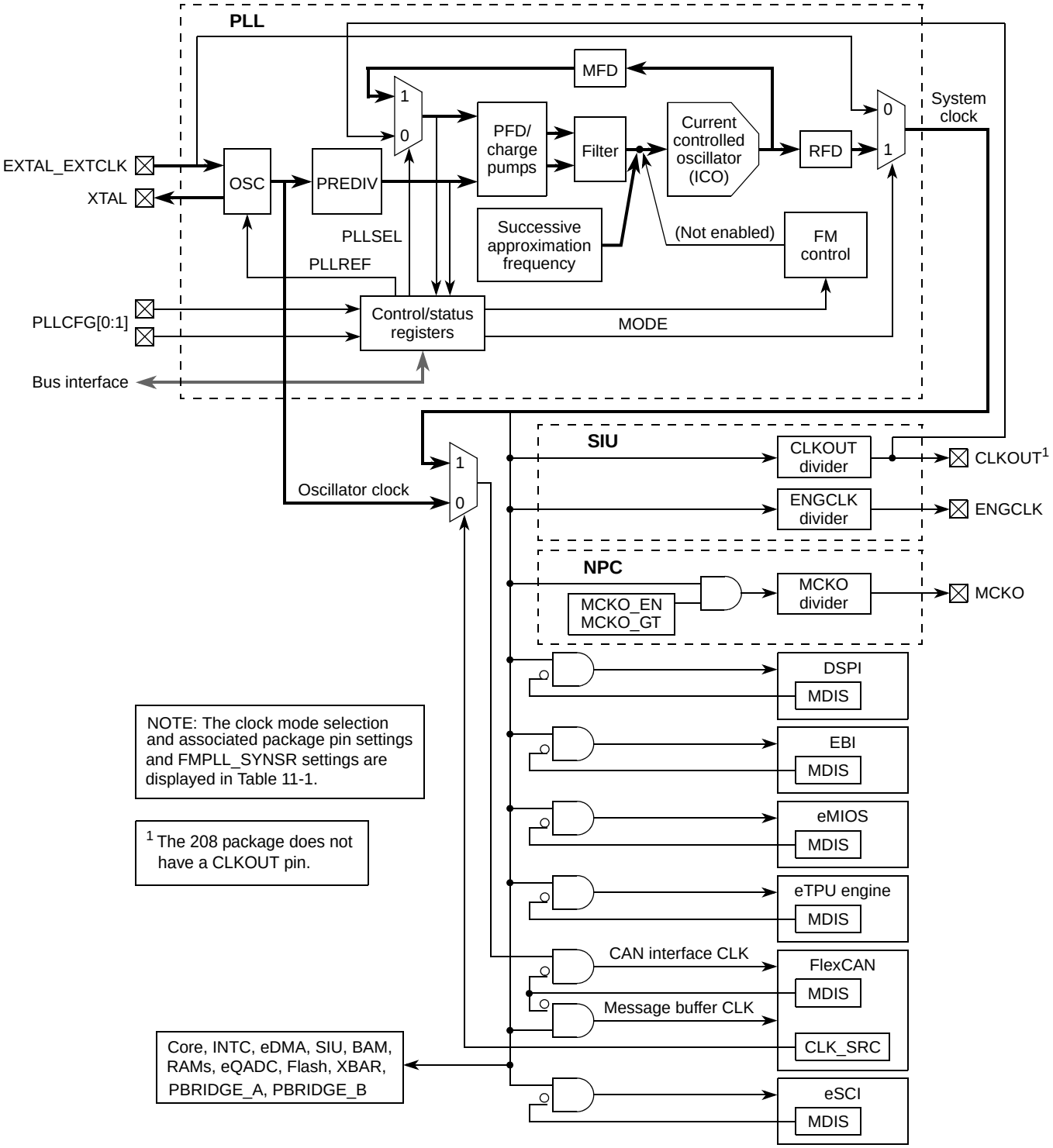


Figure 11-4. FMPLL Crystal Reference Mode without FM

11.1.1.5 FMPLL Crystal Reference Mode With FM

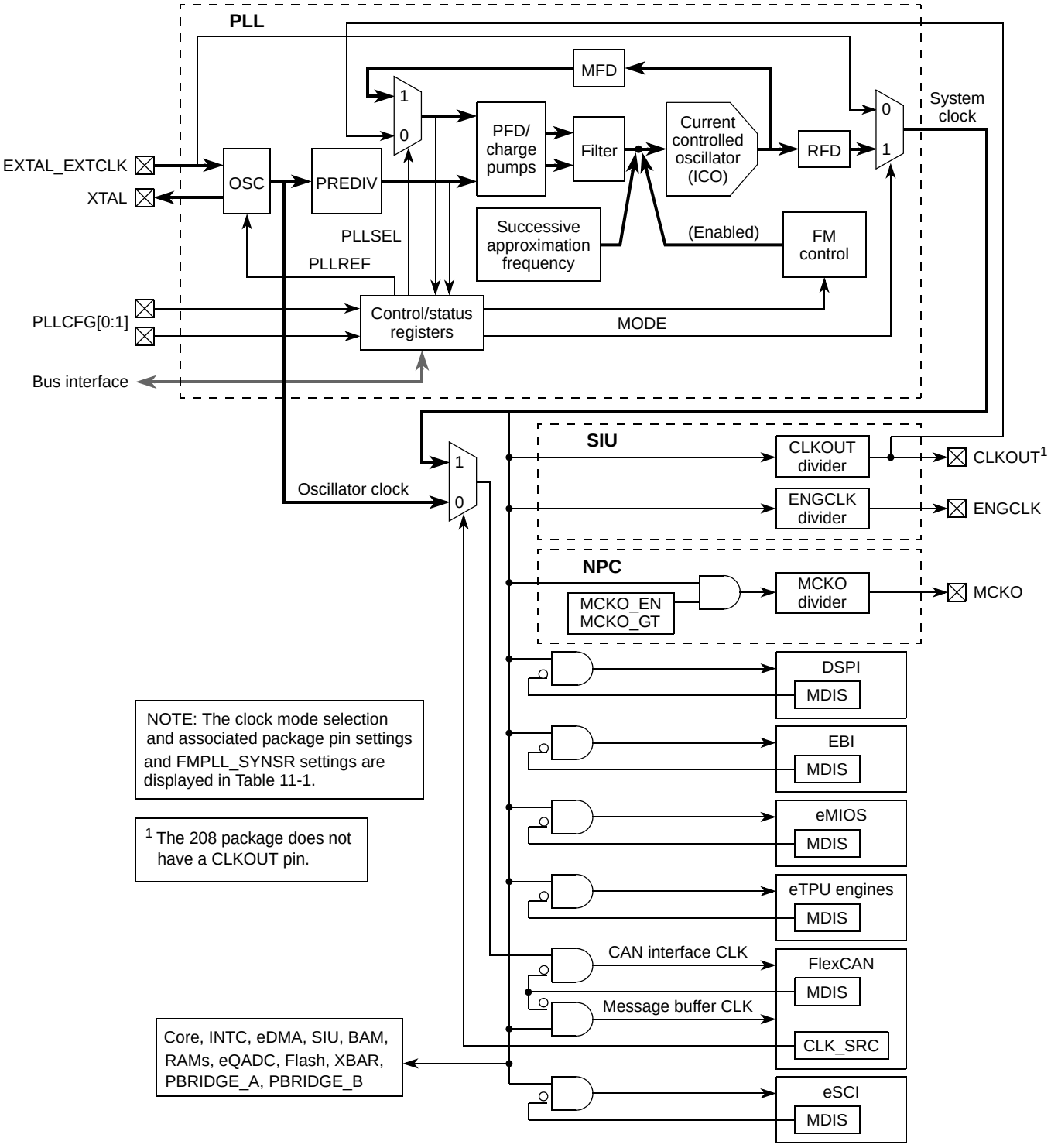


Figure 11-5. FMPLL Crystal Reference Mode with FM

11.1.1.6 FMPLL Dual-Controller Mode (1:1)

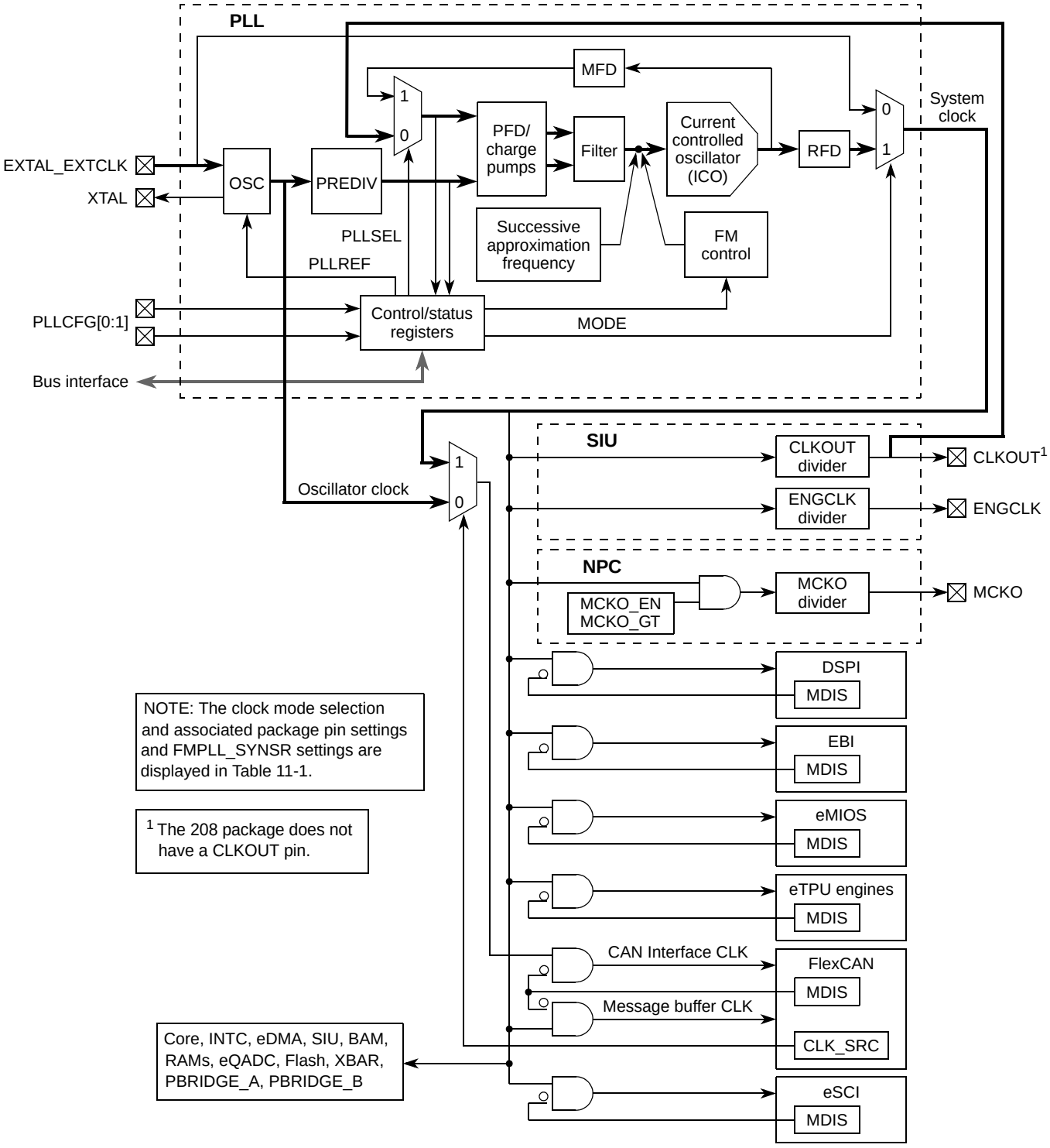


Figure 11-6. FMPLL Dual Controller (1:1) Mode

11.1.2 Overview

The frequency modulated phase locked loop (FMPLL) allows you to generate high speed system clocks from an 8–20 MHz crystal oscillator or from an external clock generator. Further, the FMPLL supports programmable frequency modulation of the system clock. The FMPLL multiplication factor, reference clock pre-divider factor, output clock divider ratio, modulation depth, and modulation rate are all controllable through a bus interface.

11.1.3 Features

The FMPLL has the following major features:

- Input clock frequency from 8–20 MHz
- Current controlled oscillator (ICO) range from 48 MHz to maximum device frequency
- Reference frequency pre-divider (PREDIV) for finer frequency synthesis resolution
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the FMPLL to re-lock
- Four modes of operation:
  - Bypass mode.
  - Crystal reference mode. This is the default mode for the 324 package (with or without the 496 assembly). See Section 11.1.4.1, “Crystal Reference.”
  - External reference mode. See Section 11.1.4.2, “External Reference Mode.”
  - PLL dual-controller (1:1) mode for EXTAL_EXTCLK to CLKOUT skew minimization.
- Programmable frequency modulation
  - Modulation enabled/disabled via bus interface
  - Triangle wave modulation
  - Register programmable modulation depth (±1% to ±2% deviation from center frequency)
  - Register programmable modulation frequency dependent on reference frequency; limited to 100–250 MHz.
- Lock detect circuitry reports when the FMPLL has achieved frequency lock and continuously monitors lock status to report loss of lock conditions
  - User-selectable ability to generate an interrupt request upon loss of lock. See Chapter 10, “Interrupt Controller (INTC),” for details.
  - User-selectable ability to generate a system reset upon loss of lock. See Chapter 4, “Reset,” for details.
- Loss-of-clock (LOC) detection for reference and feedback clocks
  - User-selectable ability to generate an interrupt request upon loss of clock. See Chapter 10, “Interrupt Controller (INTC),” for details.
  - User-selectable ability to generate a system reset upon loss of clock See Chapter 4, “Reset,” for details.
- Self-clocked mode (SCM) operation in event of input clock failure

11.1.4 FMPLL Modes of Operation

The FMPLL operational mode is configured during reset. Table 11-1 shows clock mode selection during reset configuration. Additional information on reset configuration options for the FMPLL are in Chapter 4, "Reset."

Table 11-1. Clock Mode Selection

Clock Mode	Package Pins			Synthesizer Status Register (FMPLL_SYNSR)[1] Bits		
	$\overline{\text{RSTCFG}}$[2]	PLLCFG[0]	PLLCFG[1]	MODE	PLLSEL	PLLREF
Crystal reference (324 package only)	1	PLLCFG pins ignored.		1	1	1
	0	1	0			
External reference	0	0	1	1	1	0
Bypass	0	0	0	0	0	0
Dual-controller	0	1	1	1	0	0

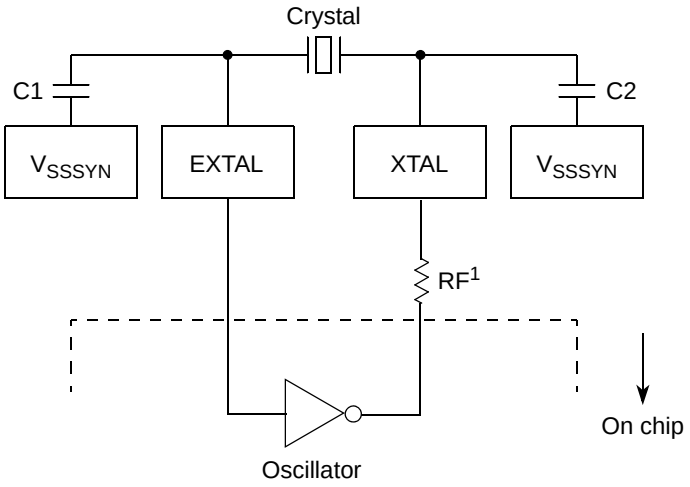
1 See Section 11.3.1.2, "Synthesizer Status Register (FMPLL_SYNSR)" for more information.

2 Because the 208 package has no $\overline{\text{RSTCFG}}$ pin, the signal is internally asserted (driven to 0), therefore the PLLCFG pins are always used to configure the FMPLL. After the device resets, the PLLCFG values remain the same as before the reset. The device does not reset to the crystal reference mode. Bypass mode is not enabled in the 208 package.

11.1.4.1 Crystal Reference

In crystal reference mode, the FMPLL receives an input clock frequency ($F_{ref\_crystal}$) from the crystal oscillator circuit (EXTAL_EXTCLK) and the pre-divider, and multiplies the frequency to create the FMPLL output clock. You must supply a crystal oscillator that is within the device input frequency range, the crystal manufacturer's recommended external support circuitry, and a short signal route from the MCU to the crystal.

The external support circuitry for the crystal oscillator is shown in Figure 11-7. Example component values are shown as well. Review the actual circuit with the crystal manufacturer. A block diagram illustrating crystal reference mode is shown in Figure 11-4.



1 For an 8–20 MHz crystal, the resistor must be 1–2.8 KΩ. The exact value depends on the crystal characteristics. Consult the crystal manufacturer's specifications.

Figure 11-7. Crystal Oscillator Network

In crystal reference mode, the FMPLL can generate a frequency modulated clock or a non-modulated clock (locked on a single frequency). The modulation rate, modulation depth, output clock divide ratio (RFD), and whether the FMPLL is modulating or not can be programmed by writing to the FMPLL registers. Crystal reference is the default clock mode for the 324 pin package. It is not necessary to force PLLCFG[0:1] to enter this mode.

In the 208 package size, because it has no $\overline{\text{RSTCFG}}$ pin, the crystal reference mode can only be selected through the PLLCFG pins.

11.1.4.2 External Reference Mode

The external reference mode functions the same as crystal reference mode except that EXTAL_EXTCLK is driven by an external clock generator rather than a crystal oscillator. The input frequency range (F_{ref_ext}) in external reference mode is the same as the input frequency reference range ($F_{ref\text{-}crystal}$) in the crystal reference mode, and frequency modulation is also available. To enter external reference mode, follow the procedure outlined in Section 11.1.4, "FMPLL Modes of Operation." A block diagram illustrating external reference mode is shown in Figure 11-3.

NOTE

In addition to supplying power for the CLKOUT signal, when the FMPLL is configured for external reference mode of operation, the V_{DDE5} supply voltage also controls the voltage level at which the signal presented to the EXTAL_EXTCLK pin causes a switch in the clock logic levels. The EXTAL_EXTCLK accepts a clock source with a voltage range of 1.6–3.6 V, however the transition voltage is determined by V_{DDE5} supply voltage divided by two. As an example, if V_{DDE5} is 3.3 V, then the clock transitions at approximately 1.6 V. The V_{DDE5} supply voltage and the voltage level of the external clock reference must be compatible, or the device does not clock correctly.

11.1.4.3 Bypass Mode

In FMPLL bypass mode, the FMPLL is completely bypassed and you must supply an external clock on the EXTAL_EXTCLK pin. The external clock is used directly to produce the internal system clocks. In bypass mode, the analog portion of the FMPLL is disabled and no clocks are generated at the FMPLL output. Consequently, frequency modulation is not available. In bypass mode the pre-divider is bypassed and has no effect on the system clock. The frequency in bypass mode is F_{ref_ext}.

To enter bypass mode, follow the procedure outlined in Section 11.1.4, "FMPLL Modes of Operation." A block diagram illustrating bypass mode is shown in Figure 11-2.

11.1.4.4 Dual-Controller Mode (1:1)

FMPLL dual-controller mode is used by the slave MCU device of a dual-controller system. The slave FMPLL facilitates skew reduction between the input and output clock signals. To enter dual-controller mode, follow the procedure outlined in Section 11.1.4, "FMPLL Modes of Operation."

In this mode, the system clock runs at twice the frequency of the EXTAL_EXTCLK input pin and is phase aligned. Crystal operation is not supported in dual-controller mode and an external clock must be provided. In this mode, the frequency and phase of the signal at the EXTAL_EXTCLK pin and the CLKOUT pin of the slave MCU are matched. A block diagram illustrating dual-controller mode (1:1) is shown in Figure 11-6.

Frequency modulation is not available when configured for dual-controller mode for both the master and slave devices. Enabling frequency modulation on the device supplying the reference clock to the slave in dual-controller mode produces unreliable clocks on the slave.

NOTE

When using dual-controller mode, do not change the CLKOUT clock divider on the slave device from its reset state of divide-by-two. Increasing or decreasing this divide ratio can produce unpredictable results from the FMPLL.

11.2 External Signal Description

Table 11-2 lists external signals used by the FMPLL during normal operation.

Table 11-2. PLL External Pin Interface

Name	I/O Type	Function	Pull
$\overline{\text{RSTCFG}}$_GPIO[210][1]	I/O	Determines the configuration to use during reset. GPIO used otherwise.	Up
PLLCFG[0]_GPIO[208]	I/O	Configures the mode during reset. GPIO used otherwise.	Up
PLLCFG[1]_GPIO[209]	I/O	Configures the mode during reset. GPIO used otherwise.	Up
XTAL	Output	Output drive for external crystal	—
EXTAL_EXTCLK	Input	Crystal external clock input	—
V_{DDSYN}	Power	Analog power supply (3.3 V ±10%)	—
V_{SSSYN}	Ground	Analog ground	—

[1] The 208 package does not have a $\overline{\text{RSTCFG}}$ pin, therefore the signal is internally asserted (driven to 0).

11.3 Memory Map/Register Definition

Table 11-3 shows the FMPLL memory map locations.

Table 11-3. FMPLL Module Memory Map

Address	Register Name	Register Description	Bits
Base (0xC3F8_0000)	FMPLL_SYNCR	Synthesizer control register	32
Base + 0x0004	FMPLL_SYNSR	Synthesizer status register	32
(Base + 0x0008)–0xC3F8_3FFF	—	Reserved	—

11.3.1 Register Descriptions

The clock operation is controlled by the synthesizer control register (FMPLL_SYNCR) and status is reported in the synthesizer status register (FMPLL_SYNSR). The following sections describe these registers in detail.

11.3.1.1 Synthesizer Control Register (FMPLL_SYNCR)

The synthesizer control register (FMPLL_SYNCR) contains bits for defining the clock operation for the system.

NOTE

To ensure proper operation for all MPC5500s, execute an **mbar** or **msync** instruction between: the write to change the FMPLL_SYNCR[MFD], and the read to check the lock status shown by FMPLL_SYNSR[LOCK].

Buffered writes to the FMPLL, as controlled by PBRIDGE_A_OPACR[BW0], must be disabled.

Address: Base + 0x0000 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	PREDIV			MFD				0	RFD				LOC EN	LOL RE	LOC RE
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DIS CLK	LOL IRQ	LOC IRQ	RATE	DEPTH				EXP							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-8. Synthesizer Control Register (FMPLL_SYNCR)

Table 11-4. FMPLL_SYNCR Field Descriptions

Field	Description
0	Reserved
1–3 PREDIV [0:2]	The PREDIV bits control the value of the divider on the input clock. The output of the pre-divider circuit generates the reference clock (F_{prediv}) to the FMPLL analog loop. When the PREDIV bits are changed, the FMPLL immediately loses lock. To prevent an immediate reset, the LOLRE bit must be cleared before writing the PREDIV bits. In 1:1 (dual-controller) mode, the PREDIV bits are ignored and the input clock is fed directly to the analog loop. 000 Divide by 1 001 Divide by 2 010 Divide by 3 011 Divide by 4 100 Divide by 5 101–111 Invalid values **Note:** Programming a PREDIV value such that the ICO operates outside its specified range causes unpredictable results and the FMPLL does not lock. See the device *Data Sheet* for details on the ICO range. **Note:** To avoid unintentional interrupt requests, disable LOLIRQ before changing PREDIV and then reenable it after acquiring lock. **Note:** When using crystal reference mode or external reference mode, The PREDIV value must not be set to any value that causes the phase/frequency detector to go below 4 MHz. That is, the crystal ($F_{ref\_crystal}$) or external clock (F_{ref_ext}) frequency divided by the PREDIV value creates the F_{prediv} frequency that must be greater than or equal to 4 MHz. See the device *Data Sheet* for F_{prediv} values. **Note:** To use the 8–20 MHz OSC, the PLL predivider must be configured for divide-by-two operation by tying PLLCFG[2] low (set PREDIV to 0b000).
4–8 MFD [0:4]	Multiplication factor divider. The MFD bits control the value of the divider in the FMPLL feedback loop. The value specified by the MFD bits establish the multiplication factor applied to the reference frequency. The decimal equivalent of the MFD binary number is substituted into the equation from Table 11-9 for F_{sys} to determine the equivalent multiplication factor. When the MFD bits are changed, the FMPLL loses lock. At this point, if modulation is enabled, the calibration sequence is reinitialized. To prevent an immediate reset, the LOLRE bit must be cleared before writing the MFD bits. In dual-controller mode, the MFD bits are ignored and the multiplication factor is equivalent to 2X. In bypass mode the MFD bits have no effect. **Note:** Programming an MFD value such that the ICO operates outside its specified range causes unpredictable results and the FMPLL does not lock. See the device *Data Sheet* for details on the ICO range. **Note:** To avoid unintentional interrupt requests, disable LOLIRQ before changing MFD and then reenable it after acquiring lock.
9	Reserved

Table 11-4. FMPLL_SYNCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>10–12
RFD
[0:2]</td><td>Reduced frequency divider. The RFD bits control a divider at the output of the FMPLL. The value specified by the RFD bits establish the divisor applied to the FMPLL frequency.
<table>
<tr><th>RFD[0:2]</th><th>Output Clock Divide Ratio</th></tr>
<tr><td>000</td><td>Divide by 1</td></tr>
<tr><td>001</td><td>Divide by 2</td></tr>
<tr><td>010</td><td>Divide by 4</td></tr>
<tr><td>011</td><td>Divide by 8</td></tr>
<tr><td>100</td><td>Divide by 16</td></tr>
<tr><td>101</td><td>Divide by 32</td></tr>
<tr><td>110</td><td>Divide by 64</td></tr>
<tr><td>111</td><td>Divide by 128</td></tr>
</table>
Changing the RFD bits does not affect the FMPLL; hence, no re-lock delay is incurred. Resulting changes in clock frequency are synchronized to the next falling edge of the current system clock. However these bits must only be written when the lock bit (LOCK) is set, to avoid exceeding the allowable system operating frequency. In bypass mode, the RFD bits have no effect.</td></tr>
<tr><td>13
LOCEN</td><td>Loss-of-clock enable. The LOCEN bit determines whether the loss of clock function is operational. See Section 11.4.2.6, “Loss-of-Clock Detection” and Section 11.4.2.6.1, “Alternate and Backup Clock Selection” for more information.
In bypass mode, this bit has no effect.
LOCEN does not affect the loss of lock circuitry.
0 Loss of clock disabled.
1 Loss of clock enabled.</td></tr>
<tr><td>14
LOLRE</td><td>Loss-of-lock reset enable. The LOLRE bit determines how the system integration module (the SIU) handles a loss of lock indication. When operating in crystal reference, external reference, or dual-controller mode, the FMPLL must be locked before setting the LOLRE bit. Otherwise reset is immediately asserted. The LOLRE bit has no effect in bypass mode.
0 Ignore loss of lock, reset not asserted.
1 Assert reset on loss of lock. Reset remains asserted, regardless of the source of reset, until after the FMPLL has locked.</td></tr>
<tr><td>15
LOCRE</td><td>Loss-of-clock reset enable. The LOCRE bit determines how the system integration module (the SIU) handles a loss of clock condition when LOCEN = 1. LOCRE has no effect when LOCEN = 0. If the LOCF bit in the SYNSR indicates a loss of clock condition, setting the LOCRE bit causes an immediate reset. In bypass mode LOCRE has no effect.
0 Ignore loss of clock, reset not asserted.
1 Assert reset on loss of clock.</td></tr>
<tr><td>16
DISCLK</td><td>Disable CLKOUT. The DISCLK bit determines whether CLKOUT is active. When CLKOUT is disabled it is driven low.
0 CLKOUT driven normally
1 CLKOUT driven low</td></tr>
</table>

Table 11-4. FMPLL_SYNCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>17
LOLIRQ</td><td>Loss-of-lock interrupt request. The LOLIRQ bit enables an interrupt request for LOLF when it (LOLIRQ) is asserted and when LOLF is asserted. If either LOLF or LOLIRQ is negated, the interrupt request is negated. When operating in crystal reference, external reference, or dual-controller mode, the FMPLL must be locked before setting the LOLIRQ bit. Otherwise an interrupt is immediately requested. The LOLIRQ bit has no effect in bypass mode.
0 Ignore loss of lock, interrupt not requested
1 Request interrupt</td></tr>
<tr><td>18
LOCIRQ</td><td>Loss-of-clock interrupt request. The LOCIRQ bit determines how the system integration module (the SIU) handles a loss of clock condition when LOCEN = 1. LOCIRQ has no effect when LOCEN = 0. If the LOCF bit in the SYNSR indicates a loss of clock condition, setting (or having previously set) the LOCIRQ bit causes an interrupt request. In bypass mode LOCIRQ has no effect.
0 Ignore loss of clock, interrupt not requested
1 Request interrupt on loss of clock.</td></tr>
<tr><td>19
RATE</td><td>Modulation rate. Controls the rate of frequency modulation applied to the system frequency. The allowable modulation rates are shown below. Changing the rate by writing to the RATE bit initiates the FM calibration sequence.
<table>
<tr><th>RATE</th><th>Modulation Rate (Hz)</th></tr>
<tr><td rowspan="2">0</td><td>$F_{mod} = F_{ref\_crystal} \div [(PREDIV + 1) \times 80]$</td></tr>
<tr><td>$F_{mod} = F_{ref\_ext} \div [(PREDIV + 1) \times 80]$</td></tr>
<tr><td rowspan="2">1</td><td>$F_{mod} = F_{ref\_crystal} \div [(PREDIV + 1) \times 40]$</td></tr>
<tr><td>$F_{mod} = F_{ref\_ext} \div [(PREDIV + 1) \times 40]$</td></tr>
</table>
Note: To prevent unintentional interrupt requests, clear LOLIRQ before changing RATE.
Note: F_{mod} must be between 100–250 MHz. See Section , “Changing the MFD or PREDIV values causes the FMPLL to perform a search for the lock frequency that results in the system clock frequency changing rapidly across the complete frequency range. All MCU peripherals, including the external bus are subjected to this frequency sweep. Operation of timers and serial communications during this search sequence produces unpredictable results..”</td></tr>
<tr><td>20–21
DEPTH
[0:1]</td><td>Controls the frequency modulation depth and enables the frequency modulation. When programmed to a value other than 0x0000, the frequency modulation is automatically enabled. The programmable frequency deviations from the system frequency are shown below. If the depth is changed to a value other than 0x0000, the calibration sequence is reinitialized.
<table>
<tr><th>DEPTH[1]</th><th>DEPTH[0]</th><th>Modulation Depth (% of Fsys)</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1.0 ± 0.2</td></tr>
<tr><td>1</td><td>0</td><td>2.0 ± 0.2</td></tr>
<tr><td>1</td><td>1</td><td>Invalid value</td></tr>
</table>
Note: To prevent unintentional interrupt requests, clear LOLIRQ before changing DEPTH.</td></tr>
<tr><td>22–31
EXP
[0:9]</td><td>Expected difference value. Holds the expected value of the difference of the reference and the feedback counters. See Section 11.4.3.3, “FM Calibration Routine” to determine the value of these bits. This field is written by the application before entering calibration mode.</td></tr>
</table>

11.3.1.2 Synthesizer Status Register (FMPLL_SYNSR)

The synthesizer status register (FMPLL_SYNSR) is a 32-bit register. Only the LOLF and LOCF flag bits are writable in this register. Writes to bits other than the LOLF and LOCF have no effect.

Address: Base + 0x0004 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	LOLF	LOC	MODE	PLL SEL	PLL REF	LOCKS	LOCK	LOCF	CALD ONE	CAL PASS
W							w1c							w1c		
Reset	0	0	0	0	0	0	0	0	—[1]	—[1]	—[1]	—[1]	—[2]	0	0	0

[1] Reset state determined during reset configuration. (See Section 11.1.4, “FMPLL Modes of Operation,” for more information.)

[2] Reset state determined during reset.

Note: “w1c” signifies that this bit is cleared by writing a 1 to it.

Figure 11-9. Synthesizer Status Register (FMPLL_SYNSR)

Table 11-5. FMPLL_SYNSR Field Descriptions

Field	Description
0–21	Reserved
22 LOLF	Loss-of-lock flag. Provides the interrupt request flag. This is a write 1 to clear (w1c) bit; to clear the flag, you must write a 1 to the bit. Writing 0 has no effect. This flag is not set and an interrupt is not requested, if the loss-of-lock condition was caused by: • a system reset • a write to the FMPLL_SYNCR which modifies the MFD bits • enabling frequency modulation If the flag is set due to a system failure, writing the MFD bits or enabling FM does not clear the flag. Asserting reset clears the flag. This flag bit is sticky; if lock is reacquired, the bit remains set until either a write of 1 or reset is asserted. 0 Interrupt service not requested 1 Interrupt service requested **Note:** Upon a loss-of-lock that is not generated by: •System reset •Write to the FMPLL_SYNCR that modifies the MFD or PREDIV bits •Enabling of frequency modulation the LOLF is set *only if* LOLIRQ is set. If the FMPLL reacquires lock and any of the previous conditions in the bulleted list occurs, the LOLF is set again. To avoid generating an unintentional interrupt, clear LOLIRQ before changing MFD or PREDIV, or before enabling FM after a previous interrupt and relock occurred.

Table 11-5. FMPLL_SYNSR Field Descriptions (continued)

Field	Description
23 LOC	Loss-of-clock status. Indicates whether a loss-of-clock condition is present when operating in crystal reference, external reference, or dual-controller mode, If LOC = 0, the system clocks are operating normally. If LOC = 1, the system clocks have failed due to a reference failure or a FMPLL failure. If the read of the LOC bit and the loss-of-clock condition occur simultaneously, the bit does not reflect the current loss of clock condition. If a loss-of-clock condition occurs which sets this bit and the clocks later return to normal, this bit is cleared. A loss of clock condition can only be detected if LOCEN = 1. LOC is always 0 in bypass mode. 0 Clocks are operating normally 1 Clocks are not operating normally.
24 MODE	Clock mode. This bit is read only and the value is determined at reset. The value of this bit combined with the values of the PLLSEL and PLLREF bits, set the system clocking mode used. See Chapter 4, “Reset,” for details on how to configure the system clock mode during reset. 0 PLL bypass mode used. 1 PLL clock mode used.
25 PLLSEL	PLL mode select. This bit is read only and the value is determined at reset. The value of this bit combined with the values of the MODE and PLLREF bits, indicates the system clocking mode used. This bit indicates the FMPLL operating mode used: dual controller or reference mode. This bit is cleared in dual-controller and bypass mode. See Chapter 4, “Reset,” for details on how to configure the system clock mode during reset. See Table 11-1 and Table 11-2 for more information. 0 Dual-controller mode used. 1 Crystal reference or external reference mode used.
26 PLLREF	PLL clock reference source. This bit is read only and the value is determined at reset. The value of this bit combined with the values of the MODE and PLLSEL bits, indicates the system clocking mode used. This bit determines whether an external clock or a crystal reference is used as a the PLL reference source. This bit is cleared in dual controller mode and bypass mode. See Chapter 4, “Reset,” for details on how to configure the system clock mode during reset. 0 External clock reference used. 1 Crystal clock reference used.
27 LOCKS	Sticky FMPLL lock status bit. This bit is a read-only sticky bit that indicates the FMPLL lock status. LOCKS is set by the lock detect circuitry when the FMPLL acquires lock after one of the following: • System reset • Write to the FMPLL_SYNCR that modifies the MFD and PREDIV bits • Enable frequency modulation Whenever the FMPLL loses lock, LOCKS is cleared. LOCKS remains cleared even after the FMPLL relocks, until one of the three previously-stated conditions occurs. Furthermore, if the LOCKS bit is read when the FMPLL simultaneously loses lock, the bit does not reflect the current loss of lock condition. If operating in bypass mode, LOCKS remains cleared after reset. In crystal reference, external reference, and dual-controller mode, LOCKS is set after reset. 0 PLL has lost lock since last system reset, a write to FMPLL_SYNCR to modify the MFD and PREDIV bit fields, or frequency modulation enabled. 1 PLL has not lost lock since last system reset, a write to FMPLL_SYNCR to modify the MFD and PREDIV bit fields, or frequency modulation enabled.
28 LOCK	PLL lock status bit. This bit is a read-only bit that indicates whether the FMPLL has acquired lock. If the LOCK bit is read when the FMPLL simultaneously loses lock or acquires lock, the bit does not reflect the current condition of the FMPLL. If operating in bypass mode, LOCK remains cleared after reset. See the frequency as defined in the *MPC5534 Microcontroller Data Sheet* for the lock/unlock range. 0 PLL is unlocked. 1 PLL is locked.

Table 11-5. FMPLL_SYNSR Field Descriptions (continued)

Field	Description
29 LOCF	Loss-of-clock flag. This bit provides the interrupt request flag. This is a write 1 to clear (w1c) bit; to clear the flag, you must write a 1 to the bit. Writing 0 has no effect. Asserting reset clears the flag. This flag is sticky in the sense that if clocks return to normal after the flag has been set, the bit remains set until cleared by either writing 1 or asserting reset. 0 Interrupt service not requested 1 Interrupt service requested
30 CALDONE	Calibration complete. Indicates whether the calibration sequence has been completed since the last time modulation was enabled. If CALDONE = 0 then the calibration sequence is either in progress or modulation is disabled. If CALDONE = 1 then the calibration sequence has been completed, and frequency modulation is operating. 0 Calibration not complete. 1 Calibration complete. **Note:** FM relocking does not start until calibration is complete.
31 CALPASS	Calibration passed. Indicates whether the calibration routine was successful. If CALPASS = 1 and CALDONE = 1 then the routine was successful. If CALPASS = 0 and CALDONE = 1, then the routine was unsuccessful. When the calibration routine is initiated the CALPASS is asserted. CALPASS remains asserted until either modulation is disabled by clearing the DEPTH bits in the FMPLL_SYNCR or a failure occurs within the FMPLL calibration sequence. 0 Calibration unsuccessful. 1 Calibration successful. If calibration is unsuccessful, then actual depth is not guaranteed to match the desired depth.

11.4 Functional Description

This section explains clock architecture, clock operation, and clock configuration.

11.4.1 Clock Architecture

This section describes the clocks and clock architecture in the MCU.

The system clocks are generated from one of four FMPLL modes: crystal reference mode, external reference mode, dual-controller (1:1) mode, and bypass mode. See Section 11.1, “Introduction” for information on the different clocking modes available in the FMPLL.

The MCU has three clock output pins that are driven by programmable clock dividers. The clock dividers divide the system clock down by even integer values. The three clock output pins are the following:

- CLKOUT – External address/data bus clock
- MCKO – Nexus auxiliary port clock
- ENGCLK – Engineering clock

The MCU has been designed so that the oscillator clock can be selected as the clock source for the CAN interface in the FlexCAN blocks resulting in very low jitter performance.

Figure 11-1 shows a block diagram of the FMPLL and the system clock architecture.

11.4.1.1 Software Controlled Power Management/Clock Gating

The peripheral IP modules are designed to let software gate the clocks to the non-memory-mapped logic of the modules.

Some of the IP modules on this device support software controlled power management/clock gating whereby the application software can disable the non-memory-mapped portions of the modules by writing to module disable (MDIS) bits in registers within the modules. The memory-mapped portions of the modules are clocked by the system clock when they are being accessed. The Nexus Port Controller (NPC) can be configured to disable the MCKO signal when there are no Nexus messages pending. The flash array can be disabled by writing to a bit in the flash register map.

The modules that implement software controlled power management and clock gating are listed in Table 11-6 along with the registers and bits that disable each module. The software controlled clocks are enabled when the MCU comes out of reset.

Table 11-6. Software Controlled Power Management/Clock Gating Support

Module Name	Register Name	Bit Names
DSPI B	DSPI_B_MCR	MDIS
DSPI C	DSPI_C_MCR	MDIS
DSPI D	DSPI_D_MCR	MDIS
EBI	EBI_MCR	MDIS
eTPU engine A	ETPU_ECR_1	MDIS
FlexCAN A	CAN_A_MCR	MDIS
FlexCAN C	CAN_C_MCR	MDIS
eMIOS	EMIOS_MCR	MDIS
eSCI A	ESCI_A_CR2	MDIS
eSCI B	ESCI_B_CR2	MDIS
Nexus port controller (NPC)	NPC_PCR	MCKO_EN, MCKO_GT [1]
Flash array	FLASH_MCR	STOP [2]

[1] See Chapter 24, “Nexus Development Interface.”

[2] See Chapter 13, “Flash Memory.”

11.4.1.2 Clock Dividers

Each of the CLKOUT, MCKO, and ENGCLK dividers provides a nominal 50% duty cycle clock to an output pin. There is no guaranteed phase relationship between CLKOUT, MCKO, and ENGCLK. ENGCLK is not synchronized to any I/O pins.

11.4.1.2.1 External Bus Clock (CLKOUT)

The external bus clock (CLKOUT) divider can be programmed to divide the system clock by two or four based on the settings of the EBDF bit field in the SIU external clock control register (SIU_ECCR). The reset value of the EBDF selects a CLKOUT frequency of one half of the system clock frequency. The EBI

supports gating of the CLKOUT signal when there are no external bus accesses in progress. See the Chapter 6, “System Integration Unit (SIU)” for more information on CLKOUT.

The hold-time for the external bus pins can be changed by writing to the external bus tap select (EBTS) bit in the SIU_ECCR. See Chapter 6, “System Integration Unit (SIU)” for more information.

11.4.1.2.2 Nexus Message Clock (MCKO)

The Nexus message clock (MCKO) divider can be programmed to divide the system clock by two, four or eight based on the MCKO_DIV bit field in the port configuration register (PCR) in the Nexus port controller (NPC). The reset value of the MCKO_DIV selects an MCKO clock frequency one half of the system clock frequency. The MCKO divider is configured by writing to the NPC through the JTAG port. See Chapter 24, “Nexus Development Interface” for more information.

11.4.1.2.3 Engineering Clock (ENGCLK)

The engineering clock (ENGCLK) divider can be programmed to divide the system clock by factors from 2 to 126 in increments of two. The ENGDIV bit field in the SIU_ECCR determines the divide factor. The reset value of ENGDIV selects an ENGCLK frequency of system clock divided by 32.

11.4.1.2.4 FlexCAN_*x* Clock Domains

The FlexCAN modules have two distinct software controlled clock domains. One of the clock domains is always derived from the system clock. This clock domain includes the message buffer logic. The source for the second clock domain can be either the system clock or a direct feed from the oscillator pin EXTAL_EXTCLK. The logic in the second clock domain controls the CAN interface pins. The CLK_SRC bit in the FlexCAN CTRL register selects between the system clock and the oscillator clock as the clock source for the second domain. Selecting the oscillator as the clock source ensures very low jitter on the CAN bus. System software can gate both clocks by writing to the MDIS bit in the FlexCAN MCR register. Figure 11-1 shows the two clock domains in the FlexCAN modules.

See Chapter 21, “FlexCAN2 Controller Area Network” for more information on the FlexCAN modules.

11.4.2 Clock Operation

11.4.2.1 Input Clock Frequency

The FMPLL is designed to operate over an input clock frequency range as determined by the operating mode. The operating ranges for each mode are given in Table 11-7.

Table 11-7. Input Clock Frequency

Mode	Symbol	Input Frequency Range
Crystal reference External reference	$F_{ref\_crystal}$ F_{ref_ext}	8–20 MHz
Bypass	F_{extal}	0–132 MHz
Dual-controller (1:1)	$F_{ref\_1:1}$	25–66 MHz

11.4.2.2 Reduced Frequency Divider (RFD)

The RFD can be used for reducing the FMPLL system clock frequency. To protect the system from frequency overshoot during the PLL lock detect phase, the RFD must be programmed to be greater than or equal to 1 when changing MFD or PREDIV or when enabling frequency modulation.

11.4.2.3 Programmable Frequency Modulation

The FMPLL provides for frequency modulation of the system clock. The modulation is applied as a triangular waveform with modulation depth and rate controlled by fields in the FMPLL_SYNCR. The modulation depth can be set to ±1% or ±2% of the system frequency. The modulation rate is dependent on the reference clock frequency.

Complete details for configuring the programmable frequency modulation is given in Section 11.4.3, “Clock Configuration.” Changing the MFD or PREDIV values causes the FMPLL to perform a search for the lock frequency that results in the system clock frequency changing rapidly across the complete frequency range. All MCU peripherals, including the external bus are subjected to this frequency sweep. Operation of timers and serial communications during this search sequence produces unpredictable results.

11.4.2.4 FMPLL Lock Detection

A pair of counters monitor the reference and feedback clocks to determine when the system has acquired frequency lock. After the FMPLL has locked, the counters continue to monitor the reference and feedback clocks and reports if/when the FMPLL has lost lock. The FMPLL_SYNCR provides the flexibility to select whether to generate an interrupt, assert system reset, or do nothing in the event that the FMPLL loses lock. See Section 11.3.1.1, “Synthesizer Control Register (FMPLL_SYNCR) for details.

When the frequency modulation is enabled, the loss of lock continues to function as described but with the lock and loss of lock criteria reduced to ensure that false loss of lock conditions are not detected.

In bypass mode, the FMPLL cannot lock since the FMPLL is disabled.

11.4.2.5 FMPLL Loss-of-Lock Conditions

After the FMPLL acquires lock after reset, the FMPLL_SYNSR[LOCK] and FMPLL_SYNSR[LOCKS] status bits are set. If the MFD is changed or if an unexpected loss of lock condition occurs, the LOCK and LOCKS status bits are negated. While the FMPLL is in an unlocked condition, the system clocks continue to be sourced from the FMPLL as the FMPLL attempts to re-lock. Consequently, during the re-locking process, the system clock frequency is not well defined and can exceed the maximum system frequency thereby violating the system clock timing specifications (when changing MFD and PREDIV, this is avoided by following the procedure detailed in Section 11.4.3, “Clock Configuration”). Because this condition can arise during unexpected loss of lock events, it is recommended to use the loss of lock reset functionality, See Section 11.4.2.5.1, “FMPLL Loss-of-Lock Reset,” below. However, LOLRE must be cleared while changing the MFD otherwise a reset occurs.

After the FMPLL has relocked, the LOCK bit is set. The LOCKS bit remains cleared if the loss of lock was unexpected. The LOCKS bit is set to 1 when the loss of lock was caused by changing the MFD.

11.4.2.5.1 FMPLL Loss-of-Lock Reset

The FMPLL provides the ability to assert reset when a loss of lock condition occurs by programming the FMPLL_SYNCR[LOLRE] bit. Reset is asserted if LOLRE is set and loss-of-lock occurs. Because the FMPLL_SYNSR[LOCK] and FMPLL_SYNSR[LOCKS] bits are reinitialized after reset, the system reset status register (SIU_RSR) must be read to determine that a loss of lock condition occurred.

To exit reset, the reference must be present and the FMPLL must acquire lock. In bypass mode, the FMPLL cannot lock. Therefore a loss of lock condition cannot occur, and LOLRE has no effect.

11.4.2.5.2 FMPLL Loss-of-Lock Interrupt Request

The FMPLL provides the ability to request an interrupt when a loss of lock condition occurs by programming the FMPLL_SYNCR[LOLIRQ] bit. An interrupt is requested by the FMPLL if LOLIRQ is set and loss-of-lock occurs.

In bypass mode, the FMPLL cannot lock. Therefore a loss of lock condition cannot occur, and the LOLIRQ bit has no effect.

11.4.2.6 Loss-of-Clock Detection

The FMPLL continuously monitors the reference and feedback clocks. In the event either of the clocks fall below a threshold frequency, the system reports a loss of clock condition. You can enable a feature to have the FMPLL switch the system clocks to a backup clock in the event of such a failure. Additionally, you can enter a system RESET, assert an interrupt request, or do nothing if the FMPLL reports this condition.

11.4.2.6.1 Alternate and Backup Clock Selection

If you enable loss-of-clock by setting FMPLL.SYNCR[LOCEN] = 1, then the FMPLL transitions system clocks to a backup clock source in the event of a clock failure as per Table 11-8.

If loss of clock is enabled and the reference clock is the source of the failure, the FMPLL enters self-clock mode (SCM). The exact frequency during self-clock mode operation is indeterminate due to process, voltage, and temperature variation but is guaranteed to be below the maximum system frequency. If the FMPLL clocks have failed, the FMPLL transitions the system clock source to the reference clock.

The FMPLL remains in SCM until the next reset. If the FMPLL is operated in SCM, writes to FMPLL_SYNCR[RFD] have no effect on clock frequency. The SCM system frequency stated in the device *Data Sheet* assumes that the RFD has been programmed to 0x0.

If loss of clock is enabled and the loss-of-clock is due to a FMPLL failure (for example, loss of feedback clock), the FMPLL reference becomes the system clock's source until the next reset, even if the FMPLL regains itself and re-locks.

Table 11-8. Loss-of-Clock Summary

Clock Mode	System Clock Source before Failure	REFERENCE FAILURE Alternate Clock Selected by LOC Circuitry until Reset	PLL FAILURE Alternate Clock Selected by LOC Circuitry until Reset
Crystal Reference External Reference	PLL	PLL self-clocked mode	PLL reference
Bypass	External clock(s)	None	—

A special loss of clock condition occurs when both the reference and the FMPLL fail. The failures can be simultaneous or the FMPLL can fail first. In either case, the reference clock failure takes priority and the FMPLL attempts to operate in SCM. If successful, the FMPLL remains in SCM until the next reset. During SCM, modulation is always disabled. If the FMPLL cannot operate in SCM, the system remains static until the next reset. Both the reference and the FMPLL must be functioning correctly to exit reset.

11.4.2.6.2 Loss-of-Clock Reset

When a loss-of-clock condition is recognized, reset is asserted if the FMPLL_SYNCR[LOCRE] bit is set. The LOCF and LOC bits in FMPLL_SYNSR are cleared after reset, therefore, the SIU_RSR must be read to determine that a loss of clock condition occurred. LOCRE has no effect in bypass mode.

To exit reset, the reference must be present and the FMPLL must acquire lock.

11.4.2.6.3 Loss-of-Clock Interrupt Request

When a loss-of-clock condition is recognized, the FMPLL requests an interrupt if the FMPLL_SYNCR[LOCIRQ] bit is set. The LOCIRQ bit has no effect in bypass mode or if FMPLL_SYNCR[LOCEN] = 0.

11.4.3 Clock Configuration

In crystal reference and external reference clock mode, the default system frequency is determined by the MFD, RFD, and PREDIV reset values. See Section 11.3.1.1, “Synthesizer Control Register (FMPLL_SYNCR).” The frequency multiplier is determined by the RFD, PREDIV, and multiplication frequency divisor (MFD) bits in FMPLL_SYNCR.

Table 11-9 shows the clock-out to clock-in frequency relationships for the possible clock modes.

Table 11-9. Clock-out vs. Clock-in Relationships

Clock Mode	PLL Option
Crystal Reference Mode	$F_{sys} = F_{ref\_crystal} \times \frac{(MFD + 4)}{[(PREDIV + 1) \times 2^{RFD}]}$
External Reference Mode	$F_{sys} = F_{ref\_ext} \times \frac{(MFD + 4)}{[(PREDIV + 1) \times 2^{RFD}]}$

Table 11-9. Clock-out vs. Clock-in Relationships (continued)

Clock Mode	PLL Option
Dual Controller (1:1) Mode	$F_{sys} = 2 \times F_{ref\_1:1}$
Bypass Mode	$F_{sys} = F_{ref\_ext}$

NOTES:
F_{sys} = system frequency
F_{prediv} = clock frequency after PREDIV.
$F_{ref\_crystal}$ and F_{ref_ext} = clock frequencies at the EXTAL_EXTCLK signal. (See Figure 11-1).
MFD ranges from 0–31.
RFD ranges from 0–7.
PREDIV normal reset value is 0. Caution: Programming a PREDIV value such that the ICO operates outside its specified range causes unpredictable results and the FMPLL does not lock. See the device *Data Sheet* for details on the ICO range.

When programming the FMPLL, do not violate the maximum system clocks frequency, or maximum and minimum ICO frequency specifications. For determining the MFD value, use a value of zero for the RFD (translates to divide-by-one). This ensures that the FMPLL does not try to synthesize a frequency out of its range. See the device *Data Sheet* for more information.

11.4.3.1 Programming System Clock Frequency Without Frequency Modulation

The following steps are required to accommodate the frequency overshoot that can occur when the PREDIV or MFD bits are changed. If frequency modulation is going to be enabled, the maximum allowable frequency must be reduced by the programmed ΔF_m.

NOTE

Following these steps produces immediate changes in supply current, therefore make sure the power supply is decoupled with low ESR capacitors.

The following steps program the clock frequency without frequency modulation:

1. Determine the value for the PREDIV, MFD, and RFD fields in the synthesizer control register (FMPLL_SYNCR). Remember to include the ΔF_m if frequency modulation is enabled. The amount of jitter in the system clocks can be minimized by selecting the maximum MFD factor that can be paired with an RFD factor to provide the desired frequency. The maximum MFD value that can be used is determined by the ICO range. See the *Data Sheet* for the maximum frequency of the ICO.
2. Change the following in FMPLL_SYNCR:
   a) Make sure frequency modulation is disabled (FMPLL_SYNCR[DEPTH] = 00). A change to PREDIV, MFD, or RATE while modulation is enabled invalidates the previous calibration results.
   b) Clear FMPLL_SYNCR[LOLRE]. If this bit is set, the MCU goes into reset when MFD is written.

c) Initialize the FMPLL for less than the desired final system frequency (done in one single write to FMPLL_SYNCR):
   - Disable LOLIRQ.
   - Write FMPLL_SYNCR[PREDIV] to a desired final value.
   - Write FMPLL_SYNCR[MFD] to a desired final value.
   - Write the RFD control field value to a desired final RFD value plus one. RFD must be set to greater than one to protect from overshoot.

3. Wait for the FMPLL to lock by monitoring the FMPLL_SYNSR[LOCK] bit. See Section 11.3.1.1, "Synthesizer Control Register (FMPLL_SYNCR)," for memory synchronization between changing FMPLL_SYNCR[MFD] and monitoring the lock status.
4. Initialize the FMPLL to the desired final system frequency by changing FMPLL_SYNCR[RFD]. The FMPLL does not need to re-lock if only the RFD changes.
5. Re-enable LOLIRQ.

When using crystal reference mode or external reference mode, do not set the PREDIV value to any value that causes the phase and frequency detector to go below 4 MHz. That is, the crystal or external clock frequency divided by the PREDIV value must be in the range of 4–20 MHz.

This first register write causes the FMPLL to switch to an initial system frequency which is less than the final one. Keeping the change of frequency to a lower initial value helps minimize the current surge to the external power supply caused by the change in frequency. The last step changes the RFD to get the desired final frequency.

Changing the MFD or PREDIV values causes the FMPLL to perform a search for the lock frequency that results in the system clock frequency changing rapidly across the complete frequency range. All MCU peripherals, including the external bus are subjected to this frequency sweep. Operation of timers and serial communications during this search sequence produces unpredictable results.

11.4.3.2 Programming System Clock Frequency with Frequency Modulation

In crystal reference and external reference clock modes, the default mode is without frequency modulation enabled. When frequency modulation is enabled, however, three parameters must be set to generate the desired level of modulation: the RATE, DEPTH, and EXP bit fields of the FMPLL_SYNCR. RATE and DEPTH determine the modulation rate and the modulation depth. The EXP field controls the FM calibration routine. Section 11.4.3.3, "FM Calibration Routine," shows how to obtain the values to be programmed for EXP. Figure 11-10 illustrates the effects of the parameters and the modulation waveform built into the modulation hardware. The modulation waveform is always a triangle wave and its shape is not programmable.

The modulation rates given are specific to a reference frequency of 8 MHz. F_{prediv} is the frequency after the predivider.

$$F_{mod} = F_{ref\_crystal} \quad \text{or} \quad F_{ref\_ext} \div [(PREDIV + 1) \times Q]$$

where:

Q = 40 or 80. This gives modulation rates of 200 kHz and 100 kHz, respectively.

NOTE

The following relationship between F_{mod} and modulation rates must be maintained:

$$100\ KHz \leq F_{mod} \leq 250\ KHz$$

Therefore, the use of a non 8 MHz reference results in scaled modulation rates.

The steps to program the clock frequency with frequency modulation ensure the calibration routine operates correctly and prevents frequency overshoot:

1. Change the following in FMPLL_SYNCR:
   a) Make sure frequency modulation is disabled (FMPLL_SYNCR[DEPTH] = 00). A change to PREDIV, MFD, or RATE while modulation is enabled invalidates the previous calibration results.
   b) Clear FMPLL_SYNCR[LOLRE]. If this bit is set, the MCU goes into reset when MFD is written.
   c) Initialize the FMPLL for less than the desired final frequency:
   — Disable LOLIRQ.
   — Write FMPLL_SYNCR[PREDIV] to the desired final value.
   — Write FMPLL_SYNCR[MFD] to the desired final value.
   — Write FMPLL_SYNCR[EXP] to the desired final value.
   — Write FMPLL_SYNCR[RATE] to the desired final value.
   — Write the RFD control field to 1 plus the desired final RFD value (RFD must be greater than one to protect from overshoot).
2. Wait for the FMPLL to lock by monitoring the FMPLL_SYNSR[LOCK] bit. See Section 11.3.1.1, “Synthesizer Control Register (FMPLL_SYNCR),” for memory synchronization between changing FMPLL_SYNCR[MFD] and monitoring the lock status.
3. If using the frequency modulation feature, then:
   a) Enable FM by setting FMPLL_SYNCR[DEPTH] = 1 or 2.
   b) Also set FMPLL_SYNCR[RATE] if not done previously in step 2.
4. Calibration starts. After calibration is done, then the FMPLL re-locks. Wait for the FMPLL to re-lock by monitoring the FMPLL_SYNSR[LOCK] bit.
5. Verify FM calibration completed and was successful by testing the FMPLL_SYNSR[CALDONE] and FMPLL_SYNSR[CALPASS] bitfields.
6. If FM calibration did not complete or was not successful, attempt again by going back to step 1.
7. Initialize the FMPLL to the desired final system frequency by changing FMPLL_SYNCR[RFD]. The FMPLL does not need to re-lock when only changing the RFD.
8. Re-enable LOLIRQ.

NOTE

This first register write causes the FMPLL to switch to an initial frequency which is less than the final one. Keeping the change of frequency to a lower initial value helps minimize the current surge to the external power supply caused by change of frequency. The last step changes the RFD to get the final frequency.

NOTE

Changing the MFD or PREDIV values causes the FMPLL to perform a search for the lock frequency that results in the system clock frequency changing rapidly across the complete frequency range. All MCU peripherals, including the external bus, are subjected to this frequency sweep. Operation of timers and serial communications during this search sequence produces unpredictable results.

The frequency modulation system is dependent upon several the accuracies of these factors:

- V_{DDSYN} and V_{SSSYN} voltages
- Crystal oscillator frequency
- Manufacturing variation

For example, if a 5% accurate supply voltage is used, then a 5% modulation depth error results. If the crystal oscillator frequency is skewed from 8 MHz, the resulting modulation frequency is proportionally skewed. Finally, the error due to the manufacturing and environment variation alone can cause the frequency modulation depth error to be greater than 20%.

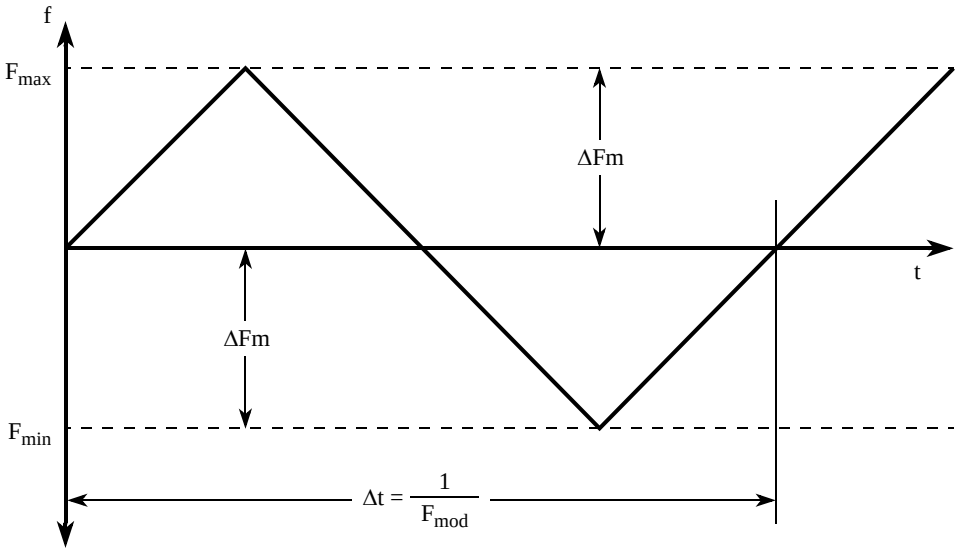


Figure 11-10. Frequency Modulation Waveform

11.4.3.3 FM Calibration Routine

Upon enabling frequency modulation, a new calibration routine is performed. This routine tunes a reference current into the modulation D/A so that the modulation depth (F_{max} and F_{min}) remains within specification.

Entering the FM calibration mode requires you to program SYNCR[EXP]. The EXP is the expected value of the difference between the reference and feedback counters used in the calibration of the FM equation:

$$EXP = \frac{((MFD + 4) \times M \times P)}{100}$$

For example, if 80 MHz is the desired final frequency and an 8 MHz crystal is used, the final values of MFD = 6 and RFD = 0 produces the desired 80 MHz. For a desired frequency modulation with a 1% depth, then EXP is calculated using P = 1, MFD = 6 and M = 480. See Table 11-10 for a complete list of values to be used for the variable (M) based on MFD setting. To obtain a percent modulation (P) of 1%, the EXP field must be set at:

$$EXP = ((6 + 4) \times 480 \times 1) \div 100 = 48$$

Rounding this value to the closest integer yields 48, which is entered into the EXP field for this example.

Table 11-10. Multiplied Factor Dividers with M Values

MFD	M
0–2	960
3–5	640
6–8	480
9–14	320
15–20	240
21–31	160

This routine corrects for process variations, but because temperature can change after calibration is performed, the variation caused by temperature drift remains. This frequency modulation calibration system is also voltage dependent, so if the supply changes after the sequence occurs, errors incurred are not corrected. The calibration system reuses the two counters in the lock detect circuit, and the reference and feedback counters. The reference counter remains clocked by the reference clock, but the feedback counter is clocked by the ICO clock.

When the calibration routine is initiated by writing to the DEPTH bits, the CALPASS status bit is immediately set and the CALDONE status bit is immediately cleared.

When calibration is induced, the ICO is given time to settle. Then both the feedback and reference counters start counting. Full ICO clock cycles are counted by the feedback counter during this time to give the initial center frequency count. When the reference counter has counted to the programmed number of reference count cycles, the input to the feedback counter is disabled and the result is placed in the COUNT0 register. The calibration system then enables modulation at programmed ΔFm. The ICO is given time to settle. Both counters are reset and restarted. The feedback counter begins to count full ICO clock cycles again to obtain the delta-frequency count. When the reference counter has counted to the new programmed number of reference count cycles, the feedback counter is stopped again.

The delta-frequency count minus the center frequency count (COUNT0) results in a delta count proportional to the reference current into the modulation D/A. That delta count is subtracted from the expected value given in the EXP field of the FMPLL_SYNCR resulting in an error count. The sign of this error count determines the direction taken by the calibration D/A to update the calibration current. After obtaining the error count for the present iteration, both counters are cleared. The stored count of COUNT0

is preserved while a new feedback count is obtained, and the process to determine the error count is repeated. The calibration system repeats this process eight times, once for each bit of the calibration D/A.

After the last decision is made, the CALDONE bit of the SYNSR is written to a one. If an error occurs during the calibration routine, then CALPASS is immediately written to a zero. If the routine completed successfully then CALPASS remains a one.

Figure 11-11 shows a block diagram of the calibration circuitry and its associated registers. Figure 11-12 shows a flow chart showing the steps taken by the calibration circuit.

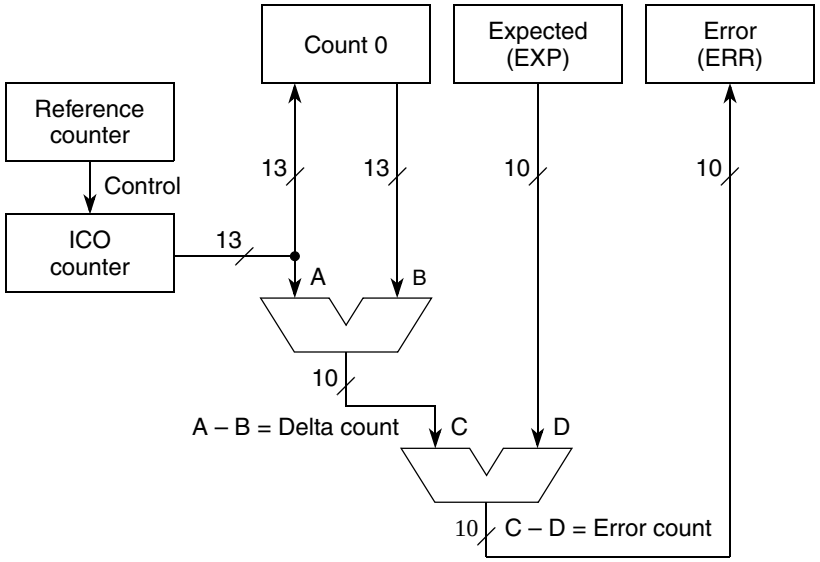


Figure 11-11. FM Auto-Calibration Data Flow

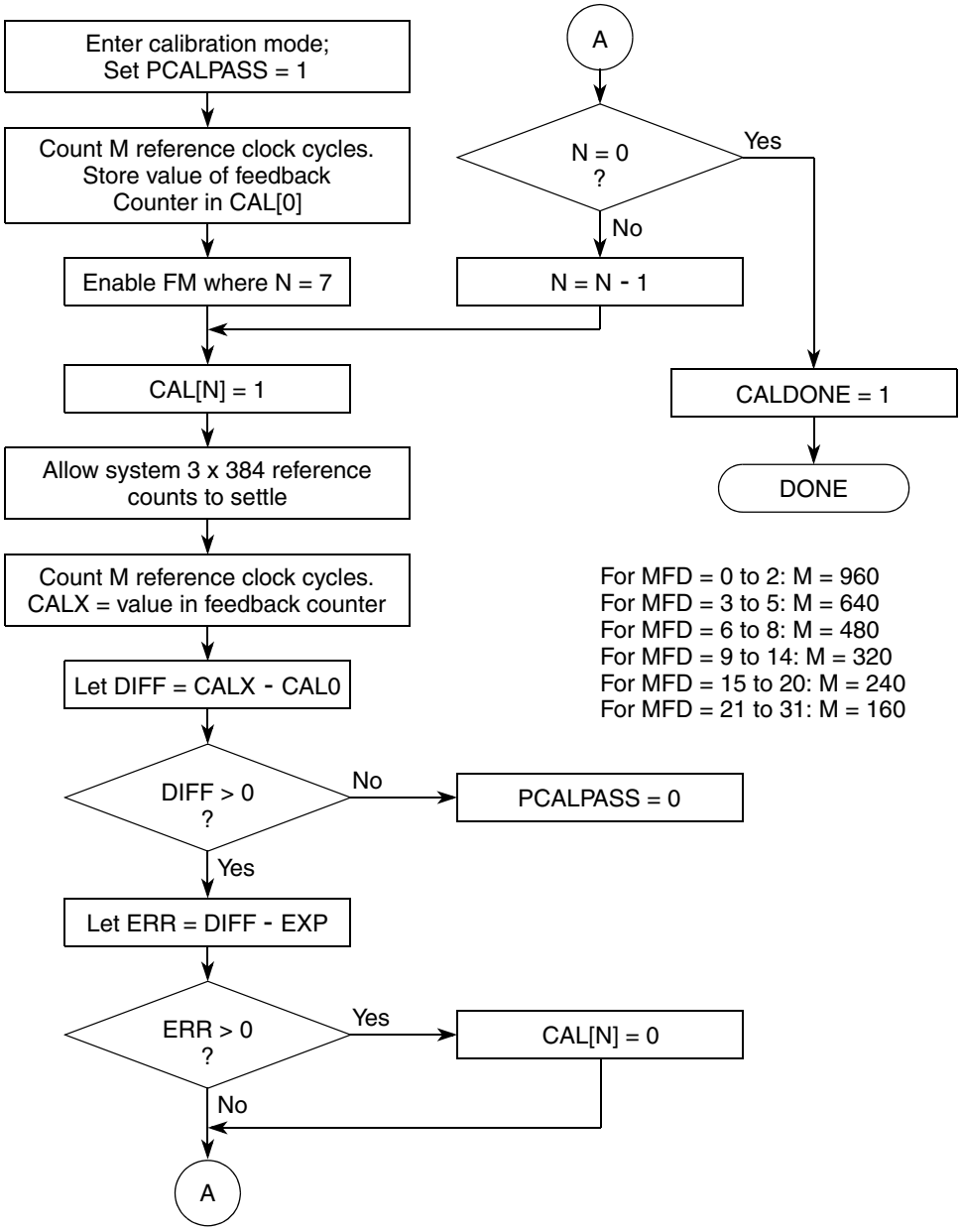


Figure 11-12. FM Auto-Calibration Flow Chart

Chapter 12
External Bus Interface (EBI)

NOTE

The 208 package does not have an external bus interface. This chapter pertains to devices in the 324 package, with and without the 496 assembly.

12.1 Introduction

This chapter describes the external bus interface (EBI) that manages the transfer of information between the internal buses and the memories or peripherals in the external address space and enables an external master to access internal address space.

This device has a 16-bit data bus only—it does not have a 32-bit data bus (internally or externally).

The EBI includes a memory controller that generates interface signals to support a variety of external memories. This includes single data rate (SDR) burst mode flash, external SRAM, and asynchronous memories. It supports up to four regions (via chip selects), each with its own programmed attributes.

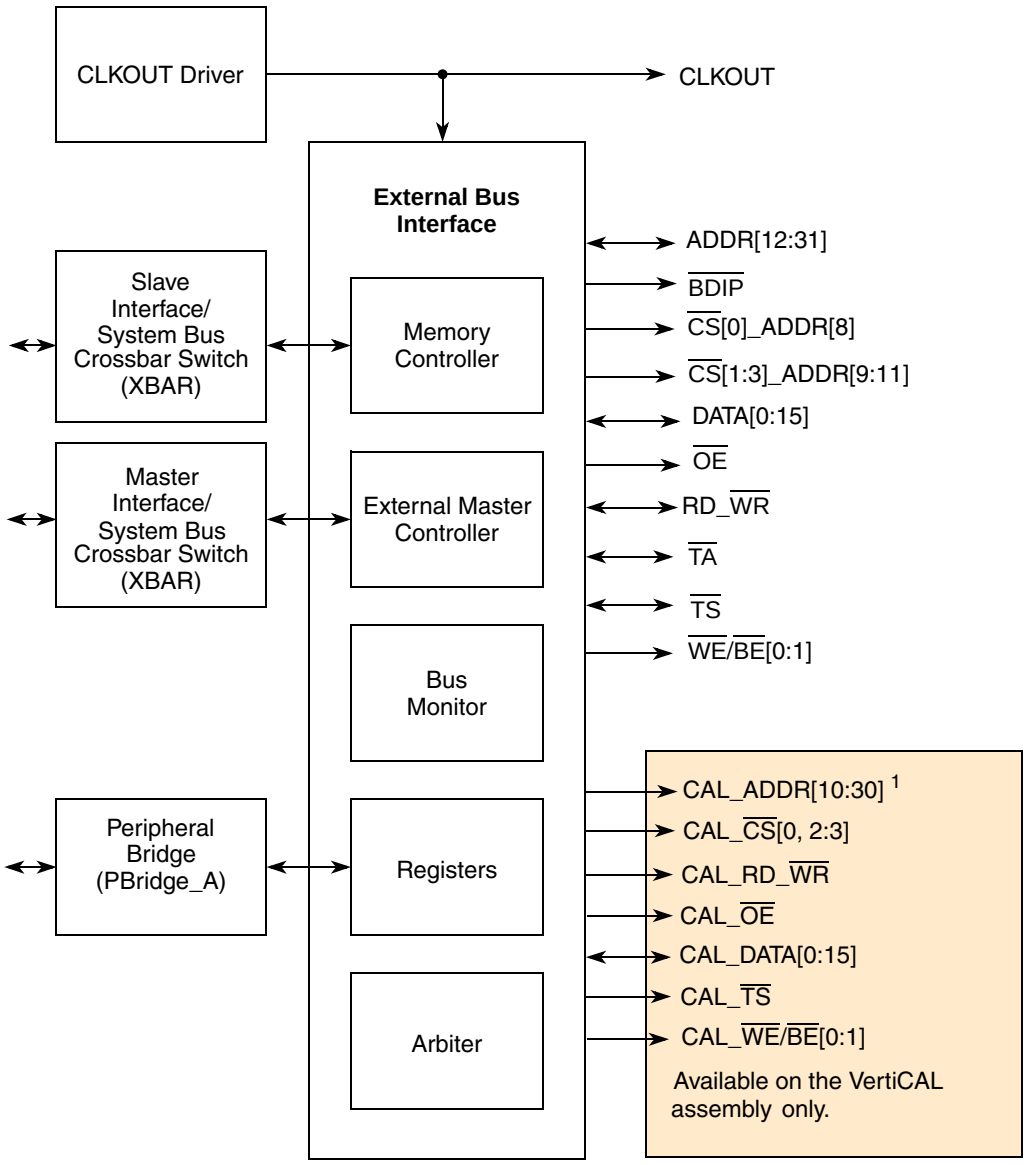
See Section 12.5.6, “Summary of Differences from MPC500,” for an overview of how the MPC5500 EBI differs from the EBI used in MPC500 devices.

12.1.1 Block Diagram

Figure 12-1 is a block diagram of the EBI. The signals shown are external pins to the MCU.

NOTE

See Chapter 2, “Signals,” as not all signals are implemented in all device packages.



1 The MPC5534 calibration bus and calibration signals are only available on the 496 VertiCal assembly.

Figure 12-1. EBI Block Diagram

12.1.2 Features

The device is designed with the following features:

- 1.8–3.3 V I/O
- Address bus—32-bit internal address bus with transfer size indication
  - 20 bits is the default EBI size for the 324 package (ADDR[12:31]).
  - 24 bits available: ADDR[12:31] is the default pin set, then $\overline{CS}$[0:3]_ADDR[8:11]_GPIO [0:3] must be configured by PCR to ADDR[8:11] to attain the 24-bit size.
- Data bus—16-bit data bus for both external memory accesses and transactions involving an external master. Although the device is designed to support a 32-bit internal data bus, the 324 package only supplies 16 balls for the external (EBI) data bus (DATA[0:15]).
- Support for external master accesses to internal addresses
- Memory controller with support for various memory types:
  - Standard SRAM
  - Synchronous burst SDR (flash or SRAM)
  - Asynchronous/legacy memory (flash or SRAM)
- Burst support (this device has no cache, therefore only the DMA can generate a burst transfer to the EBI—the core cannot.)
- Bus monitor
- Configurable wait states
- Four chip select ($\overline{CS}$[0:3]) signals in the 324 package
- Two $\overline{WE}$/$\overline{BE}$ signals ($\overline{WE}$/$\overline{BE}$[0:1])
- Calibration support is only available using the 496 VertiCal assembly:
  - Up to three calibration chip selects (CAL_$\overline{CS}$[0, 2:3])
  - Calibration address bus is 19 bits muxed with CAL_ADDR[12:30]. You can use the muxed signals CAL_ADDR[10:11] of the primary function CAL_$\overline{CS}$[2:3] to maximize the calibration bus width to 21 bits.
- Configurable bus speed modes (½ or ¼ of system clock frequency)
- Optional automatic CLKOUT gating to save power and reduce EMI
- Compatible with MPC500 external bus
  See Section 12.4.1.14, “Compatible with MPC500 External Bus (with Some Limitations).”

12.1.3 Modes of Operation

The mode of the EBI is determined by the MDIS and EXTM bits in the EBI_MCR. Configurable bus speed modes and debug mode are modes that the MCU can enter, in parallel to the EBI being configured in one of its module-specific modes.

See Section 12.3.1.3, “EBI Module Configuration Register (EBI_MCR)” for details.

12.1.3.1 Single Master Mode

In single master mode, the EBI responds to internal requests matching one of its regions, but ignores all externally-initiated bus requests. The MCU is the only master allowed to initiate transactions on the external bus in this mode; therefore, it acts as a parked master and does not have to arbitrate for the bus before starting each cycle. Single master mode is entered when EXTM = 0 and MDIS = 0 in the EBI_MCR.

12.1.3.2 External Master Mode

When the MCU is in external master mode, the EBI responds to internal requests matching one of its regions, and to external master accesses to internal address space. External master mode is entered when EXTM = 1 and MDIS = 0 in the EBI_MCR register.

Dual-master operation (multiple masters initiating external bus cycles) is not supported. A multi-MCU system with one master and one slave is supported. In a dual-controller system, the EBI is configured to internal arbitration (EARB=0 in EBI_MCR) and must be the system master.

Use the SIZEN and SIZE fields in the EBI_MCR for MCU-to-MCU transfers to indicate transfer size.

See Section 12.5.5, “Dual-MCU Operations.”

Section 12.4.2.10, “Bus Operation in External Master Mode” describes external master mode operation.

12.1.3.3 Module Disable Mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the EBI is stopped while in module disable mode. Do not make requests (other than to memory-mapped logic) to the EBI while it is in module disable mode—even if the clocks are not stopped. In this case, the operation is undefined. Module disable mode is entered when MDIS = 1 in the EBI_MCR.

12.1.3.4 Configurable Bus Speed Modes

In configurable bus speed modes, the external CLKOUT frequency is scaled to 1/2 or 1/4 of the internal system clock frequency, which remains unchanged. The EBI drives and samples signals at the scaled CLKOUT frequency rate rather than the internal system clock. This mode is selected by writing to the external clock control register in the system integration module (SIU_ECCR).

12.1.3.5 16-Bit Data Bus Mode

The EBI is limited to a 16-bit data bus, therefore, the EBI supports the 16-bit data bus mode only DATA[0:15].

To enter 16-bit data bus mode, set the data bus mode field [DBM] in the EBI master control register (EBI_MCR[DBM], EBI_BR*n*[PS] = x) to one. The reset value of DBM is 0.

External master accesses and EBI-mastered non-chip select accesses of exactly 32-bits are supported using a two (16-bit) beat transfer for both reads and writes. Data transfers that are not chip select transfers and exactly 32-bits wide are supported in standard non-burst fashion.

See Section 12.4.2.11, “Non-Chip-Select Burst in 16-bit Data Bus Mode.”

12.1.3.6 Debug Mode

When the MCU is in debug mode, the EBI remains operational.

12.2 External Signal Description

Table 12-1 alphabetically lists the external signals used by the EBI and Calibration bus. See Chapter 2, “Signals,” as not all signals are implemented in all device packages.

NOTE

The 208 package does not have an external bus interface. This chapter pertains only to the 324 package, with and without the 496 assembly.

Table 12-1. Signal Properties

Name	I/O Type	Function	Pull[1]	Package and Assembly
ADDR[8:11]	I/O	Address Bus	—	496
ADDR[12:31]	I/O	Address Bus	—	496, 324
$\overline{\text{BDIP}}$	Output	Burst Data in Progress	Up	496, 324
CAL_$\overline{\text{CS}}$[0, 2:3]	Output	Calibration Chip Selects	Up	496
CAL_ADDR[10:11] [2]	Output	Calibration address bus (output)	Up	496
CAL_ADDR[12:30]	Output	Calibration address bus (output)	Up	496
CAL_DATA[0:15]	I/O	Calibration data bus	Up	496
CAL_RD_$\overline{\text{WR}}$	I/O	Calibration read/write	Up	496
CAL_$\overline{\text{OE}}$	Output	Calibration output enable	Up	496
CAL_$\overline{\text{TS}}$	Output	Calibration transfer start	Up	496
CAL_$\overline{\text{WE}}$/$\overline{\text{BE}}$[0:1]	I/O	Calibration write/byte enables	Up	496
CLKOUT[3]	Output	Clockout	—	496, 324
$\overline{\text{CS}}$[0]	Output	Chip Selects	Up	496, 324

Table 12-1. Signal Properties (continued)

Name	I/O Type	Function	Pull[1]	Package and Assembly
$\overline{CS}$[1:3]	Output	Chip Selects	Up	496, 324
DATA[0:15]	I/O	Data Bus	—	496, 324
$\overline{OE}$	Output	Output Enable	Up	496, 324
RD_$\overline{WR}$	I/O	Read/Write	Up	496, 324
$\overline{TA}$	I/O	Transfer Acknowledge	Up	496, 324
$\overline{TS}$	I/O	Transfer Start	Up	496, 324
$\overline{WE}$/$\overline{BE}$[0:1]	Output	Write/Byte Enables	Up	496, 324

1 This column shows signals that require a weak pull (up or down) on the pin. The weak pullup/pulldown mechanisms are not available in the EBI module. Use the pad configuration registers in the system integration module (SIU_PCRs) to set the weak pullup or pulldown characteristic for each pin.

2 CAL_ADDR[10:11] are separate signals from the EBI block, and are muxed onto CAL_$\overline{CS}$[2:3] pins on MCU.

3 The CLKOUT signal is driven by the FMPLL module.

12.2.1 Detailed Signal Descriptions

See Chapter 2, “Signals,” as not all signals are implemented in all device packages.

12.2.1.1 Address Lines 8–31 (ADDR[8:31])

The ADDR[8:31] signals specify the physical address of the bus transaction. The 24 address lines are bits 8–31 of the EBI 32-bit internal address bus. Bits 0–7 are internally driven by the EBI for externally initiated accesses depending on the internal slave accessed.

See Section 12.4.2.10.1, “Address Decoding for External Master Accesses,” for more details.

ADDR[8:31] is driven by the EBI or an external master depending on which device controls the external bus. The 324 BGA packaged devices use ADDR[12:31] (24 bits available if CS[0:3] are configured to ADDR[8:11].

See Section 6.4.1.12.1, “Pad Configuration Registers 0–3 (SIU_PCR0–SIU_PCR3)”).

12.2.1.2 Burst Data in Progress ($\overline{BDIP}$)

$\overline{BDIP}$ is asserted by a master requesting the next data beat to follow the current data beat.

$\overline{BDIP}$ is driven by the EBI or an external master depending on which one is in control of the external bus. This signal is driven by the EBI on all EBI-mastered external burst cycles, but is only sampled by burst mode memories that have a burst pin.

See Section 12.4.2.5, “Burst Transfer.”

12.2.1.3 Clockout (CLKOUT)

CLKOUT is a general-purpose clock output signal to connect to the clock input of SDR external memories and in some cases to the input clock of another MCU in multi-master configurations.

12.2.1.4 Chip Selects 0–3 ($\overline{CS}$[0:3])

$\overline{CS}$[*n*] is asserted by the master to indicate that this transaction is targeted for a particular memory bank.

The chip selects are driven by the EBI or an external master depending on which module controls the external bus. The chip select is driven in the same clock as the assertion of $\overline{TS}$ and valid address, and is kept valid until the cycle is terminated.

See Section 12.4.1.4, “Memory Controller with Support for Various Memory Types” for details on chip select operation.

12.2.1.5 Calibration Chip Selects (CAL_$\overline{CS}$[0, 2:3]) — 496 Assembly Only

CAL_$\overline{CS}$[*n*] is asserted by the master to indicate the transaction is targeted for a memory bank on the calibration external bus.

The calibration chip selects are driven only by the EBI. External master accesses on the calibration bus are not supported. In all other aspects, the calibration chip selects operate exactly as the primary chip selects. See Section 12.4.1.4, “Memory Controller with Support for Various Memory Types for details on chip select operation.

12.2.1.6 Calibration Signals

Calibration signals are only available using the 496 VertiCal assembly and the 324 package for this device.

DATA is not driven by the EBI during a calibration bus access. During a calibration bus access, the non-calibration bus signals (other than DATA) are held in a negated state, with the exception of RD_$\overline{WR}$ and ADDR, which reflect the same values shown on the calibration version of those signals. Because the $\overline{TS}$ and $\overline{CS}$ signals are held negated on the EBI (non-calibration bus) during calibration accesses, no transfer occurs on the EBI.

During a EBI bus access, the calibration bus signals (other than CAL_DATA) are held in a negated state. CAL_DATA is not driven during non-calibration accesses.

12.2.1.7 Data Lines 0–15 (DATA[0:15])

In the 324 BGA package, DATA[0:15] transmits the data for the current transaction.

DATA[0:15] is driven by the EBI when it owns the external bus and it initiates a write transaction to an external device. The EBI also drives DATA[0:15] when an external master owns the external bus and initiates a read transaction to an internal module.

DATA[0:15] is driven by an external device during a read transaction from the EBI. An external master drives DATA[0:15] when it owns the bus and initiates a write transaction to an internal module or shared external memory.

For 8-bit transactions, the byte lanes not selected for the transfer do not supply valid data.

12.2.1.8 Output Enable ($\overline{OE}$)

$\overline{OE}$ is used to indicate when an external memory is permitted to drive back read data. External memories must have their data output buffers off when $\overline{OE}$ is negated. $\overline{OE}$ is only asserted for chip select accesses.

$\overline{OE}$ is driven by the EBI or an external master depending on who owns the external bus.

- Read cycles—$\overline{OE}$ is asserted one clock after $\overline{TS}$ asserts, and is held until the transfer terminates.
- Write cycles—$\overline{OE}$ is negated throughout the cycle.

During a calibration bus access, $\overline{OE}$ is held negated.

12.2.1.9 Read/Write (RD_$\overline{WR}$)

RD_$\overline{WR}$ indicates whether the current transaction is a read access or a write access.

RD_$\overline{WR}$ is driven by the EBI or an external master depending on who owns the external bus. RD_$\overline{WR}$ is driven in the same clock as the assertion of $\overline{TS}$ and valid address, and is kept valid until the cycle is terminated.

During a calibration bus access, RD_$\overline{WR}$ reflects the same value as the CAL_RD_$\overline{WR}$ signal.

12.2.1.10 Transfer Acknowledge ($\overline{TA}$)

$\overline{TA}$ is asserted to indicate that the slave device has received the data (and completed the access) for a write cycle, or returned data for a read cycle. If the transaction is a burst read, $\overline{TA}$ is asserted for each one of the transaction beats. For write transactions, $\overline{TA}$ is only asserted once at access completion, even if more than one write data beat is transferred.

$\overline{TA}$ is driven by the EBI when the access is controlled by the chip selects or when an external master initiates the transaction to an internal module. Otherwise, $\overline{TA}$ is driven by the slave device to which the current transaction was addressed.

During a calibration bus access, $\overline{TA}$ is held negated.

See Section 12.4.2.9, “Termination Signals Protocol” for more details.

12.2.1.11 Transfer Start ($\overline{TS}$)

$\overline{TS}$ is asserted by the current bus owner to indicate the start of a transaction on the external bus.

$\overline{TS}$ is driven by the EBI or an external master depending on who owns the external bus. $\overline{TS}$ is only asserted for the first clock cycle of the transaction, and is negated in the successive clock cycles until the end of the transaction.

During a calibration bus access, $\overline{TS}$ is held negated.

12.2.1.12 Write/Byte Enables ($\overline{WE}/\overline{BE}$)

Write and byte enables ($\overline{WE}/\overline{BE}$[0:1]) are used to enable program operations to a particular memory. Write enable is used for write operations only. Byte enable is used for read and write operations to configure the byte lanes. These signals are set by the WEBS bit in the SIU_PCR registers. $\overline{WE}/\overline{BE}$ are only asserted for chip select accesses.

$\overline{WE}/\overline{BE}$ signals are driven by the EBI or an external master depending on which one controls the external bus. During a calibration bus access, $\overline{WE}/\overline{BE}$ signals are held negated.

See Section 12.4.1.12, “Two Write/Byte Enable (WE/BE) Signals” for more details on $\overline{WE}/\overline{BE}$ functionality.

12.2.2 Signal Function and Direction by Mode

The EBI operating mode is configured using two fields in the EBI Master Control register (EBI_MCR): EXTM and MDIS. Their settings determine which EBI signals are valid and the I/O direction. When a signal is configured for non-EBI function in the EBI_MCR, the EBI always negates the signal if the EBI controls the corresponding pad (determined by SIU configuration). Table 12-2 lists the function and direction of the external signals in each of the EBI modes of operation. The clock signals are not included because they are output only (from the FMPLL module) and are not affected by EBI modes.

See Section 12.3.1.3, “EBI Module Configuration Register (EBI_MCR)” for details on the EXTM and MDIS bits.

Table 12-2. Signal Function According to EBI Mode Settings

Signal Name	Modes		
	Module Disable Function **EXTM = *n*, MDIS = 1**	**Single Master Function I/O Direction** **EXTM = 0, MDIS = 0**	**External Master Function I/O Direction** **EXTM = 1, MDIS = 0**
ADDR[8:11] [1]	non-EBI function	Address bus (output)	Address bus (I/O)[2]
ADDR[12:30]	non-EBI function	Address bus (output)	Address bus (I/O)[2]
$\overline{BDIP}$	non-EBI function	Burst data in progress (output)[3]	
$\overline{CS}$[0:3] [1]	non-EBI function	Chip selects (output)[3]	
DATA[0:15]	non-EBI function	Data bus (I/O)	
$\overline{OE}$	non-EBI function	Output enable (output)	
RD_$\overline{WR}$	non-EBI function	Read/write (output)	Read/write (I/O)
$\overline{TA}$	non-EBI function	Transfer acknowledge (I/O)	
$\overline{TS}$	non-EBI function	Transfer start (output)	Transfer start (I/O)
$\overline{WE}/\overline{BE}$[0:1]	non-EBI function	Write/byte enables (output)[3]	
CAL_$\overline{CS}$[0, 2:3] [4]	non-EBI function	Chip selects (output)	
CAL_ADDR[12:30] [4]	non-EBI function	Calibrate the address bus (output)	

Table 12-2. Signal Function According to EBI Mode Settings (continued)

Signal Name	Modes		
	Module Disable Function EXTM = *n*, MDIS = 1	Single Master Function I/O Direction EXTM = 0, MDIS = 0	External Master Function I/O Direction EXTM = 1, MDIS = 0
CAL_DATA[0:15] [4]	non-EBI function	Calibrate the data bus (I/O)	
CAL_$\overline{OE}$ [4]	non-EBI function	Calibrate the bus to enable output	
CAL_$\overline{TS}$ [4]	non-EBI function	Calibrate the transfer start (output)	
CAL_$\overline{WE}$/$\overline{BE}$[0:1] [4]	non-EBI function	Write/byte enables (output)[3]	

1 These signals are muxed with the chip select ($\overline{CS}$) signals on this device. Use the pad configuration registers (PCR) in the system integration module (SIU) to configure the balls to use the address signals *or* chip select signals–not both.

2 All I/O signals are tri-stated by the EBI when not actively involved in a transfer.

3 Although external master accesses can drive these pins, the EBI three-states the pins and does not sample them for input.

4 The calibration signals for this device are available on the 324 package with the VertiCal assembly only.

NOTE

The open-drain mode of the pad configuration module is not used for any EBI signals. For a description of how signals are driven by multiple devices in external master mode, see Section 12.4.2.10, “Bus Operation in External Master Mode.”

12.3 Memory Map and Register Definition

NOTE

The 208 package does not have an external bus interface. This chapter pertains to the 324 and 496 packages only.

Table 12-3 is a memory map of the EBI registers.

Table 12-3. EBI Memory Map

Address	Register Name	Register Description	Bits
Base (0xC3F8_4000)	EBI_MCR	EBI module configuration register	32
Base + 0x0004	—	Reserved	—
Base + 0x0008	EBI_TESR	EBI transfer error status register	32
Base + 0x000C	EBI_BMCR	EBI bus monitor control register	32
Base + 0x0010	EBI_BR0	EBI base register bank 0	32
Base + 0x0014	EBI_OR0	EBI option register bank 0	32
Base + 0x0018	EBI_BR1	EBI base register bank 1	32
Base + 0x001C	EBI_OR1	EBI option register bank 1	32

Table 12-3. EBI Memory Map (continued)

Base + 0x0020	EBI_BR2	EBI base register bank 2	32
Base + 0x0024	EBI_OR2	EBI option register bank 2	32
Base + 0x0028	EBI_BR3	EBI base register bank 3	32
Base + 0x002C	EBI_OR3	EBI option register bank 3	32
Calibration Registers			
Base + (0x0030–0x003C)	—	Reserved	—
Base + 0x0040	EBI_CAL_BR0	EBI Calibration Base Register Bank 0	32
Base + 0x0044	EBI_CAL_OR0	EBI Calibration Option Register Bank 0	32
Base + 0x0048	EBI_CAL_BR1	EBI Calibration Base Register Bank 1	32
Base + 0x004C	EBI_CAL_OR1	EBI Calibration Option Register Bank 1	32
Base + 0x0050	EBI_CAL_BR2	EBI Calibration Base Register Bank 2	32
Base + 0x0054	EBI_CAL_OR2	EBI Calibration Option Register Bank 2	32
Base + 0x0058	EBI_CAL_BR3	EBI Calibration Base Register Bank 3	32
Base + 0x005C	EBI_CAL_OR3	EBI Calibration Option Register Bank 3	32

12.3.1 Register Descriptions

12.3.1.1 Writing EBI Registers While a Transaction is in Progress

When an EBI transaction is in progress, do *not* write to EBI registers except when the transaction is:

- From the internal or external master
- Within two CLKOUT cycles after a transaction completes, which allows the internal state machines to enter the IDLE state.

Exceptions that can be written while an EBI transaction is in progress are:

- All bits in EBI_TESR
- SIZE, SIZEN fields in EBI_MCR

If you write to other fields in the EBI registers, or when an EBI transaction is in progress and is not one of the exception cases described, the operations are indeterminable.

See Section 12.5.1, "Booting from External Memory," for additional information.

12.3.1.2 Separate Input Clock for Registers

The EBI registers are accessed with a clock signal separate from the clock used by the rest of the EBI. In module disable mode, the clock used by the non-register portion of the EBI is disabled to reduce power consumption. The clock signal dedicated to the registers, however, allows access to the registers even while the EBI is in the module disable mode. Flag bits in the EBI transfer error status register (EBI_TESR), however, are set and cleared with the clock used by the non-register portion of the EBI. Consequently, in module disable mode, the EBI_TESR does not have a clock signal and is therefore not writable.

12.3.1.3 EBI Module Configuration Register (EBI_MCR)

The EBI_MCR contains bits that configure various attributes associated with EBI operation.

Base (0xC3F8_4000) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SIZEN	SIZE		0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ACGE	EXTM	EARB	0	0	0	0	0	0	MDIS	0	0	0	0	0	DBM
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Figure 12-2. EBI Module Configuration Register (EBI_MCR)

Table 12-4. EBI_MCR Field Descriptions

Field	Description
0–4	Reserved
5 SIZEN	SIZE enable. The SIZEN bit enables the control of transfer size by the SIZE field for external master transactions to internal address space. See Section 12.5.5.3, “Setting the transfer size.” 0 Invalid value 1 Enable transfer size controlled by SIZE field **Note:** You must change this value to 1 to control the data transfer size.
6–7 SIZE	Transfer size. The SIZE field defines the transfer size of external master transactions to internal address space when SIZEN=1. See Section 12.5.5.3, “Setting the transfer size.” This field is ignored when SIZEN=0. SIZE encoding: 00 Invalid value 01 Byte 10 16-bit 11 Invalid value
8–15	Reserved
16 ACGE	Automatic CLKOUT gating enable. Enables the EBI feature of turning off CLKOUT (holding it high) during idle periods in-between external bus accesses. 0 Automatic CLKOUT gating is disabled 1 Automatic CLKOUT gating is enabled
17 EXTM	External master mode. Enables the external master mode of operation when MDIS = 0. When MDIS = 1, the EXTM bit is not used, and is treated as 0. In external master mode, an external master on the external bus can access any internal memory-mapped space while the internal e200z3 core is fully operational. When EXTM = 0, only internal masters can access the internal memory space. See Section 12.5.5, “Dual-MCU Operations.“ 0 External master mode is inactive (single master mode) 1 External master mode is active **Note:** Only master/slave systems support the EXTM functionality.
18–24	Reserved
25 MDIS	Module disable mode. Allows the clock to be stopped to the non-memory mapped logic in the EBI, effectively putting the EBI in a software controlled power-saving state. No external bus accesses can be performed when the EBI is in module disable mode (MDIS = 1). Most registers remain accessible in this mode. See Section 12.1.3.3, “Module Disable Mode,” for more information. 0 Module disable mode is inactive 1 Module disable mode is active
26–30	Reserved
31 DBM	Data bus mode. Sets the EBI to 16-bit data bus mode. 0 Invalid value 1 16-bit data bus mode is used

12.3.1.4 EBI Transfer Error Status Register (EBI_TESR)

The EBI_TESR contains a bit for each type of transfer error on the external bus. A bit set to logic 1 indicates what type of transfer error occurred since the last time the bits were cleared. Each bit can be cleared by reset or by writing a 1 to it. Writing a 0 has no effect.

Base + 0x0008 Access: R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	BMTF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-3. EBI Transfer Error Status Register (EBI_TESR)

Table 12-5. EBI_TESR Field Descriptions

Field	Description
0–30	Reserved
31 BMTF	Bus monitor timeout flag. Set if the cycle was terminated by a bus monitor timeout. 0 No error 1 Bus monitor timeout occurred This bit can be cleared by writing a 1 to it.

12.3.1.5 EBI Bus Monitor Control Register (EBI_BMCR)

The EBI_BMCR controls the timeout period of the bus monitor, and whether it is enabled or disabled.

Base + 0x000C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BMT [0:7]								BME	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0

Figure 12-4. EBI Bus Monitor Control Register (EBI_BMCR)

Table 12-6. EBI_BMCR Field Descriptions

Field	Description
0–15	Reserved
16–23 BMT [0:7]	Bus monitor timing. Defines the timeout period, in 8 external bus clock resolution, for the bus monitor. See Section 12.4.1.6, "Bus Monitor," for more details on bus monitor operation. $\text{Timeout Period} = \frac{2 + (8 \times \text{BMT})}{\text{External Bus Clock Frequency}}$
24 BME	Bus monitor enable. Controls whether the bus monitor is enabled for internal to external bus cycles. Regardless of the BME value, the bus monitor is always disabled for chip select accesses, since these always use internal $\overline{TA}$ and thus have no danger of hanging the system. 0 Disable bus monitor 1 Enable bus monitor (for non-chip select accesses only)
25–31	Reserved

12.3.1.6 EBI Base Registers 0–3 (EBI_BR*n*) and EBI Calibration Base Registers 0–3 (EBI_CAL_BR*n*)

The EBI_BR*n* are used to define the base address and other attributes for the corresponding chip select. The EBI_CAL_BR*n* are used to define the base address and other attributes for the corresponding calibration chip select.

Base + 0x0010 (EBI_BR0) Access: R/W
Base + 0x0018 (EBI_BR1)
Base + 0x0020 (EBI_BR2)
Base + 0x0028 (EBI_BR3)
Base + 0x0040 (EBI_CAL_BR0)
Base + 0x0048 (EBI_CAL_BR1)
Base + 0x0050 (EBI_CAL_BR2)
Base + 0x0058 (EBI_CAL_BR3)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	1	BA												
W																
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BA	0	0	0	PS	0	0	0	0	BL	WEBS	TBDIP	0	0	BI	V
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 12-5. EBI Base Registers 0–3 (EBI_BR*n*) and EBI Calibration Base Registers 0–3 (EBI_CAL_BR*n*)

Table 12-7. EBI_BR*n* and EBI_CAL_BR*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0–16
BA
[0:16]</td><td>Base address. Compared to the corresponding unmasked address signals among ADDR[0:16] of the internal address bus to determine if a memory bank controlled by the memory controller is being accessed by an internal bus master.
Note: The upper 3 bits of the base address (BA) field, EBI_BRn[0:2], and EBI_CAL_BRn[0:2], are tied to a fixed value of 001. These bits reset to their fixed value.</td></tr>
<tr><td>17–19</td><td>Reserved</td></tr>
<tr><td>20
PS</td><td>Port size. Determines the data bus width of transactions to this chip select bank.[1]
0 Invalid value
1 16-bit port. The calibration port size must be 16-bits wide.</td></tr>
<tr><td>21–24</td><td>Reserved</td></tr>
<tr><td>25
BL</td><td>Burst length. Determines the amount of data transferred in a burst for this chip select, measured in 32-bit words. The number of beats in a burst is automatically determined by the EBI according to the port size bit (PS) so the burst fetches the number of words chosen by BL.
0 Invalid value
1 4-word burst length
Note:
<table>
<tr><th>Value</th><th>Burst Length [1]</th><th>PS</th><th># Beats in Burst [2]</th></tr>
<tr><td>1</td><td>4-word</td><td>1</td><td>8</td></tr>
</table>
[1] Total amount of data fetched in a burst transfer.
[2] Number of external data beats used in external burst transfer. The size of each beat is determined by PS value.</td></tr>
<tr><td>26
WEBS</td><td>Write enable/byte select. Controls the functionality of the $\overline{WE}$/$\overline{BE}$[0:1] signals.
0 The $\overline{WE}$/$\overline{BE}$[0:1] signals function as $\overline{WE}$[0:1].
1 The $\overline{WE}$/$\overline{BE}$[0:1] signals function as $\overline{BE}$[0:1].</td></tr>
<tr><td>27
TBDIP</td><td>Toggle burst data in progress. Determines how long the $\overline{BDIP}$ signal is asserted for each data beat in a burst cycle. See Section 12.4.2.5.1, “TBDIP Effect on Burst Transfer,” for details.
0 Assert $\overline{BDIP}$ throughout the burst cycle, regardless of wait state configuration.
1 Only assert $\overline{BDIP}$ (BSCY + 1) external bus cycles before expecting subsequent burst data beats.</td></tr>
<tr><td>28–29</td><td>Reserved</td></tr>
<tr><td>30
BI</td><td>Burst inhibit. Determines whether or not burst read accesses are allowed for this chip select bank.
0 Enable burst accesses for this bank.
1 Disable burst accesses for this bank. This is the default value out of reset.</td></tr>
<tr><td>31
V</td><td>Valid bit. Indicates that the contents of this base register and option register pair are valid. The appropriate $\overline{CS}$ signal does not assert unless the corresponding V-bit is set.
0 This bank is not valid.
1 This bank is valid.</td></tr>
</table>

[1] The the value of PS bit in the EBI_MCR[DBM] register is not used and the value used is always 1.

12.3.1.7 EBI Option Registers 0–3 (EBI_OR*n*) and EBI Calibration Option Registers 0–3 (EBI_CAL_OR*n*)

The EBI_OR*n* registers are used to define the address mask and other attributes for the corresponding chip select. The EBI_CAL_OR*n* registers are used to define the address mask and other attributes for the corresponding calibration chip select.

Base + 0x0014 (EBI_OR0) Access: R/W
Base + 0x001C (EBI_OR1)
Base + 0x0024 (EBI_OR2)
Base + 0x002C (EBI_OR3)
Base + 0x0044 (EBI_CAL_OR0)
Base + 0x004C (EBI_CAL_OR1)
Base + 0x0054 (EBI_CAL_OR2)
Base + 0x005C (EBI_CAL_OR3)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	1	1	1	AM												
W																
Reset	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AM	0	0	0	0	0	0	0	SCY				0	BSCY		0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-6. EBI Option Registers 0–3 (EBI_OR*n*) and EBI Calibration Option Registers

Table 12-8. EBI_OR*n* and EBI_CAL_OR*n* Field Descriptions

Field	Description
0–16 AM [0:16]	Address mask. Allows masking of any corresponding bits in the associated base register. Masking the address independently allows external devices of different size address ranges to be used. Any clear bit masks the corresponding address bit. Any set bit causes the corresponding address bit to be used in comparison with the address pins. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. This field can be read or written at any time. **Note:** The upper 3 bits of the address mask (AM) field, EBI_ORx[0:2], and EBI_CAL_ORn[0:2], are tied to a fixed value of 111. These bits reset to their fixed value.
17–23	Reserved
24–27 SCY [0:3]	Cycle length in clocks. Represents the number of wait states (external bus cycles) inserted after the address phase in the single cycle case, or in the first beat of a burst, when the memory controller handles the external memory access. Values range from 0 to 15. This is the main parameter for determining the length of the cycle. • The total cycle length for the first beat (including the $\overline{TS}$ cycle): (2 + SCY) external clock cycles See Section 12.5.3.1, “Example Wait State Calculation”.

Table 12-8. EBI_OR*n* and EBI_CAL_OR*n* Field Descriptions (continued)

Field	Description
28	Reserved
29–30 BSCY [0:1]	Burst beats length in clocks. This field determines the number of wait states (external bus cycles) inserted in all burst beats except the first, when the memory controller starts handling the external memory access and thus is using SCY[0:3] to determine the length of the first beat. • Total memory access length for each beat: (1 + BSCY) External Clock Cycles • Total cycle length (including the $\overline{TS}$ cycle): (2 + SCY) + [(Number of Beats – 1) x (BSCY + 1)] **Note:** The number of beats (4, 8, 16) is determined by BL and PS bits in the base register. 00 0-clock cycle wait states (1 clock per data beat) 01 1-clock cycle wait states (2 clocks per data beat) 10 2-clock cycle wait states (3 clocks per data beat) 11 3-clock cycle wait states (4 clocks per data beat)

12.4 Functional Description

NOTE

The 208 package does not have an external bus interface. This chapter pertains to the 324 package and 496 assembly only.

12.4.1 External Bus Interface Features

12.4.1.1 32-Bit Address Bus

The transfer size for an external transaction is indicated by the SIZE and SIZEN fields in the EBI_MCR register during the clock when the address is valid. Valid transaction sizes are 8, 16, and 32 bits. The 324 package has 20 address lines pinned out externally (24 bits available if $\overline{CS}$[0:3] are configured as ADDR[8:11]). A full 32-bit decode is done internally to determine the target of the transaction and whether to assert a chip select.

See Section 6.4.1.12.1, “Pad Configuration Registers 0–3 (SIU_PCR0–SIU_PCR3)”

12.4.1.2 16-Bit Data Bus

A 16-bit data bus mode is available using the DBM bit in EBI_MCR.

See Section 12.1.3.5, “16-Bit Data Bus Mode.”

12.4.1.3 Support for External Master Accesses to Internal Addresses

The EBI allows an external master to access internal address space when the EBI is configured for external master mode in the EBI_MCR.

Section 12.4.2.10, “Bus Operation in External Master Mode” describes the external master operations.

12.4.1.4 Memory Controller with Support for Various Memory Types

The EBI contains a memory controller that supports a variety of memory types:

- Standard SRAM
- Synchronous burst mode to memory (flash or external SRAM)
- Asynchronous memory (flash or external SRAM) and peripherals

Each $\overline{CS}$ bank is configured with a pair of base and option registers. Each time an internal to external bus cycle access is requested, the following occurs:

As shown in Figure 12-7, the internal address is compared with the base address of each valid base register (17 bits are masked).

If a match occurs in one memory bank, the BR and OR bank attribute values control the memory access.

If a match occurs in more than one memory bank, the matched bank with the lowest bank address handles the memory access. For example, bank 0 is selected over memory bank 1.

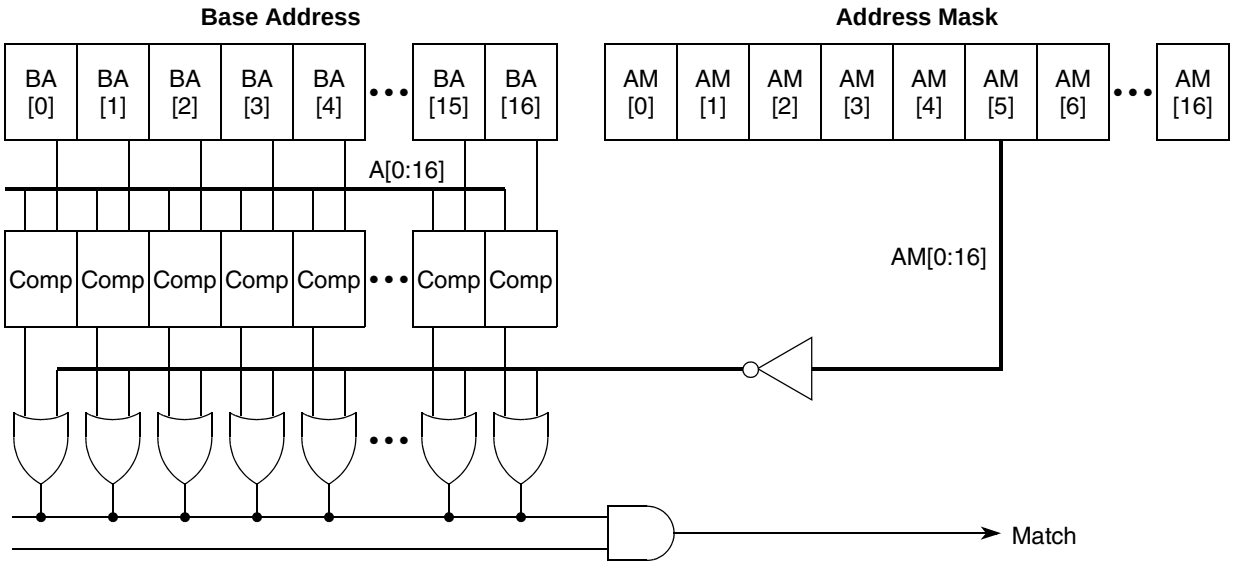


Figure 12-7. Bank Base Address and Match Structure

A match on a valid calibration chip select register overrides a match on any non-calibration chip select register, with CAL_$\overline{CS}$[0] having the highest priority. Thus the full priority of the chip selects is: CAL_$\overline{CS}$[0, 2, 3] and then $\overline{CS}$[0, 1, 2, 3].

When a match occurs on one of the chip select banks, all its attributes (from the base and option registers) are selected for the functional operation of the external memory access:

- Number of wait states for a single memory access, and for any beat in a burst access
- Burst enable
- Port size for the external accessed device

See the following sections for a full description of all chip select attributes:

Section 12.3.1.6, “EBI Base Registers 0–3 (EBI_BRn) and EBI Calibration Base Registers 0–3 (EBI_CAL_BRn),”

Section 12.3.1.7, “EBI Option Registers 0–3 (EBI_ORn) and EBI Calibration Option Registers 0–3 (EBI_CAL_ORn),”

When no match occurs on any of the chip select banks, the default transfer attributes shown in Table 12-9 are used.

NOTE

The port size (PS) value defaults to 32-bits. You must ensure the port size (PS) is set to 16-bits before initiating the transfer.

Table 12-9. Default Attributes for Transfers Other than Chip Select

Chip Select Attribute ($\overline{CS}$[0:3])	Default Value	Comment
PS	0	32-bit port size
BL	0	BL = “don’t care” (burst is disabled)
WEBS	0	Write enables
TBDIP	0	TBDIP = “don’t care” (burst is disabled)
BI	1	Burst inhibited
SCY	0	SCY = “don’t care”
BSCY	0	BSCY = “don’t care”

12.4.1.5 Burst Support (Wrapped Only)

This device has no cache, therefore the core does not support burst transfers. The eDMA only can launch a burst transfer to external memory.

The EBI supports burst read accesses to external burstable memory. The EBI in 16-bit data bus mode (EBI_MCR[DBM] = 1) does not support burst writes, except for 32-bit two-beat non-chip select burst writes to 32-bit. This allows 32-bit coherent accesses to another MCU.

Internal requests to write more than 32-bits externally are divided into separate 16-bit external transactions according to the port size.

To enable bursts to a memory region, clear the BI (Burst Inhibit) bit in the base register. External burst lengths of four and eight words are supported. Burst length is configured for each chip select by using the BL bit in the base register.

See Section 12.4.2.5, “Burst Transfer” for more details.

In 16-bit data bus mode, a special two-beat burst case is supported for reads and writes for 32-bit non-chip select accesses only.

See Section 12.4.2.11, “Non-Chip-Select Burst in 16-bit Data Bus Mode”.

Burst writes are not supported except from a 32-bit non-chip-select writes in 16-bit data bus mode. Internal requests to write more than 32 bits externally are divided into separate 16-bit external transactions according to the port size.

See Section 12.4.2.6, “Small Accesses (Small Port Size and Short Burst Length)” for more detail on these cases.

12.4.1.6 Bus Monitor

When enabled (via the BME bit in the EBI_BMCR), the bus monitor detects when no $\overline{\text{TA}}$ assertion is received within a maximum timeout period for non-chip select accesses (accesses that use external $\overline{\text{TA}}$). The timeout for the bus monitor is specified by the BMT field in the EBI_BMCR. Each time a timeout error occurs, the BMTF bit is set in the EBI_TESR. The timeout period is measured in external bus (CLKOUT) cycles. Thus the effective real-time period is multiplied (by two or four) when a configurable bus speed mode is used, even though the BMT field itself is unchanged.

12.4.1.7 Port Size Configuration per Chip Select (16 Bits)

The EBI supports memories with data widths of 16 bits. The port size (PS) for a chip select is configured using the PS bit in the base register.

12.4.1.8 Port Size Configuration per Calibration Chip Select (16 Bits)

The port size for calibration must be 16 bits wide.

12.4.1.9 Configurable Wait States

From 0 to 15 wait states can be programmed for any cycle that the memory controller generates, using the SCY bits in the option register. From zero to three wait states between burst beats can be programmed using the BSCY bits in the option register.

12.4.1.10 Four Chip Select ($\overline{\text{CS}}$[0:3]) Signals

The EBI contains four chip select signals, controlling four independent memory banks.

See Section 12.4.1.4, “Memory Controller with Support for Various Memory Types,” for more details on chip select bank configuration.

12.4.1.11 Support for Dynamic Calibration with Up to Three Chip Selects

The EBI contains three calibration chip select signals (CAL_$\overline{\text{CS}}$[0,2,3]), controlling three independent memory banks on an optional second external bus for calibration.

See Section 12.4.2.12, “Calibration Bus Operation” for more details on using the calibration bus.

12.4.1.12 Two Write/Byte Enable ($\overline{WE}$/$\overline{BE}$) Signals

The functionality of the $\overline{WE}$/$\overline{BE}$[0:1] signals depends on the value of the WEBS bit in the corresponding base register. Setting WEBS to 1 configures these pins as $\overline{BE}$[0:1], while clearing it configures the pins as $\overline{WE}$[0:1]. $\overline{WE}$[0:1] signals are asserted only during write accesses, while $\overline{BE}$[0:1] signals are asserted for both read and write accesses. The timing of the $\overline{WE}$/$\overline{BE}$[0:1] signals remains the same in both cases.

The upper write/byte enable ($\overline{WE}$/$\overline{BE}$[0]) indicates that the upper eight bits of the data bus (DATA[0:7]) contain valid data during a write/read cycle. The lower write/byte enable ($\overline{WE}$/$\overline{BE}$[1]) indicates that the lower eight bits of the data bus (DATA[8:15]) contain valid data during a write/read cycle.

The write/byte enable lines affected in a transaction are shown in Table 12-10. Only big endian byte ordering is supported by the EBI.

Table 12-10. Write/Byte Enable Signals Function -- 324 BGA

Transfer Size	Address		16-Bit Port Size[1]	
	A[30]	A[31]	$\overline{WE}$/$\overline{BE}$[0]	$\overline{WE}$/$\overline{BE}$[1]
8 bits	0	0	0	1
	0	1	1	0
	1	0	0	1
	1	1	1	0
16 bits	0	0	0	0
	1	0	0	0
Burst	0	0	0	0

1 Also applies when DBM =1 for 16-bit data bus mode.

12.4.1.13 Optional Automatic CLKOUT Gating

The EBI can hold the external CLKOUT pin high when the EBI internal master state machine is idle and no requests are pending. The EBI outputs a signal to the pads logic in the MCU to disable CLKOUT. This feature is disabled out of reset, and can be enabled or disabled by the ACGE bit in the EBI_MCR.

12.4.1.14 Compatible with MPC500 External Bus (with Some Limitations)

The EBI is compatible with the external bus of the MPC500 parts, meaning that it supports most devices supported by the MPC500 family of parts. However, there are some differences between this EBI and that of the MPC500 parts that you must be aware of before assuming that an MPC500-compatible device works with this EBI.

See Section 12.5.6, “Summary of Differences from MPC500,” for details.

NOTE

Due to testing and complexity concerns, master/slave operation between an MPC55xx and MPC5xx is not guaranteed. Multi-master operations are not supported on this device.

12.4.2 External Bus Operations

The following sections provide a functional description of the external bus, the bus cycles provided for data transfer operations, bus arbitration, and error conditions.

12.4.2.1 External Clocking

The CLKOUT signal sets the frequency of operation for the bus interface directly. Internally, the MCU uses a phase-locked loop (PLL) circuit to generate a master clock for all of the MCU circuitry (including the EBI) which is phase-locked to the CLKOUT signal. In general, all signals for the EBI are specified with respect to the rising-edge of the CLKOUT signal, and they are guaranteed to be sampled as inputs or changed as outputs with respect to that edge.

12.4.2.2 Reset

Upon detection of internal reset, the EBI immediately terminates all transactions.

12.4.2.3 Basic Transfer Protocol

The basic transfer protocol defines the sequence of actions that must occur on the external bus to perform a complete bus transaction. A simplified scheme of the basic transfer protocol is shown in Figure 12-8.

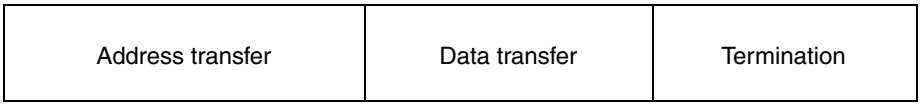


Figure 12-8. Basic Transfer Protocol

In single-master mode, the EBI is the permanent bus owner.

The address transfer phase specifies the address for the transaction and the transfer attributes that describe the transaction. The signals related to the address transfer phase are $\overline{TS}$, ADDR, $\overline{CS}$[0:3], RD_$\overline{WR}$, and $\overline{BDIP}$. The address and its related signals (with the exception of $\overline{TS}$, $\overline{BDIP}$) are driven on the bus with the assertion of the $\overline{TS}$ signal, and kept valid until the bus master receives $\overline{TA}$ asserted (the EBI holds them one cycle beyond $\overline{TA}$ for writes and external $\overline{TA}$ accesses). For writes with internal $\overline{TA}$, RD_$\overline{WR}$ is not held one cycle past $\overline{TA}$.

The data transfer phase transfers data from master to slave (on write cycles), or from slave to master (on read cycles). The data phase can transfer a:

- Single beat of data (1–2 bytes) for non-burst operations; or
- A 2-, 4-, 8-, or 16-beat burst of data (EBI_MCR[DBM] = 1, at 2 bytes per beat) when burst is enabled.

On a write cycle, the master must not drive write data until after the address transfer phase is complete. This avoids electrical contentions when switching between drivers. The master must start driving write data one cycle after the address transfer cycle. The master can stop driving the data bus as soon as it samples the $\overline{TA}$ line asserted on the rising edge of CLKOUT.

For chip select accesses, use the output enable ($\overline{OE}$) signal to indicate that the external device can drive data onto the bus during an MCU read cycle. To prevent bus contentions for chip select accesses, you must use $\overline{OE}$ to determine when the external device can drive the bus.

Read Timing—On a read cycle, the master accepts the data bus contents as valid on the rising edge of CLKOUT when the $\overline{TA}$ signal asserts and is sampled. See Figure 12-10 for an example of read timing. The termination phase completes by the assertion of $\overline{TA}$ (normal termination).

Write Timing—To facilitate asynchronous write support, the EBI keeps driving valid write data on the data bus until one clock after the rising edge, when RD_$\overline{WR}$ (and $\overline{WE}$ for chip select accesses) are negated. See Figure 12-15 for an example of write timing.

Section 12.4.2.9, “Termination Signals Protocol.” describes in detail the termination phase.

12.4.2.4 Single-Beat Transfer

The flow and timing diagrams in this section assume that the EBI is configured in single master mode. Therefore, arbitration is not needed and is not shown in these diagrams.

See Section 12.4.2.10, “Bus Operation in External Master Mode,” to read how the flow and timing diagrams change for external master mode.

12.4.2.4.1 Single-Beat Read Flow

The handshakes for a single-beat read cycle are illustrated in the following flow and timing diagrams.

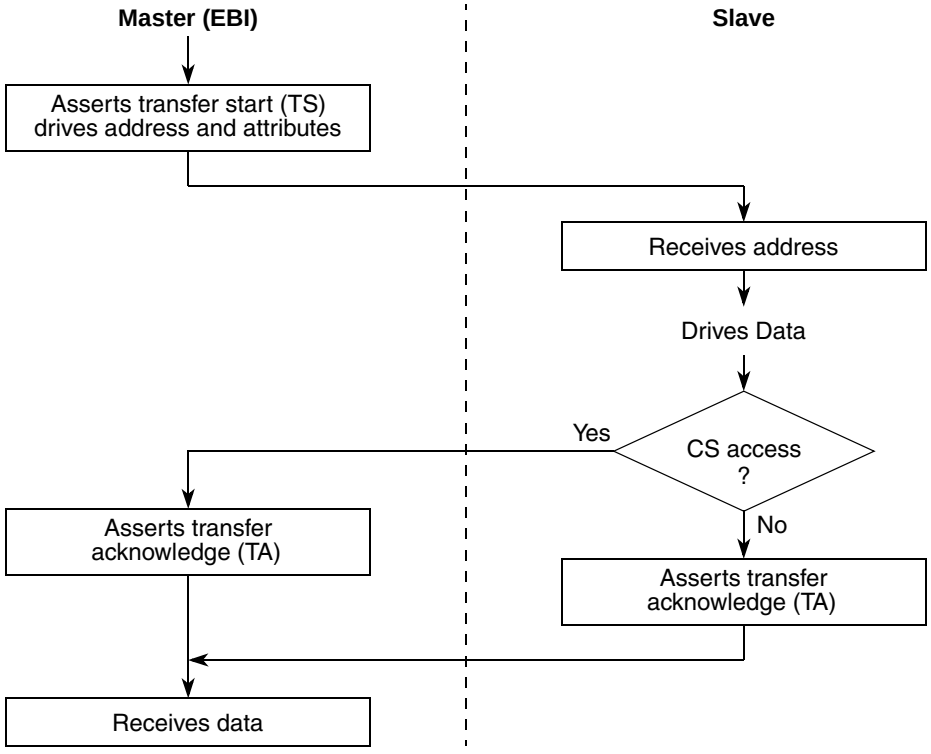


Figure 12-9. Basic Flow Diagram of a Single-Beat Read Cycle

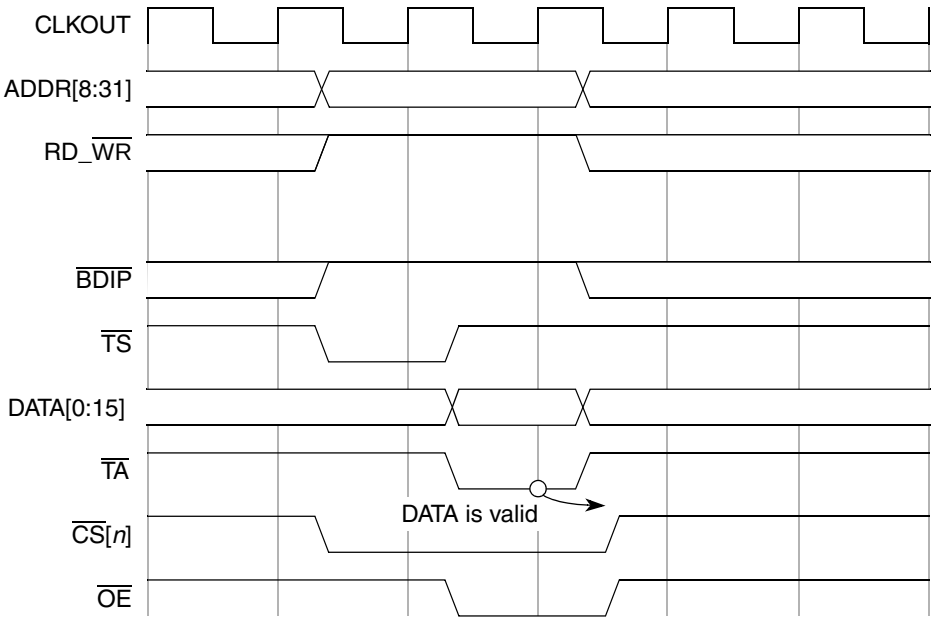
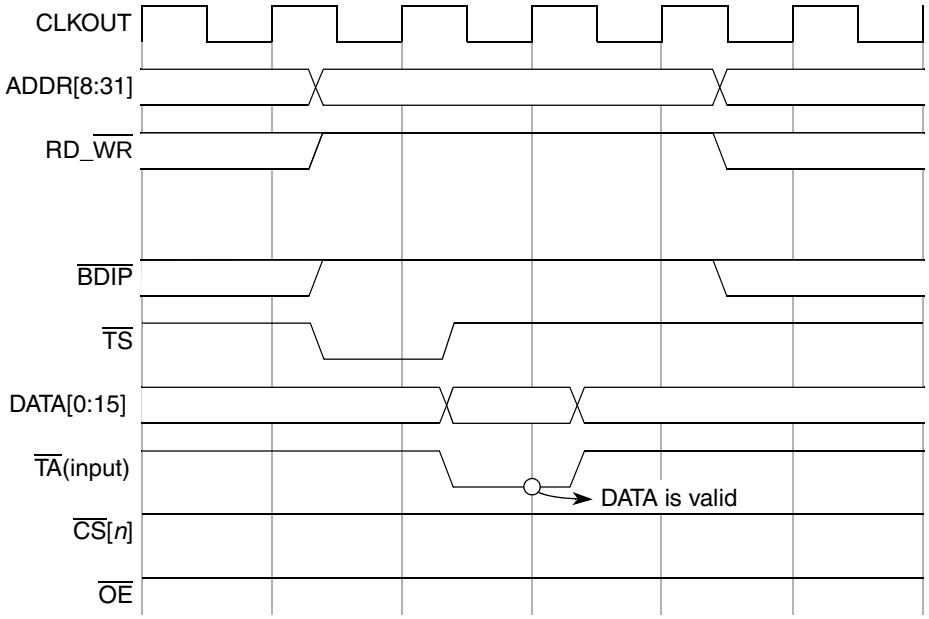


Figure 12-10. Single-Beat 16-bit Read Cycle, CS Access, Zero Wait States



* The EBI drives address and control signals an extra cycle because it uses a latched version of the external TA (1 cycle delayed) to terminate the cycle.

Figure 12-11. Single-Beat 16-bit Read Cycle, CS Access, One Wait State

12.4.2.4.2 Single-beat Write Flow

The handshakes for a single-beat write cycle are illustrated in the following flow and timing diagrams.

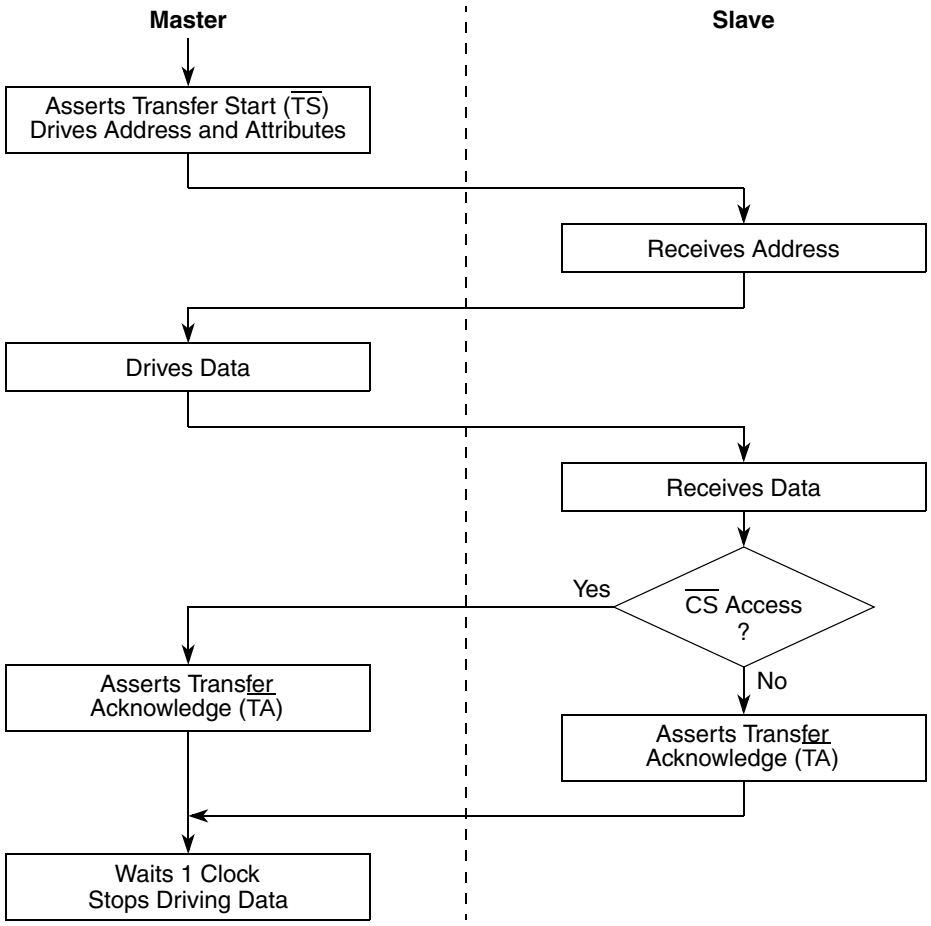


Figure 12-12. Basic Flow Diagram of a Single-beat Write Cycle

12.4.2.4.3 Back-to-Back Accesses

Due to internal bus protocol, one dead cycle is necessary between back-to-back external bus accesses that are not part of a set of small accesses. A dead cycle refers to a cycle between the $\overline{TA}$ of a previous transfer and the $\overline{TS}$ of the next transfer.

See Section 12.4.2.6, “Small Accesses (Small Port Size and Short Burst Length)” for small access timing.

NOTE

In some cases, $\overline{CS}$ remains asserted during a dead cycle, such as the cases of back-to-back writes or read-after-write to the same chip-select. See Figure 12-16 and Figure 12-17.

Besides a dead cycle, in most cases, back-to-back accesses on the external bus do not cause any change in the timing from that shown in the previous diagrams, and the two transactions are independent of each

other. Back-to-back accesses where the first access ends with an externally-driven $\overline{TA}$. In these cases, an extra cycle is required between the end of the first access and the $\overline{TS}$ assertion of the second access.

See Section 12.4.2.9, "Termination Signals Protocol," for more details.

The following diagrams show a few examples of back-to-back accesses on the external bus.

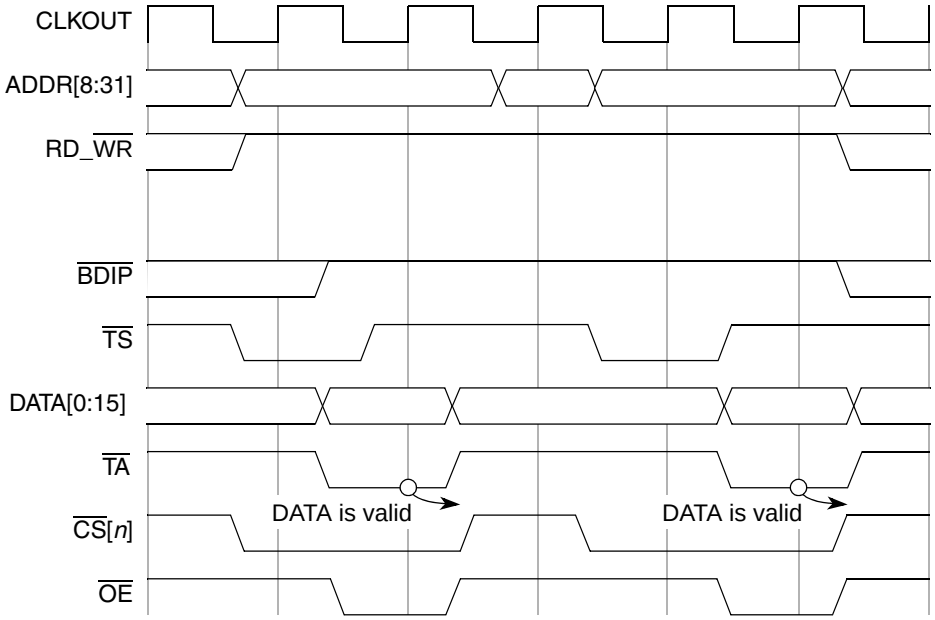


Figure 12-13. Back-to-Back 16-bit Reads to the Same $\overline{CS}$ Bank

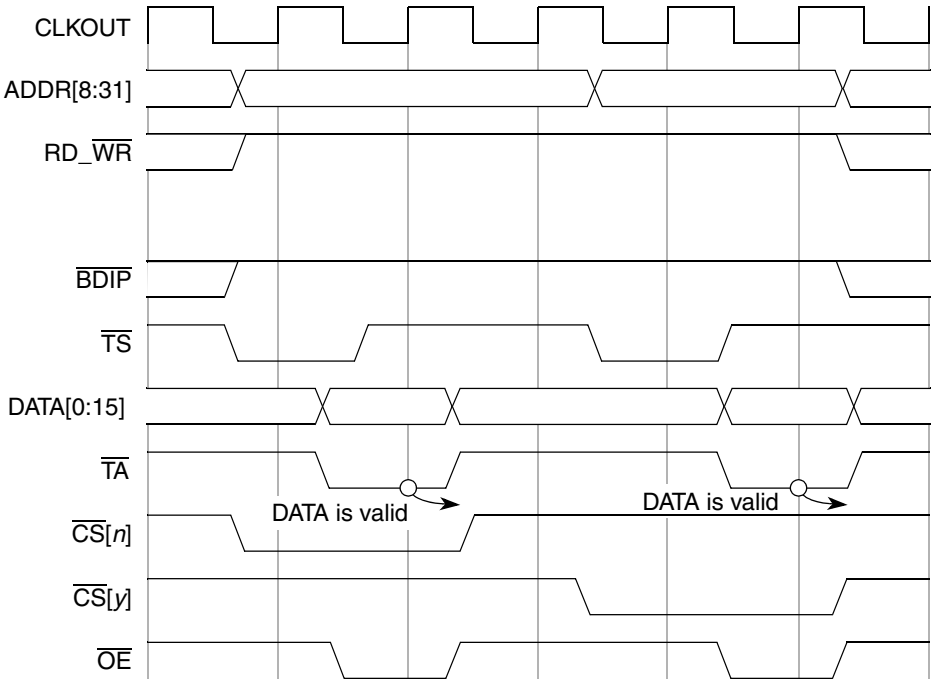


Figure 12-14. Back-to-Back 16-bit Reads to Different $\overline{CS}$ Banks

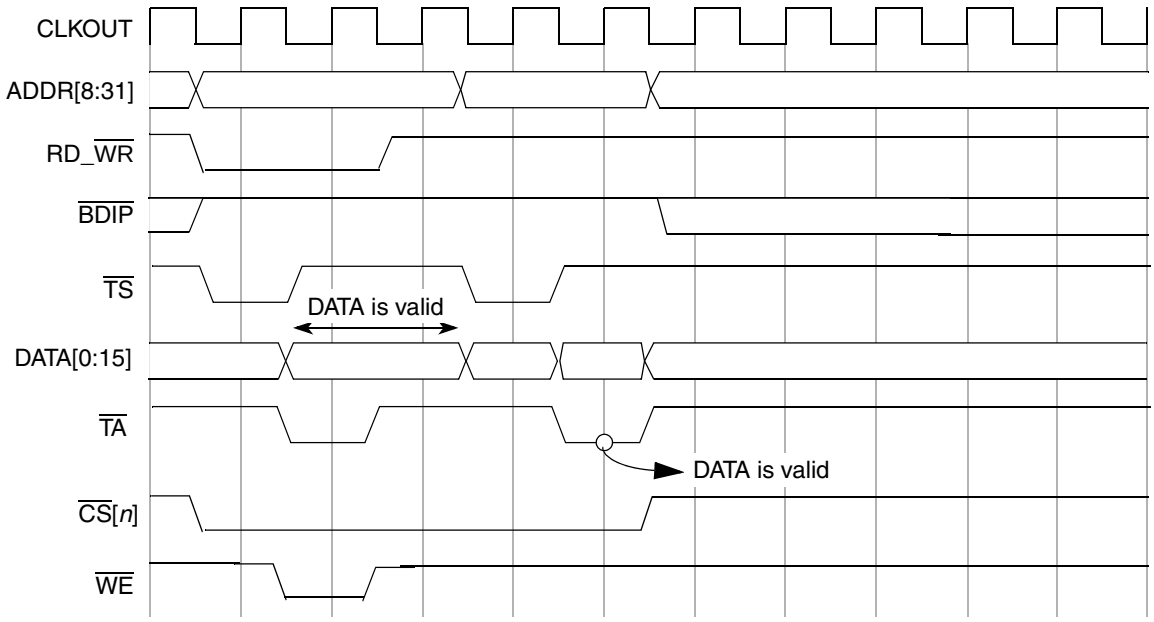


Figure 12-15. Write After Read to the Same CS Bank

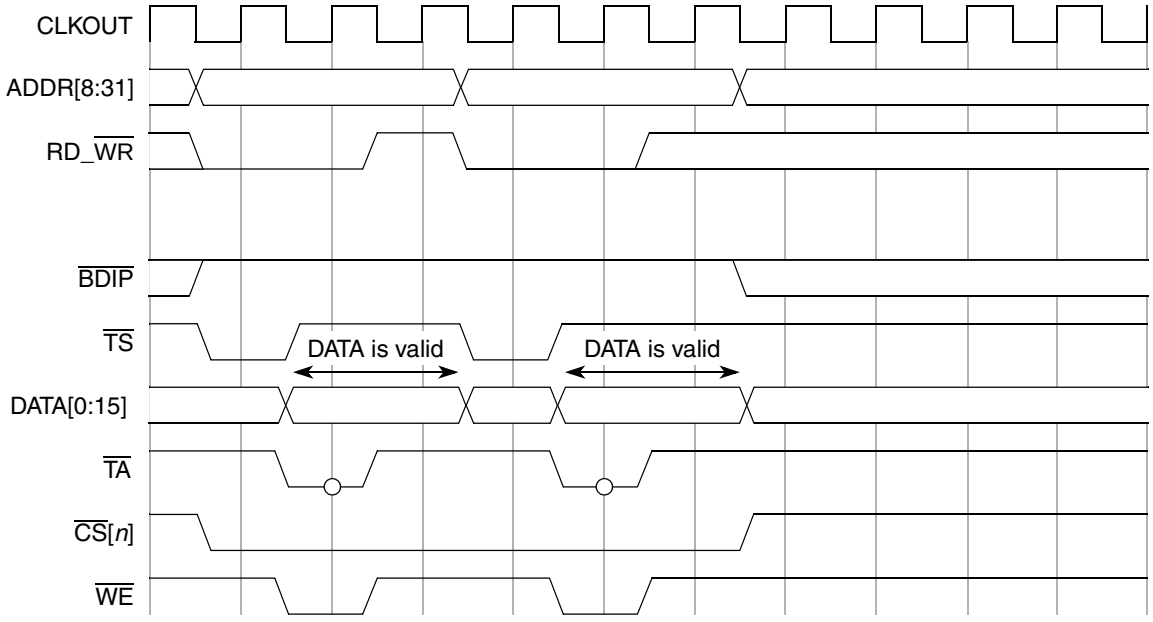


Figure 12-16. Back-to-Back 16-bit Writes to the Same CS Bank

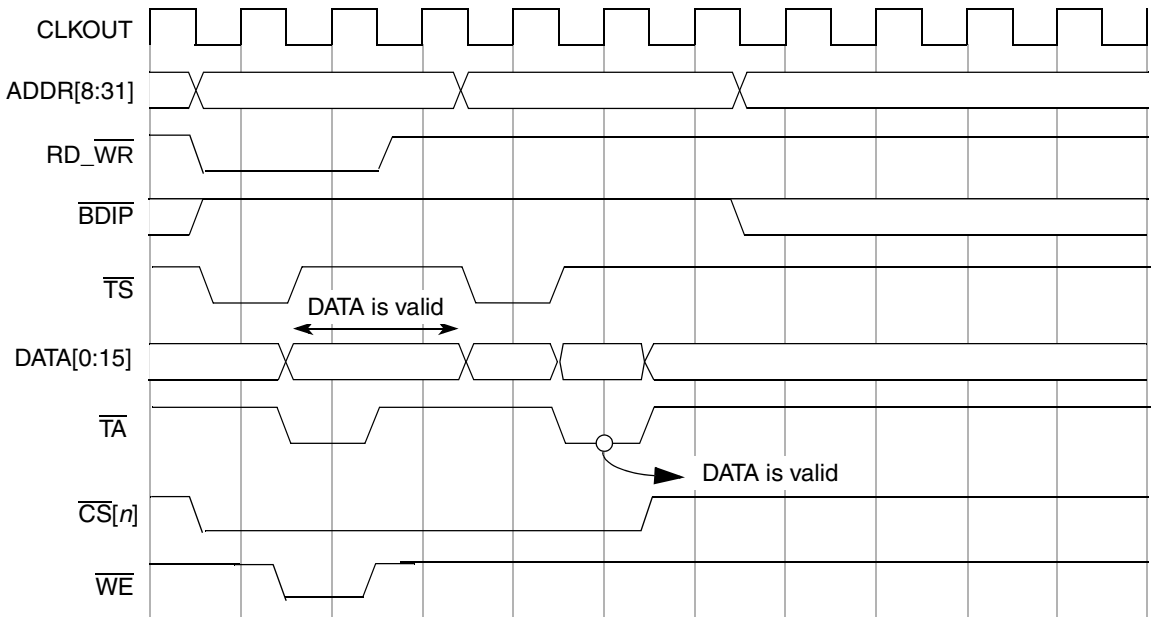


Figure 12-17. Read After Write to the Same CS Bank

12.4.2.5 Burst Transfer

On all burst cycles, the EBI requires that addresses are aligned on a doubleword boundary. The EBI supports burst wrapping for 32-byte critical-doubleword-first transfers. Bursting is supported for internally-requested 32-byte read accesses to external devices that use:

- Chip select ($\overline{CS}$) accesses only
- 32-bit non-chip select accesses using a two-beat 16-bit for each word

Other than these exceptions, all accesses from an external master to devices operating without a chip select are always single beat. If an internal request to the EBI indicates a burst transfer less than 32 bytes, one or more single-beat external transfers are used—not by an external burst transfer.

An 8-word wrapping burst reads eight 32-bit words by supplying a starting address that points to one of the words (doubleword aligned), and requires the memory device to sequentially drive each word on the data bus.

The selected slave device must internally increment ADDR[27:30] of the supplied address for each transfer until the address of the 8-word boundary is reached, and then wraps the address to the beginning of the 8-word boundary. The address and transfer attributes supplied by the EBI do not change during the transfers, and the EBI terminates each beat transfer by asserting $\overline{TA}$.

Table 12-11 shows the burst order of beats returned for an 8-word burst to a 32-bit memory interface using two-beat 16-bit accesses.

Table 12-11. Wrap Bursts Order

Burst Starting Address ADDR[27:30]	Burst Order (two-beat 16-bit access = 1 word)
00	word0 –> word1 –> word2 –> word3 –> word4 –> word5 –> word6 –> word7
01	word2 –> word3 –> word4 –> word5 –> word6 –> word7 –> word0 –> word1
10	word4 –> word5 –> word6 –> word7 –> word0 –> word1 –> word2 –> word3
11	word6 –> word7 –> word0 –> word1 –> word2 –> word3 –> word4 –> word5

Burst transfers default to external memory with a 32-bit bus and 8-word burst length. The EBI can burst from 16-bit port size memories, using twice as many external beats to fetch the data. The EBI can also burst from 16-bit or 32-bit memories that have a 4-word burst length (BL = 1 in the appropriate base register). In this case, two external 4-word burst transfers (wrapping on 4-word boundary) are performed to fulfill the internal 8-word request. This operation is considered atomic by the EBI, so the EBI does not allow other master accesses to intervene between the transfers.

During burst cycles, the $\overline{\text{BDIP}}$ (burst data in progress) signal indicates the duration of the burst data. During the data phase of a burst read cycle, the EBI receives data from the addressed slave. If the EBI needs more than one data transfer, it asserts the $\overline{\text{BDIP}}$ signal. Upon receiving the word prior to the last word, the EBI negates $\overline{\text{BDIP}}$. Therefore, the slave stops driving new data on the rising edge of the clock after $\overline{\text{BDIP}}$ negates.

Some slave devices internally configure burst length and timing, which does not support using the $\overline{\text{BDIP}}$ signal. In this case, $\overline{\text{BDIP}}$ is driven by the EBI normally, but the output is ignored by memory and the burst transfer mechanism is determined by the internal configuration of the EBI and slave device. When the TBDIP bit is set in the base register, the timing for $\overline{\text{BDIP}}$ is altered.

See Section 12.4.2.5.1, “TBDIP Effect on Burst Transfer,” for this timing.

Burst writes are not supported by the EBI except for 32-bit non-chip select accesses in 16-bit data bus mode. For all other burst writes, the EBI negates $\overline{\text{BDIP}}$ during write cycles.

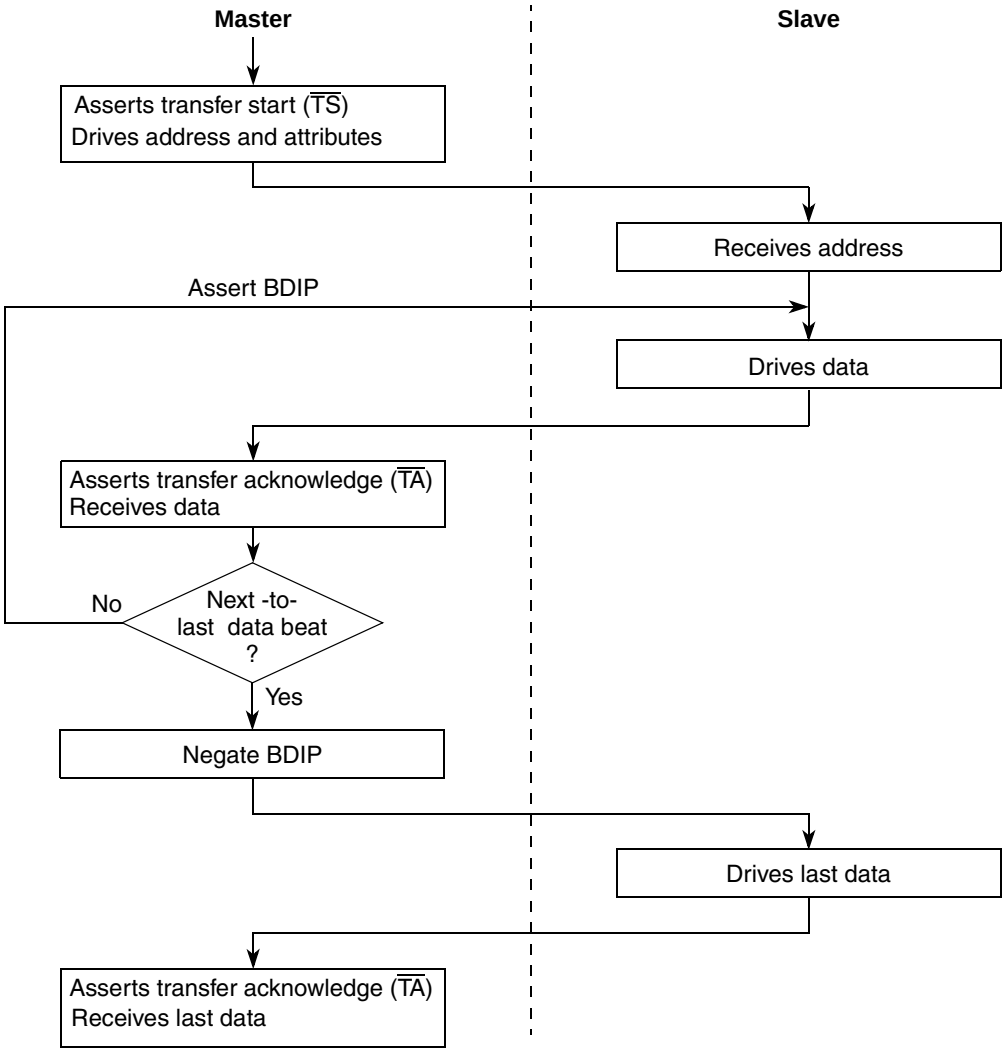


Figure 12-18. Basic Flow Diagram of a Burst Read Cycle

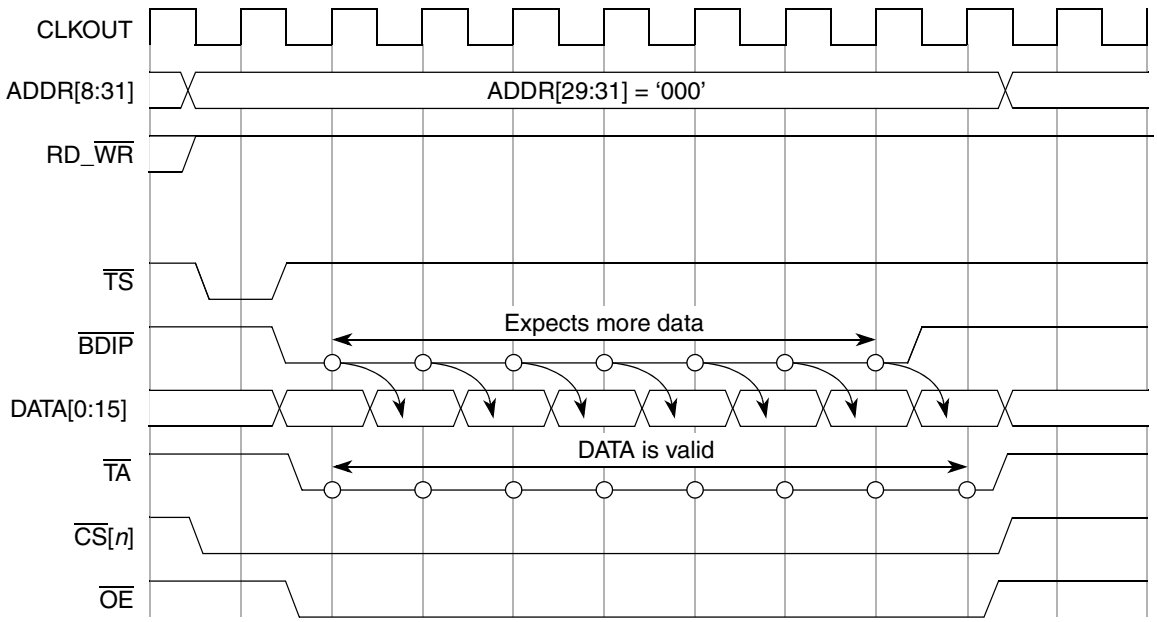


Figure 12-19. Burst 16-bit Read Cycle, Zero Wait States

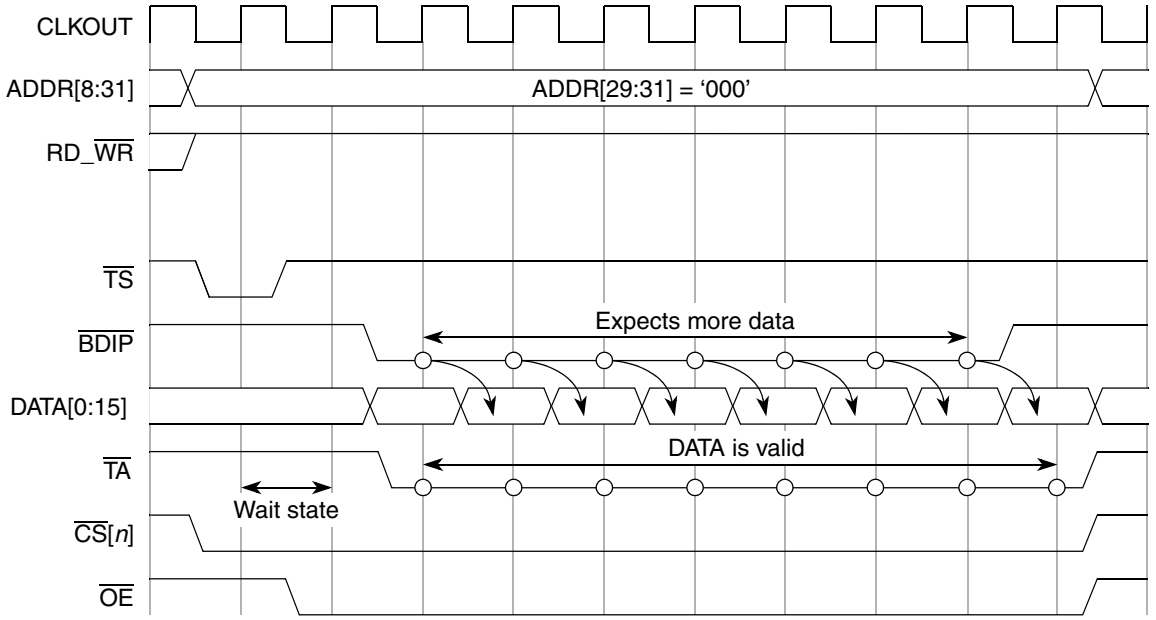


Figure 12-20. Burst 16-bit Read Cycle, One Initial Wait State

12.4.2.5.1 TBDIP Effect on Burst Transfer

Some memories require different timing on the $\overline{\text{BDIP}}$ signal than the default to run burst cycles. Using the default value of TBDIP = 0 in the appropriate EBI base register results in $\overline{\text{BDIP}}$ being asserted (SCY+1) cycles after the address transfer phase, and being held asserted throughout the cycle regardless of the wait

states between beats (BSCY). Figure 12-21 shows an example of the TBDIP = 0 timing for a 4-beat burst with BSCY = 1.

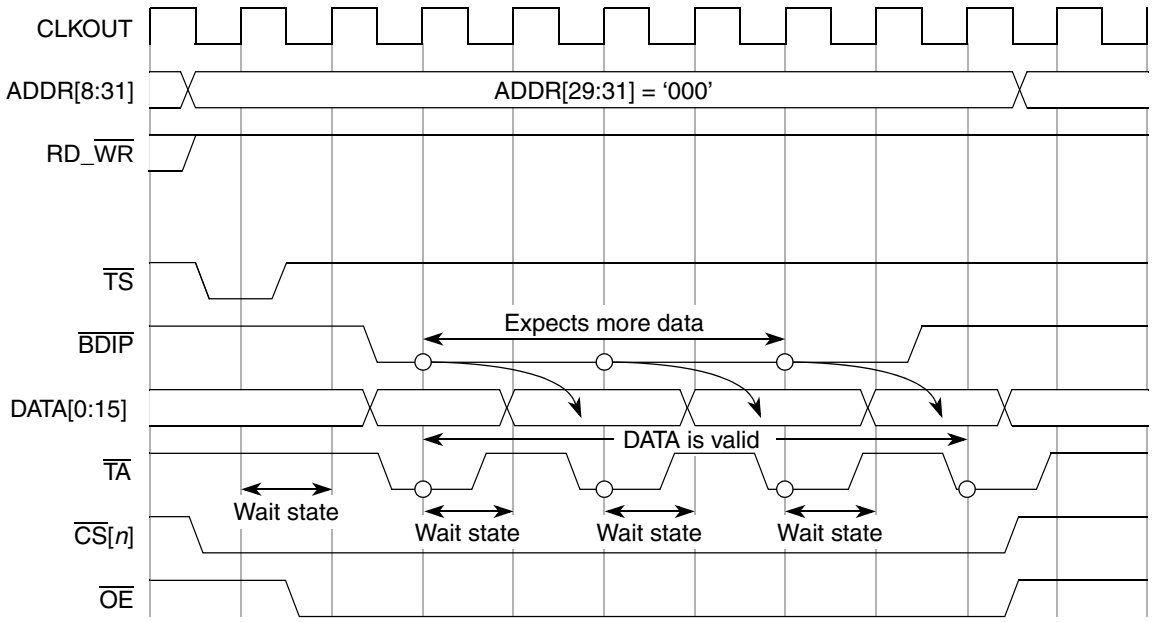


Figure 12-21. Burst 32-bit Read Cycle, One Wait State between Beats, TBDIP = 0

When using TBDIP = 1, the BDIP behavior changes to toggle between every beat when BSCY is a non-zero value. Figure 12-22 shows an example of the TBDIP = 1 timing for the same four-beat burst shown in Figure 12-21.

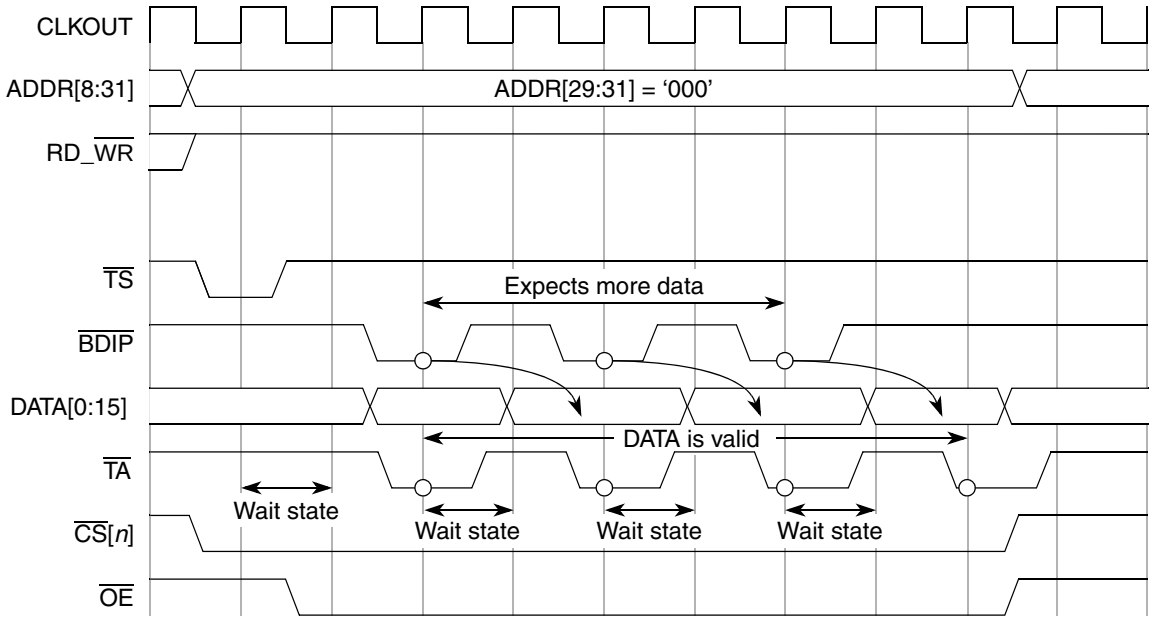


Figure 12-22. Burst 32-bit Read Cycle, One Wait State between Beats, TBDIP = 1

12.4.2.6 Small Accesses (Small Port Size and Short Burst Length)

In this context, a small access refers to an access whose burst length and port size are such that the number of bytes requested by the internal master cannot all be fetched (or written) in one external transaction. This is the case when the base register's burst length bit (EBI_BR*n*[BL]) and port size bit (EBI_BR*n*[PS]) are set such that one of two situations occur:

- Burst accesses are inhibited and the number of bytes requested by the master is greater than the port size (16) can accommodate in a single access.
- Burst accesses are enabled and the number of bytes requested by the master is greater than the selected burst length (8 words).

If this is the case, the EBI initiates multiple transactions until all the requested data is transferred. All the transactions initiated to complete the data transfer are considered as an atomic transaction, so the EBI does not allow other unrelated master accesses to intervene between the transfers.

Table 12-12 shows all the combinations of burst length, port size, and requested byte count that cause the EBI to run multiple external transactions to fulfill the request.

Table 12-12. Small Access Cases

Byte Count Requested by Internal Master	Burst Length	Port Size	# External Accesses to Fulfill Request
Non-burstable Chip-Select Banks (BI = 1) or Non-Chip-Select Access			
4	1 beat	16-bit	1[1]
8	1 beat	16-bit	4
32	1 beat	16-bit	16
Burstable Chip-Select Banks (BI = 0)			
32 (8 words)	8 beats	16-bit	2

[1] 16-bit data bus mode (DBM = 1), one 2-beat burst access is performed and this is not considered a small access case. See Section 12.4.2.11, "Non-Chip-Select Burst in 16-bit Data Bus Mode" for this special DBM = 1 case.

In most cases, the timing for small accesses is the same as for normal single-beat and burst accesses, except that multiple back-to-back external transfers are executed for each internal request. These transfers have no additional dead cycles between external accesses that are not present for back-to-back stand-alone transfers except for the case of writes with an internal request size greater than 64 bits.

See Section 12.4.2.6.2, "Small Access Example #2: 32-byte Write with External TA."

The following sections show a few examples of small accesses. The timing for the remaining cases in Table 12-12 can be extrapolated from these and the other timing diagrams in this document.

12.4.2.6.1 Small Access Example #1: 32-bit Write to 16-bit Port

Figure 12-23 shows an example of a 32-bit write to a 16-bit port, requiring two 16-bit external transactions.

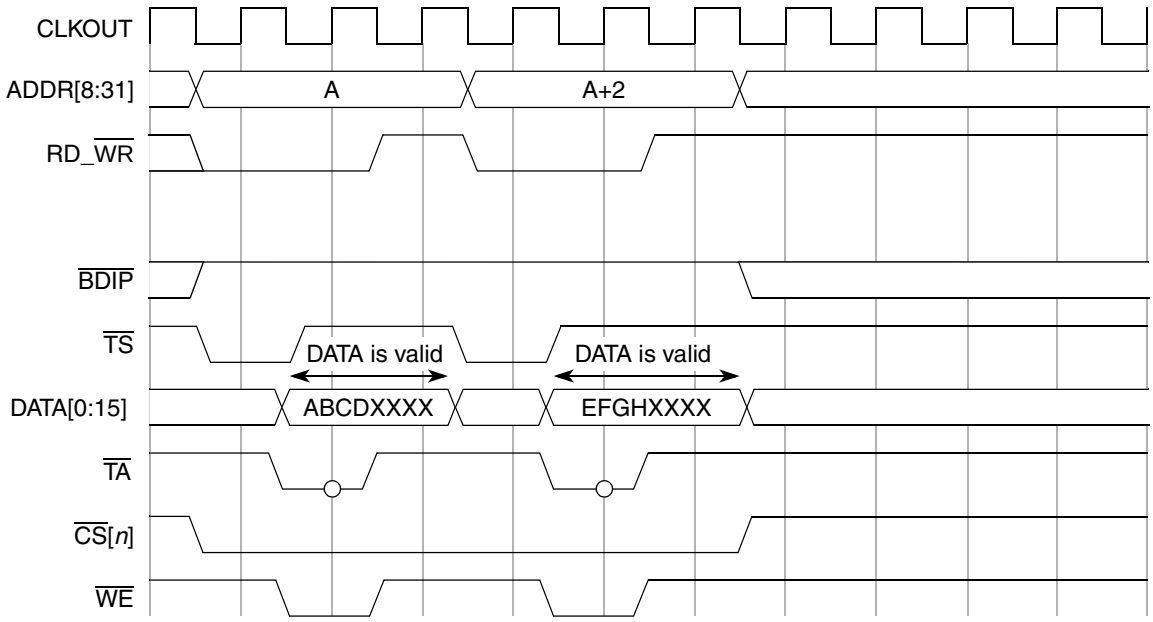
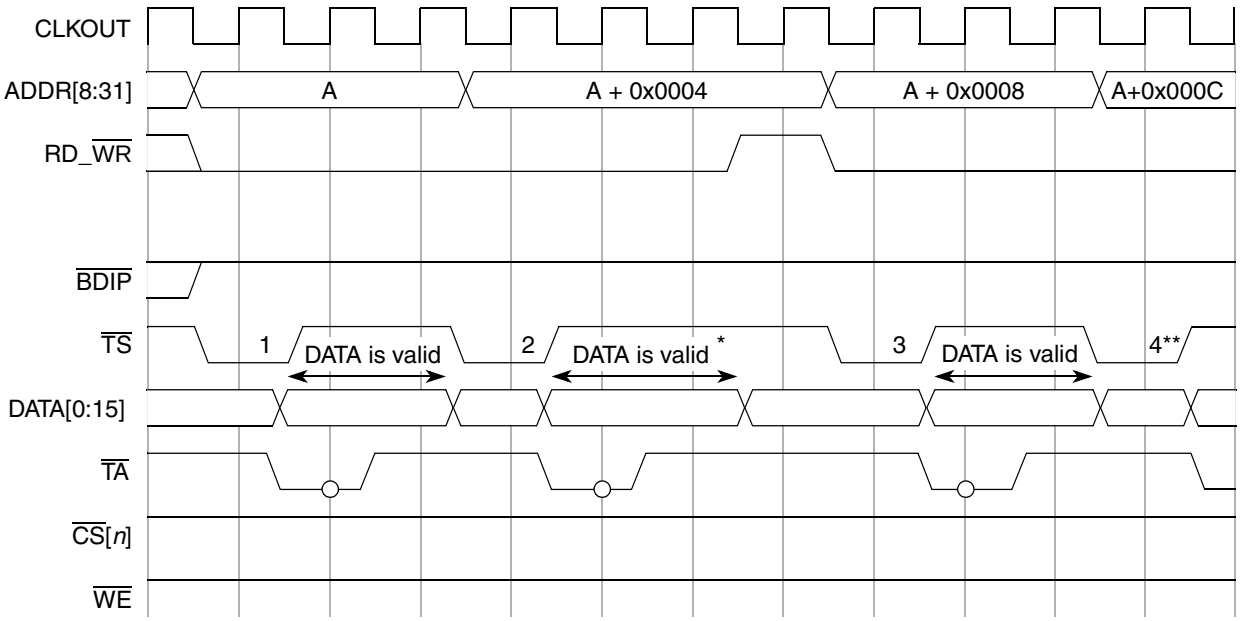


Figure 12-23. Single-beat 32-bit Write Cycle, 16-bit Port Size, Basic Timing

12.4.2.6.2 Small Access Example #2: 32-byte Write with External TA

Figure 12-24 shows an example of a 32-byte write to a non-chip select device, such as an external master, using external TA, requiring eight 32-bit external transactions. Due to the use of external TA, RD_WR does not toggle between the accesses unless that access is the end of a 64-bit boundary. In this case, an extra cycle is required between TA and the next TS to get the next 64-bits of write data internally and RD_WR negates during this extra cycle.



* This extra cycle is required after accesses 2, 4, and 6 to get the next 64-bits of internal write data.

** Four more external accesses (not shown) are required to complete the internal 32-byte request. The timing of these is the same as accesses 1–4 shown in this diagram.

Figure 12-24. 32-Byte Write Cycle with External $\overline{TA}$, Basic Timing

12.4.2.7 Size, Alignment, and Packaging on Transfers

Table 12-13 shows the allowed sizes that an internal or external master can request from the EBI. The behavior of the EBI for request sizes not shown below is undefined. No error signal is asserted for these erroneous cases.

Table 12-13. Transaction Sizes Supported by EBI

Number of Bytes	
Internal Master	**External Master**
1	1
2	2
4	4
8	
32	

Even though misaligned non-burst transfers from internal masters are supported, the EBI naturally aligns the accesses when it sends them out to the external bus, splitting them into multiple aligned accesses if necessary. Natural alignment for the EBI means:

- Byte access can have any address.
- 16-bit access, address bit 31 must be 0.

- 32-bit access, address bits 30–31 must be 0.
- For burst accesses of any size, address bits 29–31 must be 0.

The EBI never generates a misaligned external access, so a multi-master system with two MCUs can never have a misaligned external access. In the erroneous case that an externally-initiated misaligned access does occur, the EBI errors the access and does not initiate the access on the internal bus.

The EBI requires that the portion of the data bus used for a transfer to/from a particular port size be fixed. A 16-bit port must reside on bits 0–15.

In the following figures and tables the following convention is adopted:

- The most significant byte of a 32-bit operand is OP0, and OP3 is the least significant byte.
- The two bytes of a 16-bit operand are OP0 (most significant) and OP1, or OP2 (most significant) and OP3, depending on the address of the access.
- The single byte of a byte-length operand is OP0, OP1, OP2, or OP3, depending on the address of the access.

The convention can be seen in Figure 12-25.

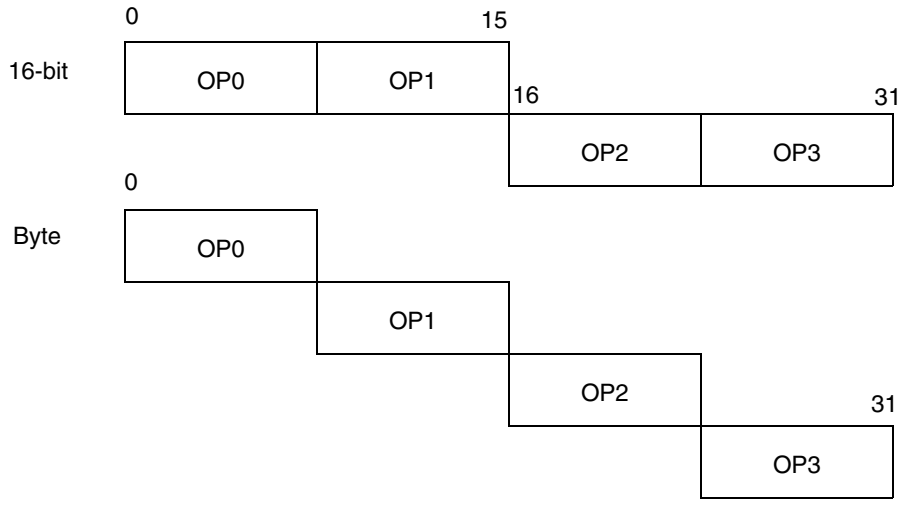


Figure 12-25. Internal Operand Representation

Table 12-14 lists the bytes required on the data bus for read cycles. The bytes indicated as '—' are not required during that read cycle.

Table 12-14. Data Bus Requirements for Read Cycles

Transfer Size	SIZE [0:1]	Address		16-Bit Port Size	
		A[30]	A[31]	D[0:7]	D[8:15]
8 bits	01	0	0	OP0	—
	01	0	1	—	OP1
	01	1	0	OP2	—
	01	1	1	—	OP3
16 bits	10	0	0	OP0	OP1
	10	1	0	OP2	OP3
32 bits	00	0	0	OP0 and OP2 [1]	OP1 and OP3

1 This case consists of two 16-bit external transactions, the first fetching OP0 and OP1, the second fetching OP2 and OP3.

Table 12-15 lists the patterns of the data transfer for write cycles when accesses are initiated by the MCU. The bytes indicated as '—' are not driven during that write cycle.

Table 12-15. Data Bus Contents for Write Cycles

Transfer Size	SIZE[0:1]	Address		16-Bit Port Size[1]	
		A[30]	A[31]	D[0:7]	D[8:15]
8 bits	01	0	0	OP0	—
	01	0	1	—	OP1
	01	1	0	OP2	—
	01	1	1	—	OP3
16 bits	10	0	0	OP0	OP1
	10	1	0	OP2	OP3
32 bits	00	0	0	OP0 and OP2 [2]	OP1 and OP3

1 DBM = 1 for 16-bit data bus mode.

2 This case consists of two 16-bit external transactions, the first writing OP0 and OP1, the second writing OP2 and OP3.

12.4.2.8 Arbitration

This device does not have arbitration pins, so multi-master operation with arbitration is not supported. However, limited dual-MCU functionality is supported for the case of a Master and Slave configuration.

See Section 12.5.5, "Dual-MCU Operations."

12.4.2.9 Termination Signals Protocol

The termination signals protocol was defined to avoid electrical contention on lines that can be driven by various sources. To do that, a slave must not drive signals associated with the data transfer until the address phase is completed and it recognizes the address as its own. The slave must disconnect from signals immediately after it acknowledges the cycle and not later than the termination of the next address phase cycle.

For EBI-mastered non-chip select accesses, the EBI requires assertion of $\overline{TA}$ from an external device to signal that the bus cycle is complete. The EBI uses a latched version of $\overline{TA}$ (1 cycle delayed) for these accesses to help make timing at high frequencies. This results in the EBI driving the address and control signals 1 cycle longer than required, as seen in Figure 12-33. However, the DATA does not need to be held 1 cycle longer by the slave, because the EBI latches DATA every cycle during non-chip select accesses. During these accesses, the EBI does not drive the $\overline{TA}$ signal, leaving it up to an external device (or weak internal pull-up) to drive $\overline{TA}$.

For EBI-mastered chip select accesses, the EBI drives $\overline{TA}$ the entire cycle, asserting according to internal wait state counters to terminate the cycle. During idle periods on the external bus, the EBI drives $\overline{TA}$ negated as long as it is granted the bus; when it no longer owns the bus it lets go of $\overline{TA}$. When an external master does a transaction to internal address space, the EBI only drives $\overline{TA}$ for the cycle it asserts $\overline{TA}$ to return data and for 1 cycle afterwards to ensure fast negation.

Table 12-16 summarizes how the EBI recognizes the termination signals provided from an external device.

Table 12-16. Termination Signals Protocol

$\overline{TA}$[1]	Action
Negated	No termination
X	Transfer error termination
Asserted	Normal transfer termination

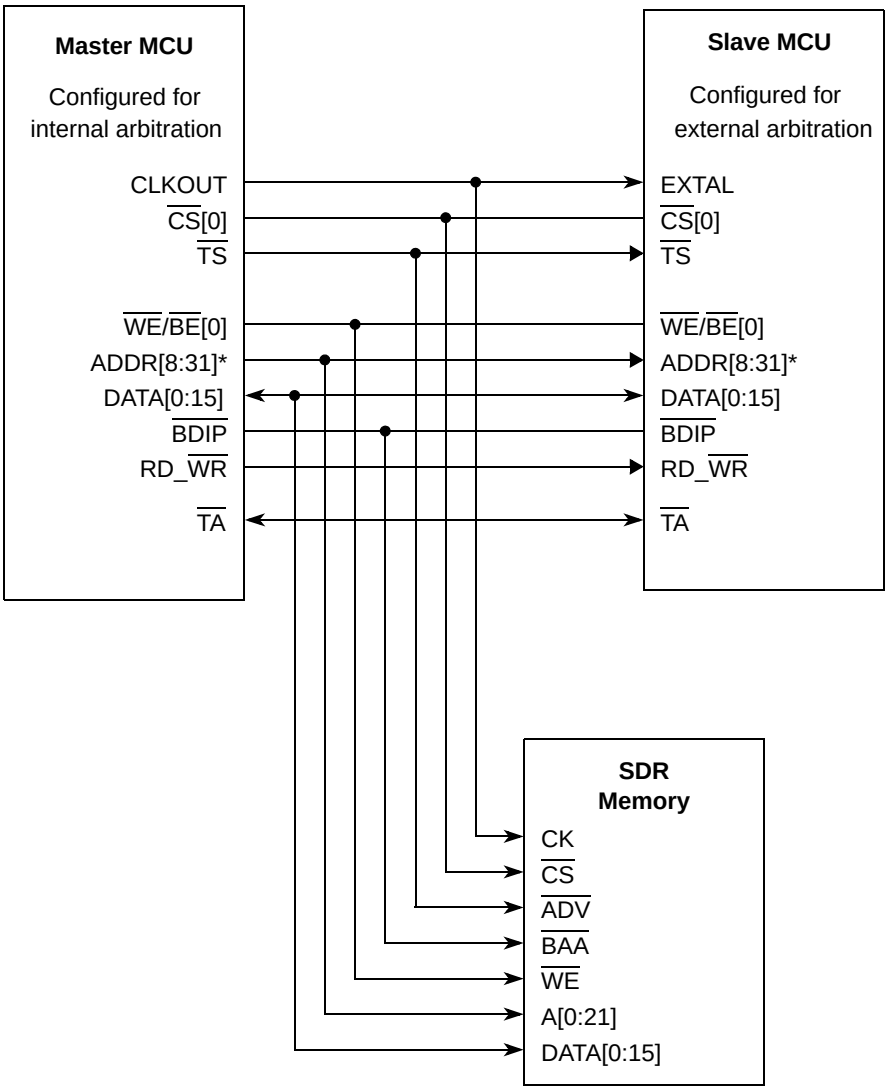
1 Latched version (1 cycle delayed) used for externally driven $\overline{TA}$.

Figure 12-34 shows an example of the termination signals protocol for back-to-back reads to two different slave devices that correctly take turns driving the termination signals. This assumes a system using slave devices that drive termination signals.

12.4.2.10 Bus Operation in External Master Mode

External master mode enables an external master to access the internal address space of the MCU. Figure 12-26 shows how to connect an MCU to an external master (a second MCU) and a shared SDR memory to operate in external master mode. Limited support for external master accesses (master/slave systems only) is available in this device.

See Section 12.5.5, “Dual-MCU Operations.”



* Only ADDR[8:29] are connected to the 32-bit SDR memory.

Figure 12-26. MCU Connected to External Master and SDR Memory

When the external master requires external bus accesses, it takes ownership on the external bus, and the direction of most of the bus signals is inverted, relative to its direction when the MCU owns the bus.

Most of the bidirectional signals shown in Figure 12-26 are only driven by the EBI when the EBI owns the external bus. The only exception is the TA signal and the DATA bus, which are driven by the EBI for external master reads to internal address space. As long as the external master device follows the same protocol for driving signals as this EBI, there is no need to use the open drain mode of the pads configuration module for any EBI pins.

See Section 12.4.2.9, “Termination Signals Protocol” for more information.

The Power Architecture storage reservation protocol is not supported by the EBI. Coherency between multiple masters must be maintained via software techniques, such as event passing.

The EBI does not provide memory controller services to an external master that accesses shared external memories. Each master must correctly configure its own memory controller and drive its own chip selects when sharing a memory between two masters.

The EBI does not support burst accesses from an external master; only single accesses of 8-, 16-, or 32-bits can be performed.[1]

12.4.2.10.1 Address Decoding for External Master Accesses

The EBI allows external masters to access internal address space when the EBI is configured for external master mode. The external address is compared for any external master access, to determine if EBI operation is required. Because only 24 address bits are available on the external bus, special decoding logic is required to allow an external master to access on-chip locations whose upper eight address bits are non-zero. This is done by using the upper four external address bits (ADDR[8:11]) as a code to determine whether the access is on-chip and if so, for which internal slave it is targeted.

NOTE

External master accesses are not supported to the calibration bus.

The options for the address compare sequence are explained in the following bullets:

- External master access to another device — If ADDR[8] = 0, then the access is assumed to be to another device and is ignored by the EBI.
- External master access to valid internal slave — If ADDR[8] = 1, then ADDR[9:11] are checked versus a list of 3-bit codes to determine which internal slave to forward the access to. The upper 8 internal address bits are set appropriately by the EBI according to this 3-bit code, and internal address bits [8:11] are set appropriately to match the internal slave selected.
- External master access to invalid internal slave — If the 3-bit code does not match a valid internal slave, then the EBI responds with a bus error.

1. Except for the special case of a 32-bit non-chip select access in 16-bit data bus mode. See Section 12.4.2.11, “Non-Chip-Select Burst in 16-bit Data Bus Mode”.

Table 12-17 shows the possible 3-bit codes that are associated with various slaves in the MCU, as well as the resulting upper 12 address bits required to appropriately match up with the memory map of each internal slave.

Table 12-17. EBI Internal Slave Address Decoding

Internal Slave	External ADDR[8:11][1]	Internal ADDR[0:7][2]	Internal ADDR[8:11][3]
(off-chip)	0b0xxx	—	—
Internal flash	0b10xx	0b0000_0000	0b00, ADDR[10:11]
Internal SRAM	0b1100	0b0100_0000	0b0000
Reserved	0b1101	0b0110_0000	0b0000
Bridge A peripherals	0b1110	0b1100_0011	0b1111
Bridge B peripherals	0b1111	0b1111_1111	0b1111

1 Value on upper 4 bits of 24-bit external address bus ADDR[8:31]. ADDR[8] determines whether the access is on or off chip.

2 Value on upper 8 bits of 32-bit internal address bus.

3 Value on bits 8:11 of 32-bit internal address bus.

12.4.2.10.2 Bus Transfers Initiated by an External Master

The external master gets ownership of the bus and asserts $\overline{TS}$ to initiate an external master access. The access is directed to the internal bus only if the input address matches to the internal address space. The access is terminated with $\overline{TA}$. If the access was successfully completed, the MCU asserts $\overline{TA}$, and the external master can proceed with another external master access, or relinquish the bus.

Figure 12-27 and Figure 12-28 illustrate the basic flow of read and write external master accesses.

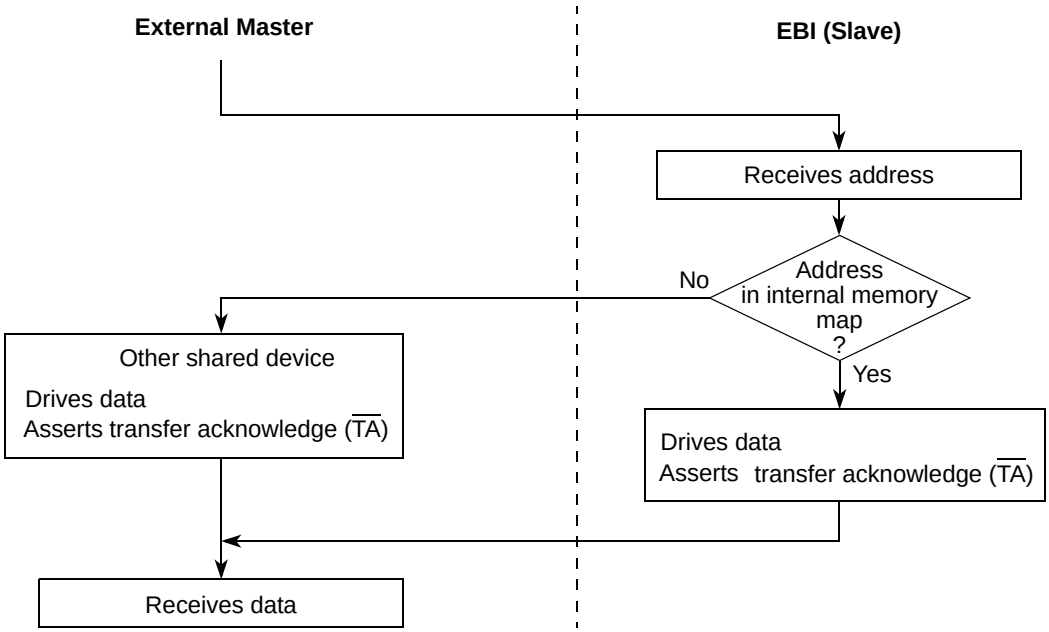


Figure 12-27. Basic Flow Diagram of an External Master Read Access (EARB = 1)

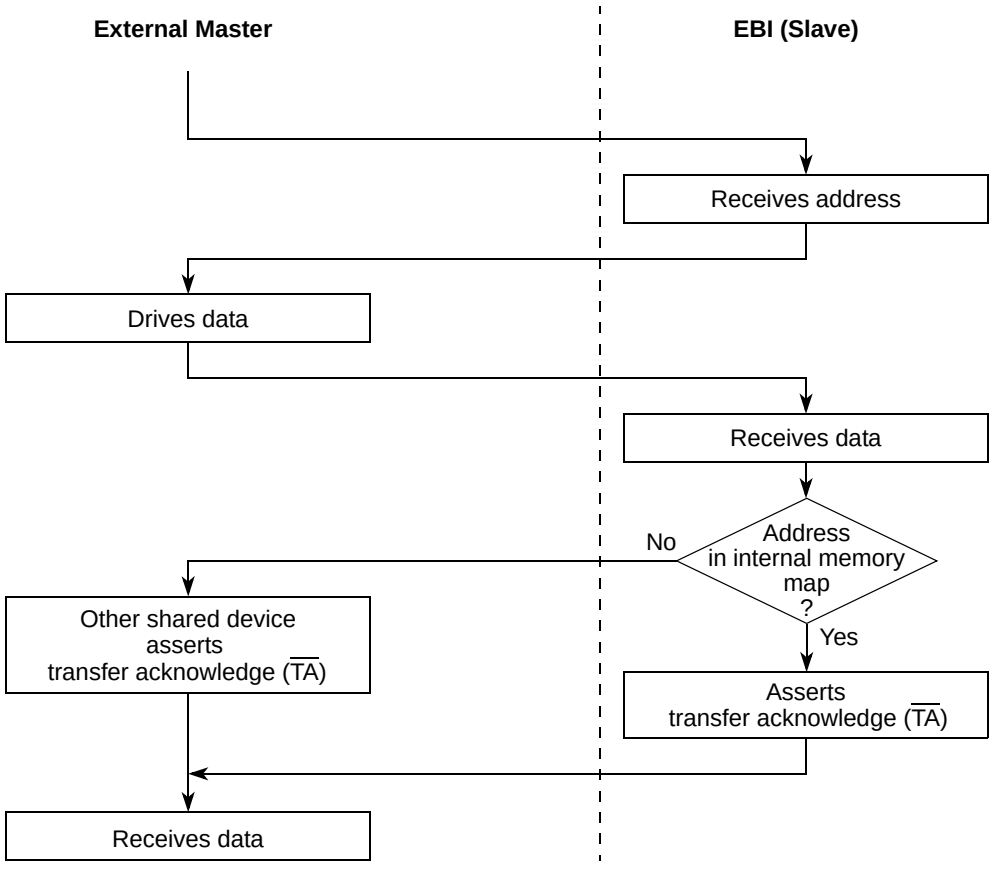


Figure 12-28. Basic Flow Diagram of an External Master Write Cycle (EARB = 1)

Figure 12-29 and Figure 12-30 describe read and write cycles from an external master accessing internal space in the MCU. The minimal latency for an external master access is three clock cycles. The actual latency of an external to internal cycle varies depending on which internal module is being accessed and how much internal bus traffic is going on at the time of the access.

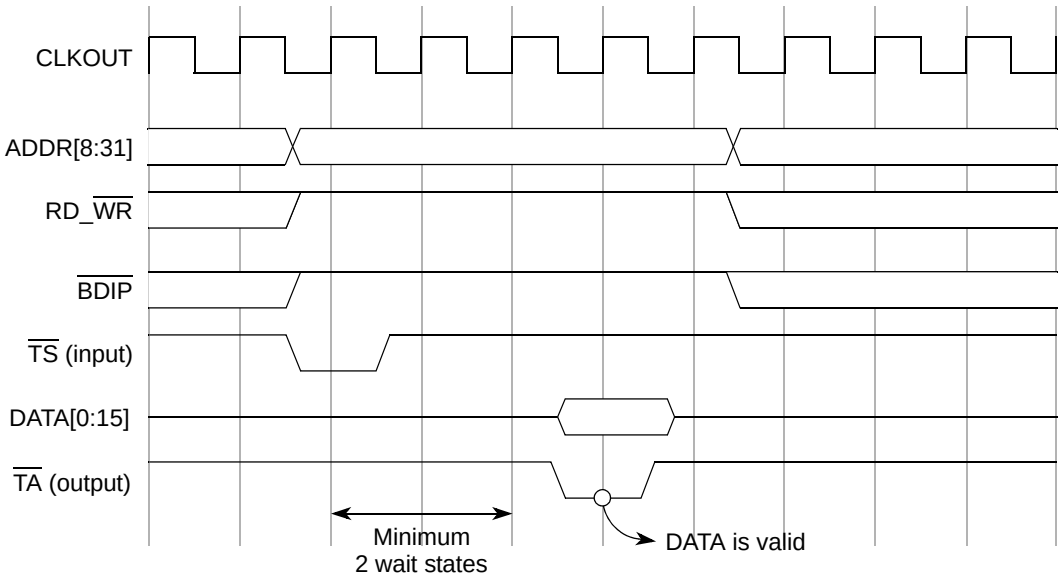
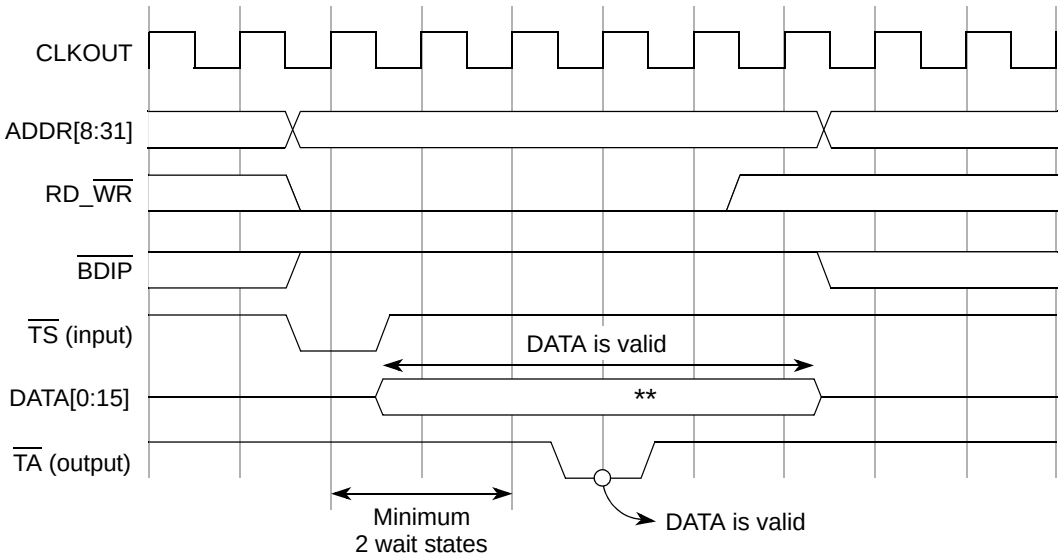


Figure 12-29. External Master Read from MCU



** If the external master is another MCU with this EBI, then DATA remains valid as shown due to use of latched TA internally. These extra data valid cycles (past TA) are not required by the slave EBI.

Figure 12-30. Basic Flow Diagram of an EBI Read Access in External Master Mode (EARB = 0)

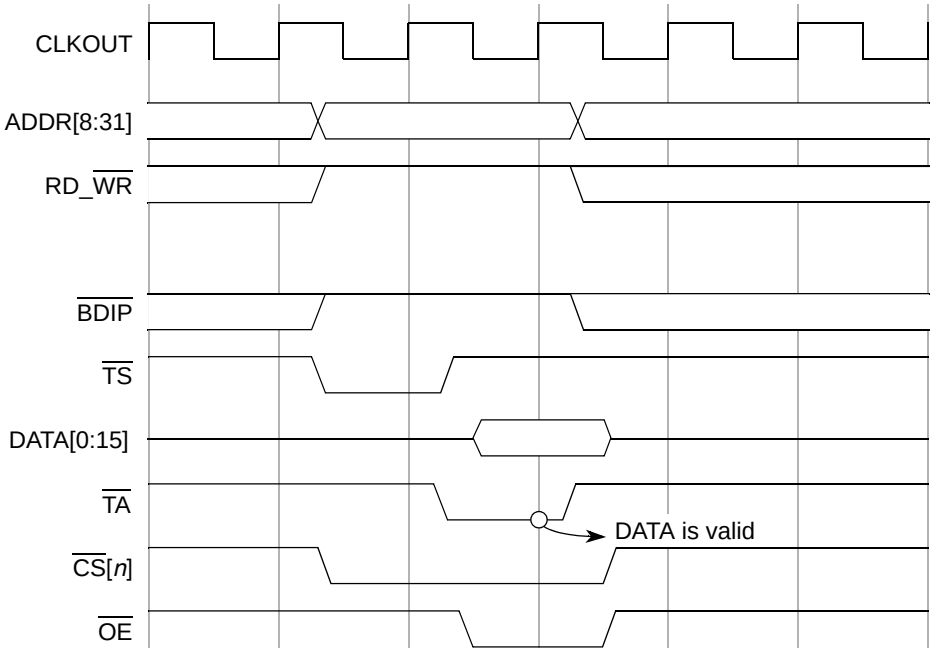


Figure 12-31. Single-Beat CS Read Cycle in External Master Mode, Zero Wait States

12.4.2.11 Non-Chip-Select Burst in 16-bit Data Bus Mode

The timing diagrams in this section apply only to the special case of a non-chip select 32-bit access in 16-bit data bus mode. They specify the behavior for both the EBI-master and EBI-slave, as the external master is expected to be another MCU with this EBI.

For this case, a special two-beat burst protocol is used for reads and writes, so that the EBI-slave can internally generate one 32-bit read or write access (thus 32-bit coherent), as opposed to two separate 16-bit accesses.

Figure 12-32 shows a 32-bit read from an external master in 16-bit data bus mode.

Figure 12-33 shows a 32-bit write from an external master in 16-bit data bus mode.

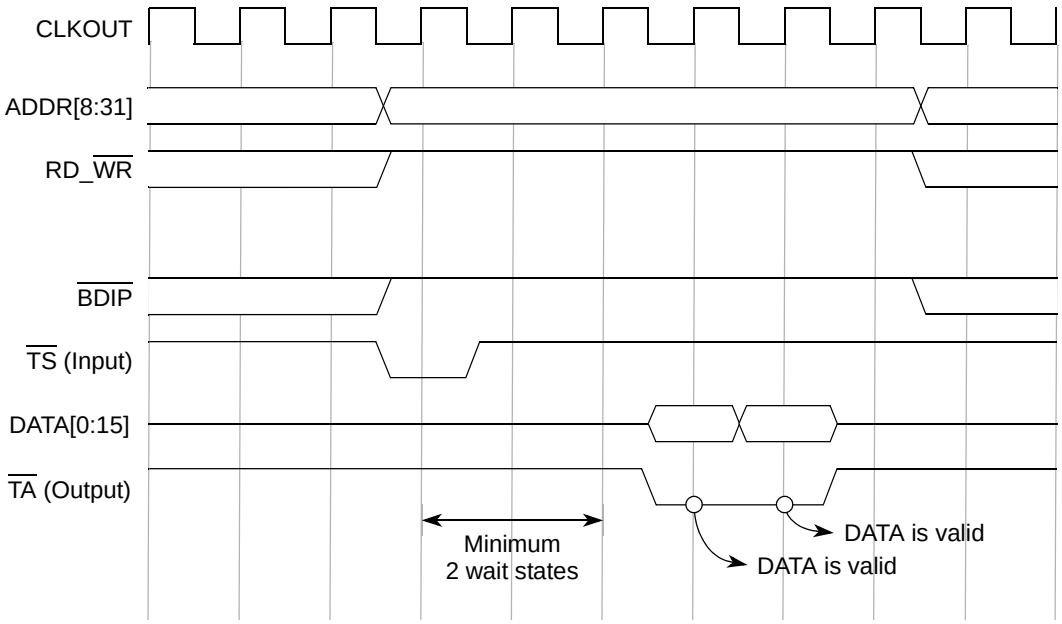


Figure 12-32. External Master 32-bit Read from MCU with DBM = 1

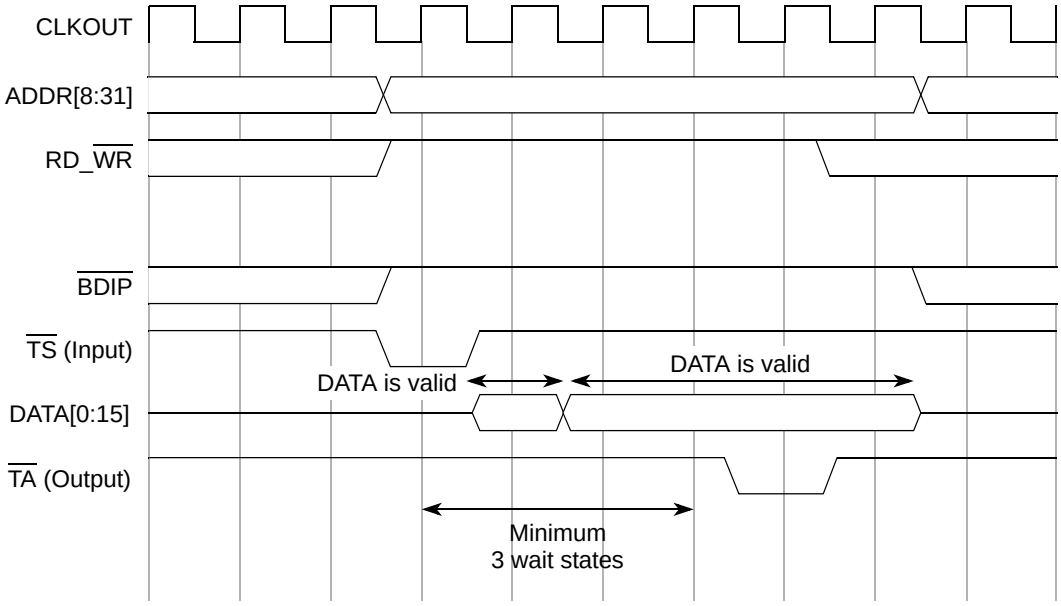


Figure 12-33. External Master 32-bit Write to MCU with DBM=1

12.4.2.12 Calibration Bus Operation

The EBI has a second external bus, intended for calibration use. This bus consists of a second set of the same signals present on the primary external bus, except that some signals are excluded. Both busses are supported by the EBI by using the calibration chip selects to steer accesses to the calibration bus instead of to the primary external bus.

Because the calibration bus has no arbitration signals, the arbitration on the primary bus controls accesses on the calibration bus as well, and no external master accesses can be performed on the calibration bus.

Accesses cannot be performed in parallel on both external busses. However, back-to-back accesses can switch from one bus to the other, as determined by the type of chip select each address matches.

See Appendix B, “Calibration” for more information on how to use the calibration bus.

The timing diagrams and protocol for the calibration bus are identical to those for the primary bus, except that some signals are not available on the calibration bus.

There is an inherent dead cycle between a calibration chip select access and a non-calibration access (chip select or non-chip select), just like the one between accesses to two different non-calibration chip selects (described in Section 12.4.2.4.3, “Back-to-Back Accesses”).

Figure 12-34 shows an example of a non-calibration chip select read access followed by a calibration chip select read access. This figure is identical to Figure 12-14, except the $\overline{CS}$[y] is replaced by CAL_$\overline{CS}$[y]. Timing for other cases on the calibration bus can similarly be derived from other figures in this document (by replacing $\overline{CS}$ with CAL_$\overline{CS}$).

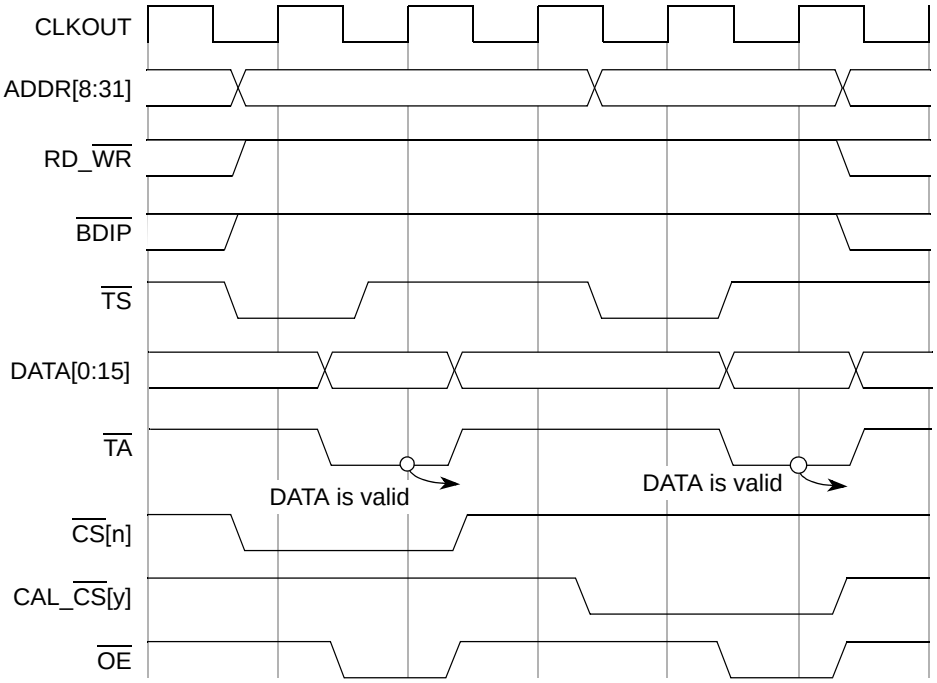


Figure 12-34. Back-to-Back 32-bit Reads to $\overline{CS}$, CAL_$\overline{CS}$ Banks

12.5 Initialization and Application Information

NOTE

The 208 package does not have an external bus interface. This chapter pertains to the 324 package and 496 assembly only.

12.5.1 Booting from External Memory

The EBI block does not support booting directly from external memory (fetching the first instruction after an external RESET). The MCU uses an internal boot assist module (BAM), which executes after each reset and configures the EBI block, allowing for external boot if desired.

See Chapter 15, "Boot Assist Module (BAM)," for detail information about the boot modes supported by the MCU.

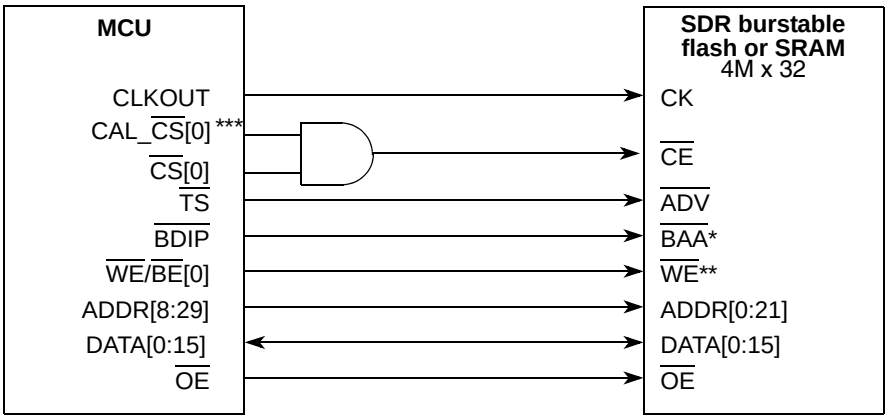
You cannot boot from external memory on the calibration bus.

Do not modify the EBI registers during external accesses. If external memory must write to the EBI registers, avoid modifying EBI registers:

- Copy the code that writes to the EBI registers (plus the return branches) to internal SRAM
- Branch to internal SRAM to run the code, ending with a branch back to external memory

12.5.2 Running with SDR (Single Data Rate) Burst Memories

This includes flash and external SRAM memories with a compatible burst interface. $\overline{\text{BDIP}}$ is required only for some SDR memories. Figure 12-32 shows a block diagram of an MCU connected to a 32-bit SDR burst memory.



* Connection depending on the type of memory.
** Flash memories typically use one $\overline{\text{WE}}$ signal as shown, RAMs use 2 (16-bit).
*** Not available on all devices, see the Signals chapter.

Figure 12-35. MCU Connected to SDR Burst Memory

See Figure 12-19 for an example of the timing of a typical burst read operation to an SDR burst memory. See Figure 12-20 for an example of the timing of a typical single write operation to SDR memory.

12.5.3 Using Asynchronous Memory

The EBI supports asynchronous memory, even though the EBI does not have an asynchronous mode. Asynchronous memories do not support bursting, and do not require the CLKOUT, $\overline{\text{TS}}$, and $\overline{\text{BDIP}}$ signals. The EBI drives the output, and latches all signals at the positive edge of CLKOUT. The data timing is

controlled by setting the SCY bits in the option register to a valid number of wait states for the access time to asynchronous memory.

12.5.3.1 Example Wait State Calculation

This example applies to any chip select memory, synchronous or asynchronous.

As an example, say we have a memory with 50 ns access time, and we are running the external bus at 66 MHz (CLKOUT period: 15.2 ns).

When the input data specification for the MCU is 4 ns:

Number of wait states = (access time) ÷ (CLKOUT period) + (0 or 1) (depending on setup time)

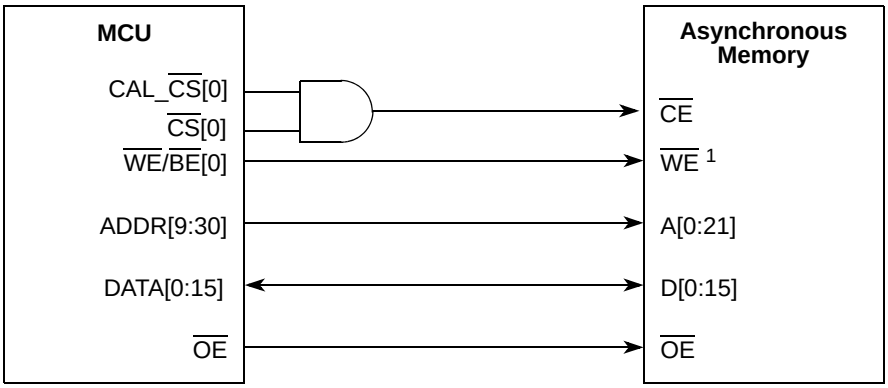
50 ÷ 15.2 = 3 with 4.4 ns remaining (minimum of three wait states, then check setup time)

15.2 - 4.4 = 10.8 ns (this is the achieved input data setup time)

Because actual input setup (10.8 ns) is greater than the input setup specification (4.0 ns), three wait states is sufficient. If the input setup is less than 4.0 ns, use four wait states.

12.5.3.2 Timing and Connections for Asynchronous Memories

The connections to an asynchronous memory are the same as for a synchronous memory, except that the CLKOUT, $\overline{TS}$, and $\overline{BDIP}$ signals are not used. Figure 12-36 shows a block diagram of an MCU connected to an asynchronous memory.



[1] Flash memories typically use $\overline{WE}$[0] signal as shown, RAMs use two ($\overline{WE}$/$\overline{BE}$[0:1]).

Figure 12-36. MCU Connected to Asynchronous Memory

Figure 12-37 shows a timing diagram of a read operation to a 16-bit asynchronous memory using three wait states. Figure 12-38 shows a timing diagram of a write operation to a 16-bit asynchronous memory using three wait states.

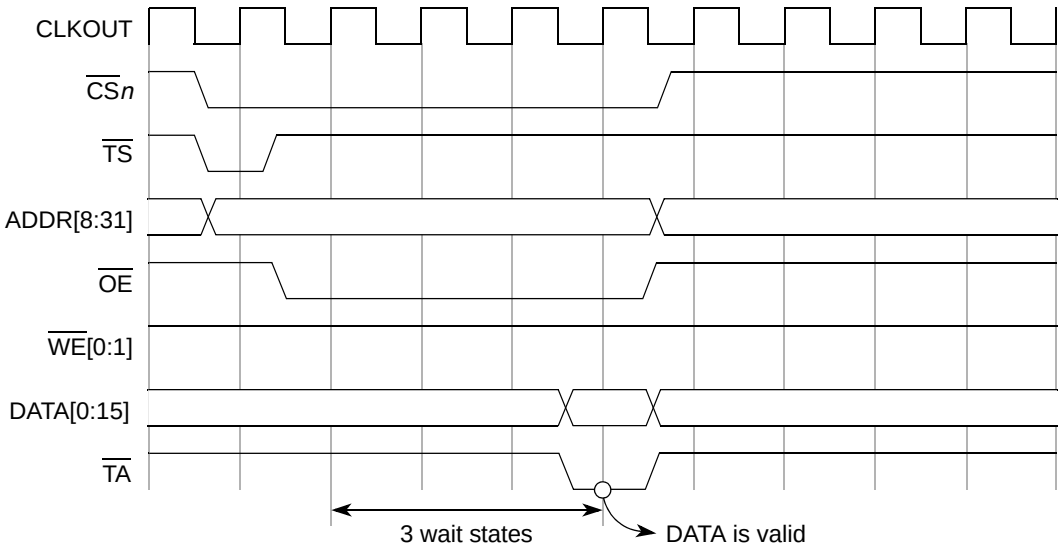


Figure 12-37. Read Operation to Asynchronous Memory, Three Initial Wait States

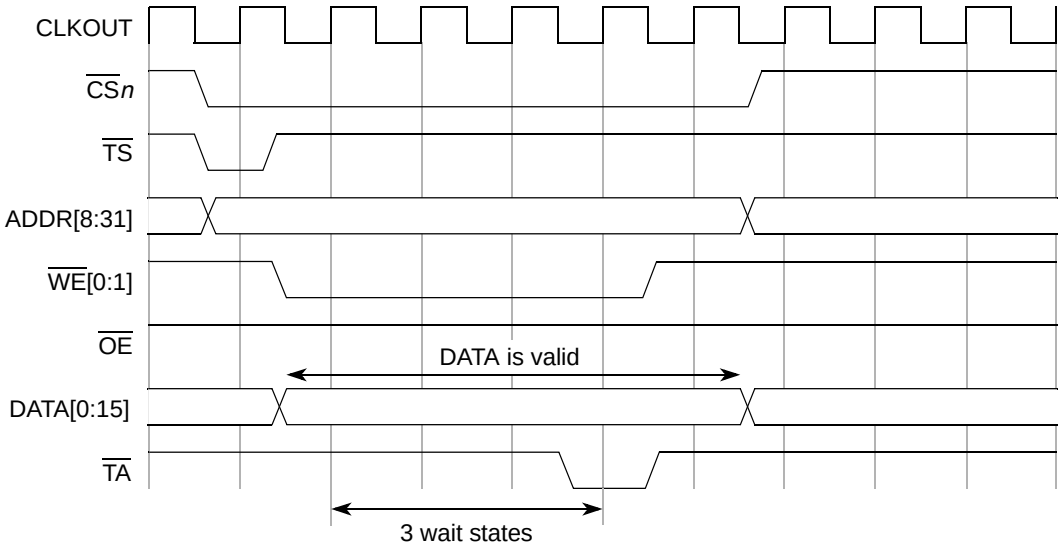
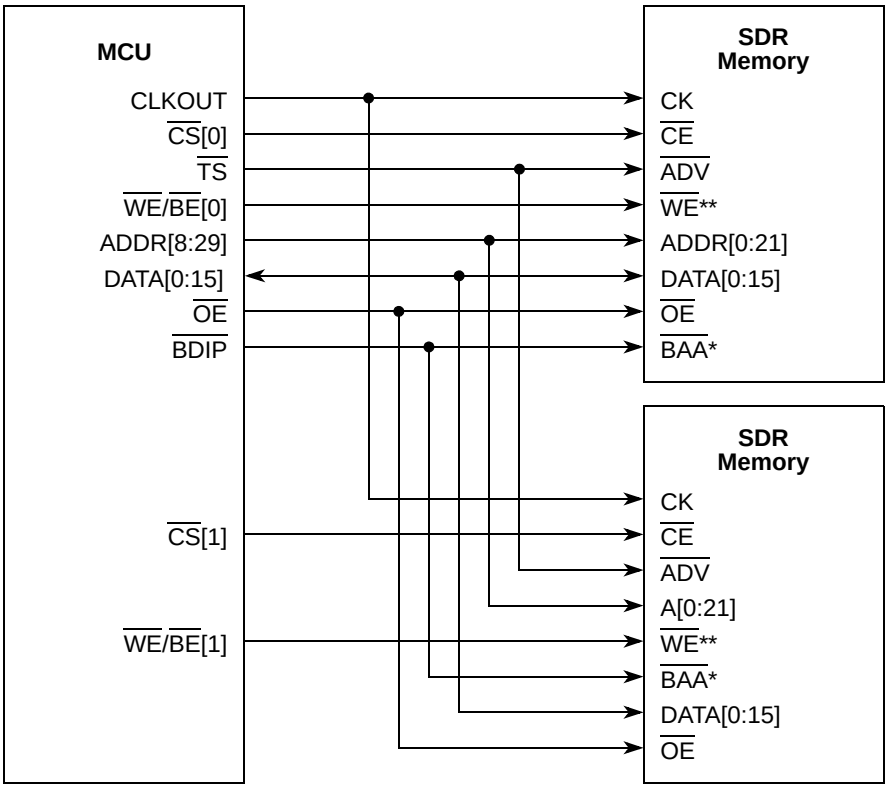


Figure 12-38. Write Operation to Asynchronous Memory, Three Initial Wait States

12.5.4 Connecting an MCU to Multiple Memories

The MCU can be connected to more than one memory at a time.

Figure 12-39 shows an example of how two memories could be connected to one MCU.



* The connection depends on the memory used.
** Flash memories typically use WE[0] signal as shown, RAMs use two (WE/BE[0:1]).

Figure 12-39. MCU Connected to Multiple Memories

12.5.5 Dual-MCU Operations

This section describes how to configure dual-MCU systems when a signal or pin is not available. More than one section can apply if the signals or pins are not present on one or both MCUs.

12.5.5.1 Connecting 16-bit MCU to 32-bit MCU (Master and Slave)

Connect DATA[0:15] between both MCUs, and configure both for 16-bit data bus mode operation (DBM=1 in EBI_MCR). Does not support 32-bit external memories.

12.5.5.2 Arbiting a Master and Slave configuration

A dual master system is not supported, because the two masters have no method to arbitrate access to the external bus without conflicts. However, you can configure a master/slave system for arbitration.

To implement a master/slave system, you must configure the master MCU for internal arbitration (EARB=0 in EBI_MCR) and the slave MCU for external arbitration (EARB=1). The slave MCU never attempts to start an access on the external bus. The master MCU maintains control of the bus without arbitration delays. If the slave MCU executes internal code to access an external address, the access never completes and eventually times-out in the slave MCU.

12.5.5.3 Setting the transfer size

To set the block size the Master uses to access the slave device, set the SIZEN bit in the internal SIZE field of the EBI_MCR for the slave MCU. To access the slave MCU using a different block size, the Master MCU must first write the new transfer size to the slave MCU's SIZE field before processing subsequent transaction.

12.5.5.4 Acknowledging a transfer

You must configure the chip select and external memory to access valid chip select regions only. This ensures the EBI latches the data to the correct cycle count for the valid chip region.

The EBI does not have built-in protection to prevent external accesses to invalid chip and memory regions. Without logic to identify the valid chip region, the EBI cannot latch the data to the correct cycle.

12.5.5.5 Detecting a transfer error

If an access times out in the EBI bus monitor, the EBI (master) terminates the access early, but the MCU does not detect the access termination. Therefore, the slave device can drive the data much later, colliding with a subsequent access that is already underway. Therefore, disable the EBI bus monitor.

12.5.5.6 Detecting Burst Data in Progress

If an MCU does not have a $\overline{\text{BDIP}}$ signal, burst support is available if the memory does not require $\overline{\text{BDIP}}$ to support data burst. Many external memories use a self-timed configurable burst mechanism that does not require a dynamic burst indicator. Check the applicable external memory specification to see if $\overline{\text{BDIP}}$ is required.

12.5.6 Summary of Differences from MPC500

The following summary lists of the significant differences between this EBI used in the MPC5500 devices and that of the MPC500 devices:

- SETA feature not supported: chip select devices cannot use the external $\overline{\text{TA}}$ signal, instead must use wait state configuration.
- No memory controller support for external masters: no support for multi-master system to drive its own chip selects
- Changes in bit fields:
  - Removed these variable timing attributes from the option register: CSNT, ACS, TRLX, EHTR
  - Removed LBDIP base register bit, now late $\overline{\text{BDIP}}$ assertion is the default

  - The BL field of the base register has inverted logic from the MPC56x devices (0 = 8-beat burst on the MPC5500, 1 = 8-beat burst on the MPC56x)
- Removed reservation support on external bus
- Removed address type (AT), write-protect (WP), and dual-mapping features because these functions can be replicated by memory management unit (MMU) in e200z3 core
- Removed support for 8-bit ports
- Removed boot chip select operation: on-chip boot assist module (BAM) handles boot (and configuration of EBI registers)
- Added support for 32-bit coherent read and write non-chip select accesses in 16-bit data bus mode
- Misaligned accesses are not supported
- Calibration features implemented by three calibration chip selects
- Removed support for 3-master systems

Chapter 13
Flash Memory

13.1 Introduction

This section provides information about the flash bus interface unit (FBIU) and the flash memory block.

13.1.1 Block Diagram

Figure 13-1 shows a block diagram of the flash memory module. The FBIU is addressed through the system bus while the flash control and status registers are addressed through the slave (peripheral) bus.

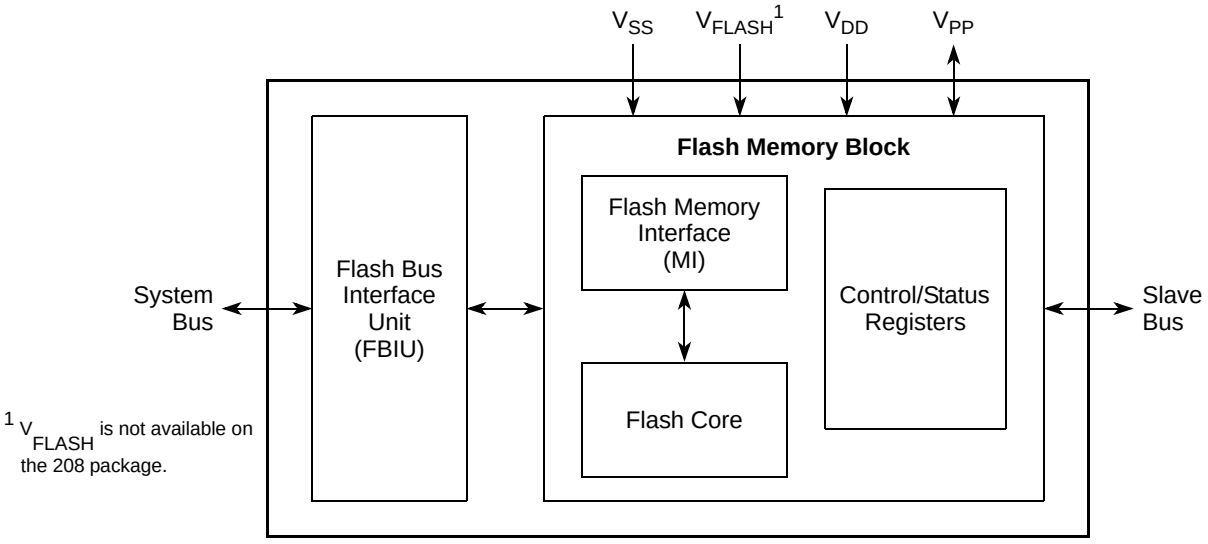


Figure 13-1. Flash System Block Diagram

13.1.2 Overview

The flash module serves as electrically programmable and erasable non-volatile memory (NVM) that is ideal for program and data storage for single-chip applications allowing for field reprogramming without requiring external programming voltage sources. The module is a solid-state silicon memory device consisting of blocks of single-transistor storage elements.

The device flash contains a flash bus interface unit (FBIU) and a flash memory array. The flash BIU interfaces the system bus to a dedicated flash memory array controller. The FBIU supports a 64-bit data bus width at the system bus port, and a 128-bit read data interface from the flash memory array. If enabled, the flash BIU contains a four-entry prefetch buffer, each entry containing 128 bits of data, and an associated controller that prefetches sequential lines of data from the flash array into the buffer. Prefetch

buffer hits support zero-wait responses. Normal flash array accesses (accesses that don't go to the prefetch buffers) are registered in the FBIU in a single cycle, and are forwarded to the system bus on the next cycle, incurring at least two wait states (depending on the frequency). Additional wait states are indicated in FLASH_BUICR[RWSC]. See Table 13-14 for more information.

The flash memory block is arranged as two functional units, the first being the flash core. The flash core is composed of arrayed non-volatile storage elements, sense amplifiers, row selects, column selects, charge pumps, ECC logic and redundancy logic. The arrayed storage elements in the flash core are subdivided into physically separate units referred to as blocks.

The second functional unit of flash memory is the memory interface (MI). The MI contains the registers and logic that control the operation of the flash core. The MI is also the interface between the flash module and the FBIU. The FBIU connects the MCU system bus to the flash module, and provides all system level customization and configuration functionality.

The flash array has three address spaces. Low address space (LAS) is 256-KB in size. Mid address space (MAS) is also 256-KB in size. High address space (HAS) is 512 KB in size. Total address space is 1.0 MB.

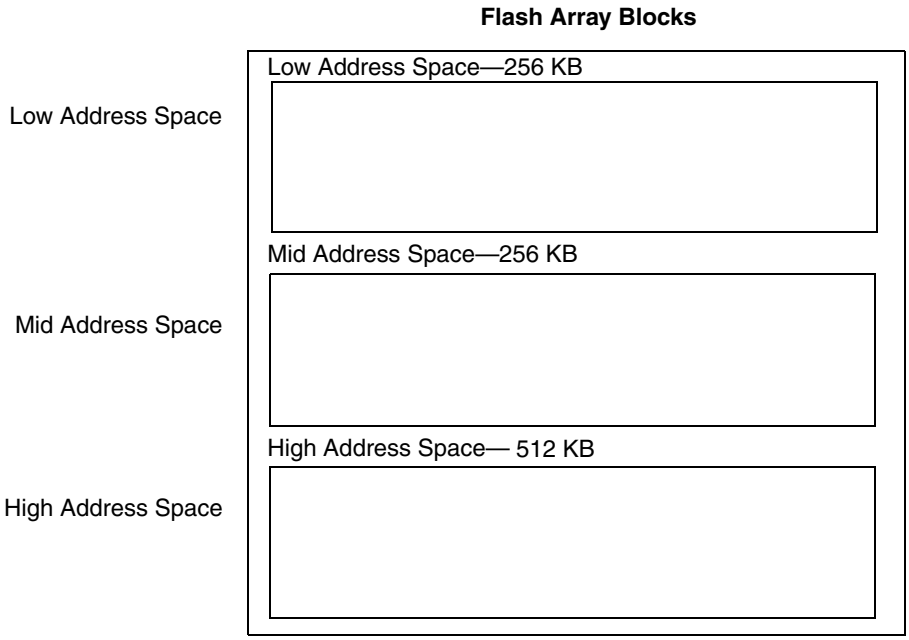


Figure 13-2. Flash Array Diagram

13.1.3 Features

The following list summarizes the key features of the FBIU:

- The FBIU system bus interface supports a 64-bit data bus. All byte, halfword, word, and doubleword reads are supported. Only aligned word and doubleword writes are supported.
- The flash array interface supports a 128-bit read data bus and a 64-bit write data bus.
- The FBIU provides configurable read buffering and line prefetch support. Four line read buffers (each 128 bits wide) and a prefetch controller are used to support single-cycle read responses (zero wait-states) for hits in the buffers.
- The FBIU provides hardware and software configurable read and write access protections on a per-master basis.
- The FBIU interface to the flash array is pipelined with a depth of 1.
- The FBIU allows configurable access timing.
- The FBIU provides multiple-mapping support and mapping-based block access timing allowing use for emulation of other memory types.

The flash memory array has the following features:

- Software programmable block program/erase restriction control for low, mid, and high address spaces.
- Erase of selected blocks.
- ECC with single-bit correction, double-bit detection.
- Page program size of 128 bits allows programming from one to two consecutive 64-bit doublewords in a page.
- Embedded hardware program and erase algorithm.
- Read while write with multiple partitions.
- Stop mode for low power stand-by.
- Erase suspend, program suspend, and erase-suspended program.
- Automotive flash that meets automotive endurance and reliability requirements. Shadow information is stored in a non-volatile shadow block.
- Independent program/erase of the shadow block.

13.1.4 Modes of Operation

13.1.4.1 User Mode

User mode is the default operating mode of the flash memory block. In this mode, you can read, write, program, and erase the flash. See Section 13.4.2, “Flash Memory Array: User Mode.”

13.1.4.2 Stop Mode

In stop mode (FLASH_MCR[STOP] = 1), all DC current sources in the flash are disabled. See Section 13.4.3, “Flash Memory Array: Stop Mode.”

13.2 External Signal Description

Table 13-1 shows a list of signals required for flash.

Table 13-1. Signal Properties

Name	Function	Reset State
V_{FLASH}	Flash read power supply	—
V_{PP}	Flash program/erase power supply	—

13.2.1 Voltage for Flash Only V_{FLASH}

V_{FLASH} is a supply required for reads of the flash core. This voltage is specified as 3.3 V with a tolerance of ± 0.3 V.

208 Package: The V_{FLASH} pin is not available on the 208 package.

13.2.2 Program and Erase Voltage for Flash Only V_{PP}

V_{PP} is a supply required for program and erase of the flash core. This voltage is specified as 5 V with a tolerance of -0.5 V to +0.25 V during program and erase operations. V_{PP} is required at all times, even during normal reads of flash memory. During read operations, V_{PP} can be as high as 5.3 V and as low as 3.0 V.

13.3 Memory Map/Register Description

The flash BIU occupies a 512-MB portion of the address space. The actual flash array is multiply-mapped within this space.

The MCU internal flash has a feature that allows the internal flash timing to be modified to emulate an external memory, hence the name, external emulation mode. The upper five address lines are used to provide additional timing control that allows the FBIU response timing on the system bus (which must be controlled to provide for timing emulation of alternate memory types). See Figure 13-3.

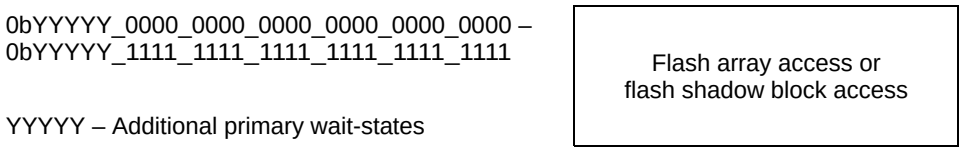


Figure 13-3. Flash BIU Address Scheme

This feature allows calibration parameters to be tested using an external memory; and then in production, the internal flash access timing is modified to match timing of the external memory. The access time of the internal flash is lengthened based on the address range being accessed. To access an area with a slower access time, the address is modified per Table 13-2.

Table 13-2. Internal Flash External Emulation Mode

Address Range		YYYYY	Wait States
0x0000_0000	0x001F_FFFF	00000	0
0x0100_0000	0x011F_FFFF	01000	8
0x0200_0000	0x021F_FFFF	10000	16
0x0300_0000	0x031F_FFFF	11000	24
0x0400_0000	0x041F_FFFF	00001	1
0x0500_0000	0x051F_FFFF	01001	9
0x0600_0000	0x061F_FFFF	10001	17
0x0700_0000	0x071F_FFFF	11001	25
0x0800_0000	0x081F_FFFF	00010	2
0x0900_0000	0x091F_FFFF	01010	10
0x0A00_0000	0x0A1F_FFFF	10010	18
0x0B00_0000	0x0B1F_FFFF	11010	26
0x0C00_0000	0x0C1F_FFFF	00011	3
0x0D00_0000	0x0D1F_FFFF	01011	11
0x0E00_0000	0x0E1F_FFFF	10011	19
0x0F00_0000	0x0F1F_FFFF	11011	27
0x1000_0000	0x101F_FFFF	00100	4
0x1100_0000	0x111F_FFFF	01100	12
0x1200_0000	0x121F_FFFF	10100	20
0x1300_0000	0x131F_FFFF	11100	28
0x1400_0000	0x141F_FFFF	00101	5
0x1500_0000	0x151F_FFFF	01101	13
0x1600_0000	0x161F_FFFF	10101	21
0x1700_0000	0x171F_FFFF	11101	29
0x1800_0000	0x181F_FFFF	00110	6
0x1900_0000	0x191F_FFFF	01110	14
0x1A00_0000	0x1A1F_FFFF	10110	22
0x1B00_0000	0x1B1F_FFFF	11110	30
0x1C00_0000	0x1C1F_FFFF	00111	7
0x1D00_0000	0x1D1F_FFFF	01111	15
0x1E00_0000	0x1E1F_FFFF	10111	23
0x1F00_0000	0x1F1F_FFFF	11111	31

13.3.1 Flash Memory Map

Table 13-3 shows the flash array memory map and how it is mapped using byte addressing.

Base addresses for the device are the following:

- Shadow base address = 0x00FF_FC00
- Array base address = 0x0000_0000
- Control registers base address = 0xC3F8_8000

Table 13-3. Module Flash Array Memory Map

Byte Address	Type and Amount of Space Used	Access
Shadow base (0x0000_0000–0x0000_03FF)	Shadow block space (1024 bytes)	User
Array base + (0x0000_0000–0x0003_FFFF)	Low address space (256 KB)	User
Array base + (0x0004_0000–0x0007_FFFF)	Mid address space (256 KB)	User
Array base + (0x0008_0000–0x000F_FFFF)	High address space (512 KB)	User

Table 13-4 shows how the array is partitioned into three address spaces — low, mid, and high — and into partitions and blocks.

Table 13-4. Flash Partitions

Address (Array base + offset)	Use	Block	Bytes	Partition
Array Base + 0x0000_0000	Low address space	Low 0	16 KB	1
Array Base + 0x0000_4000		Low 1	48 KB	
Array Base + 0x0001_0000		Low 2	48 KB	
Array Base + 0x0001_C000		Low 3	16 KB	
Array Base + 0x0002_0000		Low 4	64 KB	2
Array Base + 0x0003_0000		Low 5	64 KB	
Array Base + 0x0004_0000	Mid address space	Med 0	128 KB	3
Array Base + 0x0006_0000		Med 1	128 KB	
Array Base + 0x0008_0000	High address space	High 0	128 KB	4
Array Base + 0x000A_0000		High 1	128 KB	
Array Base + 0x000C_0000		High 2	128 KB	5
Array Base + 0x000E_0000		High 3	128 KB	
Array Base + 0x00FF_FC00	Shadow block space	Shadow	472	All [1]
Array Base + 0x00FF_FDD8	Flash shadow block, serial passcode	Shadow	8	All [1]
Array Base + 0x00FF_FDE0	Flash shadow block, control word	Shadow	4	All [1]
Array Base + 0x00FF_FDE4	General use	Shadow	4	All [1]
Array Base + 0x00FF_FDE8	Flash shadow block, FLASH_LMLR reset configuration	Shadow	4	All [1]
Array base + 0x00FF_FDEC	General use	Shadow	4	All [1]

Table 13-4. Flash Partitions (continued)

Address (Array base + offset)	Use	Block	Bytes	Partition
Array base + 0x00FF_FDF0	Flash shadow block, FLASH_HLR reset configuration	Shadow	4	All [1]
Array base + 0x00FF_FDF4	General use	Shadow	4	All [1]
Array base + 0x00FF_FDF8	Flash shadow block, FLASH_SLMLR reset configuration	Shadow	4	All [1]
Array base + 0x00FF_FE00	Flash shadow block, FLASH_BIUCR2 reset configuration	Shadow	4	All [1]
Array base + 0x00FF_FE04–0x00FF_FFFF	General use	Shadow	508	All [1]

[1] The shadow block does not support RWW. See Section 13.4.2.5, "Flash Shadow Block."

Table 13-5 shows the register set for the flash module.

Table 13-5. Module Register Memory Map

Byte Address	Register Name	Register Description	Bits
Register Base + 0x0000	FLASH_MCR	Module configuration register	32
Register Base + 0x0004	FLASH_LMLR	Low/mid address space block locking register	32
Register Base + 0x0008	FLASH_HLR	High address space block locking register	32
Register Base + 0x000C	FLASH_SLMLR	Secondary low/mid address space block locking register	32
Register Base + 0x0010	FLASH_LMSR	Low/mid address space block select register	32
Register Base + 0x0014	FLASH_HSR	High address space block select register	32
Register Base + 0x0018	FLASH_AR	Address register	32
Register Base + 0x001C	FLASH_BIUCR	Flash bus interface unit control register	32
Register Base + 0x0020	FLASH_BIUAPR	Flash bus interface unit access protection register	32
Register Base + 0x0024	FLASH_BIUCR2	Flash bus interface unit control register 2	32
Register Base + (0x0028–0x7FFF)	—	Reserved	—

13.3.2 Register Descriptions

The flash registers are detailed in the following sections.

13.3.2.1 Module Configuration Register FLASH_MCR

A number of module configuration register (FLASH_MCR) bits are protected from a write while another bit or set of bits are in a specific state. These locks are discussed in relationship to each bit in this section. Simultaneously writing bits which lock each other out is discussed in Section 13.3.2.1.1, "MCR Simultaneous Register Writes." The MCR is always available to be read except when the flash module is disabled.

Address: Base (0xC3F8_8000) + 0x0000 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SIZE				0	LAS			0	0	0	MAS
W																
Reset	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0

	16	17	18	19	20	21	22	238	24	25	26	27	28	29	30	31
R	EER	RWE	1	1	PEAS	DONE	PEG	0	PRD	STOP	0	PGM	PSUS	ERS	ESUS	EHV
W	w1c	w1c														
Reset	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0

Figure 13-4. Module Configuration Register (FLASH_MCR)

Table 13-6. FLASH_MCR Field Descriptions

Field	Description
0–3	Reserved
4–7 SIZE[0:3]	Array space size. Dependent upon the size of the flash module. SIZE is read only. 0000–0010 Invalid value 0011 Total array size is 1.0 MB 0100–1111 Invalid value
8	Reserved
9–11 LAS[0:2]	Low address space. Corresponds to the configuration of the low address space. All possible values of LAS and the configuration to which each value corresponds are shown below. LAS is read only. 110 The LAS value of 110 provides two 16-KB blocks, two 48-KB blocks, and two 64-KB blocks.
12–14	Reserved
15 MAS	Mid address space size. Corresponds to the configuration of the mid address space. MAS is read only. 0 Two 128-KB blocks are available1Invalid value
16 EER	ECC event error. Provides information on previous reads; if a double-bit detection occurred, the EER bit is set to 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. This bit cannot be set by the application. In the event of a single bit detection and correction, this bit is not set. If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) were correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0 Reads are occurring normally. 1 An ECC Error occurred during a previous read. **Note:** This bit can be set on speculative prefetches that cause double bit error detection. Therefore, use the ECSM[FNCE] flag for detecting non-correctable ECC errors in the flash instead of using FLASH_MCR[EER].
17 RWE	Read while write event error. Provides information on previous RWW reads. If a read while write error occurs, this bit is set to 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. This bit cannot be set to 1 by the application. If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) were correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0 Reads are occurring normally. 1 A read while write error occurred during a previous read.
18–19	Reserved

Table 13-6. FLASH_MCR Field Descriptions (continued)

Field	Description
20 PEAS	Program/erase access space. Indicates which space is valid for program and erase operations, either main array space or shadow space. PEAS is read only. 0 Shadow address space is disabled for program/erase and main address space enabled. 1 Shadow address space is enabled for program/erase and main address space disabled.
21 DONE	State machine status. Indicates if the flash module is performing a high voltage operation. DONE is set to a 1 on termination of the flash module reset and at the end of program and erase high voltage sequences. 0 Flash is executing a high voltage operation. 1 Flash is not executing a high voltage operation.
22 PEG	Program/erase good. Indicates the completion status of the last flash program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations. Aborting a program/erase high voltage operation causes PEG to be cleared, indicating the sequence failed. PEG is set to a 1 when the module is reset. PEG is read only. The value of PEG is valid only when PGM = 1 and/or ERS = 1 and after DONE has transitioned from 0 to 1 due to an abort or the completion of a program/erase operation. PEG is valid until PGM/ERS makes a 1 to 0 transition or EHV makes a 0 to 1 transition. The value in PEG is not valid after a 0 to 1 transition of DONE caused by PSUS or ESUS being set to logic 1. A diagram presenting PEG valid times is presented in Figure 13-5. If PGM and ERS are both 1 when DONE makes a qualifying 0 to 1 transition the value of PEG indicates the completion status of the PGM sequence. This happens in an erase-suspended program operation. 0 Program or erase operation failed. 1 Program or erase operation successful.
23	Reserved
24 PRD	Pipelined Reads Disabled. PRD is used to allow pipelined reads to be disabled. By default PRD is 0, which enables pipelined accesses. In systems with slower clocks (<30MHz), the pipelined read feature can be disabled by writing this bit to a 1. This would allow single cycle clock accesses in systems with a slower clock. In systems with faster clocks (>30MHz), accesses are multiple cycles, and the pipelined read feature can be used to get faster throughput on successive reads (PRD = 0). 1 Pipelined Reads are disabled. 0 Pipelined Reads are enabled. **Note:** PRD must be set before setting the flash wait states to 0 (done in FLASH_BIUCR)
25 STOP	Stop mode enabled. Puts the flash into stop mode. Changing the value in STOP from a 0 to a 1 places the flash module in stop mode. A 1 to 0 transition of STOP returns the flash module to normal operation. STOP can be written only when PGM and ERS are low. When STOP = 1, only the STOP bit in the MCR can be written. In STOP mode all address spaces, registers, and register bits are deactivated except for the FLASH_MCR[STOP] bit. 0 Flash is not in stop mode; the read state is active. 1 Flash is in stop mode.
26	Reserved
27 PGM	Program. Used to set up flash for a program operation. A 0 to 1 transition of PGM initiates an flash program sequence. A 1 to 0 transition of PGM ends the program sequence. PGM can be set only under one of the following conditions: • User mode read (STOP and ERS are low). • Erase suspend[1] (ERS and ESUS are 1) with EHV low. PGM can be cleared by you only when EHV are low and DONE is high. PGM is cleared on reset. 0 Flash is not executing a program sequence. 1 Flash is executing a program sequence.

Table 13-6. FLASH_MCR Field Descriptions (continued)

Field	Description
28 PSUS	Program suspend. Indicates the flash module is in program suspend or in the process of entering a suspend state. The flash module is in program suspend when PSUS = 1 and DONE = 1. PSUS can be set high only when PGM and EHV are high. A 0 to 1 transition of PSUS starts the sequence which sets DONE and places the flash in program suspend. PSUS can be cleared only when DONE and EHV are high. A 1 to 0 transition of PSUS with EHV = 1 starts the sequence which clears DONE and returns the flash module to program. The flash module cannot exit program suspend and clear DONE while EHV is low. PSUS is cleared on reset. 0 Program sequence is not suspended. 1 Program sequence is suspended.
29 ERS	Erase. Used to set up flash for an erase operation. A 0 to 1 transition of ERS initiates an flash erase sequence. A 1 to 0 transition of ERS ends the erase sequence. ERS can be set only in a normal operating mode read (STOP and PGM are low). ERS can be cleared by you only when ESUS and EHV are low and DONE is high. ERS is cleared on reset. 0 Flash is not executing an erase sequence. 1 Flash is executing an erase sequence.
30 ESUS	Erase suspend. Indicates that the flash module is in erase suspend or in the process of entering a suspend state. The flash module is in erase suspend when ESUS = 1 and DONE = 1. ESUS can be set high only when ERS and EHV are high and PGM is low. A 0 to 1 transition of ESUS starts the sequence which sets DONE and places the flash in erase suspend. ESUS can be cleared only when DONE and EHV are high and PGM is low. A 1 to 0 transition of ESUS with EHV = 1 starts the sequence which clears DONE and returns the flash module to erase mode. The flash module cannot exit erase suspend and clear DONE while EHV is low. ESUS is cleared on reset. 0 Erase sequence is not suspended. 1 Erase sequence is suspended.
31 EHV	Enable high voltage. Enables the flash module for a high voltage program/erase operation. EHV is cleared on reset. EHV must be set after an interlock write to start a program/erase sequence. EHV can be set, initiating a program/erase, after an interlock write under one of the following conditions: • Erase (ERS = 1, ESUS = 0). • Program (ERS = 0, ESUS = 0, PGM = 1, PSUS = 0). • Erase-suspended program (ERS = 1, ESUS = 1, PGM = 1, PSUS = 0). If a program operation is to be initiated while an erase is suspended you must clear EHV while in erase suspend before setting PGM. In normal operation, a 1 to 0 transition of EHV with DONE high, PSUS and ESUS low terminates the current program/erase high voltage operation. When an operation is aborted[2], there is a 1 to 0 transition of EHV with DONE low and the suspend bit for the current program/erase sequence low. An abort causes the value of PEG to be cleared, indicating a failed program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted. EHV can be written during suspend. EHV must be high for the flash to exit suspend. Do not write the EHV bit after a suspend bit is set high and before DONE has transitioned high. Do not set the EHV bit low after the current suspend bit is set low and before DONE has transitioned low. 0 Flash is not enabled to perform a high voltage operation. 1 Flash is enabled to perform a high voltage operation.

[1] In an erase-suspended program, programming flash locations in blocks which were being operated on in the erase can corrupt flash core data. Avoid this due to reliability implications.

[2] Aborting a high voltage operation leaves flash core addresses in an indeterminate data state. This can be recovered by executing an erase on the affected blocks.

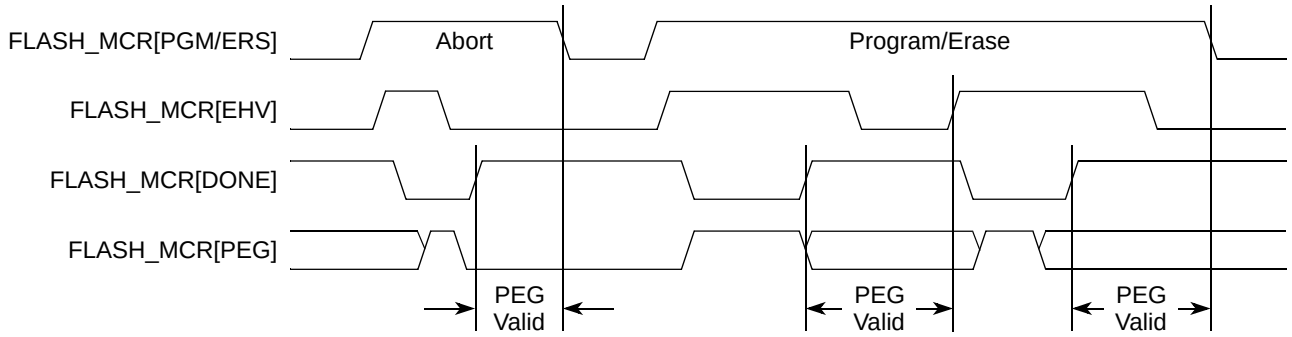


Figure 13-5. PEG Valid Times

13.3.2.1.1 MCR Simultaneous Register Writes

A number of MCR bits are protected against write when another bit or set of bits is in a specific state. These write locks are covered on a bit by bit basis in Section 13.3.2.1, "Module Configuration Register FLASH_MCR." The write locks detailed in that section do not consider the effects of trying to write two or more bits simultaneously. The effects of writing bits simultaneously which would put the flash module in an illegal state are detailed here.

The flash does not allow you to write bits simultaneously which would put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in Table 13-7.

Table 13-7. MCR Bit Set/Clear Priority Levels

Priority Level	MCR Bits
1	STOP
2	ERS
3	PGM
4	EHV
5	ESUS, PSUS

If you try to write two or more MCR bits simultaneously then only the bit with the highest priority level is written. Setting two bits with the same priority level is prevented by existing write locks and does not put the flash in an illegal state.

For example, setting FLASH_MCR[STOP] and FLASH_MCR[PGM] simultaneously results in only FLASH_MCR[STOP] being set. Attempting to clear FLASH_MCR[EHV] while setting FLASH_MCR[PSUS] results in FLASH_MCR[EHV] being cleared, while FLASH_MCR[PSUS] remains unaffected.

13.3.2.2 Low/Mid Address Space Block Locking Register FLASH_LMLR

The low and mid address block locking register provides a means to protect blocks from being modified. These bits along with bits in the secondary LMLOCK field (FLASH_SLMLR), determine if the block is locked from program or erase. An “OR”’ of FLASH_LMLR and FLASH_SLMLR determine the final lock status. See Section 13.3.2.4, “Secondary Low/Mid Address Space Block Locking Register FLASH_SLMLR” for more information on FLASH_SLMLR.

NOTE

In the event that blocks are not present (due to configuration or total memory size), the LOCK bits defaults to locked, and are not writable. The reset value is always 1 (independent of the shadow block), and register writes have no effect.

Address: Base (0xC3F8_8000) + 0x0004 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LME	0	0	0	0	0	0	0	0	0	0	SLOCK	1	1	MLOCK		1	1	1	1	1	1	1	1	1	1	LLOCK					
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1	1^1

1 The reset value of these bits is determined by flash values in the shadow block. Erasing the array sets the reset value to 1.

Figure 13-6. Low/Mid Address Space Block Locking Register (FLASH_LMLR)

Table 13-8. FLASH_LMLR Field Descriptions

Field	Description
0 LME	Low and mid address lock enable. Enables the locking register fields (SLOCK, MLOCK and LLOCK) to be set or cleared by register writes. This bit is a status bit only, and cannot be written or cleared, and the reset value is 0. The method to set this bit is to write a password, and if the password matches, the LME bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME, the password 0xA1A1_1111 must be written to the FLASH_LMLR. 0 Low and mid address locks are disabled, and cannot be modified. 1 Low and mid address locks are enabled and can be written.
1–10	Reserved
11 SLOCK	Shadow lock. Locks the shadow block from programs and erases. The SLOCK bit is not writable if a high voltage operation is suspended. Upon reset, information from the shadow block is loaded into the SLOCK bit. The SLOCK bit can be written as a register. Reset causes the bits to go back to their shadow block value. The default value of the SLOCK bit (assuming the corresponding shadow block bit is erased) would be locked. SLOCK is not writable unless LME is high. 0 Shadow block is available to receive program and erase pulses. 1 Shadow block is locked for program and erase.
12–13	Reserved

Table 13-8. FLASH_LMLR Field Descriptions (continued)

Field	Description
14–15 MLOCK[1:0]	Mid address block lock. A value of 1 in a bit of the lock register signifies that the corresponding block is locked for program and erase. A value of 0 in the lock register signifies that the corresponding block is available to receive program and erase pulses. Likewise the lock register is not writable if a high voltage operation is suspended. Upon reset, information from the shadow block is loaded into the block registers. The LOCK bits can be written as a register. Reset causes the bits to go back to their shadow block value. The default value of the LOCK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the LOCK bits default to locked, and are not writable. The reset value is always 1 (independent of the shadow block), and register writes have no effect. MLOCK is not writable unless LME is high.
16–25	Reserved
26–31 LLOCK[5:0]	Low address block lock. These bits have the same description and attributes as MLOCK. As an example of how the LLOCK bits are used, if a configuration has six 16-KB blocks in the low address space, the block residing at address array base + 0, corresponds to LLOCK0. The next 16-KB block corresponds to LLOCK1, and so on up to LLOCK5.

13.3.2.3 High Address Space Block Locking Register (FLASH_HLR)

The high address space block locking register provides a means to protect blocks from being modified.

Address: Base (0xC3F8_8000) + 0x0008 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	HBE	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	HLOCK			
W																																
Reset	0	0	0	0	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]

[1] The reset value of these bits is determined by flash values in the shadow block. An erased array causes the reset value to be 1.

Figure 13-7. High Address Space Block Locking Register (FLASH_HLR)

Table 13-9. FLASH_HLR Field Descriptions

Field	Description
0 HBE	High address lock enable. Enables the locking field (HLOCK) to be set or cleared by register writes. This bit is a status bit only, and cannot be written to or cleared, and the reset value is 0. The method to set this bit is to provide a password, and if the password matches, the HBE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE, the password 0xB2B2_2222 must be written to FLASH_HLR. 0 High address locks are disabled, and cannot be modified. 1 High address locks are enabled to be written.
1–27	Reserved
28–31 HLOCK[3:0]	High address space block lock. Has the same characteristics as MLOCK. See Section 13.3.2.2, “Low/Mid Address Space Block Locking Register FLASH_LMLR” for more information. The block numbering for High Address space starts with HLOCK[0] and continues until all blocks are accounted. HLOCK is not writable unless HBE is set. In the event that blocks are not present (due to configuration or total memory size), the HLOCK bits default to locked, and are not writable.

13.3.2.4 Secondary Low/Mid Address Space Block Locking Register FLASH_SLMLR

The FLASH_SLMLR provides an alternative means to protect blocks from being modified. These bits along with bits in the LMLOCK field (FLASH_LMLR), determine if the block is locked from program or erase. An “OR” of FLASH_LMLR and FLASH_SLMLR determine the final lock status. See Section 13.3.2.2, “Low/Mid Address Space Block Locking Register FLASH_LMLR” for more information on FLASH_LMLR.

Address: Base (0xC3F8_8000) + 0x000C Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLE	0	0	0	0	0	0	0	0	0	0	SS LOCK	1	1	SM LOCK		1	1	1	1	1	1	1	1	1	1	SLLOCK					
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	1[1]	1	1	1[1]	1[1]	1	1	1	1	1	1	1	1	1	1	1[1]	1[1]	1[1]	1[1]	1[1]	1[1]

[1] The reset value of these bits is determined by flash values in the shadow block. An erased array sets the reset value to 1.

Figure 13-8. Secondary Low/Mid Address Space Block Locking Register (FLASH_SLMLR)

Table 13-10. FLASH_SLMLR Field Descriptions

Field	Description
0 SLE	Secondary low and mid address lock enable. Enables the secondary lock fields (SSLOCK, SMLOCK, and SLLOCK) to be set or cleared by register writes. This bit is a status bit only, and cannot be written to or cleared, and the reset value is 0. The method to set this bit is to provide a password, and if the password matches, the SLE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE, the password 0xC3C3_3333 must be written to the FLASH_SLMLR. 0 Secondary low and mid address locks are disabled, and cannot be modified. 1 Secondary low and mid address locks are enabled to be written.
1–10	Reserved
11 SSLOCK	Secondary shadow lock. An alternative method to lock the shadow block from programs and erases. SSLOCK has the same description as SLOCK in Section 13.3.2.2, “Low/Mid Address Space Block Locking Register FLASH_LMLR.” SSLOCK is not writable unless SLE is high.
12–13	Reserved
14–15 SMLOCK [1:0]	Secondary mid address block lock. Alternative method to lock the mid address space blocks from programs and erases. SMLOCK has the same description as MLOCK in section Section 13.3.2.2, “Low/Mid Address Space Block Locking Register FLASH_LMLR.” SMLOCK is not writable unless SLE is set. In the event that blocks are not present (due to configuration or total memory size), the SMLOCK bits default to locked, and are not writable.
16–25	Reserved
26–31 SLLOCK [5:0]	Secondary low address block lock. These bits are an alternative method to lock the low address space blocks from programs and erases. SLLOCK has the same description as LLOCK in Section 13.3.2.2, “Low/Mid Address Space Block Locking Register FLASH_LMLR. SLLOCK is not writable unless SLE is high. In the event that blocks are not present (due to configuration or total memory size), the SLLOCK bits default to locked, and are not writable.

13.3.2.5 Low/Mid Address Space Block Select Register FLASH_LMSR

The FLASH_LMSR provides a means to select blocks to be operated on during erase.

Address: Base (0xC3F8_8000) + 0x0010 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSEL		0	0	0	0	0	0	0	0	0	0	LSEL					
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-9. Low/Mid Address Space Block Select Register (FLASH_LMSR)

Table 13-11. FLASH_LMSR Field Descriptions

Field	Description
0–13	Reserved
14–15 MSEL[1:0]	Mid address space block select. Values in the selected register signify that a block(s) is or is not selected for erase. The reset value for the select registers is 0. The blocks must be selected (or unselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable after an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding SELECT bits default to unselected, and are not writable. The reset value is always 0, and register writes have no effect. A description of how blocks are numbered is detailed in Section 13.3.2.2, “Low/Mid Address Space Block Locking Register FLASH_LMLR.” 0b0000 Mid address space blocks are *not* selected for erase 0b0001 One mid address space block is selected for erase 0b0011 Two mid address space blocks are selected for erase
16–25	Reserved
26–31 LSEL[5:0]	Low address space block select. Used to select blocks in the low address space; these have the same description and attributes as the MSEL bits 0b0000 Low address space blocks are *not* selected for erase 0b0001 One low address space block is selected for erase 0b0011 Two low address space blocks are selected for erase 0b0111 Three low address space blocks are selected for erase 0b1111 Four low address space blocks are selected for erase 0b1_1111 Five low address space blocks are selected for erase 0b11_1111 Six low address space blocks are selected for erase

13.3.2.6 High Address Space Block Select Register FLASH_HSR

The FLASH_HSR allows the application to select the high address flash blocks on which to operate.

Address: Base (0xC3F8_8000) + 0x0014 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	HBSEL			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-10. High Address Space Block Select Register (FLASH_HSR)

Table 13-12. FLASH_HSR Field Descriptions

Field	Description
0–27	Reserved
28–31 HBSEL[3:0]	High address space block select. Has the same characteristics as MSEL. For more information see Section 13.3.2.5, "Low/Mid Address Space Block Select Register FLASH_LMSR." 0b0000 High address space blocks are *not* selected for erase 0b0001 One high address space block is selected for erase 0b0011 Two high address space blocks are selected for erase 0b0111 Three high address space blocks are selected for erase 0b1111 Four high address space blocks are selected for erase

13.3.2.7 Address Register FLASH_AR

The FLASH_AR provides the first failing address in the event of ECC event error (FLASH_MCR[EER] set), as well as providing the address of a failure that occurs in a state machine operation (FLASH_MCR[PEG] cleared). ECC event errors take priority over state machine errors. This is especially valuable in the event of a RWW operation, where the read senses an ECC error and the state machine fails simultaneously. This address is always a doubleword address that selects 64 bits.

In normal operating mode, the FLASH_AR is not writable.

Address: Base (0xC3F8_8000) + 0x0018 Access: User R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	ADDR																			0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-11. Address Register (FLASH_AR)

Table 13-13. FLASH_AR Field Descriptions

Field	Description
0–9	Reserved
10–28 ADDR[3:21]	Doubleword address of first failing address in the event of an ECC error, or the address of a failure occurring during state machine operation.
29–31 ADDR[0:2]	Always read as 0.

13.3.2.8 Flash Bus Interface Unit Control Register FLASH_BIUCR

The FLASH_BIUCR is the control register for the set up and control of the flash interface. This register must not be written while executing from flash. Only use a 32-bit write operation to write to this register.

Address: Base (0xC3F8_8000) + 0x001C Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	M3 PFE	M2 PFE	M1 PFE	M0 PFE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	APC			WWSC		RWSC			0	DPF EN	0	IPF EN	PFLIM			BFEN
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Figure 13-12. Flash Bus Interface Unit Control Register (FLASH_BIUCR)

Table 13-14. FLASH_BIUCR Field Descriptions

Bits	Description
0–11	Reserved. Do not set these bits.
12–15 M*n*PFE	Master *n* prefetch enable. Used to control whether prefetching can be triggered based on the master ID of a requesting master. These bits are cleared by hardware reset. 0 No prefetching can be triggered by this master 1 Prefetching can be triggered by this master These fields are identified as follows: M3PFE = EBI M2PFE = eDMA M1PFE = Nexus M0PFE = MCU core
16–18 APC [1]	Address pipelining control. Used to control the number of cycles between pipelined access requests. This field must be set to a value corresponding to the operating frequency of the system clock. The required settings are documented in Table 13-15. 000 Accesses can be pipelined back-to-back 001 Access requests require one additional hold cycle 010 Access requests require two additional hold cycles ... 110 Access requests require six additional hold cycles 111 No address pipelining
19–20 WWSC [1]	Write wait state control. Used to control the number of wait-states added to the best-case flash array access time for writes. This field must be set to a value corresponding to the operating frequency of the system clock. The required settings are documented in Table 13-15. 00 No additional wait states are added 01 One additional wait state is added 10 Two additional wait states are added 11 Three additional wait states are added
21–23 RWSC [1]	Read wait state control. Used to control the number of wait states added to the best-case flash array access time for reads. This field must be set to a value corresponding to the operating frequency of the system clock. The required settings are documented in Table 13-16. 000 No additional wait states are added 001 One additional wait state is added ... 111 Seven additional wait states are added

Table 13-14. FLASH_BIUCR Field Descriptions (continued)

Bits	Description
24	Reserved. Do not set this bit.
25 DPFEN	Data prefetch enable. Enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. 0 No prefetching is triggered by a data read access 1 Prefetching can be triggered by any data read access
26	Reserved. Do not set this bit.
27 IPFEN	Instruction prefetch enable. Enables or disables prefetching initiated by an instruction read access. This field is cleared by hardware reset. 0 No prefetching is triggered by an instruction read access 1 Prefetching can be triggered by any instruction read access
28–30 PFLIM	Prefetch limit. Controls the prefetch algorithm used by the FBIU prefetch controller. This field defines a limit on the maximum number of sequential prefetches attempted between buffer misses. This field is cleared by hardware reset. 000 No prefetching is performed 001 The referenced line is prefetched on a buffer miss (i.e. prefetch on miss) 01x The referenced line is prefetched on a buffer miss, or the next sequential line is prefetched on a buffer hit (if not already present), i.e., prefetch on miss or hit. 1xx See 01x (support for legacy code)
31 BFEN	FBIU line read buffers enable. Enables or disables line read buffer hits. It is also used to invalidate the buffers. These bits are cleared by hardware reset. 0 The line read buffers are disabled from satisfying read requests, and all buffer valid bits are cleared. 1 The line read buffers are enabled to satisfy read requests on hits. Buffer valid bits can be set when the buffers are successfully filled. **Note:** Disable prefetching before invalidating the buffers. This includes starting a program or erase operation, or turning on and off the buffers.

[1] APC, WWSC, and RWSC values are determined by the maximum frequency of operation. See Table 13-15.

Table 13-15. FLASH_BIU Settings vs. Frequency of Operation[1]

Target Maximum Frequency (MHz)	APC	WWSC	RWSC	DPFEN [2]	IPFEN [2]	PFLIM [3]	BFEN [2]
up to and including 27 MHz [4, 5]	0b000	0b01	0b000	0b0, 0b1	0b0, 0b1	0b000 to 0b010	0b0, 0b1
up to and including 52 MHz [6]	0b001	0b01	0b001	0b0, 0b1	0b0, 0b1	0b000 to 0b010	0b0, 0b1
up to and including 77 MHz [7]	0b010	0b01	0b010	0b0, 0b1	0b0, 0b1	0b000 to 0b010	0b0, 0b1
up to and including 82 MHz [8]	0b011	0b01	0b011	0b0, 0b1	0b0, 0b1	0b000 to 0b010	0b0, 0b1
Reset values:	0b111	0b11	0b111	0b0	0b0	0b000	0b0

1 Illegal combinations exist. Therefore, all entries must be taken from the same row in this table.
2 For maximum flash performance, set to 0b1.
3 For maximum flash performance, set to 0b010.
4 27 MHz parts allow for 25 MHz system clock + 2% frequency modulation (FM).
5 The APC/RWSC/WWSC combination requires setting the flash MCR register bit PRD=1.
6 52 MHz parts allow for 50 MHz system clock + 2% frequency modulation (FM).
7 77 MHz parts allow for 75 MHz system clock + 2% frequency modulation (FM).
8 82 MHz parts allow for 80 MHz system clock + 2% frequency modulation (FM).

13.3.2.9 Flash Bus Interface Unit Access Protection Register FLASH_BIUAPR

The FLASH_BIUAPR controls access protection for the flash from masters on the crossbar switch. Use a 32-bit write operation only to this register.

Address: Base (0xC3F8_8000) + 0x0020 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	1	1	1	1	1	1	1	1	M3AP		M2AP		M1AP		M0AP	
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 13-13. Flash Bus Interface Unit Access Protection Register (FLASH_BIUAPR)

Table 13-16. FLASH_BIUAPR Field Descriptions

Field	Description
0–23	Reserved. Reads/Writes have no effect.
24–31 M*n*AP [0:1]	Master *n* access protection. Controls whether read and write accesses to the flash are allowed based on the master ID of a requesting master. These fields are initialized by hardware reset. See Table 7-4. 00 No accesses can be performed by this master 01 Only read accesses can be performed by this master 10 Only write accesses can be performed by this master 11 Both read and write accesses can be performed by this master These fields are identified as follows: M0AP= MCU core M1AP= Nexus M2AP= eDMA M3AP= EBI

13.3.2.10 Flash Bus Interface Unit Control Register 2 FLASH_BIUCR2

The FLASH_BIUCR2 defines the operations of the four line buffers.

Address: Base (0xC3F8_8000) + 0x0024 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LBCFG		0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	–[1]	–[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	238	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] See bit description in Table 13-17 for reset values.

Figure 13-14. Flash Bus Interface Unit Control Register 2 (FLASH_BIUCR2)

Table 13-17. FLASH_BIUCR2 Field Descriptions

Bits	Description
0–1 LBCFG	Line Buffer Configuration.This field controls the configuration of the four line buffers in the FBIU controller. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. In all cases, when a buffer miss occurs, it is allocated to the least-recently-used buffer within the group and the just-fetched entry then marked as most-recently-used. If the flash access is for the next-sequential line, the buffer is not marked as most-recently-used until the given address produces a buffer hit. This field is initialized by hardware reset to the value contained in the address (0x0200 + the shadow base address) of the flash array. An erased or unprogrammed flash sets this field to 0b11. 00 All four buffers are available for any flash access, i.e., there is no partitioning of the buffers based on the access type. 01 Reserved 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11 The buffers are partitioned into two groups with buffers 0,1, 2 allocated for instruction fetches and buffer 3 for data accesses.
2–31	Reserved

To temporarily change the values of any of the fields in the FLASH_BIUCR2, a write to the IPS-mapped register is performed. To change the values loaded into the FLASH_BIUCR2 at reset, the word location at address (0x0200 + the shadow base address) in the flash array must be programmed using the normal sequence of operations.

13.4 Functional Description

13.4.1 Flash Bus Interface Unit (FBIU)

The flash BIU interfaces between the system bus and the flash memory interface unit and generates read and write enables, the flash array address, write size, and write data as inputs to the flash memory interface unit (MI). The flash BIU captures read data from the MI and drives it on the system bus. Up to four lines (1 line is a 128-bit width) of data or instructions are buffered by the flash BIU. Lines can be prefetched in advance of being requested by the system bus interface, allowing single-cycle read data responses on buffer hits.

Several prefetch control algorithms are available for controlling line read buffer fills. Prefetch triggering can be restricted to instruction accesses only, data accesses only, or can be unrestricted. Prefetch triggering can also be controlled on a per-master basis.

Buffers can also be selectively enabled or disabled for allocation by instruction and data prefetch (see Section 13.3.2.10, “Flash Bus Interface Unit Control Register 2 FLASH_BIUCR2”).

Access protections can be applied on a per-master basis for both reads and writes to support security and privilege mechanisms.

13.4.1.1 FBIU Basic Interface Protocol

The flash BIU interfaces to the flash array by driving addresses and read or write enable signals to the flash memory interface unit. The access time of the flash is determined by the settings of the wait state control bits in the FLASH_BIUCR, as well as the pipelining of addresses.

The flash BIU also has the capability of extending the normal system bus access timing by inserting additional primary (initial access) wait states for reads and burst reads. This capability is provided to allow emulation of other memories which have different access time characteristics.

13.4.1.2 FBIU Access Protections

The flash BIU provides hardware configurable access protections for both read and write cycles from masters. It allows restriction of read and write requests on a per-master basis. The FBIU also supports software configurable access protections. Detection of a protection violation results in an error response from the flash BIU to the system bus.

13.4.1.3 Flash Read Cycles—Buffer Miss

Read data is normally stored in the least-recently updated line read buffer in parallel with the requested data being forwarded to the system bus. If the flash access was directly the result of a system bus transaction, the line buffer is marked as most-recently-used as it is being loaded. If the flash access was the result of a speculative prefetch to the next sequential line, it is first loaded into the least-recently-used buffer. The status of this buffer is not changed to most-recently-used until a subsequent buffer hit occurs.

13.4.1.4 Flash Read Cycles—Buffer Hit

Single clock read responses to the system bus are possible with the flash BIU when the requested read access is buffered.

13.4.1.5 Flash Access Pipelining

Accesses to the flash array can be pipelined by driving a subsequent access address and control signals while waiting for the current access to complete. Pipelined access requests are always run to completion and are not aborted by the flash BIU. Request pipelining allows for improved performance by reducing the access latency seen by the system bus master. Access pipelining can be applied to both read and write cycles by the flash array.

13.4.1.6 Flash Error Response Operation

The flash array can terminate a requested access with an error. This can occur due to an ECC error that is uncorrectable, an access control violation, or because of improper access sequencing during program/erase operations. When an error response is received, the flash BIU marks a line read buffer as invalid. An error response can be signaled on read or write operations.

13.4.1.7 FBIU Line Read Buffers and Prefetch Operation

The flash BIU contains four 128-bit line read buffers which are used to hold data read from the flash array. Each buffer operates independently and is filled using a single array access. The buffers are used for both prefetch and normal demand fetches.

Prefetch triggering is controllable on a per-master and access-type basis. Bus masters can be enabled or disabled from triggering prefetches, and triggering can be further restricted based on whether a read access is for instruction or data and whether or not it is a burst access. A read access to the flash BIU can trigger a prefetch to the next sequential line of array data on the cycle following the request. The access address is incremented to the next-higher 16-byte boundary, and a flash array prefetch is initiated if the data is not already resident in a line read buffer. Prefetched data is always loaded into the least-recently-used buffer.

Buffers can be in one of six states, listed here in prioritized order:

- Invalid—the buffer contains no valid data.
- Used—the buffer contains valid data which has been provided to satisfy a burst type read.
- Valid—the buffer contains valid data which has been provided to satisfy a single type read.
- Prefetched—the buffer contains valid data which has been prefetched to satisfy a potential future access.
- Busy—the buffer is currently being used to satisfy a burst read.
- Busy fill—the buffer has been allocated to receive data from the flash array, and the array access is still in progress.

Selection of a buffer to be loaded on a miss is based on the following replacement algorithm:

1. First, the buffers are examined to determine if there are any invalid buffers. If there are multiple invalid buffers, the one to be used is selected using a reverse numeric priority, where buffer 0 is selected first, then buffer 1, etc.
2. If there are no invalid buffers, the least-recently-used buffer is selected for replacement.

Once the candidate line buffer has been selected, the flash array is accessed and read data loaded into the buffer. If the buffer load was in response to a miss, the just-loaded buffer is immediately marked as most-recently-used. If the buffer load was in response to a speculative fetch to the next-sequential line address after a buffer hit, the recently-used status is not changed. Rather, it is marked as most-recently-used only after a subsequent buffer hit. This policy maximizes performance based on reference patterns of flash accesses and allows for prefetched data to remain valid when non-prefetch enabled bus masters are granted flash access.

Several algorithms are available for prefetch control which trade off performance for power. They are described in Section 13.3.2.8, “Flash Bus Interface Unit Control Register FLASH_BIUCR.” More aggressive prefetching increases power due to the number of wasted (discarded) prefetches, but can increase performance by lowering average read latency.

13.4.1.8 Prefetch Triggering

Prefetch triggering can be enabled for instruction and data reads, but never by write cycles. Prefetch triggering can be controlled for individual bus masters.

13.4.1.9 FBIU Buffer Invalidation

The line read buffers can be invalidated under hardware and software control. Buffers are automatically invalidated whenever the buffers are turned on or off, or at the beginning of a program or erase operation.

NOTE

Disable prefetching before invalidating the buffers. This includes starting a program or erase operation, or turning on and off the buffers.

13.4.1.10 Flash Wait-state Emulation

Emulation of other memory array timings are supported by the flash BIU. This functionality can be useful to maintain the access timing for blocks of memory which were used to overlay flash blocks for the purpose of system calibration or tuning during code development.

The flash BIU inserts additional wait states according to the upper address lines ADDR[28:24]. When these address lines are non-zero, additional cycles are added to system bus transfers. Normal system bus termination is extended. In addition, no line read buffer prefetches are initiated, and buffer hits are ignored.

13.4.2 Flash Memory Array: User Mode

In user (normal) operating mode the flash module can be read, written (register writes and interlock writes), programmed, or erased. The following subsections define all actions that can be performed in normal operating mode. The registers mentioned in these sections are detailed in Section 13.3.2, “Register Descriptions.”

13.4.2.1 Flash Read and Write

The default state of the flash module is read. The main and shadow address space can be read only in the read state. The module configuration register (FLASH_MCR) is always available for read. The flash module enters the read state on reset. The flash module is in the read state under four sets of conditions:

- The read state is active when FLASH_MCR[STOP] = 0 (user mode read).
- The read state is active when FLASH_MCR[PGM] = 1 and/or FLASH_MCR[ERS] = 1 and high voltage operation is ongoing (read while write).

NOTE

Reads done to the partitions being operated on (either erased or programmed) result in errors and the FLASH_MCR[RWE] bit is set.

- The read state is active when FLASH_MCR[PGM] = 1 and FLASH_MCR[PSUS] = 1 in the MCR. (Program suspend).
- The read state is active when FLASH_MCR[ERS] = 1 and FLASH_MCR[ESUS] = 1 and FLASH_MCR[PGM] = 0 in the MCR. (Erase suspend).

NOTE

Flash core reads are done through the BIU. In many cases the BIU does page buffering to allow sequential reads to be done with higher performance. This can create a data coherency issue that must be handled with software. Data coherency can be an issue after a program or erase operation, as well as shadow block operations.

In flash normal operating mode, registers can be written and the flash array can be written to do interlock writes. Reads attempted to invalid locations result in indeterminate data. Invalid locations occur when addressing is done to blocks that do not exist in non 2^n array sizes. Interlock writes attempted to invalid locations (due to blocks that do not exist in non 2^n array sizes), result in an interlock occurring, but attempts to program or erase these blocks do not occur since they are forced to be locked.

See the following sections for more information:

Section 13.3.2.2, “Low/Mid Address Space Block Locking Register FLASH_LMLR”
Section 13.3.2.3, “High Address Space Block Locking Register (FLASH_HLR)”
Section 13.3.2.4, “Secondary Low/Mid Address Space Block Locking Register FLASH_SLMLR”

13.4.2.2Read While Write (RWW)

The flash core is divided into partitions. Partitions are always comprised of two or more blocks. Partitions are used to determine read while write (RWW) groupings. While a write (program or erase) is being done within a given partition, a read can be simultaneously executed to any other partition. Partitions are listed in Table 13-4. Each partition in high address space comprises of two 128-KB blocks. The shadow block has unique RWW restrictions described in Section 13.4.2.5, “Flash Shadow Block.”

The flash core is also divided into blocks to implement independent erase or program protection. The shadow block exists outside the normal address space and is programmed, erased and read independently of the other blocks. The shadow block is included to support systems that require NVM for security or system initialization information.

A software mechanism is provided to independently lock or unlock each block in high-, mid-, and low-address space against program and erase.

13.4.2.3 Flash Programming

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1. Addresses in locked/disabled blocks cannot be programmed. You can program the values in any or all of four words within a page in a single program sequence. Word addresses are selected using bits 3:2 of the page-bound word.

Whenever a program operation occurs, ECC bits are programmed. ECC is handled on a 64-bit boundary. Thus, if only 1 word in any given 64-bit ECC segment is programmed, do not program the adjoining word (in that segment) because the ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word results in an operation failure. All programming operations must be from 64 bits to 128 bits, and be 64-bit aligned. The programming operation must completely fill the selected ECC segments within the page.

The program operation consists of the following sequence of events:

1. Change the value in the FLASH_MCR[PGM] bit from a 0 to a 1.

NOTE

Ensure the block that contains the address to be programmed is unlocked.

2. Write the first address to be programmed in the flash module with the program data. This write is referred to as a program data interlock write. An interlock write can either be an aligned word or doubleword.
3. To program more than one word or doubleword, write the data to be programmed into each additional address in the page, which is called a program data write. All unwritten data words default to 0xFFFF_FFFF.
4. Write a logic 1 to the FLASH_MCR[EHV] bit to start the internal program sequence or skip to step 9 to terminate.
5. Wait until the FLASH_MCR[DONE] bit goes high.
6. Confirm FLASH_MCR[PEG] = 1.
7. Write a logic 0 to the FLASH_MCR[EHV] bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the FLASH_MCR[PGM] bit to terminate the program sequence.

The program sequence is presented graphically in Figure 13-15. The program suspend operation detailed in Figure 13-15 is discussed in Section 13.4.2.3.2, “Flash Program Suspend/Resume.”

The first write after a program is initiated determines the page address to be programmed. The program can be initiated with the 0 to 1 transition of the FLASH_MCR[PGM] bit or by clearing the FLASH_MCR[EHV] bit at the end of a previous program. This first write is referred to as an interlock write. If the program is not an erase-suspended program, the interlock write determines if the shadow or normal array space is programmed and causes FLASH_MCR[PEAS] to be set/cleared.

In the case of an erase-suspended program, the value in FLASH_MCR[PEAS], is retained from the erase.

An interlock write must be performed before setting FLASH_MCR[EHV]. You can terminate a program sequence by clearing FLASH_MCR[PGM] prior to setting FLASH_MCR[EHV].

If multiple writes are done to the same location the data for the last write is used in programming.

While FLASH_MCR[DONE] is low, FLASH_MCR[EHV] is high and FLASH_MCR[PSUS] is low you can clear FLASH_MCR[EHV], resulting in a program abort. A program abort forces the module to step 8 of the program sequence. An aborted program results in FLASH_MCR[PEG] being set low, indicating a failed operation. The data space being operated on before the abort contains indeterminate data. You cannot abort a program sequence while in program suspend.

WARNING

Aborting a program operation leaves the flash core addresses being programmed in an indeterminate data state. This can be recovered by executing an erase on the affected blocks.

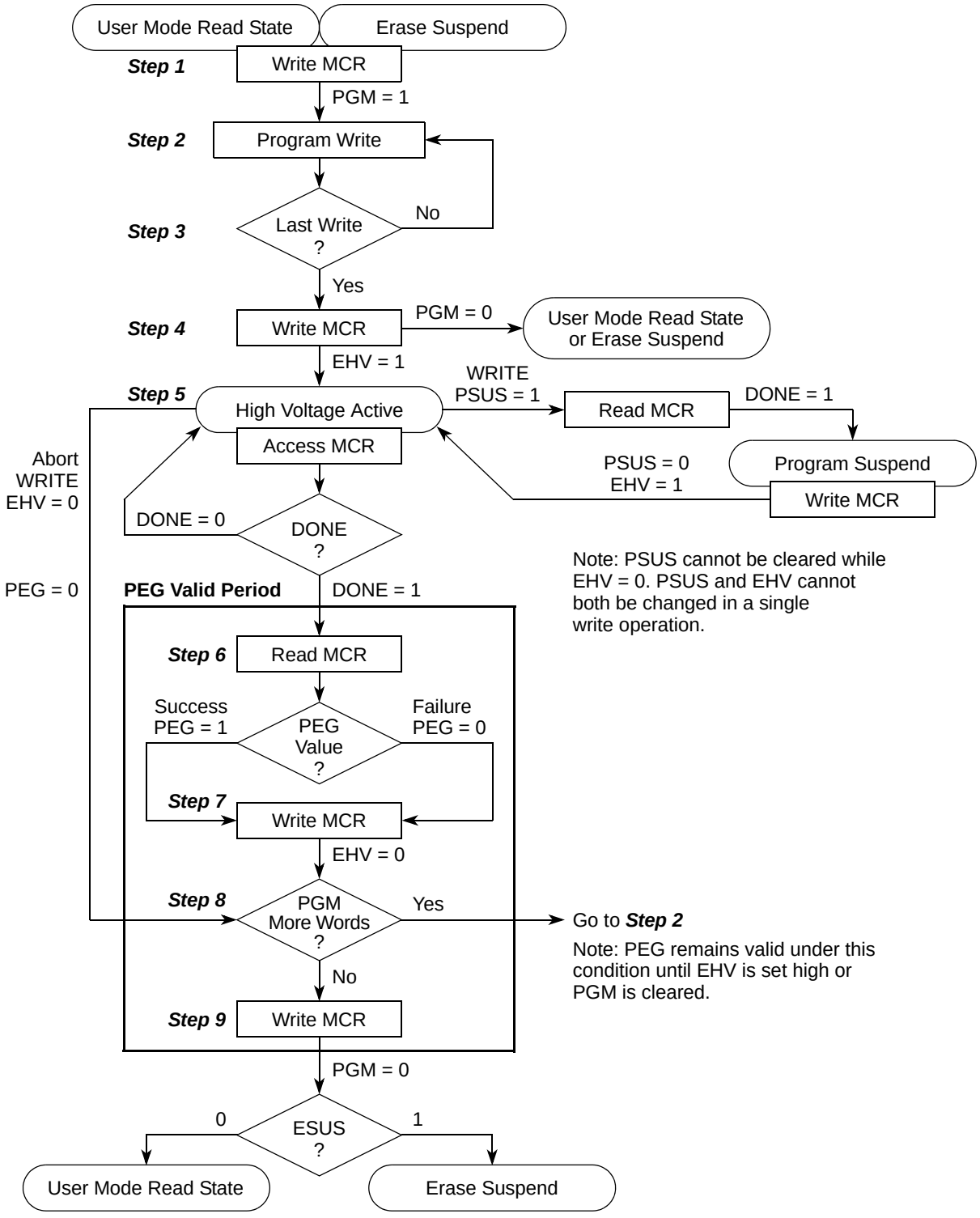


Figure 13-15. Program Sequence

13.4.2.3.1 Software Locking

A software mechanism is provided to independently lock/unlock each high, mid, and low address space against program and erase.

Software Locking is done through the FLASH_LMLR (low/mid address space block locking register), FLASH_SLMLR (secondary low/mid address space block locking register), or FLASH_HLR (high address space block locking register). These can be written through register writes, and can be read through register reads.

When the program/erase operations are enabled through hardware, software locks are enforced through doing register writes.

13.4.2.3.2 Flash Program Suspend/Resume

The program sequence can be suspended to allow read access to the flash core. It is not possible to erase or program during a program suspend. Do not attempt interlock writes during program suspend.

A program suspend can be initiated by changing the value of the FLASH_MCR[PSUS] bit from a 0 to a 1. FLASH_MCR[PSUS] can be set high at any time when FLASH_MCR[PGM] and FLASH_MCR[EHV] are high. A 0 to 1 transition of FLASH_MCR[PSUS] causes the flash module to start the sequence to enter program suspend, which is a read state. The module is not suspended until FLASH_MCR[DONE] = 1. At this time, flash core reads can be attempted. After it is suspended, the flash core can only be read. Reads to the blocks being programmed/erased return indeterminate data.

The program sequence is resumed by writing a logic 0 to FLASH_MCR[PSUS]. FLASH_MCR[EHV] must be set to a 1 before clearing FLASH_MCR[PSUS] to resume operation. When the operation resumes, the flash module continues the program sequence from one of a set of predefined points. This can extend the time required for the program operation.

13.4.2.4 Flash Erase

Erase changes the value stored in all bits of the selected blocks to logic 1. Locked or disabled blocks cannot be erased. If multiple blocks are selected for erase during an erase sequence, the blocks are erased sequentially starting with the lowest numbered block and terminating with the highest. Aborting an erase operation leaves the flash core blocks being erased in an indeterminate data state. This can be recovered by executing an erase on the affected blocks.

The erase sequence consists of the following sequence of events:

1. Change the value in the FLASH_MCR[ERS] bit from 0 to a 1.
2. Select the block, or blocks to be erased by writing ones to the appropriate registers in FLASH_LMSR or FLASH_HSR. If the shadow block is to be erased, this step can be skipped, and FLASH_LMSR and FLASH_HSR are ignored. For shadow block erase, see section Section 13.4.2.5, “Flash Shadow Block” for more information.

NOTE

Lock and Select are independent. If a block is selected and locked, no erase occurs. Write to any address in flash. This is referred to as an erase interlock write.

3. Write a logic 1 to the FLASH_MCR[EHV] bit to start an internal erase sequence or skip to step 8 to terminate.
4. Wait until the FLASH_MCR[DONE] bit goes high.
5. Confirm FLASH_MCR[PEG] = 1.
6. Write a logic 0 to the FLASH_MCR[EHV] bit.
7. If more blocks are to be erased, return to step 2.
8. Write a logic 0 to the FLASH_MCR[ERS] bit to terminate the erase.

The erase sequence is presented graphically in Figure 13-16. The erase suspend operation detailed in Figure 13-16 is discussed in section Section 13.4.2.4.1, “Flash Erase Suspend/Resume.”

After setting FLASH_MCR[ERS], one write, referred to as an interlock write, must be performed before FLASH_MCR[EHV] can be set to a 1. Data words written during erase sequence interlock writes are ignored. You can terminate the erase sequence by clearing FLASH_MCR[ERS] before setting FLASH_MCR[EHV].

An erase operation can be aborted by clearing FLASH_MCR[EHV] assuming FLASH_MCR[DONE] is low, FLASH_MCR[EHV] is high and FLASH_MCR[ESUS] is low. An erase abort forces the module to step 7 of the erase sequence. An aborted erase results in FLASH_MCR[PEG] being set low, indicating a failed operation. The blocks being operated on before the abort contain indeterminate data. You cannot abort an erase sequence while in erase suspend.

WARNING

Aborting an erase operation leaves the flash core blocks being erased in an indeterminate data state. This can be recovered by executing an erase on the affected blocks.

13.4.2.4.1 Flash Erase Suspend/Resume

The erase sequence can be suspended to allow read access to the flash core. The erase sequence can also be suspended to program (erase-suspended program) the flash core. A program started during erase suspend can in turn be suspended. Only one erase suspend and one program suspend are allowed at a time during an operation. It is not possible to erase during an erase suspend, or program during a program suspend. During suspend, all reads to flash core locations targeted for program and blocks targeted for erase return indeterminate data. Programming locations in blocks targeted for erase during erase-suspended program can result in corrupted data.

An erase suspend operation is initiated by setting the FLASH_MCR[ESUS] bit. FLASH_MCR[ESUS] can be set to a 1 at any time when FLASH_MCR[ERS] and FLASH_MCR[EHV] are high and FLASH_MCR[PGM] is low. A 0 to 1 transition of FLASH_MCR[ESUS] causes the flash module to start the sequence which places it in erase suspend. You must wait until FLASH_MCR[DONE] = 1 before the module is suspended and further actions are attempted. After it is suspended, the array can be read or a program sequence can be initiated (erase-suspended program). Before initiating a program sequence you must first clear FLASH_MCR[EHV]. If a program sequence is initiated the value of the FLASH_MCR[PEAS] is not reset. These values are fixed at the time of the first interlock of the erase. Flash core reads while FLASH_MCR[ESUS] = 1 from the blocks being erased return indeterminate data.

The erase operation is resumed by clearing the FLASH_MCR[ESUS] bit. The flash continues the erase sequence from one of a set of predefined points. This can extend the time required for the erase operation.

WARNING

In an erase-suspended program, programming flash locations in blocks which were being operated on in the erase can corrupt flash core data.

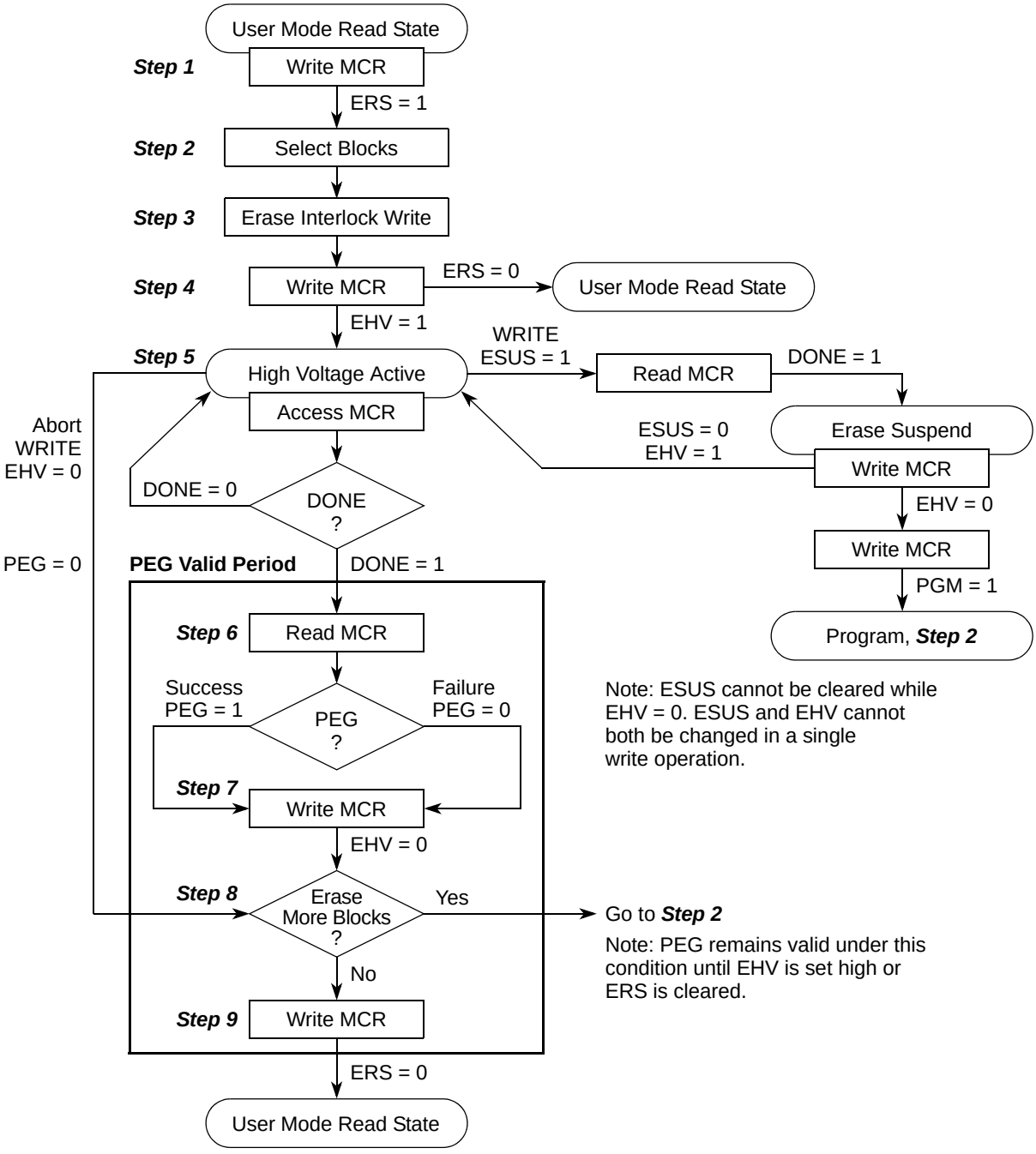


Figure 13-16. Erase Sequence

13.4.2.5 Flash Shadow Block

The flash shadow block is a memory-mapped block in the flash memory map. Program and erase of the shadow block are enabled only when FLASH_MCR[PEAS] = 1. After you has begun an erase operation on the shadow block, the operation cannot be suspended to program the main address space and vice-versa. You must terminate the shadow erase operation to program or erase the main address space.

NOTE

If an erase of user space is requested, and a suspend is done with attempts to erase suspend program shadow space, this attempted program is directed to user space as dictated by the state of FLASH_MCR[PEAS]. Likewise an attempted erase suspended program of user space, while the shadow space is being erased, is directed to shadow space as dictated by the state of FLASH_MCR[PEAS].

The shadow block cannot utilize the RWW feature. After an operation is started in the shadow block, a read cannot be done to the shadow block, or any other block. Likewise, after an operation is started in a block in low/mid/high address space, a read cannot be done in the shadow block.

The shadow block contains information on how the lock registers are reset. The first and second words can be used for reset configuration words. All other words can be used for user defined functions or other configuration words. The shadow block also contains information on how FLASH_BIUCR2 is reset.

The shadow block can be locked/unlocked against program or erase by using the FLASH_LMLR or FLASH_SLMLR discussed in Section 13.3.2, “Register Descriptions.”

Programming of the shadow block has similar restrictions to programming the array in terms of how ECC is calculated. See Section 13.4.2.3, “Flash Programming” for more information. Only one program is allowed per 64 bit ECC segment between erases. Erase of the shadow block is done similarly as an array erase. See section Section 13.4.2.4, “Flash Erase” for more information.

13.4.2.6 Censorship

Censorship logic disables access to internal flash based on the censorship control word value and the BOOTCFG[0:1] bits in the SIU_RSR. This prevents modification of the FLASH_BIUAPR bitfields associated with all masters except the core based on the censorship control word value, the BOOTCFG[0:1] bits in the SIU_RSR, and the EXTM bit in the EBI_MCR. Also, censorship logic sets the boot default value to external-with-external-master access disabled based on the value of the censorship control word and a TCU input signal.

208 Package: BOOTCFG[0] is not available due to pin limitations and internally asserted (driven to 0). The value of the censorship control word defaults to internal flash on the 208 package.

13.4.2.6.1 Censorship Control Word

The censorship control word is a 32-bit value located at the base address of the shadow block plus 0x1E0. The flash module latches the value of the control word prior to the negation of system reset. Censorship logic uses the value latched in the flash module to disable access to internal flash, disable the NDI, prevent modification of the FLASH_BIUAPR bitfields, and/or set the boot default value.

13.4.2.6.2 Flash Disable

Censorship logic disables read and write access to internal flash according to the logic presented in Table 13-18.

Table 13-18 shows the encoding of the BOOTCFG signals in conjunction with the value stored in the Censorship word in the shadow block of internal flash memory. The table also shows: the name of the boot mode; whether the internal flash memory is enabled or disabled; whether the Nexus port is enabled or disabled; whether the password downloaded in serial boot mode is compared with a fixed ‘public’ password or compared to a user programmable flash password.

Table 13-18. Flash Access Disable Logic

BOOTCFG[1] [0:1]	Censorship Control 0x00FF_FDE0 (Upper Half)	Serial Boot Control 0x00FF_FDE2 (Lower Half)	Boot Mode Name	Internal Flash State	Nexus State[2]	Serial Password
00	!0x55AA	Don't care	Internal – Censored	Enabled	Disabled	Flash
	0x55AA		Internal – Public	Enabled	Enabled	Public
01	Don't care	0x55AA	Serial – Flash Password	Enabled	Disabled	Flash
		!0x55AA	Serial – Public Password	Disabled	Enabled	Public
10	!0x55AA	Don't care	External – No Arbitration – Censored	Disabled	Enabled	Public
	0x55AA		External – No Arbitration -– Public	Enabled	Enabled	Public
11	!0x55AA	Don't care	External – External Arbitration – Censored	Disabled	Enabled	Public
	0x55AA		External – External Arbitration – Public	Enabled	Enabled	Public
’!’ = ’NOT’ (any value other than the value specified)						

1 BOOTCFG[0:1] bits are located in the SIU_RSR.
BOOTCFG[0] is not available on the 208 package and is internally asserted (driven to 0).

2 The Nexus port controller is held in reset when in censored mode.

The FBIU returns a bus error if an access is attempted while flash access is disabled. Flash access is any read, write or execute access.

13.4.2.6.3 FLASH_BIUAPR Modification

Censorship logic prevents modification of the access protection register (FLASH_BIUAPR) bit fields associated with all masters except the core according to the logic presented in Table 13-19.

Table 13-19. PFBAPR Modification Logic

BOOTCFG[1]		Censorship Control Word		EXTM[2]	PFBAPR Bitfields Writable
[0]	[1]	Upper Half	Lower Half		
0	0	0x55AA	0xXXXX	0	Yes
0	0	!0x55AA	0xXXXX	0	Yes
1	0	0x55AA	0xXXXX	0	Yes
1	0	!0x55AA	0xXXXX	0	Yes
1	1	0x55AA	0xXXXX	0	Yes
1	1	!0x55AA	0xXXXX	0	Yes
0	1	0xXXXX	0x55AA	0	Yes
0	1	0xXXXX	!0x55AA	0	Yes
0	0	0x55AA	0xXXXX	1	Yes
0	0	!0x55AA	0xXXXX	1	No
1	0	0x55AA	0xXXXX	1	Yes
1	0	!0x55AA	0xXXXX	1	No
1	1	0x55AA	0xXXXX	1	Yes
1	1	!0x55AA	0xXXXX	1	No
0	1	0xXXXX	0x55AA	1	No
0	1	0xXXXX	!0x55AA	1	No

1 BOOTCFG[0:1] bits are located in the SIU_RSR.

2 EXTM bit is located in the EBI_MCR.

13.4.2.6.4 External Boot Default

The SIU latches the boot default value in the SIU_RSR BOOTCFG[0:1] bits if and only if $\overline{\text{RSTCFG}}$ is negated. Censorship logic sets the boot default value before the SIU latches the value to external-with-external-master access disabled (EXTM=0) if the lower half of the censorship control word equals 0xFFFF or 0x0000. Otherwise, censorship logic sets the boot default value to internal flash.

208 Package: BOOTCFG[0] and $\overline{\text{RSTCFG}}$ are not available due to pin limitations. This signal is internally asserted (driven to 0) therefore, the device defaults to internal flash on the 208 package.

13.4.3 Flash Memory Array: Stop Mode

Stop mode is entered by setting the FLASH_MCR[STOP] bit. The FLASH_MCR[STOP] bit cannot be written when FLASH_MCR[PGM] = 1 or FLASH_MCR[ERS] = 1. In stop mode all DC current sources in the flash module are disabled. Stop mode is exited by clearing the FLASH_MCR[STOP] bit.

Accessing the flash memory array when STOP is asserted results in an error response from the flash BIU to the system bus. Memory array accesses must not be attempted until the flash transitions out of stop mode.

13.4.4 Flash Memory Array: Reset

A reset is the highest priority operation for the flash and terminates all other operations.

The flash uses reset to initialize register and status bits to their default reset values. If the flash is executing a program or erase operation and a reset is issued, the operation is aborted and the flash disables the high voltage logic without damage to the high voltage circuits. Reset aborts all operations and forces the flash into normal operating mode ready to receive accesses. FLASH_MCR[DONE] is set to 1 at the exit of reset.

After reset is negated, register accesses can be performed, even though registers that require updating from shadow information, or other inputs, cannot read updated values until flash exits reset. FLASH_MCR[DONE] can be polled to determine if reset has been exited.

Chapter 14
Internal Static RAM (SRAM)

14.1 Introduction

The SRAM provides 64 KB of general-purpose system SRAM. The first 32 KB of SRAM is powered by a separate power supply pin for standby operation. Figure 14-1 shows the internal SRAM block diagram.

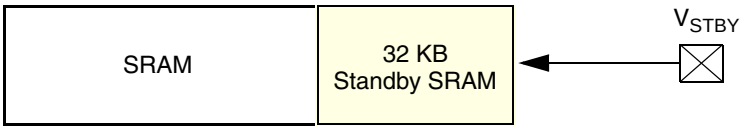


Figure 14-1. Internal SRAM Block Diagram

The SRAM controller has these features:

- Read/write accesses can map to SRAM from any master
- 32 KB with a separate power source for standby operation
- Byte, halfword, word, and doubleword addressable
- Single-bit correction and double-bit error detection

14.2 SRAM Operating Modes

Table 14-1 lists and describes the SRAM operating modes.

Table 14-1. SRAM Operating Modes

Mode	Description
Normal (functional)	Allows reads and writes of SRAM.
Standby	Preserves the 32 KB of standby memory when the V_{DD} (1.5 V) power drops below the level of V_{STBY} (0.8–1.2 V). Updates to standby SRAM are inhibited during system reset or during standby mode.

14.3 External Signal Description

The external signal for SRAM is the V_{STBY} RAM power supply. If the standby feature of the SRAM is not used, tie the V_{STBY} pin to V_{SS}.

14.4 Register Memory Map

The SRAM occupies 64 KB of memory starting at the base address as shown in Table 14-2.

Table 14-2. SRAM Memory Map

Address	Register Name	Register Description	Size
Base (0x4000_0000)	—	SRAM powered by V_{STBY}	32 KB
Base + 0x8000	—	32-KB RAM	32 KB

The internal SRAM has no registers. Registers for the SRAM ECC are located in the ECSM. See Chapter 8, "Error Correction Status Module (ECSM)," for more information.

14.5 Functional Description

ECC checks are performed during the read portion of an SRAM ECC read/write (R/W) operation, and ECC calculations are performed during the write portion of a read/write (R/W) operation. Because the ECC bits can contain random data after the device is powered on, you must initialize the SRAM by executing 32-bit write instructions to the entire SRAM. The platform RAM for the e200z3 is segmented on a 32-bit boundary, instead of the 64-bit organization for MPC5500 family members that are based on the e200z6. For software compatibility with other members of the MPC5500 family, use 64-bit writes to initialize the ECC bits of two 32-bit words simultaneously. For more information, see Section 14.7, "Initialization and Application Information."

14.6 SRAM ECC Mechanism

The SRAM ECC detects the following conditions and produces the following results:

- Detects and corrects all 1-bit errors
- Detects and flags all 2-bit errors as non-correctable errors
- Detects 39-bit reads (32-bit data bus + 7-bit ECC) that return all zeros or all ones, asserts an error indicator on the bus cycle, and sets the error flag

SRAM does not detect all errors greater than two bits. Internal SRAM writes are done on byte boundaries:

- 1 byte (0:7 bits)
- 2 bytes (0:15 bits)
- 4 bytes or 1 word (0:31 bits)
- 8 bytes or a doubleword (0:63 bits)

If the entire 32 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 32-bit data bus. The 7-bit ECC is appended to the data segment and written to SRAM. If the write operation is less than the entire 32-bit data width (1- or 2-byte segment), the following occurs:

1. The ECC mechanism checks the entire 32-bit data bus for errors, detecting and either correcting or flagging errors.
2. The write data bytes (1- or 2-byte segment) are merged with the corrected 64 bits on the data bus.
3. The ECC is then calculated on the resulting 64 bits formed in the previous step.

4. The 7-bit ECC result is appended to the 32 bits from the data bus, and the 39-bit value is then written to SRAM.

14.6.1 Access Timing

The system bus is a two-stage pipelined bus, which makes the timing of any access dependent on the access during the previous clock. Table 14-3 lists the various combinations of read and write operations to SRAM and the number of wait states used for the each operation. The table columns contain the following information:

- **Current operation** — Lists the type of SRAM operation executing currently
- **Previous operation** — Lists the valid types of SRAM operations that can precede the current SRAM operation (valid operation during the preceding clock)
- **Wait states** — Lists the number of wait states (bus clocks) the operation requires which depends on the combination of the current and previous operation

Table 14-3. Number of Wait States Required for RAM Operation

Current Operation	Previous Operation	Number of Wait States
Read	Idle	0
	Read	0
	32 or 64-bit write	0
	8 or 16-bit write	1
32 or 64-bit write	Idle	0
	Read	0
	32 or 64-bit write	0
	8 or 16-bit write	1
8 or 16-bit write	Idle	0
	Read	0
	32 or 64-bit write	0
	8 or 16-bit write	1

14.6.2 Reset Effects on SRAM Accesses

If a reset event asserts during a read or write operation to SRAM, the completion of that access depends on the cycle at which the reset occurs. Data read from or written to SRAM before the reset event occurred is retained, and no other address locations are accessed or changed.

14.7 Initialization and Application Information

To use the SRAM, the ECC must check all bits that require initialization after power on. Use a 64-bit write to each SRAM location to initialize the SRAM array as part of the application initialization code. All writes must specify an even number of registers performed on 64-bit word-aligned boundaries. If the write

is not the entire 64-bits (8-, 16-, or 32-bits), a read / modify / write operation is generated that checks the ECC value upon the read. See Section 14.6, “SRAM ECC Mechanism.”

NOTE

You *must* initialize SRAM, even if the application does not use ECC reporting.

14.7.1 Example Code

Because the ECC uses 64-bit based instructions, use 64-bit writes for this device, even though you must initialize the SRAM with 32-bit writes.

To initialize SRAM correctly, use a store multiple word **(stmw)** instruction to implement 64-bit writes to all SRAM locations. The **stmw** instruction concatenates two 32-bit registers to implement a single 64-bit write. To ensure the writes are 64-bits, specify an even number of registers and write on 64-bit word-aligned boundaries.

The following example code illustrates the use of the **stmw** instruction to initialize the SRAM ECC bits.

```
init_RAM:
lis      r11,0x4000        # base address of the SRAM, 64-bit word aligned
ori      r11,r11,0         # not needed for this address but could be for others
li       r12,512           # loop counter to get all of SRAM;
                           # 64k/4 bytes/32 GPRs = 512
mtctr    r12
init_ram_loop:
stmw     r0,0(r11)         # write all 32 GPRs to SRAM
addi     r11,r11,128       # inc the ram ptr; 32 GPRs * 4 bytes = 128
bdnz     init_ram_loop     # loop for 64k of SRAM
blr                        # done
```

Chapter 15
Boot Assist Module (BAM)

15.1 Introduction

This chapter describes the boot assist module (BAM).

15.1.1 Overview

The BAM contains the MCU boot program code. The BAM control block is connected to peripheral bridge B and occupies the last 16 KB of the MCU memory space. The BAM program supports the following booting modes:

- Internal flash
- External memory (324 package only; not available on the 208 package)
- Serial boot using an eSCI interface
- Serial boot using a FlexCAN interface

The BAM program is executed by the e200z3 core just after the MCU reset. Depending on the boot mode, the program initializes the minimum MCU resources to start application code execution.

Figure 15-1 is a block diagram of the BAM.

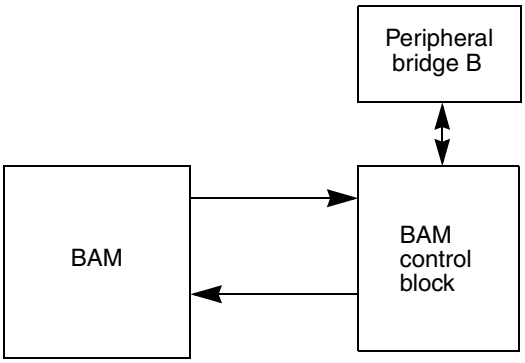


Figure 15-1. BAM Block Diagram

15.1.2 Features

The BAM program provides the following features:

- Initial e200z3 core MMU setup with minimum address translation for all internal MCU resources and external memory address space
- Locate and detect application boot code
- Automatic switch to serial boot mode if the flash is not initialized or is invalid:
  - Internal flash, or
  - External flash (324 package only; not available on the 208 package)
- Programmable 64-bit password protection for serial boot mode
- Boot application code from:
  - Internal flash module
  - External memory without arbitration (324 package only; not available on the 208 package)
- Serial boot can load the application boot code from a FlexCAN or eSCI bus into internal SRAM
- Censorship protection for internal flash module
- Watchdog timer enable option in the e200z3 core
- Configurable memory map for use with the legacy PowerPC Book E code or Freescale VLE code

15.1.3 Modes of Operation

15.1.3.1 Normal Mode

In normal operation, the BAM responds to all read requests within its address space. The BAM program is executed following the negation of reset.

15.1.3.2 Debug Mode

The BAM program is not executed when the MCU comes out of reset in OnCE debug mode. Use the development tool to configure and initialize the MCU before accessing the MCU resources.

15.1.3.3 Internal Boot Mode

Use internal boot mode to boot from internal flash memory. Configuration information, initialization, and boot code are kept in internal flash. If the application requires, the BAM program can complete the boot process before the application enables the external bus interface.

15.1.3.4 External Boot Modes

Use external boot mode for systems that have application code and configuration information in external memory connected by the EBI. Do not select external boot mode for devices without an external bus.

15.1.3.5 Serial Boot Mode

This mode of operation can load a user program into internal SRAM using either the eSCI or FlexCAN serial interface, then execute the downloaded program. The program can then control the downloading of data, as well as erasing and programming the internal or external flash memory.

Serial boot mode downloads:

- 64-bit password
- 32-bit start address
- 32-bit download consisting of 1-bit VLE flag (most significant bit)
  followed by a 31-bit length field containing the number of bytes to receive (download length)

Set the VLE flag to 1 for devices that support variable length encoding and must run in VLE mode. When the VLE flag is set, the BAM programs the external bus interface (EBI), RAM, and the flash memory map unit (MMU) TLB entries 1, 2, and 3 with the VLE attribute.

Clear the VLE bit to 0 for devices that use the PowerPC Book E or Power Architecture instruction set mode.

15.2 Memory Map

The BAM has 16 KB of memory, from 0xFFFF_C000 through 0xFFFF_FFFF, which is divided into four 4-KB segments, each containing a copy of the BAM program code. The BAM code resides in the 4-KB memory segment beginning at 0xFFFF_F000. A copy of the BAM code resides in the three preceding 4-KB segments, as shown in Table 15-1. The BAM program executes from the reset vector at address 0xFFFF_FFFC. Table 15-1 shows the addresses for the BAM code in the memory map.

Table 15-1. BAM Memory Map

Address	Description
0xFFFF_C000–0xFFFF_CFFF	BAM program mirrored
0xFFFF_D000–0xFFFF_DFFF	BAM program mirrored
0xFFFF_E000–0xFFFF_EFFF	BAM program mirrored
0xFFFF_F000–0xFFFF_FFFF	BAM program

15.3 Functional Description

15.3.1 BAM Program Resources

The BAM program initializes and uses the following MCU resources:

- BOOTCFG field in the reset status register (SIU_RSR) to determine the boot option
  - For devices using the 208 package, the BAM determines the value of BOOTCFG[1] only, since BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).
- Location and value of the reset configuration halfword (RCHW), which contains the address and configuration options for the boot code. See Chapter 4, “Reset,” for information about the RCHW.
- DISNEX bit in the SIU_CCR register to determine if the Nexus port is enabled

- MMU allows core access to MCU internal resources and the EBI
- EBI registers and external bus pads, when using external boot modes
- FlexCAN A, eSCI A and their pads, when using serial boot mode
- eDMA during serial boot mode

15.3.2 BAM Program Operation

The MCU core accesses the BAM after $\overline{\text{RSTOUT}}$ negates and before the application program executes.

The BAM program configures the e200z3 core MMU access for all MCU internal resources and external memory, according to Table 15-2. The memory map configuration remains the same for internal flash boot mode.

Table 15-2. MMU Configuration for Internal Flash Boot

TLB Entry	Region	Attributes	Logical Base Address	Physical Base Address	Size
0	Peripheral bridge B and BAM	• Guarded • Big endian • Global PID	0xFFF0_0000	0xFFF0_0000	1 MB
1	Internal flash	• Not guarded • Big endian • Global PID	0x0000_0000	0x0000_0000	16 MB
2	EBI	• Not guarded • Big endian • Global PID	0x2000_0000	0x2000_0000	16 MB
3	Internal SRAM	• Not guarded • Big endian • Global PID	0x4000_0000	0x4000_0000	256 KB

The MMU maps the logical addresses to the same physical addresses for all modules except for the external bus interface (EBI). The logical EBI addresses are mapped to physical addresses in internal flash memory. This allows code developed to run from external memory to run from internal flash memory.

The BAM program reads the following data and determines the boot mode for the boot sequence:

- BOOTCFG[0:1] located in the reset status register (SIU_RSR)
- Censorship control field located at 0x00FF_FDE0 in the shadow block of internal flash
- Serial boot control field located at 0x00FF_FDE2 in the shadow block of internal flash

The boot mode determines the following:

- Enables or disables internal flash memory
- Enables or disables the Nexus port
- Compares the password received in serial boot mode to a preset public password or a programmable password located in internal flash

Table 15-3 summarizes the different boot modes.

Table 15-3. Boot Modes

BOOTCFG [0:1]	Censorship Control 0x00FF_FDE0	Serial Boot Control 0x00FF_FDE2	Boot Mode Name	Internal Flash State	Nexus State	Serial Password
00	!0x55AA[1]	Any value	Internal—Censored	Enabled	Disabled	Flash
	0x55AA		Internal—Public	Enabled	Enabled	Public
01	Any value	0x55AA	Serial—Flash password	Enabled	Disabled	Flash
		!0x55AA	Serial—Public password	Disabled	Enabled	Public
10	!0x55AA	Any value	External—No arbitration—Censored	Disabled	Enabled	Public
	0x55AA		External—No arbitration—Public	Enabled	Enabled	Public
11	Invalid value					

1 '!' = 'NOT,' as in !0x55AA, means all values except 0x55AA. Do not use 0x0000 or 0xFFFF for the value of the censorship control or serial boot control words.

The 32-bit censorship word, which contains the censorship control and serial boot control fields, is read and interpreted during the boot process. Its value is used in conjunction with the BOOTCFG[0:1] values to enable or disable the internal flash memory and the Nexus interface. The censorship word is programmed at the factory to contain 0x55AA_55AA, which uses a password in internal flash to activate serial boot mode for an uncensored (public) device.

The censorship word starts at address 0x00FF_FDE0 and contains a 16-bit censorship control field [0:15] and a 16-bit serial boot control field [16:31]. The factory default settings are shown in Figure 15-2:

Address: 0x00FF_FDE0 Value: 0x55AA

MSB	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary value	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
Hex value	5				5				A				A			

Censorship control field–default value configures the device as uncensored.

Address: 0x00FF_FDE2 Value: 0x55AA

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Binary value	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
Hex value	5				5				A				A			

Serial boot control field–default value reads a password from internal flash.

Figure 15-2. Censorship Word

The BAM program reads the DISNEX bit to determine which of the following passwords to compare with the serial password received in serial boot mode:

- Public password (64-bit fixed value of 0xFEED_FACE_CAFE_BEEF); or
- Flash password (64-bit value in the shadow block of internal flash at address 0x00FF_FDD8).

Serial boot flash password starts at address 0x00FF_FDD8:

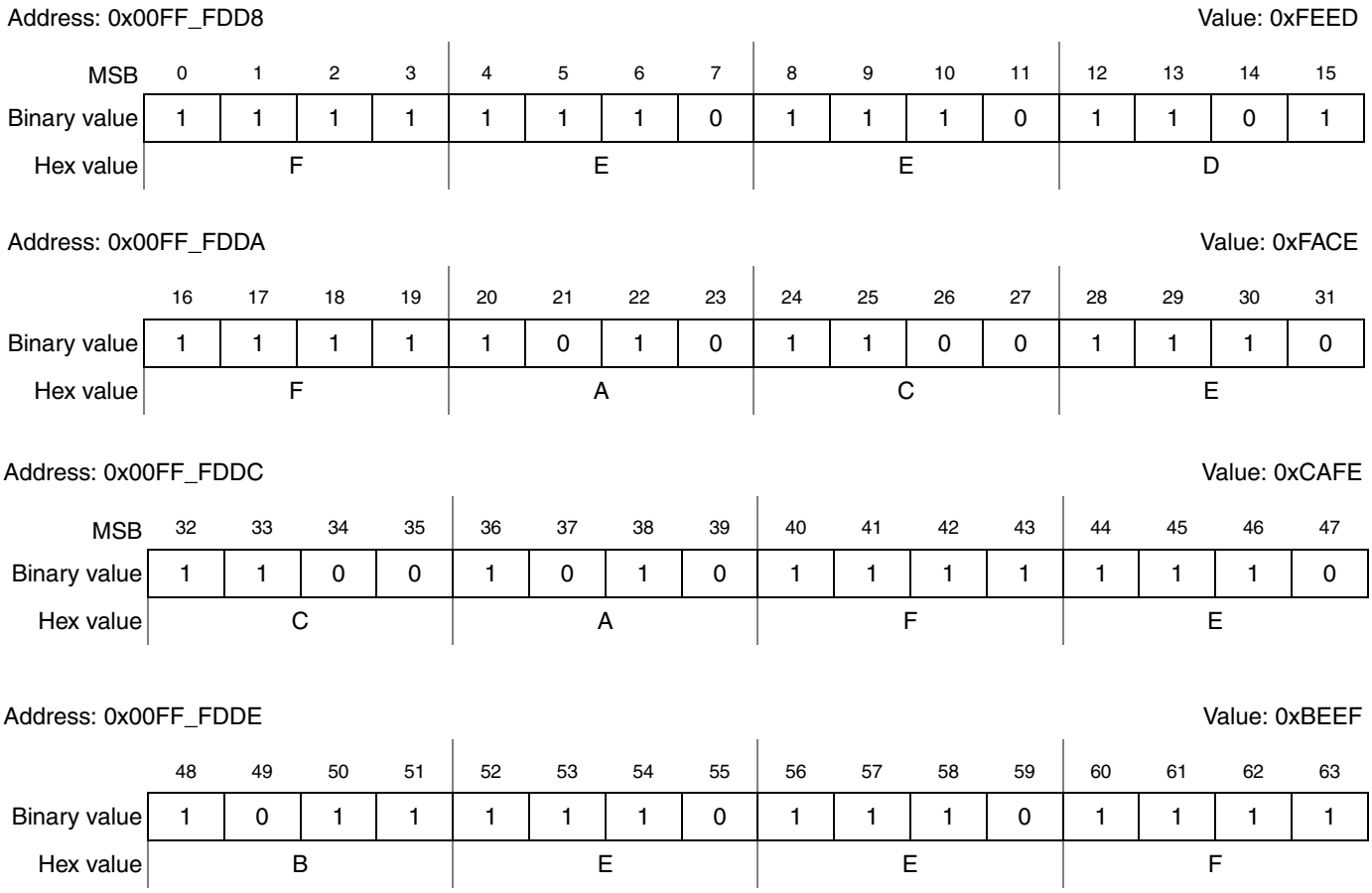


Figure 15-3. Serial Boot Flash Password

15.3.2.1 Boot Mode Features

The BAM program continues to initialize in one of the following boot modes:

- Internal boot mode from flash
- External boot mode from:
  - External memory without bus arbitration (324 package only; not available on the 208 package)
- Serial boot mode from one of the following interfaces:
  - eSCI interface
  - FlexCAN interface

15.3.2.2 Internal Boot Mode Flow

When the BAM detects internal flash boot mode, a bus error exception handler is set up to avoid bus errors and manage corrupt data from internal flash memory. The BAM program then reads up to six locations to find a valid reset configuration halfword (RCHW). When the RCHW is:

- *Valid*—BAM sets the e200z3 watchdog timer enable bit RCHW[WTE]
- *Invalid* —BAM uses the serial boot mode

15.3.2.2.1 Finding the Reset Configuration Halfword

The BAM searches internal flash memory for a valid reset configuration halfword (RCHW). A valid RCHW is a 16-bit value that contains a fixed 8-bit boot identifier and the reset configuration halfword (RCHW). The RCHW is the first halfword in one of the six low address flash blocks as shown in Table 15-4.

Table 15-4. Low Address Space (LAS) Block Memory Addresses

Block	Address
0	0x0000_0000
1	0x0000_4000
2	0x0001_0000
3	0x0001_C000
4	0x0002_0000
5	0x0003_0000

Read Section 4.4.3.5.1, “Reset Configuration Half Word (RCHW) Definition” for the definition and description of the RCHW.

If a valid RCHW value is located, the BAM:

1. Sets the watchdog timer enable bit (RCHW[WTE])
2. Fetches the reset vector from BOOT_BLOCK_ADDRESS + 0x0004
3. Branches to the reset boot vector

The application must have a valid instruction at the reset boot vector address.

If the BAM fails to find a valid RCHW, the validity of the flash memory is unreliable and the device defaults to serial boot operating mode.

BOOT_BLOCK_ADDRESS + 0x0000_0004

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31

Figure 15-4. Reset Boot Vector

If the watchdog timer is enabled for internal boot mode, the watchdog timeout is set to 2.5×2^{17} system clock cycles.

15.3.2.3 External Boot Modes Flow

Use external boot mode to boot application code from an external asynchronous memory device that is connected to the EBI. External boot mode is controlled by $\overline{CS}$[0].

15.3.2.3.1 External Boot MMU Configuration

As shown in Table 15-5, the BAM program sets up two MMU regions differently than in internal flash boot mode. The internal flash logical addresses are mapped to the physical addresses of the EBI.

Table 15-5. MMU Configuration for External Boot Mode

TLB Entry	Region	Attributes	Logical Base Address	Physical Base Address	Size
1	Internal flash memory	• Not guarded • Big endian • Global PID	0x0000_0000	0x2000_0000	16 MB
2	EBI	• Not guarded • Big endian • Global PID	0x2000_0000	0x2000_0000	16 MB

This allows code written to run from internal flash memory to execute from external memory.

15.3.2.3.2 Single Bus Master

Use External boot with no arbitration mode for single-master systems where the MCU is the only bus master, therefore no arbitration of the external bus is necessary. See Section 15.3.2.3.3, “Configure the EBI for External Boot—Single Master with no Arbitration.”

The boot modes are specified by the BOOTCFG[0:1] value.

208 Package: BOOTCFG[0] is not available, and is internally asserted (driven to 0), therefore the 208 package is limited to boot from internal flash memory or a serial port.

15.3.2.3.3 Configure the EBI for External Boot—Single Master with no Arbitration

The BAM program configures:

- Chip select $\overline{CS}$[0] as a 16-bit port starting at base address 0x2000_0000 with:
  - no burst
  - 15 wait states
  - 8 MB
- EBI for no external master (clears EXTM bit)
- Enables the EBI for normal operation
- Configures the following EBI signals:
  - ADDR[8:31]
  - DATA[0:15]
  - $\overline{WE}$[0]
  - $\overline{OE}$
  - $\overline{TS}$
  - $\overline{CS}$[0]

15.3.2.3.4 Read the Reset Configuration Halfword

The BAM program reads the first location in external memory (i.e. address 0x2000_0000) for a valid reset configuration halfword (RCHW). If the BAM program finds a valid RCHW, the following occurs:

1. The $\overline{CS}$[0] port size and data pins are configured according to the RCHW[PS0] bit
2. The e200z3 core watchdog timeout is enabled or disabled using the RCHW[WTE] bit. The watchdog timeout interval is 2.5×2^{17} system clock periods when using the RCHW.
3. MMU boots using PowerPC Book E code or Freescale VLE code according to the RCHW[VLE] setting.

The BAM program then reads the reset vector from the address 0x2000_0004 and branches to that reset vector address, starting application program execution. See Figure 15-4 for more information.

15.3.2.4 Serial Boot Mode Operation

Serial boot operating mode configures:

- FlexCAN A and the eSCI A GPIO signals
- Unused message buffers in FlexCAN A are designated as scratch pad SRAM
- Flash memory map unit (MMU) TLB entries
- Watchdog timer is enabled and set to 2.5×2^{27} system clock cycles

Serial boot mode downloads:

- 64-bit password
- 32-bit start address
- 32-bit download consisting of a 1-bit VLE flag followed by a 31-bit length field

Set the VLE flag to 1 for devices that support variable length encoding and must run in VLE mode. When the VLE flag is set, the BAM configures these components:

- External bus interface (EBI) signals and port size
- SRAM structure
- VLE attribute is set in flash memory map unit (MMU) TLB entries 1, 2, and 3

Clear the VLE bit to 0 for devices that use the PowerPC Book E or Power Architecture instruction set mode.

15.3.2.4.1 Serial Boot Mode MMU and EBI Configuration

The BAM program sets up the MMU for all peripheral and memory regions in one of two different modes and sets up the EBI in one of three different modes; depending on how serial boot mode was entered.

If serial boot mode is entered directly by choosing the mode with the BOOTCFG signals, or was entered indirectly from internal boot mode because no valid RCHW was found, then the MMU is configured the same way as for internal boot mode. The EBI is disabled and all bus pins function as GPIO.

If serial boot mode is entered indirectly from external boot/single-master because no valid RCHW was found, then the MMU and EBI are configured the for an external boot mode with a 16-bit data bus.

See Table 15-3 and Table 15-5 for more information.

15.3.2.4.2 Serial Boot Mode FlexCAN and eSCI Configuration

In serial boot mode, the BAM program configures FlexCAN A and eSCI A to receive messages. The CNRXA and RXDA signals are configured as inputs to the FlexCAN and eSCI modules. The CNTXA signal is configured as an output from the FlexCAN module. The TXDA signal of the eSCI A remains configured as GPIO input. The BAM program writes 0x0000_0000_0000_0000 to the e200z3 core timebase registers (TB), enables the e200z3 core setting the watchdog timer to use 2.5×2^{27} system clock cycles before a reset occurs.

See Table 15-6 for examples of time out periods.

In serial boot mode the FlexCAN controller is configured to operate at a baud (bit) rate equal to the system clock frequency divided by 60 with one message buffer (MB) using the standard 11-bit identifier format detailed in the CAN 2.0A specification.

See Section 21.4.5.4, “Protocol Timing,” for information on FlexCAN bit rate generation.

Coming out of reset, the default system clock is 1.5 times the crystal frequency. The baud rate with PLL enabled is equal to the crystal frequency divided by 40.

See Chapter 11, “Frequency Modulated Phase Locked Loop and System Clocks (FMPLL),” for more information. Table 15-6 shows FlexCAN operation at reset.

Table 15-6. BAM FlexCAN Frequency at Reset (FMPLL Enabled out of Reset)

FMPLL Clock Mode	System Clock Frequency (f_{sys}) after Reset	Serial Boot Mode Frequency[1] (FlexCAN Baud Rate)
Crystal reference mode or External reference mode	1.5 x crystal reference frequency ($f_{ref\_crystal}$) [2]	Crystal reference frequency ÷ 40
Dual controller mode	2 x EXTCLK	EXTCLK ÷ 30

1 Serial boot mode frequency is set in software as the system clock frequency divided by 60.

2 Crystal reference frequency is set at 8–20 MHz.

The BAM ignores the following errors:

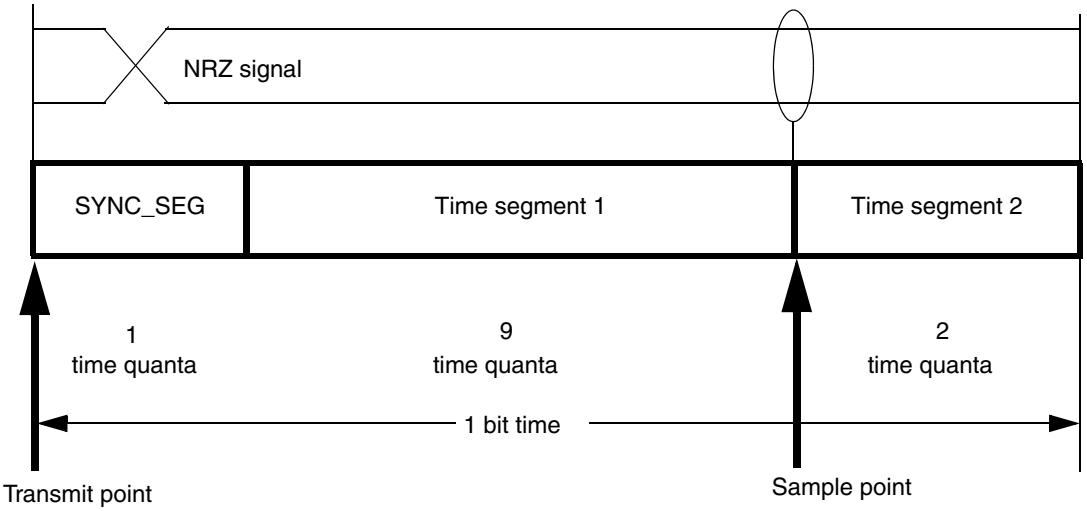
- Bit 1 errors
- Bit 0 errors
- Acknowledge errors
- Cyclic redundancy code errors
- Form errors
- Stuffing errors
- Transmit error counter errors
- Receive error counter errors

All data received is accepted as valid and is echoed out on the CNTXA signal.

NOTE

The host computer must compare the ‘echoed data’ to the sent data and restart the process if an error is detected.

See Figure 15-5 for details of FlexCAN bit timing.



1 time quanta = 5 system clock periods = 3.3 crystal clock periods (with PLL enabled)

Figure 15-5. FlexCAN Bit Timing

The eSCI is configured for one start bit, eight data bits, no parity and one stop bit and to operate at a baud rate equal to the system clock divided by 1250. See Table 15-7 for examples of baud rates.

The BAM ignores the following eSCI errors:

- Overrun errors
- Noise errors
- Framing errors
- Parity errors

All data received is accepted as valid and is echoed out on the TXD signal. The host computer is responsible for comparing the echoes with the sent data, and restarting the process if an error is detected.

Table 15-7. Serial Boot Mode—Baud Rate and Watchdog Summary

Crystal Frequency (MHz)	System Clock Frequency (MHz)	SCI Baud Rate	FlexCAN Baud Rate	Watchdog Timeout Period (seconds)
$f_{ref\_crystal}$	$f_{sys} = 1.5 \times f_{ref\_crystal}$	$f_{sys} \div 1250$	$f_{sys} \div 60$	$(2.5 \times 2^{27}) \div f_{sys}$
8	12	9600	200 K	28.0
12	18	14400	300 K	18.6
16	24	19200	400 K	14.0
20	30	24000	500 K	11.2

Upon receiving a valid FlexCAN message with an ID equal to 0x0011 that contains eight data bytes, or a valid eSCI message, the BAM uses a serial boot submode: FlexCAN serial boot mode, or eSCI serial boot mode.

In FlexCAN serial boot mode, the eSCI A signal RXDA reverts to GPIO input. All data is downloaded on the FlexCAN bus and eSCI messages are ignored.

In eSCI serial boot mode, the FlexCAN A signals CNRXA and CNTXA revert to GPIO inputs and the TXDA signal is configured as an output. All data is downloaded on the eSCI bus and FlexCAN messages are ignored.

Table 15-8. FlexCAN and eSCI Reset Configuration for FlexCAN and eSCI Boot

Pin Label	Reset Function	Initial Serial Boot Mode	Serial Boot Mode after Receiving a Valid	
			FlexCAN Message	eSCI Message
CNTXA	GPIO	CNTXA	CNTXA	GPIO
CNRXA	GPIO	CNRXA	CNRXA	GPIO
TXDA	GPIO	GPIO	GPIO	TXDA
RXDA	GPIO	RXDA	GPIO	RXDA

Table 15-9. FlexCAN and eSCI Reset Pin Configuration

Pins	I/O	Weak Pullup State	Hysteresis	Driver Configuration	Slew Rate	Input Buffer Enable
CNTXA_TXDA	Output	Enabled Up	—	Push/pull	Medium	N
CNRXA_RXDA	Input	Enabled Up	Y	—	—	—
GPIO	Input	Enabled Up	Y	—	—	—

15.3.2.4.3 Download Process for FlexCAN Serial Boot Mode

The download process contains the following steps:

1. Download the 64-bit password.
2. Download the start address, VLE flag, and the number of data bytes to download.
3. Download the data.
4. Execute the boot code from the start address.

Each step of the process must complete before the next step starts.

1. Download the 64-bit password.

   The host computer must send a FlexCAN message with ID = 0x011 that contains the 64-bit serial download password. FlexCAN messages with other IDs or fewer bytes of data are ignored. When a valid message is received, the BAM transmits a FlexCAN message with ID = 0x001 that contains the data received. The host must not send a second FlexCAN message until it receives the echo of the first message. A FlexCAN message sent before the echo is received is ignored.

   The received 64-bit password is validated to ensure that none of the four 16-bit halfwords have a value of 0x0000 or 0xFFFF, which are invalid passwords. A valid password must have at least one 0 and one 1 in each halfword lane.

   The BAM program then reads the disable Nexus bit [DISNEX] in the SIU_CCR register to determine the censorship status of the MCU. If Nexus is disabled, the MCU is censored and the password is compared to a password stored in the shadow block in internal flash memory.

   If Nexus is enabled, the MCU is not censored or booting from external flash and the password is compared to the constant value = 0xFEED_FACE_CAFE_BEEF.

   If the password fails any validity tests, the MCU stops responding to all stimulus. To repeat boot operation, the MCU needs to be reset by external reset or by watchdog. If the password is valid, the BAM program refreshes the e200z3 watchdog timer and the next step in the protocol can be performed.

2. Download the start address, VLE bit, and the download size.

   The host computer must send a FlexCAN message with an ID = 0x012 that contains:

   — 32-bit start address in internal SRAM indicating where to store the succeeding data in memory;

   — 32-bit number containing a 1-bit variable length encoded (VLE) flag followed by a 31-bit length field that contains the number of data bytes to receive and store in memory before switching to execute the code just loaded.

The start address is expected on a 32-bit word boundary, therefore the least significant 2 bits of the address are ignored. FlexCAN messages with other IDs or fewer data bytes are ignored.

Set the VLE bit in the serial download data (most significant bit in the LENGTH word) if the code to download uses VLE instructions.

When a valid message is received, the BAM transmits a FlexCAN message with an ID = 0x002 that contains the received data. The host computer must not send another FlexCAN message until it receives the echo from the previous message. A FlexCAN message sent before the echo is received is ignored.

3. Download the data.

The host computer must send a succession of FlexCAN messages with ID = 0x013 that contains raw binary data (the data length is variable). Each data byte received is stored in MCU memory, starting at the address specified in the previous step and incrementing through memory until the number of data bytes received and stored in memory matches the number specified in the previous step. FlexCAN messages with ID values other than 0x013 are ignored.

When a valid message is received, the BAM transmits a FlexCAN message with an ID = 0x003 that contains the data received. The host computer must not send another FlexCAN message until it receives the echo from the previous message. A FlexCAN message sent before the echo is received is ignored.

NOTE

Internal SRAM is protected by 64-bit error correction (ECC) hardware. All writes to uninitialized internal SRAM must be 64-bits wide, or an ECC error occurs. The BAM buffers 8 bytes of downloaded data before executing a single 64-bit write. Only internal SRAM supports 64-bit writes. Downloading data to other RAM causes errors.

If the start address of the downloaded data is not at an 8-byte boundary, the BAM writes 0x00 to the memory locations from the preceding 8-byte boundary to the start address (maximum 4 bytes). The BAM also writes 0x00 to all memory locations from the last byte of data downloaded to the following 8-byte boundary (maximum 7 bytes).

4. Execute code.

The BAM waits for the last FlexCAN message transmission to complete. Then the FlexCAN controller is disabled. CNTXA and CNRXA become GPIO inputs. The BAM branches to the starting address of the downloaded code, as specified in step 2.

NOTE

The code that downloads and executes must:

- Periodically refresh the e200z3 watchdog timer; or
- Change the timeout period to a value that does not cause resets during normal operation.

Table 15-10. FlexCAN Serial Boot Mode Download Process

Step	Host Message Sent	MCU Response Message	Action
1	FlexCAN ID = 0x011 + 64-bit password	FlexCAN ID = 0x001 + 64-bit password	Password checked for validity and compared against stored password. e200z3 watchdog timer is refreshed if the password check is successful.
2	FlexCAN ID = 0x012 + 32-bit store address + 32-bit number of bytes	FlexCAN ID = 0x002 + 32-bit store address + 32-bit number of bytes	The load address and the number of bytes to download are stored for future use.
3	FlexCAN ID = 0x013 + 8 to 64 bits of raw binary data	FlexCAN ID = 0x003 + 8 to 64 bits of raw binary data	Each data byte received is written to MCU memory, starting at the address specified in the previous step and incrementing until the number of data bytes received and stored matches the number of bytes to download and store (specified in step 2).
4	None	None	The BAM program returns I/O pins and the FlexCAN module to their reset state, then branches to the start address of the stored data (specified in step 2).

15.3.2.4.4 eSCI Serial Boot Mode Download Process

The eSCI serial boot mode download process contains the following steps:

1. Download the 64-bit password.
2. Download the start address, VLE flag, and the number of data bytes to download.
3. Download the data.
4. Execute the code from start address.

Each step in the following process must complete before the next step starts. The eSCI operates in half duplex mode where the host sends a byte of data, then waits for the echo back from the MCU before proceeding with the next byte. Bytes sent from the host before the previous echo from the MCU is received are ignored.

1. Download the 64-bit password.

   The first 8 bytes of eSCI data the host computer sends must contain the 64-bit serial download password. For each valid eSCI message received, the BAM transmits the same data on the eSCI A TXDA signal.

   The received 64-bit password is checked for validity. It is checked to ensure that none of the 4 x 16-bit halfwords are illegal passwords (0x0000 or 0xFFFF). A password must have at least one 0 and one 1 in each halfword to qualify as legal.

   The BAM program then checks the censorship status of the MCU by checking the DISNEX bit in the SIU_CCR. If Nexus is disabled, the MCU is considered censored and the password is compared with a password stored in the shadow block of internal flash memory.

   If Nexus is enabled, the MCU is not censored or is booting from external flash and the password is compared to the constant value of 0xFEED_FACE_CAFE_BEEF.

   If the password fails a validity test, the MCU stops responding to all stimulus. To repeat the boot operation, assert the $\overline{\text{RESET}}$ signal or wait for the watchdog timer to reset the MCU. If the password is valid, the BAM refreshes the e200z3 watchdog timer and proceeds to step 2.

2. Download the start address, VLE bit, and the download size.

   The host computer must send the next eight bytes of eSCI data that contains:

   — 32-bit start address in internal SRAM indicating where to store the succeeding data in memory;

   — 32-bit number containing a 1-bit variable length encoded (VLE) flag followed by a 31-bit length field that contains the number of data bytes to receive and store in memory before switching to execute the code just loaded.

   The start address is normally located on a word boundary (4-bytes), therefore the least significant 2 bits of the address are ignored. For each valid eSCI message received, the BAM transmits the same data on the eSCI A TXDA signal.

   Set the VLE bit in the serial download data (most significant bit in the LENGTH word) if the code to download uses VLE instructions.

3. Download the data.

   The host computer must then send a succession of eSCI messages, each containing raw binary data. Each byte of data received is stored in the MCU's memory, starting at the address specified in the previous step and incrementing through memory until the number of data bytes received and stored in memory matches the number specified in the previous step. For each valid eSCI message received, the BAM transmits the same data on the TXDA signal.

NOTE

Internal SRAM is protected by 64-bit wide error correction coding hardware (ECC). All writes to uninitialized internal SRAM must be 64 bits wide, or an ECC error occurs. The BAM buffers download data until 8-bytes are received, and then executes a single 64-bit wide write. Only internal SRAM supports 64-bit writes. Downloading data to RAM other than internal SRAM causes errors.

If the start address of the downloaded data is not on an 8-byte boundary, the BAM writes 0x00 beginning at the preceding 8-byte boundary memory location to the start address (4 byte maximum). The BAM also writes 0x00 to all memory locations from the last data byte downloaded to the following 8-byte boundary (7 byte maximum).

4. Execute the code.

   The BAM waits for the last eSCI message transmission to complete and then disables the eSCI. TXDA and RXDA revert to general-purpose inputs. The BAM branches to the starting address where the downloaded code is stored (specified in step 2) and executes the code.

NOTE

The code that downloads and executes must periodically refresh the e200z3 watchdog timer or change the timeout period to a value that does not cause resets during normal operation.

Table 15-11. eSCI Serial Boot Mode Download Process

Step	Host Sent Message	BAM Response Message	Action
1	64-bit password MSB first	64-bit password	Password checked for validity and compared against stored password. e200z3 watchdog timer is refreshed if the password check is successful
2	32-bit store address + 32-bit number of bytes MSB first	32-bit store address + 32-bit number of bytes	Load address and size of download are stored for future use
3	8 bits of raw binary data	8 bits of raw binary data	Each byte of data received is stored in MCU memory, starting at the address specified in the previous step and incrementing until the number of data bytes received and stored matches the number of data bytes specified in the preceding step.
4	None	None	The BAM returns I/O pins and the eSCI module to their reset state, except it asserts ESCI_A_CR2[MDIS] instead of negates. Then the BAM branches to the starting address of the stored data specified in step 2.

15.3.3 Interrupts

No interrupts are generated or enabled by the BAM.

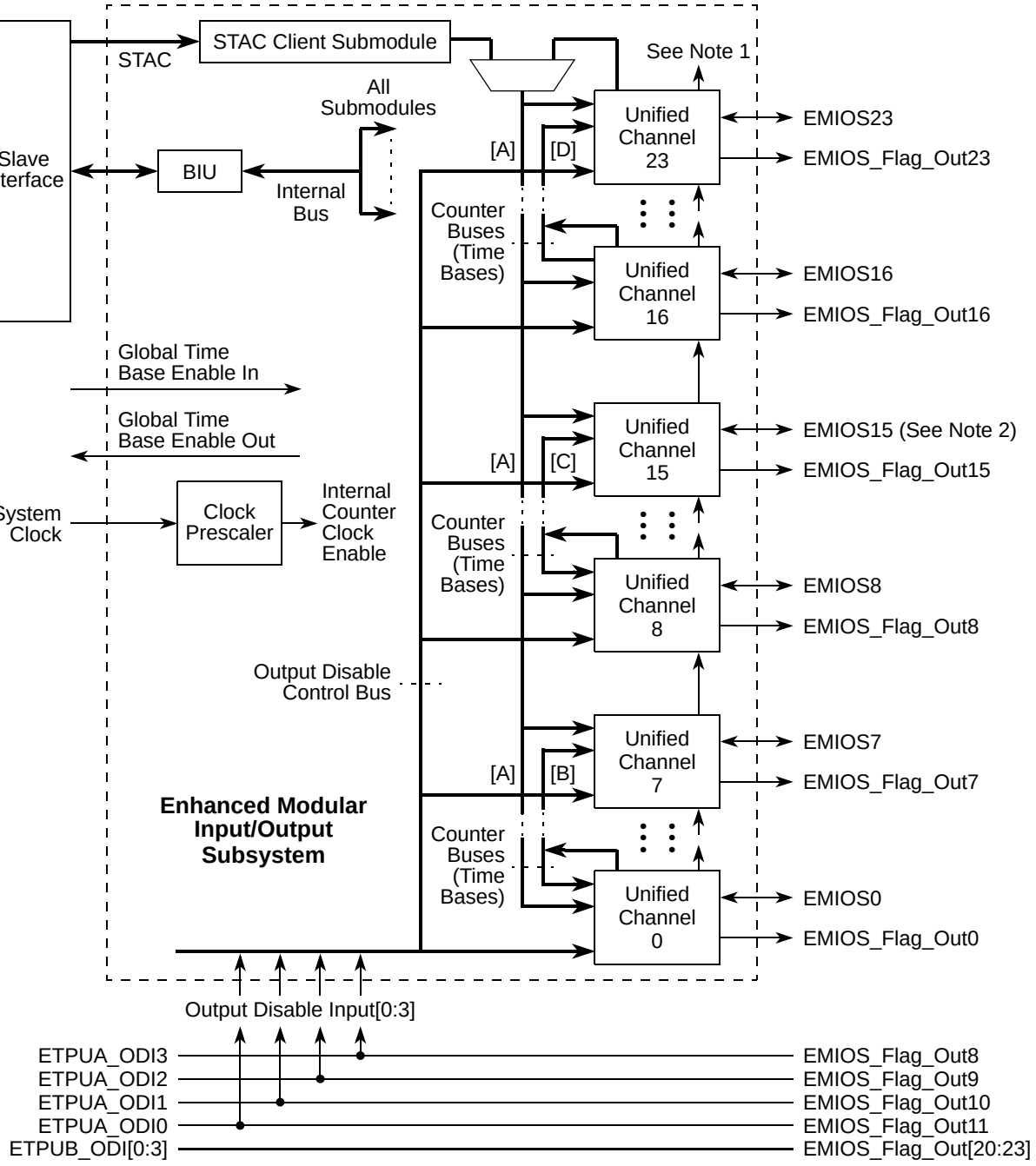
Chapter 16
Enhanced Modular Input/Output Subsystem (eMIOS)

16.1 Introduction

This chapter describes the enhanced modular input/output subsystem (eMIOS), which provides functionality to generate or measure timed events.

16.1.1 Block Diagram

Figure 16-1 shows the block diagram of the eMIOS.



Note 1: Connection between UC[n-1] and UC*n* necessary to implement QDEC mode.
Note 2: On channels 12–15, there is no input from EMIOS[12:15], but only from the DSPI module.

Figure 16-1. eMIOS Block Diagram

16.1.2 Overview

The eMIOS builds on the MIOS concept by using a unified channel (UC) module that provides a superset of the functionality of all the individual MIOS channels, while providing a consistent user interface. This allows for more flexibility because each unified channel can be programmed for different functions.

16.1.3 Features

- 24 unified channels
- Unified channels features
  - 24-bit registers for captured/match values
  - 24-bit internal counter
  - Internal prescaler
  - Dedicated output pin for buffer direction control
  - Selectable time base
  - Can generate its own time base
- Four 24-bit wide counter buses
  - Counter bus A can be driven by unified channel 23 or by the STAC bus.
  - Counter bus B, C, and D are driven by unified channels 0, 8, and 16, respectively.
  - Counter bus A can be shared among all unified channels (UC). UCs 0–7, 8–15, and 16–23 can share counter buses B, C, and D, respectively.
- One global prescaler
- Shared time bases through the counter buses
- Synchronization among internal and external time bases
- Shadow FLAG register
- State of module can be frozen for debug purposes
- DMA request capability for some channels
- Motor control capability

16.1.4 Modes of Operation

16.1.4.1 eMIOS Modes

The eMIOS operates in one of the following modes:

- User mode—The normal operating mode. When EMIOS_MCR[FRZ] = 0, and EMIOS_CCR[FREN] = 0, the eMIOS is in user mode.
- Debug mode—Debug mode is programmed individually for each channel. When entering debug mode, the values in the UC registers' are frozen, but remain available to the debugger for read and write accesses. After leaving debug mode, the values in the UC register counters prior to entering debug mode are restored before resuming operations.

In debug mode, all clocks are running and all registers are accessible for use during software debugging; there is no power saving during debug mode.

- Freeze mode—Freeze mode enables the eMIOS to freeze the value in the unified channels registers' when debug mode is requested at the MCU level. While in freeze mode, the eMIOS continues to operate to allow the MCU access to the unified channels' registers. The unified channel values remain frozen until one of these events occurs:
  - EMIOS_MCR[FRZ] bit is written to zero
  - MCU exits debug mode
  - Unified channel's EMIOS_CCR[FREN] bit is cleared

  In freeze mode, all clocks are running and all registers are accessible for use debugging software; there is no power saving during freeze mode.

16.1.4.2 Unified Channel Modes

The unified channels can be configured to operate in the following modes:

Table 16-1. Unified Channel Modes

Mode
General purpose input/output
Single action input capture
Single action output compare
Input pulse width measurement
Input period measurement
Double action output compare
Pulse/edge accumulation
Pulse/edge counting
Quadrature decode
Windowed programmable time accumulation
Modulus counter, normal
Modulus counter, buffered
Output pulse width and frequency modulation, normal
Output pulse width and frequency modulation, buffered
Center aligned output pulse width modulation with dead time insertion, normal
Center aligned output pulse width modulation with dead time insertion, buffered
Output pulse width modulation, normal
Output pulse width modulation, buffered

These modes are described in Section 16.4, "Functional Description."

16.2 External Signal Description

16.2.1 Overview

Each unified channel has one input and one output signal connected to the channel's I/O pin. See the SIU, eTPU, and DSPI sections for details about connecting eMIOS to pads and other modules.

NOTE

Channels 12–15 are input for data from the DSPI module.
Channels 12–15 cannot be input from eMIOS[12:15] because these are not pinned out. See Figure 2-1.

The internal *output disable input* signals 0–3 (see Table 16-3), are provided to implement the output disable feature needed for motor control. They are connected to EMIOS_Flag_Out signals according to Section 16.2.1.2, "Output Disable Input—eMIOS Output Disable Input Signals."

16.2.1.1 External Signals

When configured as an input, EMIOS*n* is synchronized and filtered by the programmable input filter (PIF). The output of the PIF is then used by the channel logic, and can be read by the MCU through the UCIN bit of the EMIOS_CSR*n*.

When configured as an output, EMIOS*n* is a registered output and can be read by the MCU through the UCOUT bit of the EMIOS_CSR*n*.

Table 16-2. External Signals

Signal	Direction	Function	Reset State
EMIOS[0:11, 16:23]	Input	eMIOS Unified Channel n input	—
EMIOS[12:15]	Input	From DSPI	—
EMIOS[0:23]	Output	eMIOS Unified Channel n output	0 / Hi-Z[1]

[1] A value of 0 refers to the reset value of the signal. Hi-Z refers to the state of the external pin if a tri-state output buffer is controlled by the corresponding eMIOS signal.

16.2.1.2 Output Disable Input—eMIOS Output Disable Input Signals

Output disable inputs to both the eMIOS and the eTPU modules are connected to EMIOS_Flag_Out*n* signals according to Table 16-3.

Table 16-3. eMIOS Output Disable Input Signals

eMIOS Channel[1]	eMIOS Output Disable Input Signal[2]	eTPU Output Disable Input Signal[3]
EMIOS_Flag_Out8	output disable input 3	**ETPUA_ODI3**
EMIOS_Flag_Out9	output disable input 2	ETPUA_ODI2

Table 16-3. eMIOS Output Disable Input Signals (continued)

eMIOS Channel[1]	eMIOS Output Disable Input Signal[2]	eTPU Output Disable Input Signal[3]
EMIOS_Flag_Out10	output disable input 1	ETPUA_ODI1
EMIOS_Flag_Out11	output disable input 0	ETPUA_ODI0
EMIOS_Flag_Out20	—	ETPUB_ODI0
EMIOS_Flag_Out21	—	ETPUB_ODI1
EMIOS_Flag_Out22	—	ETPUB_ODI2
EMIOS_Flag_Out23	—	ETPUB_ODI3

1 All other EMIOS_Flag_Out*n* output signals are not connected.

2 Each of the four internal eMIOS *output disable input* signals can be programmed to disable the output of any eMIOS channel if that channel has selected output disable capability by the setting of its EMIOS_CCR*n*[ODIS] bit, and by specifying the output disable input in its EMIOS_CCRn[ODISSL] field.

3 ETPU*x*_ODI*y* input signals disable outputs for eTPU engine *x*, channels (*y**8) through (*y**8+7). See the ETPU chapter for more details.

16.3 Memory Map and Register Definitions

Addresses of unified channel (UC) registers are specified as offsets from the channel's base address, otherwise the eMIOS base address is used as reference.

The overall address map organization is shown in Table 16-4. Table 16-5 describes the unified channel registers. All registers are cleared on reset.

Table 16-4. eMIOS Memory Map

Address	Register Name	Register Description	Bits
Base (0xC3FA_0000)	EMIOS_MCR	Module Configuration Register	32
Base + 0x0004	EMIOS_GFR	Global Flag Register	32
Base + 0x0008	EMIOS_OUDR	Output Update Disable Register	32
Base + (0x000C–0x001F)	—	Reserved	—
Base + 0x0020	UC0	Unified Channel 0 Registers	256
Base + 0x0040	UC1	Unified Channel 1 Registers	256
Base + 0x0060	UC2	Unified Channel 2 Registers	256
Base + 0x0080	UC3	Unified Channel 3 Registers	256
Base + 0x00A0	UC4	Unified Channel 4 Registers	256
Base + 0x00C0	UC5	Unified Channel 5 Registers	256
Base + 0x00E0	UC6	Unified Channel 6 Registers	256
Base + 0x0100	UC7	Unified Channel 7 Registers	256

Table 16-4. eMIOS Memory Map (continued)

Address	Register Name	Register Description	Bits
Base + 0x0120	UC8	Unified Channel 8 Registers	256
Base + 0x0140	UC9	Unified Channel 9 Registers	256
Base + 0x0160	UC10	Unified Channel 10 Registers	256
Base + 0x0180	UC11	Unified Channel 11 Registers	256
Base + 0x01A0	UC12	Unified Channel 12 Registers	256
Base + 0x01C0	UC13	Unified Channel 13 Registers	256
Base + 0x01E0	UC14	Unified Channel 14 Registers	256
Base + 0x0200	UC15	Unified Channel 15 Registers	256
Base + 0x0220	UC16	Unified Channel 16 Registers	256
Base + 0x0240	UC17	Unified Channel 17 Registers	256
Base + 0x0260	UC18	Unified Channel 18 Registers	256
Base + 0x0280	UC19	Unified Channel 19 Registers	256
Base + 0x02A0	UC20	Unified Channel 20 Registers	256
Base + 0x02C0	UC21	Unified Channel 21 Registers	256
Base + 0x02E0	UC22	Unified Channel 22 Registers	256
Base + 0x0300	UC23	Unified Channel 23 Registers	256

Table 16-5. UC Memory Map

Address	Register Name	Register Description	Bits
UC*n* Base + 0x0000	EMIOS_CADR*n*	Channel A Data Register	32
UC*n* Base + 0x0004	EMIOS_CBDR*n*	Channel B Data Register	32
UC*n* Base + 0x0008	EMIOS_CCNTR*n*	Channel Counter Register	32
UC*n* Base + 0x000C	EMIOS_CCR*n*	Channel Control Register	32
UC*n* Base + 0x0010	EMIOS_CSR*n*	Channel Status Register	32
UC*n* Base + (0x0014–0x001F)	—	Reserved	—

16.3.1 Register Description

All registers are 32-bit wide. This section illustrates the eMIOS with 24 unified channels supporting 24-bit wide data.

16.3.1.1 eMIOS Module Configuration Register EMIOS_MCR

EMIOS_MCR contains global control bits for the eMIOS module.

Address: Base + 0x0000 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MDIS	FRZ	GTBE	ETB	GPREN	0	0	0	0	0	0	SRV			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPRE								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-2. eMIOS Module Configuration Register (EMIOS_MCR)

Table 16-6. EMIOS_MCR Field Descriptions

Field	Description
0	Reserved. This bit is readable/writable, but has no effect.
1 MDIS	Module disable. Puts the eMIOS in low power mode. The MDIS bit is used to stop the clock of the module, except the access to registers EMIOS_MCR and EMIOS_OUDR. 0 Clock is running 1 Enter low power mode
2 FRZ	Freeze. Enables the eMIOS to freeze the registers of the unified channels when debug mode is requested at MCU level. Each unified channel must have FREN bit set to enter freeze mode. While in freeze mode, the eMIOS continues to operate to allow the MCU access to the unified channels registers. The unified channel remains frozen until the FRZ bit is written to zero or the MCU exits debug mode or the unified channel FREN bit is cleared. 0 Allows unified channels to continue to operate when device enters debug mode and the EMIOS_CCR*n*[FREN] bit is set 1 Stops unified channels operation when in debug mode and the EMIOS_CCR*n*[FREN] bit is set
3 GTBE[1]	Global time base enable. Used to export a global time base enable from the module and provide a method to start time bases of several modules simultaneously. 0 Global time base enable out signal negated 1 Global time base enable out signal asserted **Note:** The global time base enable input signal controls the internal counters. When asserted, internal counters are enabled. When negated, internal counters disabled.

Table 16-6. EMIOS_MCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>4
ETB</td><td>External time base. Selects the time base source that drives counter bus[A].
0 Unified channel 23 drives counter bus[A]
1 STAC drives counter bus[A]
Note: If ETB is set to select STAC as the counter bus[A] source, the GTBE must be set to enable the STAC to counter bus[A]. See Section 16.4.2, “STAC Client Submodule” and the shared time and angle clock (STAC) bus interface section and the STAC bus configuration register (ETPU_REDCR) section of the eTPU chapter for more information about the STAC.</td></tr>
<tr><td>5
GPREN</td><td>Global prescaler enable. Enables the prescaler counter.
0 Prescaler disabled (no clock) and prescaler counter is cleared
1 Prescaler enabled</td></tr>
<tr><td>6–11</td><td>Reserved</td></tr>
<tr><td>12–15
SRV
[0:3]</td><td>Server time slot. Selects the address of a specific STAC server to which the STAC client submodule is assigned (see Section 16.4.2, “STAC Client Submodule,” for details)
0000 eTPU engine A, TCR1
0001 Invalid value
0010 eTPU engine A, TCR2
0011 Invalid value
0100–1111 Invalid value</td></tr>
<tr><td>16–23
GPRE
[0:7]</td><td>Global prescaler. Selects the clock divider value for the global prescaler, as shown below.
<table>
<tr><th>GPRE[0:7]</th><th>Divide Ratio</th></tr>
<tr><td>00000000</td><td>1</td></tr>
<tr><td>00000001</td><td>2</td></tr>
<tr><td>.
.
.
.</td><td>.
.
.
.</td></tr>
<tr><td>11111111</td><td>256</td></tr>
</table>
</td></tr>
<tr><td>24–31</td><td>Reserved</td></tr>
</table>

[1] The GTBE signal is an inter-module signal, not an external pin on the device.

16.3.1.2 eMIOS Global Flag Register EMIOS_GFR

The EMIOS_GFR is a read-only register that groups the FLAG bits from all channels. This organization improves interrupt handling on simpler devices. These bits are mirrors of the FLAG bits of each channel register (EMIOS_CSR) and flag bits in those channel registers cannot be cleared by accessing this ‘mirror’ register.

Address: Base + 0x0004 Access: User R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	F23	F22	F21	F20	F19	F18	F17	F16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-3. eMIOS Global Flag Register (EMIOS_GFR)

16.3.1.3 eMIOS Output Update Disable Register EMIOS_OUDR

The EMIOS_OUDR serves to disable transfers from the A2 to the A1 channel registers and from the B2 to the B1 channel registers when values are written to these registers, and the channel is running in modulus counter (MC) mode or an output mode.

Address: Base + 0x0008 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	OU23	OU22	OU21	OU20	OU19	OU18	OU17	OU16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OU15	OU14	OU13	OU12	OU11	OU10	OU9	OU8	OU7	OU6	OU5	OU4	OU3	OU2	OU1	OU0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-4. eMIOS Output Update Disable Register (EMIOS_OUDR)

Table 16-7. EMIOS_OUDR Field Descriptions

Field	Description
0–7	Reserved
8–31 OU*n*	Channel *n* output update disable. When running in MC mode or an output mode, values are written to registers A2 and B2. OU*n* bits are used to disable transfers from registers A2 to A1 and B2 to B1. Each bit controls one channel. 0 Transfer enabled. Depending on the operating mode, transfer can occur immediately or in the next period. Unless stated otherwise, transfer occurs immediately. 1 Transfers disabled

16.3.1.4 eMIOS Channel A Data Register EMIOS_CADR*n*

Depending on the mode of operation, internal registers A1 or A2, used for matches and captures, can be assigned to address EMIOS_CADR*n*. Both A1 and A2 are cleared by reset. Table 16-8 summarizes the EMIOS_CADR*n* writing and reading accesses for all operating modes. For more information see section Section 16.4.4.4, "Modes of Operation of the Unified Channels."

Address: UC*n* Base + 0x0000 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	A							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	A															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-5. eMIOS Channel A Data Register (EMIOS_CADR*n*)

16.3.1.5 eMIOS Channel B Data Register (EMIOS_CBDR*n*)

Depending on the mode of operation, internal registers B1 or B2 are used to address EMIOS_CBDR*n*. Both B1 and B2 are cleared by reset. Table 16-8 summarizes the EMIOS_CBDR*n* writing and reading accesses for all operating modes. For more information see section Section 16.4.4.4, "Modes of Operation of the Unified Channels."

NOTE

Do not read the EMIOS_CBDR*n* registers must not be read speculatively. For future compatibility, the TLB entry covering the EMIOS_CBDR*n* must be configured to be guarded.

Address: UC*n* Base + 0x0004 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	B							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	B															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-6. eMIOS Channel B Data Register (EMIOS_CBDR*n*)

Table 16-8. EMIOS_CADR*n* and EMIOS_CBDR*n* Value Assignments

Operating Mode	Register Access			
	Write	Read	Write	Read
GPIO	A1, A2	A1	B1,B2	B1
SAIC[1]	—	A2	B2	B2
SAOC[1]	A2	A1	B2	B2
IPWM	—	A2	—	B1
IPM	—	A2	—	B1
DAOC	A2	A1	B2	B1
PEA	A1	A2	—	B1
PEC[1]	A1	A1	B1	B1
QDEC[1]	A1	A1	B2	B2
WPTA	A1	A1	B1	B1
MC – Normal[1]	A2	A1	B2	B2
MC – Buffered	A2	A1	B2	B2
OPWFM – Normal	A2	A1	B2	B1
OPWFM – Buffered	A2	A1	B2	B1
OPWMC – Normal	A2	A1	B2	B1
OPWMC – Buffered	A2	A1	B2	B1
OPWM – Normal	A2	A1	B2	B1
OPWM – Buffered	A2	A1	B2	B1

[1] In these modes, the register EMIOS_CBDR*n* is not used, but B2 can be accessed.

16.3.1.6 eMIOS Channel Counter Register EMIOS_CCNTR*n*

The EMIOS_CCNTR*n* contains the value of the internal counter. When GPIO mode is selected or the channel is frozen, the EMIOS_CCNTR*n* is readable and writable. For all others modes, the EMIOS_CCNTR*n* is a read-only register. When entering some operating modes, this register is automatically cleared (see section Section 16.4.4.4, “Modes of Operation of the Unified Channels,” for details).

Address: UC*n* Base + 0x0008 Access: User R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	C							
W[1]																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	C															
W[1]																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 In GPIO mode or freeze action, this register is writable.

Figure 16-7. eMIOS Channel Counter Register (EMIOS_CCNTR*n*)

16.3.1.7 eMIOS Channel Control Register EMIOS_CCR*n*

The eMIOS_CCR*n* enables the setting of several control parameters for a unified channel. Among these controls are the setting of a channel prescaler, channel mode selection, input trigger sensitivity and filtering, interrupt and DMA request enabling, and output mode control.

Address: UC*n* Base + 0x000C Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FREN	ODIS	ODISSL		UCPRE		UCP REN	DMA	0	IF				FCK	FEN	0
W[1]																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	BSL		EDSEL	EDPOL	MODE						
W[1]			FORCMA	FORCMB												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-8. eMIOS Channel Control Register (EMIOS_CCR*n*)

Table 16-9. EMIOS_CCR*n* Field Description

Field	Description
0 FREN	Freeze enable. If set and validated by FRZ bit in EMIOS_MCR, freezes all registers values when in debug mode, allowing the MCU to perform debug functions. 0 Normal operation 1 Freeze UC registers values
1 ODIS	Output disable. Allows output disable in any output mode except GPIO. 0 The output pin operates normally 1 If the selected *output disable input* signal is asserted, the output pin goes to the complement of EDPOL for OPWFM, OPWFMB, and OPWMB modes, but the unified channel continues to operate normally; that is, it continues to produce FLAG and matches. When the selected *output disable input* signal is negated, the output pin operates normally.

Table 16-9. EMIOS_CCR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2–3
ODISSL
[0:1]</td><td>Output disable select. Selects one of the four output disable input signals.
00 output disable input 0
01 output disable input 1
10 output disable input 2
11 output disable input 3</td></tr>
<tr><td>4–5
UCPRE
[0:1]</td><td>Prescaler. Selects the clock divider value for the unified channel internal prescaler, as follows;
<table>
<tr><th>UCPRE[0:1]</th><th>Divide Ratio</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>2</td></tr>
<tr><td>10</td><td>3</td></tr>
<tr><td>11</td><td>4</td></tr>
</table></td></tr>
<tr><td>6
UCPREN</td><td>Prescaler enable. Enables the prescaler counter.
0 Prescaler disabled (no clock) and prescaler counter is loaded with UCPRE value
1 Prescaler enabled</td></tr>
</table>

Table 16-9. EMIOS_CCR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7
DMA</td><td>Direct memory access. Selects between generating an interrupt request (IRQ) or a DMA request.
0 FLAG assigned to Interrupt request
1 FLAG assigned to DMA request
Not all eMIOS channels support DMA, as shown below.
For a channel that does not support DMA, do not change the default value of 0 for the DMA bit.
<table>
<tr><th>eMIOS Channel</th><th>DMA = 0</th><th>DMA = 1</th></tr>
<tr><td>0</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>1</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>2</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>3</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>4</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>5</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>6</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>7</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>8</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>9</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>10</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>11</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>12</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>13</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>14</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>15</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>16</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>17</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>18</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>19</td><td>Interrupt</td><td>DMA request</td></tr>
<tr><td>20</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>21</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>22</td><td>Interrupt</td><td>Reserved</td></tr>
<tr><td>23</td><td>Interrupt</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>8</td><td>Reserved</td></tr>
</table>

Table 16-9. EMIOS_CCR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>9–12
IF
[0:3]</td><td>Input filter. Controls the programmable input filter, selecting the minimum input pulse width that can pass through the filter, as shown below. For output modes, these bits have no meaning.
<table>
<tr><th>IF[0:3][1]</th><th>Minimum input pulse width [filter clock periods]</th></tr>
<tr><td>0000</td><td>Bypassed[2]</td></tr>
<tr><td>0001</td><td>2 filter clock periods</td></tr>
<tr><td>0010</td><td>4 filter clock periods</td></tr>
<tr><td>0100</td><td>8 filter clock periods</td></tr>
<tr><td>1000</td><td>16 filter clock periods</td></tr>
<tr><td>all others</td><td>Reserved</td></tr>
</table>
1 Filter latency is 3 clock cycles.
2 The input signal is synchronized before arriving at the digital filter.</td></tr>
<tr><td>13
FCK</td><td>Filter clock select. Selects the clock source for the programmable input filter.
0 Prescaled clock
1 System clock</td></tr>
<tr><td>14
FEN</td><td>FLAG enable. Allows the unified channel FLAG bit to generate an interrupt signal or a DMA request signal (The type of signal to be generated is defined by the DMA bit).
0 Disable (FLAG does not generate an interrupt or DMA request)
1 Enable (FLAG generates an interrupt or DMA request)</td></tr>
<tr><td>15–17</td><td>Reserved</td></tr>
<tr><td>18
FORCMA</td><td>Force match A. For output modes, the FORCMA bit is equivalent to a successful comparison on comparator A (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operating mode which uses comparator A, otherwise it has no effect.
0 Has no effect
1 Force a match at comparator A
For input modes, the FORCMA bit is not used and writing to it has no effect.</td></tr>
<tr><td>19
FORCMB</td><td>Force match B. For output modes, the FORCMB bit is equivalent to a successful comparison on comparator B (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operating mode which uses comparator B, otherwise it has no effect.
0 Has no effect
1 Force a match at comparator B
For input modes, the FORCMB bit is not used and writing to it has no effect.</td></tr>
<tr><td>20</td><td>Reserved</td></tr>
</table>

Table 16-9. EMIOS_CCR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>21–22
BSL
[0:1]</td><td>Bus select. Used to select either one of the counter buses or the internal counter to be used by the unified channel.

<table>
<tr><th>BSL[0:1]</th><th>Selected Bus</th></tr>
<tr><td>00</td><td>All channels: counter bus[A]</td></tr>
<tr><td>01</td><td>Channels 0 to 7: counter bus[B]
Channels 8 to 15: counter bus[C]
Channels 16 to 23: counter bus[D]</td></tr>
<tr><td>10</td><td>Reserved</td></tr>
<tr><td>11</td><td>All channels: internal counter (see Note)</td></tr>
</table>

Note: In certain modes the internal counter is used internally and therefore cannot be used as the channel time base.</td></tr>
<tr><td>23
EDSEL</td><td>Edge selection bit. For input modes, the EDSEL bit selects whether the internal counter is triggered by both edges of a pulse or just by a single edge as defined by the EDPOL bit. When not shown in the mode of operation description, this bit has no effect.
0 Single edge triggering defined by the EDPOL bit
1 Both edges triggering

For GPIO input mode, the EDSEL bit selects if a FLAG can be generated.
0 A FLAG is generated as defined by the EDPOL bit
1 No FLAG is generated

For SAOC mode, the EDSEL bit selects the behavior of the output flip-flop at each match.
0 The EDPOL value is transferred to the output flip-flop
1 The output flip-flop is toggled</td></tr>
</table>

Table 16-9. EMIOS_CCR*n* Field Description (continued)

Field	Description
24 EDPOL	Edge polarity. For input modes (except QDEC and WPTA mode), the EDPOL bit asserts which edge triggers either the internal counter or an input capture or a FLAG. When not shown in the mode of operation description, this bit has no affect. 0 Trigger on a falling edge 1 Trigger on a rising edge For WPTA mode, the internal counter is used as a time accumulator and counts up when the input gating signal has the same polarity of EDPOL bit. 0 Counting occurs when the input gating signal is low 1 Counting occurs when the input gating signal is high For QDEC (MODE[6] cleared), the EDPOL bit selects the count direction according to *direction* signal (UC*n* input). 0 Counts down when UC*n* is asserted 1 Counts up when UC*n* is asserted NOTE: UC[n-1] EDPOL bit selects which edge clocks the internal counter of UC*n* 0 Trigger on a falling edge 1 Trigger on a rising edge For QDEC (MODE[6] set), the EDPOL bit selects the count direction according to the phase difference. 0 Internal counter decrements if phase A is ahead phase B signal 1 Internal counter increments if phase A is ahead phase B signal NOTE: To operate correctly, EDPOL bit must contain the same value in UC*n* and UC[n-1] For output modes, the EDPOL bit is used to select the logic level on the output pin. When software selects any output mode except GPIO, the initial state of the output flip-flop is the complement of EDPOL. 0 A match on comparator A clears the output flip-flop, while a match on comparator B sets it 1 A match on comparator A sets the output flip-flop, while a match on comparator B clears it
25–31 MODE	Mode selection. Selects the mode of operation of the unified channel, as shown in Table 16-10.

Table 16-10. Unified Channel MODE Bits

MODE[0:6]	Unified Channel Mode of Operation
0b0000000	General purpose input/output mode (input)
0b0000001	General purpose input/output mode (output)
0b0000010	Single action input capture
0b0000011	Single action output compare
0b0000100	Input pulse width measurement
0b0000101	Input period measurement
0b0000110	Double action output compare (with FLAG set on the second match)
0b0000111	Double action output compare (with FLAG set on both match)
0b0001000	Pulse/edge accumulation (continuous)

Table 16-10. Unified Channel MODE Bits (continued)

MODE[0:6]	Unified Channel Mode of Operation
0b0001001	Pulse/edge accumulation (single shot)
0b0001010	Pulse/edge counting (continuous)
0b0001011	Pulse/edge counting (single shot)
0b0001100	Quadrature decode (for count and direction encoders type)
0b0001101	Quadrature decode (for phase A and phase B encoders type)
0b0001110	Windowed programmable time accumulation
0b0001111	Reserved
0b0010000	Modulus counter (up counter, internal clock source)
0b0010001	Modulus counter (up counter, external clock source)
0b0010010–0b0010011	Reserved
0b0010100	Modulus counter (up/down counter, no change in counter direction upon match of input counter and register B1, internal clock source)
0b0010101	Modulus counter (up/down counter, no change in counter direction upon match of input counter and register B1, external clock source)
0b0010110	Modulus counter (up/down counter, change in counter direction upon match of input counter and register B1 and sets the FLAG, internal clock source)
0b0010111	Modulus counter (up/down counter, change in counter direction upon match of input counter and register B1 and sets the FLAG, external clock source)
0b0011000	Output pulse width and frequency modulation (FLAG set at match of internal counter and comparator B, immediate update)
0b0011001	Output pulse width and frequency modulation (FLAG set at match of internal counter and comparator B, next period update)
0b0011010	Output pulse width and frequency modulation (FLAG set at match of internal counter and comparator A or comparator B, immediate update)
0b0011011	Output pulse width and frequency modulation (FLAG set at match of internal counter and comparator A or comparator B, next period update)
0b0011100	Center aligned output pulse width modulation (FLAG set in trailing edge, trailing edge dead-time)
0b0011101	Center aligned output pulse width modulation (FLAG set in trailing edge, leading edge dead-time)
0b0011110	Center aligned output pulse width modulation (FLAG set in both edges, trailing edge dead-time)
0b0011111	Center aligned output pulse width modulation (FLAG set in both edges, leading edge dead-time)
0b0100000	Output pulse width modulation (FLAG set at match of internal counter and comparator B, immediate update)
0b0100001	Output pulse width modulation (FLAG set at match of internal counter and comparator B, next period update)

Table 16-10. Unified Channel MODE Bits (continued)

MODE[0:6]	Unified Channel Mode of Operation
0b0100010	Output pulse width modulation (FLAG set at match of internal counter and comparator A or comparator B, immediate update)
0b0100011	Output pulse width modulation (FLAG set at match of internal counter and comparator A or comparator B, next period update)
0b1100100–0b1111111	Reserved
0b1010000	Modulus up counter, buffered, internal clock
0b1010001	Modulus up counter, buffered, external clock
0b1010010–0b1010001	Reserved
0b1010100	Modulus up/down counter, buffered (FLAG set on one event, internal clock)
0b1010101	Modulus up/down counter, buffered (FLAG set on one event, external clock)
0b1010110	Modulus up/down counter, buffered (FLAG set on both events, internal clock)
0b1010111	Modulus up/down counter, buffered (FLAG set on both events, external clock)
0b1011000	Output pulse width and frequency modulation, buffered (FLAG set at match of internal counter and comparator B)
0b1011001	Reserved
0b1011010	Output pulse width and frequency modulation, buffered (FLAG set at match of internal counter and comparator A or comparator B)
0b1011011	Reserved
0b1011100	Center aligned output pulse width modulation, buffered (FLAG set on trailing edge, trailing edge dead-time)
0b1011101	Center aligned output pulse width modulation, buffered (FLAG set on trailing edge, leading edge dead-time)
0b1011110	Center aligned output pulse width modulation, buffered (FLAG set on both edges, trailing edge dead-time)
0b1011111	Center aligned output pulse width modulation, buffered (FLAG set on both edges, leading edge dead-time)
0b1100000	Output pulse width modulation, buffered (FLAG set on second match)
0b1100001	Reserved
0b1100010	Output pulse width modulation, buffered (FLAG set on both matches)

16.3.1.8 eMIOS Channel Status Register (EMIOS_CSR*n*)

EMIOS_CSR*n* reflects the status of the UC input/output signals and the overflow condition of the internal counter, as well as the occurrence of a trigger event.

Address: UC*n* Base + 0x0010 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OVR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVFL	0	0	0	0	0	0	0	0	0	0	0	0	UCIN	UCOUT	FLAG
W	w1c															w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-9. eMIOS Channel Status Register (EMIOS_CSR*n*)

Table 16-11. EMIOS_CSR*n* Field Descriptions

Field	Description
0 OVR	Overrun. Indicates that FLAG generation occurred when the FLAG bit was already set. This bit can be cleared by writing a 1 to it or by clearing the FLAG bit. 0 Overrun has not occurred 1 Overrun has occurred
1–15	Reserved
16 OVFL	Overflow. Indicates that an overflow has occurred in the internal counter. OVFL is cleared by writing a 1 to it. 0 No overflow 1 An overflow had occurred
17–28	Reserved
29 UCIN	Unified channel input pin. Reflects the input pin state after being filtered and synchronized.
30 UCOUT	Unified channel output pin. The UCOUT bit reflects the output pin state.
31 FLAG	FLAG. Set when an input capture or a match event in the comparators occurred. This bit is cleared by writing a 1 to it. 0 FLAG cleared 1 FLAG set event has occurred **Note:** When EMIOS_CCR[DMA] bit is set, the FLAG bit is cleared by the eDMA controller.

16.4 Functional Description

The eMIOS provides independent channels (UC) that can be configured and accessed by the MCU. Four time bases can be shared by the channels through four counter buses and each unified channel can generate its own time base. Optionally, the counter A bus can be driven by an external time base from the eTPU imported through the STAC interface.

NOTE

Counter bus A can be driven by unified channel 23 or by the STAC bus. Counter bus B, C, and D are driven by unified channels 0, 8, and 16, respectively. Counter bus A can be shared among all unified channels. UCs 0 to 7, 8 to 15, and 16 to 23 can share counter buses B, C, and D, respectively.

The following four components of the eMIOS are discussed below:

- Bus interface unit
- STAC client submodule
- Global clock prescaler
- Unified channels and their modes of operation

16.4.1 Bus Interface Unit (BIU)

The bus interface unit provides the interface between the internal bus and the slave interface, allowing communication among all submodules and the slave interface.

The BIU allows 8-, 16-, and 32-bit accesses. They are performed over a 32-bit data bus in a single cycle clock.

16.4.1.1 Effect of Freeze on the BIU

When the FRZ bit in the EMIOS_MCR is set and the module is in debug mode, the operation of the BIU is not affected.

16.4.2 STAC Client Submodule

The shared time and angle count (STAC) bus provides access to one external time base, imported from the STAC bus to the eMIOS unified channels. The eTPU module's time bases and angle count can be exported and/or imported through the STAC client submodule interface. Time bases and/or angle information of either eTPU engine can be exported to the other eTPU engine and to the eMIOS module, which is only a STAC client. There are restrictions on engine export/import targets: one engine cannot export from or import to itself, nor can it import time base and/or angle count if in angle mode.

The device’s STAC server identification assignment is shown in Table 16-12. The time slot assignment is fixed, so only time bases running at system clock ÷ 4 or slower can be integrally exported. The STAC client submodule runs with the system clock, and its time slot timing is synchronized with the eTPU timing on reset. The time slot sequence is 0-1-2-3, such that they are alternated between engines 1 and 2.

Table 16-12. STAC Client Submodule Server Slot Assignment

Engine	Time Base	Server ID
1	TCR1	0
1	TCR2	2
2	TCR1	1
2	TCR2	3

Figure 16-10 provides a block diagram for the STAC client submodule.

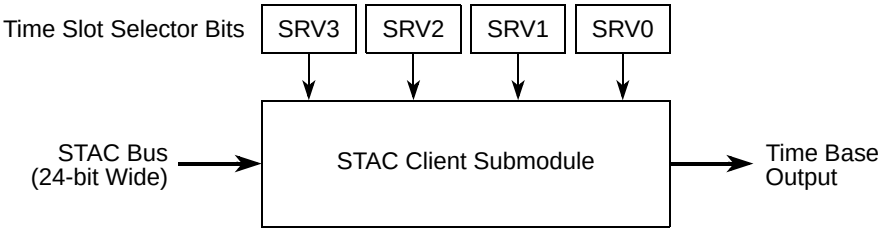
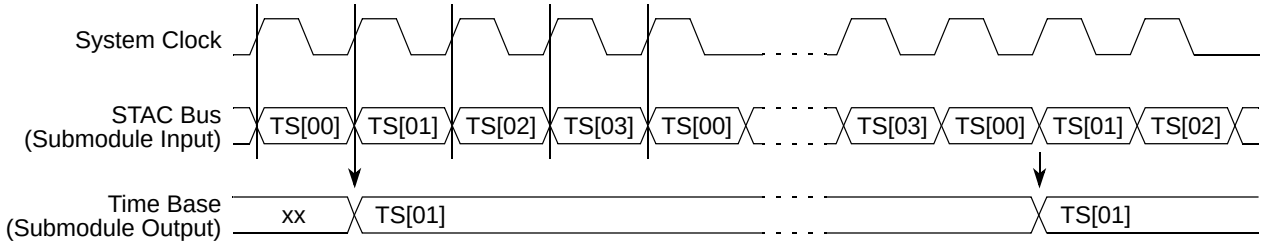


Figure 16-10. STAC Client Submodule Block Diagram

Bits SRV[0:3] in register EMIOS_MCR, selects the desired time slot of the STAC bus to be output. Figure 16-11 shows a timing diagram for the STAC client submodule.



Note: In this case, SRV bits were set to capture TS[01].

Figure 16-11. Timing Diagram for the STAC Bus and STAC Client Submodule Output

Every time the selected time slot changes, the STAC Client Submodule output is updated.

16.4.2.1 Effect of Freeze on the STAC Client Submodule

When the FRZ bit in the EMIOS_MCR is set and the module is in debug mode, the operation of the STAC client submodule is not affected; that is, there is no freeze function in this submodule.

16.4.3 Global Clock Prescaler Submodule (GCP)

The GCP divides the system clock to generate a clock for the clock prescalers of the unified channels. The system clock is prescaled by the value defined in Table 16-7 according to the GPRE[0:7] bits in the EMIOS_MCR. Counting is enabled by setting EMIOS_MCR[GPREN]. The counter can be stopped at any time by clearing this bit, thereby stopping the internal counter in all the unified channels.

16.4.3.1 Effect of Freeze on the GCP

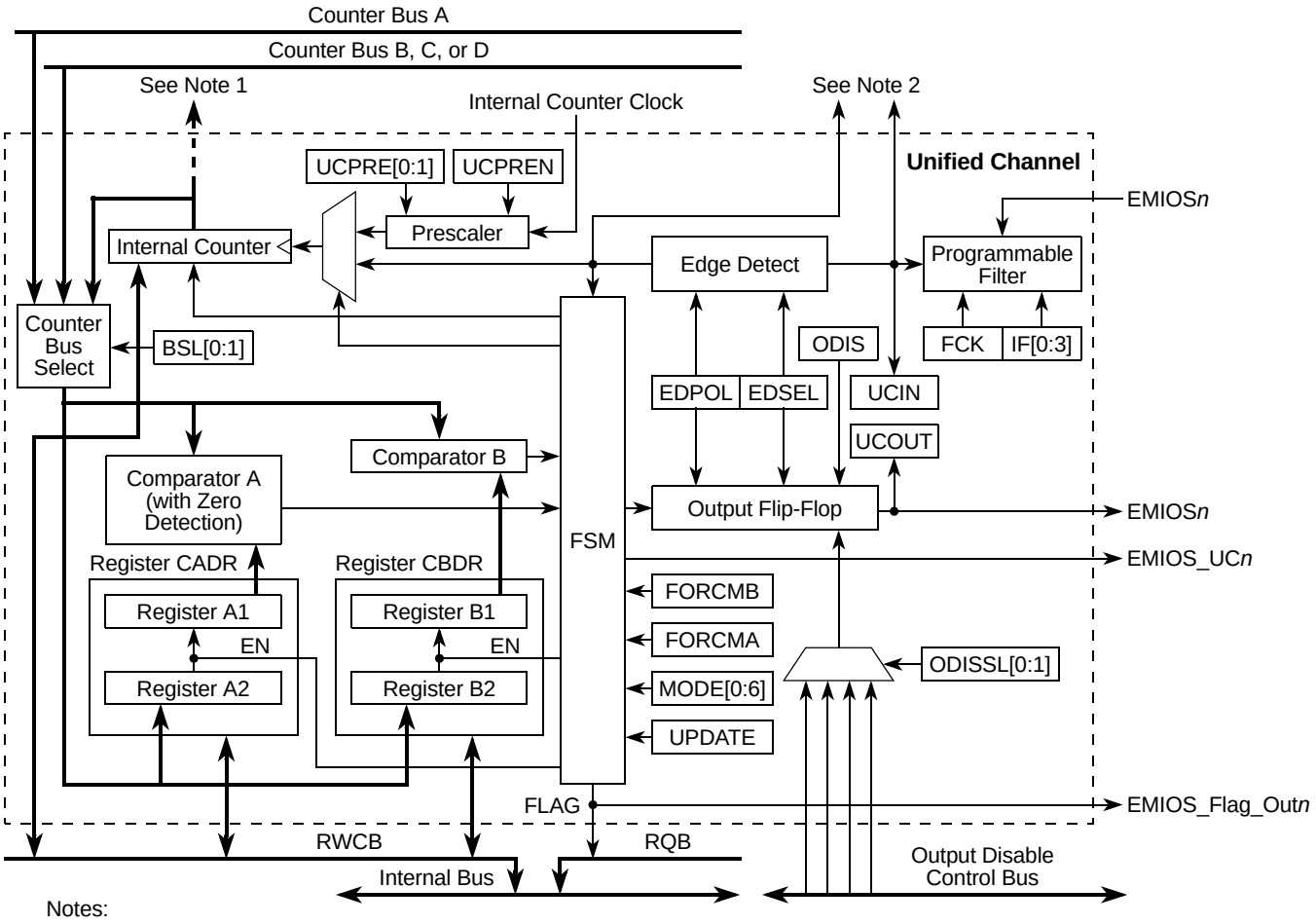
When the FRZ bit in the EMIOS_MCR is set and the module is in debug mode, the operation of GCP submodule is not affected; that is, there is no freeze function in this submodule.

16.4.4 Unified Channel (UC)

Figure 16-12 shows the unified channel block diagram. Each unified channel consists of the following:

- Counter bus selector that selects the time base to be used by the channel for all timing functions
- Programmable clock prescaler
- Two double buffered data registers A and B that allow up to two input capture and/or output compare events to occur before software intervention is needed.
- Two comparators (equal only) A and B that compare the selected counter bus with the value in the data registers
- Internal counter that can be used as a local time base or to count input events
- Programmable input filter that ensures that only valid pin transitions are received by a channel
- Programmable input edge detector that detects rising, falling, or both edges
- Output flip-flop that holds the logic level to be applied to the output pin
- eMIOS status and control registers
- Output disable input selector that selects the output disable input signal to be used as the unified channel output disable
- Control state machine (FSM)

The major components and functions of the unified channels are discussed in Section 16.4.4.1, “Programmable Input Filter (PIF)” through Section 16.4.4.4, “Modes of Operation of the Unified Channels.”



Notes:

1. Counter bus A can be driven by either the STAC bus or channel 23. See EMIOS_MCR[ETB]. Channel 0 drives counter bus B, channel 8 drives counter bus C and channel 16 drives counter bus D. Counter bus B can be selected as the counter for channels 0-7, counter bus C for channels 8-15, and counter bus D for channels 16-23. See Figure 1-1 and EMIOS_CCR*n*[BS].
2. Goes to the finite state machine of the UC[n-1]. These signals are used for QDEC mode.

Figure 16-12. Unified Channel Block Diagram

16.4.4.1 Programmable Input Filter (PIF)

The PIF ensures that only valid input pin transitions are received by the unified channel edge detector. A block diagram of the PIF is shown in Figure 16-13.

The PIF is a 5-bit programmable up counter that is incremented by the selected clock source, according to bits IF[0:3] in EMIOS_CCR*n*. The clock source is selected by the EMIOS_CCR*n*[FCK] bit.

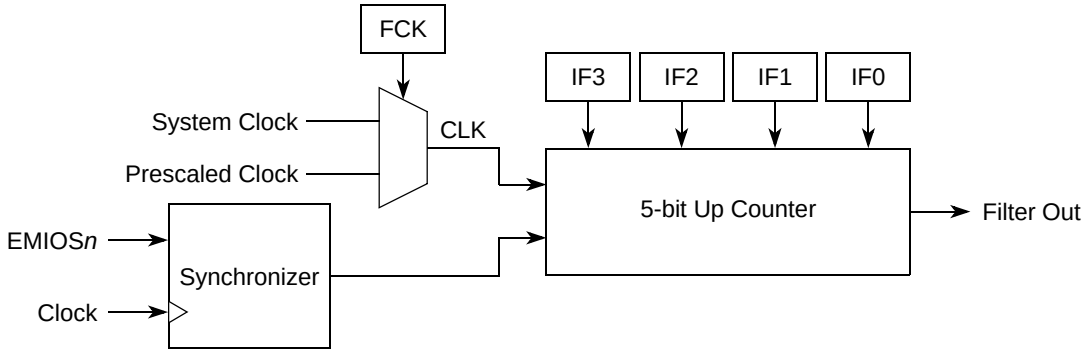


Figure 16-13. Programmable Input Filter Submodule Diagram

The input signal is synchronized by the system clock. When a state change occurs in this signal, the 5-bit counter starts counting up. As long as the new state is stable on the pin, the counter continues incrementing. If a counter overflows occurs, the new pin value is validated. In this case, it is transmitted as a pulse edge to the edge detector. If the opposite edge appears on the pin before validation (overflow), the counter is reset. At the next synchronized pin transition, the counter starts counting again. Any pulse that is shorter than a full range of the masked counter is regarded as a glitch, and it is not passed on to the edge detector. A timing diagram of the input filter is shown in Figure 16-14.

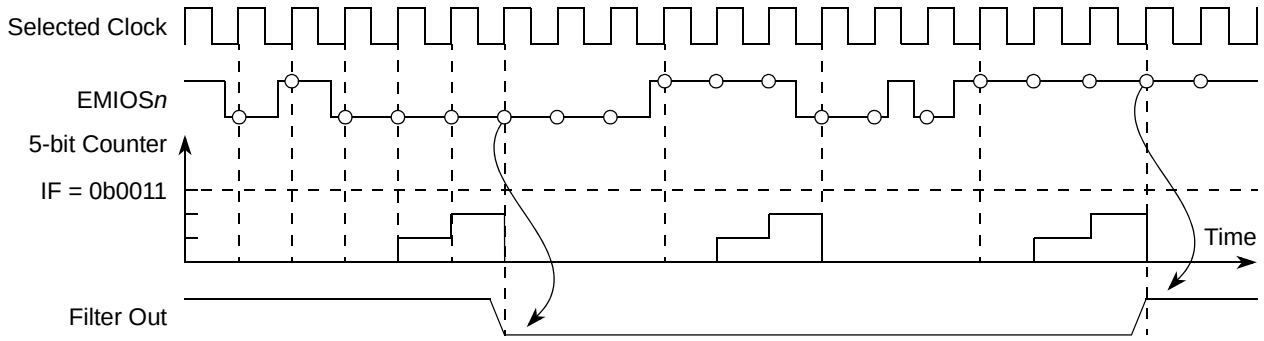


Figure 16-14. Programmable Input Filter Example

16.4.4.2 Clock Prescaler (CP)

A unified channel has a clock prescaler (CP) that divides the global clock prescaler (see Section 16.4.3, "Global Clock Prescaler Submodule (GCP)") output signal to generate a clock enable for the internal counter of the unified channel. It is a programmable 2-bit down counter. The global clock prescaler submodule (GCP) output signal is prescaled by the value defined in Table 16-9 according to the UCPRE[0:1] bits in the EMIOS_CCR*n*. The output is clocked every time the counter reaches zero. Counting is enabled by setting the UCPREN bit in the EMIOS_CCR*n*. The counter can be stopped at any time by clearing this bit, thereby stopping the internal counter in the unified channel.

16.4.4.3 Effect of Freeze on the Unified Channel

When in debug mode and the EMIOS_MCR[FRZ] bit and the EMIOS_CCR*n*[FREN] bit are both set, the internal counter and the unified channel's capture and compare functions are halted. The UC is frozen in its current state.

During freeze, all registers are accessible. When the unified channel is operating in an output mode, the force match functions remain available, allowing the software to force the output to the desired level.

For input modes, any input events that occur while the channel is frozen are ignored.

When exiting debug mode or freeze enable bit is cleared (FRZ in the EMIOS_MCR or FREN in the EMIOS_CCR*n*) the channel actions resume.

16.4.4.4 Modes of Operation of the Unified Channels

The mode of operation of a unified channel is determined by the mode select bits MODE[0:6] in the EMIOS_CCR*n*. See Table 16-10 for details.

When entering an output mode (except for GPIO mode), the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR*n*.

Because the internal counter EMIOS_CCNTR*n* continues to run in all modes (except for GPIO mode), it is possible to use this counter as the UC time base unless it (the internal counter) is a required resource in the operation of the selected mode.

To provide smooth waveform generation while allowing A and B registers to be changed during operation, the double-buffered modes MCB, OPWFMB, OPWMB, and OPWMCB are provided (beginning at Section 16.4.4.4.15, “Modulus Counter, Buffered Mode (MCB)”). In these modes the A and B registers are double buffered. Descriptions of the double-buffered modes are presented separately, because there are several basic differences from the single-buffered MC, OPWFM, OPWM, and OPWMC modes.

Section 16.4.4.4.2, “Single Action Input Capture Mode (SAIC)” through Section 16.4.4.4.18, “Output Pulse Width Modulation, Buffered Mode (OPWMB)” below explain in detail the unified channels’ modes of operation.

16.4.4.4.1 General Purpose Input/Output Mode (GPIO)

In GPIO mode, all input capture and output compare functions of the UC are disabled, the internal counter (EMIOS_CCNTRn register) is cleared and disabled. All control bits remain accessible. To prepare the UC for a new operating mode, writing to registers EMIOS_CADR*n* or EMIOS_CBDR*n* stores the same value in registers A1 and A2, or B1 and B2, respectively.

MODE[6] bit selects between input (MODE[6] = 0) and output (MODE[6] = 1) modes:

MODE[0:6]	Unified Channel Mode of Operation
0b0000000	General purpose input/output mode (input)
0b0000001	General purpose input/output mode (output)

It is required that when changing MODE[0:6], the application software goes into GPIO mode first to reset the UC internal functions correctly. Failure to do this can lead to invalid and unexpected output compares and input capture results, or can cause the FLAGs to be set incorrectly.

In GPIO input mode, the FLAG generation is determined according to EDPOL and EDSEL bits and the input pin status can be determined by reading the UCIN bit.

In GPIO output mode, the unified channel is used as a single output port pin and the value of the EDPOL bit is permanently transferred to the output flip-flop.

NOTE

The GPIO modes provided in the eMIOS are particularly useful as interim modes when certain other eMIOS modes are being dynamically configured and enabled or disabled during the execution of the application. For normal GPIO function on the eMIOS pins, it is recommended that the SIU be used to configure those pins as system GPIO. See Section 6.3.1.3, "General-Purpose I/O Pins (GPIO[0:213])".

16.4.4.4.2 Single Action Input Capture Mode (SAIC)

MODE[0:6]	Unified Channel Mode of Operation
0b0000010	Single Action Input Capture Mode

In SAIC mode, when a triggering event occurs on the input pin, the value on the selected time base is captured into register A2. At the same time, the FLAG bit is set to indicate that an input capture has occurred. Register EMIOS_CADR*n* returns the value of register A2.

The input capture is triggered by a rising, falling or either edges in the input pin, as configured by EDPOL and EDSEL bits in EMIOS_CCR*n*.

Figure 16-15 shows how the unified channel can be used for input capture.

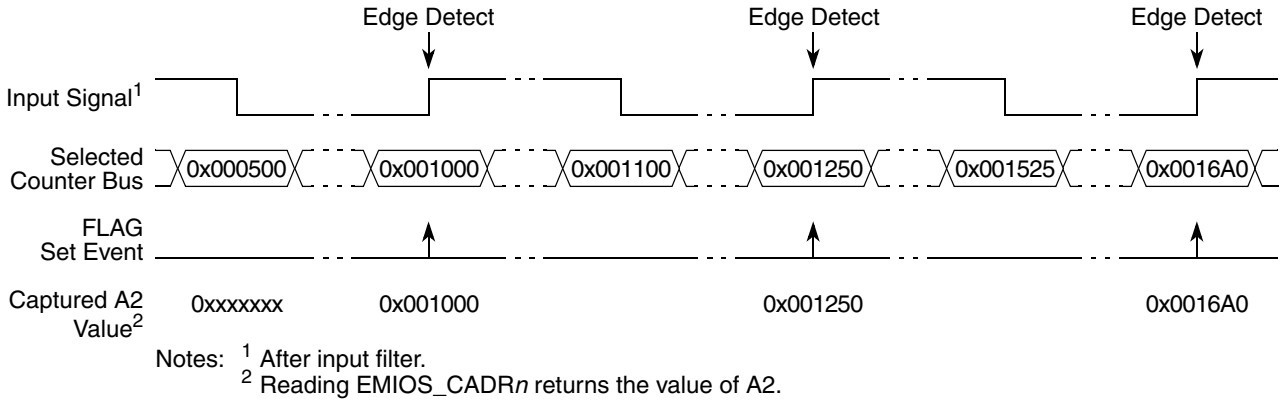


Figure 16-15. Single Action Input Capture Example

16.4.4.4.3 Single Action Output Compare Mode (SAOC)

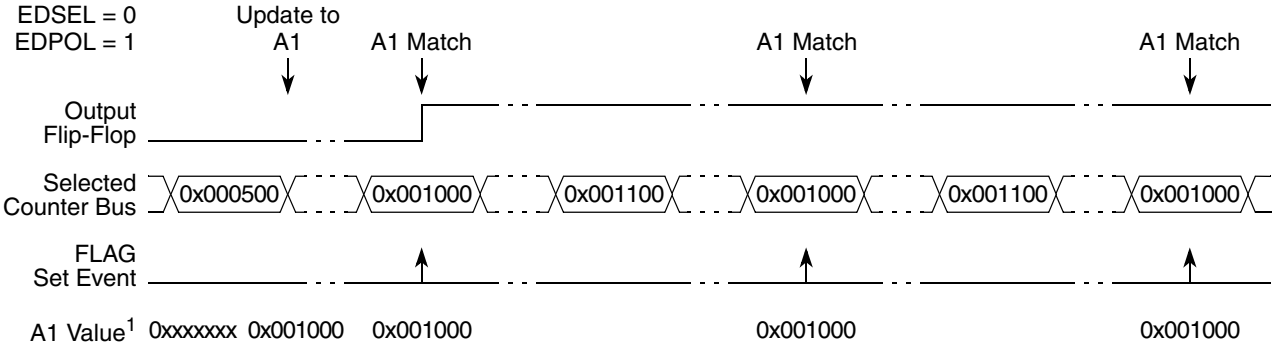
MODE[0:6]	Unified Channel Mode of Operation
0b0000011	Single Action Output Compare Mode

In SAOC mode a match value is loaded in register A2 and then transferred to register A1 to be compared with the selected time base. When a match occurs, the EDSEL bit selects if the output flip-flop is toggled or if the value in EDPOL is transferred to it. At the same time, the FLAG bit is set to indicate that the output

compare match has occurred. Writing to register EMIOS_CADR*n* stores the value in register A2 and reading to register EMIOS_CADR*n* returns the value of register A1.

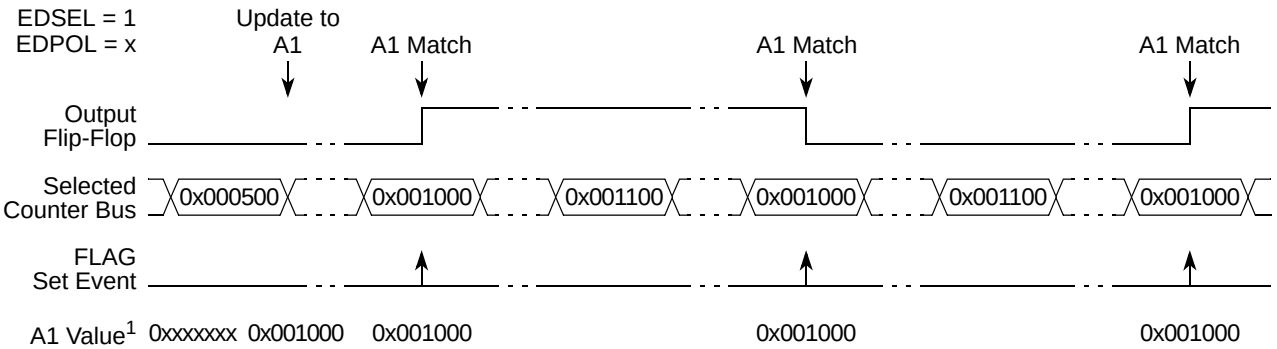
An output compare match can be simulated in software by setting the FORCMA bit in EMIOS_CCR*n*. In this case, the FLAG bit is not set.

Figure 16-16 and Figure 16-17 show how the unified channel can be used to perform a single output compare with EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively.



Notes: [1] EMIOS_CADR*n* writes to A2.
A2 value transferred to A1 according to OU*n* bit.

Figure 16-16. SAOC Example with EDPOL Value Transferred to the Output Flip-flop



Notes: [1] EMIOS_CADR*n* writes to A2.
A2 value transferred to A1 according to OU*n* bit.

Figure 16-17. SAOC Example Toggling the Output Flip-flop

16.4.4.4.4 Input Pulse Width Measurement Mode (IPWM)

MODE[0:6]	Unified Channel Mode of Operation
0b0000100	Input Pulse Width Measurement Mode

The IPWM mode allows the measurement of the width of a positive or negative pulse by capturing the leading edge on register B1 and the trailing edge on register A2. Successive captures are done on consecutive edges of opposite polarity. The leading edge sensitivity (that is, pulse polarity) is selected by

EDPOL bit in the EMIOS_CCR*n*. Registers EMIOS_CADR*n* and EMIOS_CBDR*n* return the values in register A2 and B1, respectively.

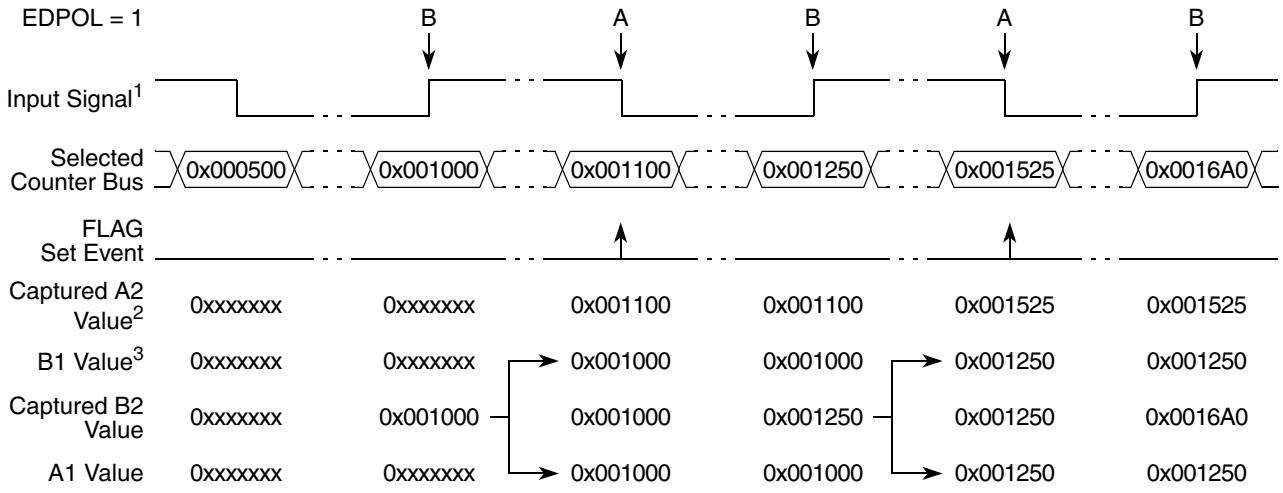
The capture function of register A2 remains disabled until the first leading edge triggers the first input capture on register B2. When this leading edge is detected, the count value of the selected time base is latched into register B2; the FLAG bit is not set. When the trailing edge is detected, the count value of the selected time base is latched into register A2 and, at the same time, the FLAG bit is set and the content of register B2 is transferred to register B1.

If subsequent input capture events occur while the corresponding FLAG bit is set, registers A2 and B1 are updated with the latest captured values and the FLAG remain set. Registers EMIOS_CADR*n* and EMIOS_CBDR*n* return the value in registers A2 and B1, respectively.

To guarantee coherent access, reading EMIOS_CADR*n* disables transfers between B2 and B1 until reading EMIOS_CBDR*n*. After that, transfer is re-enabled.

The input pulse width is calculated by subtracting the value in B1 from A2.

Figure 16-18 shows how the unified channel can be used for input pulse width measurement.



Notes: 1 After input filter.
2 Reading EMIOS_CADRn returns the value of A2, writing EMIOS_CADRn writes to A2.
3 Reading EMIOS_CBDRn returns the value of B1, writing EMIOS_CBDRn writes to B1.

Figure 16-18. Input Pulse Width Measurement Example

16.4.4.4.5 Input Period Measurement Mode (IPM)

MODE[0:6]	Unified Channel Mode of Operation
0b0000101	Input Period Measurement Mode

The IPM mode allows the measurement of the period of an input signal by capturing two consecutive rising edges or two consecutive falling edges. Successive input captures are done on consecutive edges of the same polarity. The edge polarity is defined by the EDPOL bit in the EMIOS_CCR*n*.

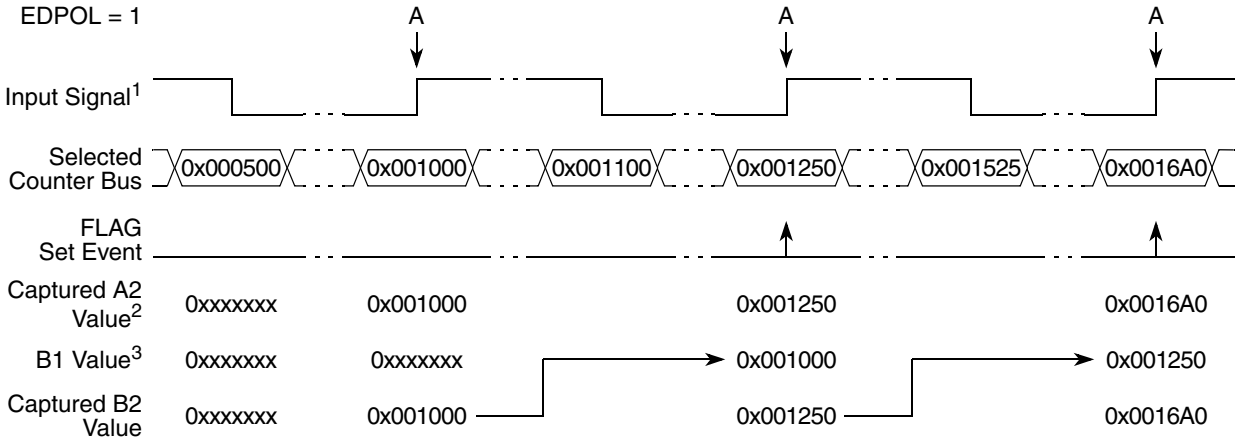
When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set, and the values in registers B1 is meaningless. On the second and subsequent captures, the FLAG line is set and data in register B2 is transferred to register B1.

When the second edge of the same polarity is detected, the counter bus value is latched into registers A2 and B2, the data previously held in register B2 is transferred to data register B1, and the FLAG bit is set to indicate the start and end points of a complete period have been captured. This sequence of events is repeated for each subsequent capture. Registers EMIOS_CADR*n* and EMIOS_CBDR*n* return the values in register A2 and B1, respectively.

To guarantee coherent access, reading EMIOS_CADR*n* disables transfers between B2 and B1 until reading EMIOS_CBDR*n* register, then any pending transfer is re-enabled.

The input pulse period is calculated by subtracting the value in B1 from A2.

Figure 16-19 shows how the unified channel can be used for input period measurement.



Notes: 1 After input filter.
2 Reading EMIOS_CADR*n* returns the value of A2, writing EMIOS_CADR*n* writes to A2.
3 Reading EMIOS_CBDR*n* returns the value of B1, writing EMIOS_CBDR*n* writes to B1.

Figure 16-19. Input Period Measurement Example

16.4.4.4.6 Double Action Output Compare Mode (DAOC)

MODE[0:6]	Unified Channel Mode of Operation
0b0000110	Double action output compare (with FLAG set on the second match)
0b0000111	Double action output compare (with FLAG set on both match)

In the DAOC mode the leading and trailing edges of the variable pulse width output are generated by matches occurring on comparators A and B, respectively.

When the DAOC mode is first selected (coming from GPIO mode) both comparators are disabled. Comparators A and B are enabled by updating registers A1 and B1 respectively and remain enabled until a match occurs on that comparator, when it is disabled again. To update registers A1 and B1, a write to A2 and B2 must occur and the EMIOS_CCR*n*[ODIS] bit must be cleared.

The output flip-flop is set to the value of EMIOS_CCR*n*[EDPOL] when a match occurs on comparator A and to the complement of EDPOL when a match occurs on comparator B.

MODE[6] controls if the EMIOS_CSR*n*[FLAG] is set on both matches or just on the second match (see Table 16-10 for details).

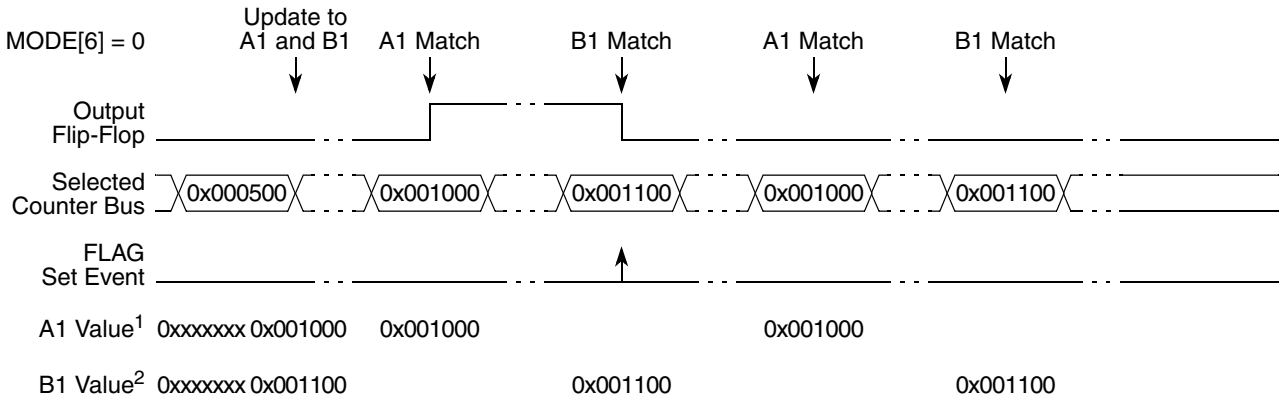
If subsequent enabled output compares occur on registers A1 and B1, pulses continue to be generated, regardless of the state of the FLAG bit.

At any time, the EMIOS_CCR*n*[FORCMA] and EMIOS_CCR*n*[FORCMB] bits allow the software to force the output flip-flop to the level corresponding to a comparison event in comparator A or B, respectively. The FLAG bit is not affected by these forced operations.

NOTE

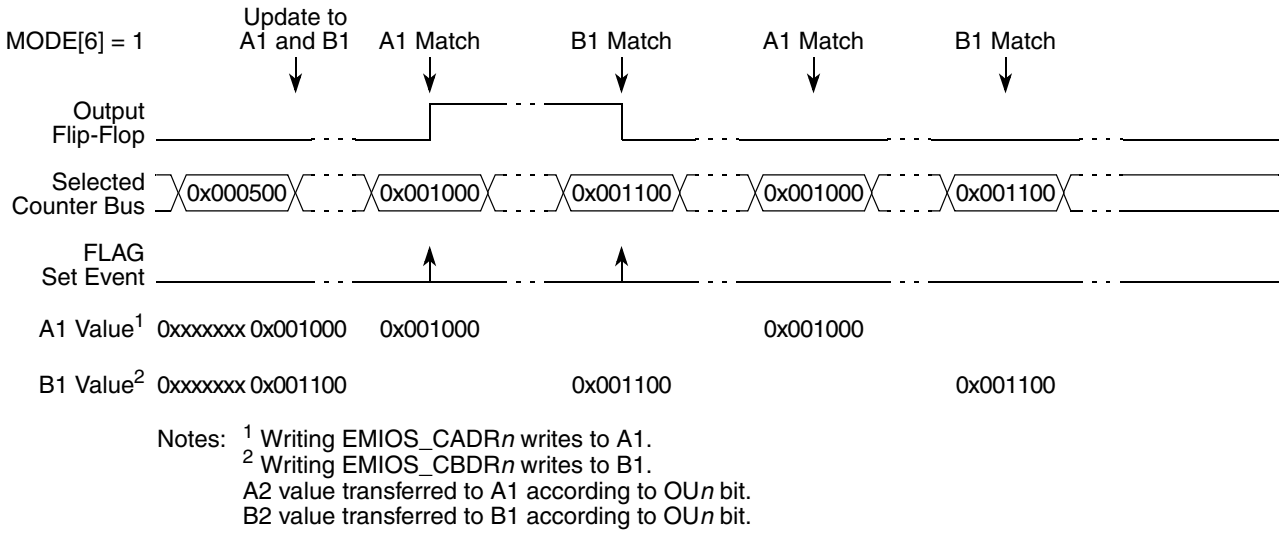
If both registers (A1 and B1) are loaded with the same value, the unified channel behaves as if a single match on comparator B had occurred; the output flip-flop is set to the complement of EDPOL bit and the FLAG bit is set.

Figure 16-20 and Figure 16-21 show how the unified channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.



Notes: [1] Writing EMIOS_CADR*n* writes to A1.
[2] Writing EMIOS_CBDR*n* writes to B1.
A2 value transferred to A1 according to OU*n* bit.
B2 value transferred to B1 according to OU*n* bit.

Figure 16-20. Double Action Output Compare with FLAG Set on the Second Match



Notes: [1] Writing EMIOS_CADR*n* writes to A1.
[2] Writing EMIOS_CBDR*n* writes to B1.
A2 value transferred to A1 according to OU*n* bit.
B2 value transferred to B1 according to OU*n* bit.

Figure 16-21. Double Action Output Compare with FLAG Set on Both Matches

16.4.4.4.7 Pulse/Edge Accumulation Mode (PEA)

MODE[0:6]	Unified Channel Mode of Operation
0b0001000	Pulse/edge accumulation (continuous)
0b0001001	Pulse/edge accumulation (single shot)

The PEA mode returns the time taken to detect a desired number of input events. MODE[6] bit selects between continuous or single shot operation.

After writing to register A1, the internal counter is cleared on the first input event, ready to start counting input events and the selected timebase is latched into register B2. On the match between the internal counter and register A1, a counter bus capture is triggered to register A2 and B2. The data previously held in register B2 is transferred to register B1 and the FLAG bit is set to indicate that an event has occurred. The desired time interval can be determined subtracting register B1 from A2. Registers EMIOS_CADR*n* and EMIOS_CBDR*n* return the values in register A2 and B1, respectively.

To guarantee coherent access, reading EMIOS_CADR*n* disables transfers between B2 and B1 until reading EMIOS_CBDR*n* register, then any pending transfer is re-enabled.

Triggering of the counter clock (input event) is done by a rising or falling edge or both edges on the input pin. The polarity of the triggering edge is selected by the EDSEL and EDPOL bits in EMIOS_CCR*n*.

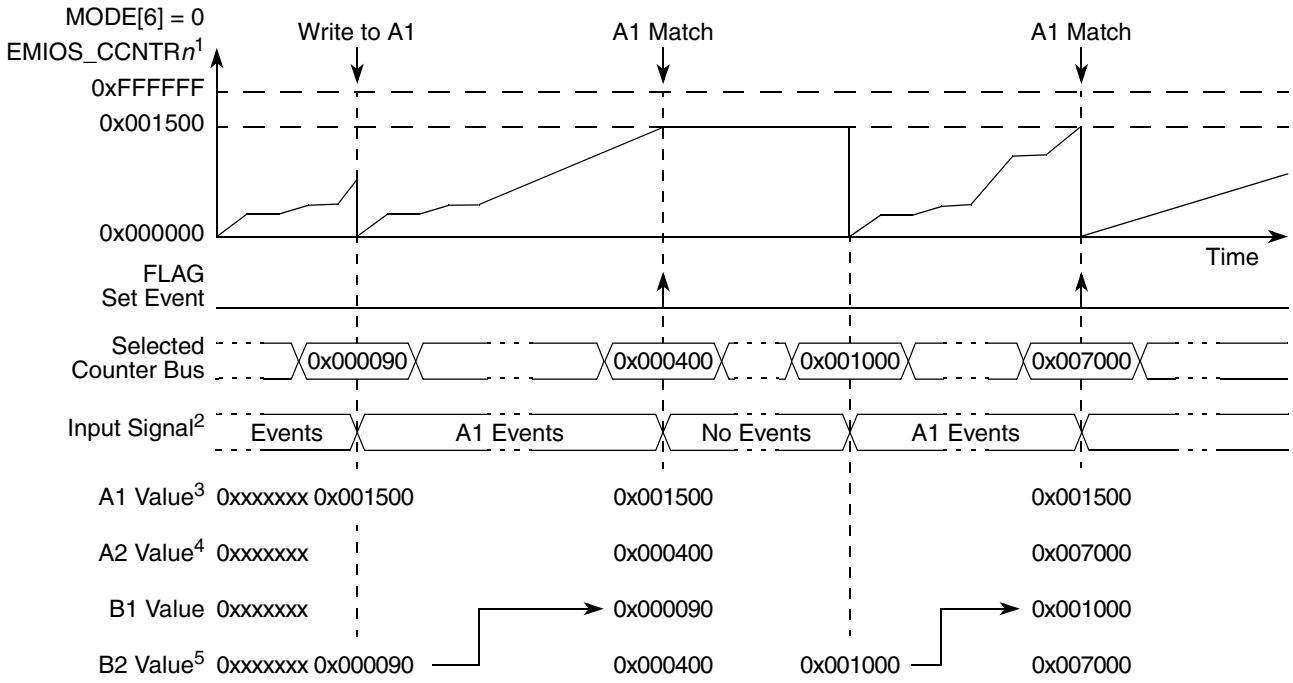
For continuous operating mode (MODE[6] cleared), the counter is cleared on the next input event after a FLAG generation and continues to operate as described above.

For single shot operation (MODE[6] set), the counter is not cleared or incremented after a FLAG generation, until a new writing operation to register A is performed.

NOTE

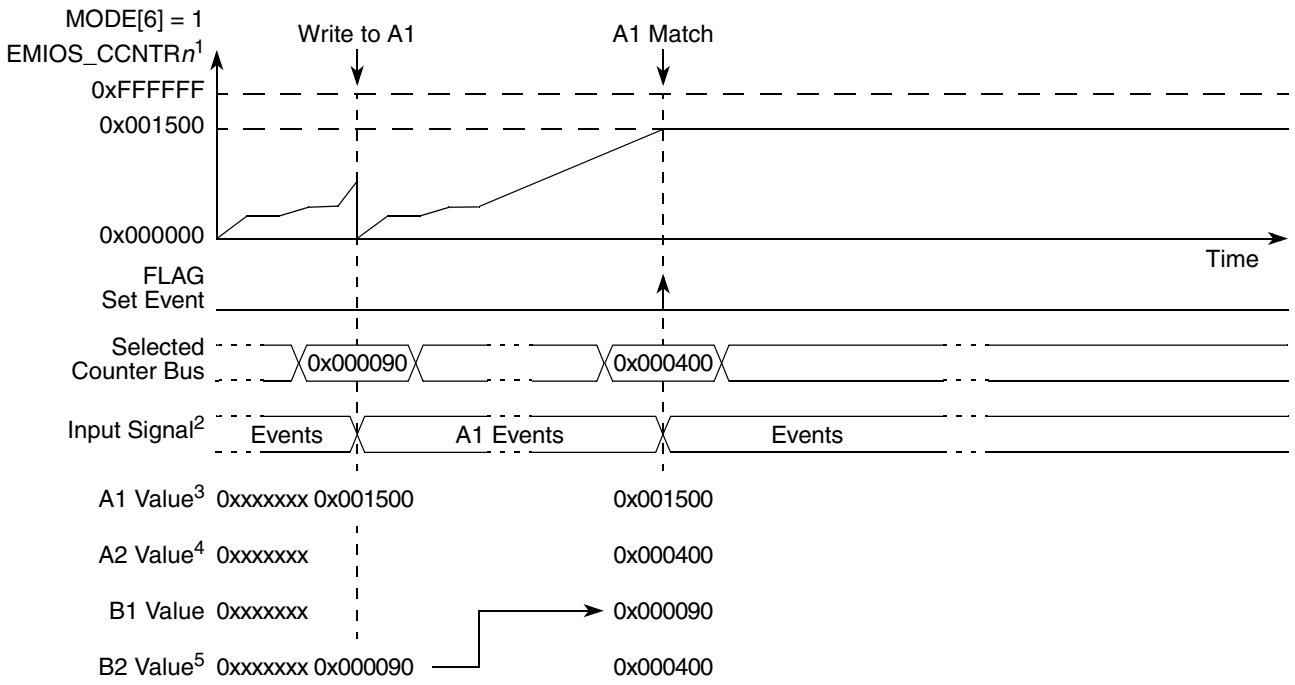
The FORCMA and FORCMB bits have no effect when the unified channel is configured for PEA mode.

Figure 16-22 and Figure 16-23 show how the unified channel can be used for continuous and single shot pulse/edge accumulation mode.



Notes: [1] Cleared on the first input event after writing to register A1.
[2] After input filter.
[3] Writing EMIOS_CADR*n* writes to A1.
[4] Reading EMIOS_CADR*n* returns the value of A2.
[5] Reading EMIOS_CBDR*n* returns the value of B1.

Figure 16-22. Pulse/Edge Accumulation Continuous Mode Example



Notes: [1] Cleared on the first input event after writing to register A1.
[2] After input filter.
[3] Writing EMIOS_CADR*n* writes to A1.
[4] Reading EMIOS_CADR*n* returns the value of A2.
[5] Reading EMIOS_CBDR*n* returns the value of B1.

Figure 16-23. Pulse/Edge Accumulation Single-shot Mode Example

16.4.4.4.8 Pulse/Edge Counting Mode (PEC)

MODE[0:6]	Unified Channel Mode of Operation
0b0001010	Pulse/edge counting (continuous)
0b0001011	Pulse/edge counting (single shot)

The PEC mode returns the amount of pulses or edges detected on the input for a desired time window. MODE[6] bit selects between continuous or single shot operation.

Triggering of the internal counter is done by a rising or falling edge or both edges on the input signal. The polarity and the triggering edge is selected by EDSEL and EDPOL bits in EMIOS_CCR*n*.

Register A1 holds the start time and register B1 holds the stop time for the time window. After writing to register A1, when a match occur between comparator A and the selected timebase, the internal counter is cleared and it is ready to start counting input events. When the time base matches comparator B1, the internal counter is disabled and the FLAG bit is set. Reading the EMIOS_CCNTR*n* returns the amount of detected pulses.

For continuous operation (MODE[6] cleared), the next match between comparator A and the selected time base clears the internal counter and counting is enabled again. To guarantee the accuracy when reading

EMIOS_CCNTR*n* after the flag is set, the software must check if the time base value is out of the time interval defined by registers A1 and B1.

For single shot operation (MODE[6] set), the next match between comparator A and the selected time base has no effect, until a new write to register A is performed.

NOTE

The FORCMA and FORCMB bits have no effect when the unified channel is configured for PEC mode.

Figure 16-24 and Figure 16-25 show how the unified channel can be used for continuous or single shot pulse/edge counting mode.

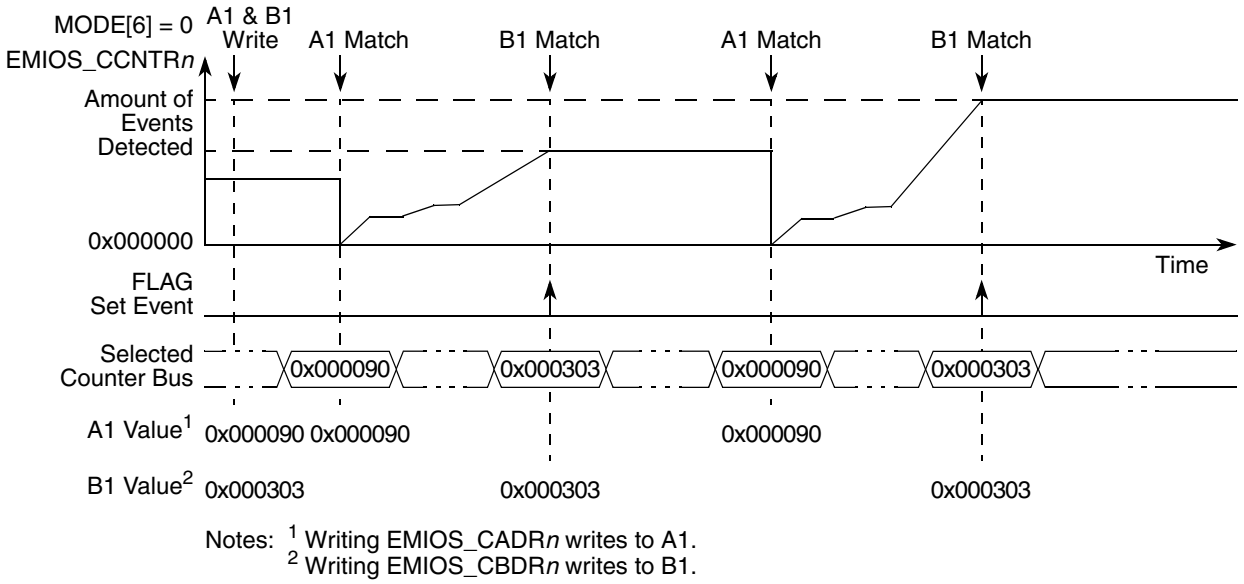


Figure 16-24. Pulse/Edge Counting Continuous Mode Example

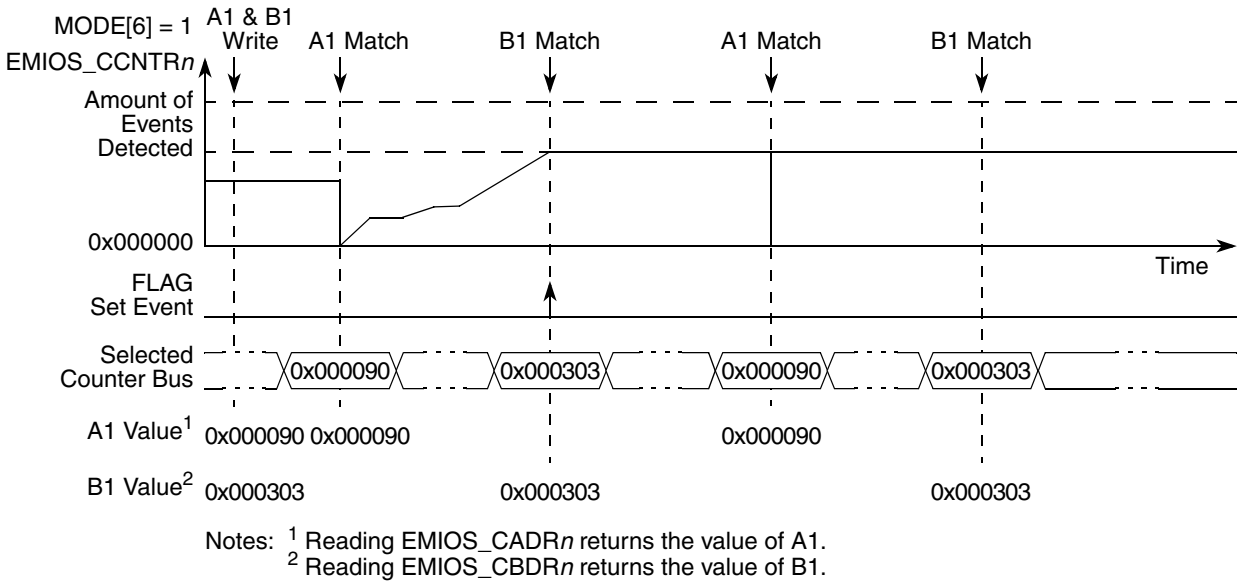


Figure 16-25. Pulse/Edge Counting Single-Shot Mode Example

16.4.4.4.9 Quadrature Decode Mode (QDEC)

MODE[0:6]	Unified Channel Mode of Operation
0b0001100	Quadrature decode (for count and direction encoders type)
0b0001101	Quadrature decode (for phase A and phase B encoders type)

Quadrature decode mode uses UC*n* operating in QDEC mode and the programmable input filter (PIF) from UC[n-1]. UC[n-1] can be configured, at the same time, to an operation mode that does not use I/O pins, such as MC mode (modulus counter). The connection among the UCs is circular; that is, when UC0 is running in QDEC mode, the programmable input filter from UC23 is being used.

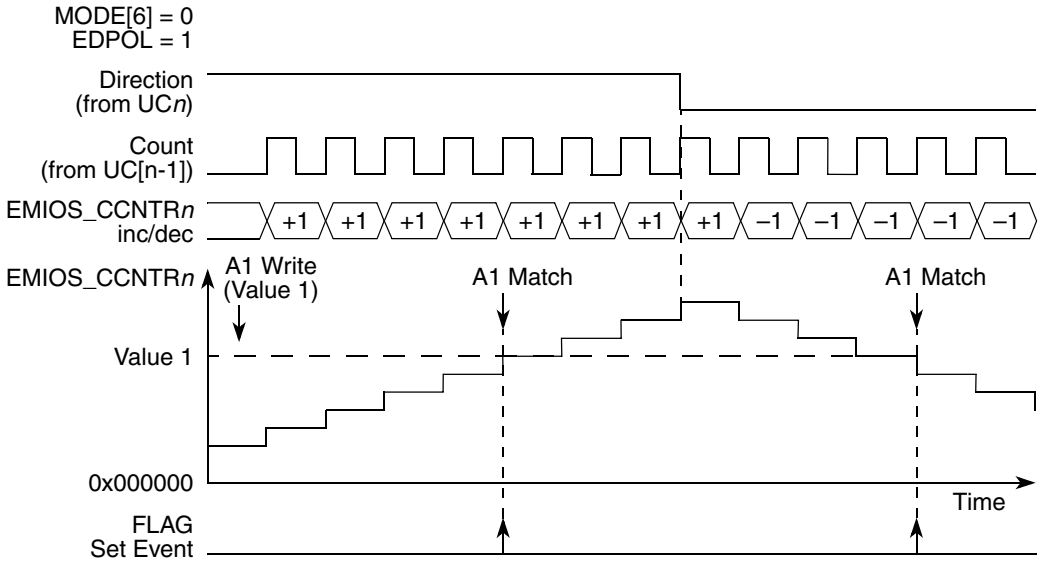
This mode generates a FLAG every time the internal counter matches A1 register. The internal counter is automatically selected and is not cleared when entering this mode.

MODE[6] bit selects which type of encoder is used: count and direction encoder or phase A and phase B encoders.

When operating with count and direction encoder (MODE[6] cleared), UC*n* input pin must be connected to the direction signal and UC[n-1] input pin must be connected to the count signal of the quadrature encoder. UC*n* EDPOL bit selects count direction according to direction signal and UC[n-1] EDPOL bit selects if the internal counter is clocked by the rising or falling edge of the count signal.

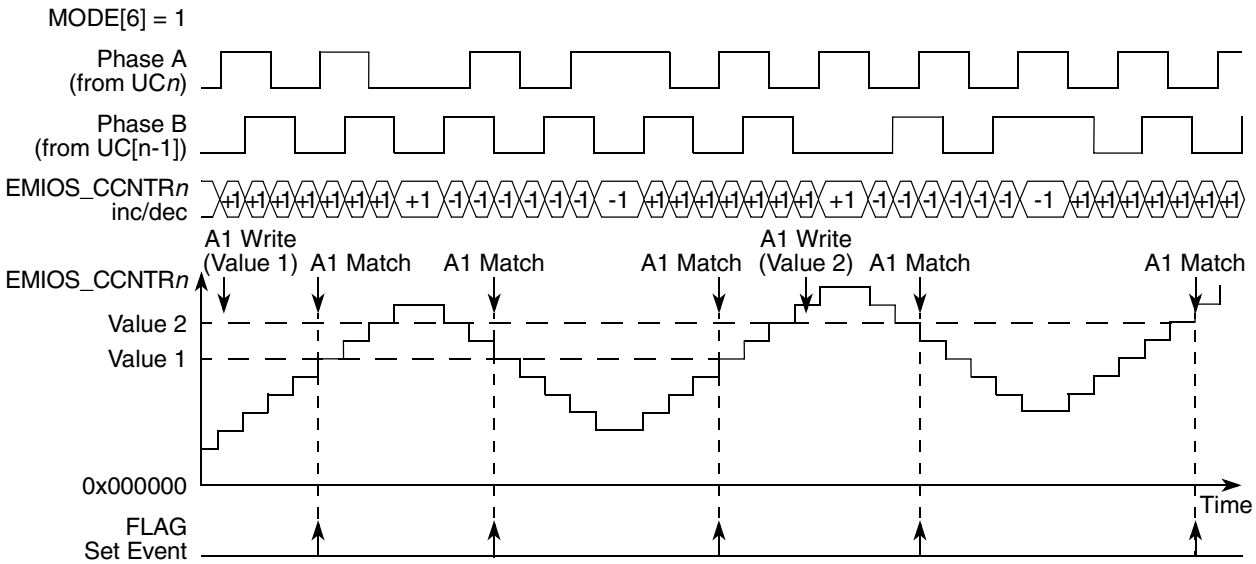
When operating with phase A and phase B encoder (MODE[6] set), UC*n* input pin must be connected to the phase A signal and UC[n-1] input pin must be connected to the phase B signal of the quadrature encoder. EDPOL bit selects the count direction according to the phase difference between phase A and phase B signals.

Figure 16-26 and Figure 16-27 show two unified channels configured to quadrature decode mode for count and direction encoder and phase A and phase B encoders, respectively.



Note: Writing EMIOS_CADR*n* writes to A1.

Figure 16-26. Quadrature Decode Mode Example with Count and Direction Encoder



Note: Writing EMIOS_CADR*n* writes to A1.

Figure 16-27. Quadrature Decode Mode Example with Phase A and Phase B Encoder

16.4.4.4.10 Windowed Programmable Time Accumulation Mode (WPTA)

MODE[0:6] = 0001110

The WPTA mode accumulates the sum of the total high time or low time of an input signal over a programmable interval (time window).

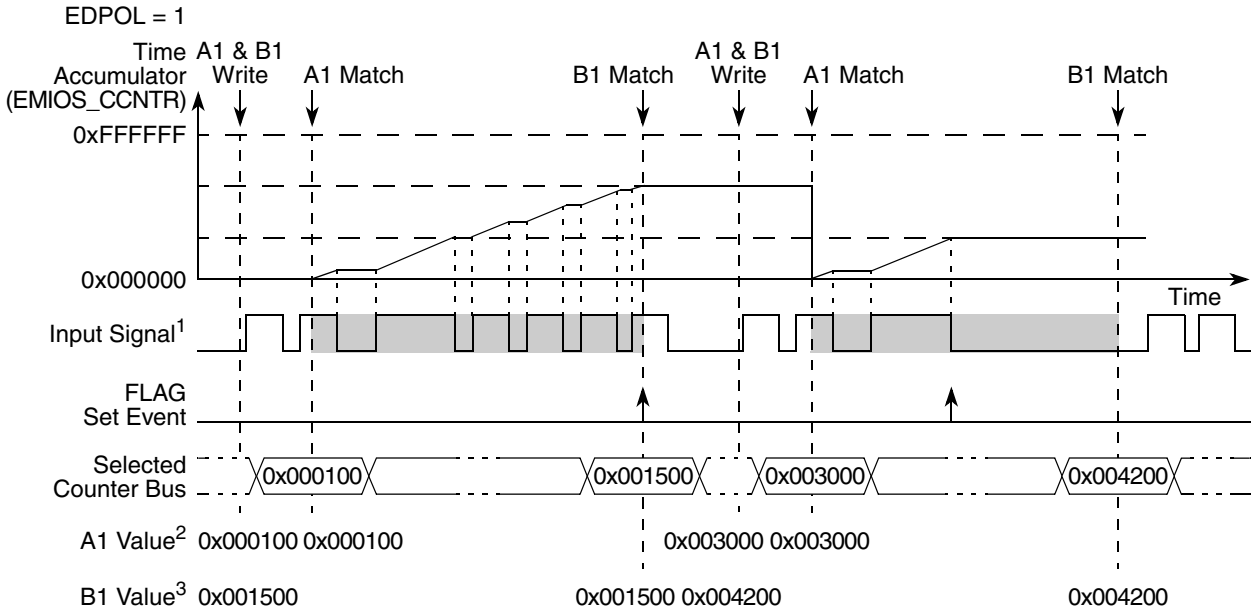
The prescaler bits UCPRE[0:1] in EMIOS_CCR*n* define the increment rate of the internal counter.

Register A1 holds the start time and register B1 holds the stop time of the programmable time interval. When a match occurs between register A and the selected timebase, the internal counter is cleared and it is ready to start counting. The internal counter is used as a time accumulator; that is, it counts up when the input signal has the same polarity of EDPOL bit in EMIOS_CCR*n* and does not count otherwise. When a match occurs in comparator B, the internal counter is disabled regardless of the input signal polarity and the FLAG bit is set. Reading EMIOS_CCNTR*n* returns the high or low time of the input signal.

NOTE

The FORCMA and FORCMB bits have no effect when the unified channel is configured for WPTA mode.

Figure 16-28 shows how the unified channel can be used to accumulate high time.



Notes: 1 After input filter.
2 Writing EMIOS_CADR*n* writes to A1.
3 Writing EMIOS_CBDR*n* writes to B1.

Figure 16-28. Windowed Programmable Time Accumulation Example

16.4.4.4.11 Modulus Counter Mode (MC)

MODE[0:6]	Modulus Counter Operating Modes
0b0010000	Modulus counter: • up counter • internal clock source
0b0010001	Modulus counter: • up counter • external clock source
0b0010010–0b0010011	Reserved

MODE[0:6]	Modulus Counter Operating Modes
0b0010100	Modulus counter: • up/down counter • no change in counter direction upon match of input counter and register B1 • internal clock source
0b0010101	Modulus counter: • up/down counter • no change in counter direction upon match of input counter and register B1 • external clock source
0b0010110	Modulus counter: • up/down counter • change in counter direction upon match of input counter and register B1 and sets the FLAG • internal clock source
0b0010111	Modulus counter: • up/down counter • change in counter direction upon match of input counter and register B1 and sets the FLAG • external clock source

The MC mode can be used to provide a time base for a counter bus or as a general purpose timer.

MODE[6] bit selects internal or external clock source when cleared or set, respectively. When external clock is selected, the input signal pin is used as the source and the triggering polarity edge is selected by the EDPOL and EDSEL in the EMIOS_CCR*n*.

When software selects the modulus counter mode, the internal counter is initially reset to 0. The internal counter counts up from the current value until it matches the value in register A1. Register B1 is cleared and is not accessible to the MCU. MODE[4] bit selects up mode or up/down mode, when cleared or set, respectively.

When in up count mode, a match between the internal counter and register A1 sets the FLAG and clears the internal counter.

When in up/down count mode, a match between the internal counter and register A1 sets the FLAG and changes the counter direction from increment to decrement. A match between register B1 and the internal counter changes the counter direction from decrement to increment and sets the FLAG only if MODE[5] bit is set.

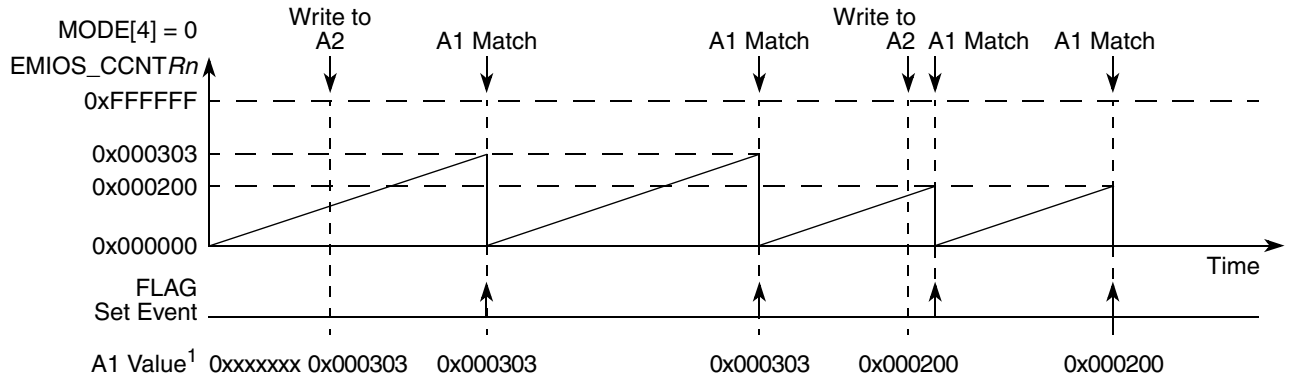
NOTE

The FORCMA and FORCMB bits have no effect when the unified channel is configured for MC mode.

NOTE

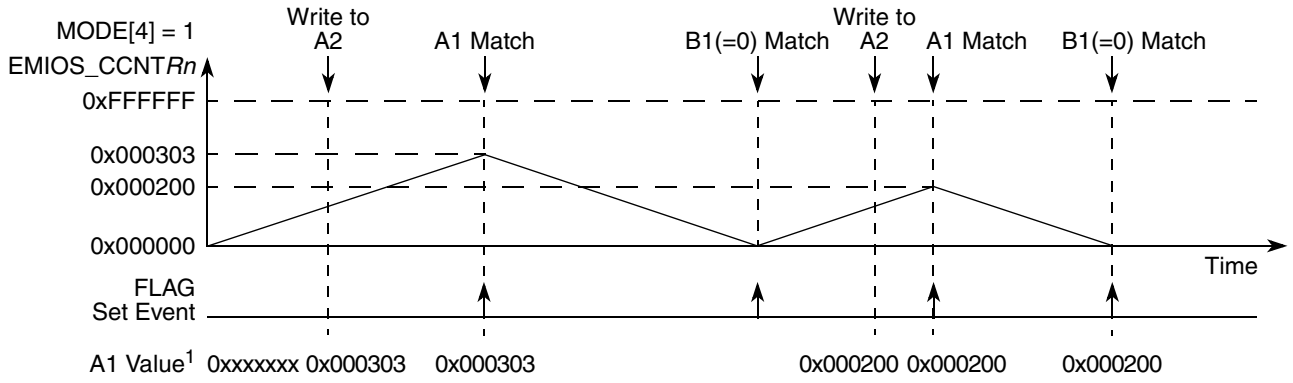
Any update to the A register occurs immediately, regardless of the current state of the counter and whether the counter is in up mode, or up/down mode.

Figure 16-29 and Figure 16-30 shows how the unified channel can be used as modulus counter in up mode and up/down mode, respectively.



Notes: [1] Writing EMIOS_A*n* writes to A2.
A2 value transferred to A1 according to OU*n* bit.

Figure 16-29. Modulus Counter Up Mode Example



Notes: [1] Writing EMIOS_A*n* writes to A2.
A2 value transferred to A1 according to OU*n* bit.

Figure 16-30. Modulus Counter Up/Down Mode Example

16.4.4.4.12 Output Pulse Width and Frequency Modulation Mode (OPWFM)

MODE[0:6]	Unified Channel Mode of Operation
0b0011000	Output pulse width and frequency modulation (FLAG set at match of internal counter and comparator B, immediate update)
0b0011001	Output pulse width and frequency modulation (FLAG set at match of internal counter and comparator B, next period update)
0b0011010	Output pulse width and frequency modulation (FLAG set at match of internal counter and comparator A or comparator B, immediate update)
0b0011011	Output pulse width and frequency modulation (FLAG set at match of internal counter and comparator A or comparator B, next period update)

In this mode, register A1 contains the duty cycle and register B1 contains the period of the output signal. MODE[6] bit controls the transfer from register B2 to B1, which can be done either immediately (MODE[6] cleared), providing the fastest change in the duty cycle, or at every match of register A1 (MODE[6] set).

The internal counter is automatically selected as a time base, therefore the BSL[0:1] bits in register EMIOS_CCR*n* have no meaning. The output flip-flop's active state is the complement of EDPOL bit. The output flip-flop is active during the duty cycle (from the start of the cycle until a match occurs in comparator A). After the match in comparator A the output flip-flop is in the inactive state (the value of EDPOL) until the next cycle starts. When a match on comparator A occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit and the internal counter is cleared.

FLAG can be generated at match B, when MODE[5] is cleared, or in both matches, when MODE[5] is set.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Also, FORCMB clears the internal counter. The FLAG bit is not set by the FORCMA or FORCMB operations.

If subsequent comparisons occur on comparators A and B, the PWFM pulses continue to be output, regardless of the state of the FLAG bit.

To achieve 0% duty cycle, both registers A1 and B1 must be set to the same value. When a simultaneous match occurs on comparators A and B, the output flip-flop is set at every period to the value of EDPOL bit.

To temporarily change from the current duty cycle to 0% and then return to the current duty cycle, the sequence is the following:

1. If not currently stored, store value of register A.
2. Set A=B.
3. If immediate 0% duty cycle is desired, set FORCA=1.
4. To return to the previous duty cycle, restore register A with its former value.

100% duty cycle is possible by writing 0x000000 to register A. When a match occurs, the output flip-flop is set at every period to the complement of EDPOL bit. The transfer from register B2 to B1 is still controlled by MODE[6] bit.

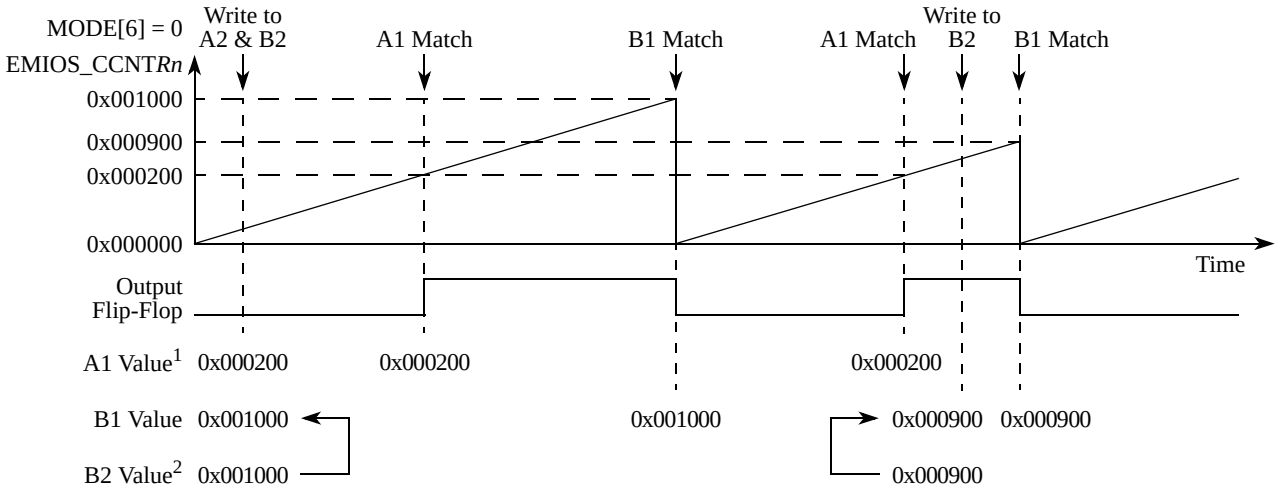
To temporarily change from the current duty cycle to 100% and then return to the current duty cycle, the sequence is the following:

1. If not currently stored, store value of register A.
2. Set A=0.
3. If immediate 100% duty cycle is desired, set FORCB=1.
4. To return to the previous duty cycle, restore register A with its former value.

NOTE

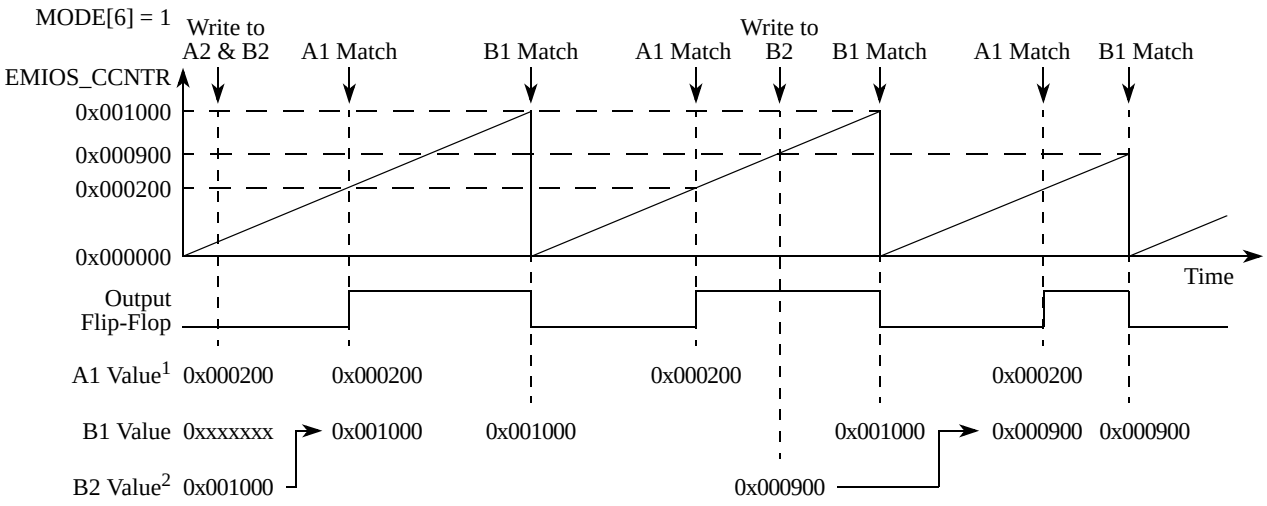
Updates to the A register always occur immediately. If next period update is selected via the mode[6] bit, only the B register update is delayed until the next period.

Figure 16-31 shows the unified channel running in OPFWM mode with immediate register update and Figure 16-32 shows the unified channel running in OPFWM mode with next period update PFWM mode. In both figures EDPOL = 1, so the output is low during the duty cycle. Table 16-13 has additional illustrative examples.



Notes: [1] Writing EMIOS_A*n* writes to A2.
[2] Writing EMIOS_B*n* writes to B2.
A2 value transferred to A1 according to OU*n* bit.
B2 value transferred to B1 according to OU*n* bit.

Figure 16-31. OPWFM with Immediate Update



Notes: [1] Writing EMIOS_A*n* writes to A2.
[2] Writing EMIOS_B*n* writes to B2.
A2 value transferred to A1 according to OU*n* bit.
B2 value transferred to B1 according to OU*n* bit.

Figure 16-32. OPWFM with Next Period Update

Table 16-13. Examples of Output Waveforms

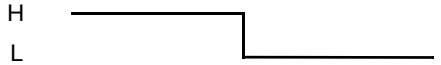
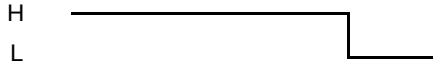
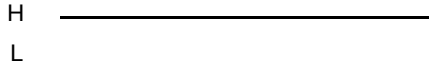
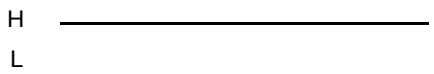
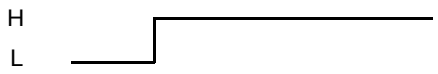
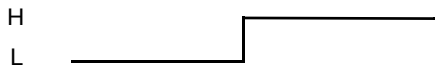
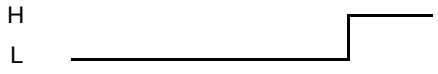
EDPOL	Duty Cycle	A (decimal)	B (decimal)	Waveform
0 (active high output)	0%	1000	1000	
	25%	250	1000	
	50%	500	1000	
	75%	750	1000	
	100%	0	1000	

Table 16-13. Examples of Output Waveforms

EDPOL	Duty Cycle	A (decimal)	B (decimal)	Waveform
1 (active low output)	0%	1000	1000	
	25%	250	1000	
	50%	500	1000	
	75%	750	1000	
	100%	0	1000	

16.4.4.4.13 Center Aligned Output Pulse Width Modulation with Dead-time Mode (OPWMC)

MODE[0:6]	Unified Channel Mode of Operation
0b0011100	Center aligned output pulse width modulation (FLAG set in trailing edge, trailing edge dead-time)
0b0011101	Center aligned output pulse width modulation (FLAG set in trailing edge, leading edge dead-time)
0b0011110	Center aligned output pulse width modulation (FLAG set in both edges, trailing edge dead-time)
0b0011111	Center aligned output pulse width modulation (FLAG set in both edges, leading edge dead-time)

This operating mode generates a center aligned PWM with dead time insertion in the leading or trailing edge.

The selected counter bus must be running an up/down time base, as shown in Figure 16-30. BSL[0:1] bits select the time base. Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base. Register B1 contains the dead time value and is compared with the internal counter. For a leading edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. MODE[6] bit selects between trailing and leading dead time insertion, respectively.

NOTE

Set the internal prescaler of the OPWMCB channel to the same value as the MCB channel prescaler, and the prescalers must be synchronized. This allows the A1 and B1 registers to represent the same time scale for duty cycle and dead time insertion.

When operating with leading edge dead time insertion, the first match between A1 and the selected time base clears the internal counter and switches the selected time base to the internal counter. When a match occurs between register B1 and the selected time base, the output flip-flop is set to the value of the EDPOL bit and the time base is switched to the selected counter bus. In the next match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

When operating with trailing edge dead time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit. In the next match between register A1 and the selected time base, the internal counter is cleared and the selected time base is switched to the internal counter. When a match occurs between register B1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit and the time base is switched to the selected counter bus. This sequence repeats continuously.

FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or in both edges, when MODE[5] is set.

At any time, the FORCMA or FORCMB bits are equivalent to a successful comparison on comparator A or B with the exception that the FLAG bit is not set.

NOTE

When in freeze mode, the FORCMA or FORCMB bits only allow the software to force the output flip-flop to the level corresponding of a match on A or B respectively.

If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

To achieve a duty cycle of 100%, both registers A1 and B1 must be set to the same value. When a simultaneous match occurs between the selected time base and registers A1 and B1, the output flip-flop is set at every period to the value of EDPOL bit and the selected time base switches to the selected counter bus, allowing a new cycle to begin at any time, as previously described. 0% duty cycle is possible by writing 0x000000 to register A. When a match occurs, the output flip-flop is set at every period to the complement of EDPOL bit and the selected time base switches to the selected counter bus, allowing a new cycle to begin at any time, as previously described. In both cases, FLAG is generated regardless of MODE[5] bit.

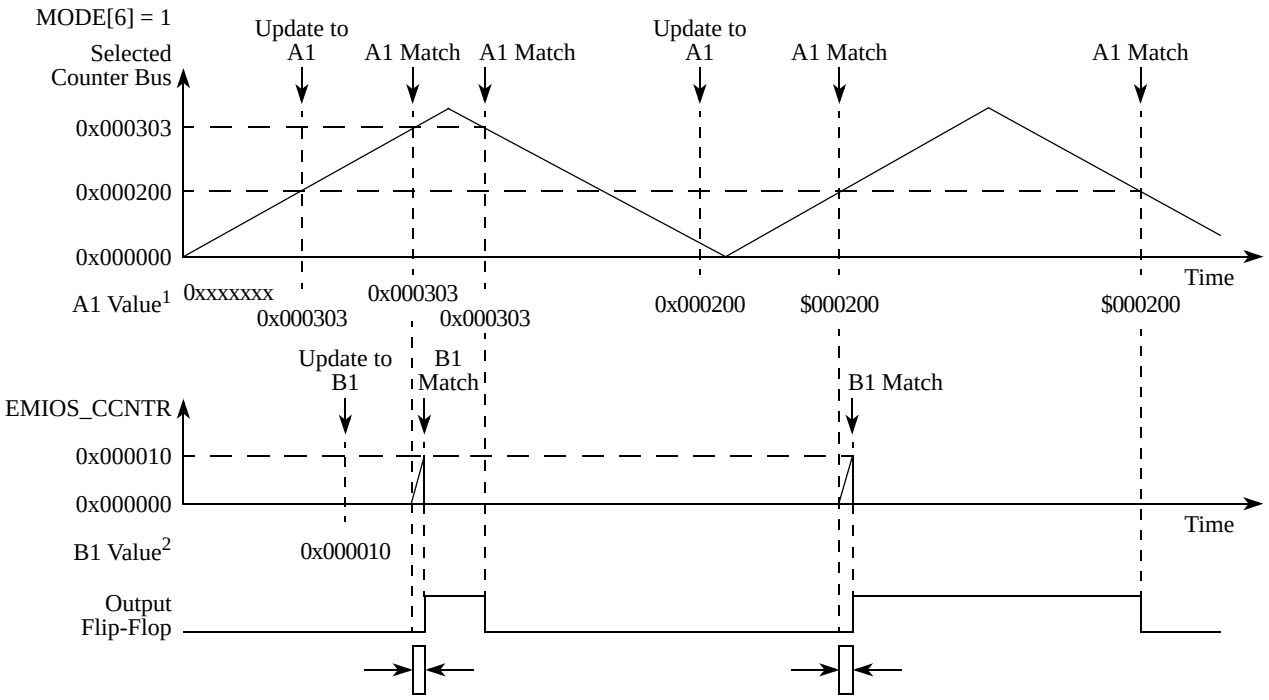
NOTE

If A1 and B1 are set to the value 0x000000, a 0% duty cycle waveform is produced.

NOTE

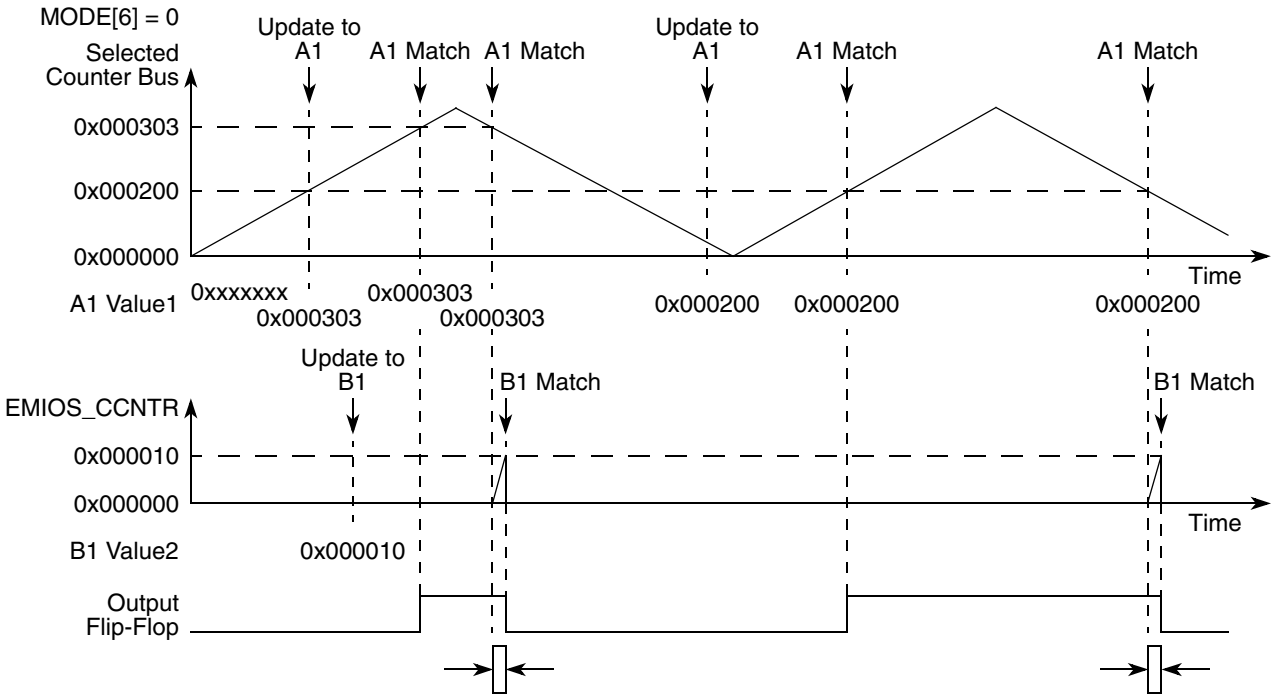
Any updates to the A or B register occurs immediately.

Figure 16-33 and Figure 16-34 show the unified channel running in OPWMC with leading and trailing dead time, respectively.



Notes: [1] Writing EMIOS_A*n* writes to A2.
[2] Writing EMIOS_B*n* writes to B1.
A2 value transferred to A1 according to OU*n* bit.
B2 value transferred to B1 according to OU*n* bit.

Figure 16-33. Output PWMC with Leading Dead-time Insertion



Notes: 1 Writing EMIOS_An writes to A2.
2 Writing EMIOS_Bn writes to B1.
A2 value transferred to A1 according to OUn bit.
B2 value transferred to B1 according to OUn bit.

Figure 16-34. Output PWMC with Trailing Dead-time Insertion

16.4.4.4.14 Output Pulse Width Modulation Mode (OPWM)

MODE[0:6]	Unified Channel Mode of Operation
0b0100000	Output pulse width modulation (FLAG set at match of internal counter and comparator B, immediate update)
0b0100001	Output pulse width modulation (FLAG set at match of internal counter and comparator B, next period update)
0b0100010	Output pulse width modulation (FLAG set at match of internal counter and comparator A or comparator B, immediate update)
0b0100011	Output pulse width modulation (FLAG set at match of internal counter and comparator A or comparator B, next period update)

Registers A1 and B1 define the leading and trailing edges of the PWM output pulse, respectively. MODE[6] bit controls the transfer from register B2 to B1, which can be done either immediately (MODE[6] cleared), providing the fastest change in the duty cycle, or at every match of register A1 (MODE[6] set).

The value loaded in register A1 is compared with the value on the selected time base. When a match on comparator A occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit.

FLAG can be generated at match B, when MODE[5] is cleared, or in both matches, when MODE[5] is set.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. The FLAG bit is not set by the FORCMA and FORCMB operations.

If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

To achieve 0% duty cycle, registers A1 and B1 must be set to the same value. When a simultaneous match on comparators A and B occur, the output flip-flop is set at every period to the value of EDPOL bit. 0% duty cycle is possible by writing 0x000000 to register A. When a match occurs, the output flip-flop is set at every period to the complement of EDPOL bit. The transfer from register B2 to B1 is still controlled by MODE[6] bit.

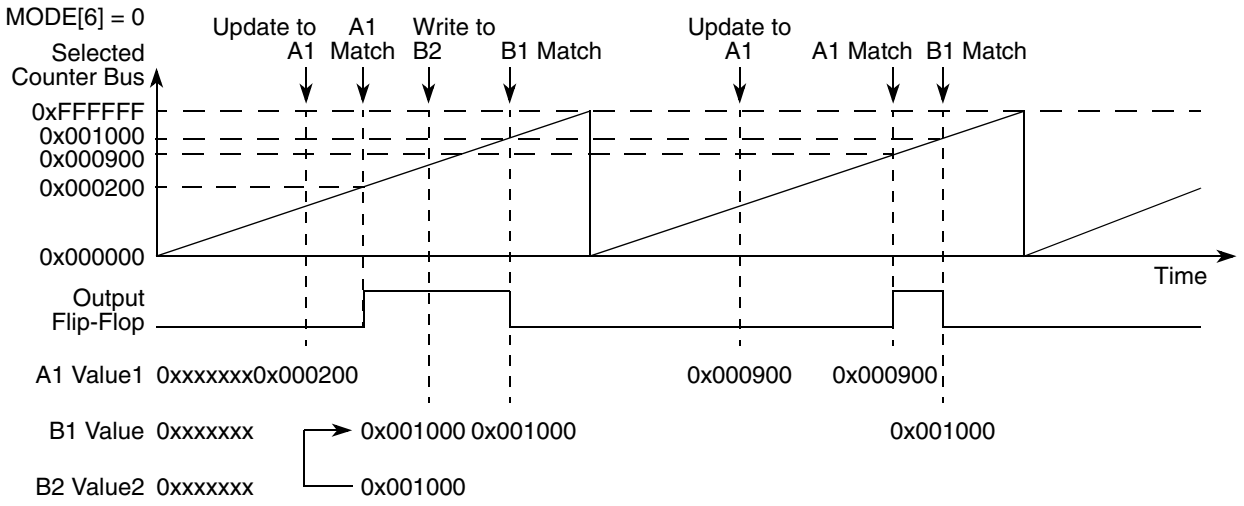
NOTE

If A1 and B1 are set to the value 0x000000, a 100% duty cycle waveform is produced.

NOTE

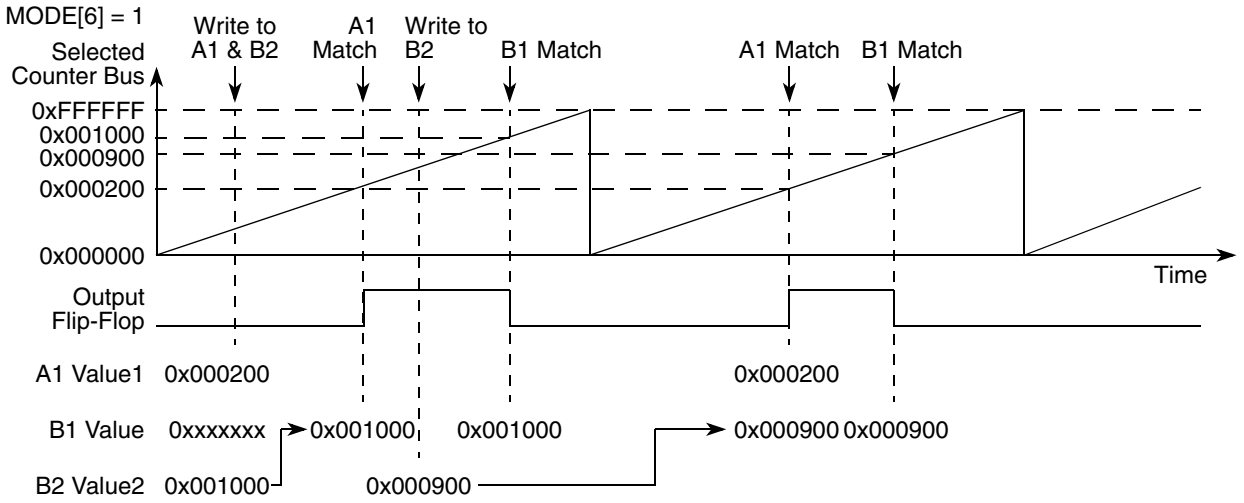
Updates to the A register always occur immediately. If next period update is selected via the mode[6] bit, only the B register update is delayed until the next period.

Figure 16-35 and Figure 16-36 show the unified channel running in OPWM with immediate update and next period update, respectively.



Notes: 1 Writing EMIOS_A*n* writes to A2.
2 Writing EMIOS_B*n* writes to B2.
A2 value transferred to A1 according to OU*n* bit.
B2 value transferred to B1 according to OU*n* bit.

Figure 16-35. Output PWM with Immediate Update



Notes: 1 Writing EMIOS_A*n* writes to A2.
2 Writing EMIOS_B*n* writes to B2.
A2 value transferred to A1 according to OU*n* bit.
B2 value transferred to B1 according to OU*n* bit.

Figure 16-36. Output PWM with Next Period Update

16.4.4.4.15 Modulus Counter, Buffered Mode (MCB)

MODE[0:6]	Unified Channel Mode of Operation
0b1010000	Modulus up counter, buffered, internal clock
0b1010001	Modulus up counter, buffered, external clock
0b1010010–0b1010001	Reserved
0b1010100	Modulus up/down counter, buffered (FLAG set on one event, internal clock)
0b1010101	Modulus up/down counter, buffered (FLAG set on one event, external clock)
0b1010110	Modulus up/down counter, buffered (FLAG set on both events, internal clock)
0b1010111	Modulus up/down counter, buffered (FLAG set on both events, external clock)

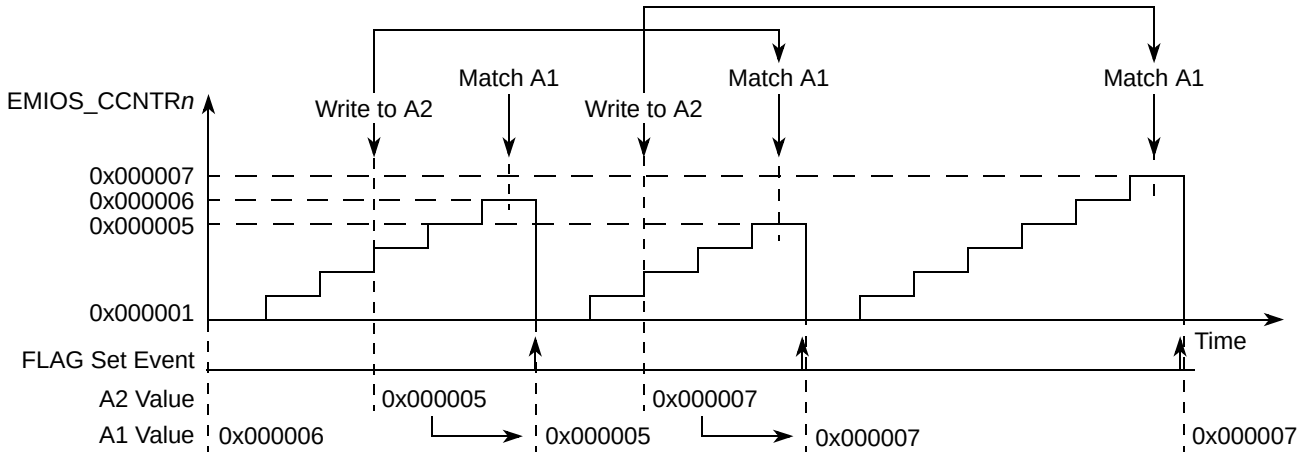
The MCB mode provides a time base which can be shared with other channels through the internal counter buses. Register A1 is double buffered, thus allowing smooth transitions between cycles when changing the A2 register value during operation. The A1 register is updated at the cycle boundary, which is defined as when the internal counter reaches the value one. The internal counter values are within a range from one up to register A1 value in MCB mode.

The MODE[6] bit selects the internal clock source if clear or external if set. When an external clock is selected, the channel input pin is used as the channel clock source. The active edge of this clock is defined by EDPOL and EDSEL bits in the EMIOS_CCR channel register.

When entering the MCB mode, if up counter is selected (MODE[4] = 0), the internal counter starts counting up from its current value until an A1 match occurs. On the next system clock cycle after an A1 match occurs, the internal counter is set to one and the counter continues counting up. If up/down mode is selected (MODE[4] = 1), the counter changes direction at the A1 match and counts down until it reaches one and is then set to count up again. In this mode B1 is set to one and cannot be changed, as it is used to generate a match to switch from down count to up count.

Versus the MC mode, the MCB mode counts between one and the A1 register value. The counter cycle period in up count mode is equal to the A1 value. In up/down counter mode the period is defined by the formula: $(2 \times A1) - 2$.

Figure 16-37 illustrates the counter cycle for several A1 values. Register A1 is loaded with the A2 value at the cycle boundary. Thus any value written to A2 within cycle (n) are updated to A1 at the next cycle boundary, and therefore are used on cycle (n+1). The cycle boundary between cycle (n) and cycle (n+1) is defined as the first clock cycle of cycle (n+1). Flags are set when A1 matches occur.



Note: A2 value transferred to A1 according to OU*n* bit.

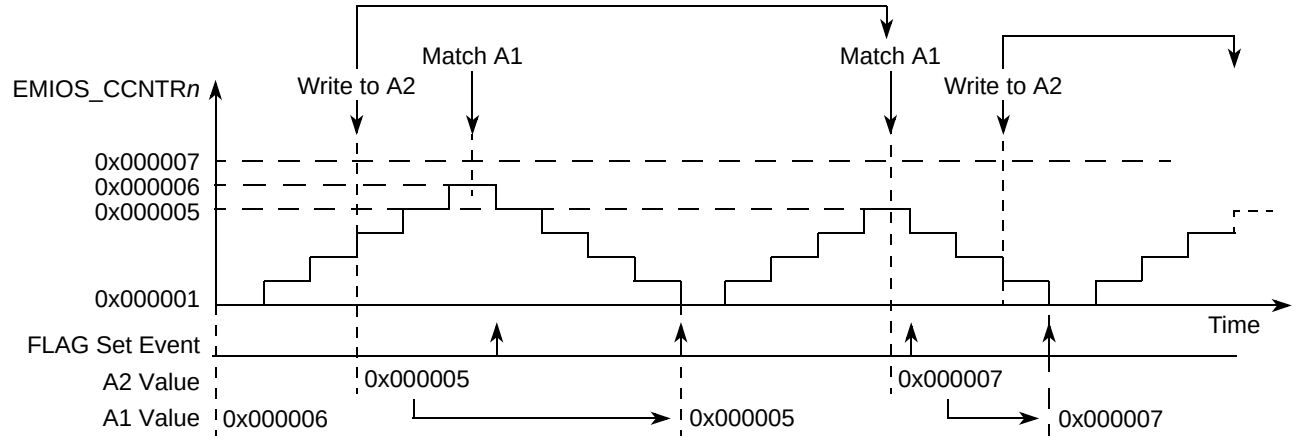
Figure 16-37. eMIOS MCB Mode Example — Up Operation

NOTE

If a prescaler greater than 1 is used, there are several system clock cycles between when the flag is asserted and the counter is set to one. You must consider this when the A value is changed every cycle, because A1 is updated on the cycle boundary, which is after the flag is set.

Figure 16-38 illustrates the MCB up/down counter mode. The A1 register is updated at the cycle boundary. If A2 is written in cycle (*n*), this new value is used in cycle (*n*+1) for the next A1 match.

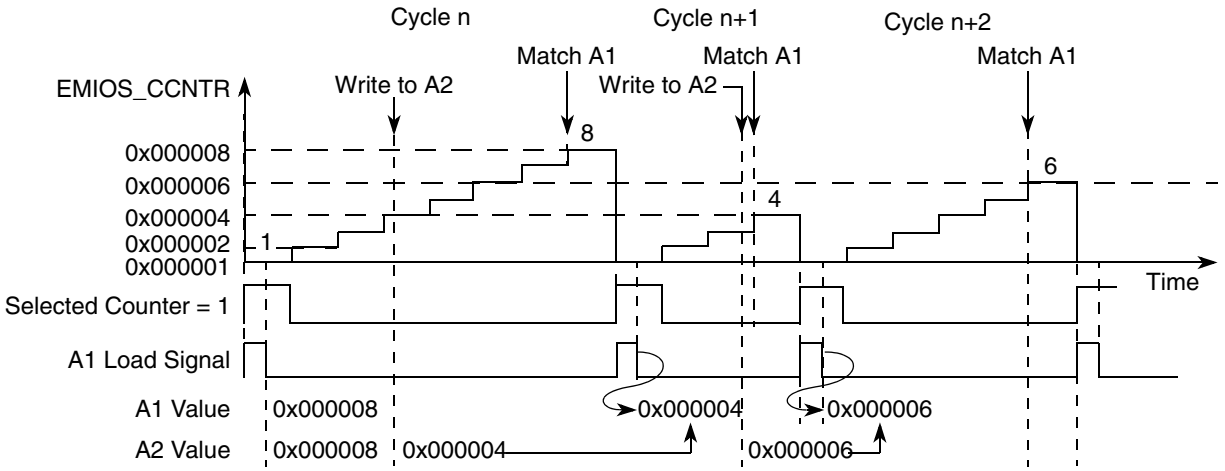
Flags are generated only at an A1 match if MODE[5] is 0. If MODE[5] is 1, flags are also generated at the cycle boundary.



Note: A2 value transferred to A1 according to OU*n* bit.

Figure 16-38. eMIOS MCB Mode Example — Up/Down Operation

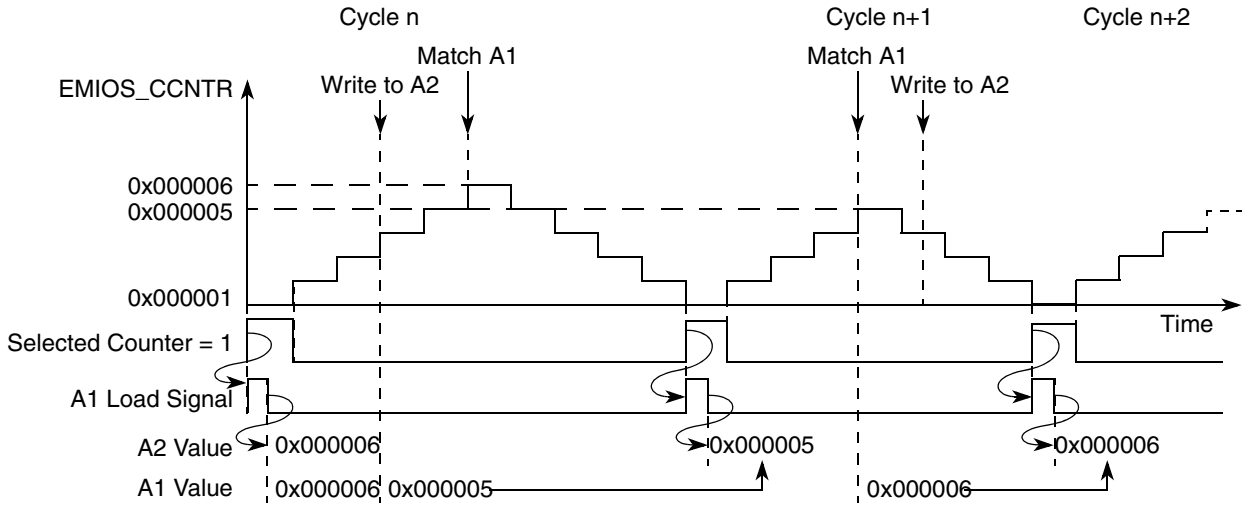
Figure 16-39 provides a more detailed illustration of the A1 update process in up counter mode. The A1 load signal is generated based on the detection of the internal counter reaching one, and has the duration of one system clock cycle. During the load pulse A1 still holds its previous value. It is actually updated at the second system clock cycle.



A2 value transferred to A1 according to OU*n* bit.

Figure 16-39. eMIOS MCB Mode Example — Up Operation A1 Register Update

Figure 16-40 illustrates the A1 register update process in up/down counter mode. A2 can be written at any time within cycle (*n*) to use in cycle (*n*+1). Thus A1 receives the new value at the next cycle boundary. The EMIOS_OUDR[n] bits can be used to disable the update of A1 register.



A2 value transferred to A1 according to OU*n* bit (the transfer is triggered by the A1 load signal)

Figure 16-40. eMIOS MCB Mode Example — Up/Down Operation A1 Register Update

16.4.4.4.16 Output Pulse Width and Frequency Modulation, Buffered Mode (OPWFMB)

MODE[0:6]	Unified Channel Mode of Operation
0b1011000	Output pulse width and frequency modulation, buffered (FLAG set at match of internal counter and comparator B)
0b1011001	Reserved
0b1011010	Output pulse width and frequency modulation, buffered (FLAG set at match of internal counter and comparator A or comparator B)

This mode generates waveforms with variable duty cycle and frequency. The internal channel counter is automatically selected as the time base, A1 sets the duty cycle and B1 determines the frequency. Both A1 and B1 are double buffered to allow smooth signal generation when changing the register values during operation. 0% and 100% duty cycles are supported.

To provide smooth and consistent channel operation, this mode differs substantially from the OPWFM mode. The main differences are in how A1 and B1 are updated, the delay from the A1 match to the output flip-flop transition, and the range of the internal counter which ranges from 1 up to B1 value.

When a match on comparator A occurs, the output register is set to the value of EDPOL. When a match on comparator B occurs, the output register is set to the complement of EDPOL. A B1 match also causes the internal counter to transition to 1, thus re-starting the counter cycle.

Figure 16-41 shows an example of OPWFMB mode operation. The output flip-flop transition occurs when the A1 or B1 match signal is negated, as detected by the negative edge of the A1 and B1 match signals. For example, if register A1 is set to 0x000004, the output flip-flop transitions 4 counter periods after the cycle starts, plus one system clock cycle. In Figure 16-41 the prescaler ratio is set to two (see Section 16.5.3, “Time Base Generation).

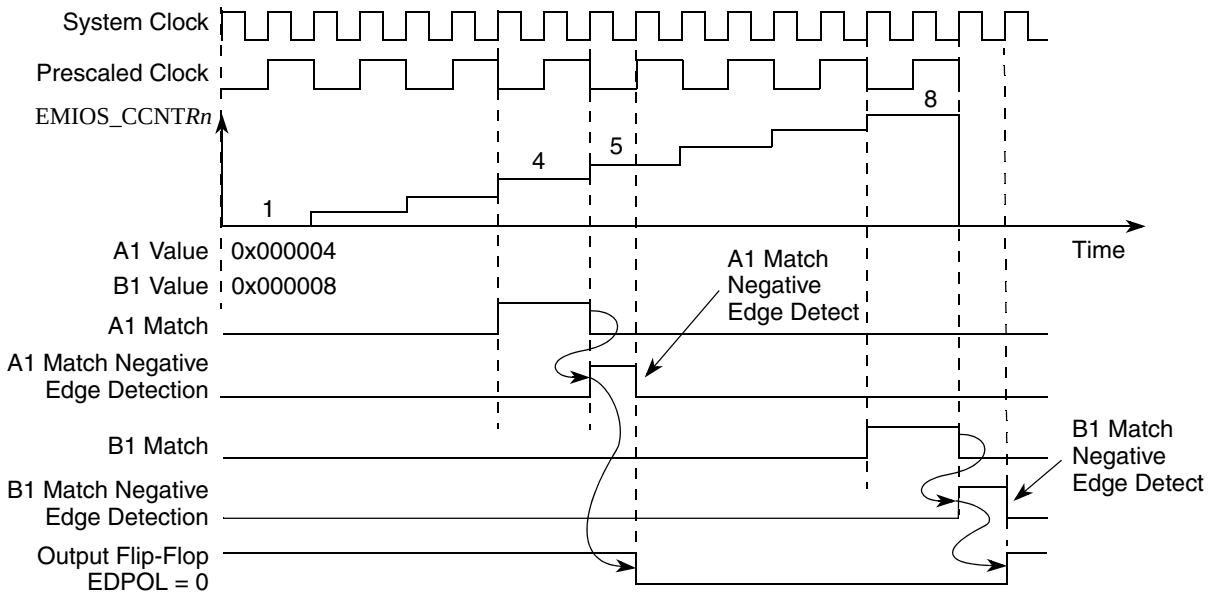


Figure 16-41. eMIOS OPWFMB Mode Example — A1/B1 Match to Output Register Delay

Figure 16-42 shows the generated output signal if A1 is 0. Since the counter does not reach zero in this mode, the channel internal logic infers a match as if A1 = 1, with the difference that in this case the positive edge of the match signal is used to trigger the output flip-flop transition instead of the negative edge that is used when A1 = 1. The A1 positive edge match signal from cycle (*n*+1) occurs at the same time as the B1 match negative edge from cycle (*n*). This allows the use of the A1 match positive edge to mask the B1 match negative edge when they occur at the same time. The result is that no transition occurs on the output flip-flop, and a 0% duty cycle is generated.

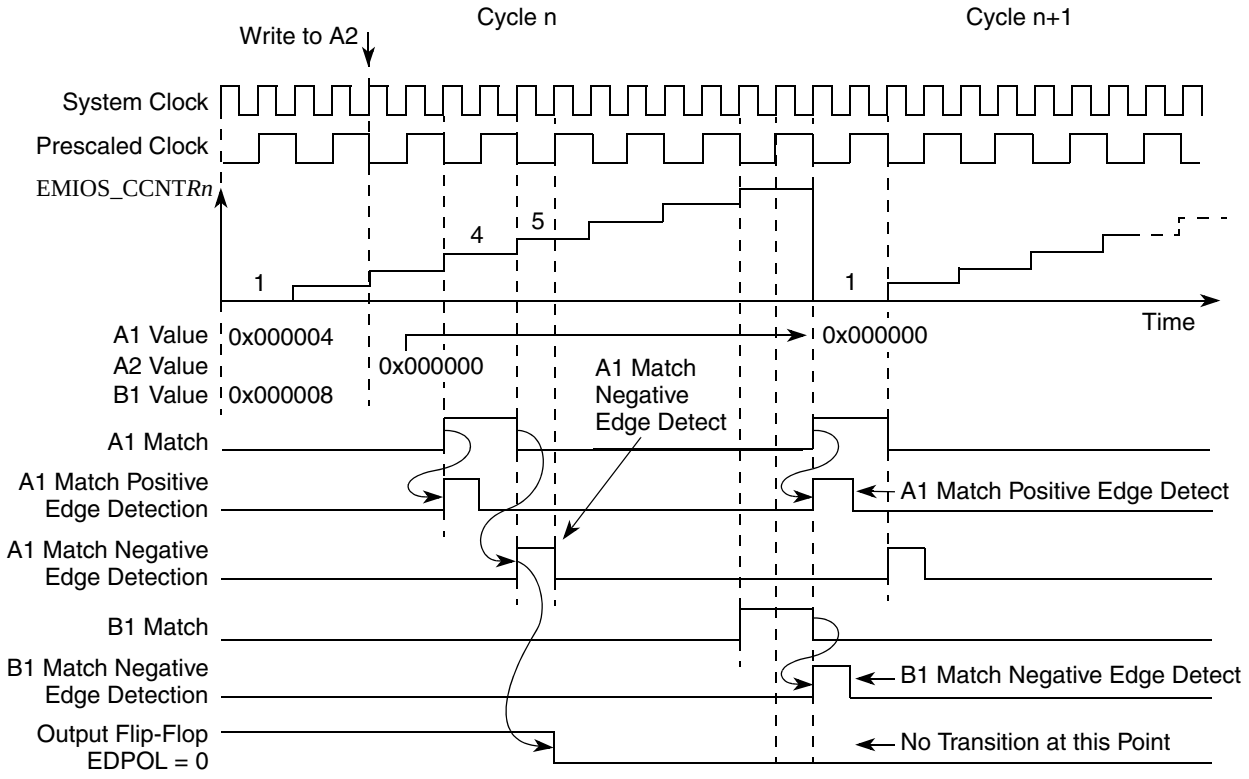


Figure 16-42. eMIOS OPWFMB Mode Example — A1 = 0 (0% Duty Cycle)

Figure 16-43 shows the timing for the A1 and B1 loading. A1 and B1 use the same signal to trigger a load, which is generated based on the selected counter reaching one. This event is defined as the cycle boundary. The load signal pulse has the duration of one system clock cycle and occurs at the first system clock period of every cycle of the counter. If A2 and B2 are written within cycle (*n*), their values are loaded into A1 and B1, respectively, at the first clock of cycle (*n*+1). The update disable bits, EMIOS_OUDR, can be used to control the update of these registers, thus allowing the delay of A1 and B1 update for synchronization purposes.

During the load pulse A1 still holds its old value, which is updated on the following system clock cycle. During the A1 load pulse, an internal by-pass allows the use of A2 instead of A1 for matches if A2 is either 0 or 1, thus allowing matches to be generated even when A1 is being loaded. This approach allows a uniform channel operation for any A2 value, including 1 and 0.

In Figure 16-43 it is assumed that the channel and global prescalers are set to one, meaning that the channel internal counter transition at every system clock cycle. FLAGs can be generated only on B1 matches when

MODE[5] is cleared, or on both A1 and B1 matches when MODE[5] is set. Since B1 FLAG occurs at the cycle boundary, this flag can be used to indicate that A2 or B2 data written on cycle (*n*) were loaded to A1 or B1, respectively, thus generating matches in cycle (*n*+1).

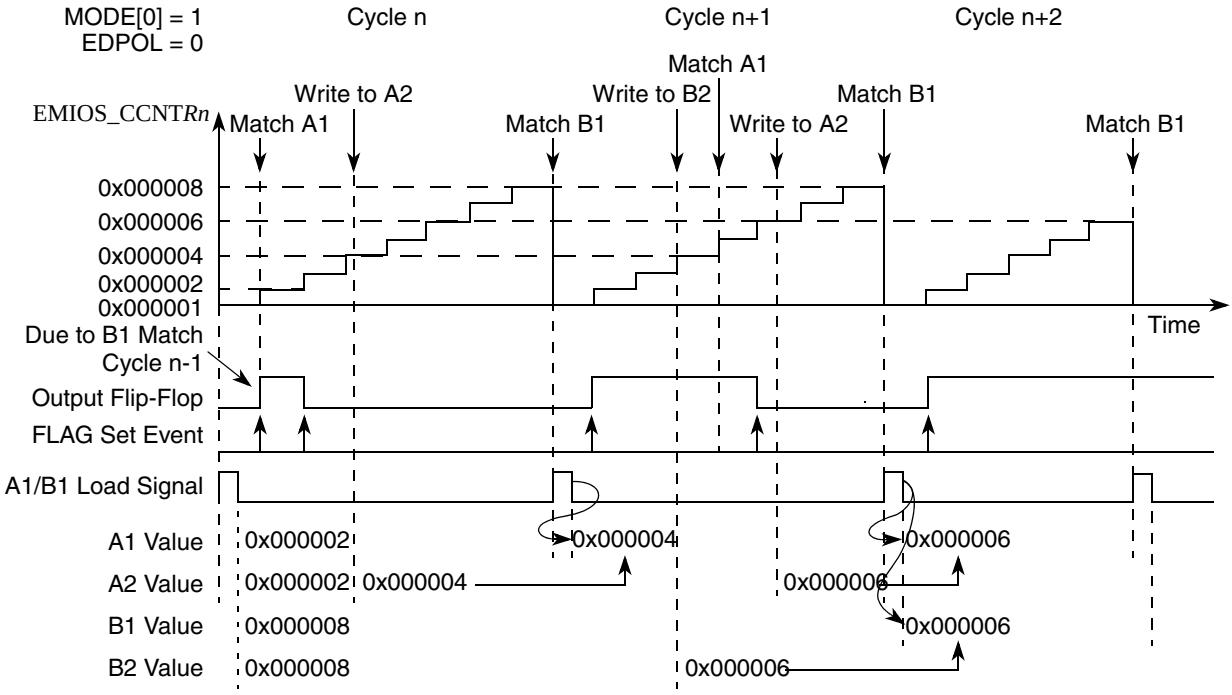


Figure 16-43. eMIOS OPWFMB Mode Example — A1/B1 Updates and Flags

Figure 16-44 shows the operation of the output disable feature in OPWFMB mode. Unlike OPWFM mode, the output disable forces the channel output flip-flop to the EDPOL bit value. This functionality targets applications that use active high signals and a high to low transition at A1 match. For such cases EDPOL must be 0.

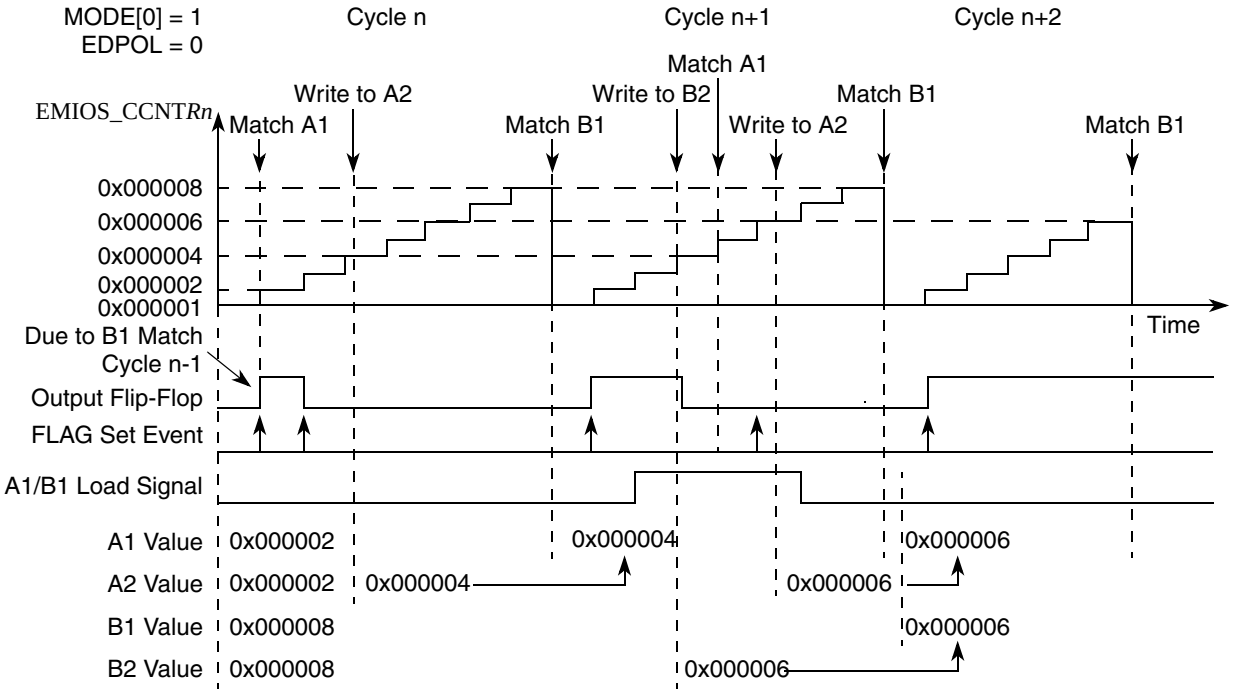


Figure 16-44. eMIOS OPWFMB Mode Example — Active Output Disable

The output disable has a synchronous operation, meaning that the assertion of the output disable input signal causes the channel output flip-flop to transition to EDPOL at the next system clock cycle. If the output disable input is negated, the output flip-flop transitions at the following A1 or B1 match.

In Figure 16-44 it is assumed that the output disable input is enabled and selected for the channel (see Section 16.3.1.7, “eMIOS Channel Control Register EMIOS_CCRn,” for a detailed description of the ODIS and ODISSL bits and selection of the output disable inputs).

The FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on comparators A or B respectively. Similar to a B1 match, FORCMB clears the internal counter. The FLAG bit is not set when the FORCMA or FORCMB bits are set.

Figure 16-45 illustrates the generation of 100% and 0% duty cycle signals. It is assumed that EDPOL = 0 and the prescaler ratio is 1. Initially A1 = 0x000008 and B1 = 0x000008. In this case, a B1 match has precedence over an A1 match, thus the output flip-flop is set to the complement of EDPOL. This cycle corresponds to a 100% duty cycle signal. The same output signal can be generated for any A1 value greater than or equal to B1.

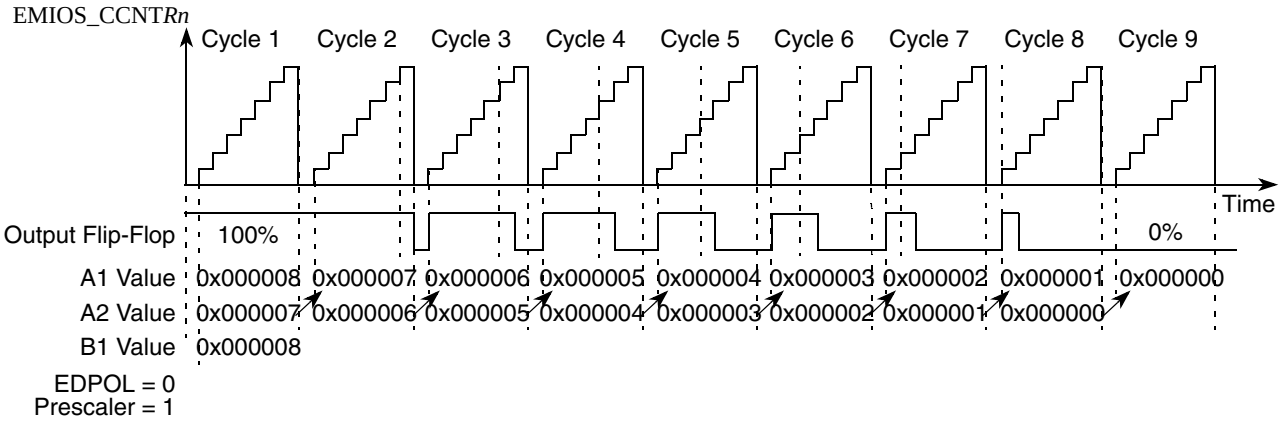


Figure 16-45. eMIOS OPWFMB Mode Example — 100% to 0% Duty Cycle

A 0% duty cycle signal is generated if A1 = 0 as shown in Figure 16-45 cycle 9. In this case the B1 = 0x000008 match from cycle 8 occurs at the same time as the A1 = 0x000000 match from cycle 9. See Figure 16-42 for a description of A1 and B1 match generation for a case where A1 match has precedence over B1 match and the output signal transitions to EDPOL.

16.4.4.4.17 Center Aligned Output Pulse Width Modulation, Buffered Mode (OPWMCB)

MODE[0:6]	Unified Channel Mode of Operation
0b1011100	Center aligned output pulse width modulation, buffered (FLAG set on trailing edge, trailing edge dead-time)
0b1011101	Center aligned output pulse width modulation, buffered (FLAG set on trailing edge, leading edge dead-time)
0b1011110	Center aligned output pulse width modulation, buffered (FLAG set on both edges, trailing edge dead-time)
0b1011111	Center aligned output pulse width modulation, buffered (FLAG set on both edges, leading edge dead-time)

This mode generates a center aligned PWM with dead time insertion on the leading or trailing edge. A1 and B1 registers are double buffered to allow smooth output signal generation when changing A2 or B2 values during operation.

The selected counter bus for a channel configured to OPWMCB mode must be another channel running in MCB up/down counter mode (see Section 16.4.4.4.15, “Modulus Counter, Buffered Mode (MCB)”). Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base. Register B1 contains the dead time value and is compared against the internal counter. For a leading edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. The MODE[6] bit selects between trailing and leading dead time insertion, respectively.

NOTE

It is recommended that the internal prescaler of the OPWMCB channel be set to the same value as the MCB channel prescaler, and the prescalers must also be synchronized. This allows the A1 and B1 registers to represent the same time scale for duty cycle and dead time insertion.

Figure 16-46 illustrates loading of the A1 and B1 registers, which occurs when the selected counter bus reaches the value one. This counter value defines the cycle boundary. Values written to A2 or B2 within cycle (*n*) are loaded into A1 or B1 registers and are used to generate matches in cycle (n+1).

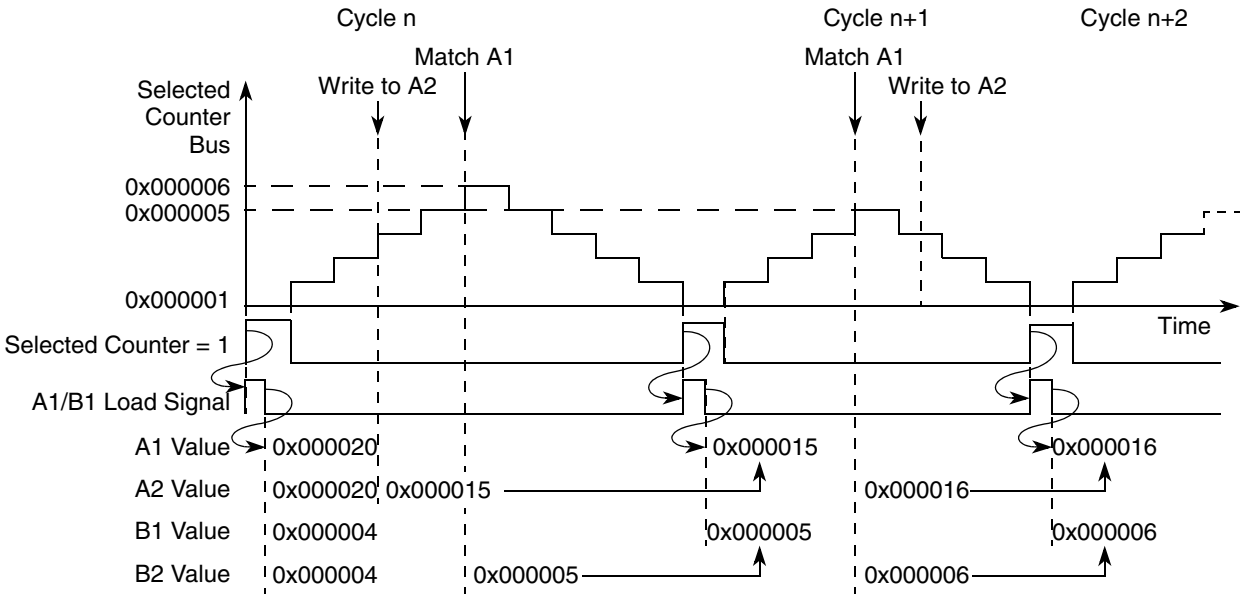


Figure 16-46. eMIOS OPWMCB Mode Example — A1/B1 Register Loading

The EMIOS_OUDR[n] bit can be used to disable the A1 and B1 updates, thus allowing the loading of these registers to be synchronized with the load of A1 or B1 registers in others channels. By using the update disable bit, the A1 and B1 registers can be updated in the same counter cycle.

In this mode A1 matches set the internal counter to one. When operating with leading edge dead time insertion, the first A1 match resets the internal counter to 0x000001. When a match occurs between register B1 and the internal time base, the output flip-flop is set to the value of the EDPOL bit. In the following match between A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously. Figure 16-47 shows two cycles of a center aligned PWM signal. Both A1 and B1 register values are changing within the same cycle, which allows the duty cycle and dead time values to be changed at simultaneously.

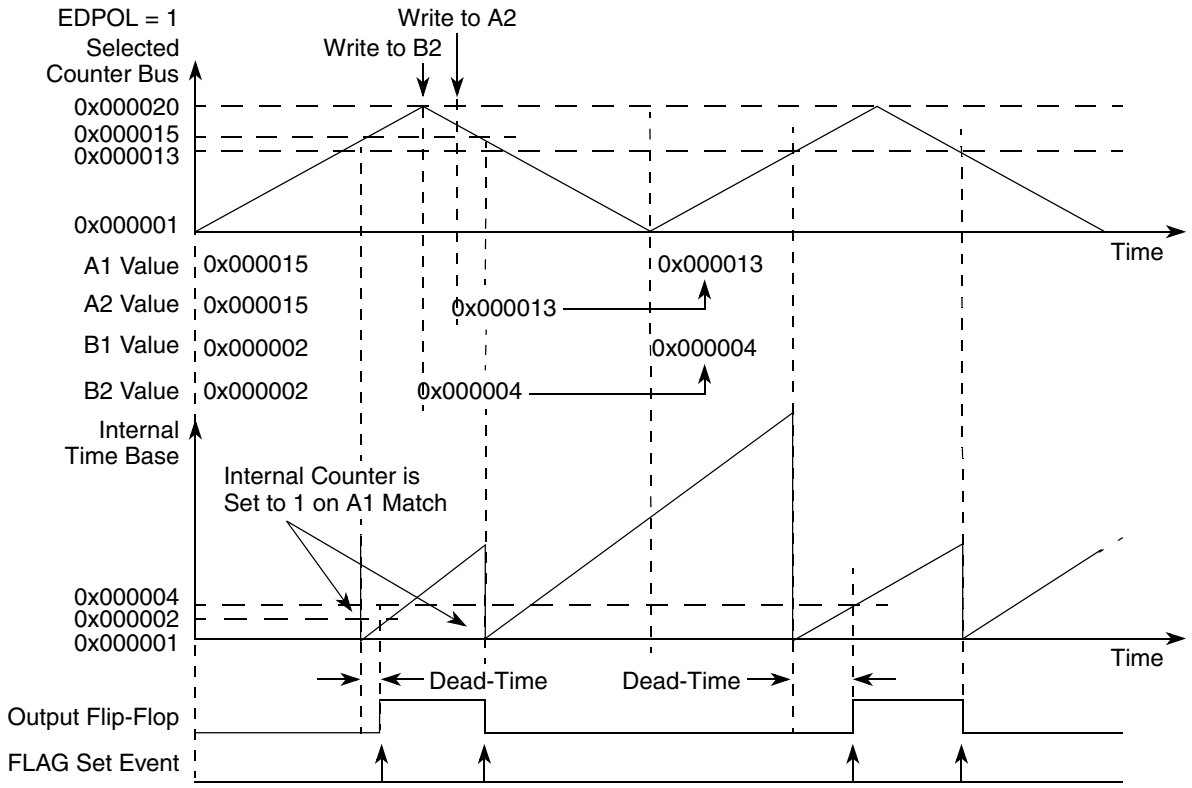


Figure 16-47. eMIOS PWMCB Mode Example — Lead Dead Time Insertion

As shown in Figure 16-48, when operating with trailing edge dead time insertion the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit and resets the internal counter to 0x000001. In the second match between register A1 and the selected time base, the internal counter is reset to 0x000001 and B1 matches are enabled. When the match between register B1 and the selected time base occurs the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

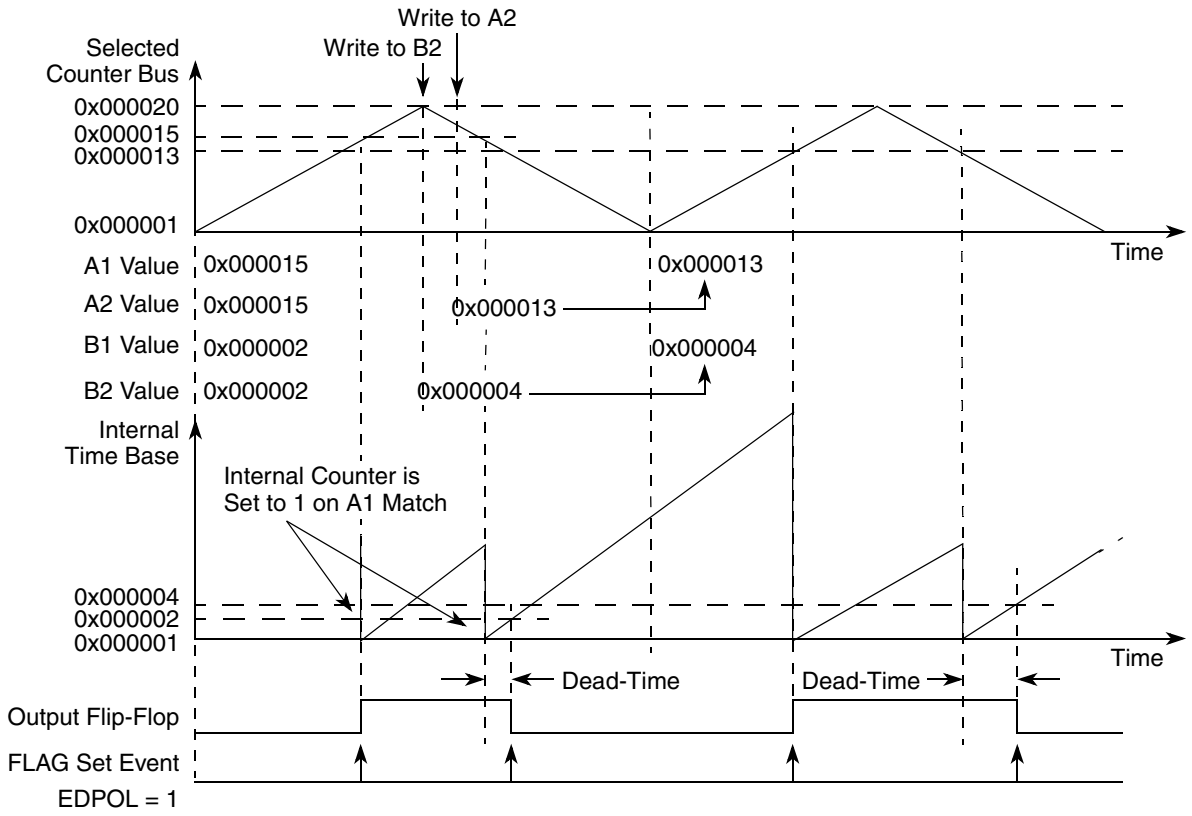


Figure 16-48. eMIOS PWMCB Mode Example — Trailing Dead Time Insertion

FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or on both edges when MODE[5] is set. If subsequent matches occur on A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

NOTE

In OPWMCB mode, FORCMA and FORCMB do not have the same behavior as a regular match:

FORCMA has different behaviors depending on the selected dead time insertion mode. In leading dead time insertion mode, writing one to FORCMA sets the output flip-flop to the compliment of EDPOL. In trailing dead time insertion mode, the output flip-flop is forced to the value of EDPOL.

If FORCMB is set, the output flip-flop value depends on the selected dead time insertion mode. In leading dead time insertion mode, FORCMB sets the output flip-flop to the value of EDPOL. In trailing dead time insertion mode, the output flip-flop is forced to the compliment of EDPOL.

NOTE

Setting the FORCMA bit does not reset the internal time base to 0x000001 as a regular A1 match does. FORCMA and FORCMB have the same behavior even in freeze or normal mode regarding the output flip-flop transition.

The FLAG bit is not set in the case of the FORCMA, FORCMB or both bits being set at the same time.

When FORCMA and FORCMB are both set, the output flip-flop is set to the compliment of the EDPOL bit. This is equivalent to FORCMA having precedence over FORCMB when lead dead time insertion is selected and FORCMB having precedence over FORCMA when trailing dead time insertion is selected.

Duty cycles from 0% to 100% can be generated by setting appropriate A1 and B1 values relative to the period of the external time base. Setting A1 = 1 or A1 = 0 generates a 100% duty cycle waveform. If A1 > period ÷ 2, where period refers to the selected counter bus period, then a 0% duty cycle is produced. Assuming EDPOL is one and OPWMCB mode with trailing dead time insertion mode is selected, 100% duty cycle signals can be generated if B1 occurs at or after the cycle boundary (external counter = 1).

NOTE

A special case occurs when A1 is set to the external counter bus period ÷ 2, which is the maximum value of the external counter. In this case the output flip-flop is constantly set to the EDPOL bit value.

Internal channel logic prevents matches from one cycle to propagate to the next cycle. In trailing dead time insertion mode, a B1 match from cycle (*n*) could eventually cross the cycle boundary and occur in cycle (n+1). In this case the B1 match is masked out and does not cause the output flip-flop to transition. Therefore matches in cycle (n+1) are not affected by the late B1 matches from cycle (*n*).

Figure 16-49 shows a 100% duty cycle output signal generated by setting A1 = 4 and B1 = 3. In this case the trailing edge is positioned at the boundary of cycle (n+1), which is actually considered to belong to cycle (n+2) and therefore does not cause the output flip-flip to transition.

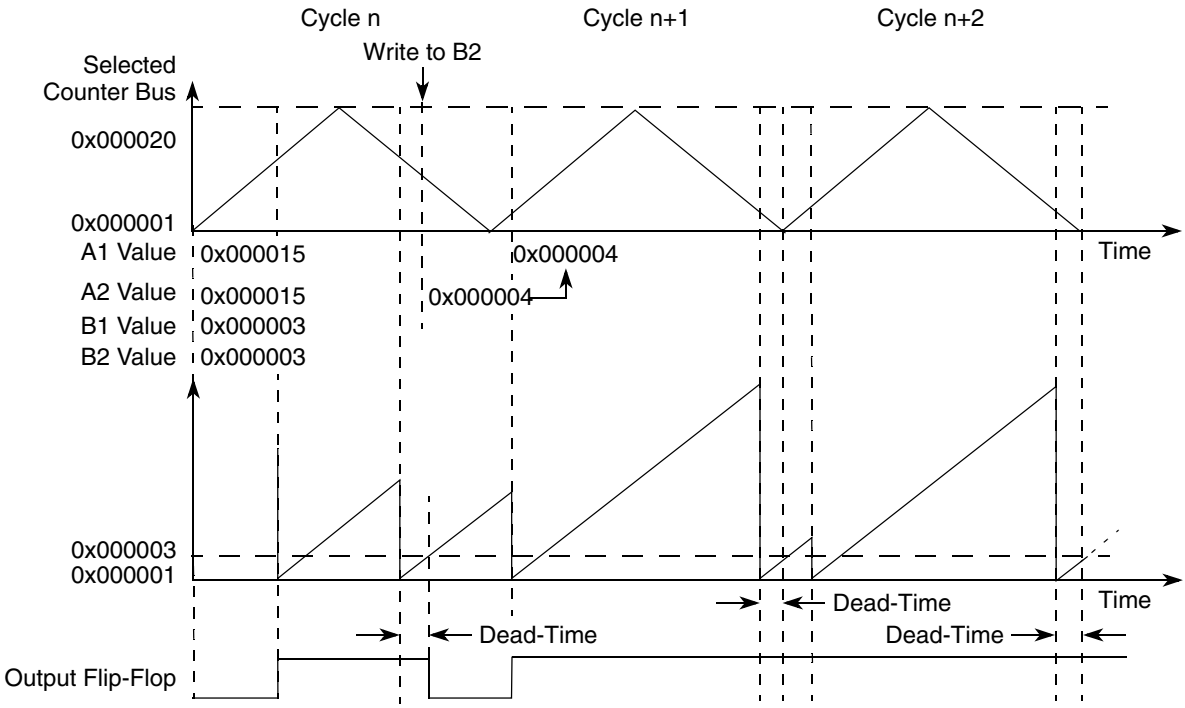


Figure 16-49. eMIOS PWMCB Mode Example — 100% Duty Cycle (A1 = 4, B1 = 3)

The output disable input, if enabled, causes the output flip-flop to transition to the compliment of EDPOL. This allows to the channel output flip-flop to be forced to a safety state. The internal channel matches continue to occur in this case, thus generating flags. When the output disable is negated, the channel output flip-flop is again controlled by A1 and B1 matches. This process is synchronous, meaning that the output channel pin transitions only occur on system clock edges.

As in OPWMB and OPWFMB modes, the match signal used to set or clear the channel output flip-flop is generated on the negation of the channel comparator output signal which compares the selected time base with A1 or B1. See Figure 16-41, which illustrates the delay from matches to output flip-flop transition in OPWFMB mode.

16.4.4.4.18 Output Pulse Width Modulation, Buffered Mode (OPWMB)

MODE[0:6]	Unified Channel Mode of Operation
0b1100000	Output pulse width modulation, buffered (FLAG set on second match)
0b1100001	Reserved
0b1100010	Output pulse width modulation, buffered (FLAG set on both matches)

OPWMB mode is used to generate pulses with programmable leading and trailing edge placement. An external counter is selected from one of the counter buses. The A1 register value defines the first edge and B1 defines the second edge. The output signal polarity is defined by the EDPOL bit. If EDPOL is zero, a negative edge occurs when A1 matches the selected counter bus and a positive edge occurs when B1 matches the selected counter bus.

The A1 and B1 registers are double buffered and updated from A2 and B2, respectively, at the cycle boundary. The load operation is similar to the OPWFMB mode. See Figure 16-43 for more information on A1 and B1 register updates.

Flags are generated at B1 matches when MODE[5] is cleared, or on both A1 and B1 matches when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated regardless of the state of the FLAG bit.

The FORCMA and FORCMB bits allow software to force the output flip-flop to the level corresponding to a match on A1 or B1 respectively. FLAG is not set by the FORCMA and FORCMB operations.

The following rules apply to the OPWMB mode:

- B1 matches have precedence over A1 matches if they occur at the same time within the same counter cycle.
- A1 = 0 match from cycle (*n*) has precedence over a B1 match from cycle (n-1).
- A1 matches are masked if they occur after a B1 match within the same cycle.
- Values written to A2 or B2 on cycle (*n*) are loaded to A1 or B1 at the following cycle boundary (assuming EMIOS_OUDR[*n*] is not asserted). Thus the new values is used for A1 and B1 matches in cycle (*n*+1).

Figure 16-50 illustrates operation in OPWMB mode with A1/B1 matches and the transition of the channel output flip-flop. In this example EDPOL is zero.

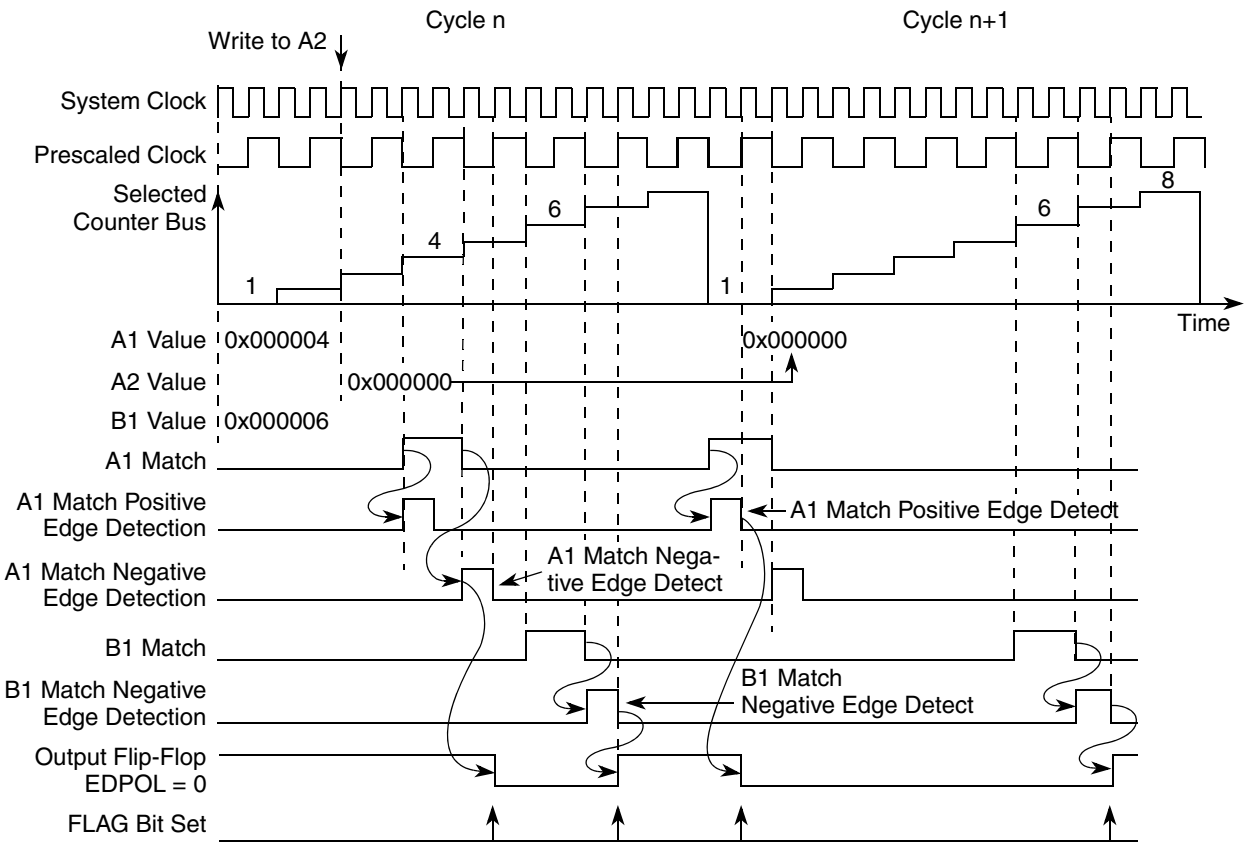


Figure 16-50. eMIOS OPWMB Mode Example—Matches and Flags

The output flip-flop transitions are based on the negative edges of the A1 and B1 match signals. Figure 16-50 shows the value of A1 being set to zero in cycle (n+1). In this case the match positive edge is used instead of the negative edge to transition the output flip-flop.

Figure 16-51 illustrates the channel operation for 0% duty cycle. The A1 match signal positive edge occurs at the same time as the B1 = 8 signal negative edge. In this case the A1 match has precedence over the B1 match, causing the output flip-flop to remain at the EDPOL value, thus generating a 0% duty cycle.

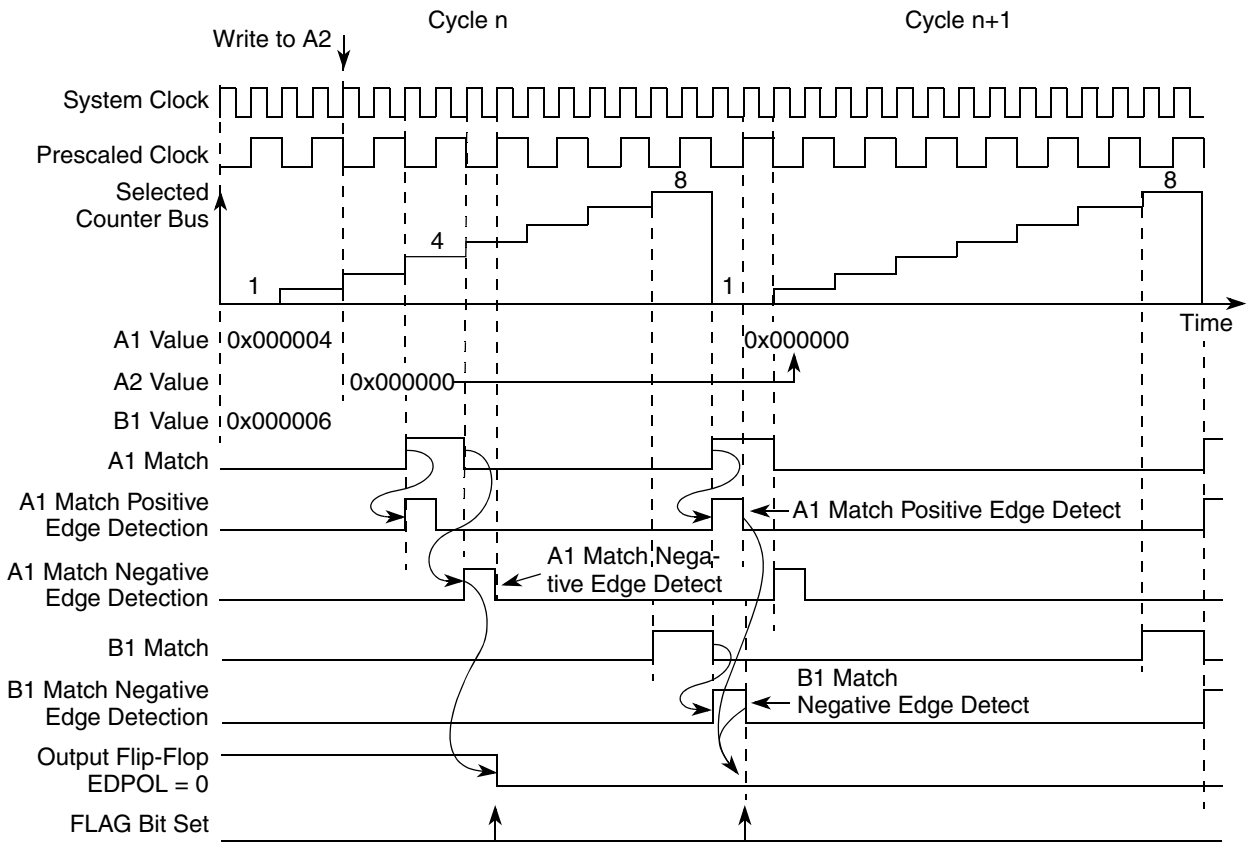


Figure 16-51. eMIOS OPWMB Mode Example—0% Duty Cycle

Figure 16-52 shows the operation of the OPWMB mode with the output disable signal asserted. The output disable forces a transition in the output flip-flop to the EDPOL bit value. After the output disable is negated, the output flip-flop is allowed to transition at the next A1 or B1 match. The output disable does not modify the flag bit behavior. There is one system clock delay between the assertion of the output disable signal and the transition of the output flip-flop.

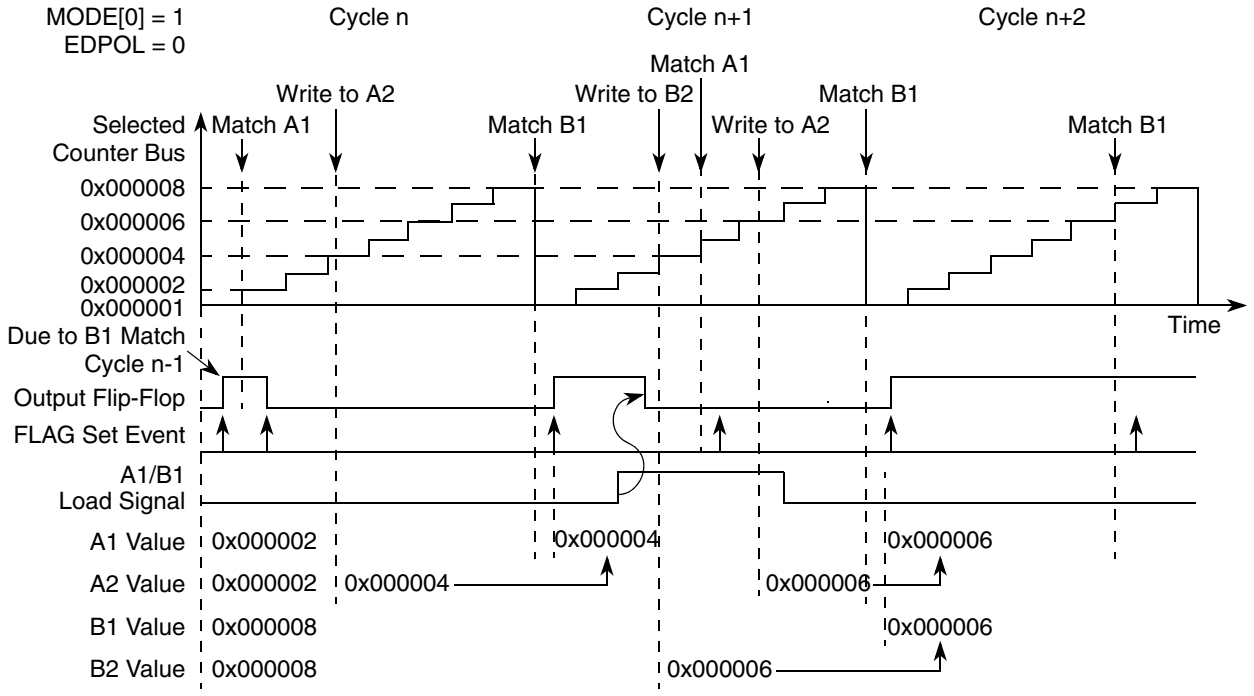


Figure 16-52. eMIOS OPWMB Mode Example—Active Output Disable

Figure 16-53 shows a waveform changing from 100% to 0% duty cycle. In this case EDPOL is zero and B1 is set to the same value as the period of the selected external time base.

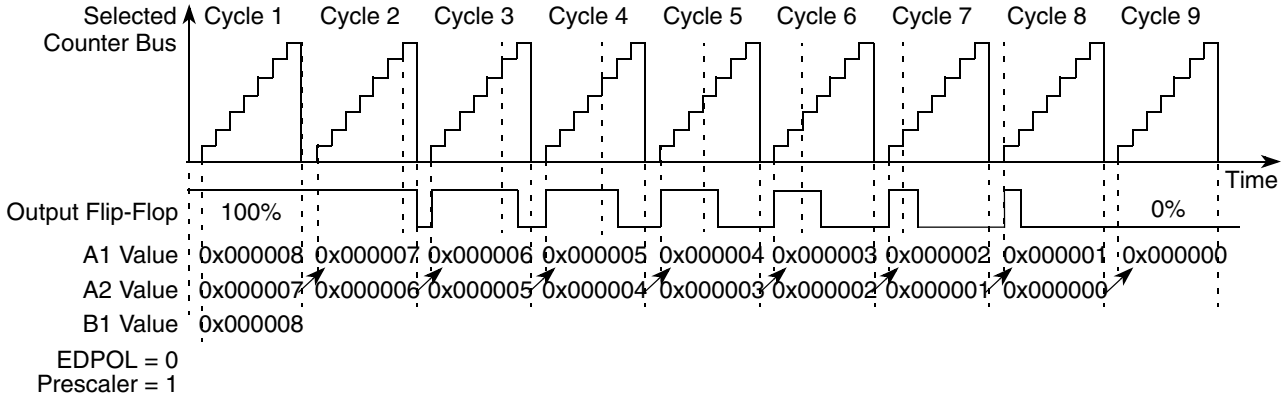


Figure 16-53. eMIOS OPWMB Mode Example—100% to 0% Duty Cycle

In Figure 16-53 if B1 is set to a value lower than 0x000008 it is not possible to achieve 0% duty cycle by only changing A1 register value. Since B1 matches have precedence over A1 matches, the output flip-flop transitions to the compliment of EDPOL at B1 matches. In this example, if B1 = 0x000009, a B1 match does not occur, and thus a 0% duty cycle signal is generated.

16.5 Initialization and Application Information

Upon reset all of the unified channels of the eMIOS default to general purpose inputs (GPIO input mode).

16.5.1 Considerations on Changing a UC Mode

Before changing an operating mode, the UC must be programmed to GPIO mode, and EMIOS_CADR*n* and EMIOS_CBDR*n* must be updated with the correct values for the next operating mode. Then the EMIOS_CCR*n* can be written with the new operating mode. If a UC is changed from one mode to another without performing this procedure, the first operating cycle of the selected time base is unpredictable.

NOTE

When interrupts are enabled and an interrupt is generated, clear the FLAG bits before exiting the interrupt service routine.

16.5.2 Generating Correlated Output Signals

Correlated output signals can be generated by all output operating modes. Bits ODIS*n* can be used to control the update of these output signals.

To guarantee that the internal counters of correlated channels are incremented in the same clock cycle, the internal prescalers must be set up before enabling the global prescaler. If the internal prescalers are set after enabling the global prescaler, the internal counters can increment in the same ratio, but at a different clock cycle.

When an output disable condition occurs, the software interrupt routine must service the output channels before servicing the channels running SAIC. This procedure avoid glitches in the output pins.

16.5.3 Time Base Generation

For all channel operation modes that generate a time base (MC, OPWFM, OPWM, MCB, OPWFMB and OPWMB), the clock prescaler can use several ratios calculated as:

$$\text{Ratio} = (\text{GPRE} + 1) \times (\text{UCPRE} + 1)$$

The prescaled clocks in Figure 16-55, Figure 16-56, and Figure 16-57 illustrate this ratio. For example, if the ratio is 1, the prescaled clock is high and continuously enables the internal counter (EMIOS_CCNTR*n*) (Figure 16-55); if the ratio is 3, then it pulses every 3 clock cycles (Figure 16-56) and the internal counter increments every 3 clock cycles; if the ratio is 9, it pulses every 9 clock cycles, etc. This high pulse enables the EMIOS_CCNTR*n* to increment as long as no other conditions disable this counter. The match signal is generated by pulsing every time the internal counter matches the programmed match value. For the same programmed match value, the period is shorter when using a prescaler ratio greater than one.

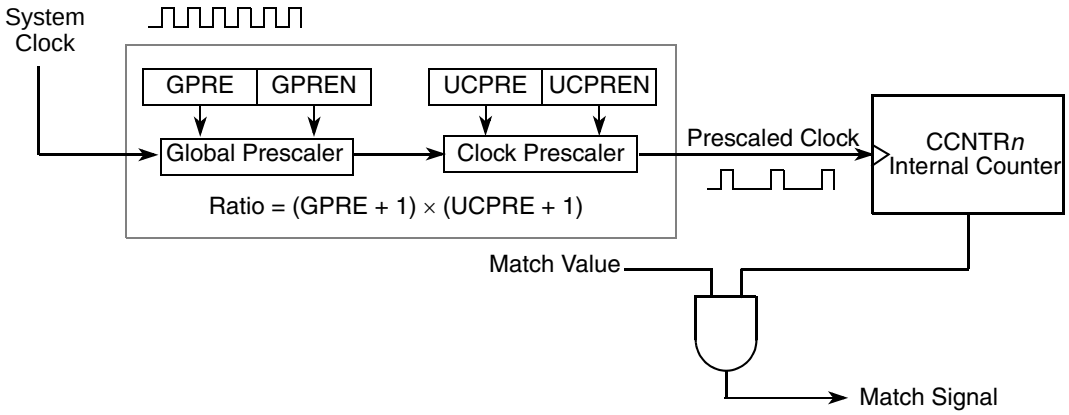
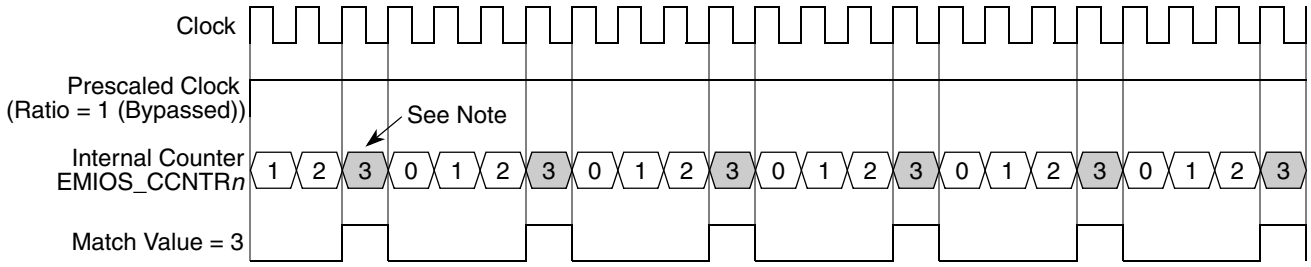
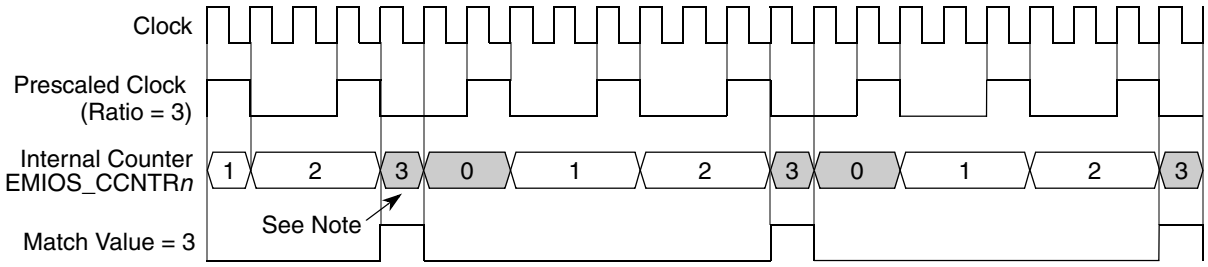


Figure 16-54. eMIOS Time Base Generation Block Diagram



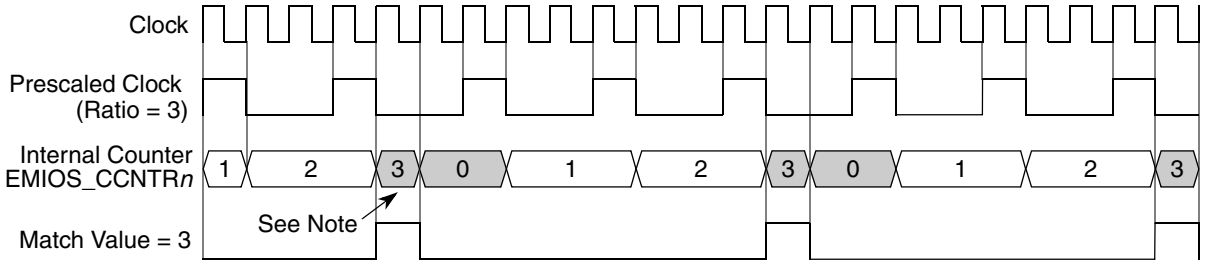
NOTE: The period of the time base includes the match value. When a match occurs, the first clock cycle is used to clear the internal counter, starting another period

Figure 16-55. eMIOS Time Base Example—Fastest Prescaler Ratio



NOTE: The period of the time base does not include the match value. When a match occurs, the first clock cycle is used to clear the internal counter, starting another period

Figure 16-56. eMIOS Time Base Example—Prescale Ratio = 3, Match Value = 3



NOTE: The period of the time base does not include the match value. When a match occurs, the first clock cycle is used to clear the internal counter, starting another period

Figure 16-57. eMIOS Time Base Example—Prescale Ratio = 2, Match Value = 5

Chapter 17
Enhanced Time Processing Unit (eTPU)

17.1 Introduction

The enhanced time processing unit (eTPU) is the timing unit featured on this microcontroller that operates in parallel with the device core (CPU). The eTPU does the following:

- Executes programs independently from the host core
- Detects and precisely records timing of input events
- Generates complex output waveforms
- Is controlled by the core without a requirement for real-time host processing

The host core setup and service times for each input and output event are greatly minimized. This device contains one eTPU.

The eTPU improves the performance of the device by providing high resolution timing:

- eTPU dedicated channels that include two match and two capture registers, as opposed to the previous generation TPUs which only had one of each register
- eTPU engines that are optimized with specific instructions to service channel hardware
- The fast instruction execution rate of the eTPU engine that reduces service time

Because responding to hardware service requests is primarily done by the eTPU engine, the host is free to handle higher level operations.

NOTE

All references to an eTPU Reference Manual are referring to the *Enhanced Time Processing Unit (eTPU) Reference Manual*.

17.1.1 eTPU Implementation

For more detailed information regarding the eTPU module and compiler, see the *Enhanced Time Processing (eTPU) Reference Manual.* The device contains a specific implementation of the eTPU's full functionality. This chapter focuses only on an eTPU overview and those details that are different than the full instantiation of the module. These differences include the following:

- 2.5 KBs of shared data memory (SDM). This memory is alternately referred to as eTPU shared parameter (data) RAM (SPRAM).
- 12 KBs of shared code memory (SCM).

The eTPU debug interface is built into the device's debug module. See Section 10.2.1 of the eTPU reference manual for details on eTPU debug.

- Data transfer requests are implemented as a single DMA request to the DMA controller. All 32 channels' data transfer request signals are logically ORed to produce the single DMA request.
- I/O channel pairs can be shared on a common pin. The output buffer enable (OBE) is not used in this device. The outputs are enabled in the SIU; see Chapter 6, "System Integration Unit (SIU)."

Because of the above differences between this device's implementation of the eTPU and the full eTPU, full register bit descriptions are included within this chapter as well as in the *Enhanced Time Processing (eTPU) Reference Manual.*

17.1.2 Block Diagram

Figure 17-1 shows a top-level eTPU block diagram. This device has a single eTPU engine configuration.

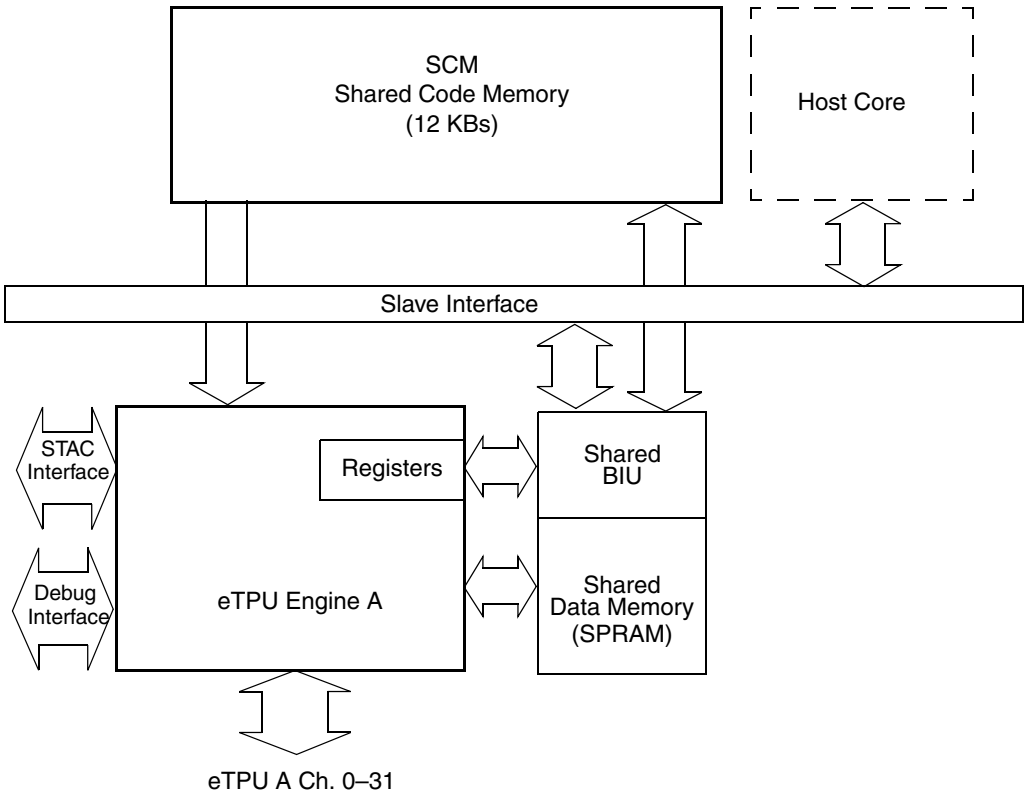


Figure 17-1. eTPU Block Diagram

Figure 17-2 shows the block diagram for the eTPU engine.

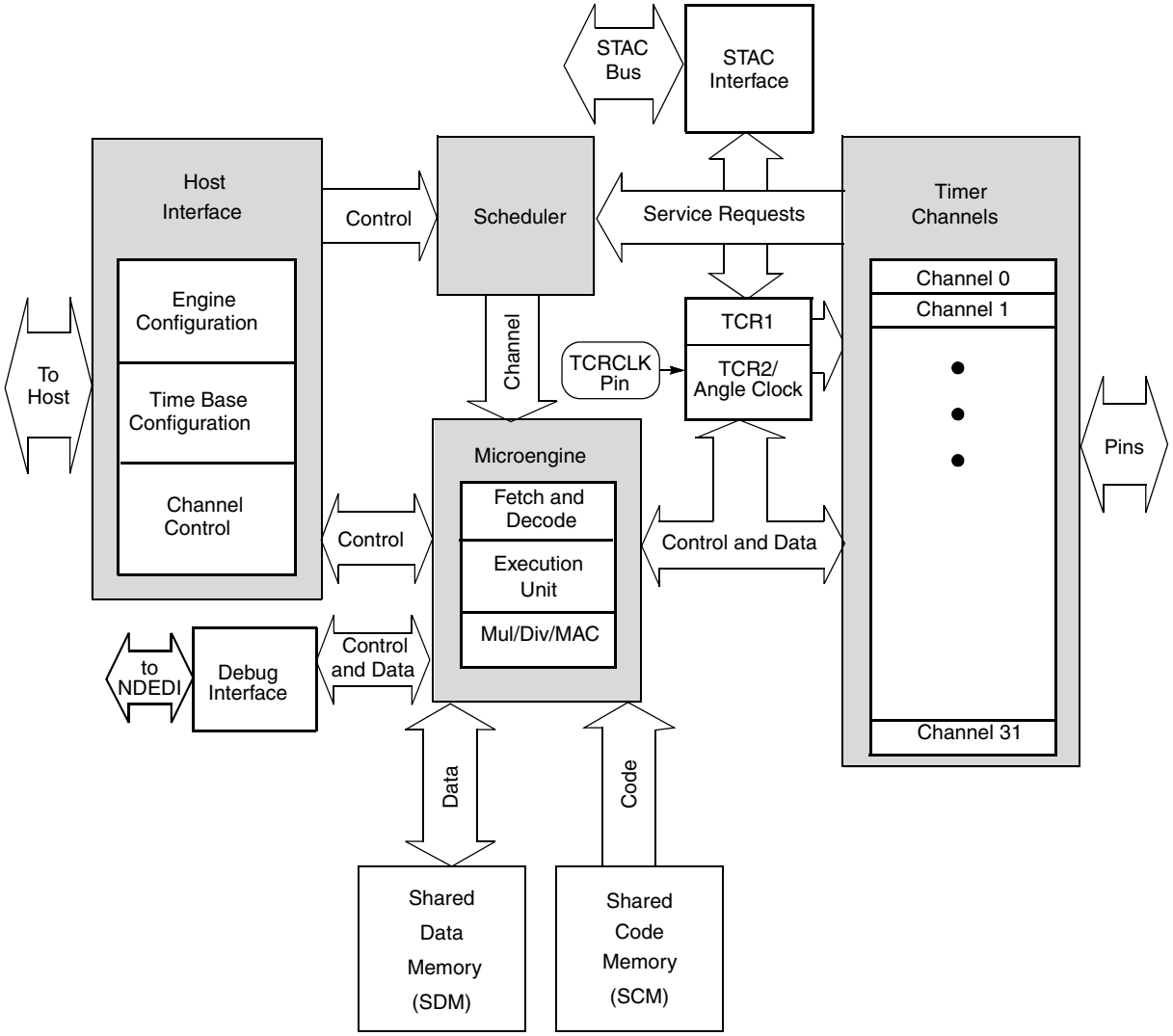


Figure 17-2. eTPU Engine Block Diagram

17.1.3 eTPU Operation Overview

The eTPU is a real-time microprocessor subsystem. Therefore it runs microengine code from instruction memory (SCM) to handle specific events and accesses data memory (SDM) for parameters, work data, and application state information. Events can originate from I/O channels (due to pin transitions and/or time base matches), device core requests, or inter-channel requests. Events that call for local eTPU processing activate the microengine by issuing a service request. The service request microcode can send an interrupt to the device core, but the core cannot be directly interrupted by I/O channel events.

Each channel is associated with a function that defines its behavior. A function is a software entity consisting of a set of microengine routines, called threads, that respond to eTPU service requests. Function routines, which reside in the SCM, are also responsible for channel configuration. A function can be assigned to several channels, but a channel can only be associated with one function at a given moment.

If the device core reconfigures the channel function, the eTPU can change the function assigned to that channel. The association between functions and channels is defined by the device core.

The eTPU hardware supplies resource sharing features that support concurrency:

- A hardware scheduler dispatches the service request microengine routines based on a set of priorities defined by the device's core. Each channel has its own unique priority assignment that primarily depends on CPU assignment. The channel's number is an inherent property also used to determine priority.
- A service request routine cannot be interrupted by another service request until it ends, that is, until an end instruction is executed. This sequence of uninterrupted instruction execution is called a thread. The core can terminate the thread by writing 1 to the FEND bit in the ETPU_ECR register.
- Channel-specific contexts (registers and flags) are automatically switched between the end of a thread and the beginning of the next one.
- SDM arbitration, a dual-parameter coherency controller, and semaphores can be used to ensure coherent access to eTPU data shared by both eTPU engines and the device core.

17.1.3.1 eTPU Engine

The eTPU engine processes input pin transitions and generates output pin waveforms. These events are triggered by eTPU timers (time bases) that are driven by a system clock to give absolute time control or by an asynchronous counter such as an angle clock that can track the angle of a rotating shaft.

Each eTPU engine consists of the following blocks: 32 independent timer channels, a task scheduler, a host interface, and a microprocessor (hereinafter called a microengine) that has dedicated hardware for input signal processing and output signal generation over the 32 I/O channels. Each channel can also choose between two 24-bit counter registers for a time base.

The microengines fetch microinstructions from shared code memory (SCM). eTPU application parameters and global and local variables, referred to as work data, are held in 32-bit shared data memory (SDM), which is also used for passing information between the device's core and both (or one) microengines. The bus interface unit (BIU) allows the device's core to access eTPU registers, SDM, and SCM.

17.1.3.2 Time Bases

The eTPU engine has two 24-bit count registers TCR1 and TCR2 that provide reference time bases for all match and input capture events. Prescalers for both time bases are controlled by the device core through bit fields in the eTPU engine configuration registers.

The values for each of TCR1 and TCR2 counter registers can be independently derived from the system clock or from an external input via the TCRCLK pin. In addition, the TCR2 time base can be derived from special angle-clock hardware that enables implementing angle-based functions. This feature is added to support advanced angle-based engine control applications.

The TCRs can also drive an eMIOS time base through the shared time and counter (STAC) bus, or they can be written by eTPU function software.

17.1.3.3 eTPU Timer Channels

Each eTPU engine has 32 identical, independent channels. Each channel corresponds to an input/output signal pair. Every channel has access to two 24-bit counter registers, TCR1 and TCR2.

Each channel consists of event logic which supports a total of four events, two capture and two match events. The event logic contains two 24-bit capture registers and two 24-bit match registers. The match registers are compared to a selected TCR by greater-than-or-equal-to and equal-only comparators. The match and compare register pairs enable many combinations of single and double-action functions.

The channel configuration can be changed by, with restrictions, the microengine. Each channel can perform double capture, double match or a variety of other capture-match combinations. Service requests can be generated on one or both of the match events and/or on one of the capture events.

Digital filters that have different filtering modes are provided for the input signals.

Every channel can use any time base or angle counter for either match or capture operation. For example, a match on TCR1 can capture the value of TCR2. The channels can request service from the microengine due to recognized pin transitions (input events) or time base matches.

Every eTPU channel can be configured with the following combinations:

- Single input capture, no match (TPU3 functionality)
- Single input capture with single match time-out (TPU3 functionality)
- Single input capture with double match time-out with several double match submodes
- Double input capture with single or double match time-out with several double match submodes
- Single output match (TPU3 functionality)
- Double output match with several double match submodes
- Input-dependent output generation

The double match functionality has various combinations for generation of service request and determining pin actions.

17.1.3.3.1 Host Interface

The engine’s host interface allows the device core to control the operation of the eTPU. For the eTPU to start operation, the device core must initialize the eTPU by writing to the appropriate host interface registers to assign a function and priority to each channel. In addition, the device core writes to the host service request and channel configuration registers to further define operation for each initialized channel.

NOTE

The host transfers the code image for the eTPU microcode to the SCM, then the host enables eTPU access to the SCM (which also disables host access).

17.1.3.3.2 Shared Data Memory (SDM)

The SDM works as data RAM that can be accessed by the device core and the eTPU engine. This memory is used for either:

- Information transfer between the device core and the eTPU
- Data storage for the eTPU microcode program

The SDM width is 32 bits, and is accessible by the host in any of the three formats: byte, 16-bit, or 32-bit. The eTPU can access the full 32 bits of the SDM, lower 24 bits or upper byte (8-bit).

The host can also access the SDM space mirrored in an alternate area with parameter sign extension (PSE). PSE allows for 24-bit data to be accessed as 32 bit sign-extended data without using the device's bandwidth to extend the data.

Parameter signal extension accesses differ from the usual host accesses to the original SDM area as follows:

- Writes are effective only to the lower 3 bytes of a word: the word's most significant byte (byte address) is kept unaltered in SDM.

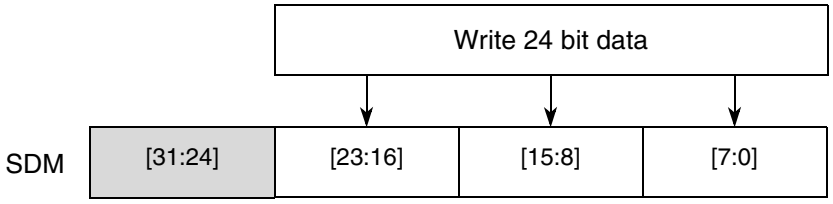


Figure 17-3. SDM PSE Area Write

NOTE

For the most significant byte, the word format is big endian, as in the default PowerPC word format.

- Reads return the lower 3 bytes of a word sign-extended to 32 bits, that is: the most significant bit of the word's second most significant byte (byte addresses) is copied in all 8 bits of the most significant read byte.

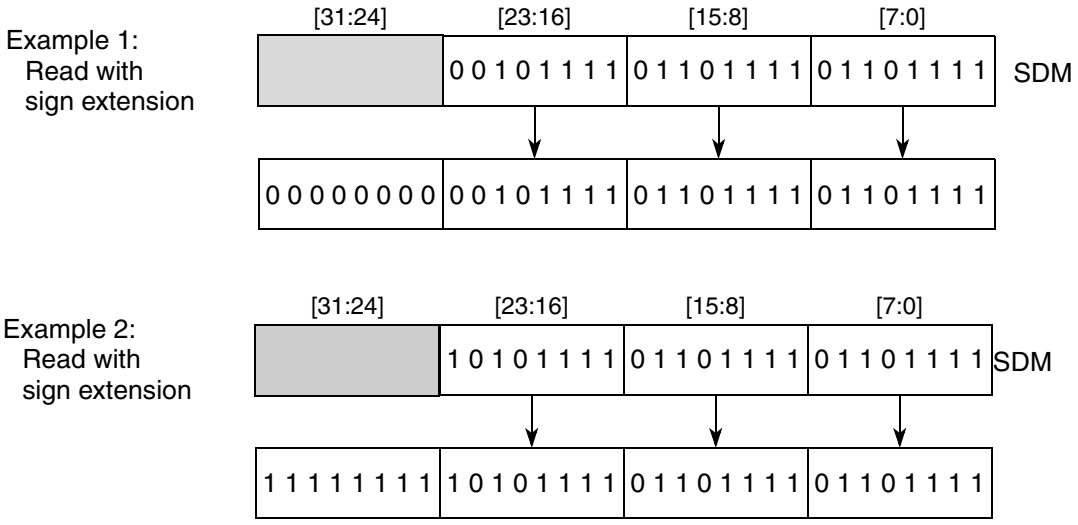


Figure 17-4. PSE Accesses

Each eTPU channel can be associated with a variable number of parameters located in the SDM, according to its selected function. In addition, the SDM can be fully shared between two eTPU engines, enabling communication between them. Each function can require a different number of parameters. During eTPU initialization the host has to program channel base addresses, allocating proper parameters for each channel according to its selected function.

In the host address space each parameter occupies four bytes (32 bits). eTPU usage of the upper byte is achieved by having a 32-bit Preload (P) register that can access the upper byte, the lower 24 bits, or all the 32 bits. The microcode can switch between access sizes at any time.

Each function can require a different number of parameters. During the eTPU initialization the host has to program channel base addresses, allocating proper parameters for each channel according to its selected function.

17.1.3.3.3 Task Scheduler

As mentioned in Section 17.1.3, “eTPU Operation Overview,” every channel function is composed of one or more threads, and threads cannot be interrupted by host or channel events, such as channel servicing. The function of the task scheduler, therefore, is to recognize and prioritize the channels needing service and grant execution time to each channel. The time given to an individual thread for execution or service is called a time slot. The duration of a time slot is determined by the number of instructions executed in the thread plus SDM wait-states received, and varies in length. Although several channels can request service at the same time, the function threads must be executed serially.

At any time, an arbitrary number of channels can require service. The channel logic, eTPU microcode, or the host application notifies the scheduler by issuing a service request.

Out of reset, all channels are disabled. The device core makes a channel active by assigning it one of three priorities: high, middle, or low. The scheduler determines the order in which channels are serviced based on channel number and assigned priority. The priority mechanism, implemented in hardware, ensures that all requesting channels are serviced.

17.1.3.3.4 Microengine

The eTPU microengine is a simple RISC implementation that performs each instruction in a microcycle of two system clocks, while pre-fetching the next instruction through an instruction pipeline. Instruction execution time is constant for the arithmetic logic unit (ALU) unless it gets wait states from SDM arbitration.

Microcode is stored in shared code memory (SCM) that is 32 bits wide. The microengine instruction set provides basic arithmetic and logic operations, flow control (jumps and subroutine calls), SDM access, and channel configuration and control. The instruction formats are defined in such a way that allow particular combinations of two or three of these operations with unconflicting resources to be executed in parallel in the same microcycle, thus improving performance.

The microengine also has an independent multiply/divide/MAC unit that performs these complex operations in parallel with other microengine instructions.

Channel functionality is integrated to the instruction set through channel control operations and conditional branch operations, which support jumps/calls on channel-specific conditions. This allows quick and terse channel configuration and control code, contributing to reduced service time.

17.1.3.4 Debug Interface

Nexus level 3 debug support is available through the eTPU Nexus development interface (NDEDI). See Chapter 24, “Nexus Development Interface.”

17.1.4 Features

The eTPU includes these distinctive features:

- Up to 32 channels for each eTPU engine: each channel is associated with an I/O signal pair
  - Enhanced input digital filters on the input pins for improved noise immunity. The eTPU digital filter can use two samples, three samples, or work in continuous mode.
  - Orthogonal channels, except for channel 0: each channel can perform any time function. Each time function can be assigned to more than one channel at a given time, so each signal can have any functionality. Channel 0 has the same capabilities of the others, but can also work with special angle counter logic (see below).
  - A link service request allows activation of a channel thread by request of another channel, even between eTPU engines.
  - A host service request allows activation of a channel thread by the device core request.
  - Each channel has an event mechanism that supports single and double action functionality in various combinations. It includes two 24-bit capture registers, two 24-bit match registers, 24-bit greater-equal or equal-only comparator.

- Two independent 24-bit time bases for channel synchronization
  - The first time base can be clocked by the system clock with programmable prescaler division from 2 to 512 (in steps of 2).
  - The first time base can also be clocked by an external signal with programmable prescaler divisions of 1 to 256.
  - The second time base can be clocked by an external signal with programmable prescaler divisions from 1 to 64 or by the system clock divided by 8.
  - The second time base has a programmable prescaler that applies to all TCR2 clock inputs except the angle counter.
  - The second time base counter can work as an angle counter, enabling angle-based applications to match angle instead of time.
  - The second time base can alternatively be used as a pulse accumulator gated by an external signal.
  - Either time base can be written or read by the eTPU engine at any time.
  - Either time base can be read, but not written, by the host.
  - Both time bases can be exported or imported from engine to engine through the STAC (shared time and counter) bus.

NOTE

An engine cannot export/import to/from itself. An engine cannot import a time base and/or angle count if it is in angle mode.

- Event-triggered RISC processor (microengine)
  - 2-stage pipeline implementation (fetch and execution), with separate instruction memory (SCM) and data memory (SDM).
  - Two-system-clock microcycle fixed-length instruction execution for the ALU.
  - 12 KBs of shared code memory (SCM).
  - 2.5 KBs of shared data memory (SDM).
  - Instruction set with embedded channel support, including specialized channel control subinstructions and conditional branching on channel-specific flags.
  - Channel-oriented addressing: channel-bound address mode with host configured channel base address allows the same function to operate independently on different channels.
  - Channel-bound data address space of up to 128 32-bit parameters (512 bytes).
  - Global parameter address mode allows access to common channel data of up to 256 32-bit parameters (1024 bytes).
  - Support for indirect and stacked data access schemes.
  - Parallel execution of: data access, ALU, channel control and flow control subinstructions in selected combinations.
  - 24-bit registers and ALU, plus one 32-bit register for full-width SDM access.
  - Additional 24-bit multiply/MAC/divide unit which supports all signed/unsigned/ multiply/MAC combinations, and unsigned 24-bit divide. The MAC/divide unit works in parallel with the regular microcode commands.

- Resource sharing features resolve channel contention for common use of channel registers, memory and microengine time
  - Hardware scheduler works as a ‘task management’ unit, dispatching event service routines by predefined, host-configured priority.
  - Hardware breakpoints on data access, qualified by address and/or data values.
  - Hardware breakpoints on instruction address.
  - Automatic channel context switch when a ‘task switch’ occurs; that is, one function thread ends and another begins to service a request from another channel. Channel-specific registers, flags and parameter base address are automatically loaded for the next serviced channel.
  - Individual channel priority setting in three levels: high, middle, and low.
  - Scheduler priority scheme allows calculation of worst case latency for event servicing and ensures servicing of all channels by preventing permanent blockage.
  - SDM shared between host core and both eTPU engines, supporting channel-channel or host-channel communication.
  - Hardware implementation of four semaphores allows for resource arbitration between channels in both eTPU engines.
  - Hardware semaphores are directly supported by the microengine instruction set.
  - Dual-parameter coherency hardware support allows coherent (to host) access to 2 parameters by microengines in back-to-back accesses.
  - Coherent dual-parameter controller allows coherent (to microengines) accesses to two parameters by the host.
- Test and development support features
  - Nexus level 3 debug support through the eTPU Nexus block (NDEDI)
  - Software breakpoints
  - SCM (code memory) continuous signature-check built-in code integrity test multiple input signature calculator (MISC): runs concurrently with eTPU normal operation

17.2 Modes of Operation

The eTPU is capable of working in the following modes.

17.2.1 User Configuration Mode

By having access to the shared code memory (SCM), the core has the ability to program the eTPU cores with time functions.

17.2.2 User Mode

In user mode the core does not access the eTPU shared code memory, and pre-defined eTPU functions are used.

17.2.3 Debug Mode

The core debugs eTPU code, accessing special trace/debug features via Nexus interface:

— Hardware breakpoint/watchpoint setting
— Access to internal registers
— Single-step execution
— Forced instruction execution
— Software breakpoint insertion and removal

17.2.4 Module Disable Mode

eTPU engine clocks are stopped through a register write to ETPU_ECR bit MDIS, saving power. Input sampling stops. eTPU engines can be in disable mode independently. Module disable mode stops only the engine clock, so that the shared BIU and global channel registers can be accessed, and interrupts and DMA requests can be cleared and enabled/disabled. An engine only enters module disable mode when any currently running thread is finished.

17.2.5 eTPU Mode Selection

User and user configuration are the production operating modes, and differ from each other only in access to SCM. Module disable mode is entered by setting ETPU_ECR[MDIS].

17.3 External Signal Description

There are 65 external signals for the eTPU engine:

- 32 channel input signals
- 32 channel output signals
- TCRCLK clock input

Additionally there are four internal output disable signals that implement the output disable feature needed for motor control. See Section 16.2.1.2, “Output Disable Input—eMIOS Output Disable Input Signals,” for more information.

17.4 eTPU Detailed Signal Description

17.4.1 Output and Input Channel Signals

The channel signal connections for the eTPU engine are described in Table 17-1 and Table 19-3, respectively. Each eTPU channel has an input and output associated with it. In Table 17-1 and Table 19-3 this is represented by the Input/output column. The eTPU channels can be connected to external pins or wired internally to other peripheral devices. In the device, some of the eTPU channels are connected to pins. The pin connections are represented by the Pin Number column in Table 17-1 and Table 19-22. To the right of the pin number column is the eTPU channel connections column that shows the channel number that corresponds to each input or output pin. Many of these pins are multipurpose, that is they are multiplexed. Table 17-1 and Table 19-3 shows the other non-eTPU signals listed in the Signals with Which eTPU Signal is Shared column.

To reduce the number of pins required by the device's eTPU while still maintaining the eTPU's functionality, the eTPU is also internally wired to the DSPI (Chapter 19, "Deserial Serial Peripheral Interface (DSPI)"). The DSPI connections are shown in the column labeled DSPI Serial Channel Connections in Table 17-1 and Table 19-22. The eTPU microcode can be programmed to set the output level of an eTPU channel in one of two manners:

- By forcing the logic level to a specified value
- By specifying the logic level output action when a match or transition event occurs

Every eTPU channel input has a digital filter to filter out noise pulses that have a width less than a specified value. This prevents small noise glitches from being recognized by the transition detect logic. Any pulses wider than the specified filter width are passed to the channel transition detect logic.

Table 17-1. eTPU A Channel Connection Table [1]

eTPU Channel Number	I/O	eTPU Channel Connections	DSPI Serial Channel Connections	eTPU A Signal	Signal muxed with the eTPU Signal
0–9	Input	0, 1–4, 5–8, 9	Not connected	eTPUA[0:9]	eTPUA[12:21]_GPIO[114:123][2]
	Output	0–9	DSPI C[4:13]		eMIOS[0:9]_GPIO[179:188]
10–11	Input	10–11	Not connected	eTPUA[10:11]	GPIO[124:125]
	Output	10–11	DSPI C[14:15]		eTPUA[22:23]_GPIO[124:125][2]
12–15	Input	12, 13–15	Not connected	eTPUA[12:15]	GPIO[126:129]
	Output	12, 13–15	DSPI C[0:3]		eTPUA[0:3]_GPIO[114:117]
16–19	Input	16, 17–19	Not connected	eTPUA[16:19]	GPIO[130:133]
	Output	16, 17–19	DSPI B[7:4][1] DSPI D[5:2][1]		eTPUA[4:7]_GPIO[118:121]
20–21	Input	20, 21	Not connected	eTPUA[20:21]	$\overline{\text{IRQ}}$[8:9]_GPIO[134:135][3]
	Output	20, 21	DSPI B[3:2][1] DSPI D[1:0][1]		eTPU A[8:9]_GPIO[122:123]
22–23	Input	22, 23	Not connected	eTPUA[22:23]	$\overline{\text{IRQ}}$[10:11]_GPIO[136:137][3]
	Output	22, 23			eTPU A[10:11]_GPIO[124:125]
24–27	Input	Not connected	DSPI B[13:10][1]	eTPUA[24:27]	Not connected
	Output	24, 25, 26, 27	DSPI B[13:10][1] DSPI D[15:12][1]	eTPUA[24:27]	$\overline{\text{IRQ}}$[12:15]_GPIO[138:141][3]
28–29	Input	Not connected	DSPI B[9:8][1]	eTPUA[28:29]	Not connected
	Output	28, 29	DSPI B[9:8][1] DSPI D[11:10][1]		GPIO[142:143]
30–31	Input	30, 31	Not connected	eTPUA[30:31]	GPIO[144:145]
	Output				

1 See the Signals chapter for package pin locations of these signals.

2 These signals are output only

3 These signals are input only

17.4.1.1 Time Base Clock Signal (TCRCLK[A])

The TCRCLK[A] input signals are used to control the TCR1 and TCR2 time bases for eTPU A.

NOTE

Throughout this document, TCRCLKA is referred to as TCRCLK.

There is one independent TCRCLK input for the engine. Table 17-2 shows the TCRCLK pin connections. For pulse accumulator operations TCRCLK can be used as a gate for a counter based on the system clock divided by eight. For angle operations TCRCLK can be used to get the tooth transition indications in angle mode. See the eTPU reference manual, sections 5.9 and 5.10 for further details.

Table 17-2. TCRCLK Signals

Signal Name	Pin Connection		Other Signals Muxed on Same Pin
	208 BGA	324 BGA	
TCRCLKA	L4	M2	$\overline{\text{IRQ}}$[7] (input only) GPIO[113]

17.4.1.2 Channel Output Disable Signals

The eTPU engine has four input signals that are used to force the outputs of a group of eight channels to an inactive level. These signals originate from the eMIOS. When an output disable signal is active, all eight channels assigned to the disable signal that have their ODIS bits set to 1 in ETPU_C*n*CR register have their outputs forced to the opposite of the value specified in the ETPU_C*n*CR[OPOL] bit. Therefore, individual channels can be selected to be affected by the output disable signals, as well as their disabling forced polarity.

The output disable channel groups are defined in Table 17-3.

Table 17-3. Output Disable Channel Groups

eMIOS Channel	Engine	eTPU Channels Disabled
11	A	0–7
10		8–15
9		16–23
8		24–31

17.5 Memory Map and Register Definition

17.5.1 Memory Map

The eTPU system simplified memory map is shown in Table 17-4. The base address for the eTPU module is listed as BASE. Each of the register areas shown can have their own reserved address areas.

Table 17-4 shows a high-level memory map.

NOTE

Devices with eTPU A engine only do not implement the eTPU B registers. Do not access these addresses and treat the memory as reserved.

Table 17-4. eTPU High-Level Memory Map

Address	Register Description
Base (0xC3FC_0000)–Base + 0x0000_001F	eTPU system module configuration registers
Base + 0x0000_0020– 0x0000_002F	eTPU A time base registers
Base + 0x0000_0030–0x0000_001FF	Reserved
Base + 0x0000_0200–0x0000_02FF	eTPU[A] global channel registers
Base + 0x0000_0300–0x0000_03FF	Reserved
Base + 0x0000_0400–0x0000_07FF	eTPU A channel registers
Base + 0x0000_0800–0x0000_7FFF	Reserved
Base + 0x0000_8000–0x0000_8BFF[1]	SDM (2.5 KBs)
Base + 0x0000_8C00–0x0000_BFFF	Reserved
Base + 0x0000_C000–0x0000_CBFF	SDM PSE mirror[2] (3 KBs)
Base + 0xCC00–0xFFFF	Reserved
Base + 0x0001_0000–0x0001_3FFF[3]	SCM (12 KBs)
Base + 0x0001_4000–0x0001_FFFF	Not writable. Reads the return value of ETPU_SCMOFFDATAR register.

1 Do not access addresses in the SDM memory block that are unused. These addresses are reserved.

2 Parameter Sign Extension access area. See the eTPU reference manual.

3 Do not access addresses in the SCM memory block that are unused. These addresses are reserved.

17.5.2 Register Description

Table 17-5 shows the eTPU registers and their locations, without examples or explanation of how the fields are used. For a complete description of these registers, see the *Enhanced Time Processing Unit (eTPU) Reference Manual*. The features are explained in detail there.

Table 17-5. Detailed Memory Map

Address	Register Name	Register Description	Bits
Base (0xC3FC_0000)	ETPU_MCR	eTPU module configuration register	32
Base + 0x0000_0004	ETPU_CDCR	eTPU coherent dual-parameter controller register	32
Base + 0x0000_0008	—	Reserved	—
Base + 0x0000_000C	ETPU_MISCCMPR	eTPU MISC compare register	32
Base + 0x0000_0010	ETPU_SCMOFFDATAR	eTPU SCM off-range data register	32
Base + 0x0000_0014	ETPU_ECR_A	eTPU A engine configuration register	32
Base + 0x0000_001C	—	Reserved	—
Base + 0x0000_0020	ETPU_TBCR_A	eTPU A time base configuration register	32
Base + 0x0000_0024	ETPU_TB1R_A	eTPU A time base 1	32
Base + 0x0000_0028	ETPU_TB2R_A	eTPU A time base 2	32
Base + 0x0000_002C	ETPU_REDCR_A	eTPU A STAC bus interface configuration	32
Base + 0x0000_0030–0x0000_01FF	—	Reserved	—
Base + 0x0000_0200	ETPU_CISR_A	eTPU A channel interrupt status	32
Base + 0x0000_0208–0x0000_020C	—	Reserved	—
Base + 0x0000_0210	ETPU_CDTRSR_A	eTPU A channel data transfer request status	32
Base + 0x0000_0218–0x0000_021C	—	Reserved	—
Base + 0x0000_0220	ETPU_CIOSR_A	eTPU A channel interrupt overflow status	32
Base + 0x0000_0228–0x0000_022C	—	Reserved	—
Base + 0x0000_0230	ETPU_CDTROSR_A	eTPU A channel data transfer request overflow status register	32
Base + 0x0000_0238–0x0000_023C	—	Reserved	—
Base + 0x0000_0240	ETPU_CIER_A	eTPU A channel interrupt enable register	32
Base + 0x0000_0248–0x0000_024C	—	Reserved	—
Base + 0x0000_0250	ETPU_CDTRER_A	eTPU A channel data transfer request enable register	32
Base + 0x0000_0258–0x0000_027F	—	Reserved	—
Base + 0x0000_0280	ETPU_CPSSR_A	eTPU A channel pending service status register	32
Base + 0x0000_0288–0x0000_028C	—	Reserved	—
Base + 0x0000_0290	ETPU_CSSR_A	eTPU A channel service status register	32
Base + 0x0000_0298–0x0000_03FF	—	Reserved	—

Table 17-5. Detailed Memory Map (continued)

Address	Register Name	Register Description	Bits
Base + 0x0000_0400	ETPU_C0CR_A	eTPU A channel 0 configuration register	32
Base + 0x0000_0404	ETPU_C0SCR_A	eTPU A channel 0 status and control register	32
Base + 0x0000_0408	ETPU_C0HSRR_A	eTPU A channel 0 host service request register	32
Base + 0x0000_040C	—	Reserved	—
Base + 0x0000_0410	ETPU_C1CR_A	eTPU A channel 1 configuration register	32
Base + 0x0000_0414	ETPU_C1SCR_A	eTPU A channel 1 status and control register	32
Base + 0x0000_0418	ETPU_C1HSRR_A	eTPU A channel 1 host service request register	32
Base + 0x0000_041C	—	Reserved	—
. . .	. . .	. . .	. . .
Base + 0x0000_05F0	ETPU_C31CR_A	eTPU A channel 31 configuration register	32
Base + 0x0000_05F4	ETPU_C31SCR_A	eTPU A channel 31 status and control register	32
Base + 0x0000_05F8	ETPU_C31HSRR_A	eTPU A channel 31 host service request register	32
Base + 0x0000_05FC–0x0000_7FFF	—	Reserved	—
Base + 0x0000_8000–0x0000_8BFF	—	Shared data memory (parameter RAM)	2.5 KBs
Base + 0x0000_8C00–0x0000_BFFF	—	Reserved	—
Base + 0x0000_C000–0x0000_CBFF	—	SDM PSE mirror[1]	2.5 KBs
Base + 0x0000_CC00–0x0000_FFFF	—	Reserved	—
Base + 0x0001_0000–0x0001_2FFF	SCM	Shared code memory[2]	12 KBs
Base + 0x0001_3000–0x0001_FFFF	—	Reserved	—

1 Parameter sign extension access area. See the eTPU reference manual.

2 SCM access is only available under certain conditions when ETPU_MCR[VIS] = 1. The SCM can only be written in 32-bit accesses.

17.5.2.1 System Configuration Registers

17.5.2.1.1 eTPU Module Configuration Register (ETPU_MCR)

This register is global to both eTPU engines, and resides in the shared BIU. ETPU_MCR gathers global configuration and status in the eTPU system, including global exception. It is also used for configuring the SCM (shared code memory) operation and test.

Address: Base + 0x0000_0000 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	MGEA	MGEB	ILFA	ILFB	0	0	0	SCMSIZE				
W	GEC															
Reset	0	0	0	0	0	0	0	0	0	0	0	SCMSIZE				

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	SCMMISF	SCMMISEN	0	0	VIS	0	0	0	0	0	GTBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-5. ETPU_MCR Register

Table 17-6. ETPU_MCR Field Descriptions

Field	Description
0 GEC	Global exception clear. Negates global exception request and clears global exception status bits MGEA, MGEB, ILFA, ILFB and SCMMISF. A read always returns 0. Writes have the following effect: 0 Keep global exception request and status bits ILFA, ILFB, MGEA, MGEB, and SCMMISF as is. 1 Negate global exception, clear status bits ILFA, ILFB, MGEA, MGEB, and SCMMISF. GEC works the same way with either one or both engines in stop mode.
1–3	Reserved
4 MGEA	Microcode global exception engine A. Indicates that a global exception was asserted by microcode executed on the respective engine. The determination of the reason why the global exception was asserted is application dependent: it can be coded in an SDM status parameter, for instance. This bit is cleared by writing 1 to GEC. 0 No microcode-requested global exception pending. 1 Global exception requested by microcode is pending.
5 MGEB	Microcode global exception engine B. Indicates that a global exception was asserted by microcode executed on the respective engine. The determination of the reason why the global exception was asserted is application dependent: it can be coded in an SDM status parameter, for instance. This bit is cleared by writing 1 to GEC. 0 No microcode requested global exception pending. 1 Global exception requested by microcode is pending.
6 ILFA	Illegal instruction flag eTPU A. Set by the microengine to indicate that an illegal instruction was decoded in engine A. This bit is cleared by host writing 1 to GEC. For more information about illegal instructions, see Section 9.6 in the eTPU reference manual. 0 Illegal Instruction not detected. 1 Illegal Instruction detected by eTPU A.

Table 17-6. ETPU_MCR Field Descriptions (continued)

Field	Description
7–10	Reserved
11–15 SCMSIZE [0:4]	SCM size. Holds the number of 2 KB SCM Blocks minus 1. This value is MCU-dependent.
16–20	Reserved
21 SCMMISF	SCM MISC Flag. Set by the SCM MISC (multiple input signature calculator) logic to indicate that the calculated signature does not match the expected value, at the end of a MISC iteration. For more details, see the eTPU reference manual for more details. 0 Signature mismatch not detected. 1 MISC has read entire SCM array and the expected signature in ETPU_MISCCMPR does not match the value calculated. This bit is cleared by writing 1 to GEC.
22 SCM MISEN	SCM MISC enable. Used for enabling/disabling the operation of the MISC logic. SCMMISEN is readable and writable at any time. The MISC logic only operates when this bit is set to 1. When the bit is reset the MISC address counter is set to the initial SCM address. When enabled, the MISC continuously cycles through the SCM addresses, reading each and calculating a CRC. To save power, the MISC can be disabled by clearing the SCMMISEN bit. For more details, see the eTPU reference manual. 0 MISC operation disabled. The MISC logic is reset to its initial state. 1 MISC operation enabled. (Toggling to 1 clears the SCMMISF bit) SCMMISEN is cleared automatically when MISC logic detects an error; that is, when SCMMISF transitions from 0 to 1, disabling the MISC operation.
23–24	Reserved
25 VIS	SCM visibility. Determines SCM visibility to the slave bus interface and resets the MISC state (but SCMMISEN keeps its value). 0 SCM is not visible to the slave bus. Accessing SCM address space issues a bus error. 1 SCM is visible to the slave bus. The MISC state is reset. This bit is write protected when any of the engines are not in halt or stop states. When VIS=1, the ETPU_ECR MDIS bits are write protected, and only 32-bit aligned SCM writes are supported. The value written to SCM is unpredictable if other transfer sizes are used.
26–30	Reserved
31 GTBE	Global time base enable. Enables time bases in both engines, allowing them to be started synchronously. An assertion of GTBE also starts the eMIOS time base[1]. This enables the eTPU time bases and the eMIOS time base to all start synchronously. 1 time bases in both eTPU engines and eMIOS are enabled to run. 0 time bases in both engines are disabled to run. **Note:** When GTBE is turned off with Angle Mode enabled, the EAC must be reinitialized before GTBE is turned on again.

[1] The eMIOS also has an GTBE bit. Assertion of either the eMIOS or eTPU GTBE bit starts time bases for the eMIOS and eTPU, see the eTPU reference manual.

17.5.2.1.2 eTPU Coherent Dual-Parameter Controller Register (ETPU_CDCR)

ETPU_CDCR configures and controls dual-parameter coherent transfers. For more information, see the eTPU reference manual.

Address: Base + 0x0000_0004 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STS	CTBASE							PBBASE							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PWIDTH	PARM0							WR	PARM1						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 17-6. eTPU Coherent Dual-Parameter Controller Register (ETPU_CDCR)

Table 17-7. ETPU_CDCR Field Descriptions

Field	Description
0 STS	Start. Set by the host to start the data transfer between the parameter buffer pointed by PBBASE and the target addresses selected by the concatenation of fields CTBASE and PARM0/1. The host receives wait-states until the data transfer is complete. Coherency logic resets STS once the data transfer is complete. For more information, see the eTPU reference manual. 0 (Write) does not start a coherent transfer. 1 (Write) starts a coherent transfer.
1–5 CTBASE [0:4]	Channel transfer base. This field concatenates with fields PARM0/PARM1 to determine the absolute offset (from the SDM base) of the parameters to be transferred: Parameter 0 address = {CTBASE, PARM0} × 4 + SDM base Parameter 1 address = {CTBASE, PARM1} × 4 + SDM base
6–15 PBBASE [0:9]	Parameter buffer base address. Points to the base address of the parameter buffer location, with granularity of 2 parameters (8 bytes). The host (byte) address of the first parameter in the buffer is PBBASE × 8 + SDM Base Address.
16 PWIDTH	Parameter width selection. Selects the width of the parameters to be transferred between the PB and the target address. 0 Transfer 24-bit parameters. The upper byte remains unchanged in the destination address. 1 Transfer 32-bit parameters. All 32 bits of the parameters are written in the destination address.
17–23 PARM0 [0:6]	Channel parameter number 0. This field in concatenation with CTBASE[3:0] determine the address offset (from the SDM base address) of the parameter which is the destination or source (defined by WR) of the coherent transfer. The SDM address offset of the parameter is {CTBASE, PARM0} × 4. PARM0 allows non-contiguous parameters to be transferred coherently[1].

Table 17-7. ETPU_CDCR Field Descriptions (continued)

Field	Description
24 WR	Read/Write selection. This bit selects the direction of the coherent data transfer. 0 Read operation. Data transfer is from the selected parameter RAM address to the PB. 1 Write operation. Data transfer is from the PB to the selected parameter RAM address.
25–31 PARM1 [0:6]	Channel parameter number 1. This field in concatenation with CTBASE[3:0] determines the address offset (from the SDM base) of the parameter which is the destination or source (defined by WR) of the coherent transfer. The SDM address offset of the parameter is {CTBASE, PARM1} × 4. PARM1 allows non-contiguous parameters to be transferred coherently[1].

[1] The parameter pointed by {CTBASE, PARM0} is the first transferred.

17.5.2.1.3 eTPU MISC Compare Register (ETPU_MISCCMPR)

The multiple input signature calculator compare register (ETPU_MISCCMPR) holds the 32-bit signature expected from the whole shared code memory (SCM) array. This register must be written by the host with the 32-bit word to be compared against the calculated signature at the end of the MISC cycle. For more details, see the *eTPU Reference Manual*.

Address: Base + 0x0000_000C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMISCCMP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMISCCMP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-7. eTPU MISC Compare Register (ETPU_MISCCMPR)

Table 17-8. ETPU_MISCCMPR Field Descriptions

Field	Description
0–31 EMISCCMP [0:31]	Expected multiple input signature calculator compare register value. For more information, see the eTPU reference manual.

17.5.2.1.4 eTPU SCM Off-Range Data Register (ETPU_SCMOFFDATAR)

ETPU_SCMOFFDATAR holds the 32-bit value returned when the SCM array is accessed at non implemented addresses, either by the host or by the microengine. This register can be written by the host with the 32-bit instruction to be executed by the microengine to recover from runaway code.

NOTE

The ETPU_SCMOFFDATAR reset value is the opcode of an instruction that disables matches, clears the TDLs and the MRLs; the opcode also issues an illegal instruction Global Exception, and ends the thread.

Address: Base + 0x0000_0010 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ETPUSCMOFFDATA[0:15]															
W																
Reset	1	1	1	1	0	0	1	1	0	1	1	1	0	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ETPUSCMOFFDATA[16:31]															
W																
Reset	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1

Figure 17-8. eTPU SCM Off-Range Data Register (ETPU_SCMOFFDATAR)

Table 17-9. ETPU_SCMOFFDATAR Field Descriptions

Field	Description
0–31 ETPU SCMOFF DATAR	SCM Off-range read data value.

17.5.2.1.5 eTPU Engine Configuration Register (ETPU_ECR)

Each engine has its own ETPU_ECR. The ETPU_ECR holds configuration and status fields that are programmed the eTPU engine.

Address: Base + 0x0000_0014 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEND	MDIS	0	STF	0	0	0	0	HLTF	0	0	0	0	FPSCK		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CDFC		0	0	0	0	0	0	0	0	0	ETB				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-9. eTPU Engine Configuration Register (ETPU_ECR)

Table 17-10. ETPU_ECR Field Descriptions

Field	Description
0 FEND	Force end. Assertion terminates any current running thread as if an END instruction have been executed. For more information, see the eTPU reference manual. 0 Normal operation. 1 Terminates current thread. This bit is self-negating.
1 MDIS	Module disable internal stop. This is the low power stop bit. When MDIS is set, the engine shuts down its internal clocks. TCR1 and TCR2 cease to increment, and input sampling stops. The engine asserts the stop flag (STF) bit to indicate that it has stopped. However, the BIU continues to run, and the host can access all registers except for the channel registers[1] and writes to time base registers. Section 17.6.0.3, “Channel Configuration and Control Registers.”After MDIS is set, even before STF asserts, data read from the channel registers is not meaningful, a Bus Error is issued, and writes are unpredictable. When the MDIS bit is asserted while the microcode is executing, the eTPU stops when the thread is complete. 0 eTPU engine runs. 1 Commands engine to stop its clocks. Stop completes on the next system clock after the stop condition is valid. The MDIS bit is write-protected when ETPU_MCR[VIS]=1. **Note:** After the MDIS has been switched from 1 to 0 or vice-versa, do not switch its value again until STF is switched to the same value.
2	Reserved
3 STF	Stop flag bit. Each engine asserts its stop flag (STF) to indicate that it has stopped. Only then the host can assume that the engine has actually stopped. The eTPU system is fully stopped when the STF bits of the eTPU engine is asserted. The engine only stops when any ongoing thread is complete in this case. 0 The engine is operating. 1 The engine has stopped (after the local MDIS bit has been asserted). Summarizing engine stop conditions, which STF reflects: STF A:= (after stop completed) MDIS A. STF A means the STF bit from engine A.
4–7	Reserved

Table 17-10. ETPU_ECR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>8
HLTF</td><td>Halt mode flag. If eTPU engine entered halt state, this flag is asserted. The flag remains asserted while the microengine is in halt state, even during a single-step or forced instruction execution. See the eTPU reference manual for further details about entering halt mode.
0 eTPU engine is not halted.
1 eTPU engine is halted.</td></tr>
<tr><td>9–12</td><td>Reserved</td></tr>
<tr><td>13–15
FPSCK
[0:2]</td><td>Filter prescaler clock control. Controls the prescaling of the clocks used in digital filters for the channel input signals and TCRCLK input. The following table illustrates filter prescaler clock control.
<table>
<tr><th>Filter Control</th><th>Sample on System Clock Divided by:</th></tr>
<tr><td>000</td><td>2</td></tr>
<tr><td>001</td><td>4</td></tr>
<tr><td>010</td><td>8</td></tr>
<tr><td>011</td><td>16</td></tr>
<tr><td>100</td><td>32</td></tr>
<tr><td>101</td><td>64</td></tr>
<tr><td>110</td><td>128</td></tr>
<tr><td>111</td><td>256</td></tr>
</table>
Filtering can be controlled independently by the engine, but all input digital filters in the same engine have same clock prescaling. For more details, see the eTPU reference manual.</td></tr>
<tr><td>16–17
CDFC
[0:1]</td><td>Channel digital filter control. Select a digital filtering mode for the channels when configured as inputs for improved noise immunity. Channel digital filter control is illustrated in the following table.
<table>
<tr><th>CDFC</th><th>Selected Digital Filter</th></tr>
<tr><td>00</td><td>TPU2/3 two sample mode: Using the filter clock which is the system clock divided by (2, 4, 8,..., 256) as a sampling clock (selected by FPSCK field in ETPU_ECR), comparing two consecutive samples which agree with each other sets the input signal state. This is the default reset state.</td></tr>
<tr><td>01</td><td>Invalid value.</td></tr>
<tr><td>10</td><td>eTPU three sample mode: Similar to the TPU2/3 two sample mode, but comparing three consecutive samples which agree with each other sets the input signal state.</td></tr>
<tr><td>11</td><td>eTPU continuous mode: Signal needs to be stable for the whole filter clock period. This mode compares all the values at the rate of system clock divided by two, between two consecutive filter clock pulses. If all the values agree with each other, input signal state is updated.</td></tr>
</table>
The eTPU has three digital filtering modes for the channels which provide programmable trade-off between signal latency and noise immunity. For more information on filtering, see the eTPU reference manual. Changing CDFC during eTPU normal input channel operation is not recommended since it changes the behavior of the transition detection logic while executing its operation.</td></tr>
</table>

Table 17-10. ETPU_ECR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>18–26</td><td>Reserved</td></tr>
<tr><td>27–31
ETB
[0:4]</td><td>Entry table base. Determines the location of the microcode entry table for the eTPU functions in SCM. More information about entry points is located in the eTPU reference manual. The following table shows the entry table base address options.
<table>
<tr><th>ETB</th><th>Entry Table Base Address for CPU Host Address (byte format)</th><th>Entry Table Base Address for Microcode Address (word format)</th></tr>
<tr><td>00000</td><td>0x0000_0000</td><td>0x0000_0000</td></tr>
<tr><td>00001</td><td>0x0000_0800</td><td>0x0000_0200</td></tr>
<tr><td>00010</td><td>0x0000_1000</td><td>0x0000_0400</td></tr>
<tr><td>.
.
.
.</td><td>.
.
.
.</td><td>.
.
.
.</td></tr>
<tr><td>11110</td><td>0x0000_F000</td><td>0x0000_3C00</td></tr>
<tr><td>11111</td><td>0x0000_F800</td><td>0x0000_3E00</td></tr>
</table>
</td></tr>
</table>

[1] The time base registers can still be read in stop mode, but writes are ineffective and a bus error is issued. Global channel registers and SDM can be accessed normally.

17.6 Time Base Registers

17.6.0.1 Time Base Registers

Time base registers allow the configuration and visibility of internally-generated time bases TCR1 and TCR2. There is one of these registers for the eTPU engine.

NOTE

Writes to this register issue a bus error and are ineffective when MDIS = 1.
Reads are always allowed.

17.6.0.1.1 eTPU Time Base Configuration Register (ETPU_TBCR)

This register configures several time base options.

Address: Base + 0x0000_0020 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCR2CTL			TCRCF		0	AM	0	0	0	TCR2P					
W																
Reset	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR1CTL		0	0	0	0	0	0	TCR1P							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-10. eTPU Time Base Configuration Register (ETPU_TBCR)

Table 17-11. ETPU_TBCR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0–2
TCR2CTL</td><td>TCR2 clock/gate control. Part of the TCR2 clocking system. These bits determine the clock source for TCR2 before the prescaler. TCR2 can count on any detected edge of the TCRCLK signal or use it for gating system clock divided by 8. After reset, the TCRCLK signal rising edge is selected. TCR2 can also be clocked by the system clock divided by 8. TCR2CTL also determines the TCRCLK edge selected for angle tooth detection in angle mode. See the eTPU Reference Manual for more information. TCR2 clock sources are listed in the following table.
<table>
<tr><th>TCR2CTL</th><th>AM=0
(TCR2 Clock)</th><th>AM=1
(Angle Tooth Detection)</th></tr>
<tr><td>000</td><td>Gated DIV8 clock (system clock / 8). When the external TCRCLK signal is low, the DIV8 clock is blocked, preventing it from incrementing the TCR2 prescaler. When the external TCRCLK signal is high, TCR2 prescaler is incremented at the frequency of the system clock divided by 8.</td><td>Do not use 000 with AM=1.</td></tr>
<tr><td>001</td><td>Rise transition on TCRCLK signal increments TCR2 prescaler.</td><td>Rising Edge</td></tr>
<tr><td>010</td><td>Fall transition on TCRCLK signal increments TCR2 prescaler.</td><td>Falling Edge</td></tr>
<tr><td>011</td><td>Rise or fall transition on TCRCLK signal increments TCR2 prescaler.</td><td>Rising or Falling Edge</td></tr>
<tr><td>100</td><td>DIV8 clock (system clock / 8)</td><td rowspan="4">Do not use 1XX with AM=1.</td></tr>
<tr><td>101</td><td>Invalid value</td></tr>
<tr><td>110</td><td>Invalid value</td></tr>
<tr><td>111</td><td>TCR2CTL shuts down TCR2 clocking, except on Angle Mode. TCR2 can also change as STAC client.</td></tr>
</table>
</td></tr>
</table>

Table 17-11. ETPU_TBCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>3–4
TCRCF</td><td>TCRCLK signal filter control. Controls the TCRCLK digital filter determining whether the TCRCLK signal input (after a synchronizer) is filtered with the same filter clock as the channel input signals or uses the system clock divided by 2, and also whether the TCRCLK digital filter works in integrator mode or two sample mode. The following table describes TCRCLK filter clock/mode.
<table>
<tr><th>TCRCF</th><th>Filter Input</th><th>Filter Mode</th></tr>
<tr><td>00</td><td>system clock divided by 2</td><td>two sample</td></tr>
<tr><td>01</td><td>filter clock of the channels</td><td>two sample</td></tr>
<tr><td>10</td><td>system clock divided by 2</td><td>integration</td></tr>
<tr><td>11</td><td>filter clock of the channels</td><td>integration</td></tr>
</table>
For more information, see the eTPU Reference Manual.</td></tr>
<tr><td>5</td><td>Reserved</td></tr>
<tr><td>6
AM</td><td>Angle mode selection. When the AM bit is set the EAC (eTPU Angle Clock) hardware provides angle information to the channels using the TCR2 bus. When the AM bit is cleared (non-angle mode), EAC operation is disabled, and its internal registers can be used as general purpose registers.
0 EAC operation is disabled
1 TCR2 works in angle mode
AM must not be changed when ETPU_MCR[GTBE] = 1.
Note: Changing AM can cause expurious transition detections on channel 0, depending on the channel mode and state.
For more information, see the eTPU Reference Manual.</td></tr>
<tr><td>7–9</td><td>Reserved</td></tr>
<tr><td>10–15
TCR2P</td><td>Timer count register 2 prescaler control. Part of the TCR2 clocking system. TCR2 is clocked from the output of a prescaler. The prescaler divides its input by (TCR2P+1) allowing frequency divisions from 1 to 64. The prescaler input is the system clock divided by 8 (in gated or non-gated clock mode) or Internal Timebase input, or TCRCLK filtered input. This field has no effect on TCCR2 in Angle Mode. For more information on TCR2, see the eTPU Reference Manual.</td></tr>
<tr><td>16–23
TCR1CTL</td><td>TCR1 clock/gate control. Part of the TCR1 clocking system. It determines the clock source for TCR1. TCR1 can count on detected rising edge of the TCRCLK signal or the system clock divided by 2. After reset TCRCLK signal is selected. The following table shows the selection of the TCR1 clock source.
<table>
<tr><th>TCR1CTL</th><th>TCR1 Clock</th></tr>
<tr><td>00</td><td>selects TCRCLK as clock source for the TCR1 prescaler (must not be use in Angle Mode)</td></tr>
<tr><td>01</td><td>Invalid value</td></tr>
<tr><td>10</td><td>selects system clock divided by 2 as clock source for the TCR1 prescaler</td></tr>
<tr><td>11</td><td>TCR1CTL shuts down TCR1 clock. TCR1 can still change if STAC client.</td></tr>
</table>
For more information on the TCR1 clocking system, see the Reference User's Manual.</td></tr>
<tr><td>24–31
TCR1P</td><td>Timer count register 1 prescaler control. Clocked from the output of a prescaler. The input to the prescaler is the internal eTPU system clock divided by 2 or the output of TCRCLK filter, or Peripheral Timebase input. The prescaler divides this input by (TCR1P+1) allowing frequency divisions from 1 up to 256.</td></tr>
</table>

17.6.0.1.2 eTPU Time Base 1 (TCR1) Visibility Register (ETPU_TB1R)

This register provides visibility of the TCR1 time base for core host read access. This register is read-only. The value of the TCR1 time base shown can be driven by the TCR1 counter or imported, depending on the configuration set in ETPU_REDCR. For more information, see the eTPU reference manual.

Address: Base + 0x0000_0024 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TCR1							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR1															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-11. eTPU Time Base 1 (TCR1) Visibility Register (ETPU_TB1R)

Table 17-12. ETPU_TB1R Field Descriptions

Field	Description
0–7	Reserved
8–31 TCR1 [0:23]	TCR1 value. Used on matches and captures. For more information, see the eTPU reference manual.

17.6.0.1.3 eTPU Time Base 2 (TCR2) Visibility Register (ETPU_TB2R)

This register provides visibility of the TCR2 time base for core host read access. This register is read-only. The value of the TCR2 time base shown can be driven by the TCR2 counter, the angle mode logic, or imported from the STAC interface, depending on angle mode (an engine cannot import when in angle mode) and STAC interface configurations set in registers ETPU_TBCR and ETPU_REDCR. For more information on time bases, see the eTPU reference manual.

Address: Base + 0x0000_0028 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TCR2							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR2															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-12. eTPU Time Base 2 (TCR2) Visibility Register (ETPU_TB2R)

Table 17-13. ETPU_TB2R Bit Field Descriptions

Field	Description
0–7	Reserved
8–31 TCR2 [0:23]	TCR2 value. Used on matches and captures. For information on TCR2, see the eTPU reference manual.

17.6.0.1.4 STAC Bus Configuration Register (ETPU_REDCR)

This register configures the eTPU STAC bus interface module and operation. For more information on the STAC interface, see the eTPU reference manual.

Address: Base + 0x0000_002C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REN1	RSC1	0	0	SERVER_ID1				0	0	0	0	SRV1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REN2	RSC2	0	0	SERVER_ID2				0	0	0	0	SRV2			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-13. STAC Bus Configuration Register (ETPU_REDCR)

Table 17-14. ETPU_REDCR Field Descriptions

Field	Description
0 REN1	TCR1 resource[1] client/server operation enable. Enables or disables client/server operation for the eTPU STAC interface. REN1 enables TCR1. 0 Server/client operation for resource 1 is disabled. 1 Server/client operation for resource 1 is enabled.
1 RSC1	TCR1 resource server/client assignment. Selects the eTPU data resource assignment to be used as a server or client. RSC1 selects the functionality of TCR1. For server mode, external plugging determines the unique server address assigned to each TCR. For a client mode, the SRV1 field determines the server address to which the client listens. 0 Resource client operation. 1 Resource server operation.
2–3	Reserved
4–7 SERVER_ID1	STAC bus address for TCR1 as a server. For more information on the STAC interface, see the eTPU reference manual.
8–11	Reserved
12–15 SRV1[0:3]	TCR1 resource server. Selects the address of the specific STAC Server the local TCR1 monitors when configured as a STAC client. For more information on the STAC interface, see the eTPU reference manual.
16 REN2	TCR2 resource[1] client/server operation enable. Enables or disables client/server operation for eTPU slave resources. REN2 enables TCR2 slave bus operations. 1 Server/client operation for resource 2 is enabled. 0 Server/client operation for resource 2 is disabled.
17 RSC2	TCR2[2] resource server/client assignment. Selects the eTPU data resource assignment to be used as a server or client. RSC2 selects the functionality of TCR2. For server mode, external plugging determines the unique server address assigned to each TCR. For a client mode, the SRV2 field determines the Server address to which the client listens. 0 Resource Client operation. 1 Resource Server operation.
18–19	Reserved
20–23 SERVER_ID2	STAC bus address for TCR2 as a server.
24–27	Reserved
28–31 SRV2[0:3]	TCR2 resource server. Selects the address of the specific STAC server the local TCR2 listens to when configured as a STAC Client. For more information on the STAC interface, see the eTPU reference manual.

[1] Resource identifies any parameter that changes in time and can be exported / imported from other device. For the eTPU, a resource can be TCR1 or TCR2 (either time or angle values).

[2] When TCR2 is configured as a STAC bus client (REN2 = 1, RSC2 = 0) the angle clock hardware must be disabled (ETPU_TBCR[AM] = 0).

17.6.0.2 Global Channel Registers

The registers in this section group, by type, the interrupt status and enable bits from all the channels. This organization eases management of all channels or groups of channels by a single interrupt handler routine. These bits are mirrored by the individual channel registers.

17.6.0.2.1 eTPU Channel Interrupt Status Register (ETPU_CISR)

Host interrupt status from all channels are grouped in ETPU_CISR. The bits are mirrored by the channels' status/control registers. For more information, see Section 17.6.0.3.3, "eTPU Channel n Status Control Register (ETPU_CnSCR)," and the eTPU reference manual.

NOTE

The host core must write 1 to clear an interrupt status bit.

Address: Base + 0x0000_0200 Access: R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIS31	CIS30	CIS29	CIS28	CIS27	CIS26	CIS25	CIS24	CIS23	CIS22	CIS21	CIS20	CIS19	CIS18	CIS17	CIS16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIS15	CIS14	CIS13	CIS12	CIS11	CIS10	CIS9	CIS8	CIS7	CIS6	CIS5	CIS4	CIS3	CIS2	CIS1	CIS0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-14. eTPU Channel Interrupt Status Register (ETPU_CISR)

Table 17-15. ETPU_CISR Field Descriptions

Field	Description
0–31 CIS*n*	Channel *n* interrupt status. 0 indicates that channel *n* has no pending interrupt to the host core. 1 indicates that channel *n* has a pending interrupt to the host core. To clear a status bit, the host must write 1 to it. For details about interrupts see the eTPU reference manual.

17.6.0.2.2 eTPU Channel Data Transfer Request Status Register (ETPU_CDTRSR)

Data transfer request status from all channels are grouped in ETPU_CDTRSR. The bits are mirrored by the channels' status/control registers. For more information on data transfers and channel control registers, see the eTPU reference manual.

Address: Base + 0x0000_0210 Access: R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DTRS 31	DTRS 30	DTRS 29	DTRS 28	DTRS 27	DTRS 26	DTRS 25	DTRS 24	DTRS 23	DTRS 22	DTRS 21	DTRS 20	DTRS 19	DTRS 18	DTRS 17	DTRS 16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DTRS 15	DTRS 14	DTRS 13	DTRS 12	DTRS 11	DTRS 10	DTRS 9	DTRS 8	DTRS 7	DTRS 6	DTRS 5	DTRS 4	DTRS 3	DTRS 2	DTRS 1	DTRS 0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-15. eTPU Channel Data Transfer Request Status Register (ETPU_CDTRSR)

Table 17-16. ETPU_CDTRSR Field Descriptions

Field	Description
0–31 DTRS*n*	Channel *n* data transfer request status. 0 Indicates that channel *n* has no pending data transfer request. 1 Indicates that channel *n* has a pending data transfer request. To clear a status bit, the host must write 1 to it. For details about data transfer requests see the eTPU reference manual.

17.6.0.2.3 eTPU Channel Interrupt Overflow Status Register (ETPU_CIOSR)

An interrupt overflow occurs when an interrupt is issued for a channel when the previous interrupt status bit for the same channel has not been cleared. Interrupt overflow status from all channels are grouped in ETPU_CIOSR. The bits are mirrored by the channels' status/control registers. For information about channel status registers and overflow, see Section 17.6.0.3.3, "eTPU Channel n Status Control Register (ETPU_CnSCR)," and the eTPU reference manual.

NOTE

The host must write 1 to clear an interrupt overflow status bit.

Address: Base + 0x0000_0220 Access: R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIOS 31	CIOS 30	CIOS 29	CIOS 28	CIOS 27	CIOS 26	CIOS 25	CIOS 24	CIOS 23	CIOS 22	CIOS 21	CIOS 20	CIOS 19	CIOS 18	CIOS 17	CIOS 16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIOS 15	CIOS 14	CIOS 13	CIOS 12	CIOS 11	CIOS 10	CIOS 9	CIOS 8	CIOS 7	CIOS 6	CIOS 5	CIOS 4	CIOS 3	CIOS 2	CIOS 1	CIOS 0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-16. eTPU Channel Interrupt Overflow Status Register (ETPU_CIOSR)

Table 17-17. ETPU_CIOSR Field Descriptions

Field	Description
0–31 CIOS*n*	Channel *n* interrupt overflow status. 0 indicates that no interrupt overflow occurred in the channel. 1 indicates that an interrupt overflow occurred in the channel. To clear a status bit, the host must write 1 to it. For details about interrupts see the eTPU reference manual.

17.6.0.2.4 eTPU Channel Data Transfer Request Overflow Status Register (ETPU_CDTROSR)

Data transfer request overflow status from all channels are grouped in ETPU_CDTROSR. The bits are mirrored by the channels' status/control registers. For more information on channel status registers and data transfer request overflow, see Section 17.6.0.3.3, "eTPU Channel n Status Control Register (ETPU_CnSCR)," and the eTPU reference manual.

NOTE

The host must write 1 to clear a data transfer request overflow status bit.

Address: Base + 0x0000_0230 Access: R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DTR OS31	DTR OS30	DTR OS29	DTR OS28	DTR OS27	DTR OS26	DTR OS25	DTR OS24	DTR OS23	DTR OS22	DTR OS21	DTR OS20	DTR OS19	DTR OS18	DTR OS17	DTR OS16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DTR OS15	DTR OS14	DTR OS13	DTR OS12	DTR OS11	DTR OS10	DTR OS9	DTR OS8	DTR OS7	DTR OS6	DTR OS5	DTR OS4	DTR OS3	DTR OS2	DTR OS1	DTR OS0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-17. eTPU Channel Data Transfer Request Overflow Status Register (ETPU_CDTROSR)

Table 17-18. ETPU_CDTROSR Field Descriptions

Field	Description
0–31 DTROS*n*	Channel *n* data transfer request overflow status. 0 indicates that no data transfer request overflow occurred in the channel. 1 indicates that a data transfer request overflow occurred in the channel. To clear a status bit, the host must write 1 to it. For details about data transfer request overflow, see the eTPU reference manual.

17.6.0.2.5 eTPU Channel Interrupt Enable Register (ETPU_CIER)

The host interrupt enable bits for all 32 channels are grouped in ETPU_CIER. The bits are mirrored by the channel configuration registers. For more information on channel configuration registers and interrupt enable, see Section 17.6.0.3.2, "eTPU Channel n Configuration Register (ETPU_CnCR)," and the eTPU reference manual.

Address: Base + 0x0000_0240 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CIE 31	CIE 30	CIE 29	CIE 28	CIE 27	CIE 26	CIE 25	CIE 24	CIE 23	CIE 22	CIE 21	CIE 20	CIE 19	CIE 18	CIE 17	CIE 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CIE 15	CIE 14	CIE 13	CIE 12	CIE 11	CIE 10	CIE 9	CIE 8	CIE 7	CIE 6	CIE 5	CIE 4	CIE 3	CIE 2	CIE 1	CIE 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-18. eTPU Channel Interrupt Enable Register (ETPU_CIER)

Table 17-19. ETPU_CIER Field Descriptions

Field	Description
0–31 CIE*n*	Channel *n* interrupt enable. Enable the eTPU channels to interrupt the device core. 0 Interrupt disabled for channel *n*. 1 Interrupt enabled for channel *n*. For details about interrupts see the eTPU reference manual.

17.6.0.2.6 eTPU Channel Data Transfer Request Enable Register (ETPU_CDTRER)

Data transfer request enable status bits from all channels are grouped in ETPU_CDTRER. The bits are mirrored in the channels' configuration registers. For more on configuration registers and data transfer request enable, see Section 17.6.0.3.2, "eTPU Channel n Configuration Register (ETPU_CnCR)," and the eTPU reference manual."

Address: Base + 0x0000_0250 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DTRE 31	DTRE 30	DTRE 29	DTRE 28	DTRE 27	DTRE 26	DTRE 25	DTRE 24	DTRE 23	DTRE 22	DTRE 21	DTRE 20	DTRE 19	DTRE 18	DTRE 17	DTRE 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DTRE 15	DTRE 14	DTRE 13	DTRE 12	DTRE 11	DTRE 10	DTRE 9	DTRE 8	DTRE 7	DTRE 6	DTRE 5	DTRE 4	DTRE 3	DTRE 2	DTRE 1	DTRE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-19. eTPU Channel Data Transfer Request Enable Register (ETPU_CDTRER)

Table 17-20. ETPU_CDTRER Field Descriptions

Field	Description
0–31 DTRE*n*	Channel *n* data transfer request enable. Enable data transfer requests for their respective channels. 0 Data transfer request disabled for channel *n*. 1 Data transfer request enabled for channel *n*. For details about interrupts see the eTPU reference manual.

17.6.0.2.7 eTPU Channel Pending Service Status Register (ETPU_CPSSR)

ETPU_CPSSR is a read-only register that holds the status of the pending channel service requests. For information on channel service requests, see the eTPU reference manual.

NOTE

More than one source can request service when a channel's service request bit is set.

Address: Base + 0x0000_0280 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SR31	SR30	SR29	SR28	SR27	SR26	SR25	SR24	SR23	SR22	SR21	SR20	SR19	SR18	SR17	SR16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SR15	SR14	SR13	SR12	SR11	SR10	SR9	SR8	SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-20. eTPU Channel Pending Service Status Register (ETPU_CPSSR)

Table 17-21. ETPU_CPSSR Bit Field Descriptions

Field	Description
0–31 SR*n*	Pending service request *n*. Indicates a pending service request for channel *n*. The SR status for the pending request is negated at the time slot transition for the respective service thread. 0 no service request pending for channel *n* 1 pending service request for channel *n*

NOTE

The pending service status bit for a channel is set when a service request is pending, even if the Channel is disabled (CPR*n* = 0).

17.6.0.2.8 eTPU Channel Service Status Register (ETPU_CSSR)

ETPU_CSSR holds the current channel service status on whether it is being serviced or not. Only one bit can be asserted in this register at a given time. When no channel is being serviced the register read value is 0x0000_0000. ETPU_CSSR is a read-only register. The register can be read during normal eTPU operation for monitoring the scheduler activity. For more information on channels being serviced, see the eTPU reference manual.

NOTE

The ETPU_CSSR is not an absolute indication of channel status. If more than one source is requesting service, the asserted status bit only indicates that one of the requests has been granted.

NOTE

Channel service status does not always reflect decoding of the CHAN register, since the CHAN register can be changed by the service thread microcode.

Address: Base + 0x0000_0290 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SS31	SS30	SS29	SS28	SS27	SS26	SS25	SS24	SS23	SS22	SS21	SS20	SS19	SS18	SS17	SS16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SS15	SS14	SS13	SS12	SS11	SS10	SS9	SS8	SS7	SS6	SS5	SS4	SS3	SS2	SS1	SS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-21. ETPU_CSSR Register

Table 17-22. ETPU_CSSR Field Descriptions

Field	Description
0–31 SS*n*	Service status *n*. Indicates that channel *n* is currently being serviced. It is updated at the 1st microcycle of a time slot transition. 0 channel *n* is not currently being serviced 1 channel *n* is currently being serviced See the eTPU reference manual for more information on time slot transitions.

17.6.0.3 Channel Configuration and Control Registers

Each channel, for both eTPU engines, has a group of three registers used to control, configure and check status of that channel as shown in Table 17-23.

Table 17-23. Channel Registers Structure

Channel Offset	Register Name
0x0000	eTPU channel configuration register (ETPU_C*n*CR)
0x0004	eTPU channel status/control register (ETPU_C*n*SCR)
0x0008	eTPU channel host service request register (ETPU_C*n*HSRR)
0x000C	Reserved

17.6.0.3.1 Channel Registers Layout

One contiguous area is used to map all channel registers of each eTPU engine as shown in Table 17-24.

Table 17-24. eTPU Channel Register Map

Address	Registers Structure
Base + 0x0000_0400	eTPU A channel 0 register structure
Base + 0x0000_0410	eTPU A channel 1 register structure
Base + 0x0000_0420	eTPU A channel 2 register structure
Base + 0x0000_0430–0x0000_05D0	. . .
Base + 0x0000_05E0	eTPU A channel 30 register structure
Base + 0x0000_05F0	eTPU A channel 31 register structure
Base + 0x0000_0600–0x0000_07FF	Reserved

There are 32 structures defined, one for each available channel in the eTPU. The base address for the structure presented can be calculated by using the following equation:

Channel_Register_Structure_Base_Address =
ETPU_Engine_Channel_Base + (channel_number × 0x0000_0010)

where:

ETPU_Engine_Channel_Base = ETPU_Base + (0x0000_0400).

17.6.0.3.2 eTPU Channel *n* Configuration Register (ETPU_C*n*CR)

The ETPU_C*n*CR is a collection of the configuration bits related to an individual channel. Some of these bits are mirrored from the global channel registers.

Address: Channel_Register_Base + 0x0000 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIE	DTRE	CPR		0	0	ETPD	ETCS	0	0	0	CFS				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ODIS	OPOL	0	0	0	CPBA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-22. eTPU Channel *n* Configuration Register (ETPU_C*n*CR)

Table 17-25. ETPU_C*n*CR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0
CIE</td><td>Channel interrupt enable. This bit is mirrored from the ETPU_CIER
0 Disable interrupt for this channel.
1 Enable interrupt for this channel.
For more information, see the eTPU reference manual.</td></tr>
<tr><td>1
DTRE</td><td>Channel data transfer request enable. This bit is mirrored from the ETPU_CDTRER.
0 Disable data transfer request for this channel.
1 Enable data transfer request for this channel.
See the eTPU reference manual for more information.</td></tr>
<tr><td>2–3
CPR
[0:1]</td><td>Channel priority. Defines the priority level for the channel. The priority level is used by the hardware scheduler. The values for CPR[1:0] and levels are:
<table>
<tr><th>CPR</th><th>Priority</th></tr>
<tr><td>00</td><td>Disabled</td></tr>
<tr><td>01</td><td>Low</td></tr>
<tr><td>10</td><td>Middle</td></tr>
<tr><td>11</td><td>High</td></tr>
</table>
For more information on the hardware scheduler, see the eTPU reference manual.</td></tr>
<tr><td>4–5</td><td>Reserved</td></tr>
<tr><td>6
ETPD</td><td>This bit selects which channel signal, input or output, is used in the entry point selection. The ETPD value has to be compatible with the function chosen for the channel, selected in the field CFS. For details about entry table and condition encoding schemes, see the eTPU reference manual.
0 Use PSTI for entry point selection.
1 Use PSTO for entry point selection.</td></tr>
<tr><td>7
ETCS</td><td>Entry table condition select. Determines the channel condition encoding scheme that selects the entry point to be taken in an entry table. The ETCS value has to be compatible with the function chosen for the channel, selected in ETPU_CnCR[CFS]. Two condition encoding schemes are available.
1 Select alternate entry table condition encoding scheme.
0 Select standard entry table condition encoding scheme.For details about entry table and condition encoding schemes, see the eTPU reference manual.</td></tr>
<tr><td>8–10</td><td>Reserved</td></tr>
<tr><td>11–15
CFS
[0:4]</td><td>Channel function select. Defines the function to be performed by the channel. The function assigned to the channel has to be compatible with the channel condition encoding scheme, selected by ETPU_CnCR[ETCS]. For more information about functions, see the eTPU reference manual.</td></tr>
<tr><td>16
ODIS</td><td>Output disable. Enables the channel to have its output forced to the value opposite to OPOL when the output disable input signal corresponding to the channel group that it belongs is active.
0 Turns off the output disable feature for the channel.For more information on output disable, see the eTPU reference manual.
1 Turns on the output disable feature for the channel.</td></tr>
<tr><td>17
OPOL</td><td>Output polarity. Determines the output signal polarity. The activation of the output disable signal forces, when enabled by ETPU_CnCR[ODIS], the channel output signal to the opposite of this polarity.
0 Output active low (output disable drives output to high)
1 Output active high (output disable drives output to low)</td></tr>
</table>

Table 17-25. ETPU_C*n*CR Field Descriptions (continued)

Field	Description
18–20	Reserved
21–31 CPBA [0:10]	Channel *n* parameter base address. The value of this field multiplied by 8 specifies the SDM parameter base host (byte) address for channel *n* (2-parameter granularity). The formula for calculating the absolute channel parameter base (byte) address, as seen by the host, is eTPU_Base + 0x8000 + CPBA x 8. The SDM is mirrored in the parameter sign extension (PSE) area. The formula to calculate the absolute channel parameter base (byte) address in the PSE area is eTPU_Base + 0xC000 + CPBA x 8. For more information on SDM addresses, see the eTPU reference manual.

17.6.0.3.3 eTPU Channel *n* Status Control Register (ETPU_C*n*SCR)

ETPU_C*n*SCR is a collection of the interrupt status bits of the channel, and also the function mode definition (read-write). Bits CIS, CIOS, DTRS, and DTROS for each channel can also be accessed from ETPU_CISR, ETPU_CIOSR, ETPU_CDTRSR, and ETPU_CDTROSR respectively. For more information on the three previously mentioned registers, see the eTPU reference manual.

NOTE

The device core must write 1 to clear a status bit.

Address: Channel_Register_Base + 0x0004 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIS	CIOS	0	0	0	0	0	0	DTRS	DTROS	0	0	0	0	0	0
W	w1c	w1c							w1c	w1c						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IPS	OPS	0	0	0	0	0	0	0	0	0	0	0	0	FM	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-23. eTPU Channel *n* Status Control Register (ETPU_C*n*SCR)

Table 17-26. ETPU_C*n*SCR Field Descriptions

Field	Description
0 CIS	Channel interrupt status. 0 Channel has no pending interrupt to the device core. 1 Channel has a pending interrupt to the device core. CIS is mirrored in the ETPU_CISR. The core must write 1 to clear CIS. For more information on ETPU_CISR and interrupts, see Section 17.6.0.2.1, "eTPU Channel Interrupt Status Register (ETPU_CISR)," and the eTPU reference manual.
1 CIOS	Channel interrupt overflow status. 0 Interrupt overflow negated for this channel 1 Interrupt overflow asserted for this channel CIOS is mirrored in the ETPU_CIOSR. The core must write 1 to clear CIOS. For more information on the ETPU_CIOSR and interrupt overflow, see Section 17.6.0.2.3, "eTPU Channel Interrupt Overflow Status Register (ETPU_CIOSR)." and the eTPU reference manual.
2–7	Reserved
8 DTRS	Data transfer request status. 0 Channel has no pending data transfer request. 1 Channel has a pending data transfer request. DTRS is mirrored in the ETPU_CDTRSR. The core must write 1 to clear DTRS. For more information on the ETPU_CDTRSR and data transfer, see Section 17.6.0.2.2, "eTPU Channel Data Transfer Request Status Register (ETPU_CDTRSR)." and the eTPU reference manual.
9 DTROS	Data transfer request overflow status. 0 Data transfer request overflow negated for this channel. 1 Data transfer request overflow asserted for this channel. DTROS is mirrored in the ETPU_CDTROSR. The core must write 1 to clear DTROS. See Section 17.6.0.2.4, "eTPU Channel Data Transfer Request Overflow Status Register (ETPU_CDTROSR)." and the eTPU reference manual for more information on ETPU_CDTROSR and data transfer overflows.
10–15	Reserved
16 IPS	Channel input pin state. Shows the current value of the filtered channel input signal state.
17 OPS	Channel output pin state. Shows the current value driven in the channel output signal, including the effect of the external output disable feature. If the channel input and output signals are connected to the same pad, OPS reflects the value driven to the pad. This is not necessarily the actual pad value, which drives the value in the IPS bit.
18–29	Reserved
30–31 FM [0:1]	Channel function mode.[1] Each function can use this field for specific configuration. These bits can be tested by microengine code.

[1] These bits are equivalent to the TPU/TPU2/TPU3 host sequence (HSQ) bits.

17.7 Functional Description

See the eTPU User's Manual for information regarding the functional description of the eTPU module.

17.8 Initialization and Application Information

After initial power-on reset, the eTPU remains in an idle state (except when debug is asserted on power-on reset—in this case, the microengine awakens in halt state). In addition, the SCM must be initialized with the eTPU application prior to configuring the eTPU.

Chapter 18
Enhanced Queued Analog-to-Digital Converter (eQADC)

18.1 Introduction

The enhanced queued analog-to-digital converter (eQADC) provides accurate and fast conversions for a wide range of applications. The eQADC provides a parallel interface to two on-chip analog-to-digital converters (ADCs), and a single master to single slave serial interface to an off-chip external device. The two on-chip ADCs are architected to allow access to all the analog channels.

NOTE

The 324 package supports 40 channels in the dual eQADC engines. The 208 package supports 34 channels only.

18.1.1 Block Diagram

Figure 18-1 shows the primary components inside the eQADC.

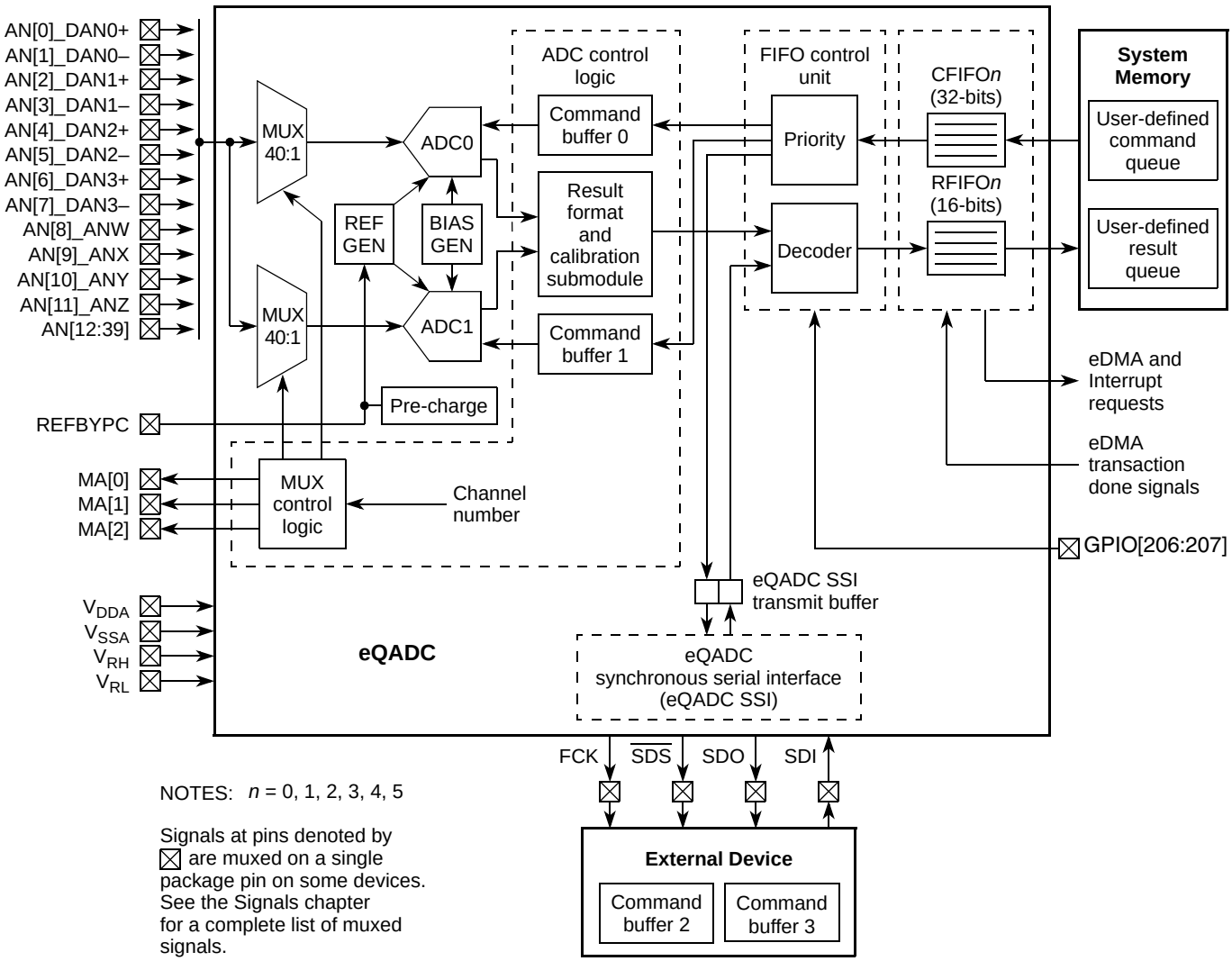


Figure 18-1. Simplified eQADC Block Diagram

18.1.2 Overview

The eQADC transfers commands from multiple command FIFOs (CFIFOs) to the on-chip ADCs or to the external device. The module can also in parallel (independently of the CFIFOs) receive data from the on-chip ADCs or from an off-chip external device into multiple result FIFOs (RFIFOs). The eQADC supports software and external hardware triggers from other modules to initiate transfers of commands from the CFIFOs to the on-chip ADCs or to the external device. (See Section 6.5.5.1, “eQADC External Trigger Input Multiplexing.”) It also monitors the amount of memory currently in use by each the CFIFO and RFIFO to detect underflow and overflow conditions.

A CFIFO underflow occurs when a CFIFO:

- Is in the TRIGGERED state; and
- Becomes empty.

An RFIFO overflow occurs when an RFIFO:

- Becomes full; and
- Host CPU or eDMA data is waiting to transmit to the RFIFO.

The eQADC generates eDMA or interrupt requests to control data movement between the FIFOs and the system memory, which is external to the eQADC.

The eQADC consists of the FIFO control unit which controls the CFIFOs and the RFIFOs, two ADCs with control logic, and the eQADC synchronous serial interface (eQADC SSI) which allows communication with an external device. There are six CFIFOs and six RFIFOs, each with four entries.

The FIFO control unit performs the following functions:

- Prioritizes the CFIFOs to determine which CFIFOs transfer commands
- Supports software and hardware triggers to start command transfers from a particular CFIFO
- Decodes command data from the CFIFOs and sends the commands to one of the two on-chip ADCs or to the external device
- Decodes result data from on-chip ADCs or from the external device, and transfers data to the RFIFO

The ADC control logic manages the execution of commands bound for on-chip ADCs from the CFIFOs and with the RFIFOs via the result format and calibration submodule. The ADC control logic performs the following functions:

- Buffers command data for execution
- Decodes command data and accordingly generates control signals for the two on-chip ADCs
- Formats and calibrates conversion result data coming from the on-chip ADCs
- Generates the internal multiplexer control signals and the select signals used by the external multiplexers

The eQADC SSI allows for a full duplex, synchronous, serial communication between the eQADC and an external device.

Figure 18-1 also depicts data flow through the eQADC. Commands are contained in system memory in a user-defined queue data structure. Command data is moved from the command queue you defined to the CFIFOs by either the host CPU or by the eDMA. After a CFIFO is triggered and becomes the highest priority, CFIFO command data is transferred from the CFIFO to the on chip ADCs, or to the external device. The ADC executes the command, and the result is moved through the result format and calibration submodule and to the RFIFO. The RFIFO target is specified by a field in the command that initiated the conversion. Data from the external device bypasses the result format and calibration submodule and is moved directly to its specified RFIFO. When data is stored in an RFIFO, data is moved from the RFIFO by the host CPU or by the eDMA to a data structure in system memory depicted in Figure 18-1 as a user-defined result queue.

For users familiar with the QADC, the eQADC system upgrades the functionality provided by that module. See Section 18.5.7, “eQADC versus QADC,” for a comparison between the eQADC and QADC.

18.1.3 Features

The eQADC includes these distinctive features:

- Two independent on-chip RSD cyclic ADCs
  - 12 bit AD resolution.
  - Targets up to 10 bit accuracy at 400 kilosamples per second (ADC_CLK = 6 MHz) and eight bit accuracy at 800 kilosamples per second (ADC_CLK = 12 MHz) for differential conversions.
  - Differential conversions (range -2.5 V to +2.5 V).
  - Single-ended signal range from 0–5 V.
  - Sample times of two (default), 8, 64, or 128 ADC clock cycles.
  - Sample time stamp information when requested.
  - Parallel interface to eQADC CFIFOs and RFIFOs.
  - Supports right-justified unsigned and signed formats for conversion results.
- Optional automatic application of ADC calibration constants: provision of reference voltages (25% V_{REF}[1] and 75% V_{REF}) for ADC calibration purposes
- 40 input channels available to the two on-chip ADCs
- Four pairs of differential analog input channels
- Full duplex synchronous serial interface to an external device
  - A free-running clock is provided for use by the external device
  - Supports a 26-bit message length
  - Transmits a null message when there are no triggered CFIFOs with commands bound for external command buffers, or when there are triggered CFIFOs with commands bound for external command buffers but the external command buffers are full
- Priority-based CFIFOs
  - Supports six CFIFOs with fixed priority. The lower the CFIFO number, the higher its priority. Supports software and several hardware trigger modes to arm a particular CFIFO.
  - Generates interrupt when command coherency is not achieved.
- External hardware triggers
  - Supports rising edge, falling edge, high level and low level triggers
  - Supports configurable digital filter
- Upgrades the functionality of the QADC

18.1.4 Modes of Operation

This section describes the operating modes of the eQADC.

1. $V_{REF} = V_{RH} - V_{RL}$.

18.1.4.1 Normal Mode

This is the default operational mode when the eQADC is not in background debug or stop mode.

18.1.4.2 Debug Mode

Upon a debug mode entry request, eQADC behavior varies according to the status of the DBG field in Section 18.3.2.1, “eQADC Module Configuration Register (EQADC_MCR).” If DBG is programmed to 0b00, the debug mode entry request is ignored. If DBG is programmed to 0b10 or to 0b11, the eQADC enters debug mode. In case the eQADC SSI is enabled, the free running clock (FCK) output to external device does not stop when DBG is programmed to 0b11, but FCK stops in low phase, when DBG is programmed to 0b10.

During debug mode, the eQADC does not transfer commands from any CFIFOs, no null messages are transmitted to the external device, no data is returned to any RFIFO, no hardware trigger event is captured, and all eQADC registers can be accessed as in normal mode. Access to eQADC registers implies that CFIFOs can still be triggered using software triggers, because no scheme is implemented to write-protect registers during debug mode. eDMA and interrupt requests continue to be generated as in normal mode.

If at the time the debug mode entry request is detected, there are commands in the ADC that were already under execution, these commands are completed but the generated results, if any, are not sent to the RFIFOs until debug mode is exited. Commands that have not begun to execute are not executed until after exiting debug mode. The clock with an on-chip ADC stops, during its low phase, after the ADC stops executing commands. The time base counter only stops after all on-chip ADCs stop executing commands.

When exiting debug mode, the eQADC relies on the FIFO control unit and on the CFIFO status to determine the next command entry to transfer.

The eQADC internal behavior after the debug mode entry request is detected differs depending on the status of command transfers.

- No command transfer is in progress.

  The eQADC immediately halts future command transfers from any CFIFO.

  If a null message is being transmitted, eQADC completes the serial transmission before halting future command transfers. If valid data (conversion result or data read from an ADC register) is received by the result format and calibration submodule at the end of transmission, this data is not sent to an RFIFO until debug mode is exited.

  If the null message transmission is aborted, the eQADC completes the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission is transmitted only after exiting debug mode.
- Command transfer is in progress.

  eQADC completes the transfer and updates CFIFO status before halting future command transfers from any CFIFO.

  Command transfers to the external device are considered completed when the serial transmission of the command is completed. If valid data (conversion result or data read from an ADC register) is received at the end of a serial transmission, it is not sent to an RFIFO until debug mode is exited. The CFIFO status bits are still updated after the completion of the serial transmission, therefore,

after debug mode entry request is detected, the eQADC status bits stop changing several system clock cycles after the on-going serial transmission completes.

If the command message transmission aborts, the eQADC completes the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission is transmitted only after debug mode exits.

- Command/null message transfer through serial interface was aborted but next serial transmission did not start.

  If the debug mode entry request is detected between the time a previous serial transmission was aborted and the start of the next transmission, the eQADC completes the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission is transmitted only after debug mode exits.

18.1.4.3 Stop Mode

Upon a stop mode entry request detection, the eQADC progressively halts its operations until it reaches a static, stable state from which it can recover when returning to normal mode. The eQADC then asserts an acknowledge signal, indicating that it is static and that the clock input can be stopped. In stop mode, the free running clock (FCK) output to external device stops during its low phase if the eQADC SSI is enabled, and no hardware trigger events is captured. No capturing of hardware trigger events means that — as long as the system clock is running — CFIFOs can still be triggered using software triggers because no scheme is implemented to write-protect registers during stop mode.

If at the time the stop mode entry request is detected, there are commands in the ADC that were already under execution, these commands are completed but the generated results, if any, are not sent to the RFIFOs until stop mode is exited. Commands whose execution has not started do not execute until stop mode exits.

After these remaining commands are executed, the clock input to the ADCs is stopped. The time base counter stops after all on-chip ADCs cease executing commands and then the stop acknowledge signal is asserted. When exiting stop mode, the eQADC relies on the CFIFO operation modes and on the CFIFO status to determine the next command entry to transfer.

The eQADC internal behavior after the stop mode entry request is detected differs depending on the status of the command transfer.

- No command transfer is in progress

  The eQADC immediately halts future command transfers from any CFIFO.

  If a null message is being transmitted, eQADC completes the transmission before halting future command transfers. If valid data (conversion result or data read from an ADC register) is received at the end of the transmission, it is not sent to an RFIFO until stop mode exits.

  If the null message transmission is aborted, the eQADC completes the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission is only transmitted after stop mode exits.

- Command transfer is in progress.

  The eQADC completes the transfer and update CFIFO status before halting future command transfers from any CFIFO.

Command transfers to the external device are considered completed when the serial transmission of the command is completed. If valid data (conversion result or data read from an ADC register) is received at the end of a serial transmission, it is not sent to an RFIFO until stop mode exits. The CFIFO status bits are still updated after the completion of the serial transmission, therefore, after stop mode entry request is detected, the eQADC status bits stop changing several system clock cycles after the on-going serial transmission completes.

If the command message transmission is aborted, the eQADC completes the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission are transmitted only after stop mode exits.

- Command/null message transfer through serial interface was aborted but next serial transmission did not start.

  If the stop mode entry request is detected between the time a previous serial transmission was aborted and the start of the next transmission, the eQADC completes the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission is transferred only after stop mode is exited.

18.2 External Signal Description

These signals are external to the eQADC module, but are not necessarily physical pins. See Chapter 2, “Signals” for a complete list of all physical pins and signals.

Table 18-1. eQADC External Signals

Function	Description	I/O Type	Status During Reset[1]	Status After Reset[2]	Type	Package
AN[0]_ DAN0+	Single-ended analog input 0 Positive terminal differential input	I	I / —	AN[0] / —	Analog	496 324 208
AN[1]_ DAN0-	Single-ended analog input 1 Negative terminal differential input	I	I / —	AN[1] / —	Analog	496 324 208
AN[2]_ DAN1+	Single-ended analog input 2 Positive terminal differential input	I	I / —	AN[2] / —	Analog	496 324 208
AN[3]_ DAN1-	Single-ended analog input 3 Negative terminal differential input	I	I / —	AN[3] / —	Analog	496 324 208
AN[4]_ DAN2+	Single-ended analog input 4 Positive terminal differential input	I	I / —	AN[4] / —	Analog	496 324 208
AN[5]_ DAN2-	Single-ended analog input 5 Negative terminal differential input	I	I / —	AN[5] / —	Analog	496 324 208
AN[6]_ DAN3+	Single-ended analog input 6 Positive terminal differential input	I	I / —	AN[6] / —	Analog	496 324 208

Table 18-1. eQADC External Signals (continued)

Function	Description	I/O Type	Status During Reset[1]	Status After Reset[2]	Type	Package
AN[7]_ DAN3-	Single-ended analog input 7 Negative terminal differential input	I	I / —	AN[7] / —	Analog	496 324 208
AN[8]_ ANW	Single-ended analog input 8 External multiplexed analog input W	I	I / —	AN[8] / —	Analog	496 324
AN[9]_ ANX	Single-ended analog input 9 External multiplexed analog input X	I	I / —	AN[9] / —	Analog	496 324 208
AN[10]_ ANY	Single-ended analog input 10 External multiplexed analog input Y	I	I / —	AN[10] / —	Analog	496 324
AN[11]_ ANZ	Single-ended analog input 11 External multiplexed analog input Z	I	I / —	AN[11] / —	Analog	496 324 208
AN[12]_ MA[0]_ $\overline{\text{SDS}}$	Single-ended analog input 12 Mux address 0 eQADC SSI serial data select	I O O	I / —	AN[12] / —	Analog/ Digital/ Digital	496 324 208
AN[13]_ MA[1]_ SDO	Single-ended analog input 13 Mux address 1 eQADC SSI serial data out	I O O	I / —	AN[13] / —	Analog/ Digital/ Digital	496 324 208
AN[14]_ MA[2]_ SDI	Single-ended analog input 14 Mux address 2 eQADC SSI serial data in	I O I	I / —	AN[14] / —	Analog/ Digital/ Digital	496 324 208
AN[15]_ FCK	Single-ended analog input 15 eQADC free running clock	I O	I / —	AN[15] / —	Analog/ Digital	496 324 208
AN[16]	Single-ended analog input 16	I	I / —	AN[16] / —	Analog	496 324 208
AN[17:18]	Single-ended analog input 17–18	I	I / —	AN[17:19] / —	Analog	496 324 208
AN[19:20]	Single-ended analog input 19–20	I	I / —	AN[19:20] / —	Analog	496 324
AN[21]	Single-ended analog input	I	I / —	AN[21] / —	Analog	496 324 208
AN[22:25]	Single-ended analog input	I	I / —	AN[22:25] / —	Analog	496 324 208
AN[26]	Single-ended analog input	I	I / —	AN[26] / —	Analog	496 324

Table 18-1. eQADC External Signals (continued)

Function	Description	I/O Type	Status During Reset[1]	Status After Reset[2]	Type	Package
AN[27:28]	Single-ended analog input	I	I / —	AN[27:28] / —	Analog	496 324 208
AN[29]	Single-ended analog input	I	I / —	AN[29] / —	Analog	496 324
AN[30:32]	Single-ended analog input	I	I / —	AN[30:32] / —	Analog	496 324 208
AN[33]	Single-ended analog input	I	I / —	AN[33] / —	Analog	496 324 208
AN[34:35]	Single-ended analog input	I	I / —	AN[34:35] / —	Analog	496 324 208
AN[36]	Single-ended analog input	I	I / —	AN[36] / —	Analog	496 324 208
AN[37:39]	Single-ended analog input 37–39	I	I / —	AN[37:39] / —	Analog	496 324 208
Power Supplies						
V_{RH}	Voltage reference high	I	— / —	V_{RH}	Power	496 324 208
V_{RL}	Voltage reference low	I	— / —	V_{RL}	Power	496 324 208
REFBYPC	Reference bypass capacitor input	I	— / —	REFBYPC	Power	496 324 208
V_{DDA}	Analog positive power supply	I	—	V_{DDA}	Power	496 324 208
V_{SSA}	Analog negative power supply	I	—	V_{SSA}	Power	496 324 208

1 Terminology is O — output, I — input, Up — weak pullup enabled, Down — weak pulldown enabled, Low — output driven low, High — output driven high. A dash on the left side of the slash denotes that both the input and output buffers for the pin are off. A dash on the right side of the slash denotes that there is no weak pullup/down enabled on the pin. The signal name to the left or right of the slash indicates the pin is enabled.

2 Function after reset of GPI is general-purpose input. A dash on the left side of the slash denotes that both the input and output buffers for the pin are off. A dash on the right side of the slash denotes that there is no weak pullup/down enabled on the pin.

18.3 Memory Map and Register Definition

This section provides memory maps and detailed descriptions of all registers. Data written to or read from reserved areas of the memory map is undefined.

18.3.1 eQADC Memory Map

This section provides memory maps for the eQADC.

Table 18-2. eQADC Memory Map

Address	Register Name	Register Description	Bits
Base (0xFFF8_0000)	EQADC_MCR	EQADC module configuration register	32
Base + 0x0004	—	Reserved	—
Base + 0x0008	EQADC_NMSFR	eQADC null message send format register	32
Base + 0x000C	EQADC_ETDFR	eQADC external trigger digital filter register	32
Base + 0x0010	EQADC_CFPR0	eQADC command FIFO push register 0	32
Base + 0x0014	EQADC_CFPR1	eQADC command FIFO push register 1	32
Base + 0x0018	EQADC_CFPR2	eQADC command FIFO push register 2	32
Base + 0x001C	EQADC_CFPR3	eQADC command FIFO push register 3	32
Base + 0x0020	EQADC_CFPR4	eQADC command FIFO push register 4	32
Base + 0x0024	EQADC_CFPR5	eQADC command FIFO push register 5	32
Base + 0x0028	—	Reserved	—
Base + 0x002C	—	Reserved	—
Base + 0x0030	EQADC_RFPR0	eQADC result FIFO pop register 0	32 [1]
Base + 0x0034	EQADC_RFPR1	eQADC result FIFO pop register 1	32 [1]
Base + 0x0038	EQADC_RFPR2	eQADC result FIFO pop register 2	32 [1]
Base + 0x003C	EQADC_RFPR3	eQADC result FIFO pop register 3	32 [1]
Base + 0x0040	EQADC_RFPR4	eQADC result FIFO pop register 4	32 [1]
Base + 0x0044	EQADC_RFPR5	eQADC result FIFO pop register 5	32 [1]
Base + 0x0048	—	Reserved	—
Base + 0x004C	—	Reserved	—
Base + 0x0050	EQADC_CFCR0	eQADC command FIFO control register 0	16
Base + 0x0052	EQADC_CFCR1	eQADC command FIFO control register 1	16
Base + 0x0054	EQADC_CFCR2	eQADC command FIFO control register 2	16
Base + 0x0056	EQADC_CFCR3	eQADC command FIFO control register 3	16
Base + 0x0058	EQADC_CFCR4	eQADC command FIFO control register 4	16
Base + 0x005A	EQADC_CFCR5	eQADC command FIFO control register 5	16

Table 18-2. eQADC Memory Map (continued)

Address	Register Name	Register Description	Bits
Base + 0x005C	—	Reserved	—
Base + 0x0060	EQADC_IDCR0	eQADC interrupt and eDMA control register 0	16
Base + 0x0062	EQADC_IDCR1	eQADC interrupt and eDMA control register 1	16
Base + 0x0064	EQADC_IDCR2	eQADC interrupt and eDMA control register 2	16
Base + 0x0066	EQADC_IDCR3	eQADC interrupt and eDMA control register 3	16
Base + 0x0068	EQADC_IDCR4	eQADC interrupt and eDMA control register 4	16
Base + 0x006A	EQADC_IDCR5	eQADC interrupt and eDMA control register 5	16
Base + 0x006C	—	Reserved	—
Base + 0x0070	EQADC_FISR0	eQADC FIFO and interrupt status register 0	32
Base + 0x0074	EQADC_FISR1	eQADC FIFO and interrupt status register 1	32
Base + 0x0078	EQADC_FISR2	eQADC FIFO and interrupt status register 2	32
Base + 0x007C	EQADC_FISR3	eQADC FIFO and interrupt status register 3	32
Base + 0x0080	EQADC_FISR4	eQADC FIFO and interrupt status register 4	32
Base + 0x0084	EQADC_FISR5	eQADC FIFO and interrupt status register 5	32
Base + 0x0088	—	Reserved	—
Base + 0x008C	—	Reserved	—
Base + 0x0090	EQADC_CFTCR0	eQADC command FIFO transfer counter register 0	16
Base + 0x0092	EQADC_CFTCR1	eQADC command FIFO transfer counter register 1	16
Base + 0x0094	EQADC_CFTCR2	eQADC command FIFO transfer counter register 2	16
Base + 0x0096	EQADC_CFTCR3	eQADC command FIFO transfer counter register 3	16
Base + 0x0098	EQADC_CFTCR4	eQADC command FIFO transfer counter register 4	16
Base + 0x009A	EQADC_CFTCR5	eQADC command FIFO transfer counter register 5	16
Base + 0x009C	—	Reserved	—
Base + 0x00A0	EQADC_CFSSR0	eQADC command FIFO status snapshot register 0	32
Base + 0x00A4	EQADC_CFSSR1	eQADC command FIFO status snapshot register 1	32
Base + 0x00A8	EQADC_CFSSR2	eQADC command FIFO status snapshot register 2	32
Base + 0x00AC	EQADC_CFSR	eQADC command FIFO status register	32
Base + 0x00B0	—	Reserved	—
Base + 0x00B4	EQADC_SSICR	eQADC synchronous serial interface control register	32
Base + 0x00B8	EQADC_SSIRDR	eQADC synchronous serial interface receive data register	32
Base + (0x00BC–0x00FC)	—	Reserved	—
Base + (0x0100–0x010C)	EQADC_CF0R*n*	eQADC CFIFO0 registers 0–3	32

Table 18-2. eQADC Memory Map (continued)

Address	Register Name	Register Description	Bits
Base + (0x0110–0x013C)	—	Reserved	—
Base + (0x0140–0x014C)	EQADC_CF1R*n*	eQADC CFIFO1 registers 0–3	32
Base + (0x0150–0x017C)	—	Reserved	—
Base + (0x0180–0x018C)	EQADC_CF2R*n*	eQADC CFIFO2 registers 0–3	32
Base + (0x0190–0x01BC)	—	Reserved	—
Base + (0x01C0–0x01CC)	EQADC_CF3R*n*	eQADC CFIFO3 registers 0–3	32
Base + (0x01D0–0x01FC)	—	Reserved	—
Base + (0x0200–0x020C)	EQADC_CF4R*n*	eQADC CFIFO4 registers 0–3	32
Base + (0x0210–0x023C)	—	Reserved	—
Base + (0x0240–0x024C)	EQADC_CF5R*n*	eQADC CFIFO5 registers 0–3	32
Base + (0x0250–0x02FC)	—	Reserved	—
Base + (0x0300–0x030C)	EQADC_RF0R*n*	eQADC RFIFO0 registers 0–3	32
Base + (0x0310–0x033C)	—	Reserved	—
Base + (0x0340–0x034C)	EQADC_RF1R*n*	eQADC RFIFO1 registers 0–3	32
Base + (0x0350–0x037C)	—	Reserved	—
Base + (0x0380–0x038C)	EQADC_RF2R*n*	eQADC RFIFO2 registers 0–3	32
Base + (0x0390–0x03BC)	—	Reserved	—
Base + (0x03C0–0x03CC)	EQADC_RF3R*n*	eQADC RFIFO3 registers 0–3	32
Base + (0x03D0–0x03FC)	—	Reserved	—
Base + (0x0400–0x040C)	EQADC_RF4R*n*	eQADC RFIFO4 registers 0–3	32
Base + (0x0410–0x043C)	—	Reserved	—
Base + (0x0440–0x044C)	EQADC_RF5R*n*	eQADC RFIFO5 registers 0–3	32
Base + (0x0450–0x07FC)	—	Reserved	—

[1] Result FIFOs are 16-bits wide [0:15]; bits [16:31] are filled with zeros to allow for 32-bit read access.

18.3.2 eQADC Register Descriptions

18.3.2.1 eQADC Module Configuration Register (EQADC_MCR)

The EQADC_MCR contains bits used to control how the eQADC responds to a debug mode entry request, and to enable the eQADC SSI interface.

Address: Base+ 0x0000 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	ESSIE		0	DBG	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-2. eQADC Module Configuration Register (EQADC_MCR)

Table 18-3. EQADC_MCR Field Descriptions

Field	Description
0–26	Reserved
27–28 ESSIE [0:1]	eQADC synchronous serial interface enable. Defines the eQADC synchronous serial interface operation. 00 eQADC SSI is disabled 01 Invalid value 10 eQADC SSI is enabled, FCK is free running, and serial transmissions are disabled 11 eQADC SSI is enabled, FCK is free running, and serial transmissions are enabled
29	Reserved
30–31 DBG [0:1]	Debug enable. Defines the eQADC response to a debug mode entry request. 00 Do not enter debug mode 01 Invalid value 10 Enter debug mode. If the eQADC SSI is enabled, FCK stops while the eQADC is in debug mode. 11 Enter debug mode. If the eQADC SSI is enabled, FCK is free running while the eQADC is in debug mode

NOTE

Disabling the eQADC SSI (0b00 write to ESSIE) or serial transmissions from the eQADC SSI (0b10 write to ESSIE) while a serial transmission is in progress results in the abort of that transmission.

NOTE

When disabling the eQADC SSI, the FCK does not stop until it reaches its low phase.

18.3.2.2 eQADC Null Message Send Format Register (EQADC_NMSFR)

The EQADC_NMSFR defines the format of the null message sent to the external device.

Address: Base + 0x0008 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	NMF									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NMF															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-3. eQADC Null Message Send Format Register (EQADC_NMSFR)

Table 18-4. EQADC_NMSFR Field Descriptions

Field	Description
0–5	Reserved
6–31 NMF [0:25]	Null message format. Contains the programmable null message send value for the eQADC. The value written to this register is sent as a null message when serial transmissions from the eQADC SSI are enabled (ESSIE field is configured to 0b11 in EQADC_MCR (Section 18.3.2.1, “eQADC Module Configuration Register (EQADC_MCR)) and either • there are no triggered CFIFOs with commands bound for external command buffers, or; • there are triggered CFIFOs with commands bound for external command buffers but the external command buffers are full. See Section , “Null Message Format for External Device Operation” for more information on the format of a null message.

NOTE

The eQADC null message send format register (eQADC_NMSFR) only defines the format of the null message that eQADC sends. The null message register does not control how the eQADC detects a null message from the input source. The eQADC detects a null message by decoding the MESSAGE_TAG field on the receive data. See Table 18-31 for more information on the MESSAGE_TAG field.

NOTE

Writing to the eQADC null message send format register while serial transmissions are enabled is not recommended. See EQADC_MCR[ESSIE] field in Section 18.3.2.1, “eQADC Module Configuration Register (EQADC_MCR).”

18.3.2.3 eQADC External Trigger Digital Filter Register (EQADC_ETDFR)

NOTE

This register is not available on the MPC5561 device due to pin limitations on the 324 package. Do not read or write to this register.

The EQADC_ETDFR is used to set the minimum time a signal must be held in a logic state on the CFIFO triggers inputs to be recognized as an edge or level gated trigger. The digital filter length field specifies the minimum number of system clocks that must be counted by the digital filter counter to recognize a logic state change.

Address: Base + 0x000C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DFL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-4. eQADC External Trigger Digital Filter Register (EQADC_ETDFR)

18.3.2.4 eQADC CFIFO Push Registers 0–5 (EQADC_CFPR*n*)

The EQADC_CFPRs provide a mechanism to fill the CFIFOs with command messages from the command queues. See Section 18.4.3, “eQADC Command FIFOs,” for more information on the CFIFOs and to Section 18.4.1.2, “Message Format in eQADC,” for a description on command message formats.

Address: Base + 0x0010 (EQADC_CFPR0) Access: WO
Base + 0x0014 (EQADC_CFPR1);
Base + 0x0018 (EQADC_CFPR2)
Base + 0x001C (EQADC_CFPR3)
Base + 0x0020 (EQADC_CFPR4)
Base + 0x0024 (EQADC_CFPR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	CF_PUSH*n*															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	CF_PUSH*n*															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-5. eQADC CFIFO Push Registers 0–5 (EQADC_CFPR*n*)

Table 18-5. EQADC_CFPR*n* Field Descriptions

Field	Description
0–31 CF_PUSH*n* [0:31]	CFIFO push data *n*. When CFIFO*n* is not full, writing to the whole word or any bytes of EQADC_CFPR*n* pushes the 32-bit CF_PUSH*n* value into CFIFO*n*. Writing to the CF_PUSH*n* field also increments the corresponding CFCTR*n* value by one in Section 18.3.2.8, "eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)." When the CFIFO*n* is full, the eQADC ignores any write to the CF_PUSH*n*. Reading the EQADC_CFPR*n* always returns 0. **Note:** Write only whole words to the EQADC_CFPR*n* registers. Writing halfwords or bytes to EQADC_CFPR pushes the entire 32-bit CF_PUSH field into the CFIFO, but undefined data fills the areas of CF_PUSH that were not specifically designated as target locations for the write.

18.3.2.5 eQADC Result FIFO Pop Registers 0–5 (EQADC_RFPR*n*)

The eQADC_RFPRs allow you to retrieve data from RFIFOs.

NOTE

Do not read the EQADC_RFPR*n* unless absolutely necessary, since the data is lost when the read occurs. For compatibility, configure the TLB entry for the EQADC_RFPR*n* registers as guarded.

Address: Base + 0x0030 (EQADC_RFPR0) Access: RO
Base + 0x0034 (EQADC_RFPR1)
Base + 0x0038 (EQADC_RFPR2)
Base + 0x003C (EQADC_RFPR3)
Base + 0x0040 (EQADC_RFPR4)
Base + 0x0044 (EQADC_RFPR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RF_POP*n*															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-6. eQADC RFIFO Pop Registers 0–5 (EQADC_RFPR*n*)

Table 18-6. EQADC_RFPR*n* Field Descriptions

Field	Description
0–15	Reserved
16–31 RF_POP*n* [0:15]	Result FIFO pop data *n*. When RFIFO*n* is not empty, the RF_POP*n* contains the next unread entry value of RFIFO*n*. Reading the whole word, a halfword, or any bytes of EQADC_RFPR*n* pops one entry from RFIFO*n*, and the RFCTR*n* value is decremented by 1. See Section 18.3.2.8, "eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)." When the RFIFO*n* is empty, any read on EQADC_RFPR*n* returns undefined data value and does not decrement the RFCTR*n* value. Writing to EQADC_RFPR*n* has no effect.

18.3.2.6 eQADC CFIFO Control Registers 0–5 (EQADC_CFCR*n*)

The eQADC_CFCRs contain bits that affect CFIFOs. These bits specify the CFIFO operation mode and can invalidate all of the CFIFO contents.

Address: EQADC_BASE + 0x0050 (EQADC_CFCR0) Access: R/W
EQADC_BASE + 0x0052 (EQADC_CFCR1)
EQADC_BASE + 0x0054 (EQADC_CFCR2)
EQADC_BASE + 0x0056 (EQADC_CFCR3)
EQADC_BASE + 0x0058 (EQADC_CFCR4);
EQADC_BASE + 0x005A (EQADC_CFCR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MODE*n*				0	0	0	0
W						SSE*n*	CFINV*n*									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-7. eQADC CFIFO Control Registers (EQADC_CFCR*n*)

Table 18-7. EQADC_CFCR*n* Field Descriptions

Field	Description
0–4	Reserved
5 SSE*n*	CFIFO single-scan enable bit *n*. Used to set the SSS*n* bit, as described in Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn).” Writing a 1 to SSE*n* sets the SSS*n* if the CFIFO is in single-scan mode. When SSS*n* is already asserted, writing a 1 to SSE*n* has no effect. If the CFIFO is in continuous-scan mode or is disabled, writing a 1 to SSE*n* does not set SSS*n*. Writing a 0 to SSE*n* has no effect. SSE*n* always is read as 0. 0 No effect. 1 Set the SSS*n* bit.
6 CFINV*n*	CFIFO invalidate bit *n*. Causes the eQADC to invalidate all entries of CFIFO*n*. Writing a 1 to CFINV*n* resets the value of CFCTR*n* in the EQADC_FISR register (see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn).” Writing a 1 to CFINV*n* also resets the push next data pointer, transfer next data pointer to the first entry of CFIFO*n* in Figure 18-35. Reading CFINV*n* always returns a 0. Writing a 0 has no effect. 0 No effect. 1 Invalidate all of the entries in the corresponding CFIFO. **Note:** Writing CFINV*n* only invalidates commands stored in CFIFO*n*; previously transferred commands that are waiting for execution (commands stored in the ADC command buffers) are executed, and the results are stored in the RFIFO. **Note:** Do not write to CFINV*n* unless MODE*n* is disabled, and CFIFO status is IDLE.
7	Reserved
8–11 MODE*n* [0:3]	CFIFO operation mode *n*. Selects the CFIFO operation mode for CFIFO*n*. See Section 18.4.3.4, “CFIFO Scan Trigger Modes,” for more information on CFIFO trigger mode. **Note:** If MODE*n* is not disabled, it must not be changed to any other mode besides disabled. If MODE*n* is disabled and the CFIFO status is IDLE, MODE*n* can be changed to any other mode.
12–15	Reserved

Table 18-8. CFIFO Operation Mode Table

MODE*n*[0:3]	CFIFO Operation Mode
0b0000	Disabled
0b0001	Software trigger, single scan
0b0010	Low level gated external trigger, single scan
0b0011	High level gated external trigger, single scan
0b0100	Falling edge external trigger, single scan
0b0101	Rising edge external trigger, single scan
0b0110	Falling or rising edge external trigger, single scan
0b0111–0b1000	Reserved
0b1001	Software trigger, continuous scan
0b1010	Low level gated external trigger, continuous scan
0b1011	High level gated external trigger, continuous scan
0b1100	Falling edge external trigger, continuous scan
0b1101	Rising edge external trigger, continuous scan
0b1110	Falling or rising edge external trigger, continuous scan
0b1111	Reserved

18.3.2.7 eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCR*n*)

The eQADC_IDCRs contain bits to enable the generation of interrupt or eDMA requests when the corresponding flag bits are set in EQADC_FISRn. See Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn).”

Address: EQADC_BASE + 0x0060 (EQADC_IDCR0) Access: R/W
EQADC_BASE + 0x0062 (EQADC_IDCR1)
EQADC_BASE + 0x0064 (EQADC_IDCR2)
EQADC_BASE + 0x0066 (EQADC_IDCR3)
EQADC_BASE + 0x0068 (EQADC_IDCR4)
EQADC_BASE + 0x006A (EQADC_IDCR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCI E*n*	TORI E*n*	PIE*n*	EOQI E*n*	CFUI E*n*	0	CFF E*n*	CFF S*n*	0	0	0	0	RFOI E*n*	0	RFD E*n*	RFD S*n*
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-8. eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCR*n*)

Table 18-9. EQADC_IDCR*n* Field Descriptions

Field	Description
0 NCIE*n*	Non-coherency interrupt enable *n*. Enables the eQADC to generate an interrupt request when the corresponding NCF*n*, described in Section 18.3.2.8, "eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)," is asserted. 0 Disable non-coherency interrupt request 1 Enable non-coherency interrupt request
1 TORIE*n*	Trigger overrun interrupt enable *n*. Enables the eQADC to generate an interrupt request when the corresponding TORF*n* (described in Section 18.3.2.8, "eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)") is asserted. Apart from generating an independent interrupt request for a CFIFO*n* trigger overrun event, the eQADC also provides a combined interrupt at which the result FIFO overflow interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun interrupt requests of all CFIFOs are ORed. When RFOIE*n*, CFUIE*n*, and TORIE*n* are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOF*n*, CFUF*n*, and TORF*n* (assuming that all interrupts are enabled). See Section 18.4.7, "eQADC eDMA or Interrupt Request," for details. 0 Disable trigger overrun interrupt request 1 Enable trigger overrun interrupt request
2 PIE*n*	Pause interrupt enable *n*. Enables the eQADC to generate an interrupt request when the corresponding PFx in EQADC_FISRn is asserted. See Section 18.3.2.8, "eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)." 0 Disable pause interrupt request 1 Enable pause interrupt request
3 EOQIE*n*	End-of-queue interrupt enable *n*. Enables the eQADC to generate an interrupt request when the corresponding EOQF*n* in EQADC_FISRn is asserted. See Section 18.3.2.8, "eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)." 0 Disable end of queue interrupt request. 1 Enable end of queue interrupt request.
4 CFUIE*n*	CFIFO underflow interrupt enable *n*. Enables the eQADC to generate an interrupt request when the corresponding CFUF*n* in EQADC_FISRn is asserted. Apart from generating an independent interrupt request for a CFIFO*n* underflow event, the eQADC also provides a combined interrupt at which the result FIFO overflow interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun interrupt requests of all CFIFOs are ORed. When RFOIE*n*, CFUIE*n*, and TORIE*n* are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOF*n*, CFUF*n*, and TORF*n* (assuming that all interrupts are enabled). See Section 18.4.7, "eQADC eDMA or Interrupt Request," for details. 0 Disable underflow interrupt request 1 Enable underflow interrupt request
5	Reserved
6 CFFE*n*	CFIFO fill enable *n*. Enables the eQADC to generate an interrupt request (CFFS*n* is asserted) or eDMA request (CFFS*n* is negated) when CFFF*n* in EQADC_FISRn is asserted. See Section 18.3.2.8, "eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)." 0 Disable CFIFO fill eDMA or interrupt request 1 Enable CFIFO fill eDMA or interrupt request **Note:** CFFE*n* must not be negated while an eDMA transaction is in progress.
7 CFFS*n*	CFIFO fill select *n*. Selects if an eDMA or interrupt request is generated when CFFF*n* in EQADC_FISRn (See Section 18.3.2.8, "eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)") is asserted. If CFFE*n* is asserted, the eQADC generates an interrupt request when CFFS*n* is negated, or it generates an eDMA request if CFFS*n* is asserted. 0 Generate interrupt request to move data from the system memory to CFIFO*n*. 1 Generate eDMA request to move data from the system memory to CFIFO*n*. **Note:** CFFS*n* must not be negated while an eDMA transaction is in progress.

Table 18-9. EQADC_IDCR*n* Field Descriptions (continued)

Field	Description
8–11	Reserved
12 RFOIE*n*	RFIFO overflow interrupt enable *n*. Enables the eQADC to generate an interrupt request when the corresponding RFOF*n* in EQADC_FISRn is asserted. See Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn).” Apart from generating an independent interrupt request for an RFIFO*n* overflow event, the eQADC also provides a combined interrupt at which the result FIFO overflow Interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun interrupt requests of all CFIFOs are ORed. When RFOIE*n*, CFUIE*n,* and TORIE*n* are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOF*n,* CFUF*n*, and TORF*n* (assuming that all interrupts are enabled). See Section 18.4.7, “eQADC eDMA or Interrupt Request,” for details. 0 Disable overflow interrupt request 1 Enable overflow Interrupt request
13	Reserved
14 RFDE*n*	RFIFO drain enable *n*. Enables the eQADC to generate an interrupt request (RFDS*n* is asserted) or eDMA request (RFDS*n* is negated) when RFDF*n* in EQADC_FISRn (See Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) is asserted. 0 Disable RFIFO drain eDMA or interrupt request 1 Enable RFIFO drain eDMA or interrupt request **Note:** RFDE*n* must not be negated while an eDMA transaction is in progress.
15 RFDS*n*	RFIFO drain select *n*. Selects if an eDMA or interrupt request is generated when RFDF*n* in EQADC_FISRn (See Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) is asserted. If RFDE*n* is asserted, the eQADC generates an interrupt request when RFDS*n* is negated, or it generates an eDMA request when RFDS*n* is asserted. 0 Generate interrupt request to move data from RFIF*n* to the system memory 1 Generate eDMA request to move data from RFIFO*n* to the system memory **Note:** RFDS*n* must not be negated while an eDMA transaction is in progress.

18.3.2.8 eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR*n*)

The EQADC_FISRs contain flag and status bits for each CFIFO and RFIFO pair. Writing 1 to a flag bit clears it. Writing 0 has no effect. Status bits are read only. These bits indicate the status of the FIFO itself.

Address: Base + 0x0070 (EQADC_FISR0) Access: R/W1c
Base + 0x0074 (EQADC_FISR1)
Base + 0x0078 (EQADC_FISR2)
Base + 0x007C (EQADC_FISR3)
Base + 0x0080 (EQADC_FISR4)
Base + 0x0084 (EQADC_FISR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCF*n*	TORF*n*	PF*n*	EOQF*n*	CFUF*n*	SSS*n*	CFFF*n*	0	0	0	0	0	RFOF*n*	0	RFDF*n*	0
W	w1c	w1c	w1c	w1c	w1c		w1c						w1c		w1c	
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFCTR*n*				TNXTPTR*n*				RFCTR*n*				POPNXTPTR*n*			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-9. eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR*n*)

Table 18-10. EQADC_FISR*n* Field Descriptions

Field	Description
0 NCF*n*	Non-coherency flag *n*. NCF*n* is set whenever a command sequence being transferred through CFIFO*n* becomes non-coherent. If NCIE*n* in EQADC_IDCRn (See Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and NCF*n* are asserted, an interrupt request is generated. Writing a 1 clears NCF*n*. Writing a 0 has no effect. More for information on non-coherency see Section 18.4.3.5.5, “Command Sequence Non-Coherency Detection.” 0 Command sequence being transferred by CFIFO*n* is coherent 1 Command sequence being transferred by CFIFO*n* became non-coherent **Note:** Non-coherency means that a command in the command FIFO was not immediately executed, but delayed. This can occur if the command is pre-empted, where a higher priority queue is triggered and has a competing conversion command for the same converter.
1 TORF*n*	Trigger overrun flag for CFIFO *n*. TORF*n* is set when trigger overrun occurs for the specified CFIFO in edge or level trigger mode. Trigger overrun occurs when an already triggered CFIFO receives an additional trigger. When EQADC_IDCRn[TORIE*n*] is set (See Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and TORF*n* are asserted, an interrupt request is generated. Apart from generating an independent interrupt request for a CFIFO*n* trigger overrun event, the eQADC also provides a combined interrupt at which the result FIFO overflow interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun Interrupt requests of all CFIFOs are ORed. When RFOIE*n*, CFUIE*n*, and TORIE*n* are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOF*n*, CFUF*n*, and TORF*n* (assuming that all interrupts are enabled). See Section 18.4.7, “eQADC eDMA or Interrupt Request,” for details. Write 1 to clear the TORF*n* bit. Writing 0 has no effect. 0 No trigger overrun occurred 1 Trigger overrun occurred **Note:** The trigger overrun flag is not set for CFIFOs configured for software trigger mode.

Table 18-10. EQADC_FISR*n* Field Descriptions (continued)

Field	Description
2 PF*n*	Pause flag *n*. PF behavior changes according to the CFIFO trigger mode. • In edge trigger mode, PF*n* is set when the eQADC completes the transfer of an entry with an asserted pause bit from CFIFO*n*. • In level trigger mode, when CFIFO*n* is in the TRIGGERED state, PF*n* is set when CFIFO status changes from TRIGGERED due to the detection of a closed gate. An interrupt routine, generated due to the asserted PF, can be used to verify if a complete scan of the command queue you defined was performed. If a closed gate is detected while no command transfers are taking place, it has an immediate effect on the CFIFO status. If a closed gate is detected while a command transfer to an on-chip ADC is taking place, it only affects the CFIFO status when the transfer completes. If a closed gate is detected during the serial transmission of a command to the external device, it has no effect on the CFIFO status until the transmission completes. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate ADC command buffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed. In software trigger mode, PF*n* is never asserted. If PIE*n* (See Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and PF*n* are asserted, an interrupt is generated. Writing a 1 clears the PF*n*. Writing a 0 has no effect. See Section 18.4.3.5.3, “Pause Status,” for more information on pause flag. 0 Entry with asserted pause bit was not transferred from CFIFO*n* (CFIFO in edge trigger mode), or CFIFO status did not change from the TRIGGERED state due to detection of a closed gate (CFIFO in level trigger mode). 1 Entry with asserted pause bit was transferred from CFIFO*n* (CFIFO in edge trigger mode), or CFIFO status changes from the TRIGGERED state due to detection of a closed gate (CFIFO in level trigger mode). **Note:** In edge trigger mode, an asserted PF*n* only implies that the eQADC has finished transferring a command with an asserted pause bit from CFIFO*n*. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO. **Note:** In software or level trigger mode, when the eQADC completes the transfer of an entry from CFIFO*n* with an asserted pause bit, PF*n* is not set and commands continue to transfer without pausing.
3 EOQF*n*	End-of-queue flag *n*. Indicates that an entry with an asserted EOQ bit was transferred from CFIFO*n* to the on-chip ADCs or to the external device. See Section 18.4.1.2, “Message Format in eQADC,” for details about command message formats. When the eQADC completes the transfer of an entry with an asserted EOQ bit from CFIFO*n*, EOQF*n* is set. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate command buffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed. If the EOQIE*n* bit (See Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and EOQF*n* are asserted, an interrupt is generated. Writing a 1 clears the EOQF*n* bit. Writing a 0 has no effect. See Section 18.4.3.5.2, “Command Queue Completion Status,” for more information on end-of-queue flag. 0 Entry with asserted EOQ bit was not transferred from CFIFO*n* 1 Entry with asserted EOQ bit was transferred from CFIFO*n* **Note:** An asserted EOQF*n* only implies that the eQADC has finished transferring a command with an asserted EOQ bit from CFIFO*n*. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

Table 18-10. EQADC_FISR*n* Field Descriptions (continued)

Field	Description
4 CFUF*n*	CFIFO underflow flag *n*. Indicates an underflow event on CFIFO*n*. CFUF*n* is set when CFIFO*n* is in the TRIGGERED state and it becomes empty. No commands are transferred from an underflowing CFIFO, and command transfers from lower priority CFIFOs are not blocked. When CFUIE*n* (see Section Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and CFUF*n* are both asserted, the eQADC generates an interrupt request. Apart from generating an independent interrupt request for a CFIFO*n* underflow event, the eQADC also provides a combined interrupt at which the result FIFO overflow interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun interrupt requests of all CFIFOs are ORed. When RFOIE*n*, CFUIE*n*, and TORIE*n* are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOF*n*, CFUF*n*, and TORF*n* (assuming that all interrupts are enabled). See Section 18.4.7, “eQADC eDMA or Interrupt Request,” for details. Writing a 1 clears CFUF*n*. Writing a 0 has no effect. 0 No CFIFO underflow event occurred 1 A CFIFO underflow event occurred
5 SSS*n*	CFIFO single-scan status bit *n*. When asserted, enables the detection of trigger events for CFIFOs programmed into single-scan level- or edge-trigger mode, and works as trigger for CFIFOs programmed into single-scan software-trigger mode. See Section 18.4.3.4.2, “Single-Scan Mode,” for further details. The SSS*n* bit is set by writing a 1 to the SSE*n* bit (see Section Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”). The eQADC clears the SSS*n* bit when a command with an asserted EOQ bit is transferred from a CFIFO in single-scan mode, when a CFIFO is in single-scan level trigger mode and its status changes from the TRIGGERED state due to the detection of a closed gate, or when the value of the CFIFO operation mode MODE*n* (see Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”) is changed to disabled. Writing to SSS*n* has no effect. SSS*n* has no effect in continuous-scan or in disabled mode. 0 CFIFO in single-scan, level-, or edge-trigger mode ignores trigger events, or CFIFO in single-scan software-trigger mode is not triggered. 1 CFIFO in single-scan level- or edge-trigger mode detects a trigger event, or CFIFO in single-scan software-trigger mode is triggered.
6 CFFF*n*	CFIFO fill flag *n*. CFFF*n* is set when the CFIFO*n* is not full. When CFFE*n* (see Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and CFFF*n* are both asserted, an interrupt or an eDMA request is generated depending on the status of the CFFS*n* bit. When CFFS*n* is negated (interrupt requests selected), software clears CFFF*n* by writing a 1 to it. Writing a 0 has no effect. When CFFS*n* is asserted (eDMA requests selected), CFFF*n* is automatically cleared by the eQADC when the CFIFO becomes full. 0 CFIFO*n* is full. 1 CFIFO*n* is not full. **Note:** When generation of interrupt requests is selected (CFFS*n*=0), CFFF*n* must only be cleared in the ISR after the CFIFO*n* push register is accessed. **Note:** Do not clear CFFF*n* when CFFSn is asserted (eDMA requests selected).
7–11	Reserved

Table 18-10. EQADC_FISR*n* Field Descriptions (continued)

Field	Description
12 RFOF*n*	RFIFO overflow flag *n*. Indicates an overflow event on RFIFO*n*. RFOF*n* is set when RFIFO*n* is already full, and a new data is received from the on-chip ADCs or from the external device. The RFIFO*n* does not overwrite older data in the RFIFO, and the new data is ignored. When RFOIE*n* (see Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and RFOF*n* are both asserted, the eQADC generates an interrupt request. Apart from generating an independent interrupt request for an RFIFO*n* overflow event, the eQADC also provides a combined interrupt at which the result FIFO overflow interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun interrupt requests of all CFIFOs are ORed. When RFOIE*n*, CFUIE*n*, and TORIE*n* are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOF*n*, CFUF*n*, and TORF*n* (assuming that all interrupts are enabled). See Section 18.4.7, “eQADC eDMA or Interrupt Request,” for details. Write 1 to clear RFOF*n*. Writing a 0 has no effect. 0 No RFIFO overflow event occurred. 1 An RFIFO overflow event occurred.
13	Reserved
14 RFDF*n*	RFIFO drain flag *n*. Indicates if RFIFO*n* has valid entries that can be drained or not. RFDF*n* is set when the RFIFO*n* has at least one valid entry in it. When RFDE*n* (see Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and RFDF*n* are both asserted, an interrupt or an eDMA request is generated depending on the status of the RFDS*n* bit. When RFDS*n* is negated (interrupt requests selected), software clears RFDF*n* by writing a 1 to it. Writing a 0 has no effect. When RFDS*n* is asserted (eDMA requests selected), RFDF*n* is automatically cleared by the eQADC when the RFIFO becomes empty. 0 RFIFO*n* is empty. 1 RFIFO*n* has at least one valid entry. **Note:** In the interrupt service routine, RFDF must be cleared only after the RFIFO*n* pop register is read. **Note:** Do not clear RFDF*n* when RFDS*n* is asserted (eDMA requests selected).
15	Reserved
16–19 CFCTR*n* [0:3]	CFIFO*n* entry counter. Indicates the number of commands stored in the CFIFO*n*. When the eQADC completes transferring a piece of new data from the CFIFO*n*, it decrements CFCTR*n* by 1. Writing a word or any bytes to the corresponding CFIFO Push Register (see Section 18.3.2.4, “eQADC CFIFO Push Registers 0–5 (EQADC_CFPRn)”) increments CFCTR*n* by 1. Writing any value to CFCTR*n* has no effect.
20–23 TNX TPTR*n* [0:3]	CFIFO*n* transfer next pointer. Indicates the index of the next entry to be removed from CFIFO*n* when it completes a transfer. When TNXTPTR*n* is 0, it points to the entry with the smallest memory-mapped address inside CFIFO*n*. TNXTPTR*n* is only updated when a command transfer is completed. If the maximum index number (CFIFO depth minus 1) is reached, TNXTPTR*n* is wrapped to 0, else, it is incremented by 1. For details, see Section 18.4.3.1, “CFIFO Basic Functionality.” Writing any value to TNXTPTR*n* has no effect.
24–27 RFCTR*n* [0:3]	RFIFO*n* entry counter. Indicates the number of data items stored in the RFIFO*n*. When the eQADC stores a piece of new data into RFIFO*n*, it increments RFCTR*n* by 1. Reading the whole word, halfword or any bytes of the corresponding Result FIFO pop register (see Section 18.3.2.5, “eQADC Result FIFO Pop Registers 0–5 (EQADC_RFPRn)”) decrements RFCTR*n* by 1. Writing any value to RFCTR*n* itself has no effect.
28–31 POPNX TPTR*n* [0:3]	RFIFO*n* pop next pointer. Indicates the index of the entry that is returned when EQADC_RFPR*n* is read. When POPNXTPTR*n* is 0, it points to the entry with the smallest memory-mapped address inside RFIFO*n*. POPNXTPTR*n* is updated when EQADC_RFPR*n* is read. If the maximum index number (RFIFO depth minus 1) is reached, POPNXTPTR*n* is wrapped to 0, else, it is incremented by 1. For details see Section 18.4.4.1, “RFIFO Basic Functionality.” Writing any value to POPNXTPTR*n* has no effect.

18.3.2.9 eQADC CFIFO Transfer Counter Registers 0–5 (EQADC_CFTCR*n*)

The EQADC_CFTCRs record the number of commands transferred from a CFIFO. The EQADC_CFTCR supports the monitoring of command transfers from a CFIFO.

Address: EQADC_BASE + 0x0090 (EQADC_CFTCR0) Access: R/W
EQADC_BASE + 0x0092 (EQADC_CFTCR1)
EQADC_BASE + 0x0094 (EQADC_CFTCR2)
EQADC_BASE + 0x0096 (EQADC_CFTCR3)
EQADC_BASE + 0x0098 (EQADC_CFTCR4)
EQADC_BASE + 0x009A (EQADC_CFTCR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	TC_CF*n*										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-10. eQADC CFIFO Transfer Counter Registers (EQADC_CFTCR*n*)

Table 18-11. EQADC_CFTCR*n* Field Descriptions

Field	Description
0–4	Reserved
5–15 TC_CF*n* [0:10]	Transfer counter for CFIFO*n*. TC_CF*n* counts the number of commands that have been completely transferred from CFIFO*n*. TC_CF*n*=2, for example, signifies that two commands have been transferred. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate command buffer. The transfer of entries bound for an external device is considered completed when the serial transmission of the entry is completed. The eQADC increments the TC_CF*n* value by 1 after a command is transferred. TC_CF*n* resets to 0 after eQADC completes transferring a command with an asserted EOQ bit. Writing any value to TC_CF*n* sets the counter to that written value. **Note:** If CFIFO*n* is in the TRIGGERED state when its MODE*n* field is programmed to disabled, the exact number of entries transferred from the CFIFO until that point (TC_CF*n*) is only known after the CFIFO status changes to IDLE, as indicated by CFS*n*. For details see Section 18.4.3.4.1, “Disabled Mode.”

18.3.2.10 eQADC CFIFO Status Snapshot Registers 0–2

The eQADC_CFSSRs contain status fields to track the operation status of each CFIFO and the transfer counter of the last CFIFO to initiate a command transfer to the internal ADCs and the external command buffers. EQADC_CFSSR0–1 are related to the on-chip ADC command buffers (buffers 0 and 1) while EQADC_CFSSR2 is related to the external command buffers (buffers 2 and 3). All fields of a particular EQADC_CFSSR are captured at the beginning of a command transfer to the buffer associated with that register.

Captured status register values are associated with a previous command transfer. This means that the eQADC_CFSSR registers capture the status registers before the status registers change, because of the transfer of the current command that is about to be popped from the CFIFO. The EQADC_CFSSRs are read only. Writing to the EQADC_CFSSRs has no effect.

18.3.2.10.1 eQADC CFIFO Status Snapshot Registers 0 EQADC_CFSSR0

The first eQADC CFIFO status snapshot register is displayed in Figure 18-11.

Address: Base + 0x00A0 Access: RO

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS0_T0		CFS1_T0		CFS2_T0		CFS3_T0		CFS4_T0		CFS5_T0		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	LCFT0				TC_LCFT0										
W																
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Figure 18-11. eQADC CFIFO Status Snapshot Register 0 (EQADC_CFSSR0)

The first eQADC CFIFO status snapshot register is described in Table 18-12.

Table 18-12. EQADC_CFSSR0 Field Descriptions

Field	Description
0–11 CFS*n*_T0 [0:1]	CFIFO status at transfer to ADC*n* command buffer. Indicates the CFIFO*n* status at the time a command transfer to ADC*n* command buffer is initiated. CFS*n*_T0 is a copy of the corresponding CFS*n* in EQADC_CFSR (see Section 18.3.2.11, “eQADC CFIFO Status Register EQADC_CFSR”) captured at the time a command transfer to buffer*n* is initiated.
12–16	Reserved
17–20 LCFT0 [0:3]	Last CFIFO to transfer to ADC*n* command buffer. Holds the CFIFO number of last CFIFO to have initiated a command transfer to ADC*n* command buffer. LCFT0 has the following values: **LCFT0[0:3]** / **LCFT0 Meaning** 0b0000 – Last command was transferred from CFIFO0 0b0001 – Last command was transferred from CFIFO1 0b0010 – Last command was transferred from CFIFO2 0b0011 – Last command was transferred from CFIFO3 0b0100 – Last command was transferred from CFIFO4 0b0101 – Last command was transferred from CFIFO5 0b0110–0b1110 – Reserved 0b1111 – No command was transferred to ADC*n* command buffer
21–31 TC_LCFT0 [0:10]	Transfer counter for last CFIFO to transfer commands to ADC*n* command buffer. Indicates the number of commands which have been completely transferred from CFIFO*n* when a command transfer from CFIFO*n* to ADC*n* command buffer is initiated. TC_LCFT0 is a copy of the corresponding TC_CF*n* in EQADC_CFTCR*n* (see Section 18.3.2.9) captured at the time a command transfer from CFIFO*n* to ADC*n* command buffer is initiated. This field has no meaning when LCFT0 is 0b1111.

18.3.2.10.2 eQADC CFIFO Status Snapshot Registers 1 EQADC_CFSSR1

The second eQADC CFIFO status snapshot register is displayed in Figure 18-12.

Address: Base + 0x00A4 Access: RO

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS0_T1		CFS1_T1		CFS2_T1		CFS3_T1		CFS4_T1		CFS5_T1		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	LCFT1				TC_LCFT1										
W																
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Figure 18-12. eQADC CFIFO Status Snapshot Register 1 (EQADC_CFSSR1)

The second eQADC CFIFO status snapshot register is described in Table 18-13.

Table 18-13. EQADC_CFSSR1 Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0–11
CFSn_T1
[0:1]</td><td>CFIFO status at transfer to ADCn command buffer. Indicates the CFIFOn status at the time a command transfer to ADCn command buffer is initiated. CFSn_T1 is a copy of the corresponding CFSn in EQADC_CFSR (see Section 18.3.2.11, “eQADC CFIFO Status Register EQADC_CFSR”) captured at the time a command transfer to buffern is initiated.</td></tr>
<tr><td>12–16</td><td>Reserved</td></tr>
<tr><td>17–20
LCFT1
[0:3]</td><td>Last CFIFO to transfer to ADCn command buffer. Holds the CFIFO number of last CFIFO to have initiated a command transfer to ADCn command buffer. LCFT1 has the following values:
<table>
<tr><th>LCFT1[0:3]</th><th>LCFT1 Meaning</th></tr>
<tr><td>0b0000</td><td>Last command was transferred from CFIFO0</td></tr>
<tr><td>0b0001</td><td>Last command was transferred from CFIFO1</td></tr>
<tr><td>0b0010</td><td>Last command was transferred from CFIFO2</td></tr>
<tr><td>0b0011</td><td>Last command was transferred from CFIFO3</td></tr>
<tr><td>0b0100</td><td>Last command was transferred from CFIFO4</td></tr>
<tr><td>0b0101</td><td>Last command was transferred from CFIFO5</td></tr>
<tr><td>0b0110–0b1110</td><td>Reserved</td></tr>
<tr><td>0b1111</td><td>No command was transferred to ADCn command buffer</td></tr>
</table></td></tr>
<tr><td>21–31
TC_LCFT1
[0:10]</td><td>Transfer counter for last CFIFO to transfer commands to ADCn command buffer. Indicates the number of commands which have been completely transferred from CFIFOn when a command transfer from CFIFOn to ADCn command buffer is initiated. TC_LCFT1 is a copy of the corresponding TC_CFn in EQADC_CFTCRn (see Section 18.3.2.9, “eQADC CFIFO Transfer Counter Registers 0–5 (EQADC_CFTCRn)”) captured at the time a command transfer from CFIFOn to ADCn command buffer is initiated. This field has no meaning when LCFT1 is 0b1111.</td></tr>
</table>

18.3.2.10.3 eQADC CFIFO Status Snapshot Registers 2 EQADC_CFSSR2

The third eQADC CFIFO status snapshot register is displayed in Figure 18-13.

Address: Base + 0x00A8 Access: RO

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS0_TSSI		CFS1_TSSI		CFS2_TSSI		CFS3_TSSI		CFS4_TSSI		CFS5_TSSI		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ENI	LCFTSSI				TC_LCFTSSI										
W																
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Figure 18-13. eQADC CFIFO Status Snapshot Register 2 (EQADC_CFSSR2)

The third eQADC CFIFO status snapshot register is described in Table 18-14.

Table 18-14. EQADC_CFSSR2 Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0–11
CFSn_TSSI
[0:1]</td><td>CFIFO Status at Transfer through the eQADC SSI. Indicates the CFIFOn status at the time a serial transmission through the eQADC SSI is initiated. CFSn_TSSI is a copy of the corresponding CFSn in EQADC_CFSR (see Section 18.3.2.11, “eQADC CFIFO Status Register EQADC_CFSR”) captured at the time a serial transmission through the eQADC SSI is initiated.</td></tr>
<tr><td>12–15</td><td>Reserved</td></tr>
<tr><td>16
ENI</td><td>External command buffer Number Indicator. Defines the external command buffer the last command was sent.
0 Last command was transferred to command buffer 2.
1 Last command was transferred to command buffer 3.</td></tr>
<tr><td>17–20
LCFTSSI
[0:3]</td><td>Last CFIFO to transfer commands through the eQADC SSI. Holds the CFIFO number of last CFIFO to have initiated a command transfer to an external command buffer through the eQADC SSI. LCFTSSI does not indicate the transmission of null messages. LCFTSSI has the following values:
<table>
<tr><th>LCFTSSI[0:3]</th><th>LCFTSSI Meaning</th></tr>
<tr><td>0b0000</td><td>Last command was transferred from CFIFO0</td></tr>
<tr><td>0b0001</td><td>Last command was transferred from CFIFO1</td></tr>
<tr><td>0b0010</td><td>Last command was transferred from CFIFO2</td></tr>
<tr><td>0b0011</td><td>Last command was transferred from CFIFO3</td></tr>
<tr><td>0b0100</td><td>Last command was transferred from CFIFO4</td></tr>
<tr><td>0b0101</td><td>Last command was transferred from CFIFO5</td></tr>
<tr><td>0b0110–0b1110</td><td>Reserved</td></tr>
<tr><td>0b1111</td><td>No command was transferred to an external command buffer</td></tr>
</table>
</td></tr>
</table>

Table 18-14. EQADC_CFSSR2 Field Descriptions (continued)

Field	Description
21–31 TC_LCFTSSI [0:10]	Transfer counter for last CFIFO to transfer commands through eQADC SSI. Indicates the number of commands which have been completely transferred from a particular CFIFO at the time a command transfer from that CFIFO to an external command buffer is initiated. TC_LCFTSSI is a copy of the corresponding TC_CF*n* in EQADC_CFTCRn (see Section 18.3.2.9, "eQADC CFIFO Transfer Counter Registers 0–5 (EQADC_CFTCRn)") captured at the time a command transfer to an external command buffer is initiated. This field has no meaning when LCFTSSI is 0b1111.

18.3.2.11 eQADC CFIFO Status Register EQADC_CFSR

The EQADC_CFSR contains the current CFIFO status. The EQADC_CFSRs are read only. Writing to the EQADC_CFSR has no effect.

Address: Base + 0x00AC Access: RO

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS0		CFS1		CFS2		CFS3		CFS4		CFS5		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-14. eQADC CFIFO Status Register (EQADC_CFSR)

Table 18-15. EQADC_CFSR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0–11
CFSn
[0:1]</td><td>CFIFO status. Indicates the current status of CFIFOn.
<table>
<tr><th>Value</th><th>Status</th><th>Explanation</th></tr>
<tr><td>0b00</td><td>IDLE</td><td>• CFIFO is disabled.
• CFIFO is in single-scan edge or level trigger mode and EQADC_FISRn[SSS] is not asserted.
• eQADC completes transferring the last entry of the command queue you defined in single-scan mode.</td></tr>
<tr><td>0b01</td><td>Reserved</td><td>Not applicable</td></tr>
<tr><td>0b10</td><td>WAITING FOR TRIGGER</td><td>• CFIFO mode changes to continuous-scan edge or level trigger mode.
• CFIFO mode changes to single-scan edge or level trigger mode and EQADC_FISRn[SSS] asserts.
• CFIFO mode changes to single-scan software trigger mode and EQADC_FISRn[SSS] negates.
• CFIFO pauses.
• eQADC transfers the last entry of the queue in continuous-scan edge trigger mode.</td></tr>
<tr><td>0b11</td><td>TRIGGERED</td><td>CFIFO is triggered</td></tr>
</table></td></tr>
<tr><td>12–31</td><td>Reserved</td></tr>
</table>

18.3.2.12 eQADC SSI Control Register EQADC_SSICR

The EQADC_SSICR configures the SSI submodule.

Address: Base + 0x00B4 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	MDT			0	0	0	0	BR			
W																
Reset	0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	1

Figure 18-15. eQADC SSI Control Register (EQADC_SSICR)

Table 18-16. EQADC_SSICR Field Descriptions

Field	Description
0–20	Reserved
21–23 MDT [0:2]	Minimum delay after transmission. Defines the minimum delay after transmission time (t_{MDT}) expressed in serial clock (FCK) periods. t_{MDT} is the minimum time $\overline{SDS}$ must be kept negated between two consecutive serial transmissions. Table 18-17 lists the minimum delay after transfer time according to how MDT is set. The MDT field must only be written when the serial transmissions from the eQADC SSI are disabled - See EQADC_MCR[ESSIE] field in Section 18.3.2.1, “eQADC Module Configuration Register (EQADC_MCR)”.
24–27	Reserved
28–31 BR [0:3]	Baud rate. Selects system clock divide factor as shown in Table 18-18. The baud clock is calculated by dividing the system clock by the clock divide factor specified with the BR field. **Note:** The BR field must only be written when the eQADC SSI is disabled - See EQADC_MCR[ESSIE] field in Section 18.3.2.1, “eQADC Module Configuration Register (EQADC_MCR).”

Table 18-17. Minimum Delay After Transmission (t_{MDT}) Time

MDT	t_{MDT} (FCK period)
0b000	1
0b001	2
0b010	3
0b011	4
0b100	5
0b101	6

Table 18-17. Minimum Delay After Transmission (t_{MDT}) Time (continued)

MDT	t_{MDT} (FCK period)
0b110	7
0b111	8

Table 18-18. System Clock Divide Factor for Baud Clock

BR[0:3]	System Clock Divide Factor [1]
0b0000	2
0b0001	3
0b0010	4
0b0011	5
0b0100	6
0b0101	7
0b0110	8
0b0111	9
0b1000	10
0b1001	11
0b1010	12
0b1011	13
0b1100	14
0b1101	15
0b1110	16
0b1111	17

1 If the system clock is divided by a odd number then the serial clock has a duty cycle different from 50%.

18.3.2.13 eQADC SSI Receive Data Register EQADC_SSIRDR

The eQADC SSI receive data register (EQADC_SSIRDR) records the last message received from the external device.

Address: Base + 0x00B8 Access: RO

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RDV	0	0	0	0	0	R_DATA									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	R_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-16. eQADC SSI Receive Data Register (EQADC_SSIRDR)

Table 18-19. EQADC_SSIRDR Field Descriptions

Field	Description
0 RDV	Receive data valid. Indicates if the last received data is valid. This bit is cleared automatically whenever the EQADC_SSIRDR is read. Writes have no effect. 0 Receive data is not valid. 1 Receive data is valid.
1–5	Reserved
6–31 R_DATA [0:25]	eQADC receive DATA. Contains the last result message that was shifted in. Writes to the R_DATA have no effect. Messages that were not completely received due to a transmission abort is not copied into EQADC_SSIRDR.

18.3.2.14 eQADC CFIFO Registers (EQADC_CF[0–5]R*n*)

EQADC_CF[0–5]R*n* provide visibility of the contents of a CFIFO for debugging purposes. Each CFIFO has four registers that are uniquely mapped to its four 32-bit entries. See Section 18.4.3, “eQADC Command FIFOs,” for more information on CFIFOs. These registers are read only. Data written to these registers is ignored.

Address: CFIFO0: Base + 0x0100 (CF0R0) Access: RO
Base + 0x0104 (CF0R1)
Base + 0x0108 (CF0R2)
Base + 0x010C (CF0R3)
CFIFO1: Base + 0x0140 (CF1R0)
Base + 0x0144 (CF1R1)
Base + 0x0148 (CF1R2)
Base + 0x014C (CF1R3)
CFIFO2: Base + 0x0180 (CF2R0)
Base + 0x0184 (CF2R1)
Base + 0x0188 (CF2R2)
Base + 0x018C (CF2R3)

CFIFO3: Base + 0x01C0 (CF3R0)
Base + 0x01C4 (CF3R1)
Base + 0x01C8 (CF3R2)
Base + 0x01CC (CF3R3)
CFIFO4: Base + 0x0200 (CF4R0)
Base + 0x0204 (CF4R1)
Base + 0x0208 (CF4R2)
Base + 0x020C (CF4R3)
CFIFO5: Base + 0x0240 (CF5R0)
Base + 0x0244 (CF5R1)
Base + 0x0248 (CF5R2)
Base + 0x024C (CF5R3)

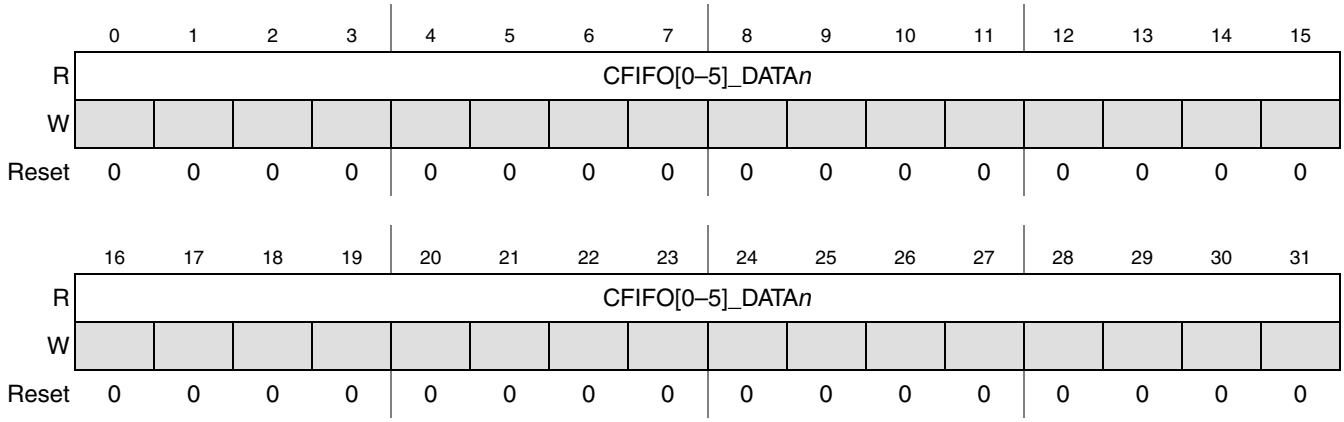


Figure 18-17. eQADC CFIF0[0–5] Registers (EQADC_CF[0–5]R*n*)

Table 18-20. EQADC_CF[0–5]R*n* Field Descriptions

Field	Description
0–31 CFIFO[0–5]_DATA*n* [0:31]	CFIFO[0–5]_data*n*. Returns the value stored within the entry of CFIFO[0–5]. Each CFIFO is composed of four 32-bit entries, with register 0 being mapped to the entry with the smallest memory mapped address.

18.3.2.15 eQADC RFIFO Registers (EQADC_RF[0–5]R*n*)

EQADC_RF[0–5]R*n* allows you to read the contents of a RFIFO for debugging purposes. Each RFIFO has four registers that are uniquely mapped to its four 16-bit entries. See Section 18.4.4, "Result FIFOs," for more information on RFIFOs. These registers are read only. Data written to these registers is ignored.

Address: RFIFO0: Base + 0x0300 (RF0R0) Access: RO
Base + 0x0304 (RF0R1)
Base + 0x0308 (RF0R2)
Base+0x030C (RF0R3)
RFIFO1: Base + 0x0340 (RF1R0)
Base + 0x0344 (RF1R1)
Base + 0x0348 (RF1R2)
Base + 0x034C (RF1R3)
RFIFO2: Base + 0x0380 (RF2R0)
Base + 0x0384 (RF2R1)
Base + 0x0388 (RF2R2)
Base + 0x038C (RF2R3)

RFIFO3: Base + 0x03C0 (RF3R0)
Base + 0x03C4 (RF3R1)
Base + 0x03C8 (RF3R2)
Base + 0x03CC (RF3R3)
RFIFO4: Base + 0x0400 (RF4R0)
Base + 0x0404 (RF4R1)
Base + 0x0408 (RF4R2
Base + 0x040C (RF4R3)
RFIFO5: Base + 0x0440 (RF5R0)
Base + 0x0444 (RF5R1)
Base + 0x0448 (RF5R2)
Base + 0x044C (RF5R3)

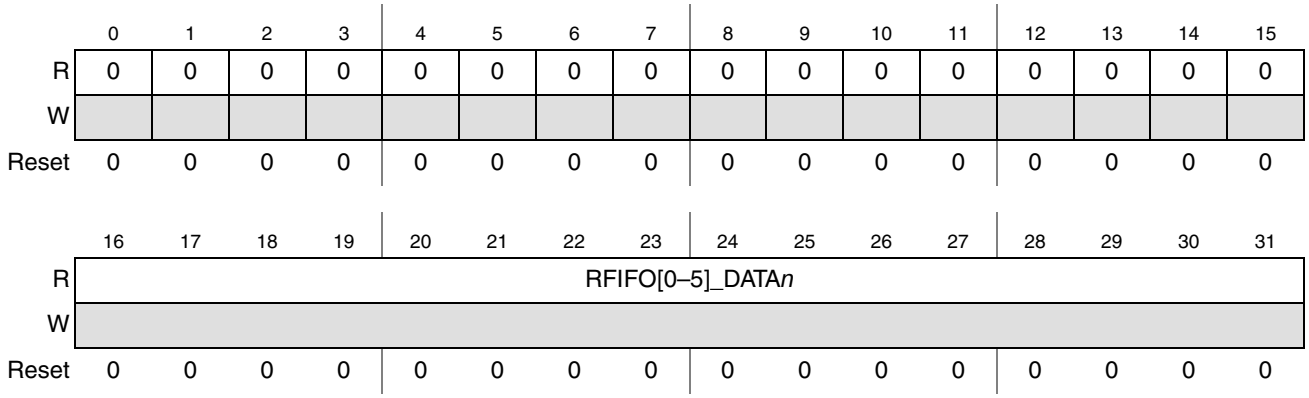


Figure 18-18. eQADC RFIFO*n* Registers (EQADC_RF[0–5]R*n*)

The following table lists the field descriptions in the eQADC receive FIFO registers.

Table 18-21. EQADC_RF[0–5]R*n* Field Descriptions

Field	Description
0–15	Reserved
16–31 RFIFO[0–5]_DATA*n* [0:15]	RFIFO[0–5] data *n*. Returns the value in the RFIFO[0–5] entry. Each RFIFO has four 16-bit entries, with register 0 being mapped to the entry with the smallest memory mapped address.

18.3.3 On-Chip ADC Registers

This section describes a list of registers that control on-chip ADC operation. The ADC registers are not part of the CPU accessible memory map. These registers can only be accessed indirectly through configuration commands. There are five non memory mapped registers per ADC, five for ADC0 and five for ADC1. The address, usage, and access privilege of each register is shown in Table 18-22 and Table 18-23. Data written to or read from reserved areas of the memory map is undefined.

Their assigned addresses are the values used to set the ADC_REG_ADDRESS field of the read/write configuration commands bound for the on-chip ADCs. These are halfword addresses. Further, the following restrictions apply when accessing these registers:

- Registers ADC0_CR, ADC0_GCCR, and ADC0_OCCR can only be accessed by configuration commands sent to the ADC0 command buffer.
- Registers ADC1_CR, ADC1_GCCR, and ADC1_OCCR can only be accessed by configuration commands sent to the ADC1 command buffer.
- Registers ADC_TSCR and ADC_TBCR can be accessed by configuration commands sent to the ADC0 command buffer or to the ADC1 command buffer. A data write to ADC_TSCR through a configuration command sent to the ADC0 command buffer writes the same memory location as when writing to it through a configuration command sent to the ADC1 command buffer. The same is valid for ADC_TBCR.

NOTE

Simultaneous write accesses from the ADC0 and ADC1 command buffers to ADC_TSCR or to ADC_TBCR are not allowed.

Table 18-22. ADC0 Registers

ADC0 Register Address	Use	Access
0x0000	ADC0 Address 0x00 is used for conversion command messages.	—
0x0001	ADC0 Control Register (ADC0_CR)	Write/Read
0x0002	ADC Time Stamp Control Register (ADC_TSCR) [1]	Write/Read
0x0003	ADC Time Base Counter Register (ADC_TBCR) [1]	Write/Read
0x0004	ADC0 Gain Calibration Constant Register (ADC0_GCCR)	Write/Read
0x0005	ADC0 Offset Calibration Constant Register (ADC0_OCCR)	Write/Read
0x0006–0x00FF	Reserved	—

[1] This register is also accessible by configuration commands sent to the ADC1 command buffer.

Table 18-23. ADC1 Registers

ADC1 Register Address	Use	Access
0x0000	ADC1 Address 0x00 is used for conversion command messages.	—
0x0001	ADC1 Control Register (ADC1_CR)	Write/Read

Table 18-23. ADC1 Registers (continued)

0x0002	ADC Time Stamp Control Register (ADC_TSCR) [1]	Write/Read
0x0003	ADC Time Base Counter Register (ADC_TBCR) [1]	Write/Read
0x0004	ADC1 Gain Calibration Constant Register (ADC1_GCCR)	Write/Read
0x0005	ADC1 Offset Calibration Constant Register (ADC1_OCCR)	Write/Read
0x0006–0x00FF	Reserved	—

[1] This register is also accessible by configuration commands sent to the ADC0 command buffer.

18.3.3.1 ADC*n* Control Registers (ADC0_CR and ADC1_CR)

The ADC*n* control registers (ADC*n*_CR) are used to configure the on-chip ADCs.

Address: 0x0001 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ADC0_EN	0	0	0	ADC0_EMUX	0	0	0	0	0	0	ADC0_CLK_PS				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ADC1_EN	0	0	0	ADC1_EMUX	0	0	0	0	0	0	ADC1_CLK_PS				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Figure 18-19. ADC*n* Control Registers (ADC0_CR and ADC1_CR)

Table 18-24. ADC*n*_CR Field Descriptions

Field	Description
0 ADC*n*_EN	ADC*n* enable. Enables ADC*n* to perform A/D conversions. See Section 18.4.5.1, "Enabling and Disabling the on-chip ADCs," for details. 0 ADC is disabled. Clock supply to ADC0/1 is stopped. 1 ADC is enabled and ready to perform A/D conversions. **Note:** The bias generator circuit inside the ADC ceases functioning when both ADC0_EN and ADC1_EN bits are negated. **Note:** Conversion commands sent to a disabled ADC are ignored by the ADC control hardware. **Note:** When the ADC*n*_EN status is changed from asserted to negated, the ADC clock does not stop until it reaches its low phase.
1–3	Reserved

Table 18-24. ADC*n*_CR Field Descriptions (continued)

Field	Description
4 ADC*n*_ EMUX	ADC*n* external multiplexer enable. When ADC*n*_EMUX is asserted, the MA pins output digital values to the external channel number selected to convert external multiplexer inputs. See Section 18.4.6, "Internal and External Multiplexing," for a detailed description about how ADC*n*_EMUX affects channel number decoding. 0 External multiplexer disabled; no external multiplexer channels can be selected. 1 External multiplexer enabled; external multiplexer channels can be selected. **Note:** Both ADC*n*_EMUX bits must not be asserted at the same time. **Note:** The ADC*n*_EMUX bit must only be written when the ADC*n*_EN bit is negated. ADC*n*_EMUX can be set during the same write cycle used to set ADC*n*_EN.
5–10	Reserved
11–15 ADC*n*_ CLK_PS [0:4]	ADC*n* clock prescaler. The ADC*n*_CLK_PS field controls the system clock divide factor for the ADC*n* clock as in Table 18-25. See Section 18.4.5.2, "ADC Clock and Conversion Speed," for details about how to set ADC0/1_CLK_PS. The ADC*n*_CLK_PS field must only be written when the ADC*n*_EN bit is negated. This field can be configured during the same write cycle used to set ADC*n*_EN.

Table 18-25. System Clock Divide Factor for ADC Clock

ADC*n*_CLK_PS[0:4]	System Clock Divide Factor
0b00000	2
0b00001	4
0b00010	6
0b00011	8
0b00100	10
0b00101	12
0b00110	14
0b00111	16
0b01000	18
0b01001	20
0b01010	22
0b01011	24
0b01100	26
0b01101	28
0b01110	30
0b01111	32
0b10000	34
0b10001	36
0b10010	38

Table 18-25. System Clock Divide Factor for ADC Clock (continued)

ADC*n*_CLK_PS[0:4]	System Clock Divide Factor
0b10011	40
0b10100	42
0b10101	44
0b10110	46
0b10111	48
0b11000	50
0b11001	52
0b11010	54
0b11011	56
0b11100	58
0b11101	60
0b11110	62
0b11111	64

18.3.3.2 ADC Time Stamp Control Register (ADC_TSCR)

The ADC_TSCR contains a system clock divide factor used in the making of the time base counter clock. It determines at what frequency the time base counter runs. ADC_TSCR can be accessed by configuration commands sent to ADC0 or to ADC1. A data write to ADC_TSCR using a configuration command sent to ADC0 writes to the same memory location as a write using a configuration command sent to ADC1.

NOTE

Simultaneous write accesses from ADC0 and ADC1 to ADC_TSCR are not allowed.

Address: 0x0002 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	TBC_CLK_PS			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-20. ADC Time Stamp Control Register (ADC_TSCR)

Table 18-26. ADC_TSCR Field Descriptions

Field	Description
0–11	Reserved
12–15 TBC_ CLK_PS [0:3]	Time base counter clock prescaler. Contains the system clock divide factor for the time base counter. It controls the accuracy of the time stamp. The prescaler is disabled when TBC_CLK_PS is set to 0b0000.

Table 18-27. Clock Divide Factor for Time Stamp

TBC_CLK_PS[0:3]	System Clock Divide Factor	Clock to Time Stamp Counter for a 120 MHz System Clock (MHz)
0b0000	Disabled	Disabled
0b0001	1	120
0b0010	2	60
0b0011	4	30
0b0100	6	20
0b0101	8	15
0b0110	10	12
0b0111	12	10
0b1000	16	7.5
0b1001	32	3.75
0b1010	64	1.88
0b1011	128	0.94
0b1100	256	0.47
0b1101	512	0.23
0b1110–0b1111	Reserved	—

NOTE

If TBC_CLK_PS is not set to disabled, it must not be changed to any other value besides disabled. If TBC_CLK_PS is set to disabled it can be changed to any other value.

18.3.3.3 ADC Time Base Counter Registers (ADC_TBCR)

The ADC_TBCR contains the current value of the time base counter. ADC_TBCR can be accessed by configuration commands sent to ADC0 or to ADC1. A data write to ADC_TBCR using a configuration command sent to ADC0 writes the same memory location as a write using a configuration command sent to ADC1.

NOTE

Simultaneous write accesses from ADC0 and ADC1 to ADC_TBCR are not allowed.

Address: 0x0003 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TBC_VALUE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-21. ADC Time Base Counter Register (ADC_TBCR)

Table 18-28. ADC_TBCR Field Descriptions

Field	Description
0–15 TBC_ VALUE [0:15]	Time base counter VALUE. Contains the current value of the time base counter. Reading TBC_VALUE returns the current value of time base counter. Writes to TBC_VALUE register load the written data to the counter. The time base counter counts from 0x0000 to 0xFFFF and wraps when reaching 0xFFFF.

18.3.3.4 ADC*n* Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)

The ADC*n*_GCCR contains the gain calibration constant used to fine-tune the ADC*n* conversion results. See Section 18.4.5.4, "ADC Calibration Feature," for details about the calibration scheme used in the eQADC.

Address: 0x0004 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	GCC0														
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	GCC1														
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-22. ADC*n* Gain Calibration Constant Registers (ADC*n*_GCCR)

Table 18-29. ADC*n*_GCCR Field Descriptions

Field	Description
0	Reserved
1–15 GCC*n* [0:14]	ADC*n* gain calibration constant (GCC1) is an unsigned 15-bit fixed pointed number expressed in the *GCC_INT.GCC_FRAC* binary format. Use the gain calibration constant to fine-tune ADC*n* conversion results. The integer part of the gain constant (GCC_INT) contains a single binary digit while its fractional part (GCC_FRAC) contains 14 digits. For details about the GCC data format see Section 18.4.5.4.2, "MAC Unit and Operand Data Format."

18.3.3.5 ADC*n* Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)

The ADC*n*_OCCR contains the offset calibration constant used to fine-tune of ADC0/1 conversion results. The offset constant is a signed 14-bit integer value. See Section 18.4.5.4, "ADC Calibration Feature," for details about the calibration scheme used in the eQADC.

Address: 0x0005 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	OCC0													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	OCC1													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-23. ADC*n* Offset Calibration Constant Registers (ADC*n*_OCCR)

Table 18-30. ADC*n*_OCCR Field Descriptions

Field	Description
0–1	Reserved
2–15 OCC*n* [0:13]	ADC*n* offset calibration constant. Contains the offset calibration constant used to fine-tune ADC*n* conversion results. Use the two's complement representation for expressing negative values.

18.4 Functional Description

The eQADC provides a parallel interface to two on-chip ADCs, and a single master to single slave serial interface to an off-chip external device. The two on-chip ADCs can access all the analog channels.

Initially, command data is contained in system memory in a user defined data queue structure. Command data is moved between queues you defined and CFIFOs by the host CPU or by the eDMA which responds to interrupt and eDMA requests generated by the eQADC. The eQADC supports software and hardware triggers from other modules or external pins to initiate transfers of commands from the multiple CFIFOs to the on-chip ADCs or to the external device.

CFIFOs can be configured to be in single-scan or continuous-scan mode. When a CFIFO is configured to single-scan mode, the eQADC scans the command queue you defined once. The eQADC stops transferring commands from the triggered CFIFO after detecting the EOQ bit set in the last transfer. After an EOQ bit is detected, software involvement is required to rearm the CFIFO so that it can detect new trigger events.

When a CFIFO is configured for continuous-scan mode, the whole user command queue is scanned multiple times. After the detection of an asserted EOQ bit in the last command transfer, command transfers can continue or not depending on the mode of operation of the CFIFO.

The eQADC can also in parallel and independently of the CFIFOs receive data from the on-chip ADCs or from off-chip external device into multiple RFIFOs. Result data is moved from the RFIFOs to the result queues you defined in system memory by the host CPU or by the eDMA.

18.4.1 Data Flow in the eQADC

Figure 18-24 shows how command data flows inside the eQADC system. A command message is the predefined format in which command data is stored in command queues you defined. A command message has 32 bits and is composed of two parts: a CFIFO header and an ADC command. Command messages are moved from command queues you defined to the CFIFOs by the host CPU or by the eDMA as they respond to interrupt and eDMA requests generated by the eQADC. The eQADC generates these requests whenever a CFIFO is not full. The FIFO control unit transfers only the command part of the command message to the ADC. Information in the CFIFO header together with the upper bit of the ADC command is used by the FIFO control unit to arbitrate which triggered CFIFO transfers the next command. Because command transfer through the serial interface can take significantly more time than a parallel transfer to the on-chip ADCs, command transfers for on-chip ADCs occur concurrently with the transfers through the serial interface. Commands sent to the ADCs are executed in a first-in-first-out (FIFO) basis and three types of results can be expected: data read from an ADC register, a conversion result, or a time stamp. The order at which ADC commands sent to the external device are executed, and the type of results that can be expected depends on the architecture of that device with the exception of unsolicited data like null messages for example.

NOTE

While the eQADC pops commands out from a CFIFO, it also is checking the number of entries in the CFIFO and generating requests to fill it. The process of pushing and popping commands to and from a CFIFO can occur simultaneously.

The FIFO control unit expects all incoming results to be shaped in a pre-defined result message format. Figure 18-25 shows how result data flows inside the eQADC system. Results generated on the on-chip ADCs are formatted into result messages inside the result format and calibration submodule. Results returning from the external device are already formatted into result messages and therefore bypass the result format and calibration submodule located inside the eQADC. A result message is composed of an RFIFO header and an ADC Result. The FIFO control unit decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result are sent. After in an RFIFO, the ADC result is moved to the corresponding user result queue by the host CPU or by the eDMA as they respond to interrupt and eDMA requests generated by the eQADC. The eQADC generates these requests whenever an RFIFO has at least one entry.

NOTE

While conversion results are returned, the eQADC is checking the number of entries in the RFIFO and generating requests to empty it. The process of pushing and popping ADC results to and from an RFIFO can occur simultaneously.

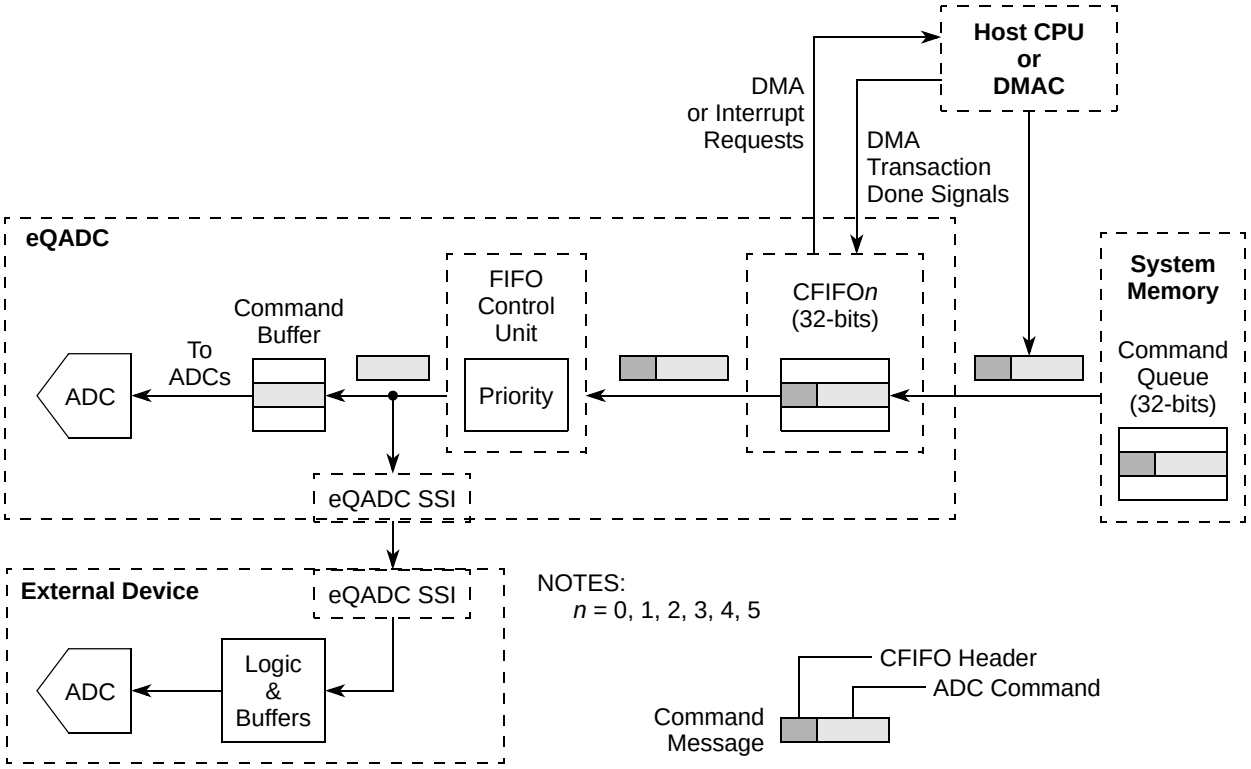


Figure 18-24. Command Flow During eQADC Operation

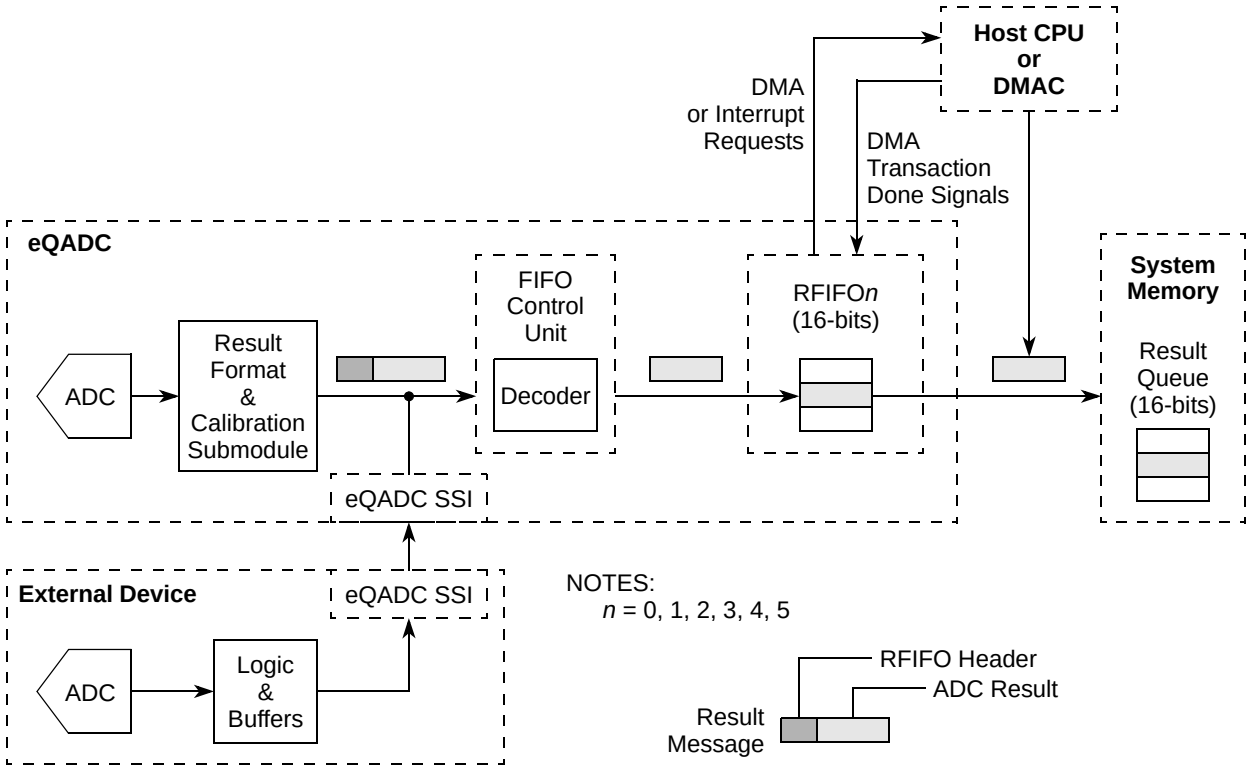


Figure 18-25. Result Flow During eQADC Operation

18.4.1.1 Assumptions/Requirements Regarding the External Device

The external device exchanges command and result data with the eQADC through the eQADC SSI interface. This section explains the minimum requirements an external device must meet to correctly interface with the eQADC. Some assumptions about the architecture of the external device are also described.

18.4.1.1.1 eQADC SSI Protocol Support

The external device must fully support the eQADC SSI protocol as specified in Section 18.4.8, “eQADC Synchronous Serial Interface (SSI) Submodule,” section of this document. Support for the abort feature is optional. When aborts are not supported, all command messages bound for an external command buffer must have the ABORT_ST bit negated. See Section , “ Command Message Format for External Device Operation.”

18.4.1.1.2 Number of Command Buffers and Result Buffers

The external device must have a minimum of one and a maximum of two command buffers to store command data sent from the eQADC. If more than two command buffers are implemented in the external device, they are not recognized by the eQADC as valid destinations for commands. In this document, the two valid external command buffers are referred to as command buffer 2 and command buffer 3 (the two on-chip ADCs being command buffer 0 and 1). The external device decides to which external command buffer a command is sent by decoding the upper bit (BN bit) of the ADC command. See Section ,

" Command Message Format for External Device Operation." An external device that only implements one command buffer can ignore the BN bit.

The limit of two command buffers does not limit the number of result buffers in the slave device.

18.4.1.1.3 Command Execution and Result Return

Commands sent to a specific external command buffer must be executed in the order they were received.

Results generated by the execution of commands in an external command buffer must be returned in the order that the command buffer received these commands.

18.4.1.1.4 Null and Result Messages

The external device must be capable of correctly processing null messages as specified in the Section 18.3.2.2, "eQADC Null Message Send Format Register (EQADC_NMSFR)."

In case no valid result data is available to be sent to the eQADC, the external device must send data in the format specified in Section , "Null Message Format for External Device Operation."

In case valid result data is available to sent to the eQADC, the external device must send data in the format specified in Section , "Result Message Format for External Device Operation."

The BUSY0/1 fields of all messages sent from the external device to the eQADC must be correctly encoded according to the latest information on the fullness state of the command buffers. For example, if external command buffer 2 is empty before the end of the current serial transmission and if at the end of this transmission the external device receives a command to command buffer 2, then the BUSY0 field, that is to be sent to the eQADC on the next serial transmission, must be encoded assuming that the external command buffer has one entry.

18.4.1.2 Message Format in eQADC

This section explains the command and result message formats used for on-chip ADC operation and for external device operation.

A command message is the pre-defined format at which command data is stored in the command queues you defined. A command message has 32 bits and is composed of two parts: a CFIFO header and an ADC command. The size of the CFIFO header is fixed to 6 bits, and it works as inputs to the FIFO control unit. The header controls when a command queue ends, when it pauses, if commands are sent to internal or external buffers, and if it can abort a serial data transmission. Information contained in the CFIFO header, together with the upper bit of the ADC command, is used by the FIFO control unit to arbitrate which triggered CFIFO transfers the next command. ADC commands are encoded inside the least significant 26 bits of the command message.

A result message is composed of an RFIFO header and an ADC result. The FIFO control unit decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result is sent. The ADC result field is always 16 bits.

18.4.1.2.1 Message Formats for On-Chip ADC Operation

This section describes the command/result message formats used for on-chip ADC operation.

NOTE

Although this subsection describes how the command and result messages are formatted to communicate with the on-chip ADCs, nothing prevents the programmer from using a different format when communicating with an external device through the serial interface. See Section 18.4.1.2.2, "Message Formats for External Device Operation." Apart from the BN bit, the ADC command of a command message can be formatted to communicate to an arbitrary external device provided that the device returns an RFIFO header in the format expected by the eQADC. When the FIFO control unit receives return data message, it decodes the message tag field and stores the 16-bit data into the corresponding RFIFO.

Conversion Command Message Format for On-Chip ADC Operation

Figure 18-26 describes the command message format for conversion commands when interfacing with the on-chip ADCs. A conversion result is always returned for conversion commands and time stamp information can be optionally requested. Conversion commands are sent to the ADC internal memory map address zero, therefore the lower byte of the lower byte of conversion commands is always cleared to 0 to distinguish it from configuration commands.

Address: 0x0005 Access: R/W

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	Reserved			EB (0b0)	BN	CAL	MESSAGE_TAG				LST		TSR	FMT
CFIFO Header						ADC Command									

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CHANNEL_NUMBER								ADC_REG_ADDRESS							
ADC Command								0	0	0	0	0	0	0	0

Figure 18-26. Conversion Command Message Format for On-Chip ADC Operation

Table 18-31. On-Chip ADC Field Descriptions: Conversion Command Message Format

Field	Description
0 EOQ	End-of-queue. Asserted in the last command of a command queue to indicate to the eQADC that a scan of the queue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to 0. Depending on the CFIFO operating mode, the CFIFO status changes when it detects when the EOQ bit on the last transferred command is asserted. See Section 18.4.3.4, "CFIFO Scan Trigger Modes," for details. 0 Not the last entry of the command queue. 1 Last entry of the command queue. **Note:** If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
1 PAUSE	Pause. Allows software to create sub-queues within a command queue. When the eQADC completes the transfer of a command with an asserted pause bit, the CFIFO enters the WAITING FOR TRIGGER state. See Section 18.4.3.5.1, "CFIFO Operation Status," for a description of the state transitions. The pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 0 Do not enter WAITING FOR TRIGGER state after transfer of the current command message. 1 Enter WAITING FOR TRIGGER state after transfer of the current command message. **Note:** If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
2–4	Reserved
5 EB	External buffer bit. A negated EB bit indicates that the command is sent to an on chip ADC. 0 Command is sent to an internal buffer. 1 Command is sent to an external buffer.
6 BN	Buffer number. For internal commands, indicates the ADC to which the message is sent. For external commands, indicates to which command FIFO the messages is sent. ADCs 0 and 1 can be internal or external depending on the EBI bit setting. 0 Message sent to ADC 0. 1 Message sent to ADC 1.
7 CAL	Calibration. Indicates if the returning conversion result must be calibrated. 0 Do not calibrate conversion result. 1 Calibrate conversion result.

Table 18-31. On-Chip ADC Field Descriptions: Conversion Command Message Format (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>8–11
MESSAGE_
TAG
[0:3]</td><td>MESSAGE_TAG field. Allows the eQADC to separate returning results into different RFIFOs. When the eQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the external device/on-chip ADC returns the result with the same MESSAGE_TAG. The eQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data.
<table>
<tr><th>MESSAGE_TAG[0:3]</th><th>MESSAGE_TAG Meaning</th></tr>
<tr><td>0b0000</td><td>Result is sent to RFIFO 0</td></tr>
<tr><td>0b0001</td><td>Result is sent to RFIFO 1</td></tr>
<tr><td>0b0010</td><td>Result is sent to RFIFO 2</td></tr>
<tr><td>0b0011</td><td>Result is sent to RFIFO 3</td></tr>
<tr><td>0b0100</td><td>Result is sent to RFIFO 4</td></tr>
<tr><td>0b0101</td><td>Result is sent to RFIFO 5</td></tr>
<tr><td>0b0110–0b0111</td><td>Reserved</td></tr>
<tr><td>0b1000</td><td>Null message received</td></tr>
<tr><td>0b1001</td><td>Reserved for customer use. [1]</td></tr>
<tr><td>0b1010</td><td>Reserved for customer use. [1]</td></tr>
<tr><td>0b1011–0b1111</td><td>Reserved</td></tr>
</table>
[1] These messages are treated as null messages. Therefore, they must obey the format for incoming null messages and return valid BUSY0/1 fields. See Section , “Null Message Format for External Device Operation.”</td></tr>
<tr><td>12–13
LST
[0:1]</td><td>Long sampling time. These two bits determine the duration of the sampling time in ADC clock cycles.
Note: For external mux mode, 64 or 128 sampling cycles is recommended.
<table>
<tr><th>LST[0:1]</th><th>Sampling cycles (ADC Clock Cycles)</th></tr>
<tr><td>0b00</td><td>2</td></tr>
<tr><td>0b01</td><td>8</td></tr>
<tr><td>0b10</td><td>64</td></tr>
<tr><td>0b11</td><td>128</td></tr>
</table></td></tr>
<tr><td>14
TSR</td><td>Time stamp request. TSR indicates the request for a time stamp. When TSR is asserted, the on-chip ADC control logic returns a time stamp for the current conversion command after the conversion result is sent to the RFIFOs. See Section 18.4.5.3, “Time Stamp Feature,” for details.
0 Return conversion result only.
1 Return conversion time stamp after the conversion result.</td></tr>
<tr><td>15
FMT</td><td>Conversion data format. FMT specifies to the eQADC how to format the 12-bit conversion data returned by the ADCs into the 16-bit format which is sent to the RFIFOs. See Section , “ADC Result Format for On-Chip ADC Operation,” for details.
0 Right justified unsigned.
1 Right justified signed.</td></tr>
</table>

Table 18-31. On-Chip ADC Field Descriptions: Conversion Command Message Format (continued)

Field	Description
16–23 CHANNEL_ NUMBER [0:7]	Channel number. Selects the analog input channel. The software programs this field with the channel number corresponding to the analog input pin to be sampled and converted. See Section 18.4.6.1, "Channel Assignment," for details.
24–31	

Write Configuration Command Message Format for On-Chip ADC Operation

Figure 18-27 describes the command message format for a write configuration command when interfacing with the on-chip ADCs. A write configuration command is used to set the control registers of the on-chip ADCs. No conversion data is returned for a write configuration command. Write configuration commands are differentiated from read configuration commands by a negated R/W bit.

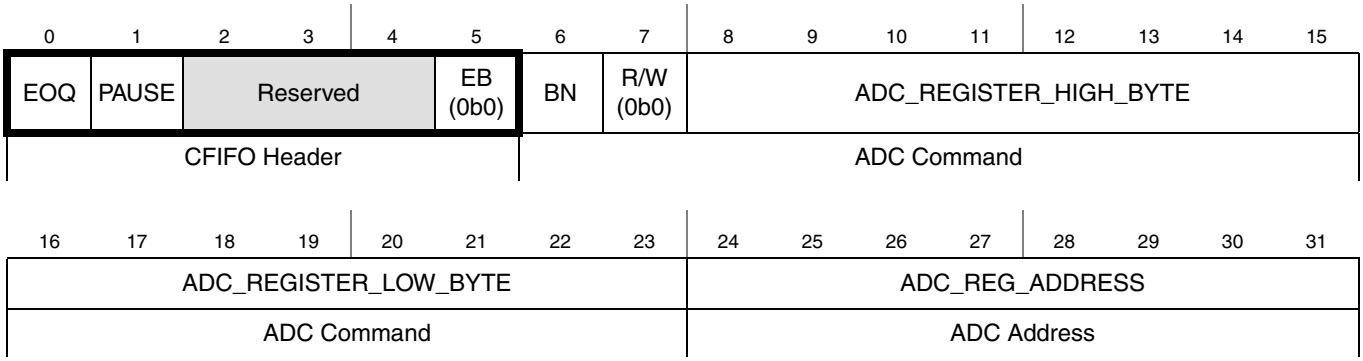


Figure 18-27. Write Configuration Command Message Format for On-chip ADC Operation

Table 18-32. On-Chip ADC Field Descriptions: Write Configuration

Field	Description
0 EOQ	End-of-queue. Asserted in the last command of a command queue to indicate to the eQADC that a scan of the queue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to 0. Depending on the CFIFO mode of operation, the CFIFO status also changes upon the detection of an asserted EOQ bit on the last transferred command. See Section 18.4.3.4, "CFIFO Scan Trigger Modes," for details. 0 Not the last entry of the command queue. 1 Last entry of the command queue. **Note:** If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
1 PAUSE	Pause bit. Allows software to create sub-queues within a command queue. When the eQADC completes the transfer of a command with an asserted pause bit, the CFIFO enters the WAITING FOR TRIGGER state. See Section 18.4.3.5.1, "CFIFO Operation Status," for a description of the state transitions. The pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 0 Do not enter WAITING FOR TRIGGER state after transfer of the current command message. 1 Enter WAITING FOR TRIGGER state after transfer of the current command message. **Note:** If both the pause and EOQ bits are asserted in the same command message, the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
2–4	Reserved

Table 18-32. On-Chip ADC Field Descriptions: Write Configuration (continued)

Field	Description
5 EB	External buffer bit. Always clear this bit for messages sent to an on-chip ADC. 0 Command is sent to an internal command buffer. 1 Command is sent to an external command buffer.
6 BN	Buffer number. Indicates to which buffer the message is sent.Buffers 1 and 0 can either be internal or external depending on the EBI bit setting. 0 Message buffer 0. 1 Message buffer 1.
7 R/W	Read/write. A negated R/W indicates a write configuration command. 0 Write 1 Read
8–15 ADC_REGISTER_HIGH_BYTE [0:7]	ADC register high byte. The value to be written into the most significant 8 bits of control/configuration register when the R/W bit is negated.
16–23 ADC_REGISTER_LOW_BYTE [0:7]	ADC register low byte. The value to be written into the least significant 8 bits of a control/configuration register when the R/W bit is negated.
24–31 ADC_REG_ADDRESS [0:7]	ADC register address. Identifies to which ADC register the read or write is performed. Only use 16-bit (halfword) addresses. See Table 18-22. See Table 18-22.

Read Configuration Command Message Format for On-Chip ADC Operation

Figure 18-28 describes the command message format for a read configuration command when interfacing with the on-chip ADCs. A read configuration command is used to read the contents of the on-chip ADC registers which are only accessible via command messages. Read configuration commands are differentiated from write configuration commands by an asserted R/W bit.

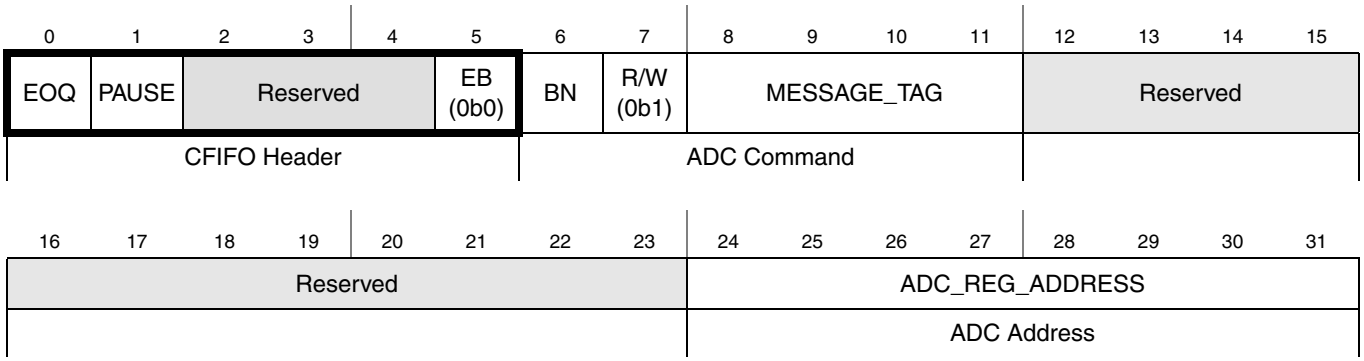


Figure 18-28. Read Configuration Command Message Format for On-Chip ADC Operation

Table 18-33. On-Chip ADC Field Descriptions: Read Configuration

Field	Description
0 EOQ	End-of-queue. Asserted in the last command of a command queue to indicate to the eQADC that a scan of the queue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to 0. Depending on the CFIFO mode of operation, the CFIFO status changes upon the detection of an asserted EOQ bit on the last transferred command. See Section 18.4.3.4, "CFIFO Scan Trigger Modes," for details. 0 Not the last entry of the command queue. 1 Last entry of the command queue. **Note:** If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
1 PAUSE	Pause bit. Allows software to create sub-queues within a command queue. When the eQADC completes the transfer of a command with an asserted pause bit, the CFIFO enters the WAITING FOR TRIGGER state. See Section 18.4.3.5.1, "CFIFO Operation Status," for a description of the state transitions. The pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 0 Do not enter WAITING FOR TRIGGER state after transfer of the current command message. 1 Enter WAITING FOR TRIGGER state after transfer of the current command message. **Note:** If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
2–4	Reserved
5 EB	External buffer bit. Always clear this bit for messages sent to an on-chip ADC. 0 Command is sent to an internal command buffer. 1 Command is sent to an external command buffer.
6 BN	Buffer number. Indicates to which buffer the message is sent. Buffers 1 and 0 can either be internal or external depending on the EBI bit setting. 0 Message buffer 0. 1 Message buffer 1.

Table 18-33. On-Chip ADC Field Descriptions: Read Configuration (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7
R/W</td><td>Read/write. An asserted R/W bit indicates a read configuration command.
0 Write
1 Read</td></tr>
<tr><td>8–11
MESSAGE_TAG
[0:3]</td><td>MESSAGE_TAG field. Allows the eQADC to separate returning results into different RFIFOs. When the eQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the external device/on-chip ADC returns the result with the same MESSAGE_TAG. The eQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data.
<table>
<tr><th>MESSAGE_TAG[0:3]</th><th>MESSAGE_TAG Meaning</th></tr>
<tr><td>0b0000</td><td>Result is sent to RFIFO 0</td></tr>
<tr><td>0b0001</td><td>Result is sent to RFIFO 1</td></tr>
<tr><td>0b0010</td><td>Result is sent to RFIFO 2</td></tr>
<tr><td>0b0011</td><td>Result is sent to RFIFO 3</td></tr>
<tr><td>0b0100</td><td>Result is sent to RFIFO 4</td></tr>
<tr><td>0b0101</td><td>Result is sent to RFIFO 5</td></tr>
<tr><td>0b0110–0b0111</td><td>Reserved</td></tr>
<tr><td>0b1000</td><td>Null message received</td></tr>
<tr><td>0b1001</td><td>Reserved for customer use. [1]</td></tr>
<tr><td>0b1010</td><td>Reserved for customer use. [1]</td></tr>
<tr><td>0b1011–0b1111</td><td>Reserved</td></tr>
</table>
[1] These messages are treated as null messages. Therefore, they must obey the format for incoming null messages and return valid BUSY0/1 fields. See Section , “Null Message Format for External Device Operation.”</td></tr>
<tr><td>12–23</td><td>Reserved</td></tr>
<tr><td>24–31
ADC_REG_
ADDRESS
[0:7]</td><td>ADC register address. Identifies to which ADC register the read or write is performed. Only use 16-bit (halfword) addresses. See Table 18-22. See Table 18-22.</td></tr>
</table>

ADC Result Format for On-Chip ADC Operation

When the FIFO control unit receives a return data message, it decodes the MESSAGE_TAG field and stores the 16-bit data into the appropriate RFIFO. This section describes the ADC result portion of the result message returned by the on-chip ADCs.

The 16-bit data stored in the RFIFOs can be the following:

- Data read from an ADC register with a read configuration command. In this case, the stored 16-bit data corresponds to the contents of the ADC register that was read.
- A time stamp. In this case, the stored 16-bit data is the value of the time base counter latched when the eQADC detects the end of the analog input voltage sampling. For details see Section 18.4.5.3, “Time Stamp Feature.”

- A conversion result. In this case, the stored 16-bit data contains a right justified 14-bit result data. The conversion result can be calibrated or not depending on the status of CAL bit in the command that requested the conversion. When the CAL bit is negated, this 14-bit data is obtained by executing a 2-bit left-shift on the 12-bit data received from the ADC. When the CAL bit is asserted, this 14-bit data is the result of the calculations performed in the EQADC MAC unit using the12-bit data received from the ADC and the calibration constants GCC and OCC (See Section 18.4.5.4, “ADC Calibration Feature”). Then, this 14-bit data is further formatted into a 16-bit format according to the status of the FMT bit in the conversion command. When FMT is asserted, the 14-bit result data is reformatted to look as if it was measured against an imaginary ground at V_{REF} / 2 (the most significant bit (MSB) bit of the 14-bit result is inverted), and is sign-extended to a 16-bit format as in Figure 18-29. When FMT is negated, the eQADC zero-extends the 14-bit result data to a 16-bit format as in Figure 18-30. Correspondence between the analog voltage in a channel and the calculated digital values is shown in Table 18-36.

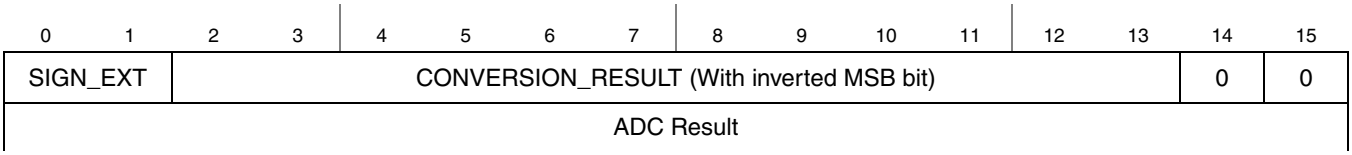


Figure 18-29. ADC Result Format when FMT = 1 (Right Justified Signed)—On-Chip ADC Operation

Table 18-34. ADC Result Format when FMT = 1 Field Descriptions

Field	Description
0–1 SIGN_EXT [0:1]	Sign extension. Only has meaning when FMT is asserted. SIGN_EXT is 0b00 when CONVERSION_RESULT is positive, and 0b11 when CONVERSION_RESULT is negative.
2–15 CONVERSION_ RESULT [0:13]	Conversion result. A digital value corresponding to the analog input voltage in a channel when the conversion command was initiated. The two’s complement representation is used to express negative values.

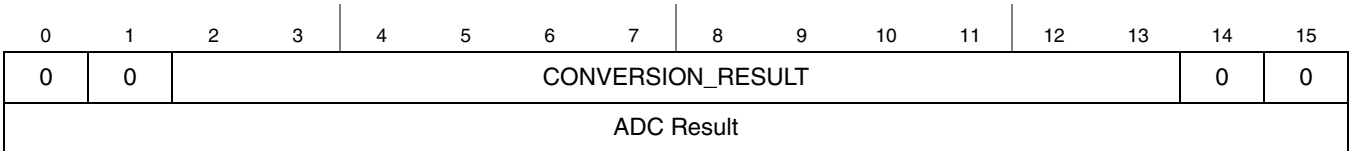


Figure 18-30. ADC Result Format when FMT = 0 (Right Justified Unsigned)—On-Chip ADC Operation

Table 18-35. ADC Result Format when FMT = 0 Field Descriptions

Field	Description
0–1 SIGN_EXT [0:1]	Sign extension. These two bits are always zero for FMT=0 because unsigned results are positive.
2–15 CONVERSION_ RESULT [0:13]	Conversion result. A digital value corresponding to the analog input voltage in a channel when the conversion command was initiated.

Table 18-36. Correspondence between Analog Voltages and Digital Values[1, 2]

	Voltage Level on Channel (V)	Corresponding 12-bit Conversion Result Returned by the ADC	16-bit Result Sent to RFIFOs (FMT=0) [3]	16-bit Result Sent to RFIFOs (FMT=1) [3]
Single-ended Conversions	5.12	0xFFF	0x3FFC	0x1FFC
	5.12 – LSB	0xFFF	0x3FFC	0x1FFC
	...	...	...	...
	2.56	0x800	0x2000	0x0000
	...	...	...	...
	1 LSB	0x001	0x0004	0xE004
	0	0x000	0x0000	0xE000
Differential Conversions	2.56	0xFFF	0x3FFC	0x1FFC
	2.56 – LSB	0xFFF	0x3FFC	0x1FFC
	...	...	...	...
	0	0x800	0x2000	0x0000
	...	...	...	...
	-2.56 + LSB	0x001	0x0004	0xE004
	-2.56	0x000	0x0000	0xE000

1 $V_{REF} = V_{RH} - V_{RL} = 5.12$ V. Resulting in one 12-bit count (LSB) = 1.25 mV.

2 The two's complement representation is used to express negative values.

3 Assuming an accurately calibrated ADC with an ideal gain and a zero offset.

18.4.1.2.2 Message Formats for External Device Operation

This section describes the command messages, data messages, and null messages formats used for external device operation.

Command Message Format for External Device Operation

Figure 18-31 describes the command message format for external device operation. Command message formats for on-chip operation and for external device operation share the same CFIFO header format. However, there are no limitations regarding the format an ADC Command used to communicate to an arbitrary external device. Only the upper bit of an ADC Command has a fixed format (BN field) to indicate the FIFO control unit/external device to which the command and the external command buffer is sent. The remaining 25 bits can be anything decodable by the external device. Only the ADC command portion of a command message is transferred to the external device.

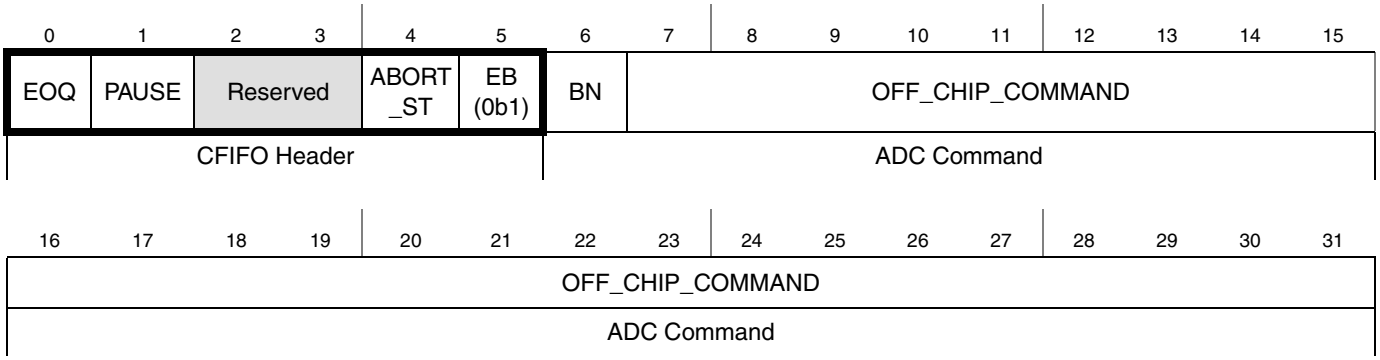


Figure 18-31. Command Message Format for External Device Operation

Table 18-37. On-Chip ADC Field Descriptions: External Device Operation

Field	Description
0 EOQ	End-of-queue. Asserted in the last command of a command queue to indicate to the eQADC that a scan of the queue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to 0. Depending on the CFIFO mode of operation, the CFIFO status changes upon the detection of an asserted EOQ bit on the last transferred command. See Section 18.4.3.4, “CFIFO Scan Trigger Modes,” for details. 0 Not the last entry of the command queue. 1 Last entry of the command queue. **Note:** If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
1 PAUSE	Pause bit. Allows software to create sub-queues within a command queue. When the eQADC completes the transfer of a command with an asserted pause bit, the CFIFO enters the WAITING FOR TRIGGER state. See Section 18.4.3.5.1, “CFIFO Operation Status,” for a description of the state transitions. The pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 0 Do not enter WAITING FOR TRIGGER state after transfer of the current command message. 1 Enter WAITING FOR TRIGGER state after transfer of the current command message. **Note:** If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
2–3	Reserved
4 ABORT _ST	ABORT serial transmission. Indicates whether to abort an on-going serial transmission. All CFIFOs can abort null message transmissions when triggered but only CFIFO0 can abort command transmissions of lower priority CFIFOs. For more on serial transmission aborts, see Section 18.4.3.2, “CFIFO Prioritization and Command Transfer.” 0 Do not abort current serial transmission. 1 Abort current serial transmission.
5 EB	External buffer. Always set this bit for messages sent to an external ADC. 0 Command is sent to an internal command buffer. 1 Command is sent to an external command buffer.
6 BN	See Section , “Conversion Command Message Format for On-Chip ADC Operation.”
7–31 OFF_CHIP_ COMMAND [0:24]	OFF-CHIP COMMAND Field. The OFF_CHIP_COMMAND field can be anything decodable by the external device. It is 25 bits long and it is transferred together with the BN bit to the external device when the CFIFO is triggered. See Section , “Conversion Command Message Format for On-Chip ADC Operation,” for a description of the command message used when interfacing with the on-chip ADCs.

Result Message Format for External Device Operation

Data is returned from the ADCs in the form of result messages. A result message is composed of an RFIFO header and an ADC result. The FIFO control unit decodes the information contained in the RFIFO header and sends the contents of the ADC result to the appropriate RFIFO. Only data stored on the ADC_RESULT field is stored in the RFIFOs result queues. The ADC result of any received message with a null data message tag is ignored. The format of a result message returned from the external device is shown in Figure 18-32. It is 26 bits long, and is composed of a MESSAGE_TAG field, information about the status of the buffers (BUSY fields), and result data. The BUSY fields are needed to inform the eQADC about when it is appropriate to transfer commands to the external command buffers.

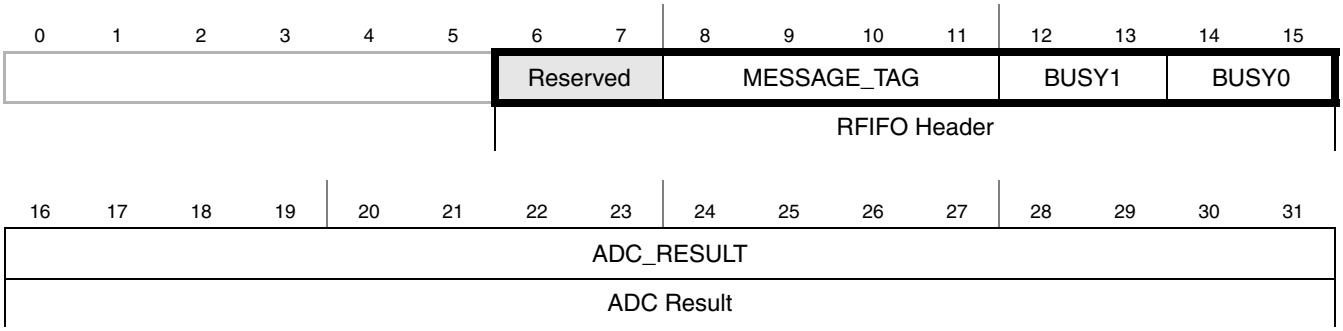


Figure 18-32. Result Message Format for External Device Operation

Table 18-38. Result Message Format for External Device Operation

Field	Description
6–7	Reserved
8–11 MESSAGE_TAG [0:3]	MESSAGE_TAG Field. See Section , “Conversion Command Message Format for On-Chip ADC Operation.”
12–13 BUSY1 [0:1]	BUSY1 status. The BUSY1 field indicates if the external device can receive more commands. Table 18-39 shows how these two bits are encoded. When an external device cannot accept any more commands, it must set BUSY1 to “Do not send commands” in the returning message. The BUSY1 field of values 0b10 and 0b10can be set by the external device to view the BUSY1 status of the external command buffers for debug. As an example, set the BUSY1 status to the number of entries in an external command buffer.
14–15 BUSY0 [0:1]	BUSY status. The BUSY fields indicate if the external device can receive more commands. Table 18-39 shows how these two bits are encoded. When an external device cannot accept any more new commands, it must set BUSY*n* to a value indicating “Do not send commands” in the returning message. The BUSY fields of values 0b10 and 0b10 can be freely encoded by the external device to allow visibility of the status of the external command buffers for debug. As an example, they could indicate the number of entries in an external command buffer.
16–31 ADC_ RESULT [0:15]	ADC RESULT Field. The result data received from the external device or on-chip ADC can be any value produced by the external device, such as the result of a conversion command, data requested via a read configuration command, or a time stamp value. The ADC_RESULT of any incoming message with a null message tag is ignored. When the MESSAGE_TAG is for an RFIFO, the eQADC extracts the 16-bit ADC_RESULT from the raw message and stores it into the appropriate RFIFO.

Table 18-39. Command BUFFER*n* BUSY Status[1]

BUSY*n*[0:1]	Meaning
0b00	Send available commands—command buffer is empty
0b01	Send available commands
0b10	Send available commands
0b11	Do not send commands

1 After reset, the eQADC always assumes that the external command buffers are full and cannot receive commands.

Null Message Format for External Device Operation

Null messages are only transferred through the serial interface to allow results and unsolicited control data, like the status of the external command buffers, to return when there are no more commands pending to transfer. Null messages are only transmitted when serial transmissions from the eQADC SSI are enabled (see ESSIE field in Section 18.3.2.1, “eQADC Module Configuration Register (EQADC_MCR),”), and when one of the following conditions apply:

- There are no triggered CFIFOs with commands bound for external command buffers.
- There are triggered CFIFOs with commands bound for external command buffers but the external buffers are full. The eQADC detected returning BUSY*n* fields indicating “Do not send commands.”

Figure 18-33 illustrates the null message send format. When the eQADC transfers a null message, it directly shifts out the 26-bit data content inside the Section 18.3.2.2, “eQADC Null Message Send Format Register (EQADC_NMSFR).” The register must be programmed with the null message send format of the external device.

Figure 18-34 illustrates the null message receive format. It has the same fields found in a result message with the exception that the ADC result is not used. See section “Result Message Format for External Device Operation,” for more information. The MESSAGE_TAG field must be set to the null message tag (0b1000). The eQADC does not store into an RFIFO any incoming message with a null message tag.

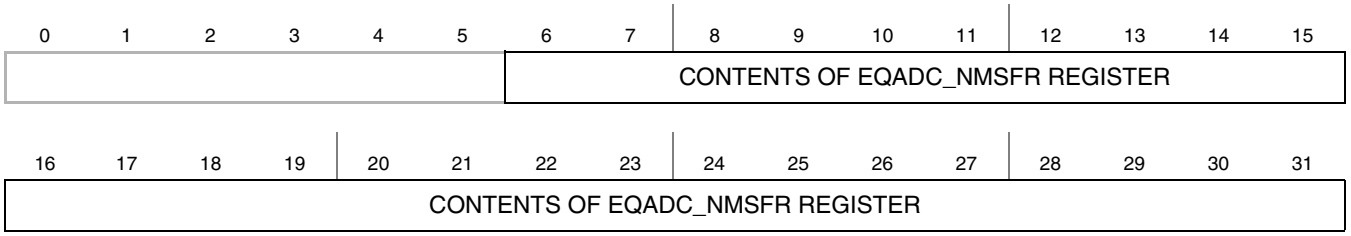


Figure 18-33. Null Message Send Format for External Device Operation

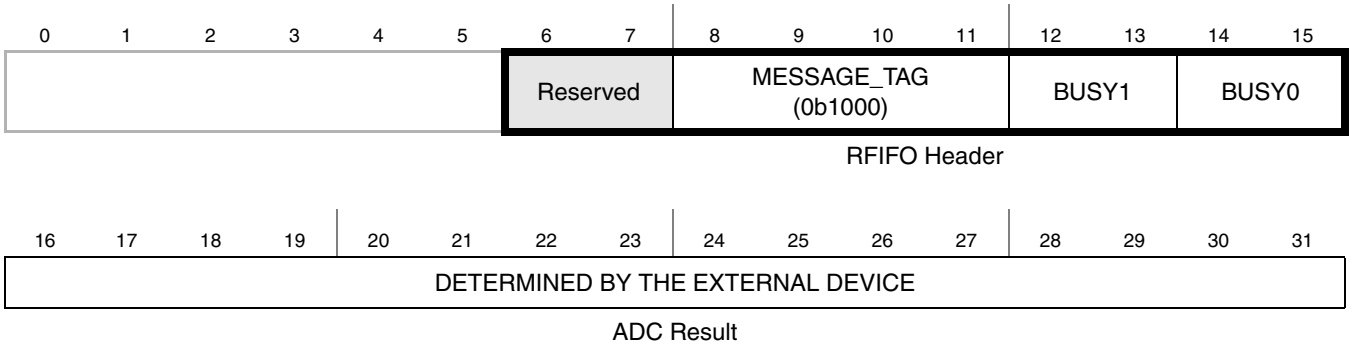


Figure 18-34. Null Message Receive Format for External Device Operation

Table 18-40. Null Message Receive Format for External Device Operation

Field	Description
6–7	Reserved
8–11 MESSAGE_TAG[0:3]	MESSAGE_TAG field. See Section , “Conversion Command Message Format for On-Chip ADC Operation.”
12–15 BUSY*n* [0:1]	BUSY status. See Section , “Result Message Format for External Device Operation.”
16–31	Determined by the external device.

18.4.2 Command and Result Queues

The command and result queues are actually part of the eQADC system although they are not hardware implemented inside the eQADC. Instead command and result queues are user-defined queues located in system memory. Each command queue entry is a 32-bit command message.The last entry of a command queue has the EOQ bit asserted to indicate that it is the last entry of the queue. The result queue entry is a 16-bit data item.

See Section 18.1.4, “Modes of Operation,” for a description of the message formats and their flow in eQADC.

See Section 18.5.5, “Command Queue and Result Queue Usage,” for examples of how command queues and result queues can be used.

18.4.3 eQADC Command FIFOs

18.4.3.1 CFIFO Basic Functionality

There are six prioritized CFIFOs located in the eQADC. Each CFIFO is four entries deep, and each CFIFO entry is 32 bits long. A CFIFO serves as a temporary storage location for the command messages stored in the command queues in system memory. When a CFIFO is not full, the eQADC sets the corresponding CFFF bit in Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn).” If

CFFE is asserted as in Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn),” the eQADC generates requests for more commands from a command queue. An interrupt request, served by the host CPU, is generated when CFFS is negated, and a eDMA request, served by the eDMA, is generated when CFFS is asserted. The host CPU or the eDMA respond to these requests by writing to the Section 18.3.2.4, “eQADC CFIFO Push Registers 0–5 (EQADC_CFPRn),” to fill the CFIFO.

NOTE

Only whole words must be written to EQADC_CFPR. Writing halfwords or bytes to EQADC_CFPR pushes the entire 32-bit CF_PUSH field into the corresponding CFIFO, but undefined data fills the areas of CF_PUSH that were not specifically designated as target locations for writing.

Figure 18-35 describes the important components in the CFIFO. Each CFIFO is implemented as a circular set of registers to avoid the need to move all entries at each push/pop operation. The push next data pointer points to the next available CFIFO location for storing data written into the eQADC command FIFO push register. The transfer next data pointer points to the next entry to be removed from CFIFO*n* when it completes a transfer. The CFIFO transfer counter control logic counts the number of entries in the CFIFO and generates eDMA or interrupt requests to fill the CFIFO. TNXTPTR in Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn),” indicates the index of the entry that is currently being addressed by the transfer next data pointer, and CFCTR, in the same register, provides the number of entries stored in the CFIFO.

Using TNXTPTR and CFCTR, the absolute addresses for the entries indicated by the transfer next data pointer and by the push next data pointer can be calculated using the following formulas:

```
Transfer Next Data Pointer Address = CFIFOn_BASE_ADDRESS + TNXTPTRn x 4
Push Next Data Pointer Address = CFIFOn_BASE_ADDRESS +
[(TNXTPTRn+CFCTRn) mod CFIFO_DEPTH] x 4
```

where

- *a mod b* returns the remainder of the division of *a* by *b*.
- CFIFO*n*_BASE_ADDRESS is the smallest memory mapped address allocated to a CFIFO*n* entry.
- CFIFO_DEPTH is the number of entries contained in a CFIFO - four in this implementation.

When CFS*n* in Section 18.3.2.11, “eQADC CFIFO Status Register EQADC_CFSR,” is in the TRIGGERED state, the eQADC generates the proper control signals for the transfer of the entry pointed by transfer next data pointer. CFUF*n* in Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn),” is set when a CFIFO*n* underflow event occurs. A CFIFO underflow occurs when the CFIFO is in the TRIGGERED state and it becomes empty. No commands are transferred from an underflowing CFIFO, and command transfers from lower priority CFIFOs are not blocked. CFIFO*n* is empty when the transfer next data pointer *n* equals the push next data pointer *n* and CFCTR*n* is 0. CFIFO*n* is full when the transfer next data pointer *n* equals the push next data pointer *n* and CFCTR*n* is not 0.

When the eQADC completes the transfer of an entry from CFIFO*n*: the transferred entry is popped from CFIFO*n*, the CFIFO counter CFCTR in the Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn),” is decremented by 1, and transfer next data pointer *n* is incremented by 1 (or wrapped around) to point to the next entry in the CFIFO. The transfer of entries bound for the on-chip

ADCs is considered completed when they are stored in the appropriate ADC command buffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed.

When the EQADC_CFPR*n* is written and CFIFO*n* is not full, the CFIFO counter CFCTR*n* is incremented by 1, and the push next data pointer *n* then is incremented by 1 (or wrapped around) to point to the next entry in the CFIFO.

When the EQADC_CFPR*n* is written but CFIFO*n* is full, the eQADC does not increment the counter value and does not overwrite any entry in CFIFO*n*.

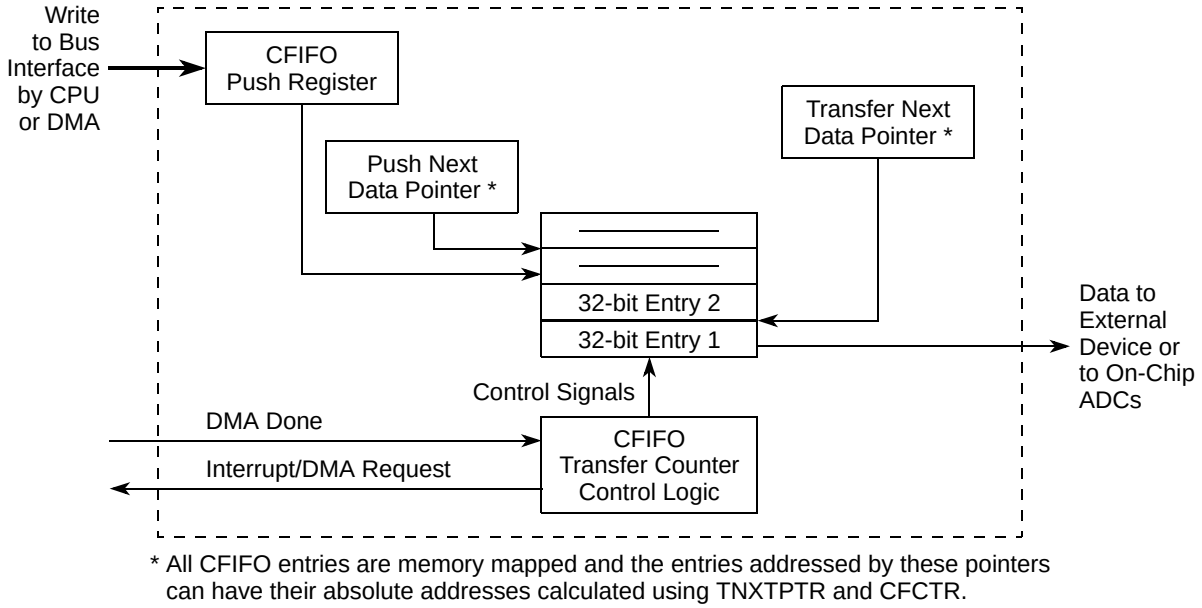


Figure 18-35. CFIFO Diagram

The detailed behavior of the push next data pointer and transfer next data pointer is described in the example shown in Figure 18-36 where a CFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four entries. In this example, CFIFO*n* with 16 entries is shown in sequence after pushing and transferring entries.

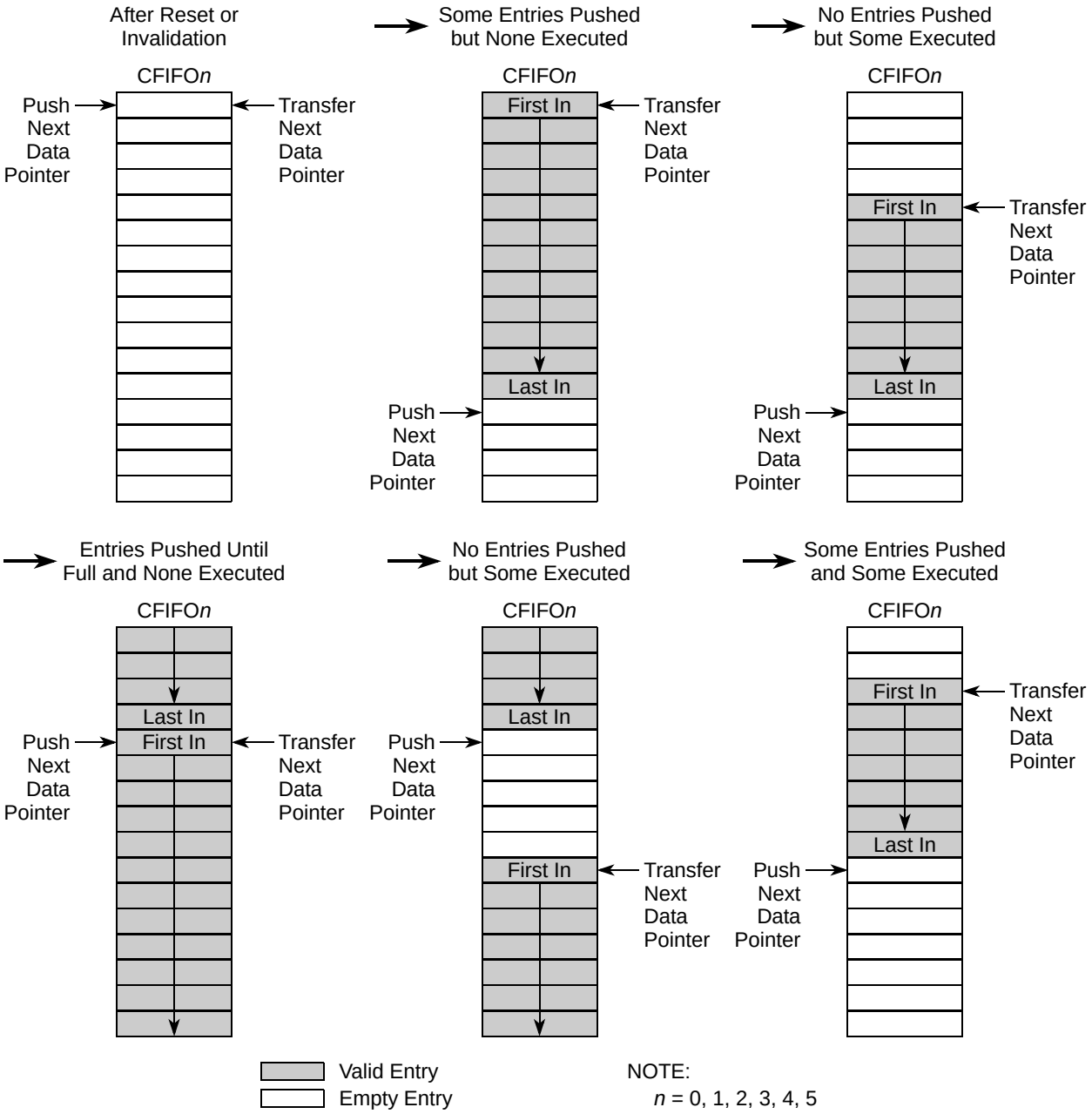


Figure 18-36. CFIFO Entry Pointer Example

18.4.3.2 CFIFO Prioritization and Command Transfer

The CFIFO priority is fixed according to the CFIFO number. A CFIFO with a smaller number has a higher priority. When commands from different CFIFOs are sent to the same destination (such as the same on-chip ADC), the higher priority CFIFO is always served first. A triggered, not-underflowing CFIFO starts to transfer commands when the following occur:

- Its commands are bound for an internal command buffer that is not full, and it is the highest priority triggered CFIFO sending commands to that buffer.
- Its commands are bound for an external command buffer that is not full, and it is the highest priority triggered CFIFO sending commands to an external buffer that is not full.

A triggered CFIFO with commands bound for a certain command buffer consecutively transfers its commands to the buffer until one of the following occurs:

- An asserted end of queue bit is reached.
- An asserted pause bit is encountered and the CFIFO is configured for edge trigger mode.
- CFIFO is configured for level trigger mode and a closed gate is detected.
- In case its commands are bound for an internal command buffer, a higher priority CFIFO that uses the same internal buffer is triggered.
- In case its commands are bound for an external command buffer, a higher priority CFIFO that uses an external buffer is triggered.

The prioritization logic of the eQADC, depicted in Figure 18-37, is composed of three independent submodules: one that prioritizes CFIFOs with commands bound for ADC0, another that prioritizes CFIFOs with commands for ADC1, and a last one that prioritizes CFIFOs with commands for external command buffer 2 and buffer 3. As these three submodules are independent, simultaneous commands to ADC0, to ADC1, and to eQADC SSI transmit buffer are allowed. The hardware identifies the destination of a command by decoding the EB and BN bits in the command message (see Section 18.4.1.2, “Message Format in eQADC,” for details).

NOTE

Triggered but empty CFIFOs, underflowing CFIFOs, are not considered for prioritization. No data from these CFIFOs is sent to the on-chip ADCs or the external command buffers, and lower priority CFIFOs are not stopped from transferring commands.

Whenever ADC0 is able to receive new commands, the prioritization submodule selects the highest-priority triggered CFIFO with a command bound for ADC0, and sends it to the ADC. In case ADC0 is able to receive new entries but there are no triggered CFIFOs with commands bound for it, nothing is sent. The submodule prioritizing ADC1 usage behaves in the same way.

When the eQADC SSI is enabled and ready to start serial transmissions, the submodule prioritizing eQADC SSI usage writes command or null messages into the eQADC SSI transmit buffer. Data written to the eQADC SSI transmit buffer is subsequently transmitted to the external device through the eQADC SSI link. The submodule writes commands to the eQADC SSI transmit buffer when there are triggered CFIFOs with commands bound for not-full external command buffers. The command written to the transmit buffer belongs to the highest priority CFIFO sending commands to an external buffer that is not full. This implies

that a lower priority CFIFO can have its commands sent if a higher priority CFIFO cannot send its commands due to a full command buffer. The submodule writes null messages to the eQADC SSI transmit buffer when there are no triggered CFIFOs with commands bound for external command buffers, or when there are triggered CFIFOs with commands bound for external buffers but the external buffers are full. The eQADC monitors the status of the external buffers by decoding the BUSY fields of the incoming result messages from the external device (see Section , "Result Message Format for External Device Operation," for details).

NOTE

When a higher priority CFIFO cannot send commands due to a full external command buffer, a lower priority CFIFO is served. When the higher priority CFIFO is ready to send commands, an interrupt to the command transfers from the lower priority CFIFO can result in CFIFO incoherence. Whether the lower priority CFIFO becomes non-coherent depends on the following:

- Rate at which commands on the external ADCs are executed
- Rate at which commands are transmitted to the external command buffers
- Depth of the buffers

After a serial transmission starts, the submodule monitors triggered CFIFOs and manages the abort of the serial transmissions. If a null message is transmitted, the serial transmission is aborted when all of the following conditions are met:

- A not-underflowing CFIFO in the TRIGGERED state has commands bound for an external command buffer that is not full, and it is the highest priority CFIFO sending commands to an external buffer that is not full.
- The ABORT_ST bit of the command to be transmitted is asserted.
- The 26th bit of the currently transmitting null message is not shifted out.

The command from the CFIFO is then written into eQADC SSI transmit buffer, allowing for a new serial transmission to initiate.

In case a command is being transmitted, the serial transmission is aborted when all following conditions are met:

- CFIFO0 is in the TRIGGERED state, is not underflowing, and its current command is bound for an external command buffer that is not full.
- The ABORT_ST bit of the command to be transmitted is asserted.
- The 26th bit of the currently transmitting command has not being shifted out.

The command from CFIFO0 is then written into eQADC SSI transmit buffer, allowing for a new serial transmission to initiate.

NOTE

The aborted command is not popped from the pre-empted CFIFO, but is retransmitted as soon as it's the highest priority CFIFO sending commands to an unfilled external command buffer.

After a serial transmission is completed, the eQADC prioritizes the CFIFOs and schedules a command or a null message to be sent in the next serial transmission. After the data for the next transmission has been defined and scheduled, the eQADC can, under certain conditions, stretch the $\overline{\text{SDS}}$ negation time to allow the schedule of new data for that transmission. This occurs when the eQADC acknowledges that the status of a higher-priority CFIFO has changed to the TRIGGERED state and attempts to schedule that CFIFO command before $\overline{\text{SDS}}$ is asserted. Only commands of CFIFOs that have the ABORT_ST bit asserted can be scheduled in this manner. Under such conditions:

1. A CFIFO0 command is scheduled for the next transmission independently of the type of data that was previously scheduled. The time during which $\overline{\text{SDS}}$ is negated is stretched to allow the eQADC to load the CFIFO0 command and start its transmission.
2. CFIFO1-5 commands are only scheduled for the next transmission if the previously scheduled data was a null message. The time during which $\overline{\text{SDS}}$ is negated is stretched to allow the eQADC to load that command and start its transmission. However, if the previously scheduled data was a command, no rescheduling occurs and the next transmission starts without delays.

If a CFIFO becomes triggered while $\overline{\text{SDS}}$ is negated, but the eQADC only attempts to reschedule that CFIFO command after $\overline{\text{SDS}}$ is asserted, then the current transmission is aborted depending on if the conditions for that are met or not.

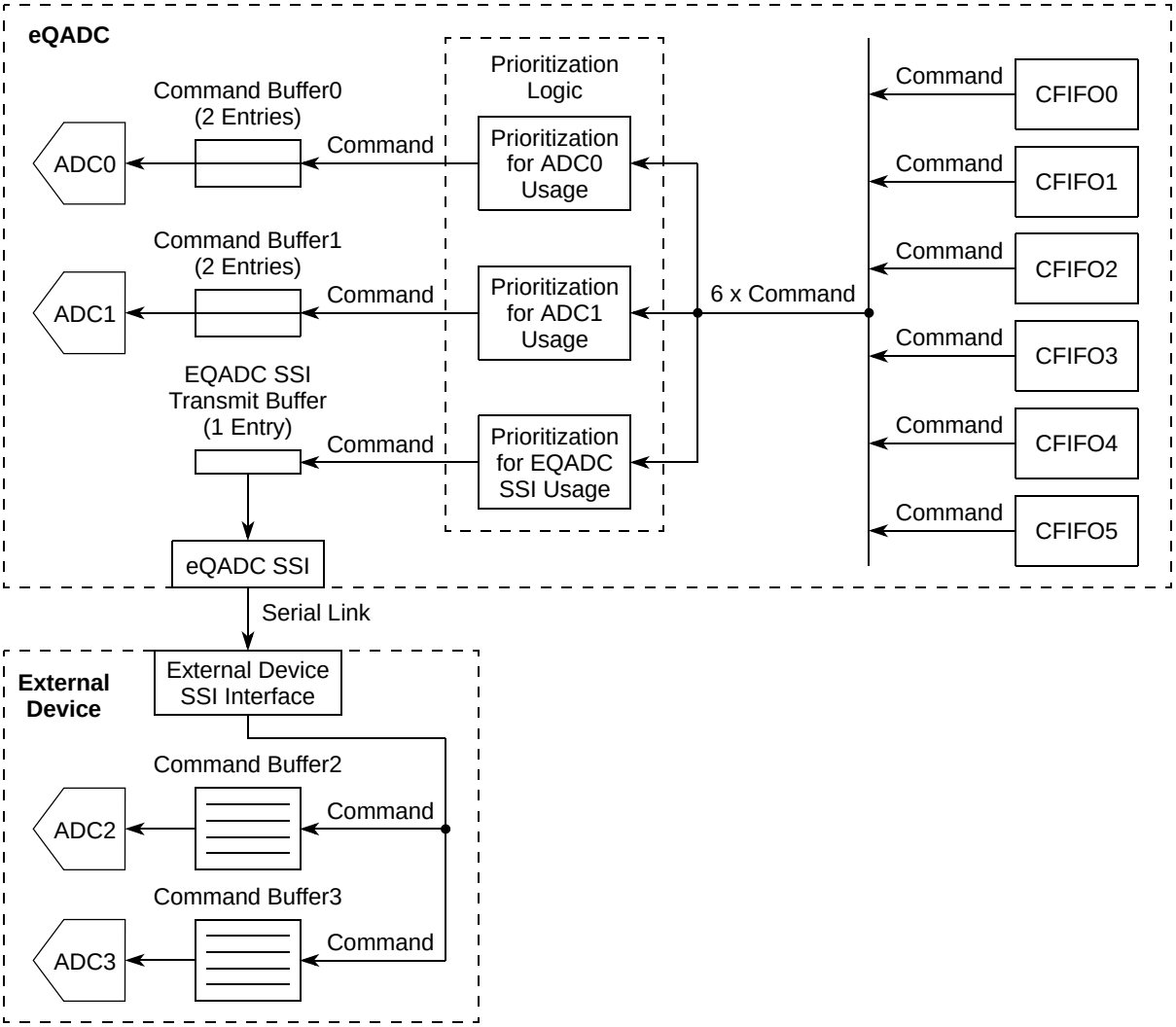


Figure 18-37. CFIFO Prioritization Logic

18.4.3.3 External Trigger from eTPU or eMIOS Channels

The six eQADC external trigger inputs can be connected to either an external pin (GPIO206, or GPIO207), an eTPU channel, or an eMIOS channel. The input source for each eQADC external trigger is individually specified in the eQADC trigger input select register (SIU_ETISR) in the SIU block.

The eQADC trigger numbers specified by SIU_ETISR[TSEL(0–5)] correspond to CFIFO numbers 0–5. To calculate the CFIFO number that each trigger is connected to, divide the eDMA channel number by 2.

A complete description of the eTPU and eMIOS trigger function and configuration is found in Section 6.5.5.1, “eQADC External Trigger Input Multiplexing.”

18.4.3.4 CFIFO Scan Trigger Modes

The eQADC supports two different scan modes, single-scan and continuous-scan. See Table 18-41 for a summary of these two scan modes. When a CFIFO is triggered, the eQADC scan mode determines whether the eQADC stops command transfers from a CFIFO, and waits for software intervention to rearm the CFIFO to detect new trigger events, upon detection of an asserted EOQ bit in the last transfer. See Section 18.4.1.2, "Message Format in eQADC," for details about command formats.

CFIFOs can be configured in single-scan or continuous-scan mode. When a CFIFO is configured in single-scan mode, the eQADC scans the command queue one time. The eQADC stops future command transfers from the triggered CFIFO after detecting the EOQ bit set in the last transfer. After a EOQ bit is detected, software involvement is required to rearm the CFIFO so that it can detect new trigger events.

When a CFIFO is configured for continuous-scan mode, no software involvement is necessary to rearm the CFIFO to detect new trigger events after an asserted EOQ is detected. In continuous-scan mode the whole command queue is scanned multiple times.

The eQADC also supports different triggering mechanisms for each scan mode. The eQADC does not transfer commands from a CFIFO until the CFIFO is triggered. The combination of scan modes and triggering mechanisms allows the support of different requirements for scanning input channels. The scan mode and trigger mechanism are configured by programming the MODE*n* field in Section 18.3.2.6, "eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)."

Enabled CFIFOs can be triggered by software or external trigger events. The elapsed time from detecting a trigger to transferring a command is a function of clock frequency, trigger synchronization, trigger filtering, programmable trigger events, command transfer, CFIFO prioritization, ADC availability, etc. Fast and predictable transfers can be achieved by ensuring that the CFIFO is not underflowing and that the target ADC can accept commands when the CFIFO is triggered.

18.4.3.4.1 Disabled Mode

The MODE*n* field in Section 18.3.2.6, "eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)," for all of the CFIFOs can be changed from any other mode to disabled at any time. No trigger event can initiate command transfers from a CFIFO which has its MODE field programmed to disabled.

NOTE

If MODE*n* is not disabled, it must not be changed to any other mode besides disabled. If MODE*n* is disabled and the CFIFO status is IDLE, MODE*n* can be changed to any other mode.

If MODE*n* is changed to disabled:

- The CFIFO execution status changes to IDLE. The timing of this change depends on whether a command is being transferred or not:
  - When no command transfer is in progress, the eQADC switches the CFIFO to IDLE status immediately.
  - When a command transfer to an on-chip ADC is in progress, the eQADC completes the transfer, updates TC_CF, and switches CFIFO status to IDLE. Command transfers to the internal ADCs are considered completed when a command is written to the relevant buffer.
  - When a command transfer to an external command buffer is in progress, the eQADC aborts the transfer and switches CFIFO status to IDLE. If the eQADC cannot abort the transfer, that is when the 26th bit of the serial message has being already shifted out, the eQADC completes the transfer, updates TC_CF and then switches CFIFO status to IDLE.
- The CFIFOs are not invalidated automatically. The CFIFO still can be invalidated by writing a 1 to the CFINV*n* bit (see Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”). Certify that CFS has changed to IDLE before setting CFINV*n*.
- The TC_CF*n* value also is not reset automatically, but it can be reset by writing 0 to it.
- The EQADC_FISRn[SSS] bit (see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) is negated. The SSS bit can be set even if a 1 is written to the EQADC_CFCR[SSE] bit (see Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”) in the same write that the MODE*n* field is changed to a value other than disabled.
- The trigger detection hardware is reset. If MODE*n* is changed from disabled to an edge trigger mode, a new edge, matching that edge trigger mode, is needed to trigger the command transfers from the CFIFO.

NOTE

CFIFO fill requests, generated when the CFFF asserts, are not automatically halted when MODE*n* is changed to disabled. CFIFO fill requests are still generated until EQADC_IDCRn[CFFE] bit is cleared. See Section Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn).”

18.4.3.4.2 Single-Scan Mode

In single-scan mode, a single pass through a sequence of command messages in command queue you defined is performed.

In single-scan software trigger mode, the CFIFO is triggered by an asserted single-scan status bit, EQADC_FISRn[SSS] (see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”). The SSS bit is set by writing 1 to the single-scan enable bit, EQADC_CFCRn[SSE] (see Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”).

In single-scan edge or level trigger mode, the respective triggers are only detected when the SSS bit is asserted. When the SSS bit is negated, all trigger events for that CFIFO are ignored. Writing a 1 to the SSE bit can be done during the same write cycle that the CFIFO operation mode is configured.

Only the eQADC can clear the SSS bit. After SSS is asserted, it remains asserted until the eQADC completes the command queue scan, or the CFIFO operation mode, EQADC_CFCRn[MODE*n*] (see Section 18.3.2.6) is changed to disabled. The SSS*n* bit is negated while MODE*n* is disabled.

Single-Scan Software Trigger

When single-scan software trigger mode is selected, the CFIFO is triggered by an asserted SSS bit. The SSS bit is asserted by writing 1 to the SSE bit. Writing to SSE while SSS is already asserted does not have any effect on the state of the SSS bit, nor does it cause a trigger overrun event.

The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC or an external command buffer that is not full. When an asserted EOQ bit is encountered, the eQADC clears the SSS bit. Setting the SSS bit is required for the eQADC to start the next scan of the queue.

The pause bit has no effect in single-scan software trigger mode.

Single-Scan Edge Trigger

When SSS is asserted and an edge triggered mode is selected for a CFIFO, an appropriate edge on the associated trigger signal causes the CFIFO to become triggered. For example, if rising-edge trigger mode is selected, the CFIFO becomes triggered when a rising edge is sensed on the trigger signal. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC, or an external command buffer that is not full.

When an asserted EOQ bit is encountered, the eQADC clears SSS and stops command transfers from the CFIFO. An asserted SSS bit and a subsequent edge trigger event are required to start the next scan for the CFIFO. When an asserted pause bit is encountered, the eQADC stops command transfers from the CFIFO, but SSS remains set. Another edge trigger event is required for command transfers to continue. A trigger overrun happens when the CFIFO is in a TRIGGERED state and an edge trigger event is detected.

Single-Scan Level Trigger

When SSS is asserted and a level gated trigger mode is selected, the input level on the associated trigger signal puts the CFIFO in a TRIGGERED state. When the CFIFO is set to high-level gated trigger mode, a high level signal opens the gate, and a low level closes the gate. When the CFIFO is set to low-level gated trigger mode, a low level signal opens the gate, and a high level closes the gate. If the corresponding level is already present, setting the SSS bit triggers the CFIFO. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC or an external command buffer that is not full.

The eQADC clears the SSS bit and stops transferring commands from a triggered CFIFO when an asserted EOQ bit is encountered or when CFIFO status changes from triggered due to the detection of a closed gate. If a closed gate is detected while no command transfers are taking place and the CFIFO status is triggered, the CFIFO status is immediately changed to IDLE, the SSS bit is negated, and the PF flag is asserted. If a closed gate is detected during the serial transmission of a command to the external device, it has no affect on the CFIFO status until the transmission completes. After the transmission is completed, the TC_CF counter is updated, the SSS bit is negated, the PF flag is asserted, and the CFIFO status is changed to IDLE.

An asserted SSS bit and a level trigger are required to restart the CFIFO. Command transfers restart from the point they have stopped.

If the gate closes and opens during the same serial transmission of a command to the external device, it has no effect on the CFIFO status or on the PF flag, but the TORF flag asserts as shown in Figure 18-39. Therefore, closing the gate for a period less than a serial transmission time interval does not guarantee that the closure affects command transfers from a CFIFO.

The pause bit has no effect in single-scan level trigger mode.

18.4.3.4.3 Continuous-Scan Mode

In continuous-scan mode, multiple passes looping through a sequence of command messages in a command queue are executed. When a CFIFO is programmed for a continuous-scan mode, the EQADC_CFCRn[SSE] (see Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”) does not have any effect.

Continuous-Scan Software Trigger

When a CFIFO is programmed to continuous-scan software trigger mode, the CFIFO is triggered immediately. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC or an external command buffer that is not full. When a CFIFO is programmed to run in continuous-scan software trigger mode, the eQADC does not halt transfers from the CFIFO until the CFIFO operation mode is modified to disabled or a higher priority CFIFO preempts it. Although command transfers do not stop upon detection of an asserted EOQ bit, the EOQF is set and, if enabled, an EOQ interrupt request is generated.

The pause bit has no effect in continuous-scan software trigger mode.

Continuous-Scan Level Trigger

When high or low level gated trigger mode is selected, the input level on the associated trigger signal places the CFIFO in a TRIGGERED state. When high-level gated trigger is selected, a high-level signal opens the gate, and a low level closes the gate. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC or an external buffer that is not full. Although command transfers do not stop upon detection of an asserted EOQ bit at the end of a command transfer, the EOQF is asserted and, if enabled, an EOQ interrupt request is generated.

The eQADC stops transferring commands from a triggered CFIFO when CFIFO status changes from triggered due to the detection of a closed gate. If a closed gate is detected while no command transfers are taking place and the CFIFO status is TRIGGERED, the CFIFO status is immediately changed to waiting for trigger and the PF flag is asserted. If a closed gate is detected during the serial transmission of a command to the external device, it has no effect on the CFIFO status until the transmission completes. After the transmission is completed, the TC_CF counter is updated, the PF flag is asserted, and the CFIFO status is changed to waiting for trigger. Command transfers restart as the gate opens.

If the gate closes and opens during the same serial transmission of a command to the external device, it has no affect on the CFIFO status or on the PF flag, but the TORF flag asserts as shown in Figure 18-39.

Therefore, closing the gate for a period less than a serial transmission time interval does not guarantee that command transfers from a CFIFO are closed.

The pause bit has no effect in continuous-scan level trigger mode.

18.4.3.4.4 CFIFO Scan Trigger Mode Start/Stop Summary

Table 18-41 summarizes the start and stop conditions of command transfers from CFIFOs for all of the single-scan and continuous-scan trigger modes.

Table 18-41. CFIFO Scan Trigger Mode—Command Transfer Start/Stop Summary

Trigger Mode	Requires Asserted SSS to Recognize Trigger Events?	Command Transfer Start/Restart Condition	Stop on asserted EOQ bit[1]?	Stop on asserted Pause bit[2]?	Other Command Transfer Stop Condition[3] [4]
Single Scan Software	Not Applicable	Asserted SSS bit.	Yes	No	None.
Single Scan Edge	Yes	A corresponding edge occurs when the SSS bit is asserted.	Yes	Yes	None.
Single Scan Level	Yes	Gate is opened when the SSS bit is asserted.	Yes	No	The eQADC also stops transfers from the CFIFO when CFIFO status changes from triggered due to the detection of a closed gate.[5]
Continuous Scan Software	No	CFIFO starts automatically after being configured into this mode.	No	No	None.
Continuous Scan Edge	No	A corresponding edge occurs.	Yes	Yes	None.
Continuous Scan Level	No	Gate is opened.	No	No	The eQADC also stops transfers from the CFIFO when CFIFO status changes from triggered due to the detection of a closed gate.[5]

1 See Section 18.4.3.5.2, “Command Queue Completion Status,” for more information on EOQ.

2 See Section 18.4.3.5.3, “Pause Status,” for more information on pause.

3 The eQADC always stops command transfers from a CFIFO when the CFIFO operation mode is disabled.

4 The eQADC always stops command transfers from a CFIFO when a higher priority CFIFO is triggered. See Section 18.4.3.2, “CFIFO Prioritization and Command Transfer,” for information on CFIFO priority.

5 If a closed gate is detected while no command transfers are taking place, it has an immediate effect on the CFIFO status. If a closed gate is detected during the serial transmission of a command to the external device, it has no effect on the CFIFO status until the transmission completes.

18.4.3.5 CFIFO and Trigger Status

18.4.3.5.1 CFIFO Operation Status

Each CFIFO has its own CFIFO status field. CFIFO status (CFS) can be read from EQADC_CFSSR (see Section 18.3.2.11, “eQADC CFIFO Status Register EQADC_CFSR.” Figure 18-38 and Table 18-42 indicate the CFIFO status switching condition. See Table 18-15 for the meaning of each CFIFO operation status. The last CFIFO to transfer a command to an on-chip ADC can be read from the LCFT*n* (*n*=0,1) fields (see Section 18.3.2.10, “eQADC CFIFO Status Snapshot Registers 0–2.” The last CFIFO to transfer a command to a specific external command buffer can be identified by reading the EQADC_CFSSR*n*[LCFTSSI] and EQADC_CFSSR*n*[ENI] fields (see Section 18.3.2.10, “eQADC CFIFO Status Snapshot Registers 0–2.”

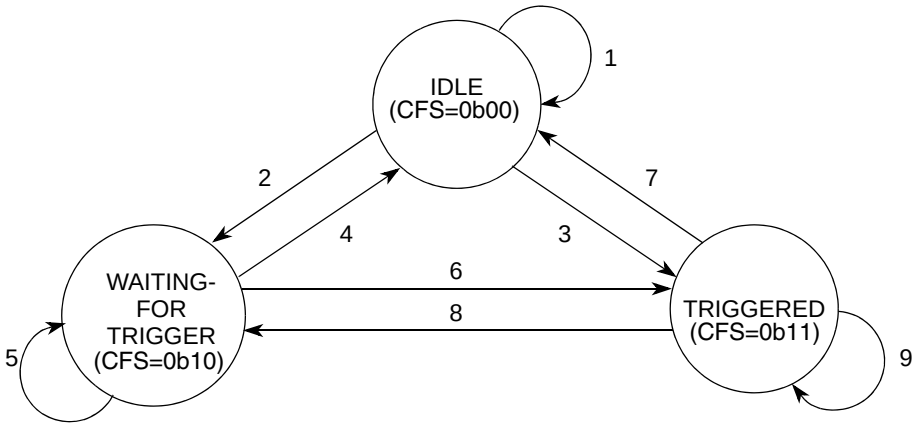


Figure 18-38. State Machine of CFIFO Status

Table 18-42. Command FIFO Status Switching Condition

No.	From Current CFIFO Status (CFS)	To New CFIFO Status (CFS)	Status Switching Condition
1	IDLE (00)	IDLE (0b00)	• CFIFO mode is programmed to disabled, OR • CFIFO mode is programmed to single-scan edge or level trigger mode and SSS is negated.
2		WAITING FOR TRIGGER (0b10)	• CFIFO mode is programmed to continuous-scan edge or level trigger mode, OR • CFIFO mode is programmed to single-scan edge or level trigger mode and SSS is asserted, OR • CFIFO mode is programmed to single-scan software trigger mode.
3		TRIGGERED (0b11)	• CFIFO mode is programmed to continuous-scan software trigger mode

Table 18-42. Command FIFO Status Switching Condition (continued)

No.	From Current CFIFO Status (CFS)	To New CFIFO Status (CFS)	Status Switching Condition
4	WAITING FOR TRIGGER (10)	IDLE (0b00)	• CFIFO mode is modified to disabled mode.
5		WAITING FOR TRIGGER (0b10)	• No trigger occurred.
6		TRIGGERED (0b11)	• Appropriate edge or level trigger occurred, OR • CFIFO mode is programmed to single-scan software trigger mode and SSS bit is asserted.
7	TRIGGERED (11)	IDLE (0b00)	• CFIFO in single-scan mode, eQADC detects the EOQ bit asserted at end of command transfer, and CFIFO mode is not modified to disabled, OR • CFIFO, in single-scan level trigger mode, and the gate closes while no commands are being transferred from the CFIFO, and CFIFO mode is not modified to disabled, OR • CFIFO, in single-scan level trigger mode, and eQADC detects a closed gated at end of command transfer, and CFIFO mode is not modified to disabled, OR • CFIFO mode is modified to disabled mode and CFIFO was not transferring commands. • CFIFO mode is modified to disabled mode while CFIFO was transferring commands, and CFIFO completes or aborts the transfer.
8		WAITING FOR TRIGGER (0b10)	• CFIFO in single or continuous-scan edge trigger mode, eQADC detects the pause bit asserted at the end of command transfer, the EOQ bit in the same command is negated, and CFIFO mode is not modified to disabled, OR • CFIFO in continuous-scan edge trigger mode, eQADC detects the EOQ bit asserted at the end of command transfer, and CFIFO mode is not modified to disabled, OR • CFIFO, in continuous-scan level trigger mode, and the gate closes while no commands are being transferred from the CFIFO, and CFIFO mode is not modified to disabled, OR • CFIFO, in continuous-scan level trigger mode, and eQADC detects a closed gated at end of command transfer, and CFIFO mode is not modified to disabled.
9		TRIGGERED (0b11)	• No event to switch to IDLE or WAITING FOR TRIGGER status has happened.

18.4.3.5.2 Command Queue Completion Status

The end of queue flag, EQADC_FISRn[EOQF] (see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) is asserted when the eQADC completes the transfer of a CFIFO entry with an asserted EOQ bit. Software sets the EOQ bit in the last command message of a user-defined command queue to indicate that this entry is the end of the queue. See Section 18.4.1.2, “Message Format in eQADC,” for information on command message formats. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate command buffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed.

The command with a EOQ bit asserted is valid and is transferred. When EQADC_CFCRn[EOQIE] (see Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”) and EQADC_FISRn[EOQF] are asserted, the eQADC generates an end of queue interrupt request.

In single-scan modes, command transfers from the corresponding CFIFO cease when the eQADC completes the transfer of a entry with an asserted EOQ. Software involvement is required to rearm the CFIFO so that it can detect new trigger events.

NOTE

An asserted EOQF*n* only implies that the eQADC has finished transferring a command with an asserted EOQ bit from CFIFO*n*. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

18.4.3.5.3 Pause Status

In edge trigger mode, when the eQADC completes the transfer of a CFIFO entry with an asserted pause bit, the eQADC stops future command transfers from the CFIFO and sets EQADC_FISRn[PF]. The eQADC ignores the pause bit in command messages in any software level trigger mode. The eQADC sets the PF flag only in single or continuous-scan edge trigger mode when the pause bit set. When the PF flag is set for a CFIFO in single-scan edge trigger mode, the EQADC_FISRn[SSS] bit is not cleared.

See the following sections for more information:

Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”

Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”

Section 18.4.1.2, “Message Format in eQADC,” for information on command message formats.

In level trigger mode, the PF flag designates that a CFIFO*n* is in the TRIGGERED status. The PF*n* bit is set when a closed gate is detected, triggering the CFIFO status change. The pause flag interrupt routine can be used to verify if a complete scan of the command queue was performed. If a closed gate is detected while no command transfers are taking place, it has an immediate effect on the CFIFO status. If a closed gate is detected during the serial transmission of a command to the external device, it has no effect on the CFIFO status until the transmission completes.

When EQADC_CFCR[PIE] and EQADC_FISRn[PF] are asserted, the eQADC generates a pause interrupt request. See Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)” for more information.

NOTE

In edge-trigger mode, an asserted PF*n* only implies that the eQADC finished transferring a command with an asserted pause bit from CFIFO*n*. It does not imply that result data for the current command and for all previously transferred commands has been returned to the RFIFO.

NOTE

In software or level trigger mode, when the eQADC completes the transfer of an entry from CFIFO*n* with an asserted pause bit, PF*n* is not set and the command transfers continues without pausing.

18.4.3.5.4 Trigger Overrun Status

When a CFIFO is configured for edge or level trigger mode and is in a TRIGGERED state, an additional trigger event for the same CFIFO causes a trigger overrun:

1. The trigger overrun bit for the CFIFO is set (EQADC_FISRn[TORF*n*] = 1)
2. The EQADC_CFCRn[TORIE] and EQADC_FISRn[TORF] assert
3. The eQADC generates a trigger overrun interrupt request.

See the following sections for more information:

Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”

Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”

For CFIFOs configured for level trigger mode, a trigger overrun event is detected only when the gate closes and reopens during a single serial command transmission as shown in Figure 18-39.

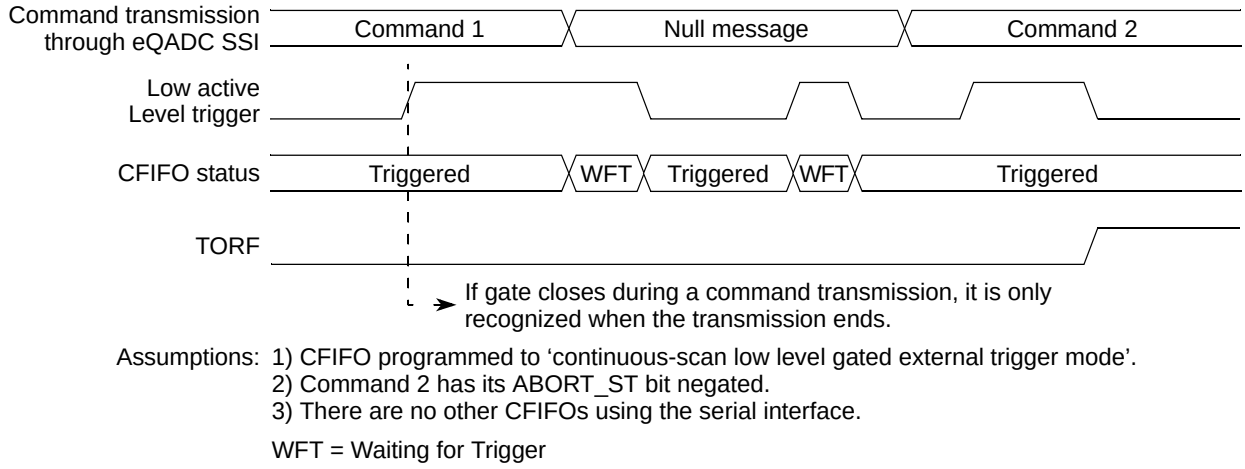


Figure 18-39. Trigger Overrun on Level Trigger Mode CFIFOs

NOTE

The trigger overrun flag is not set for CFIFOs configured for software trigger mode.

18.4.3.5.5 Command Sequence Non-Coherency Detection

The eQADC provides a mechanism to indicate if a command sequence has been completely executed without interruptions. A command sequence is defined as a group of consecutive commands bound for the same ADC and it is expected to be executed without interruptions. A command sequence is coherent if its commands are executed in order without interruptions. Because commands are stored in the ADC's command buffers before being executed in the eQADC, a command sequence is coherent if, while it is transferring commands to an on-chip ADC command buffer, the buffer is only fed with commands from that sequence without ever becoming empty.

A command sequence starts when:

- A CFIFO in TRIGGERED state transfers its first command to an on-chip ADC.
- The CFIFO is constantly transferring commands and the previous command sequence ended.
- The CFIFO resumes command transfers after being interrupted.

And a command sequence ended when:

- An asserted EOQ bit is detected on the last transferred command.
- CFIFO is in edge-trigger mode and asserted pause bit is detected on the last transferred command.
- The ADC to which the next command is sent is different from the ADC command last transferred.

Figure 18-40 shows examples of how the eQADC would detect command sequences when transferring commands from a CFIFO. The smallest possible command sequence can have a single command as shown in example 3 of Figure 18-40.

User Command Queue with Two Command Sequences

1	CF5_ADC1_CM0
2	CF5_ADC1_CM1
3	CF5_ADC1_CM2
4	CF5_ADC1_CM3(Pause=1)
5	CF5_ADC1_CM4
6	CF5_ADC1_CM5
7	CF5_ADC1_CM6(EOQ=1)

Example 1

Assuming that these commands are transferred by a CFIFO configured for edge trigger mode and the command transfers are never interrupted, the eQADC checks for non-coherency of two command sequences: one formed by commands 0, 1, 2, 3, and the other by commands 4, 5, 6.

User Command Queue with Three Command Sequences

1	CF5_ADC1_CM0
2	CF5_ADC1_CM1
3	CF5_ADC1_CM2
4	CF5_ADC0_CM3
5	CF5_ADC0_CM4
6	CF5_ADC1_CM5
7	CF5_ADC1_CM6(EOQ=1)

Example 2

Assuming that command transfers from the CFIFO are never interrupted, the eQADC checks for non-coherency of three command sequences. The first being formed by commands 0, 1, 2, the second by commands 3, 4 and the third by commands 5, 6. Even when the commands of this queue are transferred through a CFIFO in continuous-scan mode, the first three commands and the last two commands of this command queue still constitute two distinct command sequences, although they are all bound for the same ADC, because an asserted EOQ ends a command sequence.

User Command Queue with a Seven Command Sequence

1	CF5_ADC1_CM0
2	CF5_ADC2_CM1
3	CF5_ADC3_CM2
4	CF5_ADC1_CM3
5	CF5_ADC0_CM4
6	CF5_ADC2_CM5
7	CF5_ADC1_CM6(EOQ=1)

Example 3

The eQADC checks for non-coherency of seven command sequences, all containing a single command, but NCF is never set.

CF*n*_ADCa_CMD*n* – Command *n* in CFIFO*n* bound for ADCa (ADC3 and ADC4 are external devices associated with external command buffers 2 and 3).

Figure 18-40. Command Sequence Examples

The NCF flag is used to indicate command sequence non-coherency. When the NCF*n* flag is asserted, it indicates that the command sequence being transferred through CFIFO*n* became non-coherent. The NCF flag only becomes asserted for CFIFOs in a TRIGGERED state.

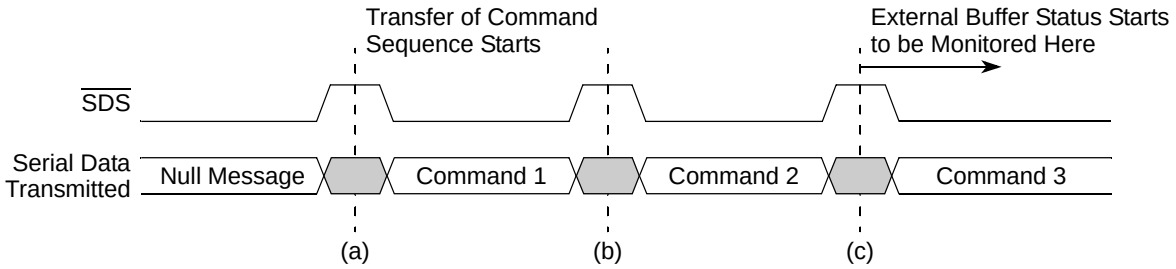
A command sequence is non-coherent when, after transferring the first command of a sequence from a CFIFO to a buffer, it cannot successively send all the other commands of the sequence before any of the following conditions are true:

- The CFIFO through which commands are being transferred is pre-empted by a higher priority CFIFO which sends commands to the same ADC. The NCF flag becomes asserted immediately after the first command transfer from the pre-empting CFIFO, that is the higher priority CFIFO, to the ADC in use is completed. See Figure 18-42.
- The external command buffer in use becomes empty. (Only the fullness of external buffers is monitored because the fill rate for internal ADC buffers is many times faster than the drain rate,

and each has a dedicated priority engine.) This case happens when different CFIFOs attempt to use different external command buffers and the higher priority CFIFO bars the lower priority one from sending new commands to its buffer—see Figure 18-43. An external command buffer is considered empty when the corresponding BUSY field in the last result message received from external device is encoded as “Send available commands - buffer is empty”. See Section , “Result Message Format for External Device Operation.” The NCF flag becomes asserted immediately after the eQADC detects that the external buffer in use becomes empty.

NOTE

After the transfer of a command sequence to an external command buffer starts, the eQADC ignores, for non-coherency detection purposes, the BUSY fields captured at the end of the first serial transmission. Thereafter, all BUSY fields captured at the end of consecutive serial transmissions are used to check the fullness of that external command buffer. This is done because the eQADC only updates its external ADC command buffer status record when it receives a serial message, resulting that the record kept by the eQADC is always outdated by, at least, the length of one serial transmission. This prevents a CFIFO from immediately becoming non-coherent when it starts transferring commands to an empty external command buffer. See Figure 18-41 for an example.



Assumptions: 1) The CFIFO starts sending commands to an external command buffer when triggered.
2) Execution of a command on the external device takes longer than the time to complete two serial transmissions.

Figure 18-41. External Command Buffer Status Detection at Command Sequence Transfer Start

Table 18-43. External Buffer Status

Capture Point at eQADC	Buffer Status at External Device	Buffer Status as Captured by the eQADC	Used for NCF Detection on the eQADC?
(a)	EMPTY	EMPTY	No change
(b)	1 ENTRY	EMPTY	No
(c)	2 ENTRY	1 ENTRY	Yes

After the start of command sequence transfer, the eQADC checks for the command sequence coherency until the command sequence ends or one of the following conditions are true:

- Command sequence is non-coherent.
- CFIFO status changes from the TRIGGERED state
- CFIFO underflow occurs

NOTE

The NCF flag is asserted if an external command buffer empty event is detected at the same time the eQADC stops checking for the coherency of a command sequence.

After command transfers restart or continue, the non-coherency hardware operate as if the command sequence started from that point. Figure 18-44 depicts how the non-coherency hardware operates when a non-coherency event is detected.

NOTE

If MODE*n* is changed to disabled while a CFIFO is transferring commands, the NCF flag for that CFIFO is not asserted.

NOTE

When the eQADC enters debug or stop mode while a command sequence is executing, the NCF asserts if an empty external command buffer is detected after debug or stop mode exits.

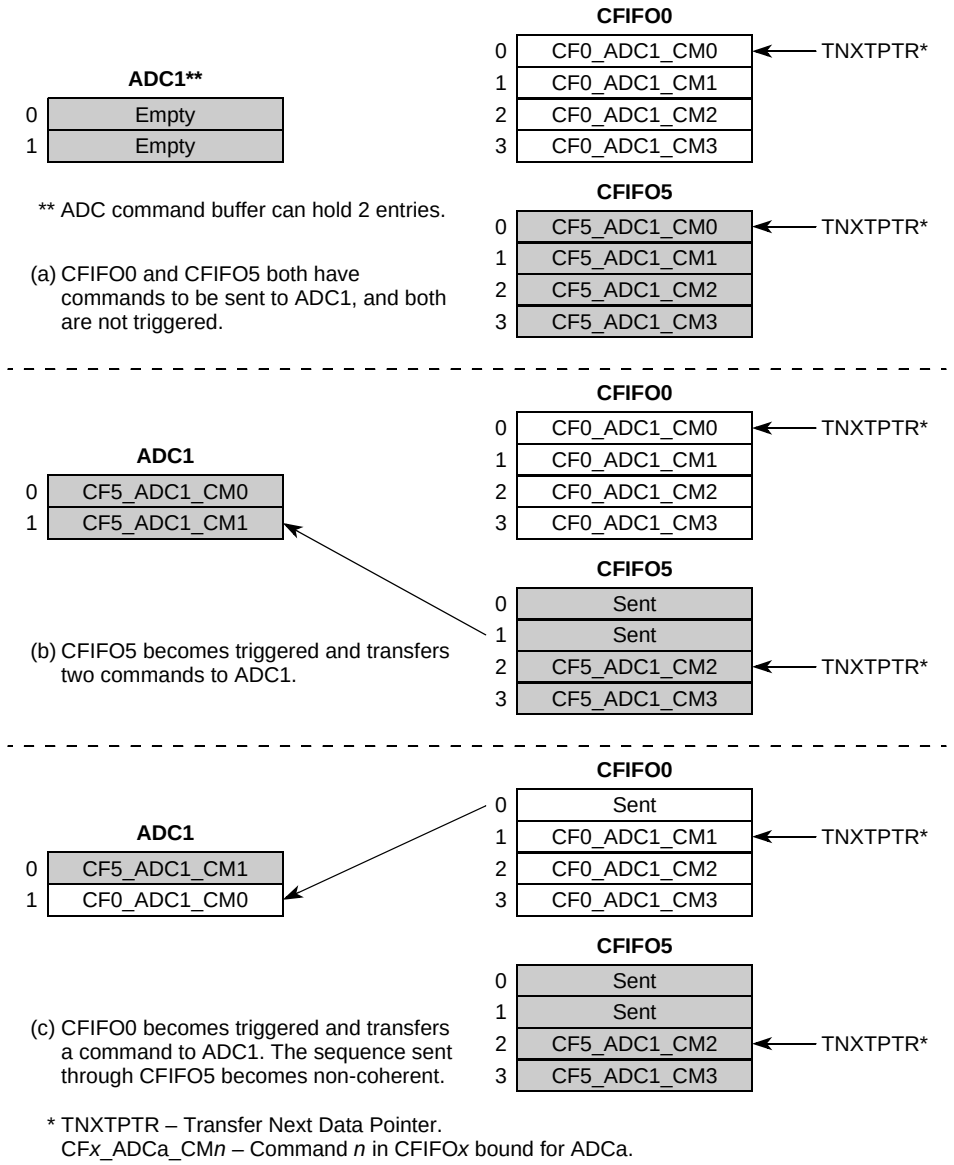
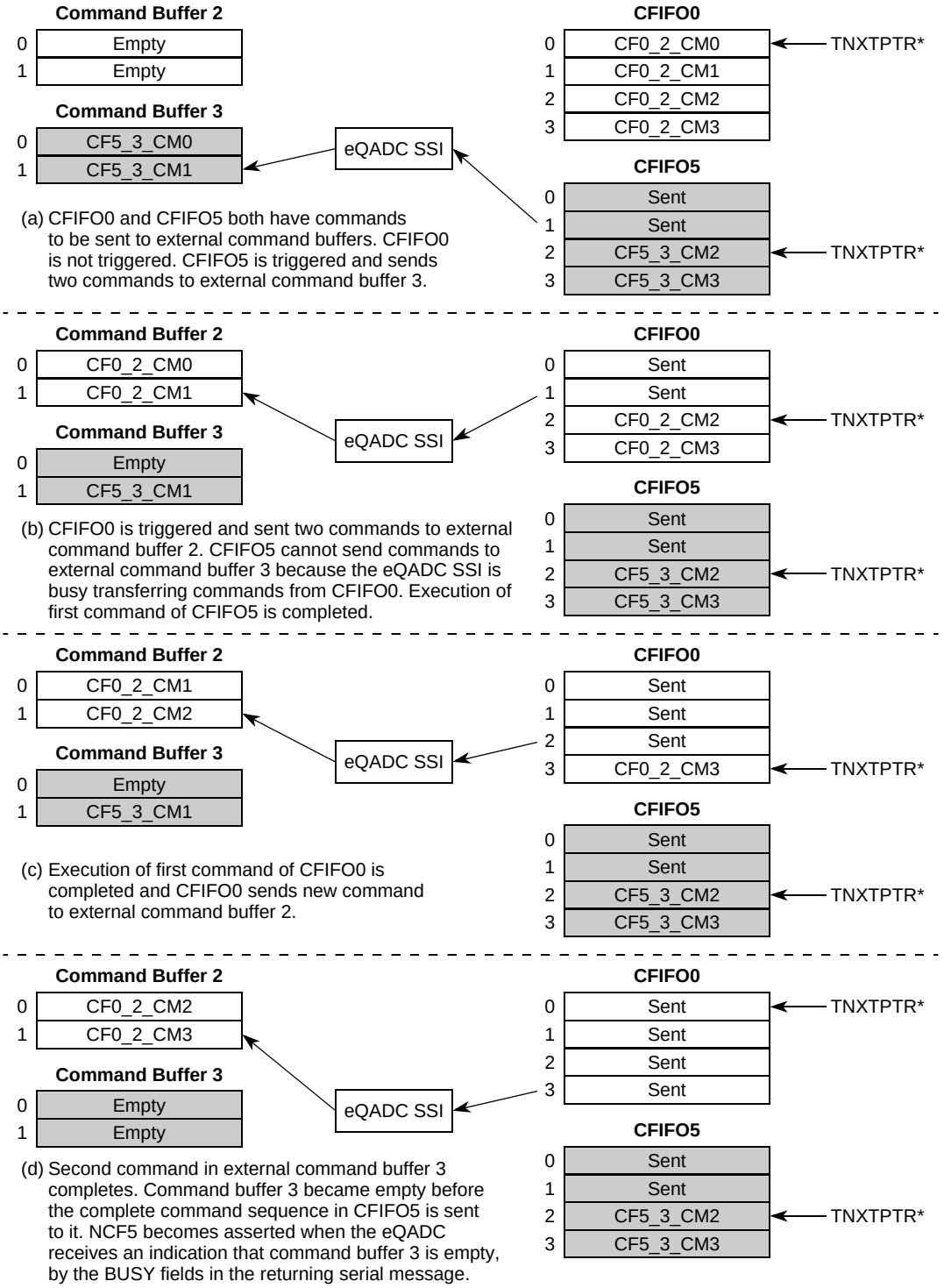


Figure 18-42. Non-Coherency Event When Different CFIFOs Use the Same Buffer



* TNXTPTR – Transfer Next Data Pointer.
CF*x*_ADCa_CM*n* – Command *n* in CFIFO*x* bound for external command buffer a.

Figure 18-43. Non-Coherency Event When Different CFIFOs Are Using Different External Command Buffers

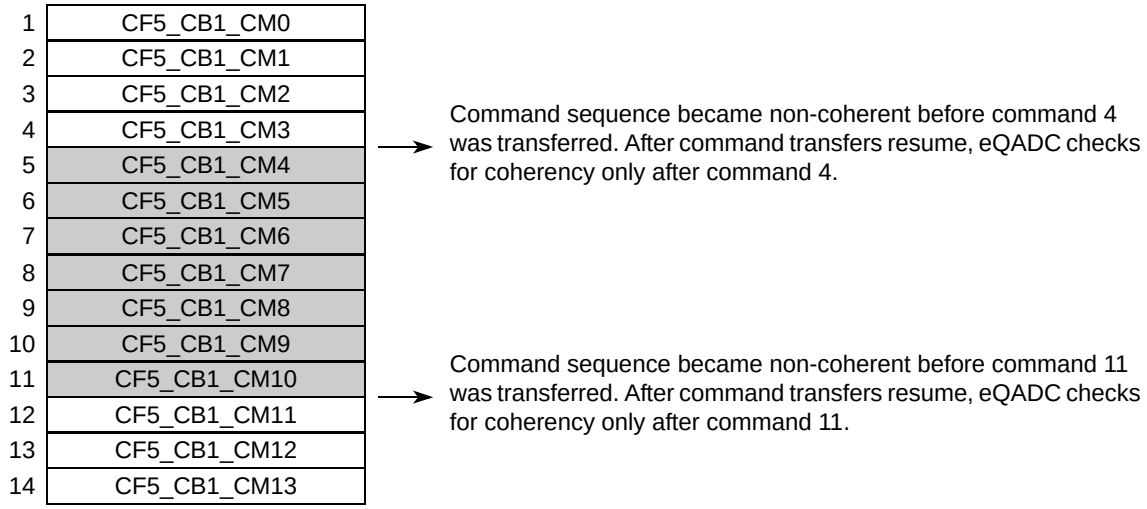


Figure 18-44. Non-coherency Detection When Transfers From a Command Sequence Are Interrupted

18.4.4 Result FIFOs

18.4.4.1 RFIFO Basic Functionality

There are six RFIFOs located in the eQADC. Each RFIFO is four entries deep, and each RFIFO entry is 16 bits long. Each RFIFO serves as a temporary storage location for the one of the result queues allocated in system memory. All result data is saved in the RFIFOs before being moved into the system result queues. When an RFIFO is not empty, the eQADC sets the corresponding EQADC_FISRn[RFDF] (see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”). If EQADC_IDCR*n*[RFDE] is asserted (see Section 18.3.2.7), the eQADC generates a request so that the RFIFO entry is moved to a result queue. An interrupt request, served by the host CPU, is generated when EQADC_IDCR*n*[RFDS] is negated, and an eDMA request, served by the eDMA, is generated when RFDS is asserted. The host CPU or the eDMA responds to these requests by reading EQADC_RFPRn (see Section 18.3.2.5, “eQADC Result FIFO Pop Registers 0–5 (EQADC_RFPRn)”) to retrieve data from the RFIFO.

NOTE

Reading a word, halfword, or any bytes from EQADC_RFPR*n* pops an entry from RFIFO*n*, and the RFCTR*n* field decrements by 1.

Configure the eDMA controller to read a single result (16-bit data) from the RFIFO pop registers for every asserted eDMA request it acknowledges. See Section 18.5.2, “eQADC to eDMA Controller Interface” for eDMA controller configuration guidelines.

Figure 18-45 describes the important components in the RFIFO. Each RFIFO is implemented as a circular set of registers to avoid the need to move all entries at each push/pop operation. The pop next data pointer always points to the next RFIFO message to be retrieved from the RFIFO when reading eQADC_RFPR. The receive next data pointer points to the next available RFIFO location for storing the next incoming

message from the on-chip ADCs or from the external device. The RFIFO counter logic counts the number of entries in RFIFO and generates interrupt or eDMA requests to drain the RFIFO.

EQADC_FISRn[POPNXTPTR] (see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) indicates which entry is currently being addressed by the pop next data pointer, and EQADC_FISRn[RFCTR] provides the number of entries stored in the RFIFO. Using POPNXTPTR and RFCTR, the absolute addresses for pop next data pointer and receive next data pointer can be calculated using the following formulas:

```
Pop Next Data Pointer Address= RFIFOn_BASE_ADDRESS + POPNXTPTRn x 4
Receive Next Data Pointer Address = RFIFOn_BASE_ADDRESS +
[(POPNXTPTRn + RFCTRn) mod RFIFO_DEPTH] x 4
```

where

- *a mod b* returns the remainder of the division of *a* by *b*.
- RFIFO*n*_BASE_ADDRESS is the smallest memory mapped address allocated to an RFIFO*n* entry.
- RFIFO_DEPTH is the number of entries contained in a RFIFO - four in this implementation.

When a new message arrives and RFIFO*n* is not full, the eQADC copies its contents into the entry pointed by receive next data pointer. The RFIFO counter EQADC_FISR*n*[RFCTR*n*] (see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) is incremented by 1, and the receive next data pointer *n* is also incremented by 1 (or wrapped around) to point to the next empty entry in RFIFO*n*. However, if the RFIFO*n* is full, the eQADC sets the EQADC_FISR*n*[RFOF] (see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”). The RFIFO*n* does not overwrite the older data in the RFIFO, the new data is ignored, and the receive next data pointer *n* is not incremented or wrapped around. RFIFO*n* is full when the receive next data pointer *n* equals the pop next data pointer *n* and RFCTR*n* is not 0. RFIFO*n* is empty when the receive next data pointer *n* equals the pop next data pointer *n* and RFCTR*n* is 0.

When the eQADC RFIFO pop register *n* is read and the RFIFO*n* is not empty, the RFIFO counter RFCTR*n* is decremented by 1, and the pop next data pointer is incremented by 1 (or wrapped around) to point to the next RFIFO entry.

When the eQADC RFIFO pop register *n* is read and RFIFO*n* is empty, eQADC does not decrement the counter value and the pop next data pointer *n* is not updated. The read value is undefined.

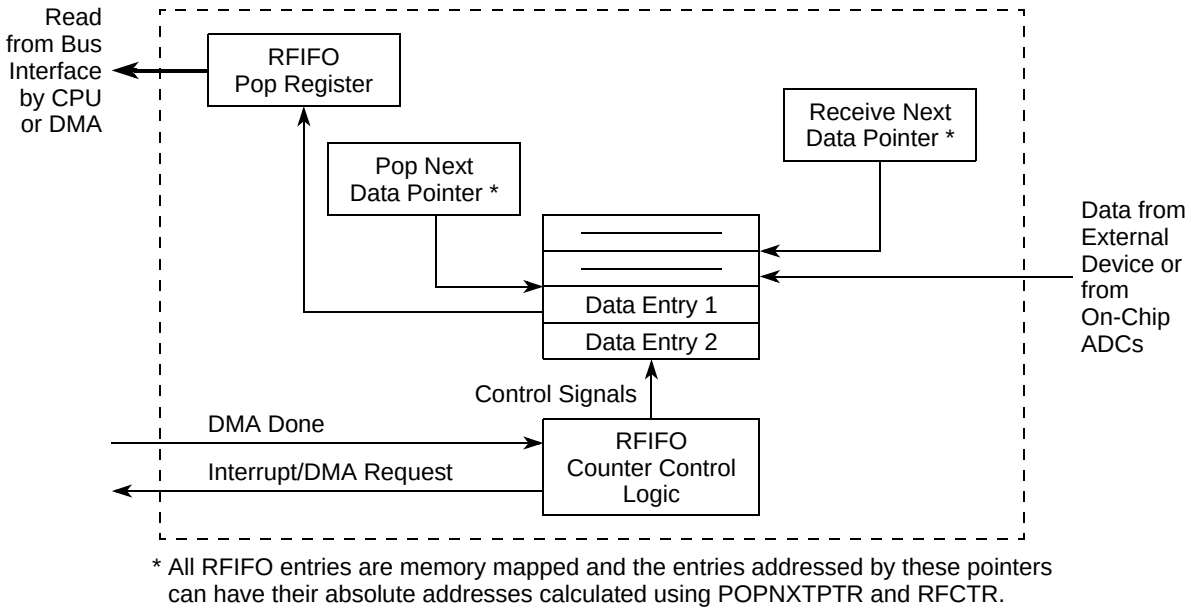


Figure 18-45. RFIFO Diagram

The detailed behavior of the pop next data pointer and receive next data pointer is described in the example shown in Figure 18-46 where an RFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four entries. In this example, RFIFO*n* with 16 entries is shown in sequence after popping or receiving entries.

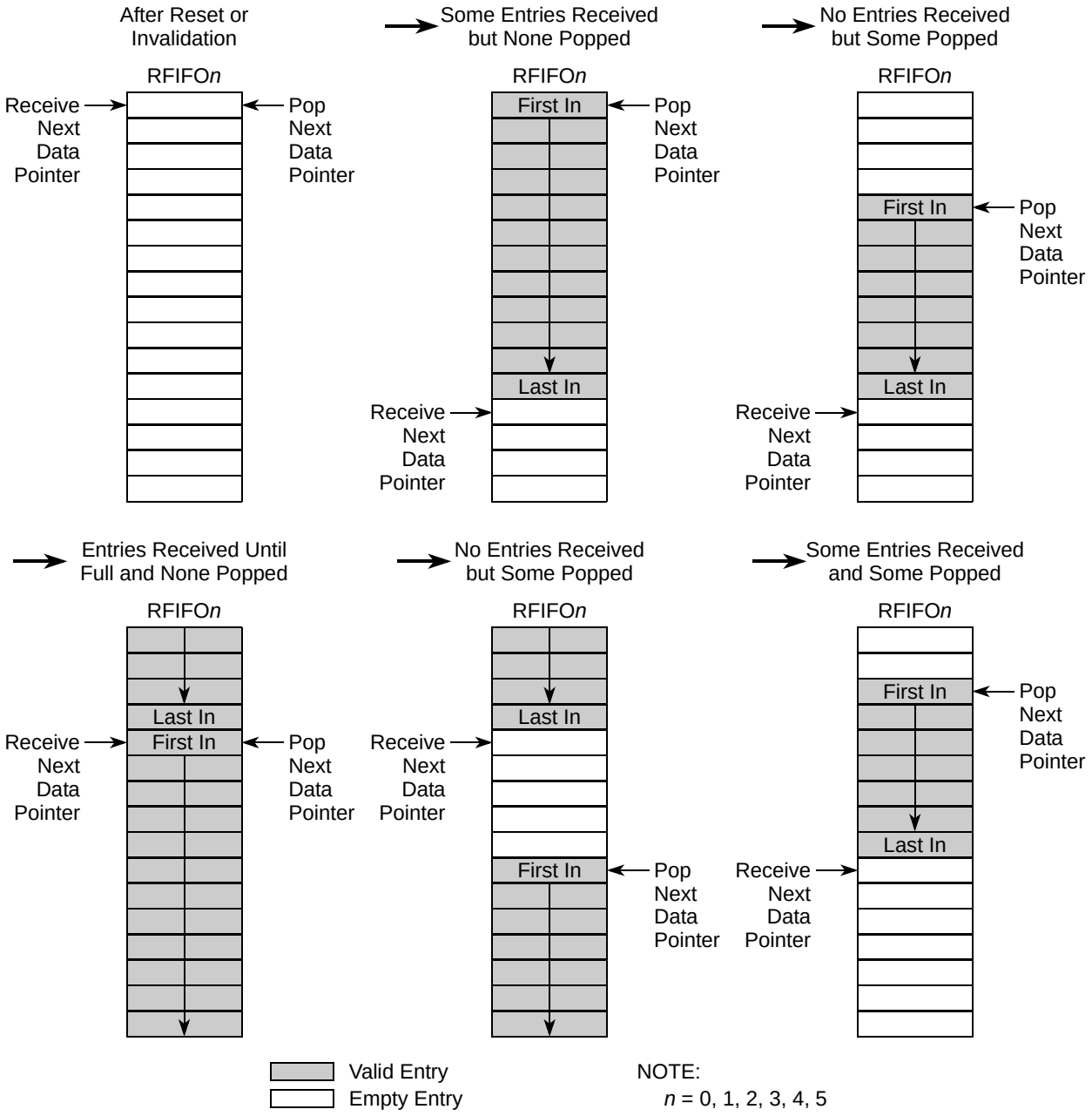


Figure 18-46. RFIFO Entry Pointer Example

18.4.4.2 Distributing Result Data into RFIFOs

Data to be moved into the RFIFOs can come from three sources: from ADC0, from ADC1, or from the external device. All result data comes with a MESSAGE_TAG field defining what to do with the received data. The FIFO control unit decodes the MESSAGE_TAG field and:

- Stores the 16-bit data into the appropriate RFIFO if the MESSAGE_TAG indicates a valid RFIFO number; or
- Ignores the data in case of a null or “reserved for customer use” MESSAGE_TAG

In general, received data is moved into RFIFOs as the data becomes available, while an exception happens when multiple results from different sources become available at the same time. In that case, result data from ADC0 is processed first, result data from ADC1 is only processed after all ADC0 data is processed, and result data from the external device is only processed after all data from ADC0/1 is processed.

When time-stamped results return from the on-chip ADCs, the conversion result and the time stamp are always moved to the RFIFOs in consecutive clock cycles to guarantee they are always stored in consecutive RFIFO entries.

18.4.5 On-Chip ADC Configuration and Control

18.4.5.1 Enabling and Disabling the on-chip ADCs

The on-chip ADCs have an enable bit (ADC0_CR[ADC0_EN] and ADC1_CR[ADC1_EN], see Section 18.3.3.1, “ADCn Control Registers (ADC0_CR and ADC1_CR)”) which allows the enabling of the ADCs only when necessary. When the enable bit for an ADC is negated, the clock input to that ADC is stopped. The ADCs are disabled out of reset - ADC0/1_EN bits are negated - to allow for their safe configuration. The ADC must only be configured when its enable bit is negated. After the enable bit of an ADC is asserted, clock input is started, and the bias generator circuit is turned on. When the enable bits of both ADCs are negated, the bias circuit generator is stopped.

NOTE

An 8ms wait time from V_{DDA} power up to enabling an ADC is required to pre-charge the external 100nf capacitor on REFBYPC. This time must be guaranteed by the crystal startup time plus the reset duration, or the host application. The ADC internal bias generator circuit starts up after 10υs upon VRH/VRL power up to stabilize the required bias current to the pre-charge circuit; the current to the other analog circuits are disabled until the ADCs are enabled. As soon as the ADCs are enabled, the bias currents to other analog circuits are ready.

NOTE

The eQADC is designed to wait 120 ADC clocks after an on-chip ADC enables or the eQADC exits stop mode before the first conversion command is issued. Two independent counters monitor the delay, one clocked by ADC0_CLK and another by ADC1_CLK. Conversion commands can begin executing when one of the counters reaches 120 ADC clocks. Conversion commands sent to a disabled ADC are ignored by the ADC control hardware.

18.4.5.2 ADC Clock and Conversion Speed

The clock input to the ADCs is defined by setting the ADC0_CR[ADC0_CLK_PS] and ADC1_CR[ADC1_CLK_PS] fields. See Section 18.3.3.1, “ADCn Control Registers (ADC0_CR and ADC1_CR).” The ADC0/1_CLK_PS field selects the clock divide factor by which the system clock is divided as showed in Table 18-25. The ADC clock frequency is calculated as below and it must not exceed 12 MHz.

$$ADCClockFrequency = \frac{SystemClockFrequency(MHz)}{SystemClockDivideFactor}; (ADCClockFrequency \leq 12MHz)$$

Figure 18-47 depicts how the ADC clocks for ADC0 and ADC1 are generated.

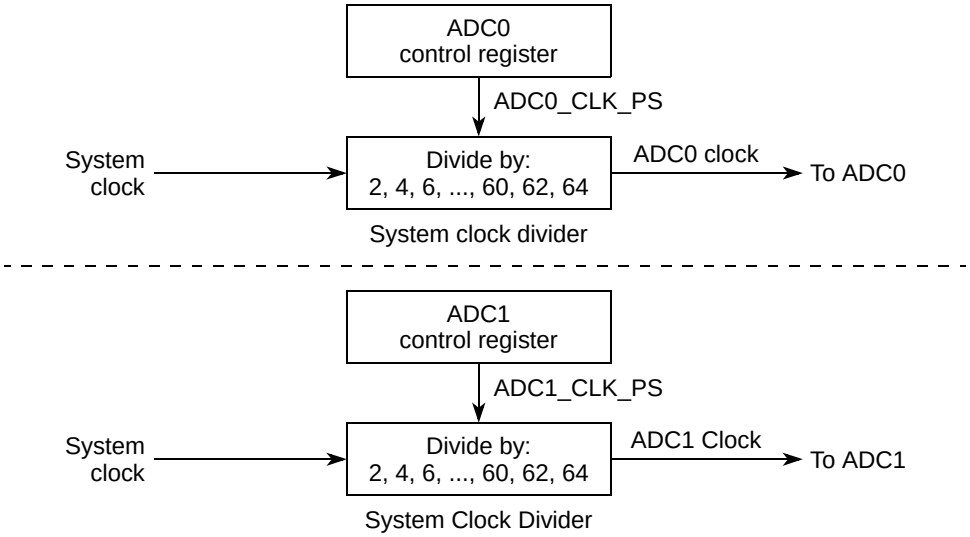


Figure 18-47. ADC0 and ADC1 Clock Generation

The ADC conversion speed (in kilosamples per second – ksamp/s) is calculated by the following formula. The number of sampling cycles is determined by the LST bits in the command message— see Section , “Conversion Command Message Format for On-Chip ADC Operation,” — and it can take one of the following values: 2, 8, 64, or 128 ADC clock cycles. The number of AD conversion cycles is 13 for differential conversions and 14 for single-ended conversions. The maximum conversion speed is achieved when the ADC Clock frequency is set to its maximum (12 Mhz) and the number of sampling cycles set to its minimum (2 cycles). The maximum conversion speed for differential and single-ended conversions are 800 k samples/sec and 750 k samples/sec, respectively.

$$ADCConversionSpeed = \frac{ADCClockFrequency(MHz)}{(NumberOfSamplingCycles + NumberOfADConversionCycles)}$$

Table 18-44 shows an example of how the ADC0/1_CLK_PS can be set when using a 72 MHz system clock and the corresponding conversion speeds for all possible ADC clock frequencies. The table also shows that according to the system clock frequency, certain clock divide factors are invalid (2, 4, 6, 8 clock divide factors in the example) since their use would result in a ADC clock frequency higher than the maximum one supported by the ADC. ADC clock frequency must not exceed 12 Mhz.

Table 18-44. ADC Clock Configuration Example—System Clock Frequency = 72 MHz

ADC0 or ADC1 CLK_PS[0:4]	System Clock Divide Factor	ADC Clock in MHz System Clock = 72 MHz	Differential Conversion Speed with Default Sampling Time (13 + 2 cycles) in ksamples/s	Single-Ended Conversion Speed with Default Sampling Time (14 + 2 cycles) in ksamples/s
0b00000	2	—	—	—
0b00001	4	—	—	—
0b00010	6	12	800	750
0b00011	8	9	600	563
0b00100	10	7.2	480	450
0b00101	12	6	400	375
0b00110	14	5.14	343	321
0b00111	16	4.5	300	281
0b01000	18	4	267	250
0b01001	20	3.6	240	225
0b01010	22	3.27	218	204
0b01011	24	3	200	188
0b01100	26	2.77	185	173
0b01101	28	2.57	171	161
0b01110	30	2.4	160	150
0b01111	32	2.25	150	141
0b10000	34	2.12	141	133
0b10001	36	2	133	125
0b10010	38	1.89	126	118
0b10011	40	1.8	120	113
0b10100	42	1.71	114	107
0b10101	44	1.64	109	103
0b10110	46	1.57	105	98
0b10111	48	1.5	100	94
0b11000	50	1.44	96	90
0b11001	52	1.38	92	86
0b11010	54	1.33	89	83
0b11011	56	1.29	86	81
0b11100	58	1.24	83	78
0b11101	60	1.2	80	75

Table 18-44. ADC Clock Configuration Example—System Clock Frequency = 72 MHz (continued)

ADC0 or ADC1 CLK_PS[0:4]	System Clock Divide Factor	ADC Clock in MHz System Clock = 72 MHz	Differential Conversion Speed with Default Sampling Time (13 + 2 cycles) in ksamples/s	Single-Ended Conversion Speed with Default Sampling Time (14 + 2 cycles) in ksamples/s
0b11110	62	1.16	77	73
0b11111	64	1.13	75	71

18.4.5.3 Time Stamp Feature

The on-chip ADCs can provide a time stamp for the conversions they execute. A time stamp is the value of the time base counter latched when the eQADC detects the end of the analog input voltage sampling. A time stamp for a conversion command is requested by setting the TSR bit in the corresponding command. When TSR is negated, that is a time stamp is not requested, the ADC returns a single result message containing the conversion result. When TSR is asserted, that is a time stamp is requested, the ADC returns two result messages; one containing the conversion result, and another containing the time stamp for that conversion. The result messages are sent in this order to the RFIFOs and both messages are sent to the same RFIFO as specified in the MESSAGE_TAG field of the executed conversion command.

The time base counter is a 16-bit up counter and wraps after reaching 0xFFFF. It is disabled after reset and it is enabled according to the setting of ADC_TSCR[TBC_CLK_PS] field (see Section 18.3.3.2, “ADC Time Stamp Control Register (ADC_TSCR)”). TBC_CLK_PS defines if the counter is enabled or disabled, and, if enabled, at what frequency it is incremented. The time stamps are returned regardless of whether the time base counter is enabled or disabled. The time base counter can be reset by writing 0x0000 to the ADC_TBCR (Section 18.3.3.3, “ADC Time Base Counter Registers (ADC_TBCR)”) with a write configuration command.

18.4.5.4 ADC Calibration Feature

18.4.5.4.1 Calibration Overview

The eQADC provides a calibration scheme to remove the effects of gain and offset errors from the results generated by the on-chip ADCs. Only results generated by the on-chip ADCs are calibrated. The results generated by ADCs on the external device are directly sent to RFIFOs unchanged. The main component of calibration hardware is a multiply-and-accumulate (MAC) unit, one per on-chip ADC, that is used to calculate the following transfer function which relates a calibrated result to a raw, uncalibrated one.

```
CAL_RES = GCC x (RAW_RES + OCC + 2);
```

where:

- CAL_RES is the calibrated result corresponding the input voltage V_i.
- GCC is the gain calibration constant.
- RAW_RES is the raw, uncalibrated result corresponding to an specific input voltage V_i.
- OCC is the offset calibration constant.
- The addition of two reduces the maximum quantization error of the ADC. See Section 18.5.6.3, “Quantization Error Reduction During Calibration.”

Calibration constants GCC and OCC are determined by taking two samples of known reference voltages and using these samples to calculate their values. For details and example about how to calculate the calibration constants and use them in result calibration see Section 18.5.6, “ADC Result Calibration.” After it is calculated, the GCC coefficients are stored in ADC0_GCCR and ADC1_GCCR (see Section 18.3.3.4, “ADCn Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)”) and the OCC coefficients in ADC0_OCCR and ADC1_OCCR (see Section 18.3.3.5, “ADCn Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)”) from where their values are fed to the MAC unit. Since the analog characteristics of each on-chip ADC differs, each ADC has an independent pair of calibration constants.

A conversion result is calibrated according to the status of CAL bit in the command that initiated the conversion. If the CAL bit is asserted, the eQADC automatically calculates the calibrated result before sending the result to the appropriate RFIFO. If the CAL bit is negated, the result is not calibrated, it bypasses the calibration hardware, and is directly sent to the appropriate RFIFO.

18.4.5.4.2 MAC Unit and Operand Data Format

The MAC unit diagram is shown in Figure 18-48. Each on-chip ADC has a separate MAC unit to calibrate its conversion results.

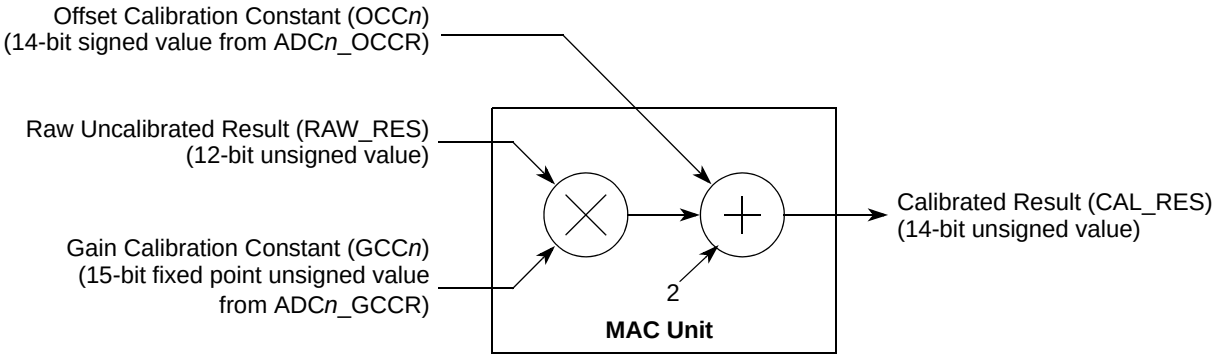


Figure 18-48. MAC Unit Diagram

The OCC*n* operand is a 14-bit signed value and it is the upper 14 bits of the value stored in ADC0_OCCR and ADC1_OCCR. The RAW_RES operand is the raw uncalibrated result, and it is a direct output from the on-chip ADCs.

The GCC*n* operand is a 15-bit fixed point unsigned value, and it is the upper 15 bits of the value stored in ADC0_GCCR and ADC1_GCCR. The GCC is expressed in the *GCC_INT.GCC_FRAC* binary format. The integer part of the GCC (GCC_INT = GCC[1]) contains a single binary digit while its fractional part (GCC_FRAC = GCC[2:15]) contains 14 bits. See Figure 18-49 for more information. The gain constant equivalent decimal value ranges from 0 to 1.999938..., as shown in Table 18-46. Two is always added to the MAC output: see Section 18.5.6.3, “Quantization Error Reduction During Calibration. CAL_RES output is the calibrated result, and it is a 14-bit unsigned value. CAL_RES is truncated to 0x3FFF, in case of a overflow, and to 0x0000, in case of an underflow.

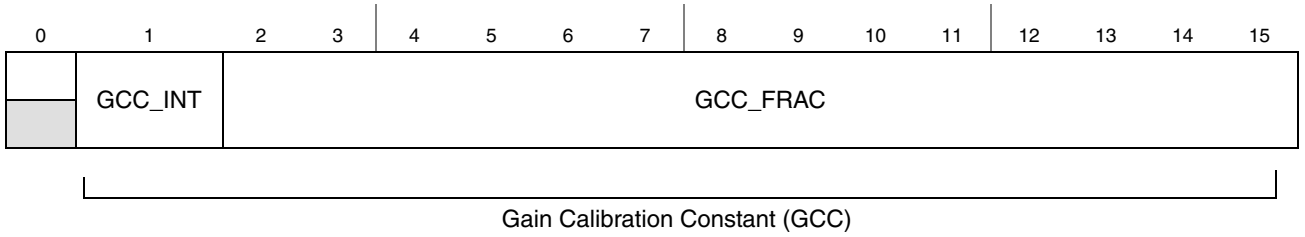


Figure 18-49. Gain Calibration Constant Format

Table 18-45. Gain Calibration Constant Format Field Descriptions

Field	Description
0	Reserved
1 GCC_INT	Integer part of the gain calibration constant for ADC*n*. GCC_INT is the integer part of the gain calibration constant (GCC) for ADC0/1. 0 = ADC0 1 = ADC1
2–15 GCC_FRAC [1:14]	Fractional part of the gain calibration constant for ADC*n*. GCC_FRAC is the fractional part of the gain calibration constant (GCC) for ADC*n*. GCC_FRAC can expresses decimal values ranging from 0 to 0.999938...

Table 18-46. Correspondence between Binary and Decimal Representations of the Gain Constant

Gain Constant (GCC_INT.GCC_FRAC binary format)	Corresponding Decimal Value
0.0000_0000_0000_00	0
...	...
0.1000_0000_0000_00	0.5
...	...
0.1111_1111_1111_11	0.999938...
1.0000_0000_0000_00	1
...	...
1.1100_0000_0000_00	1.75
...	...
1.1111_1111_1111_11	1.999938...

18.4.5.5 ADC Control Logic Overview and Command Execution

Figure 18-50 shows the basic logic blocks involved in the ADC control and how they interact. CFIFOs/RFIFOs interact with ADC command/result message return logic through the FIFO control unit. The EB and BN bits in the command message uniquely identify the ADC to which the command is sent. The FIFO control unit decodes these bits and sends the ADC command to the proper ADC. Other blocks of logic are the result format and calibration submodule, the time stamp logic, and the MUX control logic.

The result format and calibration submodule formats the returning data into result messages and sends them to the RFIFOs. The returning data can be data read from an ADC register, a conversion result, or a time stamp. The formatting and calibration of conversion results also take place inside this submodule.

The time stamp logic latches the value of the time base counter when detecting the end of the analog input voltage sampling, and sends it to the result format and calibration submodule as time stamp information.

The MUX control logic generates the proper MUX control signals and, when the ADC0/1_EMUX bits are asserted, the MA signals based on the channel numbers extracted from the ADC Command.

ADC commands are stored in the ADC command buffers (2 entries) as they come in and they are executed on a first-in-first-out basis. After the execution of a command in ENTRY1 finishes, all commands are shifted one entry. After the shift, ENTRY0 is always empty and ready to receive a new command. Execution of configuration commands only starts when they reach ENTRY1. Consecutive conversion commands are pipelined, and their execution can start while in ENTRY0. This is explained below.

A/D conversion accuracy can be affected by the settling time of the input channel multiplexers. Some time is required for the channel multiplexer’s internal capacitances to settle after the channel number is changed. If the time prior to and during sampling is not long enough to permit this settling, then the voltage on the sample capacitors do not accurately represent the voltage to be read. This is a problem in particular when external muxes are used.

To maximize settling time, when a conversion command is in buffer ENTRY1 and another conversion command is identified in ENTRY0, then the channel number of ENTRY0 is sent to the *MUX control logic* half an ADC clock before the start of the sampling phase of the command in ENTRY0. This pipelining of sample and settling phase is shown in Figure 18-51(b).

This provides more accurate sampling, which is specially important for applications that require high conversion speeds, i.e., with the ADC running at maximum clock frequency and with the analog input voltage sampling time set to a minimum (2 ADC clock cycles). In this case, the short sampling time can prevent the multiplexers to completely settle. The second advantage of pipelining conversion commands is to provide equal conversion intervals even though the sample time increases on second and subsequent conversions. See Figure 18-51. This is important for any digital signal process application.

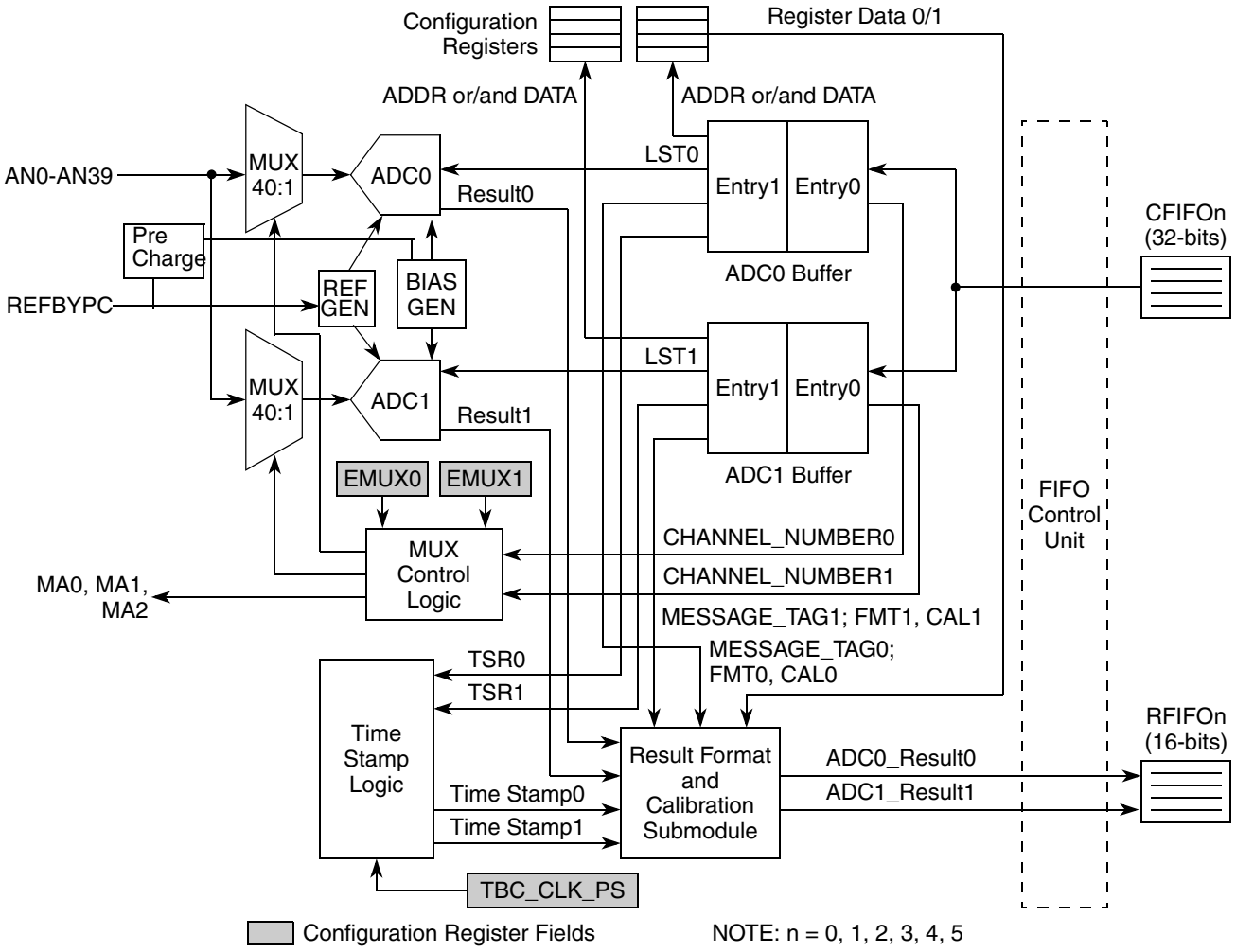


Figure 18-50. On-Chip ADC Control Scheme

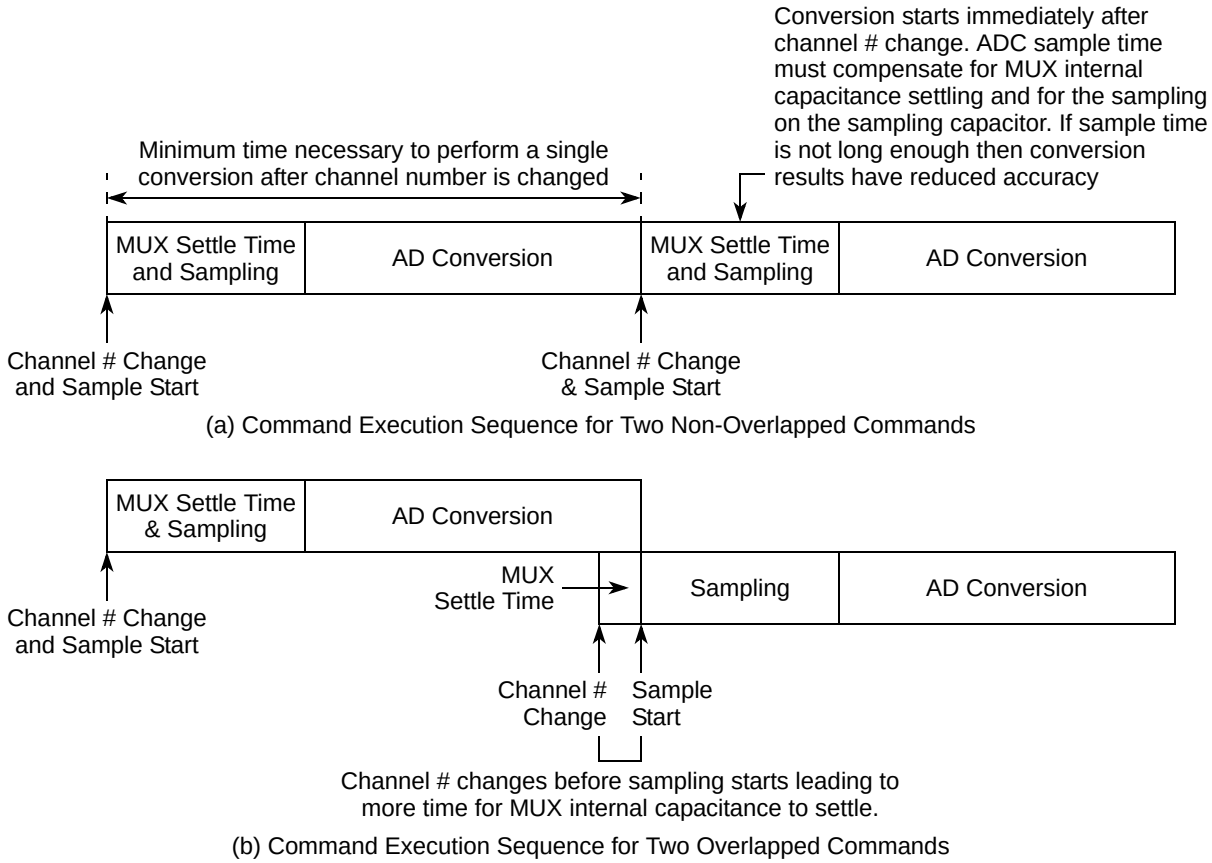


Figure 18-51. Overlapping Consecutive Conversion Commands

18.4.6 Internal and External Multiplexing

18.4.6.1 Channel Assignment

The internal analog multiplexers select one of the 40 analog input pins for conversion, based on the CHANNEL_NUMBER field of a Command Message. The analog input pin channel number assignments and the pin definitions vary depending on how the ADC0/1_EMUX are configured. Only one ADC can drive the external mux address pins: therefore ADC0_EMUX and ADC1_EMUX must not be asserted at the same time.

During differential conversions the analog multiplexer passes differential signals to both the positive and negative terminals of the ADC. The differential conversions can only be initiated on four channels: DAN0, DAN1, DAN2, and DAN3. See Table 18-48 and Figure 18-49 for the channel numbers used to select differential conversions.

Table 18-47. ADC*n*_EMUX Bits Combinations

ADC0_EMUX	ADC1_EMUX	Set CHANNEL_NUMBER	
		ADC0	ADC1
0	0	No external mux	No external mux
0	1	ADC1 uses the external mux	ADC1 uses the external mux
1	0	ADC0 uses the external mux	ADC0 uses the external mux
1	1	Reserved. Do not use.	

Table 18-48 shows the channel number assignments for the non-multiplexed mode. The 40 single-ended channels and 4 differential pairs are shared between the two ADCs.

Table 18-48. Non-multiplexed Channel Assignments [1]

Input Pins			Channel Number in CHANNEL_NUMBER Field	
Analog Pin Name	Other Functions	Conversion Type	Binary	Decimal
AN[0]–AN[39]		Single-ended	0000_0000–0010_0111	0–39
V_{RH}		Single-ended	0010_1000	40
V_{RL}		Single-ended	0010_1001	41
	(V_{RH} - V_{RL}) ÷ 2 see footnote[2]	Single-ended	0010_1010	42
	75% x (V_{RH} - V_{RL})	Single-ended	0010_1011	43
	25% x (V_{RH} - V_{RL})	Single-ended	0010_1100	44
Reserved			0010_1101–0101_1111	45–95
DAN0+ and DAN0- DAN1+ and DAN1- DAN2+ and DAN2- DAN3+ and DAN3-		Differential Differential Differential Differential	0110_0000 0110_0001 0110_0010 0110_0011	96 97 98 99
Reserved			0110_0100–1111_1111	100–255

1 The two on-chip ADCs can access the same analog input pins but simultaneous conversions are not allowed. Also, when one ADC is performing a differential conversion on a pair of pins, the other ADC must not access either of these two pins as single-ended channels.

2 This equation only applies before calibration. After calibration, the 50% reference point returns approximately 20mV lower than the expected 50% of the difference between the High Reference Voltage (V_{RH}) and the Low Reference Voltage (V_{RL}). For calibration of the ADC, only use the 25% and 75% points as described in Section 18.5.6.1, "MAC Configuration Procedure"

Table 18-49 shows the channel number assignments for multiplexed mode. The ADC with the ADC*n*_EMUX bit asserted can access at most 33 single-ended and 32 externally multiplexed channels. See Section 18.4.6.2, "External Multiplexing," for a detailed explanation about how external multiplexing can be achieved.

Table 18-49. Multiplexed Channel Assignments[1]

Input Pins			Channel Number in CHANNEL_NUMBER Field	
Analog Pin Name	**Other Functions**	**Conversion Type**	**Binary**	**Decimal**
AN[0]–AN[7]		Single-ended	0000_0000–0000_0111	0–7
Used for digital address lines of the external mux			0000_1000–0000_1011	8–11
AN[12]–AN[39]		Single-ended	0000_1100–0010_0111	12–39
V_{RH}		Single-ended	0010_1000	40
V_{RL}		Single-ended	0010_1001	41
	$(V_{RH} - V_{RL}) \div 2$	Single-ended	0010_1010	42
	75% x $(V_{RH} - V_{RL})$	Single-ended	0010_1011	43
	25% x $(V_{RH} - V_{RL})$	Single-ended	0010_1100	44
Reserved			0010_1101–0011_1111	45–63
ANW ANX ANY ANZ	— — — —	Single-ended Single-ended Single-ended Single-ended	0100_0xxx 0100_1xxx 0101_0xxx 0101_1xxx	64–71 72–79 80–87 88–95
DAN0+ and DAN0- DAN1+ and DAN1- DAN2+ and DAN2- DAN3+ and DAN3-		Differential Differential Differential Differential	0110_0000 0110_0001 0110_0010 0110_0011	96 97 98 99
Reserved			0011_0100–1111_1111	100–255

1 The two on-chip ADCs can access the same analog input pins but simultaneous conversions are not allowed. Also, when one ADC is performing a differential conversion on a pair of pins, the other ADC must not access either of these two pins as single-ended channels.

18.4.6.2 External Multiplexing

The eQADC can use from one to four external multiplexers to expand the number of analog signals that can be converted. Using this configuration, up to 25 additional channels can be created.

- The first external multiplexer requires one common analog pin and three address pins, so although eight additional ADC channels are created, four existing channels are lost, so there is a net addition of four channels.
- For subsequent external multiplexers, only one additional internal channel is lost so there is a net addition of seven channels.

The externally multiplexed channels are automatically selected by the CHANNEL_NUMBER field of a command message, in the same way done with internally multiplexed channels. The software selects the external multiplexed mode by setting the ADC0/1_EMUX bit in either ADC0_CR or ADC1_CR depending on which ADC performs the conversion. Figure 18-49 shows the channel number assignments for the multiplexed mode. Only one ADC can have its ADC0/1_EMUX bit asserted at a time.

Figure 18-52 shows the maximum configuration of four external multiplexer chips connected to the eQADC. The external multiplexer chip selects one of eight analog inputs and connects it to a single analog output, which is fed to a specific input of the eQADC. The eQADC provides three multiplexed address signals, MA[0], MA[1], and MA[2], to select one of eight inputs. These three multiplexed address signals are connected to all four external multiplexer chips. The analog output of the four multiplex chips are each connected to four separate eQADC inputs, ANW, ANX, ANY, and ANZ. The MA pins correspond to the three least significant bits of the channel number that selects ANW, ANX, ANY, and ANZ with MA[0] being the most significant bit. See Table 18-50.

Table 18-50. Encoding of MA Pins[1]

Channel Number selecting ANW, ANX, ANY, ANZ (decimal)				MA0	MA1	MA2
ANW	ANX	ANY	ANZ			
64	72	80	88	0	0	0
65	73	81	89	0	0	1
66	74	82	90	0	1	0
67	75	83	91	0	1	1
68	76	84	92	1	0	0
69	77	85	93	1	0	1
70	78	86	94	1	1	0
71	79	87	95	1	1	1

1 0 means pin is driven LOW and 1 that pin is driven HIGH.

When the external multiplexed mode is selected for either ADC, the eQADC automatically creates the MA output signals from CHANNEL_NUMBER field of a command message. The eQADC also converts the proper input channel (ANW, ANX, ANY, and ANZ) by interpreting the CHANNEL_NUMBER field. As a result, up to 32 externally multiplexed channels appear to the conversion queues as directly connected signals.

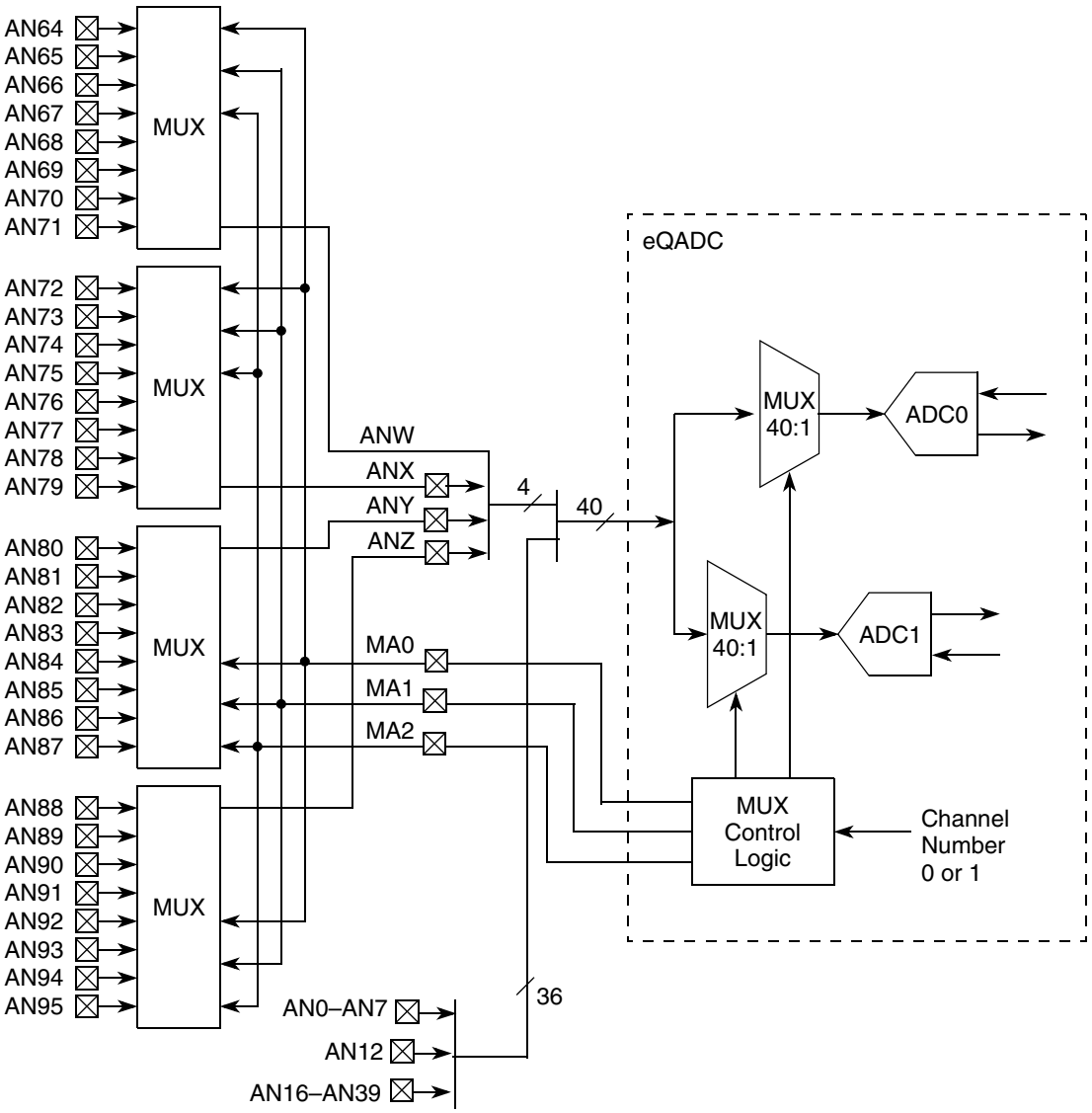


Figure 18-52. Example of External Multiplexing

18.4.7 eQADC eDMA or Interrupt Request

Table 18-51 lists methods to generate interrupt requests in the eQADC queuing control and triggering control. The eDMA/interrupt request select bits and the eDMA/interrupt enable bits are described in Section 18.3.2.7, "eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)," and the interrupt flag bits are described in Section 18.3.2.8, "eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)." Table 18-53 depicts all interrupts and eDMA requests generated by the eQADC.

Table 18-51. eQADC FIFO Interrupt Summary[1]

Interrupt	Condition	Clearing Mechanism
Non Coherency Interrupt	NCIE*n* = 1 NCF*n* = 1	Clear NCF*n* bit by writing a 1 to the bit.
Trigger Overrun Interrupt [2]	TORIE*n* = 1 TORF*n* =1	Clear TORF*n* bit by writing a 1 to the bit.
Pause Interrupt	PIE*n* = 1 PF*n* =1	Clear PF*n* bit by writing a 1 to the bit.
End of Queue Interrupt	EOQIE*n* = 1 EOQF*n* = 1	Clear EOQF*n* bit by writing a 1 to the bit.
Command FIFO Underflow Interrupt [2]	CFUIE*n* = 1 CFUF*n* = 1	Clear CFUF*n* bit by writing a 1 to the bit.
Command FIFO Fill Interrupt	CFFE*n* = 1 CFFS*n* = 0 CFFF*n* = 1	Clear CFFF*n* bit by writing a 1 to the bit.
Result FIFO Overflow Interrupt [2]	RFOIE*n* = 1 RFOF*n* = 1	Clear RFOF*n* bit by writing a 1 to the bit.
Result FIFO Drain Interrupt	RFDE*n* = 1 RFDS*n* = 0 RFDF*n* = 1	Clear RFDF*n* bit by writing a 1 to the bit.

1 For details see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn),” and Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn).”

2 Apart from generating an independent interrupt request for when a RFIFO overflow interrupt, a CFIFO underflow interrupt, and a CFIFO trigger overrun interrupt occurs, the eQADC also provides a combined interrupt request at which these requests from ALL CFIFOs are ORed. See Figure 18-53 for details.

Table 18-52 describes a list of methods to generate eDMA requests by the eQADC.

Table 18-52. eQADC FIFO eDMA Summary[1]

eDMA Request	Condition	Clearing Mechanism
Result FIFO Drain eDMA Request	RFDE*n* = 1 RFDS*n* = 1 RFDF*n* = 1	The eQADC automatically clears the RFDF*n* when RFIFO*n* becomes empty. Writing 1 to the RFDF*n* bit is not allowed while RDFS = 1.
Command FIFO Fill eDMA Request	CFFE*n* = 1 CFFS*n* = 1 CFFF*n* = 1	The eQADC automatically clears the CFFF*n* when CFIFO*n* becomes full. Writing 1 to the CFFF*n* bit is not allowed while CFDS = 1.

1 For details see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn),” and Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn).”

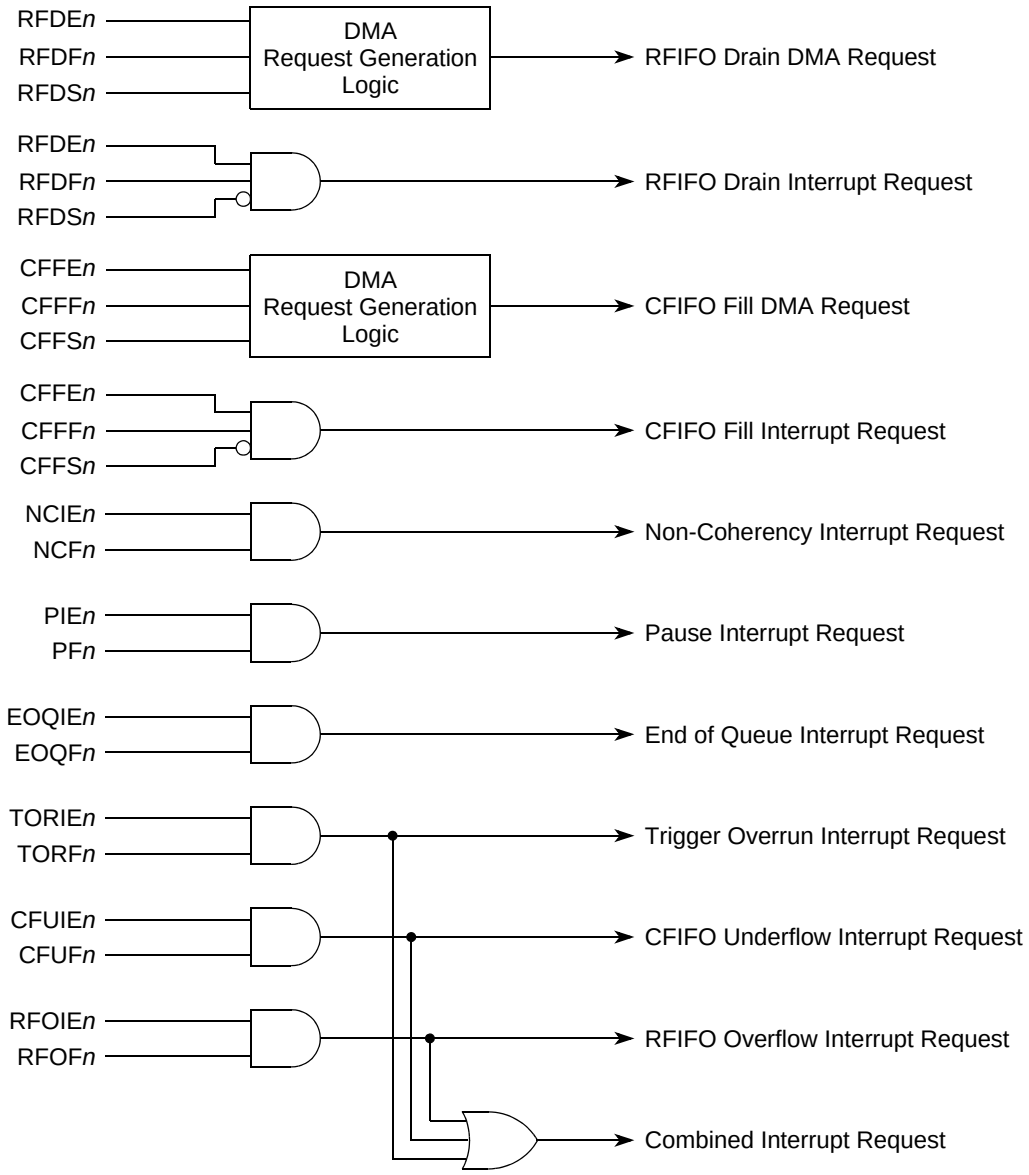


Figure 18-53. eQADC eDMA and Interrupt Requests

18.4.8 eQADC Synchronous Serial Interface (SSI) Submodule

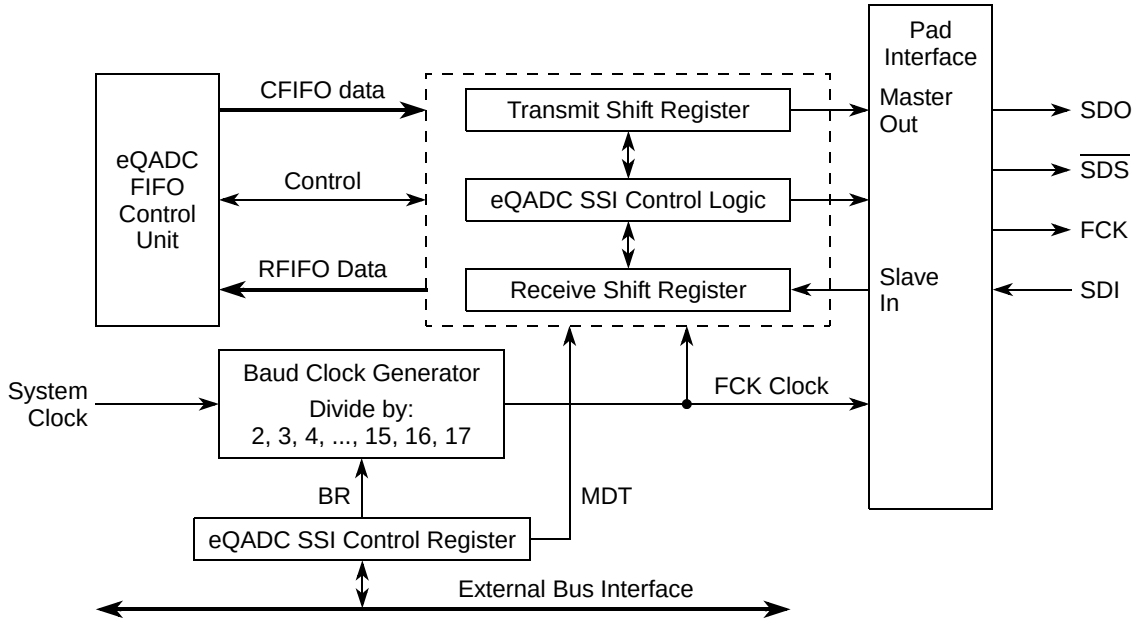


Figure 18-54. eQADC Synchronous Serial Interface Block Diagram

The eQADC SSI protocol allows for a full duplex, synchronous, serial communication between the eQADC and a single external device. Figure 18-54 shows the different components inside the eQADC SSI. The eQADC SSI submodule on the eQADC is always configured as a master. The eQADC SSI has four associated port pins:

- Free running clock (FCK)
- Serial data select ($\overline{\text{SDS}}$)
- Serial data in (SDI)
- Serial data out (SDO)

The FCK clock signal times the shifting and sampling of the two serial data signals and it is free running between transmissions, allowing it to be used as the clock for the external device. The $\overline{\text{SDS}}$ signal is asserted to indicate the start of a transmission, and negated to indicate the end or the abort of a transmission. SDI is the master serial data input and SDO the master serial data output.

The eQADC SSI submodule is enabled by setting the EQADC_MCR[ESSIE] (see Section 18.3.2.1, “eQADC Module Configuration Register (EQADC_MCR)”). When enabled, the eQADC SSI can be optionally capable of starting serial transmissions. When serial transmissions are disabled (ESSIE set to 0b10), no data is transmitted to the external device but FCK is free-running. This operation mode permits the control of the timing of the first serial transmission, and can be used to avoid the transmission of data to an unstable external device, for example, a device that is not fully reset. This mode of operation is specially important for the reset procedure of an external device that uses the FCK as its main clock.

The main elements of the eQADC SSI are the shift registers. The 26-bit transmit shift register in the master and 26-bit receive shift register in the slave are linked by the SDO pin. In a similar way, the 26-bit transmit shift register in the slave and 26-bit receive shift register in the master are linked by the SDI pin. See

<f-helvetica>Figure 18-55.. When a data transmission operation is performed, data in the transmit registers is serially shifted twenty-six bit positions into the receive registers by the FCK clock from the master; data is exchanged between the master and the slave. Data in the master transmit shift register in the beginning of a transmission operation becomes the output data for the slave, and data in the master receive shift register after a transmission operation is the input data from the slave.

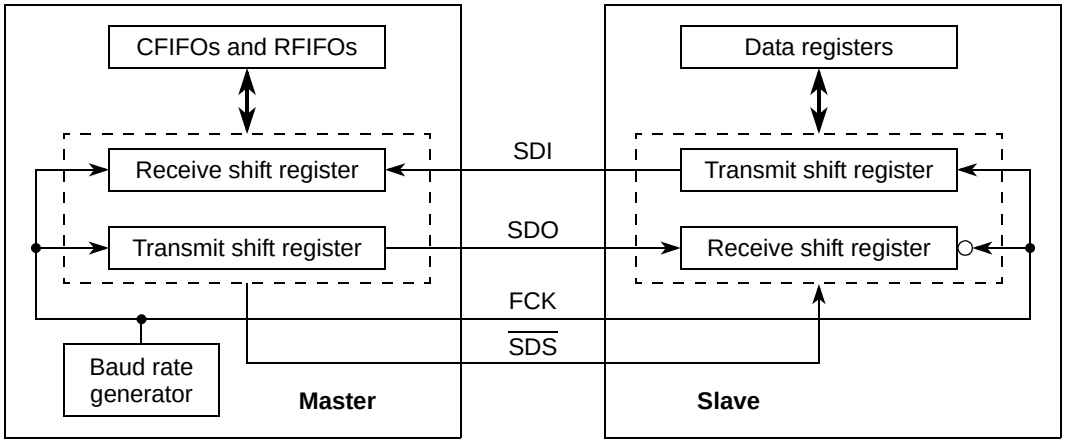


Figure 18-55. Full Duplex Pin Connection

18.4.8.1 eQADC SSI Data Transmission Protocol

Figure 18-56 shows the timing of an eQADC SSI transmission operation. The main characteristics of this protocol are the following:

- FCK is free running, it does not stop between data transmissions. FCK is driven low:
  - When the serial interface is disabled
  - In stop/debug mode
  - Immediately after reset
- Frame size is fixed to 26 bits.
- Msb bit is always transmitted first.
- Master drives data on the positive edge of FCK and latches incoming data on the next positive edge of FCK.
- Slave drives data on the positive edge of FCK and latches incoming data on the negative edge of FCK.

Master initiates a data transmission by driving $\overline{SDS}$ low, and its msb bit on SDO on the positive edge of FCK. After an asserted $\overline{SDS}$ is detected, the slave shifts its data out, one bit at a time, on every FCK positive edge. Both the master and the slave drive new data on the serial lines on every FCK positive edge. This process continues until all the initial 26-bits in the master shift register are moved into the slave shift register. t_{DT} is the delay between two consecutive serial transmissions, time during which $\overline{SDS}$ is negated. When ready to start of the next transmission, the slave must drive the msb bit of the message on every positive edge of FCK regardless of the state of the $\overline{SDS}$ signal. On the next positive edge, the second bit of the message is conditionally driven according to if an asserted $\overline{SDS}$ was detected by the slave on the preceding FCK negative edge. This is an important requisite since the $\overline{SDS}$ and the FCK are not

synchronous. The $\overline{SDS}$ signal is not generated by FCK, rather both are generated by the system clock, so that it is not guaranteed that FCK edges precede $\overline{SDS}$ edges. While $\overline{SDS}$ is negated, the slave continuously drives its msb bit on every positive edge of FCK until it detects an asserted $\overline{SDS}$ on the immediately next FCK negative edge. See Figure 18-57 for three cases that show how the slave operates when $\overline{SDS}$ is asserted.

NOTE

On the master, the FCK is not used as a clock. Although, the eQADC SSI behavior is described in terms of the FCK positive and negative edges, all eQADC SSI related signals (SDI, SDS, SDO, and FCK) are synchronized by the system clock on the master side. There are no restrictions regarding the use of the FCK as a clock on the slave device.

18.4.8.1.1 Abort Feature

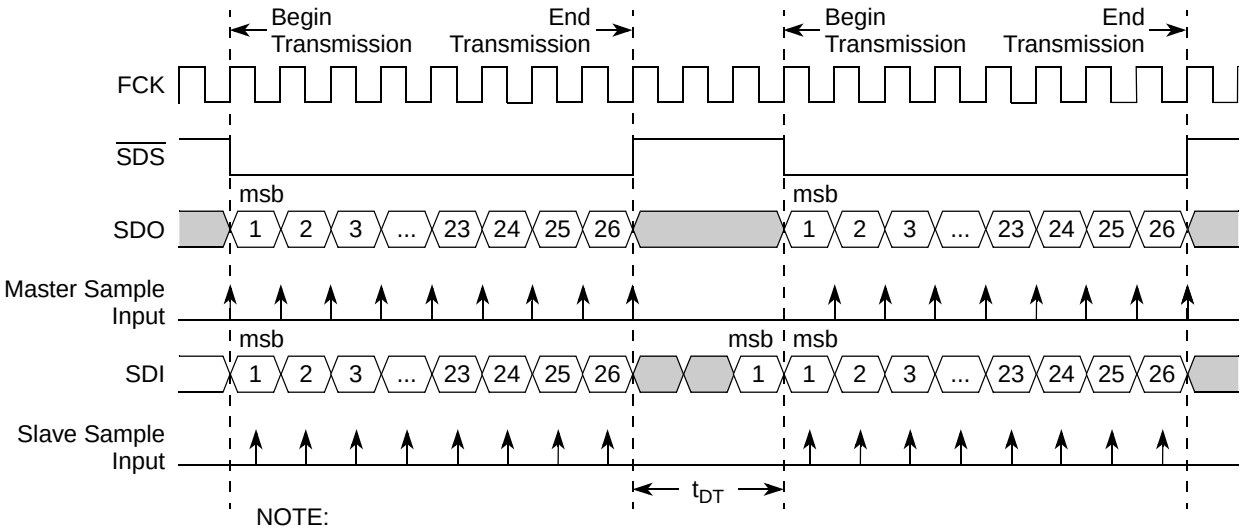
The master indicates it is ending the current transfer by negating $\overline{SDS}$ before the whole data frame has being shifted out, that is the 26th bit of data being transferred has not being shifted out. The eQADC ignores the incompletely received message. The eQADC re-sends the aborted message whenever the corresponding CFIFO becomes again the highest priority CFIFO with commands bound for an external command buffer that is not full. See Section 18.4.3.2, “CFIFO Prioritization and Command Transfer,” for more information on aborts and CFIFO priority.

18.4.8.2 Baud Clock Generation

As shown in Figure 18-54, the baud clock generator divides the system clock to produce the baud clock. The EQADC_SSICR[BR] field (see Section 18.3.2.12, “eQADC SSI Control Register EQADC_SSICR”) selects the system clock divide factor as in Table 18-18. [1]

$$\text{BaudClockFrequency} = \frac{\text{SystemClockFrequency(MHz)}}{\text{SystemClockDivideFactor}}$$

1. Maximum FCK frequency is highly dependable on track delays, master pad delays, and slave pad delays.



NOTE:
t_{MDT} = Minimum t_{DT} is programmable and defined in Section 18.3.2.12, 'eQADC SSI Control Register (EQADC_SSICR).'

Figure 18-56. Synchronous Serial Interface Protocol Timing

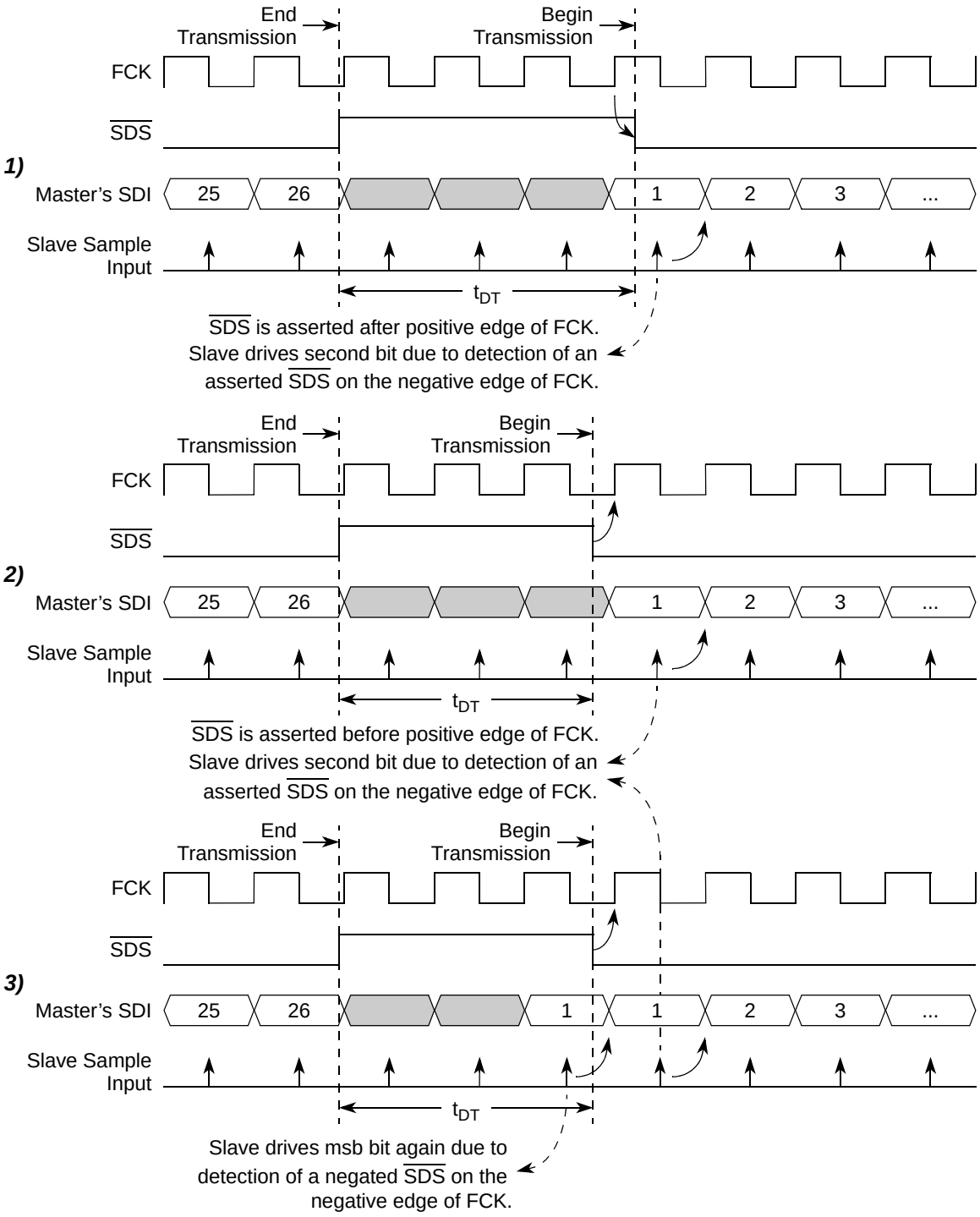


Figure 18-57. Slave Driving the msb and Consecutive Bits in a Data Transmission

18.4.9 Analog Submodule

18.4.9.1 Reference Bypass

The reference bypass capacitor (REFBYPC) signal requires a 100 nF capacitor connected to V_{RL} to filter noise on the internal reference used by the ADC.

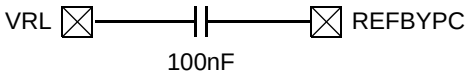


Figure 18-58. Reference Bypass Circuit

18.4.9.2 Analog-to-Digital Converter (ADC)

18.4.9.2.1 ADC Architecture

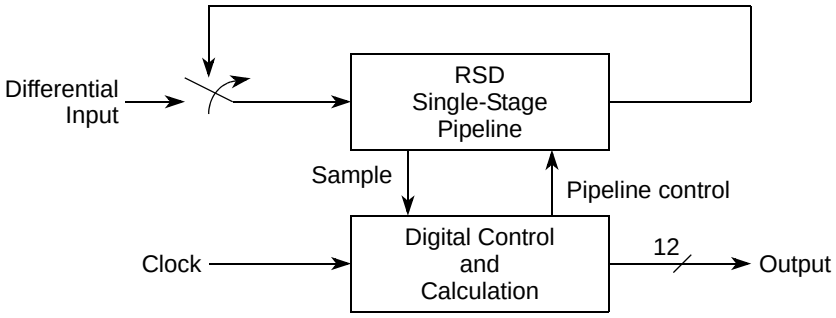


Figure 18-59. RSD ADC Block Diagram

The redundant signed digit (RSD) cyclic ADC consists of two main portions, the analog RSD stage, and the digital control and calculation module, as shown in Figure 18-59. To begin an analog-to-digital conversion, a differential input is passed into the analog RSD stage. The signal is passed through the RSD stage, and then from the RSD stage output, back to its input to be passed again. To complete a 12-bit conversion, the signal must pass through the RSD stage 12 times. Each time an input signal is read into the RSD stage, a digital sample is taken by the digital control/calculation module. The digital control/calculation module uses this sample to tell the analog module how to condition the signal. The digital module also saves each successive sample and adds them according to the RSD algorithm at the end of the entire conversion cycle.

18.4.9.2.2 RSD Overview

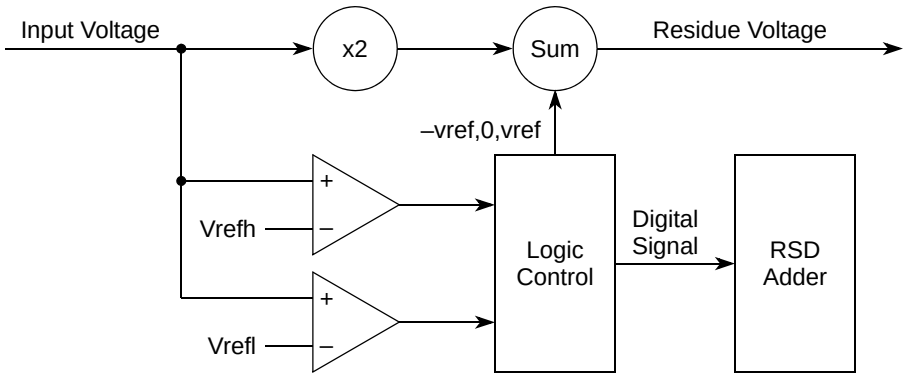


Figure 18-60. RSD Stage Block Diagram

On each pass through the RSD stage, the input signal are multiplied by exactly two, and summed with either –vref, 0, or vref, depending on the logic control. The logic control determines –vref, 0, or vref, depending on the two comparator inputs. As the logic control sets the summing operation, it also sends a digital value to the RSD adder. Each time an analog signal passes through the RSD single-stage, a digital value is collected by the RSD adder. At the end of an entire AD conversion cycle, the RSD adder uses these collected values to calculate the 12-bit digital output.

Figure 18-61 shows the transfer function for the RSD stage. Note how the digital value (AB) is dependent on the two comparator inputs.

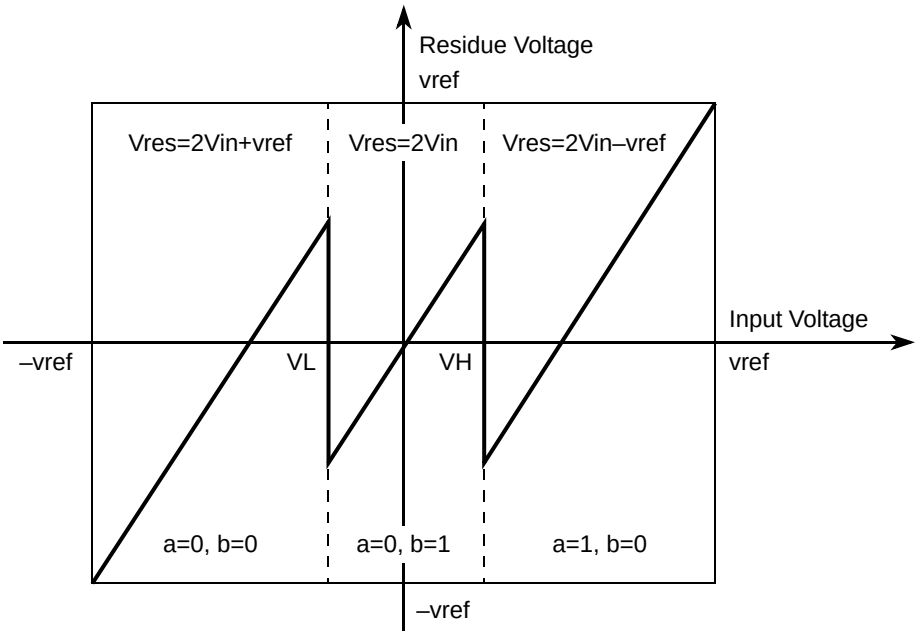


Figure 18-61. RSD Stage Transfer Function

In each pass through the RSD stage, the remainder is sent back as the new input, and the digital signals, a and b, are stored. For the 12-bit ADC, the input signal is sampled during the input phase, and after each of the 12 passes through the RSD stage. Thus, 13 total a and b values are collected. Upon collecting all these

values, they are added according to the RSD algorithm to create the 12-bit digital representation of the original analog input.

18.4.9.2.3 RSD Adder

The array, s1 to s12, are the digital output of the RSD ADC with s1 being the MSB (most significant bit) and s12 being the LSB (least significant bit).

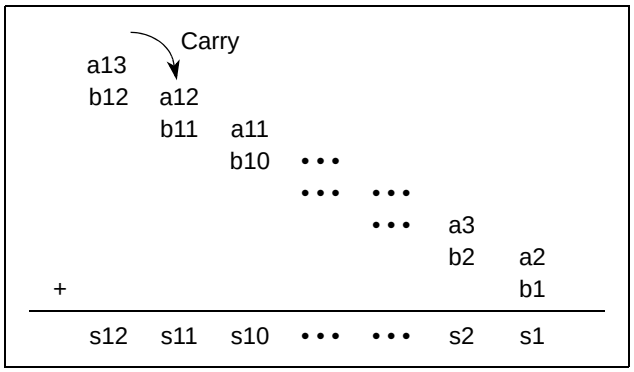


Figure 18-62. RSD Adder

18.5 Initialization and Application Information

18.5.1 Multiple Queues Control Setup Example

This section provides an example of how to configure multiple user command queues. Table 18-53 describes how each queue can be used for a different application. Also documented in this section are general guidelines on how to initialize the on-chip ADCs and the external device, and how to configure the command queues and the eQADC.

Table 18-53. Example Applications of Each Command Queue

Command Queue Number	Queue Type	Running Speed	Number of Contiguous Conversions	Example
0	Very fast burst time-based queue	Every 2 μs for 200 μs; pause for 300 μs and then repeat	2	Injector current profiling
1	Fast hardware-triggered queue	Every 900 μs	3	Current sensing of PWM controlled actuators
2	Fast repetitive time-based queue	Every 2 ms	8	Throttle position
3	Software-triggered queue	Every 3.9 ms	3	Command triggered by software strategy
4	Repetitive angle-based queue	Every 625 μs	7	Airflow read every 30 degrees at 8000 RPM
5	Slow repetitive time-based queue	Every 100 ms	10	Temperature sensors

18.5.1.1 Initialization of On-Chip ADCs and an External Device

The following steps provide an example of configuring the eQADC to initialize the on-chip ADCs and the external device. In this example, commands are sent through CFIFO0.

1. Load all required configuration commands in the RAM in such way that they form a queue; this data structure is referred to below as Queue0. Figure 18-63 shows an example of a command queue able to configure the on-chip ADCs and external device at the same time.
2. Configure Section 18.3.2.2, "eQADC Null Message Send Format Register (EQADC_NMSFR)."
3. Configure Section 18.3.2.12, "eQADC SSI Control Register EQADC_SSICR," to communicate with the external device.
4. Enable the eQADC SSI by programming the ESSIE field the Section 18.3.2.1, "eQADC Module Configuration Register (EQADC_MCR)."
   a) Write 0b10 to ESSIE field to enable the eQADC SSI. FCK is free running but serial transmissions are not started.
   b) Wait until the external device becomes stable after reset.
   c) Write 0b11 to ESSIE field to enable the eQADC SSI to start serial transmissions.
5. Configure the eDMA to transfer data from Queue0 to CFIFO0 in the eQADC.
6. Configure Section 18.3.2.7, "eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)."
   a) Set CFFS0 to configure the eQADC to generate an eDMA request to load commands from Queue0 to the CFIFO0.
   b) Set CFFE0 to enable the eQADC to generate an eDMA request to transfer commands from Queue0 to CFIFO0; Command transfers from the RAM to the CFIFO0 starts immediately.
   c) Set EOQIE0 to enable the eQADC to generate an interrupt after transferring all of the commands of Queue0 through CFIFO0.
7. Configure Section 18.3.2.6, "eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)."
   a) Write 0b0001 to the MODE0 field in eQADC_CFCR0 to program CFIFO0 for software single-scan mode.
   b) Write 1 to SSE0 to assert SSS0 and trigger CFIFO0.
8. Because CFIFO0 is in single-scan software mode and it is also the highest priority CFIFO, the eQADC starts to transfer configuration commands to the on-chip ADCs and to the external device.
9. When all of the configuration commands are transferred, EQADC_FISR*n*[CF0] is set. See Section 18.3.2.8, "eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)." The eQADC generates an end of queue interrupt. The initialization procedure is complete.

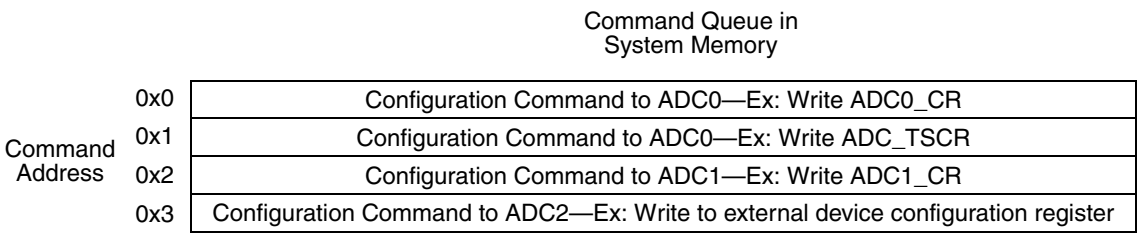


Figure 18-63. Example of a Command Queue Configuring the On-Chip ADCs/External Device

18.5.1.2 Configuring eQADC for Applications

This section provides an example based on the applications in Table 18-53. The example describes how to configure multiple command queues to be used for those applications and provides a step-by-step procedure to configure the eQADC and the associated command queue structures. In the example, the "Fast hardware-triggered command queue," described on the second row of Table 18-53, transfer its commands to ADC1; the conversion commands are executed by ADC1. The generated results are returned to RFIFO3 before being transferred to the result queues in the RAM by the eDMA.

NOTE

There is no fixed relationship between CFIFOs and RFIFOs with the same number. The results of commands being transferred through CFIFO1 can be returned to any RFIFO, regardless of its number. The destination of a result is determined by the MESSAGE_TAG field of the command that requested the result. See Section 18.4.1.2, "Message Format in eQADC," for details.

Step One: Set up the command queues and result queues.

1. Load the RAM with configuration and conversion commands. Table 18-54 is an example of how to set commands for command queue 1.
   a) Each trigger event causes four commands to be executed. When the eQADC detects the pause bit asserted, it waits for another trigger to restart transferring commands from the CFIFO.
   b) At the end of the command queue, the "EOQ" bit is asserted as shown in Table 18-54.
   c) Results are returned to RFIFO3 as specified in the MESSAGE_TAG field of commands.
2. Reserve memory space for storing results.

Table 18-54. Example of Command Queue Commands [1]

	0	1	2 3	4	5	6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
	EOQ	PAUSE	RESERVED	ABORT_ST	EB (0b1)	BN	CAL	MESSAGE_TAG	ADC COMMAND
CMD1	0	0	0	0	0	1	0	0b0011	Conversion command
CMD2	0	0	0	0	0	1	0	0b0011	Conversion command
CMD3	0	0	0	0	0	1	0	0b0011	Conversion command
CMD4	0	1	0	0	0	1	0	0b0011 [2]	Configure peripheral device for next conversion sequence
CMD5	0	0	0	0	0	1	0	0b0011	Conversion command
CMD6	0	0	0	0	0	1	0	0b0011	Conversion command
CMD7	0	0	0	0	0	1	0	0b0011	Conversion command
CMD8	0	1	0	0	0	1	0	0b0011[2]	Configure peripheral device for next conversion sequence
≈								etc.....	≈
	CFIFO header								ADC command

Table 18-54. Example of Command Queue Commands [1] (continued)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	EOQ	PAUSE	RESERVED		ABORT_ST	EB (0b1)	BN	CAL	MESSAGE_ TAG				ADC COMMAND																			
CMDEOQ	1	0	0		0	0	1	0	0b0011				EOQ message																			
	CFIFO header								ADC command																							

[1] Fields LST, TSR, FMT, and CHANNEL_NUMBER are not shown for clarity. See Section , “Conversion Command Message Format for On-Chip ADC Operation,” for details.

[2] MESSAGE_TAG field is only defined for read configuration commands.

Step Two: Configure the eDMA to handle data transfers between the command/result queues in RAM and the CFIFOs/RFIFOs in the eQADC.

1. For transferring, set the source address of the eDMA TCD*n* to point to the start address of command queue 1. Set the destination address of the eDMA to point to EQADC_CFPR1. See Section 18.3.2.4, “eQADC CFIFO Push Registers 0–5 (EQADC_CFPRn).”
2. For receiving, set the source address of the eDMA TCD*n* to point to EQADC_RFPR3. See Section 18.3.2.5, “eQADC Result FIFO Pop Registers 0–5 (EQADC_RFPRn).” Set the destination address of the eDMA to point to the starting address of result queue 1.

Step Three: Configure the eQADC control registers.

1. Configure Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn).”
   a) Set EOQIE1 to enable the End of Queue Interrupt request.
   b) Set CFFS1 and RFDS3 to configure the eQADC to generate eDMA requests to push commands into CFIFO1 and to pop result data from RFIF03.
   c) Set CFINV1 to invalidate the contents of CFIFO1.
   d) Set RFDE3 and CFFE1 to enable the eQADC to generate eDMA requests. Command transfers from the RAM to the CFIFO1 starts immediately.
   e) Set RFOIE3 to indicate if RFIFO3 overflows.
   f) Set CFUIE1 to indicate if CFIFO1 underflows.
2. Configure MODE1 to continuous-scan rising edge external trigger mode in Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn).”

Step Four: Command transfer to ADCs and result data reception.

When an external rising edge event occurs for CFIFO1, the eQADC automatically begins transferring commands from CFIFO1 when it becomes the highest priority CFIFO trying to send commands to ADC1. The received results are placed in RFIFO3 and then moved to result queue 1 by the eDMA.

18.5.2 eQADC to eDMA Controller Interface

This section provides an overview of the eQADC/eDMA interface and general guidelines about how to configure the eDMA to correctly transfer data between the queues in system memory and the eQADC FIFOs.

18.5.2.1 Command Queue and CFIFO Transfers

In transfers involving command queues and CFIFOs, the eDMA moves data from a queued source to a single destination as shown in Figure 18-64.

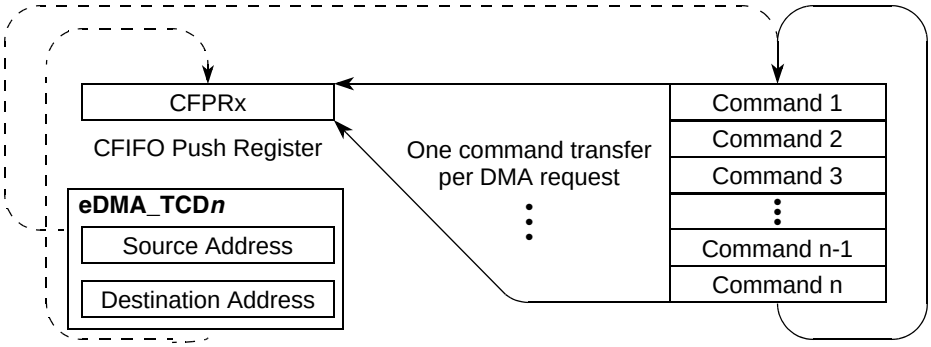


Figure 18-64. Command Queue/CFIFO Interface

The location of the data to be moved is indicated by the source address, and the final destination for that data, by the destination address. The eDMA has transfer control descriptors (TCDs) containing these addresses and other parameters used in the control of data transfers.

See Section 9.2.2.17, “Transfer Control Descriptor (TCD)” for more information.

For every eDMA request issued by the eQADC, the eDMA must be configured to transfer a single command (32-bit data) from the command queue, pointed to by the source address, to the CFIFO push register, pointed to by the destination address. After the service of an eDMA request is completed, the source address has to be updated to point to the next valid command. The destination address remains unchanged. When the last command of a queue is transferred, do one of the following:

- Disable the eDMA channel. This might be desirable for CFIFOs in single scan mode.
- Update the source address to point to a valid command for the first command in the queue was transferred (cyclic queue), or the first command of any other command queue. This is desirable for CFIFOs in continuous scan mode, or in some cases, for CFIFOs in single-scan mode.

See Chapter 9, “Enhanced Direct Memory Access (eDMA)” for details about how this functionality is supported.

18.5.2.2 Receive Queue/RFIFO Transfers

In transfers involving receive queues and RFIFOs, the eDMA controller moves data from a single source to a queue destination as shown in Figure 18-65. The location of the data to be moved is indicated by the source address, and the final destination for that data, by the destination address. For every eDMA request issued by the EQADC, the eDMA controller has to be configured to transfer a single result (16-bit data),

pointed to by the source address, from the RFIFO pop register to the receive queue, pointed to by the destination address. After the service of an eDMA request is completed, the destination address has to be updated to point to the location where the next 16-bit result are stored. The source address remains unchanged. When the last expected result is written to the receive queue, do one of the following:

- Disable the eDMA channel.
- Update the destination address to point to the next location where in-coming results are stored: the first entry of the current receive queue (cyclic queue); or the beginning of a new receive queue.

See Chapter 19, “Deserial Serial Peripheral Interface (DSPI)” for details about how this functionality is supported.

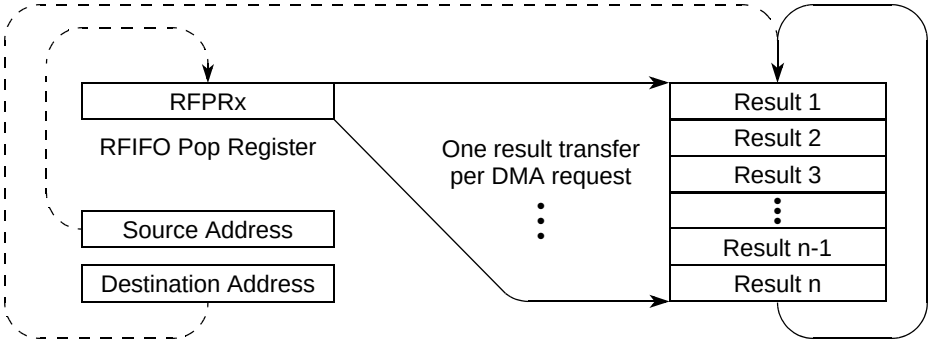


Figure 18-65. Receive Queue/RFIFO Interface

18.5.3 Sending Immediate Command Setup Example

In the eQADC, there is no immediate command register for sending a command immediately after writing to that register. However, a CFIFO can be configured to perform the same function as an immediate command register. The following steps illustrate how to configure CFIFO5 as an immediate command CFIFO. This eliminates the use of the eDMA. The results are returned to RFIFO5.

1. Configure the Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn).”
   a) Clear CFIFO fill enable5 (CFFE5 = 0) in EQADC_IDCR5.
   b) Clear CFIFO underflow interrupt enable5 (CFUIE5 = 0) in EQADC_IDCR2.
   c) Clear RFDS5 to configure the eQADC to generate interrupt requests to pop result data from RFIF05.
   d) Set RFIFO drain enable5 (RFDE5 = 1) in EQADC_IDCR5.
2. Configure the Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn).”
   a) Write 1 to CFINV5 in EQADC_CFCR5. This invalidates the contents of CFIFO5.
   a) Set MODE5 to continuous-scan software trigger mode in EQADC_CFCR5.
3. To transfer a command, write it to the eQADC CFIFO push register 5 (EQADC_CFPR5) with message tag = 0b0101. See Section 18.3.2.4, “eQADC CFIFO Push Registers 0–5 (EQADC_CFPRn).”

4. Up to 4 commands can be queued in CFIFO5. Check the CFCTR5 status in EQADC_FISR5 before pushing another command to avoid overflowing the CFIFO. See Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn).”
5. When the eQADC receives a conversion result for RFIFO5, it generates an interrupt request. RFIFO pop register 5 (EQADC_RFPR5) can be popped to read the result. See Section 18.3.2.5, “eQADC Result FIFO Pop Registers 0–5 (EQADC_RFPRn).”

18.5.4 Modifying Queues

Some applications require more command queues than the six supported by the eQADC. These additional command queues can be supported by interrupting command transfers from a configured CFIFO, even if it is triggered and transferring, modifying the corresponding command queue in the RAM or associating another command queue to it, and restarting the CFIFO. More details on disabling a CFIFO are described in Section 18.4.3.4.1, “Disabled Mode.”

1. Determine the resumption conditions when later resuming the scan of the command queue at the point before it was modified.
   a) Change EQADC_CFCRn[MODE*n*] (see Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”) to disabled. See Section 18.4.3.4.1, “Disabled Mode,” for a description of what happens when MODE*n* is changed to disabled.
   b) Poll EQADC_CFSR[CFS*n*] until it becomes IDLE (see Section 18.3.2.11, “eQADC CFIFO Status Register EQADC_CFSR”).
   c) Read and save EQADC_CFTCRn[TC_CF*n*] (see Section 18.3.2.9, “eQADC CFIFO Transfer Counter Registers 0–5 (EQADC_CFTCRn)”) for later resuming the scan of the queue. The TC_CF*n* provides the point of resumption.
   d) Because all the result data may not have been stored in the RFIFO at the time MODE*n* is changed to disable, wait for all expected results to be stored in the RFIFO/result queue before reconfiguring the eDMA to work with the modified result queue. The number of results that must return can be estimated from the TC_CF*n* value obtained above.
2. Disable the eDMA from responding to the eDMA request generated by EQADC_FISRn[CFFF*n*] and EQADC_FISRn[RFDF*n*] (see Section 18.3.2.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”).
3. Write “0x0000” to the TC_CF*n* field.
4. Load the new configuration and conversion commands into RAM. Configure the eDMA to support the new command/result queue, but do not configure it yet to respond to eDMA requests from CFIFO*n*/RFIFO*n*.
5. If necessary, modify the EQADC_IDCRn registers (see Section 18.3.2.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) to suit the modified command queue.
6. Write 1 to EQADC_CFCRn[CFINV*n*] (see Section 18.3.2.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”) to invalidate the entries of CFIFO*n*.
7. Configure the eDMA to respond to eDMA requests generated by CFFF*n* and RFDF*n*.
8. Change MODE*n* to the modified CFIFO operation mode. Write 1 to SSE*n* to trigger CFIFO*n* if MODE*n* is software trigger.

18.5.5 Command Queue and Result Queue Usage

Figure 18-66 is an example of command queue and result queue usage. It shows the command queue 0 commands requesting results that are stored in result queue 0 and result queue 1, and command queue 1 commands requesting results that are stored only in result queue 1. Some command messages request data to be returned from the on-chip ADC/external device, but some only configure them and do not request returning data. When a command queue contains both write and read commands like command queue 0, the command queue and result queue entries are not aligned, in Figure 18-66, the result for the second command of command queue 0 is the first entry of result queue 0. The figure also shows that command queue and result queue entries can also become unaligned even if all commands in a command queue request data as command queue 1. Command queue 1 entries became unaligned to result queue 1 entries because a result requested by the forth command queue 0 command was sent to result queue 1. This happens because the system can be configured so that several command queues can have results sent to a single result queue.

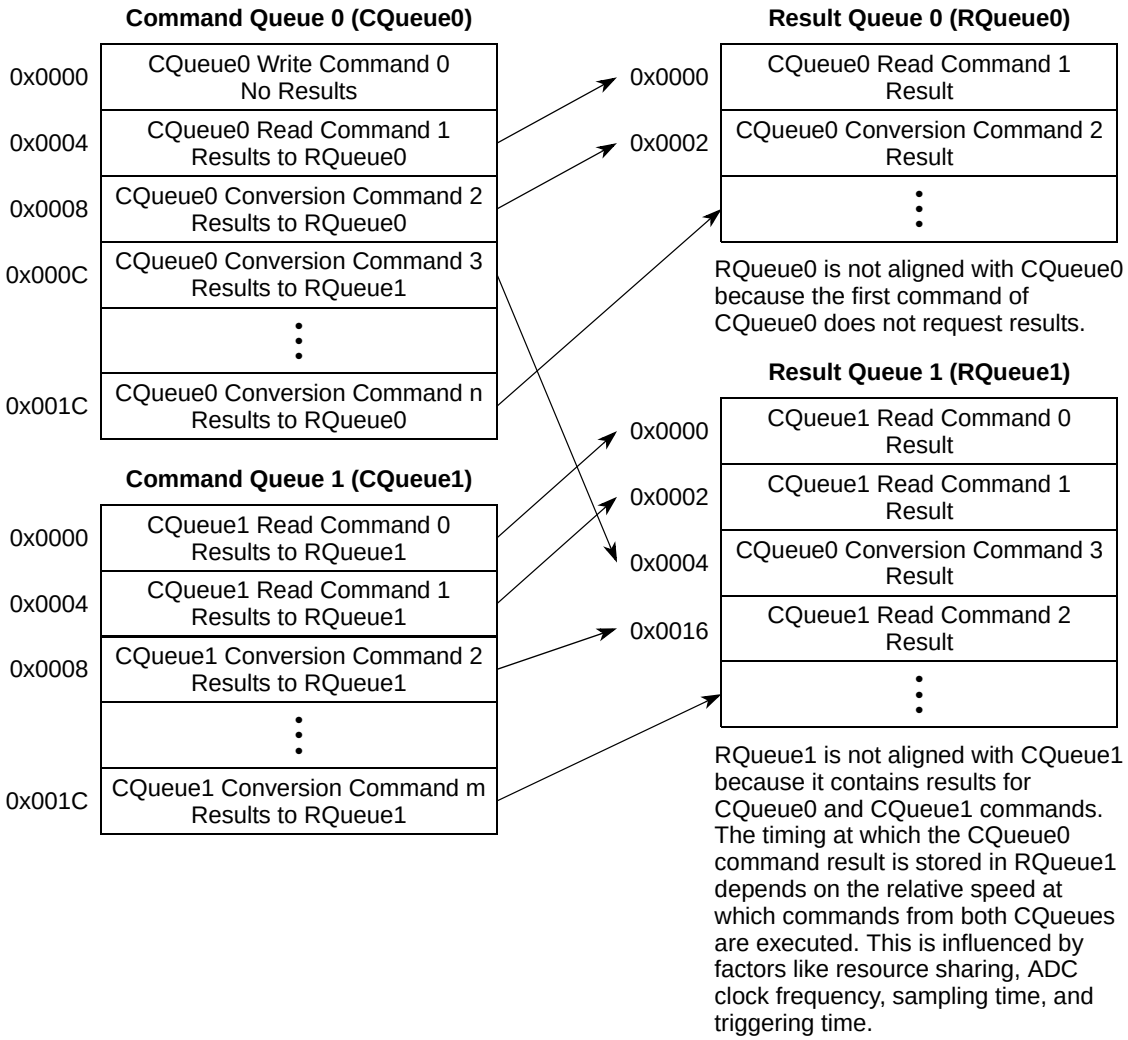


Figure 18-66. eQADC Command and Result Queues

18.5.6 ADC Result Calibration

The ADC result calibration process consists of two steps: determining the gain and offset calibration constants, and calibrating the raw results generated by the on-chip ADCs by solving the following equation discussed in Section 18.4.5.4.1, “Calibration Overview.”

$$CAL\_RES = GCC \times RAW\_RES + OCC + 2; \quad \textit{Eqn. 18-1}$$

The calibration constants GCC and OCC can be calculated from Equation 18-1 provided that two pairs of expected (CAL_RES) and measured (RAW_RES) result values are available for two different input voltages. Most likely calibration points to be used are 25% V_{REF}[1] and 75% V_{REF} since they are far apart but not too close to the end points of the full input voltage range. This allows for calculations of more representative calibration constants. The eQADC provides these voltages via channel numbers 43 and 44. The raw, uncalibrated results for these input voltages are obtained by converting these channels with conversion commands that have the CAL bit negated.

The transfer equations for when sampling these reference voltages are:

$$CAL\_RES_{75\%VREF} = GCC \times RAW\_RES_{75\%VREF} + OCC + 2; \quad \textbf{\textit{Eqn. 18-2}}$$

$$CAL\_RES_{25\%VREF} = GCC \times RAW\_RES_{25\%VREF} + OCC + 2; \quad \textbf{\textit{Eqn. 18-3}}$$

Thus;

$$GCC = (CAL\_RES_{75\%VREF} - CAL\_RES_{25\%VREF}) / (RAW\_RES_{75\%VREF} - RAW\_RES_{25\%VREF}); \quad \textbf{\textit{Eqn. 18-4}}$$

$$OCC = CAL\_RES_{75\%VREF} - GCC \times RAW\_RES_{75\%VREF} - 2; \quad \textbf{\textit{Eqn. 18-5}}$$

or

$$OCC = CAL\_RES_{25\%VREF} - GCC \times RAW\_RES_{25\%VREF} - 2; \quad \textbf{\textit{Eqn. 18-6}}$$

After being calculated, the GCC and OCC values must be written to ADC0_GCCR and ADC1_GCCR registers (see Section 18.3.3.4, “ADCn Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)”) and the ADC0_OCCR and ADC1_OCCR registers (see Section 18.3.3.5, “ADCn Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)”) using write configuration commands.

The eQADC automatically calibrates the results, according to Equation 18-1, of every conversion command that has its CAL bit asserted using the GCC and OCC values stored in the ADC calibration registers.

For more information on calibrating the eQADC, see applications note AN2989 ‘Design, Accuracy, and Calibration of Analog to Digital Converters on the MPC5500 Family’ available from www.freescale.com.

1. $V_{REF} = V_{RH} - V_{RL}$

18.5.6.1 MAC Configuration Procedure

The following steps illustrate how to configure the calibration hardware, that is, determining the values of the gain and offset calibration constants, and the writing these constants to the calibration registers. This procedure must be performed for both ADC0 and ADC1.

1. Convert channel 44 with a command that has its CAL bit negated and obtain the raw, uncalibrated result for 25% V_{REF} ($RAW\_RES_{25\%VREF}$).
2. Convert channel 43 with a command that has its CAL bit negated and obtain the raw, uncalibrated result for 75% V_{REF} ($RAW\_RES_{75\%VREF}$).
3. Because the expected values for the conversion of these voltages are known ($CAL\_RES_{25\%VREF}$ and $CAL\_RES_{75\%VREF}$), GCC and OCC values can be calculated from Equation 18-4 and Equation 18-5 using these values, and the results determined in steps 1 and 2.
4. Reformat GCC and OCC to the proper data formats as specified in Section 18.4.5.4.2, “MAC Unit and Operand Data Format.” GCC is an unsigned 15-bit fixed point value and OCC is a signed 14-bit value.
5. Write the GCC value to ADCn gain calibration registers (see Section Section 18.3.3.4, “ADCn Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)”) and the OCC value to ADCn offset calibration constant registers (see Section Section 18.3.3.5, “ADCn Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)”) using write configuration commands.

18.5.6.2 Example Calculation of Calibration Constants

The raw results obtained when sampling reference voltages 25% V_{REF} and 75% V_{REF} were, respectively, 3798 and 11592. The results obtained from the conversion of these reference voltages are, respectively, 4096 and 12288. Therefore, using Equation 18-4 and Equation 18-5, the gain and offset calibration constants are:

```
GCC=(12288-4096)÷(11592-3798) = 1.05106492-> 1.05102539[1] = 0x4344
    OCC=12288-1.05106492*11592 -2 = 102.06-> 102 = 0x0066
```

Table 18-55 shows, for this particular case, examples of how the result values change according to GCC and OCC when result calibration is executed (CAL=1) and when it is not (CAL=0).

Table 18-55. Calibration Example

Input Voltage	Raw result (CAL=0)		Calibrated result (CAL=1)	
	Hexadecimal	Decimal	Hexadecimal	Decimal
25% VREF	0x0ED6	3798	0x1000	4095.794
75% VREF	0x2D48	11592	0x3000	12287.486

1. This calculation is rounded down due to binary approximation.

18.5.6.3 Quantization Error Reduction During Calibration

Figure 18-67 shows how the ADC transfer curve changes due to the addition of two to the MAC output during the calibration. See MAC output equation in Section 18.4.5.4, “ADC Calibration Feature.” The maximum absolute quantization error is reduced by half leading to an increase in accuracy.

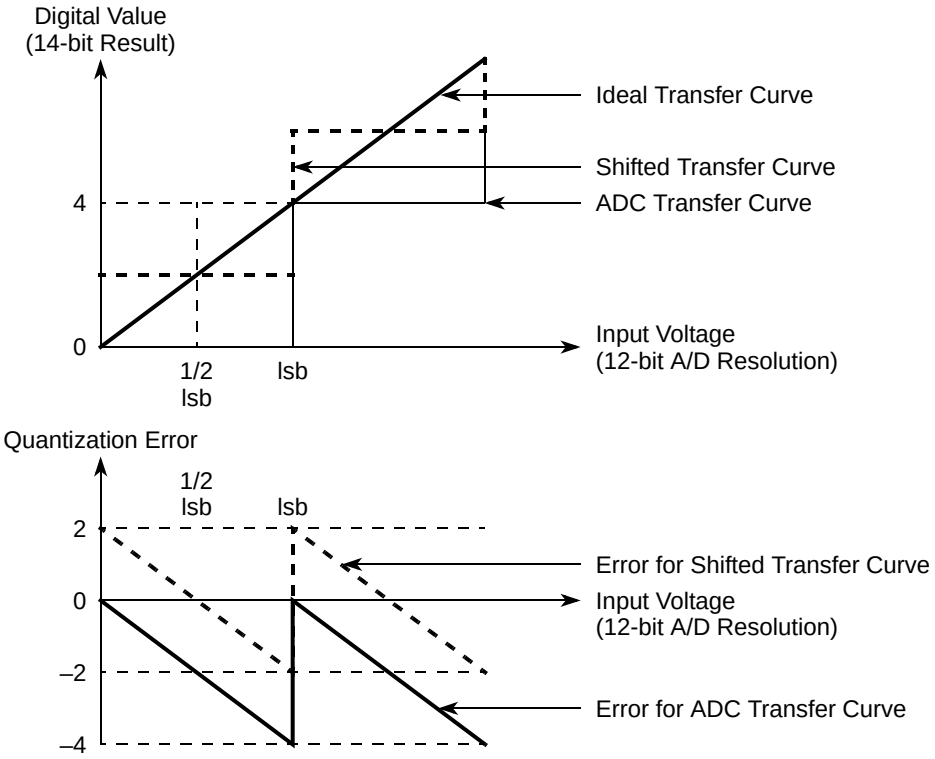


Figure 18-67. Quantization Error Reduction During Calibration

18.5.7 eQADC versus QADC

This section describes how the eQADC upgrades the QADC functionality. The section also provides a comparison between the eQADC and QADC in terms of their functionality. You must be familiar with QADC terminology to fully comprehend the following sections.

Figure 18-68 is an overview of a QADC.

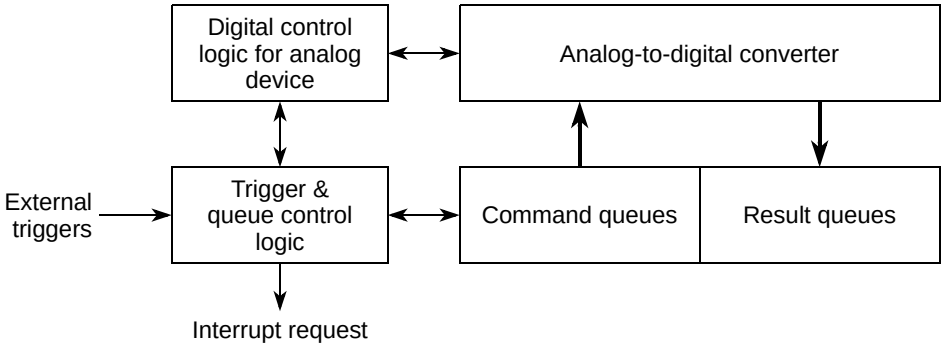


Figure 18-68. QADC Overview

Figure 18-69 is an overview of the eQADC system.

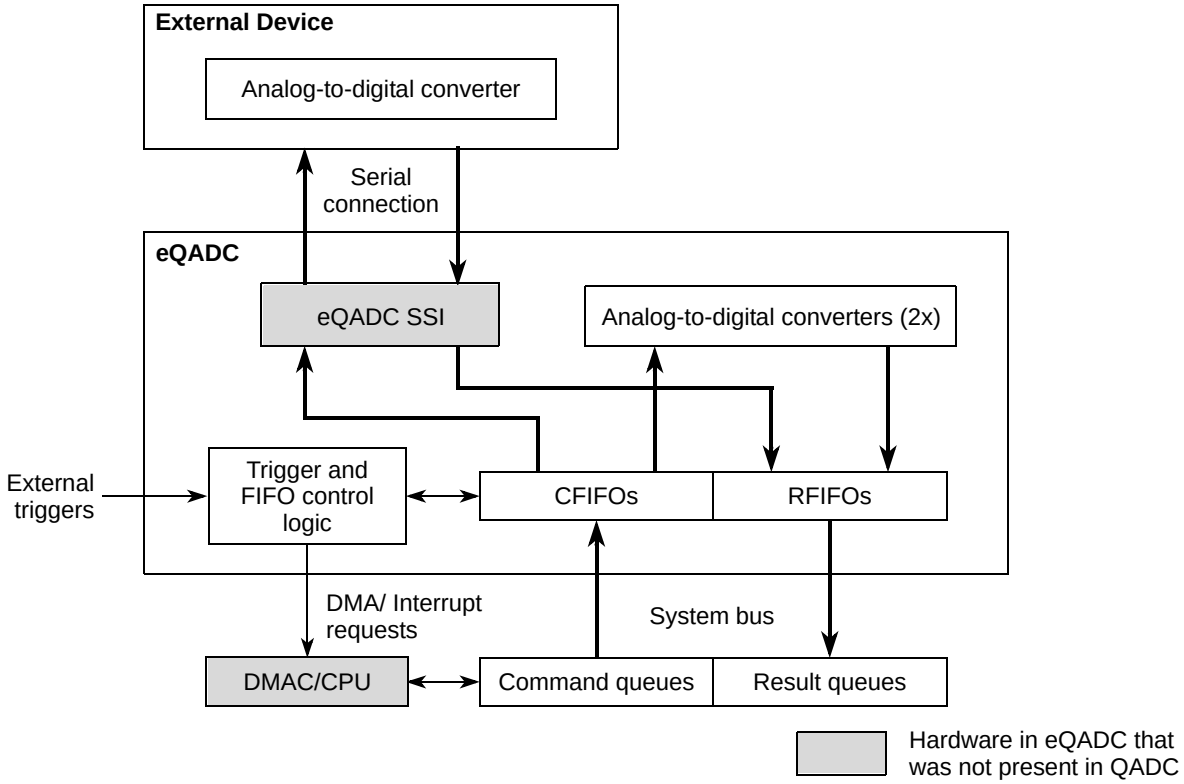


Figure 18-69. eQADC System Overview

The eQADC system consists of four parts: queues in system memory, the eQADC, on-chip ADCs, and an external device. As compared with the QADC, the eQADC system requires two pieces of extra hardware.

1. An eDMA or an MCU is required to move data between the eQADC's FIFOs and queues in the system memory.
2. A serial interface [eQADC synchronous serial interface (SSI)] is implemented to transmit and receive data between the eQADC and the external device.

Because there are only FIFOs inside the eQADC, much of the terminology or use of the register names, register contents, and signals of the eQADC involve FIFO instead of queue. These register names, register contents, and signals are functionally equivalent to the queue counterparts in the QADC. Table 18-56 lists how the eQADC register, register contents, and signals are related to QADC.

Table 18-56. Terminology Comparison between QADC and eQADC

QADC Terminology	eQADC Terminology	Function
CCW	Command Message	In the QADC, the hardware only executes conversion command words. In the eQADC, not all commands are conversion commands; some are configuration commands.
Queue Trigger	CFIFO Trigger	In the QADC, a trigger event is required to start the execution of a queue. In the eQADC, a trigger event is required to start command transfers from a CFIFO. When a CFIFO is triggered and transferring, commands are continuously moved from command queues to CFIFOs. Thus, the trigger event initiates the “execution of a queue” indirectly.
Command Word Pointer Queue *n* (CWPQ*n*)	Counter Value of Commands Transferred from Command FIFO*n* (TC_CF*n*)	In the QADC, CWPQ*n* allows the last executed command on queue *n* to be determined. In the eQADC, the TC_CF*n* value allows the last transferred command on command queue *n* to be determined.
Queue Pause Bit (P)	CFIFO Pause Bit	In the QADC, detecting a pause bit in the CCW pauses the queue execution. In the eQADC, detecting a pause bit in the command pauses command transfers from a CFIFO.
Queue Operation Mode (MQ*n*)	CFIFO Operation Mode (MODE*n*)	The eQADC supports all queue operation modes in the QADC except operation modes related to a periodic timer.
Queue Status (QS)	CFIFO Status (CFS*n*)	In the QADC, the queue status is read to check whether a queue is idle, active, paused, suspended, or trigger pending. In the eQADC, the CFIFO status is read to check whether a queue is IDLE, WAITING FOR TRIGGER (idle or paused in QADC), or triggered (suspended or trigger pending in QADC).

The eQADC and QADC also have similar procedures for the configuration or execution of applications. Table 18-57 shows the steps required for the QADC versus the steps required for the eQADC system.

Table 18-57. Usage Comparison between QADC and eQADC System

Procedure	QADC	eQADC System
Analog Control Configuration	Configure analog device by writing to the QADCs.	Program configuration commands into command queues.
Prepare Scan Sequence	Program scan commands into command queues.	Program scan commands into command queues.
Queue Control Configuration	Write to the QADC control registers.	Write to the eQADC control registers.
Data Transferred between Queues and Buffers	Not Required.	Program the eDMA or the CPU to handle the data transfer.
Serial Interface Configuration	Not Required.	Write to the eQADC SSI registers.
Queue Execution	Require software or external trigger events to start queue execution.	Require software or external trigger events to start command transfers from a CFIFO.

Chapter 19
Deserial Serial Peripheral Interface (DSPI)

19.1 Introduction

This chapter describes the deserial serial peripheral interface (DSPI), which provides a synchronous serial bus for communication between the MCU and an external peripheral device.

Microcontroller chips in the MPC55xx family implement several DSPI module configurations. Most devices implement DSPI B, C, and D; several devices implement B and C; and some devices implement A, B, C, and D. The "*x*" appended to signal names signifies the DSPI module to which the signal applies. Thus PCS*x*[0] specifies that the PCS signal applies to module A, B, and so on.

This device implements DSPI B, C, and D only.

19.1.1 Block Diagram

A block diagram of the DSPI is shown in Figure 19-1.

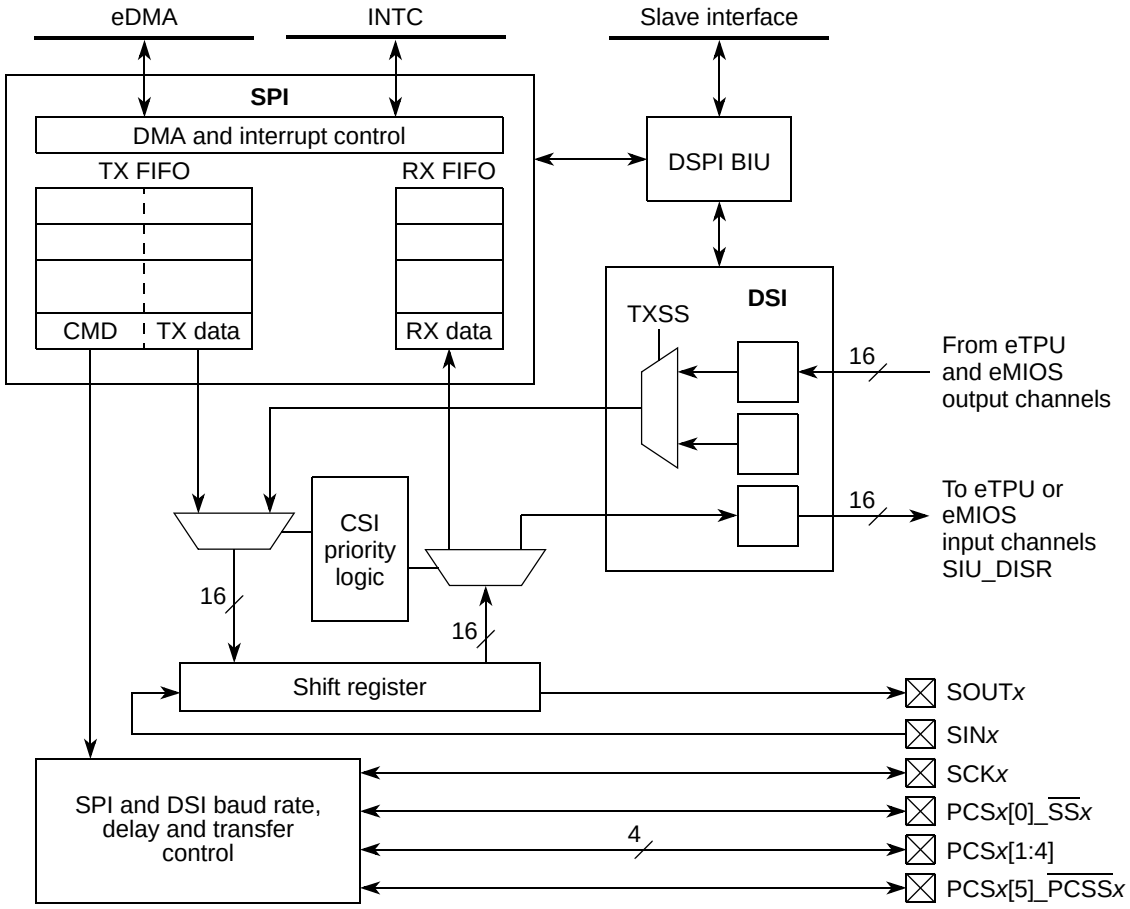


Figure 19-1. DSPI Block Diagram

19.1.2 Overview

The deserial serial interface (DSPI) in this device supports pin count reduction through serialization and deserialization of eTPU channels, two SPI channels, and memory-mapped registers, as well as transfer control capabilities. Incoming serial data triggers an external interrupt request to the DSPI deserialized output connections configured in the SIU. The channels and register content are transmitted using an SPI protocol. There are three identical DSPI modules (DSPI B, DSPI C, and DSPI D).

The DSPI has the following configurations:

- Serial peripheral interface (SPI) configuration where the DSPI operates as a SPI with support for queues.
- Deserial serial interface (DSI) configuration where the DSPI serializes eTPU and eMIOS output channels and deserializes the received data by placing the data on the eTPU and eMIOS input channels and as inputs to the external interrupt request submodule of the SIU.
- Combined serial interface (CSI) configuration where the DSPI operates in both SPI and DSI configurations interleaving DSI frames with SPI frames, giving priority to SPI frames.

For queued operations, the SPI queues reside in internal SRAM, which is external to the DSPI. Data transfers between the queues and the DSPI FIFOs are done using the eDMA controller or host software. Figure 19-2 shows a SPI port servicing external queues in internal SRAM.

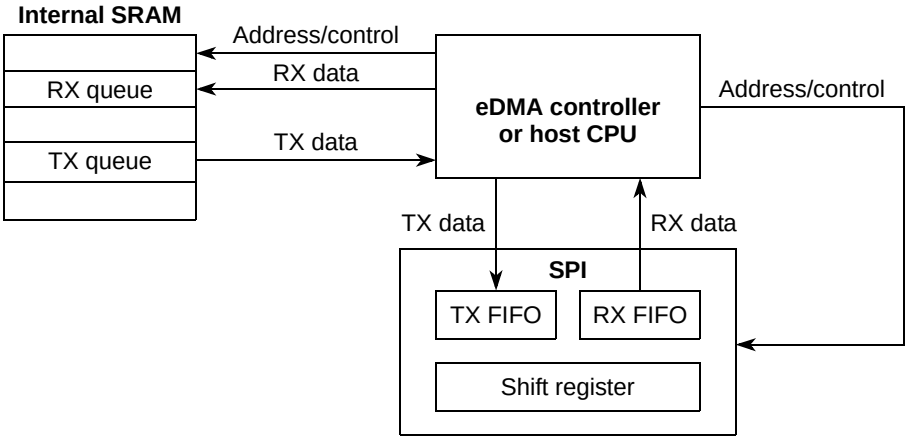


Figure 19-2. DSPI with Queues and eDMA

19.1.3 Features

The DSPI supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave mode
- Buffered transmit and receive operation using the TX and RX FIFOs, with depths of four entries
- Visibility of TX and RX FIFOs to simplify debugging
- FIFO bypass mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis
  - — Eight clock and transfer attribute registers
  - — Serial clock with programmable polarity and phase
  - — Programmable delays
    - – PCS to SCK delay
    - – SCK to PCS delay
    - – Delay between frames

  - Programmable serial frame size of 4 to 16 bits, expandable with software control
  - Continuously held chip select capability
- Six peripheral chip selects, expandable to 64 with external demultiplexer
- Deglitching support for up to 32 peripheral chip selects with external demultiplexer
- Two DMA conditions for SPI queues residing in RAM or flash
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- Six interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - RX FIFO is not empty (RFDF)
  - FIFO overrun
    - Attempt to transmit an empty TX FIFO
    - Serial frame was received while RX FIFO is full (RFOF)
  - FIFO under flow
    - Use for a slave device only running in SPI mode. Request to the slave for a data transfer when the TX FIFO is empty (TFUF)
- Modified SPI transfer formats for communication with slower peripheral devices
- Supports all functional modes from QSPI subblock of QSMCM (MPC500 family)
- Continuous serial communications clock (SCK)

When configured for DSI or CSI operation, the DSPI supports pin reduction through serialization and deserialization.

- Serialized data sources
  - eTPUA, and eMIOS output channels
  - Memory-mapped register in the DSPI
- Deserialized data destinations
  - eTPUA and eMIOS input channels
  - SIU external interrupt request inputs
  - Memory-mapped register in the DSPI
- Transfer initiation conditions
    - Continuous
    - Edge-sensitive hardware trigger
    - Change in data
- Support for parallel and serial chaining of DSPI modules
- Pin serialization/deserialization with interleaved SPI frames for control and diagnostics

19.1.4 Modes of Operation

The DSPI has four modes of operation. These modes can be divided into two categories: module-specific modes such as master, slave and module disable modes; and a second category that is an MCU-specific mode: debug mode.

The module-specific modes are entered by host software writing to a register. The MCU-specific mode is controlled by signals external to the DSPI. The MCU-specific mode is a mode that the entire device can enter, in parallel to the DSPI running in one of its module-specific modes.

19.1.4.1 Master Mode

Master mode allows the DSPI to initiate and control serial communication. In this mode the SCK, PCS*n* and SOUT signals are controlled by the DSPI and configured as outputs.

For more information, see Section 19.4.1.1, “Master Mode.”

19.1.4.2 Slave Mode

Slave mode allows the DSPI to communicate with SPI / DSI bus masters. In this mode, the DSPI responds to externally controlled serial transfers. The DSPI cannot initiate serial transfers in slave mode. In slave mode, the SCK signal and the PCS*x*[0]_$\overline{SS}$ signal are configured as inputs and provided by a bus master. PCS*x*[0]_$\overline{SS}$ must be configured as input and pulled high. If the internal pullup used, the bits in the SIU_PCR for that input must be set (SIU_PCR [WPE = 1], [WPS = 1]).

For more information, see Section 19.4.1.2, “Slave Mode.”

19.1.4.3 Module Disable Mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI*x*_MCR is set.

For more information, see Section 19.4.1.3, “Module Disable Mode.”

19.1.4.4 Debug Mode

Debug mode is used for system development and debugging. If the device enters debug mode while the FRZ bit in the DSPI*x*_MCR is set, the DSPI halts operation on the next frame boundary. If the device enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI.

For more information, see Section 19.4.1.4, “Debug Mode.”

19.2 External Signal Description

19.2.1 Signal Overview

Table 19-1 lists DSPI signals used to communicate with an external device.

Table 19-1. Signal Properties

Name	I/O Type	Function	
		Master Mode	Slave Mode
PCS*x*[0]_$\overline{SS}$	Output / input	Peripheral chip select 0	Slave select
PCS*x*[1:3]	Output	Peripheral chip select 1–3	Unused [1]
PCS*x*[4]_$\overline{MTRIG}$	Output	Peripheral chip select 4	Unused [1]_Master trigger [2]
PCS*x*[5]_$\overline{PCSS}$	Output	Peripheral chip select 5 / Peripheral chip select strobe	Unused [1]
SIN*x*	Input	Serial data in	Serial data in
SOUT*x*	Output	Serial data out	Serial data out
SCK*x*	Output / input	Serial clock (output)	Serial clock (input)

[1] The SIU allows you to select alternate or GPIO signals on these pins for the device.

[2] MTRIG is an internal master trigger for a slave device.

19.2.2 Signal Descriptions

19.2.2.1 Peripheral Chip Select / Slave Select PCS*x*[0]_$\overline{SS}$

In master mode, the PCS*x*[0] signal is a peripheral chip select output that determines the slave device for the current transmission.

In slave mode, the $\overline{SS}$ signal is a slave select input signal that allows a SPI master to select the DSPI as the target for transmission. PCS*x*[0]_$\overline{SS}$ must be configured as input and pulled high. If the internal pullup is used then the bits in the SIU_PCR must be set (SIU_PCR [WPE = 1], [WPS = 1]).

Set the input buffer enable (IBE) and the output buffer enable (OBE) bits to 1 in the SIU_PCR for all DSPI chip select PCSx[0] or slave select $\overline{SS}$ function is configured for that pin. When the pin is used for DSPI master mode as a chip select output, set the OBE bit. When the pin is used in DSPI slave mode as a slave select input, set the IBE bit to 1.

See Section 6.4.1.12, “Pad Configuration Registers (SIU_PCR),” for more information.

19.2.2.2 Peripheral Chip Selects 1–3 PCS*x*[1:3]

PCS*x*[1:3] are peripheral chip select output signals in master mode. In slave mode, only the GPIO signal is available.

19.2.2.3 Peripheral Chip Select 4 / Master Trigger PCS*x*[4]_MTRIG

PCS*x*[4] is a peripheral chip select output signal in master mode.In slave mode, it is an internal master output trigger that identifies a change to the serialize data has occurred. If the multiple transfer operation (MTO) is disabled in slave mode, only the GPIO signal is available.

19.2.2.4 Peripheral Chip Select 5 / Peripheral Chip Select Strobe PCS*x*[5]_PCSS

PCSx[5] is a peripheral chip select output signal. When the DSPI is in master mode and the PCSSE bit in the DSPI*x*_MCR is cleared to 0, the PCS*x*[5] signal determines the slave device that receives the transfer.

PCSS is a strobe signal used by external logic for deglitching the PCS signals. When the DSPI is in master mode and the PCSSE bit in the DSPI*x*_MCR is set to 1, the PCSS signal indicates the timing used to decode PCS*x*[0:4] signals, which prevents glitches from occurring.

PCS*x*[5]_PCSS is not used in slave mode.

19.2.2.5 Serial Input (SIN*x*)

SIN*x* is a serial data input signal.

19.2.2.6 Serial Output (SOUT*x*)

SOUT*x* is a serial data output signal.

19.2.2.7 Serial Clock (SCK*x*)

SCK*x* is a serial communication clock signal. In master mode, the DSPI generates the SCK. In slave mode, SCK*x* is an input from an external bus master.

19.2.2.8 Internal Hardware Trigger

The hardware trigger (HT) is an input signal used with Multiple Transfer Operations in DSI Configuration.

In slave mode, the DSPI generates a trigger pulse on the MTRIG pin when a rising- or falling-edge is detected on the HT.

19.3 Memory Map and Register Definition

19.3.1 Memory Map

Table 19-2 shows the DSPI memory map.

Table 19-2. DSPI Detailed Memory Map

Address	Register Name	Register Description	Bits
Base: 0xFFF9_4000 (DSPI B) 0xFFF9_8000 (DSPI C) 0xFFF9_C000 (DSPI D)	DSPI*x*_MCR	DSPI module configuration register	32
Base + 0x0004 [1]	—	Reserved	—
Base + 0x0008	DSPI*x*_TCR	DSPI transfer count register	32
Base + 0x000C	DSPI*x*_CTAR0	DSPI clock and transfer attributes register 0	32
Base + 0x0010	DSPI*x*_CTAR1	DSPI clock and transfer attributes register 1	32
Base + 0x0014	DSPI*x*_CTAR2	DSPI clock and transfer attributes register 2	32
Base + 0x0018	DSPI*x*_CTAR3	DSPI clock and transfer attributes register 3	32
Base + 0x001C	DSPI*x*_CTAR4	DSPI clock and transfer attributes register 4	32
Base + 0x0020	DSPI*x*_CTAR5	DSPI clock and transfer attributes register 5	32
Base + 0x0024	DSPI*x*_CTAR6	DSPI clock and transfer attributes register 6	32
Base + 0x0028	DSPI*x*_CTAR7	DSPI clock and transfer attributes register 7	32
Base + 0x002C	DSPI*x*_SR	DSPI status register	32
Base + 0x0030	DSPI*x*_RSER	DSPI DMA/interrupt request select and enable register	32
Base + 0x0034	DSPI*x*_PUSHR	DSPI push TX FIFO register	32
Base + 0x0038	DSPI*x*_POPR	DSPI pop RX FIFO register	32
Base + 0x003C	DSPI*x*_TXFR0	DSPI transmit FIFO register 0	32
Base + 0x0040	DSPI*x*_TXFR1	DSPI transmit FIFO register 1	32
Base + 0x0044	DSPI*x*_TXFR2	DSPI transmit FIFO register 2	32
Base + 0x0048	DSPI*x*_TXFR3	DSPI transmit FIFO register 3	32
Base + 0x004C–0x0078 [1]	—	Reserved	—
Base + 0x007C	DSPI*x*_RXFR0	DSPI receive FIFO register 0	32
Base + 0x0080	DSPI*x*_RXFR1	DSPI receive FIFO register 1	32
Base + 0x0084	DSPI*x*_RXFR2	DSPI receive FIFO register 2	32
Base + 0x0088	DSPI*x*_RXFR3	DSPI receive FIFO register 3	32
Base + 0x008C–0x00B8 [1]	—	Reserved	—
Base + 0x00BC	DSPI*x*_DSICR	DSPI DSI configuration register	32
Base + 0x00C0	DSPI*x*_SDR	DSPI DSI serialization data register	32

Table 19-2. DSPI Detailed Memory Map (continued)

Address	Register Name	Register Description	Bits
Base + 0x00C4	DSPI*x*_ASDR	DSPI DSI alternate serialization data register	32
Base + 0x00C8	DSPI*x*_COMPR	DSPI DSI transmit comparison register	32
Base + 0x00CC	DSPI*x*_DDR	DSPI DSI deserialization data register	32

[1] Do not read or write to 'Reserved' memory. Writing to 'Reserved' memory can cause unpredictable operations.

19.3.2 Register Descriptions

19.3.2.1 DSPI Module Configuration Register (DSPI*x*_MCR)

The DSPI*x*_MCR contains bits that configure and control the DSPI operation. You can changes the values of the HALT and MDIS bits during runtime, but the effect begins on the next frame boundary. The HALT and MDIS bits in the DSPI*x*_MCR are the only bit values software can change while the DSPI is running.

Address: Base + 0x0000 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MSTR	CONT_SCKE	DCONF		FRZ	MTFE	PCSSE	ROOE	0	0	PCSIS5	PCSIS4	PCSIS3	PCSIS2	PCSIS1	PCSIS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	MDIS	DIS_TXF	DIS_RXF	CLR_TXF	CLR_RXF	SMPL_PT		0	0	0	0	0	0	0	HALT
W					w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 19-3. DSPI Module Configuration Register (DSPI*x*_MCR)

The following table describes the fields in the DSPI module configuration register:

Table 19-3. DSPI*x*_MCR Field Descriptions

Field	Description
0 MSTR	Master or slave mode select. Configures the DSPI for master mode or slave mode. 0 DSPI is in slave mode 1 DSPI is in master mode
1 CONT_SCKE	Continuous SCK enable. Enables the serial communication clock (SCK) to run continuously. See Section 19.4.8, "Continuous Serial Communications Clock," for details. 0 Continuous SCK disabled 1 Continuous SCK enabled

Table 19-3. DSPI*x*_MCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2–3
DCONF
[0:1]</td><td>DSPI configuration. Configures operating mode of the DSPI. The following table lists the DCONF values for the various configurations.
<table>
<tr><th>DCONF</th><th>Configuration</th><th>Description</th></tr>
<tr><td>00</td><td>SPI</td><td>Serial peripheral interface</td></tr>
<tr><td>01</td><td>DSI</td><td>Deserial interface</td></tr>
<tr><td>10</td><td>CSI</td><td>Combination Deserial and Serial</td></tr>
<tr><td>11</td><td>Invalid value</td><td>Not available</td></tr>
</table></td></tr>
<tr><td>4
FRZ</td><td>Freeze. Allows the DSPI to stop transfers on the next frame boundary when the device enters debug mode.

0 Do not halt serial transfers
1 Halt serial transfers</td></tr>
<tr><td>5
MTFE</td><td>Modified timing format enable. Provides a modified transfer format.
See Section 19.4.7.4, “Modified Transfer Format Enabled (MTFE = 1) with Classic SPI Transfer Format Set (CPHA = 1) for SPI and DSI.”

0 Modified SPI transfer format disabled
1 Modified SPI transfer format enabled</td></tr>
<tr><td>6
PCSSE</td><td>Peripheral chip select strobe enable. Enables the PCSx[5]_$\overline{\text{PCSS}}$ to operate as a PCS strobe output signal.
See Section 19.4.6.5, “Peripheral Chip Select Strobe Enable (PCSS).”
0 PCSx[5]_$\overline{\text{PCSS}}$ is used as the peripheral chip select 5 signal
1 PCSx[5] _$\overline{\text{PCSS}}$ is used as an active-low PCS strobe signal</td></tr>
<tr><td>7
ROOE</td><td>Receive FIFO overflow overwrite enable. Enables an RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer that generated the overflow is ignored or put in the shift register.

If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored. See Section 19.4.9.6, “Receive FIFO Overflow Interrupt Request (RFOF).”
0 Incoming data is ignored
1 Incoming data is put in the shift register</td></tr>
<tr><td>8–9</td><td>Reserved, but implemented. These bits are writable, but have no effect.</td></tr>
<tr><td>10–15
PCSISn</td><td>Peripheral chip select inactive state. Determines the inactive state of the PCSx[n] signal.
You must configure PCSx[0]_$\overline{\text{SS}}$ as inactive high for slave mode operation.
0 The inactive state of PCSx[n] is low
1 The inactive state of PCSx[n] is high</td></tr>
<tr><td>16</td><td>Reserved</td></tr>
<tr><td>17
MDIS</td><td>Module disable. Allows the clock to stop to non-memory mapped logic in the DSPI, effectively putting the DSPI in a software controlled power-saving state. The reset value of the MDIS bit is parameterized, with a default reset value of 0. See Section 19.4.10, “Power Saving Features.”
0 Enable DSPI clocks
1 Allow external logic to disable DSPI clocks</td></tr>
</table>

Table 19-3. DSPI*x*_MCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>18
DIS_TXF</td><td>Disable transmit FIFO. Enables and disables the TX FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See Section 19.4.3.3, “FIFO Disable Operation for details.”
0 TX FIFO is enabled
1 TX FIFO is disabled</td></tr>
<tr><td>19
DIS_RXF</td><td>Disable receive FIFO. Enables and disables the RX FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See Section 19.4.3.3, “FIFO Disable Operation for details.”
0 RX FIFO is enabled
1 RX FIFO is disabled</td></tr>
<tr><td>20
CLR_TXF</td><td>Clear TX FIFO. Flushes the TX FIFO. Write a 1 to the CLR_TXF bit to clear the TX FIFO counter. The CLR_TXF bit is always read as zero.
0 Do not clear the TX FIFO counter
1 Clear the TX FIFO counter</td></tr>
<tr><td>21
CLR_RXF</td><td>Clear RX FIFO. Flushes the RX FIFO. Write a 1 to the CLR_RXF bit to clear the RX counter. The CLR_RXF bit is always read as zero.
0 Do not clear the RX FIFO counter
1 Clear the RX FIFO counter</td></tr>
<tr><td>22–23
SMPL_PT
[0:1]</td><td>Sample point. Allows the host software to select when the DSPI master samples SIN in modified transfer format. Figure 19-34 shows where the master can sample the SIN pin. The following table lists the delayed sample points.
<table>
<tr><th>SMPL_PT</th><th>Number of system clock cycles between odd-numbered edge of SCK[x] and sampling of SIN[x].</th></tr>
<tr><td>00</td><td>0</td></tr>
<tr><td>01</td><td>1</td></tr>
<tr><td>10</td><td>2</td></tr>
<tr><td>11</td><td>Invalid value</td></tr>
</table>
</td></tr>
<tr><td>24–30</td><td>Reserved</td></tr>
<tr><td>31
HALT</td><td>Halt. Allows software to start and stop DSPI transfers on the next boundary.
See Section 19.4.2, “Start and Stop of DSPI Transfers,” for details on the operation of this bit.

0 Start transfers
1 Stop transfers</td></tr>
</table>

19.3.2.2 DSPI Transfer Count Register (DSPI*x*_TCR)

The DSPI*x*_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. You must not write to the DSPI*x*_TCR while the DSPI is running.

Address: Base + 0x0008 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SPI_TCNT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-4. DSPI Transfer Count Register (DSPI*x*_TCR)

The following table describes the field in the DSPI transfer count register:

Table 19-4. DSPI*x*_TCR Field Descriptions

Field	Description
0–15 SPI_TCNT [0:15]	SPI transfer counter. Counts the number of SPI transfers the DSPI makes. The SPI_TCNT field increments every time the last bit of an SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The transfer counter 'wraps around,' incrementing the counter past 65535 resets the counter to zero.
16–31	Reserved

19.3.2.3 DSPI Clock and Transfer Attributes Registers 0–7 (DSPI*x*_CTAR*n*)

The DSPI modules each contain eight clock and transfer attribute registers (DSPI*x*_CTAR*n*) which are used to define different transfer attribute configurations. Each DSPI*x*_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB or LSB first

DSPI*x*_CTARs support compatibility with the QSPI module in the MPC5xx family of MCUs. See Section 19.5.4, "MPC5xx QSPI Compatibility with the DSPI," for a discussion on DSPI and QSPI compatibility. At the initiation of an SPI or DSI transfer, control logic selects the DSPI*x*_CTAR that contains the transfer attributes. Do not write to the DSPI*x*_CTARs while the DSPI is running.

In master mode, the DSPI*x*_CTAR*n* registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. In slave mode, a subset of the bit fields in the DSPI*x*_CTAR0 and DSPI*x*_CTAR1 registers are used to set the slave transfer attributes. See the individual bit descriptions for details.

When the DSPI is configured as a SPI master, the CTAS field in the command portion of the TX FIFO entry identifies the DSPI*x*_CTAR registers that are used on a per-frame basis. When the DSPI is configured as a SPI bus slave, the DSPI*x*_CTAR0 register is used.

When the DSPI is configured as a DSI master, the DSICTAS field in the DSPI DSI configuration register (DSPI*x*_DSICR) selects which of the DSPI*x*_CTAR register is used. See Section 19.3.2.10, “DSPI DSI Configuration Register (DSPIx_DSICR).” When the DSPI is configured as a DSI bus slave, the DSPI*x*_CTAR1 register is used.

In CSI configuration, the transfer attributes are selected based on whether the current frame is SPI data or DSI data. SPI transfers in CSI configuration follow the protocol described for SPI configuration, and DSI transfers in CSI configuration follow the protocol described for DSI configuration. CSI configuration is only valid in conjunction with master mode. See Section 19.4.5, “Combined Serial Interface (CSI) Configuration” for more details.

Address:
Base + 0x000C (DSPI*x*_CTAR0)
Base + 0x0010 (DSPI*x*_CTAR1)
Base + 0x0014 (DSPI*x*_CTAR2)
Base + 0x0018 (DSPI*x*_CTAR3)
Base + 0x001C (DSPIx_CTAR4)
Base + 0x0020 (DSPI*x*_CTAR5)
Base + 0x0024 (DSPI*x*_CTAR6)
Base + 0x0028 (DSPI*x*_CTAR7)

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DBR	FMSZ				CPOL	CPHA	LSB FE	PCSSCK		PASC		PDT		PBR	
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-5. DSPI Clock and Transfer Attributes Registers 0–7 (DSPI*x*_CTAR*n*)

The following table describes the fields in the DSPI clock and transfer attributes register:

Table 19-5. DSPI*x*_CTAR*n* Field Description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0
DBR</td><td>Double baud rate. The DBR bit doubles the effective baud rate of the serial communications clock (SCK). This field is only used in master mode. It effectively halves the baud rate division ratio supporting faster frequencies and odd division ratios for the serial communications clock (SCK). When the DBR bit is set, the duty cycle of the serial communications clock (SCK) depends on the value in the baud rate prescaler and the clock phase bit as listed in the following table. See the BR field and Section 19.4.6.1, “Baud Rate Generator” for details on how to compute the baud rate. If the overall baud rate is divided by two or three system clocks, do not enable continuous SCK or the modified timing format.
0 Baud rate is computed normally with a 50/50 duty cycle
1 Baud rate is doubled with the duty cycle depending on the baud rate prescaler
<table>
<tr><th>DBR</th><th>CPHA</th><th>PBR</th><th>SCK Duty Cycle</th></tr>
<tr><td>0</td><td>any</td><td>any</td><td>50/50</td></tr>
<tr><td>1</td><td>0</td><td>00</td><td>50/50</td></tr>
<tr><td>1</td><td>0</td><td>01</td><td>33/66</td></tr>
<tr><td>1</td><td>0</td><td>10</td><td>40/60</td></tr>
<tr><td>1</td><td>0</td><td>11</td><td>43/57</td></tr>
<tr><td>1</td><td>1</td><td>00</td><td>50/50</td></tr>
<tr><td>1</td><td>1</td><td>01</td><td>66/33</td></tr>
<tr><td>1</td><td>1</td><td>10</td><td>60/40</td></tr>
<tr><td>1</td><td>1</td><td>11</td><td>57/43</td></tr>
</table>
</td></tr>
<tr><td>1–4
FMSZ
[0:3]</td><td>FMSZ. Selects the number of bits transferred per frame. The FMSZ field is used in master mode and slave mode. The following table lists the frame sizes.
<table>
<tr><th>FMSZ</th><th>Frame Size</th><th>FMSZ</th><th>Frame Size</th></tr>
<tr><td>0000</td><td>Invalid value</td><td>1000</td><td>9</td></tr>
<tr><td>0001</td><td>Invalid value</td><td>1001</td><td>10</td></tr>
<tr><td>0010</td><td>Invalid value</td><td>1010</td><td>11</td></tr>
<tr><td>0011</td><td>4</td><td>1011</td><td>12</td></tr>
<tr><td>0100</td><td>5</td><td>1100</td><td>13</td></tr>
<tr><td>0101</td><td>6</td><td>1101</td><td>14</td></tr>
<tr><td>0110</td><td>7</td><td>1110</td><td>15</td></tr>
<tr><td>0111</td><td>8</td><td>1111</td><td>16</td></tr>
</table>
</td></tr>
</table>

Table 19-5. DSPI*x*_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>5
CPOL</td><td>Clock polarity. Selects the inactive state of the serial communications clock (SCKx). This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock polarities. When the continuous selection format is selected (CONT = 1 or DCONT = 1), switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. For more information on continuous selection format, see Section 19.4.7.5, “Continuous Selection Format.”
0 The inactive state value of SCKx is low
1 The inactive state value of SCKx is high</td></tr>
<tr><td>6
CPHA</td><td>Clock phase. Selects which edge of SCKx causes data to change and which edge causes data to be captured. This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock phase settings.
0 Data is captured on the leading edge of SCKx and changed on the following edge
1 Data is changed on the leading edge of SCKx and captured on the following edge</td></tr>
<tr><td>7
LSBFE</td><td>LSB first enable. Selects if the LSB or MSB of the frame is transferred first. This bit is only used in master mode.
0 Data is transferred MSB first
1 Data is transferred LSB first</td></tr>
<tr><td>8–9
PCSSCK
[0:1]</td><td>PCSx to SCKx delay prescaler. Selects the prescaler value for the delay between assertion of PCSx and the first edge of the SCKx. Use in master mode only. The following table lists the prescaler values. The description for bitfield CSSCK in Table 19-5 details how to compute the PCS to SCK delay.
<table>
<tr><th>PCSSCK Value</th><th>PCSx to SCKx Delay Prescaler Value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>10–11
PASC
[0:1]</td><td>After SCKx delay prescaler. Selects the prescaler value for the delay between the last edge of SCKx and the negation of PCSx. Use in master mode only. The following table lists the prescaler values. The description for bitfield ASC in Table 19-5 details how to compute the after SCKx delay.
<table>
<tr><th>PASC Value</th><th>After SCKx Delay Prescaler Value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
</table>

Table 19-5. DSPI*x*_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>12–13
PDT
[0:1]</td><td>Delay after transfer prescaler. The PDT field selects the prescaler value for the delay between the negation of the PCSx signal at the end of a frame and the assertion of PCSx at the beginning of the next frame. The PDT field is only used in master mode. The following table lists the prescaler values. The description for bitfield DT in Table 19-5 details how to compute the delay after transfer.
<table>
<tr><th>PDT Value</th><th>Delay after Transfer Prescaler Value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>14–15
PBR
[0:1]</td><td>Baud rate prescaler. Selects the prescaler value for the baud rate. Use in master mode only. The baud rate is the frequency of the serial communications clock (SCKx). The system clock is divided by the prescaler value before the baud rate selection takes place. The baud rate prescaler values are listed in the following table. The description for PBR in Section 19.4.6.1, “Baud Rate Generator” details how to compute the baud rate.
<table>
<tr><th>PBR Value</th><th>Baud Rate Prescaler Value</th></tr>
<tr><td>00</td><td>2</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
</table>

Table 19-5. DSPI*x*_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>16–19
CSSCK
[0:3]</td><td>PCSx to SCKx delay scaler. Selects the scaler value for the PCSx to SCKx delay. Use in master mode only. The PCSx to SCKx delay is the delay between the assertion of PCSx and the first edge of the SCKx. The following table lists the scaler values.
<table>
<tr><th>CSSCK Value</th><th>PCS to SCK Delay Scaler Value</th><th>CSSCK Value</th><th>PCS to SCK Delay Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>512</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>1024</td></tr>
<tr><td>0010</td><td>8</td><td>1010</td><td>2048</td></tr>
<tr><td>0011</td><td>16</td><td>1011</td><td>4096</td></tr>
<tr><td>0100</td><td>32</td><td>1100</td><td>8192</td></tr>
<tr><td>0101</td><td>64</td><td>1101</td><td>16384</td></tr>
<tr><td>0110</td><td>128</td><td>1110</td><td>32768</td></tr>
<tr><td>0111</td><td>256</td><td>1111</td><td>65536</td></tr>
</table>
The PCSx to SCKx delay is a multiple of the system clock period. It is computed using the following equation:
$t_{CSC} = \frac{1}{f_{SYS}} \times \text{PCSSCK prescaler value} \times \text{CSSCK scaler value}$
Note: See Section 19.4.6.2, “PCS to SCK Delay (tCSC),” for more details.</td></tr>
<tr><td>20-23
ASC
[0:3]</td><td>After SCKx delay scaler. Selects the scaler value for the After SCKx delay. Use in master mode only. The after SCKx delay is the delay between the last edge of SCKx and the negation of PCSx. The following table lists the scaler values.
<table>
<tr><th>ASC Value</th><th>After SCK Delay Scaler Value</th><th>ASC Value</th><th>After SCK Delay Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>512</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>1024</td></tr>
<tr><td>0010</td><td>8</td><td>1010</td><td>2048</td></tr>
<tr><td>0011</td><td>16</td><td>1011</td><td>4096</td></tr>
<tr><td>0100</td><td>32</td><td>1100</td><td>8192</td></tr>
<tr><td>0101</td><td>64</td><td>1101</td><td>16384</td></tr>
<tr><td>0110</td><td>128</td><td>1110</td><td>32768</td></tr>
<tr><td>0111</td><td>256</td><td>1111</td><td>65536</td></tr>
</table>
The after SCKx delay is a multiple of the system clock period, and it is computed using the following equation:
$t_{ASC} = \frac{1}{f_{SYS}} \times \text{PASC Prescaler value} \times \text{ASC Scaler value}$
Note: See Section 19.4.6.3, “After SCK Delay (tASC),” for more details.</td></tr>
</table>

Table 19-5. DSPI*x*_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>24–27
DT
[0:3]</td><td>Delay after transfer scaler. The DT field selects the delay after transfer scaler. Use in master mode only. The delay after transfer is the time between the negation of the PCSx signal at the end of a frame and the assertion of PCSx at the beginning of the next frame. The following table lists the scaler values.
<table>
<tr><th>DT Value</th><th>Delay after Transfer Scaler Value</th><th>DT Value</th><th>Delay after Transfer Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>512</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>1024</td></tr>
<tr><td>0010</td><td>8</td><td>1010</td><td>2048</td></tr>
<tr><td>0011</td><td>16</td><td>1011</td><td>4096</td></tr>
<tr><td>0100</td><td>32</td><td>1100</td><td>8192</td></tr>
<tr><td>0101</td><td>64</td><td>1101</td><td>16384</td></tr>
<tr><td>0110</td><td>128</td><td>1110</td><td>32768</td></tr>
<tr><td>0111</td><td>256</td><td>1111</td><td>65536</td></tr>
</table>
The delay after transfer is a multiple of the system clock period. It is computed using the following equation:
$$t_{DT} = \frac{1}{f_{SYS}} \times \text{PDT Prescaler value} \times \text{DT Scaler value}$$
Note: See Section 19.4.6.4, “Delay after Transfer (tDT),” for more details</td></tr>
</table>

Table 19-5. DSPI*x*_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>28–31
BR
[0:3]</td><td>Baud rate scaler. Selects the scaler value for the baud rate. Use in master mode only. The pre-scaled system clock is divided by the baud rate scaler to generate the frequency of the SCK. The following table lists the baud rate scaler values.
<table>
<tr><th>BR Value</th><th>Baud Rate Scaler Value</th><th>BR Value</th><th>Baud Rate Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>256</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>512</td></tr>
<tr><td>0010</td><td>6</td><td>1010</td><td>1024</td></tr>
<tr><td>0011</td><td>8</td><td>1011</td><td>2048</td></tr>
<tr><td>0100</td><td>16</td><td>1100</td><td>4096</td></tr>
<tr><td>0101</td><td>32</td><td>1101</td><td>8192</td></tr>
<tr><td>0110</td><td>64</td><td>1110</td><td>16384</td></tr>
<tr><td>0111</td><td>128</td><td>1111</td><td>32768</td></tr>
</table>
The baud rate is computed using the following equation:
$$\text{SCK baud rate} = \frac{f_{SYS}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$$
Note: See Section 19.4.6.1, “Baud Rate Generator,” for more details.</td></tr>
</table>

19.3.2.4 DSPI Status Register (DSPI*x*_SR)

The DSPI*x*_SR contains status and flag bits. The bits are set by the hardware and reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI*x*_SR by writing a 1 to clear it (w1c). Writing a 0 to a flag bit has no effect.

Address: Base + 0x002C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	0	0	RFOF	0	RFDF	0
W	w1c			w1c	w1c		w1c						w1c		w1c	
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-6. DSPI Status Register (DSPI*x*_SR)

The following table describes the fields in the DSPI status register:

Table 19-6. DSPI*x*_SR Field Descriptions

Field	Description
0 TCF	Transfer complete flag. Indicates that all bits in a frame have been shifted out. The TCF bit is set after the last incoming databit is sampled, but before the tASC delay starts. See Section 19.4.7.1, “Classic SPI Transfer Format (CPHA = 0)” and Section 19.4.7.2, “Classic SPI Transfer Format (CPHA = 1) for details. The TCF bit is cleared by writing 1 to it. 0 Transfer not complete 1 Transfer complete
1 TXRXS	TX and RX status. Reflects the status of the DSPI. See Section 19.4.2, “Start and Stop of DSPI Transfers” for information on what clears and sets this bit. 0 TX and RX operations are disabled (DSPI is in STOPPED state) 1 TX and RX operations are enabled (DSPI is in RUNNING state)
2	Reserved
3 EOQF	End of queue flag. Indicates that transmission in progress is the last entry in a queue. The EOQF bit is set when the TX FIFO entry has the EOQ bit set in the command halfword and after the last incoming databit is sampled, but before the tASC delay starts. See Section 19.4.7.1, “Classic SPI Transfer Format (CPHA = 0)” and Section 19.4.7.2, “Classic SPI Transfer Format (CPHA = 1)” for details. The EOQF bit is cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executing command 1 EOQ bit is set in the executing SPI command **Note:** EOQF does not function in slave mode.
4 TFUF	Transmit FIFO underflow flag. Indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit is cleared by writing 1 to it. 0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred
5	Reserved
6 TFFF	Transmit FIFO fill flag. Indicates that the TX FIFO can be filled. Provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by writing 1 to it, or an by acknowledgement from the eDMA controller when the TX FIFO is full. 0 TX FIFO is full 1 TX FIFO is not full
7–11	Reserved
12 RFOF	Receive FIFO overflow flag. Indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit is cleared by writing 1 to it. 0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred
13	Reserved
14 RFDF	Receive FIFO drain flag. Indicates that the RX FIFO can be drained. Provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing 1 to it, or by acknowledgement from the eDMA controller when the RX FIFO is empty. 0 RX FIFO is empty 1 RX FIFO is not empty **Note:** In the interrupt service routine, RFDF must be cleared only after the DSPIx_POPR register is read.

Table 19-6. DSPI*x*_SR Field Descriptions (continued)

Field	Description
15	Reserved
16–19 TXCTR [0:3]	TX FIFO counter. Indicates the number of valid entries in the TX FIFO. The TXCTR increments every time the DSPI _PUSHR is written. The TXCTR is decremented every time an SPI command is executed and the SPI data is transferred to the shift register.
20–23 TXNXTPTR [0:3]	Transmit next pointer. Indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See Section 19.4.3.4, “Using the TX FIFO Buffering Mechanism” for more details.
24–27 RXCTR [0:3]	RX FIFO counter. Indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI _POPR is read. The RXCTR increments after the last incoming databit is sampled, but before the tASC delay starts. See Section 19.4.7.1, “Classic SPI Transfer Format (CPHA = 0)” and Section 19.4.7.2, “Classic SPI Transfer Format (CPHA = 1)” for details.
28–31 POPNXTPTR [0:3]	Pop next pointer. Contains a pointer to the RX FIFO entry that is returned when the DSPI*x*_POPR is read. The POPNXTPTR is updated when the DSPI*x*_POPR is read. See Section 19.4.3.5, “Using the RX FIFO Buffering Mechanism” for more details.

19.3.2.5 DSPI DMA and Interrupt Request Select and Enable Register (DSPI*x*_RSER)

The DSPI*x*_RSER serves two purposes: enables flag bits in the DSPI*x*_SR to generate DMA requests or interrupt requests, and selects the type of request to generate. See the bit descriptions for the type of requests that are supported. Do not write to the DSPI*x*_RSER while the DSPI is running. See section LINK for more information on the global Interrupt Vectors (table) and DMA channel assignments (list).

Address: Base + 0x0030 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF_RE	0	0	EOQF_RE	TFUF_RE	0	TFFF_RE	TFFF_DIRS	0	0	0	0	RFOF_RE	0	RFDF_RE	RFDF_DIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-7. DSPI DMA / Interrupt Request Select and Enable Register (DSPI*x*_RSER)

The following table describes the fields in the DSPI DMA / interrupt request and enable register:

Table 19-7. DSPI*x*_RSER Field Descriptions

Field	Description
0 TCF_RE	Transmission complete request enable. Enables TCF flag in the DSPI*x*_SR to generate an interrupt request. 0 TCF interrupt requests are disabled 1 TCF interrupt requests are enabled
1–2	Reserved
3 EOQF_RE	DSPI finished request enable. Enables the EOQF flag in the DSPI*x*_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled 1 EOQF interrupt requests are enabled
4 TFUF_RE	Transmit FIFO underflow request enable. The TFUF_RE bit enables the TFUF flag in the DSPI*x*_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled 1 TFUF interrupt requests are enabled
5	Reserved
6 TFFF_RE	Transmit FIFO fill request enable. Enables the TFFF flag in the DSPI*x*_SR to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled 1 TFFF interrupt requests or DMA requests are enabled
7 TFFF_DIRS	Transmit FIFO fill DMA or interrupt request select. Selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI*x*_SR is set, and the TFFF_RE bit in the DSPI*x*_RSER is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is selected 1 DMA request is selected
8–11	Reserved
12 RFOF_RE	Receive FIFO overflow request enable. Enables the RFOF flag in the DSPI*x*_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled 1 RFOF interrupt requests are enabled
13	Reserved
14 RFDF_RE	Receive FIFO drain request enable. Enables the RFDF flag in the DSPI*x*_SR to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled 1 RFDF interrupt requests or DMA requests are enabled
15 RFDF_DIRS	Receive FIFO drain DMA or interrupt request select. Selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI*x*_SR is set, and the RFDF_RE bit in the DSPI*x*_RSER is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is selected 1 DMA request is selected
16–31	Reserved

19.3.2.6 DSPI PUSH TX FIFO Register (DSPI*x*_PUSHR)

The DSPI*x*_PUSHR provides a means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. See Section 19.4.3.4, "Using the TX FIFO Buffering Mechanism," for more information. Write accesses of 8- or 16-bits to the DSPI*x*_PUSHR transfers 32 bits to the TX FIFO.

NOTE

TXDATA is used in master and slave modes.

Address: Base + 0x0034 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CONT	CTAS			EOQ	CT CNT	0	0	0	0	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-8. DSPI PUSH TX FIFO Register (DSPI*x*_PUSHR)

The following table describes the fields in the DSPI push transmit FIFO register:

Table 19-8. DSPI*x*_PUSHR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0
CONT</td><td>Continuous peripheral chip select enable. Selects a continuous selection format. The bit is used in SPI master mode. The bit enables the selected PCS signals to remain asserted between transfers. See Section 19.4.7.5, “Continuous Selection Format,” for more information.
0 Return peripheral chip select signals to their inactive state between transfers
1 Keep peripheral chip select signals asserted between transfers</td></tr>
<tr><td>1–3
CTAS
[0:2]</td><td>Clock and transfer attributes select. Selects which of the DSPIx_CTARs is used to set the transfer attributes for the SPI frame. In SPI slave mode, DSPIx_CTAR0 is used. The following table shows how the CTAS values map to the DSPIx_CTARs. There are eight DSPIx_CTARs in the device DSPI implementation.
Note: Use in SPI master mode only.
<table>
<tr><th>CTAS</th><th>Use Clock and Transfer Attributes from</th></tr>
<tr><td>000</td><td>DSPIx_CTAR0</td></tr>
<tr><td>001</td><td>DSPIx_CTAR1</td></tr>
<tr><td>010</td><td>DSPIx_CTAR2</td></tr>
<tr><td>011</td><td>DSPIx_CTAR3</td></tr>
<tr><td>100</td><td>DSPIx_CTAR4</td></tr>
<tr><td>101</td><td>DSPIx_CTAR5</td></tr>
<tr><td>110</td><td>DSPIx_CTAR6</td></tr>
<tr><td>111</td><td>DSPIx_CTAR7</td></tr>
</table></td></tr>
<tr><td>4
EOQ</td><td>End of queue. Provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPIx_SR is set.
Note: Use in SPI master mode only.
0 The SPI data is not the last data to transfer
1 The SPI data is the last data to transfer</td></tr>
<tr><td>5
CTCNT</td><td>Clear SPI_TCNT. Provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the SPI_TCNT field in the DSPIx_TCR. The SPI_TCNT field is cleared before transmission of the current SPI frame begins.
Note: Use in SPI master mode only.
0 Do not clear SPI_TCNT field in the DSPIx_TCR
1 Clear SPI_TCNT field in the DSPIx_TCR</td></tr>
<tr><td>6–7</td><td>Reserved</td></tr>
<tr><td>8–9</td><td>Reserved, but implemented. These bits are writable, but have no effect.</td></tr>
<tr><td>10–15
PCSx</td><td>Peripheral chip select x. Selects which PCSx signals are asserted for the transfer.
Note: Use in SPI master mode only.
0 Negate the PCSx signal
1 Assert the PCSx signal</td></tr>
<tr><td>16–31
TXDATA
[0:15]</td><td>Transmit data. Holds SPI data for transfer according to the associated SPI command.
Note: Use TXDATA in master and slave modes.</td></tr>
</table>

19.3.2.7 DSPI POP RX FIFO Register (DSPI*x*_POPR)

The DSPI*x*_POPR allows you to read the RX FIFO. See Section 19.4.3.5, “Using the RX FIFO Buffering Mechanism” for a description of the RX FIFO operations. Eight-bit or 16-bit read accesses to the DSPI*x*_POPR fetches the RX FIFO data, and updates the counter and pointer.

NOTE

Reading the DSPI*x*_POPR field fetches the data from the current RX FIFO entry. Once the data is read, the read data pointer moves to the next RX FIFO entry, which prevents access to the data in the current entry. Therefore, do not read DSPI*x*_POPR unless you need the data. For compatibility, configure the TLB (MMU table) entry for DSPI*x*_POPR as guarded.

Address: Base + 0x0038 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-9. DSPI POP RX FIFO Register (DSPI*x*_POPR)

The following table describes the fields in the DSPI pop receive FIFO register:

Table 19-9. DSPI*x*_POPR Field Descriptions

Field	Description
0–15	Reserved, must be cleared.
16–31 RXDATA [0:15]	Received data. The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (POPNXTPTR).

19.3.2.8 DSPI Transmit FIFO Registers 0–3 (DSPI*x*_TXFR*n*)

The DSPI*x*_TXFR*n* registers provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI*x*_TXFR*n* registers does not alter the state of the TX FIFO. The MCU uses four registers to implement the TX FIFO, that is DSPI*x*_TXFR0–DSPI*x*_TXFR3 are used.

Address:
Base + 0x003C (DSPI*x*_TXFR0)
Base + 0x0040 (DSPI*x*_TXFR1)
Base + 0x0044 (DSPI*x*_TXFR2)
Base + 0x0048 (DSPI*x*_TXFR3)

Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TXCMD															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-10. DSPI Transmit FIFO Register 0–3 (DSPI*x*_TXFR*n*)

The following table describes the fields in the DSPI transmit FIFO register:

Table 19-10. DSPI*x*_TXFR*n* Field Descriptions

Field	Description
0–15 TXCMD [0:15]	Transmit command. Contains the command that sets the transfer attributes for the SPI data. See Section 19.3.2.6, “DSPI PUSH TX FIFO Register (DSPIx_PUSHR),” for details on the command field.
16–31 TXDATA [0:15]	Transmit data. Contains the SPI data to be shifted out.

19.3.2.9 DSPI Receive FIFO Registers 0–3 (DSPI*x*_RXFR*n*)

The DSPI*x*_RXFR*n* registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI*x*_RXFR registers are read-only. Reading the DSPI*x*_RXFR*n* registers does not alter the state of the RX FIFO. The device uses four registers to implement the RX FIFO, that is DSPI*x*_RXFR0–DSPI*x*_RXFR3 are used.

Address:
Base + 0x007C (DSPI*x*_RXFR0)
Base + 0x0080 (DSPI*x*_RXFR1)
Base + 0x0084 (DSPI*x*_RXFR2)
Base + 0x0088 (DSPI*x*_RXFR3)

Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-11. DSPI Receive FIFO Registers 0–3 (DSPI*x*_RXFR*n*)

The following table describes the field in the DSPI receive FIFO register:

Table 19-11. DSPI*x*_RXFR*n* Field Description

Field	Description
0–15	Reserved, must be cleared.
16–31 RXDATA [15:0]	Receive data. Contains the received SPI data.

19.3.2.10 DSPI DSI Configuration Register (DSPI*x*_DSICR)

The DSPI*x*_DSICR selects attributes for DSI and CSI configurations. Do not write to the DSPI*x*_DSICR while the DSPI is running.

Address: Base + 0x00BC Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTOE	0	MTOCNT						0	0	0	0	TXSS	TPOL	TRRE	CID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DCONT	DSICTAS			0	0	0	0	0	0	DPCS5	DPCS4	DPCS3	DPCS2	DPCS1	DPCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-12. DSPI DSI Configuration Register (DSPI*x*_DSICR)

The following table describes the fields in the DSPI deserial serial interface configuration register:

Table 19-12. DSPI*x*_DSICR Field Descriptions

Field	Description
0 MTOE	Multiple transfer operation enable. Enables multiple DSPIs connected in a parallel or serial configuration. See Section 19.4.4.7, “Multiple Transfer Operation (MTO),” for more information. 0 Multiple transfer operation disabled 1 Multiple transfer operation enabled
1	Reserved
2–7 MTOCNT [0:5]	Multiple transfer operation count. Selects number of bits to be shifted out during a transfer in multiple transfer operation. The field sets the number of SCK cycles that the bus master needs to generate to complete the transfer. The number of SCK cycles used are one more than the value in the MTOCNT field. The number of SCK cycles defined by MTOCNT must be equal to or greater than the frame size.
8–11	Reserved
12 TXSS	Transmit data source select. Selects the source of data to be serialized. The source can be data from host software written to the DSPI DSI alternate serialization data register (DSPI*x*_ASDR), or parallel output pin states latched into the DSPI DSI serialization data register (DSPI*x*_SDR). 0 Source of serialized data is the DSPI*x*_SDR 1 Source of serialized data is the DSPI*x*_ASDR
13 TPOL	Trigger polarity. Selects the active edge of the internal hardware trigger input signal (*hi*). The bit selects which edge initiates a transfer in the DSI configuration. See Section 19.4.4.5, “DSI Transfer Initiation Control,” for more information. 0 Falling-edge initiates a transfer 1 Rising-edge initiates a transfer
14 TRRE	Trigger reception enable. Enables the DSPI to initiate a transfer when an external trigger signal is received. The bit is only valid in DSI configuration. See Section 19.4.4.5, “DSI Transfer Initiation Control,” for more information. 0 Trigger signal reception disabled 1 Trigger signal reception enabled

Table 19-12. DSPI*x*_DSICR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15
CID</td><td>Change in data transfer enable. Enables a change in serialization data to initiate a transfer. The bit is used in master mode in DSI and CSI configurations to control when to initiate transfers. When the CID bit is set, serialization is initiated when the current DSI data differs from the previous DSI data shifted out. The DSPIx_COMPR is compared with the DSPIx_SDR or DSPIx_ASDR to detect a change in data. See Section 19.4.4.5, “DSI Transfer Initiation Control,” for more information.
0 Change in data transfer operation disabled
1 Change in data transfer operation enabled</td></tr>
<tr><td>16
DCONT</td><td>DSI continuous peripheral chip select enable. Enables the PCSx signals to remain asserted between transfers. The DCONT bit only affects the PCS signals in DSI master mode. See Section 19.4.7.5, “Continuous Selection Format,” for details.
0 Return peripheral chip select signals to their inactive state after transfer is complete
1 Keep peripheral chip select signals asserted after transfer is complete</td></tr>
<tr><td>17–19
DSICTAS
[0:2]</td><td>DSI clock and transfer attributes select. The DSICTAS field selects which of the DSPIx_CTARs is used to provide transfer attributes in DSI configuration. The DSICTAS field is used in DSI master mode. In DSI slave mode, the DSPIx_CTAR1 is always selected. The following table lists the DSICTAS to DSPIx_CTARs mapping.
<table>
<tr><th>DSICTAS</th><th>DSI Clock and Transfer Attributes Controlled by</th></tr>
<tr><td>000</td><td>DSPIx_CTAR0</td></tr>
<tr><td>001</td><td>DSPIx_CTAR1</td></tr>
<tr><td>010</td><td>DSPIx_CTAR2</td></tr>
<tr><td>011</td><td>DSPIx_CTAR3</td></tr>
<tr><td>100</td><td>DSPIx_CTAR4</td></tr>
<tr><td>101</td><td>DSPIx_CTAR5</td></tr>
<tr><td>110</td><td>DSPIx_CTAR6</td></tr>
<tr><td>111</td><td>DSPIx_CTAR7</td></tr>
</table>
</td></tr>
<tr><td>20–23</td><td>Reserved</td></tr>
<tr><td>24–25</td><td>Reserved, but implemented. These bits are writable, but have no effect.</td></tr>
<tr><td>26–31
DPCSx</td><td>DSI peripheral chip select n. The DPCS bits select which of the PCSx signals to assert during a DSI transfer. The DPCS bits assert and negate the PCSx signals in DSI master mode only.
0 Negate PCSx
1 Assert PCSx</td></tr>
</table>

19.3.2.11 DSPI DSI Serialization Data Register (DSPI*x*_SDR)

The DSPI*x*_SDR contains the signal states of the parallel input signals from the eTPU or the eMIOS. The pin states of the parallel input signals are latched into the DSPI*x*_SDR on the rising edge of every system clock. The DSPI*x*_SDR is read-only. When the TXSS bit in the DSPI*x*_DSICR is negated, the data in the DSPI*x*_SDR is the source of the serialized data.

Address: Base + 0x00C0 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SER_DATA [15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-13. DSPI DSI Serialization Data Register (DSPI*x*_SDR)

The following table describes the field in the DSPI deserial serial interface serialization data register:

Table 19-13. DSPI*x*_SDR Field Description

Bits	Description
0–15	Reserved
16–31 SER_DATA [15:0]	Serialized data. The SER_DATA field contains the signal states of the parallel input signals. SER_DATA [15:0] maps to DSPI serialization inputs IN[15:0]. See Section 19.4.4.6, “DSPI Connections to eTPUA, eMIOS and SIU.”

19.3.2.12 DSPI DSI Alternate Serialization Data Register (DSPI*x*_ASDR)

The DSPI*x*_ASDR allows the host software to write data to be serialized. When the TXSS bit in the DSPI*x*_DSICR is set, the data in the DSPI*x*_ASDR is the source of the serialized data. Writes to the DSPI*x*_ASDR take effect on the next frame boundary.

Address: Base + 0x00C4 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ASER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-14. DSPI DSI Alternate Serialization Data Register (DSPI*x*_ASDR)

The following table describes the field in the DSPI deserial serial interface alternate serialization data register:

Table 19-14. DSPI*x*_ASDR Field Description

Field	Description
0–15	Reserved
16–31 ASER_DATA [0:15]	Alternate serialized data. The ASER_DATA field holds the alternate data to be serialized.

19.3.2.13 DSPI DSI Transmit Comparison Register (DSPI*x*_COMPR)

The DSPI*x*_COMPR holds a copy of the last transmitted DSI data. The DSPI*x*_COMPR is read-only. DSI data is transferred to this register as it is loaded into the transmit shift register.

Address: Base + 0x00C8 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	COMP_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-15. DSPI DSI Transmit Comparison Register (DSPI*x*_COMPR)

The following table describes the field in the DSPI deserial serial interface transmit comparison register:

Table 19-15. DSPI*x*_COMPR Field Description

Field	Description
0–15	Reserved
16–31 COMP_DATA [0:15]	Compare data. The COMP_DATA field holds the last serialized DSI data.

19.3.2.14 DSPI DSI Deserialization Data Register (DSPI*x*_DDR)

The DSPI*x*_DDR holds the signal states for the parallel output signals. The DSPI*x*_DDR is read-only and it is memory mapped so that host software can read the incoming DSI frames.

Address: Base + 0x00CC Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DESER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-16. DSPI Deserialization Data Register (DSPI*x*_DDR)

The following table describes the field in the DSPI deserialization data register:

Table 19-16. DSPI*x*_DDR Field Description

Field	Description
0–15	Reserved
16–31 DESER_DATA	Deserialized data. Holds deserialized data which is presented as signal states to the parallel output signals.

19.4 Functional Description

The DSPI supports full-duplex, synchronous serial communications between the MCU and peripheral devices. The DSPI can also be used to reduce the number of pins required for I/O by serializing and deserializing up to 16 parallel input/output signals from the eTPU and eMIOS. All communications use a protocol very similar to SPI.

You can configure the DSPI to a serial peripheral interface (SPI) in which the DSPI operates as a basic SPI or a queued SPI.

The DSPI has three configurations:

- Serial peripheral interface (SPI) configuration in which the DSPI operates as a basic SPI or a queued SPI.
- Deserial serial interface (DSI) configuration where the DSPI serializes eTPU and eMIOS output channels and deserializes the received data by placing it on the eTPU and eMIOS input channels and as inputs to the External Interrupt Request subblock of the SIU.
- Combined serial interface (CSI) configuration where the DSPI operates in both SPI and DSI configurations interleaving DSI frames with SPI frames, giving priority to SPI frames.

The DCONF field in the DSPI*x*_MCR register determines the DSPI configuration. See Table 19-3 for the DSPI configuration values.

The DSPI*x*_CTAR0–DSPI*x*_CTAR7 registers hold clock and transfer attributes.The manner in which a CTAR is selected depends on the DSPI configuration (SPI, DSI, or CSI). The SPI configuration can select which CTAR to use on a frame-by-frame basis by setting the CTAS field in the DSPI*x*_PUSHR.The DSI configuration statically selects which CTAR to use. In CSI configuration, priority logic determines if SPI data or DSI data is transferred. The type of data transferred (whether DSI or SPI) dictates which CTAR the CSI configuration uses. See Section 19.3.2.3, "DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx_CTARn)," for information on DSPI*x*_CTAR fields.

The 16-bit shift register in the master and the 16-bit shift register in the slave are linked by the SOUT*x* and SIN*x* signals to form a distributed 32-bit register. When a data transfer operation is performed, data is serially shifted a pre-determined number of bit positions. Because the registers are linked, data is exchanged between the master and the slave; the data that was in the master's shift register is now in the shift register of the slave, and vice versa. At the end of a transfer, the TCF bit in the DSPI*x*_SR is set to indicate a completed transfer. Figure 19-17 illustrates how master and slave data is exchanged.

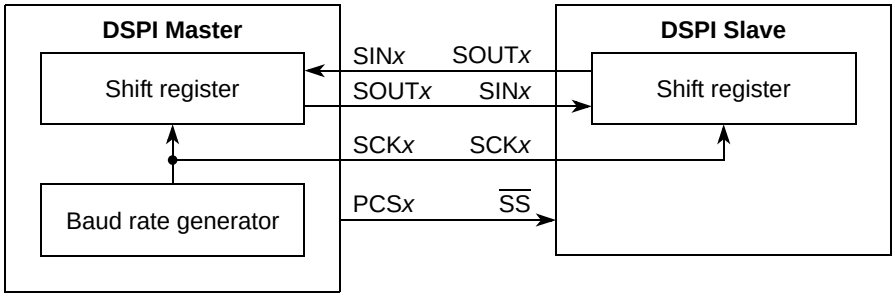


Figure 19-17. SPI and DSI Serial Protocol Overview

The DSPI has six peripheral chip select (PCS*x*) signals that are used to communicate with slave devices.

Transfer protocols and timing properties are shared by the three DSPI configurations; these properties are described independently of the configuration in Section 19.4.7, "Transfer Formats." The transfer rate and delay settings are described in section Section 19.4.6, "DSPI Baud Rate and Clock Delay Generation."

See Section 19.4.10, "Power Saving Features" for information on the power-saving features of the DSPI.

19.4.1 Modes of Operation

The DSPI modules have four available distinct modes:

- Master mode
- Slave mode
- Module disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes while debug mode is a device-specific mode.

The module-specific modes are determined by bits in the DSPI*x*_MCR. Debug mode is a mode that the entire device can enter in parallel with the DSPI being configured in one of its module-specific modes.

19.4.1.1 Master Mode

Master mode allows the DSPI to initiate and control communications. The DSPI operates as bus master when the MSTR bit in the DSPI*x*_MCR is set. The serial communications clock (SCK) is controlled by the master DSPI. All three DSPI configurations are valid in master mode.

In SPI configuration, master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPI*x*_CTARs are used to set the transfer attributes. Transfer attribute control is on a frame-by-frame basis.

See Section 19.4.3, “Serial Peripheral Interface (SPI) Configuration” for more details.

In DSI configuration, master mode transfer attributes are controlled by the DSPI*x*_DSCIR. A detailed description of the DSPI*x*_DSCIR is located in Section 19.3.2.10, “DSPI DSI Configuration Register (DSPIx_DSICR).” The DSISCTAS field in the DSPI*x*_DSICR selects which of the DSPI*x*_CTARs are used to set the transfer attributes. Transfer attributes are set up during initialization and must not be changed between frames.

See Section 19.4.4, “Deserial Serial Interface (DSI) Configuration,” for more details.

The CSI configuration is only available in master mode. In CSI configuration, the DSI data is transferred using DSI configuration transfer attributes and SPI data is transferred using the SPI configuration transfer attributes. For the bus slave to distinguish between DSI and SPI frames, the transfer attributes for the two types of frames must utilize different peripheral chip select signals.

See Section 19.4.5, “Combined Serial Interface (CSI) Configuration,” for details.

19.4.1.2 Slave Mode

In slave mode the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the MSTR bit in the DSPI*x*_MCR is cleared to zero. The DSPI slave is selected by a bus master when the slave’s $\overline{SS}$ signal asserts. In slave mode, the SCK is provided by the bus master. All transfer attributes are controlled by the bus master, except the clock polarity, clock phase and the number of bits to transfer which must be configured in the DSPI slave to communicate correctly.

The SPI and DSI configurations are valid in slave mode. CSI configuration is not available in slave mode. In SPI slave mode, the slave transfer attributes are set in the DSPI*x*_CTAR0. In DSI slave mode the slave transfer attributes are set in the DSPI*x*_CTAR1. In slave mode, for both SPI and DSI configurations, data is transferred MSB first. The LSBFE field of the related CTAR is not used.

19.4.1.3 Module Disable Mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI*x*_MCR is set.

See Section 19.4.10, “Power Saving Features,” for more details on the module disable mode.

19.4.1.4 Debug Mode

The debug mode is used for system development and debugging. If the MCU enters debug mode while the FRZ bit in the DSPI*x*_MCR is set, the DSPI stops all serial transfers and enters a stopped state. If the MCU enters debug mode while the FRZ bit is cleared, the DSPI is unaffected and remains in the module-specific mode and configuration of the DSPI. The DSPI enters debug mode when a debug request is asserted by an external controller.

See Figure 19-18 for a state diagram.

19.4.2 Start and Stop of DSPI Transfers

The DSPI has two operating states: STOPPED and RUNNING. The states are independent of DSPI configuration. The default state of the DSPI is STOPPED. In the STOPPED state no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The STOPPED state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. The TXRXS bit in the DSPI*x*_SR is cleared in this state. In the RUNNING state, serial transfers take place. The TXRXS bit in the DSPI*x*_SR is set in the RUNNING state.

Figure 19-18 shows a state diagram of the start and stop mechanism.

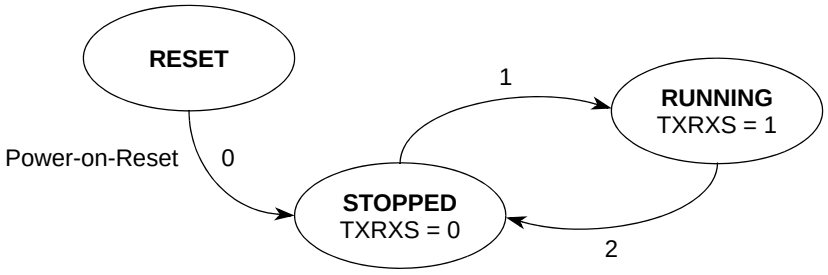


Figure 19-18. DSPI Start and Stop State Diagram

The transitions are described in Table 19-17.

Table 19-17. State Transitions for Start and Stop of DSPI Transfers

Transition #	Current State	Next State	Description
0	RESET	STOPPED	Generic power-on-reset transition
1	STOPPED	RUNNING	The DSPI starts (transitions from STOPPED to RUNNING) when all of the following conditions are true: • EOQF bit is clear • Debug mode is deselected or the FRZ bit is clear • HALT bit is clear
2	RUNNING	STOPPED	The DSPI stops (transitions from RUNNING to STOPPED) after the current frame for any one of the following conditions: • EOQF bit is set • Debug mode is selected and the FRZ bit is set • HALT bit is set

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or on the next system clock cycle if no transfers are in progress.

19.4.3 Serial Peripheral Interface (SPI) Configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI configuration when the DCONF field in the DSPI*x*_MCR is 0b00. The SPI frames can be from 4 to 16 bits long. The data to be transmitted can come from queues stored in RAM external to the DSPI. Host software or an eDMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or an eDMA controller transfers the received data from the RX FIFO to memory external to the DSPI.

The FIFO buffer operations are described in Section 19.4.3.4, “Using the TX FIFO Buffering Mechanism,” and Section 19.4.3.5, “Using the RX FIFO Buffering Mechanism.”

The interrupt and DMA request conditions are described in Section 19.4.9, “Interrupts and DMA Requests.”

The SPI configuration supports two module-specific modes; master mode and slave mode. The FIFO operations are similar for the master mode and slave mode. The main difference is that in master mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO entry. In slave mode the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field of the TX FIFO entry is not used.

19.4.3.1 SPI Master Mode

In SPI master mode the DSPI initiates the serial transfers by controlling the serial communications clock (SCK*x*) and the peripheral chip select (PCS*x*) signals. The SPI command field in the executing TX FIFO entry determines which CTARs are used to set the transfer attributes and which PCS*x* signal to assert. The command field also contains various bits that help with queue management and transfer protocol. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the serial out (SOUT*x*) pin. In SPI master mode, each SPI frame that is transmitted has a command that controls the transfer on a frame-by-frame basis.

See Section 19.3.2.6, “DSPI PUSH TX FIFO Register (DSPIx_PUSHR),” for details on the SPI command fields.

19.4.3.2 SPI Slave Mode

In SPI slave mode the DSPI responds to transfers initiated by an SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for successful communication with an SPI master. The SPI slave mode transfer attributes are set in the DSPI*x*_CTAR0.

19.4.3.3 FIFO Disable Operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The TX and RX FIFOs are disabled separately.

- Disable the TX FIFO by writing a 1 to the DIS_TXF bit in the DSPI*x*_MCR.
- Disable the RX FIFO is disabled by writing a 1 to the DIS_RXF bit in the DSPI*x*_MCR.

The FIFO disable mechanisms are transparent to the host software; transmit data and commands are written to the DSPI*x*_PUSHR and received data is read from the DSPI*x*_POPR.

- When the TX FIFO is disabled, the TFFF, TFUF, and TXCTR fields in DSPI*x*_SR use one-entry FIFO. The contents of the DSPI*x*_TXFRs and TXNXTPTR are undefined.
- When the RX FIFO is disabled, the RFDF, RFOF, and RXCTR fields in the DSPI*x*_SR behave as if there is a one-entry FIFO but the contents of the DSPI*x*_RXFRs and POPNXTPTR are undefined.

Disable the TX and RX FIFOs only if required by the application's operating mode. You must disable a FIFO before it is accessed. Failure to disable a FIFO prior to the first FIFO access can result in invalid results and is not supported.

19.4.3.4 Using the TX FIFO Buffering Mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds four entries, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI push TX FIFO register (DSPI*x*_PUSHR). For more information on DSPI*x*_PUSHR. TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO.

See Section 19.3.2.6, “DSPI PUSH TX FIFO Register (DSPIx_PUSHR).”

The TX FIFO counter field (TXCTR) in the DSPI status register (DSPI*x*_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI _PUSHR is written or SPI data is transferred into the shift register from the TX FIFO.

See Section 19.3.2.4, “DSPI Status Register (DSPIx_SR)” for more information on DSPI*x*_SR.

The TXNXTPTR field indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI*x*_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPI*x*_TXFR2 contains the SPI data and command for the next transfer. The TXNXTPTR field increments every time SPI data is transferred from the TX FIFO to the shift register.

19.4.3.4.1 Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPI*x*_PUSHR. When the TX FIFO is not full, the TX FIFO fill flag (TFFF) in the DSPI*x*_SR is set. The TFFF bit is cleared when the TX FIFO is full and the eDMA controller indicates that a write to DSPI*x*_PUSHR is complete or alternatively by host software writing a 1 to the TFFF in the DSPI*x*_SR. The TFFF can generate a DMA request or an interrupt request.

See Section 19.4.9.2, “Transmit FIFO Fill Interrupt or DMA Request (TFFF),” for details.

The DSPI ignores attempts to push data to a full TX FIFO; that is, the state of the TX FIFO is unchanged. No error condition is indicated.

19.4.3.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO counter is decremented by one. At the end of a transfer, the TCF bit in the DSPI*x*_SR is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a 1 to the CLR_TXF bit in DSPI*x*_MCR.

If an external SPI bus master initiates a transfer with a DSPI slave while the slave’s DSPI TX FIFO is empty, the transmit FIFO underflow flag (TFUF) in the slave’s DSPI*x*_SR is set.

See Section 19.4.9.4, “Transmit FIFO Underflow Interrupt Request (TFUF),”for details.

19.4.3.5 Using the RX FIFO Buffering Mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds four received SPI data frames. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data is captured from the RX FIFO by reading the DSPI*x*_POPR register. RX FIFO entries are removed from the RX FIFO by reading the DSPI*x*_POPR or by flushing the RX FIFO.

See Section 19.3.2.7, “DSPI POP RX FIFO Register (DSPIx_POPR)” for more information on the DSPI*x*_POPR.

The RX FIFO counter field (RXCTR) in the DSPI status register (DSPI*x*_SR) indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPI _POPR is read or SPI data is copied from the shift register to the RX FIFO.

The POPNXTPTR field in the DSPI*x*_SR points to the RX FIFO entry that is returned when the DSPI*x*_POPR is read. The POPNXTPTR contains the positive, 32-bit word offset from DSPI*x*_RXFR0. For example, a POPNXTPTR equal to two means that the DSPI*x*_RXFR2 contains the received SPI data that is returned when DSPI*x*_POPR is read. The POPNXTPTR field increments every time the DSPI*x*_POPR is read, and starts over again every four frames.

19.4.3.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time an SPI frame is transferred to the RX FIFO the RX FIFO counter increments by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPI*x*_SR is set indicating an overflow condition. Depending on the state of the ROOE bit in the DSPI*x*_MCR, the data from the transfer that generated the overflow is ignored or put in the shift register. If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored.

19.4.3.5.2 Draining the RX FIFO

Host software or the eDMA can remove (pop) entries from the RX FIFO by reading the DSPI*x*_POPR. A read of the DSPI*x*_POPR decrements the RX FIFO counter by one. Attempts to pop data from an empty RX FIFO are ignored, the RX FIFO counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

See Section 19.3.2.7, “DSPI POP RX FIFO Register (DSPIx_POPR)” for more information on DSPI*x*_POPR.

When the RX FIFO is not empty, the RX FIFO drain flag (RFDF) in the DSPI*x*_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the eDMA controller indicates that a read from DSPI*x*_POPR is complete; alternatively the RFDF bit can be cleared by the host writing a 1 to it.

19.4.4 Deserial Serial Interface (DSI) Configuration

The DSI configuration supports pin count reduction by serializing parallel input signals or register bits and shifting them out in an SPI-like protocol. The received serial frames are converted to a parallel form (deserialized) and placed on the parallel output signals or in a register. The various features of the DSI configuration are set in the DSPI*x*_DSICR. For more information on the DSPI*x*_DSICR. The DSPI is in DSI configuration when the DCONF field in the DSPI*x*_MCR is 0b01.

See Section 19.4.7, “Transfer Formats” for a description of the timing and transfer protocol and Section 19.3.2.10, “DSPI DSI Configuration Register (DSPIx_DSICR).”

The DSI frames can be from 4 to 16 bits long. With multiple transfer operation (MTO), the DSPI supports serial chaining of DSPI modules within the MCU to create DSI frames consisting of concatenated bits from multiple DSPIs. The DSPI also supports parallel chaining allowing several DSPIs and off-chip SPI devices to share the same serial communications clock (SCK) and peripheral chip select (PCS) signals.

See Section 19.4.4.7, “Multiple Transfer Operation (MTO),” for details on serial and parallel chaining support.

19.4.4.1 DSI Master Mode

In DSI master mode the DSPI initiates and controls the DSI transfers. The DSI master has four different conditions that can initiate a transfer:

- Continuous
- Change in data
- Trigger signal
- Trigger signal combined with a change in data

The four transfer initiation conditions are described in Section 19.4.4.5, “DSI Transfer Initiation Control.” Transfer attributes are set during initialization. The DSICTAS field in the DSPI*x*_DSICR determines which of the DSPI*x*_CTARs controls the transfer attributes.

19.4.4.2 DSI Slave Mode

In DSI slave mode the DSPI responds to transfers initiated by an SPI or DSI bus master. In this mode the DSPI does not initiate DSI transfers. Certain transfer attributes such as clock polarity and phase must be set for successful communication with a DSI master. The DSI slave mode transfer attributes are set in the DSPI*x*_CTAR1.

If the CID bit in the DSPI*x*_DSICR is set and the data in the DSPI*x*_COMPR differs from the selected source of the serialized data, the slave DSPI asserts the $\overline{\text{MTRIG}}$ signal. If the slave's internal hardware trigger signal is asserted and the TRRE is set, the slave DSPI asserts $\overline{\text{MTRIG}}$. These features are included to support chaining of several DSPI. Details about the $\overline{\text{MTRIG}}$ signal are found in Section 19.4.4.7, "Multiple Transfer Operation (MTO)."

19.4.4.3 DSI Serialization

In the DSI configuration, 4 to 16 bits can be serialized using two different sources. The TXSS bit in the DSPI*x*_DSICR selects between the DSPI*x*_SDR and DSPI*x*_ASDR as the source of serialized data. See Section 19.3.2.11, "DSPI DSI Serialization Data Register (DSPIx_SDR)," and Section 19.3.2.12, "DSPI DSI Alternate Serialization Data Register (DSPIx_ASDR)," for more details. The DSPI*x*_SDR holds the latest parallel input signal values which is sampled at every rising edge of the system clock. The DSPI*x*_ASDR is written by host software and used as an alternate source of serialized data.

A copy of the last DSI frame shifted out of the shift register is stored in the DSPI*x*_COMPR. This register provides added visibility for debugging and it serves as a reference for transfer initiation control. Section 19.3.2.13, "DSPI DSI Transmit Comparison Register (DSPIx_COMPR)," contains details on the DSPI*x*_COMPR.

Figure 19-19 shows the DSI serialization logic.

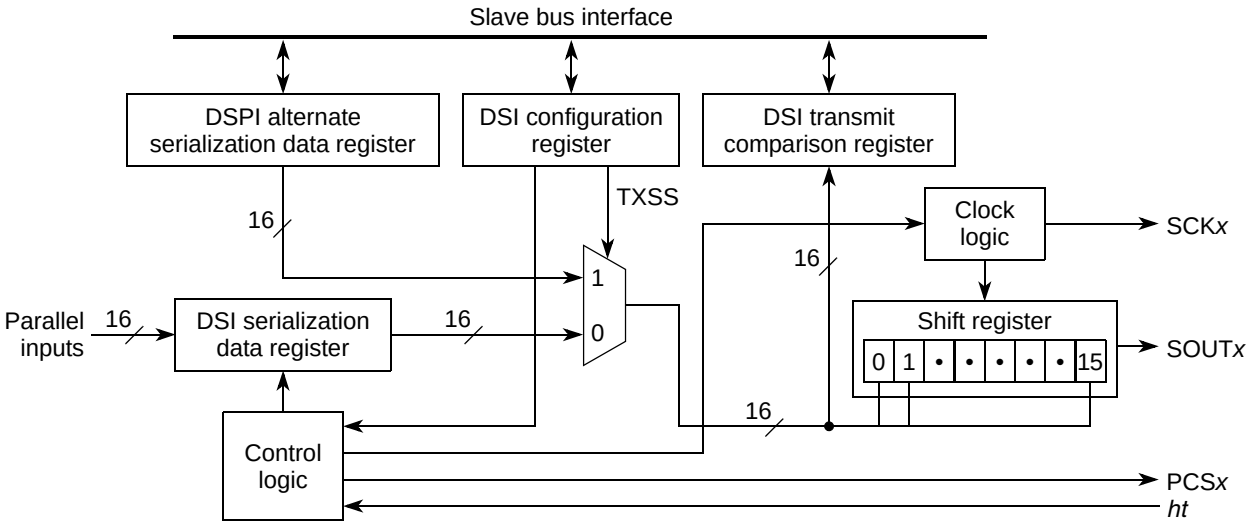


Figure 19-19. DSI Serialization Diagram

19.4.4.4 DSI Deserialization

When all bits in a DSI frame have been shifted in, the frame is copied to the DSPI*x*_DDR. This register presents the deserialized data as parallel output signal values. The DSPI*x*_DDR is memory mapped to allow host software to read the deserialized data directly. Figure 19-20 shows the DSI deserialization logic. for more information on the DSPI*x*_DDR.

See Section 19.3.2.14, "DSPI DSI Deserialization Data Register (DSPIx_DDR)."

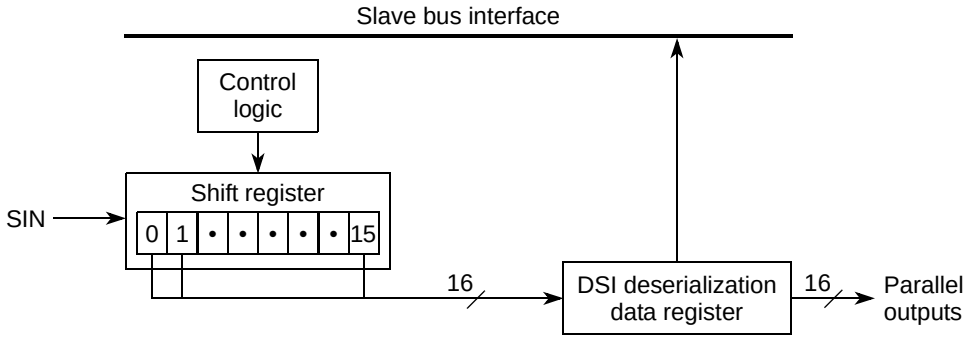


Figure 19-20. DSI Deserialization Diagram

19.4.4.5 DSI Transfer Initiation Control

Data transfers for a master DSPI in DSI configuration are initiated by a condition. When chaining DSPIs, the master and all slaves must be configured for the transfer initiation. The transfer initiation conditions are selected by the TRRE and CID bits in the DSPI*x*_DSICR.

Table 19-18 lists the four transfer initiation conditions.

Table 19-18. DSI Data Transfer Initiation Control

DSPI*x*_DSICR Bits		Type of Transfer Initiation Control
TRRE	CID	
0	0	Continuous
0	1	Change in data
1	0	Triggered
1	1	Triggered or change in data

19.4.4.5.1 Continuous Control

For continuous control, the initiation of a transfer is based on the baud rate at which data is transferred between the DSPI and the external device. The baud rate is set in the DSPI*x*_CTAR selected by the DSICTAS field in the DSPI*x*_DSICR. A new DSI frame shifts out when the previous transfer cycle has completed and the delay after transfer (t_{DT}) has elapsed.

19.4.4.5.2 Change In Data Control

For change in data control, a transfer is initiated when the data to be serialized has changed since the transfer of the last DSI frame. A copy of the previously transferred DSI data is stored in the DSPI*x*_COMPR. When the data in the DSPI*x*_SDR or the DSPI*x*_ASDR is different from the data in the DSPI*x*_COMPR, a new DSI frame is transmitted. The TXSS bit in the DSPI*x*_DSICR selects which register the DSPI*x*_COMPR is compared to. The $\overline{\text{MTRIG}}$ output signal is asserted every time a change in data is detected.

19.4.4.5.3 Triggered Control

For triggered control, initiation of a transfer is controlled by the internal hardware trigger signal (*ht*). The TPOL bit in the DSPI*x*_DSICR selects the active edge of *ht*. For *ht* to have any affect, the TRRE bit in the DSPI*x*_DSICR must be set.

19.4.4.5.4 Triggered or Change In Data Control

For triggered or change in data control, initiation of a transfer is controlled by the *ht* signal or by the detection of a change in data to be serialized.

19.4.4.6 DSPI Connections to eTPUA, eMIOS and SIU

The three DSPI blocks connect to the input and output channels of the eTPUs and the eMIOS. The MCU connects to the eTPUA, eMIOS, and SIU. Some of the DSPI outputs connect to the external interrupt input multiplexing subblock in the SIU. See Section 6.5.3, “External Interrupt” for details on how the DSPI deserialized outputs can be used to trigger external interrupt requests and Section 17.4.1, “Output and Input Channel Signals” for a discussion on eTPU connections.

19.4.4.6.1 DSPI B Connectivity

The DSPI B connects to the eMIOS, eTPUA, and SIU as shown in Figure 19-21.

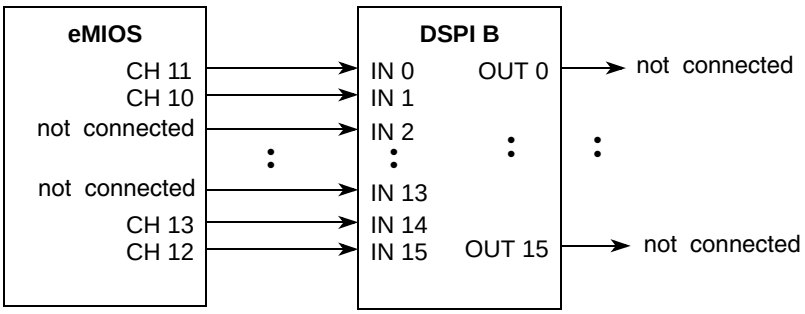


Figure 19-21. eMIOS and DSPI B Connectivity

19.4.4.6.2 DSPI B Connectivity

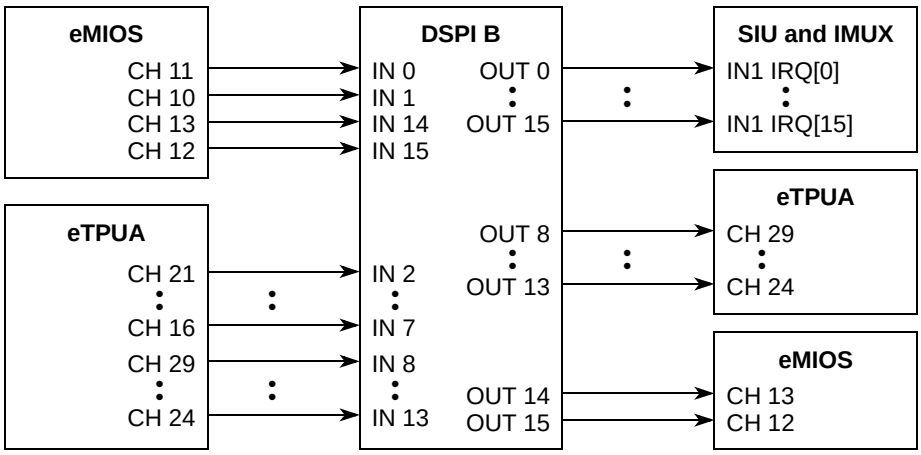


Figure 19-22. eMIOS, eTPUA and DSPI B Connectivity

Table 19-19. eMIOS and DSPI B Connectivity

Connected to:	DSPI B IN[*n*]
eMIOS output channel 11	0
eMIOS output channel 10	1
Not connected	2
Not connected	3
Not connected	4
Not connected	5
Not connected	6
Not connected	7
Not connected	8
Not connected	9
Not connected	10
Not connected	11
Not connected	12
Not connected	13
eMIOS output channel 13	14
eMIOS output channel 12	15

DSPI B OUT[*n*]	Connected to:
0	Not connected
1	Not connected
2	Not connected
3	Not connected
4	Not connected
5	Not connected
6	Not connected
7	Not connected
8	Not connected
9	Not connected
10	Not connected
11	Not connected
12	Not connected
13	Not connected
14	Not connected
15	Not connected

Table 19-20. eMIOS, eTPUA, and DSPI B Connectivity

Connected to:	DSPI B IN[*n*]
eMIOS output channel 11	0
eMIOS output channel 10	1
eTPUA output channel 21	2
eTPUA output channel 20	3
eTPUA output channel 19	4
eTPUA output channel 18	5
eTPUA output channel 17	6
eTPUA output channel 16	7
eTPUA output channel 29	8
eTPUA output channel 28	9
eTPUA output channel 27	10
eTPUA output channel 26	11
eTPUA output channel 25	12
eTPUA output channel 24	13
eMIOS output channel 13	14
eMIOS output channel 12	15

DSPI B OUT[*n*]	Connected to:
0	Input 1 on IMUX for external $\overline{\text{IRQ}}$[0]
1	Input 1 on IMUX for external $\overline{\text{IRQ}}$[1]
2	Input 1 on IMUX for external $\overline{\text{IRQ}}$[2]
3	Input 1 on IMUX for external $\overline{\text{IRQ}}$[3]
4	Input 1 on IMUX for external $\overline{\text{IRQ}}$[4]
5	Input 1 on IMUX for external $\overline{\text{IRQ}}$[5]
6	Input 1 on IMUX for external $\overline{\text{IRQ}}$[6]
7	Input 1 on IMUX for external $\overline{\text{IRQ}}$[7]
8	eTPUA input channel 29, input 1 on IMUX for external $\overline{\text{IRQ}}$[8]
9	eTPUA input channel 28, input 1 on IMUX for external $\overline{\text{IRQ}}$[9]
10	eTPUA input channel 27, input 1 on IMUX for external $\overline{\text{IRQ}}$[10]
11	eTPUA input channel 26, input 1 on IMUX for external $\overline{\text{IRQ}}$[11]
12	eTPUA input channel 25, input 1 on IMUX for external $\overline{\text{IRQ}}$[12]
13	eTPUA input channel 24, input 1 on IMUX for external $\overline{\text{IRQ}}$[13]
14	eMIOS input channel 13, input 1 on IMUX for external $\overline{\text{IRQ}}$[14]
15	eMIOS input channel 12, input 1 on IMUX for external $\overline{\text{IRQ}}$[15]

19.4.4.6.3 DSPI C Connectivity

The DSPI C is not connected, as shown in Figure 19-23.

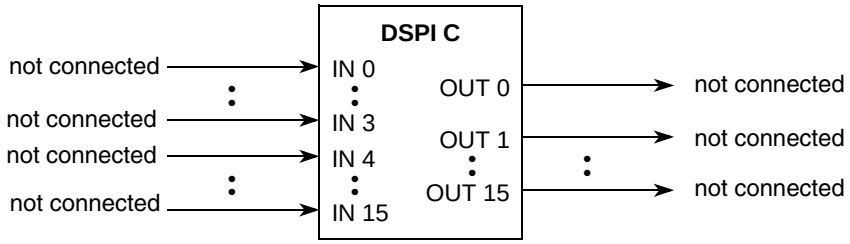


Figure 19-23. DSPI C Connectivity

Table 19-21 shows that DSPI C is not connected for this device.

Table 19-21. eTPU and DSPI C Not Connected

eTPU Channel	DSPI C IN[*n*]	DSPI C OUT[*n*]	Connected to:
Not connected	0	0	Not connected
Not connected	1	1	Not connected
Not connected	2	2	Not connected
Not connected	3	3	Not connected
Not connected	4	4	Not connected
Not connected	5	5	Not connected
Not connected	6	6	Not connected
Not connected	7	7	Not connected
Not connected	8	8	Not connected
Not connected	9	9	Not connected
Not connected	10	10	Not connected
Not connected	11	11	Not connected
Not connected	12	12	Not connected
Not connected	13	13	Not connected
Not connected	14	14	Not connected
Not connected	15	15	Not connected

The DSPI C connects to the eTPUA and the SIU as shown in Figure 19-24.

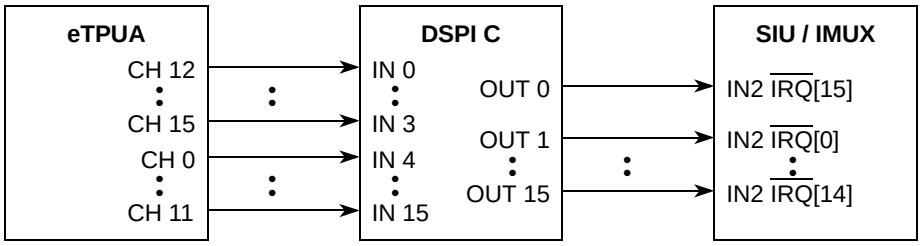


Figure 19-24. eTPUA and DSPI C Connectivity

Table 19-22 lists the eTPUA and DSPI C connections.

Table 19-22. eTPUA and DSPI C Connectivity

eTPU Channel	DSPI C IN[*n*]	DSPI C OUT[*n*]	Connected to:
eTPUA output channel 12	0	0	Input 2 on IMUX for external $\overline{\text{IRQ}}$[15]
eTPUA output channel 13	1	1	Input 2 on IMUX for external $\overline{\text{IRQ}}$[0]

Table 19-22. eTPUA and DSPI C Connectivity (continued)

eTPU Channel	DSPI C IN[*n*]	DSPI C OUT[*n*]	Connected to:
eTPUA output channel 14	2	2	Input 2 on IMUX for external $\overline{\text{IRQ}}$[1]
eTPUA output channel 15	3	3	Input 2 on IMUX for external $\overline{\text{IRQ}}$[2]
eTPUA output channel 0	4	4	Input 2 on IMUX for external $\overline{\text{IRQ}}$[3]
eTPUA output channel 1	5	5	Input 2 on IMUX for external $\overline{\text{IRQ}}$[4]
eTPUA output channel 2	6	6	Input 2 on IMUX for external $\overline{\text{IRQ}}$[5]
eTPUA output channel 3	7	7	Input 2 on IMUX for external $\overline{\text{IRQ}}$[6]
eTPUA output channel 4	8	8	Input 2 on IMUX for external $\overline{\text{IRQ}}$[7]
eTPUA output channel 5	9	9	Input 2 on IMUX for external $\overline{\text{IRQ}}$[8]
eTPUA output channel 6	10	10	Input 2 on IMUX for external $\overline{\text{IRQ}}$[9]
eTPUA output channel 7	11	11	Input 2 on IMUX for external $\overline{\text{IRQ}}$[10]
eTPUA output channel 8	12	12	Input 2 on IMUX for external $\overline{\text{IRQ}}$[11]
eTPUA output channel 9	13	13	Input 2 on IMUX for external $\overline{\text{IRQ}}$[12]
eTPUA output channel 10	14	14	Input 2 on IMUX for external $\overline{\text{IRQ}}$[13]
eTPUA output channel 11	15	15	Input 2 on IMUX for external $\overline{\text{IRQ}}$[14]

19.4.4.6.4 DSPI D Connectivity

The DSPI D connects to the eTPUA, eMIOS and SIU as shown in Figure 19-25.

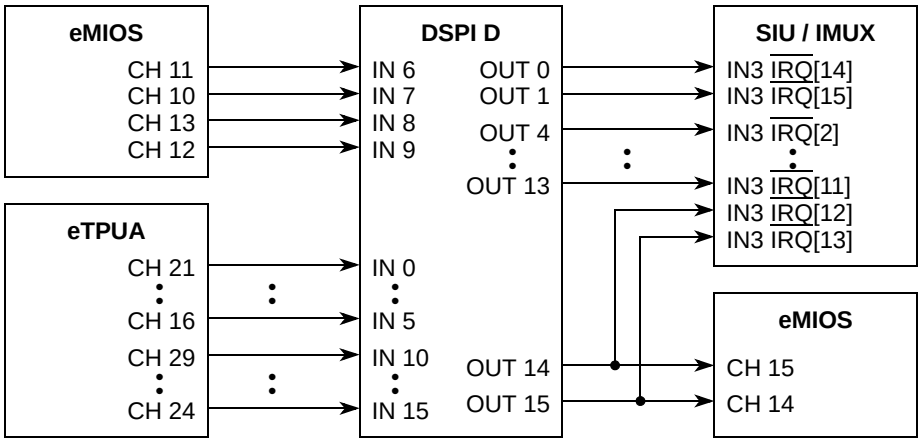


Figure 19-25. DSPI D Connectivity

Table 19-23 lists the DSPI D connections.

Table 19-23. DSPI D Connectivity Table

Connected to:	DSPI D IN[*n*]	DSPI D OUT[*n*]	Connected to:
eTPUA output channel 21	0	0	Input 3 on IMUX for external $\overline{\text{IRQ}}$[14]
eTPUA output channel 20	1	1	Input 3 on IMUX for external $\overline{\text{IRQ}}$[15]
eTPUA output channel 19	2	2	no connect
eTPUA output channel 18	3	3	no connect
eTPUA output channel 17	4	4	Input 3 on IMUX for external $\overline{\text{IRQ}}$[2]
eTPUA output channel 16	5	5	Input 3 on IMUX for external $\overline{\text{IRQ}}$[3]
eMIOS output channel 11	6	6	Input 3 on IMUX for external $\overline{\text{IRQ}}$[4]
eMIOS output channel 10	7	7	Input 3 on IMUX for external $\overline{\text{IRQ}}$[5]
eMIOS output channel 13	8	8	Input 3 on IMUX for external $\overline{\text{IRQ}}$[6]
eMIOS output channel 12	9	9	Input 3 on IMUX for external $\overline{\text{IRQ}}$[7]
eTPUA output channel 29	10	10	Input 3 on IMUX for external $\overline{\text{IRQ}}$[8]
eTPUA output channel 28	11	11	Input 3 on IMUX for external $\overline{\text{IRQ}}$[9]
eTPUA output channel 27	12	12	Input 3 on IMUX for external $\overline{\text{IRQ}}$[10]
eTPUA output channel 26	13	13	Input 3 on IMUX for external $\overline{\text{IRQ}}$[11]
eTPUA output channel 25	14	14	eMIOS input channel 15, Input 3 on IMUX for external $\overline{\text{IRQ}}$[12]
eTPUA output channel 24	15	15	eMIOS input channel 14, Input 3 on IMUX for external $\overline{\text{IRQ}}$[13]

19.4.4.7 Multiple Transfer Operation (MTO)

In DSI configuration the MTO feature allows for multiple DSPIs within the MCU to be chained together in a parallel or serial configuration. The parallel chaining allows multiple DSPIs internal to the MCU and multiple SPI devices external to the MCU to share SCK and PCS signals thereby saving pins. The serial chaining allows bits from multiple DSPIs to be concatenated into a single DSI frame from 8- to 64-bits long. MTO is enabled by setting the MTOE bit in the DSPI*x*_DSICR.

In parallel and serial chaining there is one bus master and multiple bus slaves. The bus master initiates and controls the transfers, but the DSPI slaves generate trigger signals for the bus DSPI master when an internal condition in the slave warrants a transfer. The DSPI slaves also propagate triggers from other slaves to the master. When a DSPI slave detects a trigger signal on its *ht* input, the slave generates a trigger signal on the $\overline{\text{MTRIG}}$ output.

Serial and parallel chaining require multiplexing of signals external to the DSPI. Configure SIU_DISR to serial or parallel chaining. See, Section 19.4.4.7.1, “Internal Muxing and SIU Support for Serial and Parallel Chaining, for more information.

19.4.4.7.1 Internal Muxing and SIU Support for Serial and Parallel Chaining

To support MTO, each DSPI in the device has multiplexers on the SIN*x*, SS*x*, SCK*x* inputs. The internal multiplexers are controlled by registers in the SIU block.

See Section 6.5.5.3, "Multiplexed Inputs for DSPI Multiple Transfer Operation."

Figure 19-26 shows DSPI B and the multiplexers in the IMUX subblock of the SIU. The SOUT*x*, $\overline{\text{MTRIG}}$, SCK*x* and PCS*x*0 outputs from the other two DSPIs that connect to the multiplexers on the DSPI B, DSPI C inputs.

DSPI B, DSPI C and DSPI D have similar multiplexers on their inputs.

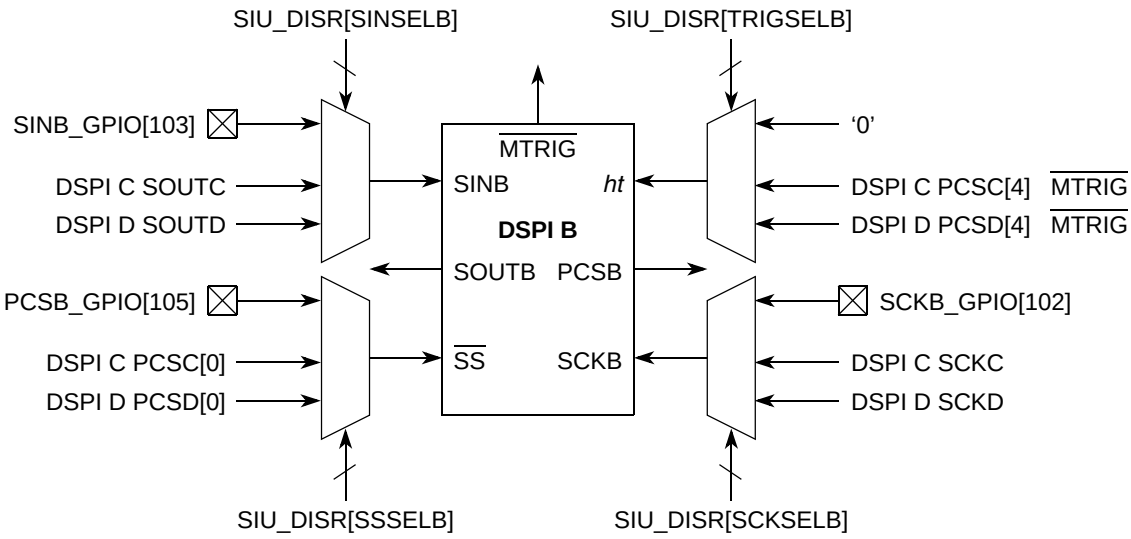


Figure 19-26. DSPI B, C, and D Inputs for Multi-transfer Operations

The source for the SIN*x* input of a DSPI can be a pin or the SOUT*x* of any of the other two DSPIs. The source for the $\overline{\text{SS}}$*x* input of a DSPI can be a pin or the PCS*x*[0] signal from any of the other DSPIs. The source for the SCK*x* input of a DSPI can be a pin or the SCK*x* output of any of the other DSPIs. The source for the hardware trigger (*ht*) input can be the $\overline{\text{MTRIG}}$ signal from any of the other DSPIs. The DSPI input select register (SIU_DSR) selects the source for each DSPI SIN*x*, $\overline{\text{SS}}$*x*, SCK*x* signal individually.

19.4.4.7.2 Parallel Chaining

Parallel chaining allows an internal slave device and an external slave device to share the PCS*x* and SCK*x* signals from a single master DSPI. Signal sharing reduces the number of DSPI pin used. An example of a parallel chain is shown in Figure 19-27.

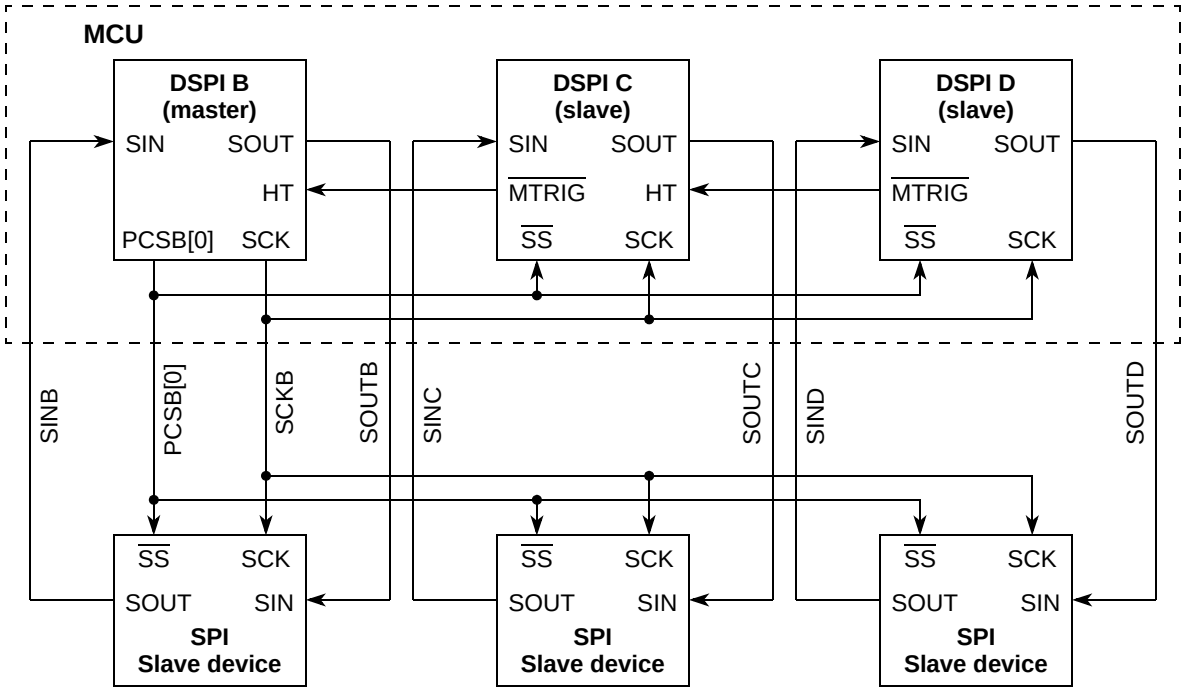


Figure 19-27. Example of Parallel Chaining of DSPI B, C and D

In the parallel chaining example, the SOUT*x* and SIN*x* of the DSPIs connect to separate external SPI devices. All internal and external SPI modules share PCS*x* and SCK*x* signals. DSPI B controls and initiates all transfers, but the DSPI slaves each have a trigger output signal $\overline{\text{MTRIG}}$ that indicates to DSPI B that a trigger condition has occurred in the DSPI slaves. DSPI B controls and initiates all transfers, but the DSPI slave has a trigger output signal $\overline{\text{MTRIG}}$ that indicates to DSPI B that a trigger condition has occurred in the DSPI slave.

When the slave DSPI has a change in data to be serialized, it asserts the $\overline{\text{MTRIG}}$ signal that propagates to DSPI B which initiates the transfer.

The MTOCNT field in the DSPI*x*_DSICR must be written with the number of bits to be transferred. In parallel chaining the number written to MTOCNT must match the FMSZ field in the selected DSPI*x*_CTAR.

19.4.4.7.3 Serial Chaining

Serial chaining allows transfers of DSI frames consisting of concatenated bits from multiple DSPIs. The concatenated frames can be from 8- to 64-bits long. Figure shows an example of how the modules can be connected.

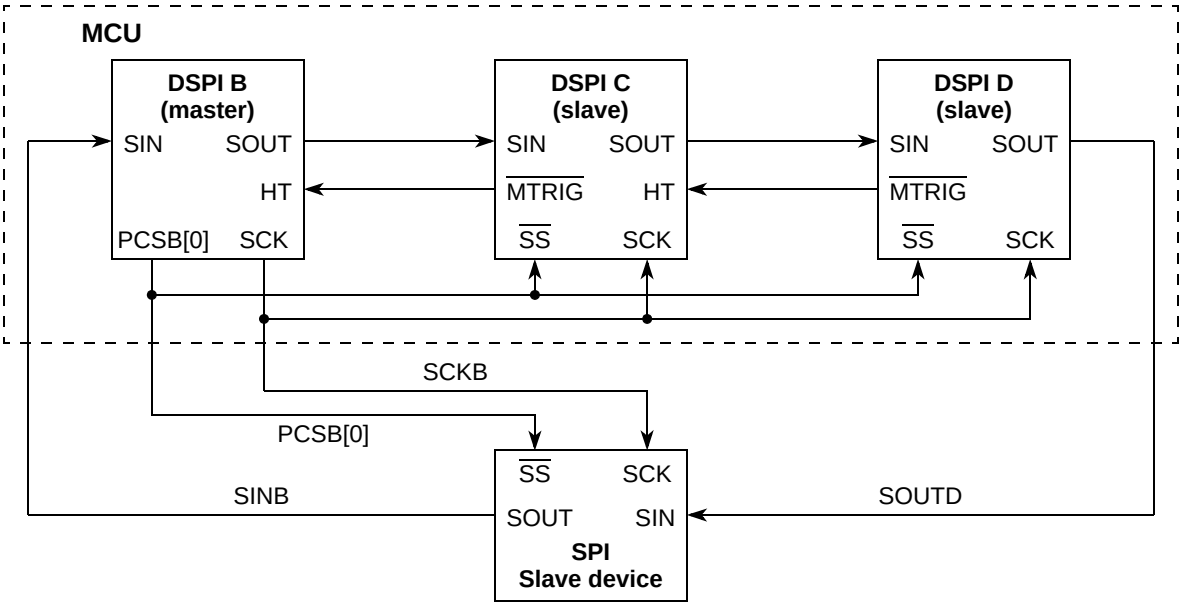


Figure 19-28. Example of Serial Chaining of DSPI B, C and D

In the MCU (master), the SOUT of DSPI B is connected to the SIN of the DSPI C (slave). The SOUT of the DSPI C (slave) is connected to the SIN input of the DSPI D slave and so on. The SOUT of the last on-chip DSPI slave is connected to the SIN of the external SPI slave. The SOUT of the external SPI slave is connected to the SIN of the DSPI B master.

The DSPI B master controls and initiates all transfers, but the slave DSPIs use the trigger output signal $\overline{\text{MTRIG}}$ to indicate to the DSPI B master that a trigger condition has occurred. When an on-chip DSPI slave has a change in data to be serialized it can assert the $\overline{\text{MTRIG}}$ signal to the DSPI master which initiates the transfer. When a DSPI slave has its *ht* signal asserted, its $\overline{\text{MTRIG}}$ signal asserts and propagates trigger signals from other DSPI slaves to the DSPI master.

The MTOCNT field in the DSPI*x*_DSICR must be written with the total number of bits to be transferred. The MTOCNT field must equal the sum of all FMSZ fields in the selected DSPI*x*_CTARs for the DSPI master and all DSPI slaves. For example, if one 16-bit DSI frame is created by concatenating 8 bits from the DSPI master, and 4 bits from each of the DSPI slaves in Figure 19-28, the DSPI master's frame size must be set to eight in the FMSZ field, and the DSPI slaves' frame size must be set to four. The largest DSI frame supported by the MTOCNT field is 48 bits. Any number of DSPIs can be connected together to concatenate DSI frames, as long as each DSPI transfers a minimum of 4 bits and a maximum of 16 bits and the total size of the concatenated frame is less than or equal to 48 bits long.

19.4.5 Combined Serial Interface (CSI) Configuration

In master mode, the CSI configuration of the DSPI is used to support SPI and DSI functions on a frame by frame basis. CSI configuration allows interleaving of DSI data frames from the parallel input signals (from the eTPU or eMIOS) with SPI commands and data from the TX FIFO. The data returned from the bus slave is either used to drive the parallel output signals (to the eTPU or eMIOS) or is stored in the RX FIFO. CSI configuration allows serialized data and configuration or diagnostic data to be transferred to a slave device using only one serial link. The DSPI is in CSI configuration when the DCONF field in the DSPI*x*_MCR is 0b10. Figure 19-29 shows an example of how a DSPI can be used with a deserializing peripheral that supports SPI control for control and diagnostic frames.

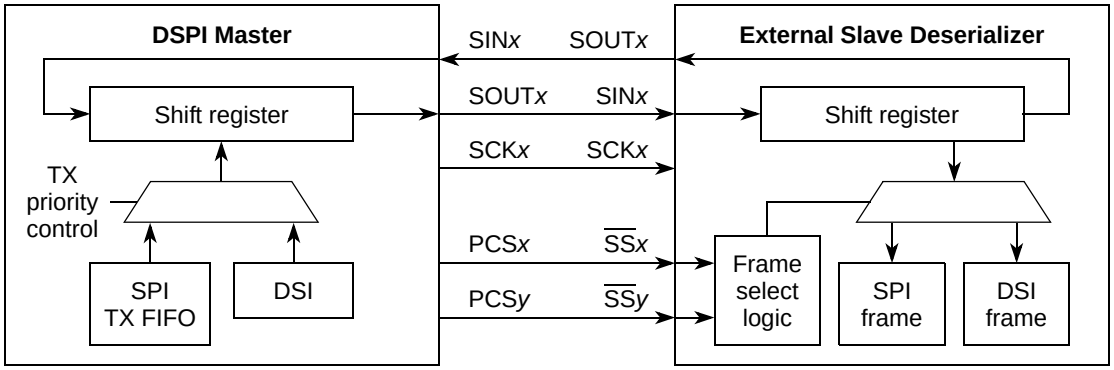


Figure 19-29. Example of System using DSPI in CSI Configuration

In CSI configuration the DSPI transfers DSI data based on Section 19.4.4.5, “DSI Transfer Initiation Control.” When there are SPI commands in the TX FIFO, the SPI data has priority over the DSI frames. When the TX FIFO is empty, DSI transfer resumes.

Two peripheral chip select signals indicate whether DSI data or SPI data is transmitted. You must configure the DSPI so the CTARs for the DSI data and SPI data assert different peripheral chip select signals denoted in the figure as PCSx and PCSy. The CSI configuration is only supported in master mode.

Data returned from the external slave while a DSI frame is transferred is placed on the parallel output signals. Data returned from the external slave while an SPI frame is transferred is moved to the RX FIFO. The TX FIFO and RX FIFO are fully functional in CSI mode.

19.4.5.1 CSI Serialization

Serialization in the CSI configuration is similar to serialization in DSI configuration. The transfer attributes for SPI frames are determined by the DSPI*x*_CTAR selected by the CTAS field in the SPI command halfword. The transfer attributes for the DSI frames are determined by the DSPI*x*_CTAR selected by the DSICTAS field in the DSPI*x*_DSICR. Figure 19-30 shows the CSI serialization logic.

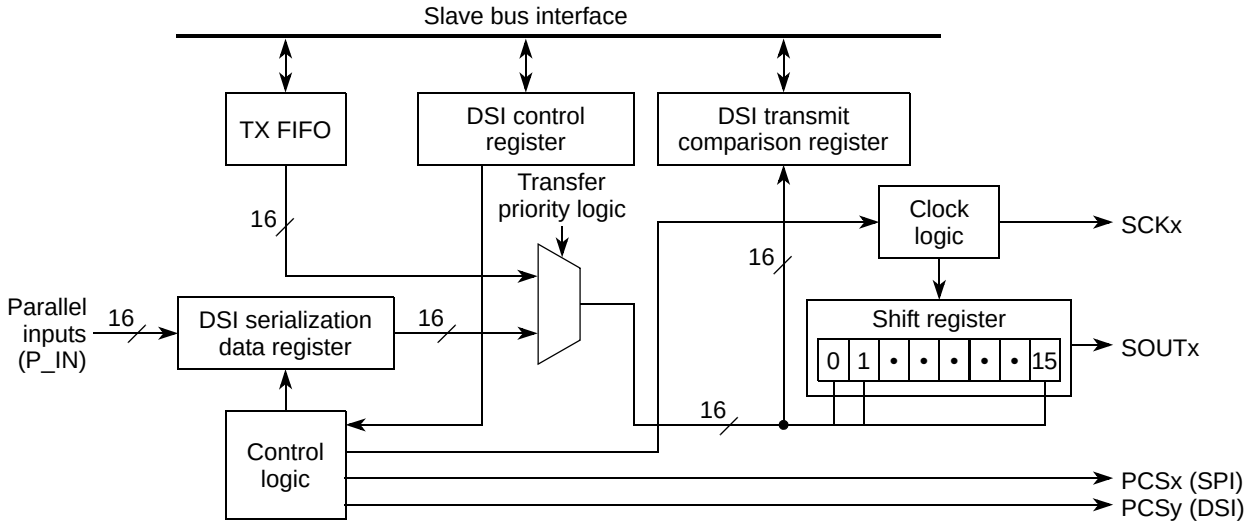


Figure 19-30. CSI Serialization Diagram

The parallel inputs signal states are latched into the DSPI*x*_SDR on the rising edge of every system clock and serialized based on the transfer initiation control settings in the DSPI*x*_DSICR. For more information on the DSPI*x*_SDR. SPI frames written to the TX FIFO have priority over DSI data from the DSPI*x*_SDR and are transferred at the next frame boundary. A copy of the most recently transferred DSI frame is stored in the DSPI*x*_COMPR. The transfer priority logic selects the source of the serialized data and asserts the chip select signal.

See Section 19.3.2.11, "DSPI DSI Serialization Data Register (DSPIx_SDR)."

19.4.5.2 CSI Deserialization

The deserialized frames in CSI configuration go into the DSPI*x*_SDR or the RX FIFO based on the transfer priority logic. When DSI frames are transferred the returned frames are deserialized and latched into the DSPI*x*_DDR. When SPI frames are transferred the returned frames are deserialized and written to the RX FIFO.

Figure 19-31 shows the CSI deserialization logic.

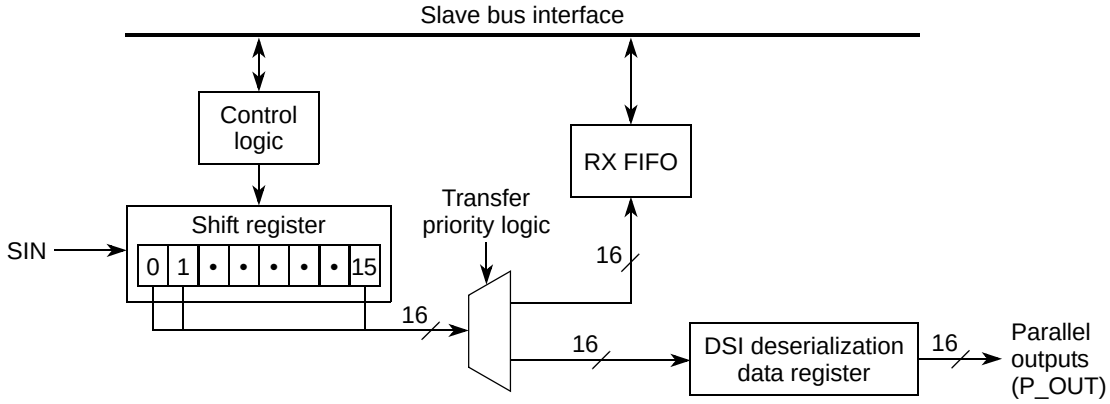


Figure 19-31. CSI Deserialization Diagram

19.4.6 DSPI Baud Rate and Clock Delay Generation

The SCK*x* frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option of doubling the baud rate.

Figure 19-32 shows conceptually how the SCK signal is generated.

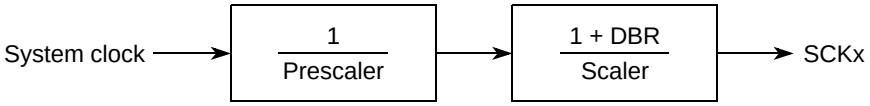


Figure 19-32. Communications Clock Prescalers and Scalers

19.4.6.1 Baud Rate Generator

The baud rate is the frequency of the serial communication clock (SCK*x*). The system clock is divided by a baud rate prescaler (defined by DSPI*x*_CTAR[PBR]) and baud rate scaler (defined by DSPI*x*_CTAR[BR]) to produce SCK*x* with the possibility of doubling the baud rate. The DBR, PBR, and BR fields in the DSPI*x*_CTARs select the frequency of SCK*x* using the following formula:

$$\text{SCK baud rate} = \frac{f_{SYS}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$$

Table 19-24 shows an example of a computed baud rate.

Table 19-24. Baud Rate Computation Example

f_{SYS}	PBR	Prescaler Value	BR	Scaler Value	DBR Value	Baud Rate
82 MHz	0b00	2	0b0000	2	0	25 Mb/sec
20 MHz	0b00	2	0b0000	2	1	10 Mb/sec

19.4.6.2 PCS to SCK Delay (t_{CSC})

The PCS*x* to SCK*x* delay is the length of time from assertion of the PCS*x* signal to the first SCK*x* edge. See Figure 19-34 for an illustration of the PCS*x* to SCK*x* delay. The PCSSCK and CSSCK fields in the DSPI*x*_CTAR*n* registers select the PCS*x* to SCK*x* delay, and the relationship is expressed by the following formula:

$$t_{CSC} = \frac{1}{f_{SYS}} \times \text{PCSSCK} \times \text{CSSCK}$$

Table 19-25 shows an example of the computed PCS to SCK delay.

Table 19-25. PCS to SCK Delay Computation Example

PCSSCK	Prescaler Value	CSSCK	Scaler Value	f_{SYS}	PCS to SCK Delay
0b01	3	0b0100	32	82 MHz	0.96 µs

19.4.6.3 After SCK Delay (t_{ASC})

The after SCK*x* delay is the length of time between the last edge of SCK*x* and the negation of PCS*x*. See Figure 19-34 and Figure 19-35 for illustrations of the after SCK*x* delay. The PASC and ASC fields in the DSPI*x*_CTAR*n* registers select the after SCK delay. The relationship between these variables is given in the following formula:

$$t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$$

Table 19-26 shows an example of the computed after SCK delay.

Table 19-26. After SCK Delay Computation Example

PASC	Prescaler Value	ASC	Scaler Value	f_{SYS}	After SCK Delay
0b01	3	0b0100	32	82 MHz	0.96 µs

19.4.6.4 Delay after Transfer (t_{DT})

The delay-after-transfer field is the amount of time between negation of the PCS*x* signal for a frame and the assertion of the PCS*x* signal for the next frame. The PDT and DT fields in the DSPI*x*_CTAR*n* registers select the delay after transfer.

See Figure 19-34 for an illustration of the delay after transfer. The following formula expresses the PDT and DT delay-after-transfer relationship:

$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$

Table 19-27 shows an example of the computed delay after transfer.

Table 19-27. Delay after Transfer Computation Example

PDT	Prescaler Value	DT	Scaler Value	f_{SYS}	Delay after Transfer
0b01	3	0b1110	32768	82 MHz	0.98 ms

19.4.6.5 Peripheral Chip Select Strobe Enable ($\overline{PCSS}$)

The $\overline{PCSS}$ signal provides a delay to allow the PCS*x* signals to settle after transitioning thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI*x*_MCR, $\overline{PCSS}$ provides a signal for an external demultiplexer to decode the PCS*x*[0:4] signals into as many as 32 glitch-free PCS*x* signals.

Figure 19-33 shows the timing of the $\overline{\text{PCSS}}$ signal relative to PCS signals.

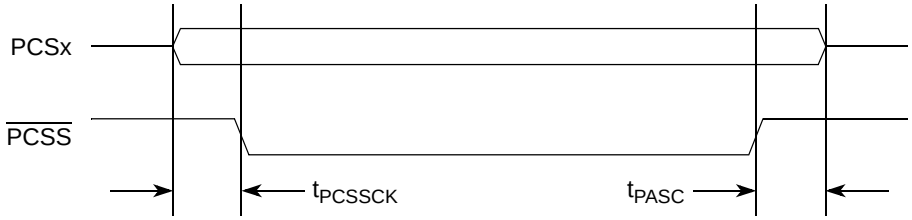


Figure 19-33. Peripheral Chip Select Strobe Timing

The delay between the assertion of the PCS*x* signals and the assertion of $\overline{\text{PCSS}}$ is selected by the PCSSCK field in the DSPI*x*_CTAR based on the following formula:

$$tPCSSCK = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between $\overline{\text{PCSS}}$ negation and PCS*x* negation is selected by the PASC field in the DSPI*x*_CTAR based on the following formula:

$$tPASC = \frac{1}{f_{SYS}} \times PASC$$

Table 19-28 shows an example of the computed t_{PCSSCK} delay.

Table 19-28. Peripheral Chip Select Strobe Assert Computation Example

PCSSCK	Prescaler	f_{SYS}	Delay before Transfer
0b11	7	82 MHz	70.0 ns

Table 19-29 shows an example of the computed the t_{PASC} delay.

Table 19-29. Peripheral Chip Select Strobe Negate Computation Example

PASC	Prescaler	f_{SYS}	Delay after Transfer
0b11	7	82 MHz	70.0 ns

19.4.7 Transfer Formats

The SPI serial communication is controlled by the serial communications clock (SCK*x*) signal and the PCS*x* signals. The SCK*x* signal provided by the master device synchronizes shifting and sampling of the data by the SIN*x* and SOUT*x* pins. The PCS*x* signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI clock and transfer attributes registers (DSPI*x*_CTAR*n*) select the polarity and phase of the serial clock, SCK*x*. The polarity bit selects the idle state of the SCK*x*. The clock phase bit selects if the data on SOUT*x* is valid before or on the first SCK*x* edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI*x*_CTAR0 (SPI slave mode) or DSPI*x*_CTAR1 (DSI slave mode) select the polarity and phase of the serial clock. Even though the bus slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master device and the slave device to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

The classic SPI formats are described in:

Section 19.4.7.1, “Classic SPI Transfer Format (CPHA = 0),” and

Section 19.4.7.2, “Classic SPI Transfer Format (CPHA = 1).”

The modified transfer formats are described in:

Section 19.4.7.3, “Modified Transfer Format Enabled (MTFE = 1) with Classic SPI Transfer Format Cleared (CPHA = 0) for SPI and DSI,” and

Section 19.4.7.4, “Modified Transfer Format Enabled (MTFE = 1) with Classic SPI Transfer Format Set (CPHA = 1) for SPI and DSI.”

A modified transfer format supports high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI*x*_MCR selects between classic SPI format and modified transfer format.

In the SPI and DSI configurations, the DSPI provides the option of keeping the PCS signals asserted between frames. See Section 19.4.7.5, “Continuous Selection Format” for details.

19.4.7.1 Classic SPI Transfer Format (CPHA = 0)

The transfer format shown in Figure 19-34 is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample the SIN*x* pins on the odd-numbered SCK*x* edges, and change the data on the SOUT*x* pins on the even-numbered SCK*x* edges.

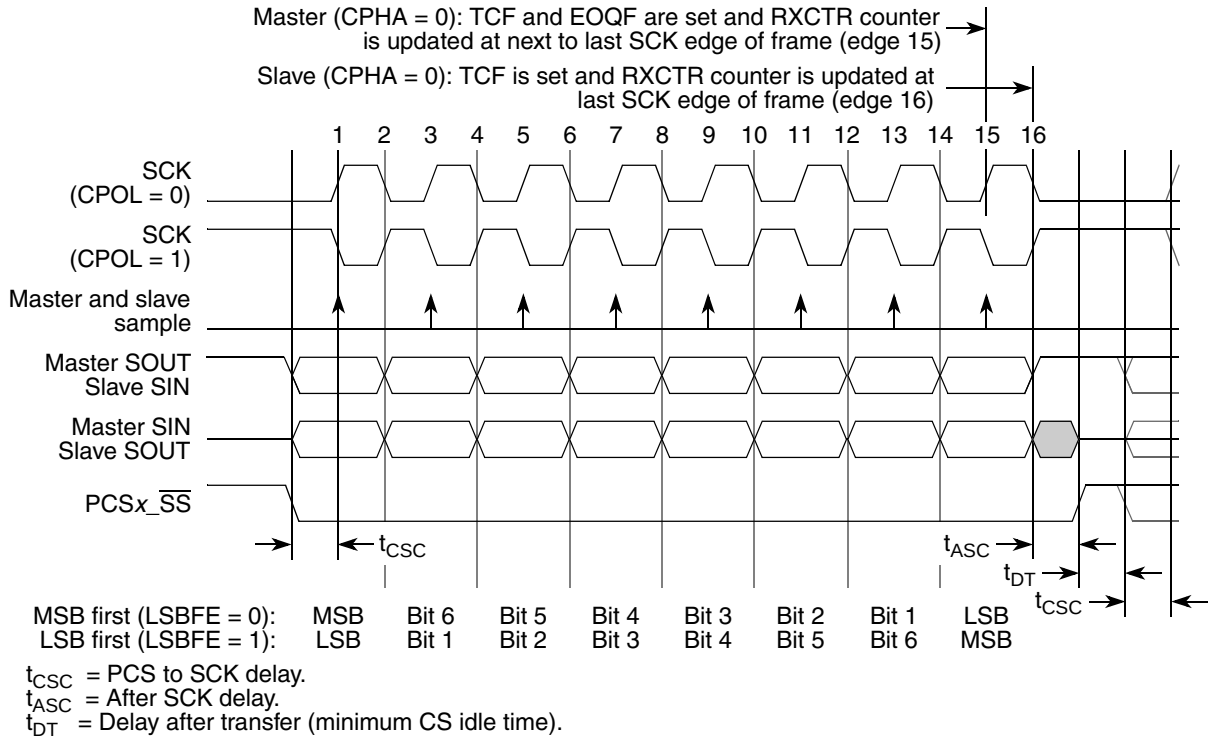


Figure 19-34. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

The master initiates the transfer by placing its first data bit on the SOUT*x* pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT*x* pin. After the t_{CSC} delay has elapsed, the master outputs the first edge of SCK*x*. This is the edge used by the master and slave devices to sample the first input data bit on their serial data input signals. At the second edge of the SCK*x* the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN*x* pins on the odd-numbered clock edges and changes the data on their SOUT*x* pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For the CPHA = 0 condition of the master, TCF and EOQF are set and the RXCTR counter is updated at the next to last serial clock edge of the frame (edge 15) of Figure 19-34.

For the CPHA = 0 condition of the slave, TCF is set and the RXCTR counter is updated at the last serial clock edge of the frame (edge 16) of Figure 19-34.

19.4.7.2 Classic SPI Transfer Format (CPHA = 1)

This transfer format shown in Figure 19-35 is used to communicate with peripheral SPI slave devices that require the first SCK*x* edge before the first data bit becomes available on the slave SOUT pin. In this format the master and slave devices change the data on their SOUT*x* pins on the odd-numbered SCK*x* edges and sample the data on their SIN*x* pins on the even-numbered SCK*x* edges.

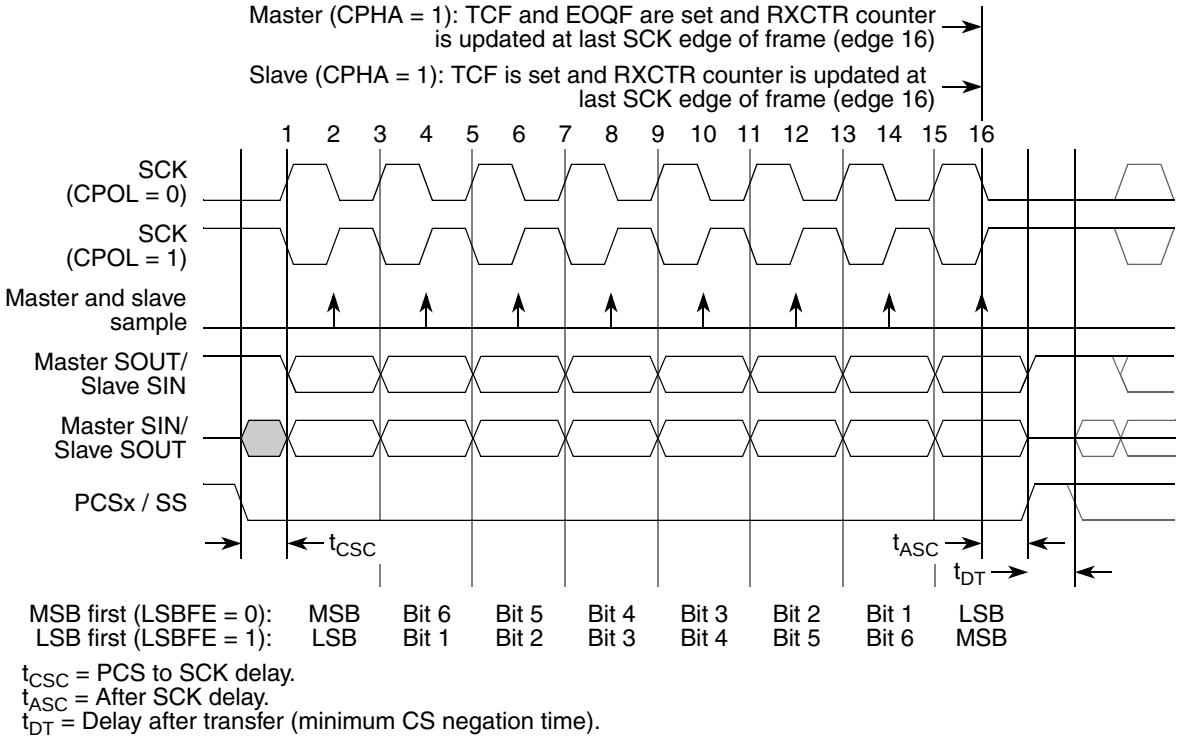


Figure 19-35. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the PCS*x* signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first SCK*x* edge and at the same time places valid data on the master SOUT*x* pin. The slave responds to the first SCK*x* edge by placing its first data bit on its slave SOUT*x* pin.

At the second edge of the SCK*x* the master and slave sample their SIN*x* pins. For the rest of the frame the master and the slave change the data on their SOUT*x* pins on the odd-numbered clock edges and sample their SIN*x* pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS*x* signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For CPHA = 1 the master EOQF and TCF and slave TCF are set at the last serial clock edge (edge 16) of Figure 19-35. For CPHA = 1 the master and slave RXCTR counters are updated on the same clock edge.

19.4.7.3 Modified Transfer Format Enabled (MTFE = 1) with Classic SPI Transfer Format Cleared (CPHA = 0) for SPI and DSI

In the modified transfer format, the master and the slave sample later in the SCK period than in classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

NOTE

For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

The master and the slave send data to the SOUT*x* pins when the PCS*x* signal asserts. After the PCS*x* to SCK*x* delay elapses the first SCK*x* edge is generated. The slave samples the master SOUT*x* signal on every odd numbered SCK*x* edge. The slave also sends more data on the slave SOUT*x* on every odd numbered clock edge.

The master sends its second data bit to the SOUT*x* pin one system clock after the odd numbered SCK*x* edge. The master samples the slave SOUT*x* pins by writing to the SMPL_PT field in the DSPI*x*_MCR. Table 19-30 lists the number of system clock cycles (between the active-edge of SCK*x* and the master sample point) for different values of the SMPL_PT bit field. The master sample point can be delayed by one or two system clock cycles.

Table 19-30. Delayed Master Sample Point

SMPL_PT	Number of System Clock Cycles between Odd-numbered Edge of SCK and Sampling of SIN
00	0
01	1
10	2
11	Invalid value

Figure 19-36 shows the modified transfer format for CPHA = 0. Only the condition where CPOL = 0 is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.

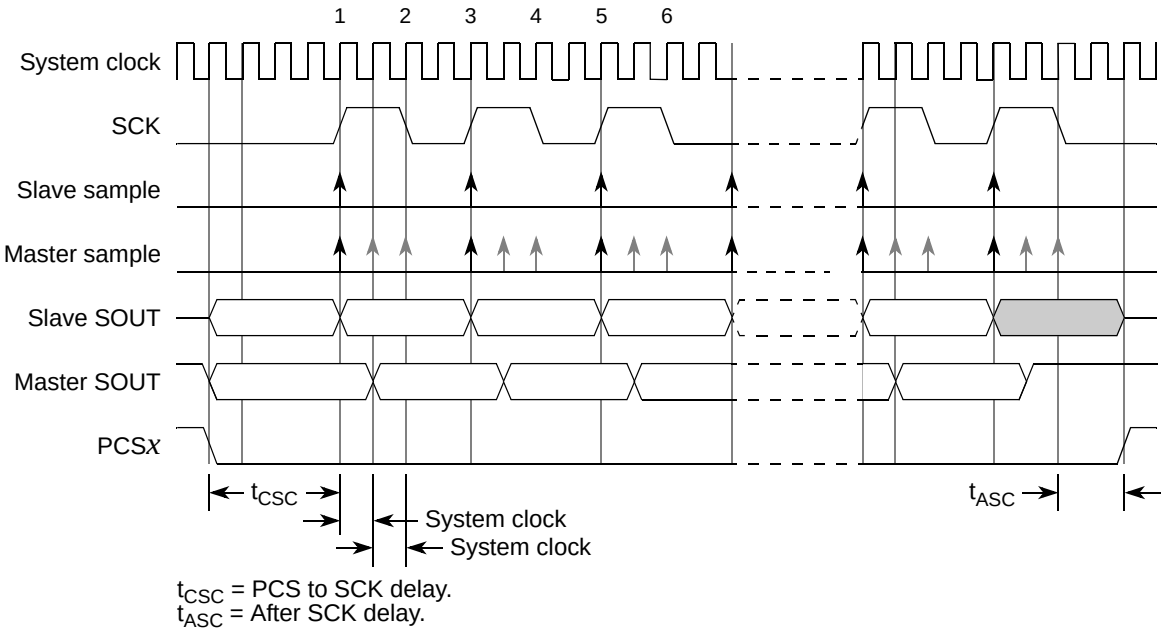


Figure 19-36. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{SCK} = f_{SYS} \div 4$)

19.4.7.4 Modified Transfer Format Enabled (MTFE = 1) with Classic SPI Transfer Format Set (CPHA = 1) for SPI and DSI

At the start of a transfer the DSPI asserts the PCS signal to the slave device. After the PCS to SCK delay has elapsed the master and the slave put data on their SOUT pins at the first edge of SCK. The slave samples the master SOUT signal on the even numbered edges of SCK. The master samples the slave SOUT signal on the odd numbered SCK edges starting with the third SCK edge. The slave samples the last bit on the last edge of the SCK. The master samples the last slave SOUT bit one half SCK cycle after the last edge of SCK. No clock edge is visible on the master SCK pin during the sampling of the last bit. The SCK-to-PCS delay must be greater or equal to half of the SCK period.

NOTE

For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

Figure 19-37 shows the modified transfer format for CPHA = 1. Only the condition where CPOL = 0 is described.

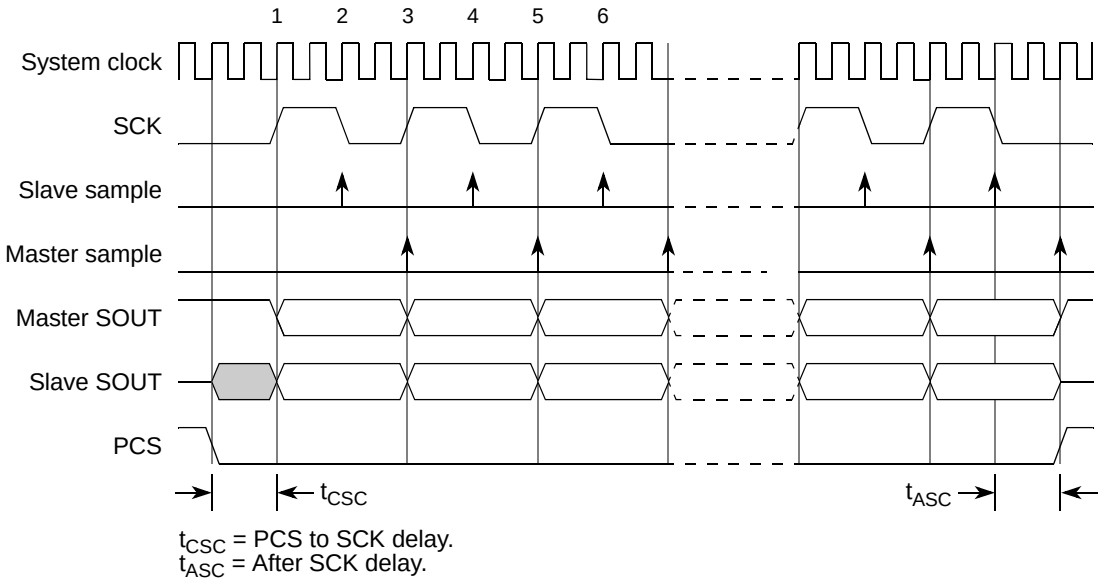


Figure 19-37. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, $f_{SCK} = f_{SYS} / 4$)

19.4.7.5 Continuous Selection Format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The continuous selection format provides the flexibility to handle both cases. The continuous selection format is enabled for the SPI configuration by setting the CONT bit in the SPI command. Continuous selection is enabled for the DSI configuration by setting the DCONT bit in the DSPI*x*_DSICR. The behavior of the PCS signals in the two configurations is identical so only the SPI configuration is described.

When the CONT bit = 0, the DSPI drives the asserted chip select signals to their idle states in between frames. The idle states of the chip select signals are selected by the PCSIS field in the DSPI*x*_MCR.

Figure 19-38 shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 0.

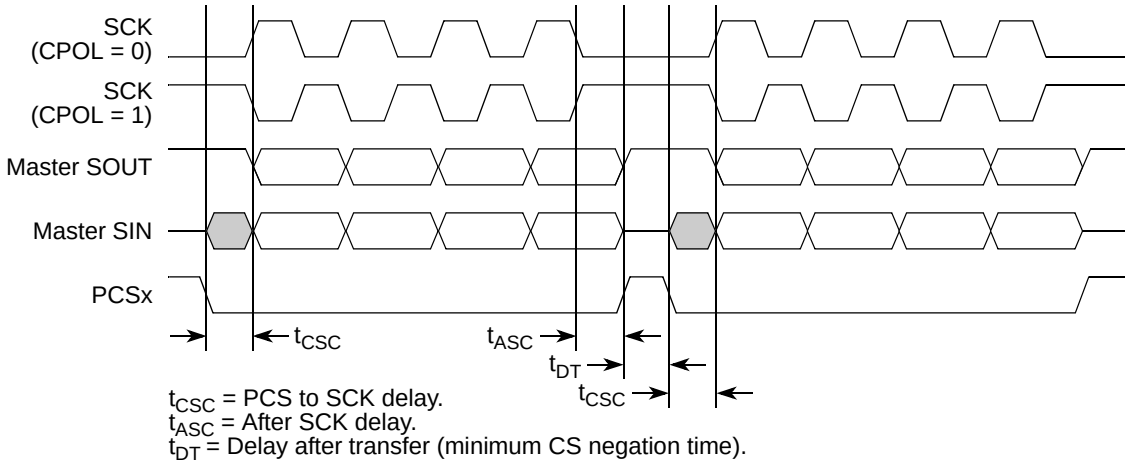


Figure 19-38. Example of Non-continuous Format (CPHA = 1, CONT = 0)

When the CONT = 1 and the PCS signal for the next transfer is the same as for the current transfer, the PCS signal remains asserted for the duration of the two transfers. The delay between transfers (t_{DT}) is not inserted between the transfers.

Figure 19-39 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 1.

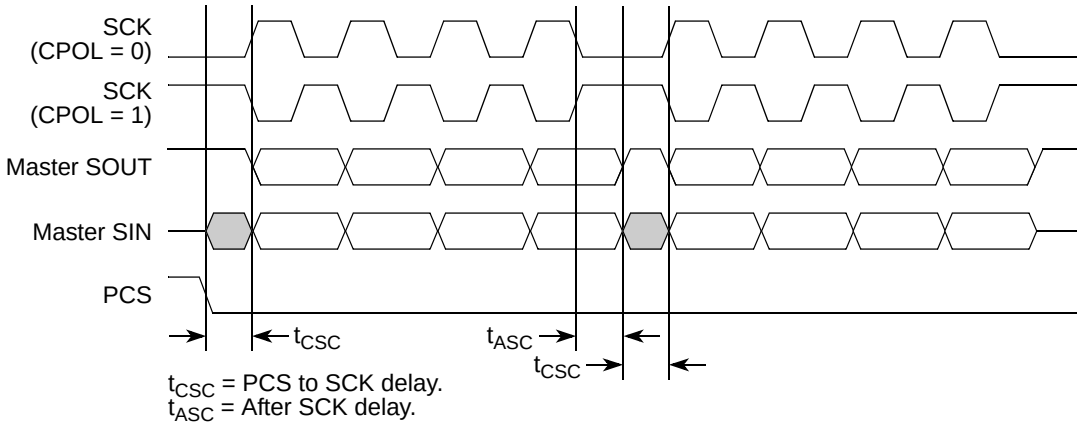


Figure 19-39. Example of Continuous Transfer (CPHA = 1, CONT = 1)

In Figure 19-39, the period length at the start of the next transfer is the sum of t_{ASC} and t_{CSC}; i.e., it does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations, t_{ASC} and t_{CSC} must be increased if a full half-clock period is required.

Switching CTARs between frames while using continuous selection can cause errors in the transfer. The PCS signal must be negated before CTAR is switched.

When the CONT bit = 1 and the PCS signals for the next transfer are different from the present transfer, the PCS signals behave as if the CONT bit was not set.

19.4.7.6 Clock Polarity Switching between DSPI Transfers

To switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the chip select pin for the next frame asserts.

See Section 19.3.2.3, “DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx_CTARn).”

In Figure 19-40, time ‘A’ shows the one clock interval. Time ‘B’ is programmable with a minimum of two system clocks.

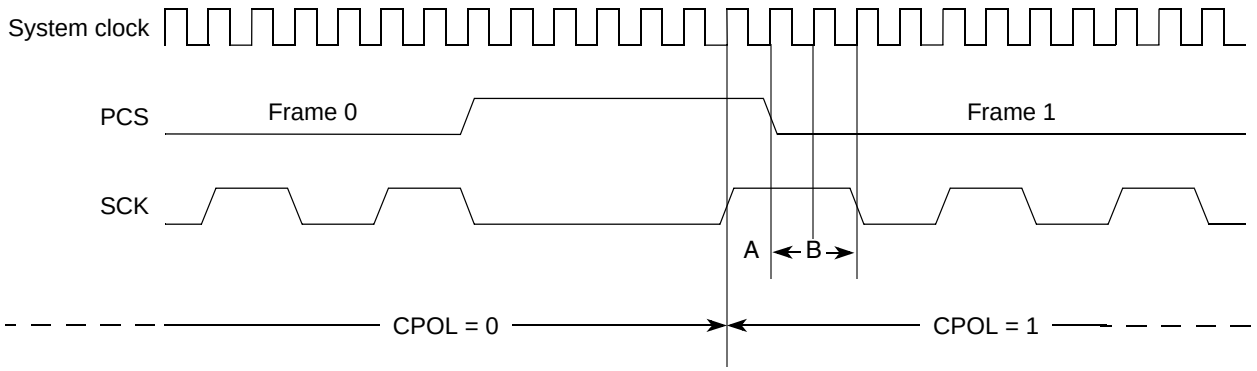


Figure 19-40. Polarity Switching between Frames

19.4.8 Continuous Serial Communications Clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting the CONT_SCKE bit in the DSPI*x*_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Setting CPHA = 0 is ignored if the CONT_SCKE bit is set. Continuous SCK is supported for modified transfer format.

Clock and transfer attributes for the continuous SCK mode are set according to the following rules:

- When the DSPI is in SPI configuration, CTAR0 is used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame is used.
- When the DSPI is in DSI configuration, the CTAR specified by the DSICTAS field is used at all times.
- When the DSPI is in CSI configuration, the CTAR selected by the DSICTAS field is used initially. At the start of an SPI frame transfer, the CTAR specified by the CTAS value for the frame is used. At the start of a DSI frame transfer, the CTAR specified by the DSICTAS field is used.
- In all configurations, the currently selected CTAR remains in use until the start of a frame with a different CTAR, or the continuous SCK mode is terminated.

The device is designed to use the same baud rate for all transfers made while using the continuous SCK. Switching clock polarity between frames while using continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into module disable mode.

Enabling continuous SCK disables the PCS to SCK delay and the After SCK delay. The delay after transfer is fixed at one SCK cycle. Figure 19-41 shows timing diagram for continuous SCK format with continuous selection disabled.

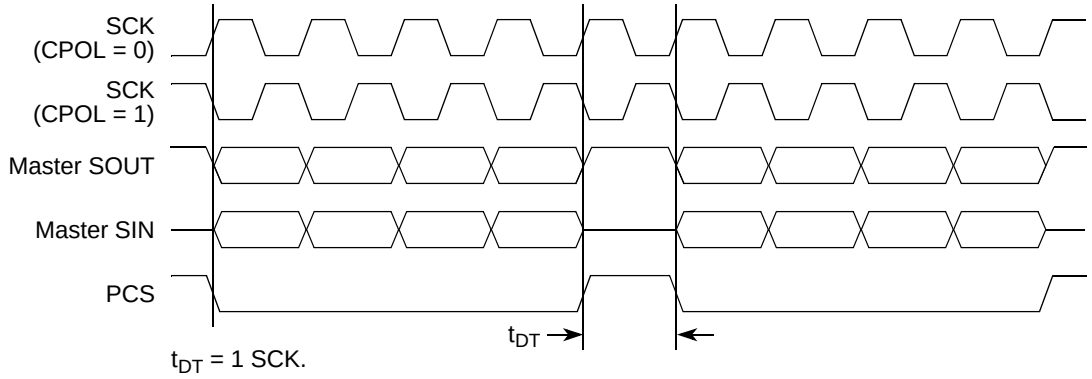


Figure 19-41. Continuous SCK Timing Diagram (CONT= 0)

If the CONT bit in the TX FIFO entry is set or the DCONT in the DSPI*x*_DSICR is set, PCS remains asserted between the transfers when the PCS signal for the next transfer is the same as for the current transfer. Figure 19-42 shows timing diagram for continuous SCK format with continuous selection enabled.

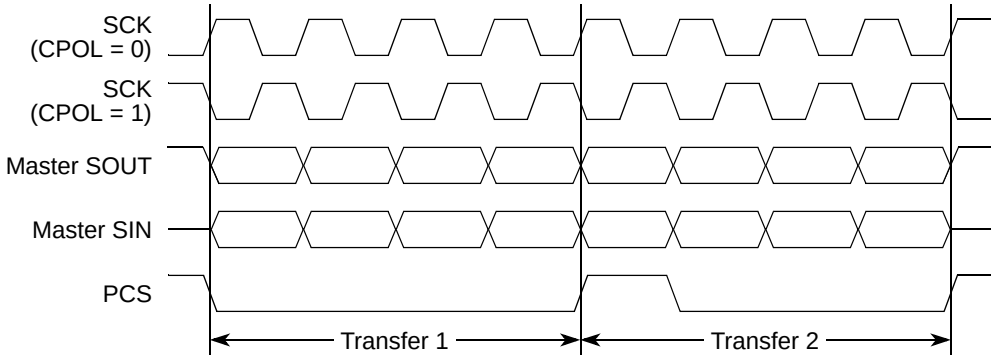


Figure 19-42. Continuous SCK Timing Diagram (CONT=1)

19.4.9 Interrupts and DMA Requests

The DSPI has five conditions that can generate interrupt requests only, and two conditions that can generate interrupt or DMA requests. Table 19-31 lists the conditions that can generate a DMA request or interrupt request.

Table 19-31. Interrupt and DMA Request Conditions

Condition	Flag	Interrupt	DMA
End of transfer queue has been reached (EOQ)	EOQF	X	
TX FIFO is not full	TFFF	X	X
Current frame transfer is complete	TCF	X	

Table 19-31. Interrupt and DMA Request Conditions (continued)

Condition	Flag	Interrupt	DMA
TX FIFO underflow has occurred	TFUF	X	
RX FIFO is not empty	RFDF	X	X
RX FIFO overflow occurred	RFOF	X	
A FIFO overrun occurred[1]	TFUF ORed with RFOF	X	

[1] The FIFO overrun condition is created by ORing the TFUF and RFOF flags together.

Each condition has a flag bit and a request enable bit. The flag bits are described in the Section 19.3.2.4, "DSPI Status Register (DSPIx_SR)" and the request enable bits are described in the Section 19.3.2.5, "DSPI DMA and Interrupt Request Select and Enable Register (DSPIx_RSER)." The TX FIFO fill flag (TFFF) and RX FIFO drain flag (RFDF) generate interrupt requests or DMA requests depending on the TFFF_DIRS and RFDF_DIRS bits in the DSPI*x*_RSER.

19.4.9.1 End-of-Queue Interrupt Request (EOQF)

The end of queue equest indicates that the end of a transmit queue is reached. The end of queue request is generated when the EOQ bit in the executing SPI command is asserted and the EOQF_RE bit in the DSPI*x*_RSER is set. See the EOQ bit description in Section 19.3.2.4, "DSPI Status Register (DSPIx_SR)." See Figure 19-34 and Figure 19-35 that illustrate when EOQF is set.

19.4.9.2 Transmit FIFO Fill Interrupt or DMA Request (TFFF)

The transmit FIFO fill request indicates that the TX FIFO is not full. The transmit FIFO fill request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the TFFF_RE bit in the DSPI*x*_RSER is set. The TFFF_DIRS bit in the DSPI*x*_RSER selects whether a DMA request or an interrupt request is generated.

19.4.9.3 Transfer Complete Interrupt Request (TCF)

The transfer complete request indicates the end of the transfer of a serial frame. The transfer complete request is generated at the end of each frame transfer when the TCF_RE bit is set in the DSPI*x*_RSER. See the TCF bit description in Section 19.3.2.4, "DSPI Status Register (DSPIx_SR)." See Figure 19-34 and Figure 19-35 that illustrate when TCF is set.

19.4.9.4 Transmit FIFO Underflow Interrupt Request (TFUF)

The transmit FIFO underflow request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode and SPI configuration is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the TFUF_RE bit in the DSPI*x*_RSER is set, an interrupt request is generated.

19.4.9.5 Receive FIFO Drain Interrupt or DMA Request (RFDF)

The receive FIFO drain request indicates that the RX FIFO is not empty. The receive FIFO drain request is generated when the number of entries in the RX FIFO is not zero, and the RFDF_RE bit in the DSPI*x*_RSER is set. The RFDF_DIRS bit in the DSPI*x*_RSER selects whether a DMA request or an interrupt request is generated.

19.4.9.6 Receive FIFO Overflow Interrupt Request (RFOF)

The receive FIFO overflow request indicates that an overflow condition in the RX FIFO has occurred. A receive FIFO overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The RFOF_RE bit in the DSPI*x*_RSER must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPI*x*_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored.

19.4.9.7 FIFO Overrun Request (TFUF) or (RFOF)

The FIFO overrun request indicates that at least one of the FIFOs in the DSPI has exceeded its capacity. The FIFO overrun request is generated by logically OR'ing together the RX FIFO overflow and TX FIFO underflow signals.

19.4.10 Power Saving Features

The DSPI supports two power-saving strategies:

- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

19.4.10.1 Module Disable Mode

Module disable mode is a module-specific mode that the DSPI can enter to save power. Host software can initiate the module disable mode by writing a 1 to the MDIS bit in the DSPI*x*_MCR. In module disable mode, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different affect when the DSPI is in the module disable mode. Reading the RX FIFO pop register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO push register does not change the state of the TX FIFO. Clearing either of the FIFOs does not have any effect in the module disable mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPI*x*_MCR does not have any affect in the module disable mode. In the module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no affect. Writing to the DSPI*x*_TCR during module disable mode does not have an effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

19.4.10.2 Slave Interface Signal Gating

The DSPI module enable signal is used to gate slave interface signals such as address, byte enable, read/write and data. This prevents toggling slave interface signals from consuming power unless the DSPI is accessed.

19.5 Initialization and Application Information

19.5.1 How to Change Queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI configuration. This section presents an example of how to change queues for the DSPI.

1. Set the EOQ bit in the command word to indicate the last entry in the queue for the DSPI after the last command word from a queue is executed.
2. Sample the command word that has the EOQ bit set at the end of the transfer. Set the EOQ flag (EOQF) in the DSPI*x*_SR is set.

   If EOQF flag is set to 1, the serial interface is disabled, preventing data transmission and reception. The DSPI is put into the STOPPED state and the TXRXS bit is negated to indicate the STOPPED state. The eDMA continues to fill the TX FIFO until one of the following conditions occur:

   — TX FIFO is full

   — Modified DMA descriptor that adds queues to the TX and RX channels is received (step 5)
3. Disable the DSPI DMA transfers by clearing the DMA channel enable bit for the DMA channel assigned to the TX FIFO and RX FIFO. This is done in the eDMA controller.
4. Ensure all received data in the RX FIFO was transferred to the memory receive queue using one of the following methods:

   — Read RXCNT in DSPI*x*_SR

   — Check RFDF in the DSPI*x*_SR after each read operation of the DSPI*x*_POPR
5. Modify the DMA descriptor for the TX and RX channels for additional queues.
6. Flush the TX FIFO and RX FIFO by writing a 1 to the CLR_TXF and the CLR_RXF bits respectively in the DSPI*x*_MCR register.
7. Clear the transfer count using one of the following methods:

   — Set the CTCNT bit in the command word of the first entry in the new queue

   — Write directly to SPI_TCNT field in DSPI*x*_TCR
8. Enable the DMA channel by setting the DMA enable request bit for the DMA channel assigned to the DSPI TX and RX FIFOs.
9. Enable serial transmission and serial reception of data by clearing the EOQF bit.

19.5.2 Baud Rate Settings

Table 19-32 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI*x*_CTARs. The values are calculated at an 82 MHz system frequency.

Table 19-32. Baud Rate Values

		Baud Rate Divider Prescaler Values (DSPI_CTAR[PBR])			
		2	**3**	**5**	**7**
Baud Rate Scaler Values (DSPI_CTAR[BR])	**2**	25.0 MHz	16.7 MHz	10.0 MHz	7.14 MHz
	4	12.5 MHz	8.33 MHz	5.00 MHz	3.57 MHz
	6	8.33 MHz	5.56 MHz	3.33 MHz	2.38 MHz
	8	6.25 MHz	4.17 MHz	2.50 MHz	1.79 MHz
	16	3.12 MHz	2.08 MHz	1.25 MHz	893 kHz
	32	1.56 MHz	1.04 MHz	625 kHz	446 kHz
	64	781 kHz	521 kHz	312 kHz	223 kHz
	128	391 kHz	260 kHz	156 kHz	112 kHz
	256	195 kHz	130 kHz	78.1 kHz	55.8 kHz
	512	97.7 kHz	65.1 kHz	39.1 kHz	27.9 kHz
	1024	48.8 kHz	32.6 kHz	19.5 kHz	14.0 kHz
	2048	24.4 kHz	16.3 kHz	9.77 kHz	6.98 kHz
	4096	12.2 kHz	8.14 kHz	4.88 kHz	3.49 kHz
	8192	6.10 kHz	4.07 kHz	2.44 kHz	1.74 kHz
	16384	3.05 kHz	2.04 kHz	1.22 kHz	872 Hz
	32768	1.53 kHz	1.02 kHz	610 Hz	436 Hz

19.5.3 Delay Settings

Table 19-33 shows the values for the delay after transfer (t_{DT}) and CS to SCK delay (t_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI*x*_CTARs. An 82 MHz system frequency was used to calculate the values in the Table 19-33.

Table 19-33. Delay Values

		Delay Prescaler Values (DSPI_CTAR[PBR])			
		1	3	5	7
Delay Scaler Values (DSPI_CTAR[DT])	2	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	32	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	64	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	128	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	256	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	512	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	1024	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	2048	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	4096	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	8192	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	16384	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	32768	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	65536	655.4 µs	2.0 ms	3.3 ms	4.6 ms

19.5.4 MPC5xx QSPI Compatibility with the DSPI

Table 19-34 shows the translation of commands written to the TX FIFO command halfword with commands written to the command RAM of the MPC5xx family QSPI. The table gives you the DSPI*x*_CTARs values to use in the control bits of the command RAM for the default cases for the combinations in the MPC5xx family. The defaults for the MPC5xx family are based on a system clock of 40 MHz.

The following delay variables generate the same delay, or as close as possible, from the DSPI 82 MHz system clock that an MPC5xx family part generates from a 40 MHz system clock. For other system clock frequencies, you can recompute the values using the information presented in Section 19.5.3, "Delay Settings."

For BITSE = 0 --> 8 bits per transfer

For DT = 0 --> 0.425 µs delay: for this value, the closest value in the DSPI is 0.480 µs

For DSCK = 0 --> 0.5 of the SCK period: for this value, the value in the DSPI is 20 ns

Table 19-34. MPC5xx QSPI Compatibility with the DSPI

MPC5xx Family Control Bits DSPI Corresponding Control Bits						Corresponding DSPI*x*_CTAR Register Configuration					
BITSE	**CTAS[0]**	**DT**	**CTAS[1]**	**DSCK**	**CTAS[2]**	**DSPI*x*_CTAR*x***	**FMSZ**	**PDT**	**DT**	**PCSSCK**	**CSSCK**
0		0		0		0	0111	10	0011	00	0000
0		0		1		1	0111	10	0011	User	User
0		1		0		2	0111	User[1]	User	00	0000
0		1		1		3	0111	User	User	User	User
1		0		0		4	User	10	0011	00	0000
1		0		1		5	User	10	0011	User	User
1		1		0		6	User	User	User	00	0000
1		1		1		7	User	User	User	User	User

1 Selected by user

19.5.5 Calculation of FIFO Pointer Addresses

You can read the TX and RX FIFO contents through the FIFO registers, and identify valid entries using a memory-mapped pointer and a memory-mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the transmit next pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the pop next pointer (POPNXTPTR).

See Section 19.4.3.4, "Using the TX FIFO Buffering Mechanism," and Section 19.4.3.5, "Using the RX FIFO Buffering Mechanism," for details on the FIFO operation. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO.

Figure 19-43 illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter.

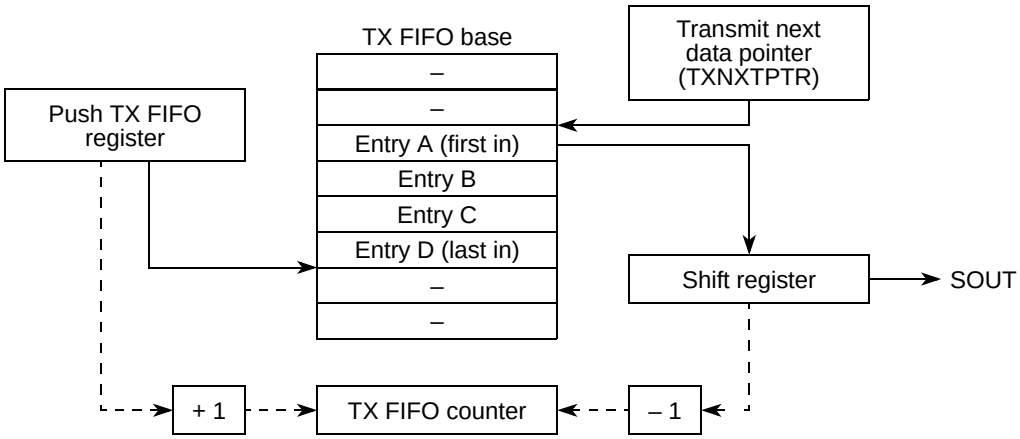


Figure 19-43. TX FIFO Pointers and Counter

19.5.5.1 Address Calculation for the First-in Entry and Last-in Entry in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

First-in entry address = TXFIFO base + 4 (TXNXTPTR)

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

Last-in entry address = TXFIFO base + (4 x [(TXCTR + TXNXTPTR - 1) modulo TXFIFO depth])

where:

TXFIFO base = base address of transmit FIFO
TXCTR = transmit FIFO counter
TXNXTPTR = transmit next pointer
TX FIFO depth = transmit FIFO depth, implementation specific

19.5.5.2 Address Calculation for the First-in Entry and Last-in Entry in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

First-in entry address = RXFIFO base + 4 x (POPNXTPTR)

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

Last-in entry address = RXFIFO base + 4 x [(RXCTR + POPNXTPTR - 1) modulo RXFIFO depth]

where:

RXFIFO base = base address of receive FIFO
RXCTR = receive FIFO counter
POPNXTPTR = pop next pointer
RX FIFO depth = receive FIFO depth, implementation specific

Chapter 20
Enhanced Serial Communication Interface (eSCI)

20.1 Introduction

This section gives an overview of the enhanced serial communication interface (eSCI) module, and presents a block diagram, its features and operating modes.

20.1.1 Block Diagram

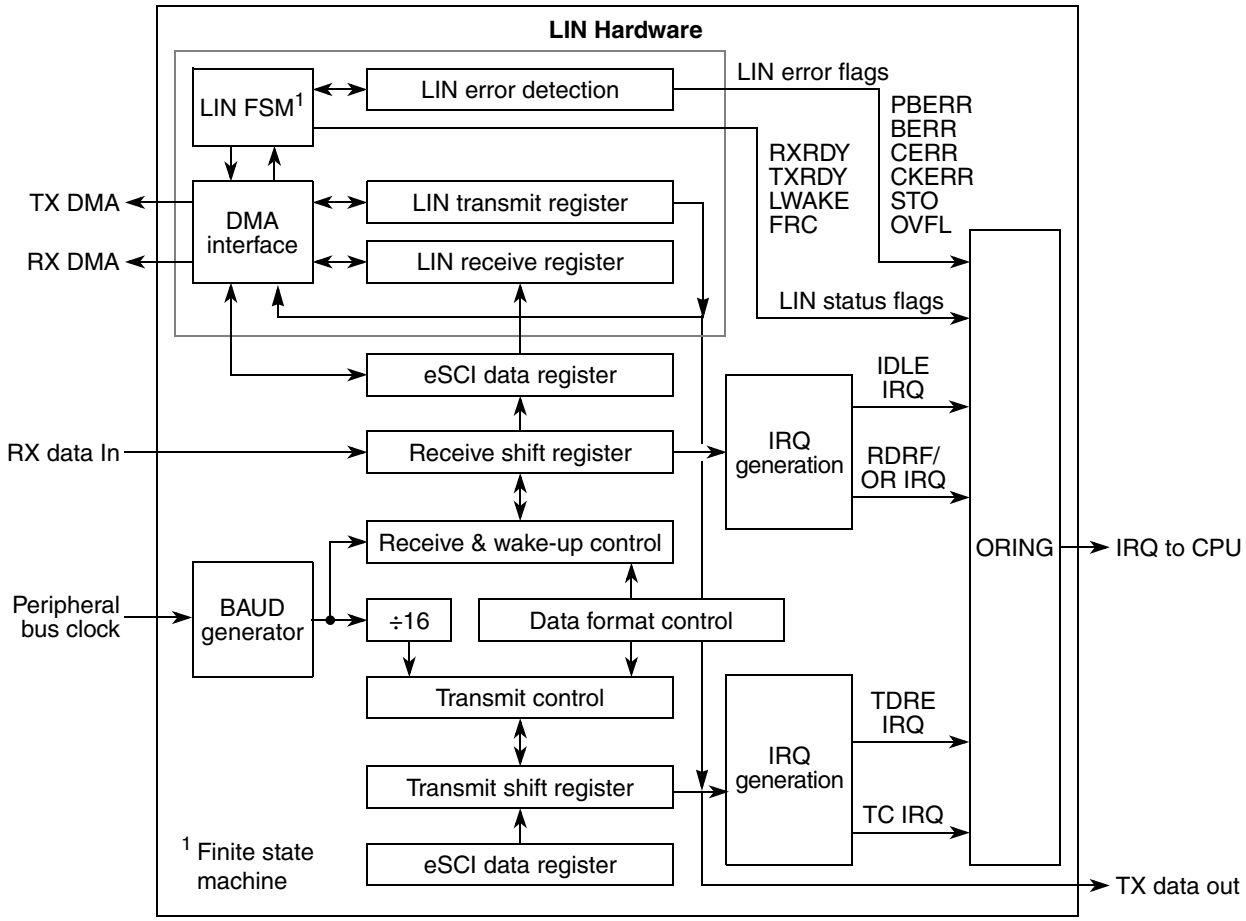


Figure 20-1. eSCI Block Diagram

20.1.2 Overview

The eSCI allows asynchronous serial communications with peripheral devices and other CPUs. The eSCI features allow it to operate as a LIN bus master, complying with the LIN 2.0 specification.

Each of the eSCI modules can be independently disabled by writing to the module disable (MDIS) bit in the module control register 2 (ESCI*x*_CR2). Disabling the module turns off the clock to the module, although the core can access some of eSCI registers using the slave bus. When the eSCI module is not used in the application, set the MDIS bit to one.

20.1.3 Features

The eSCI includes these features:

- Full-duplex operation
- Standard mark/space non-return-to-zero (NRZ) format
- Configurable 13-bit baud rate
- Programmable 8- or 9-bit data format
- LIN master node support
- Configurable CRC detection for LIN
- Separately enabled transmitter and receiver
- Programmable transmitter output parity
- Two receiver wake-up methods:
  - Idle line wake-up
  - Address mark wake-up
- Interrupt-driven operation
- Receiver framing error detection
- Hardware parity checking
- 1/16 bit-time noise detection
- Two-channel DMA interface

20.1.4 Modes of Operation

The eSCI functions the same in normal, special, and emulation modes. It has a low-power module disable mode.

20.2 External Signal Description

This section provides a description of all eSCI external to the MCU.

Each eSCI module has two I/O signals connected to the external MCU pins. These signals are summarized in Table 20-1 and described in more detail in the following sections.

Table 20-1. eSCI Signals

Signal Name[1]	I/O	Description
RXD*x*	I	eSCI receive
TXD*x*	O	eSCI transmit

1 *x* indicates eSCI module A or B

20.2.1 Detailed Signal Description

20.2.1.1 eSCI Transmit (TXDA, TXDB)

These signals transmit data out for the eSCI.

20.2.1.2 eSCI Receive Pin (RXDA, RXDB)

These signals receive data input for the eSCI.

20.3 Memory Map and Register Definition

This section provides a detailed description of all memory and registers.

20.3.1 Module Memory Map

The memory map for the eSCI module is shown in Table 20-2. The address offset is listed for each register. The total address for each register is the sum of the base address for the eSCI module (ESCI*x*_base) and the address offset for each register. There are two eSCI modules on this device:

- eSCI A base address is 0xFFFB_0000
- eSCI B base address is 0xFFFB_4000

Table 20-2. Module Memory Map

Address	Register Name	Register Description	Bits
Base 0xFFFB_0000 (A) 0xFFFB_4000 (B)	ESCI*x*_CR1	eSCI control register 1	32
Base + 0x0004	ESCI*x*_CR2	eSCI control register 2	16
Base + 0x0006	ESCI*x*_DR	eSCI data register	16
Base + 0x0008	ESCI*x*_SR	eSCI status register	32

Table 20-2. Module Memory Map (continued)

Address	Register Name	Register Description	Bits
Base + 0x000C	ESCI*x*_LCR	LIN control register	32
Base + 0x0010	ESCI*x*_LTR	LIN transmit register	32
Base + 0x0014	ESCI*x*_LRR	LIN receive register	32
Base + 0x0018	ESCI*x*_LPR	LIN cyclic redundancy check polynomial register	32

20.3.2 Register Descriptions

This section contains the register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Details of register bit and field functions follow the register diagrams, in bit order.

20.3.2.1 eSCI Control Register 1 (ESCI*x*_CR1)

Address: Base + 0x0000 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	SBR 0	SBR 1	SBR 2	SBR 3	SBR 4	SBR 5	SBR 6	SBR 7	SBR 8	SBR 9	SBR 10	SBR 11	SBR 12
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LOOPS	0	RSRC	M	WAKE	ILT	PE	PT	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-2. eSCI Control Register 1 (ESCI*x*_CR1)

Table 20-3. ESCI*x*_CR1 Field Descriptions

Field	Description
0–2	Reserved
3–15 SBR*n*	SCI baud rate. Used by the counter to determine the baud rate of the eSCI. The formula for calculating the baud rate is: $$\text{SCI baud rate} = \frac{\text{eSCI system clock}}{16 \times \text{BR}}$$ where BR is the content of the eSCI control register 1 (ESCI*x*_CR1), bits SBR0–SBR12. SBR0–SBR12 can contain a value from 1 to 8191. See the ESCIx_LCR[WU] bit description on page 20-12.

Table 20-3. ESCI*x*_CR1 Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>16
LOOPS</td><td>Loop select. Enables loop operation. In loop operation, the RXD pin is disconnected from the eSCI and the transmitter output is internally connected to the receiver input. Both the transmitter and the receiver must be enabled to use the loop function.
0 Normal operation enabled, loop operation disabled
1 Loop operation enabled
Note: The receiver input is determined by the RSRC bit.</td></tr>
<tr><td>17</td><td>Reserved</td></tr>
<tr><td>18
RSRC</td><td>Receiver source. When LOOPS = 1, the RSRC bit determines the source for the receiver shift register input.
0 Receiver input internally connected to transmitter output
1 Receiver input connected externally to transmitter
The table below shows how LOOPS and RSRC determine the loop function of the eSCI.
<table>
<tr><th>LOOPS</th><th>RSRC</th><th>Function</th></tr>
<tr><td>0</td><td>–</td><td>Normal operation</td></tr>
<tr><td>1</td><td>0</td><td>Loop mode with RXD input internally connected to TXD output</td></tr>
<tr><td>1</td><td>1</td><td>Single-wire mode with RXD input connected to TXD</td></tr>
</table>
</td></tr>
<tr><td>19
M</td><td>Data format mode. Determines whether data characters are 8 or 9 bits long.
0 1 start bit, 8 data bits, 1 stop bit
1 1 start bit, 9 data bits, 1 stop bit</td></tr>
<tr><td>20
WAKE</td><td>Wake-up condition. Determines which condition wakes up the eSCI: a logic 1 (address mark) in the most significant bit (MSB) position of a received data character or an idle condition on the RXD.
0 Idle line wake-up
1 Address mark wake-up
Note: This is not a wake-up from a power-save mode; this function applies to the receiver standby mode only.</td></tr>
<tr><td>21
ILT</td><td>Idle line type. Determines when the receiver starts counting logic 1s as idle character bits. The counting begins either after the start bit or after the stop bit. If the count begins after the start bit, then a string of logic 1s preceding the stop bit can cause false recognition of an idle character. Beginning the count after the stop bit avoids false idle character recognition, but requires correctly synchronized transmissions.
0 Idle character bit count begins after start bit
1 Idle character bit count begins after stop bit</td></tr>
<tr><td>22
PE</td><td>Parity enable. Enables the parity function. When enabled, the parity function inserts a parity bit in the most significant bit position of the transmitted word. During reception, the received parity bit is verified in the most significant bit position. The received parity bit is not masked out.
0 Parity function disabled
1 Parity function enabled</td></tr>
<tr><td>23
PT</td><td>Parity type. Determines whether the eSCI generates and checks for even parity or odd parity. With even parity, an even number of 1s clears the parity bit and an odd number of 1s sets the parity bit. With odd parity, an odd number of 1s clears the parity bit and an even number of 1s sets the parity bit.
0 Even parity
1 Odd parity</td></tr>
<tr><td>24
TIE</td><td>Transmitter interrupt enable. Enables the transmit data register empty flag ESCIx_SR[TDRE] to generate interrupt requests. The interrupt is suppressed in TX DMA mode.
0 TDRE interrupt requests disabled
1 TDRE interrupt requests enabled</td></tr>
</table>

Table 20-3. ESCI*x*_CR1 Field Descriptions (continued)

Field	Description
25 TCIE	Transmission complete interrupt enable. Enables the transmission complete flag ESCIx_SR[TC] to generate interrupt requests. The interrupt is suppressed in TX DMA mode. 0 TC interrupt requests disabled 1 TC interrupt requests enabled
26 RIE	Receiver full interrupt enable. Enables the receive data register full flag ESCIx_SR[RDRF] and the overrun flag ESCIx_SR[OR] to generate interrupt requests. The interrupt is suppressed in RX DMA mode. 0 RDRF and OR interrupt requests disabled 1 RDRF and OR interrupt requests enabled
27 ILIE	Idle line interrupt enable. Enables the idle line flag ESCIx_SR[IDLE] to generate interrupt requests. 0 IDLE interrupt requests disabled 1 IDLE interrupt requests enabled
28 TE	Transmitter enable. Enables the eSCI transmitter and configures the TXD pin as being controlled by the eSCI. The TE bit can be used to queue an idle preamble. 0 Transmitter disabled 1 Transmitter enabled
29 RE	Receiver enable. Enables the eSCI receiver. 0 Receiver disabled 1 Receiver enabled
30 RWU	Receiver wake-up. Standby state. 0 Normal operation. 1 RWU enables the wake-up function and inhibits further receiver interrupt requests. Normally, hardware wakes the receiver by automatically clearing RWU.
31 SBK	Send break. Toggling SBK sends one break character (See the description of ESCI*x*_CR2[BRK13] for break character length). Toggling implies clearing the SBK bit before the break character has finished transmitting. As long as SBK is set, the transmitter continues to send complete break characters. 0 No break characters 1 Transmit break characters

NOTES

After a reset, the baud rate generator is disabled until the TE bit or the RE bit is initialized (set for the first time).

The baud rate generator is disabled when SBR0–SBR12 = 0x0000.

The baud rate is usually written using a single write. If using 8-bit writes, writing to ESCI*x*_CR1[0–7] has no effect until ESCI*x*_CR1[8–15] is written, since ESCI*x*_CR1[0–7] is temporarily buffered until ESCI*x*_CR1[8–15] is written.

When parity is enabled, the RX Data parity bit is in the data register.

20.3.2.2 eSCI Control Register 2 (ESCI*x*_CR2)

NOTE

DMA requests are negated when in module disable mode.

Address: Base + 0x0004 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FBR	BSTP	IEB ERR	RX DMA	TX DMA	BRK 13	0	BESM 13	SB STP	0	0	ORIE	NFIE	FEIE	PFIE
W																
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-3. eSCI Control Register 2 (ESCIx_CR2)

Table 20-4. ESCI*x*_CR2 Field Description

Field	Description
0 MDIS	Module disable. By default the module is enabled, but can be disabled by writing a 1 to this bit. DMA requests are negated if the device is in module disable mode. 0 Module enabled 1 Module disabled
1 FBR	Fast bit error detection. Handles bit error detection on a per bit basis. If this is not enabled, bit errors are detected on a byte basis.
2 BSTP	Bit error/physical bus error stop. Causes DMA TX requests to be suppressed, as long as the bit error and physical bus error flags are not cleared. This stops further DMA writes, which would otherwise cause data bytes to be interpreted as LIN header information.
3 IEBERR	Enable bit error interrupt. Generates an interrupt, when a LIN bit error is detected. For a list of interrupt enables and flags, See Table 20-21.
4 RXDMA	Activate RX DMA channel. If this bit is enabled and the eSCI has received data, it raises a DMA RX request.
5 TXDMA	Activate TX DMA channel. Whenever the eSCI is able to transmit data, it raises a DMA TX request.
6 BRK13	Break transmit character length. Determines whether the transmit break character is either 10 or 11, or 13 or 14 bits long. The detection of a framing error is not affected by this bit. **Break Length:** BRK13 = 0, ESCI*x*_CR1[M] = 0: 10 BRK13 = 0, ESCI*x*_CR1[M] = 1: 11 BRK13 = 1, ESCI*x*_CR1[M] = 0: 13 BRK13 = 1, ESCI*x*_CR1[M] = 1: 14 0 Break Character is 10 or 11 bits long 1 Break character is 13 or 14 bits long **Note:** LIN 2.0 now requires that a break character is always 13 bits long, always set this bit to 1. The eSCI works with BRK13=0, but it violates LIN 2.0.
7	Reserved. This bit is readable/writable, but has no effect on the operation of the eSCI module.

Table 20-4. ESCI*x*_CR2 Field Description (continued)

Field	Description
8 BESM13	Bit error sample mode, bit 13. Determines when to sample the incoming bit to detect a bit error. This only applies when FBR is set. 0 Sample at RT clock 9 1 Sample at RT clock 13 (See Section 20.4.5.3, “Data Sampling”)
9 SBSTP	SCI bit error stop. Stops the SCI when a bit error is asserted. This allows to stop driving the LIN bus quickly after a bit error has been detected. 0 Byte is completely transmitted 1 Byte is partially transmitted
10–11	Reserved
12 ORIE	Overrun error interrupt enable. Generates an interrupt, when a frame error is detected. For a list of interrupt enables and flags, See Table 20-21.
13 NFIE	Noise flag interrupt enable. Generates an interrupt, when noise flag is set. For a list of interrupt enables and flags, See Table 20-21.
14 FEIE	Frame error interrupt enable. Generates an interrupt, when a frame error is detected. For a list of interrupt enables and flags, See Table 20-21.
15 PFIE	Parity flag interrupt enable. Generates an interrupt, when parity flag is set. For a list of interrupt enables and flags, See Table 20-21.

20.3.2.3 eSCI Data Register (ESCI*x*_DR)

Address: Base + 0x0006 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R8	T8	0	0	0	0	0	0	R7	R6	R5	R4	R3	R2	R1	R0
W									T7	T6	T5	T4	T3	T2	T1	T0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-4. eSCI Data Register (ESCI*x*_DR)

Table 20-5. ESCI*x*_DR Field Description

Field	Description
0 R8	Received bit 8. R8 is the ninth data bit received when the eSCI is configured for 9-bit data format (M = 1).
1 T8	Transmit bit 8. T8 is the ninth data bit transmitted when the eSCI is configured for 9-bit data format (M = 1). **Note:** If the value of T8 is the same as in the previous transmission, T8 does not have to be rewritten.The same value is transmitted until T8 is rewritten.
2–7	Reserved
8–15 R7–R0 T7–T0	Received bits/transmit bits 7–0 for 9-bit or 8-bit formats. Bits 7–0 from SCI communication can be read from ESCI*x*_DR[8–15] (provided that SCI communication was successful). Writing to ESCI*x*_DR [8–15] provides bits 7–0 for SCI transmission.

NOTES

In 8-bit data format, only bits 8–15 of ESCI*x*_DR need to be accessed.

When transmitting in 9-bit data format and using 8-bit write instructions, write first to ESCI*x*_DR[0–7], then ESCI*x*_DR[8–15]. For 9-bit transmissions, a single write can also be used.

Do not use ESCI*x*_DR in LIN mode, writes to this register are blocked in LIN mode.

Even if parity generation/checking is enabled via ESCIx_CR[PE], the parity bit is not masked out.

20.3.2.4 eSCI Status Register (ESCI*x*_SR)

The ESCI*x*_SR indicates the current status. The status flags can be polled, and some can also be used to generate interrupts. All bits in ESCI*x*_SR except for RAF are cleared by writing 1 to them.

Address: Base + 0x0008 Access: R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	0	0	0	BERR	0	0	0	RAF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c				w1c				
Reset	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RX RDY	TX RDY	LWAKE	STO	PB ERR	CERR	CK ERR	FRC	0	0	0	0	0	0	0	OVFL
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c								w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-5. eSCI Status Register (ESCI*x*_SR)

Table 20-6. ESCI*x*_SR Field Descriptions

Field	Description
0 TDRE	Transmit data register empty flag. TDRE is set when the transmit shift register receives a byte from the eSCI data register. When TDRE is 1, the data register (ESCI*x*_DR) is empty and can receive a new value to transmit. Clear TDRE by writing 1 to it. 0 eSCI has not transferred data to the transmit shift register since the last time software cleared TDRE 1 Byte transferred to transmit shift register; transmit data register empty
1 TC	Transmit complete flag. TC is set low when there is a transmission in progress or when a preamble or break character is loaded. TC is set high when the TDRE flag is set and no data, preamble, or break character is being transmitted. When TC is set, the TXD out signal becomes idle (logic 1). After the device is switched on (by clearing the MDIS bit, See Section 20.3.2.2, "eSCI Control Register 2 (ESCIx_CR2)," a preamble is transmitted; if no byte is written to the SCI data register then the completion of the preamble can be monitored using the TC flag. Clear TC by writing 1 to it. 0 Transmission in progress 1 No transmission in progress. Indicates that TXD out is idle.
2 RDRF	Receive data register full flag. RDRF is set when the data in the receive shift register transfers to the eSCI data register. Clear RDRF by writing 1 to it. 0 eSCI has not transferred data to the receive data register since last time software cleared RDRF 1 Received data available in eSCI data register
3 IDLE	Idle line flag. IDLE is set when 10 consecutive logic 1s (if M = 0) or 11 consecutive logic 1s (if M = 1) appear on the receiver input. After the IDLE flag is cleared, a valid frame must again set the RDRF flag before an idle condition can set the IDLE flag. Clear IDLE by writing 1 to it. 0 Receiver input is either active now or has never become active since the IDLE flag was last cleared 1 Receiver input has become idle **Note:** When the receiver wake-up bit (RWU) is set, an idle line condition does not set the IDLE flag.
4 OR	Overrun flag. OR is set when software fails to read the eSCI data register before the receive shift register receives the next frame. The OR bit is set immediately after the stop bit has been completely received for the second frame. The data in the shift register is lost, but the data already in the eSCI data registers is not affected. Clear OR by writing 1 to it. 0 No overrun 1 Overrun
5 NF	Noise flag. NF is set when the eSCI detects noise on the receiver input. NF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear NF by writing 1 to it. 0 No noise 1 Noise
6 FE	Framing error flag. FE is set when a logic 0 is accepted as the stop bit. FE bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear FE by writing 1 to it. 0 No framing error 1 Framing error
7 PF	Parity error flag. PF is set when the parity enable bit, PE, is set and the parity of the received data does not match its parity bit. Clear PE by writing 1 to it. 0 No parity error 1 Parity error
8–10	Reserved, Must be 0.

Table 20-6. ESCI*x*_SR Field Descriptions (continued)

Field	Description
11 BERR	Bit error. Indicates a bit on the bus did not match the transmitted bit. If FBR = 0, checking happens after a complete byte has been transmitted and received again. If FBR = 1, checking happens bit by bit. This bit is only used for LIN mode. BERR is also set if an unrequested byte is received (i.e. a byte that is not part of an RX frame) that is not recognized as a wake-up flag. (Because the data on the RX line does not match the idle state that was assigned to the TX line.) Clear BERR by writing 1 to it. A bit error causes the LIN finite state machine (FSM) to reset unless ESCI*x*_LCR[LDBG] is set. 0 No bit error 1 Bit error
12–14	Reserved
15 RAF	Receiver active flag. RAF is set when the receiver detects a logic 0 during the RT1 time period of the start bit search. RAF is cleared when the receiver detects an idle character. 0 No reception in progress. 1 Reception in progress.
16 RXRDY	The eSCI has received LIN data. This bit is set when the ESCI*x*_LCR receives a byte. Write a one to RXRDY to clear it to 0. 0 No receive data ready 1 Receive data ready
17 TXRDY	The LIN FSM can accept another write to ESCI*x*_LTR. This bit is set when the ESCI*x*_LTR register becomes free. Write a one to TXRDY to clear it to 0. 0 ESCIx_LTR register is not free 1 ESCIx_LTR register is free
18 LWAKE	Received LIN wake-up signal. A LIN slave has sent a wake-up signal on the bus. When this signal is detected, the LIN FSM resets. If the setup of a frame had already started, it therefore must be repeated. LWAKE is set if ESCI receives a LIN 2.0 wake-up signal (in which the baud rate is lower than 32K baud). See the WU bit. 0 LIN2.0 wakeup signal not received 1 LIN2.0 wakeup signal received
19 STO	Slave time out. Represents a NO_RESPONSE_ERROR. This is set if a slave does not complete a frame within the specified maximum frame length. For LIN 1.3 the following formula is used: $$TFRAME\_MAX = (10 \times NDATA + 44) \times 1.4$$ 0 No time out detected 1 Slave did not complete a frame within the maximum frame length specified
20 PBERR	Physical bus error. No valid message can be generated on the bus. This is set if, after the start of a byte transmission, the input remains unchanged for 31 cycles. This resets the LIN FSM. 0 No error 1 Physical bus error
21 CERR	CRC error. The CRC pattern received with an extended frame was not correct. 0 No error 1 CRC error
22 CKERR	Checksum error. Checksum error on a received frame. 0 No error 1 Checksum error
23 FRC	Frame complete. LIN frame completely transmitted. All LIN data bytes received. 0 Frame not complete 1 Frame complete

Table 20-6. ESCI*x*_SR Field Descriptions (continued)

Field	Description
24–30	Reserved
31 OVFL	ESCI*x*_LRR overflow. The LIN receive register has not been read before a new data byte, CRC, or checksum byte has been received from the LIN bus. Set when the condition is detected, and cleared by writing 1 to it. 0 No overflow 1 Overflow detected

20.3.2.5 LIN Control Register (ESCI*x*_LCR)

ESCI*x*_LCR can be written only when there are no ongoing transmissions.

Address: Base + 0x000C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LRES	0	WUD 0	WUD 1	LDBG	DSF	PRTY	LIN	RXIE	TXIE	WUIE	STIE	PBIE	CIE	CKIE	FCIE
W		WU														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	OFIE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-6. LIN Control Register (ESCI*x*_LCR)

Table 20-7. ESCI*x*_LCR Field Descriptions

Field	Description
0 LRES	LIN resynchronize. Causes the LIN protocol engine to return to start state. This happens automatically after bit errors, but software can force a return to start state manually via this bit. The bit first must be set then cleared, so that the protocol engine is operational again.
1 WU	LIN bus wake-up. Generates a wake-up signal on the LIN bus. This must be set before a transmission, if the bus is in sleep mode. This bit auto-clears, so a read from this bit always returns 0. For LIN 2.0, generating a valid wake-up character requires programming the SCI baud rate to a range of 32K baud down to 1.6K baud.

Table 20-7. ESCI*x*_LCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2–3
WUD
[0:1]</td><td>Wake-up delimiter time. Determines how long the LIN engine waits after generating a wake-up signal, before starting a new frame. The eSCI does not set ESCIx_SR[TXRDY] before this time expires. In addition to this delimiter time, the CPU and the eSCI require some setup time to start a new transmission. Typically there is an additional bit time delay. The following table shows how WUD0 and WUD1 affect the delimiter time.
<table>
<tr><th>WUD0</th><th>WUD1</th><th>Bit Times</th></tr>
<tr><td>0</td><td>0</td><td>4</td></tr>
<tr><td>0</td><td>1</td><td>8</td></tr>
<tr><td>1</td><td>0</td><td>32</td></tr>
<tr><td>1</td><td>1</td><td>64</td></tr>
</table>
</td></tr>
<tr><td>4
LDBG</td><td>LIN debug mode. Prevents the LIN FSM from automatically resetting, after an exception (bit error, physical bus error, wake-up flag) has been received. This is for debug purposes only.</td></tr>
<tr><td>5
DSF</td><td>Double stop flags. When a bit error is detected, an additional stop flag is added to the byte in which the error occurred.</td></tr>
<tr><td>6
PRTY</td><td>Activating parity generation. Generate the two parity bits in the LIN header.</td></tr>
<tr><td>7
LIN</td><td>LIN mode. Switch device into LIN mode.
0 LIN disabled
1 LIN enabled
When LIN is enabled, even if parity generation/checking is enabled via ESCIx_CR[PE], the parity bit is not masked out.</td></tr>
<tr><td>8
RXIE</td><td>LIN RXREG ready interrupt enable. Generates an Interrupt when new data is available in the LIN RXREG. For a list of interrupt enables and flags, See Table 20-21.</td></tr>
<tr><td>9
TXIE</td><td>LIN TXREG ready interrupt enable. Generates an Interrupt when new data can be written to the LIN TXREG. For a list of interrupt enables and flags, See Table 20-21.</td></tr>
<tr><td>10
WUIE</td><td>RX wake-up interrupt enable. Generates an Interrupt when a wake-up flag from a LIN slave has been received. For a list of interrupt enables and flags, See Table 20-21.</td></tr>
<tr><td>11
STIE</td><td>Slave timeout error interrupt enable. Generates an Interrupt when the slave response is too slow. For a list of interrupt enables and flags, See Table 20-21.</td></tr>
<tr><td>12
PBIE</td><td>Physical bus error interrupt enable. Generates an Interrupt when no valid message can be generated on the bus. For a list of interrupt enables and flags, See Table 20-21.</td></tr>
<tr><td>13
CIE</td><td>CRC error interrupt enable. Generates an Interrupt when a CRC error on a received extended frame is detected. For a list of interrupt enables and flags, See Table 20-21.</td></tr>
<tr><td>14
CKIE</td><td>Checksum error interrupt enable. Generates an Interrupt on a detected checksum error. For a list of interrupt enables and flags, See Table 20-21.</td></tr>
<tr><td>15
FCIE</td><td>Frame complete interrupt enable. Generates an Interrupt after complete transmission of a TX frame, or after the last byte of an RX frame is received. (The complete frame includes all header, data, CRC and checksum bytes as applicable.) For a list of interrupt enables and flags, See Table 20-21.</td></tr>
<tr><td>16–22</td><td>Reserved</td></tr>
</table>

Table 20-7. ESCI*x*_LCR Field Descriptions (continued)

Field	Description
23 OFIE	Overflow interrupt enable. Generates an Interrupt when a data byte in the ESCIx_LRR has not been read before the next data byte is received. For a list of interrupt enables and flags, See Table 20-21.
24–31	Reserved

20.3.2.6 LIN Transmit Register (ESCI*x*_LTR)

ESCI*x*_LTR can be written to only when TXRDY is set. The first byte written to the register selects the transmit address, the second byte determines the frame length, the third and fourth byte set various frame options and determine the timeout counter. Header parity is automatically generated if the ESCI*x*_LCR[PRTY] bit is set. For TX frames, the fourth byte (bits T7–T0) is skipped, since the timeout function does not apply. All following bytes are data bytes for the frame. CRC and checksum bytes are automatically appended when the appropriate options are selected.

When a bit error is detected, an interrupt is set and the transmission aborted. The register can only be written again after the interrupt is cleared. Afterwards a new frame starts, and the first byte needs to contain a header again.

Additionally it is possible to flush the ESCI*x*_LTR by setting the ESCI*x*_LCR[LRES] bit.

NOTE

Not all values written to the ESCI*x*_LTR generate valid LIN frames. The values are determined according to the LIN specification.

Address: Base + 0x0010 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R									0	0	0	0	0	0	0	0
W	P1/ L7/ HDCHK/ T7/ D7	P0/ L6/ CSUM/ T6/ D6	ID5/ L5/ CRC/ T5/ D5	ID4/ L4/ TX/ T4/ D4	ID3/ L3/ T11/ T3/ D3	ID2/ L2/ T10/ T2/ D2	ID1/ L1/ T9/ T1/ D1	ID0/ L0/ T8/ T0/ D0								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-7. LIN Transmit Register (ESCI*x*_LTR)

Address: eSCI *x* Base + 0x0010 Access: W/O

	0	1	2	3	4	5	6	7
R								
1st Write (Table 20-8) W	P[1:0]		ID[5:0]					
2nd Write (Table 20-9) W	L[7:0]							
3rd Write (Table 20-10) W	HDCHK	CSUM	CRC	TX ($\overline{RX}$)	T[11:8]			
4th Write (Table 20-11) W	T[7:0]							
5th Write (Table 20-12) W	D[7:0]							
Reset	0	0	0	0	0	0	0	0

Figure 20-8. LIN Transmit Register (ESCIx_LTR) Alternate Diagram

Table 20-8. ESCI*x*_LTR First Byte Field Description

Field	Description
0–1 P*n*	Parity bit *n*. When parity generation is enabled (ESCI*x*_LCR[PRTY] = 1), the parity bits are generated automatically. Otherwise they must be provided in this field.
2–7 ID*n*[1]	Header bit *n*. The LIN address, for LIN 1.x standard frames the length bits must be set appropriately so the extended frames are recognized by their specific patterns. See the Table 20-9.
8–31	Reserved

ID5	ID4	data bytes
0	0	2
0	1	2
1	0	4
1	1	8

[1] The values 3C, 3D, 3E and 3F of the ID-field (ID0-5) indicate command and extended frames. See LIN Specification Package Revision 2.0.

Table 20-9. ESCI*x*_LTR Second Byte Field Description

Field	Description
0–7 L*n*	Length bit *n*. Defines the length of the frame (0 to 255 data bytes). This information is needed by the LIN state machine to insert the checksum or CRC pattern as required. LIN 1.x slaves only accepts frames with 2, 4, or 8 data bytes.
8–31	Reserved

Table 20-10. ESCI*x*_LTR Third Byte Field Descriptions

Field	Description
0 HDCHK	Header checksum enable. Include the header fields into the mod 256 checksum of the standard frames.
1 CSUM	Checksum enable. Append a checksum byte to the end of a TX frame. Verify the checksum byte of an RX frame.
2 CRC	CRC enable. Append two CRC bytes to the end of a TX frame. Verify the two CRC bytes of an RX frame are correct. If both CSUM and CRC bits are set, the LIN FSM first appends the CRC bytes, then the checksum byte, and are processed in this order. If HDCHK is set, the CRC calculation includes the header and data bytes, otherwise, the CRC is performed on the data bytes only. CRC bytes are not part of the LIN standard; they are normal data bytes and belong to a higher-level protocol.
3 TX	Transmit direction. Indicates that the eSCI transmits a frame to a slave. Otherwise, an RX frame is assumed, and the eSCI only transmits the header. The data bytes are received from the slave. 0 RX frame 1 TX frame
4–7 T*n*	Timeout bit *n*. Sets the counter to determine a NO_RESPONSE_ERROR, if the frame is a read access to a LIN slave. Following LIN standard rev 1.3, the value $(10 \times N_{DATA} + 45) \times 1.4$ is recommended. For transmissions, this counter has to be set to 0. The timeout bits 7–0 are not written on a TX frame. For TX frames, the fourth byte written to the LIN transmit register (ESCI*x*_LTR) is the first data byte, for RX frames it contains timeout bits 7–0.The time is specified in multiples of bit times. The timeout period starts with the transmission of the LIN break character.
8–31	Reserved

Table 20-11. ESCI*x*_LTR Rx Frame Fourth Byte Field Description

Field	Description
0–7 T*n*	Timeout bit *n*. Sets the counter to determine a NO_RESPONSE_ERROR, if the frame is a read access to a LIN slave. Follow the LIN standard rev 1.3, the value $(10 \times N_{DATA} + 45) \times 1.4$. For transmissions, this counter must be set to 0. The timeout bits 7–0 are not written on a TX frame. For TX frames, the fourth byte written to the LIN transmit register (ESCI*x*_LTR) is the first data byte. For RX frames, it contains timeout bits 7–0.The time is specified in multiples of bit times. The timeout period starts with the transmission of the LIN break character.
8–31	Reserved

Table 20-12. ESCI*x*_LTR Tx Frame Fourth + Byte—Rx Frame Fifth + Byte Field Description

Field	Description
0–7 D*n*	Data bits for transmission.
8–31	Reserved

20.3.2.7 LIN Receive Register (ESCI*x*_LRR)

ESCI*x*_LRR can be ready only when ESCI*x*_SR[RXRDY] is set.

NOTE

Application software must ensure that ESCI*x*_LRR be read before new data or checksum bytes or CRCs are received from the LIN bus.

Address: Base + 0x0014 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-9. LIN Receive Register (ESCI*x*_LRR)

Table 20-13. ESCI*x*_LRR Field Descriptions

Field	Description
0–7 D*n*	Data bit *n*. Provides received data bytes from RX frames. Data is only valid when the ESCI*x*_SR[RXRDY] flag is set. CRC and checksum information are not available in the ESCI*x*_LRR unless they are treated as data. It is possible to treat CRC and checksum bytes as data by deactivating the CSUM respectively CRC control bits in the ESCI*x*_LTR; however, then CRC and CSUM checking has to be performed by software. Data bytes must be read from the ESCI*x*_LRR (by CPU or DMA) before any new bytes (including CRC or checksum) are received from the LIN bus otherwise the data byte is lost and OVFL is set. **Note:** The data must be collected and the LIN frame finished (including CRC and checksum if applicable) before a wake-up character can be sent.
8–31	Reserved

20.3.2.8 LIN CRC Polynomial Register (ESCI*x*_LPR)

ESCI*x*_LPR*n* can be written when there are no ongoing transmissions.

Address: Base + 0x0018 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0
Reset	1	1	0	0	0	1	0	1	1	0	0	1	1	0	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-10. LIN CRC Polynomial Register (ESCI*x*_LPR)

Table 20-14. ESCI*x*_LPR Field Description

Field	Description
0–15 P*n*	Polynomial bit x^n. Bits P15–P0 are used to define the LIN polynomial - standard is $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$ (the polynomial used for the CAN protocol).
16–31	Reserved

20.4 Functional Description

20.4.1 Overview

This section provides a complete functional description of the eSCI module, detailing the operation of the design from the end user perspective in a number of subsections.

Figure 20-11 shows the structure of the eSCI module. The eSCI allows full duplex, asynchronous, NRZ serial communication between the CPU and remote devices, including other CPUs. The eSCI transmitter and receiver operate independently, although they use the same baud rate generator. The CPU monitors the status of the eSCI, writes the data to be transmitted, and processes received data.

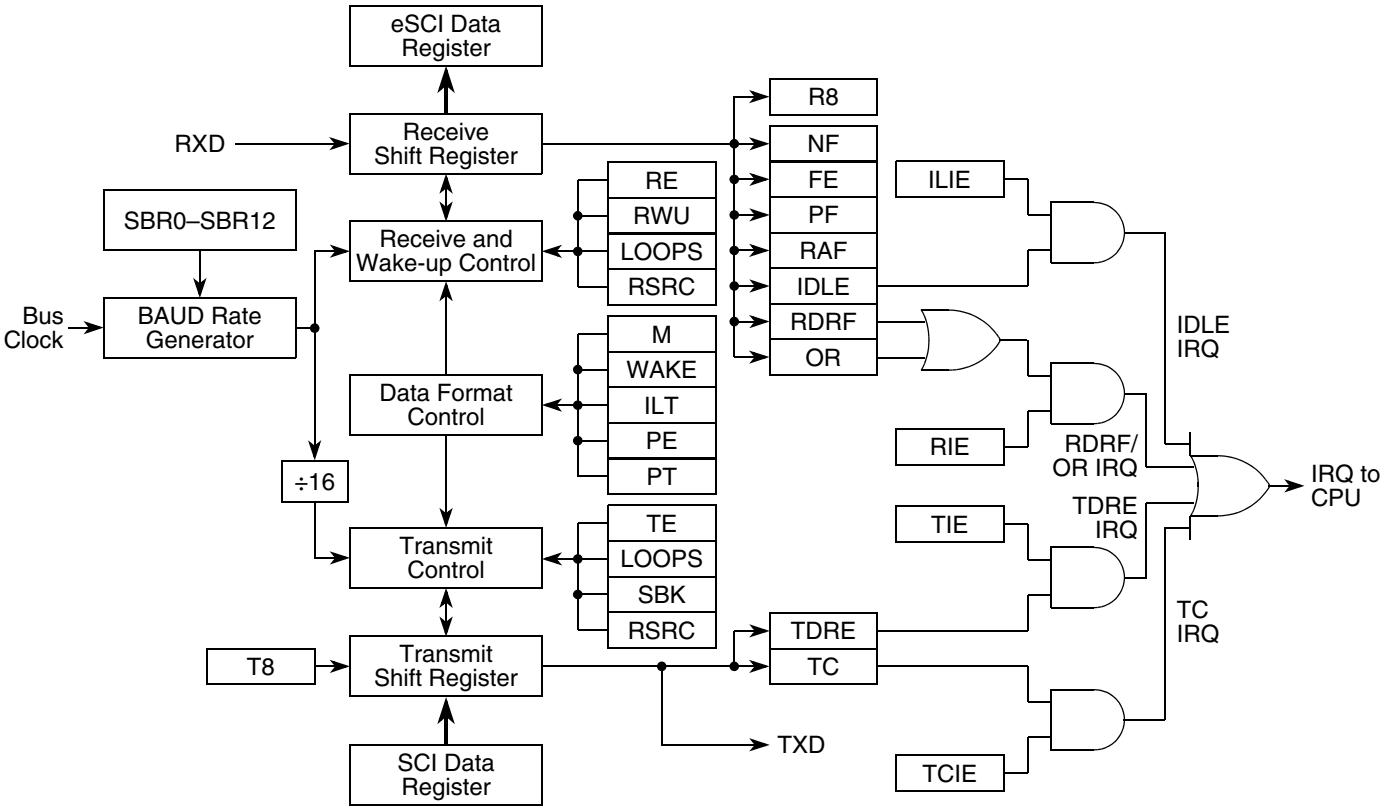


Figure 20-11. eSCI Operation Block Diagram

20.4.2 Data Format

The eSCI uses the standard NRZ mark/space data format. Each data character is contained in a frame that includes a start bit, eight or nine data bits, and a stop bit. Clearing the M bit in eSCI control register 1 configures the eSCI for 8-bit data characters. A frame with eight data bits has a total of 10 bits. Setting the M bit configures the eSCI for 9-bit data characters. A frame with nine data bits has a total of 11 bits.

When the eSCI is configured for 9-bit data characters, the ninth data bit is the T8 bit in the eSCI data register (ESCI*x*_DR). It remains unchanged after transmission and can be used repeatedly without rewriting it. A frame with nine data bits has a total of 11 bits.

The two different data formats are illustrated in Figure 20-12. Table 20-15 and Table 20-16 show the number of each type of bit in 8-bit data format and 9-bit data format, respectively.

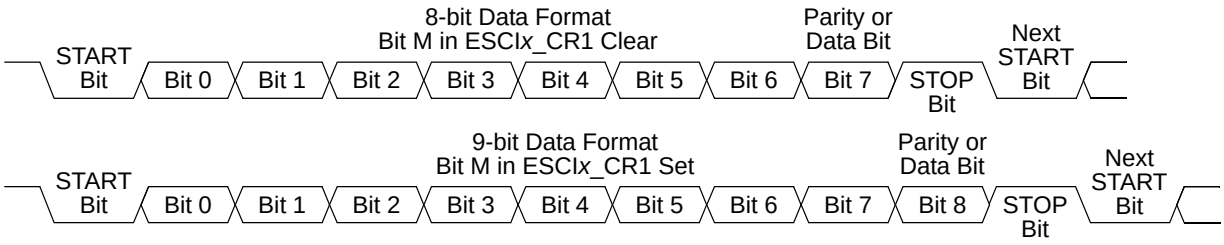


Figure 20-12. eSCI Data Formats

Table 20-15. Example of 8-bit Data Formats

Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
1	8	0	0	1
1	7	0	1	1
1	7	1 [1]	0	1

[1] The address bit identifies the frame as an address character. See Section 20.4.5.6, “Receiver Wake-up.”

Table 20-16. Example of 9-Bit Data Formats

Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
1	9	0	0	1
1	8	0	1	1
1	8	1 [1]	0	1

[1] The address bit identifies the frame as an address character. See Section 20.4.5.6, “Receiver Wake-up.”

20.4.3 Baud Rate Generation

A 13-bit modulus counter in the baud rate generator derives the baud rate for both the receiver and the transmitter. The value, 1 to 8191, written to the SBR0–SBR12 bits determines the system clock divider. The SBR bits are in the eSCI control register 1 (ESCI*x*_CR1). The baud rate clock is synchronized with the system clock and drives the receiver. The baud rate clock divided by 16 drives the transmitter. The receiver has an acquisition rate of 16 samples per bit time.

Baud rate generation is subject to one source of error when integer division of the system clock does not result in the exact target frequency.

Table 20-17 lists some examples of achieving target baud rates with a system clock frequency of 128 MHz.

$$\text{SCI baud rate} = \frac{\text{System clock}}{16 \times \text{ESCIx\_CR1[SBR]}}$$

Table 20-17. Baud Rates (Example: System Clock = 128 MHz)

Bits SBR[0:12]	Receiver Clock (Hz)	Transmitter Clock (Hz)	Target Baud Rate	Error (%)
0x0023	3,657,143	228,571	230,400	–0.79
0x0045	1,855,072	115,942	115,200	+0.64
0x008B	920,863	57,554	57,600	–0.01
0x00D0	615,385	38,462	38,400	+0.16

Table 20-17. Baud Rates (Example: System Clock = 128 MHz) (continued)

Bits SBR[0:12]	Receiver Clock (Hz)	Transmitter Clock (Hz)	Target Baud Rate	Error (%)
0x01A1	306,954	19,185	19,200	–0.08
0x022C	230,216	14,388	14,400	–0.08
0x0341	153,661	9,604	9600	+.04
0x0683	76,785	4,799	4800	–0.02
0x0D05	38,404	2,400.2	2400	+.01
0x1A0A	19,202	1,200.1	1200	+.01

20.4.4 Transmitter

Figure 20-13 illustrates the features of the eSCI transmitter.

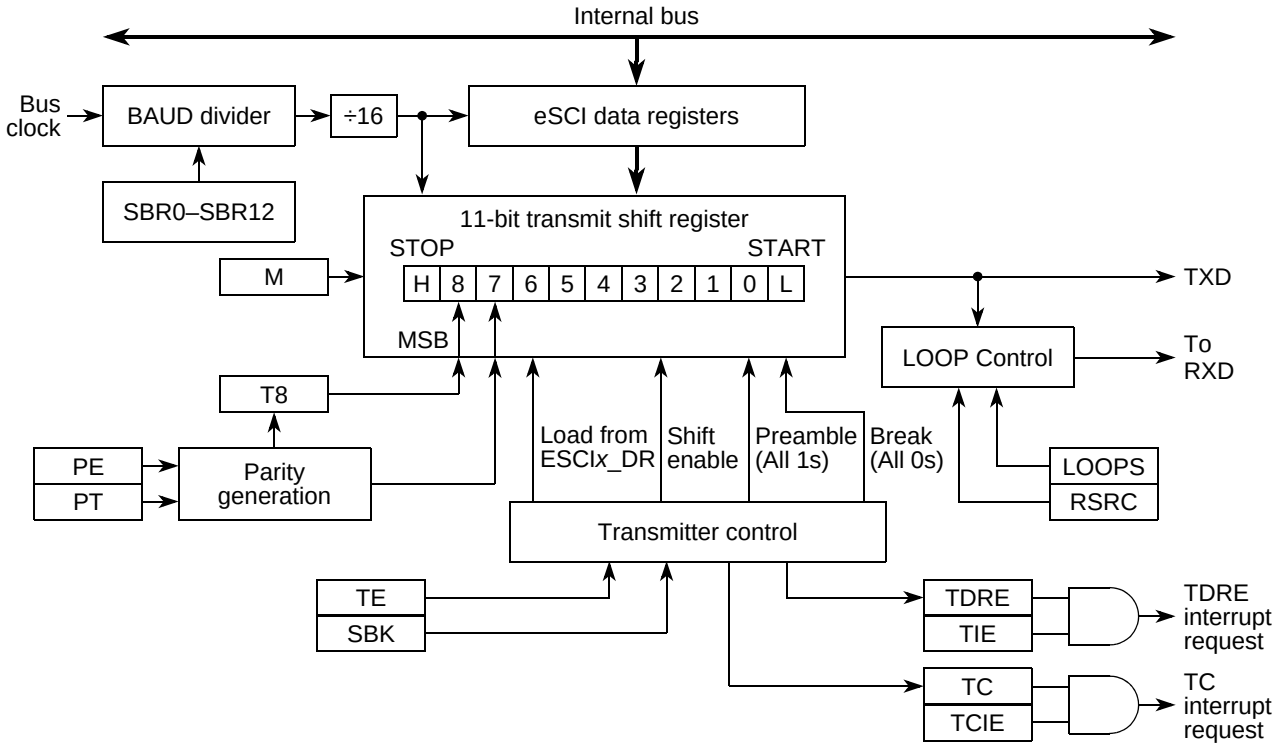


Figure 20-13. eSCI Transmitter Block Diagram

20.4.4.1 Transmitter Character Length

The eSCI transmitter can accommodate either 8-bit or 9-bit data characters. The state of the M bit in eSCI control register 1 (ESCI*x*_CR1) determines the length of data characters. When transmitting 9-bit data, bit T8 in the eSCI data register (ESCI*x*_DR) is the ninth bit (bit 8).

20.4.4.2 Character Transmission

To transmit data, the MCU writes the data bits to the eSCI data register (ESCI*x*_DR), which are transferred to the transmit shift register. The transmit shift register then shifts a frame out on the TXD signal, after it has prefaced them with a start bit and appended them with a stop bit. The eSCI data register (ESCI*x*_DR) is the buffer (write-only during transmit) between the internal data bus and the transmit shift register.

The eSCI sets the transmit data register empty flag (TDRE) every time it transfers data from the buffer (ESCI*x*_DR) to the transmit shift register. The transmit driver routine can respond to this flag by writing another byte to the transmitter buffer (ESCI*x*_DR), while the shift register is still shifting out the first byte.

To initiate an eSCI transmission:

1. Configure the eSCI:
   a) Turn on the module by clearing ESCI*x*_CR2[MDIS] if this bit is set.
   b) Select a baud rate. Write this value to the eSCI control register 1 (ESCI*x*_CR1) to start the baud rate generator. Remember that the baud rate generator is disabled when the ESCI*x*_CR1[SBR] field is zero. When using 8-bit writes, writes to the ESCI*x*_CR1[0–7] have no effect without also writing to ESCI*x*_CR1[8–15].
   c) Write to ESCIx_CR1 to configure word length, parity, and other configuration bits (LOOPS, RSRC, M, WAKE, ILT, PE, PT).
   d) Enable the transmitter, interrupts, receive, and wake-up as required, by writing to the ESCI*x*_CR1 register bits (TIE, TCIE, RIE, ILIE, TE, RE, RWU, SBK). A preamble or idle character is shifted out of the transmitter shift register.

NOTE

A single 32-bit write to ESCI_CR1 can be used to perform steps b–d above.

2. Transmit procedure for each byte:
   a) Poll the TDRE flag by reading the ESCI*x*_SR or responding to the TDRE interrupt. Keep in mind that the TDRE bit resets to 1.
   b) If the TDRE flag is set, software must then clear it, followed by writing the data to be transmitted to ESCIx_DR, where the ninth bit is written to the T8 bit in ESCIx_DR if the eSCI is in 9-bit data format.
3. Repeat step 2 for each subsequent transmission.

NOTE

The TDRE flag is set when the shift register is loaded with the next data to transmit from ESCI*x*_DR, which occurs approximately half-way through the stop bit of the previous frame. This transfer occurs 9/16ths of a bit time AFTER the start of the stop bit of the previous frame.

Toggling the TE bit from 0 to 1 automatically loads the transmit shift register with a preamble of 10 logic 1s (if M = 0) or 11 logic 1s (if M = 1). After the preamble shifts out, control logic transfers the data from the eSCI data register into the transmit shift register. A logic 0 start bit automatically goes into the least significant bit position of the transmit shift register. A logic 1 stop bit goes into the most significant bit position.

The eSCI hardware supports odd or even parity. When parity is enabled, the most significant bit (Msb) of the data character is the parity bit.

The transmit data register empty flag, TDRE, in the eSCI status register (ESCI*x*_SR) is set when the eSCI data register transfers a byte to the transmit shift register. The TDRE flag indicates that the eSCI data register can accept new data from the internal data bus. If the transmit interrupt enable bit (TIE), in eSCI control register 1 (ESCI*x*_CR1) is also set, the TDRE flag generates a transmit interrupt request.

When the transmit shift register is not transmitting a frame, the TXD output goes to the idle condition, logic 1. If at any time software clears the TE bit in eSCI control register 1 (ESCI*x*_CR1), the transmit enable signal goes low and the TXD output goes idle.

If software clears TE while a transmission is in progress (ESCI*x*_CR1[TC] = 0), the frame in the transmit shift register continues to shift out. To avoid accidentally cutting off the last frame in a message, always wait for TDRE to go high after the last frame before clearing TE.

To separate messages with preambles with minimum idle line time, use this sequence between messages:

1. Write the last byte of the first message to ESCI*x*_DR.
2. Wait for the TDRE flag to go high, indicating the transfer of the last frame to the transmit shift register.
3. Queue a preamble by clearing and then setting the TE bit.
4. Write the first byte of the second message to ESCI*x*_DR.

20.4.4.3 Break Characters

Setting the break bit, SBK, in eSCI control register 1 (ESCI*x*_CR1) loads the transmit shift register with a break character. A break character contains all logic 0s and has no start, stop, or parity bit. Break character length depends on the M bit in the eSCI control register 1 (ESCI*x*_CR1) and on the BRK13 bit in the eSCI control register 2 (ESCI*x*_CR2). As long as SBK is set, the transmitter logic continuously loads break characters into the transmit shift register. After software clears the SBK bit, the shift register finishes transmitting the last break character and then transmits at least one logic 1. The automatic logic 1 at the end of a break character guarantees the recognition of the start bit of the next frame.

NOTE

LIN 2.0 requires that a break character be 13-bits long, so always set the BRK13 bit to 1. The eSCI works with BRK13 = 0, but it violates LIN 2.0.

The eSCI recognizes a break character when a start bit is followed by eight or nine logic 0 data bits and a logic 0 in place of the stop bit. Receiving a break character has the following effects on eSCI registers:

- Sets the framing error flag, FE
- Sets the receive data register full flag, RDRF
- Clears the eSCI data register (ESCI*x*_DR)
- Can set a flag: overrun (OR), noise flag (NF), parity error flag (PF), or the receiver active flag (RAF). For more details, see Section 20.3.2.4, “eSCI Status Register (ESCIx_SR).”

20.4.4.4 Idle Characters

An idle character contains all logic 1s and has no start, stop, or parity bit. Idle character length depends on the M bit in eSCI control register 1 (ESCI*x*_CR1). The preamble is a synchronizing idle character that begins the first transmission initiated after toggling the TE bit from 0 to 1.

If the TE bit is cleared during a transmission, the TXD output becomes idle after completion of the transmission in progress. Clearing and then setting the TE bit during a transmission queues an idle character to be sent after the frame currently being transmitted.

NOTE

When queueing an idle character, return the TE bit to logic 1 before the stop bit of the current frame shifts out through the TXD output. Setting the TE bit after the stop bit shifts out through the TXD output causes data previously written to the eSCI data register to be lost. Toggle the TE bit for a queued idle character while the TDRE flag is set and immediately before writing the next byte to the eSCI data register.

20.4.4.5 Fast Bit Error Detection in LIN Mode

Fast bit error detection has been designed to allow flagging of LIN bit errors while they occur, rather than flagging them after a byte transmission has completed. To use this feature, it is assumed a physical interface connects to the LIN bus as shown in Figure 20-14.

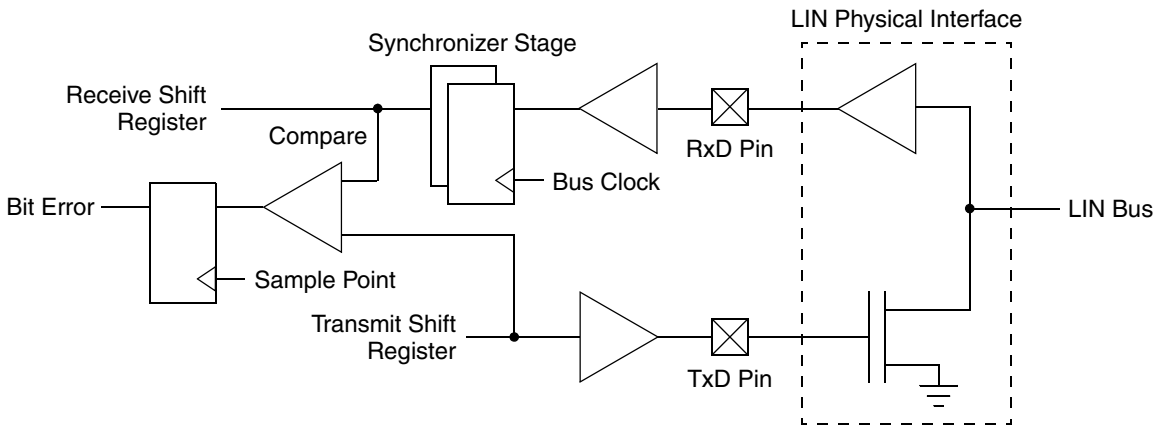


Figure 20-14. Fast Bit Error Detection on a LIN Bus

If fast bit error detection is enabled (FBR = 1), the eSCI compares the transmitted and the received data stream when the transmitter is active (not idle). After a mismatch between the transmitted data and the received data is detected the following actions are performed:

- The LIN frame is aborted (provided LDBG=0).
- The bit error flag BERR is set.
- If SBSTP is 0, the remainder of the byte is transmitted normally.
- If SBSTP is 1, the remaining bits in the byte after the error bit are transmitted as 1s (idle).

To adjust to different bus loads the sample point at which the incoming bit is compared to the one which was transmitted can be selected with the BESM13 bit (see Figure 20-15). If set, the comparison is performed at RT clock 13, otherwise at RT clock 9 (see Section 20.4.5.3, “Data Sampling.”).

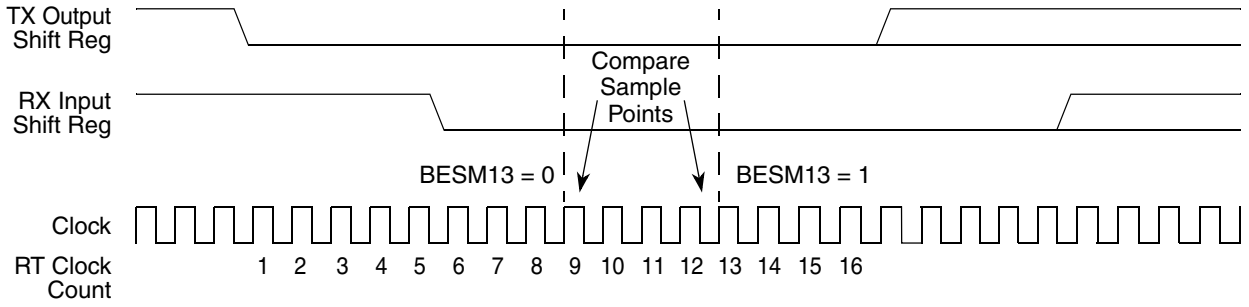


Figure 20-15. Fast Bit Error Detection Timing Diagram

20.4.5 Receiver

Figure 20-16 illustrates the eSCI receiver.

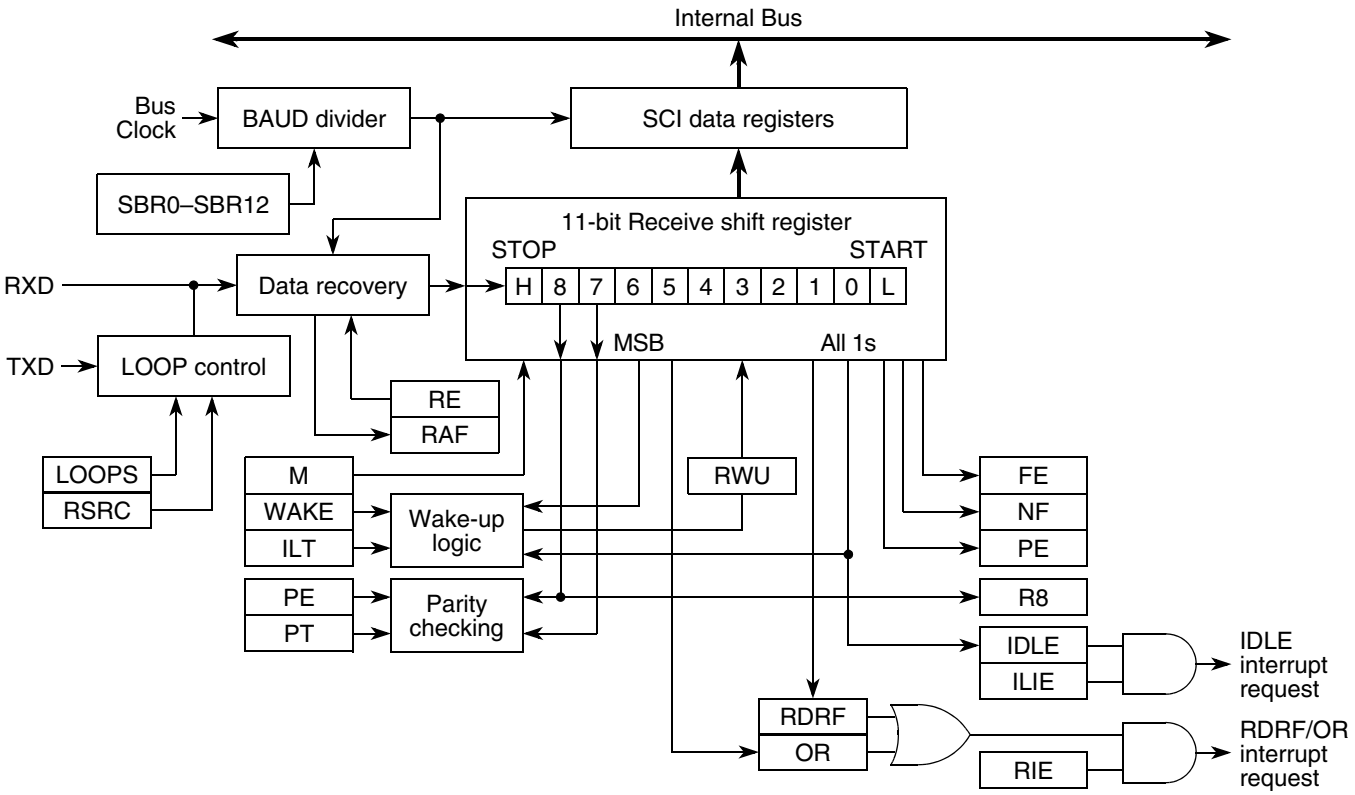


Figure 20-16. eSCI Receiver Block Diagram

20.4.5.1 Receiver Character Length

The eSCI receiver accepts 8-bit or 9-bit data characters. The state of the M bit in eSCI control register 1 (ESCI*x*_CR1) determines the bit-length of data characters. When receiving 9-bit data, bit R8 in the eSCI data register (ESCI*x*_DR) is the ninth bit (bit 8).

20.4.5.2 Character Reception

During an eSCI reception, the receive shift register shifts a frame in from the RXD input signal. The eSCI data register is the buffer (read-only during receive) between the internal data bus and the receive shift register.

After a complete frame shifts into the receive shift register, the data portion of the frame transfers to the eSCI data register. The receive data register full flag, RDRF, in eSCI status register (ESCI*x*_SR) is then set, indicating that the received byte can be read. If the receive interrupt enable bit, RIE, in eSCI control register 1 (ESCI*x*_CR1) is also set, the RDRF flag generates an RDRF interrupt request.

20.4.5.3 Data Sampling

The receiver uses a sampling clock to sample the RXD input signal at the 16 times the baud-rate frequency. This sampling clock is called the RT clock. To adjust for baud rate mismatch, the RT clock is re-synchronized (see Figure 20-17).

- After every start bit.
- After the receiver detects a data bit change from logic 1 to logic 0. This data bit change is detected when a majority of data samples return a valid logic 1 and a majority of the next data samples return a valid logic 0. Data samples are taken at RT8, RT9, and RT10, as shown in Figure 20-17.

To locate the start bit, eSCI data recovery logic performs an asynchronous search for a logic 0 preceded by three logic 1s. When the falling edge of a possible start bit occurs, the RT clock begins to count to 16.

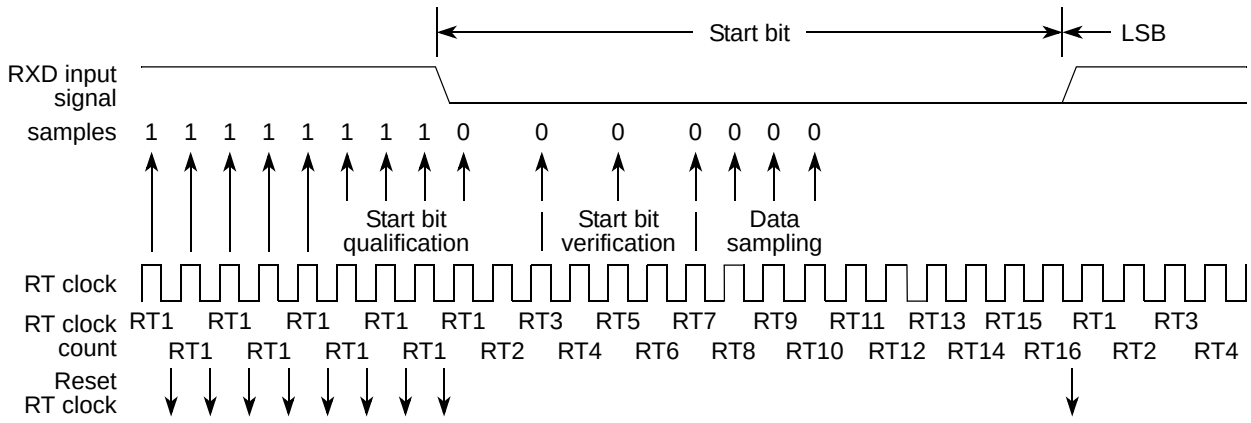


Figure 20-17. Receiver Data Sampling

To verify the start bit and to detect noise, the eSCI data recovery logic takes samples at RT3, RT5, and RT7. Table 20-18 summarizes the results of the start bit verification samples.

Table 20-18. Start Bit Verification

RT3, RT5, and RT7 Samples	Start Bit Verification	Noise Flag
000	Yes	0
001	Yes	1
010	Yes	1
011	No	0
100	Yes	1
101	No	0
110	No	0
111	No	0

If start bit verification is not successful, the RT clock is reset and a new search for a start bit begins.

To determine the value of a data bit and to detect noise, eSCI recovery logic takes samples at RT8, RT9, and RT10. Table 20-19 summarizes the results of the data bit samples.

Table 20-19. Data Bit Recovery

RT8, RT9, and RT10 Samples	Data Bit Determination	Noise Flag
000	0	0
001	0	1
010	0	1
011	1	1
100	0	1
101	1	1
110	1	1
111	1	0

NOTE

The RT8, RT9, and RT10 samples do not affect start bit verification. If any or all of the RT8, RT9, and RT10 start bit samples are logic 1s following a successful start bit verification, the noise flag (NF) is set.

To verify a stop bit and to detect noise, recovery logic takes samples at RT8, RT9, and RT10. Table 20-20 summarizes the results of the stop bit samples.

Table 20-20. Stop Bit Recovery

RT8, RT9, and RT10 Samples	Framing Error Flag	Noise Flag
000	1	0
001	1	1
010	1	1
011	0	1
100	1	1
101	0	1
110	0	1
111	0	0

In Figure 20-18 the verification samples RT3 and RT5 determine that the first low detected was noise and not the beginning of a start bit. The RT clock is reset and the start bit search begins again. The noise flag is not set because the noise occurred before the start bit was found.

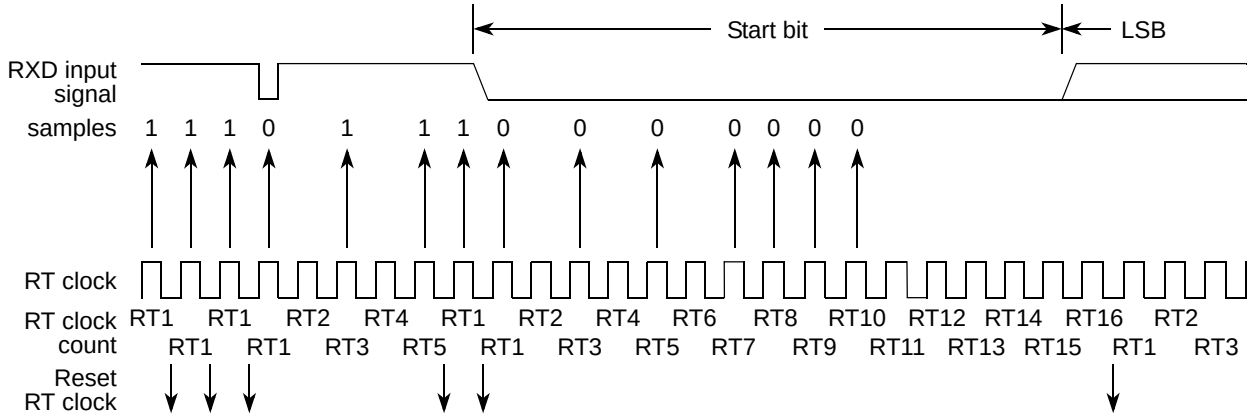


Figure 20-18. Start Bit Search Example 1

20.4.5.4 Framing Errors

If the data recovery logic sets the framing error flag, ESCI*x*_SR[FE], it does not detect a logic 1 where the stop bit must be in an incoming frame. A break character also sets the FE flag because a break character has no stop bit. The FE flag is set at the same time that the RDRF flag is set.

20.4.5.5 Baud Rate Tolerance

When a transmitting device operates at a baud rate below or above the receiver baud rate, accumulated bit-time misalignment can cause one of the three stop bit data samples (RT8, RT9, and RT10) to fall outside the stop bit. A noise error occurs if the RT8, RT9, and RT10 samples are not all the same logical values. A

framing error occurs if the receiver clock is misaligned such that the majority of the RT8, RT9, and RT10 stop bit samples are a logic zero.

The receiver samples an incoming frame and re-synchronizes the RT clock on any valid falling edge within the frame. Re-synchronization within frames corrects a misalignment between transmitter bit times and receiver bit times.

20.4.5.5.1 Slow Data Tolerance

Figure 20-19 shows how much a slow received frame can be misaligned without causing a noise error or a framing error. The slow stop bit begins at RT8 instead of RT1 but arrives in time for the stop bit data samples at RT8, RT9, and RT10.

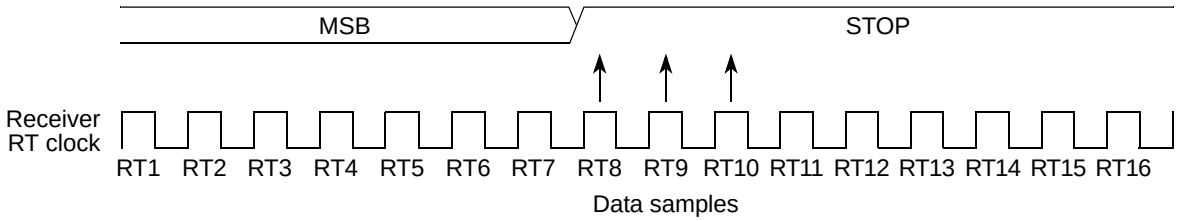


Figure 20-19. Slow Data

For an 8-bit data character, data sampling of the stop bit takes the receiver RT clock 151 clock cycles, as is shown below:

$$9 \text{ bit times} \times 16 \text{ RT cycles} + 7 \text{ RT cycles} = 151 \text{ RT cycles}$$

With the misaligned character shown in Figure 20-19, the receiver counts 151 RT cycles at the point when the count of the transmitting device is 9 bit times x 16 RT cycles = 144 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a slow 8-bit data character with no errors is 4.63%, as is shown below:

$$\frac{151 - 144}{151} \times 100 = 4.63\%$$

For a 9-bit data character, data sampling of the stop bit takes the receiver 167 RT cycles, as is shown below:

$$10 \text{ bit times} \times 16 \text{ RT cycles} + 7 \text{ RT cycles} = 167 \text{ RT cycles}$$

With the misaligned character shown in Figure 20-19, the receiver counts 167 RT cycles at the point when the count of the transmitting device is 10 bit times x 16 RT cycles = 160 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a slow 9-bit character with no errors is 4.19%, as is shown below:

$$\frac{167 - 160}{167} \times 100 = 4.19\%$$

20.4.5.5.2 Fast Data Tolerance

Figure 20-20 shows how much a fast received frame can be misaligned. The fast stop bit ends at RT10 instead of RT16 but is still sampled at RT8, RT9, and RT10.

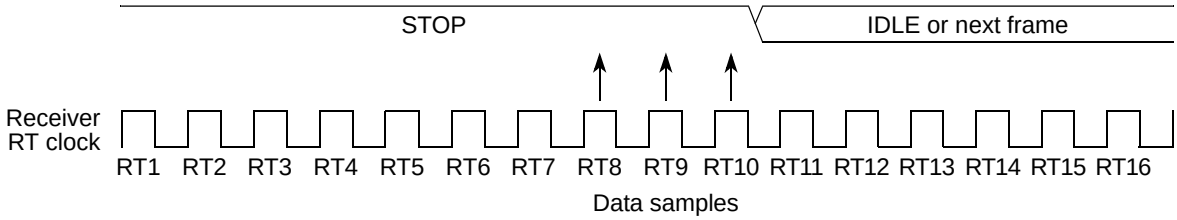


Figure 20-20. Fast Data

For an 8-bit data character, data sampling of the stop bit takes the receiver 154 RT cycles, as is shown below:

$$9 \text{ bit times} \times 16 \text{ RT cycles} + 10 \text{ RT cycles} = 154 \text{ RT cycles}$$

With the misaligned character shown in Figure 20-20, the receiver counts 154 RT cycles at the point when the count of the transmitting device is 10 bit times x 16 RT cycles = 160 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a fast 8-bit character with no errors is 3.40%, as is shown below:

$$\frac{160 - 154}{160} \times 100 = 3.40\%$$

For a 9-bit data character, data sampling of the stop bit takes the receiver 170 RT cycles, as shown below:

$$10 \text{ bit times} \times 16 \text{ RT cycles} + 10 \text{ RT cycles} = 170 \text{ RT cycles}$$

With the misaligned character shown in Figure 20-20, the receiver counts 170 RT cycles at the point when the count of the transmitting device is 11 bit times x 16 RT cycles = 176 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a fast 9-bit character with no errors is 3.40%, as is shown below:

$$\frac{176 - 170}{176} \times 100 = 3.40\%$$

20.4.5.6 Receiver Wake-up

The receiver can be put into a standby state, which disregards all input requests targeted for other receivers in multiple-receiver systems. Setting the receiver wake-up bit (RWU) in eSCI control register 1 (ESCI*x*_CR1) puts the receiver into the standby state, which disregards all receiver interrupts tar. The eSCI loads the received data into the ESCI*x*_DR, but does not set the receive data register full (RDRF) flag.

The transmitting device can address messages to selected receivers by including addressing information (address bits) in the initial frame or frames of each message. See section Section 20.4.2, “Data Format,” for an example of address bits.

The WAKE bit in eSCI control register 1 (ESCI*x*_CR1) determines how the eSCI is brought out of the standby state to process an incoming message. The WAKE bit enables either idle line wake-up or address mark wake-up.

20.4.5.6.1 Idle Input Line Wake-up (WAKE = 0)

Using the receiver idle input line wake-up method allows an idle condition on the RXD signal clears the ESCI*x*_CR1[RWU] bit and wakes up the eSCI. The initial frame or frames of every message contain addressing information. All receivers evaluate the addressing information, and receivers for which the message is addressed process the frames that follow. Any receiver for which a message is not addressed can set its RWU bit and return to the standby state. The RWU bit remains set and the receiver remains on standby until another idle character appears on the RXD signal.

Idle line wake-up requires that messages be separated by at least one idle character and that no message contains idle characters.

The idle character that wakes a receiver does not set the receiver idle bit, ESCI*x*_SR[IDLE], or the receive data register full flag, RDRF.

The idle line type bit, ESCI*x*_CR1[ILT], determines whether the receiver begins counting logic 1s as idle character bits after the start bit or after the stop bit.

20.4.5.6.2 Address Mark Wake-up (WAKE = 1)

Using the address mark wake-up method allows a logic 1 in the most significant bit (MSB) position of a frame to clear the RWU bit and wake-up the eSCI. The logic 1 in the msb position marks a frame as an address frame that contains addressing information. All receivers evaluate the addressing information, and the receivers for which the message is addressed process the frames that follow. Any receiver for which a message is not addressed can set its RWU bit and return to the standby state. The RWU bit remains set and the receiver remains on standby until another address frame appears on the RXD signal.

The logic 1 msb of an address frame clears the receiver’s RWU bit before the stop bit is received and sets the RDRF flag.

Address mark wake-up allows messages to contain idle characters but requires that the msb be reserved for use in address frames.

NOTE

With the WAKE bit clear, setting the RWU bit after the RXD signal has been idle can cause the receiver to wake-up immediately.

20.4.6 Single-Wire Operation

Normally, the eSCI uses two pins for transmitting and receiving. In single-wire operation, the RXD pin is disconnected from the eSCI. The eSCI uses the TXD pin for both receiving and transmitting.

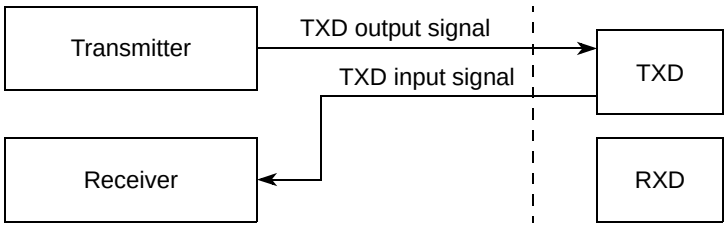


Figure 20-21. Single-Wire Operation (LOOPS = 1, RSRC = 1)

Enable single-wire operation by setting the LOOPS bit and the receiver source bit, RSRC, in eSCI control register 1 (ESCI*x*_CR1). Setting the LOOPS bit disables the path from the RXD signal to the receiver. Setting the RSRC bit connects the receiver input to the output of the TXD pin driver.

During reception, both the transmitter and receiver must be enabled (TE = 1 and RE = 1). The SIU_PCR89[PA] and SIU_PCR91[PA] bits must be set to select the TXD function for the relevant eSCI module, and the TXD pin must be set for open drain operation (SIU_PCR*nn*[ODE] = 1). Optionally, if the external transmitting device is also open drain, a weak pullup can be enabled.

See Section 6.4.1.12, “Pad Configuration Registers (SIU_PCR)”.

During transmission, the transmitter must be enabled (TE = 1); the receiver can be enabled or disabled. If the receiver is enabled (RE = 1), transmissions are echoed back on the receiver. Set or clear open drain output enable depending on desired operation.

20.4.7 Loop Operation

In loop operation the transmitter output goes to the receiver input. The RXD signal is disconnected from the eSCI.

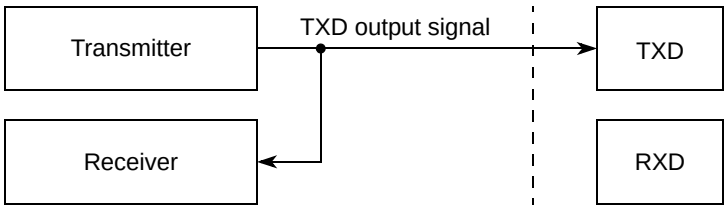


Figure 20-22. Loop Operation (LOOPS = 1, RSRC = 0)

Enable loop operation by setting the LOOPS bit and clearing the RSRC bit in eSCI control register 1 (ESCI*x*_CR1). Setting the LOOPS bit disables the path from the RXD signal to the receiver. Clearing the RSRC bit connects the transmitter output to the receiver input. Both the transmitter and receiver must be enabled (TE = 1 and RE = 1).

20.4.8 Modes of Operation

20.4.8.1 Run Mode

Run mode is the normal operating mode.

20.4.8.2 Disabling the eSCI

The module disable bit (ESCIx_CR2[MDIS]) in the eSCI control register 2 can be used to turn off the eSCI. This saves power by stopping the eSCI core from being clocked. By default the eSCI is enabled (ESCIx_CR2[MDIS]=0).

20.4.9 Interrupt Operation

Only the eSCI originates interrupt requests. The following sections describe how the eSCI generates a request and how the MCU acknowledges that request. The eSCI only has a single interrupt line (eSCI interrupt signal, active high operation) and all the following interrupts, when generated, are ORed together and issued through that port.

20.4.9.1 Interrupt Sources

There are several interrupt sources that can generate an eSCI interrupt to the CPU. They are listed with details and descriptions in Table 20-21.

Table 20-21. eSCI Interrupt Flags, Sources, Mask Bits, and Descriptions

Interrupt Source	Flag	Description	Source	Local Enable
Transmitter	TDRE	Indicates that a byte was transferred from ESCIx_DR to the transmit shift register. The transmit data register empty (TDRE) interrupt is set high by the eSCI when the transmit shift register receives data, 8 or 9 bits, from the eSCI data register, ESCI*x*_DR. A TDRE interrupt indicates that the transmit data register (ESCI*x*_DR) is empty and that a new data can be written to the ESCI*x*_DR for transmission. The TDRE bit is cleared by writing a one to the TDRE bit location in the ESCI*x*_SR.	ESCI*x*_SR[0]	TIE
Transmitter	TC	Indicates that a transmit is complete. The transmit complete (TC) interrupt is set by the eSCI when a transmission has completed. A TC interrupt indicates that there is no transmission in progress. TC is set high when the TDRE flag is set and no data, preamble, or break character is being transmitted. When TC is set, the TXD pin becomes idle (logic 1). The TC bit is cleared by writing a one to the TC bit location in the ESCI*x*_SR.	ESCI*x*_SR[1]	TCIE
Receiver	RDRF	Indicates that received data is available in the eSCI data register. The receive data register full (RDRF) interrupt is set when the data in the receive shift register transfers to the eSCI data register. An RDRF interrupt indicates that the received data has been transferred to the eSCI data register and that the received data can now be read by the MCU. The RDRF bit is cleared by writing a one to the RDRF bit location in the ESCI*x*_SR.	ESCI*x*_SR[2]	RIE

Table 20-21. eSCI Interrupt Flags, Sources, Mask Bits, and Descriptions (continued)

Interrupt Source	Flag	Description	Source	Local Enable
Receiver	IDLE	Indicates that receiver input has become idle. The idle line (IDLE) interrupt is set when 10 consecutive logic 1s (if M = 0) or 11 consecutive logic 1s (if M = 1) appear on the receiver input. After the IDLE is cleared, a valid frame must again set the RDRF flag before an idle condition can set the IDLE flag. The IDLE bit is cleared by writing a one to the IDLE bit location in the ESCI*x*_SR.	ESCI*x*_SR[3]	ILIE
Receiver	OR	Indicates that an overrun condition has occurred. The overrun (OR) interrupt is set when software fails to read the eSCI data register before the receive shift register receives the next frame. The newly acquired data in the shift register is lost in this case, but the data already in the eSCI data registers is not affected.The OR bit is cleared by writing a one to the OR bit location in the ESCI*x*_SR.	ESCI*x*_SR[4]	ORIE
Receiver	NF	Detect noise error on receiver input. The NF interrupt is set when the eSCI detects noise on the receiver input.	ESCI*x*_SR[5]	NFIE
Receiver	FE	Framing error has occurred. The interrupt is set when the stop bit is read as a 0; which violates the SCI protocol. FE is cleared by writing it with 1.	ESCI*x*_SR[6]	FEIE
Receiver	PF	Parity of received data does not match parity bit; parity error has occurred. The interrupt is set when the parity of the received data is not correct. PF is cleared by writing it with 1.	ESCI*x*_SR[7]	PFIE
LIN	BERR	Detected a bit error, only valid in LIN mode. While the eSCI is in LIN mode, the bit error (BERR) flag is set when one or more bits in the last transmitted byte is not read back with the same value. The BERR flag is cleared by writing a 1 to the bit. A bit error causes the LIN FSM to reset. Clear the BERR flag by writing a 1 to the bit.	ESCI*x*_SR[11]	IEBERR
LIN	RXRDY	Indicates LIN hardware has received a data byte. While in LIN mode, the receiver ready (RXRDY) flag is set when the eSCI receives a valid data byte in an RX frame. RXRDY is not set for bytes which the receiver obtains by reading back the data which the LIN finite state machine (FSM) has sent out. Clear the RXRDY flag by writing a 1 to the bit.	ESCI*x*_SR[16]	RXIE
LIN	TXRDY	Indicates LIN hardware can accept a control or data byte. While in LIN mode, the transmitter ready (TXRDY) flag is set when the eSCI can accept a control or data byte. Clear the TXRDY flag by writing a 1 to the bit.	ESCI*x*_SR[17]	TXIE
LIN	LWAKE	A wake-up character has been received from a LIN frame. The LIN wake-up (LWAKE) flag is set when the LIN hardware receives a wake-up character sent by one of the LIN slaves. This occurs only when the LIN bus is in sleep mode. Clear the LWAKE flag by writing a 1 to the bit.	ESCI*x*_SR[18]	WUIE
LIN	STO	The response of the slave has been too slow (slave timeout). The slave timeout (STO) flag is set during an RX frame when the LIN slave has not transmitted all requested data bytes before the specified timeout period. Clear the STO flag by writing a 1 to the bit.	ESCI*x*_SR[19]	STIE

Table 20-21. eSCI Interrupt Flags, Sources, Mask Bits, and Descriptions (continued)

Interrupt Source	Flag	Description	Source	Local Enable
LIN	PBERR	Physical bus error detected. If the RXD input remains at the same value for 15 cycles after a transmission has started, the LIN hardware sets the physical bus error (PBERR) flag. Clear the PBERR flag by writing a 1 to the bit.	ESCI*x*_SR[20]	PBIE
LIN	CERR	CRC error detected. If an RX frame has the CRC checking flag set, and the two CRC bytes do not match the calculated CRC pattern, the CRC error (CERR) flag is set. Clear the CERR flag by writing a 1 to the bit.	ESCI*x*_SR[21]	CIE
LIN	CKERR	Checksum error detected. If an RX frame has the checksum checking flag set and the last byte does not match the calculated checksum, the checksum error (CKERR) flag is set. Clear the CKERR flag by writing a 1 to the bit.	ESCI*x*_SR[22]	CKIE
LIN	FRC	LIN frame completed. The frame complete (FRC) flag is set after the last byte of a TX frame is transmitted, or after the last byte of an RX frame is received. The FRC flag indicates that the next frame can be set up. Clear the FRC flag by writing a 1 to the bit. **Note:** The last byte of an outgoing TX frame or incoming RX frame indicates that the checksum comparison occurred. **Note:** It is possible to set the FRC flag before the DMA controller has completed transferring the last byte from the eSCI port to system memory. Do not set the FRC flag if the frame is processed. For frames that are processed, use the DMA controller interrupt.	ESCI*x*_SR[23]	FCIE
LIN	OVFL	ESCIx_LRR overflow. The overflow (OVFL) flag is set when a byte is received in the ESCI*x*_LRR before the previous byte is read. Since the system is responsible for reading the register before the next byte arrives, this condition indicates a problem with CPU load. The OVFL flag is cleared by writing a 1 to the bit.	ESCI*x*_SR[31]	OFIE

20.4.10 Using the LIN Hardware

The eSCI provides special support for the LIN protocol. It can be used to automate most tasks of a LIN master. In conjunction with the DMA interface it is possible to transmit entire frames (or sequences of frames) and receive data from LIN slaves without any CPU intervention. There is no special support for LIN slave mode. If required, LIN slave mode can be implemented in software.

A LIN frame consists of a break character (10 or 13 bits), a sync field, an ID field, *n* data fields (*n* could be 0) and a checksum field. The data and checksum bytes are either provided by the LIN master (TX frame) or by the LIN slave (RX frame). The header fields are always generated by the LIN master.

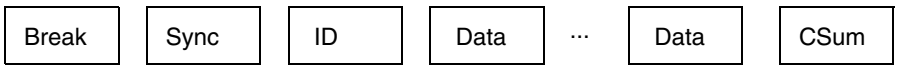


Figure 20-23. Typical LIN frame

The LIN hardware is highly configurable. This configurability allows the eSCI's LIN hardware to generate frames for LIN slaves from all revisions of the LIN standard. The settings are adjusted according to the capabilities of the slave device.

To activate the LIN hardware, the LIN mode bit in the ESCI*x*_LCR needs to be set. Other settings, such as double stop flags after bit errors and automatic parity bit generation, are also available for use in LIN mode.

The eSCI settings must be made according to the LIN specification. The eSCI must be configured for 2-wire operation (2 wires connected to the LIN transceiver) with 8 data bytes and no parity. Normally a 13-bit break is used, but the eSCI can also be configured for 10-bit breaks as required by the application.

20.4.10.1 Features of the LIN Hardware

The eSCI's LIN hardware has several features to support different revisions of the LIN slaves. The ESCI*x*_LTR can be configured to include or not include header bits in the checksum on a frame by frame basis. This feature supports LIN slaves with different LIN revisions. The LIN control register allows the application to automatically calculate the parity bits in the ID field and insert double stop flags a bit error. The BRK13 bit in ESCI*x*_CR2 determines the number of break characters generated: 10 or 13.

NOTE

LIN 2.0 requires a 13-bit break character. Set the BRK13 bit to 1. The eSCI bus works when BRK13 = 0, but the setting does not comply with LIN 2.0.

The application software can disable checksum generation/verification for individual frames to perform the functions externally and use the LIN hardware to append two CRC bytes (Figure 20-24). Although the LIN standard does not include CRCs, CRCs are processed as data bytes by the LIN protocol. CRCs are used in software applications that process very large frames. The eSCI and FlexCAN modules use the same CRC polynomial, the LIN protocol processes CAN bytes as data bytes.

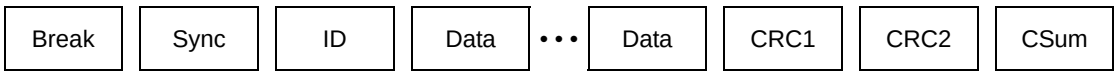


Figure 20-24. LIN Frame with CRC bytes

To force a resync of the LIN FSM, use the LRES bit in the LIN control register. Typically LIN hardware automatically discards the frame when a bit error is detected.

20.4.10.2 Generating a TX Frame

The following procedure describes how a basic TX frame is generated.

The frame is controlled via the LIN transmit register (ESCI*x*_LTR). Initially, the application software must check the TXRDY bit (either using an interrupt, the TX DMA interface, or by polling the LIN status register). If TXRDY is set, the register is writable. Before each write, TXRDY must be checked (though this step is performed automatically in DMA mode). The first write to the ESCI*x*_LTR must contain the LIN ID field. The next write to ESCI*x*_LTR specifies the length of the frame (0 to 255 Bytes). The third write to ESCI*x*_LTR contains the control byte (frame direction, checksum/CRC settings). Timeout bits are not included in TX frames, since they only see LIN slaves. The three previously mentioned writes to the

ESCI*x*_LTR specify the LIN frame data. After the LIN frame data is specified, the eSCI LIN hardware starts to generate a LIN frame.

First, the eSCI transmits a break field. The sync field is transmitted next. The third field is the ID field. After these three fields have been broadcast, the ESCIx_LTR accepts data bytes; the LIN hardware transmits these data bytes as soon as they are available and can be sent out. After the last step the LIN hardware automatically appends the checksum field.

You can set up a DMA channel to manage all the tasks required to send a TX frame, as shown in Figure 20-25. For this operation, the TX DMA channel must be activated by setting the ESCI*x*_CR2[TXDMA] bit. The control information for the LIN frame (ID, message length, TX/RX type, timeout, etc.) and the data bytes are stored at an appropriate memory location. The DMA controller is then set up to transfer this block of memory to a location (the ESCI*x*_LTR). After transmission is complete, either the DMA controller or the LIN hardware can generate an interrupt to the CPU.

NOTE

In contrast to the standard software implementation where each byte transmission requires several interrupts, the DMA controller and eSCI handle communication, bit error and physical bus error checking, checksum, and CRC generation (checking on the RX side).

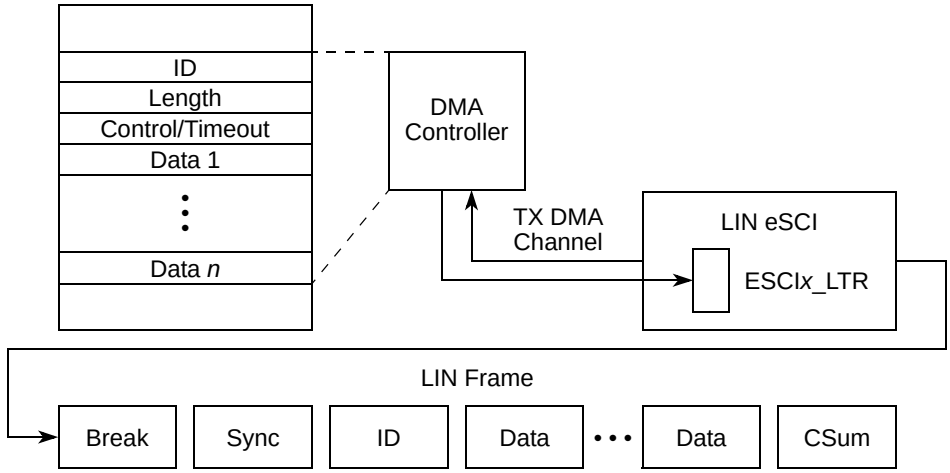


Figure 20-25. DMA Transfer of a TX Frame

20.4.10.3 Generating an RX Frame

For RX frames the header information is provided by the LIN master. The data, CRC and checksum bytes (as enabled) are provided by the LIN slave. The LIN master verifies CRC and checksum bytes transmitted by the slave.

For an RX frame, control information must be written to the ESCI*x*_LTR in the same manner as for the TX frames. Additionally the timeout bits, which define the time to complete the entire frame, must be written. Then the ESCI*x*_SR[RXRDY] bit must be checked (either with an interrupt, RX DMA interface, or by polling) to detect incoming data bytes. The checksum byte normally does not appear in the

ESCI*x*_LRR, instead the LIN hardware verifies the checksum and issue an interrupt, if the checksum value is not correct.

Two DMA channels can be used when executing an RX frame: one to transfer the header/control information from a memory location to the ESCI*x*_LTR, and one to transfer the incoming data bytes from the ESCI*x*_LRR to a table in memory. After the last byte from the RX frame has been stored, the DMA controller can indicate completion to the CPU.

NOTE

It is also possible to setup a whole sequence of RX and TX frames, and generate a single event at the end of that sequence.

See Figure 20-26 for more information.

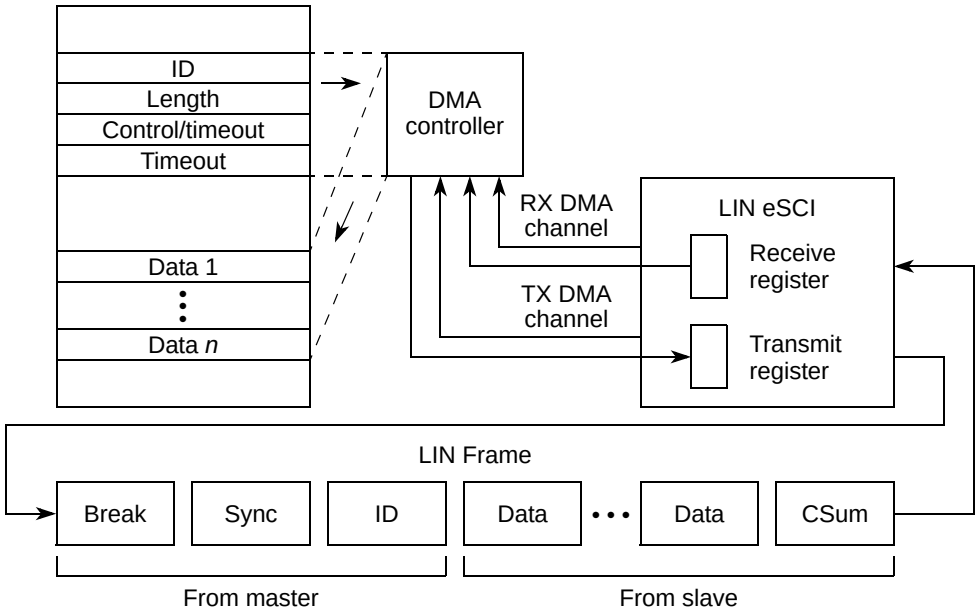


Figure 20-26. DMA Transfer of an RX frame

20.4.10.4 LIN Error Handling

The LIN hardware can detect several error conditions of the LIN protocol. LIN hardware receives all transmitted bytes, and compares the values with expected values to determine if the data is valid. If a mismatch occurs, a bit error is generated and the LIN FSM returns to its start state.

For an RX frame the LIN hardware can detect a slave timeout error. The exact slave timeout error value can be set via the timeout bits in the ESCI*x*_LTR. If the frame is not complete within the number of clock cycles specified in the register, the LIN FSM returns to its start state, and the STO interrupt is issued.

The LIN protocol supports a sleep mode. After 25,000 bus cycles of inactivity the bus is assumed to be in sleep mode. Normally entering sleep mode can be avoided, if the LIN master is regularly creating some bus activity. Otherwise the timeout state needs to be detected by the application software, for example by setting a timer.

Both LIN masters and LIN slaves can cause the bus to exit sleep mode by sending a break signal. The LIN hardware generates a break when the WU bit in the LIN control register is written. After transmitting the break, data is not sent out (TXRDY = 0) until the wake-up period expires. Define the wakeup period using the WUD bits in the LIN control register.

Break signals sent by a LIN slave are received by the LIN hardware, and so indicated by setting the WAKE flag in the LIN status register.

A physical bus error (LIN bus is permanently stuck at a fixed value) sets several error flags. If the input is permanently low, the eSCI sets the framing error (FE) flag in the eSCI status register. If the RXD input remains at the same value for 15 cycles after a transmission starts, the LIN hardware sets the PBERR flag in the LIN status register. A bit error can also occur.

20.4.10.5 LIN Setup

Since the eSCI is for general-purpose use, some of the settings are not applicable for LIN operation. The following setup applies for most applications, regardless of which kind of LIN slave is addressed:

1. Enable the module by clearing the ESCI*x*_CR2[MDIS] bit to 0.
2. Enable transmit and receive by setting ESCI*x*_CR1[TE] = 1, ESCI*x*_CR1[RE] = 1.
3. Clear the data format bit (ESCI*x*_CR1[M] = 0) to select 8 data bits, and disable the parity bit (PE = 0).
4. Use the LIN interrupts by clearing the interrupt enable bits: ESCI*x*_CR1[TIE], ESCI*x*_CR1[TCIE], and ESCI*x*_CR1[RIE]. Select LIN mode by setting ESCI*x*_LCR[LIN] = 1.
5. Set the break character (ESCI*x*_CR2[BRK13] = 1) to comply with the LIN standard requirements. The eSCI works when BRK13 = 0, but violates LIN 2.0.
6. Bit errors are commonly configured to: reset the LIN FSM, immediately stop bus transfers, and suspend DMA requests until the BERR flag is cleared. Use the following bit settings to perform these functions: ESCI*x*_LCR[LDBG] = 0, ESCI*x*_CR2[SBSTP] = 1, and ESCI*x*_CR2[BSTP] = 1.
7. Fast bit error detection provides superior error checking. Set ESCI*x*_CR2[FBR]; it is commonly used with ESCI*x*_CR2[BESM13] = 1.
8. If available, enable a pulldown on the RX input. (If the transceiver fails, the RX pin does not float).
9. Enable the following error indicators NF, FE, BERR, STO, PBERR, CERR, CKERR, and OVFL.
10. Transmit a wake-up character on the LIN bus to activate the LIN slaves.

Other settings like baud rate, length of break character etc., depend on the LIN slaves to which the eSCI is connected.

Chapter 21
FlexCAN2 Controller Area Network

21.1 Introduction

The device MCU contains two controller area network (FlexCAN2) modules. Each FlexCAN2 module is a communication controller implementing the CAN protocol according to CAN Specification version 2.0B and ISO Standard 11898.

Each FlexCAN2 module contains a 1024-byte embedded memory, capable of storing up to 64 message buffers (MBs). The respective functions are described in subsequent sections.This FlexCAN2 version implements individual mask registers and a reception queue thereby allowing queuing of received frames before requiring interrupt processing. Also included is a feature for disabling self-reception of TX frames.

21.1.1 Block Diagram

A general block diagram is shown in Figure 21-1, which describes the main submodules implemented in the FlexCAN2 module, including an embedded RAM for up to 64 message buffers.

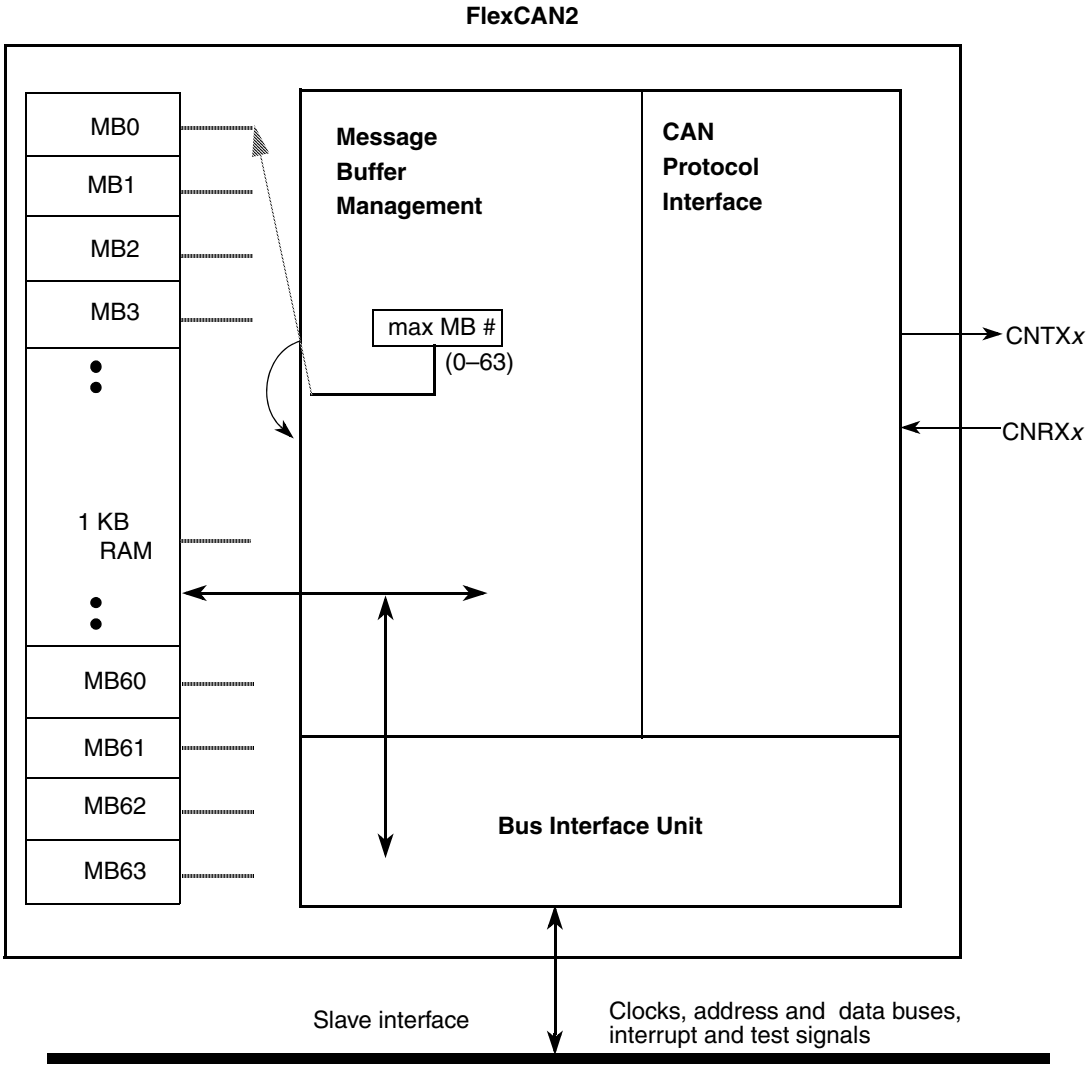


Figure 21-1. FlexCAN2 Block Diagram

21.1.2 Overview

The CAN protocol was designed primarily, but not exclusively, to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN2 module is a full implementation of the CAN protocol specification, Version 2.0 B, which supports both standard and extended message frames. Sixty-four message buffers (MBs) are stored in an embedded 1024-byte RAM dedicated to the FlexCAN2 module.

The CAN protocol interface (CPI) manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The message buffer management (MBM) handles message buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The bus interface unit (BIU) controls the access to and from the internal interface bus, to establish connection to the CPU and to any other modules. Clocks, address and data buses, interrupt outputs and test signals are accessed through the bus interface unit.

21.1.3 Features

The FlexCAN2 module includes these distinctive features:

- Based on and includes all existing features of the Freescale TouCAN module
- Reception queue available by setting more than one RX message buffer with the same ID
- Programmable for global (compatible with previous versions) or individual receive ID masking
- Maskable self-reception by setting MCR[SRXDIS]
- Full implementation of the CAN protocol specification, version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Data length of 0–8 bytes
  - Programmable bit rate up to 1 Mb/sec
- Content-related addressing
- 64 flexible message buffers of 0–8 bytes data length
- Each MB configurable as RX or TX, all supporting standard and extended messages
- Includes 1024 bytes of RAM used for message buffer storage
- Includes 256 bytes of RAM used for filtering individual RX mask registers
- Programmable clock source to the CAN protocol interface, either system clock or oscillator clock
- Listen-only mode capability
- Programmable loop-back mode supporting self-test operation
- Three programmable mask registers are turned off by default:
  - Global
  - RX Buffer 14
  - RX Buffer 15
- Programmable transmit-first scheme: lowest ID or lowest buffer number
- Time stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Multi master concept
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages

21.1.4 Modes of Operation

The device supports four FlexCAN functional modes: normal, freeze, listen-only and loop-back. One low power mode, module disabled, is supported.

21.1.4.1 Normal Mode

In normal mode, the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN protocol functions are enabled. In this device, there is no distinction between user and supervisor modes.

21.1.4.2 Freeze Mode

Freeze mode is entered when the FRZ bit in the module configuration register (CAN*x*_MCR) is asserted while the HALT bit in the CAN*x*_MCR is set or debug mode is requested by the NPC. In freeze mode no transmission or reception of frames is done, and synchronization with the CAN bus is lost. See Section 21.4.6.1, “Freeze Mode,” for more information.

21.1.4.3 Listen-Only Mode

The module enters this mode when the LOM bit in the CAN*x*_CR is asserted. In this mode, FlexCAN operates in a CAN error passive mode, freezing all error counters and receiving messages without sending acknowledgments.

21.1.4.4 Loop-Back Mode

The module enters this mode when the LPB bit in the CAN*x*_CR is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The CAN receive input pin (CNRX*x*) is ignored and the transmit output (CNTX*x*) goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

21.1.4.5 Module Disabled Mode

This low power mode is entered when the MDIS bit in the CAN_MCR is asserted. When disabled, the module shuts down the clocks to the CAN protocol interface and message buffer management submodules. Exit from this mode is done by negating the CAN_MCR[MDIS] bit. See Section 21.4.6.2, “Module Disabled Mode,” for more information.

21.2 External Signal Description

21.2.1 Overview

The FlexCAN2 module has two I/O signals connected to the external MCU pins. These signals are summarized in Table 21-1 and described in more detail in the next sub-sections.

Table 21-1. FlexCAN2 Signals

Signal Name[1]	Direction	Description
CNRX*x*	I	CAN receive
CNTX*x*	O	CAN transmit

1 *x* indicates FlexCAN2 module A or C.

21.2.2 Detailed Signal Description

21.2.2.1 CNRX*x*

This pin is the receive pin to the CAN bus transceiver. The dominant state is represented by logic level 0. The recessive state is represented by logic level 1.

21.2.2.2 CNTX*x*

This pin is the transmit pin to the CAN bus transceiver. The dominant state is represented by logic level 0. The recessive state is represented by logic level 1.

21.3 Memory Map/Register Definition

This section describes the registers and data structures in the FlexCAN2 module. The addresses presented are relative to the base address of the module.

The address space occupied by FlexCAN2 is contiguous:

- 128 bytes for registers starting at the module base address
- Extra space for message buffer storage
- 1024 bytes for 64 message buffers

21.3.1 Memory Map

The complete memory map for a FlexCAN2 module with its 64 MBs is shown in Table 21-2. Except for the base addresses, the two FlexCAN2 modules have identical memory maps. Each individual register is identified by its complete name and the corresponding mnemonic.

Table 21-2. Module Memory Map

Address	Register Name	Register Description	Bits
Base = 0xFFFC_0000 (FlexCAN A) Base = 0xFFFC_8000 (FlexCAN C)	CAN*x*_MCR	Module configuration register	32
Base + 0x0004	CAN*x*_CR	Control register	32
Base + 0x0008	CAN*x*_TIMER	Free running timer	32
Base + 0x000C	—	Reserved	—
Base + 0x0010	CAN*x*_RXGMASK	RX global mask	32
Base + 0x0014	CAN*x*_RX14MASK	RX buffer 14 mask	32
Base + 0x0018	CAN*x*_RX15MASK	RX buffer 15 mask	32
Base + 0x001C	CAN*x*_ECR	Error counter register	32
Base + 0x0020	CAN*x*_ESR	Error and status register	32
Base + 0x0024	CAN*x*_IMRH	Interrupt masks high register	32
Base + 0x0028	CAN*x*_IMRL	Interrupt masks low register	32
Base + 0x002C	CAN*x*_IFRH	Interrupt flags high register	32
Base + 0x0030	CAN*x*_IFRL	Interrupt flags low register	32
Base + (0x0034–0x005F)	—	Reserved	—
Base + (0x0060–0x007F)	—	Reserved	—
Base + (0x0080–0x017F)	MB0–MB15	Message buffers 0–15	128 per buffer
Base + (0x0180–0x027F)	MB16–MB31	Message buffers 16–31	128 per buffer
Base + (0x0280–0x047F)	MB32–MB63	Message buffers 32–63	128 per buffer
Base + (0x0880–0x08BF)	CANx_RXIMR0–CANx_RXIMR15	RX individual mask register 0–15	32
Base + (0x08C0–0x08FF)	CANx_RXIMR16–CANx_RXIMR31	RX individual mask register 16–31	32
Base + (0x0900–0x097F)	CANx_RXIMR32–CANx_RXIMR63	RX individual mask register 32–63	32

The FlexCAN2 module stores CAN messages for transmission and reception using a message buffer structure. Each individual MB is formed by 16 bytes of memory mapped as described in Table 21-3. The FlexCAN2 module can manage up to 64 message buffers.

Table 21-3 shows the standard and extended message buffer (MB0) memory map, using 16 bytes (0x80–0x8F) total space.

Table 21-3. Message Buffer MB0 Memory Mapping

Address Offset	MB Field
0x80	Control and status (C/S)
0x84	Identifier field
0x88–0x8F	Data fields 0–7 (1 byte each)

NOTE

Reading the control and status word (first word) of a message buffer locks it from receiving further messages until it is unlocked by reading: another message buffer, or the timer.

21.3.2 Message Buffer Structure

The message buffer structure used by the FlexCAN2 module is represented in Figure 21-2. Both extended and standard frames (29-bit and 11-bit identifier, respectively) used in the CAN specification (version 2.0 Part B) are represented.

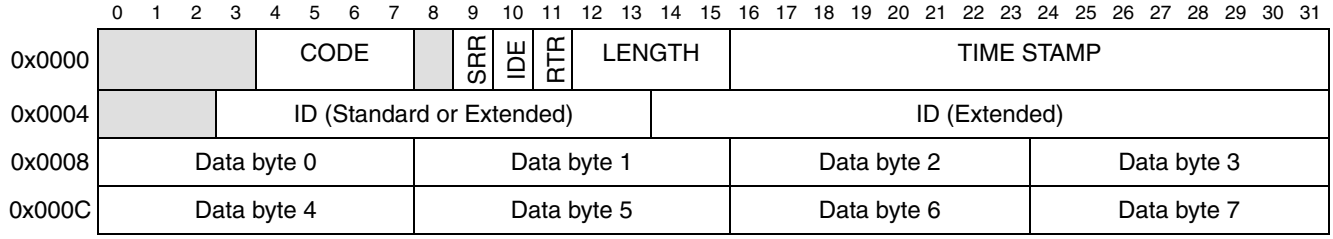


Figure 21-2. Message Buffer Structure

Table 21-4. Message Buffer Field Descriptions

Word	Bits and Field Name	Description
0x0000	0–3	Reserved
	4–7 CODE	Message buffer code. This 4-bit field can be accessed (read or write) by the CPU and by the FlexCAN2 module itself, as part of the message buffer matching and arbitration process. The encoding is shown in Table 21-5 and Table 21-6. See Section 21.4, “Functional Description,” for additional information.
	8	Reserved
	9 SRR	Substitute remote request. Fixed recessive bit, used only in extended format. You must set the SSR bit to ‘1’ for transmission (TX Buffers) and is stored with the value received on the CAN bus for RX receiving buffers. It can be received as either recessive or dominant. If FlexCAN2 receives this bit as dominant, then it is interpreted as arbitration loss. 0 Dominant is not a valid value for transmission in extended format frames 1 Recessive value is compulsory for transmission in extended format frames

Table 21-4. Message Buffer Field Descriptions (continued)

Word	Bits and Field Name	Description
0x0000 (continued)	10 IDE	ID extended bit. This bit identifies whether the frame format is standard or extended. 0 Frame format is standard 1 Frame format is extended
	11 RTR	Remote transmission request. This bit is used for requesting transmissions of a data frame. If FlexCAN2 transmits this bit as '1' (recessive) and receives it as '0' (dominant), it is interpreted as arbitration loss. If this bit is transmitted as '0' (dominant), then if it is received as '1' (recessive), the FlexCAN2 module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 0 Indicates the current MB has a data frame to be transmitted 1 Indicates the current MB has a remote frame to be transmitted
	12–15 LENGTH	Length of data in bytes. This 4-bit field is the length (in bytes) of the RX or TX data, which is located in offset 0x8 through 0xF of the MB space (see Figure 21-2). In reception, this field is written by the FlexCAN2 module, copied from the DLC (data length code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR = 1, the Frame to be transmitted is a remote frame and does not include the data field, regardless of the length field.
	16–31 TIME STAMP	Free-running counter time stamp. This 16-bit field is a copy of the free-running timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.
0x0004	0–2	Reserved
	3–13, 14–31 ID	Frame identifier. • Standard frame format: the 11 (3:13) most significant bits (MSB) are the frame ID in receive and transmit frames. The 18 (14:31) least significant bits (LSB) are ignored. • Extended frame format: all bits are the frame ID in receive and transmit frames.
0x0008 and 0x000C	DATA Bytes 0–3 Bytes 4–7	Data field. Up to eight bytes can be used for a data frame. For RX frames, the data is stored as it is received from the CAN bus. For TX frames, the CPU prepares the data field to be transmitted within the frame.

Table 21-5. Message Buffer Codes for RX Buffers

RX Code before RX Frame	Description	RX Code after RX Frame	Comment
0000	NOT ACTIVE: MB is not active.	—	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.

Table 21-5. Message Buffer Codes for RX Buffers (continued)

RX Code before RX Frame	Description	RX Code after RX Frame	Comment
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. See Section 21.4.3.1, “Matching Process for details about overrun behavior.
0110	OVERRUN: A frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB is overwritten again, and the code remains OVERRUN. See Section 21.4.3.1, “Matching Process for details about overrun behavior.
0XY1[1]	BUSY: FlexCAN is updating the contents of the MB. The CPU must not access the MB.	0010	EMPTY buffer was written with a new frame (XY was 01).
		0110	FULL/OVERRUN buffer was overwritten (XY was 11).

1 For TX message buffers (see Table 21-6), the BUSY bit must be ignored when read.

Table 21-6. Message Buffer Code for TX Buffers

RTR	Initial TX Code	Code after Successful Transmission	Description
X	1000	—	INACTIVE: MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes and RX MB with the same ID.
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the CODE field is automatically updated to ‘1110’ to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to ‘1010’ to restart the process again.
0	1110	1010	The MBM generates this code as a result of match to a remote request frame. The data frame is transmitted unconditionally once, and then the code automatically returns to ‘1010’. The CPU can write the code with the same effect.

21.3.3 Register Descriptions

The FlexCAN2 registers are described in this section. There are two separate, identical FlexCAN2 modules: A, B, C, and D. In the following sections, the '*x*' in the registers' names represents the individual module.

21.3.3.1 Module Configuration Register (CAN*x*_MCR)

CAN*x*_MCR defines global system configurations, such as the module operation mode and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which can be changed only while the module is in freeze mode.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FRZ	0	HALT	NOTRDY	0	SOFTRST	FRZACK	1	0	WRN EN	MDISACK	0	0	SRX DIS	MBFEN
W																
Reset	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	MAXMB					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 21-3. Module Configuration Register (CAN*x*_MCR)

Table 21-7. CAN*x*_MCR Field Descriptions

Field	Description
0 MDIS	Module disable. Controls whether FlexCAN2 is enabled or not. When disabled, FlexCAN2 shuts down the clock to the CAN protocol interface and message buffer management submodules. This is the only bit in CAN*x*_MCR not affected by soft reset. See Section 21.4.6.2, "Module Disabled Mode," for more information. 0 Enable the FlexCAN2 module 1 Disable the FlexCAN2 module
1 FRZ	Freeze enable. Specifies the FlexCAN2 behavior when the HALT bit in the CAN*x*_MCR is set or when debug mode is requested at MCU level. When FRZ is asserted, FlexCAN2 is enabled to enter freeze mode. Negation of this bit field causes FlexCAN2 to exit from freeze mode. 0 Not enabled to enter freeze mode 1 Enabled to enter freeze mode
2	Reserved
3 HALT	Halt FlexCAN. Assertion of this bit puts the FlexCAN2 module into freeze mode if FRZ is asserted. The CPU must clear it after initializing the message buffers and CAN*x*_CR. If FRZ is set, no reception or transmission is performed by FlexCAN2 before this bit is cleared. While in freeze mode, the CPU has write access to the CAN*x*_ECR, that is otherwise read-only. Freeze mode cannot be entered while FlexCAN2 is disabled. See Section 21.4.6.1, "Freeze Mode," for more information. 0 No freeze mode request. 1 Enters freeze mode if the FRZ bit is asserted.
4 NOTRDY	FlexCAN2 not ready. Indicates that FlexCAN2 is either disabled or in freeze mode. It is negated after FlexCAN2 has exited these modes. 0 FlexCAN2 module is either in normal mode, listen-only mode or loop-back mode 1 FlexCAN2 module is either disabled or freeze mode

Table 21-7. CAN*x*_MCR Field Descriptions (continued)

Field	Description
5	Reserved
6 SOFTRST	Soft reset. When asserted, FlexCAN2 resets its internal state machines and some of the memory-mapped registers. The following registers are affected by soft reset: • CAN*x*_MCR (except the MDIS bit) • CAN*x*_TIMER • CAN*x*_ECR • CAN*x*_ESR • CAN*x*_IMRL • CAN*x*_IMRH • CAN*x*_IFRL • CAN*x*_IFRH Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected: • CANx_CR • CAN*x*_RXGMASK • CAN*x*_RX14MASK • CAN*x*_RX15MASK • all Message buffers The SOFTRST bit can be asserted directly by the CPU when it writes to the CAN*x*_MCR, but it is also asserted when global soft reset is requested at MCU level. Because soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it can take time to fully propagate its effect. The SOFTRST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed. 0 No reset request 1 Resets values in registers indicated above.
7 FRZACK	Freeze mode acknowledge. Indicates that FlexCAN2 is in freeze mode and its prescaler is stopped. The freeze mode request cannot be granted until current transmission and reception processes have finished. Therefore the software can poll the FRZACK bit to know when FlexCAN2 has actually entered freeze mode. If freeze mode request is negated, then this bit is negated after the FlexCAN2 prescaler is running again. If freeze mode is requested while FlexCAN2 is disabled, then the FRZACK bit is set only after exiting the low power mode. See Section 21.4.6.1, “Freeze Mode,” for more information. 0 FlexCAN2 not in freeze mode, prescaler running 1 FlexCAN2 in freeze mode, prescaler stopped
8–9	Reserved
10 WRNEN	Warning interrupt enable. When asserted, this bit enables the generation of the TWRNINT and RWRNINT flags in the Error and Status Register. If WRNEN is negated, the TWRNINT and RWRNINT flags are always 0, independent of the values of the error counters, and no warning interrupt is generated. 1 = TWRNINT and RWRNINT bits are set when the respective error counter transition from less than 96 to greater than or equal to 96. 0 = TWRNINT and RWRNINT bits are zero, independent of the values in the error counters.
11 MDISACK	Low power mode acknowledge. Indicates whether FlexCAN2 is disabled. This cannot be performed until all current transmission and reception processes have finished, so the CPU can poll the MDISACK bit to know when FlexCAN2 has actually been disabled. See Section 21.4.6.2, “Module Disabled Mode,” for more information. 0 FlexCAN2 not disabled 1 FlexCAN2 is disabled
12–13	Reserved

Table 21-7. CAN*x*_MCR Field Descriptions (continued)

Field	Description
14 SRXDIS	This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module is not stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal is generated due to the frame reception. 1 = Self reception disabled 0 = Self reception enabled
15 MBFEN	Message buffer filter enable. This bit provides the capability of enabling either individual masking of every message buffer, or global masking of message buffers. By negating MBFEN, global masking is enabled and FlexCAN uses the Rx ID masking scheme of RXGMASK, RX14MASK and RX15MASK. MB14 and MB15 have individual masks and the others share the global mask. The scheme does not provide a reception queue; i.e. a received message always fills the first matching buffer, setting the CODE field to overrun if the buffer contained an unread message. See Section 21.3.3.4, “RX Mask Registers” for more information. Use global masking for compatibility with previous FlexCAN versions, which negates MBFEN at reset to retain compatibility with existing software. By asserting MBFEN, individual Rx ID masking and the reception queue features are enabled. In this scheme, individual receive mask registers (RXIM[0-63]) are provided for each MB. Upon receiving a message, FlexCAN searches the reception queue for the first empty matching MB. See Section 21.3.3.5, “RX Individual Mask Registers (CANx_RXIMR0 through CANx_RXIMR63)” and Section 21.4.3.2, “Reception Queue” for more information. 0 = Individual RX masking and reception queue features are disabled (thus the device is compatible with previous FlexCAN versions, i.e. one global mask register is used). 1 = Individual RX masking and reception queue features are enabled.
16–25	Reserved
26–31 MAXMB[0:5]	Maximum number of message buffers. This 6-bit field defines the maximum number of message buffers of the FlexCAN2 module. The reset value (0x0F) is equivalent to 16 MB configuration. FlexCAN must be in freeze mode before changing this value. Maximum MBs in use = MAXMB + 1 **Note:** MAXMB must be less than or equal to the number of available message buffers. FlexCAN2 cannot transmit or receive frames if this value is greater than the number of available message buffers.

21.3.3.2 Control Register (CAN*x*_CR)

CAN*x*_CR is defined for specific FlexCAN2 control features related to the CAN bus, such as bit-rate, programmable sampling point within an RX bit, loop-back mode, listen-only mode, bus off recovery behavior, and interrupt enabling (for example, bus-off, error). It also determines the division factor for the clock prescaler. BOFFMSK, ERRMSK, and BOFFREC bits can be accessed at any time. CANx_CR is unaffected by soft reset, which occurs when CAN_MCR[SOFTRST] is asserted.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRESDIV								RJW		PSEG1			PSEG2		
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BOFF MSK	ERR MSK	CLK_ SRC	LPB	TWRN MSK	RWRN MSK	0	0	SMP	BOFF REC	TSYN	LBUF	LOM	PROPSEG		
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] CAN*x*_CR is unaffected by soft reset (which occurs when CAN_MCR[SOFTRST] is asserted).

Figure 21-4. Control Register (CAN*x*_CR)

Table 21-8. CAN*x*_CR Field Descriptions

Bits	Description
0–7 PRESDIV[0:7]	Prescaler division factor. Defines the ratio between the CPI clock frequency and the serial clock (SCK) frequency. The SCK period defines the time quantum of the CAN protocol. For the reset value, the SCK frequency is equal to the CPI clock frequency. The maximum value of this register is 0xFF, that gives a minimum SCK frequency equal to the CPI clock frequency divided by 256. For more information, see Section 21.4.5.4, "Protocol Timing." $\text{S-clock frequency} = \frac{\text{CPI clock frequency}}{\text{PRESDIV} + 1}$
8–9 RJW[0:1]	Resync jump width. Defines the maximum number of time quanta[1] that a bit time can be changed by one re-synchronization. The valid programmable values are 0–3. $\text{Resync Jump Width} = \text{RJW} + 1$
10–12 PSEG1[0:2]	Phase segment 1. Defines the length of phase buffer segment 1 in the bit time. The valid programmable values are 0–7. $\text{Phase Buffer Segment 1} = (\text{PSEG1} + 1) \times \text{Time Quanta}$
13–15 PSEG2[0:2]	Phase segment 2. Defines the length of phase buffer segment 2 in the bit time. The valid programmable values are 1–7. $\text{Phase Buffer Segment 2} = (\text{PSEG2} + 1) \times \text{Time Quanta}$
16 BOFFMSK	Bus off mask. Provides a mask for the bus off interrupt. 0 Bus off interrupt disabled 1 Bus off interrupt enabled
17 ERRMSK	Error mask. Provides a mask for the error interrupt. 0 Error interrupt disabled 1 Error interrupt enabled
18 CLK_SRC	CAN engine clock source. Selects the clock source to the CAN Protocol Interface (CPI) to be either the system clock (driven by the PLL) or the crystal oscillator clock. The selected clock is fed into the prescaler to generate the serial clock (SCK). 0 = The CAN engine clock source is the oscillator clock 1 = The CAN engine clock source is the system clock

Table 21-8. CAN*x*_CR Field Descriptions (continued)

Bits	Description
19 LPB	Loop back. Configures FlexCAN2 to operate in loop-back mode. See Section 21.1.4, "Modes of Operation" for information about this operating mode. 0 Loop back disabled 1 Loop back enabled
20 TWRNMSK	This bit provides a mask for the TX Warning Interrupt associated with the TWRNINT flag in the Error and Status Register. This bit has no effect if the WRNEN bit in CAN*x*_MCR is negated and it is read as zero when WRNEN is negated. 1 = Tx Warning Interrupt enabled 0 = Tx Warning Interrupt disabled
21 RWRNMSK	This bit provides a mask for the RX Warning Interrupt associated with the RWRNINT flag in the Error and Status Register. This bit has no effect if the WRNEN bit in CAN*x*_MCR is negated and it is read as zero when WRNEN is negated. 1 = Rx Warning Interrupt enabled 0 = Rx Warning Interrupt disabled
22–23	Reserved
24 SMP	Sampling mode. Defines the sampling mode of each bit in the receiving messages (RX). 0 Just one sample is used to determine the RX bit value 1 Three samples are used to determine the value of the received bit: the regular one (sample point) and two preceding samples, a majority rule is used
25 BOFFREC	Bus off recovery mode. Defines how FlexCAN2 recovers from bus off state. If this bit is negated, automatic recovering from bus off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from bus off is disabled and the module remains in bus off state until you negate the bit. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then bus off recovery happens as if the BOFFREC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits are detected, then FlexCAN2 re-synchronizes to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFFREC bit can be re-asserted again during bus off, but it is only effective the next time the module enters bus off. If BOFFREC was negated when the module entered bus off, asserting it during bus off is not effective for the current bus off recovery. 0 Automatic recovering from bus off state enabled, according to CAN Spec 2.0 part B 1 Automatic recovering from bus off state disabled
26 TSYN	Timer sync mode. Enables a mechanism that resets the free-running timer each time a message is received in message buffer 0. This feature provides means to synchronize multiple FlexCAN2 stations with a special SYNC message (that is, global network time). 0 Timer sync feature disabled 1 Timer sync feature enabled **Note:** There is a possibility of 4–5 ticks count skew between the different FlexCAN2 stations that would operate in this mode.
27 LBUF	Lowest buffer transmitted first. This bit defines the ordering mechanism for message buffer transmission. 0 Buffer with lowest ID is transmitted first 1 Lowest number buffer is transmitted first

Table 21-8. CAN*x*_CR Field Descriptions (continued)

Bits	Description
28 LOM	Listen-only mode. Configures FlexCAN2 to operate in listen-only mode. In this mode, the FlexCAN2 module receives messages without giving any acknowledge. It is not possible to transmit any message in this mode. 0 FlexCAN2 module is in normal active operation, listen only mode is deactivated 1 FlexCAN2 module is in listen only mode operation
29–31 PROPSEG [0:2]	Propagation segment. Defines the length of the propagation segment in the bit time. The valid programmable values are 0–7. Propagation Segment Time = (PROPSEG + 1) × Time Quanta Time Quantum = one S clock period

1 One time quantum is equal to the S clock period.

21.3.3.3 Free Running Timer (CAN*x*_TIMER)

CAN*x*_TIMER represents a 16-bit free running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN2 bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During freeze mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the TIME STAMP entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. This operation is transparent to you, except for the time delay required for the data write to the register. Software can poll the register to verify that the data was written.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIMER															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-5. Free Running Timer (CAN*x*_TIMER)

21.3.3.4 RX Mask Registers

By negating the CAN*x*_MCR[MBFEN] bit, the CANx_RXGMASK, CANx_RX14MASK, and CANx_RX15MASK registers are used as acceptance masks for received frame IDs. Three masks are defined: a global mask, used for RX buffers 0–13 and 16–63, and two extra masks dedicated for buffers 14 and 15.

The meaning of each mask bit is the following:

- Mask bit = 0: the corresponding incoming ID bit is “don’t care.”
- Mask bit = 1: the corresponding ID bit is checked against the incoming ID bit, to see if a match exists.

These masks are used both for standard and extended ID formats. Do not change the value of mask registers while in normal operation. Locked frames which match a message buffer through a mask can be transferred into the message buffer (upon release) even if they no longer match. Table 21-9 shows some examples of ID masking for standard and extended message buffers.

Table 21-9. Mask Examples for Standard/Extended Message Buffers

Mask ID	Base ID ID28.....ID18	IDE	Extended ID ID17.....ID0	Match
MB2 ID	1 1 1 1 1 1 1 1 0 0 0	0		
MB3 ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB4 ID	0 0 0 0 0 0 1 1 1 1 1	0		
MB5 ID	0 0 0 0 0 0 1 1 1 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB14 ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
RX Global Mask	1 1 1 1 1 1 1 1 1 1 0		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1	
RX Msg in [1]	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	3
RX Msg in [2]	1 1 1 1 1 1 1 1 0 0 1	0		2
RX Msg in [3]	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0	
RX Msg in [4]	0 1 1 1 1 1 1 1 0 0 0	0		
RX Msg in [5]	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	14
RX 14 Mask	0 1 1 1 1 1 1 1 1 1 1		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0	
RX Msg in [6]	1 0 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
RX Msg in [7]	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	14

1 Match for Extended Format (message buffer 3).
2 Match for Standard Format. (message buffer 2).
3 Mismatch for message buffer 3 because of ID0.
4 Mismatch for message buffer 2 because of ID28.
5 Mismatch for message buffer 3 because of ID28, match for message buffer 14 (uses RX14MASK).
6 Mismatch for message buffer 14 because of ID27 (uses RX14MASK).
7 Match for message buffer 14 (uses RX14MASK).

21.3.3.4.1 RX Global Mask (CAN*x*_RXGMASK)

The RX global mask bits are applied to all RX identifiers excluding RX buffers 14 and 15, that have their specific RX mask registers. Access to this register is unrestricted. CANx_RXGMASK is unaffected by soft reset (which occurs when CAN_MCR[SOFTRST] asserts).

Address: Base + 0x0010 (CANx_RXGMASK)
Base + 0x0014 (CANx_RX14MASK)
Base + 0x0018 (CANx_RX15MASK)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
W																
Reset [1]	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
W																
Reset [1]	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

[1] CAN*x*_RXGMASK is unaffected by soft reset (which occurs when CAN_MCR[SOFTRST] is asserted).

Figure 21-6. RX Global Mask Register (CAN*x*_RXGMASK)

Table 21-10. CAN*x*_RXGMASK Field Descriptions

Field	Description
0–2	Reserved, must be cleared.
3–13 MI*n*	Standard ID mask bits. These bits are the same mask bits for the standard and extended formats.
14–31 MI*n*	Extended ID mask bits. These bits are used to mask comparison only in extended format.

21.3.3.4.2 RX 14 Mask (CAN*x*_RX14MASK)

The CAN*x*_RX14MASK register has the same structure as the RX global mask register and is used to mask message buffer 14. Access to this register is unrestricted. CANx_RX14MASK is unaffected by soft reset (which occurs when CAN_MCR[SOFTRST] is asserted).

- Address offset: 0x0014
- Reset value: 0x1FFF_FFFF

21.3.3.4.3 RX 15 Mask (CAN*x*_RX15MASK)

The CAN*x*_RX15MASK register has the same structure as the RX global mask register and is used to mask message buffer 15. Access to this register is unrestricted. CANx_RX15MASK is unaffected by soft reset (which occurs when CAN_MCR[SOFTRST] is asserted).

- Address offset: 0x0018
- Reset value: 0x1FFF_FFFF

21.3.3.5 RX Individual Mask Registers (CAN*x*_RXIMR0 through CAN*x*_RXIMR63)

By asserting the CAN*x*_MCR[MBFEN] bit, the CAN*x*_RXIMR[0–63] registers are used as acceptance masks for received frame IDs, in both standard and extended ID formats. One mask register is provided for each message buffer for individual ID masking per message buffer.

The meaning of each mask bit is the following:

- Mask bit = 0: The corresponding incoming ID bit is a “don’t care.”
- Mask bit = 1: The corresponding ID bit is checked against the incoming ID bit, to see if a match exists.

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in freeze mode. Out of freeze mode, write accesses are blocked and read accesses return all 0s. Furthermore, if the MBFEN bit in the MCR register is negated, any read or write operation to these RXIMR*n* registers results in access error.

Address: Base + 0x0880–0x097F Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
W																
Reset [1]	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
W																
Reset [1]	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

[1] The ‘U’ indicates the value is undefined after reset.

Figure 21-7. RX Individual Mask Registers (CANx_RXIMR0 through CANx_RXIMR63)

Table 21-11. CANx_RXIMR0–CANx_RXIMR63 Field Descriptions

Field	Description
0–2	Reserved
3–13 MI28–MI18	Standard ID mask bits. These bits are the same mask bits for the standard and extended formats.
14–31 MI17–MI0	Extended ID mask bits. These bits are used to mask comparison only in extended format.

21.3.3.6 Error Counter Register (CAN*x*_ECR)

CAN*x*_ECR has two 8-bit fields with the value of two FlexCAN2 error counters: the transmit error counter (TXECTR field) and receive error counter (RXECTR field). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN2 module. Both counters are read only except in freeze mode, where they can be written by the CPU.

Writing to the CAN*x*_ECR while in freeze mode is an indirect operation. The data is first written to an auxiliary register, and then an internal request/acknowledge procedure across clock domains is executed. This occurs internally without indication, except for the time delay required for the data write to the register. Software can poll the register to verify that the data is written.

FlexCAN2 responds to any bus state as described in the protocol: transmitting, for example, an ‘error active’ or ‘error passive’ flag, delaying its transmission start time (‘error passive’), and avoiding any influence on the bus when in the bus off state. The following are the basic rules for FlexCAN2 bus state transitions:

- If the value of TXECTR or RXECTR increases to be greater than or equal to 128, the FLTCONF field in the CAN*x*_ESR is updated to reflect the ‘error passive’ state.
- If the FlexCAN2 state is ‘error passive,’ and either TXECTR or RXECTR decrements to a value less than or equal to 127 while the other already satisfies this condition, the FLTCONF field in the CAN*x*_ESR is updated to reflect the ‘error active’ state.
- If the value of TXECTR increases to greater than 255, the FLTCONF field in the CAN*x*_ESR is updated to reflect the bus off state, and can issue an interrupt. The value of TXECTR is then reset to zero.
- If FlexCAN2 is in the bus off state, then TXECTR is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, TXECTR is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TXECTR. When TXECTR reaches the value of 128, the FLTCONF field in CAN*x*_ESR is updated to be ‘error active’ and both error counters are reset to zero. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the TXECTR value.
- If during system start-up, only one node is operating, then its TXECTR increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ACKERR bit in CAN*x*_ESR). After the transition to the ‘error passive’ state, the TXECTR does not increment anymore by acknowledge errors. Therefore the device never goes to the bus off state.
- If the RXECTR increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to ‘error active’ state.

Address: Base + 0x001C Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RXECTR								TXECTR							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-8. Error Counter Register (CAN*x*_ECR)

21.3.3.7 Error and Status Register (CAN*x*_ESR)

CAN*x*_ESR reflects various error conditions, some general status of the device, and it is the source of two interrupts to the CPU. The reported error conditions (bits 16–21) are those that occurred since the last time the CPU read this register. The CPU read action clears BIT1ERR, BIT0ERR, ACKERR, CRCERR, FRMERR, and STFERR. TXWRN, RXWRN, IDLE, TXRX, FLTCONF, BOFFINT, and ERRINT are status bits.

Most bits in this register are read-only, except BOFFINT, ERRINT, TWRNINT, and RWRNINT which are interrupt flags that can be cleared by writing 1 to them (writing 0 has no effect).

See Section 21.4.7, “Interrupts,” for more details.

NOTE

A read clears BIT1ERR, BIT0ERR, ACKERR, CRCERR, FRMERR, and STFERR, therefore these bits must not be read speculatively. For future compatibility, the TLB entry covering the CAN*x*_ESR must be configured to be guarded.

Address: Base + 0x0020 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TWRN INT	RWRN INT
W															w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BIT1 ERR	BIT0 ERR	ACK ERR	CRC ERR	FRM ERR	STF ERR	TX WRN	RX WRN	IDLE	TXRX	FLTCONF		0	BOFF INT	ERR INT	0
W														w1c	w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-9. Error and Status Register (CAN*x*_ESR)

Table 21-12. CAN*x*_ESR Field Descriptions

Field	Description
0–13	Reserved
14 TWRNINT	If the WRNEN bit in CAN*x*_MCR is asserted, the TWRNINT bit is set when the TXWRN flag transitions from 0 to 1, meaning that the TX error counter reached 96. If the corresponding mask bit in the Control Register (TWRNMSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing to 1. Writing 0 has no effect. 0 No such occurrence 1 TXECTR ≥ 96

Table 21-12. CAN*x*_ESR Field Descriptions (continued)

Field	Description
15 RWRNINT	If the WRNEN bit in CAN*x*_MCR is asserted, the RWRNINT bit is set when the RXWRN flag transitions from 0 to 1, meaning that the RX error counter reached 96. If the corresponding mask bit in the Control Register (RWRNMSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing to 1. Writing 0 has no effect. 0 No such occurrence 1 RXECTR ≥ 96
16 BIT1ERR	Bit 1 error. Indicates when an inconsistency occurs between the transmitted and the received message in a bit. A read clears BIT1ERR. 0 No such occurrence 1 At least one bit sent as recessive is received as dominant **Note:** This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.
17 BIT0ERR	Bit 0 error. Indicates when an inconsistency occurs between the transmitted and the received message in a bit. A read clears BIT0ERR. 0 No such occurrence 1 At least one bit sent as dominant is received as recessive
18 ACKERR	Acknowledge error. Indicates that an acknowledge error has been detected by the transmitter node; that is, a dominant bit has not been detected during the ACK SLOT. A read clears ACKERR. 0 No such occurrence 1 An ACK error occurred since last read of this register
19 CRCERR	Cyclic redundancy code error. Indicates that a CRC error has been detected by the receiver node; that is, the calculated CRC is different from the received. A read clears CRCERR. 0 No such occurrence 1 A CRC error occurred since last read of this register.
20 FRMERR	Form error. Indicates that a form error has been detected by the receiver node; that is, a fixed-form bit field contains at least one illegal bit. A read clears FRMERR. 0 No such occurrence 1 A form error occurred since last read of this register
21 STFERR	Stuffing error. Indicates that a stuffing error has been detected. A read clears STFERR. 0 No such occurrence. 1 A stuffing error occurred since last read of this register.
22 TXWRN	TX error counter. This status bit indicates that repetitive errors are occurring during message transmission. 0 No such occurrence 1 TXECTR ≥ 96
23 RXWRN	RX error counter. This status bit indicates when repetitive errors are occurring during messages reception. 0 No such occurrence 1 RXECTR ≥ 96
24 IDLE	CAN bus IDLE state. This status bit indicates when CAN bus is in IDLE state. 0 No such occurrence 1 CAN bus is now IDLE
25 TXRX	Current FlexCAN2 status (transmitting/receiving). This status bit indicates if FlexCAN2 is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 0 FlexCAN2 is receiving a message (IDLE = 0) 1 FlexCAN2 is transmitting a message (IDLE = 0)

Table 21-12. CAN*x*_ESR Field Descriptions (continued)

Field	Description
26–27 FLTCONF[0:1]	Fault confinement state. This status bit indicates the confinement state of the FlexCAN2 module. If the LOM bit in the CAN*x*_CR asserts, the FLTCONF field indicates “Error Passive”. Since the CAN*x*_CR is not affected by a soft reset, the FLTCONF field is not affected by a soft reset if the LOM bit asserts. 00 Error active 01 Error passive 1X Bus off
28	Reserved
29 BOFFINT	Bus off interrupt. This status bit is set when FlexCAN2 is in the bus off state. If CAN*x*_CR[BOFFMSK] is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence 1 FlexCAN2 module is in ‘Bus Off’ state
30 ERRINT	Error interrupt. This status bit indicates that at least one of the error bits (bits 16-21) is set. If CAN*x*_CR[ERRMSK] is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence 1 Indicates setting of any error bit in the CAN*x*_ESR
31	Reserved

21.3.3.8 Interrupt Masks High Register (CAN*x*_IMRH)

CAN*x*_IMRH allows any number of a range of 32 message buffer interrupts to be enabled or disabled. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (that is, when the corresponding IFRH bit is set).

Address: Base + 0x0024 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63M	BUF 62M	BUF 61M	BUF 60M	BUF 59M	BUF 58M	BUF 57M	BUF 56M	BUF 55M	BUF 54M	BUF 53M	BUF 52M	BUF 51M	BUF 50M	BUF 49M	BUF 48M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 47M	BUF 46M	BUF 45M	BUF 44M	BUF 43M	BUF 42M	BUF 41M	BUF 40M	BUF 39M	BUF 38M	BUF 37M	BUF 36M	BUF 35M	BUF 34M	BUF 33M	BUF 32M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-10. Interrupt Masks High Register (CAN*x*_IMRH)

Table 21-13. CAN*x*_IMRH Field Descriptions

Field	Description
0–31 BUF*n*M	Message buffer *n* mask. Enables or disables the respective FlexCAN2 message buffer (MB63 to MB32) Interrupt. 0 The corresponding buffer Interrupt is disabled 1 The corresponding buffer Interrupt is enabled **Note:** Setting or clearing a bit in the IMRH register can assert or negate an interrupt request, respectively.

21.3.3.9 Interrupt Masks Low Register (CAN*x*_IMRL)

CAN*x*_IMRL allows enabling or disabling any number of a range of 32 message buffer interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (that is, when the corresponding IFRL bit is set).

Address: Base + 0x0028 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31M	BUF 30M	BUF 29M	BUF 28M	BUF 27M	BUF 26M	BUF 25M	BUF 24M	BUF 23M	BUF 22M	BUF 21M	BUF 20M	BUF 19M	BUF 18M	BUF 17M	BUF 16M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15M	BUF 14M	BUF 13M	BUF 12M	BUF 11M	BUF 10M	BUF 09M	BUF 08M	BUF 07M	BUF 06M	BUF 05M	BUF 04M	BUF 03M	BUF 02M	BUF 01M	BUF 00M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-11. Interrupt Masks Low Register (CAN*x*_IMRL)

Table 21-14. CAN*x*_IMRL Field Descriptions

Field	Description
0–31 BUF*n*M	Message buffer *n* mask. Enables or disables the respective FlexCAN2 message buffer (MB31 to MB0) Interrupt. 0 The corresponding buffer Interrupt is disabled 1 The corresponding buffer Interrupt is enabled **Note:** Setting or clearing a bit in the IMRL register can assert or negate an interrupt request, respectively.

21.3.3.10 Interrupt Flags High Register (CAN*x*_IFRH)

CAN*x*_IFRH defines the flags for 32 message buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFRH bit. If the corresponding IMRH bit is set, an interrupt is generated. Write a 1 to the interrupt flag to clear its value to zero. Writing a 0 has no effect.

Address: Base + 0x002C Access: User R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BUF 63I	BUF 62I	BUF 61I	BUF 60I	BUF 59I	BUF 58I	BUF 57I	BUF 56I	BUF 55I	BUF 54I	BUF 53I	BUF 52I	BUF 51I	BUF 50I	BUF 49I	BUF 48I
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BUF 47I	BUF 46I	BUF 45I	BUF 44I	BUF 43I	BUF 42I	BUF 41I	BUF 40I	BUF 39I	BUF 38I	BUF 37I	BUF 36I	BUF 35I	BUF 34I	BUF 33I	BUF 32I
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-12. Interrupt Flags High Register (CAN*x*_IFRH)

Table 21-15. CAN*x*_IFRH Field Descriptions

Field	Description
0–31 BUF*n*I	Message buffer *n* interrupt. Each bit represents the respective FlexCAN2 message buffer (MB63–MB32) interrupt. Write 1 to clear. 0 No such occurrence 1 The corresponding buffer has successfully completed transmission or reception.

21.3.3.11 Interrupt Flags Low Register (CAN*x*_IFRL)

CAN*x*_IFRL defines the flags for 32 message buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFRL bit. If the corresponding IMRL bit is set, an interrupt is generated. Write a 1 to the interrupt flag to clear its value to zero. Writing 0 has no effect.

Address: Base + 0x0030 Access: User R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BUF 31I	BUF 30I	BUF 29I	BUF 28I	BUF 27I	BUF 26I	BUF 25	BUF 24I	BUF 23I	BUF 22I	BUF 21I	BUF 20I	BUF 19I	BUF 18I	BUF 17I	BUF 16I
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BUF 15I	BUF 14I	BUF 13I	BUF 12I	BUF 11I	BUF 10I	BUF 09I	BUF 08I	BUF 07I	BUF 06I	BUF 05I	BUF 04I	BUF 03I	BUF 02I	BUF 01I	BUF 00I
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-13. Interrupt Flags Low Register (CAN*x*_IFRL)

Table 21-16. CAN*x*_IFRL Field Descriptions

Field	Description
0–31 BUF*n*I	Message buffer *n* interrupt. Each bit represents the respective FlexCAN2 message buffer (MB31 to MB0) interrupt. Write 1 to clear. 0 No such occurrence 1 The corresponding buffer has successfully completed transmission or reception.

21.4 Functional Description

21.4.1 Overview

The FlexCAN2 module is a CAN protocol engine with a very flexible message buffer configuration scheme. The module can have up to 64 message buffers, any of which can be assigned as either a TX buffer or an RX buffer. Each message buffer has an assigned interrupt flag to indicate successful completion of transmission or reception.

21.4.2 Transmit Process

The CPU prepares a message buffer for transmission by executing the following steps:

1. Write the CODE field of the control and status word to keep the TX MB inactive (code = 1000).
2. Write the ID word.
3. Write the DATA bytes.
4. Write the LENGTH, SRR, IDE, RTR, and CODE fields of the control and status word to activate the TX MB.

The first and last steps are mandatory.

21.4.2.1 Arbitration Process

This process selects the next MB to transmit. All MBs programmed as transmit buffers are scanned to find the lowest ID[1] or the lowest MB number, depending on the LBUF bit in the CAN*x*_CR. The selected MB is transferred to an internal serial message buffer (SMB), which is not software accessible, and then transmitted. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During Intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished.
- When MBM is in idle or bus off state and the CPU writes to the C/S word of any MB
- Upon leaving freeze mode

When arbitration is complete, and an MB is selected to transmit, the frame is first transferred to the SMB. This is called ‘move out.’ After move out, the CAN transmit machine starts to transmit the frame according to the CAN protocol rules. FlexCAN2 transmits up to eight data bytes, even if the value of the data length code (DLC) is greater.

At the end of a successful transmission, the value of the free running timer at the beginning of the identifier field is written into the TIME STAMP field in the MB, the CODE field in the control and status word of the MB is updated, a status flag is set in CAN*x*_IFRL or CAN*x*_IFRH, and an MB interrupt is generated if allowed by the corresponding interrupt mask register bit.

1. If LBUF is negated, arbitration uses the ID, RTR and IDE bits inside the ID in the same positions they are transmitted in the CAN frame.

21.4.3 Receive Process

The CPU prepares a message buffer for frame reception by executing the following steps:

1. Write the CODE field of the control and status word to keep the RX MB as INACTIVE (CODE = 0000).
2. Write the ID word.
3. Write the CODE field of the control and status word to mark the MB as EMPTY.

The first and last steps are mandatory.

21.4.3.1 Matching Process

The matching process compares the IDs of all active RX message buffers to newly received frames, so that, if a match occurs, a newly received frame is transferred (moved in) to the first (that is, lowest entry) matching MB when the reception queue feature is disabled. Only MBs marked as EMPTY, FULL, or OVERRUN participate in the internal matching process at the CRC frame field. The internal matching process takes place every time the receiver receives an error free frame.

The value of the free running timer is written into the TIME STAMP field in the MB. The ID field, the DATA field (8 bytes at most), and the LENGTH field are stored, the CODE field is updated, and a status flag is set in CAN*x*_IFRL or CAN*x*_IFRH, and an interrupt is generated if the corresponding interrupt mask is enabled in CAN*x*_IMRL/H.

The CPU must read the following fields in an RX frame from its MB:

- Control and status word (mandatory, activates internal lock for this buffer)
- ID (optional, needed only if a mask was used)
- DATA field words
- Free running timer (optional, releases internal lock)

Reading the free running timer is not mandatory. If not executed, the MB remains locked, unless the CPU starts reading another MB. Only a single MB is locked at a time. The only mandatory CPU read operation is of the control and status word, to assure data coherency. If the BUSY bit is set in the CODE field, then the CPU must defer the access to the MB until this bit is negated.

The CPU must synchronize to frame reception by the status flag bit for the specific MB in one of the CANx_IFRH and CANx_IFRL registers and not by the control and status word code field of that MB. Polling the CODE field does not work because after a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the CODE field does not return to EMPTY. It remains FULL, as explained in Table 21-5. If the CPU tries to work around this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is deactivated from any current matching process. As a result, a newly received frame matching the ID of that MB can be lost. Never poll by directly reading the C/S word of the MBs. Instead, read the CANx_IFRH and CANx_IFRL registers.

The received ID field is always stored in the matching MB, thus the contents of the ID field in a MB can change if the match was due to mask.

21.4.3.2 Reception Queue

A queue of received messages can be implemented that allows the CPU more time for servicing MBs. By programming more than one MB with the same ID, received messages are queued into the MBs. Matching to a range of IDs is possible by using ID acceptance masks that mask individual MBs. During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is a don't care.

Suppose, for example, that the second and fifth MBs in an array have the same ID, and FlexCAN starts receiving messages with that ID. When FlexCAN receives the first message, the matching algorithm matches it to MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm finds MB number 2 again, but it is not free to receive[1]. Therefore, the matching process continues, finds MB number 5, and stores the message there. If yet another message with the same ID arrives, the matching algorithm determines that no matching MBs are free to receive the data, and overwrites the last matched MB (number 5). When this occurs, the code field of the MB is set to OVERRUN.

A reception queue is built that orders the messages according to the value in the time stamp field that is read by the CPU. This functionality is set by asserting the CAN*x*_MCR[MBFEN] bit. The RXIMR*n* registers are not initialized out of reset, since they reside in RAM and can only be programmed if the MBFEN bit is asserted while the module is in freeze mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (CAN*x*_RXGMASK, CAN*x*_RX14MASK, and CAN*x*_RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when CAN*x*_MCR[MBFEN] is negated.

See Section 21.3.3.4, "RX Mask Registers."

21.4.3.3 Self Received Frames

FlexCAN2 receives frames transmitted by itself if there exists an RX matching MB, but only if an ACK is generated by an external node or if loop-back mode is enabled. Note also that FlexCAN does receive frames transmitted by itself if there exists an RX matching MB, provided the MCR[SRXDIS] bit is not asserted. If SRXDIS is asserted, FlexCAN does not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal is generated due to the frame reception.

21.4.4 Message Buffer Handling

To maintain data coherency and FlexCAN2 proper operation, the CPU must obey the rules described in Section 21.4.2, "Transmit Process," and Section 21.4.3, "Receive Process." Any form of CPU accessing a MB structure within FlexCAN2 other than those specified can cause FlexCAN2 to behave in an unpredictable way.

Deactivation of a message buffer is a CPU action that causes that MB to be excluded from FlexCAN2 transmit or receive processes during the current match/arbitration round. Any CPU write access to a control and status word of the MB structure deactivates that MB, excluding it from the current RX/TX

1. If, however, the CPU has read the MB2 data and released it before the next matching process at the CRC frame, then, even if the MB2 RX code is FULL, the MB2 is free to receive and the message is stored in MB2 rather than in MB5.

process. However, deactivation is not permanent. The MB that was deactivated during the current match/arbitration pass is available to transmit or receive in the next pass.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data coherency in the MB can become corrupt, therefore that MB is deactivated.

Match and arbitration are one-pass processes. After scanning all MBs, a winner is determined. If MBs are changed after they are scanned, no re-evaluation is done to determine a new match/winner; and a frame can be lost if the matched MB has been deactivated. If two RX MBs have a matching ID to a received frame, then it is not guaranteed reception if the matching MB is deactivated after FlexCAN2 has scanned the second.

21.4.4.1 Notes on TX Message Buffer Deactivation

There is a point in time until which the deactivation of a TX MB causes it not to be transmitted (end of move out). After this point, it is transmitted but no interrupt is issued and the CODE field is not updated.

If a TX MB containing the lowest ID (or lowest buffer if LBUF is set) is deactivated after FlexCAN2 has scanned it while in arbitration process, then FlexCAN2 can transmit a MB with an ID that is not the lowest at that time.

21.4.4.2 Notes on RX Message Buffer Deactivation

If the deactivation occurs during move in, the move operation is aborted and no interrupt is issued, but the MB contains mixed data from two different frames.

In case the CPU writes data into RX MB data words while it is being moved in, the move operation is aborted and no interrupt is issued, but the control/status word can change to reflect FULL or OVRN. This action must be avoided.

21.4.4.3 Data Coherency Mechanisms

The FlexCAN2 module has a mechanism to assure data coherency in both receive and transmit processes. The mechanism includes a lock status for MBs and two internal storage areas, called serial message buffers (SMB), to buffer frame transfers within FlexCAN. The details of the mechanism are the following:

- CPU reads the control and status word of an MB, which triggers a lock for that MB; therefore a new RX frame that matches the MB cannot be written to the MB.
- To release a locked MB, the CPU must either lock another MB (by reading its control and status word), or globally release any locked MB (by reading the free-running timer).
- If the CPU reads a RX MB while it is being transferred to (from) SMB, then the BUSY bit is set in the CODE field of the control and status word. To ensure data coherency, the CPU must wait until this BUSY bit is negated before further reading from that MB. In this case, the MB is not locked.
- If the CPU deactivates a locked RX MB, then its lock status is negated, but no data is transferred to the MB.

The following bullets apply only if:

- reception queue is disabled,
- no other matching buffers in the reception queue,
- or for the last available queue element when all other MBs are not free to receive (the last queue element is overwritten in this manner for a single MB in non-queue mode).

If the reception queue is enabled, the state machine searches for the next matching message buffer.

- If while an MB is locked and an RX frame with a matching ID is received, the RX frame cannot be stored in the MB and remains in the SMB. The CAN*x*_ESR does not indicate the RX frame remained in the SMB.
- If while a MB is locked when two or more RX frames with matching ID are received, the last RX frame remains in the SMB, while all preceding RX frames are lost. The CAN*x*_ESR does not indicate that the preceding frames are discarded.
- If a locked MB is released and a matching frame exists in the SMB, the frame is transferred to the matching MB.

21.4.5 CAN Protocol Related Features

21.4.5.1 Remote Frames

A remote frame is a special kind of frame. You can program an MB to be a request remote frame by writing the MB as transmit with the RTR bit set to 1. After the remote request frame is transmitted successfully, the MB becomes a receive message buffer, with the same ID as before.

When a remote request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the CODE field ‘1010’. If there is a matching ID, then the MB frame is transmitted. If the matching MB has the RTR bit set, then FlexCAN2 transmits a remote frame as a response.

A received remote request frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame must match.

In the case that a remote request frame was received and matched a MB, this message buffer immediately enters the internal arbitration process, but is considered as normal TX MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

21.4.5.2 Overload Frames

FlexCAN2 transmits overload frames if the dominant bit satisfies any of the following conditions:

- First or second bit of intermission
- Seventh (last) bit of the end-of-frame field in RX frames
- Eighth bit (last) of error frame delimiter or overload frame delimiter

21.4.5.3 Time Stamp

The value of the free running timer is sampled at the beginning of the identifier field on the CAN bus, and is stored at the end of ‘move in’ in the TIME STAMP field, providing network activity with respect to time.

The free running timer can be reset upon a specific frame reception, enabling network time synchronization. See TSYN description in Section 21.3.3.2, “Control Register (CANx_CR).”

21.4.5.4 Protocol Timing

The clock source to the CAN protocol interface (CPI) can be either the system clock or a direct feed from the oscillator pin EXTAL. The clock source is selected by the CLK_SRC bit in the CAN_CR. The clock is fed to the prescaler to generate the serial clock (SCK).

The FlexCAN2 module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The CAN*x*_CR has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW. See Section 21.3.3.2, “Control Register (CANx_CR).”

The PRESDIV field controls a prescaler that generates the serial clock (SCK), whose period defines the ‘time quantum’ used to compose the CAN waveform. A time quantum is the atomic unit of time handled by FlexCAN.

$$f_{Tq} = \frac{f_{CANCLK}}{\text{Prescaler Value}}$$

A bit time is subdivided into three segments[1] (see Figure 21-14 and Table 21-17):

- SYNCSEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time segment 1: This segment includes the propagation segment and the phase segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CTRL register so that their sum (plus 2) is in the range of 4 to 16 time quanta.
- Time segment 2: This segment represents the phase segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CTRL register (plus 1) to be 2 to 8 time quanta long.

$$\text{Bit rate} = \frac{f_{Tq}}{(\text{Number of Time Quanta})}$$

1. For further explanation of the underlying concepts see ISO/DIS 11519–1, Section 10.3. See also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

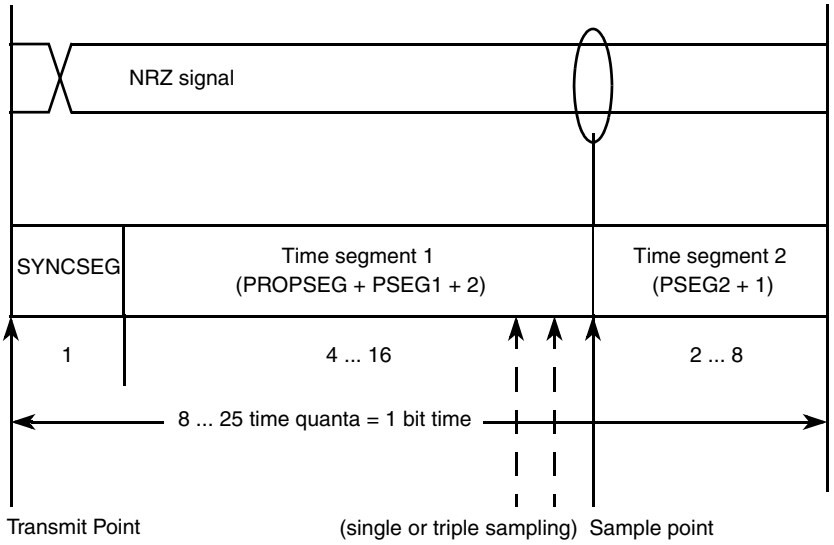


Figure 21-14. Segments within the Bit Time

Table 21-17 describes the time segment syntax:

Table 21-17. Time Segment Syntax

Syntax	Description
SYNCSEG	System expects transitions to occur on the bus during this period.
Transmit point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample point	A node in receive mode samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 21-18 gives an overview of the CAN compliant segment settings and the related parameter values.

NOTE

Ensure the bit time settings are in compliance with the CAN standard.

Table 21-18. CAN Standard Compliant Bit Time Segment Settings

Time Segment 1	Time Segment 2	Resynchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

21.4.5.5 Arbitration and Matching Timing

During normal transmission or reception of frames, the arbitration, match, move in and move out processes are executed during certain time windows inside the CAN frame, as shown in Figure 21-15. When doing matching and arbitration, FlexCAN2 needs to scan the whole message buffer memory during the available time slot. To have sufficient time to do that, the following restrictions must be observed:

- A valid CAN bit timing must be programmed, as indicated in Figure 21-15.
- The system clock frequency cannot be smaller than the oscillator clock frequency, i.e. the PLL cannot be programmed to divide down the oscillator clock.
- There must be a minimum ratio of 16 between the system clock frequency and the CAN bit rate.

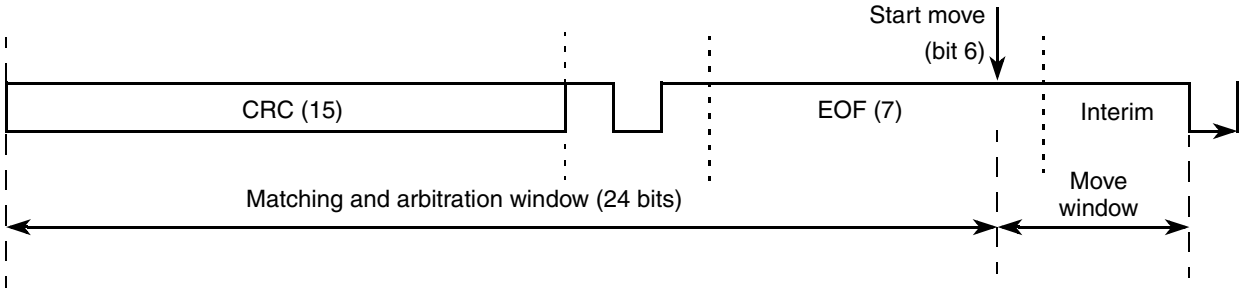


Figure 21-15. Arbitration, Match and Move Time Windows

21.4.6 Modes of Operation Details

21.4.6.1 Freeze Mode

This mode is entered by asserting the HALT bit in the CAN*x*_MCR or when the MCU is put into debug mode. In both cases it is also necessary that the FRZ bit is asserted in the CAN*x*_MCR. When freeze mode is requested during transmission or reception, FlexCAN2 does the following:

- Waits to be in either intermission, passive error, bus off or idle state
- Waits for all internal activities like move in or move out to finish
- Ignores the RX input pin and drives the TX pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the CAN*x*_ECR, which is read-only in other modes
- Sets the NOTRDY and FRZACK bits in CAN*x*_MCR

After requesting freeze mode, you must wait for the FRZACK bit to assert in CAN*x*_MCR before executing any other action, otherwise FlexCAN2 can operate in an unpredictable way. In freeze mode, all memory mapped registers are accessible; CANx_RXIMR*n* registers can be programmed only if the MBFEN bit is asserted.

Exit freeze mode using one of the following methods:

- CPU negates the FRZ bit in the CAN*x*_MCR
- The MCU exits debug mode and/or the HALT bit negates.

After exiting freeze mode, FlexCAN2 tries to re-synchronize to the CAN bus by waiting for 11 consecutive recessive bits.

21.4.6.2 Module Disabled Mode

This low power mode is entered when the CAN*x*_MCR[MDIS] bit is asserted. If the module is disabled during freeze mode, it shuts down the clocks to the CPI and MBM sub-modules, sets the CAN*x*_MCR[MDISACK] bit and negates the CAN*x*_MCR[FRZACK] bit.

If the module is disabled during transmission or reception, FlexCAN2 completes the following sequence:

1. Waits to enter the idle or bus off state, or waits for the third bit of the intermission and checks to determine if the bit is recessive.
2. Waits for all internal activities, like move-in or move-out, to finish.
3. Ignores the RX input pin and drives the TX pin as recessive.
4. Shuts down the clocks to the CPI and MBM sub-modules.
5. Sets the NOTRDY and MDISACK bits in CAN*x*_MCR.

The bus interface unit continues to operate by enabling the CPU to access memory mapped registers except the free running timer, CAN*x*_ECR, and the message buffers, which cannot be accessed when the module is disabled. To exit this mode, negate the CAN*x*_MCR[MDIS] bit, which resumes the clocks and negates the CAN*x*_MCR[MDISACK] bit.

21.4.7 Interrupts

The module can generate interrupts from 20 interrupt sources (16 interrupts due to message buffers, one interrupt due to an error condition, two interrupts for the OR'd MB16–MB31 and MB32–63, and one interrupt for one of the following: a bus off condition, a transmit warning, or a receive warning).

Each of the 64 message buffers can be an interrupt source, if its corresponding CAN*x*_IMRH or CAN*x*_IMRL bit is set. There is no distinction between TX and RX interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the CAN*x*_IFRH or CAN*x*_IFRL registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to 1.

A combined interrupt for each of two MB groups, MB16–MB31 and MB32–MB63, is also generated by an OR of all the interrupt sources from the associated MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case the CPU must read the CAN*x*_IFRH and CAN*x*_IFRL registers to determine which MB caused the interrupt.

The other two interrupt sources (bus off/transmit warning/receive warning and error) generate interrupts like the MB interrupt sources, and can be read from CAN*x*_ESR. The bus off/transmit warning/receive warning and error interrupt mask bits are located in the CAN*x*_CR.

21.4.8 Bus Interface

The CPU access to FlexCAN2 registers are subject to the following rules:

- Read and write access to unimplemented or reserved address space also results in access error. Any access to unimplemented MB locations results in access error.
- For a FlexCAN2 configuration that uses less than the total number of MBs and MAXMB is set accordingly, the remaining MB and RX mask register memory can be used as general-purpose RAM space. Byte, word and long word accesses are allowed to the unused MB space.

NOTE

Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

21.5 Initialization and Application Information

This section provides instructions for initializing the FlexCAN2 module.

21.5.1 FlexCAN2 Initialization Sequence

The FlexCAN2 module can be reset in three ways:

- MCU-level hard reset, which resets all memory mapped registers asynchronously
- MCU-level soft reset, which resets some of the memory mapped registers synchronously (see Table 21-2 for the registers affected by a soft reset)
- SOFTRST bit in CAN*x*_MCR, which has the same effect as the MCU level soft reset

A soft reset is synchronous and must follow an internal request/acknowledge procedure across clock domains. Therefore, it can take some time to fully propagate its effects. The SOFTRST bit remains asserted while a soft reset is pending, so software can poll this bit to determine when the reset completes.

After the module is enabled (CAN*x*_MCR[MDIS] bit negated), put FlexCAN2 in freeze mode before beginning the configuration. In freeze mode, FlexCAN2 is un-synchronized to the CAN bus, the HALT and FRZ bits in CAN*x*_MCR are set, the internal state machines are disabled and the FRZACK and NOTRDY bits in the CAN*x*_MCR are set. The CNTX pin is in recessive state and FlexCAN2 does not initiate frame transmission nor receives any frames from the CAN bus. The message buffer contents are not affected by reset, therefore are not automatically initialized.

For any configuration change or initialization, FlexCAN2 must be set to freeze mode (see Section 21.4.6.1, “Freeze Mode). A basic initialization sequence for the FlexCAN2 module is:

1. Initialize CANx_CR.
   - — Determine bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW.
   - — Determine the bit rate by programming the PRESDIV field.
   - — Determine internal arbitration mode (LBUF bit).
2. Initialize message buffers.
   - — The control and status word of all message buffers can be written as active or inactive.
   - — Initialize other entries in each message buffer as required.

3. Initialize CANx_MCR bits MBFEN, SRXDIS, and WRNEN.

   The initialization of FlexCAN registers for either global or individual acceptance masking depends on the configuration of MBFEN:

   — If MBFEN is negated, initialize CANx_RXGMASK, CANx_RX14MASK, and CANx_RX15MASK registers for acceptance mask.

   — If MBFEN is asserted, initialize CANx_RXIMR[0-63] for individual acceptance masking.

4. Set required mask bits in CAN*x*_IMRH and CAN*x*_IMRL registers (for all MBs interrupts), and in CAN*x*_CR (for bus off and error interrupts).
5. Negate the CAN*x*_MCR[HALT] bit.

Starting with this last event, FlexCAN2 attempts to synchronize with the CAN bus.

21.5.2 FlexCAN2 Addressing and RAM Size

There are 1024 bytes of RAM for a maximum of 64 message buffers. You can program the maximum number of message buffers (MBs) using the MAXMB field in the CAN*x*_MCR. For a 1024-byte RAM configuration, MAXMB can be any number from 0–63.

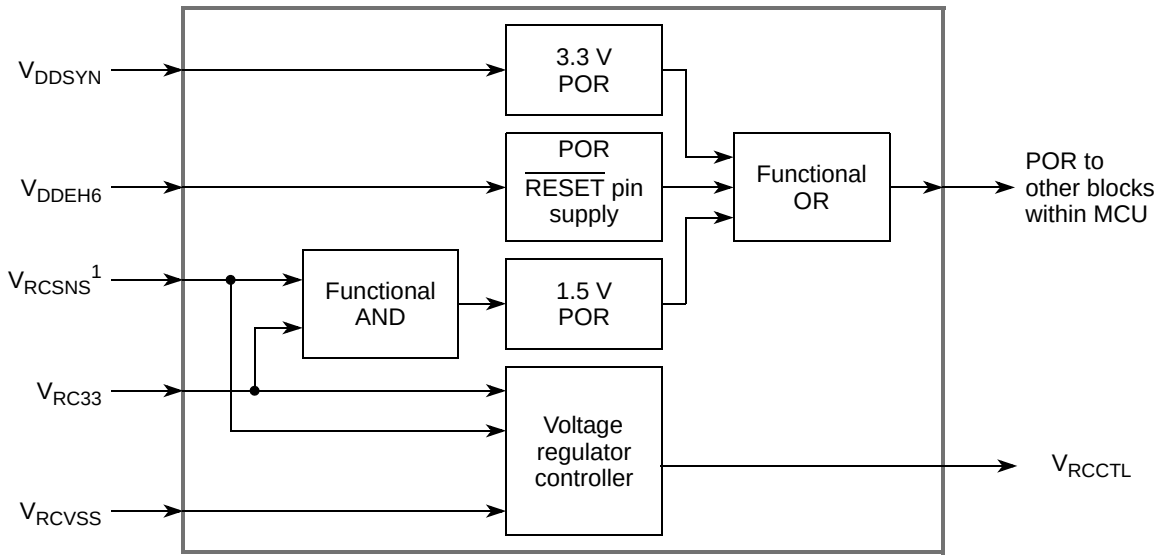
Chapter 22
Voltage Regulator Controller (VRC) and POR Module

22.1 Introduction

The voltage regulator controller (VRC) and power-on reset (POR) module contains circuitry to control regulation of the external 1.5 V supply used by the device. It also contains POR circuits for the 1.5 V supply, V_{DDSYN} and the V_{DDE} supply that powers the $\overline{\text{RESET}}$ pad.

22.1.1 Block Diagram

The block diagram of the VRC and POR module is shown in Figure 22-1. The diagram represents the various submodules as implemented on the device.



[1] This is not a package pin

Figure 22-1. Voltage Regulator Controller and POR Blocks

22.2 External Signal Description

Table 22-1 describes the VRC signals.

Table 22-1. Voltage Regulator Controller and POR Block External Signals

Signal	Type	Signal Level	Description
V_{RC33}	Supply pin	3.3 V	Regulator supply input. 3.3 V VRC supply input.
V_{DDSYN}	Supply pin	3.3 V	FMPLL supply input. This is the 3.3 V supply input for the frequency modulated phase lock loop (FMPLL) module.
V_{DDEH6}	Supply pin	3.3–5.0 V	$\overline{\text{RESET}}$ pin supply input. Power supply input for padring segment that contains the $\overline{\text{RESET}}$ pad.
V_{RCVSS}	Supply pin	0.0 V	Regulator supply ground. V_{RCVSS} is the 3.3 V ground reference for the on-chip 1.5 V regulator control circuit.
V_{RCSNS}	1.5 V sense	1.5 V	1.5 V sense used by VRC. Pad connected to V_{DD} plane in package—not a package ball. This is the 1.5 V sense from the external 1.5 V supply output of NPN transistor. This input is monitored by the VRC to determine the value for V_{RCCTL}. V_{RCSNS} is a pad on the die that is connected to a V_{DD} plane inside the package. It is not a package ball.
V_{RCCTL}	Current output	—	Regulator control output. The V_{RCCTL} sources base current to the external bypass transistor. The V_{RCCTL} signal is used with internal and external transistors to provide V_{DD}, which is the MCU 1.5 V power supply.
V_{DD}	Supply pin	1.5 V	Internal 1.5 V power supply input.

22.3 Memory Map and Register Definition

The VRC and POR module have no memory-mapped registers.

22.4 Functional Description

The VRC portion of the module contains a voltage regulator controller, and the POR portion contains circuits to monitor the voltage levels of the 1.5 V and V_{DDSYN} power supplies, as well as circuits to monitor the power supply for the $\overline{\text{RESET}}$ pad. The PORs indicate whether each supply monitored is above the specified voltage threshold. The PORs ensure that the device is correctly powered up during a power-on reset. POR holds the device in reset when any of the monitored supplies fall below the specified minimum voltage.

22.4.1 Voltage Regulator Controller

The VRC circuit has a control current for use with an external NPN transistor and an external resistor to power the 1.5 V V_{DD} supply. The control current is output on the V_{RCCTL} pin. The voltage regulator controller slowly turns on the pass transistor while the 3.3 V POR is asserted. The pass transistor is completely turned on by the time the 3.3 V POR negates.

The voltage regulator controller keeps the 1.5 V supply in regulation as long as V_{RC33} is in regulation. If more protection is desired, you can also supply an external 1.5 V low voltage reset circuit.

- If the on-chip voltage regulator controller is not used, an external 1.5 V supply is required. To avoid a power sequencing requirement when an external power supply is used, external 3.3 V must power V_{RC33} while the V_{RCCTL} pad is unconnected. In this case, the internal 1.5 V POR remains enabled.
- If V_{RC33} is not powered, the device is subject to the power sequencing requirements for the 1.5 V and 3.3 V, or $\overline{\text{RESET}}$ supplies. This ensures that the 1.5 V supply is high enough for internal logic to operate correctly during power-up.

See Section 22.5.3, “Power Sequencing Requirements” for more information.

22.4.2 POR Circuits

The individual POR circuits asserts whenever the supply being monitored drops below the specified threshold. The entire device is in power-on reset if any of these supplies are below the values specified in the device data sheet.

Power-on reset asserts as soon as possible after the voltage level of the POR supplies begin to rise. Each POR circuit negates before its supply rises to its specified range. Power-on reset remains asserted until all POR supplies rise above the maximum POR threshold. Each POR asserts after its supply drops below its specified range. The behavior for each POR during power sequencing is shown in Figure 22-2.

Before the 3.3 V POR circuit asserts when ramping up, or after it negates when ramping down, the device can exit POR but remain in system reset. In this case, MDO[0] is driving high and no clocks are toggling.

If the 3.3 V POR circuit is asserted, the device behaves as if in POR even if the 1.5 V and $\overline{\text{RESET}}$ power POR circuits have not yet asserted when ramping up or have negated when ramping down.

NOTE

The PORs for each supply are not intended to indicate that the voltage has dropped below the specified voltage range. You must monitor the supplies externally and assert $\overline{\text{RESET}}$ to achieve precise monitoring.

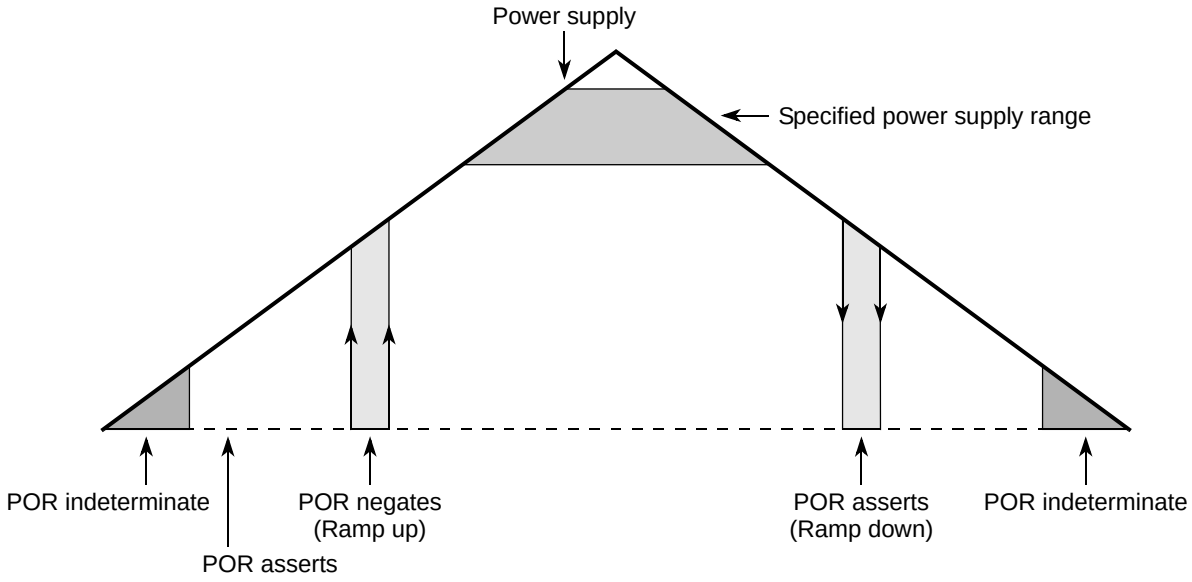


Figure 22-2. Regions POR is Asserted

22.4.2.1 1.5 V POR Circuit

The 1.5 V POR circuit monitors the voltage on the V_{RCSNS} pad. The 1.5 V POR functions if the V_{RC33} pad is powered. If you do not power V_{RC33} to the specified voltage, the 1.5 V POR is disabled and you must follow the specified power sequence.

22.4.2.2 3.3 V POR Circuit

The 3.3 V POR circuit is used to ensure that V_{DDSYN} is high enough that the FMPLL begins to operate correctly.

22.4.2.3 $\overline{\text{RESET}}$ Power POR Circuit

The $\overline{\text{RESET}}$ power POR circuit monitors the supply that powers the $\overline{\text{RESET}}$ pin, and ensures that the voltage supply to the $\overline{\text{RESET}}$ pin is high enough to reliably propagate the state of the input. The supply monitored by this POR cannot exceed 5.5 V.

22.5 Initialization and Application Information

22.5.1 Voltage Regulator Example

Figure 22-3 shows how to connect the VRC to the device.

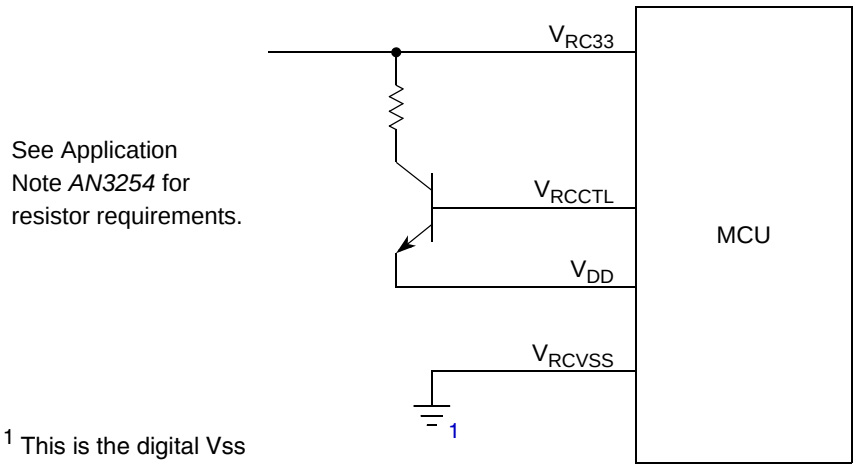


Figure 22-3. Voltage Regulator Controller Hookup

NOTE

Do not use Figure 22-3 as a reference for board design. See *Application Note AN3254* for resistor requirements.

22.5.2 Compatible Power Transistors

The following NPN transistors are compatible with the on-chip VRC:

- ON Semiconductor™ NJD2873
- Phillips Semiconductor™ BCP68

See the device data sheet for information on the operating characteristics.

22.5.3 Power Sequencing Requirements

This section describes the following power sequencing requirements for the device:

- If an external 1.5 V power supply is used and V_{RC33} is tied to ground, power sequencing is required between the 1.5 V power supply, and V_{DDSYN} and the $\overline{\text{RESET}}$ power supplies. To avoid this power sequencing requirement, power up V_{RC33} within the specified operating range, even if not using the on-chip voltage regulator controller.
  See Section 22.5.3.1, “Power-Up Sequence If VRC33 Grounded” and Section 22.5.3.2, “Power-Down Sequence If VRC33 Grounded.”
- The V_{DD33} voltage must be high enough before POR negates to ensure the values on certain pins are treated as 1s when POR negates.
  See Section 22.5.3.3, “Input Value of Pins During POR Dependent on VDD33.”

- When powering up, power sequencing is not required between V_{RC33} and V_{DDSYN}. However, for the VRC staged turn-on to operate within specification, V_{RC33} must not lead V_{DDSYN} by more than 600 mV, nor lag by more than 100 mV. Higher spikes in the emitter current of the pass transistor occur if V_{RC33} leads or lags V_{DDSYN} by exceeding these tolerances. The value of the current for the higher spikes depends on the board power supply circuitry and the amount of board-level capacitance.
- When powering down, delta tolerances between V_{RC33} and V_{DDSYN} are not required because the bypass capacitors internal and external to the device are already charged.
- When not powering up or down, delta tolerances between V_{RC33} and V_{DDSYN} are not required for the VRC to operate within specification.

22.5.3.1 Power-Up Sequence If V_{RC33} Grounded

The 1.5 V V_{DD} supply must rise to 1.35 V before the 3.3 V V_{DDSYN} and the $\overline{\text{RESET}}$ supplies rise above 2.0 V. This ensures that digital logic in the PLL for the 1.5 V supply does not begin to operate below the lower limit of the 1.35 V operation range. Because the internal 1.5 V POR is disabled, the internal 3.3 V POR or the $\overline{\text{RESET}}$ power POR must hold the device in reset. Since they can negate as low as 2.0 V, V_{DD} must be within the specification range before the 3.3 V POR and the $\overline{\text{RESET}}$ power POR negate.

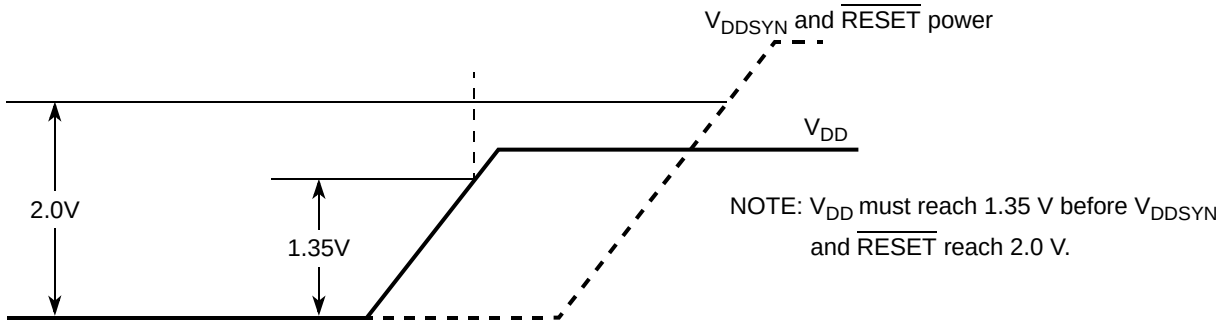


Figure 22-4. Power-Up Sequence, V_{RC33} Grounded

22.5.3.2 Power-Down Sequence If V_{RC33} Grounded

The only requirement for the power-down sequence when V_{RC33} is grounded is that if V_{DD} decreases to less than its operating range, V_{DDSYN} or the $\overline{\text{RESET}}$ power must decrease to less than 2.0 V before the V_{DD} power is allowed to increase to its operating range. This ensures that the digital 1.5 V logic, which is reset by the ORed POR only that can cause the 1.5 V supply to decrease below its specification, is reset correctly.

22.5.3.3 Input Value of Pins During POR Dependent on V_{DD33}

To avoid selecting the bypass clock because PLLCFG[0:1] and $\overline{\text{RSTCFG}}$ were not treated as 1s when POR negates, V_{DD33} must not lag V_{DDSYN} and the $\overline{\text{RESET}}$ pin power when powering the device by more than the V_{DD33_LAG} specification. V_{DD33} can independently lag V_{DDSYN} *or* $\overline{\text{RESET}}$ by more than the V_{DD33_LAG} specification. The V_{DD33_LAG} specification applies regardless of whether V_{RC33} is powered. The V_{DD33_LAG} specification only applies during power up. V_{DD33} has no lead or lag requirements when powering down. See the device data sheet for the V_{DD33_LAG} specification.

22.5.3.4 Pin Values after POR Negates

Depending on the final PLL mode required, the PLLCFG[0:1] and $\overline{\text{RSTCFG}}$ pins must have the values shown in Table 22-2 after POR negates. See application note *AN2613, "MPC5554 Minimum Board Configuration"* for one example of the external configuration circuit.

Table 22-2. Clock Mode Selection

Clock Mode	Package Pins			Synthesizer Status Register (FMPLL_SYNSR)[1] Bits		
	$\overline{\text{RSTCFG}}$[2]	PLLCFG[0]	PLLCFG[1]	MODE	PLLSEL	PLLREF
Crystal reference (324 package only)	1	PLLCFG pins ignored.		1	1	1
	0	1	0			
External reference	0	0	1	1	1	0
Bypass	0	0	0	0	0	0
Dual-controller	0	1	1	1	0	0

1 See Section 11.3.1.2, "Synthesizer Status Register (FMPLL_SYNSR)" for more information.

2 Because the 208 package has no $\overline{\text{RSTCFG}}$ pin, the signal is internally asserted (driven to 0), therefore the PLLCFG pins are always used to configure the FMPLL. After the device resets, the PLLCFG values remain the same as before the reset. The device does not reset to the crystal reference mode. Bypass mode is not enabled in the 208 package.

Table 22-3 lists external signals used by the FMPLL during normal operation.

Table 22-3. PLL External Pin Interface

Name	I/O Type	Function	Pull
$\overline{\text{RSTCFG}}$_GPIO[210][1]	I/O	Determines the configuration to use during reset. GPIO used otherwise.	Up
PLLCFG[0]_GPIO[208]	I/O	Configures the mode during reset. GPIO used otherwise.	Up
PLLCFG[1]_GPIO[209]	I/O	Configures the mode during reset. GPIO used otherwise.	Up
XTAL	Output	Output drive for external crystal	—
EXTAL_EXTCLK	Input	Crystal external clock input	—
V_{DDSYN}	Power	Analog power supply (3.3 V ±10%)	—
V_{SSSYN}	Ground	Analog ground	—

1 The 208 package does not have a $\overline{\text{RSTCFG}}$ pin, and the signal is internally asserted (driven to 0).

NOTE

After POR negates, $\overline{\text{RSTCFG}}$ and PLLCFG[0:1] can change to their final value, but do not switch through the 0, 0, 0 state on the pins.

Chapter 23
IEEE 1149.1 Test Access Port Controller (JTAGC)

23.1 Introduction

The JTAG port of the device consists of four inputs and one output. These pins include JTAG compliance select (JCOMP), test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS, and TCK are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface.

23.1.1 Block Diagram

Figure 23-1 is a block diagram of the JTAG Controller (JTAGC).

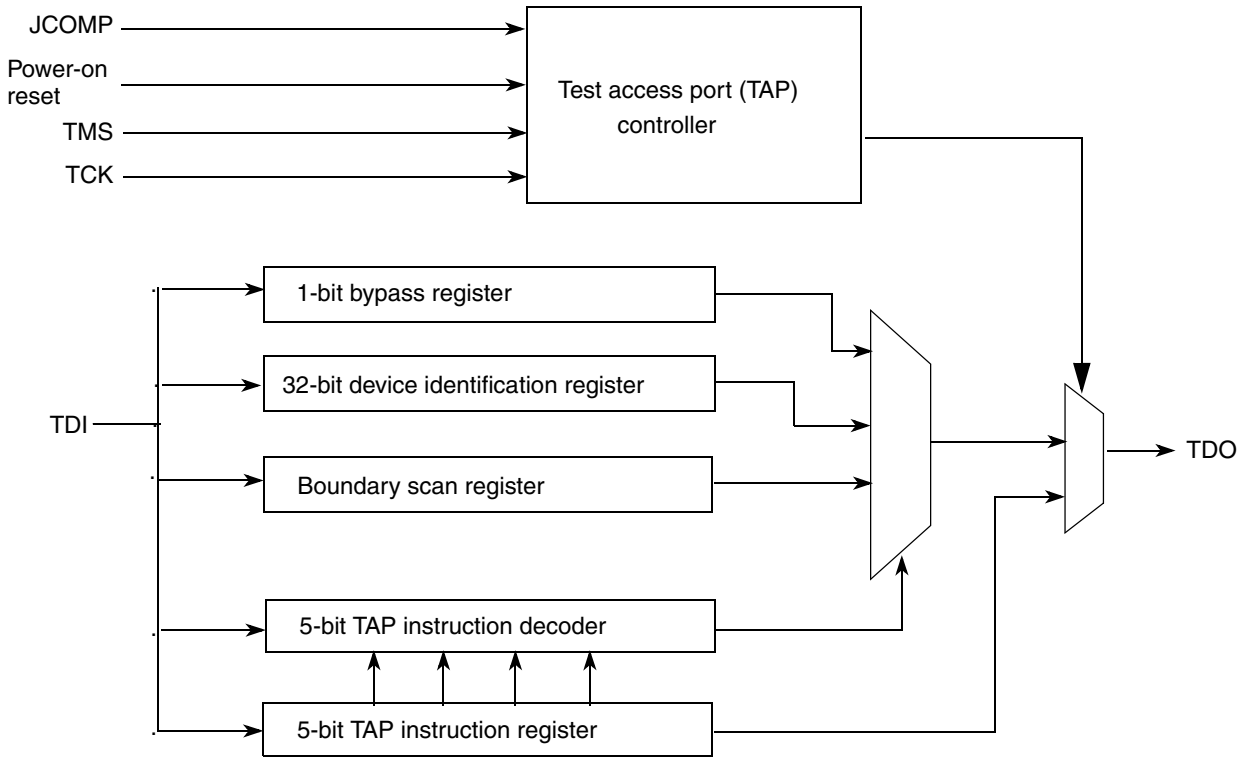


Figure 23-1. JTAG Controller Block Diagram

23.1.2 Overview

The JTAGC provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. In addition, instructions can be executed that allow the Test Access Port (TAP) to be shared with other modules on the MCU. All data input to and output from the JTAGC is communicated in serial format.

23.1.3 Features

The JTAGC is compliant with the IEEE 1149.1-2001 standard, and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface.
- 4 pins (TDI, TMS, TCK, and TDO). See Section 23.2, “External Signal Description.”
- A JCOMP input that provides the ability to share the TAP.
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions, as well as several public and private MCU specific instructions.
- Four test data registers: a bypass register, a boundary scan register, and a device identification register. The size of the boundary scan register is 432 bits.
- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry.

23.1.4 Modes of Operation

The JTAGC uses JCOMP and a power-on reset indication as its primary reset signals. Several IEEE 1149.1-2001 defined test modes are supported, as well as a bypass mode.

23.1.4.1 Reset

The JTAGC is placed in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered upon the assertion of the power-on reset signal, negation of JCOMP, or through TAP controller state machine transitions controlled by TMS. Asserting power-on reset or negating JCOMP results in asynchronous entry into the reset state. While in reset, the following actions occur:

- The TAP controller is forced into the test-logic-reset state, thereby disabling the test logic and allowing normal operation of the on-chip system logic to continue unhindered.
- The instruction register is loaded with the IDCODE instruction.

In addition, execution of certain instructions can result in assertion of the internal system reset. These instructions include EXTEST, CLAMP, and HIGHZ.

23.1.4.2 IEEE 1149.1-2001 Defined Test Modes

The JTAGC supports several IEEE 1149.1-2001 defined test modes. The test mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, HIGHZ, CLAMP, SAMPLE and SAMPLE/PRELOAD. Each instruction defines the set of data registers that can operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is enabled for serial access between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. The single-bit bypass register shift stage is enabled for serial access between TDI and TDO when the HIGHZ, CLAMP or reserved instructions are active. The functionality of each test mode is explained in more detail in Section 23.4.4, “JTAGC Instructions.”

23.1.4.3 Bypass Mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC into bypass mode. While in bypass mode, the single-bit bypass shift register is used to provide a minimum-length serial path to shift data between TDI and TDO.

23.1.4.4 TAP Sharing Mode

There are three selectable auxiliary TAP controllers that share the TAP with the JTAGC. Selectable TAP controllers include the Nexus port controller (NPC), e200 OnCE, and eTPU Nexus. The instructions required to grant ownership of the TAP to the auxiliary TAP controllers are ACCESS_AUX_TAP_NPC, ACCESS_AUX_TAP_ONCE, ACCESS_AUX_TAP_eTPUN3. Instruction opcodes for each instruction are shown in Table 23-3.

When the access instruction for an auxiliary TAP is loaded, control of the JTAG pins is transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

For more information on the TAP controllers see Chapter 24, “Nexus Development Interface.”

23.2 External Signal Description

The JTAGC consists of five signals that connect to off-chip development tools and allow access to test support functions. The JTAGC signals are outlined in the following table:

Table 23-1. JTAG Signal Properties

Name	I/O	Function	Reset State[1]
TCK	I	Test clock	Pulldown
TDI	I	Test data in	Pullup
TDO	O	Test data out	High Z / Pullup [2]
TMS	I	Test mode select	Pullup
JCOMP	I	JTAG compliancy	Pulldown

1 The pull is not implemented in the Nexus module. Pullup/down devices are implemented in the pads.

2 TDO output buffer enable is negated when JTAGC is not in the Shift-IR or Shift-DR states. A weak pullup is implemented on TDO.

23.3 Memory Map/Register Definition

This section provides a detailed description of the JTAGC registers accessible through the TAP interface, including data registers and the instruction register. Individual bit-level descriptions and reset states of each register are included. These registers are not memory-mapped and can only be accessed through the TAP.

23.3.1 Instruction Register

The JTAGC uses a 5-bit instruction register as shown in Figure 23-2. The instruction register allows instructions to be loaded into the module to select the test to be performed or the test data register to be accessed or both. Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the update-IR and test-logic-reset TAP controller states. Synchronous entry into the test-logic-reset state results in the IDCODE instruction being loaded on the falling edge of TCK. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the IDCODE instruction. During the capture-IR TAP controller state, the instruction shift register is loaded with the value 0b10101, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

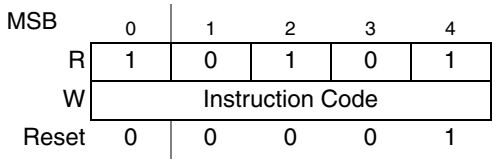


Figure 23-2. 5-Bit Instruction Register

23.3.2 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, CLAMP, HIGHZ or reserve instructions are active. After entry into the capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

23.3.3 Device Identification Register

The device identification register, shown in Figure 23-3, allows the part revision number, design center, part identification number, and manufacturer identity code to be determined through the TAP. The device identification register is selected for serial data transfer between TDI and TDO when the IDCODE instruction is active. Entry into the capture-DR state while the device identification register is selected loads the IDCODE into the shift register to be shifted out on TDO in the Shift-DR state. No action occurs in the update-DR state.

IR[4:0]: 0_0001 (IDCODE) Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRN				DC						PIN										MIC											ID
W																																
Reset	0	0	0	1	1	0	0	0	0	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 23-3. Device Identification Register

Table 23-2. Device Identification Register Field Descriptions

Field	Description
0–3 PRN	Part revision number. Contains the revision number of the device. This field changes with each revision of the device or module.
4–9 DC	Design center. Indicates the Freescale design center. For the MPC5534 this value is 0x20.
10–19 PIN	Part identification number. Contains the part number of the device. For the MPC5534, this value is 0x134.
20–30 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0xE.
31 ID	IDCODE register ID. Identifies this register as the device identification register and not the bypass register. Always set to 1.

23.3.4 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. It is used to capture input pin data, force fixed values on output pins, and select a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in Section 23.4.5, “Boundary Scan.” The size of the boundary scan register is 432 bits.

23.4 Functional Description

23.4.1 JTAGC Reset Configuration

While in reset, the TAP controller is forced into the test-logic-reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. In addition, the instruction register is loaded with the IDCODE instruction.

23.4.2 IEEE 1149.1-2001 (JTAG) Test Access Port

The JTAGC uses the IEEE 1149.1-2001 TAP for accessing registers. This port can be shared with other TAP controllers on the MCU. Ownership of the port is determined by the value of the JCOMP signal and the currently loaded instruction. For more detail on TAP sharing via JTAGC instructions see Section 23.4.4.2, “ACCESS_AUX_TAP_x Instructions.”

Data is shifted between TDI and TDO though the selected register starting with the least significant bit, as illustrated in Figure 23-4. This applies for the instruction register, test data registers, and the bypass register.

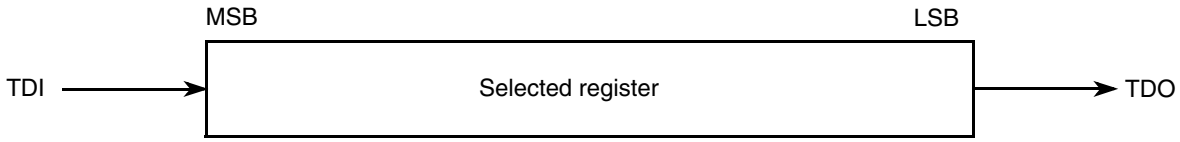
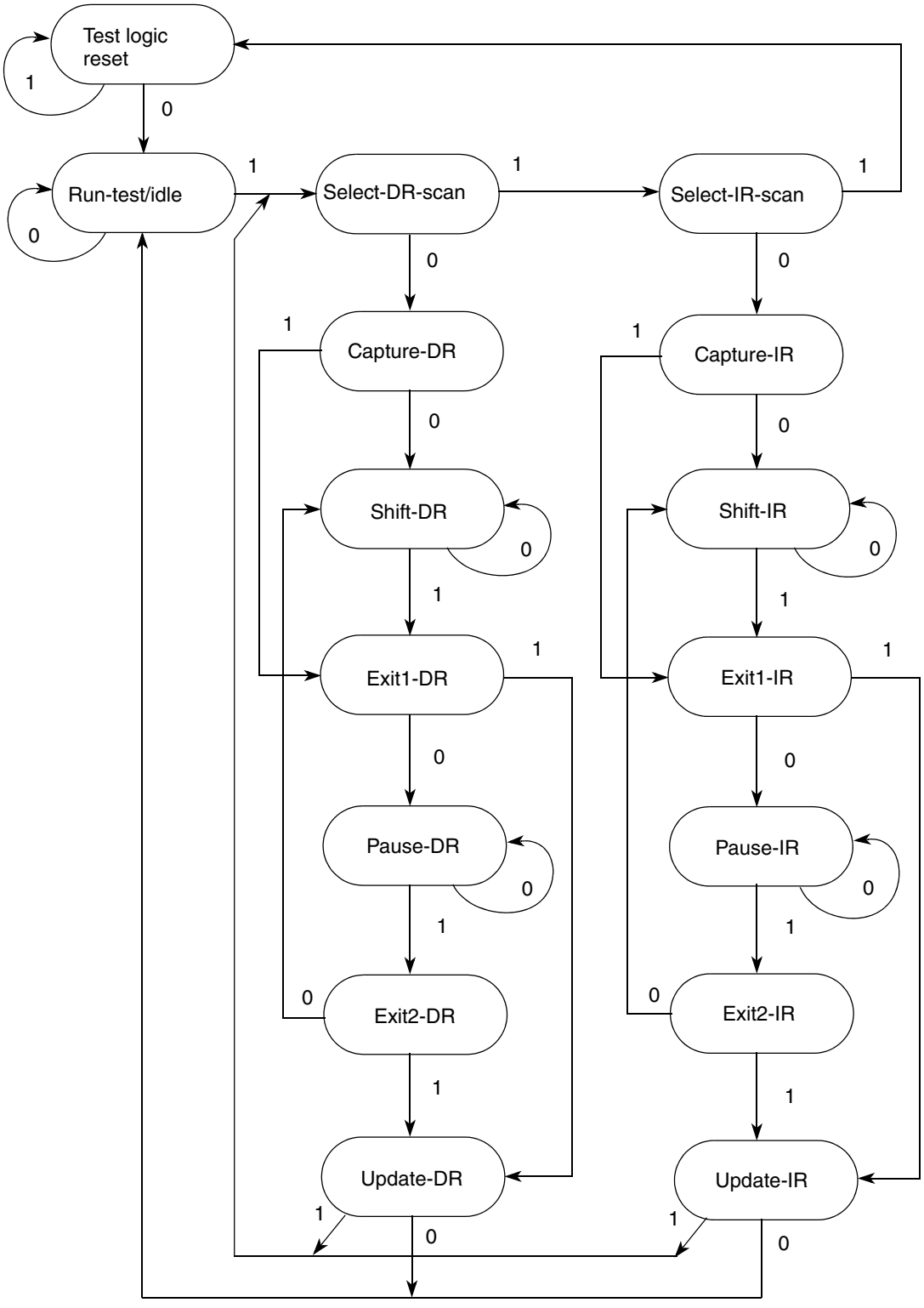


Figure 23-4. Shifting Data Through a Register

23.4.3 TAP Controller State Machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. Figure 23-5 shows the machine’s states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCK signal.

As Figure 23-5 shows, holding TMS at logic 1 while clocking TCK through a sufficient number of rising edges also causes the state machine to enter the test-logic-reset state.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 23-5. IEEE 1149.1-2001 TAP Controller Finite State Machine

23.4.3.1 Enabling the TAP Controller

The JTAGC TAP controller is enabled by setting JCOMP to a logic 1 value.

23.4.3.2 Selecting an IEEE 1149.1-2001 Register

Access to the JTAGC data registers is done by loading the instruction register with any of the JTAGC instructions while the JTAGC is enabled. Instructions are shifted in via the select-IR-scan path and loaded in the update-IR state. At this point, all data register access is performed via the select-DR-scan path.

The select-DR-scan path is used to read or write the register data by shifting in the data (LSB first) during the shift-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the capture-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the update-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting can be terminated after fetching the required number of bits.

23.4.4 JTAGC Instructions

This section gives an overview of each instruction, see the IEEE 1149.1-2001 standard for more details.

The JTAGC implements the IEEE 1149.1-2001 defined instructions listed in Table 23-3.

Table 23-3. JTAG Instructions

Instruction	Code[0:4]	Instruction Summary
IDCODE	00001	Selects device identification register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
HIGHZ	01001	Selects bypass register while three-stating all output pins and asserting functional reset
CLAMP	01100	Selects bypass register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_NPC	10000	Grants the Nexus port controller (NPC) ownership of the TAP
ACCESS_AUX_TAP_ONCE	10001	Grants the Nexus e200z3 core interface (NZ3C3) ownership of the TAP
ACCESS_AUX_TAP_eTPUN3	10010	Grants the Nexus eTPU development interface (NSEDI) ownership of the TAP
BYPASS	11111	Selects bypass register for data operations
Reserved	00101 00110 01010 10011	Do not use these settings.
Reserved[1]	All other codes	Do not use these settings. Defaults to bypass register.

[1] Freescale reserves the right to change the decoding of reserved instruction codes.

23.4.4.1 BYPASS Instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test functions. While the BYPASS instruction is active the system logic operates normally.

23.4.4.2 ACCESS_AUX_TAP_*x* Instructions

The ACCESS_AUX_TAP_*x* instructions allow the Nexus modules on the MCU to take control of the TAP. When this instruction is loaded, control of the TAP pins is transferred to the selected auxiliary TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

23.4.4.3 CLAMP Instruction

CLAMP allows the state of signals driven from MCU pins to be determined from the boundary scan register while the bypass register is selected as the serial path between TDI and TDO. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. CLAMP also asserts the internal system reset for the MCU to force a predictable internal state.

23.4.4.4 EXTEST—External Test Instruction

EXTEST selects the boundary scan register as the shift path between TDI and TDO. It allows testing of off-chip circuitry and board-level interconnections by driving preloaded data contained in the boundary scan register onto the system output pins. Typically, the preloaded data is loaded into the boundary scan register using the SAMPLE/PRELOAD instruction before the selection of EXTEST. EXTEST asserts the internal system reset for the MCU to force a predictable internal state while performing external boundary scan operations.

23.4.4.5 HIGHZ Instruction

HIGHZ selects the bypass register as the shift path between TDI and TDO. While HIGHZ is active, all output drivers are placed in an inactive drive state (for example, high impedance). HIGHZ also asserts the internal system reset for the MCU to force a predictable internal state.

23.4.4.6 IDCODE Instruction

IDCODE selects the 32-bit device identification register as the shift path between TDI and TDO. This instruction allows interrogation of the MCU to determine its version number and other part identification data. IDCODE is the instruction placed into the instruction register when the JTAGC is reset.

23.4.4.7 SAMPLE Instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when the SAMPLE instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. There is no defined action in the update-DR state. Both the data capture and the shift operation are transparent to system operation.

23.4.4.8 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- The SAMPLE part of the instruction samples the system data and control signals on the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising-edge of TCK in the capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the shift-DR state. Both the data capture and the shift operation are transparent to system operation.
- The PRELOAD part of the instruction initializes the boundary scan register cells before selecting the EXTEST or CLAMP instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the update-DR state. The data is applied to the external output pins by the EXTEST or CLAMP instruction. System operation is not affected.

23.4.5 Boundary Scan

The boundary scan technique allows signals at component boundaries to be controlled and observed through the shift-register stage associated with each pad. Each stage is part of a larger boundary scan register cell, and cells for each pad are interconnected serially to form a shift-register chain around the border of the design. The boundary scan register consists of this shift-register chain, and is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are loaded. The shift-register chain contains a serial input and serial output, as well as clock and control signals.

23.5 Initialization and Application Information

The test logic is a static logic design, and TCK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

To initialize the JTAGC module and enable access to registers, the following sequence is required:

1. Set the JCOMP signal to logic 1, thereby enabling the JTAGC TAP controller.
2. Load the appropriate instruction for the test or action to be performed.

Chapter 24
Nexus Development Interface

24.1 Introduction

The device microcontroller contains multiple Nexus clients that communicate over a single IEEE®-ISTO 5001™-2003 Nexus class 3 combined JTAG IEEE® 1149.1/auxiliary out interface. Combined, all of the Nexus clients are referred to as the Nexus development interface (NDI). Class 3 Nexus allows for program, data, and ownership trace of the microcontroller execution without access to the external data and address buses.

This chapter is organized into sections that provide a high level view of the Nexus development interface: Section 24.1, "Introduction" through Section 24.8, "NPC Initialization and Application Information."

The chapter contains sections that discuss the modules of the Nexus development interface:

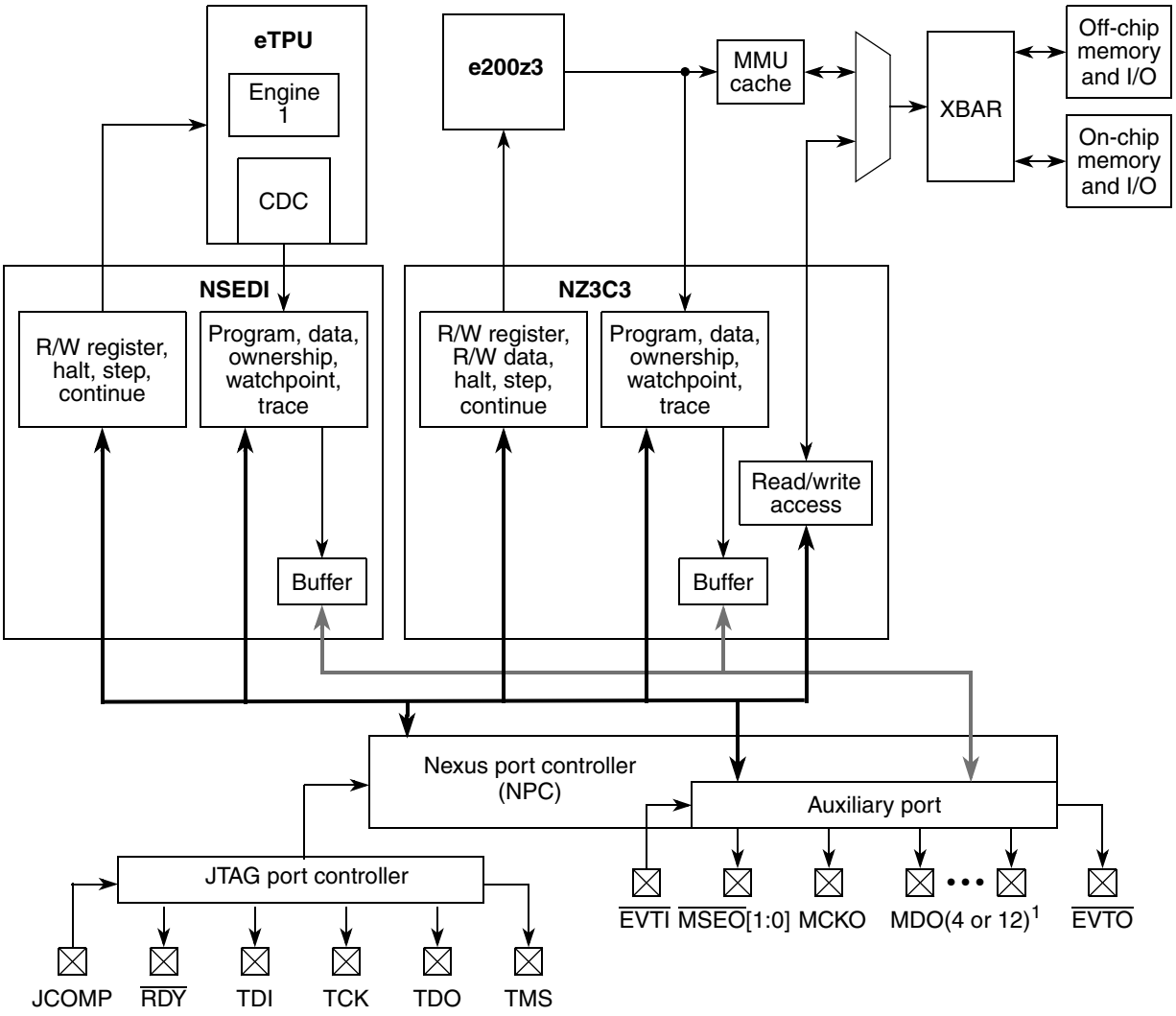
- Nexus single-eTPU development interface (NSEDI). The device has one eTPU engines. See Section 24.9, "Nexus Single eTPU Development Interface (NSEDI)" and the *eTPU Reference Manual* for information about the NSEDI.
- Nexus e200z3 core interface (NZ3C3). In this chapter, the NZ3C3 interface is discussed in Section 24.10, "e200z3 Class 3 Nexus Module (NZ3C3) through Section 24.11, "NZ3C3 Memory Map and Register Definition."

Communication to the NDI is managed via the auxiliary port and the JTAG port.

- The auxiliary port is comprised of nine or 17 output pins and 1 input pin. The output pins include one message clock out (MCKO) pin, four or 12 message data out (MDO) pins, two message start/end out ($\overline{\text{MSEO}}$) pins, one ready ($\overline{\text{RDY}}$) pin, and one event out ($\overline{\text{EVTO}}$) pin. Event in ($\overline{\text{EVTI}}$) is the only input pin for the auxiliary port.
- The JTAG port consists of four inputs and one output. These pins include JTAG compliance select (JCOMP), test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS, and TCK are compliant with the IEEE® 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface. JCOMP along with power-on reset and the TAP state machine are used to control reset for the NDI module. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAG controller (JTAGC) when JCOMP is asserted. See Table 24-4 for the JTAGC opcodes to access the different Nexus clients.

24.1.1 Block Diagram

Figure 24.1.2 shows a general block diagram of the NDI components.



[1] The 208 package does not have MDO[11:4] pins due to pin limitations. The 208 package has MDO[3:0] pins only.

Figure 24-1. NDI General Block Diagram

24.1.2 Features

The NDI module is compliant with the IEEE-ISTO 5001-2003 standard. The following features are implemented:

- 15- or 23-bit full duplex pin interface for medium and high visibility throughput
  - One of two modes selected by register configuration: full port mode (FPM) and reduced port mode (RPM). FPM comprises 12 MDO pins, and RPM comprises four MDO pins.
  - Auxiliary output port
    - One MCKO (message clock out) pin
    - Four or 12 MDO (message data out) pins
    - Two $\overline{\text{MSEO}}$ (message start/end out) pins
    - One $\overline{\text{RDY}}$ (ready) pin
    - One $\overline{\text{EVTO}}$ (event out) pin
  - Auxiliary input port uses one $\overline{\text{EVTI}}$ (event in) pin
  - Five-pin JTAG port (JCOMP, TDI, TDO, TMS, and TCK)
- Host processor (e200z3) development support features (NZ3C3)
  - IEEE-ISTO 5001-2003 standard class 3 compliant.
  - Data trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tool to trace reads and/or writes to selected internal memory resources.
  - Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing development tools to trace ownership flow.
  - Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct branches, indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code can be traced.
  - Watchpoint messaging (WPM) via the auxiliary port.
  - Watchpoint trigger enable of program and/or data trace messaging.
  - Data tracing of instruction fetches via private opcodes.
  - Subset of Power Architecture Book E software debug facilities with OnCE block (Nexus class 1 features).
- eTPU development support features (NSEDI)
  - IEEE-ISTO 5001-2002 standard Class 3 compliant for the eTPU engines.
  - Data trace via data write messaging and data read messaging. This allows the development tool to trace reads and writes to selected shared parameter RAM (SPRAM) address ranges. Two data trace windows are shared by the eTPU engine.
  - Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which channel is being serviced. An ownership trace message is transmitted to indicate when a new channel service request is scheduled, allowing the

development tools to trace task flow. A special OTM is sent when the engine enters in idle state, meaning that all requests were serviced and no new requests are yet scheduled.

  - Program trace via branch trace messaging. BTM displays program flow discontinuities (start, jumps, return, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code can be traced. The branch trace messaging method uses the branch/predicate method to reduce the number of generated messages.
  - Watchpoint messaging via the auxiliary port. WPM provides visibility of the occurrence of the eTPU's' watchpoints and breakpoints.
  - Nexus based breakpoint/watchpoint configuration and single step support.
- Run-time access to the on-chip memory map via the Nexus read/write access protocol. This feature supports accesses for run-time internal visibility, calibration variable acquisition, calibration constant tuning, and external rapid prototyping for powertrain automotive development systems.
- All features are independently configurable and controllable via the IEEE® 1149.1 I/O port.
- The NDI block reset is controlled with JCOMP, power-on reset, and the TAP state machine. These sources are independent of system reset.
- System clock locked status indication via MDO[0] following power-on reset.

24.1.3 Modes of Operation

The NDI block is in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered on the assertion of the power-on reset signal, negation of JCOMP, or through state machine transitions controlled by TMS. Assertion of JCOMP allows the NDI to move out of the reset state, and is a prerequisite to grant Nexus clients control of the TAP. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAGC controller (JTAGC) block when JCOMP is asserted.

Following negation of power-on reset, the NPC remains in reset until the system clock achieves lock. In PLL bypass mode, the NDI can transition out of the reset state immediately following negation of power-on reset. See Section 24.4.5, "System Clock Locked Indication" for more details.

24.1.3.1 Nexus Reset Mode

In Nexus reset mode, the following actions occur:

- Register values default back to their reset values.
- The message queues are marked as empty.
- The auxiliary output port pins are negated if the NDI controls the pads.
- The TDO output buffer is disabled if the NDI has control of the TAP.
- The TDI, TMS, and TCK inputs are ignored.
- The NDI block indicates to the MCU that it is not using the auxiliary output port. This indication can be used to three-state the output pins or use them for another function.

24.1.3.2 Full-Port Mode

In full-port mode, all the available MDO pins are used to transmit messages. All trace features are enabled or can be enabled by writing the configuration registers via the JTAG port. The number of MDO pins available is 12.

24.1.3.3 Reduced-Port Mode

In reduced-port mode, a subset of the available MDO pins are used to transmit messages. All trace features are enabled or can be enabled by writing the configuration registers via the JTAG port. The number of MDO pins available is four. Unused MDO (MDO[11:4]) pins can be used as GPIO. Details on GPIO functionality configuration can be found in Chapter 6, "System Integration Unit (SIU)."

24.1.3.4 Disabled-Port Mode

In disabled-port mode, message transmission is disabled. Any debug feature that generates messages can not be used. The primary features available are class 1 features and read/write access.

24.1.3.5 Censored Mode

When the device is in censored mode, reading the contents of internal flash externally is not allowed. To prevent Nexus modules from violating censorship, the NPC is held in reset when in censored mode, asynchronously holding all other Nexus modules in reset as well. This prevents Nexus read/write to memory mapped resources and the transmission of Nexus trace messages. See Table 13-18 for information on Nexus port enabling and disabling regarding censorship.

24.2 External Signal Description

The auxiliary and JTAG pin interfaces provide for the transmission of messages from Nexus modules to the external development tools and for access to Nexus client registers. The auxiliary/JTAG pin definitions are outlined in Table 24-1.

Table 24-1. Signal Properties

Signal Name	Port	Function	Reset State
$\overline{\text{EVTO}}$	Auxiliary	Event out pin	Negated
$\overline{\text{EVTI}}$	Auxiliary	Event in pin	Pullup
MCKO	Auxiliary	Message clock out pin (from NPC)	Enabled
MDO[3:0] or MDO[11:0]	Auxiliary	Message data out pins	Driven Low[1]
$\overline{\text{MSEO}}$[1:0]	Auxiliary	Message start/end out pins	Negated
$\overline{\text{RDY}}$	Auxiliary	Ready out pin	Negated
JCOMP	JTAG	JTAG compliancy and TAP sharing control	Pulldown
TCK	JTAG	Test clock input	Pulldown
TDI	JTAG	Test data input	Pullup

Table 24-1. Signal Properties (continued)

Signal Name	Port	Function	Reset State
TDO	JTAG	Test data output	High Z / Pullup
TMS	JTAG	Test mode select input	Pullup

[1] Following a power-on reset, MDO[0] remains asserted until power-on reset is exited and the system clock achieves lock.

24.2.1 Detailed Signal Descriptions

This section describes each of the signals listed in Table 24-1 in more detail.

24.2.1.1 Event Out (EVTO)

EVTO is an output pin that is asserted upon breakpoint occurrence to provide breakpoint status indication or to signify that an event has occurred. The EVTO output of the NPC is generated based on the values of the individual EVTO signals from all Nexus modules that implement the signal.

24.2.1.2 Event In (EVTI)

EVTI is used to initiate program and data trace synchronization messages or to generate a breakpoint. EVTI is edge-sensitive for synchronization and breakpoint generation.

24.2.1.3 Message Data Out (MDO[3:0] or [11:0])

Message data out (MDO) are output pins used for uploading OTM, BTM, DTM, and other messages to the development tool. The development tool must sample MDO on the rising edge of MCKO. The width of the MDO bus used is determined by the Nexus PCR[FPM] configuration.

Following a power-on reset, MDO[0] remains asserted until power-on reset is exited and the system clock achieves lock.

208 Package: MDO[11:4] pins are not available due to pin limitations.

24.2.1.4 Message Start/End Out (MSEO[1:0])

MSEO[1:0] are output pins that indicates when a message on the MDO pins has started, when a variable length packet has ended, or when the message has ended. The development tool must sample the MSEO pins on the rising edge of MCKO.

24.2.1.5 Ready (RDY)

RDY is an output pin that indicates when a device is ready for the next access.

208 Package: The RDY signal is not available due to pin limitations.

24.2.1.6 JTAG Compliancy (JCOMP)

The JCOMP signal enables or disables the TAP controller. The TAP controller is enabled when JCOMP asserts, otherwise the TAP controller remains in reset.

24.2.1.7 Test Data Output (TDO)

The TDO pin transmits serial output for instructions and data. TDO is tri-stateable and is actively driven in the SHIFT-IR and SHIFT-DR controller states. TDO is updated on the falling edge of TCK and sampled by the development tool on the rising edge of TCK.

24.2.1.8 Test Clock Input (TCK)

The TCK pin is used to synchronize the test logic and control register access through the JTAG port.

24.2.1.9 Test Data Input (TDI)

The TDI pin receives serial test instruction and data. TDI is sampled on the rising edge of TCK.

24.2.1.10 Test Mode Select (TMS)

The TMS pin is used to sequence the IEEE® 1149.1-2001 TAP controller state machine. TMS is sampled on the rising edge of TCK.

24.3 Memory Map

The NDI block contains no memory mapped registers. Nexus registers are accessed by the development tool via the JTAG port using a register index and a client select value. The client select is controlled by loading the correct access instruction into the JTAG controller; see Table 24-4. OnCE registers are accessed by loading the appropriate value in the RS[0:6] field of the OnCE command register (OCMD) via the JTAG port.

Table 24-2 shows the NDI registers by Client Source ID and Index values.

Table 24-2. Nexus Development Interface (NDI) Registers

Client Source ID	Index	Register
e200z3 Control and Status Registers[1]		
0b0000	2	e200z3 Development Control1 (NZ3C3_DC1)
0b0000	3	e200z3 Development Control2 (NZ3C3_DC2)
0b0000	4	e200z3 Development Status (NZ3C3_DS)
0b0000	6	e200z3 User Base Address (NZ3C3_UBA)
0b0000	7	Read/Write Access Control/Status (NZ3C3_RWCS)
0b0000	9	Read/Write Access Address (NZ3C3_RWA)

Table 24-2. Nexus Development Interface (NDI) Registers (continued)

Client Source ID	Index	Register
0b0000	10	Read/Write Access Data (NZ3C3_RWD)
0b0000	11	e200z3 Watchpoint Trigger (NZ3C3_WT)
0b0000	13	e200z3 Data Trace Control (NZ3C3_DTC)
0b0000	14	e200z3 Data Trace Start Address 0 (NZ3C3_DTSA1)
0b0000	15	e200z3 Data Trace Start Address 1 (NZ3C3_DTSA2)
0b0000	18	e200z3 Data Trace End Address 0 (NZ3C3_DTEA1)
0b0000	19	e200z3 Data Trace End Address 1 (NZ3C3_DTEA2)
eTPU 1 Control/Status Registers		
0b0010	0	Device ID (DID)
0b0010	2	eTPU1 Development Control (NDI_eTPU1_DC)
0b0010	4	eTPU1 Development Status (NSEDI_eTPU1_DS)
0b0010	6	eTPU1 User Base Address (NSEDI_UBA)
0b0000	7	Read/Write Access Control/Status (RWCS)
0b0000	9	Read/Write Access Address (RWA)
0b0000	10	Read/Write Access Data (RWD)
0b0010	11	eTPU1 Watchpoint Trigger (NDI_eTPU1_WT)
0b0010	13	eTPU1 Data Trace Control (NDI_eTPU1_DTC)
0b0010	22	eTPU1 Breakpoint/Watchpoint Control 1 (NSEDI_eTPU1_BWC1)
0b0010	23	eTPU1 Breakpoint/Watchpoint Control 2 (NSEDI_eTPU1_BWC2)
0b0010	24	eTPU1 Breakpoint/Watchpoint Control 3 (NSEDI_eTPU1_BWC3)
0b0010	30	eTPU1 Breakpoint/Watchpoint Address 1 (NSEDI_eTPU1_BWA1)
0b0010	31	eTPU1 Breakpoint/Watchpoint Address 2 (NSEDI_eTPU1_BWA2)
0b0010	38	eTPU1 Breakpoint/Watchpoint Data 1 (NSEDI_eTPU1_BWD1)
0b0010	39	eTPU1 Breakpoint/Watchpoint Data 1 (NSEDI_eTPU1_BWD2)
0b0010	64	eTPU1 Program Trace Channel Enable (NDI_eTPU1_PTCE)
0b0010	69	eTPU1 Microinstruction Debug Register (NSEDI_eTPU1_INST)
0b0010	70	eTPU1 Microprogram Counter Debug Register (NSEDI_eTPU1_MPC)
0b0010	71	eTPU1 Channel Flag Status Register (NSEDI_eTPU1_CFSR)

Table 24-2. Nexus Development Interface (NDI) Registers (continued)

Client Source ID	Index	Register
eTPU CDC Control/Status Registers		
0b0100	13	eTPU CDC Data Trace Control (NSEDI_CDC_DTC)
eTPU1 / CDC Shared Control/Status Registers		
0b0010 or 0b0011 or 0b0100	65	eTPU Data Trace Address Range 0 (eTPU_DTAR0)
0b0010 or 0b0011 or 0b0100	66	eTPU Data Trace Address Range 1 (eTPU_DTAR1)
0b0010 or 0b0011 or 0b0100	67	eTPU Data Trace Address Range 2 (eTPU_DTAR2)
0b0010 or 0b0011 or 0b0100	68	eTPU Data Trace Address Range 3 (eTPU_DTAR3)

[1] These e200z3 registers are described in the *e200z3 PowerPC™ Core Reference Manual.*

Table 24-3 shows the OnCE register addressing.

Table 24-3. e200z3 OnCE Register Addressing

OCMD, RS[0:6]	Register Selected
000 0000–000 0001	Invalid value
000 0010	JTAG DID (read-only)
000 0011–000 1111	Invalid value
001 0000	CPU Scan Register (CPUSCR)
001 0001	No Register Selected (Bypass)
001 0010	OnCE Control Register (OCR)
001 0011–001 1111	Invalid value
010 0000	Instruction Address Compare 1 (IAC1)
010 0001	Instruction Address Compare 2 (IAC2)
010 0010	Instruction Address Compare 3 (IAC3)
010 0011	Instruction Address Compare 4 (IAC4)
010 0100	Data Address Compare 1 (DAC1)
010 0101	Data Address Compare 2 (DAC2)
010 0110–010 1011	Invalid value
010 1100	Debug Counter Register (DBCNT)

Table 24-3. e200z3 OnCE Register Addressing (continued)

OCMD, RS[0:6]	Register Selected
010 1101	Debug PCFIFO (PCFIFO) (read-only)
010 1110–010 1111	Invalid value
011 0000	Debug Status Register (DBSR)
011 0001	Debug Control Register 0 (DBCR0)
011 0010	Debug control register 1 (DBCR1)
011 0011	Debug control register 2 (DBCR2)
011 0100	Debug control register 3 (DBCR3)
011 0101–101 1111	Invalid value (do not access)
111 0000–111 1011	General purpose register selects [0:11]
111 1100	Nexus3-access
111 1101	LSRL select
111 1110	Enable_OnCE (and bypass)
111 1111	Bypass

24.4 NDI Functional Description

24.4.1 Enabling Nexus Clients for TAP Access

After the NDI is out of the reset state, the loading of a specific instruction in the JTAG controller (JTAGC) block is required to grant the NDI ownership of the TAP. Each Nexus client has its own JTAGC instruction opcode for ownership of the TAP, granting that client the means to read/write its registers. The JTAGC instruction opcode for each Nexus client is shown in Table 24-4. After the JTAGC opcode for a client has been loaded, the client is enabled by loading its NEXUS-ENABLE instruction. The NEXUS-ENABLE instruction opcode for each Nexus client is listed in Table 24-5. Opcodes for all other instructions supported by Nexus clients can be found in the relevant sections of this chapter.

Table 24-4. JTAG Client Select Instructions

JTAGC Instruction	Opcode	Description
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller
ACCESS_AUX_TAP_ONCE	10001	Enables access to the e200z3 OnCE TAP controller
ACCESS_AUX_TAP_eTPU	10010	Enables access to the eTPU Nexus TAP controller

Table 24-5. Nexus Client JTAG Instructions

Instruction	Description	Opcode
NPC JTAG Instruction Opcodes		
NEXUS_ENABLE	Opcode for NPC Nexus Enable instruction (4-bits)	0x0
BYPASS	Opcode for the NPC BYPASS instruction (4-bits)	0xF
e200z3 OnCE JTAG Instruction Opcodes[1]		
NEXUS3_ACCESS	Opcode for e200z3 OnCE Nexus Enable instruction (10-bits)	0x7C
BYPASS	Opcode for the e200z3 OnCE BYPASS instruction (10-bits)	0x7F

1 See the e200z3 Reference Manual for a complete list of available OnCE instructions.

24.4.2 Configuring the NDI for Nexus Messaging

The NDI is placed in disabled mode upon exit of power-on reset. If message transmission via the auxiliary port is desired, a write to the port configuration register (PCR) located in the NPC is then required to enable the NDI and select the mode of operation. Asserting MCKO_EN in the PCR places the NDI in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting or negating the FPM bit selects full-port or reduced-port mode, respectively. When writing to the PCR, the PCR lsb (least significant bit) must be written to a logic 0. Setting the lsb of the PCR enables factory debug mode and prevents the transmission of Nexus messages.

Table 24-6 describes the NDI configuration options.

Table 24-6. NDI Configuration Options

JCOMP Asserted	MCKO_EN bit of the Port Configuration Register	FPM bit of the Port Configuration Register	Configuration
No	X	X	Reset
Yes	0	X	Disabled
Yes	1	1	Full-port mode
Yes	1	0	Reduced-port mode

24.4.3 Programmable MCKO Frequency

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock, and its frequency is determined by the value of the MCKO_DIV field in the port configuration register (PCR) located in the NPC. Possible operating frequencies include one-half, one-quarter, and one-eighth system clock speed.

Table 24-7 shows the MCKO_DIV encodings. In this table, SYS_CLK represents the system clock frequency. The default value selected if a reserved encoding is programmed is SYS_CLK divided by two.

Table 24-7. MCKO_DIV Values

MCKO_DIV[2:0]	MCKO Frequency
0b000	SYS_CLK
0b001	SYS_CLK ÷ 2
0b010	Invalid value
0b011	SYS_CLK ÷ 4
0b100	Invalid value
0b101	Invalid value
0b110	Invalid value
0b111	SYS_CLK ÷ 8

24.4.4 Nexus Messaging

Most of the messages transmitted by the NDI include a SRC field. This field is used to identify which source generated the message. Table 24-8 shows the values used for the SRC field by the different clients on the device. These 4-bit values are specific to the device. The same values are used for the client select values written to the client select control register.

Table 24-8. SRC Packet Encodings

SRC[3:0]	Client
0b0000	e200z3
0b0010	eTPU1 (ENGINE1_SRC)
0b0100	eTPU CDC[1] (CDC_SRC)
0b0101–0b1111	Reserved

[1] CDC is the eTPU Coherent Dual-Parameter Controller. See the *eTPU Reference Manual* for more information.

24.4.5 System Clock Locked Indication

Following a power-on reset, the lsb of the auxiliary output port pins (MDO[0]) can be monitored to provide the lock status of the system clock. MDO[0] is driven to a logic one until the system clock achieves lock after exiting power-on reset. After the system clock is locked, MDO[0] is negated and tools can begin Nexus configuration. Loss of lock conditions that occur subsequent to the exit of power-on reset and the initial lock of the system clock do not cause a Nexus reset, and therefore do not result in MDO[0] driven high.

24.5 Nexus Port Controller (NPC)

The Nexus port controller (NPC) is that part of the NDI that controls access and arbitration of the device's internal Nexus modules. The NPC contains the port configuration register (PCR) and the device

identification register (DID). The contents of the NPC DID are the same as the JTAGC device identification register.

24.5.1 Overview

The device incorporates multiple modules that require development support. Each of these modules implements a development interface based on the IEEE-ISTO 5001-2001 standard and must share the input and output ports that interface with the development tool. The NPC controls the usage of these ports in a manner that allows the individual modules to share the ports, while appearing to the development tool as a single module.

24.5.2 Features

The NPC performs the following functions:

- Controls arbitration for ownership of the Nexus auxiliary output port
- Nexus device identification register and messaging
- Generates MCKO enable and frequency division control signals
- Controls sharing of $\overline{\text{EVTO}}$
- Control of the device-wide debug mode
- Generates asynchronous reset signal for Nexus modules based on JCOMP input, censorship status, and power-on reset status
- System clock locked status indication via MDO[0] during Nexus reset
- Provides Nexus support for censorship mode

24.6 Memory Map and Register Definition

This section provides a detailed description of the NPC registers accessible to the end user. Individual bit-level descriptions and reset states of the registers are included.

24.6.1 Memory Map

Table 24-9 shows the NPC registers by index values. The registers are not memory-mapped and can only be accessed via the TAP. The NPC does not implement the client select control register because the value does not matter when accessing the registers. The bypass register (see Section 24.6.2.1, “Bypass Register”) and instruction register (see Section 24.6.2.2, “Instruction Register”) have no index values. These registers are not accessed in the same manner as Nexus client registers.

Table 24-9. NPC Memory Map

Index	Register Name	Register Description	Bits
0	DID	Device ID register	32
127	PCR	Port configuration register	32

24.6.2 Register Descriptions

This section consists of NPC register descriptions. Additional information regarding references to the TAP controller state can be found in Section 24.4.3, “TAP Controller State Machine.”

24.6.2.1 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS instruction or any unimplemented instructions are active. After entry into the Capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

24.6.2.2 Instruction Register

The NPC uses a 4-bit instruction register as shown in Figure 24-2. The instruction register is accessed via the SELECT_IR_SCAN path of the tap controller state machine, and allows instructions to be loaded into the module to enable the NPC for register access (NEXUS_ENABLE) or select the bypass register as the shift path from TDI to TDO (BYPASS or unimplemented instructions).

Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and test-logic-reset TAP controller states. Synchronous entry into the test-logic-reset state results in synchronous loading of the BYPASS instruction. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the BYPASS instruction. During the Capture-IR TAP controller state, the instruction register is loaded with the value of the previously executed instruction, making this value the register’s read value when the TAP controller is sequenced into the Shift-IR state.

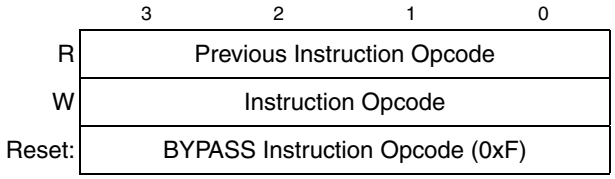


Figure 24-2. 4-Bit Instruction Register

24.6.2.3 Nexus Device ID Register (DID)

The NPC device identification register, shown in Figure 24-3, allows the part revision number, design center, part identification number, and manufacturer identity code of the part to be determined through the auxiliary output port.

Reg Index: 0 Access: User R/O

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Part Revision Number				Design Center						Part Identification Number					
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Part Identification Number (*continued*)				Manufacturer Identity Code											1
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 24-3. Nexus Device ID Register (DID)

Table 24-10. DID Register Field Descriptions

Field	Description
31–28 PRN	Part revision number. Contains the revision number of the part. This field changes with each revision of the device or module.
27–22 DC	Design center. Indicates the Freescale design center. This value is 0x0020.
21–12 PIN	Part identification number. Contains the part number of the device. The PIN for the MPC5534 is 0x0134.
11–1 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0x000E.
0	Fixed per JTAG 1149.1 Always set to 1.

24.6.2.4 Port Configuration Register (PCR)

The PCR, shown in Figure 24-4, is used to select the NPC mode of operation, enable MCKO and select the MCKO frequency, and enable or disable MCKO gating. This register must be configured as soon as the NPC is enabled.

NOTE

The mode (MCKO_GT) or clock division (MCKO_DIV) bits must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.

Reg Index: 127 Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FPM	MCKO_GT	MCKO_EN	MCKO_DIV			0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																PSTAT_EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-4. Port Configuration Register (PCR)

Table 24-11. PCR Field Descriptions

Field	Description
31 FPM	Full port mode. Determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 0 The subset of MDO[3:0] pins are used to transmit messages. 1 All MDO[11:0] pins are used to transmit messages. Section 6.4.1.12.45, “Pad Configuration Register 130 (SIU_PCR130)” shows how GPIO is enabled or disabled by the FPM setting.
30 MCKO_GT	MCKO clock gating control. Enables or disables MCKO clock gating. If clock gating is enabled, the MCKO clock is gated when the NPC is in enabled mode but not actively transmitting messages on the auxiliary output port. When clock gating is disabled, MCKO is allowed to run even if no auxiliary output port messages are being transmitted. 0 MCKO gating is disabled. 1 MCKO gating is enabled.
29 MCKO_EN	MCKO enable. Enables the MCKO clock. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. 0 MCKO clock is driven to zero. 1 MCKO clock is enabled.
28–26 MCKO_DIV [2:0]	MCKO division factor. Determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. The table below shows the meaning of MCKO_DIV values. In this table, SYS_CLK represents the system clock frequency.

MCKO_DIV[2:0]	MCKO Frequency
0	SYS_CLK
1	SYS_CLK ÷ 2
2	Invalid value
3	SYS_CLK ÷ 4
4	Invalid value
5	Invalid value
6	Invalid value
7	SYS_CLK ÷ 8

Table 24-11. PCR Field Descriptions (continued)

Field	Description
25–1	Reserved
0 PSTAT_EN	Processor status mode enable. Enables processor status (PSTAT) mode. In PSTAT mode, all auxiliary output port MDO pins are used to transmit processor status information, and Nexus messaging is unavailable. 0 PSTAT mode disabled 1 PSTAT mode enabled **Note:** PSTAT mode is intended for factory processor debug only. The PSTAT_EN bit must be written to disable PSTAT mode by the customer. No Nexus messages are transmitted under any circumstances when PSTAT mode is enabled

24.7 NPC Functional Description

24.7.1 NPC Reset Configuration

The NPC is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the PCR is then required to enable the NPC and select the mode of operation. Asserting MCKO_EN places the NPC in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting or negating the FPM bit selects full-port or reduced-port mode, respectively.

Table 24-12 describes the NPC reset configuration options.

Table 24-12. NPC Reset Configuration Options

JCOMP Asserted?	PCR[MCKO_EN]	PCR[FPM]	Configuration
No	X	X	Reset
Yes	0	X	Disabled
Yes	1	1	Full-Port Mode
Yes	1	0	Reduced-Port Mode

24.7.2 Auxiliary Output Port

The auxiliary output port is shared by each of the Nexus modules on the device. The NPC communicates with each of the individual modules and arbitrates for access to the port. Additional information about the auxiliary port is found in Section 24.2, “External Signal Description.”

24.7.2.1 Output Message Protocol

The protocol for transmitting messages via the auxiliary port is accomplished with the $\overline{\text{MSEO}}$ functions. The $\overline{\text{MSEO}}$ pins are used to signal the end of variable-length packets and the end of messages. They are not required to indicate the end of fixed-length packets. MDO and $\overline{\text{MSEO}}$ are sampled on the rising edge of MCKO.

Figure 24-5 illustrates the state diagram for $\overline{\text{MSEO}}$ transfers. All transitions not included in the figure are reserved, and must not be used.

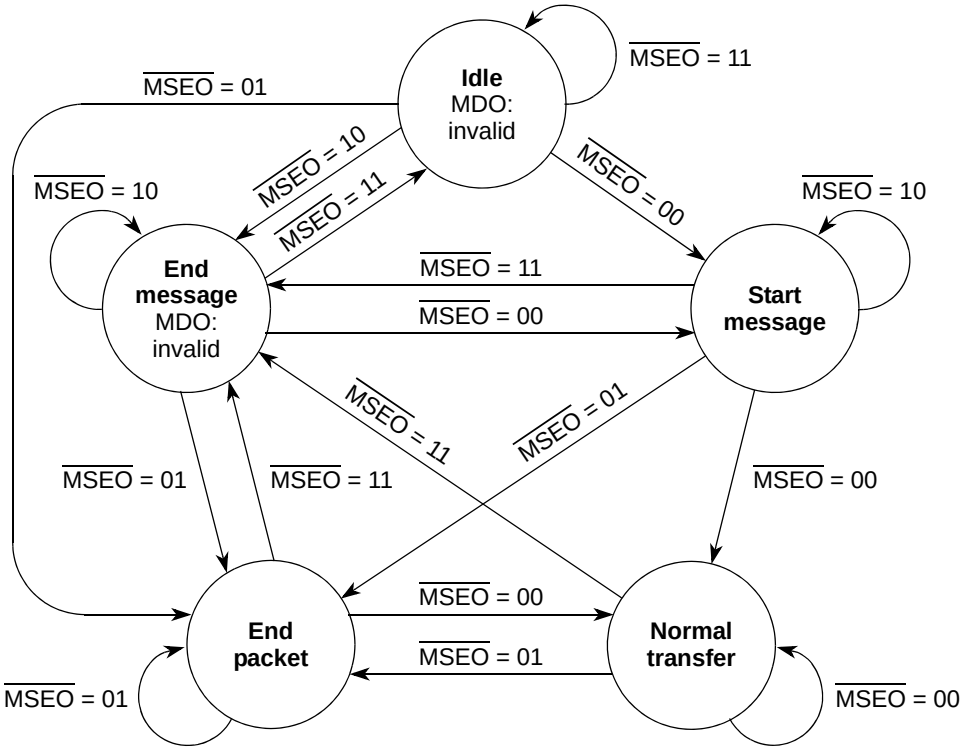


Figure 24-5. $\overline{\text{MSEO}}$ Transfers

24.7.2.2 Output Messages

In addition to sending out messages generated in other Nexus modules, the NPC can also output the device ID message contained in the device ID register on the MDO pins. The device ID message can also be sent out serially through TDO.

Table 24-13 describes the device ID message that the NPC can transmit on the auxiliary port. The TCODE is the first packet transmitted.

Table 24-13. NPC Output Messages

Message Name	Min. Packet Bits	Max Packet Bits	Packet Type	Packet Name	Packet Description
Device ID Message	6	6	Fixed	TCODE	Value = 1
	32	32	Fixed	ID	DID register contents

Figure 24-6 shows the various message formats that the pin interface formatter has to encounter.

Message	TCODE	Field #1	Field #2	Field #3	Field #4	Field #5	Min. Size[1] Bits	Max. Size[2] Bits
Device ID Message	1	Fixed = 32	—	—	—	—	38	38

1 Minimum information size. The actual number of bits transmitted depends on the number of MDO pins

2 Maximum information size. The actual number of bits transmitted depends on the number of MDO pins

Figure 24-6. Message Field Sizes

The double edges in Figure 24-6 indicate the starts and ends of messages. Fields without shaded areas between them are grouped into super-fields and can be transmitted together without end-of-packet indications between them.

24.7.2.2.1 Rules of Messages

The rules of messages include the following:

- A variable-sized field within a message must end on a port boundary. (Port boundaries depend on the number of MDO pins active with the current reset configuration.)
- A variable-sized field can start within a port boundary only when following a fixed-length field.
- Super-fields must end on a port boundary.
- When a variable-length field is sized such that it does not end on a port boundary, it is necessary to extend and zero fill the remaining bits after the highest order bit so that it can end on a port boundary.
- Multiple fixed-length packets can start and/or end on a single clock.
- When any packet follows a variable-length packet, it must start on a port boundary.
- The field containing the TCODE number is always transferred out first, followed by subsequent fields of information.
- Within a field, the lowest significant bits are shifted out first. Figure 24-7 shows the transmission sequence of a message that is made up of a TCODE followed by three fields.

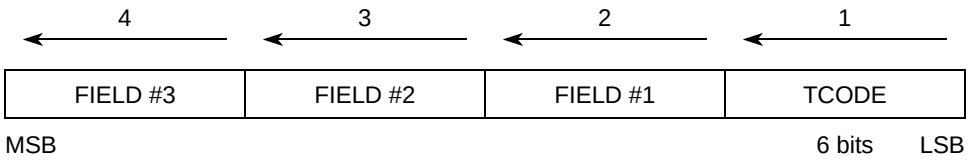


Figure 24-7. Transmission Sequence of Messages

24.7.2.3 IEEE® 1149.1-2001 (JTAG) TAP

The NPC uses the IEEE® 1149.1-2001 TAP for accessing registers. Each of the individual Nexus modules on the device implements a TAP controller for accessing its registers as well. TAP signals include TCK, TDI, TMS, and TDO. Detailed information about the TAP controller state machine can be found in Section 24.4.3, “TAP Controller State Machine.”

The IEEE® 1149.1-2001 specification can be ordered for further detail on electrical and pin protocol compliance requirements.

The NPC implements a Nexus controller state machine that transitions based on the state of the IEEE® 1149.1-2001 state machine shown in Figure 24-5. The Nexus controller state machine is defined by the IEEE-ISTO 5001-2003 standard. It is shown in Figure 24-10.

The instructions implemented by the NPC TAP controller are listed in Table 24-14. The value of the NEXUS-ENABLE instruction is 0b0000. Each unimplemented instruction acts like the BYPASS instruction. The size of the NPC instruction register is 4-bits.

Table 24-14. Implemented Instructions

Instruction Name	Private/Public	Opcode	Description
NEXUS-ENABLE	Public	0x0	Activate Nexus controller state machine to read and write NPC registers.
BYPASS	Private	0xF	NPC BYPASS instruction. Also the value loaded into the NPC IR upon exit of reset.

Data is shifted between TDI and TDO starting with the least significant bit as illustrated in Figure 24-8. This applies for the instruction register and all Nexus tool-mapped registers.

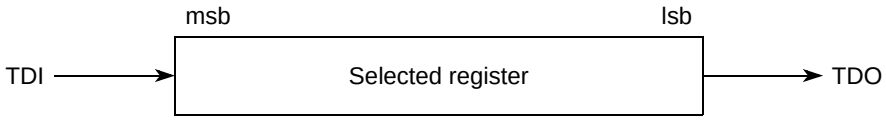
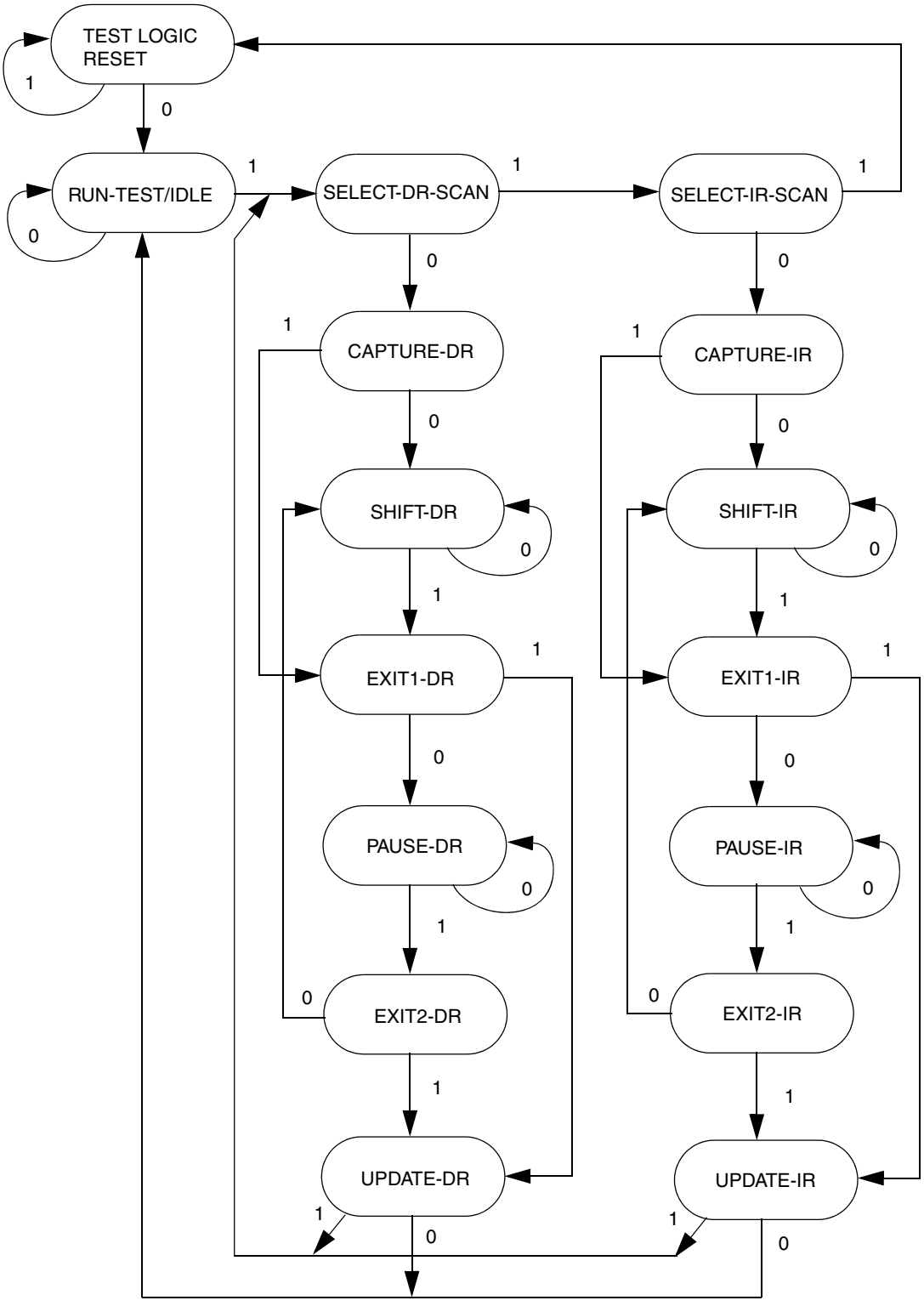


Figure 24-8. Shifting Data Into a Register

24.7.2.3.1 Enabling the NPC TAP Controller

Assertion of the power-on reset signal, entry into censored mode, or negating JCOMP resets the NPC TAP controller. When not in power-on reset or censored mode, the NPC TAP controller is enabled by asserting JCOMP and loading the ACCESS_AUX_TAP_NPC instruction in the JTAGC. Loading the NEXUS-ENABLE instruction then grants access to NPC registers.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 24-9. IEEE 1149.1-2001 TAP Controller State Machine

24.7.2.3.2 Retrieving Device IDCODE

The Nexus TAP controller does not implement the IDCODE instruction. However, the device identification message can be output by the NPC through the auxiliary output port or shifted out serially by accessing the NPC device ID register through the TAP. If the NPC is enabled, transmission of the device identification message on the auxiliary output port MDO pins occurs immediately after a write to the PCR. Transmission of the device identification message serially through TDO is achieved by performing a read of the register contents as described in Section 24.7.2.3.4, "Selecting a Nexus Client Register."

24.7.2.3.3 Loading NEXUS-ENABLE Instruction

Access to the NPC registers is enabled by loading the NPC NEXUS-ENABLE instruction when NPC has ownership of the TAP. This instruction is shifted in via the SELECT-IR-SCAN path and loaded in the UPDATE-IR state. At this point, the Nexus controller state machine, shown in Figure 24-10, transitions to the REG_SELECT state. The Nexus controller has three states: idle, register select, and data access. Table 24-15 illustrates the IEEE® 1149.1 sequence to load the NEXUS-ENABLE instruction.

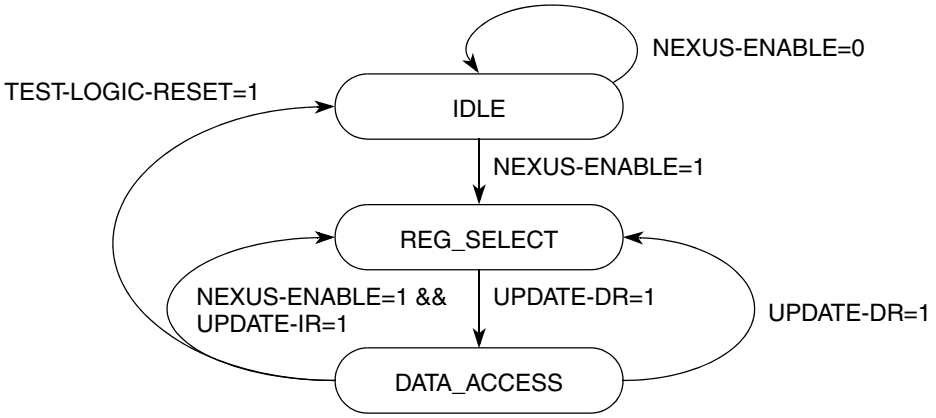


Figure 24-10. NEXUS Controller State Machine

Table 24-15. Loading NEXUS-ENABLE Instruction

Clock	TDI	TMS	IEEE® 1149.1 State	Nexus State	Description
0	—	0	RUN-TEST/IDLE	IDLE	IEEE 1149.1-2001 TAP controller in idle state
1	—	1	SELECT-DR-SCAN	IDLE	Transitional state
2	—	1	SELECT-IR-SCAN	IDLE	Transitional state
3	—	0	CAPTURE-IR	IDLE	Internal shifter loaded with current instruction
4	—	0	SHIFT-IR	IDLE	TDO becomes active, and the IEEE® 1149.1-2001 shifter is ready. Shift in all but the last bit of the NEXUS_ENABLE instruction.
5–7	0	0	3 TCKS in SHIFT-IR	IDLE	
8	0	1	EXIT1-IR	IDLE	Last bit of instruction shifted in
9	—	1	UPDATE-IR	IDLE	NEXUS-ENABLE loaded into instruction register
10	—	0	RUN-TEST/IDLE	REG_SELECT	Ready to be read/write Nexus registers

24.7.2.3.4 Selecting a Nexus Client Register

When the NEXUS-ENABLE instruction is decoded by the TAP controller, the input port allows development tool access to all Nexus registers. Each register has a 7-bit address index.

All register access is performed via the SELECT-DR-SCAN path of the IEEE® 1149.1–2001 TAP controller state machine. The Nexus controller defaults to the REG_SELECT state when enabled. Accessing a register requires two passes through the SELECT-DR-SCAN path: one pass to select the register and the second pass to read/write the register.

The first pass through the SELECT-DR-SCAN path is used to enter an 8-bit Nexus command consisting of a read/write control bit in the lsb followed by a 7-bit register address index, as illustrated in Figure 24-11. The read/write control bit is set to 1 for writes and 0 for reads.

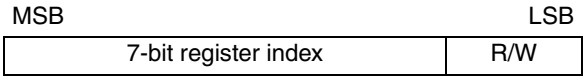


Figure 24-11. IEEE® 1149.1 Controller Command Input

The second pass through the SELECT-DR-SCAN path is used to read or write the register data by shifting in the data (lsb first) during the SHIFT-DR state. When reading a register, the register value is loaded into the IEEE® 1149.1-2001 shifter during the CAPTURE-DR state. When writing a register, the value is loaded from the IEEE® 1149.1-2001 shifter to the register during the UPDATE-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting can be terminated after the required number of bits have been acquired.

Table 24-16 illustrates a sequence that writes a 32-bit value to a register.

Table 24-16. Write to a 32-Bit Nexus Client Register

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
0	0	RUN-TEST/IDLE	REG_SELECT	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	REG_SELECT	First pass through SELECT-DR-SCAN path
2	0	CAPTURE-DR	REG_SELECT	Internal shifter loaded with current value of controller command input.
3	0	SHIFT-DR	REG_SELECT	TDO becomes active, and write bit and 6 bits of register index shifted in.
7 TCKs				
11	1	EXIT1-DR	REG_SELECT	Last bit of register index shifted into TDI
12	1	UPDATE-DR	REG_SELECT	Controller decodes and selects register
13	1	SELECT-DR-SCAN	DATA_ACCESS	Second pass through SELECT-DR-SCAN path
14	0	CAPTURE-DR	DATA_ACCESS	Internal shifter loaded with current value of register
15	0	SHIFT-DR	DATA_ACCESS	TDO becomes active, and outputs current value of register while new value is shifted in through TDI
31 TCKs				
47	1	EXIT1-DR	DATA_ACCESS	Last bit of current value shifted out TDO. Last bit of new value shifted in TDI.

Table 24-16. Write to a 32-Bit Nexus Client Register (continued)

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
48	1	UPDATE-DR	DATA_ACCESS	Value written to register
49	0	RUN-TEST/IDLE	REG_SELECT	Controller returned to idle state. It could also return to SELECT-DR-SCAN to write another register.

24.7.2.4 Nexus Auxiliary Port Sharing

Each of the Nexus modules on the MCU implements a request/grant scheme to arbitrate for control of the Nexus auxiliary port when Nexus data is ready to be transmitted.

All modules arbitrating for the port are given fixed priority levels relative to each other. If multiple modules have the same request level, this priority level is used as a tie-breaker. To avoid monopolization of the port, the module given the highest priority level alternates following each grant. Immediately out of reset the order of priority, from highest to lowest, is: NPC, NZ3C3, NSEDI. This arbitration mechanism is controlled internally and is not programmable by tools or application software.

24.7.2.5 Nexus JTAG Port Sharing

Each of the individual Nexus modules on the device implements a TAP controller for accessing its registers. When JCOMP is asserted, only the module whose ACCESS_AUX_TAP instruction is loaded has control of the TAP (see Section 24.4.4, “JTAGC Instructions”). This allows the interface to all of these individual TAP controllers to appear to be a single port from outside the device. After a Nexus module has ownership of the TAP, that module acts like a single-bit shift register, or bypass register, if no register is selected as the shift path.

24.7.2.6 MCKO

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock and its frequency is determined by the value of the MCKO_DIV[2:0] field in the PCR. Possible operating frequencies include one-half, one-quarter, and one-eighth system clock speed. MCKO is enabled by setting the MCKO_EN bit in the PCR.

The NPC also controls dynamic MCKO clock gating when in full- or reduced-port modes. The setting of the MCKO_GT bit inside the PCR determines whether or not MCKO gating control is enabled. The MCKO_GT bit resets to a logic 0. In this state gating of MCKO is disabled. To enable gating of MCKO, the MCKO_GT bit in the PCR is written to a logic 1. When MCKO gating is enabled, MCKO is driven to a logic 0 if the auxiliary port is enabled but not transmitting messages and there are no pending messages from Nexus clients.

24.7.2.7 $\overline{\text{EVTO}}$ Sharing

The NPC controls sharing of the $\overline{\text{EVTO}}$ output between all Nexus clients that produce an $\overline{\text{EVTO}}$ signal. $\overline{\text{EVTO}}$ is driven for one MCKO period whenever any module drives its $\overline{\text{EVTO}}$. When there is no active MCKO, such as in disabled mode, the NPC assumes an MCKO frequency of one-half system clock speed when driving $\overline{\text{EVTO}}$. $\overline{\text{EVTO}}$ sharing is active as long as the NPC is not in reset.

24.7.2.8 Nexus Reset Control

The JCOMP input that is used as the primary reset signal for the NPC is also used by the NPC to generate a single-bit reset signal for other Nexus modules. If JCOMP is negated, an internal reset signal is asserted, indicating that all Nexus modules must be held in reset. This internal reset signal is also asserted during a power-on reset, or if nex_disable is asserted (SIU_CCR[DISNEX]), indicating the device is in censored mode. This single bit reset signal functions much like the IEEE® 1149.1-2001 defined $\overline{\text{TRST}}$ signal and allows JCOMP reset information to be provided to the Nexus modules without each module having to sense the JCOMP signal directly or monitor the status of censored mode.

24.8 NPC Initialization and Application Information

24.8.1 Accessing NPC Tool-Mapped Registers

To initialize the TAP for NPC register accesses, the following sequence is required:

1. Enable the NPC TAP controller. This is achieved by asserting JCOMP and loading the ACCESS_AUX_TAP_NPC instruction in the JTAGC.
2. Load the TAP controller with the NEXUS-ENABLE instruction.

To write control data to NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and set the write bit to select the register with a pass through the SELECT-DR-SCAN path in the TAP controller state machine.
2. Write the register value with a second pass through the SELECT-DR-SCAN path. The prior value of this register is shifted out during the write.

To read status and control data from NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and clear the write bit to select register with a pass through SELECT-DR-SCAN path in the TAP controller state machine.
2. Read the register value with a second pass through the SELECT-DR-SCAN path. Data shifted in is ignored.

See the IEEE®-ISTO 5001-2003 standard for more detail.

24.9 Nexus Single eTPU Development Interface (NSEDI)

The enhanced timing processor unit (eTPU) has its own Nexus class 3 interface, the Nexus single eTPU development interface (NSEDINDEDI). The single eTPU engine and a coherent parameter controller (CDC) appear as two separate Nexus clients. See the *Enhanced Time Processor Unit Reference Manual* for more information about the NSEDINDEDI module.

Reg Index: 0 Access: R/O

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PRN				DC						PIN					
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIN				MIC											1
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 24-12. NSEDI Device ID Register (DID)

Table 24-17. NSEDI DID Register Field Descriptions

Field	Description
31–28 PRN	Part revision number. Contains the revision number of the part. This field changes with each revision of the device or module.
27–22 DC	Design center. Indicates the Freescale design center. This value is 0x20.
21–12 PIN	Part identification number. Contains the part number of the device. The PIN for the MPC5534 is 0x124.
11–1 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0xE.
0	Fixed per JTAG 1149.1 1 Always set

24.10 e200z3 Class 3 Nexus Module (NZ3C3)

The NZ3C3 module provides real-time development capabilities for the device core in compliance with the IEEE®-ISTO Nexus 5001-2003 standard. This module provides development support capabilities without requiring the use of address and data pins for internal visibility.

24.10.1 Introduction

This section defines the auxiliary pin functions, transfer protocols and standard development features of the NZ3C3 module. The development features supported are Program trace, data trace, watchpoint messaging, ownership trace, and read/write access via the JTAG interface.

NOTE

Throughout this section references are made to the auxiliary port and its specific signals, such as MCKO, $\overline{\text{MSEO}}$[1:0], MDO[11:0] and others. The device NPC module arbitrates the access of the single auxiliary port. To simplify the description of the function of the NZ3C3 module, the interaction of the NPC is omitted and the configuration in this chapter describes an NPC with a dedicated auxiliary port. The auxiliary port is fully described in Section 24.2, "External Signal Description."

24.10.2 Block Diagram

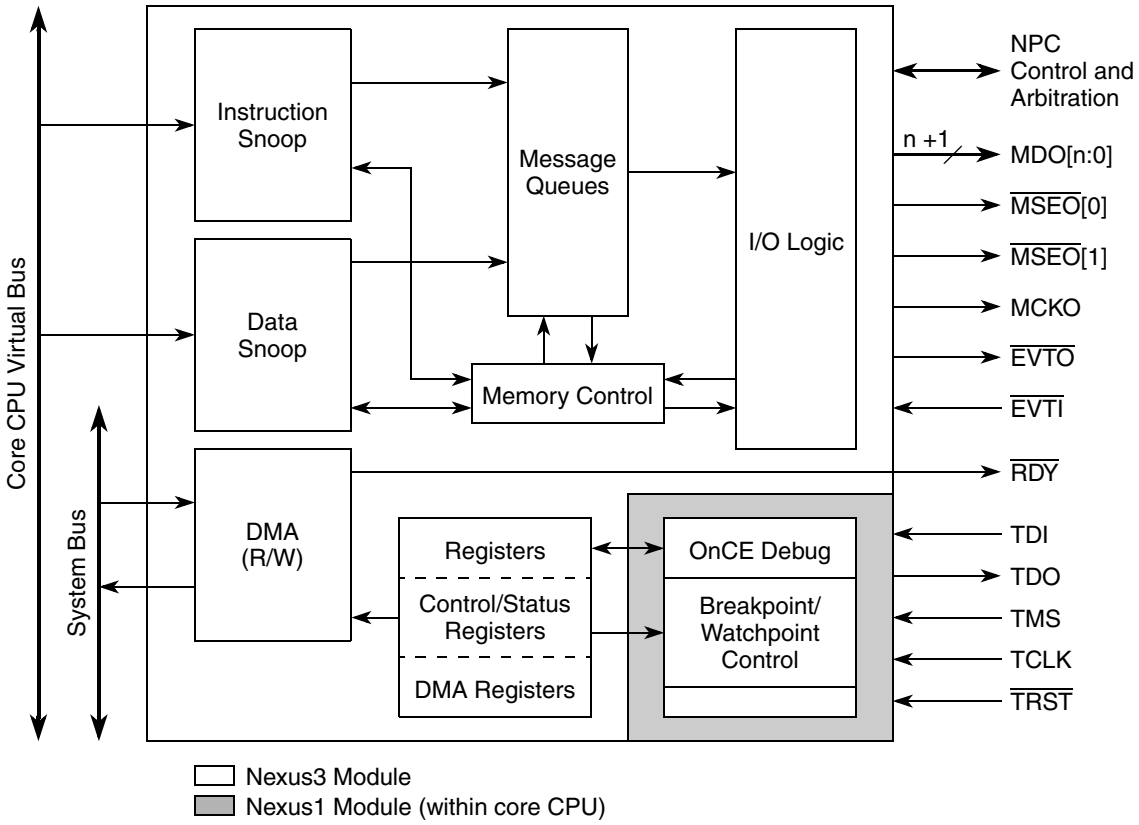


Figure 24-13. e200z3 Nexus3 Functional Block Diagram

24.10.3 Overview

Table 24-18 contains a set of terms and definitions associated with the NZ3C3 module.

Table 24-18. Terms and Definitions

Term	Description
IEEE®-ISTO 5001	Consortium and standard for real-time embedded system design. World wide Web documentation at http://www.ieee-isto.org/Nexus5001
Auxiliary Port	Refers to Nexus auxiliary port. Used as auxiliary port to the IEEE® 1149.1 JTAG interface.
Branch Trace Messaging (BTM)	Visibility of addresses for taken branches and exceptions, and the number of sequential instructions executed between each taken branch.
Client	A functional block on an embedded processor which requires development visibility and controllability. Examples are a central processing unit (CPU) or an intelligent peripheral.
Data Read Message (DRM)	External visibility of data reads to memory-mapped resources.
Data Write Message (DWM)	External visibility of data writes to memory-mapped resources.
Data Trace Messaging (DTM)	External visibility of how data flows through the embedded system. This can include DRM and/or DWM.
JTAG Compliant	Device complying to IEEE® 1149.1 JTAG standard
JTAG IR & DR Sequence	JTAG instruction register (IR) scan to load an opcode value for selecting a development register. The JTAG IR corresponds to the OnCE command register (OCMD). The selected development register is then accessed via a JTAG data register (DR) scan.
Nexus1	The e200z3 (OnCE) debug module. This module integrated with each e200z3 processor provides all static (core halted) debug functionality. This module is compliant with Class1 of the IEEE®-ISTO 5001 standard.
Ownership Trace Message (OTM)	Visibility of process/function that is currently executing.
Public Messages	Messages on the auxiliary pins for accomplishing common visibility and controllability requirements
Standard	The phrase ‘according to the standard’ is used to indicate according to the IEEE®-ISTO 5001 standard.
Transfer Code (TCODE)	Message header that identifies the number and/or size of packets to be transferred, and how to interpret each of the packets.
Watchpoint	A data or instruction breakpoint which does not cause the processor to halt. Instead, a pin is used to signal that the condition occurred. A watchpoint message is also generated.

24.10.4 Features

The NZ3C3 module is compliant with Class 3 of the IEEE®-ISTO 5001-2003 standard. The following features are implemented:

- Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code can be traced.
- Data trace via data write messaging (DWM) and data read messaging (DRM). This provides the capability for the development tool to trace reads and/or writes to selected internal memory resources.
- Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.
- Run-time access to embedded processor registers and memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint messaging via the auxiliary pins.
- Watchpoint trigger enable of program and/or data trace messaging.
- High-speed data input/output via the auxiliary port.
- Auxiliary interface for higher data input/output
  - Configurable (minimum and maximum) message data out pins (nex_mdo[n:0])
  - One or two message start/end out pins (nex_mseo_b[1:0])
  - One read/write ready pin (nex_rdy_b) pin
  - One watchpoint-event pin (nex_evto_b)
  - One event-in pin (nex_evti_b)
  - One MCKO (message clock out) pin
- Registers for program trace, data trace, ownership trace and watchpoint trigger.
- All features controllable and configurable via the JTAG port.

24.10.5 Enabling Nexus3 Operation

The Nexus module is enabled by loading a single instruction (ACCESS_AUX_TAP_ONCE, as shown in Table 24-4) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS3_ACCESS instruction (see Table 24-5). For the e200z3 Class 3 Nexus module, the OCMD value is 0b00_0111_1100. After it is enabled, the module is ready to accept control input via the JTAG pins. See Section 24.7, “NPC Functional Description” for more information.

The Nexus module is disabled when the JTAG state machine reaches the test-logic-reset state. This state can be reached by asserting the JCOMP pin or cycling through the state machine using the TMS pin. The Nexus module is also disabled if a power-on-reset (POR) event occurs. If the Nexus3 module is disabled, no trace output is provided, and the module disables (drive inactive) auxiliary port output pins MDO[n:0], $\overline{\text{MSEO}}$[1:0], MCKO. Nexus registers are not available for reads or writes.

24.10.6 TCODEs Supported by NZ3C3

The Nexus3 pins allow for flexible transfer operations via public messages. A TCODE defines the transfer format, the number and/or size of the packets to be transferred, and the purpose of each packet. The IEEE®-ISTO 5001-2003 standard defines a set of public messages. The NZ3C3 module supports the public TCODEs seen in Table 24-19. Each message contains multiple packets transmitted in the order shown in the table.

Table 24-19. Public TCODEs Supported by NZ3C3

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Debug Status	6	6	TCODE	Fixed	TCODE number = 0 (0x00)
	4	4	SRC	Fixed	Source processor identifier
	8	8	STATUS	Fixed	Debug status register (DS[31:24])
Ownership Trace Message	6	6	TCODE	Fixed	TCODE number = 2 (0x02)
	4	4	SRC	Fixed	Source processor identifier
	32	32	PROCESS	Fixed	Task/Process ID tag
Program Trace - Direct Branch Message[1]	6	6	TCODE	Fixed	TCODE number = 3 (0x03)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
Program Trace - Indirect Branch Message[1]	6	6	TCODE	Fixed	TCODE number = 4 (0x04)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	U-ADDR	Variable	Unique part of target address for taken branches/exceptions
Data Trace - Data Write Message	6	6	TCODE	Fixed	TCODE number = 5 (0x05)
	4	4	SRC	Fixed	Source processor identifier
	3	3	DSIZ	Fixed	Data size (see Table 24-23)
	1	32	U-ADDR	Variable	Unique portion of the data write address
	1	64	DATA	Variable	Data write values (see Section 24.14.6, “Data Trace,” for details)

Table 24-19. Public TCODEs Supported by NZ3C3 (continued)

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Data Trace - Data Read Message	6	6	TCODE	Fixed	TCODE number = 6 (0x06)
	4	4	SRC	Fixed	Source processor identifier
	3	3	DSIZ	Fixed	Data size (see Table 24-23)
	1	32	U-ADDR	Variable	Unique portion of the data read address
	1	64	DATA	Variable	Data read values (see Section 24.14.6, "Data Trace," for details)
Error Message	6	6	TCODE	Fixed	TCODE number = 8 (0x08)
	4	4	SRC	Fixed	Source processor identifier
	5	5	ECODE	Fixed	Error code
Program Trace - Direct Branch Message w/ Sync[1]	6	6	TCODE	Fixed	TCODE number = 11 (0x0B)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	Full target address (leading zeros truncated)
Program Trace - Indirect Branch Message w/ Sync[1]	6	6	TCODE	Fixed	TCODE number = 12 (0x0C)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	Full target address (leading zeros truncated)
Data Trace - Data Write Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 13 (0x0D)
	4	4	SRC	Fixed	Source processor identifier
	3	3	DSZ	Fixed	Data size (see Table 24-23)
	1	32	F-ADDR	Variable	Full access address (leading zeros truncated)
	1	64	DATA	Variable	Data write values (see Section 24.14.6, "Data Trace," for details)
Data Trace - Data Read Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 14 (0x0E)
	4	4	SRC	Fixed	Source processor identifier
	3	3	DSZ	Fixed	Data size (see Table 24-23)
	1	32	F-ADDR	Variable	Full access address (leading zeros truncated)
	1	64	DATA	Variable	Data read values (see Section 24.14.6, "Data Trace," for details)

Table 24-19. Public TCODEs Supported by NZ3C3 (continued)

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Watchpoint Message	6	6	TCODE	Fixed	TCODE number = 15 (0x0F)
	4	4	SRC	Fixed	Source processor identifier
	4	4	WPHIT	Fixed	Number indicating watchpoint sources
Resource Full Message	6	6	TCODE	Fixed	TCODE number = 27 (0x1B)
	4	4	SRC	Fixed	Source processor identifier
	4	4	RCODE	Fixed	Resource code indicates which resource is the cause of this message (See RCODE values in Table 24-22)
	1	32	RDATA	Variable	Branch / predicate instruction history (see Section 24.13.1, “Branch Trace Messaging (BTM)”)
Program Trace - Indirect Branch History Message	6	6	TCODE	Fixed	TCODE number = 28 (0x1C) (see footnote 1 below)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	U-ADDR	Variable	Unique part of target address for taken branches/exceptions
	1	32	HIST	Variable	Branch / predicate instruction history (see Section 24.13.1, “Branch Trace Messaging (BTM)”)
Program Trace - Indirect Branch History Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 29 (0x1D) (see footnote 1 below)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	Full target address (leading zero (0) truncated)
	1	32	HIST	Variable	Branch / predicate instruction history (see Section 24.13.1, “Branch Trace Messaging (BTM)”)
Program Trace - Program Correlation Message	6	6	TCODE	Fixed	TCODE number = 33 (0x21)
	4	4	SRC	Fixed	Source processor identifier
	4	4	EVCODE	Fixed	Event correlated w/ program flow (see Table 24-22)
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	HIST	Variable	Branch / predicate instruction history (see Section 24.13.1, “Branch Trace Messaging (BTM)”)

1 You can select between the two types of program trace. The advantages for each are discussed in Section 24.13.1, “Branch Trace Messaging (BTM). If the branch history method is selected, the shaded TCODES above are not messaged out.

Table 24-20 shows the error code encodings used when reporting an error via the Nexus3 Error Message.

Table 24-20. Error Code Encoding (TCODE = 8)

Error Code (ECODE)	Description
00000	Ownership trace overrun
00001	Program trace overrun
00010	Data trace overrun
00011	Read/write access error
00101	Invalid access opcode (Nexus register unimplemented)
00110	Watchpoint overrun
00111	(Program trace or data trace) and ownership trace overrun
01000	(Program trace or data trace or ownership trace) and watchpoint overrun
01001–10111	Invalid value
11000	BTM lost due to collision w/ higher priority message
11001–11111	Invalid value

Table 24-21 shows the encodings used for resource codes for certain messages.

Table 24-21. RCODE values (TCODE = 27)

Resource Code (RCODE)	Description	Resource Data (RDATA)
0000	Program Trace Instruction Counter overflow (reached 255 and was reset)	0xFF
0001	Program Trace, Branch and Predicate Instruction History. This type of packet is terminated by a stop bit set to 1 after the last history bit.	Branch History. This type of packet is terminated by a stop bit set to a 1 after the last history bit.

Table 24-22 shows the event code encodings used for certain messages.

Table 24-22. Event Code Encoding (TCODE = 33)

Event Code	Description
0000	Entry into Debug Mode
0001	Entry into Low Power Mode (CPU only)[1]
0010–0011	Invalid value. Reserved for future functionality
0100	Disabling Program Trace
0101–1101	Invalid value. Reserved for future functionality
1110	Entry into a VLE page from a non-VLE page
1111	Entry into a non-VLE page from a VLE page

[1] The device enters Low Power Mode when the Nexus stall mode is enabled (NZ3C3_DC1[OVC]=0b011) and a trace message is in danger of over-flowing the Nexus queue.

Table 24-23 shows the data trace size encodings used for certain messages.

Table 24-23. Data Trace Size Encodings (TCODE = 5, 6, 13, 14)

DTM Size Encoding	Transfer Size
000	Byte
001	Halfword (two bytes)
010	Word (four bytes)
011	Doubleword (eight bytes)
100	String (three bytes)
101–111	Invalid value

NOTE

Program trace can be implemented using either branch history/predicate instruction messages, or traditional direct and indirect branch messages. You can select between the two types of Program Trace. The advantages for each are discussed in Section 24.13.1, “Branch Trace Messaging (BTM). If the Branch History method is selected, the shaded TCODES are not messaged out.

24.11 NZ3C3 Memory Map and Register Definition

This section describes the NZ3C3 programmer’s model. NZ3C3 registers are accessed using the JTAG/OnCE port in compliance with IEEE® 1149.1. See Section 24.11.11, “ NZ3C3 Register Access via JTAG / OnCE” for details on NZ3C3 register access.

NOTE

NZ3C3 registers and output signals are numbered using bit 0 as the least significant bit. This bit ordering is consistent with the ordering defined by the IEEE®-ISTO 5001 standard.

Table 24-24 details the register map for the NZ3C3 module.

Table 24-24. NZ3C3 Memory Map

Access Opcode	Register Name	Register Description	Read Address	Write Address
0x0001	CSC	Client select control [1]	0x0002	—
See NPC	PCR	Port configuration register [1]	—	—
0x0002	DC1	Development control 1	0x0004	0x0005
0x0003	DC2	Development control 2	0x0006	0x0007
0x0004	DS	Development status	0x0008	—
0x0007	RWCS	Read/write access control/status	0x000E	0x000F
0x0009	RWA	Read/write access address	0x0012	0x0013
0x000A	RWD	Read/write access data	0x0014	0x0015

Table 24-24. NZ3C3 Memory Map (continued)

Access Opcode	Register Name	Register Description	Read Address	Write Address
0x000B	WT	Watchpoint trigger	0x0016	0x0017
0x000D	DTC	Data trace control	0x001A	0x001B
0x000E	DTSA1	Data trace start address 1	0x001C	0x001D
0x000F	DTSA2	Data trace start address 2	0x001E	0x001F
0x0012	DTEA1	Data trace end address 1	0x0024	0x0025
0x0013	DTEA2	Data trace end address 2	0x0026	0x0027
0x0014–0x003F	—	Reserved	0x0028–0x007E	0x0029–0x007F

[1] The CSC and PCR registers are shown in this table as part of the Nexus programmer's model. They are only present at the top level Nexus3 controller (NPC), not in the NZ3C3 module. The device's CSC register is readable through Nexus3, but the PCR is shown for reference only.

24.11.1 Port Configuration Register (PCR)

The Port Configuration Register (PCR) controls the basic port functions for all Nexus modules in a multi-Nexus environment. This includes clock control and auxiliary port width. All bits in this register are writable only once after system reset.

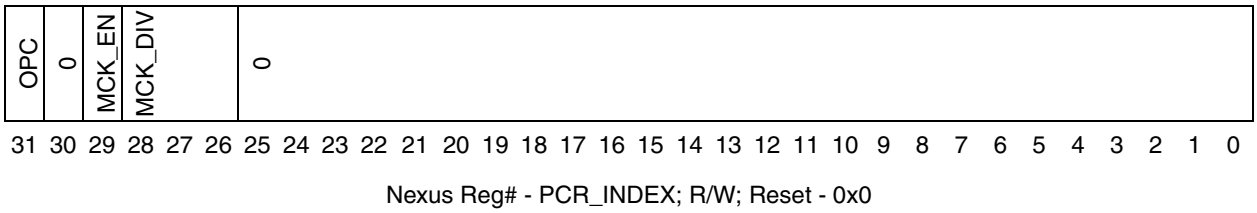


Figure 24-14. Port Configuration Register

Table 24-25. Port Configuration Register Fields

PCR[31]	OPC	OPC — Output Port Mode Control
		0 = Reduced Port Mode configuration (minimum # nex_mdo[n:0] pins)
		1 = Full Port Mode configuration (max# nex_mdo[n:0] pins defined by SOC)

PCR[30]	—	Invalid value. Reserved for future functionality
PCR[29]	MCK_EN	MCK_EN — MCKO Clock enable 0 = nex_mcko is disabled 1 = nex_mcko is enabled
PCR[28:26]	MCK_DIV	MCK_DIV — MCKO Clock Divide Ratio (read the NOTE after this table) 000 = nex_mcko is 1x processor clock frequency. 001 = nex_mcko is 1/2x processor clock frequency. 010 = Reserved (defaults to 1/2x processor clock frequency.) 011 = nex_mcko is 1/4x processor clock frequency. 100–110 = Reserved (defaults to 1/2x processor clock frequency.) 111 = nex_mcko is 1/8x processor clock frequency.
PCR[25:0]	—	Invalid value. Reserved for future functionality

NOTE

The CSC and PCR registers are in separate system module for a multi-Nexus environment. If the e200z3 Nexus3 module is the only Nexus module, the CSC and PCR registers are not implemented, and the e200z3 Nexus3 development control register 1 (DC1) set the Nexus port functionality.

24.11.2 Development Control Registers 1 and 2 (DC1, DC2)

The development control registers are used to control the basic development features of the NZ3C3 module. Development control register 1 is shown in Figure 24-15 and its fields are described in Table 24-26.

Nexus Reg: 0x0002 Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OPC	MCK_DIV		EOC		0	PTM	WEN	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									OVC			EIC		TM		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-15. Development Control Register 1 (DC1)

Table 24-26. DC1 Field Descriptions

Field	Description
31 OPC[1]	Output port mode control. 0 Reduced-port mode configuration (four MDO pins) 1 Full-port mode configuration (12 MDO pins)
30–29 MCK_DIV [1:0][1]	MCKO clock divide ratio (see note below). 00 MCKO is 1x processor clock frequency. 01 MCKO is 1/2x processor clock frequency. 10 MCKO is 1/4x processor clock frequency. 11 MCKO is 1/8x processor clock frequency.
28–27 EOC[1:0]	$\overline{\text{EVTO}}$ control. 00 $\overline{\text{EVTO}}$ upon occurrence of watchpoints (configured in DC2) 01 $\overline{\text{EVTO}}$ upon entry into debug mode 10 $\overline{\text{EVTO}}$ upon time-stamping event 11 Invalid value
26	Reserved
25 PTM	Program trace method. 0 Program trace uses traditional branch messages 1 Program trace uses branch history messages

Table 24-26. DC1 Field Descriptions (continued)

Field	Description
24 WEN	Watchpoint trace enable. 0 Watchpoint Messaging disabled 1 Watchpoint Messaging enabled
23–8	Reserved
7–5 OVC[2:0]	Overrun control. 000 Generate overrun messages 001–010 Invalid value 011 Delay processor for BTM / DTM / OTM overruns 1XX Invalid value
4–3 EIC[1:0]	EVTI control. 00 EVTI is used for synchronization (program trace/ data trace) 01 EVTI is used for debug request 1X Invalid value
2–0 TM[2:0]	Trace mode. Any or all of the TM bits can be set, enabling one or more traces. 000 No trace 1XX Program trace enabled X1X Data trace enabled XX1 Ownership trace enabled

[1] The output port mode control bit (OPC) and MCKO divide bits (MCK_DIV) are shown for clarity. These functions are controlled globally by the NPC port control register (PCR). These bits are writable in the PCR but have no effect.

Development control register 2 is shown in Figure 24-16 and its fields are described in Table 24-27.

Nexus Reg: 0x0003 Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EWC								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-16. Development Control Register 2 (DC2)

Table 24-27. DC2 Field Descriptions

Field	Description
31–24 EWC[7:0]	$\overline{\text{EVTO}}$ watchpoint configuration. Any or all of the bits in EWC can be set to configure the $\overline{\text{EVTO}}$ watchpoint. 00000000No Watchpoints trigger $\overline{\text{EVTO}}$ 1XXXXXXXWatchpoint #0 (IAC1 from Nexus1) triggers $\overline{\text{EVTO}}$ X1XXXXXXWatchpoint #1 (IAC2 from Nexus1) triggers $\overline{\text{EVTO}}$ XX1XXXXXWatchpoint #2 (IAC3 from Nexus1) triggers $\overline{\text{EVTO}}$ XXX1XXXXWatchpoint #3 (IAC4 from Nexus1) triggers $\overline{\text{EVTO}}$ XXXX1XXXWatchpoint #4 (DAC1 from Nexus1) triggers $\overline{\text{EVTO}}$ XXXXX1XXWatchpoint #5 (DAC2 from Nexus1) triggers $\overline{\text{EVTO}}$ XXXXXX1XWatchpoint #6 (DCNT1 from Nexus1) triggers $\overline{\text{EVTO}}$ XXXXXXX1Watchpoint #7 (DCNT2 from Nexus1) triggers $\overline{\text{EVTO}}$
23–0	Reserved

NOTE

The EOC bits in DC1 must be programmed to trigger $\overline{\text{EVTO}}$ on watchpoint occurrence for the EWC bits to have any effect.

24.11.3 Development Status Register (DS)

The development status register is used to report system debug status. When debug mode is entered or exited, or an e200z3-defined low power mode is entered, a debug status message is transmitted with DS[31:24]. The external tool can read this register at any time.

Nexus Reg: 0x0004 Access: R/O

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DBG	0	0	0	LPC		CHK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-17. Development Status Register (DS)

Table 24-28. DS Field Descriptions

Field	Description
31–28 DBG	Bit 31 is the e200z3 CPU debug mode transition status. Bits 31–28 are sent as the debug status message. 0 CPU not in debug mode 1 CPU in debug mode
27–26 LPC[1:0]	e200z3 CPU low power mode status. 00 Normal (run) mode 01 CPU in halted state 10 CPU in stopped state 11 Invalid value
25 CHK	e200z3 CPU checkstop status. 0 CPU not in checkstop state 1 CPU in checkstop state
24–0	Reserved

24.11.4 Read/Write Access Control and Status (RWCS)

The read write access control/status register provides control for read/write access. Read/write access provides DMA-like access to memory-mapped resources on the system bus either while the processor is halted, or during runtime. The RWCS register also provides read/write access status information as shown in Table 24-30.

Nexus Reg: 0x0007 Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AC	RW	SZ			MAP			PR		BST	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CNT														ERR	DV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-18. Read/Write Access Control and Status Register (RWCS)

Table 24-29 describes the fields and functions in the read/write control and status (RWCS) register:

Table 24-29. RWCS Field Description

Field	Description
31 AC	Access control. 0 End access 1 Start access
30 RW	Read/write select. 0 Read access 1 Write access
29–27 SZ[2:0]	Word size. 000 8-bit (byte) 001 6-bit (halfword) 010 32-bit (word) 011 64-bit (doubleword - only in burst mode) 100–111 Invalid value (default to word)
26–24 MAP[2:0]	MAP select. 000 Primary memory map 001–111 Invalid value
23–22 PR[1:0]	Read/write access priority. 00 Lowest access priority 01 Invalid value (default to lowest priority) 10 Invalid value (default to lowest priority) 11 Highest access priority
21 BST	Burst control. 0 Module accesses are single bus cycle at a time. 1 Module accesses are performed as burst operation.
20–16	Reserved
15–2 CNT[13:0]	Access control count. Number of accesses of word size SZ
1 ERR	Read/write access error. See Table 24-30.
0 DV	Read/write access data valid. See Table 24-30.

Table 24-30 details the status bit encodings.

Table 24-30. Read/Write Access Status Bit Encoding

Read Action	Write Action	ERR	DV
Read access has not completed	Write access completed without error	0	0
Read access error has occurred	Write access error has occurred	1	0
Read access completed without error	Write access has not completed	0	1
Not allowed	Not allowed	1	1

24.11.5 Read/Write Access Address (RWA)

The read/write access address register provides the system bus address to be accessed when initiating a read or a write access.

Nexus Reg: 0x0009 Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Read/Write Address																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-19. Read/Write Access Address Register (RWA)

24.11.6 Read/Write Access Data (RWD)

The read/write access data register provides the data to/from system bus memory-mapped locations when initiating a read or a write access.

Nexus Reg: 0x000A Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Read/Write Data																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-20. Read/Write Access Data Register (RWD)

24.11.7 Watchpoint Trigger Register (WT)

The watchpoint trigger register allows the watchpoints defined within the e200z3 Nexus1 logic to trigger actions. These watchpoints can control program and/or data trace enable and disable. The WT bits can be used to produce an address related ‘window’ for triggering trace messages.

Nexus Reg: 0x000B Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PTS				PTE				DTS				DTE				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-21. Watchpoint Trigger Register (WT)

Table 24-31 details the watchpoint trigger register fields.

Table 24-31. WT Field Descriptions

Field	Description
31–29 PTS[2:0]	Program trace start control. 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1) 010 Use watchpoint #1 (IAC2 from Nexus1) 011 Use watchpoint #2 (IAC3 from Nexus1) 100 Use watchpoint #3 (IAC4 from Nexus1) 101 Use watchpoint #4 (DAC1 from Nexus1) 110 Use watchpoint #5 (DAC2 from Nexus1) 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1)
28–26 PTE[2:0]	Program trace end control. 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1) 010 Use watchpoint #1 (IAC2 from Nexus1) 011 Use watchpoint #2 (IAC3 from Nexus1) 100 Use watchpoint #3 (IAC4 from Nexus1) 101 Use watchpoint #4 (DAC1 from Nexus1) 110 Use watchpoint #5 (DAC2 from Nexus1) 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1)
25–23 DTS[2:0]	Data trace start control. 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1) 010 Use watchpoint #1 (IAC2 from Nexus1) 011 Use watchpoint #2 (IAC3 from Nexus1) 100 Use watchpoint #3 (IAC4 from Nexus1) 101 Use watchpoint #4 (DAC1 from Nexus1) 110 Use watchpoint #5 (DAC2 from Nexus1) 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1)
22–20 DTE[2:0]	Data trace end control. 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1) 010 Use watchpoint #1 (IAC2 from Nexus1) 011 Use watchpoint #2 (IAC3 from Nexus1) 100 Use watchpoint #3 (IAC4 from Nexus1) 101 Use watchpoint #4 (DAC1 from Nexus1) 110 Use watchpoint #5 (DAC2 from Nexus1) 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1)
19–0	Reserved

NOTE

The WT bits control program and data trace only if the TM bits in the development control register 1 (DC1) have not been set to enable program and data trace, respectively.

24.11.8 Data Trace Control Register (DTC)

The data trace control register controls whether DTM messages are restricted to reads, writes, or both for a user programmable address range. There are two data trace channels controlled by the DTC for the Nexus3 module. Each channel can also be programmed to trace data accesses or instruction accesses.

Nexus Reg: 0x000D Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RWT1		RWT2										0	0	0	0	0	0	0	0	0	0	0	0	RC1	RC2	0	0	DI1	DI2	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-22. Data Trace Control Register (DTC)

Table 24-32 details the data trace control register fields.

Table 24-32. DTC Field Description

Field	Description
31–30 RWT1[1:0]	Read/write trace 1. 00 No trace enabled X1 Enable data read trace 1X Enable data write trace
29–28 RWT2[1:0]	Read/write trace 2. 00 No trace enabled X1 Enable data read trace 1X Enable data write trace
27–8	Reserved
7 RC1	Range control 1. 0 Condition trace on address within range 1 Condition trace on address outside of range
6 RC2	Range control 2 0 Condition trace on address within range 1 Condition trace on address outside of range
5–4	Reserved
3 DI1	Data access/instruction access trace 1. 0 Condition trace on data accesses 1 Condition trace on instruction accesses
2 DI2	Data access/instruction access trace 2 0 Condition trace on data accesses 1 Condition trace on instruction accesses
1–0	Reserved

24.11.9 Data Trace Start Address Registers 1 and 2 (DTSA*n*)

The data trace start address registers define the start addresses for each trace channel.

Nexus Reg: 0x000E Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Data Trace Start Address																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-23. Data Trace Start Address Register 1 (DTSA1)

Nexus Reg: 0x000F Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Data Trace Start Address																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-24. Data Trace Start Address Register 2 (DTSA2)

24.11.10 Data Trace End Address Registers 1 and 2 (DTEA*n*)

The data trace end address registers define the end addresses for each trace channel.

Nexus Reg: 0x0012 Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Data Trace End Address																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-25. Data Trace End Address Register 1 (DTEA1)

Nexus Reg: 0x0013 Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Data Trace End Address																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-26. Data Trace End Address Register 2 (DTEA2)

Table 24-33 illustrates the range that is selected for data trace for various cases of DTSA being less than, greater than, or equal to DTEA.

Table 24-33. Data Trace—Address Range Options

Programmed Values	Range Control Bit Value	Range Selected
DTSA < DTEA	0	DTSA –> <– DTEA
DTSA < DTEA	1	<– DTSA DTEA –>

Table 24-33. Data Trace—Address Range Options (continued)

Programmed Values	Range Control Bit Value	Range Selected
DTSA > DTEA	—	Invalid range—no trace
DTSA = DTEA	—	Invalid range—no trace

NOTE

DTSA must be less than DTEA to guarantee correct data write/read traces. Data trace ranges are exclusive of the DTSA and DTEA addresses.

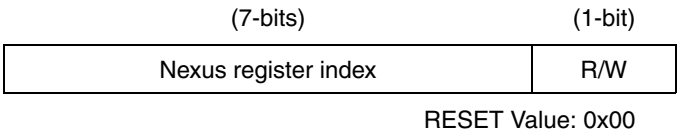
24.11.11 NZ3C3 Register Access via JTAG / OnCE

Access to Nexus3 register resources is enabled by loading a single instruction (ACCESS_AUX_TAP_ONCE) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS3_ACCESS instruction (see Table 24-5). For the NZ3C3 module, the OCMD value is 0b00_0111_1100.

After the ACCESS_AUX_TAP_ONCE instruction has been loaded, the JTAG/OnCE port allows tool/target communications with all Nexus3 registers according to the register map in Table 24-24.

Reading/writing of a NZ3C3 register then requires two (2) passes through the data-scan (DR) path of the JTAG state machine (see 24.14.10).

1. The first pass through the DR selects the NZ3C3 register to be accessed by providing an index (see Table 24-24), and the direction (read/write). This is achieved by loading an 8-bit value into the JTAG data register (DR). This register has the following format:



RESET Value: 0x00

Nexus Register Index:	Selected from values in Table 24-24
Read/Write (R/W):	0 Read 1 Write

2. The second pass through the DR then shifts the data in or out of the JTAG port, lsb first.
   a) During a read access, data is latched from the selected Nexus register when the JTAG state machine passes through the capture-DR state.
   b) During a write access, data is latched into the selected Nexus register when the JTAG state machine passes through the update-DR state.

24.12 Ownership Trace

This section details the ownership trace features of the NZ3C3 module.

Ownership trace provides a macroscopic view, such as task flow reconstruction, when debugging software written in a high level (or object-oriented) language. It offers the highest level of abstraction for tracking operating system software execution. This is especially useful when the developer is not interested in debugging at lower levels.

24.12.1 Ownership Trace Messaging (OTM)

Ownership trace information is messaged via the auxiliary port using an ownership trace message (OTM). The e200z3 processor contains a Power Architecture Book E defined process ID register within the CPU.

The process ID register is updated by the operating system software to provide task/process ID information. The contents of this register are replicated on the pins of the processor and connected to Nexus. The process ID register value can be accessed using the **mfspr**/**mtspr** instructions. Please see the *e200z3 PowerPC*^{TM} *Core Reference Manual*for more details on the process ID register.

The only condition that causes an ownership trace message occurs when the OTR register is updated or process ID register by the e200z3 processor, the data is latched within Nexus, and is messaged out via the auxiliary port, allowing development tools to trace ownership flow.

Ownership trace information is messaged out in the following format:

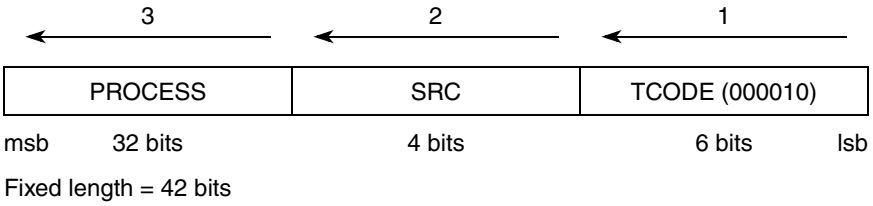


Figure 24-27. Ownership Trace Message Format

24.12.2 OTM Error Messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards incoming messages until the queue the queue is completely empty. After it empties, an error message is queued. The error encoding indicate the message types denied service to the queue while the FIFO was emptying.

If an OTM message only attempts to enter the queue while the queue is emptying, the error message incorporates the OTM error encoding (00000) only. If OTM and either BTM or DTM messages attempt to enter the queue, the error message incorporates the OTM and (program or data) trace error encoding (00111). If a watchpoint also attempts to enter the queue while the FIFO is emptying, then the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format (see Table 24-20):

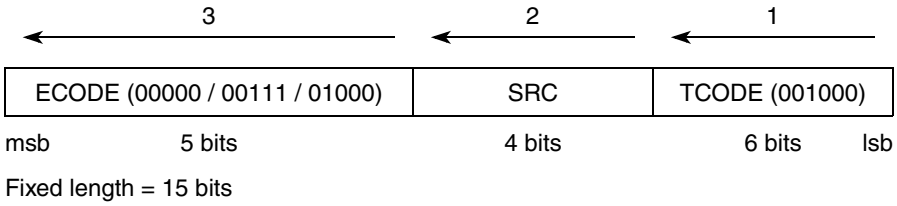


Figure 24-28. Error Message Format

24.12.3 OTM Flow

Ownership trace messages are generated when the operating system writes to the e200z3 process ID register or the memory mapped ownership trace register.

The following flow describes the OTM process:

1. The process ID register is a system control register. It is internal to the e200z3 processor and can be accessed by using PPC instructions **mtspr** and **mfspr**. The contents of this register are replicated on the pins of the processor and connected to Nexus.
2. OTR/process ID register reads do not cause ownership trace messages to be transmitted by the NZ3C3 module.
3. If the periodic OTM message counter expires (after 255 queued messages without an OTM), an OTM is sent using the latched data from the previous OTM or process ID register write.

24.13 Program Trace

This section details the program trace mechanism supported by NZ3C3 for the e200z6 processor. Program trace is implemented via branch trace messaging (BTM) as per the Class 3 IEEE®-ISTO 5001-2003 standard definition. Branch trace messaging for e200z3 processors is accomplished by snooping the e200z3 virtual address bus (between the CPU and MMU), attribute signals, and CPU status.

24.13.1 Branch Trace Messaging (BTM)

Traditional branch trace messaging facilitates program trace by providing the following types of information:

- Messages generated for direct branches that were taken indicate the number of sequential instructions executed since the last branch or exception. Branches not taken (direct or indirect) are not counted as sequential instructions.
- Messages generated for indirect branches and exceptions that were taken indicate:
  - Number of sequential instructions executed since the last branch that was taken
  - Exception with the unique portion of the branch target address or exception vector address
- History field in the branch and predicate instructions that can generate the following messages for program trace:

— Number of sequential instructions executed since the last indirect branch was taken, as well as the unique portion of the indirect branch address
— Number of sequential instructions executed since the last exception was processed, as well as the unique portion of the exception vector address
— Number of sequential instructions executed since the last predicate instruction was taken
— History field in the branch and predicate instruction unique to the branch target address or exception vector address. Each bit in the history field represents a direct branch or predicated instruction where a value of one (1) indicates taken, and a value of zero (0) indicates not taken. Certain instructions (**evsel**) generate a pair of predicate bits which are both reported as consecutive bits in the history field.

24.13.1.1 e200z3 Indirect Branch Message Instructions (Power Architecture Book E)

Table 24-34 shows the types of instructions and events which cause indirect branch messages or branch history messages to be encoded.

Table 24-34. Indirect Branch Message Sources

Source of Indirect Branch Message	Instructions
Taken branch relative to a register value	bcctr, bcctrl, bclr, bclrl
System call / trap exceptions taken	sc, tw, twi
Return from interrupts / exceptions	rfi, rfci, rfdi

24.13.1.2 e200z3 Direct Branch Message Instructions (Power Architecture Book E)

Table 24-35 shows the instruction types that cause direct branch messages, or toggle a bit in the instruction history buffer in a resource full message or branch history message before it is sent out.

Table 24-35. Direct Branch Message Sources

Source of Direct Branch Message	Instructions
Taken direct branch instructions	b, ba, bl, bla, bc, bca, bcl, bcla
Instruction synchronize	isync

24.13.1.3 BTM Using Branch History Messages

Traditional BTM messaging can accurately track the number of sequential instructions between branches, but cannot accurately indicate which instructions were conditionally executed, and which were not.

Branch history messaging solves this problem by providing a predicated instruction history field in each indirect branch message. Each bit in the history represents a predicated instruction or direct branch. A value of one (1) indicates the conditional instruction was executed or the direct branch was taken. A value of zero (0) indicates the conditional instruction was not executed or the direct branch was not taken.

Certain instructions (**evsel**) generate a pair of predicate bits which are both reported as consecutive bits in the history field.

Branch history messages solve predicated instruction tracking and save bandwidth since only indirect branches cause messages to be queued.

24.13.1.4 BTM Using Traditional Program Trace Messages

Based on the PTM bit in the DC register (DC[PTM]), program tracing can use:

- Branch history messages (DC[PTM] = 1); or
- Traditional direct and indirect branch messages (DC[PTM] = 0)

Branch history saves bandwidth and keeps consistency between methods of program trace, yet can lose temporal order between BTM messages and other types of messages. Since direct branches are not messaged, but are instead included in the history field of the indirect branch history message, other types of messages can enter the FIFO between branch history messages. The development tool cannot determine the order of “events” that occurred for direct branches by the order in which messages are sent out.

Traditional BTM messages maintain their temporal ordering because each event that queues a message to the FIFO is processed and sent in the order it was generated, and the message order is maintained when it is transmitted.

24.13.2 BTM Message Formats

The e200z3 Nexus3 module supports three types of traditional BTM messages—direct, indirect, and synchronization messages. It supports two types of branch history BTM messages—indirect branch history, and indirect branch history with synchronization messages. Debug status messages and error messages are also supported.

24.13.2.1 Indirect Branch Messages (History)

Indirect branches include all taken branches whose destination is determined at run time, interrupts and exceptions. If DC[PTM] is set, indirect branch information is messaged out in the following format:

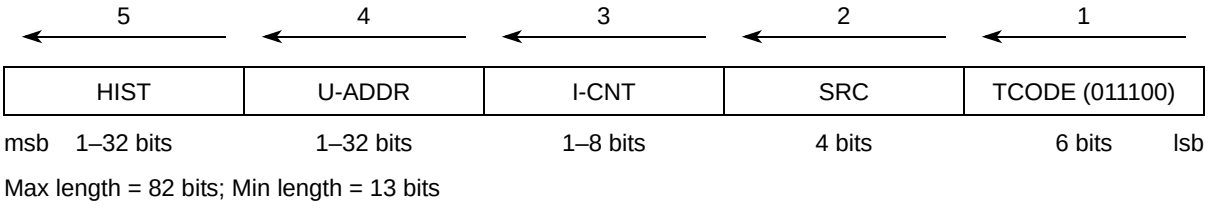


Figure 24-29. Indirect Branch Message (History) Format

24.13.2.2 Indirect Branch Messages (Traditional)

If DC[PTM] is cleared, indirect branch information is messaged out in the following format:

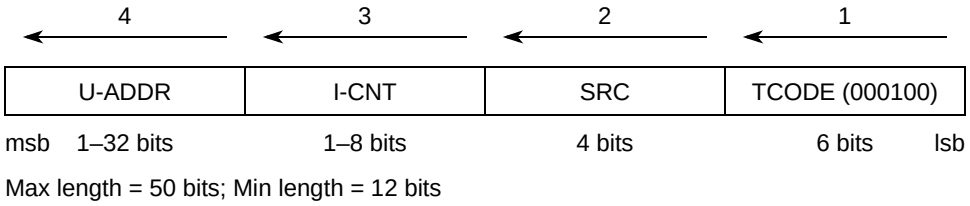


Figure 24-30. Indirect Branch Message Format

24.13.2.3 Direct Branch Messages (Traditional)

Direct branches (conditional or unconditional) are all taken branches whose destination is fixed in the instruction opcode. Direct branch information is messaged out in the following format:

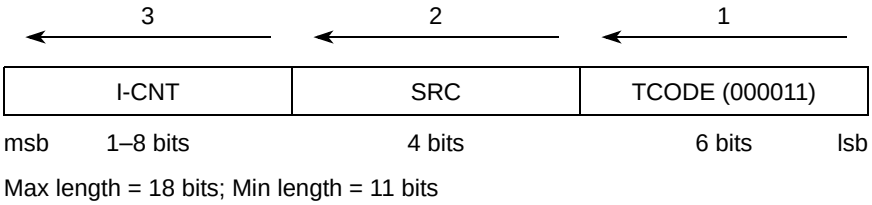


Figure 24-31. Direct Branch Message Format

NOTE

When DC[PTM] is set, direct branch messages are not transmitted. Instead, each direct branch or predicated instruction toggles a bit in the history buffer.

24.13.2.4 Resource Full Messages

The resource full message is used in conjunction with the branch history messages. The resource full message is generated when the internal branch/predicate history buffer is full, or if the BTM Instruction sequence counter (I-CNT) overflows. If synchronization is needed at the time this message is generated, the synchronization is delayed until the next branch trace message that is not a resource full message.

The current value of the history buffer is transmitted as part of the resource full message. This information can be concatenated by the tool with the branch/predicate history information from subsequent messages to obtain the complete branch history for a message. The internal history value is reset by this message, and the I-CNT value is reset as a result of a bit being added to the history buffer.

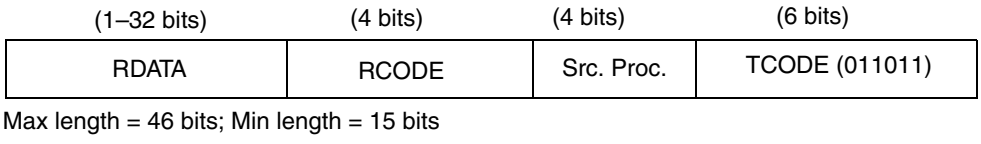


Figure 24-32. Resource Full Message Format

24.13.2.5 Debug Status Messages

NOTE

Debug Status Messages (DSMs) are enabled if the Nexus module is enabled.

Debug status messages report low power mode and debug status. Entering/exiting debug mode as well as entering a low power mode triggers a debug status message.

Debug status information is sent out in the following format:

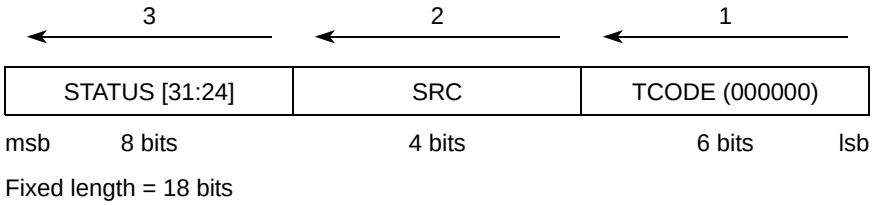


Figure 24-33. Debug Status Message Format

24.13.2.6 Program Correlation Messages

Program correlation messages are used to correlate events to the program flow that are not necessarily associated with the instruction stream. To maintain accurate instruction tracing information when entering debug mode or a CPU low power mode (where tracing can be disabled), this message is sent upon entry into one of these two modes and includes the instruction count and branch history. Program correlation is messaged out in the following format:

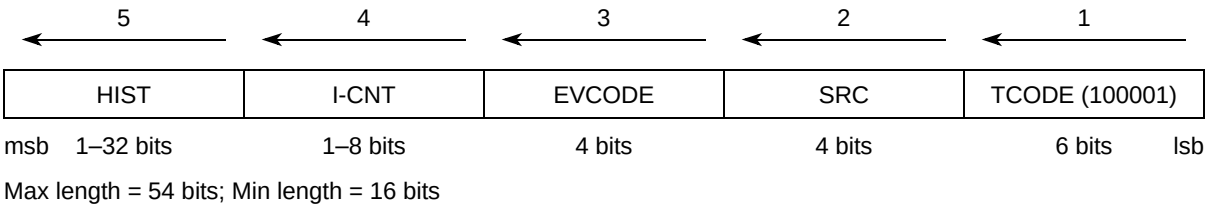


Figure 24-34. Program Correlation Message Format

24.13.2.7 BTM Overflow Error Messages

An error message occurs when a new message cannot be queued because the message queue is full. The FIFO discards incoming messages until the queue is completely empty. After it is empty, an error message is queued. The error encoding indicates which message types were denied queueing while the FIFO was emptying.

If only a program trace message attempts to enter the queue while it is being emptied, the error message incorporates the program trace only error encoding (00001). If both OTM and program trace messages attempt to enter the queue, the error message incorporates the OTM and program trace error encoding (00111). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format:

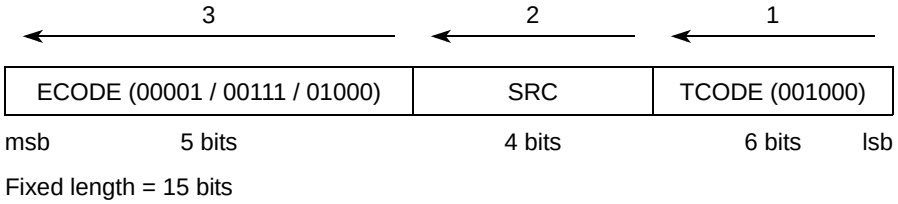


Figure 24-35. Error Message Format

24.13.3 Program Trace Synchronization Messages

A program trace direct/indirect branch with sync message is messaged via the auxiliary port (provided program trace is enabled) for the following conditions (see Table 24-36):

- Initial program trace message upon the first direct/indirect branch after exit from system reset or whenever program trace is enabled
- Upon direct/indirect branch after returning from a CPU low power state
- Upon direct/indirect branch after returning from debug mode
- Upon direct/indirect branch after occurrence of queue overrun (can be caused by any trace message), provided program trace is enabled
- Upon direct/indirect branch after the periodic program trace counter has expired indicating 255 *without-sync* program trace messages have occurred since the last *with-sync* message occurred
- Upon direct/indirect branch after assertion of the event in (EVTI) pin if the EIC bits within the DC1 register have enabled this feature
- Upon direct/indirect branch after the sequential instruction counter has expired indicating 255 instructions have occurred between branches
- Upon direct/indirect branch after a BTM message was lost due to an attempted access to a secure memory location.
- Upon direct/indirect branch after a BTM message was lost due to a collision entering the FIFO between the BTM message and either a watchpoint message or an ownership trace message

If the NZ3C3 module is enabled at reset, a EVTI assertion initiates a program trace direct/indirect branch with sync message (if program trace is enabled) upon the first direct/indirect branch. The format for program trace direct/indirect branch with sync messages is as follows:

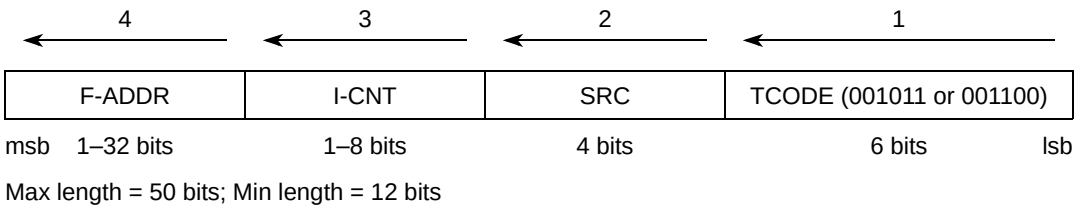


Figure 24-36. Direct/Indirect Branch with Sync Message Format

The formats for program trace direct/indirect branch with sync. messages and indirect branch history with sync. messages are as follows:

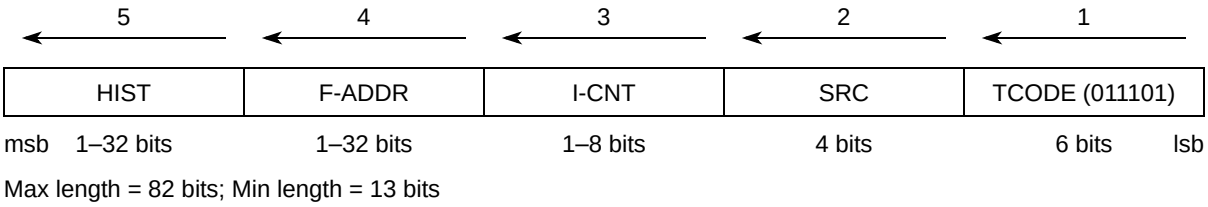


Figure 24-37. Indirect Branch History with Sync. Message Format

Exception conditions that result in program trace synchronization are summarized in Table 24-36.

Table 24-36. Program Trace Exception Summary

Exception Condition	Exception Handling
System Reset Negation	At the negation of JTAG reset (JCOMP), queue pointers, counters, state machines, and registers within the NZ3C3 module are reset. Upon the first branch out of system reset (if program trace is enabled), the first program trace message is a direct/indirect branch with sync. message.
Program Trace Enabled	The first program trace message (after program trace has been enabled) is a synchronization message.
Exit from Low Power/Debug	Upon exiting from the low power or debug modes, the next direct/indirect branch is converted to a direct/indirect branch with sync. message.
Queue Overrun	An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until the queue is completely empty. After it is empty, an error message is queued. The error encoding indicates the message types denied queueing while the FIFO was emptying. The next BTM message in the queue is a direct/indirect branch with sync. message.
Periodic Program Trace Sync.	A forced synchronization occurs periodically after 255 program trace messages have been queued. A direct/indirect branch with sync. message is queued. The periodic program trace message counter then resets.
Event In	If the Nexus module is enabled, an EVTI assertion initiates a direct/indirect branch with sync. message upon the next direct/indirect branch (if program trace is enabled and the EIC bits of the DC1 register have enabled this feature).
Sequential Instruction Count Overflow	When the sequential instruction counter reaches its maximum count (up to 255 sequential instructions can be executed), a forced synchronization occurs. The sequential counter then resets. A program trace direct/indirect branch with sync.message is queued upon execution of the next branch.
Attempted Access to Secure Memory	For devices which implement security, any attempt to branch to secure memory locations temporarily disables program trace and cause the corresponding BTM to be lost. The following direct/indirect branch queues a direct/indirect branch with sync. message. The count value within this message can be inaccurate since re-enabling program trace does not guarantee alignment on an instruction boundary.

Table 24-36. Program Trace Exception Summary (continued)

Exception Condition	Exception Handling
Collision Priority	All messages have the following priority: WPM -> OTM -> BTM -> DTM. A BTM message which attempts to enter the queue at the same time as a watchpoint message or ownership trace message is lost. An error message is sent indicating the BTM was lost. The following direct/indirect branch queues a direct/indirect branch with sync. message. The count value within this message reflects the number of sequential instructions executed after the last successful BTM message was generated. This count includes the branch which did not generate a message due to the collision.

24.14 BTM Operation

24.14.1 Enabling Program Trace

Both types of branch trace messaging can be enabled in one of two ways:

- Setting the TM field of the DC1 register to enable program trace (DC1[TM])
- Using the PTS field of the WT register to enable program trace on watchpoint hits (e200z3 watchpoints are configured within the CPU)

24.14.2 Relative Addressing

The relative address feature is compliant with the IEEE®-ISTO 5001-2003 standard recommendations, and is designed to reduce the number of bits transmitted for addresses of indirect branch messages.

The address transmitted is relative to the target address of the instruction which triggered the previous indirect branch (or sync) message. It is generated by XOR'ing the new address with the previous address, and then using only the results up to the most significant 1 in the result. To recreate this address, an XOR of the (most-significant 0-padded) message address with the previously decoded address gives the current address.

```
Previous address (A1) =0x0003FC01, New address (A2) = 0x0003F365
```

Message Generation:

A1 = 0000 0000 0000 0011 1111 1100 0000 0001
A2 = 0000 0000 0000 0011 1111 0011 0110 0101

A1 ⊕ A2 = 0000 0000 0000 0000 0000 1111 0110 0100

Address Message (M1) = 1111 0110 0100

Address Re-creation:

A1 ⊕ M1 = A2
A1 = 0000 0000 0000 0011 1111 1100 0000 0001
M1 = 0000 0000 0000 0000 0000 1111 0110 0100

A2 = 0000 0000 0000 0011 1111 0011 0110 0101

Figure 24-38. Relative Address Generation and Re-creation

24.14.3 Branch and Predicate Instruction History (HIST)

If DC[PTM] is set, BTM messaging uses the branch history format. The branch history (HIST) packet in these messages provides a history of direct branch execution used for reconstructing the program flow. This packet is implemented as a left-shifting shift register. The register is always pre-loaded with a value of one (1). This bit acts as a stop bit so that the development tools can determine which bit is the end of the history information. The pre-loaded bit itself is not part of the history, but is transmitted with the packet.

A value of one (1) is shifted into the history buffer on a taken branch (condition or unconditional) and on any instruction whose predicate condition executed as true. A value of zero (0) is shifted into the history buffer on any instruction whose predicate condition executed as false as well as on branches not taken. This includes indirect as well as direct branches were not taken. For the **evsel** instruction, two bits are shifted in, corresponding to the low element (shifted in first) and the high element (shifted in second) conditions.

24.14.4 Sequential Instruction Count (I-CNT)

The I-CNT packet, is present in all BTM messages. For traditional branch messages, I-CNT represents the number of sequential instructions, or non-taken branches in between direct/indirect branch messages.

For branch history messages, I-CNT represents the number of instructions executed since the last taken/non-taken direct branch, last taken indirect branch or exception. Not taken indirect branches are considered sequential instructions and cause the instruction count to increment. I-CNT also represents the number of instructions executed since the last predicate instruction.

The sequential instruction counter overflows when its value reaches 255. The next BTM message is converted to a synchronization type message.

24.14.5 Program Trace Queueing

NZ3C3 implements a message queue. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

NOTE

If multiple trace messages must be queued at the same time, Watchpoint Messages have the highest priority (WPM –> OTM –> BTM –> DTM).

24.14.5.1 Program Trace Timing Diagrams

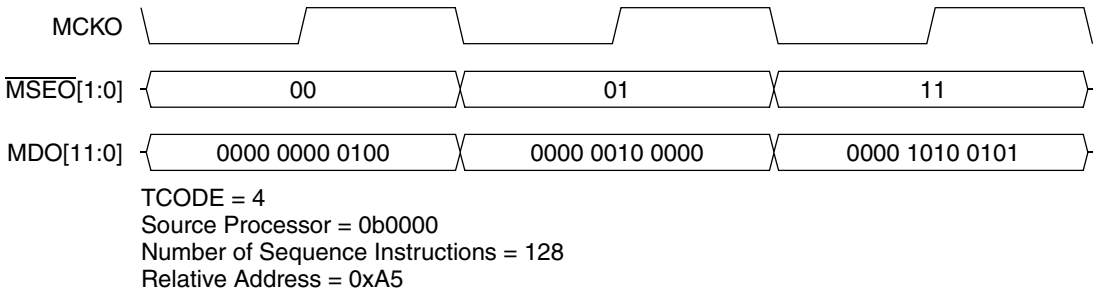


Figure 24-39. Program Trace (MDO = 12)—Indirect Branch Message (Traditional)

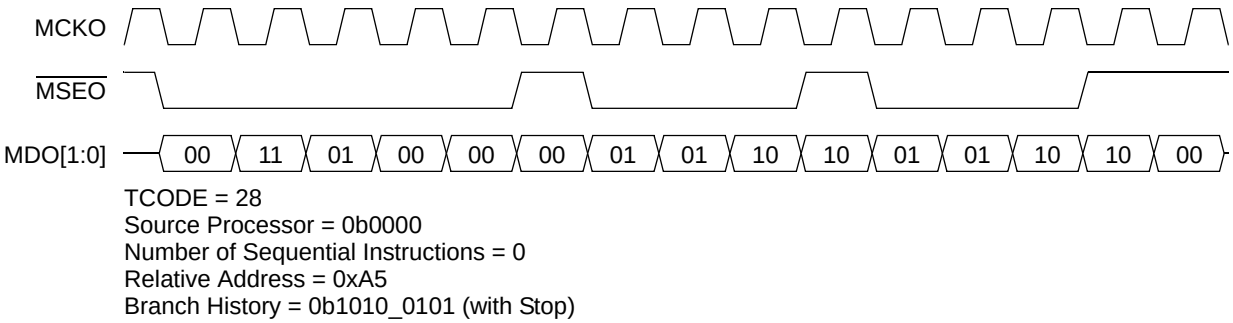


Figure 24-40. Program Trace (MDO = 2)—Indirect Branch Message (History)

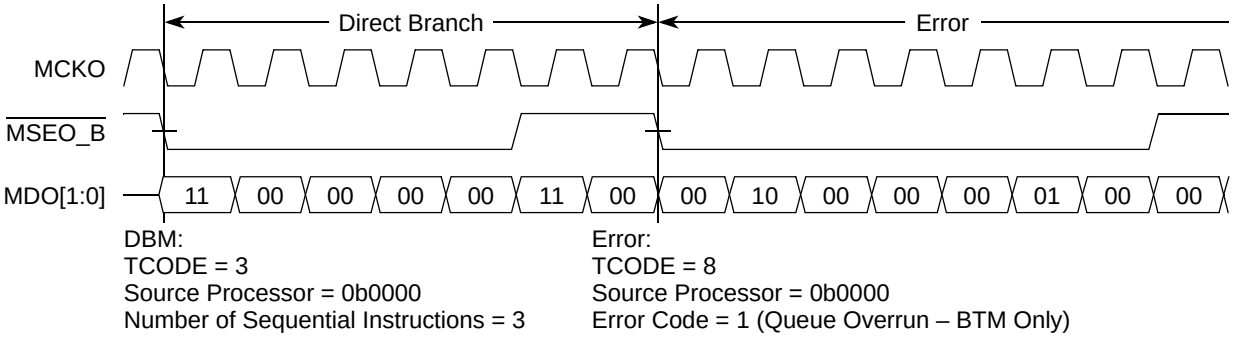


Figure 24-41. Program Trace—Direct Branch (Traditional) and Error Messages

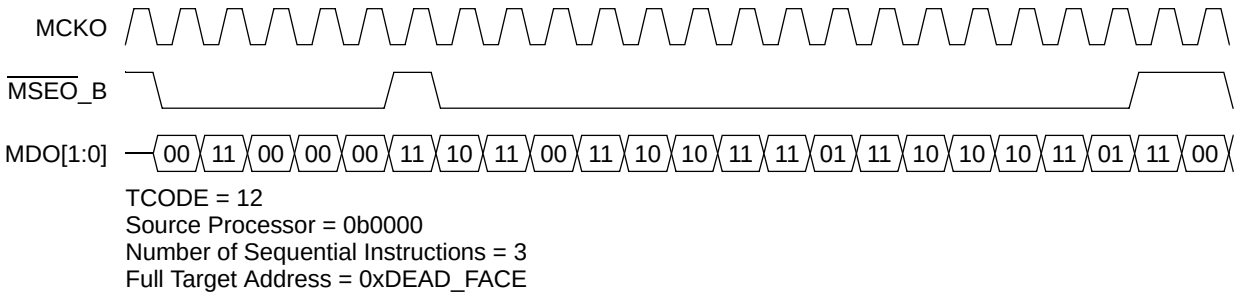


Figure 24-42. Program Trace—Indirect Branch with Sync. Message

24.14.6 Data Trace

This section deals with the data trace mechanism supported by the NZ3C3 module. Data trace is implemented via data write messaging (DWM) and data read messaging (DRM), as per the IEEE®-ISTO 5001-2003 standard.

24.14.6.1 Data Trace Messaging (DTM)

Data trace messaging for e200z3 is accomplished by snooping the e200z3 virtual data bus (between the CPU and MMU), and storing the information for qualifying accesses (based on enabled features and matching target addresses). The NZ3C3 module traces all data access that meet the selected range and attributes.

NOTE

Data trace is only performed on the e200z3 virtual data bus. This allows for data visibility for the incorporated data cache. Only e200z3 CPU initiated accesses are traced.

Data trace messaging can be enabled in one of two ways:

- Setting the TM field of the DC1 register to enable data trace (DC1[TM]).
- Using WT[DTS] to enable data trace on watchpoint hits (e200z3 watchpoints are configured within the Nexus1 module)

24.14.6.2 DTM Message Formats

The Nexus3 module supports five types of DTM messages: data write, data read, data write synchronization, data read synchronization and error messages.

24.14.6.2.1 Data Write Messages

The data write message contains the data write value and the address of the write access, relative to the previous data trace message. Data write message information is messaged out in the following format:

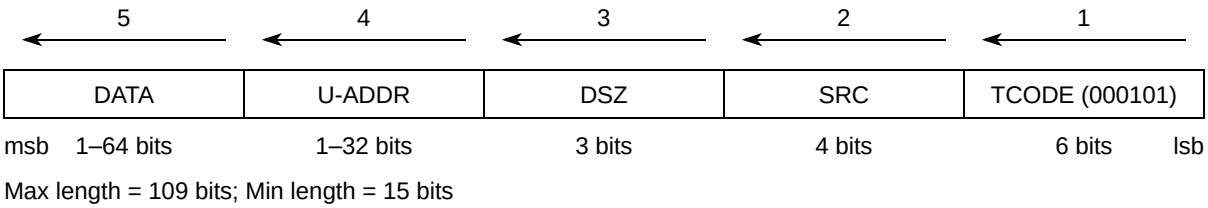


Figure 24-43. Data Write Message Format

24.14.6.2.2 Data Read Messages

The data read message contains the data read value and the address of the read access, relative to the previous data trace message. Data read message information is messaged out in the following format:

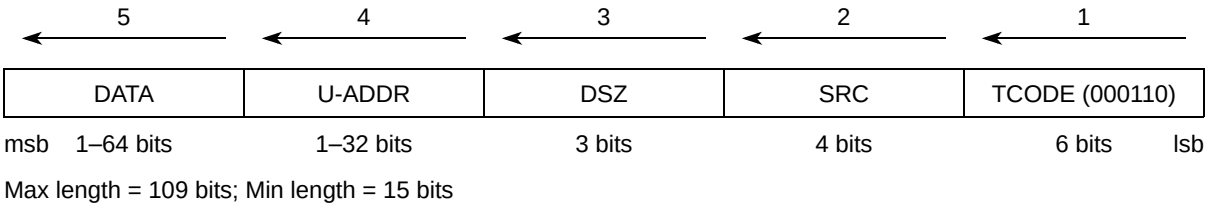


Figure 24-44. Data Read Message Format

NOTE

For the e200z3 based CPU, the doubleword encoding (data size = 0b000) indicates a doubleword access and sends out as a single data trace message with a single 64-bit data value.

24.14.6.2.3 DTM Overflow Error Messages

An error message occurs when the next message is denied service because the message queue is full. The FIFO discards all incoming messages until the queue is completely empty. After it is empty, an error message is queued that indicates the message types denied into the queue while the FIFO is emptying.

If a data trace message only attempts to enter the queue while it is emptying, the error message incorporates the data trace only error encoding (00010). If both OTM and data trace messages attempt to enter the queue, the error message incorporates the OTM and data trace error encoding (00111). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format:

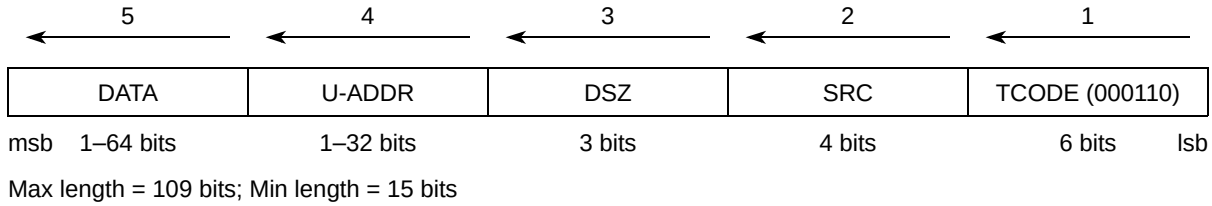


Figure 24-45. Error Message Format

24.14.6.2.4 Data Trace Synchronization Messages

A data trace write/read with sync. message is messaged via the auxiliary port (provided data trace is enabled) for the following conditions (see Table 24-37):

- Initial data trace message after exit from system reset or whenever data trace is enabled
- Upon exiting debug mode
- After occurrence of queue overrun (can be caused by any trace message), provided data trace is enabled
- After the periodic data trace counter has expired indicating 255 *without-sync* data trace messages have occurred since the last *with-sync* message occurred
- Upon assertion of the event in (EVTI) pin, the first data trace message is a synchronization message if the EIC bits of the DC1 register have enabled this feature
- Upon data trace write/read after the previous DTM message was lost due to an attempted access to a secure memory location
- Upon data trace write/read after the previous DTM message was lost due to a collision entering the FIFO between the DTM message and any of the following: watchpoint message, ownership trace message, or branch trace message

Data trace synchronization messages provide the full address (without leading zeros) and insure that development tools fully synchronize with data trace regularly. Synchronization messages provide a reference address for subsequent data messages, in which only the unique portion of the data trace address is transmitted. The format for data trace write/read with sync. messages is as follows:

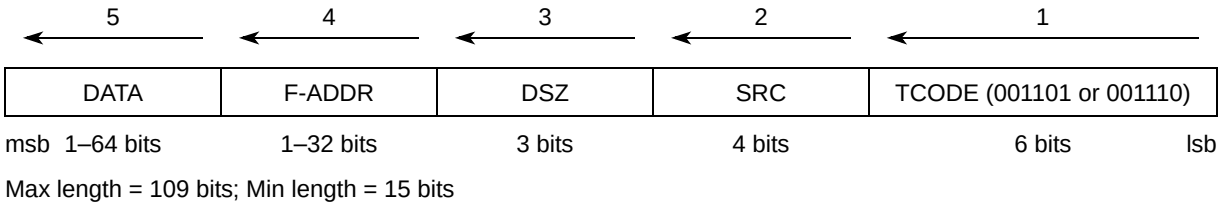


Figure 24-46. Data Write/Read with Sync. Message Format

Exception conditions that result in data trace synchronization are summarized in Table 24-37.

Table 24-37. Data Trace Exception Summary

Exception Condition	Exception Handling
System Reset Negation	At the negation of JTAG reset (JCOMP), queue pointers, counters, state machines, and registers within the NZ3C3 module are reset. If data trace is enabled, the first data trace message is a data write/read with sync. message.
Data Trace Enabled	The first data trace message (after data trace has been enabled) is a synchronization message.
Exit from Low Power/Debug	Upon exiting from low power or debug modes, the next data trace message is converted to a data write/read with sync. message.
Queue Overrun	An error message occurs when a new message cannot be queued due to a full message queue. The FIFO discards messages until it has completely emptied the queue. After the queue is empty, an error message is queued that indicates the message types denied queuing while the FIFO was emptying. The next DTM message in the queue is a data write/read with sync. message.
Periodic Data Trace Sync.	A forced synchronization occurs periodically after 255 data trace messages have been queued. A data write/read with sync. message is queued. The periodic data trace message counter then resets.
Event In	If the Nexus module is enabled, a EVTI assertion initiates a data trace write/read with sync. message upon the next data write/read (if data trace is enabled and the EIC bits of the DC1 register have enabled this feature).
Attempted Access to Secure Memory	For devices which implement security, any attempted read or write to secure memory locations temporarily disables data trace and loses the DTM. A subsequent read/write queues a data trace read/write with sync. message.
Collision Priority	All messages have the following priority: WPM –> OTM –> BTM –> DTM. A DTM message which attempts to enter the queue at the same time as a watchpoint message or ownership trace message or branch trace message can be lost. A subsequent read/write queues a data trace read/write with sync. message.

24.14.6.3 DTM Operation

24.14.6.3.1 DTM Queueing

NZ3C3 implements a message queue for DTM messages. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

NOTE

If multiple trace messages must be queued at the same time, watchpoint messages have the highest priority (WPM –> OTM –> BTM –> DTM).

24.14.6.3.2 Relative Addressing

The relative address feature is compliant with the IEEE®-ISTO 5001-2003 standard recommendations, and is designed to reduce the number of bits transmitted for addresses of data trace messages. See Section 24.14.2, “ Relative Addressing for details.

24.14.6.3.3 Data Trace Windowing

Data write/read messages are enabled via the RWT1(2) field in the data trace control register (DTC) for each DTM channel. Data trace windowing is achieved via the address range defined by the DTEA and DTSA registers and by the RC1(2) field in the DTC. All e200z3 initiated read/write accesses which fall inside or outside these address ranges, as programmed, are candidates to be traced.

24.14.6.3.4 Data Access/Instruction Access Data Tracing

The Nexus3 module is capable of tracing both instruction access data or data access data. Each trace window can be configured for either type of data trace by setting the DI1(2) field within the data trace control register for each DTM channel.

24.14.6.3.5 e200z3 Bus Cycle Special Cases

Table 24-38. e200z3 Bus Cycle Cases

Special Case	Action
e200z3 bus cycle aborted	Cycle ignored
e200z3 bus cycle with data error ($\overline{\text{TEA}}$)	Data Trace Message discarded
e200z3 bus cycle completed without error	Cycle captured & transmitted
e200z3 bus cycle initiated by NZ3C3	Cycle ignored
e200z3 bus cycle is an instruction fetch	Cycle ignored
e200z3 bus cycle accesses misaligned data (across 64-bit boundary)—both 1st and 2nd transactions within data trace range	1st and 2nd cycle captured, and 2 DTM's transmitted (see Note)
e200z3 bus cycle accesses misaligned data (across 64-bit boundary)—1st transaction within data trace range; 2nd transaction out of data trace range	1st cycle captured and transmitted; 2nd cycle ignored
e200z3 bus cycle accesses misaligned data (across 64-bit boundary)—1st transaction out of data trace range; 2nd transaction within data trace range	1st cycle ignored; 2nd cycle capture and transmitted

NOTE

For misaligned accesses (crossing 64-bit boundary), the access is broken into two accesses. If both accesses are within the data trace range, two DTMs are sent: one with a size encoding indicating the size of the original access (a word), and one with a size encoding for the portion which crossed the boundary (3 bytes).

NOTE

An STM to the cache's store buffer within the data trace range initiates a DTM message. If the corresponding memory access causes an error, a checkstop condition occurs. The debug/development tool must use this indication to invalidate the previous DTM.

24.14.6.4 Data Trace Timing Diagrams (Eight MDO Configuration)

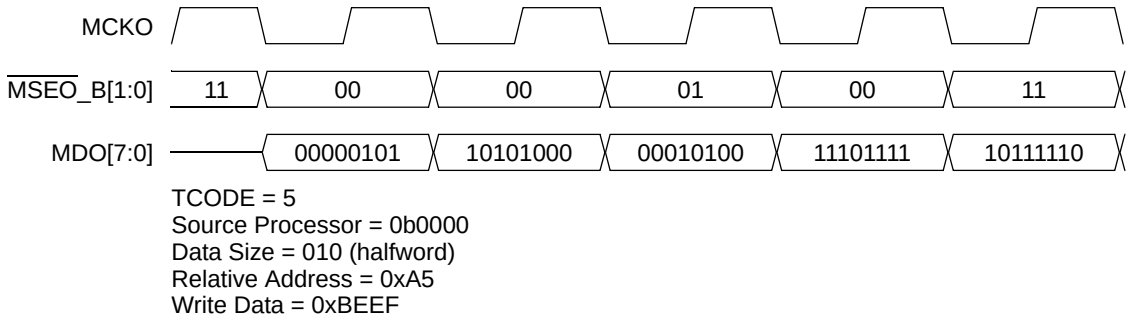


Figure 24-47. Data Trace—Data Write Message

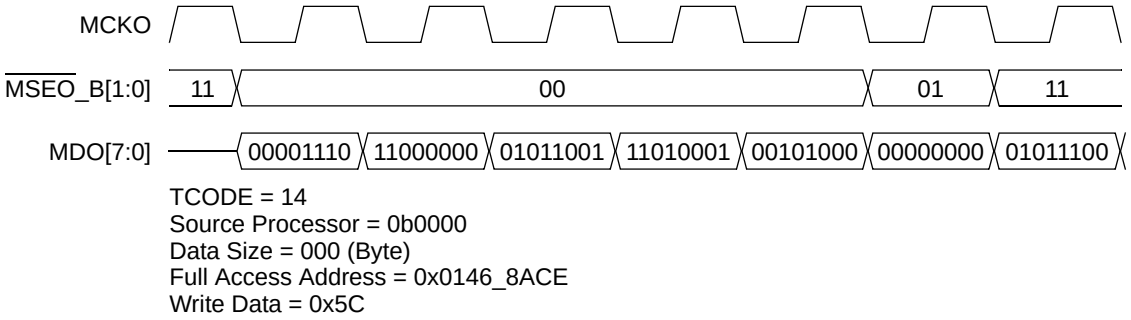


Figure 24-48. Data Trace—Data Read with Sync Message

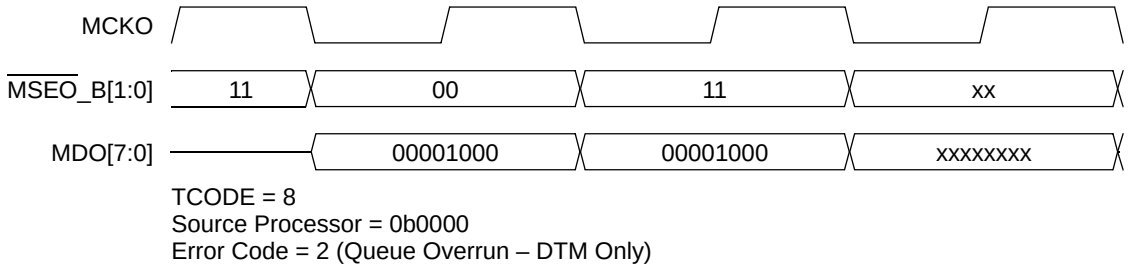


Figure 24-49. Error Message (Data Trace only encoded)

24.14.7 Watchpoint Support

This section details the watchpoint features of the NZ3C3 module.

24.14.7.1 Overview

The NZ3C3 module provides watchpoint messaging via the auxiliary pins, as defined by the IEEE®-ISTO 5001-2003 standard.

NZ3C3 is not compliant with Class4 breakpoint/watchpoint requirements defined in the standard. The breakpoint/watchpoint control register is not implemented.

24.14.7.2 Watchpoint Messaging

Enabling watchpoint messaging is done by setting the watchpoint enable bit in the DC1 register. Setting the individual watchpoint sources is supported through the e200z3 Nexus1 module. The e200z3 Nexus1 module is capable of setting multiple address and/or data watchpoints. Please see the e200z3 Core Reference Manual for more information on watchpoint initialization.

When these watchpoints occur, a watchpoint event signal from the Nexus1 module causes a message to be sent to the queue to be messaged out. This message includes the watchpoint number indicating which watchpoint caused the message.

The occurrence of any of the e200z3 defined watchpoints can be programmed to assert the event out $\overline{\text{EVTO}}$ pin for one period of the output clock (MCKO).

Watchpoint information is messaged out in the following format:

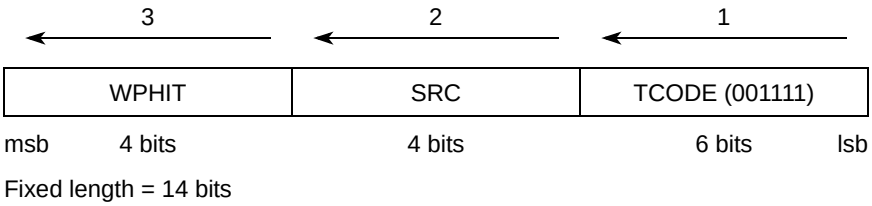


Figure 24-50. Watchpoint Message Format.

Table 24-39. Watchpoint Source Encoding

Watchpoint Source (8 bits)	Watchpoint Description
00000001	e200z3 Watchpoint #0 (IAC1 from Nexus1)
00000010	e200z3 Watchpoint #1 (IAC2 from Nexus1)
00000100	e200z3 Watchpoint #2 (IAC3 from Nexus1)
00001000	e200z3 Watchpoint #3 (IAC4 from Nexus1)
00010000	e200z3 Watchpoint #4 (DAC1 from Nexus1)
00100000	e200z3 Watchpoint #5 (DAC2 from Nexus1)
01000000	e200z3 Watchpoint #6 (DCNT1 from Nexus1)
10000000	e200z3 Watchpoint #7 (DCNT2 from Nexus1)

24.14.7.3 Watchpoint Error Message

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. After it is emptied, an error message is queued. The error encoding indicates the types of messages that attempted to be queued while the FIFO was being emptied.

If only a watchpoint message attempts to enter the queue while it is being emptied, the error message incorporates the watchpoint only error encoding (00110). If an OTM and/or program trace and/or data trace message also attempts to enter the queue while it is being emptied, the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format (see Table 24-20):

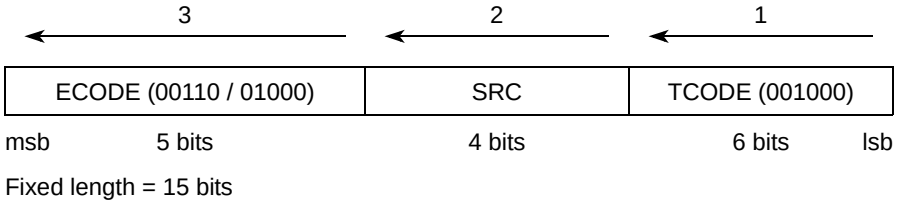


Figure 24-51. Error Message Format

24.14.7.4 Watchpoint Timing Diagram (Two MDO and One MSEO Configuration)

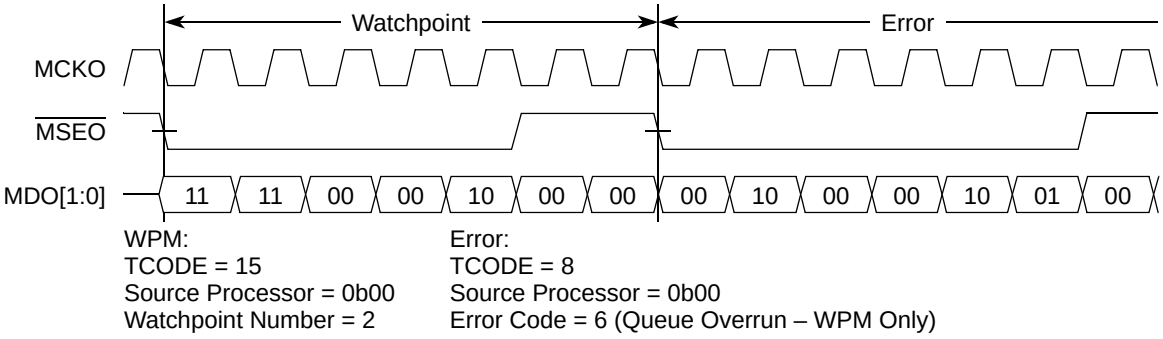


Figure 24-52. Watchpoint Message and Watchpoint Error Message

24.14.8 NZ3C3 Read/Write Access to Memory-Mapped Resources

The read/write access feature allows access to memory-mapped resources via the JTAG/OnCE port. The read/write mechanism supports single as well as block reads and writes to e200z3 system bus resources.

The NZ3C3 module is capable of accessing resources on the e200z3 system bus, with multiple configurable priority levels. Memory-mapped registers and other non-cached memory can be accessed via the standard memory map settings.

All accesses are setup and initiated by the read/write access control/status register (RWCS), as well as the read/write access address (RWA) and read/write access data registers (RWD).

Using the read/write access registers (RWCS/RWA/RWD), memory-mapped e200z3 system bus resources can be accessed through NZ3C3. The following subsections describe the steps which are required to access memory-mapped resources.

NOTE

Read/write access can only access memory mapped resources when system reset is de-asserted.

Misaligned accesses are NOT supported in the e200z3 Nexus3 module.

24.14.8.1 Single Write Access

1. Initialize the read/write access address register (RWA) through the access method outlined in Section 24.11.11, “ NZ3C3 Register Access via JTAG / OnCE” using the Nexus register index of 0x9 (see Table 24-24). Configure the write address to 0xnnnnnnnn (write address).
2. Initialize the read/write access control/status register (RWCS) through the access method outlined in Section 24.11.11, “ NZ3C3 Register Access via JTAG / OnCE,” using the Nexus Register Index of 0x7 (see Table 24-24). Configure the bits as follows:
   - Access Control RWCS[AC] –> 0b1 (to indicate start access)
   - Map Select RWCS[MAP] –> 0b000 (primary memory map)
   - Access Priority RWCS[PR] –> 0b00 (lowest priority)
   - Read/Write RWCS[RW] –> 0b1 (write access)
   - Word Size RWCS[SZ] –> 0b0xx (32-bit, 16-bit, 8-bit)
   - Access Count RWCS[CNT] –> 0x0000 or 0x0001 (single access)

NOTE

Access count RWCS[CNT] of 0x0000 or 0x0001 performs a single access.

3. Initialize the read/write access data register (RWD) through the access method outlined in Section 24.11.11, “ NZ3C3 Register Access via JTAG / OnCE,” using the Nexus register index of 0xA (see Table 24-24). Configure the write data to 0xnnnnnnnn (write data).
4. The NZ3C3 module then arbitrates for the system bus and transfer the data value from the data buffer RWD register to the memory mapped address in the read/write access address register (RWA). When the access has completed without error (ERR=1’b0), NZ3C3 asserts the RDY pin and clears the DV bit in the RWCS register. This indicates that the device is ready for the next access.

NOTE

Only the RDY pin as well as the DV and ERR bits within the RWCS provide read/write access status to the external development tool.

24.14.8.2 Block Write Access (Non-Burst Mode)

1. For a non-burst block write access, follow Steps 1, 2, and 3 outlined in Section 24.14.8.1, “Single Write Access to initialize the registers,” but using a value greater than one (0x1) for the RWCS[CNT] field.
2. The NZ3C3 module then arbitrates for the system bus and transfer the first data value from the RWD register to the memory mapped address in the read/write access address register (RWA). When the transfer has completed without error (ERR = 0), the address from the RWA register is incremented to the next word size (specified in the SZ field) and the number from the CNT field is decremented. Nexus then asserts the RDY pin. This indicates it is ready for the next access.
3. Repeat step 3 in Section 24.14.8.1, “Single Write Access” until the internal CNT value is zero (0). When this occurs, the DV bit within the RWCS is cleared to indicate the end of the block write access.

24.14.8.3 Block Write Access (Burst Mode)

1. For a burst block write access, follow Steps 1 and 2 outlined in Section 24.14.8.1, “Single Write Access” to initialize the registers, using a value of four (doublewords) for the CNT field and a RWCS[SZ] field indicating 64-bit access.
2. Initialize the burst data buffer (read/write access data register) through the access method outlined in Section 24.11.11, “ NZ3C3 Register Access via JTAG / OnCE,” using the Nexus register Index of 0xA (see Table 24-24).
3. Repeat step 2 until all doubleword values are written to the buffer.

NOTE

The data values must be shifted in 32-bits at a time lsb first (that is, doubleword write = two word writes to the RWD).

4. The Nexus module then arbitrates for the system bus and transfer the burst data values from the data buffer to the system bus beginning from the memory mapped address in the read/write access address register (RWA). For each access within the burst, the address from the RWA register is incremented to the next doubleword size (specified in the SZ field) modulo the length of the burst, and the number from the CNT field is decremented.
5. When the entire burst transfer has completed without error (ERR = 0), NZ3C3 then asserts the RDY pin, and the DV bit within the RWCS is cleared to indicate the end of the block write access.

NOTE

The actual RWA value as well as the CNT field within the RWCS are not changed when executing a block write access (burst or non-burst). The original values can be read by the external development tool at any time.

24.14.8.4 Single Read Access

1. Initialize the read/write access address register (RWA) through the access method outlined in Section 24.11.11, “ NZ3C3 Register Access via JTAG / OnCE,” using the Nexus register index of 0x9 (see Table 24-24). Configure as follows:
   - Read Address –> 0xnnnnnnnn (read address)
2. Initialize the read/write access control/status register (RWCS) through the access method outlined in Section 24.11.11, “ NZ3C3 Register Access via JTAG / OnCE,” using the Nexus register index of 0x7 (see Table 24-24). Configure the bits as follows:
   - Access Control RWCS[AC]–> 0b1 (to indicate start access)
   - Map Select RWCS[MAP] –> 0b000 (primary memory map)
   - Access Priority RWCS[PR] –> 0b00 (lowest priority)
   - Read/Write RWCS[RW] –> 0b0 (read access)
   - Word Size RWCS[SZ] –> 0b0xx (32-bit, 16-bit, 8-bit)
   - Access Count RWCS[CNT]–> 0x0000 or 0x0001 (single access)

NOTE

Access Count (CNT) of 0x0000 or 0x0001 performs a single access.

3. The NZ3C3 module then arbitrates for the system bus and the read data is transferred from the system bus to the RWD register. When the transfer is completed without error (ERR = 0), Nexus asserts the RDY pin and sets the DV bit in the RWCS register. This indicates that the device is ready for the next access.
4. The data can then be read from the read/write access data register (RWD) through the access method outlined in Section 24.11.11, " NZ3C3 Register Access via JTAG / OnCE," using the Nexus register index of 0xA (see Table 24-24).

NOTE

Only the RDY pin as well as the DV and ERR bits within the RWCS provide Read/Write Access status to the external development tool.

24.14.8.5 Block Read Access (Non-Burst Mode)

1. For a non-burst block read access, follow Steps 1 and 2 outlined in Section 24.14.8.4, "Single Read Access" to initialize the registers, but using a value greater than one (0x1) for the CNT field in the RWCS register.
2. The NZ3C3 module then arbitrates for the system bus and the read data is transferred from the system bus to the RWD register. When the transfer has completed without error (ERR=0b0), the address from the RWA register is incremented to the next word size (specified in the SZ field) and the number from the CNT field is decremented. Nexus then asserts the RDY pin. This indicates that the device is ready for the next access.
3. The data can then be read from the read/write access data register (RWD) through the access method outlined in Section 24.11.11, " NZ3C3 Register Access via JTAG / OnCE," using the Nexus register index of 0xA (see Table 24-24).
4. Repeat steps 3 and 4 in Section 24.14.8.4, "Single Read Access" until the CNT value is zero (0). When this occurs, the DV bit within the RWCS is set to indicate the end of the block read access.

24.14.8.6 Block Read Access (Burst Mode)

1. For a burst block read access, follow Steps 1 and 2 outlined in Section 24.14.8.4, "Single Read Access" to initialize the registers, using a value of four (doublewords) for the CNT field and an RWCS[SZ] field indicating 64-bit access.
2. The NZ3C3 module then arbitrates for the system bus and the burst read data is transferred from the system bus to the data buffer (RWD register). For each access within the burst, the address from the RWA register is incremented to the next doubleword (specified in the SZ field) and the number from the CNT field is decremented.
3. When the entire burst transfer has completed without error (ERR = 0), Nexus then asserts the RDY pin and the DV bit within the RWCS is set to indicate the end of the block read access.
4. The data can then be read from the burst data buffer (read/write access data register) through the access method outlined in Section 24.11.11, " NZ3C3 Register Access via JTAG / OnCE," using the Nexus register index of 0xA (see Table 24-24).
5. Repeat step 3 until all doubleword values are read from the buffer.

NOTE

The data values must be shifted out 32-bits at a time lsb first (that is, doubleword read = two word reads from the RWD).

NOTE

The actual RWA value as well as the CNT field within the RWCS are not changed when executing a block read access (burst or non-burst). The original values can be read by the external development tool at any time.

24.14.8.7 Error Handling

The NZ3C3 module handles various error conditions as follows:

24.14.8.7.1 System Bus Read/Write Error

All address and data errors that occur on read/write accesses to the e200z3 system bus returns a transfer error. If this occurs:

1. The access is terminated without re-trying (AC bit is cleared).
2. The ERR bit in the RWCS register is set.
3. The error message is sent (TCODE = 8) indicating read/write error.

24.14.8.7.2 Access Termination

The following cases are defined for sequences of the read/write protocol that differ from those described in the above sections:

1. If the AC bit in the RWCS register is set to start read/write accesses and invalid values are loaded into the RWD and/or RWA, then a system bus access error can occur. This is handled as described above.
2. If a block access is in progress (all cycles not completed), and the RWCS register is written, then the original block access is terminated at the boundary of the nearest completed access.
   a) If the RWCS is written with the AC bit set, the next read/write access begins and the RWD can be written to/ read from.
   b) If the RWCS is written with the AC bit cleared, the read/write access is terminated at the nearest completed access. This method can be used to break (early terminate) block accesses.

24.14.8.8 Read/Write Access Error Message

The read/write access error message is sent out when an system bus access error (read or write) occurs.

Error information is messaged out in the following format:

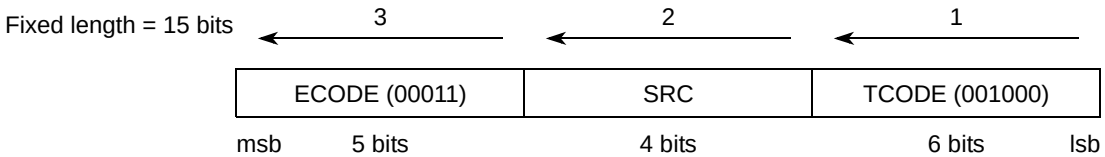


Figure 24-53. Error Message Format

24.14.9 Examples

The following are examples of program trace and data trace messages.

Table 24-40 illustrates an example indirect branch message with an eight MDO and two MSEO configuration.

T0 and S0 are the least significant bits where:

- Tx = TCODE number (fixed)
- Sx = Source processor (fixed)
- Ix = Number of instructions (variable)
- Ax = Unique portion of the address (variable)

Table 24-40. Indirect Branch Message Example (12 MDO and Two MSEO)

Clock	MDO[11:0]												$\overline{\text{MSEO}}$[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	I1	I0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	0	I5	I4	I3	I2	0	1	End Packet
3	0	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	1	1	End Packet/End Message
4	X	X	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start of Next Message

Table 24-41 illustrates an example of direct branch message with 12 MDO and two MSEO.

T0 and I0 are the least significant bits where:

- Tx = TCODE number (fixed)
- Sx = Source processor (fixed)
- Ix = Number of instructions (variable)

Table 24-41. Direct Branch Message Example (12 MDO and Two $\overline{\text{MSEO}}$)

Clock	MDO[11:0]												$\overline{\text{MSEO}}$[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	I1	I0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	0	0	0	I3	I2	1	1	End Packet and End Message
3	X	X	X	X	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start of Next Message

Table 24-42 an example data write message with 12 MDO and two MSEO configuration.

T0, A0, D0 are the least significant bits (LSB) where:

- Tx = TCODE number (fixed)
- Sx = Source processor (fixed)

- Zx = Data size (fixed)
- Ax = Unique portion of the address (variable)
- Dx = Write data (variable: 8-, 16- or 32-bit)

Table 24-42. Direct Write Message Example (12 MDO and Two $\overline{\text{MSEO}}$)

Clock	MDO[11:0]												MSEO[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	Z1	Z0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	A3	A2	A1	A0	Z2	0	1	End Packet
3	X	X	X	X	D7	D6	D5	D4	D3	D2	D1	D0	1	1	End Packet/End Message

24.14.10 IEEE® 1149.1 (JTAG) RD/WR Sequences

This section contains example JTAG/OnCE sequences used to access resources.

24.14.10.1 JTAG Sequence for Accessing Internal Nexus Registers

Table 24-43. Accessing Internal Nexus3 Registers via JTAG/OnCE

Step #	TMS Pin	Description
1	1	IDLE –> SELECT-DR_SCAN
2	0	SELECT-DR_SCAN –> CAPTURE-DR (Nexus command register value loaded in shifter)
3	0	CAPTURE-DR –> SHIFT-DR
4	0	(7) TCK clocks issued to shift in direction (read/write) bit and first 6 bits of Nexus reg. addr.
5	1	SHIFT-DR –> EXIT1-DR (7th bit of Nexus reg. shifted in)
6	1	EXIT1-DR –> UPDATE-DR (Nexus shifter is transferred to Nexus command register)
7	1	UPDATE-DR –> SELECT-DR_SCAN
8	0	SELECT-DR_SCAN –> CAPTURE-DR (Register value is transferred to Nexus shifter)
9	0	CAPTURE-DR –> SHIFT-DR
10	0	(31) TCK clocks issued to transfer register value to TDO pin while shifting in TDI value
11	1	SHIFT-DR –> EXIT1-DR (msb of value is shifted in/out of shifter)
12	1	EXIT1-DR –> UPDATE -DR (if access is write, shifter is transferred to register)
13	0	UPDATE-DR–> RUN-TEST/IDLE (transfer complete - Nexus controller to reg. select state)

24.14.10.2 JTAG Sequence for Read Access of Memory-Mapped Resources

Table 24-44. Accessing Memory-Mapped Resources (Reads)

Step #	TCLK clocks	Description
1	13	Nexus Command = write to read/write access address register (RWA)
2	37	Write RWA (initialize starting read address—data input on TDI)
3	13	Nexus Command = write to read/write control/status register (RWCS)
4	37	Write RWCS (initialize read access mode and CNT value—data input on TDI)
5	—	Wait for falling edge of RDY pin
6	13	Nexus Command = read the read/write access data register (RWD)
7	37	Read RWD (data output on TDO)
8	—	If CNT > 0, go back to Step #5

24.14.10.3 JTAG Sequence for Write Access of Memory-Mapped Resources

Table 24-45. Accessing Memory-Mapped Resources (Writes)

Step #	TCLK clocks	Description
1	13	Nexus Command = write to read/write access control/status register (RWCS)
2	37	Write RWCS (initialize write access mode and CNT value—data input on TDI)
3	13	Nexus Command = write to read/write address register (RWA)
4	37	Write RWA (initialize starting write address—data input on TDI)
5	13	Nexus Command = read the read/write access data register (RWD)
6	37	Write RWD (data output on TDO)
7	—	Wait for falling edge of RDY pin
8	—	If CNT > 0, go back to Step #5

Appendix A
MPC5534 Register Map

A.1 MPC5534 Register Map

The following table lists the base address for each module's register addresses.

Table A-1. Module Base Addresses

Module	Base Address	Page
Chapter 5, "Peripheral Bridge"	0xC3F0_0000	Page A-2
Chapter 11, "Frequency Modulated Phase Locked Loop and System Clocks (FMPLL)"	0xC3F8_0000	Page A-2
Chapter 12, "External Bus Interface (EBI)"[1]	0xC3F8_4000	Page A-2
Chapter 13, "Flash Memory"	0xC3F8_8000	Page A-3
Chapter 6, "System Integration Unit (SIU)"	0xC3F9_0000	Page A-3
Chapter 16, "Enhanced Modular Input/Output Subsystem (eMIOS)"	0xC3FA_0000	Page A-21
Chapter 17, "Enhanced Time Processing Unit (eTPU)"	0xC3FC_0000	Page A-22
Chapter 5, "Peripheral Bridge"	0xFFF0_0000	Page A-27
Chapter 7, "Crossbar Switch (XBAR)"	0xFFF0_4000	Page A-27
Chapter 8, "Error Correction Status Module (ECSM)"	0xFFF4_0000	Page A-28
Chapter 9, "Enhanced Direct Memory Access (eDMA)"	0xFFF4_4000	Page A-28
Chapter 10, "Interrupt Controller (INTC)"	0xFFF4_8000	Page A-31
Chapter 18, "Enhanced Queued Analog-to-Digital Converter (eQADC)"	0xFFF8_0000	Page A-38
Chapter 19, "Deserial Serial Peripheral Interface (DSPI)"	0xFFF9_4000 (DSPI B) 0xFFF9_8000 (DSPI C) 0xFFF9_C000 (DSPI D)	Page A-42
Chapter 20, "Enhanced Serial Communication Interface (eSCI)"	0xFFFB_0000 (A) 0xFFFB_4000 (B)	Page A-43
Chapter 21, "FlexCAN2 Controller Area Network"	0xFFFC_0000 (FlexCAN A) 0xFFFC_8000 (FlexCAN C)	Page A-44
Boot Assist Module (BAM)	0xFFFF_C000	Page A-46

[1] The External Bus Interface is not available on the 208 package.

This table lists the registers in each module.

- Signal names shown in **red** are not available on the 208 package.
- Signal names shown in **blue** are primary functions that are not designed into this device.

Table A-2. MPC5534 Detailed Register Map

Register Description	Register Name	Used Size	Address
Chapter 5, "Peripheral Bridge"			0xC3F0_0000
Peripheral bridge A master privilege control register	PBRIDGEA_MPCR	32-bit	Base + 0x0000
Reserved	—	—	(Base + 0x0004)–0xC3F7_FFFF
Chapter 11, "Frequency Modulated Phase Locked Loop and System Clocks (FMPLL)"			0xC3F8_0000
Synthesizer control register	FMPLL_SYNCR	32-bit	Base + 0x0000
Synthesizer status register	FMPLL_SYNSR	32-bit	Base + 0x0004
Reserved	—	—	(Base + 0x0008)–0xC3F8_3FFF
Chapter 12, "External Bus Interface (EBI)"			0xC3F8_4000
Module configuration register	EBI_MCR	32-bit	Base + 0x0000
Reserved	—	—	Base + 0x0004
Transfer error status register	EBI_TESR	32-bit	Base + 0x0008
Bus monitor control register	EBI_BMCR	32-bit	Base + 0x000C
Base register bank 0	EBI_BR0	32-bit	Base + 0x0010
Option register bank 0	EBI_OR0	32-bit	Base + 0x0014
Base register bank 1	EBI_BR1	32-bit	Base + 0x0018
Option register bank 1	EBI_OR1	32-bit	Base + 0x001C
Base register bank 2	EBI_BR2	32-bit	Base + 0x0020
Option register bank 2	EBI_OR2	32-bit	Base + 0x0024
Base register bank 3	EBI_BR3	32-bit	Base + 0x0028
Option register bank 3	EBI_OR3	32-bit	Base + 0x002C
Reserved	—	—	Base + (0x0030–0x003C)
EBI Calibration Base Register Bank 0	EBI_CAL_BR0	32-bit	Base + 0x0040
EBI Calibration Option Register Bank 0	EBI_CAL_OR0	32-bit	Base + 0x0044
EBI Calibration Base Register Bank 1	EBI_CAL_BR1	32-bit	Base + 0x0048
EBI Calibration Option Register Bank 1	EBI_CAL_OR1	32-bit	Base + 0x004C
EBI Calibration Base Register Bank 2	EBI_CAL_BR2	32-bit	Base + 0x0050
EBI Calibration Option Register Bank 2	EBI_CAL_OR2	32-bit	Base + 0x0054
EBI Calibration Base Register Bank 3	EBI_CAL_BR3	32-bit	Base + 0x0058

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
EBI Calibration Option Register Bank 3	EBI_CAL_OR3	32-bit	Base + 0x005C
Chapter 13, "Flash Memory"			0xC3F8_8000
Module configuration register	FLASH_MCR	32-bit	Base + 0x0000
Low/mid address space block locking register	FLASH_LMLR	32-bit	Base + 0x0004
High address space block locking register	FLASH_HLR	32-bit	Base + 0x0008
Secondary low/mid address space block locking register	FLASH_SLMLR	32-bit	Base + 0x000C
Low/mid address block select register	FLASH_LMSR	32-bit	Base + 0x0010
High address space block select register	FLASH_HSR	32-bit	Base + 0x0014
Address register	FLASH_AR	32-bit	Base + 0x0018
Bus interface unit control register	FLASH_BIUCR	32-bit	Base + 0x001C
Bus interface unit access protection register	FLASH_BIUAPR	32-bit	Base + 0x0020
Flash bus interface unit control register 2	FLASH_BIUCR2	32-bit	Base + 0x0024
Reserved	—	—	(Base + 0x0028)–0xC3F8_FFFF
Chapter 6, "System Integration Unit (SIU)"			0xC3F9_0000
MCU ID Register	SIU_MIDR		Base + 0x0004
Reserved	—	—	Base + (0x0008–0x000B)
Reset status register	SIU_RSR		Base + 0x000C
System reset control register	SIU_SRCR		Base + 0x0010
External interrupt status register	SIU_EISR		Base + 0x0014
DMA/Interrupt request enable register	SIU_DIRER		Base + 0x0018
DMA/Interrupt request status register	SIU_DIRSR		Base + 0x001C
Overrun status register	SIU_OSR		Base + 0x0020
Overrun request enable register	SIU_ORER		Base + 0x0024
$\overline{\text{IRQ}}$ rising-edge event enable register	SIU_IREER		Base + 0x0028
$\overline{\text{IRQ}}$ falling-edge event enable register	SIU_IFEER		Base + 0x002C
$\overline{\text{IRQ}}$ digital filter register	SIU_IDFR		Base + 0x0030
Reserved	—	—	Base + (0x0034–0x003F)
Pad configuration register 0 ($\overline{\text{CS}}$[0])	SIU_PCR0	16-bits	Base + 0x0040
Pad configuration register 1 ($\overline{\text{CS}}$[1])	SIU_PCR1	16-bits	Base + 0x0042
Pad configuration register 2 ($\overline{\text{CS}}$[2])	SIU_PCR2	16-bits	Base + 0x0044
Pad configuration register 3 ($\overline{\text{CS}}$[3])	SIU_PCR3	16-bits	Base + 0x0046

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Pad configuration register 8 (ADDR[12])	SIU_PCR8	16-bits	Base + 0x0050
Pad configuration register 9 (ADDR[13])	SIU_PCR9	16-bits	Base + 0x0052
Pad configuration register 10 (ADDR[14])	SIU_PCR10	16-bits	Base + 0x0054
Pad configuration register 11 (ADDR[15])	SIU_PCR11	16-bits	Base + 0x0056
Pad configuration register 12 (ADDR[16])	SIU_PCR12	16-bits	Base + 0x0058
Pad configuration register 13 (ADDR[17])	SIU_PCR13	16-bits	Base + 0x005A
Pad configuration register 14 (ADDR[18])	SIU_PCR14	16-bits	Base + 0x005C
Pad configuration register 15 (ADDR[19])	SIU_PCR15	16-bits	Base + 0x005E
Pad configuration register 16 (ADDR[20])	SIU_PCR16	16-bits	Base + 0x0060
Pad configuration register 17 (ADDR[21])	SIU_PCR17	16-bits	Base + 0x0062
Pad configuration register 18 (ADDR[22])	SIU_PCR18	16-bits	Base + 0x0064
Pad configuration register 19 (ADDR[23])	SIU_PCR19	16-bits	Base + 0x0066
Pad configuration register 20 (ADDR[24])	SIU_PCR20	16-bits	Base + 0x0068
Pad configuration register 21 (ADDR[25])	SIU_PCR21	16-bits	Base + 0x006A
Pad configuration register 22 (ADDR[26])	SIU_PCR22	16-bits	Base + 0x006C
Pad configuration register 23 (ADDR[27])	SIU_PCR23	16-bits	Base + 0x006E
Pad configuration register 24 (ADDR[28])	SIU_PCR24	16-bits	Base + 0x0070
Pad configuration register 25 (ADDR[29])	SIU_PCR25	16-bits	Base + 0x0072
Pad configuration register 26 (ADDR[30])	SIU_PCR26	16-bits	Base + 0x0074
Pad configuration register 27 (ADDR[31])	SIU_PCR27	16-bits	Base + 0x0076
Pad configuration register 28 (DATA[0])	SIU_PCR28	16-bits	Base + 0x0078
Pad configuration register 29 (DATA[1])	SIU_PCR29	16-bits	Base + 0x007A
Pad configuration register 30 (DATA[2])	SIU_PCR30	16-bits	Base + 0x007C
Pad configuration register 31 (DATA[3])	SIU_PCR31	16-bits	Base + 0x007E
Pad configuration register 32 (DATA[4])	SIU_PCR32	16-bits	Base + 0x0080
Pad configuration register 33 (DATA[5])	SIU_PCR33	16-bits	Base + 0x0082
Pad configuration register 34 (DATA[6])	SIU_PCR34	16-bits	Base + 0x0084
Pad configuration register 35 (DATA[7])	SIU_PCR35	16-bits	Base + 0x0086
Pad configuration register 36 (DATA[8])	SIU_PCR36	16-bits	Base + 0x0088
Pad configuration register 37 (DATA[9])	SIU_PCR37	16-bits	Base + 0x008A
Pad configuration register 38 (DATA[10])	SIU_PCR38	16-bits	Base + 0x008C
Pad configuration register 39 (DATA[11])	SIU_PCR39	16-bits	Base + 0x008E

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Pad configuration register 40 (DATA[12])	SIU_PCR40	16-bits	Base + 0x0090
Pad configuration register 41 (DATA[13])	SIU_PCR41	16-bits	Base + 0x0092
Pad configuration register 42 (DATA[14])	SIU_PCR42	16-bits	Base + 0x0094
Pad configuration register 43 (DATA[15])	SIU_PCR43	16-bits	Base + 0x0096
Pad configuration register 62 (RD_$\overline{WR}$)	SIU_PCR62	16-bits	Base + 0x00BC
Pad configuration register 63 ($\overline{BDIP}$)	SIU_PCR63	16-bits	Base + 0x00BE
Pad configuration register 64 ($\overline{WE}$/$\overline{BE}$[0])	SIU_PCR64	16-bits	Base + 0x00C0
Pad configuration register 65 ($\overline{WE}$/$\overline{BE}$[1])	SIU_PCR65	16-bits	Base + 0x00C2
Pad configuration register 68 ($\overline{OE}$)	SIU_PCR68	16-bits	Base + 0x00C8
Pad configuration register 69 ($\overline{TS}$)	SIU_PCR69	16-bits	Base + 0x00CA
Pad configuration register 70 ($\overline{TA}$)	SIU_PCR70	16-bits	Base + 0x00CC
Reserved	—	—	Base + (0x00D0–0x00D4)
Pad configuration register 75 (MDO[4])	SIU_PCR75	16-bits	Base + 0x00D6
Pad configuration register 76 (MDO[5])	SIU_PCR76	16-bits	Base + 0x00D8
Pad configuration register 77 (MDO[6])	SIU_PCR77	16-bits	Base + 0x00DA
Pad configuration register 78 (MDO[7])	SIU_PCR78	16-bits	Base + 0x00DC
Pad configuration register 79 (MDO[8])	SIU_PCR79	16-bits	Base + 0x00DE
Pad configuration register 80 (MDO[9])	SIU_PCR80	16-bits	Base + 0x00E0
Pad configuration register 81 (MDO[10])	SIU_PCR81	16-bits	Base + 0x00E2
Pad configuration register 82 (MDO[11])	SIU_PCR82	16-bits	Base + 0x00E4
Pad configuration register 83 (CNTXA)	SIU_PCR83	16-bits	Base + 0x00E6
Pad configuration register 84 (CNRXA)	SIU_PCR84	16-bits	Base + 0x00E8
Pad configuration register 85 (CNTXB)	SIU_PCR85	16-bits	Base + 0x00EA
Pad configuration register 86 (CNRXB)	SIU_PCR86	16-bits	Base + 0x00EC
Pad configuration register 87 (CNTXC)	SIU_PCR87	16-bits	Base + 0x00EE
Pad configuration register 88 (CNRXC)	SIU_PCR88	16-bits	Base + 0x00F0
Pad configuration register 89 (TXDA)	SIU_PCR89	16-bits	Base + 0x00F2
Pad configuration register 90 (RXDA)	SIU_PCR90	16-bits	Base + 0x00F4
Pad configuration register 91 (TXDB)	SIU_PCR91	16-bits	Base + 0x00F6
Pad configuration register 92 (RXDB)	SIU_PCR92	16-bits	Base + 0x00F8
Pad configuration register 93 (SCKA)	SIU_PCR93	16-bits	Base + 0x00FA
Pad configuration register 94 (SINA)	SIU_PCR94	16-bits	Base + 0x00FC

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Pad configuration register 95 (SOUTA)	SIU_PCR95	16-bits	Base + 0x00FE
Pad configuration register 96 (PCSA[0])	SIU_PCR96	16-bits	Base + 0x0100
Pad configuration register 97 (PCSA[1])	SIU_PCR97	16-bits	Base + 0x0102
Pad configuration register 98 (PCSA[2])	SIU_PCR98	16-bits	Base + 0x0104
Pad configuration register 99 (PCSA[3])	SIU_PCR99	16-bits	Base + 0x0106
Pad configuration register 100 (PCSA[4])	SIU_PCR100	16-bits	Base + 0x0108
Pad configuration register 101 (PCSA[5])	SIU_PCR101	16-bits	Base + 0x010A
Pad configuration register 102 (SCKB)	SIU_PCR102	16-bits	Base + 0x010C
Pad configuration register 103 (SINB)	SIU_PCR103	16-bits	Base + 0x010E
Pad configuration register 104 (SOUTB)	SIU_PCR104	16-bits	Base + 0x0110
Pad configuration register 105 (PCSB[0])	SIU_PCR105	16-bits	Base + 0x0112
Pad configuration register 106 (PCSB[1])	SIU_PCR106	16-bits	Base + 0x0114
Pad configuration register 107 (PCSB[2])	SIU_PCR107	16-bits	Base + 0x0116
Pad configuration register 108 (PCSB[3])	SIU_PCR108	16-bits	Base + 0x0118
Pad configuration register 109 (PCSB[4])	SIU_PCR109	16-bits	Base + 0x011A
Pad configuration register 110 (PCSB[5])	SIU_PCR110	16-bits	Base + 0x011C
Reserved	—	—	Base + (0x011E–0x0120)
Pad configuration register 113 (TCRCLKA)	SIU_PCR113	16-bits	Base + 0x0122
Pad configuration register 114 (ETPUA[0])	SIU_PCR114	16-bits	Base + 0x0124
Pad configuration register 115 (ETPUA[1])	SIU_PCR115	16-bits	Base + 0x0126
Pad configuration register 116 (ETPUA[2])	SIU_PCR116	16-bits	Base + 0x0128
Pad configuration register 117 (ETPUA[3])	SIU_PCR117	16-bits	Base + 0x012A
Pad configuration register 118 (ETPUA[4])	SIU_PCR118	16-bits	Base + 0x012C
Pad configuration register 119 (ETPUA[5])	SIU_PCR119	16-bits	Base + 0x012E
Pad configuration register 120 (ETPUA[6])	SIU_PCR120	16-bits	Base + 0x0130
Pad configuration register 121 (ETPUA[7])	SIU_PCR121	16-bits	Base + 0x0132
Pad configuration register 122 (ETPUA[8])	SIU_PCR122	16-bits	Base + 0x0134
Pad configuration register 123 (ETPUA[9])	SIU_PCR123	16-bits	Base + 0x0136
Pad configuration register 124 (ETPUA[10])	SIU_PCR124	16-bits	Base + 0x0138
Pad configuration register 125 (ETPUA[11])	SIU_PCR125	16-bits	Base + 0x013A
Pad configuration register 126 (ETPUA[12])	SIU_PCR126	16-bits	Base + 0x013C
Pad configuration register 127 (ETPUA[13])	SIU_PCR127	16-bits	Base + 0x013E

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Pad configuration register 128 (ETPUA[14])	SIU_PCR128	16-bits	Base + 0x0140
Pad configuration register 129 (ETPUA[15])	SIU_PCR129	16-bits	Base + 0x0142
Pad configuration register 130 (ETPUA[16])	SIU_PCR130	16-bits	Base + 0x0144
Pad configuration register 131 (ETPUA[17])	SIU_PCR131	16-bits	Base + 0x0146
Pad configuration register 132 (ETPUA[18])	SIU_PCR132	16-bits	Base + 0x0148
Pad configuration register 133 (ETPUA[19])	SIU_PCR133	16-bits	Base + 0x014A
Pad configuration register 134 (ETPUA[20])	SIU_PCR134	16-bits	Base + 0x014C
Pad configuration register 142 (ETPUA[28])	SIU_PCR142	16-bits	Base + 0x015C
Pad configuration register 143 (ETPUA[29])	SIU_PCR143	16-bits	Base + 0x015E
Pad configuration register 144 (ETPUA[30])	SIU_PCR144	16-bits	Base + 0x0160
Pad configuration register 145 (ETPUA[31])	SIU_PCR145	16-bits	Base + 0x0162
Reserved	—	—	Base + (0x0164–0x01A4)
Pad configuration register 179 (EMIOS[0])	SIU_PCR179	16-bits	Base + 0x01A6
Pad configuration register 180 (EMIOS[1])	SIU_PCR180	16-bits	Base + 0x01A8
Pad configuration register 181 (EMIOS[2])	SIU_PCR181	16-bits	Base + 0x01AA
Pad configuration register 182 (EMIOS[3])	SIU_PCR182	16-bits	Base + 0x01AC
Pad configuration register 183 (EMIOS[4])	SIU_PCR183	16-bits	Base + 0x01AE
Pad configuration register 184 (EMIOS[5])	SIU_PCR184	16-bits	Base + 0x01B0
Pad configuration register 185 (EMIOS[6])	SIU_PCR185	16-bits	Base + 0x01B2
Pad configuration register 186 (EMIOS[7])	SIU_PCR186	16-bits	Base + 0x01B4
Pad configuration register 187 (EMIOS[8])	SIU_PCR187	16-bits	Base + 0x01B6
Pad configuration register 188 (EMIOS[9])	SIU_PCR188	16-bits	Base + 0x01B8
Pad configuration register 189 (EMIOS[10])	SIU_PCR189	16-bits	Base + 0x01BA
Pad configuration register 190 (EMIOS[11])	SIU_PCR190	16-bits	Base + 0x01BC
Pad configuration register 191 (EMIOS[12])	SIU_PCR191	16-bits	Base + 0x01BE
Pad configuration register 192 (EMIOS[13])	SIU_PCR192	16-bits	Base + 0x01C0
Pad configuration register 193 (EMIOS[14])	SIU_PCR193	16-bits	Base + 0x01C2
Pad configuration register 194 (EMIOS[15])	SIU_PCR194	16-bits	Base + 0x01C4
Pad configuration register 195 (EMIOS[16])	SIU_PCR195	16-bits	Base + 0x01C6
Pad configuration register 196 (EMIOS[17])	SIU_PCR196	16-bits	Base + 0x01C8
Pad configuration register 197 (EMIOS[18])	SIU_PCR197	16-bits	Base + 0x01CA
Pad configuration register 198 (EMIOS[19])	SIU_PCR198	16-bits	Base + 0x01CC

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Pad configuration register 199 (EMIOS[20])	SIU_PCR199	16-bits	Base + 0x01CE
Pad configuration register 200 (EMIOS[21])	SIU_PCR200	16-bits	Base + 0x01D0
Pad configuration register 201 (EMIOS[22])	SIU_PCR201	16-bits	Base + 0x01D2
Pad configuration register 202 (EMIOS[23])	SIU_PCR202	16-bits	Base + 0x01D4
Pad configuration register 203 (GPIO[203])	SIU_PCR203	16-bits	Base + 0x01D6
Pad configuration register 204 (GPIO[204])	SIU_PCR204	16-bits	Base + 0x01D8
Reserved	—	—	Base + 0x01DA
Pad configuration register 206 (GPIO[206])	SIU_PCR206	16-bits	Base + 0x01DC
Pad configuration register 207 (GPIO[207])	SIU_PCR207	16-bits	Base + 0x01DE
Pad configuration register 208 (PLLCFG[0])	SIU_PCR208	16-bits	Base + 0x01E0
Pad configuration register 209 (PLLCFG[1])	SIU_PCR209	16-bits	Base + 0x01E2
Pad configuration register 210 ($\overline{\text{RSTCFG}}$)	SIU_PCR210	16-bits	Base + 0x01E4
Pad configuration register 211 (BOOTCFG[0])	SIU_PCR211	16-bits	Base + 0x01E6
Pad configuration register 212 (BOOTCFG[1])	SIU_PCR212	16-bits	Base + 0x01E8
Pad configuration register 213 (WKPCFG)	SIU_PCR213	16-bits	Base + 0x01EA
Pad configuration register 214 (ENGCLK)	SIU_PCR214	16-bits	Base + 0x01EC
Pad configuration register 215 (AN[12])	SIU_PCR215	16-bits	Base + 0x01EE
Pad configuration register 216 (AN[13])	SIU_PCR216	16-bits	Base + 0x01F0
Pad configuration register 217 (AN[14])	SIU_PCR217	16-bits	Base + 0x01F2
Pad configuration register 218 (AN[15])	SIU_PCR218	16-bits	Base + 0x01F4
Pad configuration register 219 (MCKO)	SIU_PCR219	16-bits	Base + 0x01F6
Pad configuration register 220 (MDO[0])	SIU_PCR220	16-bits	Base + 0x01F8
Pad configuration register 221 (MDO[1])	SIU_PCR221	16-bits	Base + 0x01FA
Pad configuration register 222 (MDO[2])	SIU_PCR222	16-bits	Base + 0x01FC
Pad configuration register 223 (MDO[3])	SIU_PCR223	16-bits	Base + 0x01FE
Pad configuration register 224 ($\overline{\text{MSEO}}$[0])	SIU_PCR224	16-bits	Base + 0x0200
Pad configuration register 225 ($\overline{\text{MSEO}}$[1])	SIU_PCR225	16-bits	Base + 0x0202
Pad configuration register 226 ($\overline{\text{RDY}}$)	SIU_PCR226	16-bits	Base + 0x0204
Pad configuration register 227 ($\overline{\text{EVTO}}$)	SIU_PCR227	16-bits	Base + 0x0206
Pad configuration register 228 (TDO)	SIU_PCR228	16-bits	Base + 0x0208
Pad configuration register 229 (CLKOUT)	SIU_PCR229	16-bits	Base + 0x020A
Pad configuration register 230 ($\overline{\text{RSTOUT}}$)	SIU_PCR230	16-bits	Base + 0x020C

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Pad Configuration Register 336 (CAL_$\overline{CS}$[0])	SIU_PCR336	16-bits	Base + 0x02E0
Pad Configuration Register 338 (CAL_$\overline{CS}$[2])	SIU_PCR338	16-bits	Base + 0x02E4
Pad Configuration Register 339 (CAL_$\overline{CS}$[3])	SIU_PCR339	16-bits	Base + 0x02E6
Pad Configuration Register 340 (CAL_ADDR[12:30])	SIU_PCR340	16-bits	Base + 0x02E8
Pad Configuration Register 341 (CAL_DATA[0:15])	SIU_PCR341	16-bits	Base + 0x02EA
Pad Configuration Register 342 (CAL_RD_$\overline{WR}$, CAL_$\overline{WE}$/$\overline{BE}$[0:1], CAL_$\overline{OE}$, and CAL_$\overline{TS}$)	SIU_PCR342	16-bits	Base + 0x02EC
Reserved	—	—	Base + (0x02F0–0x05FF)
GPIO data output register 0 (GPIO[0])	SIU_GPDO0	8-bits	Base + 0x0600
GPIO data output register 1 (GPIO[1])	SIU_GPDO1	8-bits	Base + 0x0601
GPIO data output register 2 (GPIO[2])	SIU_GPDO2	8-bits	Base + 0x0602
GPIO data output register 3 (GPIO[3])	SIU_GPDO3	8-bits	Base + 0x0603
Reserved	—	—	Base + (0x0604–0x0607)
GPIO data output register 8 (GPIO[8])	SIU_GPDO8	8-bits	Base + 0x0608
GPIO data output register 9 (GPIO[9])	SIU_GPDO9	8-bits	Base + 0x0609
GPIO data output register 10 (GPIO[10])	SIU_GPDO10	8-bits	Base + 0x060A
GPIO data output register 11 (GPIO[11])	SIU_GPDO11	8-bits	Base + 0x060B
GPIO data output register 12 (GPIO[12])	SIU_GPDO12	8-bits	Base + 0x060C
GPIO data output register 13 (GPIO[13])	SIU_GPDO13	8-bits	Base + 0x060D
GPIO data output register 14 (GPIO[14])	SIU_GPDO14	8-bits	Base + 0x060E
GPIO data output register 15 (GPIO[15])	SIU_GPDO15	8-bits	Base + 0x060F
GPIO data output register 16 (GPIO[16])	SIU_GPDO16	8-bits	Base + 0x0610
GPIO data output register 17 (GPIO[17])	SIU_GPDO17	8-bits	Base + 0x0611
GPIO data output register 18 (GPIO[18])	SIU_GPDO18	8-bits	Base + 0x0612
GPIO data output register 19 (GPIO[19])	SIU_GPDO19	8-bits	Base + 0x0613
GPIO data output register 20 (GPIO[20])	SIU_GPDO20	8-bits	Base + 0x0614
GPIO data output register 21 (GPIO[21])	SIU_GPDO21	8-bits	Base + 0x0615
GPIO data output register 22 (GPIO[22])	SIU_GPDO22	8-bits	Base + 0x0616
GPIO data output register 23 (GPIO[23])	SIU_GPDO23	8-bits	Base + 0x0617
GPIO data output register 24 (GPIO[24])	SIU_GPDO24	8-bits	Base + 0x0618
GPIO data output register 25 (GPIO[25])	SIU_GPDO25	8-bits	Base + 0x0619
GPIO data output register 26 (GPIO[26])	SIU_GPDO26	8-bits	Base + 0x061A

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data output register 27 (GPIO[27])	SIU_GPDO27	8-bits	Base + 0x061B
GPIO data output register 28 (GPIO[28])	SIU_GPDO28	8-bits	Base + 0x061C
GPIO data output register 29 (GPIO[29])	SIU_GPDO29	8-bits	Base + 0x061D
GPIO data output register 30 (GPIO[30])	SIU_GPDO30	8-bits	Base + 0x061E
GPIO data output register 31 (GPIO[31])	SIU_GPDO31	8-bits	Base + 0x061F
GPIO data output register 32 (GPIO[32])	SIU_GPDO32	8-bits	Base + 0x0620
GPIO data output register 33 (GPIO[33])	SIU_GPDO33	8-bits	Base + 0x0621
GPIO data output register 34 (GPIO[34])	SIU_GPDO34	8-bits	Base + 0x0622
GPIO data output register 35 (GPIO[35])	SIU_GPDO35	8-bits	Base + 0x0623
GPIO data output register 36 (GPIO[36])	SIU_GPDO36	8-bits	Base + 0x0624
GPIO data output register 37 (GPIO[37])	SIU_GPDO37	8-bits	Base + 0x0625
GPIO data output register 38 (GPIO[38])	SIU_GPDO38	8-bits	Base + 0x0626
GPIO data output register 39 (GPIO[39])	SIU_GPDO39	8-bits	Base + 0x0627
GPIO data output register 40 (GPIO[40])	SIU_GPDO40	8-bits	Base + 0x0628
GPIO data output register 41 (GPIO[41])	SIU_GPDO41	8-bits	Base + 0x0629
GPIO data output register 42 (GPIO[42])	SIU_GPDO42	8-bits	Base + 0x062A
GPIO data output register 43 (GPIO[43])	SIU_GPDO43	8-bits	Base + 0x062B
Reserved	—	—	Base + (0x062C–0x063D)
GPIO data output register 62 (GPIO[62])	SIU_GPDO62	8-bits	Base + 0x063E
GPIO data output register 63 (GPIO[63])	SIU_GPDO63	8-bits	Base + 0x063F
GPIO data output register 64 (GPIO[64])	SIU_GPDO64	8-bits	Base + 0x0640
GPIO data output register 65 (GPIO[65])	SIU_GPDO65	8-bits	Base + 0x0641
Reserved	—	—	Base + (0x0642–0x0643)
GPIO data output register 68 (GPIO[68])	SIU_GPDO68	8-bits	Base + 0x0644
GPIO data output register 69 (GPIO[69])	SIU_GPDO69	8-bits	Base + 0x0645
GPIO data output register 70 (GPIO[70])	SIU_GPDO70	8-bits	Base + 0x0646
Reserved	—	—	Base + (0x0647–0x0649)
GPIO data output register 74 (GPIO[74])	SIU_GPDO74	8-bits	Base + 0x064A
GPIO data output register 75 (GPIO[75])	SIU_GPDO75	8-bits	Base + 0x064B
GPIO data output register 76 (GPIO[76])	SIU_GPDO76	8-bits	Base + 0x064C
GPIO data output register 77 (GPIO[77])	SIU_GPDO77	8-bits	Base + 0x064D
GPIO data output register 78 (GPIO[78])	SIU_GPDO78	8-bits	Base + 0x064E

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data output register 79 (GPIO[79])	SIU_GPDO79	8-bits	Base + 0x064F
GPIO data output register 80 (GPIO[80])	SIU_GPDO80	8-bits	Base + 0x0650
GPIO data output register 81 (GPIO[81])	SIU_GPDO81	8-bits	Base + 0x0651
GPIO data output register 82 (GPIO[82])	SIU_GPDO82	8-bits	Base + 0x0652
GPIO data output register 83 (GPIO[83])	SIU_GPDO83	8-bits	Base + 0x0653
GPIO data output register 84 (GPIO[84])	SIU_GPDO84	8-bits	Base + 0x0654
GPIO data output register 85 (GPIO[85])	SIU_GPDO85	8-bits	Base + 0x0655
GPIO data output register 86 (GPIO[86])	SIU_GPDO86	8-bits	Base + 0x0656
GPIO data output register 87 (GPIO[87])	SIU_GPDO87	8-bits	Base + 0x0657
GPIO data output register 88 (GPIO[88])	SIU_GPDO88	8-bits	Base + 0x0658
GPIO data output register 89 (GPIO[89])	SIU_GPDO89	8-bits	Base + 0x0659
GPIO data output register 90 (GPIO[90])	SIU_GPDO90	8-bits	Base + 0x065A
GPIO data output register 91 (GPIO[91])	SIU_GPDO91	8-bits	Base + 0x065B
GPIO data output register 92 (GPIO[92])	SIU_GPDO92	8-bits	Base + 0x065C
GPIO data output register 93 (GPIO[93])	SIU_GPDO93	8-bits	Base + 0x065D
GPIO data output register 94 (GPIO[94])	SIU_GPDO94	8-bits	Base + 0x065E
GPIO data output register 95 (GPIO[95])	SIU_GPDO95	8-bits	Base + 0x065F
GPIO data output register 96 (GPIO[96])	SIU_GPDO96	8-bits	Base + 0x0660
GPIO data output register 97 (GPIO[97])	SIU_GPDO97	8-bits	Base + 0x0661
GPIO data output register 98 (GPIO[98])	SIU_GPDO98	8-bits	Base + 0x0662
GPIO data output register 99 (GPIO[99])	SIU_GPDO99	8-bits	Base + 0x0663
GPIO data output register 100 (GPIO[100])	SIU_GPDO100	8-bits	Base + 0x0664
GPIO data output register 101 (GPIO[101])	SIU_GPDO101	8-bits	Base + 0x0665
GPIO data output register 102 (GPIO[102])	SIU_GPDO102	8-bits	Base + 0x0666
GPIO data output register 103 (GPIO[103])	SIU_GPDO103	8-bits	Base + 0x0667
GPIO data output register 104 (GPIO[104])	SIU_GPDO104	8-bits	Base + 0x0668
GPIO data output register 105 (GPIO[105])	SIU_GPDO105	8-bits	Base + 0x0669
GPIO data output register 106 (GPIO[106])	SIU_GPDO106	8-bits	Base + 0x066A
GPIO data output register 107 (GPIO[107])	SIU_GPDO107	8-bits	Base + 0x066B
GPIO data output register 108 (GPIO[108])	SIU_GPDO108	8-bits	Base + 0x066C
GPIO data output register 109 (GPIO[109])	SIU_GPDO109	8-bits	Base + 0x066D
GPIO data output register 110 (GPIO[110])	SIU_GPDO110	8-bits	Base + 0x066E

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Reserved	—	—	Base + (0x066F–0x0670)
GPIO data output register 113 (GPIO[113])	SIU_GPDO113	8-bits	Base + 0x0671
GPIO data output register 114 (GPIO[114])	SIU_GPDO114	8-bits	Base + 0x0672
GPIO data output register 115 (GPIO[115])	SIU_GPDO115	8-bits	Base + 0x0673
GPIO data output register 116 (GPIO[116])	SIU_GPDO116	8-bits	Base + 0x0674
GPIO data output register 117 (GPIO[117])	SIU_GPDO117	8-bits	Base + 0x0675
GPIO data output register 118 (GPIO[118])	SIU_GPDO118	8-bits	Base + 0x0676
GPIO data output register 119 (GPIO[119])	SIU_GPDO119	8-bits	Base + 0x0677
GPIO data output register 120 (GPIO[120])	SIU_GPDO120	8-bits	Base + 0x0678
GPIO data output register 121 (GPIO[121])	SIU_GPDO121	8-bits	Base + 0x0679
GPIO data output register 122 (GPIO[122])	SIU_GPDO122	8-bits	Base + 0x067A
GPIO data output register 123 (GPIO[123])	SIU_GPDO123	8-bits	Base + 0x067B
GPIO data output register 124 (GPIO[124])	SIU_GPDO124	8-bits	Base + 0x067C
GPIO data output register 125 (GPIO[125])	SIU_GPDO125	8-bits	Base + 0x067D
GPIO data output register 126 (GPIO[126])	SIU_GPDO126	8-bits	Base + 0x067E
GPIO data output register 127 (GPIO[127])	SIU_GPDO127	8-bits	Base + 0x067F
GPIO data output register 128 (GPIO[128])	SIU_GPDO128	8-bits	Base + 0x0680
GPIO data output register 129 (GPIO[129])	SIU_GPDO129	8-bits	Base + 0x0681
GPIO data output register 130 (GPIO[130])	SIU_GPDO130	8-bits	Base + 0x0682
GPIO data output register 131 (GPIO[131])	SIU_GPDO131	8-bits	Base + 0x0683
GPIO data output register 132 (GPIO[132])	SIU_GPDO132	8-bits	Base + 0x0684
GPIO data output register 133 (GPIO[133])	SIU_GPDO133	8-bits	Base + 0x0685
GPIO data output register 134 (GPIO[134])	SIU_GPDO134	8-bits	Base + 0x0686
GPIO data output register 135 (GPIO[135])	SIU_GPDO135	8-bits	Base + 0x0687
GPIO data output register 136 (GPIO[136])	SIU_GPDO136	8-bits	Base + 0x0688
GPIO data output register 137 (GPIO[137])	SIU_GPDO137	8-bits	Base + 0x0689
GPIO data output register 138 (GPIO[138])	SIU_GPDO138	8-bits	Base + 0x068A
GPIO data output register 139 (GPIO[139])	SIU_GPDO139	8-bits	Base + 0x068B
GPIO data output register 140 (GPIO[140])	SIU_GPDO140	8-bits	Base + 0x068C
GPIO data output register 141 (GPIO[141])	SIU_GPDO141	8-bits	Base + 0x068D
GPIO data output register 142 (GPIO[142])	SIU_GPDO142	8-bits	Base + 0x068E
GPIO data output register 143 (GPIO[143])	SIU_GPDO143	8-bits	Base + 0x068F

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data output register 144 (GPIO[144])	SIU_GPDO144	8-bits	Base + 0x0690
GPIO data output register 145 (GPIO[145])	SIU_GPDO145	8-bits	Base + 0x0691
GPIO data output register 146 (GPIO[146])	SIU_GPDO146	8-bits	Base + 0x0692
GPIO data output register 147 (GPIO[147])	SIU_GPDO147	8-bits	Base + 0x0693
GPIO data output register 148 (GPIO[148])	SIU_GPDO148	8-bits	Base + 0x0694
GPIO data output register 149 (GPIO[149])	SIU_GPDO149	8-bits	Base + 0x0695
GPIO data output register 150 (GPIO[150])	SIU_GPDO150	8-bits	Base + 0x0696
GPIO data output register 151 (GPIO[151])	SIU_GPDO151	8-bits	Base + 0x0697
GPIO data output register 152 (GPIO[152])	SIU_GPDO152	8-bits	Base + 0x0698
GPIO data output register 153 (GPIO[153])	SIU_GPDO153	8-bits	Base + 0x0699
GPIO data output register 154 (GPIO[154])	SIU_GPDO154	8-bits	Base + 0x069A
GPIO data output register 155 (GPIO[155])	SIU_GPDO155	8-bits	Base + 0x069B
GPIO data output register 156 (GPIO[156])	SIU_GPDO156	8-bits	Base + 0x069C
GPIO data output register 157 (GPIO[157])	SIU_GPDO157	8-bits	Base + 0x069D
GPIO data output register 158 (GPIO[158])	SIU_GPDO158	8-bits	Base + 0x069E
GPIO data output register 159 (GPIO[159])	SIU_GPDO159	8-bits	Base + 0x069F
GPIO data output register 160 (GPIO[160])	SIU_GPDO160	8-bits	Base + 0x06A0
GPIO data output register 161 (GPIO[161])	SIU_GPDO161	8-bits	Base + 0x06A1
GPIO data output register 162 (GPIO[162])	SIU_GPDO162	8-bits	Base + 0x06A2
GPIO data output register 163 (GPIO[163])	SIU_GPDO163	8-bits	Base + 0x06A3
GPIO data output register 164 (GPIO[164])	SIU_GPDO164	8-bits	Base + 0x06A4
GPIO data output register 165 (GPIO[165])	SIU_GPDO165	8-bits	Base + 0x06A5
GPIO data output register 166 (GPIO[166])	SIU_GPDO166	8-bits	Base + 0x06A6
GPIO data output register 167 (GPIO[167])	SIU_GPDO167	8-bits	Base + 0x06A7
GPIO data output register 168 (GPIO[168])	SIU_GPDO168	8-bits	Base + 0x06A8
GPIO data output register 169 (GPIO[169])	SIU_GPDO169	8-bits	Base + 0x06A9
GPIO data output register 170 (GPIO[170])	SIU_GPDO170	8-bits	Base + 0x06AA
GPIO data output register 171 (GPIO[171])	SIU_GPDO171	8-bits	Base + 0x06AB
GPIO data output register 172 (GPIO[172])	SIU_GPDO172	8-bits	Base + 0x06AC
GPIO data output register 173 (GPIO[173])	SIU_GPDO173	8-bits	Base + 0x06AD
GPIO data output register 174 (GPIO[174])	SIU_GPDO174	8-bits	Base + 0x06AE
GPIO data output register 175 (GPIO[175])	SIU_GPDO175	8-bits	Base + 0x06AF

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data output register 176 (GPIO[176])	SIU_GPDO176	8-bits	Base + 0x06B0
GPIO data output register 177 (GPIO[177])	SIU_GPDO177	8-bits	Base + 0x06B1
GPIO data output register 178 (GPIO[178])	SIU_GPDO178	8-bits	Base + 0x06B2
GPIO data output register 179 (GPIO[179])	SIU_GPDO179	8-bits	Base + 0x06B3
GPIO data output register 180 (GPIO[180])	SIU_GPDO180	8-bits	Base + 0x06B4
GPIO data output register 181 (GPIO[181])	SIU_GPDO181	8-bits	Base + 0x06B5
GPIO data output register 182 (GPIO[182])	SIU_GPDO182	8-bits	Base + 0x06B6
GPIO data output register 183 (GPIO[183])	SIU_GPDO183	8-bits	Base + 0x06B7
GPIO data output register 184 (GPIO[184])	SIU_GPDO184	8-bits	Base + 0x06B8
GPIO data output register 185 (GPIO[185])	SIU_GPDO185	8-bits	Base + 0x06B9
GPIO data output register 186 (GPIO[186])	SIU_GPDO186	8-bits	Base + 0x06BA
GPIO data output register 187 (GPIO[187])	SIU_GPDO187	8-bits	Base + 0x06BB
GPIO data output register 188 (GPIO[188])	SIU_GPDO188	8-bits	Base + 0x06BC
GPIO data output register 189 (GPIO[189])	SIU_GPDO189	8-bits	Base + 0x06BD
GPIO data output register 190 (GPIO[190])	SIU_GPDO190	8-bits	Base + 0x06BE
GPIO data output register 191 (GPIO[191])	SIU_GPDO191	8-bits	Base + 0x06BF
GPIO data output register 192 (GPIO[192])	SIU_GPDO192	8-bits	Base + 0x06C0
GPIO data output register 193 (GPIO[193])	SIU_GPDO193	8-bits	Base + 0x06C1
GPIO data output register 194 (GPIO[194])	SIU_GPDO194	8-bits	Base + 0x06C2
GPIO data output register 195 (GPIO[195])	SIU_GPDO195	8-bits	Base + 0x06C3
GPIO data output register 196 (GPIO[196])	SIU_GPDO196	8-bits	Base + 0x06C4
GPIO data output register 197 (GPIO[197])	SIU_GPDO197	8-bits	Base + 0x06C5
GPIO data output register 198 (GPIO[198])	SIU_GPDO198	8-bits	Base + 0x06C6
GPIO data output register 199 (GPIO[199])	SIU_GPDO199	8-bits	Base + 0x06C7
GPIO data output register 200 (GPIO[200])	SIU_GPDO200	8-bits	Base + 0x06C8
GPIO data output register 201 (GPIO[201])	SIU_GPDO201	8-bits	Base + 0x06C9
GPIO data output register 202 (GPIO[202])	SIU_GPDO202	8-bits	Base + 0x06CA
GPIO data output register 203 (GPIO[203])	SIU_GPDO203	8-bits	Base + 0x06CB
GPIO data output register 204 (GPIO[204])	SIU_GPDO204	8-bits	Base + 0x06CC
Reserved	—	—	Base + 0x06CD
GPIO data output register 206 (GPIO[206])	SIU_GPDO206	8-bits	Base + 0x06CE
GPIO data output register 207 (GPIO[207])	SIU_GPDO207	8-bits	Base + 0x06CF

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data output register 208 (GPIO[208])	SIU_GPDO208	8-bits	Base + 0x06D0
GPIO data output register 209 (GPIO[209])	SIU_GPDO209	8-bits	Base + 0x06D1
GPIO data output register 210 (GPIO[210])	SIU_GPDO210	8-bits	Base + 0x06D2
GPIO data output register 211 (GPIO[211])	SIU_GPDO211	8-bits	Base + 0x06D3
GPIO data output register 212 (GPIO[212])	SIU_GPDO212	8-bits	Base + 0x06D4
GPIO data output register 213 (GPIO[213])	SIU_GPDO213	8-bits	Base + 0x06D5
Reserved	—	—	Base + (0x06D8–0x07FF)
GPIO data input register 0 (GPIO[0])	SIU_GPDI0	8-bits	Base + 0x0800
GPIO data input register 1 (GPIO[1])	SIU_GPDI1	8-bits	Base + 0x0801
GPIO data input register 2 (GPIO[2])	SIU_GPDI2	8-bits	Base + 0x0802
GPIO data input register 3 (GPIO[3])	SIU_GPDI3	8-bits	Base + 0x0803
GPIO data input register 4 (GPIO[4])	SIU_GPDI4	8-bits	Base + 0x0804
GPIO data input register 5 (GPIO[5])	SIU_GPDI5	8-bits	Base + 0x0805
GPIO data input register 6 (GPIO[6])	SIU_GPDI6	8-bits	Base + 0x0806
GPIO data input register 7 (GPIO[7])	SIU_GPDI7	8-bits	Base + 0x0807
GPIO data input register 8 (GPIO[8])	SIU_GPDI8	8-bits	Base + 0x0808
GPIO data input register 9 (GPIO[9])	SIU_GPDI9	8-bits	Base + 0x0809
GPIO data input register 10 (GPIO[10])	SIU_GPDI10	8-bits	Base + 0x080A
GPIO data input register 11 (GPIO[11])	SIU_GPDI11	8-bits	Base + 0x080B
GPIO data input register 12 (GPIO[12])	SIU_GPDI12	8-bits	Base + 0x080C
GPIO data input register 13 (GPIO[13])	SIU_GPDI13	8-bits	Base + 0x080D
GPIO data input register 14 (GPIO[14])	SIU_GPDI14	8-bits	Base + 0x080E
GPIO data input register 15 (GPIO[15])	SIU_GPDI15	8-bits	Base + 0x080F
GPIO data input register 16 (GPIO[16])	SIU_GPDI16	8-bits	Base + 0x0810
GPIO data input register 17 (GPIO[17])	SIU_GPDI17	8-bits	Base + 0x0811
GPIO data input register 18 (GPIO[18])	SIU_GPDI18	8-bits	Base + 0x0812
GPIO data input register 19 (GPIO[19])	SIU_GPDI19	8-bits	Base + 0x0813
GPIO data input register 20 (GPIO[20])	SIU_GPDI20	8-bits	Base + 0x0814
GPIO data input register 21 (GPIO[21])	SIU_GPDI21	8-bits	Base + 0x0815
GPIO data input register 22 (GPIO[22])	SIU_GPDI22	8-bits	Base + 0x0816
GPIO data input register 23 (GPIO[23])	SIU_GPDI23	8-bits	Base + 0x0817
GPIO data input register 24 (GPIO[24])	SIU_GPDI24	8-bits	Base + 0x0818

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data input register 25 (GPIO[25])	SIU_GPDI25	8-bits	Base + 0x0819
GPIO data input register 26 (GPIO[26])	SIU_GPDI26	8-bits	Base + 0x081A
GPIO data input register 27 (GPIO[27])	SIU_GPDI27	8-bits	Base + 0x081B
GPIO data input register 28 (GPIO[28])	SIU_GPDI28	8-bits	Base + 0x081C
GPIO data input register 29 (GPIO[29])	SIU_GPDI29	8-bits	Base + 0x081D
GPIO data input register 30 (GPIO[30])	SIU_GPDI30	8-bits	Base + 0x081E
GPIO data input register 31 (GPIO[31])	SIU_GPDI31	8-bits	Base + 0x081F
GPIO data input register 32 (GPIO[32])	SIU_GPDI32	8-bits	Base + 0x0820
GPIO data input register 33 (GPIO[33])	SIU_GPDI33	8-bits	Base + 0x0821
GPIO data input register 34 (GPIO[34])	SIU_GPDI34	8-bits	Base + 0x0822
GPIO data input register 35 (GPIO[35])	SIU_GPDI35	8-bits	Base + 0x0823
GPIO data input register 36 (GPIO[36])	SIU_GPDI36	8-bits	Base + 0x0824
GPIO data input register 37 (GPIO[37])	SIU_GPDI37	8-bits	Base + 0x0825
GPIO data input register 38 (GPIO[38])	SIU_GPDI38	8-bits	Base + 0x0826
GPIO data input register 39 (GPIO[39])	SIU_GPDI39	8-bits	Base + 0x0827
GPIO data input register 40 (GPIO[40])	SIU_GPDI40	8-bits	Base + 0x0828
GPIO data input register 41 (GPIO[41])	SIU_GPDI41	8-bits	Base + 0x0829
GPIO data input register 42 (GPIO[42])	SIU_GPDI42	8-bits	Base + 0x082A
GPIO data input register 43 (GPIO[43])	SIU_GPDI43	8-bits	Base + 0x082B
Reserved	—	—	Base + (0x082C–0x083D)
GPIO data input register 62 (GPIO[62])	SIU_GPDI62	8-bits	Base + 0x083E
GPIO data input register 63 (GPIO[63])	SIU_GPDI63	8-bits	Base + 0x083F
GPIO data input register 64 (GPIO[64])	SIU_GPDI64	8-bits	Base + 0x0840
GPIO data input register 65 (GPIO[65])	SIU_GPDI65	8-bits	Base + 0x0841
Reserved	—	—	Base + (0x0842–0x0843)
GPIO data input register 68 (GPIO[68])	SIU_GPDI68	8-bits	Base + 0x0844
GPIO data input register 69 (GPIO[69])	SIU_GPDI69	8-bits	Base + 0x0845
GPIO data input register 70 (GPIO[70])	SIU_GPDI70	8-bits	Base + 0x0846
Reserved	—	—	Base + (0x0847–0x0849)
GPIO data input register 74 (GPIO[74])	SIU_GPDI74	8-bits	Base + 0x084A
GPIO data input register 75 (GPIO[75])	SIU_GPDI75	8-bits	Base + 0x084B
GPIO data input register 76 (GPIO[76])	SIU_GPDI76	8-bits	Base + 0x084C

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data input register 77 (GPIO[77])	SIU_GPDI77	8-bits	Base + 0x084D
GPIO data input register 78 (GPIO[78])	SIU_GPDI78	8-bits	Base + 0x084E
GPIO data input register 79 (GPIO[79])	SIU_GPDI79	8-bits	Base + 0x084F
GPIO data input register 80 (GPIO[80])	SIU_GPDI80	8-bits	Base + 0x0850
GPIO data input register 81 (GPIO[81])	SIU_GPDI81	8-bits	Base + 0x0851
GPIO data input register 82 (GPIO[82])	SIU_GPDI82	8-bits	Base + 0x0852
GPIO data input register 83 (GPIO[83])	SIU_GPDI83	8-bits	Base + 0x0853
GPIO data input register 84 (GPIO[84])	SIU_GPDI84	8-bits	Base + 0x0854
GPIO data input register 85 (GPIO[85])	SIU_GPDI85	8-bits	Base + 0x0855
GPIO data input register 86 (GPIO[86])	SIU_GPDI86	8-bits	Base + 0x0856
GPIO data input register 87 (GPIO[87])	SIU_GPDI87	8-bits	Base + 0x0857
GPIO data input register 88 (GPIO[88])	SIU_GPDI88	8-bits	Base + 0x0858
GPIO data input register 89 (GPIO[89])	SIU_GPDI89	8-bits	Base + 0x0859
GPIO data input register 90 (GPIO[90])	SIU_GPDI90	8-bits	Base + 0x085A
GPIO data input register 91 (GPIO[91])	SIU_GPDI91	8-bits	Base + 0x085B
GPIO data input register 92 (GPIO[92])	SIU_GPDI92	8-bits	Base + 0x085C
GPIO data input register 93 (GPIO[93])	SIU_GPDI93	8-bits	Base + 0x085D
GPIO data input register 94 (GPIO[94])	SIU_GPDI94	8-bits	Base + 0x085E
GPIO data input register 95 (GPIO[95])	SIU_GPDI95	8-bits	Base + 0x085F
GPIO data input register 96 (GPIO[96])	SIU_GPDI96	8-bits	Base + 0x0860
GPIO data input register 97 (GPIO[97])	SIU_GPDI97	8-bits	Base + 0x0861
GPIO data input register 98 (GPIO[98])	SIU_GPDI98	8-bits	Base + 0x0862
GPIO data input register 99 (GPIO[99])	SIU_GPDI99	8-bits	Base + 0x0863
GPIO data input register 100 (GPIO[100])	SIU_GPDI100	8-bits	Base + 0x0864
GPIO data input register 101 (GPIO[101])	SIU_GPDI101	8-bits	Base + 0x0865
GPIO data input register 102 (GPIO[102])	SIU_GPDI102	8-bits	Base + 0x0866
GPIO data input register 103 (GPIO[103])	SIU_GPDI103	8-bits	Base + 0x0867
GPIO data input register 104 (GPIO[104])	SIU_GPDI104	8-bits	Base + 0x0868
GPIO data input register 105 (GPIO[105])	SIU_GPDI105	8-bits	Base + 0x0869
GPIO data input register 106 (GPIO[106])	SIU_GPDI106	8-bits	Base + 0x086A
GPIO data input register 107 (GPIO[107])	SIU_GPDI107	8-bits	Base + 0x086B
GPIO data input register 108 (GPIO[108])	SIU_GPDI108	8-bits	Base + 0x086C

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data input register 109 (GPIO[109])	SIU_GPDI109	8-bits	Base + 0x086D
GPIO data input register 110 (GPIO[110])	SIU_GPDI110	8-bits	Base + 0x086E
Reserved	—	—	Base + (0x066F–0x0670)
GPIO data input register 113 (GPIO[113])	SIU_GPDI113	8-bits	Base + 0x0871
GPIO data input register 114 (GPIO[114])	SIU_GPDI114	8-bits	Base + 0x0872
GPIO data input register 115 (GPIO[115])	SIU_GPDI115	8-bits	Base + 0x0873
GPIO data input register 116 (GPIO[116])	SIU_GPDI116	8-bits	Base + 0x0874
GPIO data input register 117 (GPIO[117])	SIU_GPDI117	8-bits	Base + 0x0875
GPIO data input register 118 (GPIO[118])	SIU_GPDI118	8-bits	Base + 0x0876
GPIO data input register 119 (GPIO[119])	SIU_GPDI119	8-bits	Base + 0x0877
GPIO data input register 120 (GPIO[120])	SIU_GPDI120	8-bits	Base + 0x0878
GPIO data input register 121 (GPIO[121])	SIU_GPDI121	8-bits	Base + 0x0879
GPIO data input register 122 (GPIO[122])	SIU_GPDI122	8-bits	Base + 0x087A
GPIO data input register 123 (GPIO[123])	SIU_GPDI123	8-bits	Base + 0x087B
GPIO data input register 124 (GPIO[124])	SIU_GPDI124	8-bits	Base + 0x087C
GPIO data input register 125 (GPIO[125])	SIU_GPDI125	8-bits	Base + 0x087D
GPIO data input register 126 (GPIO[126])	SIU_GPDI126	8-bits	Base + 0x087E
GPIO data input register 127 (GPIO[127])	SIU_GPDI127	8-bits	Base + 0x087F
GPIO data input register 128 (GPIO[128])	SIU_GPDI128	8-bits	Base + 0x0880
GPIO data input register 129 (GPIO[129])	SIU_GPDI129	8-bits	Base + 0x0881
GPIO data input register 130 (GPIO[130])	SIU_GPDI130	8-bits	Base + 0x0882
GPIO data input register 131 (GPIO[131])	SIU_GPDI131	8-bits	Base + 0x0883
GPIO data input register 132 (GPIO[132])	SIU_GPDI132	8-bits	Base + 0x0884
GPIO data input register 133 (GPIO[133])	SIU_GPDI133	8-bits	Base + 0x0885
GPIO data input register 134 (GPIO[134])	SIU_GPDI134	8-bits	Base + 0x0886
GPIO data input register 135 (GPIO[135])	SIU_GPDI135	8-bits	Base + 0x0887
GPIO data input register 136 (GPIO[136])	SIU_GPDI136	8-bits	Base + 0x0888
GPIO data input register 137 (GPIO[137])	SIU_GPDI137	8-bits	Base + 0x0889
GPIO data input register 138 (GPIO[138])	SIU_GPDI138	8-bits	Base + 0x088A
GPIO data input register 139 (GPIO[139])	SIU_GPDI139	8-bits	Base + 0x088B
GPIO data input register 140 (GPIO[140])	SIU_GPDI140	8-bits	Base + 0x088C
GPIO data input register 141 (GPIO[141])	SIU_GPDI141	8-bits	Base + 0x088D

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data input register 142 (GPIO[142])	SIU_GPDI142	8-bits	Base + 0x088E
GPIO data input register 143 (GPIO[143])	SIU_GPDI143	8-bits	Base + 0x088F
GPIO data input register 144 (GPIO[144])	SIU_GPDI144	8-bits	Base + 0x0890
GPIO data input register 145 (GPIO[145])	SIU_GPDI145	8-bits	Base + 0x0891
GPIO data input register 146 (GPIO[146])	SIU_GPDI146	8-bits	Base + 0x0892
GPIO data input register 147 (GPIO[147])	SIU_GPDI147	8-bits	Base + 0x0893
GPIO data input register 148 (GPIO[148])	SIU_GPDI148	8-bits	Base + 0x0894
GPIO data input register 149 (GPIO[149])	SIU_GPDI149	8-bits	Base + 0x0895
GPIO data input register 150 (GPIO[150])	SIU_GPDI150	8-bits	Base + 0x0896
GPIO data input register 151 (GPIO[151])	SIU_GPDI151	8-bits	Base + 0x0897
GPIO data input register 152 (GPIO[152])	SIU_GPDI152	8-bits	Base + 0x0898
GPIO data input register 153 (GPIO[153])	SIU_GPDI153	8-bits	Base + 0x0899
GPIO data input register 154 (GPIO[154])	SIU_GPDI154	8-bits	Base + 0x089A
GPIO data input register 155 (GPIO[155])	SIU_GPDI155	8-bits	Base + 0x089B
GPIO data input register 156 (GPIO[156])	SIU_GPDI156	8-bits	Base + 0x089C
GPIO data input register 157 (GPIO[157])	SIU_GPDI157	8-bits	Base + 0x089D
GPIO data input register 158 (GPIO[158])	SIU_GPDI158	8-bits	Base + 0x089E
GPIO data input register 159 (GPIO[159])	SIU_GPDI159	8-bits	Base + 0x089F
GPIO data input register 160 (GPIO[160])	SIU_GPDI160	8-bits	Base + 0x08A0
GPIO data input register 161 (GPIO[161])	SIU_GPDI161	8-bits	Base + 0x08A1
GPIO data input register 162 (GPIO[162])	SIU_GPDI162	8-bits	Base + 0x08A2
GPIO data input register 163 (GPIO[163])	SIU_GPDI163	8-bits	Base + 0x08A3
GPIO data input register 164 (GPIO[164])	SIU_GPDI164	8-bits	Base + 0x08A4
GPIO data input register 165 (GPIO[165])	SIU_GPDI165	8-bits	Base + 0x08A5
GPIO data input register 166 (GPIO[166])	SIU_GPDI166	8-bits	Base + 0x08A6
GPIO data input register 167 (GPIO[167])	SIU_GPDI167	8-bits	Base + 0x08A7
GPIO data input register 168 (GPIO[168])	SIU_GPDI168	8-bits	Base + 0x08A8
GPIO data input register 169 (GPIO[169])	SIU_GPDI169	8-bits	Base + 0x08A9
GPIO data input register 170 (GPIO[170])	SIU_GPDI170	8-bits	Base + 0x08AA
GPIO data input register 171 (GPIO[171])	SIU_GPDI171	8-bits	Base + 0x08AB
GPIO data input register 172 (GPIO[172])	SIU_GPDI172	8-bits	Base + 0x08AC
GPIO data input register 173 (GPIO[173])	SIU_GPDI173	8-bits	Base + 0x08AD

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data input register 174 (GPIO[174])	SIU_GPDI174	8-bits	Base + 0x08AE
GPIO data input register 175 (GPIO[175])	SIU_GPDI175	8-bits	Base + 0x08AF
GPIO data input register 176 (GPIO[176])	SIU_GPDI176	8-bits	Base + 0x08B0
GPIO data input register 177 (GPIO[177])	SIU_GPDI177	8-bits	Base + 0x08B1
GPIO data input register 178 (GPIO[178])	SIU_GPDI178	8-bits	Base + 0x08B2
GPIO data input register 179 (GPIO[179])	SIU_GPDI179	8-bits	Base + 0x08B3
GPIO data input register 180 (GPIO[180])	SIU_GPDI180	8-bits	Base + 0x08B4
GPIO data input register 181 (GPIO[181])	SIU_GPDI181	8-bits	Base + 0x08B5
GPIO data input register 182 (GPIO[182])	SIU_GPDI182	8-bits	Base + 0x08B6
GPIO data input register 183 (GPIO[183])	SIU_GPDI183	8-bits	Base + 0x08B7
GPIO data input register 184 (GPIO[184])	SIU_GPDI184	8-bits	Base + 0x08B8
GPIO data input register 185 (GPIO[185])	SIU_GPDI185	8-bits	Base + 0x08B9
GPIO data input register 186 (GPIO[186])	SIU_GPDI186	8-bits	Base + 0x08BA
GPIO data input register 187 (GPIO[187])	SIU_GPDI187	8-bits	Base + 0x08BB
GPIO data input register 188 (GPIO[188])	SIU_GPDI188	8-bits	Base + 0x08BC
GPIO data input register 189 (GPIO[189])	SIU_GPDI189	8-bits	Base + 0x08BD
GPIO data input register 190 (GPIO[190])	SIU_GPDI190	8-bits	Base + 0x08BE
GPIO data input register 191 (GPIO[191])	SIU_GPDI191	8-bits	Base + 0x08BF
GPIO data input register 192 (GPIO[192])	SIU_GPDI192	8-bits	Base + 0x08C0
GPIO data input register 193 (GPIO[193])	SIU_GPDI193	8-bits	Base + 0x08C1
GPIO data input register 194 (GPIO[194])	SIU_GPDI194	8-bits	Base + 0x08C2
GPIO data input register 195 (GPIO[195])	SIU_GPDI195	8-bits	Base + 0x08C3
GPIO data input register 196 (GPIO[196])	SIU_GPDI196	8-bits	Base + 0x08C4
GPIO data input register 197 (GPIO[197])	SIU_GPDI197	8-bits	Base + 0x08C5
GPIO data input register 198 (GPIO[198])	SIU_GPDI198	8-bits	Base + 0x08C6
GPIO data input register 199 (GPIO[199])	SIU_GPDI199	8-bits	Base + 0x08C7
GPIO data input register 200 (GPIO[200])	SIU_GPDI200	8-bits	Base + 0x08C8
GPIO data input register 201 (GPIO[201])	SIU_GPDI201	8-bits	Base + 0x08C9
GPIO data input register 202 (GPIO[202])	SIU_GPDI202	8-bits	Base + 0x08CA
GPIO data input register 203 (GPIO[203])	SIU_GPDI203	8-bits	Base + 0x08CB
GPIO data input register 204 (GPIO[204])	SIU_GPDI204	8-bits	Base + 0x08CC
Reserved	—	8-bits	Base + 0x08CD

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
GPIO data input register 206 (GPIO[206])	SIU_GPDI206	8-bits	Base + 0x08CE
GPIO data input register 207 (GPIO[207])	SIU_GPDI207	8-bits	Base + 0x08CF
GPIO data input register 208 (GPIO[208])	SIU_GPDI208	8-bits	Base + 0x08D0
GPIO data input register 209 (GPIO[209])	SIU_GPDI209	8-bits	Base + 0x08D1
GPIO data input register 210 (GPIO[210])	SIU_GPDI210	8-bits	Base + 0x08D2
GPIO data input register 211 (GPIO[211])	SIU_GPDI211	8-bits	Base + 0x08D3
GPIO data input register 212 (GPIO[212])	SIU_GPDI212	8-bits	Base + 0x08D4
GPIO data input register 213 (GPIO[213])	SIU_GPDI213	8-bits	Base + 0x08D5
Reserved	—	—	Base + (0x08D8–0x08FF)
eQADC trigger input select register	SIU_ETISR	32-bits	Base + 0x0900
External IRQ input select register	SIU_EIISR	32-bits	Base + 0x0904
DSPI input select register	SIU_DISR	32-bits	Base + 0x0908
Reserved	—	—	Base + (0x090C–0x097C)
Chip configuration register	SIU_CCR	32-bits	Base + 0x0980
External clock control register	SIU_ECCR	32-bits	Base + 0x0984
Compare A high register	SIU_CARH	32-bits	Base + 0x0988
Compare A low register	SIU_CARL	32-bits	Base + 0x098C
Compare B high register	SIU_CBRH	32-bits	Base + 0x0990
Compare B low register	SIU_CBRL	32-bits	Base + 0x0994
Reserved	—	—	(Base + 0x0998)–0xC3F9_FFFF
Chapter 16, "Enhanced Modular Input/Output Subsystem (eMIOS)"			0xC3FA_0000
Module configuration register	EMIOS_MCR	32-bit	Base + 0x0000
Global flag register	EMIOS_GFR	32-bit	Base + 0x0004
Output update disable register	EMIOS_OUDR	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C–0x001F)
Unified channel n, where n = 0–23	UC base addresses (UCn)	256-bit	Base + (0x0020 x (n+1))
Channel A data register n	EMIOS_CADRn	32-bit	UC*n* Base + 0x0000
Channel B data register n	EMIOS_CBDRn	32-bit	UC*n* Base + 0x0004
Channel counter register n	EMIOS_CCNTRn	32-bit	UC*n* Base + 0x0008
Channel control register n	EMIOS_CCRn	32-bit	UC*n* Base + 0x000C
Channel status register n	EMIOS_CSRn	32-bit	UC*n* Base + 0x0010
Reserved	—	—	UC*n* Base + (0x0014–0x001F)

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Chapter 17, "Enhanced Time Processing Unit (eTPU)"			0xC3FC_0000
eTPU module configuration register	ETPU_MCR	32-bit	Base + 0x0000
eTPU coherent dual-parameter controller register	ETPU_CDCR	32-bit	Base + 0x0004
Reserved	—	—	Base + (0x0008–0x000B)
eTPU miscellaneous compare register	ETPU_MISCCMPR	32-bit	Base + 0x000C
eTPU SCM off-range data register	ETPU_SCMOFFDATAR	32-bit	Base + 0x0010
eTPU A engine configuration register	ETPU_ECR_A	32-bit	Base + 0x0014
Reserved	—	—	Base + (0x0018–0x001F)
eTPU A time base configuration register	ETPU_TBCR_A	32-bit	Base + 0x0020
eTPU A time base 1	ETPU_TB1R_A	32-bit	Base + 0x0024
eTPU A time base 2	ETPU_TB2R_A	32-bit	Base + 0x0028
eTPU A STAC bus interface configuration register	ETPU_REDCR_A	32-bit	Base + 0x002C
Reserved	—	—	Base + (0x0030–0x01FF)
eTPU A channel interrupt status register	ETPU_CISR_A	32-bit	Base + 0x0200
Reserved	—	—	Base + (0x0204–0x020F)
eTPU A channel data transfer request status register	ETPU_CDTRSR_A	32-bit	Base + 0x0210
Reserved	—	—	Base + (0x0214–0x021F)
eTPU A channel interrupt overflow status register	ETPU_CIOSR_A	32-bit	Base + 0x0220
Reserved	—	—	Base + (0x0224–0x022F)
eTPU A channel data transfer request overflow status register	ETPU_CDTROSR_A	32-bit	Base + 0x0230
Reserved	—	—	Base + (0x0234–0x023F)
eTPU A channel interrupt enable register	ETPU_CIER_A	32-bit	Base + 0x0240
Reserved	—	—	Base + (0x0244–0x024F)
eTPU A channel data transfer request enable register	ETPU_CDTRER_A	32-bit	Base + 0x0250
Reserved	—	—	Base + (0x0254–0x027F)
eTPU A channel pending service status register	ETPU_CPSSR_A	32-bit	Base + 0x0280
Reserved	—	—	Base + (0x0284–0x028F)
eTPU A channel service status register	ETPU_CSSR_A	32-bit	Base + 0x0290
Reserved	—	—	Base + (0x0294–0x03FF)
eTPU A channel 0 configuration register	ETPU_C0CR_A	32-bit	Base + 0x0400

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
eTPU A channel 0 status and control register	ETPU_C0SCR_A	32-bit	Base + 0x0404
eTPU A channel 0 host service request register	ETPU_C0HSRR_A	32-bit	Base + 0x0408
Reserved	—	—	Base + (0x040C-0x040F)
eTPU A channel 1 configuration register	ETPU_C1CR_A	32-bit	Base + 0x0410
eTPU A channel 1 status and control register	ETPU_C1SCR_A	32-bit	Base + 0x0414
eTPU A channel 1 host service request register	ETPU_C1HSRR_A	32-bit	Base + 0x0418
Reserved	—	—	Base + (0x041C–0x041F)
eTPU A channel 2 configuration register	ETPU_C2CR_A	32-bit	Base + 0x0420
eTPU A channel 2 status and control register	ETPU_C2SCR_A	32-bit	Base + 0x0424
eTPU A channel 2 host service request register	ETPU_C2HSRR_A	32-bit	Base + 0x0428
Reserved	—	—	Base + (0x042C–0x042F)
eTPU A channel 3 configuration register	ETPU_C3CR_A	32-bit	Base + 0x0430
eTPU A channel 3 status and control register	ETPU_C3SCR_A	32-bit	Base + 0x0434
eTPU A channel 3 host service request register	ETPU_C3HSRR_A	32-bit	Base + 0x0438
Reserved	—	—	Base + (0x043C–0x043F)
eTPU A channel 4 configuration register	ETPU_C4CR_A	32-bit	Base + 0x0440
eTPU A channel 4 status and control register	ETPU_C4SCR_A	32-bit	Base + 0x0444
eTPU A channel 4 host service request register	ETPU_C4HSRR_A	32-bit	Base + 0x0448
Reserved	—	—	Base + (0x044C–0x044F)
eTPU A channel 5 configuration register	ETPU_C5CR_A	32-bit	Base + 0x0450
eTPU A channel 5 status and control register	ETPU_C5SCR_A	32-bit	Base + 0x0454
eTPU A channel 5 host service request register	ETPU_C5HSRR_A	32-bit	Base + 0x0458
Reserved	—	—	Base + (0x045C–0x045F)
eTPU A channel 6 configuration register	ETPU_C6CR_A	32-bit	Base + 0x0460
eTPU A channel 6 status and control register	ETPU_C6SCR_A	32-bit	Base + 0x0464
eTPU A channel 6 host service request register	ETPU_C6HSRR_A	32-bit	Base + 0x0468
Reserved	—	—	Base + (0x046C–0x046F)
eTPU A channel 7 configuration register	ETPU_C7CR_A	32-bit	Base + 0x0470
eTPU A channel 7 status and control register	ETPU_C7SCR_A	32-bit	Base + 0x0474
eTPU A channel 7 host service request register	ETPU_C7HSRR_A	32-bit	Base + 0x0478
Reserved	—	—	Base + (0x047C–0x047F)
eTPU A channel 8 configuration register	ETPU_C8CR_A	32-bit	Base + 0x0480

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
eTPU A channel 8 status and control register	ETPU_C8SCR_A	32-bit	Base + 0x0484
eTPU A channel 8 host service request register	ETPU_C8HSRR_A	32-bit	Base + 0x0488
Reserved	—	—	Base + (0x048C–0x048F)
eTPU A channel 9 configuration register	ETPU_C9CR_A	32-bit	Base + 0x0490
eTPU A channel 9 status and control register	ETPU_C9SCR_A	32-bit	Base + 0x0494
eTPU A channel 9 host service request register	ETPU_C9HSRR_A	32-bit	Base + 0x0498
Reserved	—	—	Base + (0x049C–0x049F)
eTPU A channel 10 configuration register	ETPU_C10CR_A	32-bit	Base + 0x04A0
eTPU A channel 10 status and control register	ETPU_C10SCR_A	32-bit	Base + 0x04A4
eTPU A channel 10 host service request register	ETPU_C10HSRR_A	32-bit	Base + 0x04A8
Reserved	—	—	Base + (0x04AC–0x04AF)
eTPU A channel 11 configuration register	ETPU_C11CR_A	32-bit	Base + 0x04B0
eTPU A channel 11 status and control register	ETPU_C11SCR_A	32-bit	Base + 0x04B4
eTPU A channel 11 host service request register	ETPU_C11HSRR_A	32-bit	Base + 0x04B8
Reserved	—	—	Base + (0x04BC–0x04BF)
eTPU A channel 12 configuration register	ETPU_C12CR_A	32-bit	Base + 0x04C0
eTPU A channel 12 status and control register	ETPU_C12SCR_A	32-bit	Base + 0x04C4
eTPU A channel 12 host service request register	ETPU_C12HSRR_A	32-bit	Base + 0x04C8
Reserved	—	—	Base + (0x04CC–0x04CF)
eTPU A channel 13 configuration register	ETPU_C13CR_A	32-bit	Base + 0x04D0
eTPU A channel 13 status and control register	ETPU_C13SCR_A	32-bit	Base + 0x04D4
eTPU A channel 13 host service request register	ETPU_C13HSRR_A	32-bit	Base + 0x04D8
Reserved	—	—	Base + (0x04DC–0x04DF)
eTPU A channel 14 configuration register	ETPU_C14CR_A	32-bit	Base + 0x04E0
eTPU A channel 14 status and control register	ETPU_C14SCR_A	32-bit	Base + 0x04E4
eTPU A channel 14 host service request register	ETPU_C14HSRR_A	32-bit	Base + 0x04E8
Reserved	—	—	Base + (0x04EC–0x04EF)
eTPU A channel 15 configuration register	ETPU_C15CR_A	32-bit	Base + 0x04F0
eTPU A channel 15 status and control register	ETPU_C15SCR_A	32-bit	Base + 0x04F4
eTPU A channel 15 host service request register	ETPU_C15HSRR_A	32-bit	Base + 0x04F8
Reserved	—	—	Base + (0x04FC–0x04FF)
eTPU A channel 16 configuration register	ETPU_C16CR_A	32-bit	Base + 0x0500

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
eTPU A channel 16 status and control register	ETPU_C16SCR_A	32-bit	Base + 0x0504
eTPU A channel 16 host service request register	ETPU_C16HSRR_A	32-bit	Base + 0x0508
Reserved	—	—	Base + (0x050C–0x050F)
eTPU A channel 17 configuration register	ETPU_C17CR_A	32-bit	Base + 0x0510
eTPU A channel 17 status and control register	ETPU_C17SCR_A	32-bit	Base + 0x0514
eTPU A channel 17 host service request register	ETPU_C17HSRR_A	32-bit	Base + 0x0518
Reserved	—	—	Base + (0x051C–0x051F)
eTPU A channel 18 configuration register	ETPU_C18CR_A	32-bit	Base + 0x0520
eTPU A channel 18 status and control register	ETPU_C18SCR_A	32-bit	Base + 0x0524
eTPU A channel 18 host service request register	ETPU_C18HSRR_A	32-bit	Base + 0x0528
Reserved	—	—	Base + (0x052C–0x052F)
eTPU A channel 19 configuration register	ETPU_C19CR_A	32-bit	Base + 0x0530
eTPU A channel 19 status and control register	ETPU_C19SCR_A	32-bit	Base + 0x0534
eTPU A channel 19 host service request register	ETPU_C19HSRR_A	32-bit	Base + 0x0538
Reserved	—	—	Base + (0x053C–0x053F)
eTPU A channel 20 configuration register	ETPU_C20CR_A	32-bit	Base + 0x0540
eTPU A channel 20 status and control register	ETPU_C20SCR_A	32-bit	Base + 0x0544
eTPU A channel 20 host service request register	ETPU_C20HSRR_A	32-bit	Base + 0x0548
Reserved	—	—	Base + (0x054C–0x054F)
eTPU A channel 21 configuration register	ETPU_C21CR_A	32-bit	Base + 0x0550
eTPU A channel 21 status and control register	ETPU_C21SCR_A	32-bit	Base + 0x0554
eTPU A channel 21 host service request register	ETPU_C21HSRR_A	32-bit	Base + 0x0558
Reserved	—	—	Base + (0x055C–0x055F)
eTPU A channel 22 configuration register	ETPU_C22CR_A	32-bit	Base + 0x0560
eTPU A channel 22 status and control register	ETPU_C22SCR_A	32-bit	Base + 0x0564
eTPU A channel 22 host service request register	ETPU_C22HSRR_A	32-bit	Base + 0x0568
Reserved	—	—	Base + (0x056C–0x056F)
eTPU A channel 23 configuration register	ETPU_C23CR_A	32-bit	Base + 0x0570
eTPU A channel 23 status and control register	ETPU_C23CR_A	32-bit	Base + 0x0574
eTPU A channel 23 host service request register	ETPU_C23HSRR_A	32-bit	Base + 0x0578
Reserved	—	—	Base + (0x057C–0x057F)
eTPU A channel 24 configuration register	ETPU_C24CR_A	32-bit	Base + 0x0580

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
eTPU A channel 24 status and control register	ETPU_C24SCR_A	32-bit	Base + 0x0584
eTPU A channel 24 host service request register	ETPU_C24HSRR_A	32-bit	Base + 0x0588
Reserved	—	—	Base + (0x058C–0x058F)
eTPU A channel 25 configuration register	ETPU_C25CR_A	32-bit	Base + 0x0590
eTPU A channel 25 status and control register	ETPU_C25SCR_A	32-bit	Base + 0x0594
eTPU A channel 25 host service request register	ETPU_C25HSRR_A	32-bit	Base + 0x0598
Reserved	—	—	Base + (0x059C–0x059F)
eTPU A channel 26 configuration register	ETPU_C26CR_A	32-bit	Base + 0x05A0
eTPU A channel 26 status and control register	ETPU_C26SCR_A	32-bit	Base + 0x05A4
eTPU A channel 26 host service request register	ETPU_C26HSRR_A	32-bit	Base + 0x05A8
Reserved	—	—	Base + (0x05AC–0x05AF)
eTPU A channel 27 configuration register	ETPU_C27CR_A	32-bit	Base + 0x05B0
eTPU A channel 27 status and control register	ETPU_C27SCR_A	32-bit	Base + 0x05B4
eTPU A channel 27 host service request register	ETPU_C27HSRR_A	32-bit	Base + 0x05B8
Reserved	—	—	Base + (0x05BC–0x05BF)
eTPU A channel 28 configuration register	ETPU_C28CR_A	32-bit	Base + 0x05C0
eTPU A channel 28 status and control register	ETPU_C28SCR_A	32-bit	Base + 0x05C4
eTPU A channel 28 host service request register	ETPU_C28HSRR_A	32-bit	Base + 0x05C8
Reserved	—	—	Base + (0x05CC–0x05CF)
eTPU A channel 29 configuration register	ETPU_C29CR_A	32-bit	Base + 0x05D0
eTPU A channel 29 status and control register	ETPU_C29SCR_A	32-bit	Base + 0x05D4
eTPU A channel 29 host service request register	ETPU_C29HSRR_A	32-bit	Base + 0x05D8
Reserved	—	—	Base + (0x05DC–0x05DF)
eTPU A channel 30 configuration register	ETPU_C30CR_A	32-bit	Base + 0x05E0
eTPU A channel 30 status and control register	ETPU_C30SCR_A	32-bit	Base + 0x05E4
eTPU A channel 30 host service request register	ETPU_C30HSRR_A	32-bit	Base + 0x05E8
Reserved	—	—	Base + (0x05EC–0x05EF)
eTPU A channel 31 configuration register	ETPU_C31CR_A	32-bit	Base + 0x05F0
eTPU A channel 31 status and control register	ETPU_C31SCR_A	32-bit	Base + 0x05F4
eTPU A channel 31 host service request register	ETPU_C31HSRR_A	32-bit	Base + 0x05F8
Reserved	—	—	Base + (0x05FC–0x07FF)
Reserved	—	—	Base + (0x09FC–0x7FFF)

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Shared data memory (parameter RAM)	SDM	2.5 KB	Base + (0x8000–0x8BFF)
Reserved	—	—	Base + (0x8C00–0xBFFF)
SDM PSE mirror		2.5 KB	Base + (0xC000–0xCBFF)
Reserved	—	—	Base + (0xCC00–0xFFFF)
Shared code memory	SCM	12 KB	Base + (0x0001_0000–0x0001_2FFF)
Reserved	—	—	Base + (0x0001_4000–0xFFEF_FFFF)
Chapter 5, "Peripheral Bridge"			0xFFF0_0000
Peripheral bridge B master privilege control register	PBRIDGEB_MPCR	32-bit	Base + 0x0000
Reserved	—	—	(Base + 0x0004)–0xFFF0_3FFF
Chapter 7, "Crossbar Switch (XBAR)"			0xFFF0_4000
Master priority register 0	XBAR_MPR0	32-bit	Base + 0x0000
Reserved	—	—	Base + (0x0004–0x000F)
Slave general purpose control register 0	XBAR_SGPCR0	32-bit	Base + 0x0010
Reserved	—	—	Base + (0x0014–0x00FF)
Master priority register 1	XBAR_MPR1	32-bit	Base + 0x0100
Reserved	—	—	Base + (0x0104–0x010F)
Slave general purpose control register 1	XBAR_SGPCR1	32-bit	Base + 0x0110
Reserved	—	—	Base + (0x0114–0x02FF)
Master priority register 3	XBAR_MPR3	32-bit	Base + 0x0300
Reserved	—	—	Base + (0x0304–0x030F)
Slave general purpose control register 3	XBAR_SGPCR3	32-bit	Base + 0x0310
Reserved	—	—	Base + (0x0314–0x05FF)
Master priority register 6	XBAR_MPR6	32-bit	Base + 0x0600
Reserved	—	—	Base + (0x0604–0x060F)
Slave general purpose control register 6	XBAR_SGPCR6	32-bit	Base + 0x0610
Reserved	—	—	Base + (0x0614–0x06FF)
Master priority register 7	XBAR_MPR7	32-bit	Base + 0x0700
Reserved	—	—	Base + (0x0704–0x070F)
Slave general purpose control register 7	XBAR_SGPCR7	32-bit	Base + 0x0710
Reserved	—	—	(Base + 0x0714)–0xFFF4_3FFF

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Chapter 8, "Error Correction Status Module (ECSM)"			0xFFF4_0000
Reserved	—	—	Base + (0x0000–0x0015)
Software watchdog timer control register	ECSM_SWTCR[1]	16-bit	Base + 0x0016
Reserved	—	—	Base + (0x0018–0x001A)
Software watchdog timer service register	ECSM_SWTSR [1]	8-bit	Base + 0x001B
Reserved	—	—	Base + (0x001C–0x001E)
Software watchdog timer interrupt register	ECSM_SWTIR [1]	8-bit	Base + 0x001F
Reserved	—		Base + (0x0020–0x0042)
ECC configuration register	ECSM_ECR	8-bit	Base + 0x0043
Reserved	—	—	Base + (0x0044–0x0046)
ECC status register	ECSM_ESR	8-bit	Base + 0x0047
Reserved	—	—	Base + (0x0048–0x0049)
ECC error generation register	ECSM_EEGR	16-bit	Base + 0x004A
Reserved	—	—	Base + (0x004C–0x004F)
Flash ECC address register	ECSM_FEAR	32-bit	Base + 0x0050
Reserved	—	—	Base + (0x0054–0x0055)
Flash ECC master number register	ECSM_FEMR	8-bit	Base + 0x0056
Flash ECC attributes register	ECSM_FEAT	8-bit	Base + 0x0057
Flash ECC data register high	ECSM_FEDRH	32-bit	Base + 0x0058
Flash ECC data register low	ECSM_FEDRL	32-bit	Base + 0x005C
RAM ECC address register	ECSM_REAR	32-bit	Base + 0x0060
Reserved	—	—	Base + (0x0064–0x0065)
RAM ECC master number register	ECSM_REMR	8-bit	Base + 0x0066
RAM ECC attributes register	ECSM_REAT	8-bit	Base + 0x0067
RAM ECC data register high	ECSM_REDRH	32-bit	Base + 0x0068
RAM ECC data register low	ECSM_REDRL	32-bit	Base + 0x006C
Reserved	—	—	(Base + 0x0070)–0xFFF4_3FFF
Chapter 9, "Enhanced Direct Memory Access (eDMA)"			0xFFF4_4000
Control register	EDMA_CR	32-bit	Base + 0x0000
Error status register	EDMA_ESR	32-bit	Base + 0x0004
Reserved	—	—	Base + 0x0008
Enable request register low	EDMA_ERQRL	32-bit	Base + 0x000C

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Reserved	—	—	Base + 0x0010
Enable error interrupt register low	EDMA_EEIRL	32-bit	Base + 0x0014
Set enable request register	EDMA_SERQR	8-bit	Base + 0x0018
Clear enable request register	EDMA_CERQR	8-bit	Base + 0x0019
Set enable error interrupt register	EDMA_SEEIR	8-bit	Base + 0x001A
Clear enable error interrupt request register	EDMA_CEEIR	8-bit	Base + 0x001B
Clear interrupt request register	EDMA_CIRQR	8-bit	Base + 0x001C
Clear error register	EDMA_CER	8-bit	Base + 0x001D
Set START bit register	EDMA_SSBR	8-bit	Base + 0x001E
Clear DONE status bit register	EDMA_CDSBR	8-bit	Base + 0x001F
Reserved	—	—	Base + 0x0020
Interrupt request register low	EDMA_IRQRL	32-bit	Base + 0x0024
Reserved	—	—	Base + 0x0028
Error register low	EDMA_ERL	32-bit	Base + 0x002C
Reserved	—	—	Base + (0x0030–0x00FF)
Channel priority register 0	EDMA_CPR0	8-bit	Base + 0x0100
Channel priority register 1	EDMA_CPR1	8-bit	Base + 0x0101
Channel priority register 2	EDMA_CPR2	8-bit	Base + 0x0102
Channel priority register 3	EDMA_CPR3	8-bit	Base + 0x0103
Channel priority register 4	EDMA_CPR4	8-bit	Base + 0x0104
Channel priority register 5	EDMA_CPR5	8-bit	Base + 0x0105
Channel priority register 6	EDMA_CPR6	8-bit	Base + 0x0106
Channel priority register 7	EDMA_CPR7	8-bit	Base + 0x0107
Channel priority register 8	EDMA_CPR8	8-bit	Base + 0x0108
Channel priority register 9	EDMA_CPR9	8-bit	Base + 0x0109
Channel priority register 10	EDMA_CPR10	8-bit	Base + 0x010A
Channel priority register 11	EDMA_CPR11	8-bit	Base + 0x010B
Channel priority register 12	EDMA_CPR12	8-bit	Base + 0x010C
Channel priority register 13	EDMA_CPR13	8-bit	Base + 0x010D
Channel priority register 14	EDMA_CPR14	8-bit	Base + 0x010E
Channel priority register 15	EDMA_CPR15	8-bit	Base + 0x010F
Channel priority register 16	EDMA_CPR16	8-bit	Base + 0x0110

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Channel priority register 17	EDMA_CPR17	8-bit	Base + 0x0111
Channel priority register 18	EDMA_CPR18	8-bit	Base + 0x0112
Channel priority register 19	EDMA_CPR19	8-bit	Base + 0x0113
Channel priority register 20	EDMA_CPR20	8-bit	Base + 0x0114
Channel priority register 21	EDMA_CPR21	8-bit	Base + 0x0115
Channel priority register 22	EDMA_CPR22	8-bit	Base + 0x0116
Channel priority register 23	EDMA_CPR23	8-bit	Base + 0x0117
Channel priority register 24	EDMA_CPR24	8-bit	Base + 0x0118
Channel priority register 25	EDMA_CPR25	8-bit	Base + 0x0119
Channel priority register 26	EDMA_CPR26	8-bit	Base + 0x011A
Channel priority register 27	EDMA_CPR27	8-bit	Base + 0x011B
Channel priority register 28	EDMA_CPR28	8-bit	Base + 0x011C
Channel priority register 29	EDMA_CPR29	8-bit	Base + 0x011D
Channel priority register 30	EDMA_CPR30	8-bit	Base + 0x011E
Channel priority register 31	EDMA_CPR31	8-bit	Base + 0x011F
Reserved	—	—	Base + (0x0120–0x0FFF)
Transfer control descriptor register 0	TCD0	256-bit	Base + 0x1000
Transfer control descriptor register 1	TCD1	256-bit	Base + 0x1020
Transfer control descriptor register 2	TCD2	256-bit	Base + 0x1040
Transfer control descriptor register 3	TCD3	256-bit	Base + 0x1060
Transfer control descriptor register 4	TCD4	256-bit	Base + 0x1080
Transfer control descriptor register 5	TCD5	256-bit	Base + 0x10A0
Transfer control descriptor register 6	TCD6	256-bit	Base + 0x10C0
Transfer control descriptor register 7	TCD7	256-bit	Base + 0x10E0
Transfer control descriptor register 8	TCD8	256-bit	Base + 0x1100
Transfer control descriptor register 9	TCD9	256-bit	Base + 0x1120
Transfer control descriptor register 10	TCD10	256-bit	Base + 0x1140
Transfer control descriptor register 11	TCD11	256-bit	Base + 0x1160
Transfer control descriptor register 12	TCD12	256-bit	Base + 0x1180
Transfer control descriptor register 13	TCD13	256-bit	Base + 0x11A0
Transfer control descriptor register 14	TCD14	256-bit	Base + 0x11C0
Transfer control descriptor register 15	TCD15	256-bit	Base + 0x11E0

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Transfer control descriptor register 16	TCD16	256-bit	Base + 0x1200
Transfer control descriptor register 17	TCD17	256-bit	Base + 0x1220
Transfer control descriptor register 18	TCD18	256-bit	Base + 0x1240
Transfer control descriptor register 19	TCD19	256-bit	Base + 0x1260
Transfer control descriptor register 20	TCD20	256-bit	Base + 0x1280
Transfer control descriptor register 21	TCD21	256-bit	Base + 0x12A0
Transfer control descriptor register 22	TCD22	256-bit	Base + 0x12C0
Transfer control descriptor register 23	TCD23	256-bit	Base + 0x12E0
Transfer control descriptor register 24	TCD24	256-bit	Base + 0x1300
Transfer control descriptor register 25	TCD25	256-bit	Base + 0x1320
Transfer control descriptor register 26	TCD26	256-bit	Base + 0x1340
Transfer control descriptor register 27	TCD27	256-bit	Base + 0x1360
Transfer control descriptor register 28	TCD28	256-bit	Base + 0x1380
Transfer control descriptor register 29	TCD29	256-bit	Base + 0x13A0
Transfer control descriptor register 30	TCD30	256-bit	Base + 0x13C0
Transfer control descriptor register 31	TCD31	256-bit	Base + 0x13E0
Reserved	—	—	(Base + 0x1400)–0xFFF4_7FFF
Chapter 10, "Interrupt Controller (INTC)"			0xFFF4_8000
Module configuration register	INTC_MCR	32-bit	Base + 0x0000
Reserved	—	—	Base + (0x0004–0x0007)
Current priority register	INTC_CPR	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C–0x000F)
Interrupt acknowledge register	INTC_IACKR	32-bit	Base + 0x0010
Reserved	—	—	Base + (0x0014–0x0017)
End of interrupt register	INTC_EOIR	32-bit	Base + 0x0018
Reserved	—	—	Base + (0x001C–0x001F)
Software set/clear interrupt register 0	INTC_SSCIR0	8-bit	Base + 0x0020
Software set/clear interrupt register 1	INTC_SSCIR1	8-bit	Base + 0x0021
Software set/clear interrupt register 2	INTC_SSCIR2	8-bit	Base + 0x0022
Software set/clear interrupt register 3	INTC_SSCIR3	8-bit	Base + 0x0023
Software set/clear interrupt register 4	INTC_SSCIR4	8-bit	Base + 0x0024
Software set/clear interrupt register 5	INTC_SSCIR5	8-bit	Base + 0x0025

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Software set/clear interrupt register 6	INTC_SSCIR6	8-bit	Base + 0x0026
Software set/clear interrupt register 7	INTC_SSCIR7	8-bit	Base + 0x0027
Reserved	—	—	Base + (0x0028–0x003F)
Priority select register 0	INTC_PSR0	8-bit	Base + 0x0040
Priority select register 1	INTC_PSR1	8-bit	Base + 0x0041
Priority select register 2	INTC_PSR2	8-bit	Base + 0x0042
Priority select register 3	INTC_PSR3	8-bit	Base + 0x0043
Priority select register 4	INTC_PSR4	8-bit	Base + 0x0044
Priority select register 5	INTC_PSR5	8-bit	Base + 0x0045
Priority select register 6	INTC_PSR6	8-bit	Base + 0x0046
Priority select register 7	INTC_PSR7	8-bit	Base + 0x0047
Priority select register 8	INTC_PSR8	8-bit	Base + 0x0048
Priority select register 9	INTC_PSR9	8-bit	Base + 0x0049
Priority select register 10	INTC_PSR10	8-bit	Base + 0x004A
Priority select register 11	INTC_PSR11	8-bit	Base + 0x004B
Priority select register 12	INTC_PSR12	8-bit	Base + 0x004C
Priority select register 13	INTC_PSR13	8-bit	Base + 0x004D
Priority select register 14	INTC_PSR14	8-bit	Base + 0x004E
Priority select register 15	INTC_PSR15	8-bit	Base + 0x004F
Priority select register 16	INTC_PSR16	8-bit	Base + 0x0050
Priority select register 17	INTC_PSR17	8-bit	Base + 0x0051
Priority select register 18	INTC_PSR18	8-bit	Base + 0x0052
Priority select register 19	INTC_PSR19	8-bit	Base + 0x0053
Priority select register 20	INTC_PSR20	8-bit	Base + 0x0054
Priority select register 21	INTC_PSR21	8-bit	Base + 0x0055
Priority select register 22	INTC_PSR22	8-bit	Base + 0x0056
Priority select register 23	INTC_PSR23	8-bit	Base + 0x0057
Priority select register 24	INTC_PSR24	8-bit	Base + 0x0058
Priority select register 25	INTC_PSR25	8-bit	Base + 0x0059
Priority select register 26	INTC_PSR26	8-bit	Base + 0x005A
Priority select register 27	INTC_PSR27	8-bit	Base + 0x005B
Priority select register 28	INTC_PSR28	8-bit	Base + 0x005C

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Priority select register 29	INTC_PSR29	8-bit	Base + 0x005D
Priority select register 30	INTC_PSR30	8-bit	Base + 0x005E
Priority select register 31	INTC_PSR31	8-bit	Base + 0x005F
Priority select register 32	INTC_PSR32	8-bit	Base + 0x0060
Priority select register 33	INTC_PSR33	8-bit	Base + 0x0061
Priority select register 34	INTC_PSR34	8-bit	Base + 0x0062
Priority select register 35	INTC_PSR35	8-bit	Base + 0x0063
Priority select register 36	INTC_PSR36	8-bit	Base + 0x0064
Priority select register 37	INTC_PSR37	8-bit	Base + 0x0065
Priority select register 38	INTC_PSR38	8-bit	Base + 0x0066
Priority select register 39	INTC_PSR39	8-bit	Base + 0x0067
Priority select register 40	INTC_PSR40	8-bit	Base + 0x0068
Priority select register 41	INTC_PSR41	8-bit	Base + 0x0069
Priority select register 42	INTC_PSR42	8-bit	Base + 0x006A
Priority select register 43	INTC_PSR43	8-bit	Base + 0x006B
Priority select register 44	INTC_PSR44	8-bit	Base + 0x006C
Priority select register 45	INTC_PSR45	8-bit	Base + 0x006D
Priority select register 46	INTC_PSR46	8-bit	Base + 0x006E
Priority select register 47	INTC_PSR47	8-bit	Base + 0x006F
Priority select register 48	INTC_PSR48	8-bit	Base + 0x0070
Priority select register 49	INTC_PSR49	8-bit	Base + 0x0071
Priority select register 50	INTC_PSR50	8-bit	Base + 0x0072
Priority select register 51	INTC_PSR51	8-bit	Base + 0x0073
Priority select register 52	INTC_PSR52	8-bit	Base + 0x0074
Priority select register 53	INTC_PSR53	8-bit	Base + 0x0075
Priority select register 54	INTC_PSR54	8-bit	Base + 0x0076
Priority select register 55	INTC_PSR55	8-bit	Base + 0x0077
Priority select register 56	INTC_PSR56	8-bit	Base + 0x0078
Priority select register 57	INTC_PSR57	8-bit	Base + 0x0079
Priority select register 58	INTC_PSR58	8-bit	Base + 0x007A
Priority select register 59	INTC_PSR59	8-bit	Base + 0x007B
Priority select register 60	INTC_PSR60	8-bit	Base + 0x007C

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Priority select register 61	INTC_PSR61	8-bit	Base + 0x007D
Priority select register 62	INTC_PSR62	8-bit	Base + 0x007E
Priority select register 63	INTC_PSR63	8-bit	Base + 0x007F
Priority select register 64	INTC_PSR64	8-bit	Base + 0x0080
Priority select register 65	INTC_PSR65	8-bit	Base + 0x0081
Priority select register 66	INTC_PSR66	8-bit	Base + 0x0082
Priority select register 67	INTC_PSR67	8-bit	Base + 0x0083
Priority select register 68	INTC_PSR68	8-bit	Base + 0x0084
Priority select register 69	INTC_PSR69	8-bit	Base + 0x0085
Priority select register 70	INTC_PSR70	8-bit	Base + 0x0086
Priority select register 71	INTC_PSR71	8-bit	Base + 0x0087
Priority select register 72	INTC_PSR72	8-bit	Base + 0x0088
Priority select register 73	INTC_PSR73	8-bit	Base + 0x0089
Priority select register 74	INTC_PSR74	8-bit	Base + 0x008A
Priority select register 75	INTC_PSR75	8-bit	Base + 0x008B
Priority select register 76	INTC_PSR76	8-bit	Base + 0x008C
Priority select register 77	INTC_PSR77	8-bit	Base + 0x008D
Priority select register 78	INTC_PSR78	8-bit	Base + 0x008E
Priority select register 79	INTC_PSR79	8-bit	Base + 0x008F
Priority select register 80	INTC_PSR80	8-bit	Base + 0x0090
Priority select register 81	INTC_PSR81	8-bit	Base + 0x0091
Priority select register 82	INTC_PSR82	8-bit	Base + 0x0092
Priority select register 83	INTC_PSR83	8-bit	Base + 0x0093
Priority select register 84	INTC_PSR84	8-bit	Base + 0x0094
Priority select register 85	INTC_PSR85	8-bit	Base + 0x0095
Priority select register 86	INTC_PSR86	8-bit	Base + 0x0096
Priority select register 87	INTC_PSR87	8-bit	Base + 0x0097
Priority select register 88	INTC_PSR88	8-bit	Base + 0x0098
Priority select register 89	INTC_PSR89	8-bit	Base + 0x0099
Priority select register 90	INTC_PSR90	8-bit	Base + 0x009A
Priority select register 91	INTC_PSR91	8-bit	Base + 0x009B
Priority select register 92	INTC_PSR92	8-bit	Base + 0x009C

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Priority select register 93	INTC_PSR93	8-bit	Base + 0x009D
Priority select register 94	INTC_PSR94	8-bit	Base + 0x009E
Priority select register 95	INTC_PSR95	8-bit	Base + 0x009F
Priority select register 96	INTC_PSR96	8-bit	Base + 0x00A0
Priority select register 97	INTC_PSR97	8-bit	Base + 0x00A1
Priority select register 98	INTC_PSR98	8-bit	Base + 0x00A2
Priority select register 99	INTC_PSR99	8-bit	Base + 0x00A3
Priority select register 100	INTC_PSR100	8-bit	Base + 0x00A4
Priority select register 101	INTC_PSR101	8-bit	Base + 0x00A5
Priority select register 102	INTC_PSR102	8-bit	Base + 0x00A6
Priority select register 103	INTC_PSR103	8-bit	Base + 0x00A7
Priority select register 104	INTC_PSR104	8-bit	Base + 0x00A8
Priority select register 105	INTC_PSR105	8-bit	Base + 0x00A9
Priority select register 106	INTC_PSR106	8-bit	Base + 0x00AA
Priority select register 107	INTC_PSR107	8-bit	Base + 0x00AB
Priority select register 108	INTC_PSR108	8-bit	Base + 0x00AC
Priority select register 109	INTC_PSR109	8-bit	Base + 0x00AD
Priority select register 110	INTC_PSR110	8-bit	Base + 0x00AE
Priority select register 111	INTC_PSR111	8-bit	Base + 0x00AF
Priority select register 112	INTC_PSR112	8-bit	Base + 0x00B0
Priority select register 113	INTC_PSR113	8-bit	Base + 0x00B1
Priority select register 114	INTC_PSR114	8-bit	Base + 0x00B2
Priority select register 115	INTC_PSR115	8-bit	Base + 0x00B3
Priority select register 116	INTC_PSR116	8-bit	Base + 0x00B4
Priority select register 117	INTC_PSR117	8-bit	Base + 0x00B5
Priority select register 118	INTC_PSR118	8-bit	Base + 0x00B6
Priority select register 119	INTC_PSR119	8-bit	Base + 0x00B7
Priority select register 120	INTC_PSR120	8-bit	Base + 0x00B8
Priority select register 121	INTC_PSR121	8-bit	Base + 0x00B9
Priority select register 122	INTC_PSR122	8-bit	Base + 0x00BA
Priority select register 123	INTC_PSR123	8-bit	Base + 0x00BB
Priority select register 124	INTC_PSR124	8-bit	Base + 0x00BC

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Priority select register 125	INTC_PSR125	8-bit	Base + 0x00BD
Priority select register 126	INTC_PSR126	8-bit	Base + 0x00BE
Priority select register 127	INTC_PSR127	8-bit	Base + 0x00BF
Priority select register 128	INTC_PSR128	8-bit	Base + 0x00C0
Priority select register 129	INTC_PSR129	8-bit	Base + 0x00C1
Priority select register 130	INTC_PSR130	8-bit	Base + 0x00C2
Priority select register 131	INTC_PSR131	8-bit	Base + 0x00C3
Priority select register 132	INTC_PSR132	8-bit	Base + 0x00C4
Priority select register 133	INTC_PSR133	8-bit	Base + 0x00C5
Priority select register 134	INTC_PSR134	8-bit	Base + 0x00C6
Priority select register 135	INTC_PSR135	8-bit	Base + 0x00C7
Priority select register 136	INTC_PSR136	8-bit	Base + 0x00C8
Priority select register 137	INTC_PSR137	8-bit	Base + 0x00C9
Priority select register 138	INTC_PSR138	8-bit	Base + 0x00CA
Priority select register 139	INTC_PSR139	8-bit	Base + 0x00CB
Priority select register 140	INTC_PSR140	8-bit	Base + 0x00CC
Priority select register 141	INTC_PSR141	8-bit	Base + 0x00CD
Priority select register 142	INTC_PSR142	8-bit	Base + 0x00CE
Priority select register 143	INTC_PSR143	8-bit	Base + 0x00CF
Priority select register 144	INTC_PSR144	8-bit	Base + 0x00D0
Priority select register 145	INTC_PSR145	8-bit	Base + 0x00D1
Priority select register 146	INTC_PSR146	8-bit	Base + 0x00D2
Priority select register 147 (reserved)	INTC_PSR147	8-bit	Base + 0x00D3
Priority select register 148 (reserved)	INTC_PSR148	8-bit	Base + 0x00D4
Priority select register 149	INTC_PSR149	8-bit	Base + 0x00D5
Priority select register 150 (reserved)	INTC_PSR150	8-bit	Base + 0x00D6
Priority select register 151 (reserved)	INTC_PSR151	8-bit	Base + 0x00D7
Priority select register 152	INTC_PSR152	8-bit	Base + 0x00D8
Priority select register 153	INTC_PSR153	8-bit	Base + 0x00D9
Priority select register 154	INTC_PSR154	8-bit	Base + 0x00DA
Priority select register 155	INTC_PSR155	8-bit	Base + 0x00DB
Priority select register 156	INTC_PSR156	8-bit	Base + 0x00DC

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Priority select register 157	INTC_PSR157	8-bit	Base + 0x00DD
Priority select register 158	INTC_PSR158	8-bit	Base + 0x00DE
Priority select register 159	INTC_PSR159	8-bit	Base + 0x00DF
Priority select register 160	INTC_PSR160	8-bit	Base + 0x00E0
Priority select register 161	INTC_PSR161	8-bit	Base + 0x00E1
Priority select register 162	INTC_PSR162	8-bit	Base + 0x00E2
Priority select register 163	INTC_PSR163	8-bit	Base + 0x00E3
Priority select register 164	INTC_PSR164	8-bit	Base + 0x00E4
Priority select register 165	INTC_PSR165	8-bit	Base + 0x00E5
Priority select register 166	INTC_PSR166	8-bit	Base + 0x00E6
Priority select register 167	INTC_PSR167	8-bit	Base + 0x00E7
Priority select register 168	INTC_PSR168	8-bit	Base + 0x00E8
Priority select register 169	INTC_PSR169	8-bit	Base + 0x00E9
Priority select register 170	INTC_PSR170	8-bit	Base + 0x00EA
Priority select register 171	INTC_PSR171	8-bit	Base + 0x00EB
Priority select register 172	INTC_PSR172	8-bit	Base + 0x00EC
Priority select register 173	INTC_PSR173	8-bit	Base + 0x00ED
Priority select register 174	INTC_PSR174	8-bit	Base + 0x00EE
Priority select register 175	INTC_PSR175	8-bit	Base + 0x00EF
Priority select register 176	INTC_PSR176	8-bit	Base + 0x00F0
Priority select register 177	INTC_PSR177	8-bit	Base + 0x00F1
Priority select register 178	INTC_PSR178	8-bit	Base + 0x00F2
Priority select register 179	INTC_PSR179	8-bit	Base + 0x00F3
Priority select register 180	INTC_PSR180	8-bit	Base + 0x00F4
Priority select register 181	INTC_PSR181	8-bit	Base + 0x00F5
Priority select register 182	INTC_PSR182	8-bit	Base + 0x00F6
Priority select register 183	INTC_PSR183	8-bit	Base + 0x00F7
Priority select register 184	INTC_PSR184	8-bit	Base + 0x00F8
Priority select register 185	INTC_PSR185	8-bit	Base + 0x00F9
Priority select register 186	INTC_PSR186	8-bit	Base + 0x00FA
Priority select register 187	INTC_PSR187	8-bit	Base + 0x00FB
Priority select register 188	INTC_PSR188	8-bit	Base + 0x00FC

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Priority select register 189	INTC_PSR189	8-bit	Base + 0x00FD
Priority select register 190	INTC_PSR190	8-bit	Base + 0x00FE
Priority select register 191	INTC_PSR191	8-bit	Base + 0x00FF
Priority select register 192	INTC_PSR192	8-bit	Base + 0x0100
Priority select register 193	INTC_PSR193	8-bit	Base + 0x0101
Priority select register 194 (reserved)	INTC_PSR194	8-bit	Base + 0x0102
Priority select register 195 (reserved)	INTC_PSR195	8-bit	Base + 0x0103
Priority select register 196 (reserved)	INTC_PSR196	8-bit	Base + 0x0104
Priority select register 197 (reserved)	INTC_PSR197	8-bit	Base + 0x0105
Priority select register 198 (reserved)	INTC_PSR198	8-bit	Base + 0x0106
Priority select register 199 (reserved)	INTC_PSR199	8-bit	Base + 0x0107
Priority select register 200 (reserved)	INTC_PSR200	8-bit	Base + 0x0108
Priority select register 201 (reserved)	INTC_PSR201	8-bit	Base + 0x0109
Priority select register 202	INTC_PSR202	8-bit	Base + 0x010A
Priority select register 203	INTC_PSR203	8-bit	Base + 0x010B
Priority select register 204	INTC_PSR204	8-bit	Base + 0x010C
Priority select register 205	INTC_PSR205	8-bit	Base + 0x010D
Priority select register 206	INTC_PSR206	8-bit	Base + 0x010E
Priority select register 207	INTC_PSR207	8-bit	Base + 0x010F
Priority select register 208	INTC_PSR208	8-bit	Base + 0x0110
Priority select register 209	INTC_PSR209	8-bit	Base + 0x0111
Priority select register 210 (reserved)	INTC_PSR210	8-bit	Base + 0x0112
Priority select register 211 (reserved)	INTC_PSR211	8-bit	Base + 0x0113
Chapter 18, "Enhanced Queued Analog-to-Digital Converter (eQADC)"			0xFFF8_0000
Module configuration register	EQADC_MCR	32-bit	Base + 0x0000
Reserved	—	—	Base + (0x0004–0x0007)
Null message send format register	EQADC_NMSFR	32-bit	Base + 0x0008
External trigger digital filter register	EQADC_ETDFR	32-bit	Base + 0x000C
CFIFO push register 0	EQADC_CFPR0	32-bit	Base + 0x0010
CFIFO push register 1	EQADC_CFPR1	32-bit	Base + 0x0014
CFIFO push register 2	EQADC_CFPR2	32-bit	Base + 0x0018
CFIFO push register 3	EQADC_CFPR3	32-bit	Base + 0x001C

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
CFIFO push register 4	EQADC_CFPR4	32-bit	Base + 0x0020
CFIFO push register 5	EQADC_CFPR5	32-bit	Base + 0x0024
Reserved	—	—	Base + (0x0028–0x002F)
Result FIFO pop register 0	EQADC_RFPR0	32-bit	Base + 0x0030
Result FIFO pop register 1	EQADC_RFPR1	32-bit	Base + 0x0034
Result FIFO pop register 2	EQADC_RFPR2	32-bit	Base + 0x0038
Result FIFO pop register 3	EQADC_RFPR3	32-bit	Base + 0x003C
Result FIFO pop register 4	EQADC_RFPR4	32-bit	Base + 0x0040
Result FIFO pop register 5	EQADC_RFPR5	32-bit	Base + 0x0044
Reserved	—	—	Base + (0x0048–0x004F)
CFIFO control register 0	EQADC_CFCR0	16-bit	Base + 0x0050
CFIFO control register 1	EQADC_CFCR1	16-bit	Base + 0x0052
CFIFO control register 2	EQADC_CFCR2	16-bit	Base + 0x0054
CFIFO control register 3	EQADC_CFCR3	16-bit	Base + 0x0056
CFIFO control register 4	EQADC_CFCR4	16-bit	Base + 0x0058
CFIFO control register 5	EQADC_CFCR5	16-bit	Base + 0x005A
Reserved	—	—	Base + (0x005C–0x005F)
Interrupt and DMA control register 0	EQADC_IDCR0	16-bit	Base + 0x0060
Interrupt and DMA control register 1	EQADC_IDCR1	16-bit	Base + 0x0062
Interrupt and DMA control register 2	EQADC_IDCR2	16-bit	Base + 0x0064
Interrupt and DMA control register 3	EQADC_IDCR3	16-bit	Base + 0x0066
Interrupt and DMA control register 4	EQADC_IDCR4	16-bit	Base + 0x0068
Interrupt and DMA control register 5	EQADC_IDCR5	16-bit	Base + 0x006A
Reserved	—	—	Base + (0x006C–0x006F)
FIFO and interrupt status register 0	EQADC_FISR0	32-bit	Base + 0x0070
FIFO and interrupt status register 1	EQADC_FISR1	32-bit	Base + 0x0074
FIFO and interrupt status register 2	EQADC_FISR2	32-bit	Base + 0x0078
FIFO and interrupt status register 3	EQADC_FISR3	32-bit	Base + 0x007C
FIFO and interrupt status register 4	EQADC_FISR4	32-bit	Base + 0x0080
FIFO and interrupt status register 5	EQADC_FISR5	32-bit	Base + 0x0084
Reserved	—	—	Base + (0x0088–0x008F)
CFIFO transfer counter register 0	EQADC_CFTCR0	16-bit	Base + 0x0090

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
CFIFO transfer counter register 1	EQADC_CFTCR1	16-bit	Base + 0x0092
CFIFO transfer counter register 2	EQADC_CFTCR2	16-bit	Base + 0x0094
CFIFO transfer counter register 3	EQADC_CFTCR3	16-bit	Base + 0x0096
CFIFO transfer counter register 4	EQADC_CFTCR4	16-bit	Base + 0x0098
CFIFO transfer counter register 5	EQADC_CFTCR5	16-bit	Base + 0x009A
Reserved	—	—	Base + (0x009C–0x009F)
CFIFO status snapshot register 0	EQADC_CFSSR0	32-bit	Base + 0x00A0
CFIFO status snapshot register 1	EQADC_CFSSR1	32-bit	Base + 0x00A4
CFIFO status snapshot register 2	EQADC_CFSSR2	32-bit	Base + 0x00A8
CFIFO status register	EQADC_CFSR	32-bit	Base + 0x00AC
Reserved	—	—	Base + (0x00B0–0x00B3)
SSI control register	EQADC_SSICR	32-bit	Base + 0x00B4
SSI receive data register	EQADC_SSIRDR	32-bit	Base + 0x00B8
Reserved	—	—	Base + (0x00BC–0x00FF)
CFIFO 0 register 0	EQADC_CF0R0	32-bit	Base + 0x0100
CFIFO 0 register 1	EQADC_CF0R1	32-bit	Base + 0x0104
CFIFO 0 register 2	EQADC_CF0R2	32-bit	Base + 0x0108
CFIFO 0 register 3	EQADC_CF0R3	32-bit	Base + 0x010C
Reserved	—	—	Base + (0x0110–0x013F)
CFIFO 1 register 0	EQADC_CF1R0	32-bit	Base + 0x0140
CFIFO 1 register 1	EQADC_CF1R1	32-bit	Base + 0x0144
CFIFO 1 register 2	EQADC_CF1R2	32-bit	Base + 0x0148
CFIFO 1 register 3	EQADC_CF1R3	32-bit	Base + 0x014C
Reserved	—	—	Base + (0x0150–0x017F)
CFIFO 2 register 0	EQADC_CF2R0	32-bit	Base + 0x0180
CFIFO 2 register 1	EQADC_CF2R1	32-bit	Base + 0x0184
CFIFO 2 register 2	EQADC_CF2R2	32-bit	Base + 0x0188
CFIFO 2 register 3	EQADC_CF2R3	32-bit	Base + 0x018C
Reserved	—	—	Base + (0x0190–0x01BF)
CFIFO 3 register 0	EQADC_CF3R0	32-bit	Base + 0x01C0
CFIFO 3 register 1	EQADC_CF3R1	32-bit	Base + 0x01C4
CFIFO 3 register 2	EQADC_CF3R2	32-bit	Base + 0x01C8

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
CFIFO 3 register 3	EQADC_CF3R3	32-bit	Base + 0x01CC
Reserved	—	—	Base + (0x01D0–0x01FF)
CFIFO 4 register 0	EQADC_CF4R0	32-bit	Base + 0x0200
CFIFO 4 register 1	EQADC_CF4R1	32-bit	Base + 0x0204
CFIFO 4 register 2	EQADC_CF4R2	32-bit	Base + 0x0208
CFIFO 4 register 3	EQADC_CF4R3	32-bit	Base + 0x020C
Reserved	—	—	Base + (0x0210–0x023F)
CFIFO 5 register 0	EQADC_CF5R0	32-bit	Base + 0x0240
CFIFO 5 register 1	EQADC_CF5R1	32-bit	Base + 0x0244
CFIFO 5 register 2	EQADC_CF5R2	32-bit	Base + 0x0248
CFIFO 5 register 3	EQADC_CF5R3	32-bit	Base + 0x024C
Reserved	—	—	Base + (0x0250–0x02FF)
RFIFO 0 register 0	EQADC_RF0R0	32-bit	Base + 0x0300
RFIFO 0 register 1	EQADC_RF0R1	32-bit	Base + 0x0304
RFIFO 0 register 2	EQADC_RF0R2	32-bit	Base + 0x0308
RFIFO 0 register 3	EQADC_RF0R3	32-bit	Base + 0x030C
Reserved	—	—	Base + (0x0310–0x033F)
RFIFO 1 register 0	EQADC_RF1R0	32-bit	Base + 0x0340
RFIFO 1 register 1	EQADC_RF1R1	32-bit	Base + 0x0344
RFIFO 1 register 2	EQADC_RF1R2	32-bit	Base + 0x0348
RFIFO 1 register 3	EQADC_RF1R3	32-bit	Base + 0x034C
Reserved	—	—	Base + (0x0350–0x037F)
RFIFO 2 register 0	EQADC_RF2R0	32-bit	Base + 0x0380
RFIFO 2 register 1	EQADC_RF2R1	32-bit	Base + 0x0384
RFIFO 2 register 2	EQADC_RF2R2	32-bit	Base + 0x0388
RFIFO 2 register 3	EQADC_RF2R3	32-bit	Base + 0x038C
Reserved	—	—	Base + (0x0390–0x03BF)
RFIFO 3 register 0	EQADC_RF3R0	32-bit	Base + 0x03C0
RFIFO 3 register 1	EQADC_RF3R1	32-bit	Base + 0x03C4
RFIFO 3 register 2	EQADC_RF3R2	32-bit	Base + 0x03C8
RFIFO 3 register 3	EQADC_RF3R3	32-bit	Base + 0x03CC
Reserved	—	—	Base + (0x03D0–0x03FF)

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
RFIFO 4 register 0	EQADC_RF4R0	32-bit	Base + 0x0400
RFIFO 4 register 1	EQADC_RF4R1	32-bit	Base + 0x0404
RFIFO 4 register 2	EQADC_RF4R2	32-bit	Base + 0x0408
RFIFO 4 register 3	EQADC_RF4R3	32-bit	Base + 0x040C
Reserved	—	—	Base + (0x0410–0x043F)
RFIFO 5 register 0	EQADC_RF5R0	32-bit	Base + 0x0440
RFIFO 5 register 1	EQADC_RF5R1	32-bit	Base + 0x0444
RFIFO 5 register 2	EQADC_RF5R2	32-bit	Base + 0x0448
RFIFO 5 register 3	EQADC_RF5R3	32-bit	Base + 0x044C
Reserved	—	—	Base + (0x0450–0x07FF)
ADC0 control register	ADC0_CR		No memory mapped access
ADC1 control register	ADC1_CR		
ADC time stamp control register	ADC_TSCR		
ADC time base counter register	ADC_TBCR		
ADC0 gain calibration constant register	ADC0_GCCR		
ADC1 gain calibration constant register	ADC1_GCCR		
ADC0 offset calibration constant register	ADC0_OCCR		
ADC1 offset calibration constant register	ADC1_OCCR		
Reserved	—	—	(Base + 0x0800)–0xFFF8_FFFF
Chapter 19, “Deserial Serial Peripheral Interface (DSPI)”			0xFFF9_4000 (DSPI B) 0xFFF9_8000 (DSPI C) 0xFFF9_C000 (DSPI D)
Module configuration register	DSPIx_MCR	32-bit	Base + 0x0000
Reserved	—	—	Base + (0x0004–0x0007)
Transfer count register	DSPIx_TCR	32-bit	Base + 0x0008
Clock and transfer attribute register 0	DSPIx_CTAR0	32-bit	Base + 0x000C
Clock and transfer attribute register 1	DSPIx_CTAR1	32-bit	Base + 0x0010
Clock and transfer attribute register 2	DSPIx_CTAR2	32-bit	Base + 0x0014
Clock and transfer attribute register 3	DSPIx_CTAR3	32-bit	Base + 0x0018
Clock and transfer attribute register 4	DSPIx_CTAR4	32-bit	Base + 0x001C
Clock and transfer attribute register 5	DSPIx_CTAR5	32-bit	Base + 0x0020
Clock and transfer attribute register 6	DSPIx_CTAR6	32-bit	Base + 0x0024
Clock and transfer attribute register 7	DSPIx_CTAR7	32-bit	Base + 0x0028

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Status register	DSPIx_SR	32-bit	Base + 0x002C
DMA/interrupt request select and enable register	DSPIx_RSER	32-bit	Base + 0x0030
Push TX FIFO register	DSPIx_PUSHR	32-bit	Base + 0x0034
Pop RX FIFO register	DSPIx_POPR	32-bit	Base + 0x0038
Transmit FIFO registers 0	DSPIx_TXFR0	32-bit	Base + 0x003C
Transmit FIFO registers 1	DSPIx_TXFR1	32-bit	Base + 0x0040
Transmit FIFO registers 2	DSPIx_TXFR2	32-bit	Base + 0x0044
Transmit FIFO registers 3	DSPIx_TXFR3	32-bit	Base + 0x0048
Reserved	—	—	Base + (0x004C–0x007B)
Receive FIFO registers 0	DSPIx_RXFR0	32-bit	Base + 0x007C
Receive FIFO registers 1	DSPIx_RXFR1	32-bit	Base + 0x0080
Receive FIFO registers 2	DSPIx_RXFR2	32-bit	Base + 0x0084
Receive FIFO registers 3	DSPIx_RXFR3	32-bit	Base + 0x0088
Reserved	—	—	Base + (0x008C–0x00BB)
DSI configuration register	DSPIx_DSICR	32-bit	Base + 0x00BC
DSI serialization data register	DSPIx_SDR	32-bit	Base + 0x00C0
DSI alternate serialization data register	DSPIx_ASDR	32-bit	Base + 0x00C4
DSI transmit comparison register	DSPIx_COMPR	32-bit	Base + 0x00C8
DSI deserialization data register	DSPIx_DDR	32-bit	Base + 0x00CC
Reserved	—	—	Base + 0x00D0–0xFFF9_7FFF (B) 0xFFF9_BFFF (C) 0xFFFA_FFFF (D)
Chapter 20, "Enhanced Serial Communication Interface (eSCI)"			0xFFFB_0000 (A) 0xFFFB_4000 (B)
Control register 1	ESCIx_CR1	32-bit	Base + 0x0000
Control register 2	ESCIx_CR2	16-bit	Base + 0x0004
Data register	ESCIx_DR	16-bit	Base + 0x0006
Status register	ESCIx_SR	32-bit	Base + 0x0008
LIN control register	ESCIx_LCR	32-bit	Base + 0x000C
LIN transmit register	ESCIx_LTR	32-bit	Base + 0x0010
LIN receive register	ESCIx_LRR	32-bit	Base + 0x0014
LIN CRC polynomial register	ESCIx_LPR	32-bit	Base + 0x0018

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Reserved	—	—	Base + 0x001C– 0xFFFB_3FFF (A) 0xFFFB_7FFF (B)
Chapter 21, "FlexCAN2 Controller Area Network"			0xFFFC_0000 (CAN A) 0xFFFC_8000 (CAN C)
Module configuration register	CANx_MCR	32-bit	Base + 0x0000
Control register	CANx_CR	32-bit	Base + 0x0004
Free running timer register	CANx_TIMER	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C–0x000F)
Receive global mask register	CANx_RXGMASK	32-bit	Base + 0x0010
Receive buffer 14 mask register	CANx_RX14MASK	32-bit	Base + 0x0014
Receive buffer 15 mask register	CANx_RX15MASK	32-bit	Base + 0x0018
Error counter register	CANx_ECR	32-bit	Base + 0x001C
Error and status register	CANx_ESR	32-bit	Base + 0x0020
Interrupt mask register high	CANx_IMRH	32-bit	Base + 0x0024
Interrupt mask register low	CANx_IMRL	32-bit	Base + 0x0028
Interrupt flag register high	CANx_IFRH	32-bit	Base + 0x002C
Interrupt flag register low	CANx_IFRL	32-bit	Base + 0x0030
Reserved	—	—	Base + (0x0034–0x007F)
Message buffer 0	MB0	128-bit	Base + 0x0080
Message buffer 1	MB1	128-bit	Base + 0x0090
Message buffer 2	MB2	128-bit	Base + 0x00A0
Message buffer 3	MB3	128-bit	Base + 0x00B0
Message buffer 4	MB4	128-bit	Base + 0x00C0
Message buffer 5	MB5	128-bit	Base + 0x00D0
Message buffer 6	MB6	128-bit	Base + 0x00E0
Message buffer 7	MB7	128-bit	Base + 0x00F0
Message buffer 8	MB8	128-bit	Base + 0x0100
Message buffer 9	MB9	128-bit	Base + 0x0110
Message buffer 10	MB10	128-bit	Base + 0x0120
Message buffer 11	MB11	128-bit	Base + 0x0130
Message buffer 12	MB12	128-bit	Base + 0x0140
Message buffer 13	MB13	128-bit	Base + 0x0150

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Message buffer 14	MB14	128-bit	Base + 0x0160
Message buffer 15	MB15	128-bit	Base + 0x0170
Message buffer 16	MB16	128-bit	Base + 0x0180
Message buffer 17	MB17	128-bit	Base + 0x0190
Message buffer 18	MB18	128-bit	Base + 0x01A0
Message buffer 19	MB19	128-bit	Base + 0x01B0
Message buffer 20	MB20	128-bit	Base + 0x01C0
Message buffer 21	MB21	128-bit	Base + 0x01D0
Message buffer 22	MB22	128-bit	Base + 0x01E0
Message buffer 23	MB23	128-bit	Base + 0x01F0
Message buffer 24	MB24	128-bit	Base + 0x0200
Message buffer 25	MB25	128-bit	Base + 0x0210
Message buffer 26	MB26	128-bit	Base + 0x0220
Message buffer 27	MB27	128-bit	Base + 0x0230
Message buffer 28	MB28	128-bit	Base + 0x0240
Message buffer 29	MB29	128-bit	Base + 0x0250
Message buffer 30	MB30	128-bit	Base + 0x0260
Message buffer 31	MB31	128-bit	Base + 0x0270
Message buffer 32	MB32	128-bit	Base + 0x0280
Message buffer 33	MB33	128-bit	Base + 0x0290
Message buffer 34	MB34	128-bit	Base + 0x02A0
Message buffer 35	MB35	128-bit	Base + 0x02B0
Message buffer 36	MB36	128-bit	Base + 0x02C0
Message buffer 37	MB37	128-bit	Base + 0x02D0
Message buffer 38	MB38	128-bit	Base + 0x02E0
Message buffer 39	MB39	128-bit	Base + 0x02F0
Message buffer 40	MB40	128-bit	Base + 0x0300
Message buffer 41	MB41	128-bit	Base + 0x0310
Message buffer 42	MB42	128-bit	Base + 0x0320
Message buffer 43	MB43	128-bit	Base + 0x0330
Message buffer 44	MB44	128-bit	Base + 0x0340
Message buffer 45	MB45	128-bit	Base + 0x0350

Table A-2. MPC5534 Detailed Register Map (continued)

Register Description	Register Name	Used Size	Address
Message buffer 46	MB46	128-bit	Base + 0x0360
Message buffer 47	MB47	128-bit	Base + 0x0370
Message buffer 48	MB48	128-bit	Base + 0x0380
Message buffer 49	MB49	128-bit	Base + 0x0390
Message buffer 50	MB50	128-bit	Base + 0x03A0
Message buffer 51	MB51	128-bit	Base + 0x03B0
Message buffer 52	MB52	128-bit	Base + 0x03C0
Message buffer 53	MB53	128-bit	Base + 0x03D0
Message buffer 54	MB54	128-bit	Base + 0x03E0
Message buffer 55	MB55	128-bit	Base + 0x03F0
Message buffer 56	MB56	128-bit	Base + 0x0400
Message buffer 57	MB57	128-bit	Base + 0x0410
Message buffer 58	MB58	128-bit	Base + 0x0420
Message buffer 59	MB59	128-bit	Base + 0x0430
Message buffer 60	MB60	128-bit	Base + 0x0440
Message buffer 61	MB61	128-bit	Base + 0x0450
Message buffer 62	MB62	128-bit	Base + 0x0460
Message buffer 63	MB63	128-bit	Base + 0x0470
Reserved	—	—	Base + 0x0480–0xFFFC_3FFF (A) 0xFFFF_FFFF (C)
Chapter 15, “Boot Assist Module (BAM)”			0xFFFF_C000
BAM Program Mirrored		4 KB	0xFFFF_C000–0xFFFF_CFFF
BAM Program Mirrored		4 KB	0xFFFF_D000–0xFFFF_DFFF
BAM Program Mirrored		4 KB	0xFFFF_E000–0xFFFF_EFFF
BAM Program		4 KB	0xFFFF_F000–0xFFFF_FFFF

[1] The registers mapped in the ECSM module (0xFFF4_0014–0xFFF4_001F) provide control and configuration for a software watchdog timer, and are included as part of a standard Freescale ECSM block incorporated in the MPC5534. The e200z3 core also provides this functionality and is the preferred method for watchdog implementation. To optimize code portability to other members of the eSys MPU family, use of the watchdog registers in the ECSM is not recommended.

A.2 e200z3 Core SPR Numbers

Table A-3. e200z3 Core SPR Numbers (Supervisor Mode)

Register	Description	SPR (decimal)
General Registers		
XER	Integer Exception Register	1
LR	Link Register	8
CTR	Count Register	9
GPR0–GPR31	General Purpose Registers	—
Special Purpose General Registers		
SPRG0	Special Purpose Register General 0	272
SPRG1	Special Purpose Register General 1	273
SPRG2	Special Purpose Register General 2	274
SPRG3	Special Purpose Register General 3	275
SPRG4	Special Purpose Register General 4	276
SPRG5	Special Purpose Register General 5	277
SPRG6	Special Purpose Register General 6	278
SPRG7	Special Purpose Register General 7	279
USPRG0	User Special Purpose General Register	256
BUCSR	Branch Unit Control and Status Register	1013
Exception Handling/Control Registers		
SRR0	Save and Restore Register 0	26
SRR1	Save and Restore Register 1	27
CSRR0	Critical Save and Restore Register 0	58
CSRR1	Critical Save and Restore Register 1	59
DSRR0	Debug Save and Restore Register 0	574
DSRR1	Debug Save and Restore Register 1	575
ESR	Exception Syndrome Register	62
MCSR	Machine Check Syndrome Register	572
DEAR	Data Exception Address Register	61
IVPR	Interrupt Vector Prefix Register	63
IVOR0	Interrupt Vector Offset Register 0	400
IVOR1	Interrupt Vector Offset Register 1	401
IVOR2	Interrupt Vector Offset Register 2	402
IVOR3	Interrupt Vector Offset Register 3	403

Table A-3. e200z3 Core SPR Numbers (Supervisor Mode) (continued)

Register	Description	SPR (decimal)
IVOR4	Interrupt Vector Offset Register 4	404
IVOR5	Interrupt Vector Offset Register 5	405
IVOR6	Interrupt Vector Offset Register 6	406
IVOR7	Interrupt Vector Offset Register 7	407
IVOR8	Interrupt Vector Offset Register 8	408
IVOR9	Interrupt Vector Offset Register 9	409
IVOR10	Interrupt Vector Offset Register 10	410
IVOR11	Interrupt Vector Offset Register 11	411
IVOR12	Interrupt Vector Offset Register 12	412
IVOR13	Interrupt Vector Offset Register 13	413
IVOR14	Interrupt Vector Offset Register 14	414
IVOR15	Interrupt Vector Offset Register 15	415
IVOR32	Interrupt Vector Offset Register 32	528
IVOR33	Interrupt Vector Offset Register 33	529
IVOR34	Interrupt Vector Offset Register 34	530
Processor Control Registers		
MSR	Machine State Register	—
PVR	Processor Version Register	287
PIR	Processor ID Register	286
SVR	System Version Register	1023
HID0	Hardware Implementation Dependent Register 0	1008
HID1	Hardware Implementation Dependent Register 1	1009
Timer Registers		
TBL	Time Base Lower Register	284
TBU	Time Base Upper Register	285
TCR	Timer Control Register	340
TSR	Timer Status Register	336
DEC	Decrementer Register	22
DECAR	Decrementer Auto-reload Register	54
Debug Registers		
DBCR0	Debug Control Register 0	308
DBCR1	Debug Control Register 1	309
DBCR2	Debug Control Register 2	310

Table A-3. e200z3 Core SPR Numbers (Supervisor Mode) (continued)

Register	Description	SPR (decimal)
DBCR3	Debug Control Register 3	561
DBSR	Debug Status Register	304
DBCNT	Debug Counter Register	562
IAC1	Instruction Address Compare Register 1	312
IAC2	Instruction Address Compare Register 2	313
IAC3	Instruction Address Compare Register 3	314
IAC4	Instruction Address Compare Register 4	315
DAC1	Data Address Compare Register 1	316
DAC2	Data Address Compare Register 2	317
Memory Management Registers		
MAS0	MMU Assist Register 0	624
MAS1	MMU Assist Register 1	625
MAS2	MMU Assist Register 2r	626
MAS3	MMU Assist Register 3	627
MAS4	MMU Assist Register 4	628
MAS6	MMU Assist Register 6	630
PID0	Process ID Register	48
MMUCSR0	MMU Control and Status Register 0	1012
MMUCFG	MMU Configuration Register	1015
TLB0CFG	TLB 0 Configuration Register	688
TLB1CFG	TLB 1 Configuration Register	689
Cache Registers		
L1CFG0	L1 Cache Configuration Register	515
APU Registers		
SPEFSCR	SPE APU Status and Control Register	512

Table A-4. e200z3 Core SPR Numbers (User Mode)

Register	Description	SPR (decimal)
General Registers		
CTR	Count Register	9
LR	Link Register	8
XER	Integer Exception Register	1
GPR0–GPR31	General Purpose Registers	—
Special Purpose Registers (General)		
SPRG4	Special Purpose Register General 4	260
SPRG5	Special Purpose Register General 5	261
SPRG6	Special Purpose Register General 6	262
SPRG7	Special Purpose Register General 7	263
USPRG0	User Special Purpose Register General	256
Timer Registers		
TBL	Time Base Lower Register	268
TBU	Time Base Upper Register	269
Cache Registers		
L1CFG0	L1 Cache Configuration Register	515
APU Registers		
SPEFSCR	SPE APU Status and Control Register	512

Appendix B
Calibration

NOTE

In addition to the MPC5534 device, the 496 VertiCal assembly is required to use the calibration features. The External Bus Interface (EBI) and Calibration Bus Interface (CBI) are not available due to pin limitations in the 208 package.

B.1 Overview

The MPC5500 family of microcontrollers includes various specialized features to support automotive calibration. Many of these calibration features are not available to the final application software, and some MPC5500 devices support calibration signals that are not available in the standard 208 or 324 packages.See the Signals chapter for 324 package limitations. Special calibration assembled devices with increased signal bond-out provide full access to all calibration resources for all MPC5500 variants.

Calibration hardware that uses the calibration assembled devices is detailed in Figure B-1. Freescale-produced VertiCal bases use the calibration-assembled MPC5500 device mounted on a small circuit board with a footprint which is compatible with that of the production BGA packaged MPC5500 device. A 156-way VertiCal connector on the top-side of the VertiCal base allows you to attach VertiCal compliant top-board hardware. Various types of top-board hardware to support calibration and debug are available from Freescale and commercial companies.

The VertiCal connector standard defines the set of signals used to communicate between the microcontroller on the VertiCal base board and the attached calibration development tools, called top-boards. Signal availability or sourcing for the VertiCal connector differs depending on the MPC5500 device used.

The calibration system is illustrated in Figure B-1 and the VertiCal Base is illustrated in Figure B-2.

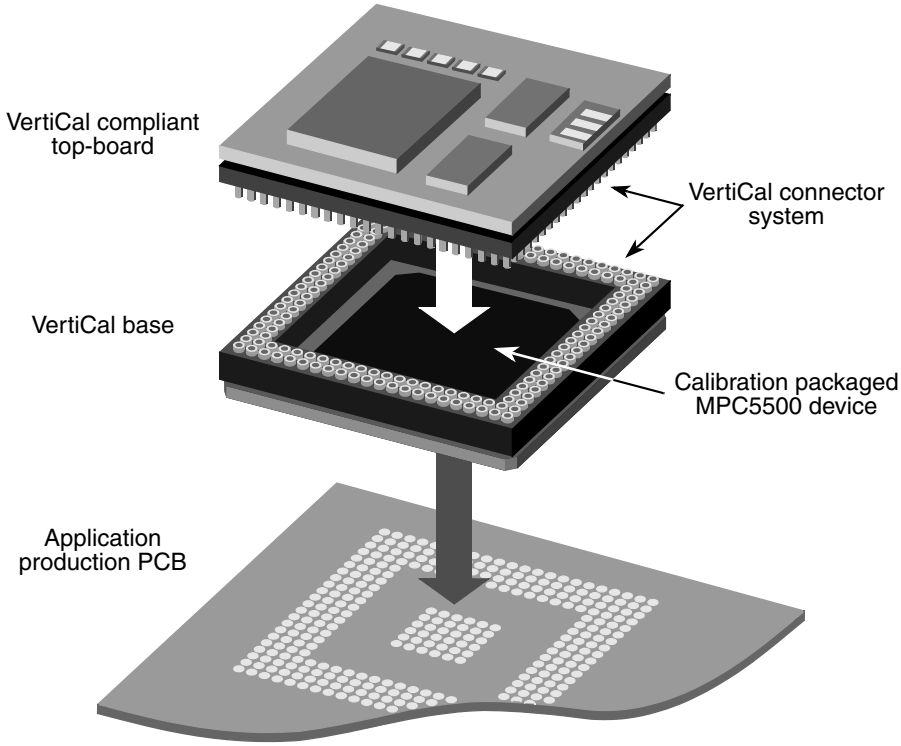


Figure B-1. Calibration Assembly

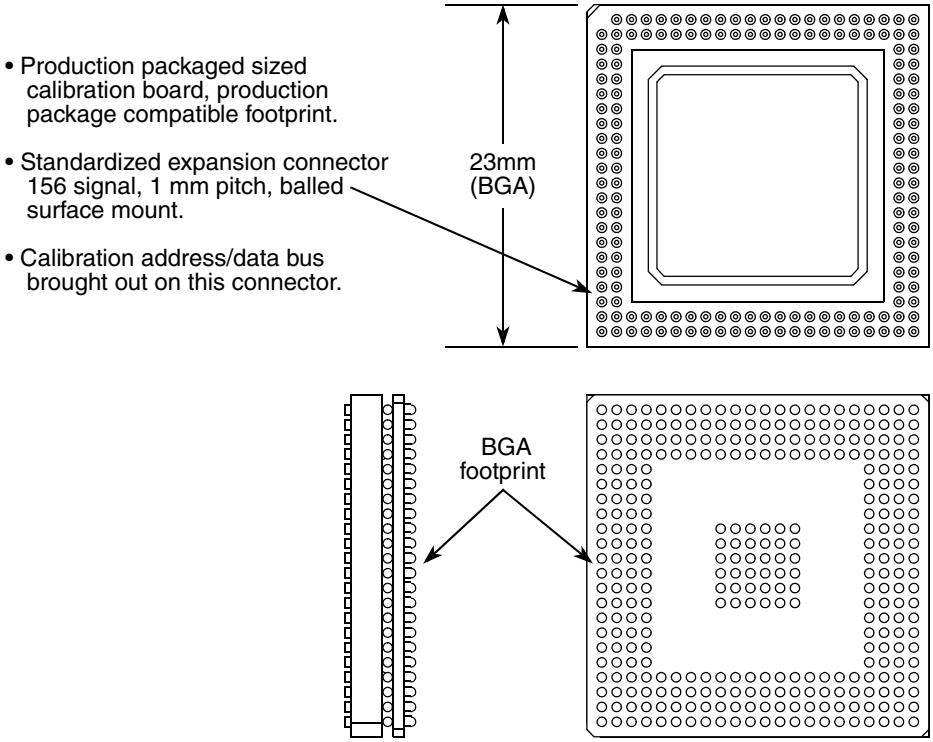


Figure B-2. VertiCal Base

B.2 Calibration Bus Interface

The calibration bus interface (CBI) has an address bus, data bus, bus control and clock signals. The calibration bus is used by tools that include memory for calibration data or other code in development. See Table B-1 for calibration bus signals. A 16-bit data bus and 19-bit address bus gives a basic addressing range of 1 MB. Alternatively, the maximum memory addressable using just one chip select is 4 MB. See Table B-2.

The VertiCal connector supports up to four chip select signals, although the actual number of chip selects available depends on the device and package. Use the CAL_$\overline{CS}$[0] chip select as the default calibration chip select to ensure maximum portability of calibration tools across devices. The three calibration chip select signals CAL_$\overline{CS}$[0, 2:3] are configured and function like the external chip select signals $\overline{CS}$[0:3], except the calibration chip selects have a higher priority in address decoding than the external chip selects, $\overline{CS}$[0:3]. See Section B.7, “Application Information,” for application information on the number of calibration chip selects.

The CAL_$\overline{CS}$[0, 2:3] chip selects have multiplexed signal functions to provide additional addressing bits that allows the flexibility of increasing the addressing range or the number of chip selects. Devices that support less than four calibration chip selects can extend the contiguous calibration addressing range by omitting chip selects starting from $\overline{CS}$[1]. For this reason $\overline{CS}$[1] is the only chip select pin not implemented on the MPC5534 device.

The calibration functionality does not use any I/O available in the 208 and 324 pin production packages for a calibration bus interface.

The P/A/G column in the following table differentiates each signal function that is multiplexed to the same pin assignment. The P/A/G value is set in the PA field of the SIU_PCR registers and determines which multiplexed signal function controls the pin. For more information on how to set the PA field, read the Signals Chapter.

Table B-1. Calibration Bus Signals

496 VertiCal Signal Name	Function	MPC5534 Signal Name	P/A/G
Address and Data Bus Signals			
CAL_ADDR[12:26]	Calibration Address bus	CAL_ADDR[12:26]	P
CAL_ADDR[27:30]	Calibration Address bus	CAL_ADDR[27:30]	P
CAL_$\overline{CS}$[3]	Calibration Chip Select	CAL_$\overline{CS}$[3]	P
CAL_$\overline{CS}$[2]	Calibration Chip Select	CAL_$\overline{CS}$[2]	P
CAL_$\overline{CS}$[1]	Calibration Chip Select	No Connect	—
CAL_$\overline{CS}$[0]	Calibration Chip Select	CAL_$\overline{CS}$[0]	P
CAL_DATA[0:15]	Calibration Data Bus	CAL_DATA[0:15]	P
CAL_$\overline{OE}$	Calibration Output Enable	CAL_$\overline{OE}$	P
CAL_RD_$\overline{WR}$	Calibration Read/Write	CAL_RD_$\overline{WR}$	P
CAL_$\overline{TS}$	Calibration Transfer Start	CAL_$\overline{TS}$	P

Table B-1. Calibration Bus Signals (continued)

496 VertiCal Signal Name	Function	MPC5534 Signal Name	P/A/G
CAL_$\overline{WE}$/$\overline{BE}$[0:1]	Calibration Write/Byte Enable	CAL_$\overline{WE}$/$\overline{BE}$[0:1] GPIO[64:65]	P G
Clock Synthesizer			
CLKOUT	System Clock Output	CLKOUT	P

B.3 Device-Specific Information

The various address bus, data bus and bus control signals are sourced from different signals depending on the MPC5500 device used, as detailed in the following sections.

B.3.1 MPC5534 Calibration Bus Implementation

The MPC5534 device has a set of external bus signals that are only used by the calibration bus. These device signals are prefixed by CAL, and their use does not affect the usage modes and electrical loading on the equivalent signals for the EBI.

B.4 Signals and Pads

The following sections detail the signal descriptions for the calibration bus.

B.4.1 CAL_$\overline{CS}$[0, 2:3] — Calibration Chip Selects 0, 2 and 3

CAL_$\overline{CS}$[n] is asserted by the master to indicate that this transaction is targeted for a particular calibration memory bank.

The calibration chip selects (CAL_$\overline{CS}$[n]) are driven by the EBI in the same clock as the assertion of $\overline{TS}$ and valid address, and is kept valid until the cycle is terminated. Bus timing is identical to standard EBI timing.

B.4.1.1 Number of Chip Selects and Maximum Memory Size

The trade-off between calibration chip selects and address lines is the same as the trade-off between non-calibration chip selects and address lines for the 324 pin package.

Table B-2. Maximum Memory Size According to Calibration Chip Selects

	Maximum Memory Allocated for Calibration		
	Device 0	Device 2	Device 3
CAL_$\overline{CS}$[0] only	4 MB	—	—
CAL_$\overline{CS}$[0] and CAL_$\overline{CS}$[2]	2 MB	2 MB	—
CAL_$\overline{CS}$[0] and CAL_$\overline{CS}$[2:3]	1 MB	1 MB	1 MB

B.4.2 Pad Ring

This section provides a list of the calibration pins and associated pad configuration registers (PCRs), including links to the detailed PCR information for each pin or pin group.

See Table B-1 for device signal names. The drive strength of the calibration pins is configured in the PCR registers. In some cases, multiple pads have their drive strengths controlled by one PCR by grouping the pins:

- CAL_ADDR[12:30]
- CAL_DATA[0:15]
- CAL_RD_$\overline{WR}$, CAL_$\overline{WE}$/$\overline{BE}$[0:1], CAL_$\overline{OE}$, CAL_$\overline{TS}$

The SIU_PCR registers control whether the CAL_$\overline{CS}$[2:3] pins are used for CAL_$\overline{CS}$[2:3] or for CAL_ADDR[10:11]. See Table B-2 for the pin assignments. Selecting between CAL_$\overline{CS}$[2:3] and CAL_ADDR[10:11] allows you to maximize the amount of calibration memory size by limiting the number of calibration chip selects to $\overline{CS}$[0]. See Section B.4.1.1, “Number of Chip Selects and Maximum Memory Size.”

Table B-3. Calibration Pin Assignments

SIU_PCR Number	Primary Function (SIU_PCR[PA] = 0b1)	Alternate Function (SIU_PCR[PA] = 0b0)
338 and 339	CAL_$\overline{CS}$[2:3]	CAL_ADDR[10:11]

B.4.3 CLKOUT

CLKOUT is supplied by the clock control block, not the EBI. Nevertheless, the same CLKOUT is used for both the non-calibration and calibration bus.

A drawback of having just one CLKOUT is that while the difference in board timing can be compensated by the adjustment in the drive strength, the CLKOUT timing, and hence the timing of the non-calibration bus, can have minor differences with a calibration tool from the production package.

B.5 Power Supplies

The signals that make up the calibration bus have their own power supply segment (V_{DDE12}). The V_{DDE12} power supply balls are not connected to any other power supply segment from the standard package ball-out but are routed on the VertiCal base to pins on the VertiCal connector. The VertiCal top board must provide voltage to the V_{DDE12} power supply pins to power up the calibration bus.

B.6 Integration Logic Functionality

The EBI connects to both the non-calibration and calibration buses. The integration logic on MPC5534 selects between the data input from both buses to the EBI.

The MPC5534 integration logic also suppresses the reflections of the outputs of the calibration bus onto the non-calibration bus. For the non-calibration bus pins that do not have a negated state to which the pins return at the end of the access, this reflection suppression is enabled by the SIU_CCR[CRSE] bit. SIU_CCR[CRSE] does not enable reflection suppression for the non-calibration bus pins that have a negated state to which the pins return at the end of an access. Those reflections always are suppressed. Furthermore, the suppression of reflections from the non-calibration bus onto the calibration bus is not enabled by CRSE. Those reflections are also always suppressed.

See Section B.7.1, “Enabling Calibration Reflection Suppression,” for when to set SIU_CCR[CRSE] or leave it in its reset negated state.

B.7 Application Information

B.7.1 Enabling Calibration Reflection Suppression

Set SIU_CCR[CRSE] to suppress reflections when calibrating. The calibration reflection suppression logic for an output that does not return to a negated state at the end of an access can introduce a small glitch on the output at the end of the access. The glitch does not interfere with the output valid or hold times. However, keep SIU_CCR[CRSE] in its reset negated state when not calibrating to prevent a glitch on the non-calibration bus outputs.

B.7.2 Communication With Development Tool Using I/O

The development tool can require some I/Os for communication between the MCU and the development tool on the VertiCal connector. Because the application cannot use these pins in the 208 and 324 pin packages, they can be used for development tool use in a VertiCal connector.

B.7.3 Matching Access Delay to Internal Flash With Calibration Memory

One use of VertiCal in the Automotive environment is engine calibration. For this application, an SRAM Top Board is added onto the VertiCal connector. This allows the engine calibrator to modify settings in SRAM, possibly using the Nexus interface or even by using the SCI port or a CAN interface.

See Table 13-2 “Internal Flash External Emulation Mode.”

After the data is calibrated, it can be copied into the internal flash. The internal flash can be accessed faster than the calibration memory, and the change in calibration data access time can change the overall system performance. To mitigate this change in system performance, the internal flash memory includes a feature that allows accesses to portions of the flash to be slowed down by adding extra wait states. This is done by multiply mapping the internal flash at different locations with different number of wait states. For example, the physical address of the flash array is 0x0000_0000 to 0x00FF_FFFF (depending on array size). That same flash data can be accessed at address 0x0100_0000 to 0x01FF_FFFF but accesses are one clock cycle slower. That same flash data can be accessed at addresses 0x0200_0000 to 0x02FF_FFFF but accesses are two clock cycles slower. This pattern is repeated through the memory map to addresses 0x1F00_0000 to 0x1FFF_FFFF where accesses are 31 clock cycles slower.

The application can use this feature by mapping the calibration data to a region of the flash memory that has access timing to match the timing of the calibration RAM used when calibrating the data. This remapping of calibration data can be achieved by either using the translation feature of the MMU or rebuilding the code with a modified link file.

Appendix C
MPC5534RM Revision History

This appendix describes corrections to the *MPC5534 Reference Manual*. For convenience, the corrections are grouped by revision.

C.1 Changes between Rev. 1 and Rev. 2

Table C-1. Changes Between Revisions 1 and 2

Chapter	Description
All chapters	Throughout, language and format cleanup.
Chapter 1, "Overview"	Changed switchpoint to 35 or 65.
	Minor editorial changes to the first three paragraphs in the Chapter Introduction. Combined sentences two and three in paragraph seven.
	Figure 1-1. MPC5534 Block Diagram: Relocated the JTAG module outside of the e200z3 core between the external connection and the Nexus Interface. Removed arrow from the eDMA to the Nexus interface.
	Table 1-1. MPC5500 Family Members: Added footnote 7: '82 MHz parts allow for 80 MHz system clock + 2% FM' to columns MPC5533 and MPC5534; and footnote 8: '132 MHz parts allow for 128 MHz system clock + 2% FM' to columns MPC5553 and MPC5554.
	Section 1.2.1 Operating Parameters: Changed 75% to 65% in the first subbullet in 'Input and output pins with 3.0–5.5 V range.
	Section 1.2.2 e200z3 Core Processor: Changed PowerPC Book E to PowerPC Architecture Book E.
	Section 1.2.7 External Bus Interface (EBI): Added BE to the 10th first-level bullet to make 'Two write/byte enable ($\overline{WE}$/$\overline{BE}$[0:1]) signals.
	Section 1.2.8 Calibration Bus: Title changes to Calibration Bus Interface. Added BE to the 7th first-level bullet to make 'Two write/byte enable ($\overline{WE}$/$\overline{BE}$[0:1]) signals.
	Section 1.3.1 External Master Mode Operation Memory Map: Table 1-3, shaded row one.
	Section 1.3.1 External Master Mode Operation Memory Map: Table 1-4, MPC5534 Slave Memory Map as seen from an External Master, shaded row two. Changed 'shadow row' to 'shadow block' throughout the entire document.
	Section 1.4.7 Calibration Bus (cal_bus): Changed title to Calibration Bus Interface (CBI), changed cal_bus to CBI.
	Section 1.4.11: Changed from 'The BAM also reads the reset configuration half word (RCHW) from Flash memory (either internal or external) and configures the MPC5534 hardware accordingly.' to 'The BAM reads the reset configuration half word (RCHW) from flash memory and configures the device. Flash memory can be either internal (208 and 324 packages) or external (324 package only).'

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 1, "Overview" *(continued)*	Table 1-1. MPC5500 Family Members: Added footnote 5 to the eQADC channel number row: 'eQADC has 34 channels on the 208 package.'
	Section 1.4.13 Enhanced Time Processing Unit (eTPU): deleted bullet that reads, Hardware implementation of four semaphores support coherent parameter sharing between both eTPU engines
	Section 1.2.7 External Bus Interface (EBI): Added NOTE at the beginning of the section: NOTE EBI features apply to devices using the 324 package, The EBI is not available in the 208 package.
Chapter 2, "Signals"	Updated "Signal Properties" table to match other MPC5500 devices (e.g. complete signal names, VertiCal information, etc.)
	Updated introductory text in section, "External Signal Description" and added figure, "Primary Function Not Available on Device".
	Added Section 2.2.1. Multiplexed Signals, including Figures 2-2 and 2-3.
	Changed eMIOS pins [0:23] all to' — / WKPCFG'
	Added Section 2.2.2. Device Signals Summary: Added at the end of the first paragraph before Table 2-1. MPC5534 Signal Properties: The signals shown in **red** are not available on the 208 package. Signals shown in **blue** are not designed into this device.
	Table MPC5534 Power/Ground Segmentation: • Changed 'MTS' to 'eMIOS' • Updated the signal names to include the alternate and GPIO signals.
	Section 2.1 Block Diagram: Added 'Signals designated in red are not available on the 208 package.' to the second paragraph.
	Table 2-1. MPC5534 Signal Properties: Marked signal names that are not available in the 208 package in red text in column one.
	Table 2-1. MPC5534 Signal Properties: Deleted column one to eliminate redundancy; added a column two to describe the signal function, and moved the PAG column after the description column.
	Table 2-1. MPC5534 Signal Properties: Included all primary signals in column one and put 'No primary signal' in the second column (Description) for all primary signals not designed in the device.
	Table 2-1. MPC5534 Signal Properties: Power / Ground signals: added V_{RCVSS}, Voltage regulator control ground, P, I —, VSSE, I / –, V_{RCVSS}, —, T21, V27.
	Table 2-1. MPC5534 Signal Properties: Footnotes 1 and 2 identical. Removed footnote 2.
	Table 2-1. MPC5534 Signal Properties: Footnote 5: 'Added the 208' to read 'The 496 assembly contains the VertiCal base and includes 324 and 208 package pins.'
	Table 2-1. MPC5534 Signal Properties: Added footnote 7: The BOOTCFG[0] and $\overline{\text{RSTCFG}}$ pin are not available in the 208 package and are internally asserted (driven to 0) in the package.
	Table 2-1. MPC5534 Signal Properties: removed footnote 9: 'Do not configure both the primary function of ADDR[8:11]_GPIO[4:7] and the alternate function of $\overline{\text{CS}}$[0:3]_ADDR[8:11]_GPIO[0:3] pins as address input pins. Only configure one set of pins for the address input. ' The 324 and 208 packages do not have ADDR[8:11]_GIOP[4:7] pins due to package limitations.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 2, "Signals" *(continued)*	Table 2-1. MPC5534 Signal Properties: • Marked footnotes 12 and 13 with red text since these signals are not available on the 208 package. • Footnote 12: Removed WE/BE[2:3] from WE/BE[2:3]_CAL_WE/BE[0:1]_GPIO[66:67] which are not available on the 324 or the 208 packages.
	Table 2-1. MPC5534 Signal Properties: Marked footnotes 23 and 24 (were 24 and 25) with red text since these signals are not available on the 208 package.
	Table 2-2: MPC5534 Power/Ground Segmentation: Added two sentences before the table: The primary signals shown in **blue** are not designed into the MPC5534 device, and are shown to locate the pin on the ball grid array (BGA). The signals shown in **red** are not available on the 208 package.
	Table 2-2: MPC5534 Power/Ground Segmentation: Footnote 1: Changed from V_{DDEH} 3.0–5.5 V to V_{DDEH} 3.0–5.25 V.
	Table 2-1. MPC5534 Signal Properties: Changed P/A/G value of AN[12:15] from P (primary=01) to MP (main primary=11).
	Section 2.3.8.4: Changed section title from PCSD2 / GPIO to DSPI D / GPIO
	Sections 2.8.2.1through 2.8.2.9: Added the DSPI A primary signals to the sections respectively: SCKA, SINA, SOUTA, PCSA[0], PCSA[1], PCSA[2], PCSA[3], PCSA[4], PCSA[5].
	Table 2-1. MPC5534 Signal Properties: Removed ETRIG[0:1]_GPIO[111:112].
	Section 2.3.9 Enhanced Queued Analog/Digital Converter (eQADC): Added NOTE at the beginning of the section: NOTE The eQADC has 40 channels in the 324 package; the 208 packages in limited to 34 channels due to pin limitations.
Chapter 3, "Core Complex (e200z3)"	Spelled out Auxiliary Processing Unit (APU) in the first paragraph.
	Added Section 3.2.1 e200z3 Core Features not Supported on this Device.
	Section 4.2. Features: From Power management bullet: removed second subbullet: Power saving modes: doze, nap, sleep.
	Section 4.5.4 Permissions: Removed the last sentence before Figure: 4-4 'Granting Access Permission' —"The current privilege level of an access is signaled to the MMU with the CPUs p_[d,i]tc[0] output signals,"
	Initial capped the headings for Sections'4.2.1 Feature Summary,' '4.7.1.1 User-level Registers,' and 4.7.1.2: Supervisor-level registers.
	Section '4.7.1.1 User-level Registers,': changed: GPROGPR31 to GPR0–GPR31; CR0CR7 to CR0–CR7.
	Section 4.10: 'Book E Instruction Extensions—VLE,' added space between Book and E, and replaced the hyphen with an emdash before VLE.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 4, “Reset”	Corrected base addresses.
	The WDRS bit in the reset status register (SIU_RSR) is set when the watchdog timer or a debug request reset occurs. A watchdog timer reset occurs and the WDRS bit is set when all the following conditions occur: • e200z3 core watchdog timer is enabled with the enable next watchdog timer (EWT) • Watchdog timer interrupt status (WIS) bits are set in the timer status register (TSR) • Watchdog reset control (WRC) field in the timer control register (TCR) is configured to reset • Time-out occurs The debug tool can issue a debug reset command by writing 2’b10 to the RST bit {DBCR0[2:3]} register in the e200z3 core, which sets the WDRS bit in the reset status register of the systems integration unit (SIU_RSR). To determine if WDRS was set by a watchdog timer or debug reset, check the WRS field in the e200z3 core TSR. The effect of a watchdog timer or debug reset request is the same on the reset controller. The debug tool can also reset the device using one of the following methods: • Debug tool asserts the $\overline{\text{RESET}}$ signal on the RESET_b pin • Debug tool sets the software system reset (SSR) bit in the system reset control register (SIU_SRCR) The debug tool writes a one to the software external reset (SER) bit in the system reset control register (SIU_SRCR) to generate an external software reset.
	The device comes out of reset using the following sequence: 1. Starting when the internal reset signal asserts, as indicated by $\overline{\text{RSTOUT}}$ asserting, the value on the WKPCFG pin is applied. At the same time, the PLLCFG[0:1] values are applied only if $\overline{\text{RSTCFG}}$ is asserted. 2. After the FMPLL is locked, the reset controller waits the predetermined number of clock cycles before negating $\overline{\text{RSTOUT}}$. When the clock count finishes, WKPCFG and BOOTCFG[0:1] are sampled. BOOTCFG[0:1] is only sampled if $\overline{\text{RSTCFG}}$ asserts. 3. The reset controller then waits 4 clock cycles before the negating $\overline{\text{RSTOUT}}$, and the associated bits/fields are updated in the SIU_RSR. See the e200z3 Core Guide for more information on the watchdog timer and debug operation. See Section 3.2.2, “Reset Output (RSTOUT).” BOOTCFG[0] is not available on the 208 package.
	Section 3.1 Introduction: Added two types of qualifying sentences for the 208 package throughout the chapter: 208 Package: BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0). 208 Package: BOOTCFG[0] and $\overline{\text{RSTCFG}}$ are not available due to pin limitations and are internally asserted (driven to 0). Therefore, BOOTCFG[1] and PLLCFG[0:1] are always sampled.
	Section 3.2.5 Boot Configuration (BOOTCFG [0:1]): changed DATA[0:31] to DATA[0:15], and $\overline{\text{WE}}$/$\overline{\text{BE}}$[0:3] to $\overline{\text{WE}}$/$\overline{\text{BE}}$[0:1].
	Section 4.4.3.2. External Reset: Added an introductory paragraph about the external reset limitations: The external reset feature is available on this device in the 324 package only, which has a 16-bit external bus interface. The 208 package does not have EBI pins, therefore the external reset feature is not supported.
	Added entire Section 4.4.3.3.1 BOOTCFG[0:1] Configuration in the 208 Package.
Chapter 5, “Peripheral Bridge”	No changes.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 6, "System Integration Unit (SIU)""	In figure titled "Pad Configuration Registers 8–27 (SIU_PCR8–SIU_PCR27)," changed PA field to be 1 bit instead of 3 bits. Changed footnote #1 to "When configured as ADDR[12:31], the OBE bit has no effect. When configured as GPO, set the OBE bit to one."
	Added "w1c" to write portion of EIF bits in EISR register.
	Modified the paragraph directly preceding Table 6-15 for clarity and to remove primary signal names not available on this device.
	Section 6.3.1.12.12 SIU_PCR75–SIU_PCR82, MDO [4:11]; Changed the register sequence to SIU_PCR82–SIU_PCR75 and the MDO pins to [11:4]
	Changed footnote 1 on PCR 105: From: "When configured as PCS, the OBE bit has no effect. When configured as GPO, set the OBE bit to 1. " To: "When configured as PCSB[0], the OBE bit has no effect. When configured as PCSD[2], set the OBE bit to 1 for master operation, and clear it to 0 for slave operation. When configured as GPO, set the OBE bit to 1."
	Modified the paragraph directly preceding Table 6-15 for clarity and to remove primary signal names not available on this device.
	Figure 6-1. SIU Block Diagram: • Added $\overline{\text{RSTCFG}}$ and PLLCFG[0]. • Changed listings of IRQs and CS signals to indicate full sequence of pins. • Added footnote on IRQ[1:5, 7:15]. The footnote reads: IRQ[6] is not designed into this device. IRQ[2] is not available due to pin limitations on the 208 package. • Added footnote on BOOTCFG[0:1] that reads: BOOTCFG[0] and $\overline{\text{RSTCFG}}$ are not available and are internally asserted (driven to 0) in the 208 package. • Added footnote on $\overline{\text{CS}}$[0:3] that reads: $\overline{\text{CS}}$[1:3] are not available due to pin limitations on the 208 package.
	Section 6.2.1 Overview: Reformatted features bulleted list into Table 6-1. SIU Features.
	Section 6.2.2 Modes of Operation: • Reformatted operating mode subsections into Table 6-2. SIU Operating Modes.
	Section 6.3.1.2 Reset Output ($\overline{\text{RSTOUT}}$): • Added: 208 Package: BOOTCFG[0] is not available due to pun limitations and is internally asserted (driven to 0) in the 208 package.
	Section 6.3.1.3 General-Purpose I/O Pins (GPIO[0:213]): • Added: NOTE: Not all GPIO pins are available on all packages. See Chapter 2, "Signals" for a listing of available GPIO pins.
	Section 6.3.1.4 BOOT Configurations Pins (BOOTCFG[0:1]): • Numbered the sequence of occurrences if RSTCFG negates while processing a reset. • Added; 208 Package: BOOTCFG[0] and $\overline{\text{RSTCFG}}$ are not available due to pin limitations and are internally asserted (driven to 0) in the 208 package.
	Section 6.3.1.6 External Interrupt Request Input Pins ($\overline{\text{IRQ}}$[0:15]): • Changed section title to External Interrupt Request Input Pins ($\overline{\text{IRQ}}$[0:5, 7:15]). • Throughout this chapter, changed $\overline{\text{IRQ}}$[0:15] to $\overline{\text{IRQ}}$[0:5, 7:15]. • Added: NOTE $\overline{\text{IRQ}}$[6] is not designed into this device. • Added: 208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 6, "System Integration Unit (SIU)" *(continued)*	Section 6.3.1.6.1 External Interrupts: • Added: NOTE $\overline{\text{IRQ}}$[6] is not designed into this device. • Added: 208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.
	Section 6.3.1.6.2 DMA Transfers: Added: 208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.
	Section 6.3.1.6.3 Overruns: • Added: NOTE $\overline{\text{IRQ}}$[6] is not designed into this device. • Added: 208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.
	Section 6.3.1.6.4 Edge Detects: • Added: NOTE $\overline{\text{IRQ}}$[6] is not designed into this device. • Added: 208 Package: $\overline{\text{IRQ}}$[2] is not available in the 208 package due to pin limitations.
	Table 6-3. SIU Signal Properties: • Added the $\overline{\text{RSTCFG}}$, PLLCFG[0], and PLLCFG[1]. • Added footnote 2: $\overline{\text{RSTCFG}}$ and BOOTCFG[0] pins are not available on the 208 package. These signals are internally asserted (driven to 0) in the 208 package. • Added to footnote 3: The GPIO and IRQ pins are multiplexed with other functions on the chip. The $\overline{\text{IRQ}}$[6] function is not designed into this device. $\overline{\text{IRQ}}$[2] is not available on the 208 package. Not all GPIO pins are available on all packages. See Chapter 2, "Signals" for a list of available IRQ and GPIO signals.
	Section 6.4.1.2 Reset Status Register (SIU_RSR): Changed numbered list to bulleted list.
	Table 6-7 Reset Source Priorities: Added 'Lowest-high' to the priority column of row 2 'Software system reset (Group1) Added 'Highest-low' to the priority column of row 3 'Loss of clock, loss of lock, watchdog, checkstop (Group2).
	Figure 6-3 Reset Status Register (SIU_RSR): Rewrote footnote 3 to read: The reset value of the BOOTCFG field is determined by the values on the BOOTCFG[0:1] pins at reset. BOOTCFG[0] is not available due to pin limitations, and is internally asserted (driven to 0) in the 208 package.
	Table 6-8 SIU_RSR Field Descriptions: BOOTCFG row: added, NOTE BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0) in the 208 package.
	Section 6.4.1.4 External Interrupt Status Register (SIU_EISR): Changed $\overline{\text{IRQ}}$[1]–$\overline{\text{IRQ}}$[15] Added page footnote that reads: $\overline{\text{IRQ}}$[6] is not available in this device. $\overline{\text{IRQ}}$[2] is not available on the 208 package due to pin limitations.
	Section 6.4.1.6 DMA/Interrupt Request Select Register (SIU_DIRSR): Added page footnote that reads: $\overline{\text{IRQ}}$[2] is not available on the 208 package.
	Section 6.4.1.11 IRQ Digital Filter Register (SIU_IDFR): Added NOTE $\overline{\text{IRQ}}$[6] is not available in this device. $\overline{\text{IRQ}}$[2] is not available on the 208 package due to pin limitations.
	Section 6.4.1.12 Pad Configuration Registers (SIU_PCR): Added NOTE $\overline{\text{IRQ}}$[6] is not available in this device. $\overline{\text{IRQ}}$[2] is not available on the 208 package due to pin limitations.
	Sections 6.4.1.10–6.4.1.11 SIU_PCR72 and SIU_PCR73: Removed. Not available on the 324 or the 208 packages.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 6, "System Integration Unit (SIU)" *(continued)*	Section 6.4.1.11 IRQ Digital Filter Register (SIU_IDFR): Table 6-17: DFL field 28-31: Changed From For a 100-MHz system clock, this gives a range of 20ns to 328µs. The minimum time of three clocks accounts for synchronization of the IRQ input pins with the system clock. To For a 82-MHz system clock, this gives a range of 24ns to 400µs. The minimum time of three clocks accounts for synchronization of the IRQ input pins with the system clock.
Chapter 7, "Crossbar Switch (XBAR)"	Updated XBAR_SGPCRn Field Descriptions table.
	Changed wording of reserved fields in registers from: "Reserved" to Reserved, must be cleared."
	Added footnote 2 to register figures 5-3 and 5-4 for the SP0 and TP0 bits: The SP0 and TP0 bits default values are always used, even though the bits are writeable. Added Notes to table 5-5 for the SP0 bit: **Note:** For PBRIDGE_A_PACR0 and PBRIDGE_B_PACR0, you must have supervisor privileges to access PBRIDGE registers. **Note:** Even though the SP0 bit (1) is writeable, the reset value for SP0 is always used. Added Notes to table 5-5 for the TP0 bit: **Note:** For PBRIDGE_A_PACR0 and PBRIDGE_B_PACR0, you must have trusted master privileges to access PBRIDGE registers. Even though the TP0 bit (1) is writeable, the reset value for TP0 is always used.
	Section 5.4 Functional Description: Changed "64-bit accesses" to "64-bit instruction accesses."
	Figure 5-3. Peripheral Access Control Registers and figure 5-4 Off-Platform Peripheral Access Control Registers. Changed bits 0, 1, and 3 to read only. Added footnote 2: to bits 1 and 3 that reads: 'The default values are always used for the SP0 and TP0 bits, even though the bits are writeable.'
	Added paragraph at the end of the chapter that reads: PBRIDGE also supports buffered writes, allowing write accesses to be terminated on the system bus in a single clock cycle, and then subsequently performed on the slave interface. Write buffering is controllable on a per-peripheral basis. The PBRIDGE implements a two-entry 32-bit write buffer.
Chapter 8, "Error Correction Status Module (ECSM)"	Added footnote numeral 1 to the RESET rows of all figures that have uninitialized values upon reset.
	Section 8.3 Initialization and Application Information expanded to include exception processing.
	Removed the 8.1 Introduction level-head making the introductory paragraph follow the Chapter tag. 'Overview' became section 8.1. Removed the bullet structure for 8.1.1 Features. Section 8.2 Memory Map and Register Definition, Table 8-1 ECSM Memory Map: changed e200z6 to e200z3 in footnote 1.
	Section 8.1 Overview: Rewrote in its entirety. • Added details on Read, 32-bit Write, and Eight- and 16-bit Writes. • Defined ECC error types: correctable and non-correctable • Clarified that SRAM ECC operates on 32 data bits plus seven check bits; Flash ECC operates on 64 data bits plus eight check bits

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 9, "Enhanced Direct Memory Access (eDMA)	In the section on DMA Performance, changed: • FROM: removed eDMA Peak Transfer Rate table • TO: Added an eDMA Peak Transfer Rates table (Table 9-20) with columns that show the effect of buffering enabled and disabled.
	Removed the Note: referring to bit 2 in the following tables: EDMA_SERQR, EDMA_CERQR, EDMA_SEEIR, EDMA_CEEIR, EDMA_CER, EDMA_SSBR, EDMA_CDSBR.
	Section 9.4.2. DMA Programming Errors: In the numbered list, changed steps 2 and 3 to subbullets of step 1. Renumbered steps, which resulted in made the last step 5. Changed the step reference in Step 5 to read 'Repeat step 4 until . . .'
	Section 9.4.5.1 Signal Request: capitalized all alpha characters in hexadecimal addresses.
Chapter 10, "Interrupt Controller (INTC)"	Changed INTC_ACKR to INTC_IACKR
	Put overbars on IRQs in table, "External Interrupt Signals"
	Added interrupt vectors 152 (FLEXCAN_A_ESR_BOFF_INT) and 173 (FLEXCAN_C_ESR_BOFF_INT).
Chapter 11, "Frequency Modulated Phase Locked Loop and System Clocks (FMPLL)"	Section Programming System Clock Frequency Without Frequency Modulation: • Moved the following paraphrase at the end of step 4 to a second sentence in the last bullet of step 2C: • *RFD must be set to greater than one to protect from overshoot*.
	Removed all references to PLLCFG[2].
	11.1.4 FMPLL Operating Modes: Removed most of the first paragraph, combined tables 11-1 and 11-2, added footnote 2 that reads: Because the 208 package has no $\overline{\text{RSTCFG}}$ pin, the signal is internally asserted (driven to 0), therefore the PLLCFG pins are always used to configure the FMPLL. After the device resets, the PLLCFG values remain the same as before the reset. The device does not reset to the crystal reference mode. Bypass mode is not enabled in the 208 package.
	Removed 'See FMPLL_SYNCR[PREDIV].' directly before section 11.1.4.2 External Reference Mode.
	Table 11-3. PLL External Pin Interface: Added the $\overline{\text{RSTCFG}}$ pin, and added a footnote that reads: The 208 package does not have a RSTCFG pin, therefore the signal is internally asserted (driven to 0).
	Table 11-5. FMPLL_SYNCR Field Descriptions: PREDIV field, changed 4–20 MHz to 4 MHz; LOLRE field, added system before 'integration module' throughout.
	Section 11.4.1 Clock Architecture: moved the following sentence to section 11.4.1.1 Software Controlled Power Management Clock Gating; 'The peripheral IP modules are designed to let software gate the clocks to the non-memory-mapped logic of the modules. Spelled out Nexus Port Controller (NPC) Removed H7FA.
	Section 11.4.3.1 Programming System Clock Frequency without Frequency Modulation: changed the NOTE format to regular text.
	Table title changed from "Clock Mode Selection in 416 Pin and 324 Pin (if available) Packages" to "Clock Mode Selection in 416 and 324 (with or without the 496 assembly)"
	Added a 2nd row for crystal reference clock mode in Clock Mode Selection tables.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 11, "Frequency Modulated Phase Locked Loop and System Clocks (FMPLL)" *(continued)*	Added sentence: "Bypass mode is not enabled in the 208 package."
	Changed sentence from "The engineering clock (ENGCLK) divider can be programmed to divide the system clock by factors from 2 to 128 in increments of two" to "The engineering clock (ENGCLK) divider can be programmed to divide the system clock by factors from 2 to 126 in increments of two."
	Modified the LOCK bit to read: PLL lock status bit. Indicates whether the FMPLL has acquired lock. If the LOCK bit is read when the FMPLL simultaneously loses lock or acquires lock, the bit does not reflect the current condition of the FMPLL. If operating in bypass mode, LOCK remains cleared after reset. Refer to the frequency as defined in the *MPC5534 Microcontroller Data Sheet* for the lock/unlock range. 0 PLL is unlocked. 1 PLL is locked.
	Clarified that PLLCFG2 is present but must be tied to ground in devices that have only 8-20 MHz range.
	In Section "Reduced Frequency Divider (RFD)": From: "The RFD must be programmed to be ≥1 when changing MFD or PREDIV or when enabling frequency modulation" To: "To protect the system from frequency overshoot during the PLL lock detect phase, the RFD must be programmed to be ≥1 when changing MFD or PREDIV or when enabling frequency modulation."
	Made changes in LOCKS bit: From "a write to the FMPLL_SYNCR which modifies the MFD bits" To: "a write to the FMPLL_SYNCR which modifies the MFD and PREDIV bits"
	In section "Alternate/Backup Clock Selection" changed sentence: From: "Note that when the FMPLL is operated in SCM the system frequency is dependent upon the value in RFD[0:2]." To: "Note that when the FMPLL is operated in SCM, writes to FMPLL_SYNCR[RFD] have no effect on clock frequency."
	Added the symbols $F_{ref\_crystal}$ and F_{ref_ext} to the diagrams, and throughout the manual added further explanation of the fact that F_{prediv} is the frequency after the predivider.
	Modified table "Input Clock Frequency" by adding a column of frequency symbols.
	Modified table "Clock Out vs. Clock In Relationships" by changing F_{ref} symbols to $F_{ref\_crystal}$ and F_{ref_ext}.
	Changed the range 16 -40 MHz to simply 40 MHz.
	Modified the crystal oscillator network figure to show resistor on the XTAL signal.
Chapter 12, "External Bus Interface (EBI)"	Updated block diagram to reflect signals available on MPC5534. Added calibration signals to block diagram.
	In "Features," changed Burst Support note to say, "The MPC5534 does not have any cache, therefore its core never generates a burst transfer to the EBI; therefore, only the DMA can cause a burst transfer to occur."
	In "Burst Support (Wrapped Only)," removed first NOTE under section heading.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 12, "External Bus Interface (EBI)" *(continued)*	In "EBI Base Registers 0–3 (EBI_BRn) and EBI Calibration Base Registers 0–3 (EBI_CAL_BRn)", Updated the BL bit description.
	In "Back-to-Back Accesses," updated text, added NOTE to intro text, added new figure, "Read After Write to the Same CS Bank."
	Removed 1 as a bus speed mode, left 1/2 and 1/4.
	In "Basic Transfer Protocol" changed: From: "To facilitate asynchronous write support, the EBI keeps driving valid write data on the data bus until 1 clock after the rising edge where RD_WR and WE are negated (for chip select accesses only)." To: "To facilitate asynchronous write support, the EBI keeps driving valid write data on the data bus until 1 clock after the rising edge where RD_WR (and WE for chip select accesses) are negated."
	Changed the name of table from "Signal Function by Mode" to "Signal Function According to EBI Mode"
	In table, "Signal Function According to EBI Mode," added the following footnote: "All I/O signals are three-stated by the EBI when not actively involved in a transfer." Added a NOTE below the table.
	To "Calibration Signals," added: "During a calibration bus access, the non-calibration bus signals (other than DATA) are held in a negated state, with the exception of RD_$\overline{WR}$ and ADDR, which reflect the same values shown on the calibration version of those signals. This is harmless because $\overline{TS}$ and $\overline{CS}$ are held negated on the non-calibration bus during calibration accesses, so no transfer takes place on the non-calibration bus. DATA is not driven by the EBI during calibration accesses. During a non-calibration bus access, the calibration bus signals (other than CAL_DATA) are held in a negated state. CAL_DATA is not driven during non-calibration accesses."
	To "Back-to-Back Accesses," added the note: "In some cases, CS remains asserted during this dead cycle, such as the cases of back-to-back writes or read-after-write to the same chip-select.
	In section "Basic Transfer Protocol," changed From: "To facilitate asynchronous write support, the EBI keeps driving valid write data on the data bus until 1 clock after the rising edge where RD_WR and WE are negated (for chip select accesses only)." To: "To facilitate asynchronous write support, the EBI keeps driving valid write data on the data bus until 1 clock after the rising edge where RD_WR (and WE for chip select accesses) are negated."
	To "External Master Mode," added: "Use the SIZEN and SIZE fields of the EBI_MCR must be used for MCU-to-MCU transfers to indicate transfer size"
	Throughout chapter, added text to indicate if a signal is only available via VertiCal.
	"External Signal Description": Moved the following text from this section to "Calibration Signals": DATA is not driven by the EBI during a calibration bus access. During a calibration bus access, the non-calibration bus signals (other than DATA) are held in a negated state, with the exception of RD_$\overline{WR}$ and ADDR, which reflect the same values shown on the calibration version of those signals. Because the $\overline{TS}$ and $\overline{CS}$ signals are held negated on the EBI (non-calibration bus) during calibration accesses, no transfer occurs on the EBI. During a EBI bus access, the calibration bus signals (other than CAL_DATA) are held in a negated state. CAL_DATA is not driven during non-calibration accesses.
	Updated table, "Signal Function According to EBI Mode Settings"

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 12, "External Bus Interface (EBI)" *(continued)*	Section 12.1 Introduction: Added a NOTE that reads: The 208 package does not have an external bus interface. This chapter pertains only to the 324 package.
	Section 12.1.4.1 Single Master Mode: Removed the NOTE because the 324 has not arbitration pins,
	Section 12.4.1.2 32-bit Data Bus: Removed this section. This device has a 16-bit bus.
	Section 12.4.1.1 32-bit Address Bus: Removed the NOTE. Removed references to TSIZ pins through the chapter. Removed references to the arbitration pins ($\overline{BB}$, $\overline{BR}$, $\overline{BG}$) throughout the chapter.
	Section 12.2.1 Basic Transfer Protocol: Figure 12-8 Basic Transfer Protocol: removed Arbitration section of drawing.
	Section 12.1.4.2 External Master Mode: Rewrote 1st sentence in the second paragraph: Dual-master operation (multiple masters initiating external bus cycles) is not supported.
	Removed references to the external arbitration pins (BB, BR, BG) throughout the chapter. Removed all references to TSIZ pins throughout the chapter, including all the timing operation figures.
	Section 12.4.2.1 External Clocking: Added '208 Package There is no CLKOUT pin on this package.'
	Section 12.4.2.7 Size, Alignment, and Packaging Transfers: Removed parenthetical reference to the TEA pin.
	Figure 12-24. Removed 32-bit section of drawing,
	Section 12.5.3 Using Asynchronous Memory: Rewrote 2nd sentence to read: Asynchronous memories do not support bursting, and do not require the CLKOUT, TS, and BDIP pins.
	Section 12.5.6 Summary of Differences from the MCP500: Changed the following: • Second bullet changed to 'No memory controller support for external masters: no support for multi-master system to drive its own chip selects' • Removed 3rd subbullet of 3rd bullet: Modified TSIZ[0:1] functionality to indicate size of the current transfer, not given information on ensuing transfers that may be part of the same atomic sequence • 5th bullet: changed e200z6 to e200z3. • Removed 8th bullet: Open drain mode and pull-up resistors no longer required for multi-master systems, extra cycle needed to switch between masters • Removed 9th bullet: Modified arbitration protocol to require extra cycles when switching between masters • Removed 16th bullet: Address decoding for external master accesses uses 4-bit code to determine the internal slave instead of the straight address decode
	Sections 12.5.5.3 to 12.5.5.6: removed reference to absent pins in the headings. These signals were never supported for the MPC553X family of devices.
	Section 12.4.1.14 Optional Automatic CLKOUT Gating: Removed the following note: NOTE This feature must be disabled for multi-master systems. In those cases, one master is getting its clock source from the other master and needs the other master to stay valid continuously.
	Tables 12-1 and 12-2: Removed tables and formatted into sub-bullets. Removed 208 package information.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 12, "External Bus Interface (EBI)" *(continued)*	Chapter Title, 12.2 External Signal Definition, 12.3 Memory Map and Register Description, 12.4 Functional Description and 12.5 Applications and Initialization: Added: NOTE The 208 package does not have an external bus interface. This chapter pertains to the 324 and 496 packages only. Deleted individual references to 208 differences throughout the chapter.
	Figure 12-1 Block Diagram: Removed individual ADD[8:11] signal; added _ADDR[8] to the CS[0] signal; added _ADDR[9:11] to the CS[1:3] signals to clarify that the 324 package does not have separate pins for ADDR[8:11].
	Sections 12.4.1.7 Port Size Configuration per Chip Select (16 or 32 Bits) and 12.4.1.8 Port Size Configuration per Calibration Chip Select (16 or 32 Bits): removed references to 32-bit port size. This device only supports 16-bit port size. Removed (16 bits) from the titles.
	Sections 12.4.1.14 Configurable Bus Speed Clock Modes and 12.4.1.15 Stop and Module Disable Modes for Power Savings. Removed both sections since they contain cross-references only. There is no power savings mode in the MPC553X devices.
	Section 12.4.2.3 Basic Transfer Protocol: Added to 5th paragraph: For chip select accesses, use the OE signal to indicate that the external device can drive data onto the bus during an MCU read cycle. To prevent bus contentions for chip select accesses, the output enable signal ($\overline{OE}$) must be used to determine when the external device can drive the bus.
	Figure 12-4. EBI Bus Monitor Control Register (EBI_BMCR): Added [0:7] after the field name BMT to completely clarify the field and bit range.
	Combined section 12.1.2 Overview into 12.1 Introduction.
	Section 12.1.2 Features: Grouped calibration features
Chapter 13, "Flash Memory"	Changed LLOCK description From: "Low address block lock. These bits have the same description and attributes as MLOCK. As an example of how the LLOCK bits are used, if a configuration has sixteen 16-KB blocks in the low address space (MCR-LAS = 3'b011), the block residing at address array base + 0, corresponds to LLOCK0. The next 16-KB block corresponds to LLOCK1, and so on up to LLOCK15." To: "Low address block lock. These bits have the same description and attributes as MLOCK. As an example of how the LLOCK bits are used, if a configuration has six 16-KB blocks in the low address space, the block residing at address array base + 0, corresponds to LLOCK0. The next 16-KB block corresponds to LLOCK1, and so on up to LLOCK5."
	Changed bullet in features list From: "Page program of 1 to 84 consecutive 32-bit words within a page (recommended minimum is 2 words due to ECC) To: "Page program size of 128 bits allows programming from one to two consecutive 64-bit doublewords within a page."
	In Flash Partitions table, corrected size value for Array Base + 0x00FF_FE04 – 0x00FF_FFFF From: 516 To: 508
	Corrected reset values in FLASH_BIUAPR (changed 0's to 1's).
	Table 13-17 FLASH_BU_CR2 Field Description and the paragraph directly succeeding the table: Corrected the word location in the flash array used to change the values loaded at reset from 0x7e00 shadow block address to 0x0200 + the shadow base address.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 13, "Flash Memory"	Figure 13-1. Flash System Block Diagram: Added footnote 1 that reads, 'V_{FLASH} is not available on the 208 package.'
	Section 13.1.2 Overview: 2nd paragraph, fourth sentence, added 'in a single cycle' to read: Normal flash array accesses (accesses that don't go to the prefetch buffers) are registered in the FBIU in a single cycle, and are forwarded to the system bus on the next cycle, incurring at least two wait states (depending on the frequency).
	Sections 13.4.2.6 Censorship and 13.4.6.4 External Boot Default: Added, '208 Package: BOOTCFG[0] and $\overline{\text{RSTCFG}}$ are not available due to pin limitations. This signal is internally asserted (driven to 0) therefore, the device defaults to internal flash on the 208 package.'
Chapter 14, "Internal Static RAM (SRAM)"	Majority of chapter re-organized.
	Corrected wait states in table, "Number of Wait States Required for SRAM Operation (continued)."
	Removed "multi-bit" error correction.
	Table 14-3 Number of Wait States Required for RAM Operation: Changed the '/' to 'or'
	Table 14-1. SRAM Operating Modes: Standby description: Added V_{DD} and (0.8–1.2 V) to the first sentence to read, 'Preserves the 32 KB of standby memory when the V_{DD} (1.5 V) power drops below the level of V_{STBY} (0.8–1.2 V). Updates to standby SRAM are inhibited during system reset or during standby mode.
	Section 14.7.1 Example Code: Added MPC5534 example code.
	Section 14.6 SRAM ECC Mechanism: throughout section changed 8-bit ECC to 7-bit ECC.
Chapter 15, "Boot Assist Module (BAM)"	Update Overview section From: The BAM contains the MCU boot program code, identical for all eSys MCUs with an e200z3 core. The BAM control block is connected to peripheral bridge B and occupies the last 16 KBs of the MCU memory space. The BAM program supports four different booting modes: from internal Flash, from external memory without bus arbitration, serial boot via SCI or CAN interfaces. The BAM program is executed by the e200z3 core just after the MCU reset. Depending on the boot mode, the program initializes appropriate minimum MCU resources to start user code execution. To: The BAM contains the MCU boot program code. The BAM control block is connected to peripheral bridge B and occupies the last 16 KB of the MCU memory space. The BAM program supports several booting modes: - Internal flash - External memory without bus arbitration - Serial boot using an eSCI interface - FlexCAN interfaces The BAM program is executed by the e200z3 core just after the MCU reset. Depending on the boot mode, the program initializes the minimum MCU resources to start application code execution.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 15, "Boot Assist Module (BAM)" *(continued)*	Added the following to "Serial Boot Mode" section: Serial boot mode downloads: 64-bit password 32-bit start address 32-bit download consisting of 1-bit VLE flag (most significant bit) followed by a 31-bit length field containing the number of bytes to receive (download length) Set the VLE flag to 1 for devices that support variable length encoding and must run in VLE mode. When the VLE flag is set, the BAM programs the external bus interface (EBI), RAM, and the flash memory map unit (MMU) TLB entries 1, 2, and 3 with the VLE attribute. Clear the VLE bit to 0 for devices that use the PowerPC Book E or Power Architecture instruction set mode.
	In "BAM Program Operation" section, changed From: "Then the BAM program reads the status of the two BOOTCFG pins from the reset status register (SIU_RSR) and the appropriate boot sequence is started as shown in the Table 15-3. Depending on the values stored in the censorship word and serial boot control word in the shadow block of internal Flash memory, the internal Flash memory can be enabled or disabled, the Nexus port can be enabled or disabled, the password received in serial boot mode is compared with a fixed public password or compared to a user programmable password in the internal Flash memory." To: "The BAM program reads the following data and determines the boot mode for the boot sequence: - BOOTCFG[0:1] located in the reset status register (SIU_RSR) - Censorship control field located at 0x00FF_FDE0 in the shadow block of internal flash - Serial boot control field located at 0x00FF_FDE2 in the shadow block of internal flash The boot mode determines the following: - Enables or disables internal flash memory - Enables or disables the Nexus port - Compares the password received in serial boot mode to a preset public password or a programmable password located in internal flash"
	Table 15-3: 'MMU Configuration for Internal Flash Boot' and Table 15-6:'MMU Configuration for External Boot (continued) Mode': Removed the reference to cache because the MPC553X devices do not have cache.
	Table 15-3: 'MMU Configuration for Internal Flash Boot' and Table 15-6:'MMU Configuration for External Boot (continued) Mode': Corrected the physical base address for the EBI from 0x0000_0000 to 0x2000_0000.
	Throughout this chapter, added 208 package information that BOOTCFG is not available and is internally asserted (Driven to 0).
	Section 15.1.3.5 Serial Boot Mode: Added serial boot mode downloads and VLE instructions.
	Section 15.1.1 Overview • Bulleted BAM modes • Deleted 'eSys,' • Added to bullet 'External memory without bus arbitration:' '(324 package only; not available on the 208 package)
	Added 'For devices using the 208 package, the BAM determines the value of BOOTCFG[1] only, since BOOTCFG[0] is not available due to pin limitations and is internally asserted (driven to 0).'

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 15, "Boot Assist Module (BAM)" *(continued)*	Section 15.3.2. BAM Program Operation, Table 15-2 MMU Configuration for Internal Flash Boot: removed reference to Cache memory in the Attributes column. No Cache available in the MPC553X family.
	Section 15.3.2. BAM Program Operation, Figure 15-2 Censorship Word, updated register format.
	Table 15-5 MMU Configuration for External Flash Boot: removed reference to Cache memory in the Attributes column. No Cache available in the MPC553X family.
	Section 15.3.2.4.2. Serial Boot Mode FlexCAN and eSCI Configuration: replaced 'All data received is assumed to be good and is echoed. . .' with 'All data received is accepted as valid and is echoed out on the TXD signal.
Chapter 16, "Enhanced Modular Input/Output Subsystem (eMIOS)"	Section 16.4.3 Global clock prescaler submodule: deleted 'The output is clocked every time the counter overflows.'
	Section 16.1.4.1 eMIOS Modes: added to the end of Freeze mode: 'In freeze mode, all clocks are running and all registers are accessible for use debugging software; there is no power saving during freeze mode.
	Table 16-11 EMIOS_CSRn Field Descriptions: Shaded reserved row (bits 17–28).
	Section 16.4.4.4.11 Modulus Counter Mode (MC) Changed table column label from 'Unified Channel Mode of Operation' to Modulus Counter Operating Modes. Bulleted the operating modes in that column.
Chapter 17, "Enhanced Time Processing Unit (eTPU)"	Section 17.5.2.1.4 eTPU SCM Off-Range Data Register (ETPU_SCMOFFDATAR): Deleted 'This register is global to both ETPU engines.
Chapter 18, "Enhanced Queued Analog-to-Digital Converter (eQADC)"	Section Overview: From: It also monitors the fullness of CFIFOs and RFIFOs, which may result in either underflow or overflow conditions. A CFIFO underflow occurs when the CFIFO is in the TRIGGERED state and it becomes empty. An RFIFO overflow occurs when an RFIFO is full and more data is ready to be moved to the RFIFO by the host CPU or by eDMA. Accordingly, the eQADC generates eDMA or interrupt requests to control data movement between the FIFOs and the system memory, which is external to the eQADC. To: It also monitors the amount of memory currently in use by each the CFIFO and RFIFO to detect underflow and overflow conditions. A CFIFO underflow occurs when a CFIFO: • Is in the TRIGGERED state; and • Becomes empty. An RFIFO overflow occurs when an RFIFO: • Becomes full; and the • Host CPU or eDMA data is waiting to transmit to the RFIFO. The eQADC generates eDMA or interrupt requests to control data movement between the FIFOs and the system memory, which is external to the eQADC.
	Section Feature Overview • From: When commands of distinct CFIFOs are bound for the same ADC, the higher priority CFIFO is always served first. • To: When commands from different CFIFOs are sent to the same ADC, the higher priority CFIFO is always served first.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 18, "Enhanced Queued Analog-to-Digital Converter (eQADC)" *(continued)*	Section Feature Overview • From: Supports four external 8-to-1 muxes that can expand the input channel number from 40 to 68 • To: Supports four external 8-to-1 muxes that can expand the number of input channels from 40 to 65.
	Section eQADC Memory Map Added footnote 1 to the eQADC result FIFOs in the eQADC Memory Map: Result FIFOs are 16-bits wide [0:15]; bits [16:31] are filled with zeros to allow for 32-bit read access.
	Section eQADC null message send format register. • From: The eQADC null message send format register only affects how the eQADC sends a null message, but it has no control on how the eQADC detects a null • To: The eQADC null message send format register (eQADC_NMSFR) only defines the format of the null message that eQADC sends. The null message register does not control how the eQADC detects a null message from the input source.
	Section eQADC FIFO POP Registers 0–5: Changed Note • From: The EQADC_RFPR*n* must not be read speculatively. For future compatibility, the TLB entry covering the EQADC_RFPR*n* must be configured to be guarded. • To: Do not read the EQADC_RFPR*n* unless absolutely necessary, since the data is lost when the read occurs. For future compatibility, configure the TLB entry for the EQADC_RFPR*n* registers as guarded.
	Section Functional Description • From: The two on-chip ADCs are architected to allow access to all the analog channels. • To: The two on-chip ADCs can access all the analog channels.
	Section Message Format in eQADC: • From: The FIFO control unit decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result should be sent. An ADC result is always 16 bits long. • To: The FIFO control unit decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result is sent. An ADC result field is always 16 bits.
	Section Conversion Command Message Format for On-Chip ADC Operation • From: The lower byte of conversion commands is always set to 0 to distinguish it from configuration commands. • To: Conversion commands are sent to the ADC internal memory map address zero, therefore the lower byte of conversion commands is always cleared to 0 to distinguish it from configuration commands.
	Removed section "Detailed Signals" from this chapter because this information is contained in the Signals chapter of the Reference Manual.
	Added this cross reference to the EQADC_NMSFR[NMF] bit: "See the section 'Null Message Format for External Device Operation' for more information."
	Table Result Message Format for External Device Operation. In the description field of the ADC_Results [16:31], changed • From: This can be the result. . . • To: This can be any value produced by the external device, such as the. . .

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 18, "Enhanced Queued Analog-to-Digital Converter (eQADC)" *(continued)*	Table. *On-chip ADC Field Descriptions: Conversion Command Message Format.* Changed description of BN field • From: Indicates which ADC the message will be sent to. • To: For internal commands, indicates the ADC to which the message is sent. For external commands, indicates to which command FIFO the messages is sent. • From: '24:31 Reserved' • To: 24:31 ADC_REG_ADDRESS [0:7] ADC register address. Identifies the address of the ADC register. Only use 16-bit (halfword) addresses. See Table 18-22. Include $00 for conversion commands.
	Figure. *Conversion Command Message Format for On-Chip ADC Operation.* Changed name for bits 16:31 from ADC Command to ADC Address.
	Table EQADC_RF[0–5]R*n* Field Descriptions: added reserved row for bits 0:15.
	Figure 18-26 Conversion Command Message Format for On-Chip ADC Operation: Gave bits 24–31 value of 0, and changed the label for bits 16–23 to ADC Command.
	Section 18.4.5.2 ADC Clock and Conversion Speed, 3rd paragraph and Table 18-44:ADC Clock Configuration Example—System Clock Frequency = 120 MHz: Changed the system clock speed from 120 MHz to 72 MHz.
	Section 18.1 Introduction: Added the following NOTE: The 324 package supports 40 channels in the dual eQADC engines. The 208 package supports 34 channels only.
	Table 18-44 ADC Clock Configuration Example—System Clock Frequency = 72 MHz: Updated table with values for 72 MHz.
Chapter 19, "Deserial Serial Peripheral Interface (DSPI)"	Changed the NOTE in the DSPI_PUSHR register: • From: "Only the TXDATA field is used for slaves." • To: "TXDATA is used in master and slave modes."
	Section 19.5.5.2, "Address Calculation for the First-in Entry and Last-in Entry in the RX FIFO" Changed • From: First-in entry address = TX FIFO base + [4 x (POPNXTPTR)] • To: First-in entry address = RX FIFO base + [4 x (POPNXTPTR)]
	Numbered and edited the steps in section 'How to Change Queues." Changed the following section titles: • From: Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 0) • To: Modified Transfer Format Enabled (MTFE = 1) and Classic SPI Transfer Format Cleared (CPHA = 0) for SPI and DSI • From: Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 1) • To: Modified Transfer Format Enabled (MTFE = 1) with Classic SPI Transfer Format Set (CPHA = 1) for SPI and DSI
	Throughout chapter: Changed f_{sys} from 100 MHz to 82 MHz.
	Tables 19-20, 19-22, and 19-23: changed IRQ[6] to reserved.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 20, "Enhanced Serial Communication Interface (eSCI)"	Section 20.1.3 Features: added 13-bit to bullet Configurable baud rate.
	Section 20.3.2.4 ESCI Status Register (ESCIx_SR): Table 20-6. ESCIx_SR Field Descriptions: Added the following: BERR bit 0 No bit error 1 Bit error RXRDY 0 No receive data ready 1 Receive data ready TXRDY 0 ESCIx_LTR is not free 1 ESCIx_LTR is free LWAKE 0 LIN2.0 wakeup signal not received 1 LIN2.0 wakeup signal received STO 0 No time out detected 1 A slave did not complete a frame within the specified maximum frame length PBERR 0 No error 1 Physical bus error CKERR 0 No error 1 Checksum error FRC 0 Frame not complete 1 Frame complete OVFL 0 No overflow 1 Overflow detected
	Section 20.3.2.5 LIN Control Register (ESCIx_LCR): Table 20-7. ESCIx_LCR Field Descriptions: LIN bit 7, added 'When LIN is enabled, even if parity generation/checking is enabled via ESCIx_CR[PE], the parity bit is not masked out.'
	Section 20.4.5.5.1 Slow Data Tolerance: changed '9 bit times x 16 RT cycles = 147 RT cycles' to '9 bit times x 16 RT cycles = 144 RT cycles.'
Chapter 21, "FlexCAN2 Controller Area Network"	No changes.
Chapter 22, "Voltage Regulator Controller (VRC) and POR Module"	Table 22-2 Values after POR Negation: Replaced table with Tables 11-1 and 11-2 from the FMPLL chapter, becoming tables 22-2 Clock Mode Selection and 22-3 PLL External Pin Interface.

Table C-1. Changes Between Revisions 1 and 2 (continued)

Chapter	Description
Chapter 23, "IEEE 1149.1 Test Access Port Controller (JTAGC)"	Table 24-1. JTAG Signal Properties: combined the Reset State and Pull columns. Changed TDO pull state to up instead of down. TDO reset state is now High Z / Pullup. Changed footnote 2 to read, 'A weak pullup is implemented on TDO.
	Figure 24-2. 5-bit Instruction Register: Changed bit order from Least Significant Bit (LSB) to Most Significant Bit (MSB).
	Tables 24-3 JTAG Instructions: • Code column label changed to Code[0:4]. • Changed row ACCESS_AUX_TAP_ONCE from e2000z6 (NZ6C3) to e200z3 (NZ3C3). • Added the row ACCESS_AUX_TAP_eTPUN3, 10010, Grants the Nexus eTPU development interface (NSEDI) ownership of the TAP • Added 10011 to the first reserved row at the end of the table. Added 'Do not use these settings.' to the second reserved row at the end of the table. Changed Decoded to select bypass register' to 'Defaults to bypass register.'
Chapter 24, "Nexus Development Interface"	Section 24.2.1.3 Message Data Output (MDO[3:0] or [11:4]): Added '208 Package: MDO[11:4] pins are not available due to pin limitations.'
	Changed e200z6 to e200z3 throughout the chapter. Changed NZ6C3 to NZ3C3 throughout the chapter.
Appendix A, "MPC5534 Register Map"	Signals not available on the 208 package appear in **red** text. Primary signals not designed into this device appear in **blue** text.
Appendix B, "Calibration"	Added footnote to the following signals in the Calibration Bus Signals table: $\overline{WE}$/$\overline{BE}$[2,3] ADDR[8:11] $\overline{TEA}$ The footnote reads: Not available on the 324 package.
	Removed references to the 416 package since it is not available for MPC5534.
	Added NOTE at the beginning of the chapter that reads: NOTE In addition to the MPC5534 device, the 496 VertiCal assembly is required to use the calibration features. The External Bus Interface (EBI) and Calibration Bus Interface (CBI) are not available due to pin limitations in the 208 package.