RM0380
Reference manual

STLUX and STNRG digital controllers designed for lighting and power conversion

Introduction

This reference manual provides complete information for application developers on how to use the STLUX and STNRG family of digital controllers.

The STLUX™ family of controllers is a part of the STMicroelectronics® digital devices tailored for lighting applications. The STLUX controllers have been successfully integrated in a wide range of architectures and applications, starting from simple buck converters for driving multiple LED strings, boost for power factor corrections, half-bridge resonant converters for high power dimmable LED strings and up to full bridge controllers for HID lamp ballasts. STLUX natively supports the DALI via the internal DALI communication module (DCM). DALI is a serial communication standard used in the lighting industry.

STNRG devices are a part of the STNRG family of STMicroelectronics digital devices designed for advanced power conversion applications. The STNRG improves the design of the STLUX™ family to support industrial power conversion applications such as PFC+LLC, interleaved LC DC-DC, interleaved PFC for smart power supplies as well as the full bridge for pilot line drivers for electric vehicles.

The heart of the STLUX (and consequently STNRG where not differently specified) is the SMED (“State Machine, Event Driven”) technology which allows the device to operate several independently configurable PWM clocks with an up to 1.3 ns resolution. An SMED is a powerful autonomous state machine which is programmed to react to both external and internal events and may evolve without any software intervention. The SMED even reaction time can be as low as 10 ns, giving the STLUX the ability of operating in time critical applications.

The SMEDs are configured and programmed via the STLUX internal low-power microcontroller (STM8). The STM8 controller extends the STLUX reliability and guarantees more than 15 years of both operating lifetime and memory data retention to the program and data memory after cycling. The STM8 device also provides powerful processing while maintaining the advantages of the CISC architecture, a 24-bit linear addressing mode, an 8-bit data bus and an optimized architecture for low power applications.

The STLUX controller 1.8 V core voltage is provided by an internal power supply DC regulator in order to simplify and reduce the application design and cost.

The STLUX device has an embedded low power, low voltage, single voltage Flash program memory designed for reliability. The same technology provides an embedded true RWW data EEPROM dedicated area and dedicated 128 byte data used for the IC configuration.

This manual describes how to program the SMED and the STM8 companion controller.

Refer to the product datasheets for ordering information, pin description, mechanical and electrical device characteristics, and for specific peripherals and implementation details.

Contents

List of tables

List of figures

1 Reference documents

- For hardware information on the STLUX controller and product specific SMED configuration, please refer to the STLUX product datasheets for STLUX385A, STLUX383A, STLUX325A and STLUX285A.
- For hardware information of the STNRG controller and product specific SMED configuration, please refer to the STNRG product datasheets for STNRG388A, STNRG328A and STNRG288A.
- For information on programming, erasing and protection of the internal Flash memory please refer to the programming manual PM0051 - “How to program STM8S and STM8A Flash program memory and data EEPROM”.
- For information about the debug module and SWIM (single wire interface module) refer to the STM8 SWIM communication protocol and debug module user manual (UM0470).
- For information on the STM8 core and assembler instruction please refer to the STM8 CPU programming manual (PM0044).
- For information on the development tools please refer to the UM1792.

2 Acronyms

The following is a list of acronyms used in this document:

Table 1. Acronym descriptions

Acronym	Description
ACU	Analog comparator unit
ADC	Analog-to-digital converter
AWU	Auto-wakeup unit
BL	Bootloader - used to load the user program without the emulator
CKC	Clock control unit
CPU	Central processing unit
CSS	Clock security system
DAC	Digital-to-analog converter
DALI	Digital addressable lighting interface
ECC	Error Correction Code
FSM	Finite state machine
FW	Firmware loaded and running on the CPU
GPIO	General purpose input/output
HSE	High-speed external crystal - ceramic resonator
HSI	High-speed internal RC oscillator

Table 1. Acronym descriptions (continued)

Acronym	Description
I^2C	Inter-integrated circuit interface
IAP	In-application programming
ICP	In-circuit programming
ITC	Interrupt controller
IWDG	Independent watchdog
LSI	Low-speed internal RC oscillator
MCU	Microprocessor central unit
MSC	Miscellaneous
PM	Power management
RFU	Reserved for future use
ROP	Read-out protection
RST	Reset control unit
RTC	Real-time clock
SMED	State machine, event driven
STMR	System timer
SW	Software is the firmware loaded and running on the CPU (synonymous of FW)
SWIM	Single wire interface module
UART	Universal asynchronous receiver transmitter
WWDG	Window watchdog

3 Register model

Table 2 shows an outline picture of the register abbreviations used in this document.

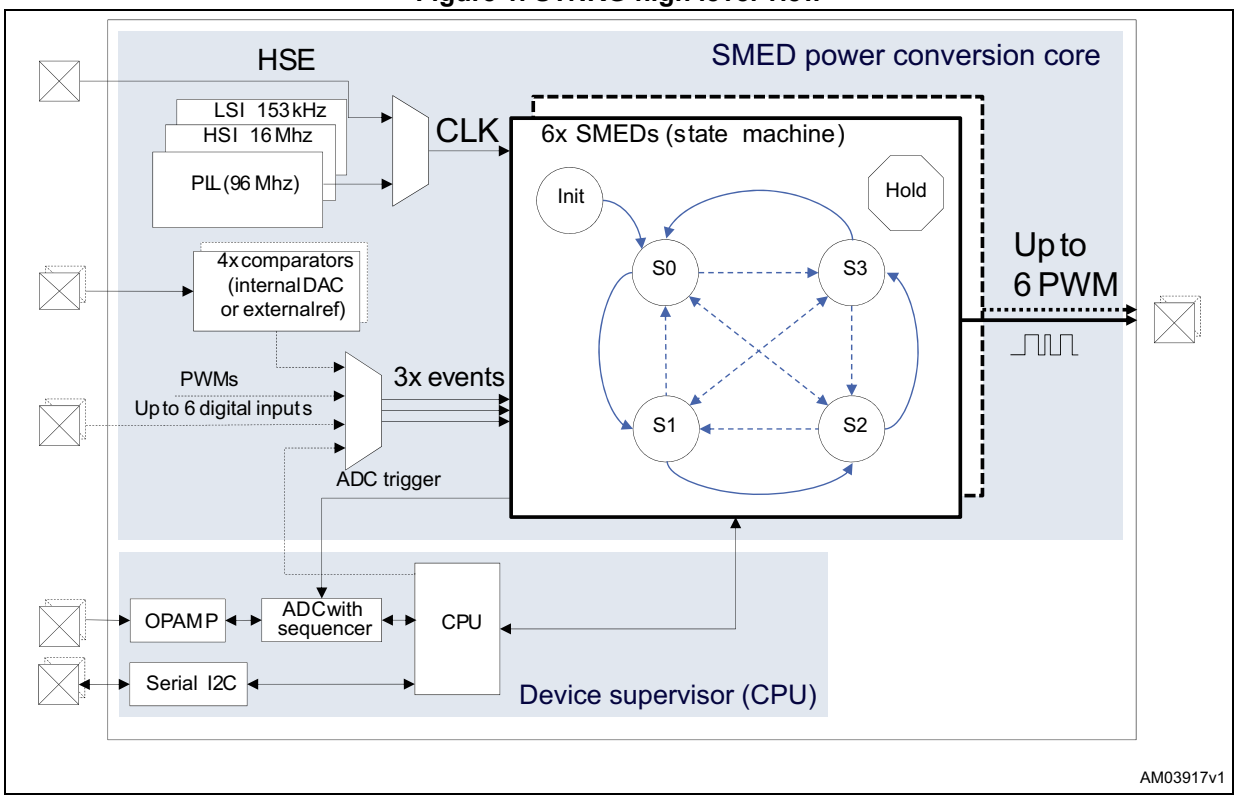
Table 2. Register abbreviations

Abbreviation	Description
read/write (rw) or (r/w)	Software can read and write to these bits.
read only (r, ro)	Software can only read these bits.
write only (w, wo)	Software can only write to this bit. Reading the bit returns a meaningless value.
read/write clear (r/wc)	SW reads and writes clear.
read/write clear (r/w0)	SW reads and writes clear when writing-bit 0.
read/write clear (r/w1)	SW reads and writes clear when writing-bit 1.
read/write once (rwo)	Software can only write once to this bit and can also read it at any time. Only a reset can return the bit to its reset value.
read/clear (rc/w1)	Software can read as well as clear this bit by writing 1. Writing ‘0’ has no effect on the bit value.
read/clear (rc/w0)	Software can read as well as clear this bit by writing 0. Writing ‘1’ has no effect on the bit value.
read/set (rs)	Software can read as well as set this bit. Writing ‘0’ has no effect on the bit value.
read/write0 (r/w0)	Software can read and writing '0'. Writing '1' has to be avoiding for future product compatibility.

4 Description

The STLUX and STNRG device families generate and control PWM signals by means of a state machine, called SMED (state machine event driven). *Figure 1* gives an overview of the internal architecture.

Figure 1. STNRG high level view



The core of the device is the SMED unit: a hardware state machine driven by system events. The SMED includes 4 states (S0, S1, S2 and S3) available during running operations. A special HOLD state is provided as well.

The SMED allows the user to configure, for every state, which system events will trigger a transaction to a new state.

During a transaction from one state to the other, the PWM output signal level can be updated.

Once a SMED is configured and running, it becomes an autonomous unit, so no interaction is required since the SMED automatically reacts to system events.

Thanks to the SMED's 96 MHz operating frequency and their automatic dithering function, the PWM maximum resolution is 1.3 ns.

The device has 6 SMEDs available. Multiple SMEDs can operate independently from each other or they can be grouped together to form a more powerful state machine.

4.1 STLUX/STNRG feature comparison

The following table summarizes the differences between the STLUX and STNRG family of products

Table 3. STLUX/STNRG feature comparison

Feature		Device	
		STLUX	STNRG
Max. pin count		38	
SMED available		6	
Max. SMED PWM output pins		6	
Max. fast digital inputs pins		6	
Max. positive comparator input pin		4	
Max. negative comparator input pins		1	3
Comparator hysteresis		No	Yes
Internal DACs		4	
Max. ADC input pins		8	
Max. ADC gain		x1 – x4	
ADC hardware trigger		No	Yes
Max. GPIO port 0 pins		6	
Communication	UART peripheral	Yes	
	DALI peripheral	Yes	No
	I2C peripheral	Yes	
HSE function		Yes	
Timers	System timer	1	
	Auxiliary timer	1	
	Basic timer	No	2
Auto-wakeup timer		1	
Watchdog	Window watchdog timer	1	
	Independent watchdog timer	1	
Flash program memory		32 Kb	
EEPROM data memory		1 Kb	
RAM		2 Kb	6 Kb

5 Central processing unit (CPU)

5.1 CPU introduction

The CPU is based on an 8-bit CISC architecture optimized for code efficiency and performance. The CPU is able to execute 80 basic instructions. It features 20 addressing modes and can address six internal registers. For the complete description of the instruction set, refer to the STM8 microcontroller family programming manual (PM0044).

5.2 CPU architecture overview

5.2.1 Architecture and registers

- Harvard architecture
- 3-stage pipeline
- 32-bit wide program memory bus - single cycle fetching for most instructions
- X and Y 16-bit index registers - enabling indexed addressing modes with or without offset and read-modify-write type data manipulations
- 8-bit accumulator
- 24-bit program counter - 16-Mbyte linear memory space
- 16-bit stack pointer - access to a 64-K-level stack
- 8-bit condition code register - 7 condition flags for the result of the last instruction

5.2.2 Addressing mode

- 20 addressing modes
- Indexed indirect addressing mode for look-up tables located anywhere in the address space
- Stack pointer relative addressing mode for local variables and parameter passing

5.2.3 Instruction set

- 80 instructions with 2-byte average instruction size
- Standard data movement and logic/arithmetic functions
- 8-bit by 8-bit multiplication
- 16-bit by 8-bit and 16-bit by 16-bit divisions
- Bit manipulation
- Data transfer between stack and accumulator (push/pop) with direct stack access
- Data transfer using the X and Y registers or direct memory-to-memory transfers

5.3 CPU registers description

The six CPU registers are shown in the programming model in *Figure 2*. Following an interrupt, the registers are pushed onto the stack in the order shown in *Figure 2*; they are popped from the stack in the reverse order.

5.3.1 Accumulator (A)

The accumulator is an 8-bit general purpose register used to hold operands and the results of the arithmetic and logic calculations as well as data manipulations.

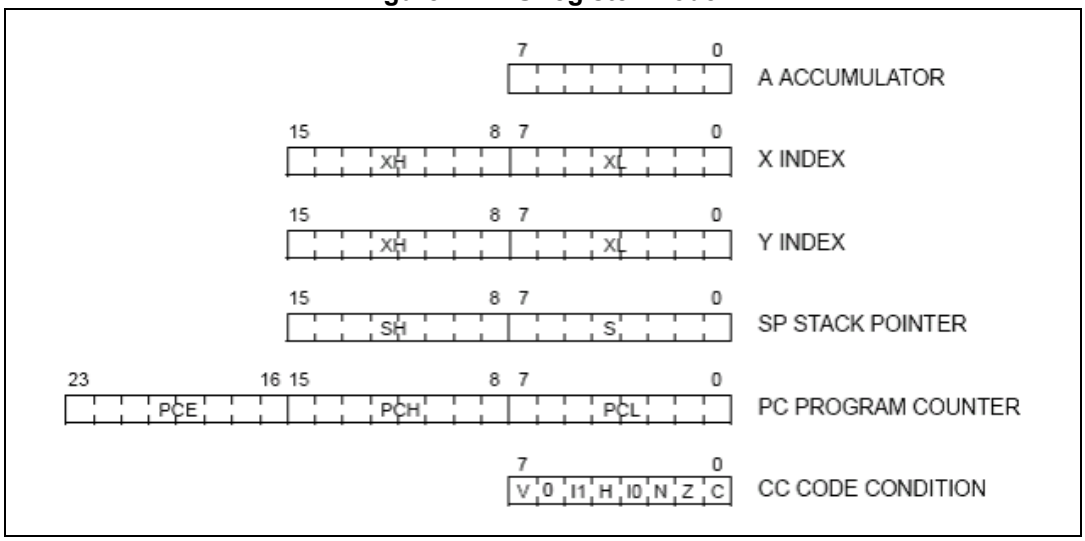
5.3.2 Index registers (X and Y)

These are 16-bit registers used to create effective addresses. They may also be used as a temporary storage area for data manipulations and have an inherent use for some instructions (multiplication/division). In most cases, the cross assembler generates a PRECODE instruction (PRE) to indicate that the following instruction refers to the Y register.

5.3.3 Program counter (PC)

The program counter is a 24-bit register used to store the address of the next instruction to be executed by the CPU. It is automatically refreshed after each processed instruction. As a result, the STM8 core can access up to 16 Mbytes of memory.

Figure 2. CPU register model



5.3.4 Stack pointer (SP)

The stack pointer is a 16-bit register. It contains the address of the next free location of the stack. The register initialization value is detailed in the product datasheet.

The stack is used to save the CPU context of subroutine calls or interrupts. The user can also directly use it through the POP and PUSH instructions.

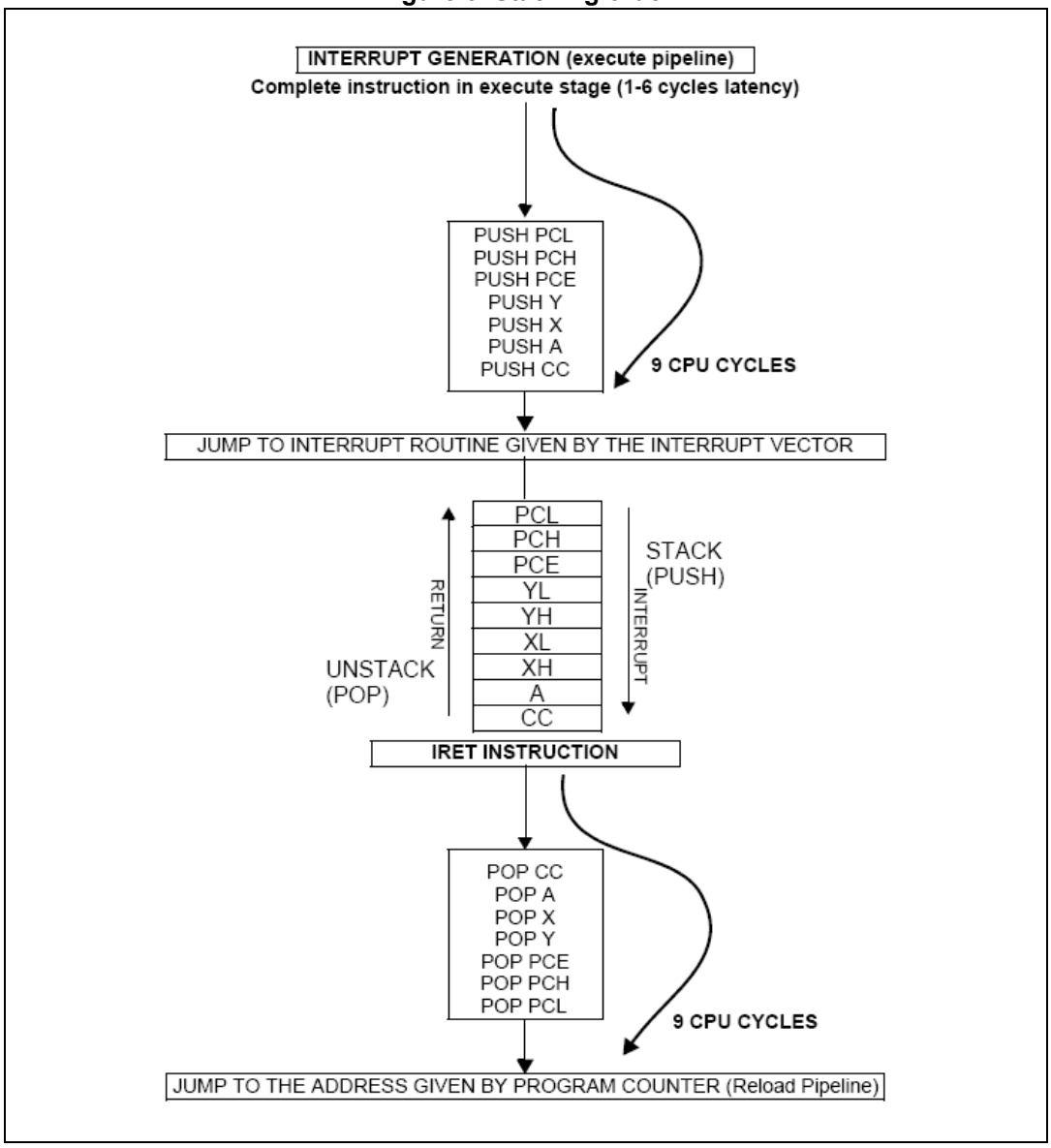
Note: The stack pointer can be initialized by the startup function provided with the C compiler. For applications written in C language, the initialization is then performed according to the address specified in the linker file. If you use your own linker file or the startup file, make sure the stack pointer is initialized properly (with the address given in the product datasheets). For applications written in other languages, be sure to properly initialize the stack pointer.

The stack pointer is decremented after data has been pushed onto the stack and incremented after data are popped from the stack. It is up to the application ensuring that the lower limit is never exceeded.

A subroutine call occupies two or three byte locations. An interrupt occupies nine byte locations to store all the internal registers (except SP). For more details refer to *Figure 3*.

Note: The WFI/HALT instructions save the context in advance. If an interrupt occurs while the CPU is in one of these modes, the latency is reduced.

Figure 3. Stacking order



5.3.5 Condition code register (CC)

The condition code register is an 8-bit register which indicates the result of the instruction just executed as well as the state of the processor. The 6th bit (MSB) of this register is reserved. These bits can be individually tested by a program and specified action taken as a result of their state. The following paragraphs describe each bit:

V: overflow

When set, V indicates that an overflow occurred during the last signed arithmetic operation on the MSB result bit. Refer to the INC, INCW, DEC, DECW, NEG, NEGW, ADD, ADDW, ADC, SUB, SUBW, SBC, CP, and CPW instructions.

I1: interrupt mask level 1

The I1 flag works in conjunction with the I0 flag to define the current interrupt level as shown in *Table 4*. These flags can be set and cleared by software through the RIM, SIM, HALT, WFI, IRET, TRAP, and POP instructions and are automatically set by hardware when CPU enters into an interrupt service routine (ISR).

Table 4. Interrupt priority level

Interrupt	Priority	I1	I0
Interruptible level 0 (main)	Lowest ↓ Highest	1	0
Interruptible level 1		0	1
Interruptible level 2		0	0
Non interruptible level 3		1	1

H: half carry bit

The H bit is set to 1 when a carry occurs between the bits 3 and 4 of the ALU during an ADD or ADC instruction. The H bit is useful in BCD arithmetic subroutines.

I0: interrupt mask level 0

Refer to Flag I1 bit field description.

N: negative

When set to 1, this bit indicates that the result of the last arithmetic, logical or data manipulation is negative (i.e. the most significant bit is at logic 1).

Z: zero

When set to 1, this bit indicates that the result of the last arithmetic, logical or data manipulation is zero.

C: carry

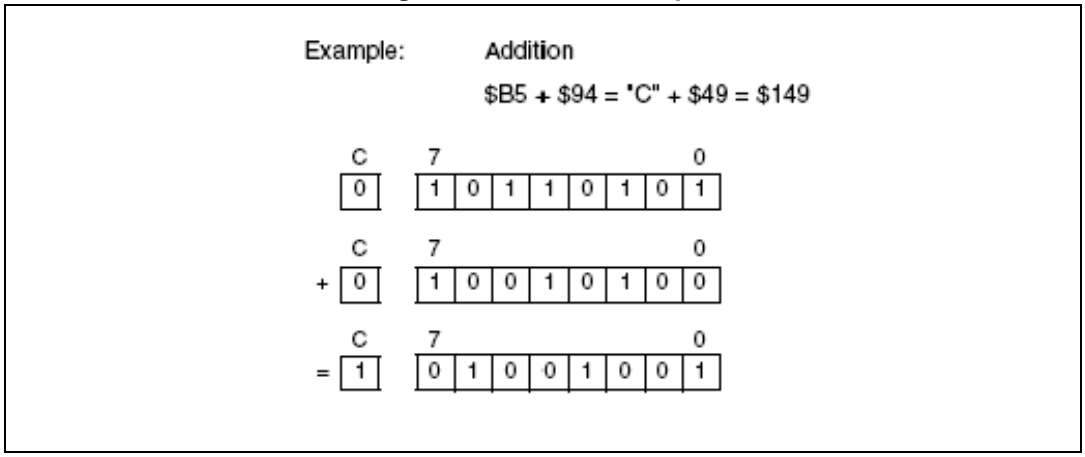
When set, C indicates that a carry or borrow out of the ALU occurred during the last arithmetic operation on the MSB operation result bit. This bit is also affected during bit test, branch, shift, rotate and load instructions. Refer to the ADD, ADC, SUB, and SBC instructions.

In a division operation, C indicates if trouble occurred during execution (quotient overflow or zero division). See the DIV instruction.

In bit test operations, C is the copy of the tested bit. See the BTJF and BTJT instructions. In shift and rotate operations, the carry is updated. Refer to the RRC, RLC, SRL, SLL, and SRA instructions.

This bit can be set, reset or complemented by software using the SCF, RCF, and CCF instructions.

Figure 4. Addition example



5.4 Core registers description

5.4.1 A (accumulator)

Offset: 0x00

Default value: 0x00

7	6	5	4	3	2	1	0
A [7:0]							
r/w							

Bit 7-0: **A[7:0]** accumulator

The accumulator is an 8-bit general purpose register used to hold operands and the results of the arithmetic and logical calculations as well as data manipulations.

5.4.2 PCE (program counter)

Offset: 0x01

Default value: 0x00

7	6	5	4	3	2	1	0
PC [23:16]							
r/w							

Bit 7-0: **PC[23:16]** program counter upper byte

This is the upper byte of the 24-bit register program counter used to store the address of the next instruction to be executed by the CPU. It's automatically refreshed after each processed instruction. As a result, the core can access up to 16-Mbyte of memory.

5.4.3 PCH (program counter)

Offset: 0x02

Default value: 0x60

7	6	5	4	3	2	1	0
PC [15:8]							
r/w							

Bit 7-0: **PC[15:8]** program counter middle byte

This is the middle byte of the 24-bit register program counter.

5.4.4 PCL (program counter)

Offset: 0x03

Default value: 0x00

7	6	5	4	3	2	1	0
PC [7:0]							
r/w							

Bit 7-0: **PC[7:0]** program counter lower byte

This is the lower byte of the 24-bit register program counter.

5.4.5 XH (X-index)

Offset: 0x04

Default value: 0x00

7	6	5	4	3	2	1	0
X[15:8]							
r/w							

Bit 7-0: **X[15:8]** index X upper byte

This is the upper byte of the index register used to create effective addresses or as a temporary storage area for data manipulations.

5.4.6 XL (X-index)

Offset: 0x05

Default value: 0x00

7	6	5	4	3	2	1	0
X[7:0]							
r/w							

Bit 7-0: **X[7:0]** index X lower byte

This is the lower byte of the index register used to create effective addresses or as a temporary storage area for data manipulations.

5.4.7 YH (Y-index)

Offset: 0x06

Default value: 0x00

7	6	5	4	3	2	1	0
Y[15:8]							
r/w							

Bit 7-0: **Y[15:8]** index Y upper byte

This is the upper byte of the index register used to create effective addresses or as a temporary storage area for data manipulations. In most of the cases the cross assembler generates a PRECODE instruction to indicate that the following instruction refers to the Y register.

5.4.8 YL (Y-index)

Offset: 0x07

Default value: 0x00

7	6	5	4	3	2	1	0
Y[7:0]							
r/w							

Bit 7-0: **Y[7:0]** index Y lower byte

This is the lower byte of the index register used to create effective addresses or as a temporary storage area for data manipulations. In most of the cases the cross assembler generates a PRECODE instruction to indicate that the following instruction refers to the Y register.

5.4.9 SPH (stack pointer)

Offset: 0x08

Default value: refer to the product datasheet

7	6	5	4	3	2	1	0
SP[15:8]							
r/w							

Bit 7-0: **SP[15:8]** stack pointer upper byte

This is the upper byte of the 16-bit register stack pointer used to point the address of the next free location of the stack. The stack is used to save the CPU context on subroutine calls or interrupts. The user can also directly use it through the POP and PUSH instructions. Two arithmetic operations (ADD/SUB) are also possible using the stack pointer.

5.4.10 SPL (stack pointer)

Offset: 0x09

Default value: refer to the product datasheet

7	6	5	4	3	2	1	0
SP[7:0]							
r/w							

Bit 7-0: **SP[7:0]** stack pointer lower byte

This is the lower byte of the 16-bit register stack pointer used to point the address of the next free location of the stack. The stack is used to save the CPU context on subroutine calls or interrupts. The user can also directly use it through the POP and PUSH instructions. Two arithmetic operations (ADD/SUB) are also possible using the stack pointer.

5.4.11 CC (code condition)

Offset: 0x0A

Default value: 0x28

7	6	5	4	3	2	1	0
V	0	I1	H	I0	N	Z	C
r	r	r	r	r	r	r	r

The condition code register is an 8-bit register which indicates the result of the instruction just executed as well as the state of the processor. These bits can be individually tested by a program and specified action can be taken as a result of their state.

For bit detailed explanation refer to *Section 5.3.5: Condition code register (CC) on page 29*.

5.5 CPU register map

The CPU registers are mapped in the STLUX address space as shown in *Table 5*. These registers can only be accessed by the debug module but not by memory access instructions executed in the core; for detailed register description refer to *Section 5.4*.

Table 5. CPU register map

Registers overview				
Name	**Description**	**Offset**	**Type(1)**	**Reset value**
A	Accumulator register	0x00	R	0x00
PCE	Program counter extend	0x01	R	0x00
PCH	Program counter high	0x02	R	0x60
PCL	Program counter low	0x03	R	0x00
XH	X index register MSB	0x04	R	0x00
XL	X index register LSB	0x05	R	0x00
YH	Y index register MSB	0x06	R	0x00
YL	Y index register LSB	0x07	R	0x00
SPH	Stack pointer MSB	0x08	R	(2)
SPL	Stack pointer LSB	0x09	R	(2)
CC	Condition code register	0x0A	R	0x28

1. The CPU registers are accessible only by the debug interface.
2. For the register initialization and base address refer to the product datasheet.

5.6 Global configuration register description (CFG_GCR)

5.6.1 SWIM disable

By default, after a MCU reset, the SWIM pin is configured to allow communication with an external tool for debugging or Flash/EEPROM programming. This pin can be disabled from the user application by setting the SWD bit in the CFG_GCR register.

5.6.2 Global configuration descriptions

CFG_GCR (global configuration register)

Offset: 0x00

Default value: 0x00

7	6	5	4	3	2	1	0
HSIT	RFU	RFU	RFU	RFU	RFU	AL	SWD
r/w	r	r	r	r	r	r/w	r/w

Bit 0: **SWD** SWIM I/O mode control

This bit is set and cleared by SW:

0: SWIM mode enable (this bit should be kept 0 in case of the debug phase).

1: SWIM mode disable (currently this selection simply disconnect the SWIM interface signal from the external pin).

Bit 1: **AL** activation level

0: main activation level, the IRET will cause the context to be retrieved from the stack and the main program will continue after the WFI instruction.

1: interrupt only activation level, the IRET will cause the CPU to go back to the WFI/HALT mode without restoring the context. The saved internal registers context will be marked as invalid in case of an event wakeup at a later moment.

Bit 6-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 7: **HSIT** high-speed oscillator trimmed

This bit is read/write. It indicates that the SWIM can work in the high-speed mode (when the HSI trimming is done by SW). It is reset by an external reset.

0: HSI not trimmed.

1: HSI trimmed.

6 Debug architecture

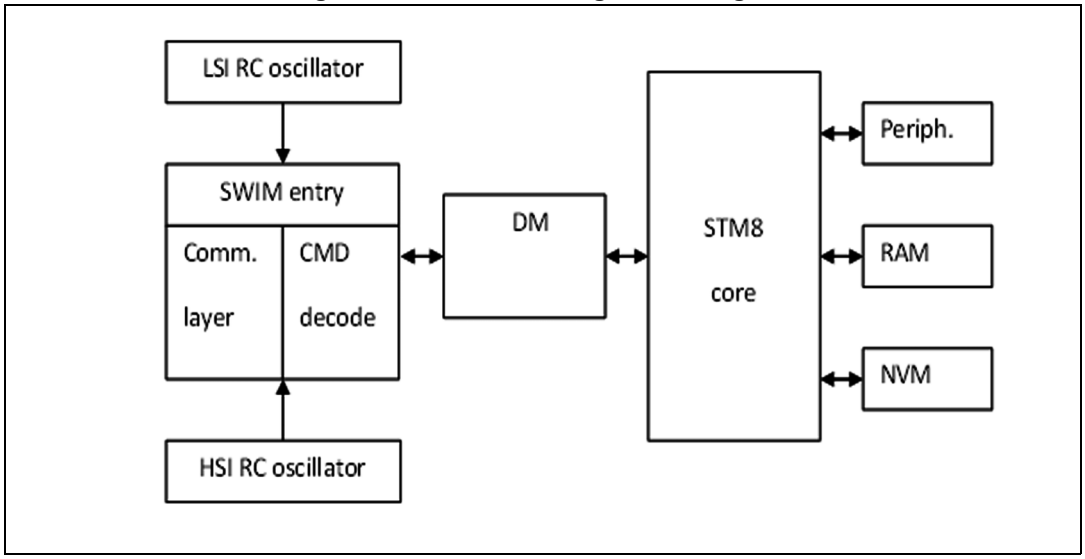
6.1 Introduction

The in-circuit debugging mode or the in-circuit programming mode are managed through a single wire hardware interface featuring ultrafast memory programming. Coupled with an in-circuit debugging module, it also offers a non-intrusive emulation mode, making the in-circuit debugger extremely powerful, close in performance to a full-featured emulator.

The embedded in-circuit debug module consists of two major functional subblocks:

- SWIM: single wire interface
- DM: debug module

Figure 5. In-circuit debug block diagram



6.2 Single wire interface module (SWIM)

After a power-on reset, the SWIM is reset and enters the OFF mode.

1. OFF: default state after power-on reset. The SWIM pin cannot be used by the application as an I/O.
2. SWIM: this state is entered when a specific sequence is performed on the SWIM pin. In this state, the SWIM pin is used by the host tool to control the STLUX with 3 commands (SRST system reset, ROTF read on the fly, WOTF write on the fly).

Note: Refer to the STM8 SWIM communication protocol and debug module user manual for description of the SWIM protocol.

6.3 SWIM main features

- Single wire communication based on an asynchronous open drain bidirectional line
- Non-intrusive read/writes to RAM and peripherals
- Interface enabled in the all IC operating mode

6.4 SWIM register description

This register is accessible only from the SWIM debug interface; the CPU cannot access this register space.

SWIM_CSR (SWIM control register)

Offset: 0x00

Default value: 0x00

7	6	5	4	3	2	1	0
SAFE_MASK	NO_ACCESS	SWIM_DM	HS	OSCOFF	RST	OBL	SWIM_PRI
r/w	r	r/w	r/w	r/w	r/w	r	r/w

Bit 0: **SWIM_PRI** SWIM direct memory access priority.

Usually the SWIM accesses to system resources are non-intrusive.

0: SWIM has the lowest priority.

1: SWIM has the highest priority.

Bit 1: **OBL** option byte loading done

0: option byte loading not done.

1: option byte loading done.

Bit 2: **RST** SWIM reset control bit

0: SWIM is not reset when an SRST command occurs.

1: SWIM is reset when an SRST command occurs. SWIM will re-enter the OFF mode.

Bit 3: **OSCOFF** oscillators control bit

0: 16 MHz oscillator remains ON in the Halt mode.

1: 16 MHz oscillator is not requested ON in the Halt mode.

Bit 4: **HS** high-speed.

The speed change occurs when the communication is IDLE.

0: slow speed bit format.

1: high-speed bit format.

Bit 5: **SWIM_DM** SWIM for debug module.

0: the SWIM module can access only the SWIM_CSR register. The SWIM reset command has no effect.

1: the whole memory range can be accessed with the ROTF and WOTF commands.

The SRST command generates a reset.

Bit 6: **NO_ACCESS** bus not accessible

0: the bus is accessible.

1: the device is in the HALT/WFI mode, or the readout protection is set.

Bit 7: **SAFE_MASK**: mask internal RESET sources.

0: internal reset sources are not masked.

1: internal reset sources are masked.

6.5 SWIM register map

Table 6 shows the SWIM internal register; for a detailed register description refer to *Section 6.4: SWIM register description*.

Table 6. SWIM internal register overview

Register overview				
Name	**Description**	**Offset**	**Type**	**Reset value**
SWIM_CSR	Control status register.	0x00	R/W	0x00

Note: This register is accessible only by the SWIM debug interface; the CPU cannot configure this register space.

6.6 Debug module

The debug module (DM) allows the developer to perform certain debugging tasks without using an emulator. For example, the DM can interrupt the MCU to break infinite loops or output the core context (stack) at a given point. The debug module is mainly used for the IC in-circuit debugging.

6.7 DM main features

- Two conditional breakpoints (break on instruction fetch, data read or write, stack access, etc.).
- Step-by-step mode
- External abort capability through SWIM
- Watchdog and peripherals control
- Monitor, RESET and abort status flags
- DM version identification capability
- IT vector table selection
- Multi debug module configuration capability

Some peripherals can be frozen (clock gate) when the CPU enters into the debug mode through the DM_ENFCT register (for details refer to register program model in *Section 6.8.11: DM_ENFCT (DM enable function register) on page 44*).

6.8 Internal registers description

Note: These registers are accessible only by the SWIM debug interface; the CPU cannot configure this register space.

6.8.1 DM_BK1E (DM breakpoint 1 ext. byte)

Offset: 0x00

Default value: 0xFF

7	6	5	4	3	2	1	0
BK1 [23:16]							
r/w							

Bit 7-0: **BK1[23:16]** DM breakpoint 1 upper byte.

6.8.2 DM_BK1H (DM breakpoint 1 high byte)

Offset: 0x01

Default value: 0xFF

7	6	5	4	3	2	1	0
BK1 [15:8]							
r/w							

Bit 7-0: **BK1[15:8]** DM breakpoint 1 middle byte.

6.8.3 DM_BK1L (DM breakpoint 1 low byte)

Offset: 0x02

Default value: 0xFF

7	6	5	4	3	2	1	0
BK1 [7:0]							
r/w							

Bit 7-0: **BK1[7:0]** DM breakpoint 1 lower byte.

6.8.4 DM_BK2E (DM breakpoint 2 ext. byte)

Offset: 0x03

Default value: 0xFF

7	6	5	4	3	2	1	0
BK2 [23:16]							
r/w							

Bit 7-0: **BK2[23:16]** DM breakpoint 2 upper byte.

6.8.5 DM_BK2H (DM breakpoint 2 high byte)

Offset: 0x04

Default value: 0xFF

7	6	5	4	3	2	1	0
BK2 [15:8]							
r/w							

Bit 7-0: **BK2[15:8]** DM breakpoint 2 middle byte.

6.8.6 DM_BK2L (DM breakpoint 2 low byte)

Offset: 0x05

Default value: 0xFF

7	6	5	4	3	2	1	0
BK2 [7:0]							
r/w							

Bit 7-0: **BK2[7:0]** DM breakpoint 2 lower byte.

6.8.7 DM_CR (DM control register)

Offset: 0x06

Default value: 0x00

7	6	5	4	3	2	1	0
WDGOFF	RFU	BC [2:0]			BIR	BIW	BKEE
r/w	r	r/w			r/w	r/w	r/w

Bit 0: **BKEE** break external enable

This bit is set and cleared by SWIM software. When set, it enables external break sources (product dependent) to generate a DM break.

0: external break disabled.

1: external break enabled.

Bit 1: **BIW** break on write control

This bit enables a breakpoint on a data write operation. It is set and cleared by software.

0: no break on data write.

1: break on data write.

Bit 2: **BIR** break on read control

This bit enables a breakpoint on a data read operation. It is set and cleared by software.

0: no break on data read.

1: break on data read.

Bit 5-3: **BC[2:0]** break on read control

These bits are set and cleared by software, they are used to configure the breakpoints.

Bit 6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 7: **WDGOFF** watchdog control enable

This bit must be set or cleared by software before the watchdog is activated. This bit has no effect if the hardware watchdog option is selected.

0: watchdog counter is not stopped while CPU is stalled by DM.

1: watchdog counter is stopped while CPU is stalled by DM.

6.8.8 DM_CR2 (DM control register 2)

Offset: 0x07

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	RFU	RFU	DEA	FVEC
r	r	r	r	r	r	r/w	r/w

Bit 0: **FVEC** forces vector table in RAM

This bit is set or cleared by software. It forces vector table to the RAM location instead of program memory (usually 8000h).

0: vector table is in the program memory area (8000h).

1: vector table is in the RAM memory area (depends of the product).

Bit 1: **DEA** disable external abort

This bit is set or cleared by software. It disables the external abort, this external abort source is product dependent.

0: external abort enable.

1: disable external abort.

Bit 7-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

6.8.9 DM_CSR (DM status/control register 1)

Offset: 0x08

Default value: 0x10

7	6	5	4	3	2	1	0
RFU	STE	STF	RST	BRW	BK2F	BK1F	BKEF
r	r/w	r	r	r	r	r	r

Bit 0: **BKEF** external breakpoint flag

This bit indicates that the DM break was generated by an external event - break event (product dependent). It is cleared by hardware when the DMCSR2.STALL bit is cleared. Writing to this bit does not change the bit value.

0: external breakpoint did not occur.

1: external breakpoint occurred.

Bit 1: **BK1F** breakpoint 1 flag

This bit indicates that the DM break was generated by the breakpoint 1. It is set by hardware depending on the control conditions and is cleared by hardware when the DMCSR2.STALL bit is cleared. Writing to this bit does not change the bit value.

0: breakpoint 1 did not occur.

1: breakpoint 1 occurred.

Bit 2: **BK2F** breakpoint 2 flag

This bit indicates that the DM break was generated by the breakpoint 2. It is set by hardware depending on the control conditions and is cleared by hardware when exiting the ICC monitor. Writing to this bit does not change the bit value.

0: breakpoint 2 did not occur.

1: breakpoint 2 occurred.

Bit 3: **BRW** break on read/write flag

This bit gives the value of the read/write signal when a break occurs. Its value is not significant for instruction fetch breaks. It is set by hardware depending on the breakpoint conditions and is cleared by hardware depending on the next breakpoint conditions. Writing to this bit does not change the bit value.

0: breakpoint on write.

1: breakpoint on read.

Bit 4: **RST** reset flag

This bit is set by hardware when a RESET occurs. It is cleared by hardware when the DMCSR2.STALL bit is cleared. Writing to this bit does not change the bit value. It indicates whether the CPU was stalled, by DM, just after RESET.

0: no RESET occurred.

1: a RESET occurred.

Bit 5: **STF** Step-by-step flag

This bit indicates that the DM break was generated by the step-by-step capability. It is set by hardware depending on the control conditions and is cleared by hardware when the DMCSR2.STALL bit is cleared. Writing to this bit does not change the bit value.

0: step-by-step interrupt did not occur.

1: step-by-step interrupt occurred.

Bit 6: **STE** step-by-step enable

This bit is set and cleared by software. It enables the step-by-step break capability.

0: step-by-step break disabled.

1: step-by-step break enabled.

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

6.8.10 DM_CSR2 (DM status/control register 2)

Offset: 0x09

Default value: 0x00

7	6	5	4	3	2	1	0
TRACE2	TRACE1	SWBKE	SWBKF	STALL	RFU	RFU	FLUSH
r/w	r/w	r/w	r	r/w	r	r	r

Bit 0: **FLUSH** FLUSH decode stage

Set by the WOTF command in order to flush the decode stage. It needs to be set when the PC was updated, before clearing the STALL bit (< Go_to_new context> command).

It is reset when the STALL bit is cleared.

Bit 2-1: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 3: **STALL** CPU STALL control bit

This bit is used to stall the CPU. This bit is kept cleared if the device is not in the SWIM mode. After RESET, this bit is set if the the SWIM is active. This will cause the CPU to be stalled after the reset vector fetch. Set by the WOTF command to generate an ABORT equivalent command or by SW when the CPU decodes a BREAK instruction. This bit is cleared by the WOTF command to restart the CPU; the clearing it's seen as a “exit” of the monitor routine by the DM (clearing of the flag).

0: CPU runs normally.

1: CPU is stalled.

Bit 4: **SWBKF** SW breakpoints status bit

This flag is set when the CPU executes the SW break instruction.

0: no SW break instruction detected (in decode stage).

1: SW break instruction detected. This bit is cleared when the STALL (SWIM) bit is cleared.

Bit 5: **SWBKE** SW breakpoint control bit

This bit is used to enable/disable the software breakpoint capability with the BREAK instruction.

0: DM does not generate any event when the BREAK instruction is fetched by the CPU.

1: DM generates an event (CPU stalled) when the S/W break instruction is fetched by the CPU.

Bit 6: **TRACE1** start trace 1 control bit (read/write)

This bit is used to enable/disable the TRACE start capability on the DM break1 event. The recording is started from the next instruction.

0 : the break1 condition is managed as a standard break event.

1: when the break1 condition is reached, the trace is triggered. The break event is not generated and the application is not stopped.

Bit 7: **TRACE2** start trace 2 control bit (read/write)

This bit is used to enable/disable the TRACE start capability on the DM break2 event. The recording is started from the next instruction.

0: the break2 condition is managed as a standard break event.

1: when the break2 condition is reached, the trace is triggered. The break event is not generated and the application is not stopped.

6.8.11 DM_ENFCT (DM enable function register)

Offset: 0x0A

Default value: 0xFF

7	6	5	4	3	2	1	0
ENFCT7	ENFCT6	ENFCT5	ENFCT4	ENFCT3	ENFCT2	ENFCT1	ENFCT0
r/w	r/w	r	r	r/w	r	r	r/w

This register field configurable by SWIM software is used to freeze the clocks of some peripherals when the IC enters in the debug mode.

Bit 0: **ENFCT0** system timer peripheral

0: peripheral frozen.

1: peripheral enabled.

Bit 1: **ENFCT1** reserved for future use.

Bit 2: **ENFCT2** reserved for future use.

Bit 3: **ENFCT3** SMED (SMED0 to SMED5) peripherals

0: peripheral frozen.

1: peripheral enabled.

Bit 4: **ENFCT4** reserved for future use.

Bit 5: **ENFCT5** reserved for future use.

Bit 6: **ENFCT6** ADC peripheral

0: peripheral frozen.

1: peripheral enabled.

Bit 7: **ENFCT7** I^2C peripheral

0: peripheral frozen.

1: peripheral enabled.

6.8.12 DM_VER (DM version register)

Offset: 0x0B

Default value: debug module version

7	6	5	4	3	2	1	0
DM_VER [7:0]							
r							

Bit 7-0: **DM_VER[7:0]** DM version

6.9 Internal registers overview

Table 7 shows the debug module internal registers starting from base address reported in the corresponding device datasheet; for detailed register description refer to *Section 6.8*.

Table 7. DM internal registers overview

Name	Description	Offset	Type	Reset value
DM_BK1E	Breakpoint 1 byte extend	0x00	R/W	0xFF
DM_BK1H	Breakpoint 1 byte high	0x01	R/W	0xFF
DM_BK1L	Breakpoint 1 byte low	0x02	R/W	0xFF
DM_BK2E	Breakpoint 2 byte extend	0x03	R/W	0xFF
DM_BK2H	Breakpoint 2 byte high	0x04	R/W	0xFF
DM_BK2L	Breakpoint 2 byte low	0x05	R/W	0xFF
DM_CR	Control register	0x06	R/W	0x00
DM_CR2	Control register2	0x07	R/W	0x00
DM_CSR	Control and status register	0x08	R/W	0x10
DM_CSR2	Control and status register2	0x09	R/W	0x00
DM_ENFCT	Enable function register	0x0A	R/W	0xFF
DM_VER	Version register	0x0B	R	DM - version number

Note: These registers are accessible only from the SWIM debug interface; the CPU cannot configure these registers space.

7 Boot ROM

The bootloader is stored inside the internal 2 Kbytes ROM, and its main task is to download the application program to the internal Flash memory through the UART peripherals interface without using the SWIM protocol or a dedicated programming tool; for the UART boot device configuration refer to the product datasheet.

The boot loader starts executing after reset.

Refer to the STLUX bootloader application note (AN4656) for more details.

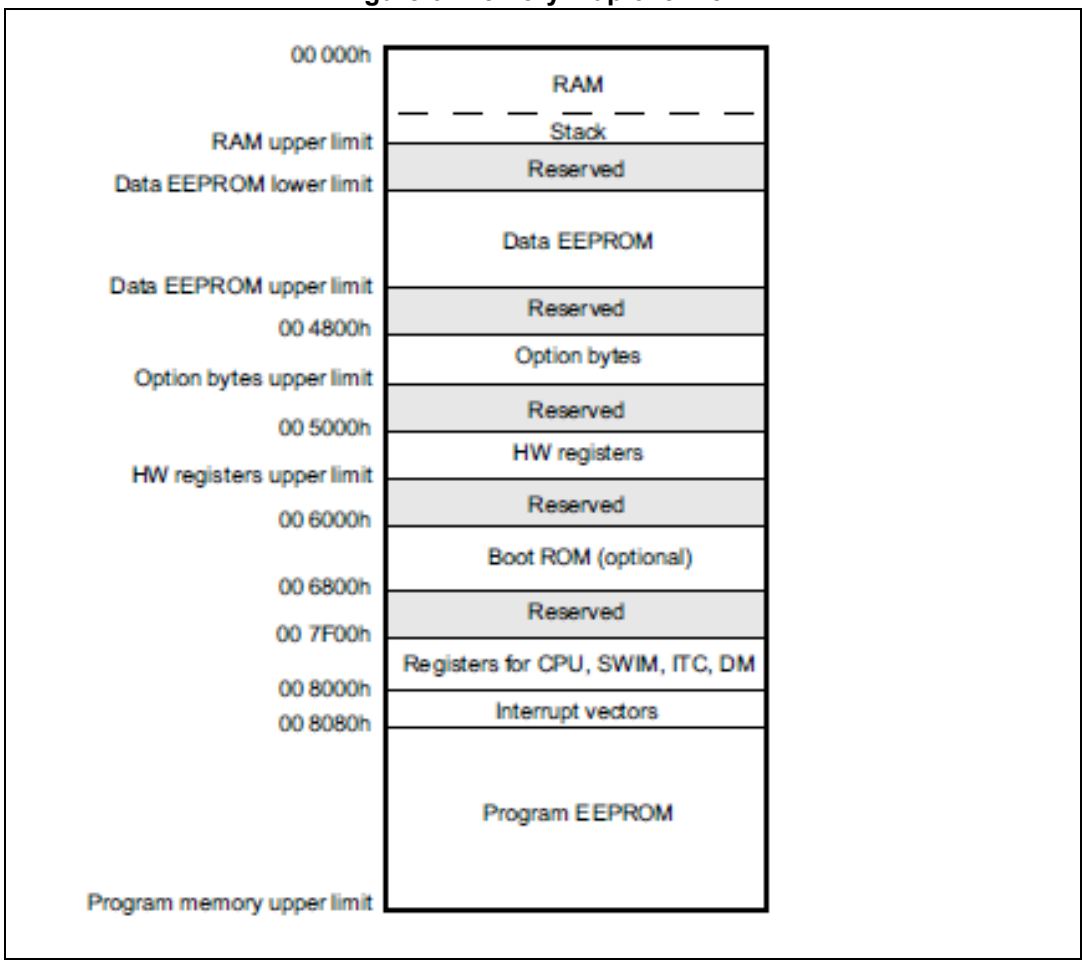
8 Memory and register map

For details on the memory map, I/O port hardware register map and interrupt controller registers, refer to the product datasheets.

8.1 Memory layout

8.1.1 Memory map

Figure 6. Memory map overview



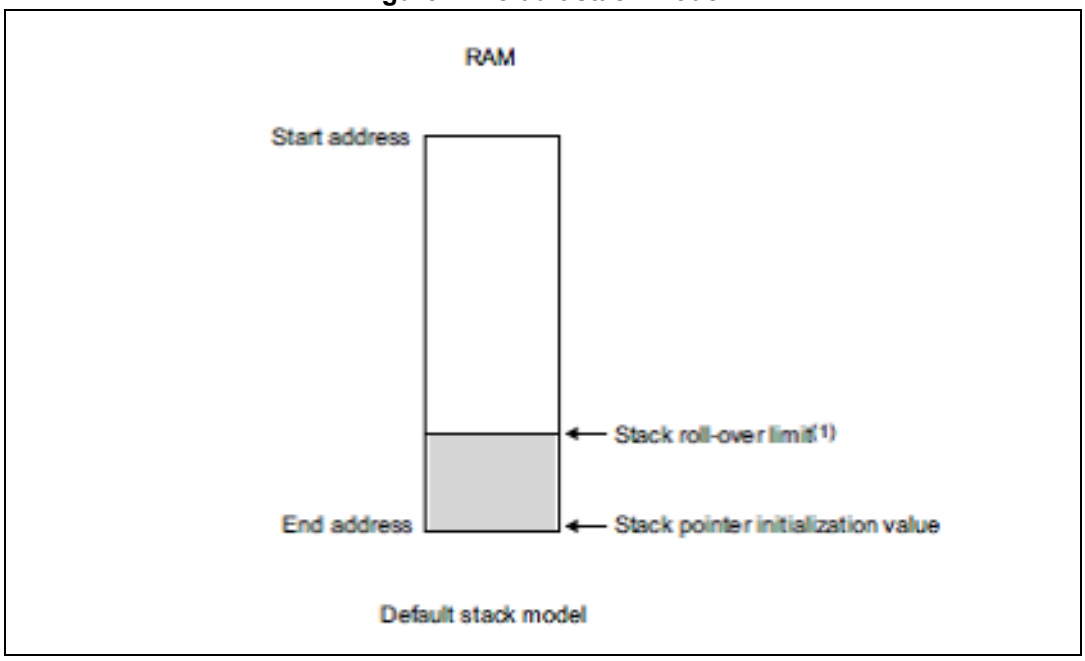
The RAM upper limit, data EEPROM upper and lower limit, option byte upper limit, hardware (HW) registers upper limit, and the program memory upper limit are specific to the device configuration. Please refer to the datasheets for quantitative information.

8.1.2 Stack handling

Default stack model

The stack of the STLUX microcontrollers is implemented in the user RAM area. The default stack model is shown in *Figure 7*.

Figure 7. Default stack model



Note: The stack rollover limit is not implemented on all devices. Refer to the datasheets for detailed information.

Stack pointer initialization value

This is the default value of the stack pointer. The user must take care to initialize this pointer. Correct loading of this pointer is usually performed by the initialization code generated by the development tools (linker file). In the default stack model this pointer is initialized to the RAM end address.

Stack rollover limit

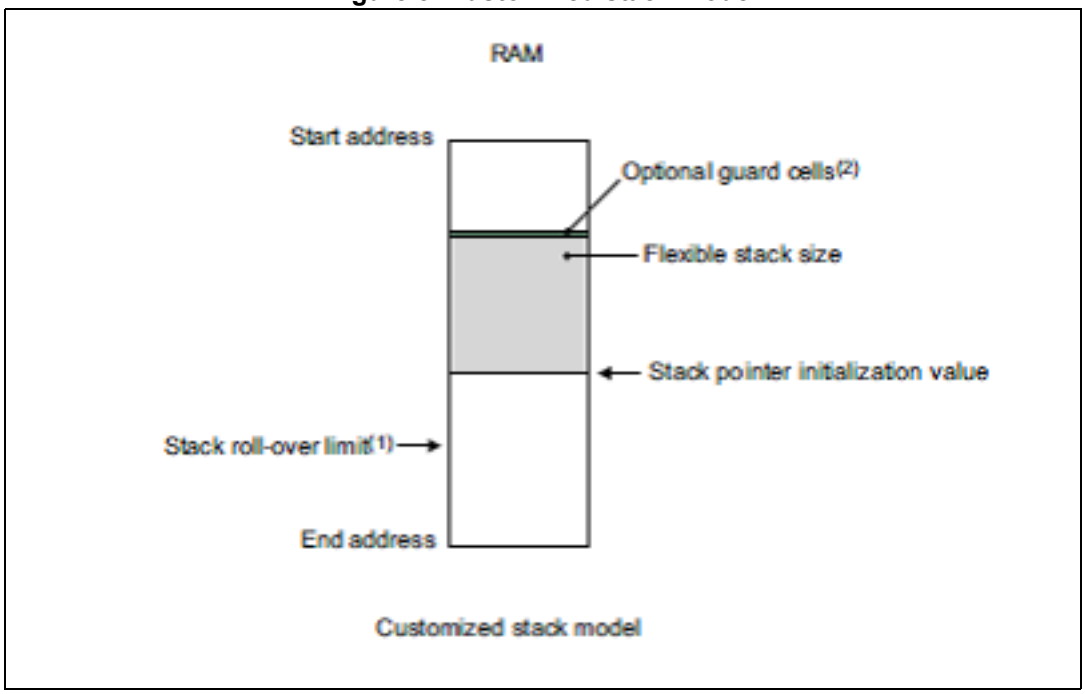
A stack rollover limit is implemented at a fixed address. If the stack pointer is decreased below the stack rollover limit, using a push operation or during context saving for subroutines or interrupt routines, it is reset to the RAM end address. The stack pointer does not rollover if the stack pointer arithmetic is used.

Such behavior of the stack pointer is of particular importance when developing software on a device with a different memory configuration than the target device. SW should avoid the stack rollover condition.

Customized stack model

STLUX stack pointer handling allows a customized stack model to be implemented. This permits a flexible stack size without restrictions due to the stack rollover limit. Implementing the customized stack also benefits portability of the software on products with different memory configurations. *Figure 8* shows the customized stack model.

Figure 8. Customized stack model



Note: The guard cells are RAM locations that have to be continuously polled by the application program to detect whether a stack overflow has taken place.

In this stack model, the initial stack pointer must be placed beyond the stack rollover limit. Consequently, the growing stack never reaches the stack rollover limit. It is clear that in this implementation the stack size is not limited by the rollover mechanism. Nevertheless, the user has to define the stack position and stack size in the link file, and he has to ensure that the stack pointer does not exceed the defined stack area (stack overflow or underrun).

The RAM locations above and below the customized stack can be regularly used as the RAM to store variables or other information. Guard cells can be implemented at the lower end of the stack to detect if the stack pointer exceeds the defined limit. These cells are standard RAM locations, initialized with fixed values that the stack overwrites if an overflow occurs. The user software can regularly poll these cells, detect the overflow condition, and put the application in a failsafe state.

During the software validation phase hardware breakpoints can be set at both limits of the stack to validate that neither a stack overflow nor an underrun happens.

9 Flash program memory and data EEPROM

The embedded program Flash and data EEPROM memories are controlled by a common set of registers; through these registers the user can program or erase the memory contents, set the write protection features, configure specific low power modes and program the device option bytes memory fields which configure the IC functional operating mode after reset.

9.1 Main Flash memory features

- STLUX provide two memory areas:
  - Flash: up to 32 Kbytes of the Flash program memory. The memory density differs according to the product device (refer to the product datasheet).
  - EEPROM: up to 1 Kbytes of the data EEPROM including option bytes. The data EEPROM density differs according to the device (refer to the product datasheet).
- Read while write capability (RWW)
- ECC (Error Correction Code): 6 bits of the error correction for every 32 bits data word.
- Configurable low power mode:
  - Memory power-down (when the MCU is in HALT or Active-halt mode)
- Supported operating mode:
  - In-application programming (IAP)
  - In-circuit programming (ICP)
- Programming modes:
  - Byte programming and automatic fast byte programming (without erase operation)
  - Word programming
  - Block programming and fast block programming mode (without erase operation)
  - Interrupt generation on end of program/erase operation and on illegal program operation
- Protection features:
  - Memory readout protection (ROP)
  - Program memory write protection with memory access security system (MASS keys)
  - Data memory write protection with memory access security system (MASS keys)
  - Programmable write-protected user boot code area (UBC).

9.2 Flash memory organization

The memories consist of three logical areas:

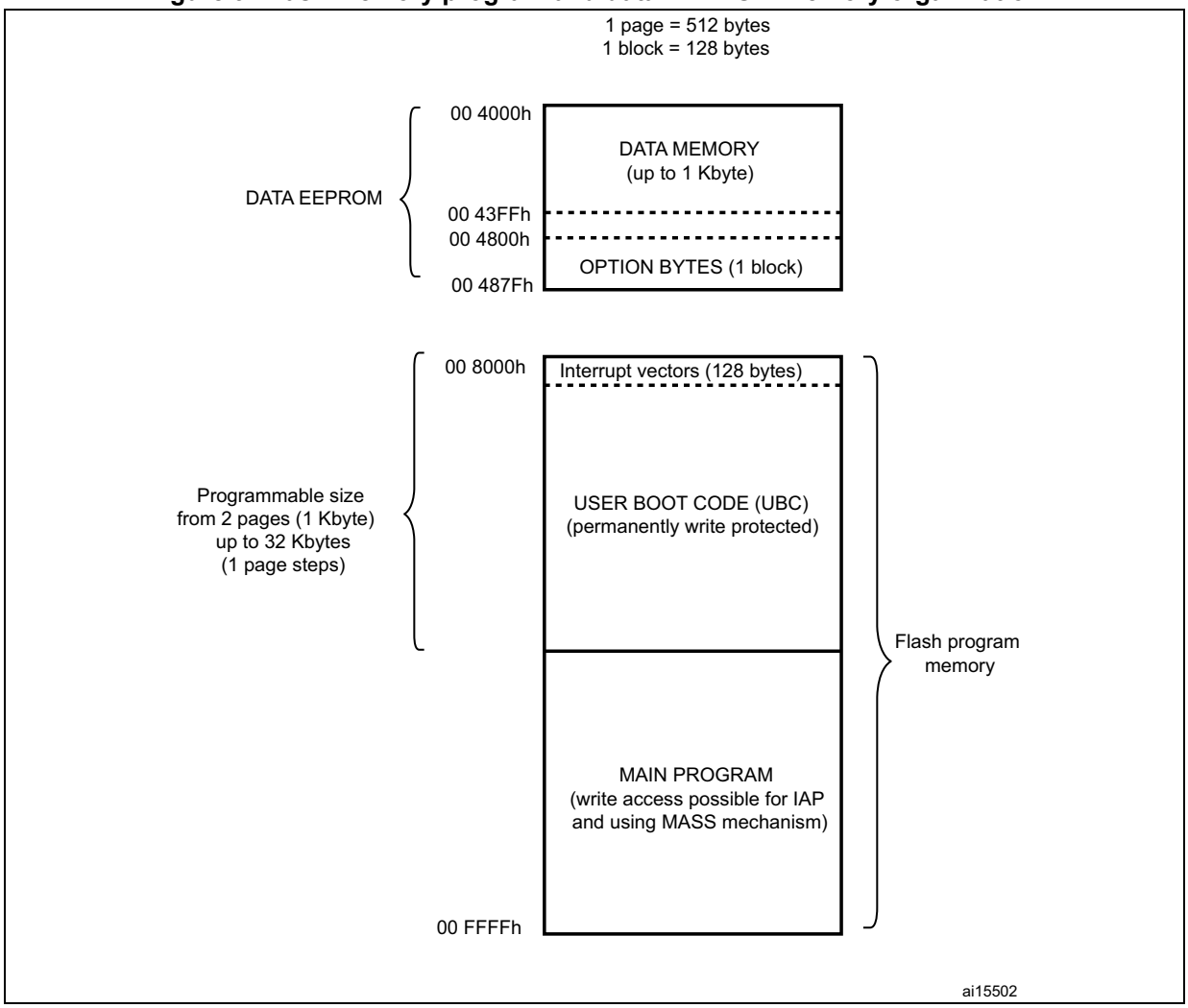
- Flash: up to 32 Kbytes of the Flash program memory. Density differs according to the device product datasheet. The Flash program memory is based on 64 pages of 512 bytes organized in 128 byte blocks.
- EEPROM: up to 1 Kbyte of the data EEPROM. Data EEPROM density differs according to the device product datasheet. The data EEPROM are based on 2 pages of 512 bytes organized in 128 byte blocks.
- EEPROM: up to 128bytes of the option byte data EEPROM.

The memories are organized in 32-bit data width protected by a 6 bit of ECC syndrome that improves the memory reliability.

The Error Correction Code (ECC) mechanism is able to detect and correct an exceptional single bit error in any word of data.

An outline view of the internal Flash program memory and data EEPROM memory organization is shown in *Figure 9*.

Figure 9. Flash memory program and data EEPROM memory organization



9.2.1 Memory access / wait state configuration

The program/data memory access time allows the device to run at up to 16 MHz without wait states.

9.2.2 Program memory

The Flash program memory is divided into 2 areas, the user boot code area (UBC), with size configurable through the option byte, and the main program memory area used to store the application code. The Flash program memory is mapped in the upper part of the STLUX addressing space and includes the reset and interrupt vector table.

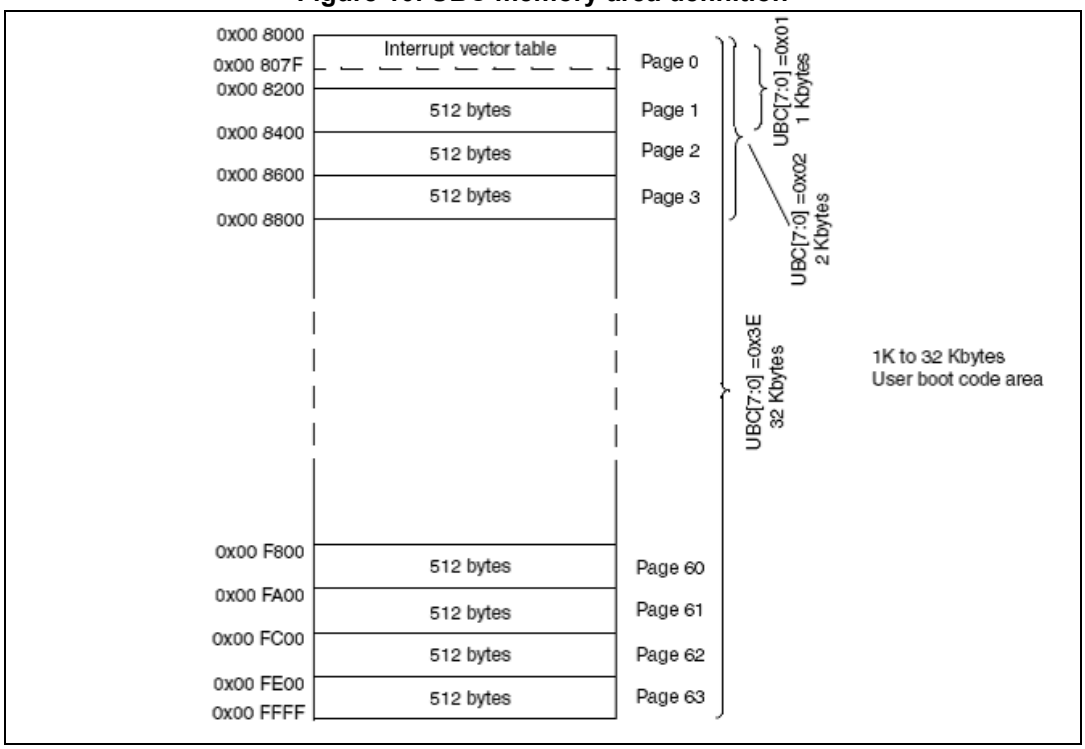
User boot area (UBC)

The user boot area (UBC) contains the reset and the interrupt vectors. It can be used to store the IAP and communication routines. The UBC area has a second level of protection to prevent unintentional erasing or a modification during IAP programming. This means that it is always write-protected and the write protection cannot be unlocked using the MASS keys.

The size of the UBC area can be configured in the ICP mode (using the SWIM interface) through the UBC option byte. The UBC option byte specifies the number of pages allocated for the UBC area starting from the address 0x008000.

The size of the UBC area can be obtained by reading the UBC option byte. The UBC memory mapping area is shown in *Figure 10*.

Figure 10. UBC memory area definition



Note: 1. UBC[7:0] = 0x00 means no user boot code area protection is defined.

2. The first 2 pages (1 Kbyte) contain the interrupt vectors table; only the first 128 bytes (32 IT vectors) are used for this purpose.

Program area

The Flash program area is used to store the user application SW program and static data (tables, and so on).

9.2.3 Data memory

The data EEPROM area can be used to store application data. By default, the DATA area is write-protected to prevent an unintentional modification when the main program is updated in the IAP mode. The write protection can be unlocked only using a specific MASS key sequence.

9.2.4 Option byte memory

The option bytes are used to configure device hardware features and memory protection. They are located in a dedicated memory array of one block.

The option bytes can be modified both in ICP/SWIM and in the IAP mode, with the OPT bit of the FLASH_CR2 register set to '1' and the nOPT bit of the FLASH_nCR2 register set to '0' (refer to *Section 9.6.2 on page 60*).

9.3 Memory protection mechanism

9.3.1 Readout protection (ROP)

Readout protection is enabled by programming the ROP option byte with the value 0xAA. When readout protection is enabled, reading or modifying the Flash program memory and DATA area in the ICP mode (using the SWIM interface) is forbidden, whatever the write protection settings. Even if no protection can be considered as totally unbreakable, the readout feature provides a very high level of protection for a general purpose microcontroller.

Disabling the readout protection

The readout protection involving the program and DATA areas can be disabled by resetting the ROP option byte in the ICP mode. In this case, the overall Flash program memory (including UBC area), the DATA area and the option bytes are automatically erased and the device can be reprogrammed.

9.3.2 Memory access security system (MASS)

After reset, the main program and DATA areas are protected against from unintentional write operations. They must be unlocked before attempting to modify their content. This unlock mechanism is managed by the memory access security system (MASS).

The UBC area specified in the UBC option byte is always writing protected. Once the memory has been modified, it is recommended to enable the write protection again to protect the memory content to prevent a possible data corruption.

Enabling write access to the main program memory

After a device reset, it is possible to disable the main program memory write protection by writing consecutively two values called MASS keys to the FLASH_PUKR register. These programmed keys are then compared to two hardware key values:

- First hardware key: 0b0101 0110 (0x56)
- Second hardware key: 0b1010 1110 (0xAE)

The following steps are required to disable write protection on the main program area:

1. Write the first 8-bit key into the FLASH_PUKR register. When this register is written for the first time after a reset, the data bus content is not latched into the register, but compared to the first hardware key value (0x56).
2. If the first hardware key is incorrect, then the FLASH_PUKR register remains locked until the next reset. Any new write commands attempt to this address is discarded.
3. If the first hardware key is correct, when the FLASH_PUKR register is written for the second time, the data bus content is still not latched into the register, but compared to the second hardware key value (0xAE).
4. If the second hardware key is incorrect, then the write protection on program memory remains locked until the next reset. Any new write commands sent to this address will be discarded.
5. If the second hardware key is correct, the main program memory is write unprotected and the PUL bit of the FLASH_IAPSR register is set.

Before starting to program the Flash, the application must verify that PUL bit is effectively set. The application may choose at any time to disable again write access to the Flash program memory by clearing the PUL bit.

Enabling write access to the DATA area

After a device reset, it is possible to disable the DATA area write protection by writing consecutively two values called MASS keys to the FLASH_DUKR register. These programmed keys are then compared to two hardware key values:

- First hardware key: 0b1010 1110 (0xAE)
- Second hardware key: 0b0101 0110 (0x56)

The following steps are required to disable write protection on the DATA area:

1. Write a first 8-bit key into the FLASH_DUKR register. When this register is written for the first time after a reset, the data bus content is not latched into the register, but compared to the first hardware key value (0xAE).
2. If the first hardware key is incorrect, the application can re-enter two MASS keys to try unprotecting the DATA area.
3. If the first hardware key is correct, the FLASH_DUKR register is programmed with the second key. The data bus content is still not latched into the register, but compared to the second hardware key value (0x56).
4. If the second hardware key is incorrect, the data EEPROM area remains write-protected until the next reset. Any new write command sent to this address is ignored.
5. If the second hardware key is correct, the DATA area is write unprotected and the DUL bit of the FLASH_IAPSR register is set.

Before starting to program the data area, the application must verify that the DATA area is not writing protected by checking that the DUL bit is effectively set. The application may choose at any time to disable again write access to the DATA area by clearing the DUL bit.

Enabling write access to option bytes

The procedure for enabling write access to the option byte area is the same used for data EEPROM access; however there is an additional OPT bit in the Flash control register 2 (FLASH_CR2) to be set and the corresponding nOPT bit in the Flash complementary control register 2 (FLASH_nCR2) to be cleared in order to enable write access to the option bytes.

9.4 Memory programming

The main program area and the data memory segment must be unlocked before attempting to perform any program sequences. For further detail refer to the *Section 9.3*.

9.4.1 Read-while-write (RWW)

The RWW feature provides the user the ability to perform write operation on data EEPROM while reading and executing the program memory. Execution time is therefore optimized. The opposite operation is not allowed: user cannot read data memory while writing program memory.

This RWW feature is enabled anytime without any specific operation required from the user. Writing to the Data EEPROM is controlled by the same control registers as for the program memory. Note that any access to the Flash control registers FLASH_CR1 and FLASH_CR2 while writing to the memory stalls the CPU, making RWW not feasible.

9.4.2 Programming modes

The internal Flash and the EEPROM data memories can be programmed in one of the following modes:

Byte programming

The main program memory and the DATA area can be programmed at byte level. To program one byte, the application writes directly to the target address.

- In the main program memory:
- The application stops for the duration of the byte program operation.
- In DATA area:
  - Devices with RWW capability: program execution does not stop, and the byte program operation is performed using the read-while-write (RWW) capability in the IAP mode.
  - Devices without RWW capability: The application stops for the duration of the byte program operation.

To erase a byte, simply write 0x00 to the corresponding address. The application can read the FLASH_IAPSR register to verify that the programming or erasing operation has been correctly executed:

- EOP flag is set after a successful programming operation
- WR_PG_DIS is set when the software has tried to write to a protected page. In this case, the write procedure is not performed.

As soon as one of these flags is set, a Flash interrupt is generated if it has been previously enabled by setting the IE bit of the FLASH_CR1 register.

Automatic fast byte programming

The programming duration can vary according to the initial content of the target address. If the word (4 bytes) containing the byte to be programmed is not empty, the whole word is automatically erased before the program operation. Otherwise if the word is empty, no erase operation is performed and the programming time is shorter (refer to t_{PROG} parameter present in the product datasheet).

However, the programming time can be fixed by setting the FIX bit of the FLASH_CR1 register to force the program operation to systematically erase the byte whatever its content (refer to *Section 9.6.1: FLASH_CR1 (Flash control register1) on page 60*). The programming times is consequently fixed and equal to the sum of the erase and write time (refer to t_{PROG} parameter present in the product datasheet).

Note: To write a byte fast (no erase), the whole word (4 bytes) into which it is written must be erased beforehand. Consequently, it is not possible to do consecutively two fast writes to the same word (without an erase before the second write): The first write will be fast but the second write to the other byte will require an erase.

Word programming

A word write operation allows an entire 4-byte word to be programmed at once, thus minimizing the programming time. As for byte programming, word operation is available both for the main program memory and data EEPROM. On some devices, the read-while-write (RWW) capability is also available when a word programming operation is performed on the data EEPROM. Refer to the datasheets for additional information.

- In the main program memory:
  The application stops for the duration of the byte program operation.
- In DATA area
  - Devices with RWW capability: program execution does not stop, and the byte program operation is performed using the read-while-write (RWW) capability in the IAP mode.
  - Devices without RWW capability: the application stops for the duration of the byte program operation.

To program a word, the WPRG/NWPRG bits in the FLASH_CR2 and FLASH_nCR2 registers must be previously set/cleared to enable the word programming mode (refer to *Section 9.6.2: FLASH_CR2 (Flash control register2) on page 60*). Then, the 4 bytes of the word to be programmed must be loaded starting with the first address. The programming cycle starts automatically when the 4 bytes have been written. As for byte operation, the EOP and the WR_PG_DIS control flags of FLASH_IAPSR, together with the Flash interrupt, can be used to determine if the operation has been correctly completed.

Block programming

Block program operations are much faster than byte or word program operations. In a block program operation, a whole block is programmed or erased in a single programming cycle. Block operations can be performed both to the main program memory and DATA area:

- In the main program memory:
  Block program operations to the main program memory have to be executed totally from RAM.
- In the DATA area
  - Devices with RWW capability: DATA block operations can be executed from the main program memory. However, the data loading phase (see below) has to be executed from RAM.
  - Devices without RWW capability: Block program operations must be executed totally from RAM.

There are three possible block operations:

- Block programming, also called standard block programming: the block is automatically erased before being programmed.
- Fast block programming: No previous erase operation is performed.
- Block erase

During block programming, interrupts are masked by hardware.

Standard block programming

A standard block program operation allows a whole block to be written at once. The block is automatically erased before being programmed. To program a whole block in the standard mode, the PRG/NPRG bits in the FLASH_CR2 and FLASH_nCR2 registers must be previously set/cleared to enable standard block programming (refer to *Section 9.6.2: FLASH_CR2 (Flash control register2) on page 60*). Then, the block of data to be programmed must be loaded sequentially to the destination addresses in the main program memory or DATA area. This causes all the data bytes to be latched.

To start programming the whole block, all 128 data byte belonging to the block must be written. All bytes written in a programming sequence must be in the same block. This means that they must have the same high address: Only the six least significant bits of the address can change. When the last byte of the target block is loaded, the programming starts automatically. It is preceded by an automatic erase operation of the whole block.

When programming a block in the DATA area, the application can check the HVOFF bit in the Flash status register (FLASH_IAPSR). As soon the HVOFF flag is reset the actual programming phase starts and the application can return to main program memory. The EOP and the WR_PG_DIS control flags of the FLASH_IAPSR together with the Flash interrupt can be used to determine if the operation has been correctly completed.

Fast block programming

Fast block programming allows programming without first erasing the memory contents. Fast block programming is therefore twice as fast as standard programming. This mode is intended only for programming parts that have already been erased. It is very useful for programming blank parts with the complete application code, as the time saving is significant.

Fast block programming is performed by using the same sequence as standard block programming. To enable the fast block programming mode, the FPRG/NFPRG bits of the FLASH_CR2 and FLASH_nCR2 registers must be previously set/cleared.

The HVOFF flag can also be polled by the application which can execute other instructions (RWW) during the actual programming phase of the DATA. The EOP and WR_PG_DIS bits of the FLASH_IAPSR register can be checked to determine if the fast block programming operation has been correctly completed.

Caution: The data programmed in the block are not guaranteed when the block is not blank prior to execute the fast block program operation.

Block erasing

A block erase allows a whole block to be erased. To erase a whole block, the ERASE/nERASE bits in the FLASH_CR2 and FLASH_nCR2 registers must be previously set/cleared to enable block erasing (refer to *Section 9.6.2: FLASH_CR2 (Flash control register2)* and *Section 9.6.3: FLASH_nCR2 (Flash control register2 protection)*

The block is then erased by writing '0x00 00 00 00' to any word inside the block. The word start address must end with '0', '4', '8', or 'C'. The EOP and the WR_PG_DIS control flags of the FLASH_IAPSR together with the Flash interrupt can be used to determine if the operation has been correctly completed.

Note: All memory block program/erase sequences must be executed without any interruption.

Option byte programming

Option byte programming is very similar to data EEPROM byte programming. The application writes directly to the target address. The program does not stop and the write operation is performed using the RWW capability.

Refer to the product datasheet for details on the option byte contents.

9.5 ICP and IAP

The In-Circuit programming (ICP) method is used to update the memory contents through the SWIM interface loading the user application data inside the microcontroller. The ICP offers quick and efficient configurability method removing unnecessary package handling or device socketing. The SWIM interface (single wire interface module) uses the SWIM pin to interconnect to the programming/debugging tool.

In contrast to the ICP method, in-application programming (IAP) can use any communication interface supported by the microcontroller (I/Os, I^2C, UART, etc.) to download the data to be programmed in memory. IAP allows reprogramming the EEPROM memory contents during the application execution.

Table 8. Memory access versus programming method(1)

Flash and EEPROM memories access mode			
Mode	**ROP**	**Memory area**	**Access(1)**
User, IAP, and bootloader			
	Readout protection enabled	User boot code area (UBC)	R/E
		Main program	R/W/E(2)
		Data EEPROM area (DATA)	R/W(3)
		Option bytes	R
	Readout protection disabled	User boot code area (UBC)	R/E(4)
		Main program	R/W/E(2)
		Data EEPROM area (DATA)	R/W(3)
		Option bytes	R/W(5)
ICP and SWIM	Readout protection enabled	User boot code area (UBC)	P
		Main program	P
		Data EEPROM area (DATA)	P
		Option bytes	R/WROP(6)
	Readout protection disabled	User boot code area (UBC)	R/E(4)
		Main program	R/W/E(2)
		Data EEPROM area (DATA)	R/W(3)
		Option bytes	R/W(5)

1. Flash access type:

   a) R/W/E: read; write and execute;

   b) R/E: read and execute (write operation forbidden);

   c) R: read (write and execute operations forbidden);

   d) P = the area cannot be accessed (read, execute and write operations forbidden);

   e) R/W_{ROP} = protected, write forbidden except for ROP option byte.

2. The Flash program memory is write-protected (locked) until the correct MASS key is written in the FLASH_PUKR. It is possible to lock the memory again by resetting the PUL bit in the FLASH_IAPSR register. If wrong keys are provided, the device must be reset and new keys programmed.
3. The data memory is write-protected (locked) until the correct MASS key is written in the FLASH_DUKR. It is possible to lock the memory again by resetting the DUL bit in the FLASH_IAPSR register. If wrong keys are provided, the device must be reset and new keys programmed.
4. To program the UBC area the application must first clear the UBC option byte.
5. The option bytes are write-protected (locked) until the correct MASS key is written in the FLASH_DUKR (with OPT set to '1'). It is possible to lock the memory again by resetting the DUL bit in the FLASH_IAPSR register. If wrong keys are provided, the device must be reset and new keys programmed.
6. When the ROP is removed, the whole memory is erased, including option bytes.

9.6 Flash registers description

9.6.1 FLASH_CR1 (Flash control register1)

Offset: 0x00

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	HALT	AHALT	IE	FIX
r	r	r	r	r/w	r/w	r/w	r/w

Bit 0: **FIX** fixed byte programming time

0: standard programming time of (1/2 $t_{PROG)}$ if the memory is already erased and t_{PROG} otherwise

1: programming time fixed at t_{PROG}. (refer to the product datasheet).

Bit 1: **IE** interrupt enable

0: Flash interrupt disable.

1: Flash interrupt enable. An interrupt is generated if the EOP or the WR_PG_DIS flag in the FLASH_IAPSR register is set.

Bit 2: **AHALT** power-down in Active-halt mode:

0: Flash operating when MCU is in Active-halt mode.

1: Flash power-down when MCU is in Active-halt mode.

Bit 3: **HALT** power-down in Halt mode

0: Flash in power-down when MCU is in Halt mode.

1: Flash in operating when MCU is in Halt mode.

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

9.6.2 FLASH_CR2 (Flash control register2)

Offset: 0x01

Default value: 0x00

7	6	5	4	3	2	1	0
OPT	WPRG	ERASE	FPRG	RFU	RFU	RFU	PRG
r/w	r/w	r/w	r/w	r	r	r	r/w

Bit 0: **PRG** block programming

This field configures the memory block programming operations; it's set by SW and cleared by HW when the operation is completed.

0: standard block program operation disabled.

1: standard block program operation enabled (automatically memory erasing before programming).

Bit 1-3: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 4: **FPRG**[a] fast block programming

This bit is set by SW and cleared by HW when the operation is completed. This field configures the memories fast block program operations (no memory erasing).

0: fast block program operation disabled.

1: fast block program operation enabled.

Bit 5: **ERASE**[a] erase block operation

This bit is set by SW cleared by HW when the operation is completed.

0: erase block operation disabled.

1: erase block operation enabled.

Bit 6: **WPRG** word programming

This bit is set by SW cleared by HW when the operation is completed.

0: word programming operation disabled.

1: word programming operation enabled.

Bit 7: **OPT** option byte enable programming mode

This bit is set and cleared by SW.

0: option bytes write access disabled.

1: option bytes write access enabled.

9.6.3 FLASH_nCR2 (Flash control register2 protection)

Offset: 0x02

Default value: 0xFF

7	6	5	4	3	2	1	0
nOPT	nWPRG	nERASE	nFPRG	RFU	RFU	RFU	nPRG
r/w	r/w	r/w	r/w	r	r	r	r/w

FLASH_nCR2: not(FLASH_CR2)

The configuration of the register FLASH_CR2 requires a back-to-back configuration of the same value complemented into the FLASH_nCR2 register to ensure the EMC protection mechanism. If the writing access of the FLASH_nCR2 register is delayed, the content value of the FLASH_CR2 register is cleared.

Caution: Between the configuration of the FLASH_CR2 and FLASH_nCR2 registers (above described) the interrupt request lines IRQ and NMI are masked by hardware to prevent an unwanted delay on the FLASH_nCR2 writing cycle.

9.6.4 FLASH_FPR (Flash write page protection)

Offset: 0x03

Default value: 0x00

a. The ERASE and FPRG bits are locked when the memory is busy.

7	6	5	4	3	2	1	0
WPB [7:0]							
r							

Bit 7-0: **WPB[7:0]** user boot code area protection pages

These bits show the size of the boot code area. They are loaded at a startup with the content of the UBC option byte; for detailed register description refer to the corresponding option register information in the product datasheet.

9.6.5 FLASH_nFPRP (FLASH_FPR protection)

Offset: 0x04

Default value: 0xFF

7	6	5	4	3	2	1	0
nWPB [7:0]							
r							

FLASH_nFPRP: not (FLASH_FPR) EMC byte protection.

9.6.6 FLASH_IAPSR (Flash status register)

Offset: 0x05

Default value: 0x40

7	6	5	4	3	2	1	0
RFU	HVOFF	RFU	RFU	DUL	EOP	PUL	WR_PG_DIS
r	r	r	r	rc/w0	rc/r	rc/w0	rc/r

Bit 0: **WR_PG_DIS** write attempted to protected page flag

This field is set by hardware and cleared by software by reading the register:

0: normal operating mode.

1: a write attempt to a write-protected page occurred. An interrupt is generated if the IE bit is set in the FLASH_CR1 register.

Bit 1: **PUL** Flash program memory unlocked flag

This field is set by hardware and cleared by software by programming it to '0'.

0: write protection of the main program area enabled.

1: write protection of the main program area is disabled by writing the correct MASS keys.

Bit 2: **EOP** end of programming (write or erase operation) flag

This field is set by hardware and cleared by software by reading the register or when a new write/erase operation starts.

0: no EOP event occurred.

1: an EOP operation occurred. An interrupt is generated if the IE bit is set in the FLASH_CR1 register.

Bit 3: **DUL** data EEPROM area unlocked flag

This bit is set by hardware and cleared by software by programming it to '0'.

0: data EEPROM area write protection enabled.

1: data EEPROM area write protection is disabled by writing the correct MASS keys.

Bit 5-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 6: **HVOFF** end of high voltage.

This bit is set and cleared by HW.

0: HV ON, start of actual programming.

1: HV OFF, end of high voltage.

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

9.6.7 FLASH_PUKR (Flash program memory unprotection)

Offset: 0x08

Default value: 0x00

7	6	5	4	3	2	1	0
WP [7:0]							
r/w							

Bit 7-0: **WP[7:0]** program memory unprotection key bit.

1st MASS program key: 0101 0110 (0x56)

2nd MASS program key: 1010 1110 (0xAE)

This byte is written by software (all modes). It returns 0x00 when read.

Refer to *Enabling write access to the main program memory in* Section 9.3.2 for the description of main program area write unprotecting mechanism.

9.6.8 FLASH_DUKR (Flash data memory unprotection)

Offset: 0x0A

Default value: 0x00

7	6	5	4	3	2	1	0
WD [7:0]							
r/w							

Bit 7-0: **WD[7:0]** data memory unprotect key bit.

1st MASS data key: 1010 1110 (0xAE).

2nd MASS data key: 0101 0110 (0x56).

This byte is written by software (all modes). It returns 0x00 when read.
Refer to *Enabling write access to the DATA area* in Section 9.3.2 for the description of DATA area write unprotecting mechanism.

9.6.9 FLASH_WAIT (Flash wait state register)

Offset: 0x0D

Default value: 0x00

7	6	5	4	3	2	1	0
RFU						WAIT[1:0]	
r						r	

The Flash controls the access time (Ta) register.

Bit 1-0: **WAIT[1:0]** Flash time access cycle wait state number.

00: 0 wait states cycle number (default case f_{MASTER} at 16 MHz).

01: 1 wait states cycle number.

10: 2 wait states cycle number.

11: 3 wait states cycle number.

Bit 7-2: **RFU** reserved for future use.

Note: 1. This register contains a read only copy of the two corresponding bits of the option byte WAITSTATE (0x480D). These bits are modifiable only reconfiguring the option byte. These bits are temporarily modifiable for debugging using the SWIM interface, and assume their default value (option byte value) at the next reset.

2. This register field has to be configured in accordance with the product datasheet requirement.

9.7 Flash registers overview

Table 9 shows the Flash control registers overview starting from the base address specified on the proper databook; for detailed registers description refer to *Section 9.6*.

Table 9. Flash registers overview

Name	Description	Offset	Type	Reset value
FLASH_CR1	Flash control register1	0x00	R/W	0x00
FLASH_CR2	Flash control register2	0x01	R/W	0x00
FLASH_nCR2	Flash control register2 (protection)	0x02	R/W	0xFF
FLASH_FRP	Flash UBC write protect register	0x03	R	0x00
FLASH_nFRP	Flash UBC write protect register	0x04	R	0xFF
FLASH_IAPSR	Flash status register	0x05	R/W	0x 40
RFU	Reserved	0x06-07	-	-
FLASH_PUKR	Program unprotect register	0x08	R/W	0x00
RFU	Reserved	0x09	-	-
FLASH_DUKR	Data unprotect register	0x0A	R/W	0x00
RFU	Reserved	0x0B-0C	-	-
FLASH_WAIT	Wait state register	0x0D	R	0x00

10 Power supply

The MCU has two distinct power supply lines:

- VDD/VSS: main power supply and I/O power supply (3 V to 5.5 V)
- VDDA/VSSA: power supply for the analog functions

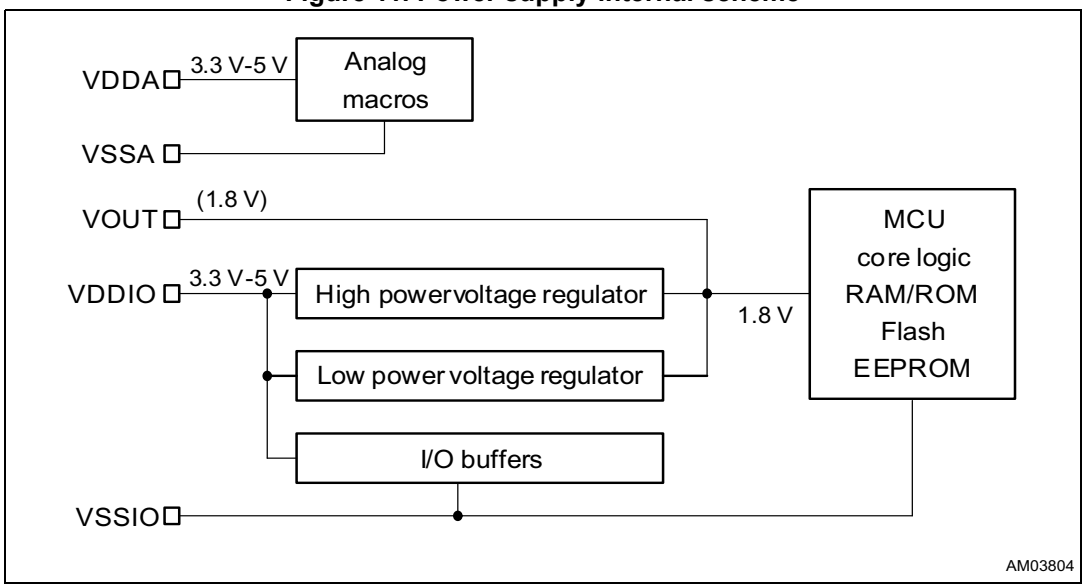
The VDD/VSS pins are used to supply the I/O pads, the internal main voltage regulator (MVR) and the internal low power voltage regulator (LPVR). The 2 regulator outputs are connected and provide the 1.8 V supply (V18) to the MCU core (CPU, Flash and RAM). In low power modes the system automatically switches from the MVR to the LPVR in order to reduce current consumption.

To stabilize the MVR, a capacitor must be connected to the VCAP pin (for more details refer to the product datasheet electrical characteristics section).

The VDDA/VSSA is connected to the analog macros (ADC, DACs and comparators).

An outline view of the IC power supply interconnections scheme is shown in *Figure 11*.

Figure 11. Power supply internal scheme



For complete information about the IC device power supply refer to the product datasheet.

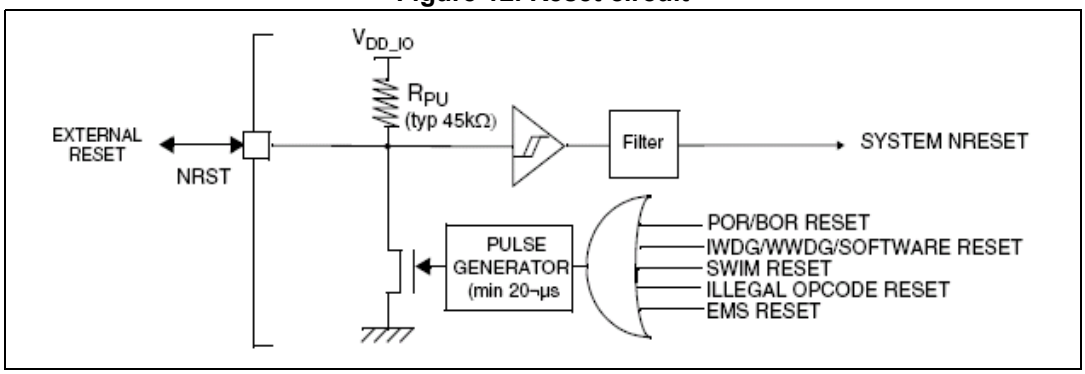
11 Reset control unit (RST)

The reset control unit is the module which controls the internal reset generation to the overall system, starting from the following nine platform reset sources:

- External reset through the NRST pin
- Power-on reset (POR)
- Brownout (BOR)
- Independent watchdog reset (IWDG) opcode
- Window watchdog reset (WWD)
- Software reset
- SWIM reset
- Illegal opcode reset
- EMC reset: generated if critical registers are corrupted or badly loaded

These sources act on the RESET pin and it is always kept low during the delay phase. The RESET service routine vector is fixed at the address specified on the related product datasheet.

Figure 12. Reset circuit



11.1 Reset state and reset in progress definition

When a reset occurs, there is a reset phase from the external pin pull-down to the internal reset signal release. During this phase, the microcontroller sets some hardware configurations before going to the reset the vector. At the end of this phase, most of the registers are configured with their “reset state” values. During the reset phase, i.e. “under reset”, some pin configurations may be different from their “reset state” configuration.

11.2 Reset circuit description

The NRST pin is both an input and an open drain output with an integrated R_{PU} weak pull-up resistor.

The low pulse duration of $t_{INFP(NRST)}$ (refer to the product datasheet) on the NRST pin generates an external reset. The reset detection is asynchronous and therefore the MCU can enter reset even in the HALT mode. The NRST pin also acts as an open drain output for resetting external devices.

An internal temporization maintains a pulse of at least $t_{ON(NRST)}$ (refer to the product datasheet) whatever the internal reset source. An additional internal weak pull-up ensures a high level on the reset pin when the reset is not driven.

11.3 Internal reset sources

Each internal reset source is linked to a specific flag bit in the reset status register (RST_SR) except the POR/BOR which have no flag.

These flags are set respectively at reset depending on the given reset source. So they are used to identify the last reset source event. These are cleared from software by writing the logic value “1” to the corresponding bit.

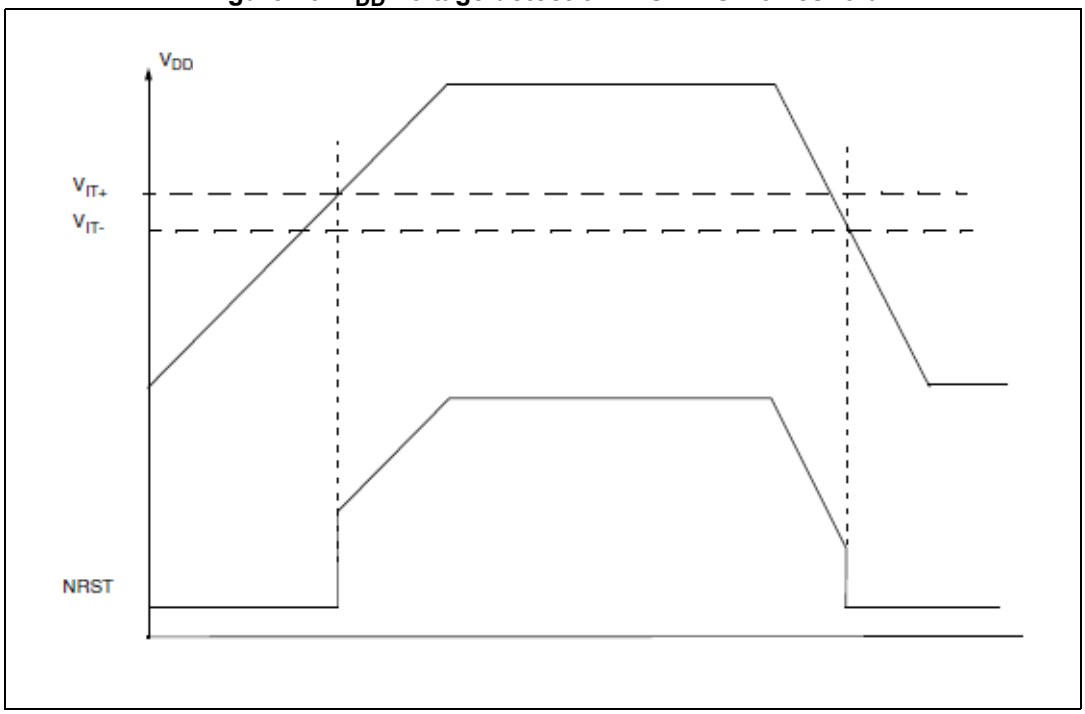
11.3.1 Power-on reset (POR) and brownout reset (BOR)

During power-on, the POR keeps the device under reset until the V_{DD} supply voltage reaches the voltage level at which the BOR starts to function. At this point, the BOR reset replaces the POR, and the POR is automatically switched off. The BOR reset is maintained till the supply voltage reaches the operating voltage range.

The BOR also generates a reset when the supply voltage drops below the V_{IT}- threshold (refer to the product datasheet). When this occurs, the POR is rearmed for the next power-on phase. A hysteresis is implemented to ensure clean detection of the voltage rise and fall.

The BOR always remains active even when the MCU is put into the low power mode.

Figure 13. V_{DD} voltage detection: POR/BOR threshold



11.3.2 Watchdog reset

Refer to *Section 16: Window watchdog (WWDG) on page 138* and *Section 15: Independent watchdog (IWDG) on page 133* for details.

11.3.3 Software reset

The application software can trigger a reset by clearing the T6 bit in the WWDG_CR register. Refer to *Section 16*.

11.3.4 SWIM reset

An external tool connected to the SWIM interface can request the SWIM block to generate an MCU reset.

11.3.5 Illegal opcode reset

In order to provide enhanced robustness to the device against unexpected behavior, a system of illegal opcode detection is implemented. If a code to be executed does not correspond to any opcode or prebyte predefined value, a reset is generated. This, combined with the watchdog, allows recovery from an unexpected fault or interference noise.

Note: A valid prebyte associated with a valid opcode forming an unauthorized combination does not generate a reset.

11.3.6 EMC reset

To protect the application against spurious write access or a system hang-up, possibly caused by electromagnetic disturbance, the most critical registers (i.e.: EEPROM configuration registers and some CKC registers) are implemented as two-bit fields that must contain complementary values. Mismatches are automatically detected by this mechanism, triggering an EMC reset and allowing the application to cleanly recover normal operations.

11.4 RST register description

For the base address refer to the related product datasheet.

RST_SR (status register)

Offset: 0x03

Default value: 0x00 after POR only

7	6	5	4	3	2	1	0
RFU	RFU	RFU	EMCF	SWIMF	ILLOPF	IWDGF	WWDGF
r	r	r	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bit 0: **WWDGF** window watchdog reset flag.

The bit indicates that the last reset was generated by the window watchdog peripheral. It's set by hardware and cleared by software (writing one) or a POR reset.

0: no reset generated by the window watchdog.

1: reset asserted by the window watchdog.

Bit 1: **IWDGF** independent watchdog reset flag.

The bit indicates that the last reset was generated by the independent watchdog peripheral. It is set by hardware and cleared by software (writing one) or a POR reset.

0: no reset generated by the independent watchdog.

1: reset asserted by the independent watchdog.

Bit 2: **ILLOPF** illegal opcode reset flag

The bit indicates that the last reset was generated by the CPU core. It is set by hardware (illegal opcode reset) and cleared by software (writing one) or a POR reset.

0: no reset generated by the CPU.

1: reset asserted by the CPU.

Bit 3:**SWIMF** SWIM reset flag

The bit indicates that the last reset was generated by the SWIM peripheral. It is set by hardware (SWIM reset) and cleared by software (writing one) or a POR reset.

0: no reset generated by the SWIM.

1: reset asserted by the SWIM.

Bit 4: **EMCF** EMC reset flag.

This bit indicates that the last reset was generated due to the EMC (protected register mismatch). It is set by hardware (comparison of protected registers) and cleared by software (writing one) or a POR reset.

0: no reset generated by the EMC.

1: reset asserted by an EMC error.

Bit 7-5: **RFU** reserved; must be kept 0 during register writing for future compatibility.

11.5 RST register overview

Table 10 shows the RST internal register starting from the base address specified on the corresponding device datasheet; for detailed register description refer to *Section 11.4*.

Table 10. Reset controller register overview

Name	Description	Offset	Type	Reset value
RST_SR	Reset status register	0x03	R/W1	0x00(1)

Note: The reset value is 0x00 after power-on reset.

12 Clock control unit (CKC)

The clock controller is the internal peripheral which generates and manages the clock signals feeding all the STLUX digital circuits.

The clock controller allows the user to configure the best performance and power reduction scheme for his application. The user can manage all the different clock sources independently and distribute them to the CPU and to the various peripherals.

A safe and glitch-free switch mechanism allows switching the system master clock on the fly from one clock source to another one and dividing it by means of the clock prescaler.

12.1 EMS - hardened clock configuration registers (optional)

To protect the application against spurious write access or a system hang-up, possibly caused by electromagnetic disturbance, the most critical CLK registers are equipped with two-bit fields that must contain complementary values. Mismatches are automatically detected by the CLK, triggering an EMS reset and allowing the application to cleanly recover normal operations. See the following paragraphs for more details.

12.2 Features overview

The clock controller is the system master module which performs the following functionalities:

- Oscillators on/off driving
- Oscillators stabilization
- Clock switch
- Low power modes

12.3 Master clock sources

Five different clock sources can be used to drive the master clock lines:

- HSI: 16 MHz high-speed internal RC oscillator
- LSI: 153.6 kHz low-speed internal RC oscillator
- PLL: internal PLL at 96 MHz (not used as the f_{MASTER} source clock)
- HSE: up to 24 MHz auxiliary external OSC/crystal
- HSE: up to 24 MHz high-speed user external clock

Each clock source can be switched on or off independently when it's not used, to optimize the power consumption.

12.3.1 HSI

The HSI is the default master clock line, generated from an internal RC oscillator with nominal frequency of 16 MHz; it has the following major features:

- RC architecture
- Glitch-free oscillation
- 3-bit user calibration circuit

Note: At a startup the master clock source is automatically selected as HSI RC clock output divided by 8 ($f_{HSI}/8$).

The HSI RC oscillator offers the advantage of providing a 16 MHz master clock source with 50% duty cycle at a low cost (no external components). It also has faster startup time than the HSE crystal oscillator. However, even with calibration, the frequency is less accurate than an external crystal oscillator or ceramic resonator. The HSIRDY flag in the internal clock register (CLK_ICKR) indicates if the HSI RC is stable or not. At a startup, the HSI RC output clock is not enabled until this bit is set by hardware. The HSI RC can be switched on and off using the HSIEN bit in the internal clock register (CLK_ICKR).

Backup source

The HSI/8 signal can also be used as a backup source (auxiliary clock) if the HSE crystal oscillator fails. Refer to *Section 12.8: Clock security system (CSS) on page 84*.

Calibration

Each device is factory calibrated by ST.

After reset, the factory calibration value is automatically loaded in an internal calibration register.

If the target application is subject to voltage or temperature variations, this may affect the RC oscillator speed. User can trim the HSI frequency in the application environment using the HSI clock calibration trimming register (CLK_HSITRIMR). In this register there are 3 bits providing an additional trimming value that is added to the internal HSI calibration register value. Each step corresponds to a mean frequency shift of 200 kHz.

Table 11. RC16 MHz oscillator user trimming

Trimming bit value	Trimming steps	Trimming bit value	Trimming steps
0b011	+3	0b111	-1
0b010	+2	0b110	-2
0b001	+1	0b101	-3
0b000	0	0b100	-4

12.3.2 LSI

The LSI is a low-speed clock line provided by an internal RC circuit. It's interconnected to the independent watchdog (IWDG) circuit, to the auto-wakeup unit (AWU); and it's also an alternative low power clock line for the master clock lines f_{MASTER}, f_{SMED} and f_{ADC}.

The LSI RC can be switched on and off using the LSIEN bit in the internal clock register (CLK_ICKR). The LSIRDY flag in the internal clock register (CLK_ICKR) indicates if the low-

speed internal oscillator is stable or not. At a startup, the clock is in the power-off state until the LSIEN bit is set by hardware (the LSI may be requested by HW at a startup in order to recognize the SWIM entry sequence).

Calibration

Like the HSI RC, the LSI RC device is factory calibrated by ST. However, it is not possible to perform further trimming.

Note: When using the independent watchdog with the LSI as a clock source, in order to guarantee that the CPU will never run on the same clock in case of a corruption, the LSI clock cannot be the master clock if the LSI_EN option bit is reset. Refer to the option bytes section in the datasheet.

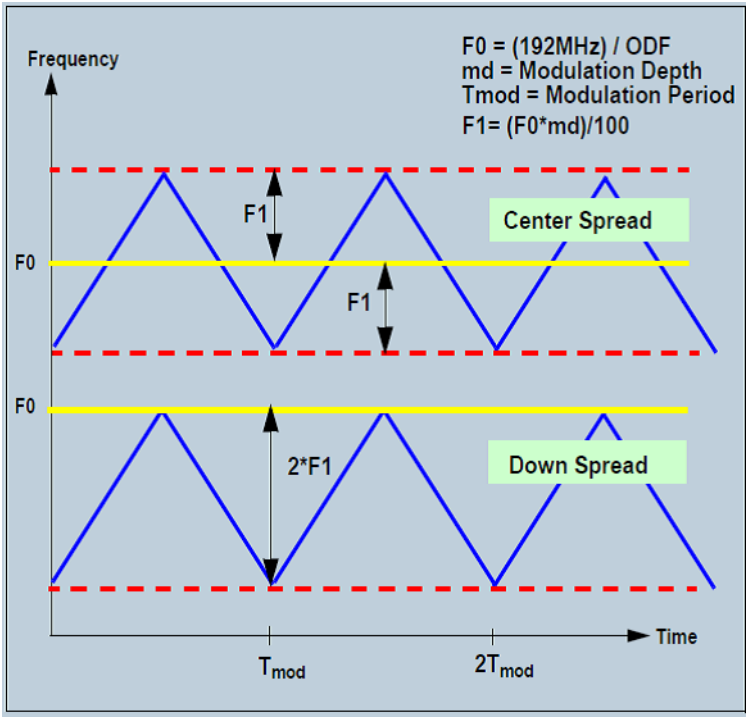
12.3.3 PLL

The PLL provides a high frequency 96 MHz clock used to generate high frequency and accurate PWM waveforms from a 16 MHz input reference clock provided either by the internal HSI RC oscillator or by the external HSE auxiliary input clock line.

If the PLL is configured in the SSCG mode (CLK_PLLR register SCG_CTRL bit = '1'), the PLL generates an output clock modulated in frequency with a TRIANGULAR profile. The modulation frequency (1/Tmod) is 20 kHz and modulation depth (MD) is 0.5%. Center spread or down spread is set using the SPREAD_CTRL bit of the CLK_PLLR register.

Figure 14 shows the SSCG PLL output with time in the two spread modes:

Figure 14. PLL output clock in the two different spread modes



12.3.4 HSE

The high-speed external clock (HSE) is an auxiliary clock source, selectable as a master clock on the f_{MASTER}, f_{SMED} and f_{ADC} clock chains interconnected to several IPs. It gives the user the ability to provide an external clock with the stability offered by oscillator circuits.

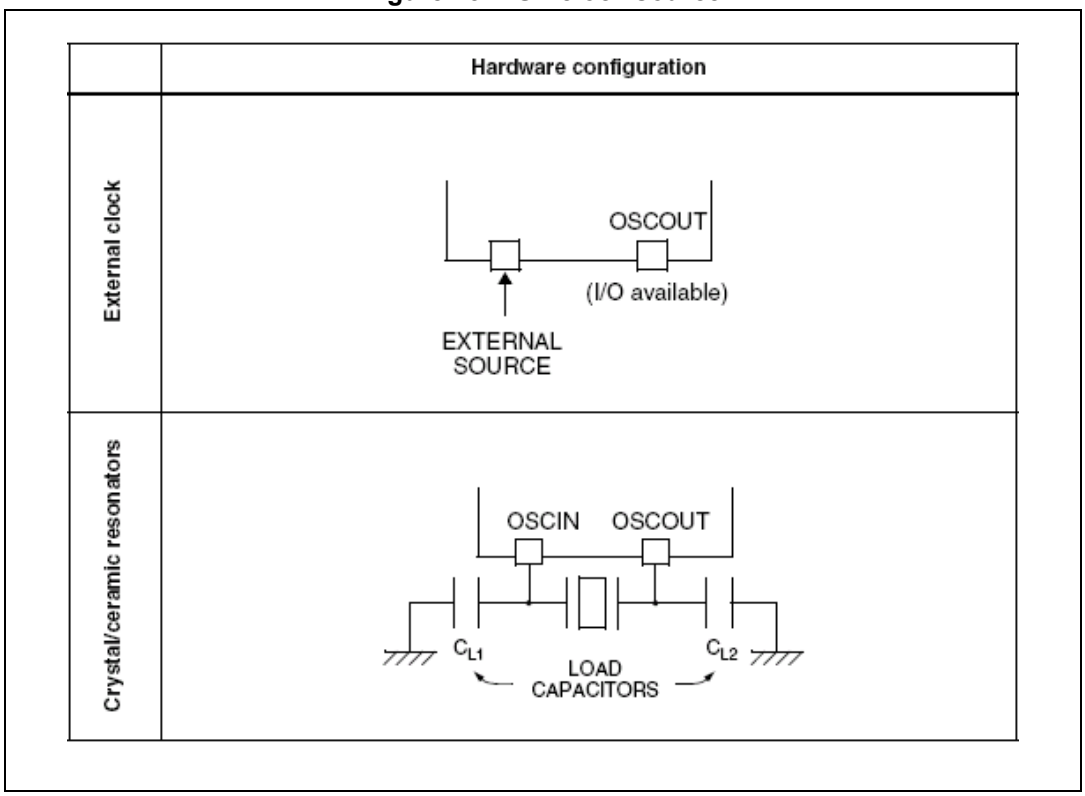
The HSE clock line supports two possible external sources:

- HSE external crystal/ceramic resonator
- HSE user external clock

The HSEOSCIN and HSEOSCOUT signals are multifunction pins configured through the I/O multiplex mechanism described in the product datasheet.

Figure 15 details the HSE different clock source configurations.

Figure 15. HSE clock source



Note: 1. When the HSE is configured as the f_{MASTER} source clock, then the HSE input frequency cannot be higher than 16 MHz.

2. If the HSE is used as a PLL input reference clock, then the HSE input frequency must be equal to16 MHz.

3. If the HSE is an auxiliary clock line for the SMED or the ADC logic, then the input frequency can be configured up to 24 MHz.

4. The resonator and the load capacitors have to be placed as close as possible to the oscillator pins in order to minimize output distortion and startup stabilization time. The loading capacitance values must be adjusted according to the selected oscillator.

External crystal/ceramic resonator (HSE crystal)

The external oscillator interconnected to the HSEOSCIN/HSEOSCOUT signal lines can be chosen in a range from 1 up to 24 MHz.

At a startup the clock signal produced by the oscillator is not stable and by default a delay of 2048 oscillator cycles is inserted before the clock signal is released. The user can program a shorter stabilization time by configuring the HSECNT option byte. Refer to the product datasheet for the option byte description.

The HSERDY flag in the external clock register (CLK_ECKR) indicates if the high-speed external oscillator is stable or not. At a startup, the clock is not released until this bit is set by hardware.

The HSE crystal can be switched on and off using the HSEEN bit in the external clock register (CLK_ECKR).

External source (HSE user-ext)

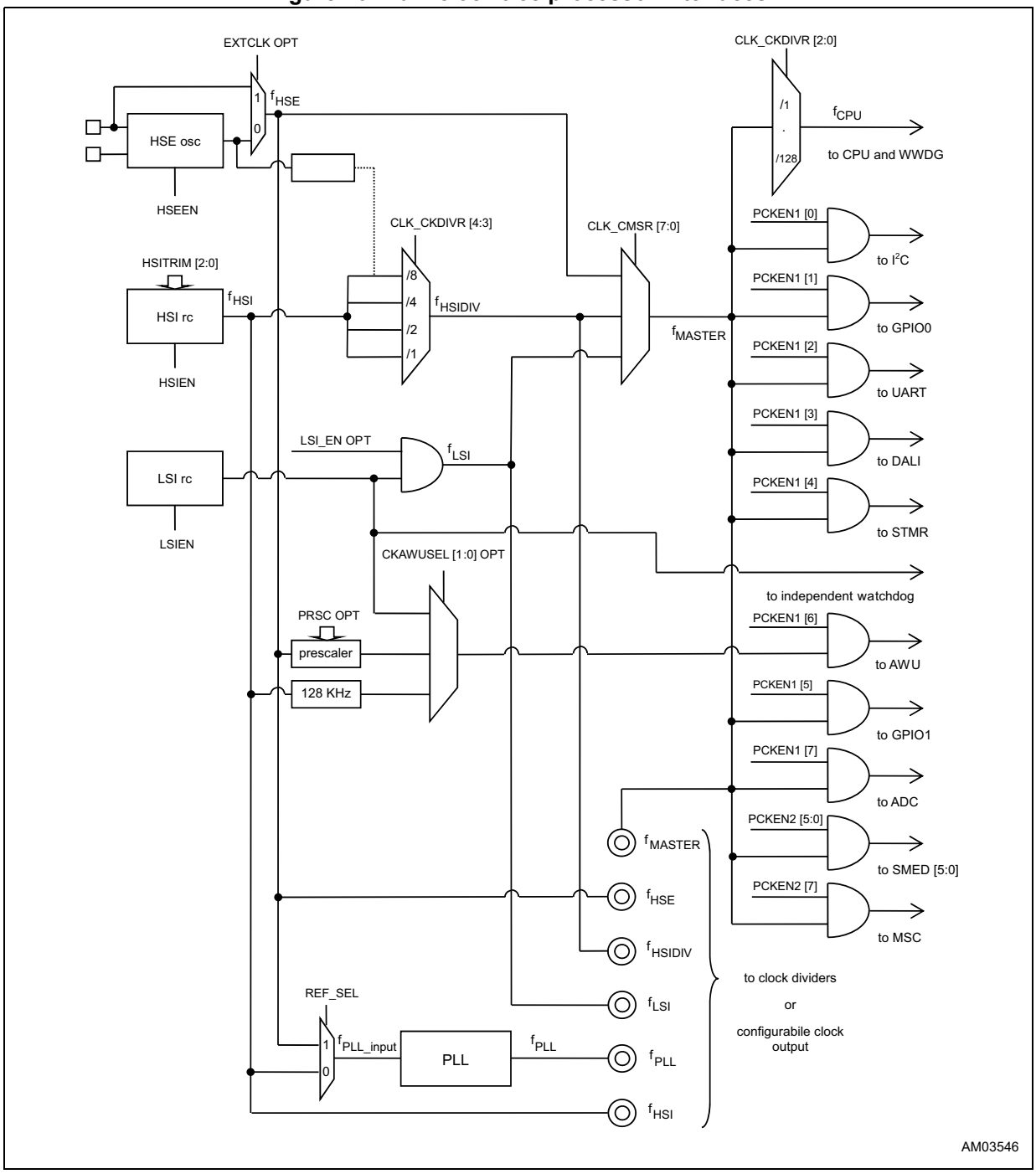
In this mode, an external clock source must be provided. It can have a frequency of up to 24 MHz. Users can select this mode by setting the EXTCLK option bit (refer to the option bytes section of the product datasheet). The external clock signal (square, sinus or triangle) with ~50% duty cycle has to drive the OSCIN pin while the OSCOUT pin is available as a standard I/O (refer to *Figure 16*).

Note: For clock frequencies above 16 MHz, Flash/Data EEPROM access must be configured with a 1 wait state. This is enabled by proper option byte. Refer to the datasheet option byte section.

12.3.5 Internal clock scheme

Figure 16 shows the internal clock tree based on the frequencies provided by the three oscillator circuits HSI, LSI, HSE and by the internal PLL running at 96 MHz. The main clock lines are divided through several clock dividers and controlled with a glitch-free clock gating logic.

Figure 16. Main clock tree processor interfaces



1. The internal frequency f_{MASTER} cannot be higher than 16 MHz in the current implementation.
2. For the SMED and ADC real-time clock dividers refer to *Section 12.7.3 on page 83* and *Section 12.7.2 on page 82*.
3. For the PLL block refer to *Section 12.7.1 on page 82*.
4. For configurable clock output refer to *Section 12.9 on page 85*.

12.4 Oscillators control

12.4.1 Oscillator enable control logic

The clock controller logic carries out the activation of the internal/external oscillator circuits, through the following resources:

- Oscillators on/off control switch (LSIEN, HSIEN and HSEEN bits).
- Flags to monitor oscillator line status (LSIRDY, HSIRDY and HSERDY bits).
- Automatic oscillator activation when executing software commands (e.g.: clock switch, CCO, etc.).
- Hardware protections to secure clock availability.

Refer to *Section 12.11.9: CLK_ICKR (internal clocks control) on page 92* and *Section 12.11.10: CLK_ECKR (external clocks control) on page 94* for details about the oscillator management features.

12.4.2 Oscillator startup

The main issue about driving oscillators concerns the clock signal stabilization time. The CKC logic prevents to deliver the clock signals during the oscillators startup phase: from the clock activation (assertion of the xxxEN bit) until the clock stabilization time (xxxRDY assertion).

Where xxx is in the place of: the LSI or HSE or HSI source clock.

The CKC waits for stabilization time by directly counting the generic number of HSI/LSI cycles.

The high-speed external quartz (EXTCLK option bit reset) oscillator is stabilized by counting HSE cycles. Stabilization time can be trimmed by configuring the option byte HSESTAB: byte values are 0.5, 8, 128, 2048 HSE cycles. No stabilization time is waited for the user external clock (EXTCLK option bit set).

Table 12 details the oscillator circuit stabilization cycle numbers.

Table 12. Oscillators stabilization cycles

Oscillator	Stabilization time configuration	Stabilization cycles
HSI	Fixed	2
LSI	Fixed	0
HSE quartz (EXTCLK = '0')	OPTION BYTE: HSESTAB	0.5, 8, 128, 2048
HSE ext (EXTCLK = '1')	None	None

12.5 Master clock switch

12.5.1 Switch trigger

The application changes the system clock master by programming the contents of the CLK_SWR (clock switch) register. As soon as CLK_SWR and CLK_CMSR contents differ, the CKC asserts the SWBSY, switches on (if it was off) the target-clock source, checks the

ready flag of the CLK_SWR oscillator and, as soon as it is true, then the clock switch is executed copying the CLK_SWR content to the CLK_CMSR register.

12.5.2 CPU clock during switch

Both CPU and peripheral clocks are not interrupted during the clock switching phase. The clock simply changes the duty cycle by stretching the low clock period according to both the current and the newer frequency. Note that the clock switch mechanism requires that the new source clock must be present, otherwise this operation is postponed until the new clock is available and the current source clock is maintained.

12.5.3 Glitch filter

When the clock switching occurs, the transition to the new clock is delayed by latency due to glitch filtering synchronization stages. Worst case latency time is 2 clock cycles of the new clock periods.

All generated clocks and their enable signals are passed through two synchronization stages avoiding any possible spurious glitch event on the output clocks.

12.5.4 SWBSY utility

The SWBSY flag is used by the application to be aware that the clock switch process is ongoing. The SWBSY is automatically asserted as soon as the clock switch is triggered and automatically cleared after the conclusion of all clock switch operations, when the CKC is ready to perform successive clock changes. To protect the clock switch mechanism, when the SWBSY flag is active, the CLK_SWR register content is frozen.

In case the clock switch operation gets stuck, never ending or requiring too much long switch time, or in case it's needed to restore the previous clock, the software is able to reset the whole clock switch process by clearing the SWBSY flag register; as a consequence the CLK_SWR will be replaced by its previous content still present in the CLK_CMSR register field, then the previous source clock is restored.

Note: If the SW application is polling the SWBSY flag after the SWI register is written with a new value, a NOP instruction should be inserted before polling this flag to let the previous instruction to be completed. By not doing so, it may be possible that polling is not done at the right time.

12.5.5 Switch enable

The application can enable/disable the clock switch mechanism to effectively change the clock master. The SWEN bit is used for this purpose. Setting the SWEN bit enables the clock switch execution but, in case the CLK_SWR oscillator is not yet stable, CKC logic keeps active the previous clock for the stabilization time required by the new source clock.

In case the application attempts to both clear the SWBSY bit and set the SWEN bit within the same software step, the SWBSY action takes precedence resetting the newer switch operations.

12.5.6 Automatic vs. manual switch

Application achieves “full automatic” clock switch by keeping the SWEN bit enabled just from the beginning of the switch procedure. Ease of use (user only writes CLK_SWR) and

maximum speed are achieved. Otherwise starting with SWEN = '0' lets the application take explicit control of execution timing.

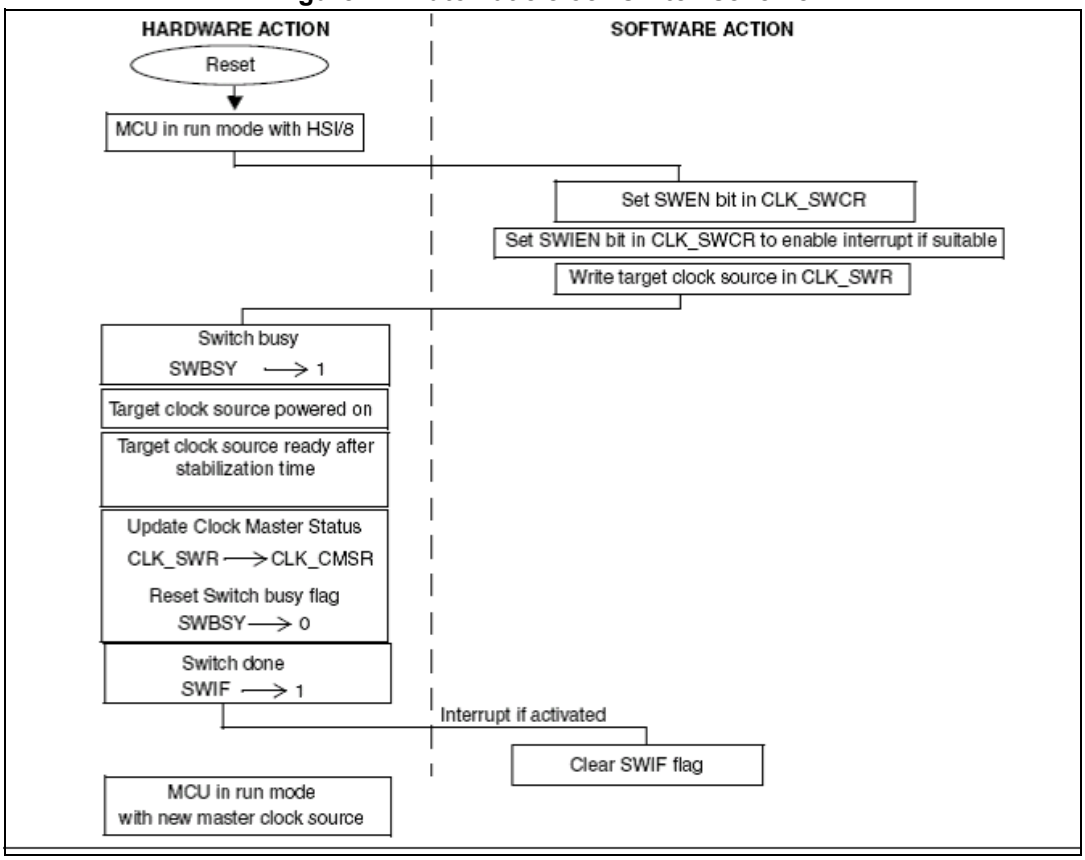
Automatic clock switch

The automatic switching enables the user to launch a clock switch with a minimum number of instructions. The software can continue doing other operations without taking care of the switch event exact time. To enable automatic switching, follow the sequence below (refer to the flowchart in *Figure 17*):

1. Enable the switching mechanism by setting the SWEN bit in the switch control register (CLK_SWCR).
2. Write the 8-bit value used to select the target clock source in the clock master switch register (CLK_SWR). The SWBSY bit in the CLK_SWCR register is set by hardware, and the target source oscillator starts. The old clock source continues to drive the CPU and peripherals.

As soon as the target clock source is ready (stabilized), the content of the CLK_SWR register is copied to the clock master status register (CLK_CMSR). The SWBSY bit is cleared and the new clock source replaces the old one. The SWIF flag in the CLK_SWCR is set and an interrupt is generated if the SWIEN bit is set.

Figure 17. Automatic clock switch scheme



Manual clock switch scheme

The manual switching is not as immediate as the automatic switching but it offers the user a precise control of the switch event time. To enable manual switching, follow the sequence

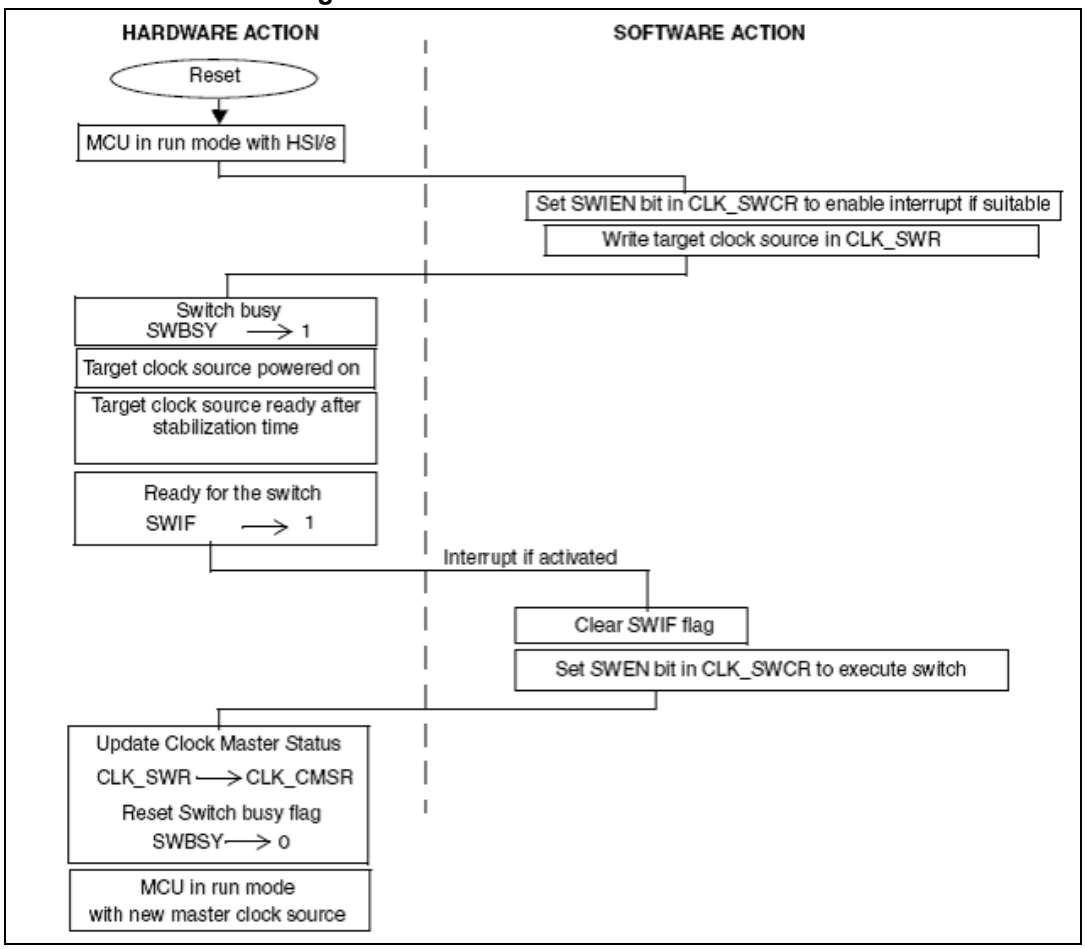
below (refer to the flowchart in *Figure 18*):

1. Write the 8-bit value used to select the target clock source in the clock master switch register (CLK_SWR). Then the SWBSY bit is set by hardware and the target source oscillator starts. The old clock source continues to drive the CPU and peripherals.
2. The software has to wait until the target clock source is ready (stabilized). This is indicated by the SWIF flag in the CLK_SWCR register and by an interrupt if the SWIEN bit is set.
3. The final software action is to set, at the chosen time, the SWEN bit in the CLK_SWCR register to execute the switch.

In both manual and automatic switching modes, the old master clock source will not be powered off automatically in case it is required by other blocks (the LSI RC may be used to drive the independent watchdog for example). The clock source can be powered off by using the bits in the internal clock register (CLK_ICKR) and external clock register (CLK_ECKR).

If the clock switch does not work for any reason, software can reset the current switch operation by clearing the SWBSY flag. This will restore the CLK_SWR register to its previous content (old master clock).

Figure 18. Manual clock switch scheme



12.5.7 Interrupts/flags for application

If clock switch execution is enabled (SWEN = '1') the SWIF flag is asserted by hardware soon after the clock change and, if the SWIE is set, an interrupt is also generated to mean the clock switch has been performed. If clock switch execution is disabled (SWEN = '0') the SWIF flag is asserted by hardware as soon as the new clock (SWR) is ready and, if the SWIE is set, an interrupt is also generated to mean the clock switch machine is ready to execute.

12.5.8 Old CKM after switch

The old clock master is never stopped after the clock switch execution since it can be used by other logic blocks. The application must explicitly switch it off, if this is needed.

12.5.9 Wait mode synchronization

The clock switch can be combined with the WAIT mode in order to synchronize the WAIT awakening interrupt with clock switch execution.

12.6 Peripheral clock gating

In order to limit the dynamic clock switching factor, saving power in the functional mode, the CKC unit provides up to 16 gated clock independently configurable peripheral control lines. The application must explicitly set the respective enable bits of the CLK_PCKENR1 and CLK_PCKENR2 registers to release the target clock. The clock enable is the preliminary step to enable the peripheral logic. The clock of the ITC interrupt controller unit is never gated.

After a device reset, all peripheral clocks are enabled. To enable or disable each peripheral clock, the corresponding PCKEN bit in the peripheral clock gating registers (CLK_PCKENR1 and CLK_PCKENR2) must be set or cleared. Each peripheral must be properly disabled, using the appropriate bit, before stopping the corresponding clock.

The AWU counter is clocked by a specific clock line (LSI or prescaled HSE), ensuring that it is operating during the CPU HALT phase, when the primary clock lines are gated.

Peripheral clocks that can be enabled or disabled are:

- I^2C
- UART
- DALI
- System TIMER
- AWU
- ADC
- SMED5
- SMED4
- SMED3
- SMED2
- SMED1
- SMED0
- MISC

12.7 Clock dividers

To reduce the dynamic power consumption, increasing the IC flexibility, the most important clock lines are individually configurable by clock divisor logic as shown below:

- CPU: $f_{CPU} = f_{MASTER} / 2^n$
  where *n* is the value of CLK_CKDIVR[2:0] register.
- SMED<m>: $f_{SMED<m>} = f_{SMED\_SOURCE<m>} / 2^n$
  where *m* is the SMED instance number 5:0; *n* is the value of CLK_SMD<m>[6:4] registers.
- ADC: $f_{ADC} = f_{ADC\_SOURCE} / (n + 1)$
  where *n* is the value of CLK_ADCR[7:4] register.
- CCO: $f_{CCO} = f_{CCO\_SOURCE} / n + 1$
  where *n* is the value of CLK_CCODIVR[7:0] register.
- AWU: $f_{AWU} = f_{AWU\_SOURCE} / 2^n$
  where *n* is the value of CLK_AWUDIVR[3:0] register and f_{AWU_SOURCE} clock is provided by the LSI or HSI prescaled (128 kHz) or by the HSE prescaled (128 kHz) clock lines.
- PLL: $f_{PLL\_PRESCALED\_} = f_{PLL}$ (96 MHz) / (CLK_PLLDIVR[1:0] (clock_divisor: 4, 5, 6, 7) * 2^n)
  where *n* is the PLL clock prescaler provided by the CLK_PLLDIVR[4:3] register field.

12.7.1 PLL clock divider

The PLL output frequency can be prescaled through the CLK_PLLDIVR register to extend the range of frequencies that can be supplied by the CCO auxiliary source clock. *Table 13* summarizes the configurable frequency values.

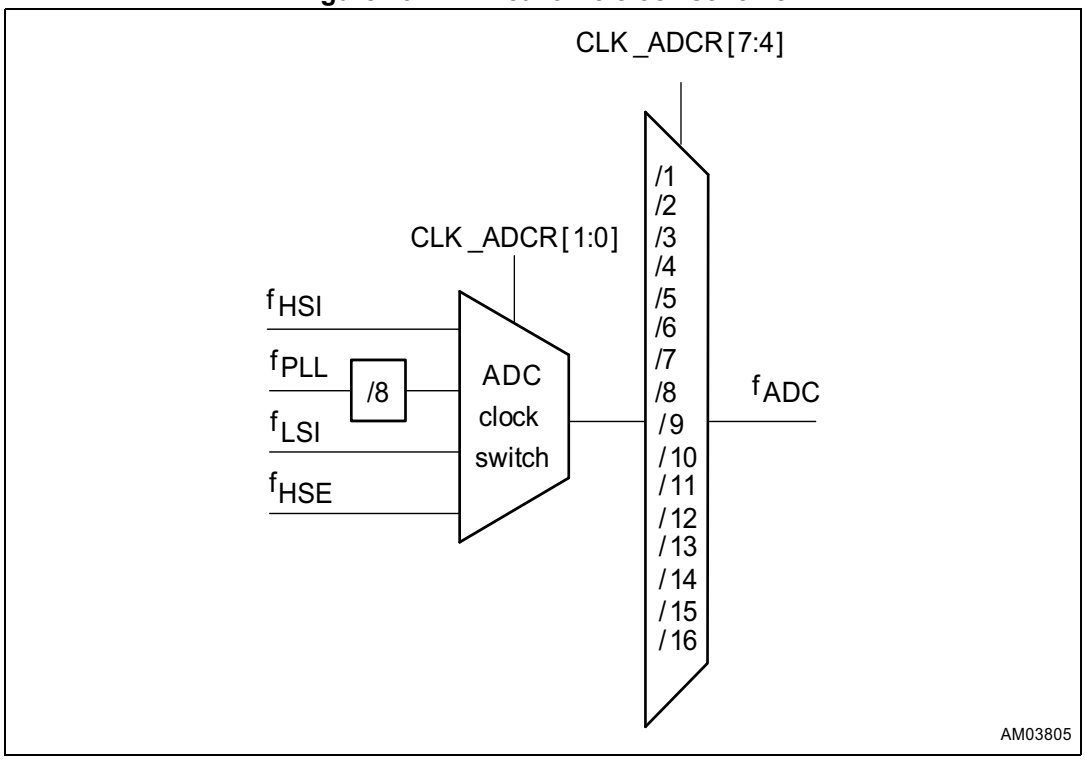
Table 13. PLL output frequency divisor

PLL at 96 MHz output frequency divisor (MHz)				
CLK_PLLDIVR[1:0]	**CLK_PLLDIVR[4:3]**			
	00 (/1)	**01 (/2)**	**10 (/4)**	**11 (/8)**
00 (/4)	24	12	6	3
01 (/5)	19.2	9.6	4.8	2.4
10 (/6)	16	8	4	2
11 (/7)	13.71	6.85	3.42	1.71

12.7.2 ADC real-time clock divider

The ADC conversion frequency is configured by the CLK_ADCR CKC register; the bit fields 1-0 select the ADC clock source, while the bit fields 7-4 configure the clock post divisor feature. *Figure 19* shows an outline view of the ADC clock scheme.

Figure 19. ADC real-time clock scheme



Note: 1. The f_{ADC} cannot be higher than 6 MHz.

2. The HSE auxiliary clock can be configured up to 24 MHz, but in this case the clock post divisor factor has to be configured in order to match the f_{ADC} maximum frequency.

3. If the HSI is the ADC default clock then it has to be divided by 3 (5.3 MHz).

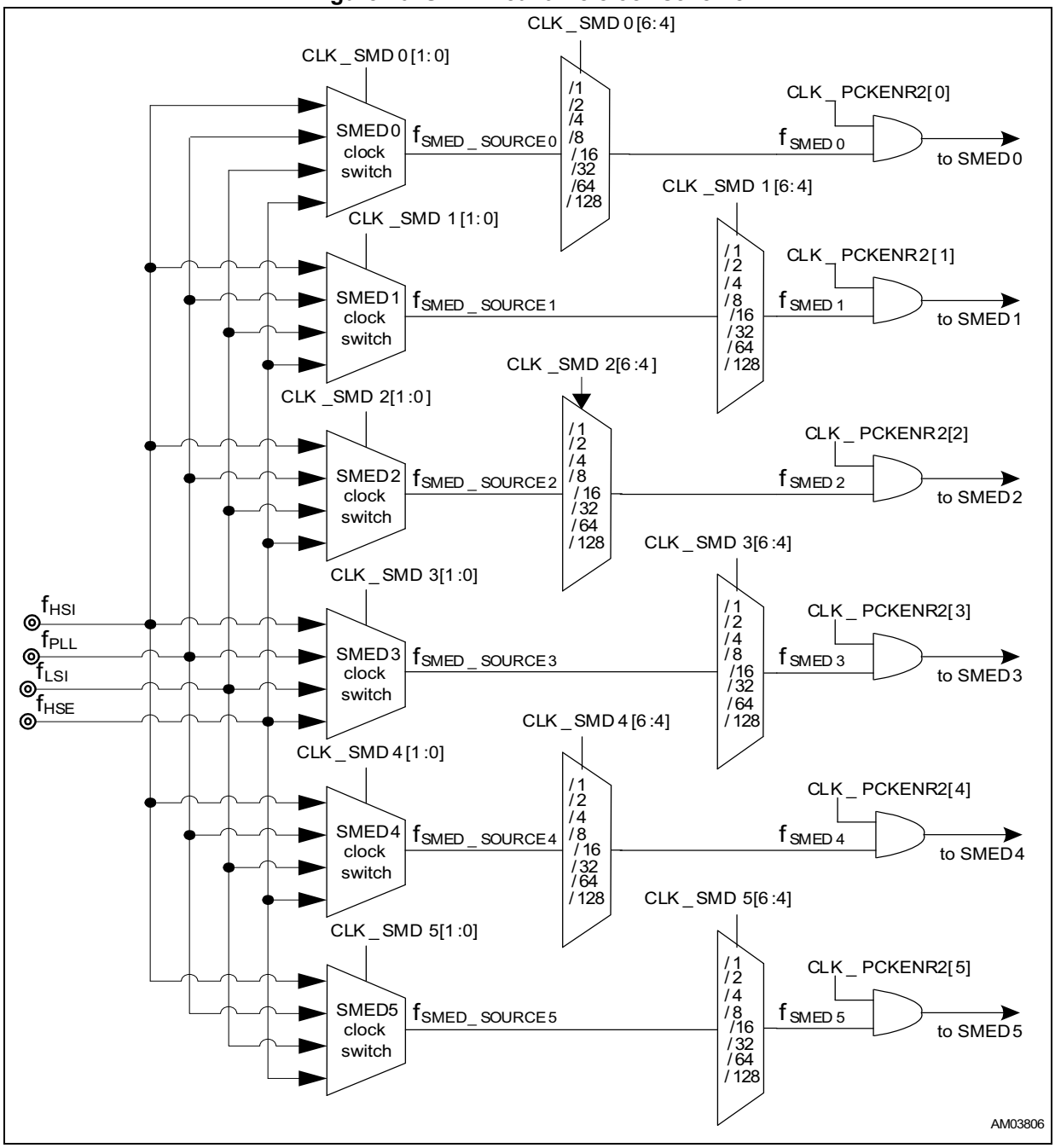
4. The PLL frequency is pre-divided by 8 (12 MHz), then it must be post-divided by 2 to provide the 6 MHz.

12.7.3 SMED real-time clock dividers

The SMED peripherals work with two asynchronous clock lines: a f_{MASTER} system clock (synchronous with CPU clock - 16 MHz default value) used for the microcontroller interface, and a RTC clock (real-time clock) configurable up to 96 MHz used to control the SMED FSM (finite state machine) logic unit and the external input events. The RTC structure is shown in *Figure 20*.

The six SMEDs belong to different clock domains and each FSM can run with an independent clock; the FW application has to configure the clock frequency by programming the source clock and the post divider registers of every used SMEDs. The clock post divider logic scales the input clock from 1 up to 128 with steps of power of 2 (1, 2, 4, 8, 16, 32, 64 and 128); while the RTC base clock can be chosen among four possible sources: f_{HSI} (16 MHz; default source), f_{PLL} (96 MHz functional sources), f_{HSE} (expansion clock), f_{LSI} (low frequency source). Each RTC root clock can be gated off when the relative SMED block is not used. *Figure 20* shows an outline view of SMED clock structures.

Figure 20. SMED real-time clock scheme



Note: The HSE external clock line can be configured up to 24 MHz.

12.8 Clock security system (CSS)

The clock security system (CSS) monitors any possible HSE crystal clock source failure when f_{MASTER} is provided by a HSE crystal oscillator. Whenever the HSE clock fails due to a broken or disconnected resonator or for any other fail reason, the clock controller activates a stall-safe recovery mechanism by automatically switching f_{MASTER} to the auxiliary clock

source (HSI/8). Once selected, the auxiliary source clock remains enabled until the MCU is reset.

The user can enable the clock security system by setting the CSSEN bit in the clock security system register (CLK_CSSR). For safety reason, once the CSS is enabled it cannot be disabled until the next reset occurs.

The following conditions must be met so that the CSS can detect HSE quartz crystal failures:

- HSE crystal on: (HSEEN = '1' in the external clock register (CLK_ECKR))
- HSE oscillator in quartz crystal configuration (EXTCLK option bit is set)
- CSS function enabled: (CSSEN = '1' in the CLK_CSSR register)

If the HSE is the current clock master when a failure is detected, the CSS performs the following actions:

- The CSSD bit is set in the CLK_CSSR register and an interrupt is generated if the CSSIEN bit is set.
- The clock master status register (CLK_CMSR), the clock master switch register (CLK_SWR) register and the HSIDIV[1:0] bits in the clock divider register (CLK_CKDIVR) are set to their reset values (CKM[7:0] = SWI[7:0] = 0xE1). HSI/8 becomes the master clock.
- The HSIEN bit in the internal clock register (CLK_ICKR) is set (HSI on).
- The HSEEN bit in the external clock register (CLK_ECKR) is cleared (HSE off).
- The AUX bit is set to indicate that the HSI/8 auxiliary clock source is forced.

The user can clear the CSSD bit by software but the AUX bit is cleared only by reset.

To select a faster clock speed, you can modify the HSIDIV[1:0] bits in the CLK_CKDIVR register after the CSSD bit in the CLK_CSSR register is cleared.

If the HSE is not the current clock master when a failure is detected, the master clock is not switched to the auxiliary clock and none of the above actions are performed except:

- The HSEEN bit is cleared in the CLK_ECKR register, the HSE is then switched OFF.
- The CSSD bit is set in the CLK_CSSR register and interrupt is generated if the CSSDIE is also set, it can be cleared by software.

If the HSE is not the current clock master and the master clock switch to HSE is ongoing, the SWBSY bit in the CLK_SWCR register must be cleared by software before clearing the CSSD bit.

If the HSE is selected by the CCOSEL to be in the output mode [see clock-out capability (CCO)] when a failure is detected, the selection is automatically changed to force HSI (HSIDIV) instead of HSE.

12.9 Configurable clock output

The configurable clock output (CCO) when selected provides an auxiliary clock line available on the CCO pin. Users have to select one of the following 14 sources clocks programming the CLK_CCOR register, then the selected frequency can be further reduced by a post divisor logic from 1 up to 256 division factors configuring the CLK_CCODIVR

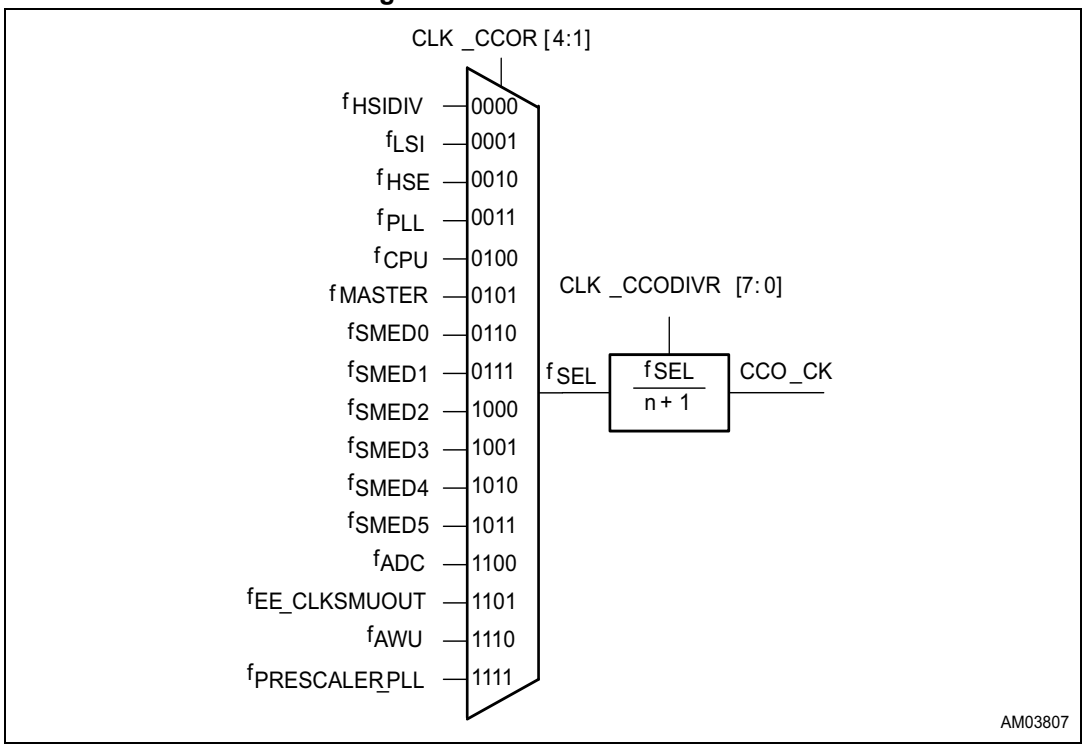
register (refer to *Section 12.11.22: CLK_CCODIVR (CCO divider) on page 104*).

- f_{HSE}
- f_{PLL} (the clock has to be post divided at least by 0x03 in order to match the I/O buffer max. speed)
- f_{HSI}
- f_{LSI}
- f_{MASTER}
- f_{CPU} (current prescaled configuration)
- f_{SMED}<n> (current prescaled configuration; where <n> = 5, 4, 3, 2, 1, 0)
- f_{ADC} (current prescaled configuration)
- f_{FLASH} (ClkSmuOut output from Flash; clock traced for debug purpose)

Note: 50% duty cycle is not guaranteed for all possible prescaled values.

Figure 21 shows the CCO clock scheme.

Figure 21. CCO clock scheme



12.10 CLK interrupts

The following interrupts can be generated by the clock controller:

- Master clock source switch event
- Clock security system event
- PLL unlock event

First two interrupts are individually maskable. The latter is a non maskable interrupt (NMI) type.

Table 14. Clock interrupt event

Interrupt event	Event flag	Enable control bit	Exit from wait	Exit from Halt
Clock security system event (CSS)	CSSD	CSSDIE	Yes	No
Master clock switch event	SWIF	SWIEN	Yes	No
PLL unlock event	LOCKP	PLL_LOCK_INT	Yes (NMI)	N. A.

12.11 Clock registers description

For the base address please refer to the related databook.

12.11.1 CLK_SMD0 (SMED0 clock configuration)

Offset: 0x00

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	SMED_0_DIV [2:0]			RFU	RFU	CK_SW [1:0]	
r	r/w			r	r	r/w	

Bit 1-0: **CK_SW[1:0]** clock switch

00: HSI at 16 MHz frequency selection.

01: PLL at 96 MHz frequency selection.

10: LSI frequency selection.

11: HSE frequency selection[(b)]

Equation 1

$$CLK_{SEL} = CK\_SW\ (HSI, PLL, LSI, HSE)$$

Bit 3-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 6-4: **SMED_0_DIV[2:0]** division factor (n) for:

b. The HSE frequency value must be chosen in accordance with the description present in *Section 12.3.4: HSE on page 74*.

Equation 2

$$CLK_{SMED} = CLKSEL / 2^n$$

Bit 7:**RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.2 CLK_SMD1 (SMED1 clock configuration)

Offset: 0x01

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	SMED_1_DIV [2:0]			RFU	RFU	CK_SW [1:0]	
r	r/w			r	r	r/w	

Bit 1-0: **CK_SW[1:0]** clock switch

00: HSI at 16 MHz frequency selection.

01: PLL at 96 MHz frequency selection.

10: LSI frequency selection.

11: HSE frequency selection(c)

Equation 3

$$CLK_{SEL} = CK\_SW(HSI, PLL, LSI, HSE)$$

Bit 3-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 6-4: **SMED_1_DIV[2:0]** division factor (n) for:

Equation 4

$$CLK_{SMED} = CLK_{SEL} / 2^n$$

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.3 CLK_SMD2 (SMED2 clock configuration)

Offset: 0x02

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	SMED_2_DIV [2:0]			RFU	RFU	CK_SW [1:0]	
r	r/w			r	r	r/w	

c. The HSE frequency value must be chosen in accordance with the description present in *Section 12.3.4: HSE on page 74*.

Bit 1-0: **CK_SW[1:0]** clock switch

00: HSI at 16 MHz frequency selection.

01: PLL at 96 MHz frequency selection.

10: LSI frequency selection.

11: HSE frequency selection(d).

Equation 5

$$CLK_{SEL} = CK\_SW\ (HSI, PLL, LSI, HSE)$$

Bit 3-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 6-4: **SMED_2_DIV[2:0]** division factor (n) for:

Equation 6

$$CLK_{SMED} = CLK_{SEL} / 2^{n}$$

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.4 CLK_SMD3 (SMED3 clock configuration)

Offset: 0x03

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	SMED_3_DIV [2:0]			RFU	RFU	CK_SW [1:0]	
r	r/w			r	r	r/w	

Bit 1-0: **CK_SW[1:0]** clock switch

00: HSI at 16 MHz frequency selection.

01: PLL at 96 MHz frequency selection.

10: LSI frequency selection.

11: HSE frequency selection(e).

Equation 7

$$CLK_{SEL} = CK\_SW\ (HSI, PLL, LSI, HSE)$$

Bit 3-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 6-4: **SMED_3_DIV[2:0]** division factor (n) for:

d. The HSE frequency value must be chosen in accordance with the description present in *Section 12.3.4: HSE on page 74*.
Resources available if the corresponding SMED IP is present on silicon, for details refer to the product datasheet.

e. HSE frequency value must be chosen in accordance with the description present in Section 9.3.4.
Resources available if the corresponding SMED IP is present on silicon device, for details refer to the product datasheet.

Equation 8

$$CLK_{SMED} = CLKSEL / 2^{n}$$

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.5 CLK_SMD4 (SMED4 clock configuration)

Address: 0x04

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	SMED_4_DIV [2:0]			RFU	RFU	CK_SW [1:0]	
r	r/w			r/w	r	r/w	

Bit 1-0: **CK_SW[1:0]** clock switch

00: HSI at 16MHz frequency selection.

01: PLL at 96MHz frequency selection.

10: LSI frequency selection.

11: HSE frequency selection(f)

Equation 9

$$CLK_{SEL} = CK\_SW\ (HSI, PLL, LSI, HSE)$$

Bit 3-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 6-4: **SMED_4_DIV[2:0]** division factor (n) for:

Equation 10

$$CLK_{SMED} = CLKSEL / {}^{2n}$$

Bit 7:**RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.6 CLK_SMD5 (SMED5 clock configuration)

Offset: 0x05

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	SMED_5_DIV [2:0]			RFU	RFU	CK_SW [1:0]	
r	r/w			r	r	r/w	

f. HSE frequency value must be chosen in accordance with the description present in *Section 12.3.4: HSE on page 74*
Resources available if the corresponding SMED IP is present on silicon device, for details refer to the product datasheet.

Bit 1-0: **CK_SW[1:0]** clock switch

00: HSI at 16 MHz frequency selection.

01: PLL at 96 MHz frequency selection.

10: LSI frequency selection.

11: HSE frequency selection(g)

Equation 11

$$CLK_{SEL} = CK\_SW(HSI, PLL, LSI, HSE)$$

Bit 3-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 6-4: **SMED_5_DIV[2:0]** division factor (n) for:

Equation 12

$$CLK_{SMED} = CLKSEL / 2^n$$

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.7 CLK_PLLDIVR (PLL prescaler configuration)

Offset: 0x0A

Default value: 0x2

7	6	5	4	3	2	1	0
RFU			PLL_PRES_DIV [1:0]		RFU	PLL_DIV [1:0]	
r			r/w		r	r/w	

Bit 1-0: **PLL_DIV[1:0]** PLL at 96 MHz clock division factor:

00: $CLK_{PLL\_DIV} = CLK_{PLL} / 4$

01: $CLK_{PLL\_DIV} = CLK_{PLL} / 5$

10: $CLK_{PLL\_DIV} = CLK_{PLL} / 6$

11: $CLK_{PLL\_DIV} = CLK_{PLL} / 7$

Bit 2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 4-3: **PLL_PRES_DIV[1:0]** PLL clock prescaled factor *n*

Equation 13

$$CLK_{PLL\_PRES\_DIV} = CLK_{PLL\_DIV} / 2^n$$

Note: Bits 1 - 0 and 4 - 3 are writable if PLL_EN = '1'.

Bit 7-5: **RFU** reserved; must be kept 0 during register writing for future compatibility.

g. HSE frequency value must be chosen in accordance with the description present in Section 9.3.4. Resources available if the corresponding SMED IP is present on silicon device, for details refer to the product datasheet.

12.11.8 CLK_AWUDIVR (AWU clock prescaler configuration)

Offset: 0x0B

Default value: 0x0

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	AWUDIV [3:0]			
r	r	r	r	r/w			

Bit 3-0: **AWUDIV[3:0]** AWU clock post divider

0000: $f_{ADC} = f_{ADC} / 1$

0001: $f_{ADC} = f_{ADC} / 2$

0010: $f_{ADC} = f_{ADC} / 4$

0011: $f_{ADC} = f_{ADC} / 8$

0100: $f_{ADC} = f_{ADC} / 16$

0101: $f_{ADC} = f_{ADC} / 32$

0110: $f_{ADC} = f_{ADC} / 64$

0111: $f_{ADC} = f_{ADC} / 128$

1xxx: $f_{ADC} = f_{ADC} / 256$

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.9 CLK_ICKR (internal clocks control)

Offset: 0x0C

Default value: 0x01

7	6	5	4	3	2	1	0
RFU	RFU	REGAH	LSIRDY	LSIEN	FHW	HSIRDY	HSIEN
r	r	r/w	r	r/w	r/w	r	r/w

Bit 0: **HSIEN** enable high-speed RC internal oscillator

Enable control bit for the high-speed internal RC oscillator. The HSIEN can be set and clear by software, but it is set and kept asserted by hardware right of priority in such case:

- CSS safe oscillator is on (AUX = '1')
- when exiting HALT/ActiveHALT by fast-halt-wakeup configuration (FHW = '1')
- HSI selected by active CCO (CCOBSY = '1' and CCOSEL = 0x0)
- clock switch to HSI is ongoing (CKM ! = SWI and SWI = HSI)
- HSI is clock master (CKM = HSI)
- HSI requested by SWIM when active for debug

0: disable high-speed internal RC oscillator.

1: enable high-speed internal RC oscillator.

Bit 1: **HSIRDY** high-speed internal oscillator ready

This bit indicates if the high-speed internal RC oscillator is stable or not. It is set and cleared by hardware. The HSI clock is not released as long as this bit is clear. The HSIRDY is automatically cleared when the HSI is turned off.

0: high-speed internal RC is not ready and HSI clock is not available.

1: high-speed internal RC reached stabilization and HSI clock is available.

Bit 2: **FHW** fast Halt/Active-halt wake up

This bit is set and cleared by software.

If the FHW is set the oscillator is automatically switched on (HSIEN = '1') and selected as a next clock master (CKM = SWI = HSI) when resuming from Halt/Active-halt power save modes.

0: fast Halt/Active-halt wakeup mode disabled

1: fast Halt/Active-halt wakeup mode enabled

Bit 3: **LSIEN** enable low-speed internal oscillator

Enable control bit for the low-speed internal RC oscillator. The LSIEN can be set and cleared by software, but it is set and kept asserted by hardware right of priority in such case:

- LSI is requested by SWIM or AWU peripheral
- LSI selected by active CCO (CCOBSY = '1' and CCOSEL = 0x1)
- Clock switch to LSI is ongoing (CKM != SWI and SWI = LSI)
- LSI is the clock master (CKM = LSI)

0: disable low-speed internal RC oscillator

1: enable low-speed internal RC oscillator

Bit 4: **LSIRDY** low-speed oscillator ready

This bit indicates if the low-speed internal RC oscillator is stable or not. It is set and cleared by hardware. The LSI clock is not released as long as this bit is clear. The LSIRDY is automatically cleared when the LSI is turned off.

0: low-speed internal RC is not ready and LSI clock is not available.

1: low-speed internal RC reached stabilization and LSI clock is available.

Bit 5: **REGAH**: regulator power off in Active-halt mode

This bit is set and cleared by software. When it is set, the main voltage regulator is powered off as soon as the MCU enters the Active-halt mode, so the wakeup time is longer.

0: MVR regulator ON in Active-halt mode

1: MVR regulator OFF in Active-halt mode

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.10 CLK_ECKR (external clocks control)

Offset: 0x0D

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	RFU	RFU	HSERDY	HSEEN
r	r	r	r	r	r	r	r/w

Bit 0: **HSEEN** enable high-speed external oscillator

Enable control bit for the high-speed external quartz oscillator. The HSEEN can be set and cleared by software. It is set and kept asserted by hardware in the following cases:

- Clock switch to HSE is ongoing (CKM != SWI and SWI = HSE).
- HSE selected by active CCO (CCOBSY = '1' and CCOSEL = 0x2).
- HSE is clock master (CKM = HSE).

Else, HSEEN is cleared and kept deasserted by hardware right of priority in such a case CSSD = '1'

0: disable high-speed external oscillator.

1: enable high-speed external oscillator.

Bit 1: **HSERDY** high-speed external quartz oscillator ready

This bit indicates if the high-speed external quartz oscillator is stable or not. It is set and cleared by hardware. The HSE clock is not released as long as this bit is clear. The number of stabilization cycles can be configured by the HSESTB bits when the HSE is chosen in quartz configuration (HSECNF = '0'). The HSERDY is soon asserted switching on the HSE user-external clock configuration (HSECNF = '1'). The HSERDY is automatically cleared when the HSE is turned off.

0: high-speed external quartz is not ready and HSE clock is not available.

1: high-speed external quartz reached stabilization and HSE clock is available.

Bit 7-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.11 CLK_PLLR (PLL configuration)

Offset: 0x0E

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	PLL_LOCK_INT	SPREAD_CTRL	SSCG_CTRL	BYPASS	REF_SEL	LOCKP	PLLON
r	r/w	r/w	r/w	r/w	r/w	r	r/w

Bit 0: **PLLON** PLL power-down

0: PLL disabled.

1: PLL enabled.

Bit 1: **LOCKP** PLL lock signal

0: PLL not locked.

1: PLL locked.

Bit 2: **REF_SEL** PLL clock input reference clock selection

0: HSI source clock selection.

1: HSE external source clock selection.

Note: The HSE frequency must be equal to 16 MHz.

Bit 3: **BYPASS** PLL bypass

0: disable PLL bypass.

1: enable PLL bypass.

Bit 4: **SSCG_CTRL PLL** SSCG input

0: output is not frequency modulated.

1: output is frequency modulated with a triangular profile.

Bit 5: **SPREAD_CTRL** PLL spread input

0: center spread in SSCG mode.

1: down spread in SSCG mode.

Bit 6: **PLL_LOCK_INT** PLL lock interrupt enable

0: disable PLL unlock interrupt generation.

1: enable PLL unlock interrupt generation if there is a falling edge on the PLL lock internal signal.

Note: This register is configurable when the PLL_EN option bit is set to '1'. Refer to the option bytes section in the datasheet.

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.12 CLK_CMSR (clock master)

Offset: 0x0F

Default value: 0xE1

7	6	5	4	3	2	1	0
nCKM [3:0]				CKM [3:0]			
r				r			

Bit 3-0: **CKM[3:0]** clock master status bits

These bits are set and cleared by HW and show the system clock master delivered to the CPU and peripherals. The CKM can be changed by hardware in 3 cases only:

- SWI content is copied to the CKM soon after clock switch execution.

- If the fast-halt-wakeup mode is selected (FHW = '1'), the CKM is reset to the HSI when
resuming from HALT/ActiveHALT (slow/fast).

- AUX = '1' is asserted by CSS. In such a case the CKM can be modified again only after external reset.

The system clock legal selections are the following:

0xE1: HSI 16 MHz.

0xD2: LSI 153.6 kHz.

0xB4: HSE external frequency.

0x78: reserved.

Any other values: HSI 16 MHz clock selection.

Note: When the HSE is one of the clock sources that concur to generate the f_{MASTER} its frequency cannot be higher than 16 MHz.

Bit 7-4:**nCKM[3:0]** not (CKM[3:0]); EMC digit protection field

This register field is the complemented value of the CKM[3:0] register field.

12.11.13 CLK_SWR (clock switch)

Offset: 0x10

Default value: 0xE1

7	6	5	4	3	2	1	0
nSWI[3:0]				SWI [3:0]			
r/w				r/w			

Bit 3-0: **SWI[3:0]** clock switch

These bits are used by application to choose/change the system clock by the clock switch procedure. In order to perform clock switch software has to fill these bits with the target oscillator code. SWI content is then frozen and write-protected as long as clock switch operation is ongoing (SWBSY = '1'). However software is enabled to reset clock switch executions by clearing the SWBSY flag: in such a case SWI is restored by its

previous content (the actual CKM). Soon after clock switch execution SWI content is copied to CKM.

SWI is reset to HSI by hardware in such cases:

- The Fast-Halt-wakeup mode is selected (FHW = '1'), SWI is reset to HSI when resuming from Halt/Active-halt (slow/fast).

- AUX = '1' is asserted by CSS. In such a case SWI can be modified again only after external reset.

The next clock master is following:

0xE1: HSI 16 MHz.

0xD2: LSI(1).

0xB4: HSE.

0x78: reserved.

Others encoding: no clock changes. This condition may occurs in case of the SWI wrong bit encoding or SWI corruption; in such cases the SWI is restored with the previous value.

Note: Writing 0xD2 to the SWI register is disabled when LSI_EN option bit is cleared. Refer to the product datasheet option bit section.

Bit 7-4: **nSWI[3:0]** not(SWI[3:0]); EMC digit protection field

This field must be configured with the complemented value of the SWI[3:0] register field.

12.11.14 CLK_SWCR (switch control)

Offset: 0x11

Default value: 0x0

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	SWIF	SWIEN	SWEN	SWBSY
r	r	r	r	rc/w0	r/w	r/w	r/w

Bit 0: **SWBSY** switch busy

This bit indicates, when asserted, that internal clock switch operations are ongoing. It is set and cleared by hardware. Nevertheless software can clear SWBSY in order to reset clock switch operations (target oscillator is broken, stabilization is longing too much, etc…) by restoring SWI = CKM. If at the same time software attempts to set SWEN and clear SWBSY, SWBSY action takes precedence.

0: clock switch operation not in progress.

1: clock switch operation in progress.

Bit 1: **SWEN** switch enable

This bit enables system clock to switch to the clock source pointed by SWI register. It also determines the SWIF bit behavior. It is set and cleared by software. If at the same time asserts SWEN and clears SWBSY, SWBSY takes precedence.

0: disable clock switch execution.

1: enable clock switch execution.

Bit 2: **SWIEN** clock switch interrupt enable

This bit enables, when set, the SWIF bit to generate clock switch interrupt. It is set and cleared by software.

0: disable clock switch interrupt.

1: enable clock switch interrupt.

Bit 3: **SWIF** clock switch interrupt flag

This bit is set by hardware and cleared by software writing '0'. Its meaning depends on the status of the SWEN bit. Refer to *Figure 17 on page 79* and *Figure 18 on page 80*.

- **In manual switching mode (SWEN = '0'):**
  - 0: target clock source not ready
  - 1: target clock source ready
- **In automatic switching mode (SWEN = '1'):**
  - 0: no clock switch event occurred
  - 1: clock switch event occurred

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.15 CLK_CKDIVR (clock dividers)

Address: 0x12

Default value: 0x18

7	6	5	4	3	2	1	0
RFU	RFU	RFU	HSIDIV [1:0]		CPUDIV [2:0]		
r	r	r	r/w		r/w		

Bit 2-0: **CPUDIV[2:0]** CPU clock prescaler

These bits select the CPU clock division factor; they are set and cleared by software:

000: $f_{CPU} = f_{MASTER}$

001: $f_{CPU} = f_{MASTER} / 2$

010: $f_{CPU} = f_{MASTER} / 4$

011: $f_{CPU} = f_{MASTER} / 8$

100: $f_{CPU} = f_{MASTER} / 16$

101: $f_{CPU} = f_{MASTER} / 32$

110: $f_{CPU} = f_{MASTER} / 64$

111: $f_{CPU} = f_{MASTER} / 128$

Bit 4-3: **HSIDIV[1:0]** high-speed internal clock prescaler

These bits select the high-speed internal RC clock division factor. Reset value is HSI/8, then HSI/8 is the user mode startup clock. These bits are normally set and cleared by software, but hardware control overcomes in such cases:

- AUX = CSSD = '1' reset configuration is forced (HSI/8)

00: HSIDIV = HSI

01: HSIDIV = HSI / 2

10: HSIDIV = HSI / 4

11: HSIDIV = HSI / 8

Note: The HSIDIV post-divisor logic provide a clock with duty cycle 50%.

Bit 7-5: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.16 CLK_PCKENR1 (peripherals clock enable)

Offset: 0x13

Default value: 0xFF

7	6	5	4	3	2	1	0
PCKEN17	PCKEN16	PCKEN15	PCKEN14	PCKEN13	PCKEN12	PCKEN11	PCKEN10
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

These bits are written by software to enable or disable the f_{MASTER} clock to the corresponding peripheral.

Bit 0:**PCKEN10** I^2C peripheral clocks enable

1: clock enabled (active clock).

0: clock gated (disable clock).

Bit 1:**PCKEN11** GPIO0 peripheral clock enable.

1: clock enabled

0: clock gated

Bit 2:**PCKEN12** UART peripheral clocks enable

1: clock enabled.

0: clock gated.

Bit 3:**PCKEN13** DALI peripheral clocks enable

1: clock enabled.

0: clock gated.

Bit 4:**PCKEN14** STMR peripheral clocks enable

1: clock enabled.

0: clock gated.

Bit 5:**PCKEN15** GPIO1 peripheral clock enable.

1: clock enabled.

0: clock gated.

Bit 6:**PCKEN16** AWU peripheral clocks enable

1: clock enabled.

0: clock gated.

Bit 7:**PCKEN17** ADC peripheral clocks enable

1: clock enabled.

0: clock gated.

12.11.17 CLK_CSSR (clock security system)

Offset: 0x14

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	CSSD	CSSDIE	AUX	CSSEN
r	r	r	r	rc/w0	r/w	r	r/w0

Bit 0: **CSSEN** clock security system enable

It's enabled by software one time, and cleared only by an external reset.

0: clock security system is off, no detection is provided for HSE

1: clock security system is on, HSE is being monitored by clock detector.

Bit 1: **AUX** auxiliary oscillator connected as clock master

This bit is set and cleared by HW. The AUX stays off if the clock security system is disabled (CSSEN = '0'). Otherwise it is set by hardware after the detection of disturbance on the high-speed external quartz / user external clock provided that one of the following assertion is true:

- CKM = HSE

It is set by hardware and can be cleared only by an external reset. When the AUX is set, the following actions are automatically performed:

- HSIEN = '1' is reset and kept forced
- CKM = SWI = HSI is reset and kept forced
- HSIDIV = 0x3 is reset until CSSD = '1'

0: auxiliary oscillator is off.

1: auxiliary oscillator is on and selected as an actual clock master source.

Bit 2: **CSSDIE** clock security system detection interrupt enable

This bit enables, when set, the CSSD bit to generate clock switch interrupt. It is set and cleared by software.

0: clock security system interrupt disabled.

1: clock security system interrupt enabled.

Bit 3: **CSSD** clock security system detection

This bit is set by HW and cleared by SW. The CSSD stays off if clock security system is disabled (CSSEN = '0'). Otherwise it is set by hardware after the detection of disturbance on the high-speed external quartz provided HSEON = '1'.

The CSSD assertion:

- is cleared by software
- causes HSEEN bit automatically cleared
- causes CCOSEL bit automatically cleared if CCOSEL = 0x2
- generates clock security system interrupt when CSSDIE = '1'

0: CSS is off of no disturbance on the high-speed external quartz / user external clock was detected.

1: disturbances on high-speed external quartz clock was detected by the CSS.

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.18 CLK_CCOR (configurable clock output)

Offset: 0x15

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	CCOBSY	CCORDY	CCOSEL [3:0]				CCOEN
r	r	r	r/w				r/w

Bit 0: **CCOEN** configurable clock output enable

Enable control bit for the configurable clock output. It is set and cleared by software.

0: disable CCO clock.

1: enable CCO clock.

Bit 4-1: **CCOSEL[3:0]** configurable clock output selection

These bits are used by application SW to configure the CCO clock source provided on CCO output. Software has to program this field with the proper target frequency code. The PLL cannot be written as a source clock if the PLL_EN option bit is cleared.

CCOSEL content is frozen and write-protected as long as the CCO clock is being switched on, stabilized and delivered (CCOBSY = '1').

The CCOSEL is automatically reset to the HSI/HSIDIV in such a case:

- CCOSEL = 0x2 and CCOBSY= '1' and CSSD = '1'.

The possible configurable CCO clock output sources are shown below (not all SMED sources clock may be available, depending on specific product):

0x0: HSI

0x1: LSI

0x2: HSE

0x3: PLL

0x4: CPU

0x5: CKM

0x6: SMED0_CK

0x7: SMED1_CK

0x8: SMED2_CK

0x9: SMED3_CK

0xA: SMED4_CK

0xB: SMED5_CK

0xC: ADC_CK

0xD: EE_CLKSMUOUT

0xE: AWU_CK

0xF: PRESCLALED_PLL_CK

Note: 1. Writing 0x3, the CCOSEL register is disabled when the PLL_EN (option bit) is '0'.

2. Before selecting any clock sources it must be previously enabled.

3. The f_{CPU} clock is generated by clock gating logic clocked by the f_{MASTER} clock then the duty is variable and depends on the division factors (the high pulse is longer as the f_{MASTER} clock period, while the lower pulse is variable, the minimum pulse is one f_{MASTER} clock period).

4. The encoding 0xD is reserved and used only for a debug purpose.

Bit 5: **CCORDY** configurable clock output ready

This bit is set and cleared by HW. The CCORDY indicates that the clock requested on CCO output is being delivered.

0: CCO clock is not available.

1: CCO clock is available.

Bit 6: **CCOBSY** configurable clock output busy

This bit is set and cleared by HW. The CCOBSY indicates that the requested CCO clock is being switched on, stabilized or delivered by the CCOEN command. It is set by hardware after the CCOEN = '1' assertion and cleared by hardware after configurable

clock output is shut down. As long as the CCOBSY is set, the CCOSEL bits are write-protected. The CCOBSY is also the same of the CCO alternate function output.

0: CCO is not busy; CCOSEL is configurable by SW.

1: CCO is busy; CCOSEL is not configurable.

Note: The procedure to write the CLK_CCOR is the following:

1. First step: write the CCOSEL register with the CCO disabled (CCOEN = '0'.).

2. Second step: enable the CCO (CCOEN = '1') and re-write the CCOSEL as first step.

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.19 CLK_PCKENR2 (peripherals clock enable)

Offset: 0x16

Default value: 0xFF

7	6	5	4	3	2	1	0
PCKEN27	RFU	PCKEN25	PCKEN24	PCKEN23	PCKEN22	PCKEN21	PCKEN20
r/w	r	r/w	r/w	r/w	r/w	r/w	r/w

Bit 0: **PCKEN20** clock enable for SMED0's processor interface and the RTC (real-time clock) logic.

1: clock enable.

0: clock gated.

Bit 1: **PCKEN21** clock enable for SMED1's processor interface and the RTC logic.

1: clock enable.

0: clock gated.

Bit 2: **PCKEN22** clock enable for SMED2's processor interface and the RTC logic[(h)].

1: clock enable.

0: clock gated.

Bit 3: **PCKEN23** clock enable for SMED3's processor interface and the RTC logic[(h)].

1: clock enable.

0: clock gated.

Bit 4: **PCKEN24** clock enable for SMED4's processor interface and the RTC logic[(h)].

1: clock enable.

0: clock gated.

Bit 5: **PCKEN25** clock enable for SMED5's processor interface and the RTC logic[(h)].

1: clock enable.

0: clock gated.

Bit 6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

h. Clock gating available if the corresponding SMED IP is present on the silicon device; refer to the product datasheet.

Bit 7: **PCKEN27** clock enable of MISC's processor interface.

1: clock enable (must be kept enable in the current silicon version).

0: clock gated.

Note: Clock gating available if the corresponding SMED IP is present on the silicon device; refer to the product datasheet.

12.11.20 CLK_HSITRIMR (HSI calibration trimmer)

Offset: 0x18

Default value: 0x00

7	6	5	4	3	2	1	0
RFU					HSITRIM [2:0]		
r					r/w		

Bit 2-0: **HSITRIM[2:0]** high-speed internal oscillator trimmer

This register is writable by software in order to refine the internal high-speed RC oscillator frequency. The resulting calibration value is achieved on physical output by adding HSITRIM plus HSICAL value.

Bit 7-3: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.21 CLK_SWIMCCR (SWIM clock division)

Offset: 0x19

Default value: 0x00

7	6	5	4	3	2	1	0
RFU							SWIMCLK
r							r/w

Bit 0: **SWIMCLK** SWIM clock division

This bit is set and cleared by SW.

0: enable the SWIM clock division by 2

1: SWIM clock unchanged.

Bit 7-1: **RFU** reserved; must be kept 0 during register writing for future compatibility.

12.11.22 CLK_CCODIVR (CCO divider)

Offset: 0x1A

Default value: 0x00

7	6	5	4	3	2	1	0
CCODIV [7:0]							
r/w							

Bit 7-0: **CCODIV[7:0]** division factor (n) for the CCO clock.

This field is set and cleared by SW. The CCODIV is the CCO clock post divider.

Equation 14

$$CLK_{CCO} = CLK / (n + 1)$$

Note: After reset the CLK_CCODIVR must be programmed before to enable the CCO output (by CCOEN of CLK_CCOR register), then the division factor can be configured on the fly.

12.11.23 CLK_ADCR (ADC clock configuration)

Offset: 0x1B

Default value: 0x20

7	6	5	4	3	2	1	0
ADC_DIV [3:0]				RFU		SEL [1:0]	
r/w				r		r/w	

This register is configurable by SW and defines the ADC conversion frequency

Bit 1-0: **SEL[1:0]** ADC clock selection

00: HSI 16 MHz selected

01: PLL at 96 MHz selected

10: LSI selected

11: HSE selected

Bit 3-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 7-4: **ADC_DIV[3:0]** division factor (n) for ADC selected clock:

Equation 15

$$f_{ADC} = CLK_{SEL} / (n+1)$$

Note: The minimum division factor (n) depends on the CLK_{SEL} selected source clock.

12.12 Clock registers overview

Table 15 shows the clock controller internal registers overview starting from the base address reported in the corresponding device datasheet; for detailed registers description refer to *Section 12.11 on page 87*.

Table 15. Clock controller registers overview

Name	Description	Offset	Type	Reset value
CLK_SMD0	SMED0 RTC clock config. register	0x00	R/W	0x00
CLK_SMD1	SMED1 RTC clock config. register	0x01	R/W	0x00
CLK_SMD2	SMED2 RTC clock config. register(1)	0x02	R/W	0x00
CLK_SMD3	SMED3 RTC clock config. register(1)	0x03	R/W	0x00
CLK_SMD4	SMED4 RTC clock config. register(1)	0x04	R/W	0x00
CLK_SMD5	SMED5 RTC clock config. register(1)	0x05	R/W	0x00
RFU	Reserved	0x06-0x09	-	-
CLK_PLLDIV	PLL divider/prescaler register(2)	0x0A	R/W	0x00
CLK_AWUDIV	AWU divider register	0x0B	R/W	0x00
CLK_ICKR	Internal clock register(2)	0x0C	R/W	0x01
CLK_ECKR	External clock register(2)	0x0D	R/W	0x00
CLK_PLLR	PLL status register(2)	0x0E	R/W	0x01
CKL_CMSR	Clock master status register	0x0F	R	0xE1
CLK_SWR	Clock master switch register	0x10	R/W	0xE1
CLK_SWCR	Switch control register(2)	0x11	R/W	0x00
CLK_CKDIVR	Clock divider register	0x12	R/W	0x18
CLK_PCKENR1	Peripheral clock enable 1	0x13	R/W	0xFF
CLK_CSSR	CSS status register(2)	0x14	R/W	0x00
CLK_CCOR	Configurable clock output reg.(2)	0x15	R/W	0x00
CLK_PCKENR2	Peripheral clock enable 2	0x16	R/W	0xFF
RFU	Reserved	0x17	-	-
CLK_HSITRIMR	HSI calibration trimmer register	0x18	R/W	0x00
CLK_SWIMCCR	SWIM CLK division factor	0x19	R/W	0x00
CLK_CCODIVR	CCO divider register	0x1A	R/W	0x00
CLK_ADCR	ADC status register	0x1B	R/W	0x20
RFU	Reserved	0x1C	-	-

1. A register available if the corresponding SMED logic is present in the product device; for further details refer to the product datasheet.
2. Some bits are protected by a hard write protection mechanism.

13 Power management (PM)

13.1 Overview

The power saving strategy is based on a compromise between performances and the power consumption acceptable by the user application. The IC offers two methods to reduce both the dynamic consumption mainly due to the dynamic clock switching and the static consumption caused by the transistor bias and leakage current:

- Clock switching reduction: used to slow down the system clock and/or gating the clock of unused IPs.
- Operating modes: different operating modes have been implemented to reduce the power consumption.

Note: The power management feature is configured by some CKC registers, for further details refer to *Section 12.11: Clock registers description on page 87**.*

13.2 Clock switching reduction

The clock switching control activities allow decreasing the dynamic power consumption through the following configurable functionalities:

- Peripheral clock gating.
- Slow down the system clocks.

13.2.1 Peripheral clock gating

In functional modes the peripherals not used by the application contribute to the overall power consumption. The input clock signal switching activity drives hidden, power consuming transitions inside registered elements even though doesn't alter the resulting logic state. In order to limit the power consumption each peripheral source clock can be switched off independently when it's not used through the CKC CLK_PCKENR1 and CLK_PCKENR2 internal registers; for further detail about the registers layout refer to *Section 12.11*.

Note: The SW application can program “on the fly” the clock gating functionality.

13.2.2 Slow down clock

Whenever the IC doesn't need to run at full speed to meet the user application performance or in a similar way during a particular FW execution sequence that doesn't not require a higher CPU computational capability, either the system clock f_{MASTER} or/and the f_{CPU} clock can be slowed down increasing the IC power saving.

The system frequency f_{MASTER} is generated by one of the three configurable sources clock lines (refer to *Figure 16: Main clock tree processor interfaces on page 76*): the HSE external crystal oscillator, LSI RC153.6 kHz internal oscillator and the HSI RC 16 MHz internal oscillator, which is the default clock line provided by the system after reset. The f_{MASTER} frequency driven by the HSI source is programmable through the HSIDIV[1:0] bits field of the CLK_CKDIV register, selecting one of the following prescaled values: /1, /2, /4 or /8.

The CPU frequency f_{CPU} is controlled by the CLK_CKDIVR register through the CPUDIV[2:0] bits field that allows configuring a clock division factor from 1 up to 128; the peripheral clocks are not affected by the f_{CPU} frequency scaling.

Note: The SW application can program "on the fly" the division factors.

13.2.3 Analog peripherals power-down

To reduce the leakage caused by the bias current and reference circuitry of the analog macros, it is recommended to turn off the unused peripherals. These may include the ADC converter, DAC reference voltages and PLL. Refer to relative sections for details about the suitable programming register. The following considerations are applied for the ADC and PLL blocks:

- ADC: the user has to assure that this peripheral is in the IDLE state, before turning off the peripheral by programming the PD bit field of the ADC_CFG register or by executing the HALT instruction. This may require postponing the ADC power-down (by adding some delay by SW) until the completion of current conversion operation; for details refer to *Section 27.8.1: ADC_CFG (configuration register) on page 326*.
- ACU: the user may keep the ACU peripheral in power-on even in the HALT mode. This is allowed in order to use the interrupt capability of the comparator related P3 port to wake up the device (available in some devices: refer to the product datasheet).
  - No specific action has to be performed if the ACU is left enabled; however the power consumption increases accordingly (refer to the product datasheet).
  - Otherwise, to power off the ACU peripheral, the MSC_DACCTR register has to be completely cleared before entering the HALT state. When the device awakes, the user has to configure the peripheral through the MSC_DACCTR register. The peripheral is fully stable and operative 400 ns after the user has completed the configuration.
- PLL: the PLL logic has to be put in power-down before entering the HALT state. This may be done by gating the clocks of the peripheral configured with the PLL source clock (refer to *Section 13.2.1: Peripheral clock gating on page 107*) and then clearing the PLLON bit field of the CKC_PLLR register.

When the IC device exits from HALT, the user has to enable the PLL setting the PLLON bit, then poll by SW the LOCKP bit of the CLK_PLLR register until this field is high; after that may restore the peripheral clocks by removing the clock gating configurations. For PLL lock time information refer to the product datasheet.

13.3 Operating modes

The IC supports four different functional operating modes to reduce the dynamic power consumption:

- RUN: functional operating mode, where all system resources are actives.
- WAIT mode: the CPU is halted while peripherals and oscillators are running.
- Active-HALT mode: all the system is halted with the exception of the LSI oscillator feeding the AWU and the IWDG IPs (alternatively the HSE can be selected to clock only the AWU through option bytes).
- HALT mode: all clock activities are stopped (best power saving condition).

The user has to select one of these four operating modes, configuring them to obtain the best compromise between lower power consumption, and faster startup time after a wakeup

event. When the IC device enters in one of the power saving mode (HALT, Active-HALT or WAIT) the IC can return to the RUN operating mode only after triggering an interrupt event.

In the WAIT mode, all internal peripheral and the external lines (P0, P1, P2 and P3 depending on the product) may be configured to generate a wakeup event.

In the HALT mode only the external lines, properly configured as asynchronous level event, can awake the IC device.

In the Active-HALT the external lines and the internal AWU may be configured to trigger the wakeup event.

Table 16 summarizes the main power management features.

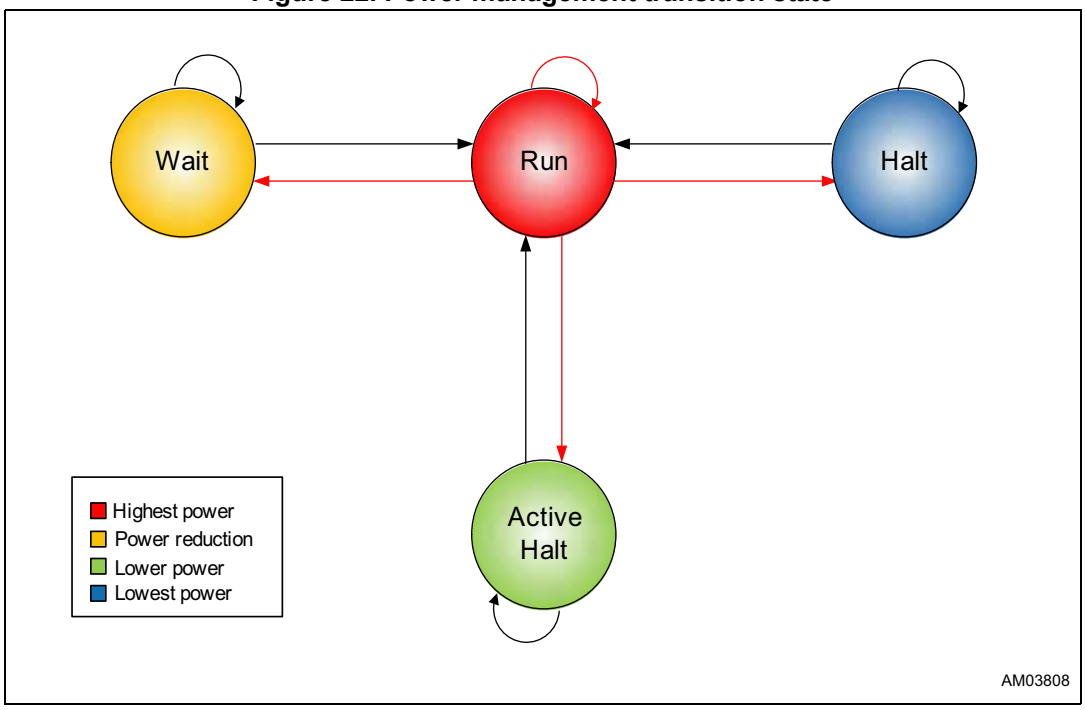
Table 16. Power management scheme

Operating mode	Voltage regulator	Flash	Oscillators PLL	CPU	Peripherals	Entry	wakeup trigger event
Run	MVR	On	On(1)	On(2)	On(3),(4)	-	-
Wait	MVR	On	On(1)	Off(2)	On(3),(4)	Execute WFI(5) instruction	All internal or external interrupts and reset
Active-halt	MVR(6)	On	Off except LSI or HSE(7)	Off	AWU and IWDG on (if activated)	Enable the AWU then execute HALT instruction	AWU or external interrupts and reset
		Off					
	LPVR(8),(9)	On	Off except LSI(10)				
		Off					
Halt	LPVR(8)	On	Off	Off	Off	Execute HALT instruction	External interrupts and reset
		Off					

1. The PLL is enabled by 'PLL_EN' option bit and the CKC_PLLR register.
2. The f_{CPU} clock can be slowed down to reduce the dynamic power switching.
3. The f_{MASTER} clock can be slowed down to reduce the dynamic power switching.
4. Unused peripherals should be disabled turning off their clock gating.
5. The WFI: wait for interrupt.
6. The Active-halt mode configured with fast wakeup functionality.
7. The HSE has to be enabled by programming the CKAWSEL[1:0] option bit to 0 b. (Refer to the product datasheet); this selects the AWU clock f_{AWU} to be provided by the f_{HSE} with a proper scaling factor, also programmable through option byte.
8. When the LPVR is used, the MVR regulator is automatically switched off.
9. The Active-halt mode configured with slow wakeup functionality.
10. Only the LSI clock source can be used, as the HSE clock current consumption is too high for the LPVR.

Figure 22 shows the power management allowed operating transition scheme.

Figure 22. Power management transition state



13.3.1 RUN mode

By default, after a reset or power-on the microcontroller enters in the run mode; the system frequency f_{MASTER} and the CPU frequency f_{CPU} are clocked by the HSI clock line divided by eight. The FW application has to program the proper frequencies by configuring the clock prescaler and the clock gating functionalities according to the target application usage.

The reduction of the power consumption can be achieved decreasing the clock switching activity, further detail can be found in *Section 13.2: Clock switching reduction*.

13.3.2 WAIT mode

The wait for the interrupt mode allows reducing the device power consumption by switching off the CPU core when it is not being used. The wait mode is mainly used when the IC device is waiting for an external or an internal interrupt event which allows the program to continue its execution. Rather than waiting for an interrupt event in the run mode, the device can be switched in the wait mode, subsequently the expected interrupt acts as a wakeup source allowing to restore the run operating mode; further power reduction can be achieved by reducing the clock switching activity as described in *Section 13.2*.

WAIT mode enter

The wait mode is entered by executing the WFI assembly instruction. This stops the CPU, leaving all the other peripherals and the interrupt controller active. When the CPU executes the WFI instruction, the interrupts are automatically enabled.

WAIT mode exit

When an internal or external interrupt request occurs, the CPU wakes up quickly from the wait mode and resumes processing. The list of wakeup interrupt sources is shown in the interrupt vector table described in the product datasheet.

13.3.3 Active-halt mode

The Active-halt mode is similar to the Halt mode except that in addition to the external interrupt it uses the AWU (auto-wakeup) unit to generate a wakeup event internally after a programmed delay, then the processor resumes its regular activity and the AWU counters are stopped.

In the Active-halt mode, the main oscillator, the CPU and almost all the peripherals are stopped. Only the LSI RC oscillator (or alternatively HSE if MVR is kept on) is running to drive the AWU counter.

This operating mode is commonly used to reduce the device power consumption in an interrupt based application where the processor activity is limited to control the peripheral interrupt request then it returns in the Active-halt mode.

Active-halt mode enter

The IC enters in the Active-halt mode when the AWU is enabled (refer to *Section 17: Auto-wakeup unit (AWU) on page 143*) and the CPU executes a Halt instruction.

Note: If the IWDG is enabled before the HALT instruction is executed, the device does not switch to the Halt mode, but, switches to the Halt phase of the Active-halt mode. In this case, if the AWU is not enabled, the MCU does not wake up automatically. The MCU wakes up by an IWDG reset or an external reset.

Active-halt mode exit

The normal run operating mode is resumed when either an AWU or an external interrupt wakeup source is triggered (refer to interrupt exception vector table in the datasheet manual).

Fast wakeup mode

The capability to get a fast wakeup time is very important in the Active-halt mode. It supplements the effect of CPU processing performance by helping to minimize the time MCU, stays in the run mode between two periods in the low power mode and thus reduces the overall average power consumption.

After a wakeup event, the IC device resumes its activities using the clock selected before to enter in the Active-halt mode. The longest wakeup time is achieved when the f_{MASTER} clock is provided by the HSE crystal; that's due to the oscillator stabilization time. To reduce the wakeup time, the device offers a feature called fast clock wakeup which allows starting automatically from the HSI clock after a wakeup event. The user can decide to switch back to the former clock or to stay on the HSI clock.

To enable the fast wakeup mode, set the FHW bit and reset the REGAH bit of the CLK_ICKR register. This mode automatically selects the HSI clock as the f_{MASTER} source clock before to enter into the Active-halt mode; by default this functionality is disabled after reset.

Additional power consumption can be gained programming the Flash in the power-down mode when the IC enters in the Active-halt mode by setting the AHALT bit of the FLASH_CR1 register.

Slow wakeup mode

To further reduce current consumption, in this mode the internal main voltage regulator is powered off automatically when the MCU enters the Halt or Active-halt mode. The MCU core is powered by the low power voltage regulator. The wakeup event restores the MVR power regulator source voltage after the MVR recovery time. To enable the slow wakeup mode the REGAH bit of the CLK_ICKR register has to be set.

Additional power consumption can be gained by programming the Flash in the power-down mode when the IC enters in the Active-halt mode. This is done by setting the AHALT bit of the FLASH_CR1 register.

Note: If the SWIM is enabled, the IC device won't switch off both the power regulators and the high-speed internal oscillator, and will enter in an operating mode similar to the fast Active-halt mode. To switch off these two macros the OSCOFF bit of the SWIM CSR register must be set.

13.3.4 Halt mode

In this mode the master clock is stopped, the CPU and all the peripherals clocked by f_{MASTER} or by derived clocks are disabled. As result, none of the peripherals are clocked and the digital part of the MCU consumes almost no power with the following exceptions:

- HSI not stopped if SWIM is enabled.
- LSI clock is stopped.

The main MVR voltage regulator is switched off to limit the power consumption, only the LPVR regulator source voltage stays active.

By default the Flash should be configured in the power-down mode by keeping the HALT bit of the FLASH_CR1 register clear; so when the device Halt condition is reached the Flash is switched off.

Note: In the Halt mode, all peripherals register values and the RAM contents are preserved; by default the clock configuration remains unchanged.

Halt mode enter

The MCU enters in the Halt mode when a HALT instruction is executed.

Halt mode exit

Wakeup from the Halt mode is triggered by an external interrupt sources, generated by the interrupt detection logic of the GPIO port (refer to the product datasheet) or by an alternate function pin capable to trigger a peripheral interrupt.

Fast wakeup mode

To speed up the wakeup start time the IC device can be configured to enable the fast wakeup functionality as described in the *Fast wakeup mode in Section 13.3.3*.

14 Interrupt controller (ITC)

The interrupt controller is used to control all the interrupt request lines originated by the internal peripherals. The ITC manages up to 32 interrupt lines, and handles:

- Hardware interrupts
- Software interrupt (TRAP instruction)
- Interrupt priority level configuration for any interrupt request line.
- Nested or concurrent interrupt management:
  - Up to 4 software programmable nested levels
  - Up to 32 interrupt vectors fixed by hardware
  - 2 not maskable events: RESET, TRAP
  - User not maskable interrupt NMI (PLL unlock, P0, P1, P2 and auxiliary/basic timers if enabled) (2)
  - Up to 22 external interrupt request lines (P0, P1, P2 and P3) configured on 4 independent interrupt vectors (2, 3)
- P3 interrupts associated to the device's comparators allowing flexible and fast reaction to external analog signals.

Note: 1. All the internal interrupt requests are level type active high.

2. The port P0, P1, P2 and auxiliary/basic timers can be configured to assert maskable or unmaskable interrupt requests.

3. Some interrupts may be not available depending on the product features.

14.1 Functional overview

The interrupt request lines are controlled by 8 internal registers which configure the interrupt request priority level. The controller generates two interrupt output request lines: IRQ maskable interrupt, NMI unmaskable and the interrupt exception vector onto 5 bits which encodes the number of the interrupt request line currently active; all of these signals are interconnected to the CPU.

The encoding vector is generated by priority network logic; the interrupt 31 has the lowest priority. The interrupt vector table is located within address specify on the related product datasheet (DS).

The CPU controls the interrupt sequence through the bit 5 and bit 3 of the condition code (CC) register (I1, I0).

Note: 1. When the CPU executes WFI or HALT instructions all interrupt request are enabled.

2. The SW task with WFI or HALT instructions should avoid the usage of RIM instruction.

14.2 Interrupt masking and processing flow

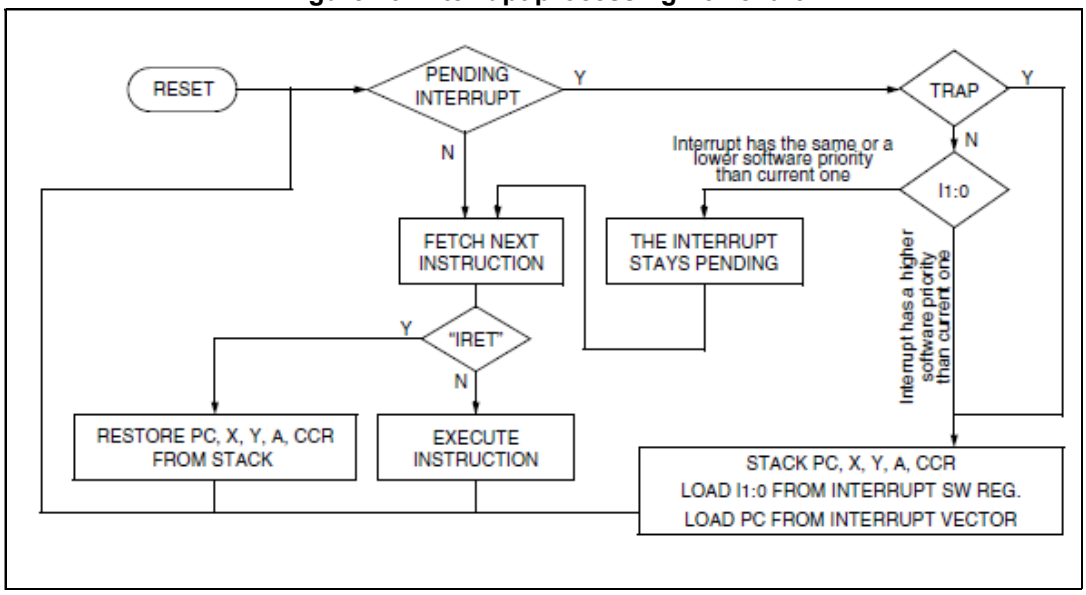
The interrupt masking is managed by bits I1 and I0 of the CCR register and by the ITC_SPRx registers which set the software priority level of each interrupt request (refer to *Table 18: Interrupt priority encoding value*). The interrupt process flow is shown in *Figure 23: Interrupt processing flowchart*.

When an interrupt request is serviced:

1. Normal processing is suspended at the end of the current instruction execution.
2. The PC, X, Y, A and CCR registers are saved onto the stack.
3. Bits I1 and I0 of the CCR register are set according to the values in the ITC_SPRx registers corresponding to the serviced interrupt request.
4. The PC is then loaded with the interrupt vector of the interrupt to the service and the first instruction of the interrupt service routine is fetched.

The interrupt service routine should end with the IRET instruction which causes the content of the saved registers to be recovered from the stack. As a consequence of the IRET instruction, bits I1 and I0 are restored from the stack and the program execution resumes.

Figure 23. Interrupt processing flowchart



Caution: If the interrupt mask bits I0 and I1 are set within an interrupt service routine (ISR) with the instruction SIM, removal of the interrupt mask with RIM causes the software priority to be set to the level 0.

To restore the correct priority when disabling and enabling interrupts inside an ISR, refer to the procedures presented in *Table 17*.

Table 17. Interrupt disable / enable inside the ISR

Disabling the interrupts	Enabling the interrupts
#asm PUSH CC POP ISR_CC(1) SIM #end asm	#asm PUSH ISR_CC(1) POP CC #end asm

Note: The IRS_CC is a variable which stores the current value of the CC register.

14.3 Interrupt priority

The interrupt priority configuration is done by setting the SPR1 and SPR0 bit accordingly with the next descriptions:

VECTxSPR[1:0] vector **"x"** software priority

These bits are written by software to define the software priority of each interrupt. The priority level is specified in *Table 18*.

Table 18. Interrupt priority encoding value

Interrupt level	Priority	VECTxSPR1	VECTxSPR0
Level 0 (normal)	Lowest ↓ Highest	1	0
Level 1		0	1
Level 2		0	0
Non interruptible		1	1

Where "x" is the interrupt vector start from 01 to 31.

The value of the "interrupt pointer" in the interrupt vector table is defined in the corresponding product datasheet.

Note: 1. The interrupt vector table is located at a fixed location (refer to the corresponding product datasheet).

2. Each interrupt vector (except RESET, NMI and TRAP) has two corresponding bits in these registers where its own software priority is stored.

3. The VECTxSPR[1:0] fields of the ITC_SPRx registers has the same meaning of the I1 and I0 bits in the CC register. They have to be configured as described in Table 18.

4. Interrupt level 0 cannot be configured (VECTxSPR[1] = 1, VECTxSPR[0] = 0); in this case the previously stored value is kept. See the register description in Section 14.11: Interrupt controller registers description on page 124.

14.4 Servicing pending interrupts

Several interrupt requests may be pending at the same time. The interrupt to be taken into account is determined by the following two-step process:

1. The highest software priority interrupt is serviced.
2. If several interrupts have the same software priority then the interrupt with the highest hardware priority number is serviced first.

When an interrupt request is not serviced immediately, it is latched and then processed when its software priority combined with the hardware priority becomes the highest one.

Note: 1. The hardware priority is exclusive while the software one is not. This allows the previous process to succeed with only one interrupt.

2. The RESET, NMI and TRAP are considered as having the highest software priority in the decision process.

14.5 Interrupt sources

Two interrupt source types are managed by the STLUX interrupt controller:

- Not maskable interrupts: RESET, TRAP and NMI or external interrupts configured as NMI
- Maskable interrupts: external interrupts configured as IRQ or interrupts issued by the internal peripherals

14.5.1 Not-maskable interrupt sources

Not maskable interrupt sources are processed regardless of the state of bits I1 and I0 of the CCR register (see *Table 18*). PC, X, Y, A and CCR registers are stacked only when a TRAP or NMI interrupts occur. The corresponding vector is then loaded in the PC register and bits I1 and I0 of the CCR register are set to disable interrupts (level 3).

- TRAP (not maskable software interrupt):

This software interrupt source is serviced when the TRAP instruction is executed. It is serviced as a TRAP according to the flowchart shown in *Figure 23*.

A TRAP interrupt does not allow the processor to exit from the Halt mode.

- RESET:

The RESET interrupt source has the highest priorities. This means that all the interrupts are disabled at the beginning of the reset routine.

These have to be re-enabled by the RIM instruction (see *Table 19 on page 124*).

A RESET interrupt allows the processor to exit from the Halt mode.

Refer to *Section 11: Reset control unit (RST) on page 67* for more details about RESET interrupt management.

- NMI (not maskable hardware interrupt):

This hardware interrupt source is serviced immediately as soon as the interrupt request is active in according to the flowchart shown in *Figure 23*. This interrupt kind allows the processor to exit from the Halt mode.

Caution: 1. A TRAP instruction must not be used in a NMI service routine.
2. Configuring more than one NMI HW interrupt sources should be avoided.

14.5.2 Maskable interrupt sources

Maskable interrupt vector sources are serviced if the corresponding interrupt is enabled and if its own interrupt software priority in ITC_SPRx registers is higher than the one currently being serviced (I1 and I0 in CCR register). If one of these two conditions is not met, the interrupt is latched and remains pending.

- External interrupts:
  External interrupts can be used to wake up the MCU from the Halt mode. The device sensitivity to external interrupts can be selected by software through the external interrupt control registers (MSC_CFGP<xy>).
  When several input pins connected to the same interrupt line are selected simultaneously, they are logically ORed.
  When external level-triggered interrupts are latched, if the given level is still present at the end of the interrupt routine, the interrupt remains activated except if it's been cleared by the interrupt service routine.
  Optionally P0, P1 (if available) and P2 interrupts can be configured as NMI.
- Peripheral interrupts:
  Some peripheral interrupts cause the MCU to wake up from the Halt mode. Refer to the interrupt vector table description present in the product datasheet.
  A peripheral interrupt occurs when a specific flag is set in the peripheral interrupt status register and the corresponding interrupt enable bit is set in the peripheral control register.
  The standard sequence for clearing a peripheral interrupt requires an access to the status register followed by a read or write to the associated interrupt register field. The clearing sequence resets the internal latch. A pending interrupt (that is an interrupt waiting to be serviced) is therefore lost when the clear sequence is executed.

14.6 Interrupts and low power modes

All interrupts allow the processor to exit from the wait mode.

Only the external and other specific interrupts allow the processor to exit from the Halt and Active-halt mode (refer to the interrupt vector table in the product datasheet).

When several pending interrupts are present while waking up from the Halt mode, the first interrupt serviced can be only an interrupt able to force the IC device to exit from the Halt mode. If the priority of the highest interrupt pending cannot wake up the device from the Halt mode, it will be serviced next.

If any internal or external interrupt (from a timer for example) occurs while the HALT instruction is executing (the HALT instruction lasts 9 clock cycles; see the STM8 programming manual PM0044), the HALT instruction is completed but the interrupt invokes the wakeup process immediately after the HALT instruction has finished executing. In this case the MCU is actually waking up from the Halt mode to the run mode, with the corresponding delay of $Twu_{(H)}$ as specified in the product datasheet.

14.7 Activation level/low power mode control

The MCU activation level is configured by programming the AL bit in the CFG_GCR register (refer to *CFG_GCR (global configuration register) in* Section 5.6.2).

This bit is used to control the low power modes of the MCU. In very low power applications, the MCU spends most of the time in WFI and is woken up (through interrupts) at specific moments in order to execute a specific task. Some of these recurring tasks are short enough to be treated directly in an ISR (interrupt service routine), rather than going back to the main program.

To cover this case, you can set the AL bit before entering low power (by executing WFI instruction), then the interrupt routine returns directly to the low power mode. The run time/ISR execution is reduced due to the fact that the register context is saved only on the first interrupt.

As a consequence, all the operations can be executed in the ISR in very simple applications. In more complex ones, an interrupt routine may re-launch the main program by simply resetting the AL bit.

For example, an application may need to be woken up by the auto-wakeup unit (AWU) every 50 ms in order to check the status of some pins/sensors/push-buttons. Most of the times, as these pins are not active, the MCU can return to the low power mode without running the main program. If one of these pins is active, the ISR decides to launch the main program by resetting the AL bit.

14.8 Concurrent and nested interrupt management

The STLUX device has two interrupt management modes:

- Concurrent mode
- Nested mode

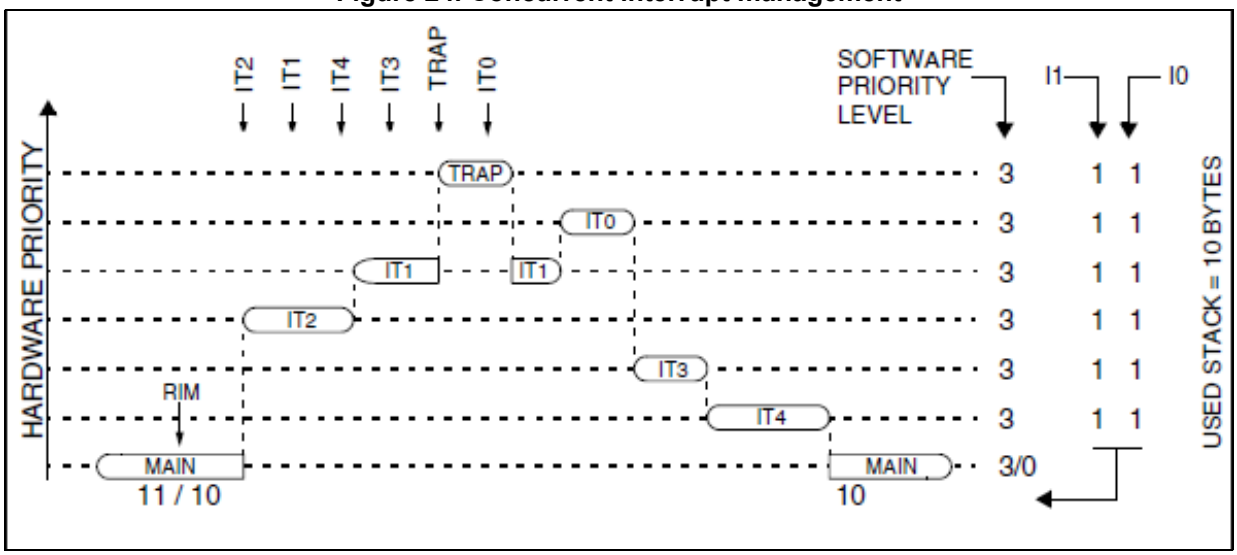
14.8.1 Concurrent interrupt management mode

In this mode, all interrupts have the priority level 3 so that none of them can be interrupted, except by the RESET or TRAP or NMI.

The hardware priority is given in the following order from the lowest to the highest priority, that is: MAIN, IT4, IT3, IT2, IT1, IT0, TRAP, NMI and RESET.

Figure 24 shows an example of the concurrent interrupt management mode.

Figure 24. Concurrent interrupt management



14.8.2 Nested interrupt management mode

In this mode, nested interrupts are allowed during interrupt routines. This mode is activated as soon as an interrupt priority level lower than level 3 is set.

The hardware priority is given in the following order from the lowest to the highest priority, that is: MAIN, IT4, IT3, IT2, IT1, IT0, TRAP, and NMI.

The software priority is configured for each interrupt vector by setting the corresponding I1_x and I0_x bits of the ITC_SPRx register. I1_x and I0_x bits have the same meaning of the CCR register fields I1 and I0.

The level 0 cannot be programmed (I1_x = 1, I0_x = 0). In this case, the previously stored value is kept. For example: if previous value is 0xCF, and programmed value equals 64 h, the result is 44 h.

The RESET, NMI and TRAP vectors have no software priorities. When one of them is serviced, bits I1 and I0 of the CCR register are both set.

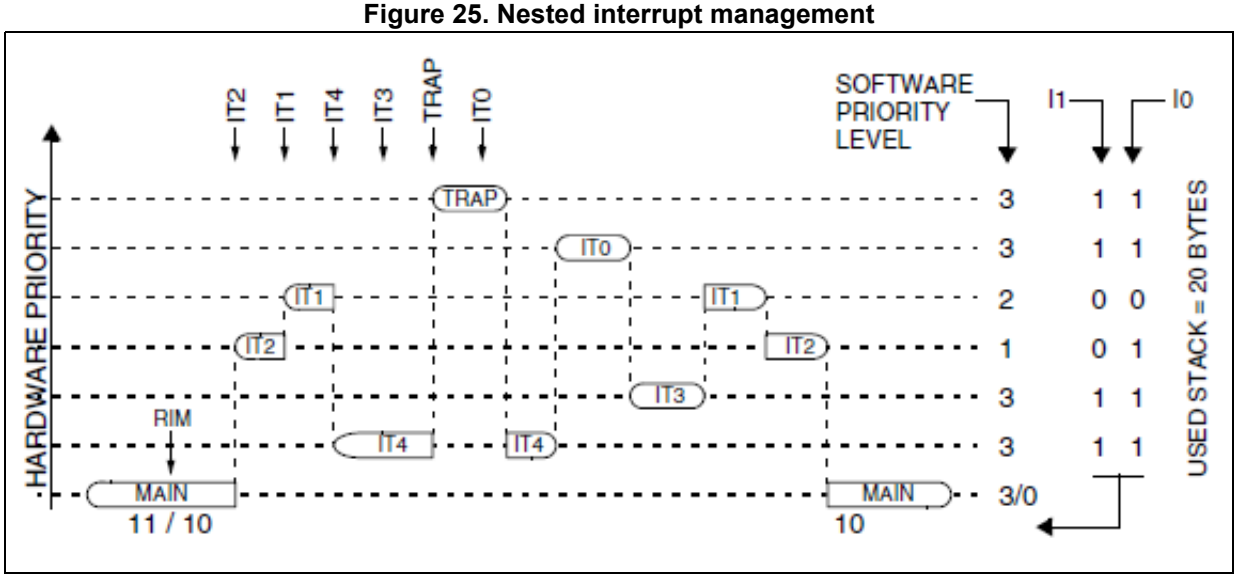
Caution: If bits I1_x and I0_x are modified while the interrupt x is executed, the device operates as follows: if the interrupt x is still pending (new interrupt or flag not cleared) and the new software priority is higher than the previous one, then the interrupt x is re-entered. Otherwise, the software priority remains unchanged till the next interrupt request (after the IRET of the interrupt x).

During the execution of an interrupt routine, the HALT, POPCC, RIM, SIM and WFI instructions change the current software priority till the next IRET instruction or one of the previously mentioned instructions is issued. Refer to *Section 14.10: Interrupt instructions on page 124* for the list of dedicated interrupt instructions.

Figure 25 shows an example of the nested interrupt management mode. In the example the interrupt IT2 has software priority 1, IT1 has software priority 2 while IT0, IT3 and IT4 have software priority 3.

Warning: A stack overflow failure may occur without any SW notification.

Figure 25. Nested interrupt management



14.9 External interrupts

The external interrupts are triggered at the occurrence of a programmed level or edge transition on configurable I/O lines. The interrupts are configured and sensed through capture logic modules (refer to *Section 14.9.4: Edge detector configuration* for further details about the SW configuration of this functionality).

The input lines of the detector modules are in this case directly connected to the I/O lines, while each module drives its own interrupt request line. Each module and its interrupt channel are associated to any port groups.

- MSC_CFGP0<y> registers and MSC_STSP0 register control the port-P0 (GPIO0)
- MSC_CFGP1<y> registers and MSC_STSP1 register control the port-P1 (PWM/GPIO1) if the feature is available in the product
- MSC_CFGP2<y> registers and MSC_STSP2 register control the port-P2 (DIGINs)
- MSC_CFGP3<y> registers and MSC_STSP3 register control the port-P3 (CMPs) if the feature is available in the product

The actual number of ports connected to the edge detector depends on the product. See the product datasheet for details on the port interrupt vector mapping and knowing the ports with interrupt capable. Refer to *Section 14.9.4* for details on the edge detector programming and interrupt handling method.

14.9.1 Interrupt on port 3 (comparators)

In some products it is possible to enable the interrupt capability on the pins associated to the internal fast analog comparators (P3) through the MSC_CFGP3<y> registers. This feature is very valuable because allows the user to configure up to 4 lines (depending on the device) to sense analog signals in a range from 0 up to 1.25 V.

The comparators respond within a short time (50 ns) and their analog output are directly connected to the edge detection logic, allowing a fast management of alarm or sense signals through interrupt service routines.

The interrupts generated by the port-P3 are maskable. Refer to *Section 26: Analog comparator unit (ACU) on page 300* for further details on the comparators and the internal DAC reference voltage generation.

14.9.2 Interrupt polling mode

The external maskable interrupt request lines may be controlled in the polling mode by masking the interrupt request and reading the MSC_STSP<x> to check the presences of an interrupt request flag. The interrupt request is cleared by writing '1' the active interrupt flag of the MSC_STSP<x> register.

The user has to enable the interrupt mask to manage the lines in the polling mode through the INPP<x>_IMSK register field of the miscellaneous register.

- P0: the INPP0_IMSK bit of the MSC_INPP2AUX2 register has to be set, then the P0_CR2[5:0] register is used to enable selectively the interrupt request lines. If the INPP0_IMSK field is cleared, the interrupt is enabled.
- P1: the P1_CR2[5:0] register is used to enable selectively the interrupt request lines of the port-P1 if the feature is available in the product.
- P2: the interrupt masks are directly and individually controlled through the INPP2_IMSK[5:0] field of the MSC_INPP2AUX2 register. The interrupt on the port is masked if the related bit in the INPP2_IMSK[5:0] field is enabled.
- P3: the interrupt mask is enabled on each pin by means of the INT_MSK field of the corresponding MSC_CFGP3<y> register (present in the MSC indirect addressing area) if the feature is available in the product.

The port current signal values may be read through the registers MSC_INPP<x>.

The port-P2 offers the SW controllability through the MSC_INPP2AUX1 register of the internal 47 kΩ pull-up resistances interconnected to the DIGIN[5:0] input lines. For compatibility with the previous product after reset the pull-up resistances are enabled.

Note: For compatibility with previous STLUX385 devices, the interrupt masks of all P0/P2 ports are disabled at reset.

The MSC_CFGP3<y>, MSC_STSP3, MSC_INPP2AUX1 and MSC_INPP2AUX2 registers are addressed by the miscellaneous register indirect mode. Refer to the product datasheet for further description about the indirect addressing mechanism.

Refer toSection 22: General purpose I/O port (GPIO) on page 221 for details on the P<x>_CR2 registers.

14.9.3 Auxiliary/basic timers interrupt functionality

The interrupt feature of the auxiliary/basic timers shares the interrupt capture logic of port-P1. When the timers are enabled through the option bit configuration the input lines of the

capture logic are alternatively assigned to the auxiliary/basic timers according to the following configurations:

- Auxiliary timer (CCO), if available in the product, is associated to the P1[5] interrupt line.
- Basic timer0, if available in the product, is associated to the P1[0] interrupt line.
- Basic timer1, if available in the product, is associated to the P1[1] interrupt line.

For timer availability and configurability refer to the specific product datasheet.

The interrupts have to be programmed in the edge mode through the MSC_CFGP1<y> registers (y = 0, 1, 5). The configuration of the register field P1_CR2[y] selects the interrupt handling mode:

0: polling sequence

1: interrupt mode

The interrupt status request is readable on the MSC_STSP1[y] bit. Refer to *Section 14.9.4* for details on the interrupt capture logic.

The timer interrupts shares the same interrupt vector of the port-P1. Refer to the product datasheet for details on the timer availability and the interrupt vector mapping.

Note: The CCO auxiliary timer functionality is an internal feature that doesn't involve the DIGIN[0] primary signal except in case of clock output configuration (for further information refer to Section 12: Clock control unit (CKC) on page 71).

Refer to Section 22: General purpose I/O port (GPIO) on page 221 for details on the P1_CR2 register.

14.9.4 Edge detector configuration

The edge detector is a module able to accept signals from its input channels and to send an interrupt request based on the configuration and behavior of the input lines.

Every module is based on up to 6 separate input channels and manages an interrupt request line directly connected to the ITC and the internal processor.

The module handles the following features:

- A dedicate interrupt request line connected to the interrupt controller logic
- Up to 6 configurable input sense lines
- Each input sense line can be programmed asynchronous (no clock needed to setup an interrupt event, supporting also wake up feature) or synchronous
- Each input sense line is individually programmable to capture a level or an edge with either polarity

The MSC_CFGP<xy> miscellaneous registers, where x is the associated module instantiation number (i.e the port number ranging from 0 to 3 depending on the product features) and y is the input line number of the module, configuring the single source line P<x>[<y>] pin (y is in a range from 0 up to 5).

These registers allow configuring:

- Interrupt capture modes: edge type, synchronous level, asynchronous level with wake up feature. The selection is achieved through the INT_SEL[1:0] field.
- Select the polarity of the sensed signal individually for each input line (raising/falling edge or high/low level). The polarity is programmed through the INT_LEV field.
- Select the interrupt type between maskable (IRQ) or unmaskable (NMI) for each input line. In the first case the occurrence of the event on the input line raises an interrupt on the module dedicated line. The interrupt type is selected through the INT_TYPE field. Only P0, P1 (where available) and P2 ports can be programmed as a source of the NMI interrupt by enabling the INT_TYPE bit of the MSC_CFGP<xy> registers.
- Enable the interrupt through the INT_ENB field. All interrupt input source lines can be enabled separately.

The MSC_STSP<x> status registers allow identifying the interrupt source when an interrupt relative to the port P<x> is raised (or when the event occurs in the polling mode). There's a direct correlation between the register bit position and the port pin index number.

When the enabled input line matches the configured capture condition (e.g.: falling edge) and the respective interrupt is enabled, the corresponding flag on the MSC_STSP<x> status register is asserted; an interrupt request (IRQ or NMI depending on the configuration) is raised and stays active until cleared by SW.

There are two ways to reset an interrupt source request:

- User can write the status register (MSC_STSP<x>) with a data pattern containing ones in correspondence to the sources event to be reset.
- Alternatively the capture mode can be re configured for the related input line (control register MSC_CFGP<xy>). This procedure resets automatically the interrupt request line of the port associated to the control register.

Note: When an input line is programmed in the *level mode (synchronous or asynchronous), the interrupt flag has to be cleared only after the corresponding interrupt source request line becomes false, otherwise a new interrupt for the same source event is retriggered.*

14.10 Interrupt instructions

Table 19 shows the interrupt instructions related to the CC register.

Table 19. Dedicated interrupt instruction set

Instruction	Description	Functional / example	I1	H	I0	N	Z	C
HALT	Entering Halt mode		1		0			
IRET	Interrupt routine return	Pop CCR, A, X, Y, PC	I1	H	I0	N	Z	C
JRM	Jump if I1:0 = 11 (level 3)							
JRNM	Jump if I1:0<>11							
POP CC	Pop CCR from the stack	Memory => CCR	I1	H	I0	N	Z	C
PUSH CC	Push CC on the stack	CC => memory						
RIM	Enable interrupt (level 0 set)	Load 10 in I1:I0 of CCR	1		0			
SIM	Disable interrupt (level 3 set)	Load 11 in I1:I0 of CCR	1		1			
TRAP	Software trap	Software NMI	1		1			
WFI	Wait for interrupt		1		0			

14.11 Interrupt controller registers description

For details about the interrupt vector number refer to the corresponding product datasheet.

Note: Pay attention that level 0 cannot be programmed (I1_x = 1, I0_x = 0). In this case, the previously mask stored value is kept. For example: if previous value is 0xCF, and programmed value equals 64 h, the result is 44 h.

14.11.1 ITC_SPR0 (interrupt SW priority 0)

Offset: 0x00

Default value: 0xFF

7	6	5	4	3	2	1	0
VECT03SPR1	VECT03SPR0	VECT02SPR1	VECT02SPR0	VECT01SPR1	VECT01SPR0	RFU	RFU
r/w	r/w	r/w	r/w	r/w	r/w	r	r

Bit 1-0: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 7-2: **VECTxSPR[1:0]** vector x software priority

These bits are written by software to define the software priority of the interrupt sources 3 - 1 agreed on the encoding value present in *Table 18: Interrupt priority encoding value on page 115*.

Note: The interrupt0 is a not maskable interrupt, so the bits1 - 0 are unused.

14.11.2 ITC_SPR1 (interrupt SW priority 1)

Offset: 0x01

Default value: 0xFF

7	6	5	4	3	2	1	0
VECT07SPR1	VECT07SPR0	VECT06SPR1	VECT06SPR0	VECT05SPR1	VECT05SPR0	VECT04SPR1	VECT04SPR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 7-0: **VECTxSPR[1:0]** vector x software priority

These bits are written by software to define the software priority of the interrupt sources 7 - 4 agreed on the encoding value present in *Table 18 on page 115*.

14.11.3 ITC_SPR2 (interrupt SW priority 2)

Offset: 0x02

Default value: 0xFF

7	6	5	4	3	2	1	0
VECT11SPR1	VECT11SPR0	VECT10SPR1	VECT10SPR0	VECT09SPR1	VECT09SPR0	VECT08SPR1	VECT08SPR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 7-0: **VECTxSPR[1:0]** vector x software priority

These bits are written by software to define the software priority of the interrupt sources 11-8 agreed on the encoding value present in *Table 18*.

14.11.4 ITC_SPR3 (interrupt SW priority 3)

Offset: 0x03

Default value: 0xFF

7	6	5	4	3	2	1	0
VECT15SPR1	VECT15SPR0	VECT14SPR1	VECT14SPR0	VECT13SPR1	VECT13SPR0	VECT12SPR1	VECT12SPR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 7-0: **VECTxSPR[1:0]** vector x software priority

These bits are written by software to define the software priority of the interrupt sources 15-12 agreed on the encoding value present in *Table 18*.

14.11.5 ITC_SPR4 (interrupt SW priority 4)

Offset: 0x04

Default value: 0xFF

7	6	5	4	3	2	1	0
VECT19SPR1	VECT19SPR0	VECT18SPR1	VECT18SPR0	VECT17SPR1	VECT17SPR0	VECT16SPR1	VECT16SPR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 7-0: **VECTxSPR[1:0]** vector x software priority

These bits are written by software to define the software priority of the interrupt sources 19 - 16 agreed on the encoding value present in *Table 18*.

14.11.6 ITC_SPR5 (interrupt SW priority 5)

Offset: 0x05

Default value: 0xFF

7	6	5	4	3	2	1	0
VECT23SPR1	VECT23SPR0	VECT22SPR1	VECT22SPR0	VECT21SPR1	VECT21SPR0	VECT20SPR1	VECT20SPR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 7-0: **VECTxSPR[1:0]** vector x software priority

These bits are written by software to define the software priority of the interrupt sources 23 - 20 agreed on the encoding value present in *Table 18*.

14.11.7 ITC_SPR6 (interrupt SW priority 6)

Offset: 0x06

Default value: 0xFF

7	6	5	4	3	2	1	0
VECT27SPR1	VECT27SPR0	VECT26SPR1	VECT26SPR0	VECT25SPR1	VECT25SPR0	VECT24SPR1	VECT24SPR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 7-0: **VECTxSPR[1:0]** vector x software priority

These bits are written by software to define the software priority of the interrupt sources 27 - 24 agreed on the encoding value present in *Table 18*.

14.11.8 ITC_SPR7 (interrupt SW priority 7)

Offset: 0x07

Default value: 0xFF

7	6	5	4	3	2	1	0
VECT31SPR1	VECT31SPR0	VECT30SPR1	VECT30SPR0	VECT29SPR1	VECT29SPR0	VECT28SPR1	VECT28SPR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 3-0: **VECTxSPR[1:0]** vector x software priority

These bits are written by software to define the software priority of the interrupt sources 31 - 28 agreed on the encoding value present in *Table 18*.

14.12 Miscellaneous interrupt registers

14.12.1 MSC_CFGP<xy> (port P<x>[<y>] input control register)

Offset: 0x00 + <y> if <x> is 0

Offset: 0x06 + <y> if <x> is 2

Offset: 0x21 + <y> if <x> is 1

Indirect address: 0x0E + <y> if <x> is 3

Default value: 0x00

7	6	5	4	3	2	1	0
RFU			INT_TYPE/INT_MSK	INT_ENB	INT_SEL[1:0]		INT_LEV
r			r/w	r/w	r/w		r/w

<y> is in a range from 0 to 5 if <x> is 0, 1 or 2 (depending on the product)

<y> is in a range from 0 to 3 if <x> is 3 (depending on the product)

Bit 0: **INT_LEV** interrupt request active level used in combination with INT_SEL[1:0]

When INT_SEL[1:0] is equal to '00' or '11', then INT_LEV has the following definition:

0: interrupt level active low.

1: interrupt level active high.

When INT_SEL[1:0] is equal to '10', then INT_LEV selects:

0: interrupt edge triggered active on falling edge.

1: interrupt edge triggered active on rising edge.

Bit 2-1: **INT_SEL[1:0]** interrupt source configuration.

00: level interrupt with the wakeup capability. The interrupt pulse width should be greater than 40 ns and should stay active until the interrupt event is processed by SW.

The interrupt request is cleared with the WC (write clear bit) functionality of MSC_STSP0 register.

01: reserved encoding value; must be not use for future product compatibility.

10: edge triggered interrupt. The interrupt pulse width should be greater than one f_{CPU} clock period. The interrupt request is cleared with the WC (write clear bit) functionality of the MSC_STSP0 register.

11: level interrupt. The interrupt pulse width min. should be greater than one f_{CPU} clock period and should stay active until the interrupt event is processed by SW. The interrupt request is cleared with the WC (write clear bit) functionality of the MSC_STSP0 register.

Bit 3: **INT_ENB** interrupt enable

0: disable interrupt request line.

1: enable interrupt request line.

If <x> is 0, 1, 2:

Bit 4: **INT_TYPE** interrupt type configuration IRQ /NMI

0: interrupt type IRQ.

1: interrupt type NMI unmaskable interrupt request.

If <x> is 3:

Bit 4: **INT_MSK** interrupt mask enable on pin <y> for polling operation

0: IRQ interrupt mask disable (interrupt enable).

1: IRQ interrupt mask enable (interrupt mask).

Bit 7-5: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Note: If <x> = 1, <y> = 0, 1, 5, the MSC_CFGP1<xy> register may control the interrupt of the associated auxiliary/basic timer if the feature is available and enabled via option bytes (see Section 14.9.3: Auxiliary/basic timers interrupt functionality on page 121.

14.12.2 MSC_STSP<x> (port P<x> interrupt status reg.; <x> = 0, 1, 2, 3)

Offset: 0x0C if <x> is 0

Offset: 0x0D if <x> is 2

Offset: 0x27 if <x> is 1

Indirect address: 0x12 if <x> is 3

Default value: 0x00

7 6	5	4	3	2	1	0
RFU	BIT_5_INT/RFU(1),(2)	BIT_4_INT/RFU(1)	BIT_3_INT	BIT_2_INT	BIT_1_INT(2)	BIT_0_INT(2)
r	r/wc(1)	r/wc(1)	r/wc	r/wc	r/wc	r/wc

1. If <x> = 3, the bit [5:4] has to be considered RFU read only.
2. If <x> = 1, these fields are alternatively associated to the interrupts of auxiliary/basic timers where available and enabled via option bytes (see *Section 14.9.3 on page 121*.).

Bit 0: **BIT_0_INT** interrupt pending for port<x>[0] input signal or basic timer 0

0: interrupt cleared.

1:interrupt pending; clearing by writing '1'.

Bit 1: **BIT_1_INT** interrupt pending for port<x>[1] input signal or basic timer 1

0: interrupt cleared.

1: interrupt pending; clearing by writing '1'.

Bit 2: **BIT_2_INT** interrupt pending for port<x>[2] input signal

0: interrupt cleared.

1: interrupt pending; clearing by writing '1'.

Bit 3: **BIT_3_INT** interrupt pending for port<x>[3] input signal

0: interrupt cleared.

1: interrupt pending; clearing by writing '1'.

If <x> is 0, 1, 2:

Bit 4: **BIT_4_INT** interrupt pending for port<x>[4] input signal

0: interrupt cleared.

1: interrupt pending; clearing by writing '1'.

Bit 5: **BIT_5_INT** interrupt pending for port<x>[5] input signal or CCO auxiliary timer

0: interrupt cleared.

1: interrupt pending; clearing by writing '1'.

If <x> is 3:

Bit 4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 5: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

14.12.3 MSC_INPP2AUX1 (INPP2 aux. register1)

Offset: 0x08 (MSC INDIRECT AREA)

Default value: 0x00

7	6	5	4	3	2	1	0
RFU		INPP2_PULCTR [5:0]					
r		r/w					

Bit 5-0: **INPP2_PULCTR[5:0]** this register field configures respectively the INPP2[5:0] pull-up functionality as follows:

0: enable pad pull-up features.

1: disable pad pull-up.

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

14.12.4 MSC_INPP2AUX2 (INPP2 aux. register2)

Idx address: 0x09 (MSC INDIRECT AREA)

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	INPP0_IMSK	INPP2_IMSK [5:0]					
r	r/w	r/w					

Bit 5-0: **INPP2_IMSK[5:0]** INPP2 interrupt mask control register:

0: disable INPP2<n> interrupt mask (interrupt sources are enabled), <n> bit position.

1: enable INPP2<n> interrupt mask (interrupt sources are masked), <n> bit position.

Bit 6: **INPP0_IMSK** INPP0 interrupt mask enable:

0: disable INPP0 interrupt mask (disabled by default for compatibility with STLUX385).

1: enable INPP0 interrupts mask through the GPIO register P0_CR2.

Bit 7:**RFU** reserved; must be kept 0 during register writing for future compatibility.

14.12.5 MSC_INPP2 (Port2 input data register)

Offset: 0x0E

Default value: 0x00

7	6	5	4	3	2	1	0
RFU		DATA_IN [5:0]					
r		r					

Bit 5-0: **DATA_IN[5:0]** port P2 input signals.

These read only registers return the port-2 input signal values after a re-synchronization stage.

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

14.13 Interrupt controller registers overview

Table 20 shows the interrupt controller internal registers starting from the base address specified in the corresponding product datasheet; for detailed registers description refer to *Section 14.11 on page 124*.

Table 20. ITC interrupt control registers

Registers overview				
Name	**Description**	**Offset**	**Type**	**Reset value**
ITC_SPR0	Interrupt SW priority reg-0	0x00	R/W	0xFF
ITC_SPR1	Interrupt SW priority reg-1	0x01	R/W	0xFF
ITC_SPR2	Interrupt SW priority reg-2	0x02	R/W	0xFF
ITC_SPR3	Interrupt SW priority reg-3	0x03	R/W	0xFF
ITC_SPR4	Interrupt SW priority reg-4	0x04	R/W	0xFF
ITC_SPR5	Interrupt SW priority reg-5	0x05	R/W	0xFF
ITC_SPR6	Interrupt SW priority reg-6	0x06	R/W	0xFF
ITC_SPR7	Interrupt SW priority reg-7	0x07	R/W	0xFF

14.14 Miscellaneous interrupt registers overview

Table 21 shows the miscellaneous interrupt registers starting from the base address specified in the corresponding product datasheet; for detailed registers description refer to *Section 14.11 on page 124*.

Table 21. MISC interrupt control registers

Registers overview				
Name	**Description**	**Offset**	**Type**	**Reset value**
MSC_CFGP00	Port P0[0] input control reg.	0x00	R/W	0x00
MSC_CFGP01	Port P0[1] input control reg.	0x01	R/W	0x00
MSC_CFGP02	Port P0[2] input control reg.	0x02	R/W	0x00
MSC_CFGP03	Port P0[3] input control reg.	0x03	R/W	0x00
MSC_CFGP04	Port P0[4] input control reg.	0x04	R/W	0x00
MSC_CFGP05	Port P0[5] input control reg.	0x05	R/W	0x00
MSC_CFGP20	Port P2[0] input control reg.	0x06	R/W	0x00
MSC_CFGP21	Port P2[1] input control reg.	0x07	R/W	0x00
MSC_CFGP22	Port P2[2] input control reg.	0x08	R/W	0x00
MSC_CFGP23	Port P2[3] input control reg.	0x09	R/W	0x00
MSC_CFGP24	Port P2[4] input control reg.	0x0A	R/W	0x00
MSC_CFGP25	Port P2[5] input control reg.	0x0B	R/W	0x00
MSC_STSP0	Port P0 interrupt status reg.	0x0C	R/WC	0x00

Table 21. MISC interrupt control registers (continued)

Registers overview				
Name	**Description**	**Offset**	**Type**	**Reset value**
MSC_STSP2	Port P2 interrupt status reg.	0x0D	R/WC	0x00
MSC_CFGP10	Port P1[0]/AddTim0 input control reg.	0x21	R/W	0x00
MSC_CFGP11	Port P1[1]/AddTim1 input control reg.	0x22	R/W	0x00
MSC_CFGP12	Port P1[2] input control reg.	0x23	R/W	0x00
MSC_CFGP13	Port P1[3] input control reg.	0x24	R/W	0x00
MSC_CFGP14	Port P1[4] input control reg.	0x25	R/W	0x00
MSC_CFGP15	Port P1[5]/aux timer input control reg.	0x26	R/W	0x00
MSC_STSP1	Port P1/Aux/bas timers interrupt status reg.	0x27	R/WC	0x00
MSC_INPP2AUX2	INPP2 aux register2	0x09(1)	R/W	0x00
MSC_CFGP30	Port P3[0] input control reg.	0x0E(1)	R/W	0x00
MSC_CFGP31	Port P3[1] input control reg.	0x0F(1)	R/W	0x00
MSC_CFGP32	Port P3[2] input control reg.	0x10(1)	R/W	0x00
MSC_CFGP33	Port P3[3] input control reg.	0x11(1)	R/W	0x00
MSC_STSP3	Port P3 interrupt status reg. (comparators)	0x12(1)	R/WC	0x00

1. Indirect address mode.

14.15 Interrupt exception vector table

For the interrupt vector address table information refer to the corresponding product datasheet.

15 Independent watchdog (IWDG)

15.1 IWDG introduction

The independent watchdog peripheral can be used to resolve processor malfunctions due to hardware or software failures. It is clocked by the 153.6 kHz LSI internal RC clock source divided by 2, and thus stays active even if the main clock fails.

15.2 IWDG functional description

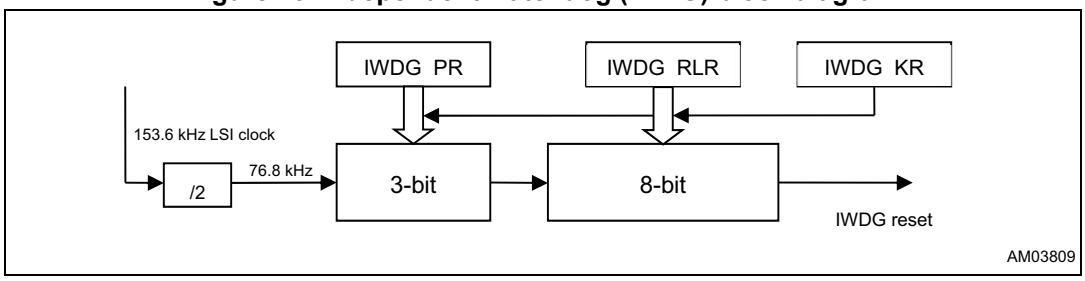
When the independent watchdog is started by writing the value 0xCC in the key register (IWDG_KR), the counter starts counting down from the reset value of 0xFF. When it reaches the end of count value (0x00) a reset signal is generated (IWDG reset).

Once enabled, the independent watchdog can be configured through the IWDG_PR, and IWDG_RLR registers. The IWDG_PR register is used to select the prescaler divider feeding the counter clock. Whenever the KEY_REFRESH value (0xAA) is written in the IWDG_KR register, the IWDG is refreshed by reloading the IWDG_RLR value into the counter and the watchdog reset is prevented.

The IWDG_PR and IWDG_RLR registers are write-protected. To modify them, first write the KEY_ACCESS code (0x55) in the IWDG_KR register. The sequence can be aborted by writing 0xAA in the IWDG_KR register to refresh it.

Refer to *Section 15.5: Independent watchdog registers description* for details on the IWDG registers.

Figure 26. Independent watchdog (IWDG) block diagram



15.2.1 IWDG HW feature

If the hardware watchdog feature has been enabled through the IWDG_HW option byte, the watchdog is automatically enabled at power-on, and generates a reset unless the key register is written by software before the counter reaches the end of the count. Refer to the option byte description in the product datasheet.

15.2.2 IWDG timeout period

The timeout period can be configured through the IWDG_PR and IWDG_RLR registers. It is determined by the following equation:

Equation 16

$$T = 2 \times T_{LSI} \times P \times R$$

where:

T = timeout period

$T_{LSI} = 1/f_{LSI}$

$P = 2^{(PR[2:0] + 2)}$

R = RLR[7:0]+1

The IWDG counter must be refreshed by software before this timeout period expires. Otherwise, an IWDG reset will be generated after the following delay has elapsed since the last refresh operation:

Equation 17

$$D = T + 6 \times T_{LSI}$$

where *D* = delay between the last refresh operation and the IWDG reset.

Table 22. Watchdog timeout period (with LSI/2 counter clock)

IWDG timeout period			
Prescaler divider	**PR[2:0] bits**	**Min. timeout RL[7:0] = 0x00**	**Max. timeout RL[7:0] = 0xFF**
/4	0	52.08 µs	13.33 ms
/8	1	104.1 µs	26.66 ms
/16	2	208.3 µs	53.33 ms
/32	3	416 µs	106 ms
/64	4	833 µs	213 ms
/128	5	1.66 ms	426 ms
/256	6	3.33 ms	853 ms

15.3 IWDG program sequence

To use the IWDG if not being enabled by HW configuration (IWDG_HW bit of MISCUOPT option byte register), the user should perform the following steps:

- Starting the IWDG function by writing 0xCC in the IWDG_KR register (key register).
- Setup the IWDG_RLR (reload counter) by writing 0x55 in the IWDG_KR and then write the desired value on the IWDG_RLR register.
- Setup the IWDG_PR (prescaler register) by writing 0x55 in the IWDG_KR and then write the desired prescaler value in the IWDG_PR register.
- Refresh the IDWG_KR register (writing value 0xAA) before the time expire.

Note: 1. After reset the prescaler is set to 0x00 and the reload counter to 0xFF. This means that between the activation of the IWDG and the first refresh there must be less than 2048 LSI clock cycles corresponding to about 13.333 ms, otherwise the IWDG triggers a hardware reset.

2. If the IWDG is enabled by HW configuration, the IDDG_KR register has to be refreshed within the proper interval time otherwise the IWDG triggers a hardware reset.

To know if the CPU was reset by the IWDG peripheral please see the RST_SR register bit IWDGF (refer to *RST_SR (status register) in Section 11.4*).

15.4 IWDG and AWU concurrent activation

The IWDG and the AWU are the only peripherals that can run also when the system is in the Active-halt mode. If both IWDG and AWU are configured at the same time, the programmer must take care of the following conditions depending on the time of activation of the IWDG (writing 0xCC to IWDG_KR) and of the AWU (execution of the HALT instruction).

- If the AWU is started before that the IWDG has completed its internal synchronization (8 + 2 * prescaler LSI clocks), the system enters the HALT mode with the IWDG in a frozen state, then when the AWU time elapses and the system wakes-up, the IWDG completes its synchronization and after the programmed period will generate a hardware reset if not retriggered.
- If the AWU is started after the IWDG internal synchronization, two different conditions apply depending on the IWDG and AWU timings:

1. AWU timing less than IWDG
   In this case the system enters the HALT mode, it is awakened by the AWU and then the IWDG will generate a hardware reset at its elapsing time if not retriggered.
2. IWDG timing less than AWU
   In this case the system enters the HALT mode, when the IWDG time elapses, an internal reset flag is latched and the system remains in HALT; when also the AWU time is elapsed, the system is awakened and immediately reset.

15.5 Independent watchdog registers description

15.5.1 IWDG_KR (key register)

Offset: 0x00

Default value: -

7	6	5	4	3	2	1	0
KEY [7:0]							
w							

Bit 7-0: **KEY[7:0]** 7 bit key value

0xCC: enables the IWDG function.

0xAA: this constant must be written by software at regular intervals, otherwise the watchdog generates an MCU reset when the counter reaches 0 (if enabled).

0x55:enables access to the IWDG_PR and IWDG_RLR register.

15.5.2 IWDG_PR (prescaler register)

Offset: 0x01

Default value: 0x00

7	6	5	4	3	2	1	0
RFU					PR [2:0]		
r					r/w		

Bit 2-0: **PR[2:0]** prescaler divider

These bits are write access protected. They are written by software to select the prescaler divider feeding the counter clock.

000: $f_{IWDG}/4$

001: f_{IWDG} /8

010: f_{IWDG} /16

011: f_{IWDG} /32

100: f_{IWDG} /64

101: f_{IWDG} /128

110: f_{IWDG} /256

111: reserved

Bit 7-3: **RFU** reserved; must be kept 0 during register writing for future compatibility.

15.5.3 IWDG_RLR (reload counter)

Offset: 0x02

Default value: 0xFF

7	6	5	4	3	2	1	0
RL [7:0]							
r/w							

Bit 7-0: **RL[7:0]** watchdog counter reload value

These bits are write access protected. This field is written by software to define the value to be loaded in the watchdog counter each time the value 0xAA is written in the IWDG_KR register. The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler (IWDG_PR); refer to *Table 22: Watchdog timeout period (with LSI/2 counter clock)*.

15.6 Watchdog registers overview

Table 23 shows the independent watchdog internal registers starting from the base address specified in the corresponding product datasheet; for detailed register description refer to *Section 15.5: Independent watchdog registers description*.

Table 23. IWDG internal registers overview

Name	Description	Offset	Type	Reset value
IWDG_KR	Key register	0x00	W	none
IWDG_PR	Prescaler register	0x01	R/W	0x00
IWDG_RLR	Counter reload register	0x02	R/W	0xFF

16 Window watchdog (WWDG)

16.1 WWDG introduction

The window watchdog is used to detect the occurrence of a software fault, usually generated by external interference or by unforeseen logical conditions, which causes the application program to abandon its normal sequence.

The watchdog circuit generates an MCU reset on expiry of a programmed time period, unless the program refreshes the contents of the down counter before the T6 bit becomes cleared.

An MCU reset is also generated if the 7-bit down counter value (in the control register) is refreshed before the down counter has reached the window register value. This implies that the counter must be refreshed in a limited window.

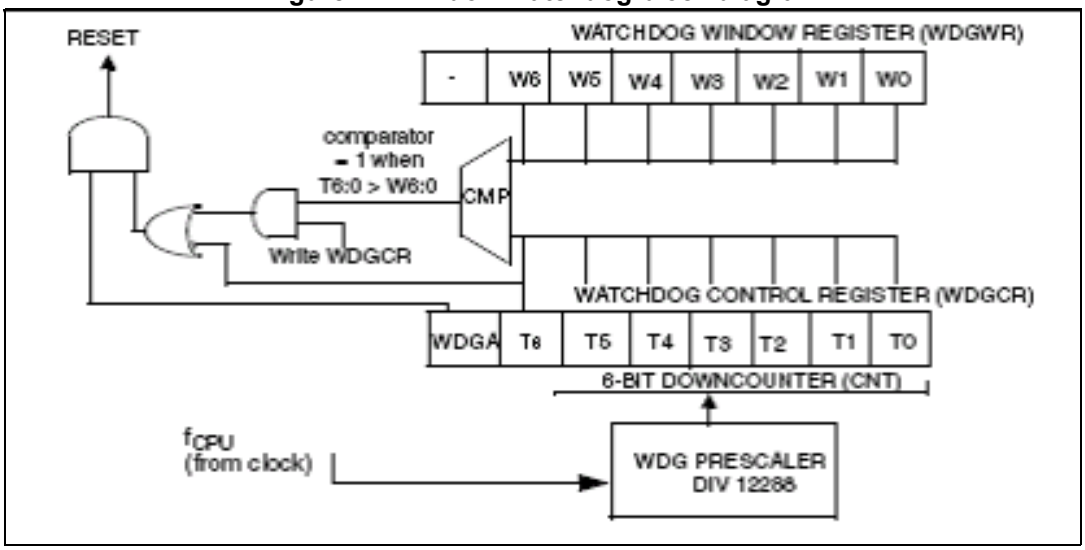
16.2 WWDG main features

- Programmable free running down counter
- Conditional reset:
  - Reset (if watchdog activated) when the down counter value becomes less than 0x40
  - Reset (if watchdog activated) if the down counter is reloaded outside the window
- Hardware/software watchdog activation (selectable by option byte)
- Optional reset on HALT instruction (configurable by option byte)

16.3 WWDG functional description

If the watchdog is activated (the WDGA bit is set) and when the 7-bit down counter (T[6:0] bits) rolls over from 0x40 to 0x3F (T6 becomes cleared), it initiates a reset cycle pulling low the reset pin. If the software refreshes the counter while the counter is greater than the value stored in the window register, then a reset is generated.

Figure 27. Window watchdog block diagram



The application program must write in the WWDG_CR register at regular intervals during normal operation to prevent an MCU reset. This operation must occur only when the counter value is lower than the window register value. The value to be stored in the WWDG_CR register must be between 0xFF and 0xC0:

- Enabling the watchdog:
  When the software watchdog is selected (by option byte), the watchdog is disabled after a reset. It is enabled by setting the WDGA bit in the WWDG_CR register, and then it cannot be disabled again except by a reset.
  When the hardware watchdog is select (by option byte), the watchdog is always active and the WDGA bit is not used.
- Controlling the down counter:
  The down counter is free running; it counts down even if the watchdog is disabled. When the watchdog is enabled, the T6 bit must be set to prevent generating an immediate reset.
  The T[5:0] bits contain the number of increments which represents the delay time before the watchdog asserts a reset signal. The timing varies between a minimum and a maximum value due to the unknown status of the prescaler when writing the WWDG_CR register.
  The window register (WWDG_WR) contains the high limit of the window: To prevent a reset, the down counter must be reloaded when its value is lower than the window register value and greater than 0x3F.

Note: The T6 bit can be used to generate a software reset (the WDGA bit is set and the T6 bit is cleared).

16.4 Watchdog reset on Halt option

If the watchdog is activated and the watchdog reset on Halt option is selected, then the HALT instruction will generate a reset.

16.5 How to program the watchdog timeout

The next formula can be used to calculate the WWDG timeout, t_{WWDG}, expressed in ms:

Equation 18

$$t_{WWDG} = T_{CPU} \times 12288 \times (T[5:0] + 1)$$

where T_{CPU} is the peripheral clock period expressed in ms.

Figure 28. Approximate timeout (ms) at 16 MHz f_{CPU}

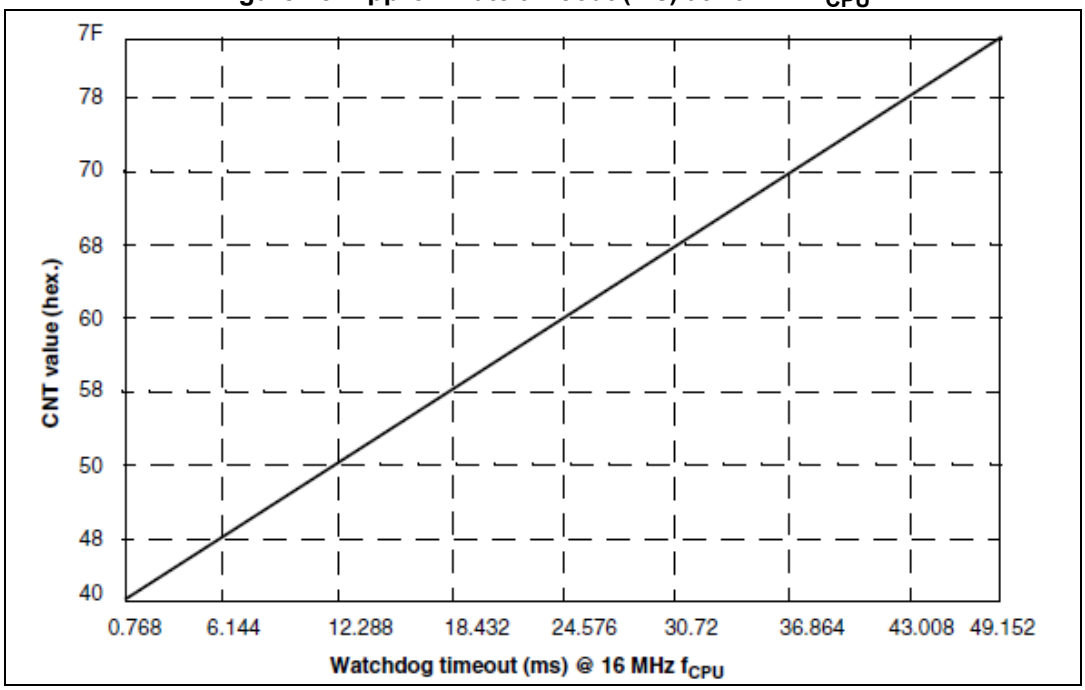
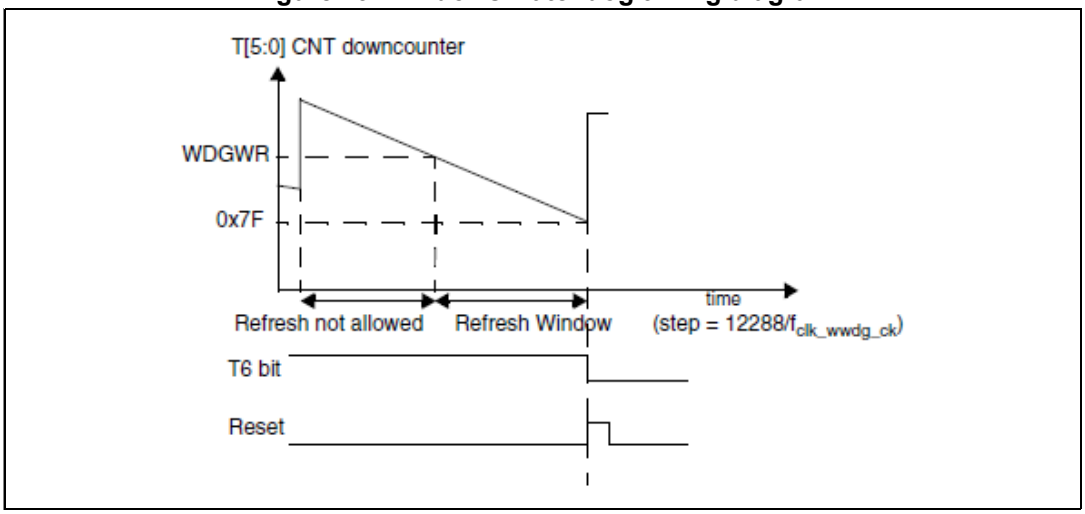


Figure 29. Windows watchdog timing diagram



16.6 WWDG low power modes

Table 24 defines effect of low power modes on WWDG:

Table 24. Low power-on WWDG

Mode	Description	
Wait	No effect on watchdog: the down counter continues to decrement.	
Halt	WWDG_HALT in option byte	
	0	No watchdog reset is generated. The MCU enters the Halt mode. The watchdog counter is decremented once and then stops counting and is no longer able to generate a watchdog reset until the MCU receives an external interrupt or a reset. If an interrupt is received (refer to interrupt vector table mapping to see interrupts which can occur in the Halt mode), the watchdog restarts counting after the stabilization delay. If a reset is generated, the watchdog is disabled (reset state) unless hardware watchdog is selected by option byte.
	1	A reset is generated instead of entering the Halt mode.
Active-Halt	X	No reset is generated. The MCU enters the Active-halt mode. The watchdog counter is not decremented. It stops counting. When the MCU receives an oscillator interrupt or external interrupt, the watchdog restarts counting immediately. When the MCU receives a reset the watchdog restarts counting after the stabilization delay.

16.6.1 Hardware watchdog option

If the hardware watchdog is selected by the option byte, the watchdog is always active and the WDGA bit in the WWDG_CR register is not used. Refer to the option byte description in the product datasheet.

16.6.2 Using Halt mode with the WWDG (WWDG_HALT option byte)

The following recommendation applies if the Halt mode is used when the watchdog is enabled. Before executing the HALT instruction, refresh the WDG counter, to avoid an unexpected WWDG reset immediately after waking up the microcontroller.

16.7 Window watchdog registers description

16.7.1 WWDG_ CR (control register)

Offset: 0x00

Default value: 0x7F

7	6	5	4	3	2	1	0
WDGA	T [6:0]						
rs	r/w						

Bit 0: **T[6:0]** 7 bit counter (MSB to LSB)

These bits contain the value of the watchdog counter. It is decremented every 12288 f_{CPU} cycles (approx.). A reset is produced when it rolls over from 0x40 to 0x3F (T6 becomes cleared).

Warning: When writing to the WWDG_CR register, always write 1 in the T6 bit to avoid generating an immediate reset.

Bit 7: **WDGA** activation bit

This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

0: watchdog disabled.

1: watchdog enabled.

Note: This bit is ignored if the HW watchdog option bit is set.

16.7.2 WWDG_WR (window register)

Offset: 0x01

Default value: 0x7F

7	6	5	4	3	2	1	0
RFU	W [6:0]						
r	r/w						

Bit 6-0: **W[6:0]** 7-bit window value

These bits contain the window value to be compared to the down counter contents.

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

16.8 Window watchdog registers overview

Table 25 details the window watchdog internal registers starting from the base address specified in the corresponding product datasheet (DS); for detailed register description refer to *Section 16.7: Window watchdog registers description*.

Table 25. WWDG internal registers overview

Name	Description	Offset	Type	Reset value
WWDG_CR	Control register	0x00	R/W	0x7F
WWDG_WR	Window register	0x01	R/W	0x7F

17 Auto-wakeup unit (AWU)

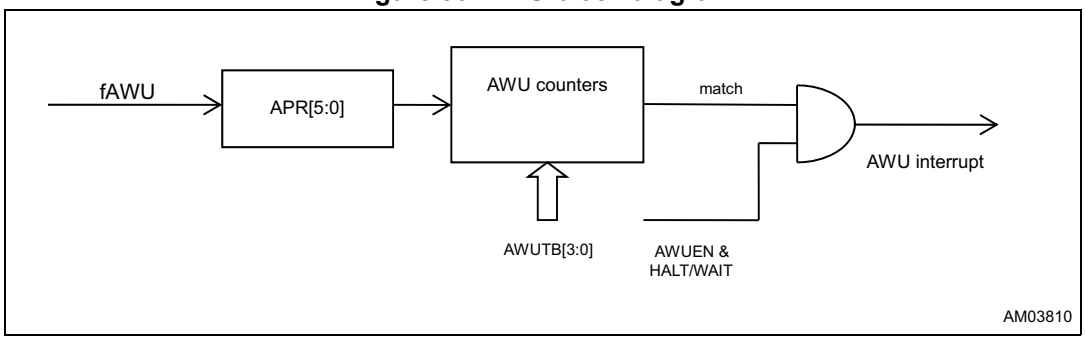
The AWU is used to provide an internal wakeup time base usable when the MCU enters into the Active-halt power saving mode. The f_{AWU} time base frequency is generated by the LSI internal RC oscillator or by the external prescaled HSE source (prescaled to 128 kHz).

17.1 AWU functional description

Figure 30 shows an outline view of the AWU block diagram.

The f_{AWU} clock is generated by the CKC block, for further details refer to *Section 12: Clock control unit (CKC) on page 71* and the requirements present in the product datasheet.

Figure 30. AWU block diagram



When the AWU logic is not in used, the AWUTB[3:0] bits of time base selection register (AWU_TBR) should be cleared to reduce the IC device power consumption.

17.2 Time base selection

Please refer to the asynchronous *Section 17.5.2: AWU_APR (prescaler divider register)* *Section 17.5.3: AWU_TBR (time base register)* descriptions.

In order to choose the right values for AWUTB[3:0] and APRDIV register fields, the user has first to search the range corresponding to the desired time interval (this gives the AWUTB[3:0] value). Then the APRDIV can be chosen to get a time interval value as close as possible to the desired one.

The above register field configuration can be done through the formulas present in *Table 26*.

Table 26 details the auto-wakeup interval range selectable through both AWUTB[3:0] and APRDIV register fields.

Table 26. AWU timer interval range

AWUTB[3:0]	APR_{DIV} range	APR_{DIV} formula for internal time definition	Interval range	Timer range f_{AWU} (LSI) at 153.6 kHz (ms)	Timer range f_{AWU} (HSE/HSI) at 128 kHz (ms)
0b0001	2 to 64	APR_{DIV}/f_{AWU}	$2/f_{AWU}$ - $64/f_{AWU}$	0,013 - 0,417	0,016 - 0,5
0b0010	32 to 64	$2xAPR_{DIV}/f_{AWU}$	$2x32/f_{AWU}$ - $2x64/f_{AWU}$	0,41 - 0,83	0,5 - 1
0b0011	32 to 64	$22xAPR_{DIV}/f_{AWU}$	$2x64/f_{AWU}$ - $2x128/f_{AWU}$	0,83 - 1,66	1 - 2
0b0100	32 to 64	$23xAPR_{DIV}/f_{AWU}$	$22x64/f_{AWU}$ - $22x128/f_{AWU}$	1,66 - 3,33	2 - 4
...	...		...	...	...
0b1100	32 to 64	$211xAPR_{DIV}/f_{AWU}$	$2^{10}x64/f_{AWU}$ - $210x128/f_{AWU}$	426,66 - 853,33	512 - 1024
0b1101	32 to 64	$212xAPRDIV/f_{AWU}$	$2^{11}x64/f_{AWU}$ -$211x128/f_{AWU}$	853,33 - 1706,66	1024 - 2048
0b1110	26 to 64	$211x5xAPR_{DIV}/f_{AWU}$	$2^{11}x130/f_{AWU}$ -$211x320/f_{AWU}$	1733,33 - 4266,66	2080 - 5120
0b1111	11 to 64	$211x30xAPR_{DIV}/f_{AWU}$	$2^{11}x330/f_{AWU}$ - $212x960/f_{AWU}$	4400 - 25600	5280 - 30720

Note: 1. If the target value is between $211x128/f_{AWU}$ and $211x130/f_{AWU}$ or between $211x320/f_{AWU}$ and $211x330/f_{AWU}$, the value closer to the target one must be chosen.

2. The AWU interval time can be further decreased by configuring the CLK_AWUDIVR prescaler register (refer to Section 12.11.8: CLK_AWUDIVR (AWU clock prescaler configuration) on page 92).

17.3 Interval time formulas

The following equations can be used to configure the T_{AWU} elapsed time:

- When AWUTB[3:0] < 0b1110: T_{AWU} = (2^(AWUTB-1) * APRDIV)/f_{AWU}
- When AWUTB[3:0] = 0b1110: T_{AWU} = (2^(AWUTB-3) * 5 * APRDIV)/f_{AWU}
- When AWUTB[3:0] = 0b1111: T_{AWU} = (2^(AWUTB-4) * 30 * APRDIV)/f_{AWU}

17.4 AWU program sequence

The AWU usage requires the following program sequences:

1. Select the appropriate clock input HSE, HSI or LSI (refer to the product datasheet description).
2. Define the appropriate prescaler value by writing to the APR [5:0] bits in the asynchronous prescaler register (AWU_APR).
3. Select the desired auto-wakeup delay by writing to the AWUTB[3:0] bits in the time base selection register (AWU_TBR).
4. Set the AWUEN bit in the control/status register (AWU_CSR).
5. Execute the HALT instruction. AWU counters are reloaded and start to count a new AWU interval time.
6. Continue the executions after AWU interrupt.

Note: *1. The AWU timer starts only when the MCU enters the Active-halt mode after a HALT instruction (refer to the Active-halt mode section in Section 13: Power management (PM) on page 107). The AWU interrupt logic is enabled in this timeframe.*

2. The prescaler counter starts to count only if APR[5:0] value is different from the reset value 0x3F.

3. The f_{AWU} time base source clock has to be select in accordance with the product datasheet requirements.

4. If the AWU logic is not used, the AWUTB[3:0] bits of time base selection register (AWU_TBR) should be cleared to reduce the IC device power consumption.

17.5 Auto wake unit registers description

17.5.1 AWU_CSR (control status register)

Offset: 0x00

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	AWUF	AWUEN	RFU	RFU	RFU	RFU
r	r	rc/r	r/w	r	r	r	r

Bit 3-0: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 4: **AWUEN** auto-wakeup enable

This bit is set and cleared by the software. It enables the auto-wakeup feature. If the microcontroller enters either the Active-halt or wait mode, the AWU logic wakes up the microcontroller after the programmed delay time.

0: AWU (auto-wakeup) disabled.

1: AWU (auto-wakeup) enabled.

Bit 5: **AWUF** auto-wakeup flag

This bit is set by the hardware when the AWU module generates an interrupt and cleared by SW reading the AWU_CSR register; the bit writing is ignored.

0: no AWU interrupt occurred.

1: AWU interrupt occurred.

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

17.5.2 AWU_APR (prescaler divider register)

Offset: 0x01

Default value: 0x3F

7	6	5	4	3	2	1	0
RFU	RFU	APR [5:0]					
r	r	r/w					

Bit 5-0: **APR[5:0]** asynchronous prescaler divider factor

These bits are written by software to select the prescaler divider feeding the counter clock.

0x00: APRDIV = 2

0x01: APRDIV = 3

0x02: APRDIV = 4

0x03: APRDIV = 5

…

0x06: APRDIV = 8

…

0x0E: APRDIV = 16

…

0x1E: APRDIV = 32

…

0x2E: APRDIV = 48

…

0x3E: APRDIV = 64

0x3F: reserved value

Note: Do not write the value 0x3F in this register.

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

17.5.3 AWU_TBR (time base register)

Offset: 0x02

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	AWUTB [3:0]			
r	r	r	r	r/w			

Bit 3-0: **AWUTB[3:0]** auto-wakeup time base selection

These bits are written by software to program the wakeup interrupt frequency.

AWU interrupt is enabled when AWUEN = '1'.

0000: no interrupt.

0001: $(2^0 \times APRDIV) / f_{AWU}$

0010: $(2^1 \times APRDIV) / f_{AWU}$

0011: $(2^2 \times APRDIV) / f_{AWU}$

0100: $(2^3 \times APRDIV) / f_{AWU}$

0101: $(2^4 \times APRDIV) / f_{AWU}$

0110: $(2^5 \times APRDIV) / f_{AWU}$

0111: $(2^6 \times APRDIV) / f_{AWU}$

1000: $(2^7 \times APRDIV) / f_{AWU}$

1001: $(2^8 \times APRDIV) / f_{AWU}$

1010: $(2^9 \times APRDIV) / f_{AWU}$

1011: $(2^{10} \times APRDIV) / f_{AWU}$

1100: $(2^{11} \times APRDIV) / f_{AWU}$

1101: $(2^{12} \times APRDIV) / f_{AWU}$

1110: $(2^{11} \times 5 \times APRDIV) / f_{AWU}$

1111: $(2^{11} \times 30 \times APRDIV) / f_{AWU}$

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

17.6 Auto wake unit registers overview

Table 27 shows the auto wake unit internal registers starting from the base address specified in the corresponding product datasheet; for detailed register description refer to *Section 17.5: Auto wake unit registers description*.

Table 27. AWU internal registers overview

Name	Description	Offset	Type	Reset value
AWU_CSR	Control status register	0x00	R/W	0x00
AWU_APR	Asynchronous prescaler register	0x01	R/W	0x3F
AWU_TBR	Time base register	0x02	R/W	0x00

18 System timer (STMR)

18.1 STMR introduction

The system timer consists of a 16-bit auto-reload upcounter driven by a programmable prescaler. It can be used for the time base generation, with the interrupt generation on the timer overflow event.

18.2 STMR main features

The system timer has the following main features

- 16-bit auto-reload upcounter
- One shoot, free running operating mode
- 3-bit programmable prescaler allowing dividing (also “on the fly”) the counter clock frequency by 1, 2, 4, 8, 16, 32, 64 and 128.
- Interrupt generation on the following events:
  - Counter overflow.
  - Counter update.
  - Counter initialization by SW.

18.3 STMR timer overview

The system timer is based on a 16-bit upcounter connected to the internal clock f_{CK_CNT} provided by a prescaler logic clocked by the f_{MASTER}. The prescaler can divide the internal clock by any power of 2 from 1 to 128.

The counter clock frequency is calculated as follows:

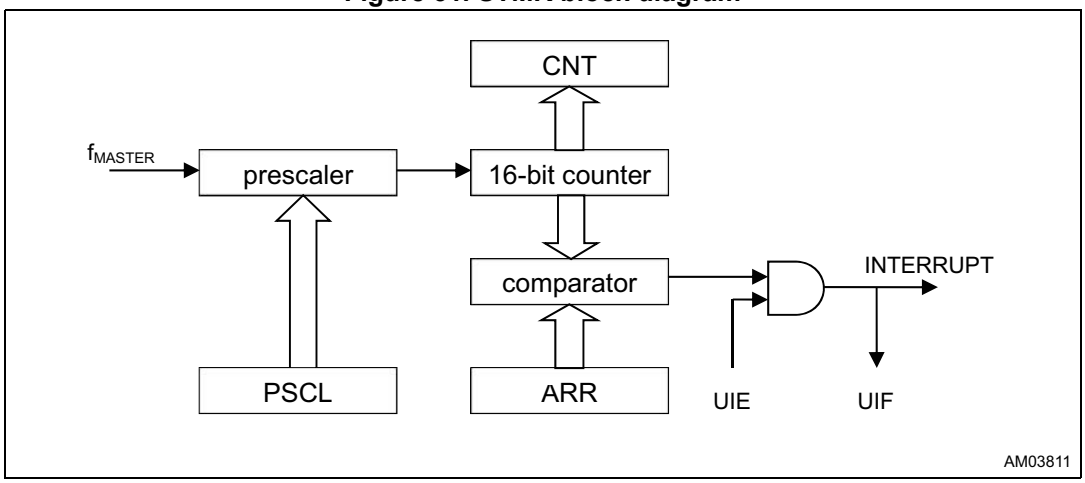
Equation 19

$$f_{CK\_CNT} = f_{MASTER}/2^{(PSC[2:0])}$$

A 16-bit auto-reload register defines the value of the overflow count. When the upcounter matches the auto-reload value, an update event is generated and the upcounter is restarted from 0.

A block diagram of the system timer is represented in *Figure 31*.

Figure 31. STMR block diagram



The 16-bit counter, prescaler and auto-reload registers may be read or written by software.

The auto-reload register is formed by a preload register plus a shadow register. The update of the auto-reload register can be done in two different modes:

- **Auto-reload preload enabled** (the ARPE bit set in the STMR_CR1 register)
  In this mode, when data are written to the auto-reload register, this it's posted in the preload register and transferred into the shadow register at the next update event (UEV).
- **Auto-reload preload disabled** (the ARPE bit cleared in the STMR_CR1 register)
  In this mode, when data are written to the auto-reload register this it's transferred into the shadow register immediately.

An update event is generated when:

- The counter overflows
- The UG bit in the STMR_EGR register is set by software

The UEV can be disabled by setting the UDIS bit in the STMR_CR1.

18.4 Configuration of the 16-bit counter

There is no buffering when writing to the counter. Both STMR_CNTH and STMR_CNTL may be written at any time, so it is suggested not to write a new value into the counter while this it's running to avoid loading an incorrect intermediate content.

An 8-bit buffer is implemented for the read. Software must read the MSB byte first, after that the LSB byte value is buffered automatically. This buffered value remains unchanged until the 16-bit read sequence is completed.

Note: Do not use the LDW instruction to read the 16-bit counter. It reads the LSB byte first and returns an incorrect result.

18.5 Write sequence for 16-bit STMR_ARR register

16-bit values are stored in the STMR_ARR register through the preload registers. This must be performed by two consecutive write instructions, one for each byte. The MSB byte must be written first.

The shadow register update is blocked as soon as the MSB byte has been written, and stays blocked until the LSB byte has been written.

Note: Do not use the LDW instruction as this writes the LSB byte first which produces incorrect results.

18.6 Prescaler

The STMR prescaler is based on a 7-bit counter controlled by the PSC[2:0] bits of the STMR_PSCL register. The register field may be changed on the fly as this control register is buffered. The prescaler divides the counter clock frequency by any power of 2 between 1 and 128.

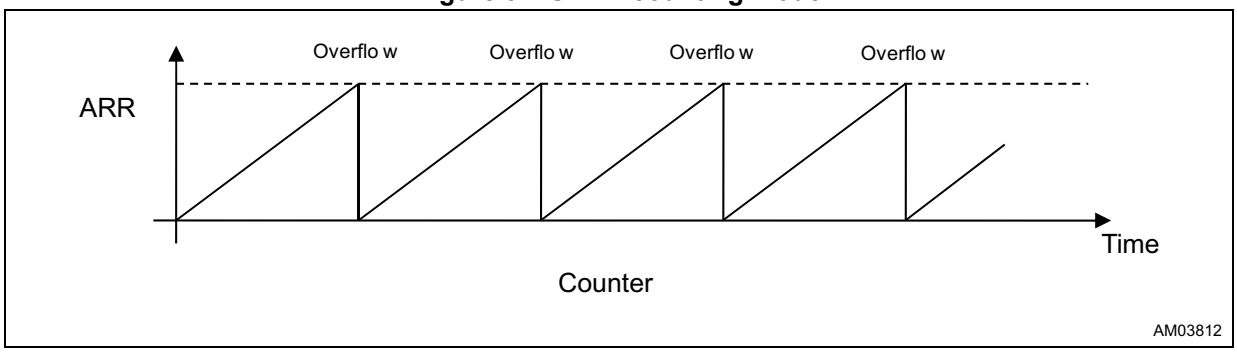
The prescaler value is loaded through a preload shadow register. The shadow register, which contains the current value to be used, is loaded as soon as a UEV is generated.

Read operations to the STMR_PSCL register access the preload register, so no special care needs to be taken to read it.

18.7 STMR counting mode

The counter counts from 0 up to the auto-reload value present into the STMR_ARR H-L registers, then restarts from 0 generating a counter overflow and a UEV event, if the UDIS bit of the STMR_CR1 register is cleared.

Figure 32. STMR counting mode



An update event can also be generated by setting the UG bit in the STMR_EGR register (by software).

The UEV update event can be disabled by software by setting the UDIS bit in the STMR_CR1 register. This is useful to avoid updating the shadow registers while writing new values in the preload registers. No update event occurs until the UDIS bit has been written to 0. However, the counter restarts from the 0 value, whereas the counter of the prescaler restarts from 0 (without any change to the prescale rate).

In addition, if the URS bit (update request selection) in the STMR_CR1 register is set, setting the UG bit generates a UEV without setting the UIF flag (thus no interrupt request is sent). This avoids generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurred, all registers are updated and the update flag UIF bit of the STMR_SR1 register is set (depending on the value of the URS bit, refer to *Section 18.11.1: STMR_CR1 (control register)* description):

- The buffer of the prescaler is reloaded with the preload value (content of the STMR_PSCL register).
- The auto-reload shadow register is updated with the preload value (content of the STMR_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one.

Figure 33. Counter update when ARPE = 0 (ARR not preloaded) with prescaler = 2

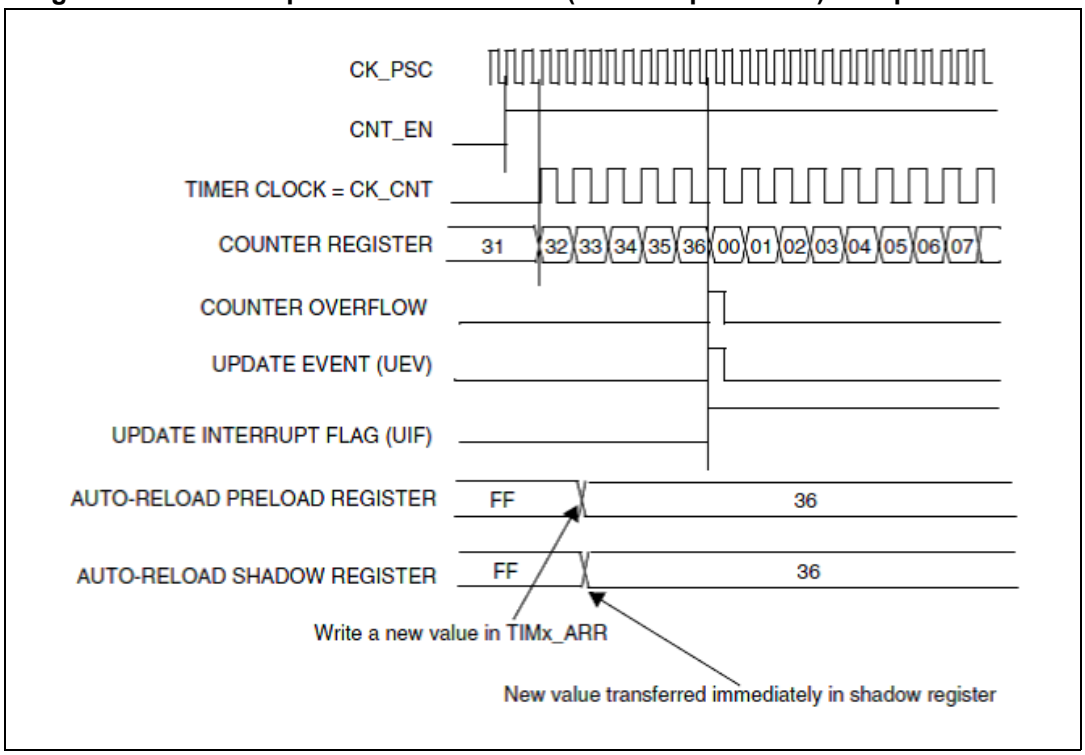
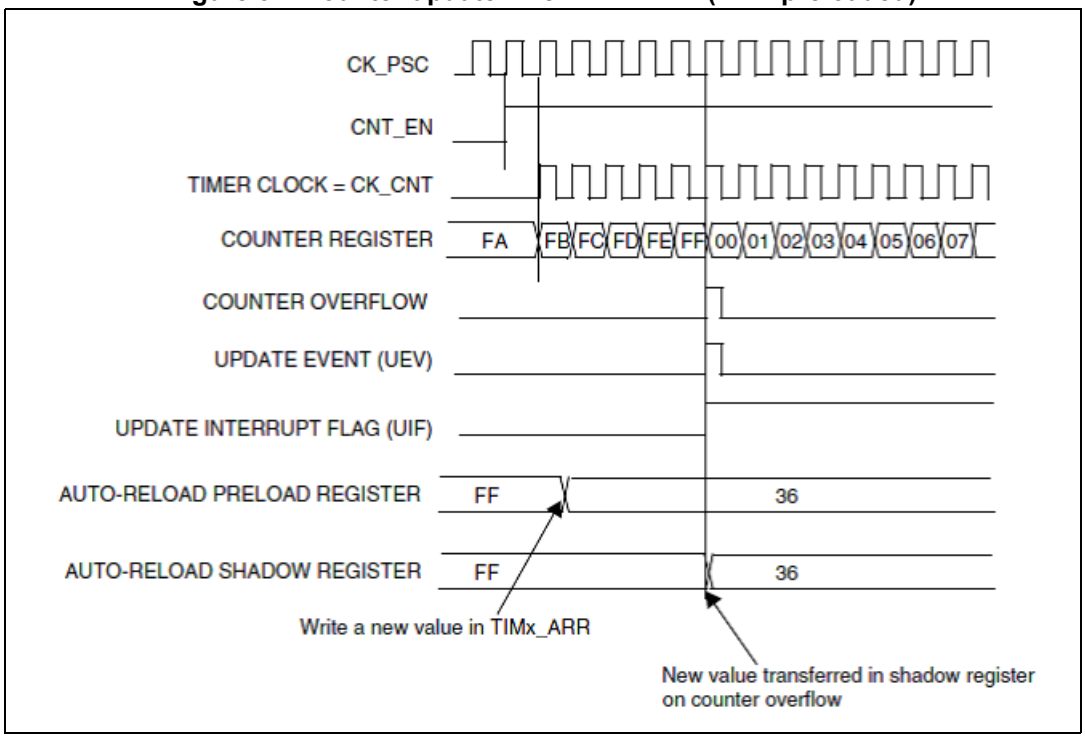


Figure 34. Counter update when ARPE = 1 (ARR preloaded)



18.8 STMR program sequence

This section gives an example of the system timer initialization flow.

- Setup CKC peripheral registers to start the STMR internal clock source
- Disable the counter (STMR_CR1 = 0x0)
- Set the UDIS bit of the STMR_CR1 register to prevent interrupt
- Setup the reload counter value by configuring the STMR_ARRH/L registers
- Set the prescaler register STMR_PSCL with a desired value
- Set the UG bit of the STMR_EGR register to generate update event
- Set the CEN to enable the counter and clear UDIS bits of the STMR_CR1 register
- Set the UIE bit of the STMR_IER register to generate an interrupt on the counter overflow event.

18.9 STMR interrupts

STMR has only one interrupt request source mapped on a dedicated interrupt vector:

- Update event interrupt (overflow, and counter initialization)

The interrupt features is enabled by setting the UIE bit of the STMR_IER register to enable the timer interrupt requests.

The update interrupt event can also be generated by software setting the UG bit of the STMR_EGR register.

18.10 System timer as ADC trigger source

The STMR peripheral may be configured to provide the ADC HW trigger functionality; refer to the product datasheet to see if this functionality is supported.

The ADC start is triggered by any event which is programmed (by means of UDIS and URS bits) to set the interrupt flag. The trigger functionality is independent of peripheral interrupt enable: when the peripheral is selected to trigger the ADC, the interrupt should be disabled in order to minimize the CPU load. Refer to the ADC chapter for further details on using the ADC trigger (refer to *Section 27: Analog-to-digital converter (ADC) on page 312*).

18.11 System timer unit registers description

18.11.1 STMR_CR1 (control register)

Offset: 0x00

Default value: 0x00

7	6	5	4	3	2	1	0
ARPE	RFU			OPM	URS	UDIS	CEN
r/w	r			r/w	r/w	r/w	r/w

Bit 0: **CEN** counter enable

0: the counter is disabled.

1: the counter is enabled.

Bit 1: **UDIS** update disable

0: the update event is generated as soon as a counter overflow occurs or a software update is generated; the buffered registers are then loaded with their preload values.

1: the update event is not generated; shadow registers keep their value (ARR, PSC). Anyway the counter and the prescaler are reinitialized if the UG bit is set.

Bit 2: **URS** update request source

0: the interrupt requests are sent as soon as registers are updated (counter overflow or set of UG bit).

1: the interrupt requests are sent only when the counter reaches the overflow.

Bit 3: **OPM** one pulse mode

0: counter doesn't stop when an update event is triggered

1: counter stops the count when the next update event is triggered (clearing the bit CEN).

Bit 6-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 7: **ARPE** auto-reload preload enable

0: STMR_ARRx register are not buffered.

1: STMR_ARRx register are buffered through a preload register.

18.11.2 STMR_IER (interrupt enable register)

Offset: 0x01

Default value: 0x00

7	6	5	4	3	2	1	0
RFU							UIE
r							r/w

Bit 0: **UIE** update interrupt enable

0: update interrupt disabled.

1: update interrupt enabled.

Bit 7-1:**RFU** reserved; must be kept 0 during register writing for future compatibility.

18.11.3 STMR_SR1 (status register 1)

Offset: 0x02

Default value: 0x00

7	6	5	4	3	2	1	0
RFU							UIF
r							rc/wo

Bit 0: **UIF** update interrupt flag

This bit is set by hardware on an update event. It is cleared by software by writing '0':

0: no update occurred.

1: update interrupt pending. This bit is set by hardware when the registers are updated:

- When the overflow occurs and if the UDIS = 0 in the STMR_CR1 register.

- When the CNT is reinitialized by software using the UG bit in the STMR_EGR register, if URS = 0 and UDIS = 0 in the STMR_CR1 register.

Bit 7-1: **RFU** reserved; must be kept 0 during register writing for future compatibility.

18.11.4 STMR_EGR (event generation register)

Offset: 0x03

Default value: 0x00

7	6	5	4	3	2	1	0
RFU							UG
r							w

Bit 0: **UG** update generation

This bit is set by software and it's automatically cleared by hardware:

0: no action.

1: re-initializes the counter and generates an update of the registers. Note that the prescaler counter is cleared too (the prescaler ratio is not changed).

Bit 7-1: **RFU** reserved; must be kept 0 during register writing for future compatibility.

18.11.5 STMR_CNTH (counter high register)

Offset: 0x04

Default value: 0x00

7	6	5	4	3	2	1	0
CNT [15:8]							
r/w							

Bit 7-0: **CNT[15:8]** counter value most significant byte (MSB).

18.11.6 STMR_CNTL (counter low register)

Offset: 0x05

Default value: 0x00

7	6	5	4	3	2	1	0
CNT [7:0]							
r/w							

Bit 7-0: **CNT[7:0]** counter value less significant byte (LSB).

18.11.7 STMR_PSCL (prescaler register)

Offset: 0x06

Default value: 0x00

7	6	5	4	3	2	1	0
RFU					PSC [2:0]		
r					r/w		

Bit 2-0: **PSC[2:0]** prescaler value

Value of the prescaling factor to be applied to the internal clock f_{MASTER}.

The f_{CK_CNT} counter clock frequency is equal to $f_{MASTER} / 2^{PSC[2:0]}$.

The PSCL contains the value which is loaded in the active prescaler register at each update event (including when the counter is cleared through the UG bit of the STMR_EGR register.

Consequently, a UEV must be generated so that a new prescaler value can be taken into account.

Bit 7-3: **RFU** reserved; must be kept 0 during register writing for future compatibility.

18.11.8 STMR_ARRH (auto-reload high register)

Offset: 0x07

Default value: 0xFF

7	6	5	4	3	2	1	0
ARR [15:8]							
r/w							

Bit 7-0: **ARR[15:8]** auto-reload value MSB

The ARR is the value to be loaded into the actual auto-reload register.

The counter is halted while the auto-reload value is null.

18.11.9 STMR_ARRL (auto-reload low register)

Offset: 0x08

Default value: 0xFF

7	6	5	4	3	2	1	0
ARR [7:0]							
r/w							

Bit 7-0: **ARR[7:0]** auto-reload value LSB

The ARR is the value to be loaded in the actual auto-reload register.

The counter is halted while the auto-reload value is null.

18.12 System timer unit registers overview

Table 28 details the system timer internal registers starting from the base address specified on the corresponding product datasheet; for detailed register description refer to *Section 18.11: System timer unit registers description*.

Table 28. System timer internal registers overview

Name	Description	Offset	Type	Reset value
STMR_CR1	Control register 1	0x00	R/W	0x00
STMR_IER	Interrupt enable register	0x01	R/W	0x00
STMR_SR1	Status register	0x02	R/W1	0x00
STMR_EGR	Event generation register	0x03	RS	0x00
STMR_CNTH	Counter high register	0x04	R/W	0x00
STMR_CNTL	Counter low register	0x05	R/W	0x00
STMR_PSCL	Prescaler register	0x06	R/W	0x00
STMR_ARRH	Auto-reload high register	0x07	R/W	0xFF
STMR_ARRL	Auto-reload low register	0x08	R/W	0xFF

19 Auxiliary timer

19.1 Auxiliary timer introduction

The auxiliary timer is a light timer built grouping some functionality already existing in the silicon device and spread on different IPs to optimize the silicon cost.

The main purpose of this timer is to provide a secondary programmable time base generation usable by SW application.

19.2 Auxiliary timer main features

The timer clock pulse is provided by the configurable clock out logic (programmable in term of the clock source and prescaler division factor), while the interrupt functionality is implemented by an interrupt edge detection logic similar to the port P0. The timer has the following main features:

- Free running mode
- Upcounter
- Timer prescaler 8 bits
- Interrupt timer request
  - Vectored interrupt
  - Interrupt IRQ/NMI or polling mode
- Timer pulse configurable as an output signal on CCO primary pin.

19.3 Auxiliary timer functional description

The main function of the auxiliary timer is to provide a periodic interrupt (or polled signal) useful as reference for a time base generation.

The timer unit uses the configurable clock out logic to generate a periodic square wave. The interrupt functionality is controlled by the edge detector logic of the P1[5] port; the MSC_CFGP15 configures the interrupt type, the interrupt flag and the reset mechanism is handled by the register MSC_STSP1[5] bit.

The interrupt mask functionality managed by the P1_CR2[5] bit acts also on the auxiliary timer interrupt. The aux. timers share the interrupt vector with the P1 port, in products where the P1 port interrupt functionality is available.

Refer to *Section 14: Interrupt controller (ITC) on page 113* and *Section 22: General purpose I/O port (GPIO) on page 221* for details about the interrupt configurability.

The auxiliary timer may be configured to generate the HW trigger to start the ADC trigger conversion sequence (refer to the product datasheet if this features is available). The ADC trigger functionality is independent of the interrupt timer request generation.

19.4 Program sequence

The following operations have to be done in order to configure the auxiliary timer functionality:

- Select the source clock, by programming the CLK_CCOR register as for standard CCO configuration.
- Initialize the clock divisor, by programming the CLK_CCODIVR register as for standard CCO configuration.
- Configure the input capture logic of the GPIO1[5] input line (rising edge active high active IRQ/NMI) using the dedicated miscellaneous registers MSC_CFGP15.
- Enable the interrupt source (GPIO1[5] input line) through the MSC_CFGP15 register.
- The interrupt or polling mode functionality is selected by the register P1_CR2 bit 5 (if it's set, the interrupt request is enabled).

If the interrupt request is masked , the timer event may be polled by checking the bit 5 of the port 1 status register (MSC_STSP1).

The aux. timer functionality is an internal feature independent of the DIGIN[0] primary signal except in case of external timer clock configuration (enabled by CCO output pin selection).

Note: 1. The auxiliary timer is halted by clearing the MSC_CFGP15 bit 3.

2. The pulse timing is also available on the CCO primary pin, via the CKC programming register.

3. The CCO internal clock is a free running clock source.

4. The auxiliary timer is not halted when the CPU enters in the debug mode; this has to be considered during this phase.

5. To be detected by the interrupt edge capture logic, the frequency of the waveform selected through the CLK_CCOR and CLK_CCODIVR register must respect the following condition: $f_{CCO} < f_{CKM}$. This rule must be considered also when the auxiliary timer is used as a source for the ADC trigger functionality.

19.5 Auxiliary timer unit register overview

Table 29 shows the auxiliary timer internal register overview; the registers are distributed across different peripherals; for register details refer to respectively peripheral chapter descriptions.

Table 29. Auxiliary timer register overview

Name	Description	Block	Offset	Type	Reset value
P1_CR2	Control register2	GPIO1	0x04	R/W	0x00
MSC_CFGP10	P15 input line control	MISC	0x26	R/W	0x00
MSC_STSP1	Port 1 status		0x27	R/W	0x00
CLK_CCODIVR	Clock dividers	CKC	0x15	R/W	0x00
CLK_CCOR	Configurable clock output		0x1A	R/W	0x00

20 Basic timer

20.1 Basic timer introduction

The STLUX architecture can include up to two additional basic timers besides the system timer and the auxiliary timer. Each basic timer (BscTim) operates with a free running counter clocked by an independently selectable and prescalable source. Refer to the product datasheet for details on the availability and the count of the BscTim in your product.

These timers can increase the flexibility of the application by allowing the user to implement additional time-based events.

20.2 Basic timer main features

Each BscTim generates an internal square signal with 50% duty cycle whose half period depends on the clock source frequency $_{fck,}$ the counter value *cnt* and the prescaler *prsc* division factor according to the following equation:

Equation 20

$$T / 2 = [prsc \times (2 + cnt)] / f_{ck}$$

When one BscTim is enabled, this square signal feeds the input of the associated edge detector logic. In the current implementation the BscTim0 and BscTim1, if available, share through a multiplexer the edge detector of the P1[0] and P1[1] port respectively. Enabling the timer function inhibits the interrupt feature on the relative P1 port (if available in the product).

Each timer has the following main features:

- Free running mode
- Timer prescaler register 8 bits
- Counter register 6 bits
- Programmable source clock (HSI, LSI, HSE, PLL)
- Interrupt timer request
  - Vectored interrupt (shared with the P1 port if available in the product)
  - Interrupt IRQ/NMI or polling mode

20.3 Basic timer functional description

The basic timers provide a periodic waveform which can be used as reference for time base generation. It is possible to associate an interrupt or to use a polling approach. In both cases the user has to properly configure and enable the edge detection circuit.

For the BscTim<x> (x = 0, 1) the MSC_CFGP1<x> register configures the interrupt type and the detection mode (for normal operation the edge mode is recommended); the interrupt flag and the reset mechanism are handled by the MSC_STSP1[x] register bit.

The interrupt functionality is enabled through the P1_CR2[x] bit. The BscTims share the programming registers and the interrupt vector with the P1 port, in products where the P1 port interrupt functionality is available.

Refer to *Section 14: Interrupt controller (ITC) on page 113* and *Section 22: General purpose I/O port (GPIO) on page 221* for further details on the interrupt programming.

The BscTims can be used also as a source of the ADC trigger functionality, available on some products. The event which can trigger an ADC SoC is the rising edge of the timer waveform. The BscTim can be selected as an ADC trigger source regardless of interrupt enabling. Refer to *Section 27: Analog-to-digital converter (ADC) on page 312* for further details on the use of the ADC trigger.

In order to synchronize the timer output, the system PLL has to be enabled and locked.

20.4 Programming sequence

The following operations have to be done in order to configure the BscTim<x> (x = 0, 1) functionality:

- Enable the system PLL and wait for its lock.
- Select the clock source of the timer through the CK_SEL<x> field of the MSC_FTM0CKSEL register (see *Section 20.5.1: MSC_FTM0CKSEL (BscTim10 clock selection)* for details).
- Select the prescaler (prsc) value through the MSC_TM<x>CKDIV register.
- Select the counter (cnt) value through the MSC_TM<x>CONF register.
- Configure the input sensitivity of the P1[x] input line (rising/falling edge and optionally NMI) by using the dedicated MSC_CFGP1<x> miscellaneous register. Refer to the *Section 14: Interrupt controller (ITC)* for further details on configuring, enabling and using the P1 detection logic (MSC_CFGP1<x> and MSC_STSP1 registers).
- Enable the edge detection through the MSC_CFGP1<x>[3] bit.
- Enable the generation of the timer waveform through the EN<x> bit of the MSC_FTM0CKSEL register (see *Section 20.5.1* for details on the bit).
- The interrupt or polling mode functionality is selected through the P1_CR2[x] bit (if it is set, the interrupt request is enabled). Refer to *Section 22: General purpose I/O port (GPIO)* for further details on P1_CR2 register.

If the interrupt request is masked, the timer event may be polled by checking the relative status register bit (MSC_STSP1[x]).

Note: 1. The BscTim<x> is halted and reset by clearing the EN<x> bit. Alternatively the user can disable the edge detection through the MSC_CFGP1<x> register in order to stop timer events.

2. The BscTim is not halted automatically when the CPU enters in the debug mode; this condition has to be considered during this phase.

3. In order to be detected by the edge detector, the frequency of the timer related waveform must respect the following condition: $f < f_{CKM}$. This consideration applies also if the BscTim is used as a source for the ADC trigger functionality.

4. In some products where the BscTim1 feature is absent, the CK_SEL1 and EN1 bits of the MSC_FTM0CKSEL register, the MSC_FTM1CKDIV and MSC_FTM1CONF registers themselves may be associated to the ADC trigger filter functionality. Refer to the product datasheet for further details.

20.5 Basic timer control registers

Note that depending on the product, all or some of the registers may not be available.

20.5.1 MSC_FTM0CKSEL (BscTim10 clock selection)

Offset: 0x00 (MSC INDIRECT AREA)

Default value: 0x00

7	6	5	4	3	2	1	0
EN1	CLK_SEL1 [2:0]			EN0	CLK_SEL0 [2:0]		
r/w	r/w			r/w	r/w		

Bit 2-0: **CLK_SEL0[2:0]** BscTim0 clock source configuration:

000: HSI source clock

001: HSE source clock

010: LSI source clock

011: PLL source clock

1xx: RFU reserved encoding values.

Bit 3: **EN0 BscTim0** enable:

0: BscTim0 logic disabled

1: BscTim0 logic enabled

Bit 6-4: **CLK_SEL1[2:0]** BscTim1 clock source configuration:

000: HSI source clock

001: HSE source clock

010: LSI source clock

011: PLL source clock

1xx: RFU reserved encoding values.

Bit 7: **EN1 BscTim1** enable:

0: BscTim1 logic disabled

1: BscTim1 logic enabled

20.5.2 MSC_FTM0CKDIV (BscTim0 clock division factor)

Offset: 0x01 MSC (INDIRECT AREA)

Default value: 0x00

7	6	5	4	3	2	1	0
DIV [7:0]							
r/w							

Bit 7-0: **DIV[7:0]** BscTim0 clock division factor:

This field represents the frequency ratio of the source clock prescaler (prsc = DIV + 1). The following equation applies:

Equation 21

$$f_{prsc} = f_{ck} / (DIV + 1)$$

where f_{prsc} is the prescaled frequency and f_{ck} is the source clock frequency.

20.5.3 MSC_FTM0CONF (BscTim0 counter)

Offset: 0x02 (MSC INDIRECT AREA)

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	COUNT [5:0]					
r/w	r/w	r/w					

Bit 5-0: **COUNT[5:0]** BscTim0 timer value:

The clock count used to synthesize the BscTim0 waveform half period. The single clock period is given by $1/f_{prsc}$. Due to the internal architecture the real count value is

Equation 22

$$cnt = COUNT + 2$$

Bit 7-6: **RFU** reserved for future use.

Keep these bits reset unless other product functionalities are associated to them.

20.5.4 MSC_FTM1CKDIV (BscTim1 clock division factor)

Offset: 0x03 MSC (INDIRECT AREA)

Default value: 0x00

7	6	5	4	3	2	1	0
DIV [7:0]							
r/w							

Bit 7-0: **DIV[7:0]** BscTim1 clock division factor:

This field represents the frequency ratio of the source clock prescaler (prsc = DIV + 1). See *Section 20.5.2: MSC_FTM0CKDIV (BscTim0 clock division factor)* for the analogous discussion about the BscTim0.

20.5.5 MSC_FTM1CONF (BscTim0 counter)

Offset: 0x04 (MSC INDIRECT AREA)

Default value: 0x00

7	6	5	4	3	2	1	0
ADC_AReload	ADC_AFlush	COUNT [5:0]					
r/w	r/w	r/w					

Bit 5-0: **COUNT[5:0]** BscTim1 timer value:

The clock count used to synthesize the BscTim1 waveform half period. See *Section 21.6.3: External input multiplexing on page 170* for the analogous discussion about BscTim0.

Bit 6: **ADC_AFlush** FIFO auto-flush for single conversion mode:

0: disable auto-flush.

1: enable auto-flush (if available, it works when the CIRCULAR bit of the ADC_CFG register is '0').

Bit 7: **ADC_AReload** FIFO auto-reload mode:

0: disable auto-reload.

1: enable auto-reload (if available, it works when the CIRCULAR bit of the ADC_CFG register is '1').

20.6 Additional basic timer register overview

Table 30 shows the BscTim internal register overview. The registers are located in the indirect address miscellaneous area. Refer to the product datasheet on details on how to manage the indirect addressing.

Table 30. BASIC timer register overview

Name	Description	Block	Offset	Type	Reset value
MSC_CFGP10	P10 input line control	MISC	0x21	R/W	0x00
MSC_STSP1	Port 1 status		0x27	R/W	0x00
MSC_FTM0CKSEL	Basic timer source clock sel.	MSC (indirect)	0x00	R/W	0x00
MSC_FTM0CKDIV	Basic Timer0 clock prescale		0x01	R/W	0x00
MSC_FTM0CONF	Basic Timer0 counter value		0x02	R/W	0x00
MSC_FTM1CKDIV	Basic Timer1 clock prescale		0x03	R/W	0x00
MSC_FTM1CONF	Basic Timer1 counter value		0x04	R/W	0x00

21 State machine, event driven (SMED)

21.1 SMED architecture overview

The SMED is an advanced programmable PWM generator signal designed to meet mainly the requirements of lighting applications, power controllers and PFC of small/medium power. Single or multiple instantiations of the SMED module are intended to be embedded in the microcontroller IC device.

The SMED (state machine, event driven) peripheral is a programmable state machine controllable by both internal events (counter timers) and external events (primary I/O signals), able to generate an output signal (PWM) depending on the evolution of the internal state machine.

The main purpose of the SMED is to generate a periodic pulsed PWM output signal with a behavior instantaneously modifiable by real-time events in terms of the duty cycle, polarity level and frequency adjustment. The deterministic evolution and the reaction to asynchronous input events are programmed by a dedicated set of three registers for each state. The SMED state machine contains four independent timed states (S0-3), one initial state (IDLE) and one not timed state (HOLD).

Each SMED block is provided with an expansion interface that allows functional cross connections between different SMED modules. The expansion i/f allows coupling two up to four SMED depending on the product features. For further information refer to the product datasheet.

21.2 SMED main features

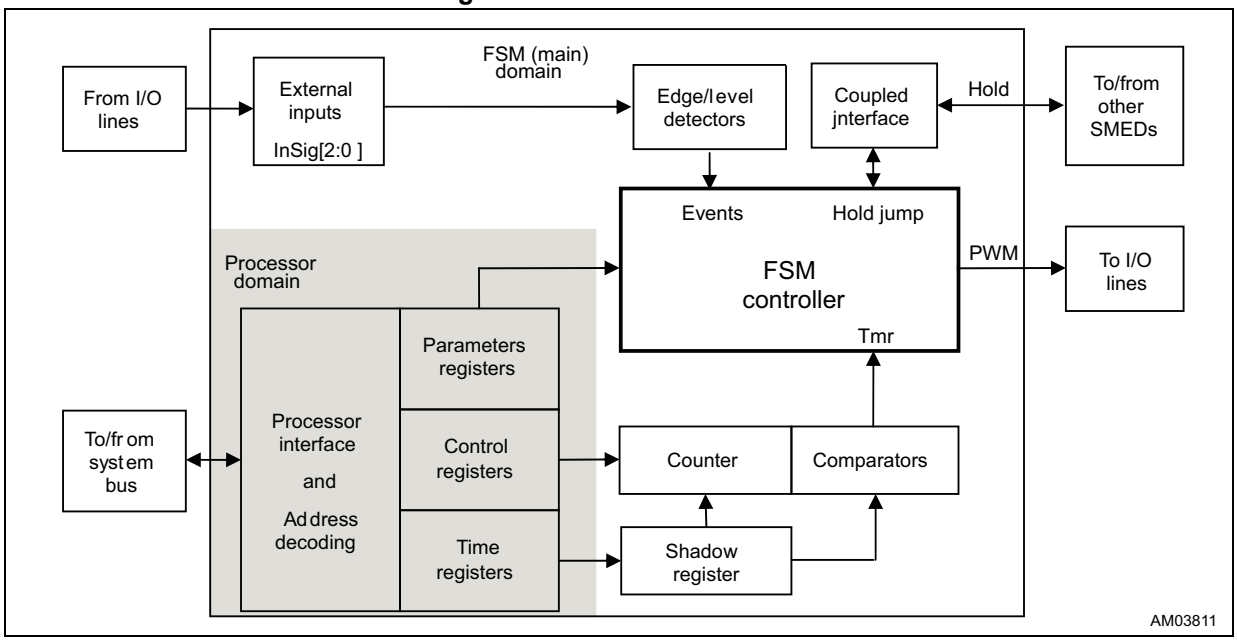
The SMED main features are summarized in the following points:

- Programmable state machine for PWM generation
- Four independent states (S0-S3) plus an initial state (IDLE) and a special state (HOLD)
- Programmable timings independent for each state (S0-S3)
- Large set of selectable trigger events:
  - I/O lines levels and edges
  - Analog comparators outputs
  - Software programmable events
- High sampling rate on input signals (up to 96 MHz)
- High resolution on output signal generation due to high-speed clock
- Output signal modulation (dithering function)
- Events timestamp (dump function)
- Interrupt generation capability on internal/external events
  - External input trigger
  - State change
  - Timer overflow
- Full status information
  - Events overflow
  - Timing overflow
  - Timing capture (dump status)
  - State change
  - Current state
  - Output line status
  - Input lines status
- ADC trigger capability (if available on the product)
- Independently programmable IDLE PWM levels via option byte (refer to the product datasheet)
- PWM independently programmable as (pseudo) open drain output (refer to the product datasheet)
- SMED coupled expansion interface.

21.3 Block diagram

A simplified architecture of a single SMED device is depicted in *Figure 35*.

Figure 35. SMED architecture



21.3.1 Subblocks descriptions

The functionalities implemented by the main blocks depicted in *Figure 35* are the following:

- Edge/level detectors: sample the external input line to detect both the signal level and the signal transitions
- Coupled interface: handles the internal signals to/from other SMED devices for coupled mode configurations
- Counters/comparators: handle the FSM timing information related to sequential state evolution
- FSM controller: is the core logic of the SMED that determines state evolution
- Processor interface: glue logic necessary to correctly interface the SMED device to the microcontroller system bus.

21.3.2 Clock domains

The SMED unit operates with two different clock logic domains as highlighted in the block diagram; the processor domain (CKM) and the FSM domain. This distinction is particularly useful to understand internal synchronization behaviors when the processor clock and the SMED clock are distinct.

The processor clock domain is related to all the interface registers that can be directly accessed by the user application, while the FSM clock domain is related to the SMED internal logic, in particular to input signals detection and FSM state evolution.

The synchronization between the two clock domains is handled internally by the SMED logic.

Note: 1. The real-time FSM clock lines feeding all SMED peripheral logics may be frozen when the CPU is stalled in the debug mode, depending on the ENFCT3 bit field of the DM_ENFCT register.

2. In order to use the SMED interrupts for clock frequency $f_{SMED} < f_{CKM}$ it is necessary to select the "interrupt low frequency" feature through the option byte. For further details refer to the product datasheet.

21.4 Processor interface

The user application can access and control the SMED operations through the processor interface logic, by reading and writing a dedicated set of registers; through the registers the user can configure and control the SMED behavior.

The SMED registers are allocated in the standard microcontroller peripheral register area.

The overall registers that control the SMED logic are grouped in different register spaces depending on their functionality:

- Registers for global SMED configuration (SMED behavior, input lines multiplexing, output lines multiplexing) are located in the MSC register space.
- Registers for clock selection and configuration reside in the CKC (clock controller) area.
- Registers for the SMED state machine configuration reside in dedicated SMED areas, one for each SMED.

The detailed description of SMED core registers and their usage can be found in *Section 21.17.1: SMED coupled interface configuration scheme on page 189* while the detailed description of the SMED environment configuration registers is found in *Section 21.17.2: Chapter assumptions: on page 190*.

Note: SMED registers configuration can be easily done also thanks to the SMED configurator tool available online on the STLUX webpage.

21.5 FSM controller

The FSM controller is the core of the SMED peripheral logic. It is up to this block to handle the selected input events and timings to generate the requested output signal.

The understanding of this block and of the input/output signals configuration is fundamental for the correct usage of SMED logic.

The FSM is a finite state machine based on 5 different states (**IDLE, S0, S1, S2** and **S3**) plus one special state (**HOLD**).

The transitions among the states are driven by both external events and internal events.

These events are controlled by a complete set of configuration registers, dedicated to each state (unless for the special state **HOLD**), and determines the evolution of the FSM core for the PWM output signal generation.

Note: The FSM is enabled, i.e. not forced in the IDLE state and thus capable to evolve and react to events, when the SMD<n>_CTR[1] register bit is at '1'; Clearing the bit isn't a full software reset, because memory elements other than the state registers of the machine, shadow and lock controls included, maintain their information. The user, who needs to force the SMED in IDLE, has to manage this process consistently.

21.6 External events

The SMED has the capability to sample three different independent external input lines called **InSig[0]**, **InSig[1]** and **InSig[2]** to detect the real-time signal variations. The input signals can be physical I/O external input of the IC device or signals generated by the internal logic.

The input **InSig[2:0]** lines are sampled with the FSM frequency, that can be configured up to 96 MHz, allowing a fast response time to input signal transitions.

To be correctly acquired a digital input signal interconnected to the InSig line should have a pulse width ≥ 4 Tck. Refer to the product datasheet for details about the frequency and slope characteristics of the DIGIN pads.

21.6.1 Level/edge detection logic

A dedicated block of the SMED logic is used to constantly monitor the status of the input lines to detect any input signal variations in the real-time mode without SW intervention.

Each input lines may be configured independently to trigger different signal conditions depending on the SMD<n>_CTR_INP register bit values (refer to register programming in *SMD<n>_CTR_INP (control input register) of Section 21.18.1*), as detailed below:

- Falling edge
- Rising edge
- Low level
- High level

In case of edge configuration the InSig line is active for one cycle when the programmed edge is detected.

When the configured capture condition is recognized by one of the enabled InSig[2:0] input lines, and the current state is programmed to react to the involved InSig, the corresponding external event is generated and causes a FSM jump to a new state.

Each one of the three input lines InSig[0], InSig[1] and InSig[2] has to be individually enabled in order to be sensed by the capture logic; this is carried out by configuring the SMD<n>_ISEL register.

The InSig signals are synchronized with a two-stage register chain.

21.6.2 External event handling

The events triggered by the InSig[2:0] external inputs can be used by the FSM to change the state evolution in two distinct ways: sequential transitions and non-sequential transitions.

These events are referred in the following description as **Event_Seq(I)** or **Event_NoSeq(I)**, depending on the way the FSM uses them for state transitions. The “I” letter is a generic index indicating that the event originates due to the InSig[I] signal.

- By convention the sequential transition events is referred as **Event_Seq(I)** (refer to *Section 21.9.3: Sequential state transitions (S0→S3) on page 174*).
- The non-sequential transition events is referred as **Event_NoSeq(I)** (refer to *Section 21.9.4: Non-sequential state transitions on page 175*).

Whenever an InSig[2:0] related event is triggered, and the related mask bit is set, the associated interrupt is triggered and the corresponding status flag bit is set. From the interrupt perspective the Event_Seq(k) and Event_NoSeq(k) for a given *k* are indistinguishable. The interrupt and the flag are associated just to the index of the InSig[k] line that caused the transition.

For each state the InSig used for the Event_Seq() and Event_NoSeq() transitions are programmed through the CEDGE and EDGE fields of the state parameter registers.

The InSig[0] line has a supplementary latch functionality used to memorize the occurrence of an InSig[0] capture condition, allowing a deferred Event_Seq(0) or Event_NoSeq(0) transition occurrence from any of the states configured to react to InSig[0] input capture. When this feature is used, the InSig[0] line is typically configured to capture a signal edge, although this is not required.

This feature is controlled by the INPUT_LAT field of the SMD<n>ISEL register; the latched information is selectively cleared by entering any of the S0-3 states when the LATCH_SEL bit of respective SMD<n>_PRM_<st>2 register is set.

21.6.3 External input multiplexing

Each of the InSig[2:0] input lines may be driven by different signals. The lines are generated by an embedded switch matrix (ConBox) capable to multiplex more input signals (either internal or external sources) to the InSig lines. The configuration of the signals connected to the InSig input lines of each SMED<n> is controlled by the MSC_CBOXS<n> register.

The signal families multiplexed are the following:

- DIGINs (primary input signals)
- CMPs (output of the internal comparator units)
- SW (internal register signal driven by SW)
- PWMs (available only for some SMED units)

For the complete list of signals configurable as the InSig of the SMED units refer to the product datasheet.

When the SMED FSM is running, the input line hot change may be done through the following program sequence:

- Disable the corresponding InSig[k] input line through the SMD<n>_ISEL register.
- Configure the new input signal changing the ConBox configuration by programming the MSC_CBOXS<n> register.
- Configure the corresponding input capture logic with the new input signal by programming the SMD<n>_CTR_INP control register
- Enable the InSig[k] signal through the SMD<n>_ISEL register.

21.7 Internal events

The SMED has an internal 16-bit counter that can be used to control the FSM sequential state evolution similarly to the Event_Seq(I) event. Any of the FSM states S0, S1, S2 and S3 except IDLE and HOLD has an associated 16-bit state timer register (TMR_T0, TMR_T1, TMR_T2 and TMR_T3).

The SMED counter is clocked by the FSM real-time clock and is incremented by 1 on every clock cycle. The registers counter value is compared with the FSM current counter; when

the counter matches or exceeds the respective state timer, then the corresponding **Tmr(S)** (timer) event is generated. By convention the “S” letter is a generic index that specifies the state in which the timer event arises.

When the counter reaches the maximum value (0xFFFF), it automatically wraps back to 0x0001; if the relative CNT_OV_E bit field of the SMD<n>_IER register is set, the counter overflow interrupt is generated and the corresponding CNT_OVER status bit is set. No Tmr(S) event is generated in this case.

Note: For the FSM clock configuration refer to Section 12: Clock control unit (CKC) on page 71.

The counter is initialized to 0x0001 whenever the FSM timer is enabled (refer to the START_CNT bit of *SMD<n>_CTR (control register) in Section 21.18.1 on page 199*) or when it's reset by the occurrence of a state transition if the suitable CNT_RSTC or CNT_RSTE state parameter register bit is set (refer to *Section 21.10.2: Counter reset on page 179*).

21.8 Events priority

In the normal priority sequence the first active event among the Event_NoSeq(I), Event_Seq(I) or Tmr(S) determines the SMED evolution state; in case two or more events are asserted at the same time, the FSM state evolution decision process uses the following fixed priority scheme:

- Event_NoSeq(I) Higher priority
- Event_Seq(I) ↓
- Tmr(S) Lower

21.9 State transition

The SMED logic, as mentioned in the previous paragraph, has five distinct states named IDLE, S0, S1, S2 and S3, plus a special state named HOLD. The FSM states and the allowable transitions between states are represented by the graph in *Figure 36*.

The state transitions graph emphasizes the coexistence of two different state transition sequences:

- Sequential evolution: shown by the continuous line, it's based on the sequential state transitions determined either by Event_Seq(I) or Tmr(S) events; the FSM state evolution is done sequentially in the circular mode from lower to the upper state (i.e. S0, S1, S2, S3, S0, S1, etc.).
- Non-sequential evolution: indicated by the dashed line, it comprises the state transitions determined by the Event_NoSeq(I) events and let the FSM jump into any of S0, S1, S2, S3 and HOLD state without order restrictions.

Mixed mode transition sequences are allowed to increase the complexity of the PWM signal modulation. Each state can be configured to wait for both Event_Seq(I) and Event_NoSeq(I) transitions in addition to the internal Tmr(S) event. (The AND_OR bit may modify the transition state policy).

The state transitions are controlled by a dedicated set of registers available for every states (refer to parameter registers SMD<n>_PRM_<id>[2:0] and SMD<n>_PRM_<st>[2:0] of *Section 21.18.1: SMED core registers on page 199*).

- When the FSM changes the state, jumping from the state x to the next state due to the Tmr(x) event, and the corresponding interrupt enable request bit is set (IT_STA_<st> field of the SMD<n>_IEN register, <st> = S<x>), the associated interrupt request is raised and the respective flag is set (the STA_<st>_IT bit field of the SMD<n>_IRQ register).
- When the transition is caused by an external event, Event_Seq(k) or Event_NoSeq(k), the system considers the InSig[k] that produced it. If the associated interrupt is enabled (the IT_EN_EX<k> bit of the SMD<n>_IEN), the interrupt request is raised and the respective flag is set (the EXT<k>_INT bit field of the SMD<n>_IRQ register).

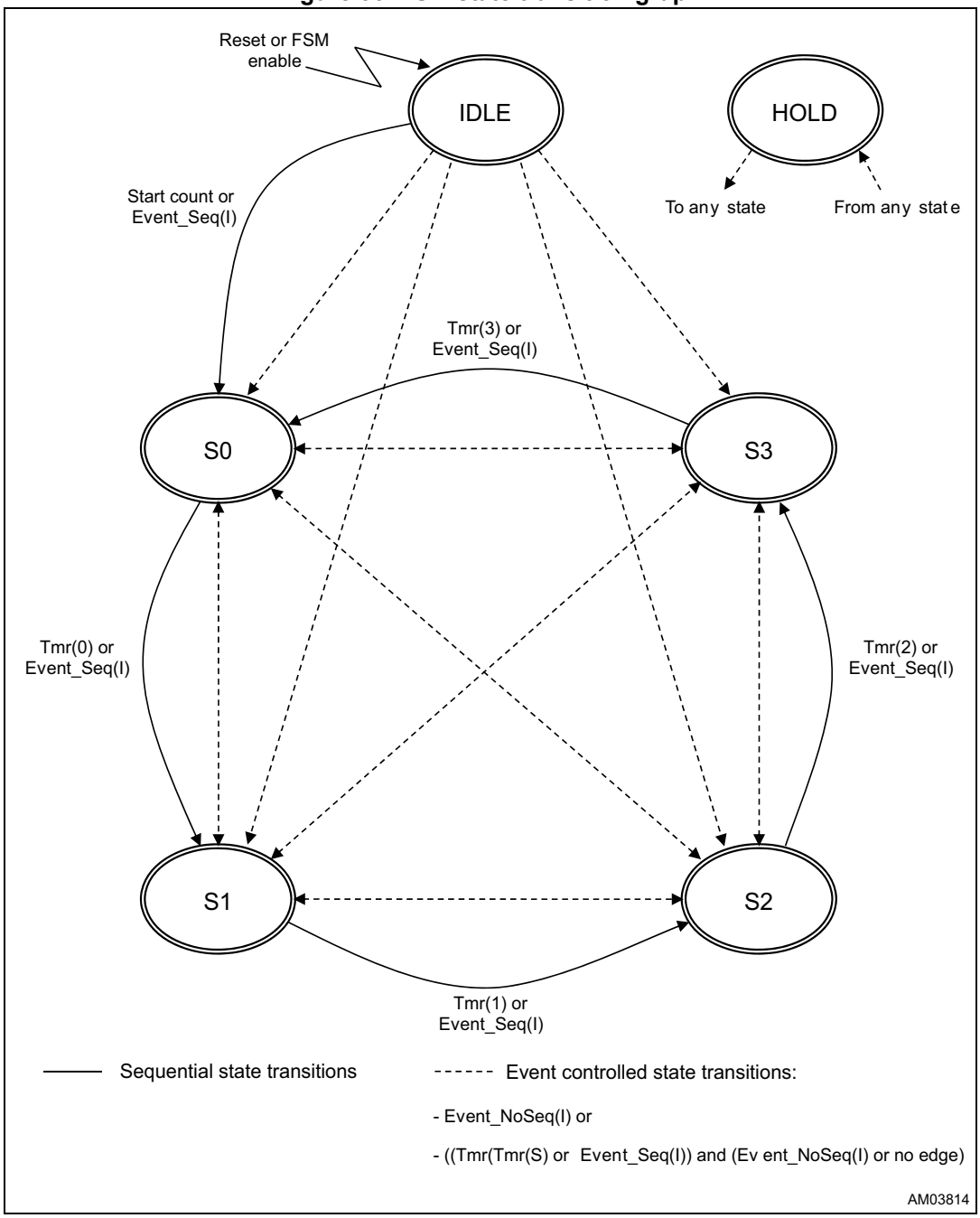
In case the FSM timer counter is not enabled (refer to START_CNT field of SMD<n>_CTR register *Section 21.18.1: SMED core registers on page 199*), the FSM state evolution depends only on both external event s according to the following rules:

- All states including the IDLE state must be configured to exit on the Event_NoSeq(I) or Event_Seq(I) event.
- In case of the Event_NoSeq(I) event the next state is determined by the value programmed into the register parameters SMD<n>_PRM_<st>0.
- In case of the Event_Seq(I) event the next state is the sequential one.
- If the AND_OR bit of the SMD<n>_PRM_<st> register (<st> = S0, S1, S2, S3, ID) the behavior is modified.

For further details refer to the AND_OR functionality description.

Figure 36 shows an outline view of the SMED finite state machine state flow diagram.

Figure 36. FSM state transition graph



21.9.1 IDLE state

The IDLE state is the initial state of the SMED FSM. This state is only entered asynchronously by a reset condition (power-on reset, microcontroller reset or peripheral reset), or synchronously whenever the FSM is disabled. The exit from this initial state may be caused by the following conditions:

- Event Event_NoSeq(I), if programmed, allows jumping to any state: S0, S1, S2, S3 and HOLD.
- Event Event_Seq(I), if programmed, allows jumping to the state S0 (refer to *Section 21.9.5: State evolution equations on page 176*).
- Start FSM counter, if enabled causes to jump to the S0 state.

The initial value of the PWM output signal before the FSM exits from the IDLE state is configured by the Rst_PWM[5:0] option bits of the GENCFG register (for further information refer to the product datasheet).

Note: If the initial value of a PWM output is configured 'high' through the Rst_PWM[5:0] option bits, the high level is applied after the system has loaded the option byte values from the EEPROM. The initialization sequence is done after about 50 µs from the release of the reset signal; during this interval time, the pad is driven low (refer to the product datasheet).

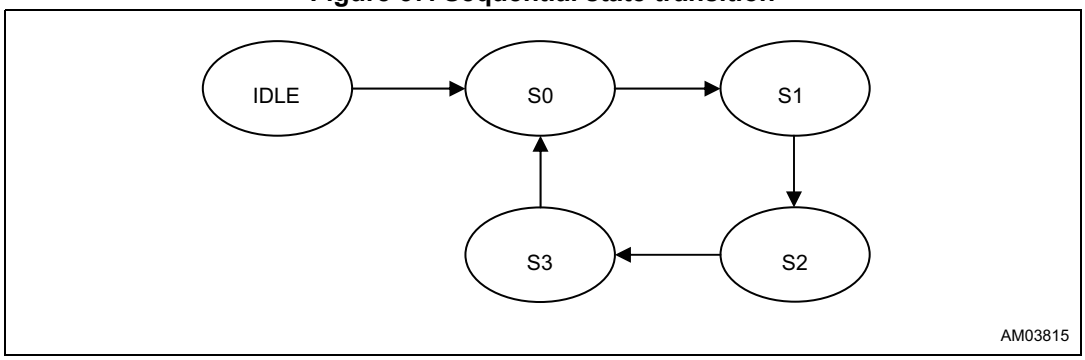
21.9.2 HOLD state

The HOLD is special state condition of the SMED FSM where the state machine is frozen until the HOLD exit condition is reached. This particular state is useful either when the SMED works in one of the available coupled modes or is configured in the single mode. In the HOLD state, the counter is stopped, freezing the current value until the SMED exits from this state resuming the functional state evolution. For the complete information about all possible SMED coupled configuration schemes refer to the product datasheet.

21.9.3 Sequential state transitions (S0→S3)

The FSM evolves sequentially among states S0, S1, S2 and S3 whenever a Tmr(S) or an Event_Seq(I) event is triggered(1).

Figure 37. Sequential state transition



The **Tmr(S)** event is triggered when the FSM 16-bit counter is equal or greater than the 16-bit register timer value (SMD<n>_TMR_T<3:0>L/H) associated to the current FSM state(2). Note that only the comparator associated to the current FSM state is active.

If the FSM counter is not enabled (refer to START_CNT bit in the SMD<n>_CTR register *Section 21.18.1: SMED core registers on page 199*), the FSM sequential evolution is determined only by the Event_Seq(I) events.

The **Event_Seq(k)** (k = 0, 1, 2) event is triggered when the following conditions are met contemporaneously:

- The InSig[k] line is enabled through the SMD<n>_ISEL register.
- The InSig[k] input signal matches the configured capture condition (falling/rising edge, low/high level) programmed through the SMD<n>_CTR_INP register.
- The InSig[k] is programmed as a source of the Event_Seq(k) sequential event in the current state. In each state only one at a time of the InSig input signals may be programmed as sequential event (refer to the CEDGE field of register *SMD<n>_PRM_<st>1 (parameter1 register) in Section 21.18.1: SMED core registers on page 199*).

Both events Tmr(S) and Event_Seq(I), whichever comes first, determine the FSM sequential evolution state in the circular mode as shown in *Figure 37*. If the events are raised simultaneous the priority scheme in *Section 21.8: Events priority on page 171* is applied.

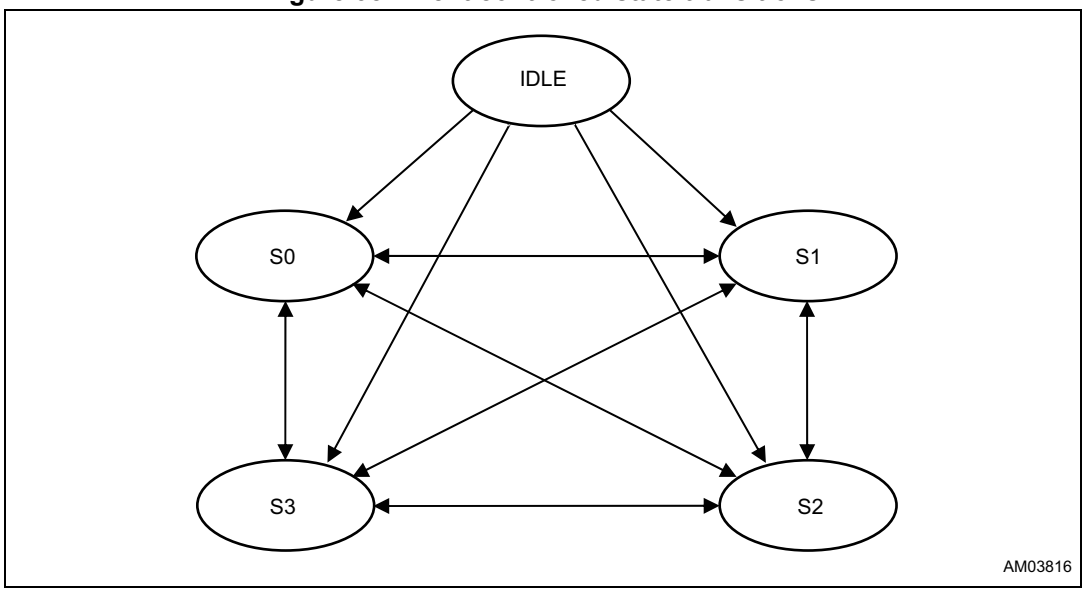
Note: This behavior may be modified by the AND_OR and HOLD_JMP bits of SMD<n>_PRM_<st>0 register (refer to Section 21.9.5: State evolution equations on page 176).

In the IDLE state the Tmr(S) event is substituted by the timer activation (SMD<n>_CTR[0] = '1') because there is no timer associated to this state.

21.9.4 Non-sequential state transitions

The FSM state evolution is not fixed; any evolution scheme is allowed by configuring the parameter registers of states S0 - S3 whenever an Event_NoSeq(I) event is triggered (1) as shown in *Figure 38*.

Figure 38. Event controlled state transitions



The **Event_NoSeq(k)** (k = 0, 1, 2) event is triggered when the following conditions are met contemporaneously:

- The InSig[k] line is enabled through the SMD<n>_ISEL register.
- The InSig[k] input signal matches the configured capture condition (falling/rising edge, low/high level) programmed through the SMD<n>_CTR_INP register.
- The InSig[k] is programmed as a source of the Event_NoSeq(k) non-sequential event in the current state. In each state only one at a time of the InSig input signals may be programmed as non-sequential event (refer to the EDGE field of the register *SMD<n>_PRM_<st>0 (parameter0 register) in* *Section 21.18.1: SMED core registers on page 199*).

The next state is programmed by the NX_STAT bits of the SMD<n>_PRM_<st>0 register (refer to register programming in *SMD<n>_PRM_<st>0 (parameter0 register)*) of *Section 21.18.1: SMED core registers on page 199*.

Note: This behavior may be modified by the AND_OR and HOLD_JMP bits of the SMD<n>_PRM_<st>0 register (refer to Section 21.9.5).

21.9.5 State evolution equations

The FSM evolves following predefined “transition equations” configured by the register parameters. These are the HOLD_JMP and the AND_OR bit of SMD<n>_PRM_<st>0 register (refer to register programming *in SMD<n>_PRM_<st>0 (parameter0 register) of* *Section 21.18.1: SMED core registers on page 199*).

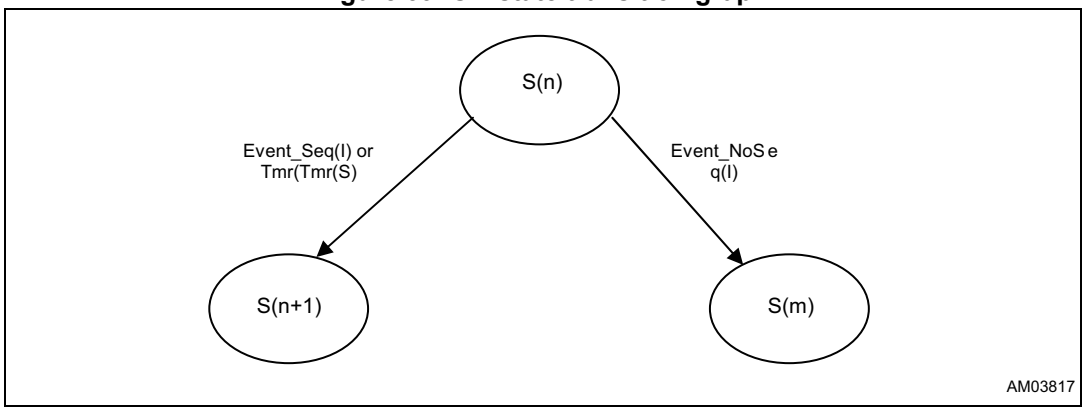
The following description assumes that the HOLD_JMP bit of the SMD<n>_PRM_<st>0 register is cleared. Refer to *SMD<n>_PRM_<st>0 (parameter0 register)* in *Section 21.18.1: SMED core registers on page 199* for further details on the HOLD_JMP parameter.

- AND_OR flag cleared (OR condition applies):

  The FSM changes the state when the first of the configured triggering events is raised.
  - If the triggering event is Tmr(S) or Event_Seq(I), the following state will be the next sequential state.
  - If the triggering event is Event_NoSeq(I), the next state may be any one of the S0, S1, S2 or S3 programmed states.

This is the case described in *Section 21.9.3: Sequential state transitions (S0→S3)* and *Section 21.9.4: Non-sequential state transitions*. The corresponding state transition graph is shown in *Figure 39*.

Figure 39. OR state transition graph



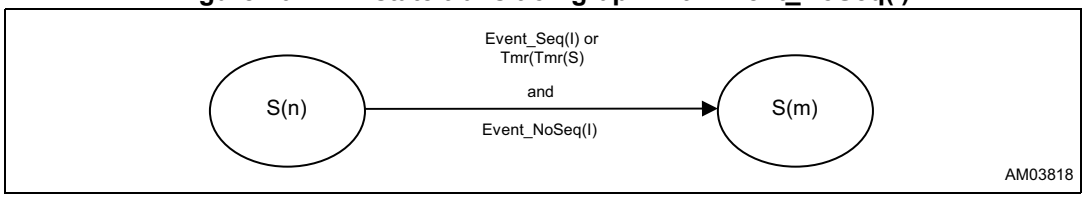
- AND_OR flag set (AND condition applies) (with Event_NoSeq(I) configured):

  The FSM changes the state when both the sequential state transition condition (Tmr(S) or the Event_Seq(I)) and the programmed state transition condition (Event_NoSeq(I)) are satisfied contemporaneously (i.e. within the same FSM clock cycle).

  The new state is the one selected by the programmed state transition (see NX_STAT bits of SMD<n>_PRM_<st>0 register *in SMD<n>_PRM_<st>0 (parameter0 register) of Section 21.18.1: SMED core registers on page 199*).

  The corresponding state transition graph is shown in *Figure 40*.

Figure 40. AND state transition graph with Event_NoSeq(I)



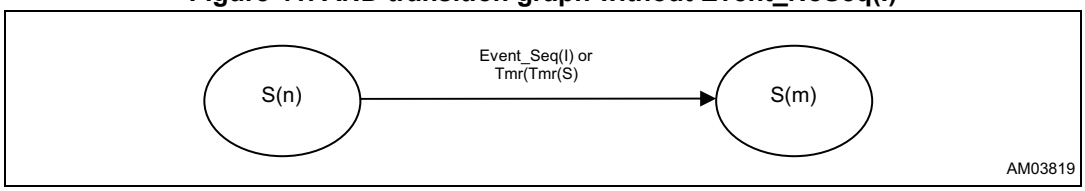
- AND_OR flag set (AND condition applies) (with no Event_NoSeq(I) configured):

  The FSM changes the state when the sequential state transition conditions (Tmr(S) or Event_Seq(I)) are satisfied.

  The new state is the one selected by the programmed state transition (see NX_STAT bits of SMD<n>_PRM_<st>0 register in *SMD<n>_PRM_<st>0 (parameter0 register) of Section 21.18.1: SMED core registers on page 199*).

  The corresponding state transition graph is shown in *Figure 41*.

Figure 41. AND transition graph without Event_NoSeq(I)



21.10 State transition occurrence

To support the generation of complex output signals two or more SMEDs can be configured in coupled schemes (for further information refer to *Section 21.17: SMED coupled expansion interface on page 189*). *Table 31* summarizes the possible FSM state transitions in every configuration.

Table 31. FSM states transition overview

Global config.(1)	Conf. parameter fields		Active events		Next state	
Coupled/!single configuration	HOLD_JMP	AND_OR	Tmr(S) OR Event_Seq(I)	Event_NoSeq(I)	HOLD	S0,S1,S2,S3
Single/coupled	0	0	1	0	No	Sequential
Single/coupled	0	0	1 or 0	1	No	NX_STAT[1:0]
Single/coupled	0	1	1	1(2)	No	NX_STAT[1:0]
Single only	1	0	1	0	No	Sequential
Single/coupled	1	0	1 or 0	1	Yes	NX_STAT[1:0](3)
Coupled only	1	0	1	X(4)	Yes	NX_STAT[1:0](3)
Single/coupled	1(4)	1(4)	1	1(2), (4)	Yes	NX_STAT[1:0](3)

1. Parameter configured by the SMD<n>GLBCONF field of MSC_SMDCFG<xy> registers.
2. In these cases the Event_NoSeq(I) condition is considered always true if the Event_NoSeq(I) option is not selected for the considered state (EDGE[1:0] = “00”).
3. Next state after the Hold exit.
4. If the No Event_NoSeq(I) is used the SMED can exit from the HOLD state only when the corresponding coupled SMED enters the HOLD state. In this case it is mandatory to use a paired configuration by setting the HOLD_EXIT bit of the SMD<n>_PRM_<st>1 register.

Two fundamental operations are carried out when a transition occurs: the output PWM level change and the timer counter reset. These actions are performed on the exit from the current state even if the FSM passes through the HOLD state. The SMD<n>_PRM_<st>0 and SMD<n>_PRM_<st>1 contains the relevant configuration bits which determine the output level (PULS_CMP, PULS_EDG) and the counter reset on transition (CNT_RSTC, CNT_RSTE).

The above bit fields are used alternatively in accordance with the following rules:

- If the AND_OR bit is set or the state transition is raised by an Event_NoSeq(I) event, the PULS_EDG and CNT_RSTE register bits are used.
- If the AND_OR bit is reset and the state transition is raised by a Tmr(S) or Event_Seq(I) event, the PULS_CMP and CNT_RSTC register bits are used.

21.10.1 Output PWM level

The PWM output signal level is configured through the register parameters available for every state; during the FSM state evolution the PWM output signal changes polarity according to the value configured into the state parameter registers; then by programming the state timing and the transition sequence it's possible to generate the desired PWM timing pulse.

The output signal is updated whenever the state transition occurs, including transitions to the HOLD state where the output signal is updated on entering the HOLD state itself, The PWM level is always maintained when the FSM exits from the HOLD state.

The PWM timing pulse accuracy increases with the SMED clock frequency; a high frequency (configurable up to 96 MHz) enhances the PWM output signal timing resolution.

Note: 1. If the PWM signal is used internally (i.e. SMD connected in coupled configuration), it doesn't have limitation in term of the toggle rate.

2. If the PWM signals is used as primary output pins its toggle rate has not to exceed the maximum toggle frequency of its own output pad (for further details refer to the product datasheet).

The level of the PWM output signal may be configured independently for sequential and non-sequential state transitions. The user has to configure the PULS_EDG bit of the SMD<n>_PRM_<st>0 register (refer to *SMD<n>_PRM_<st>0 (parameter0 register) in Section 21.18.1: SMED core registers on page 199)* for non-sequential and the PULS_CMP of the SMD<n>_PRM_<st>1 register (refer to *SMD<n>_PRM_<st>1 (parameter1 register) in Section 21.18.1: SMED core registers on page 199*) for sequential transitions.

If the AND_OR bit is set, the FSM takes into consideration the PULS_EDG bit to change the PWM level on a transition.

The PWM output signal has an additional dithering functionality that can be dynamically configured by software; for further information refer to *Section 21.11.2: Dithering*.

21.10.2 Counter reset

By default the FSM counter, when enabled, increments its value on every FSM clock cycle independently by the edge events, and wraps to 0x0001 when it exceeds the maximum count value (0xFFFF).

When the FSM state changes, the internal counter may be reset selectively either by non-sequential state transitions (bit CNT_RSTE of the SMD<n>_PRM_<st>0 register *SMD<n>_PRM_<st>0 (parameter0 register) in Section 21.18.1: SMED core registers on page 199*) or by sequential state transitions (bit CNT_RSTC of the SMD<n>_PRM_<st>1 register *in SMD<n>_PRM_<st>1 (parameter1 register)* of *Section 21.18.1: SMED core registers on page 199*).

If the AND_OR bit is set, the FSM takes into consideration the CNT_RSTE bit to change the PWM level on a transition.

In this way it is possible to have the following timing periods:

- Cumulative period for more states (no counter reset between different states).
- Independent period for each state (reset counter between any states).

A simple example of the counter reset functionality is the generation of a PWM signal for a fixed time period (TP), with a granted minimum high level initial period (T0) and a granted minimum low period at the end (T2). The states sequence is described in *Figure 42*.

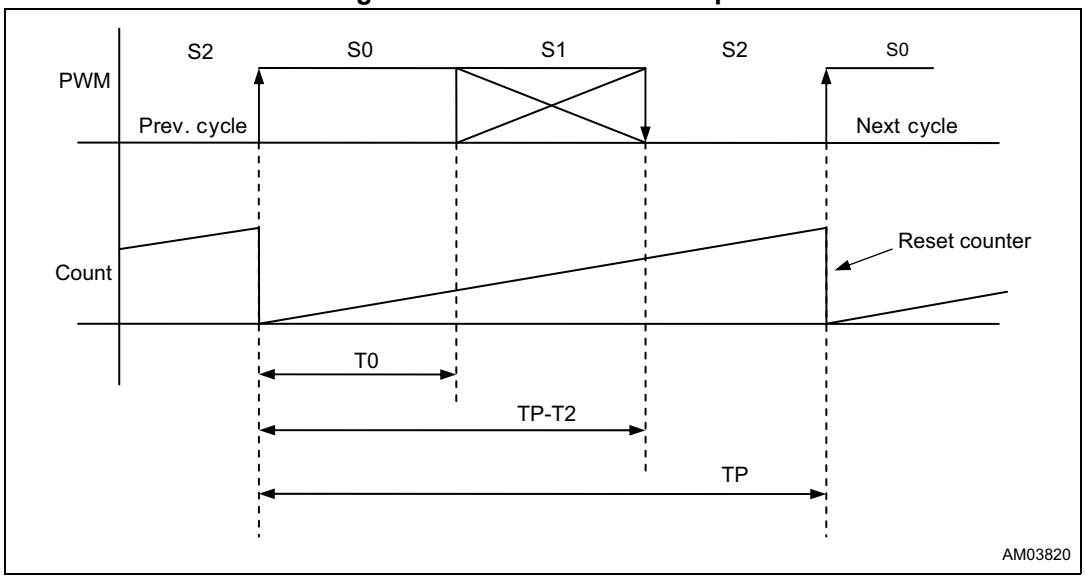
The PWM signal is generated using three states: the S0 minimum initial high level period, S1 intermediate period and S2 minimum final low level period.

The state S0 is programmed for a period T0 and is sensible only to the Tmr(0) event; this grants the minimum period of the output high voltage level. On the exit from the S0 the PWM output is left unchanged and the counter is left running.

The state S1 is programmed for a period equal to TP-T2 and is sensible both to the Event_Seq(I) and to Tmr(1) whichever came first (OR configuration). On the exit from the S1 the PWM output is cleared at low voltage and the counter is left running.

The state S2 is programmed for a period TP and is sensible only to the Tmr(2) event (with AND_OR and no_Event to S0); this grants the minimum period of low voltage output level. On the exit from the S2 the PWM output is set and the counter is reset starting a new cycle.

Figure 42. Counter reset example



21.10.3 Hold jump

The HOLD functionality is used when it's required to freeze the current state of the SMED until some external event is triggered. The HOLD state may be entered by any of the <st> = ID, S0, S1, S2 or S3 state, setting the corresponding HOLD_JMP bit of the SMD<n>_PRM_<st>0 register (refer to the register programming *in SMD<n>_PRM_<st>1 (parameter1 register) of* *Section 21.18.1: SMED core registers on page 199*).

The HOLD state is commonly used when two or more SMEDs are coupled together to form a more complex state machine able to drive more than one PWM signal at the same time or a more complex merged PWM signal (further description is found in *Section 21.17: SMED coupled expansion interface on page 189*), but it can also be used in single SMED configuration with a little different behavior.

The following paragraph the HOLD_JMP bit is assumed active.

- Single mode:
  Two cases are possible according to the value of the AND_OR register field:
  - If the AND_OR bit is cleared, the HOLD state is entered from any state when the state transition is driven by an Event_NoSeq(I) event.
  - If the AND_OR bit is set, the HOLD state is always entered when the state evolution equation is satisfied. For further information refer to the AND_OR bit programming *Section 21.9.5: State evolution equations on page 176*.
- Coupled mode:
  - If the AND_OR bit is cleared, any of the Event_Seq(I), Event_NoSeq(I) or Tmr(S) events triggers the transition to the HOLD state.
  - If the AND_OR bit is set, the HOLD state is always entered when the state evolution equation is satisfied. For further information refer to the AND_OR bit programming *Section 21.9.5: State evolution equations*.

Note: For the complete information about all possible SMED coupled configuration schemes refer to Section 21.17: SMED coupled expansion interface on page 189.

21.10.4 Hold exit

When an FSM is frozen in the HOLD state, it waits until the hold exit condition is reached. The event used to exit from the HOLD state depends on the SMED behavior (single mode or coupled mode) and on the HOLD_EXIT bit of SMD<n>PRM_<st>1 register (see register programming in *SMD<n>_PRM_<st>1 (parameter1 register) of Section 21.18.1: SMED core registers on page 199*).

When the FSM exits from the HOLD state, the next state is configured exclusively by the NX_STAT bits field of register SMD<n>_PRM_<st>0. Refer to *SMD<n>_PRM_<st>0 (parameter0 register) in Section 21.18.1: SMED core registers on page 199*. When an SMED is in HOLD, the FSM monitors the related InSig[I] programmed as a source of the Event_NoSeq(I) in the state preceding the HOLD[(i)]. In the single mode and with AND_OR = '0' the Event_NoSeq(I) is also the only event can cause the SMED to enter the HOLD state. The AND_OR bit doesn't influences the HOLD exit mechanism.

- Single mode:
  The HOLD_EXIT bit must be cleared in this configuration mode; then the FSM exits from the HOLD state when the capture condition on the “exit InSig” is triggered (if it is an edge sensible input) or becomes false (if it is a level sensible input).
- Coupled mode[(j)]:
  - HOLD_EXIT cleared
    The FSM exits from the HOLD state when the capture condition on the “exit InSig”

i. When the HOLD_EXIT register bit is cleared, the FSM can exit from the HOLD state only through the specified behavior on the “exit InSig”. In this case, the EDGE[1:0] field relative to the origin state must be different from “00”.

j. For the complete information about all possible SMED coupled configuration schemes refer to the coupled expansion interface Section 18.17.

is triggered (if it is an edge sensible input) or becomes false (if it is a level sensible input).

– HOLD_EXIT set
The FSM exits from the HOLD state when the corresponding coupled SMED enters the HOLD state.

Note that it is possible to configure two coupled SMED running at the same time with the HOLD_EXIT bit cleared. The coupling programmed through MSC_SMDCFG<xy> registers in this case just forces either the SMED to jump to HOLD, if the origin state HOLD_JUMP bit is set, even if the transition is of the Tmr(S) or Event_Seq(I) type.

21.11 Time control

The FSM has an embedded internal 16-bit upcounter that is incremented at every cycle of the programmed clock; this provides the time base for the deterministic PWM timing pulse generation (refer to *Section 21.18.1: SMED core registers on page 199*).

This counter is started by the START_CNT bit in the SMD<n>_CTR register (refer to the registers programming in *SMD<n>_CTR (control register)* of *Section 21.18.1: SMED core registers on page 199*) and runs until the SMED is active. When the counter reaches the maximum count value (0xFFFF) automatically wraps to 0x0001 continuing the counting process (for further detail refer to *Section 21.18.1: SMED core registers*).

The counter is handled by the FSM both to generate programmed timing (state timers compare) and to generate the time stamp information (dump).

The counter may be reset on a state transition if the CNT_RSTE or CNT_RSTC register bits are set (refer to *Section 21.10.2: Counter reset*).

When the FSM enters the HOLD state, the counter is frozen and resumes counting when the FSM exits from the HOLD state, unless the involved CNT_RSTE bit or CNT_RSTC bit are set, restarting the counter from 0x0001 (refer to registers programming *Section 21.10.2*).

When the counter reaches the overflow condition, it wraps back to 0x0001 and, if the CNT_OV_E bit field of the SMD<n>_IER register is '1', the CNT_OVER bit of the SMD<n>_IRQ register is set and an interrupt is generated.

21.11.1 State timer registers

The FSM single 16-bit timer counter register is compared every clock cycle with the value programmed in the timer constant registers of the current state S<x> (x = 0, 1, 2, 3) The 16-bit timer constant register is configured through the lower byte SMD<n>_TMR_T<x>L and upper byte SMD<n>_TMR_T<x>H registers, where x represent the state number (0:3). The IDLE and HOLD states are not equipped with the timer constant registers since they don't use the timer functionality.

To avoid conflict when updating the contents of the upper and lower bytes of the state timer comparison registers, a shadow register technique is used. Each state timer has a corresponding 16-bit shadow register that holds a copy of the timer registers contents.

- If the FSM is not active, the update of the state timer registers updates also the corresponding shadow register.
- If the FSM is running, the update of the state timer registers does not automatically update the shadow registers; these are updated synchronously (all 16 bits at a time)

when the corresponding TIME_Tx_VAL bit of the register SMD<n>_CTR_TMR is set, validating the new contents, and when the condition selected by the TIM_UPD[1:0] field of the SMD<n>_CFG register occurs (refer to register programming in *SMD<n>_CFG (timer configuration register)* of *Section 21.18.1: SMED core registers*).

The content of the upcounter is continuously compared with the content of the state timer shadow register of the current active state. If the counter is equal or greater than the state timer, the corresponding Tmr(S) event is generated; this event generates the state transition according to the configured transition equations (refer to *Section 21.9.5: State evolution equations on page 176*).

Note: 1. The SMD<n>_CTR_TMR[4:0] register bits can be updated only when the register content is cleared; while set by SW in single or multi-bit configuration, these fields have to be cleared only by HW. Although the register bits can also be cleared by SW for a diagnostic purpose, this operation is forbidden in the functional mode.

2. When $f_{SMED} < f_{MASTER}$, the back-to-back SMD<n>_CTR_TMR write register sequence is allowed only when the register is cleared and after $2t_{SMED}$ additional latency time (this delay is not requested in case of $f_{SMED} \geq f_{MASTER}$).

3. SMD<n>_TMR_T<i>L/H constant registers are writable by SW only when the corresponding SMD<n>_CTR_TMR[3:0] register control bit is cleared; if the control bit is set the corresponding constant register is locked until the proper control bit is cleared again; during this phase any attempt to write the constant register is ignored.

21.11.2 Dithering

In particular cases, generating a periodic signal with a precise fixed timing may cause resonance or EMI problems in the hardware application where capacitive and/or inductive loads are present.

To avoid these problems, the programmed values of the state timers and consequently the PWM output signal, may be dynamically modified at run time configuring the dithering functionality.

The dithering functionality is obtained by temporarily incrementing the contents of the state timer register of 1 cycle. The dithering is applied to timer comparator value of one of the four available states (S0, S1, S2 and S3) selected by the TIM_NUM field of the SMD<n>_CFG register. The dithering is programmable over a period of 8 state cycles, by setting the corresponding bits of the SMD<n>_CTR_DTHR register; if the register bit is set, then the corresponding cycle in the selected state timer is temporary incremented by 1 cycle (refer to the register programming *in SMD<n>_CTR_DTHR (dithering register)* of *Section 21.18.1: SMED core registers*).

Note: 1. By design the cycle index of dithering functionality is updated on the rising edge of the PWM SMED output signal; this has to be taken in account especially when the PWM idle level (reset value) is configured active high.

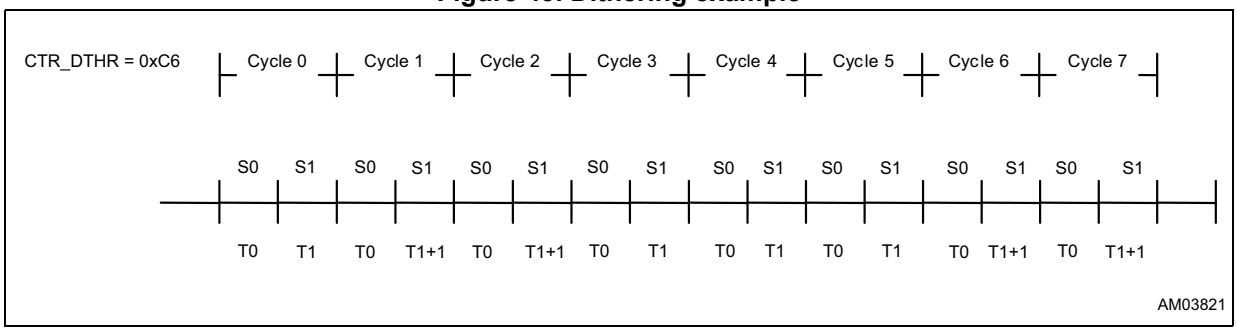
2. Note that soon after the SMED start the cycle index points to the bit position 0 of the SMD<n>_DITHER register. If the SMED pulse makes a transition to '1' exiting from IDLE, the index in the first SMED cycle will point to the position 1, in full respect of the rule explained in the preceding note.

Like the state timers, the dithering register is also implemented with a shadow mechanism.

- If the FSM is not running, the update of the dithering register also updates the shadow register.
- If the FSM is enabled, the update of the dithering register does not update the shadow register; the register is updated when the DITHER_VAL bit in the register SMD<n>_CTR_TMR is set validating the new contents, and the FSM is not in the state hosting the dither (refer to the register programming *Section : SMD<n>_CTR_TMR (control time register) on page 200*).

Figure 43 shows an example of the dithering functionality applied to an SMED that runs cyclically on states S0 - S1 with respective timings T0 and T1. The dithering is applied to cycles 1, 2, 6 and 7. Note that in the current example the SMED PWM signal toggles at every state updating the dithering cycle counter.

Figure 43. Dithering example



Note: 1. The SMD<n>_CTR_TMR[4] register bit can be updated only when the register content is cleared; while it's set by SW, this field has to be cleared only by HW. Although the register bits can also be cleared by SW for a diagnostic purpose, this operation is forbidden in the functional mode.

2. The SMD<n>_CTR_DTHR constant register is writable any time by SW only when the corresponding SMD<n>_CTR_TMR[4] register control bit is cleared; if the control bit is set the corresponding constant register is locked until the proper control bit is cleared again; during this phase any attempt to write the constant register is ignored.

21.11.3 Time stamp functionality

For control purposes sometimes it may be necessary to know the FSM current count when an external event occurs.

The time stamp functionality is triggered either by the Event_Seq(k) or Event_NoSeq(k) transition event if the dump functionality is enabled for the InSig[k] line. Tmr(S) events are ignored for this purpose.

By configuring the DMPE_EX[k] (k = 0, 1, 2) field of the SMD<n>_DMP register (refer to the register programming *Section 21.18.1: SMED core registers*) the running time value of the FSM counter is instantaneously copied in two dedicated 8-bit registers which stores the upper byte (SMD<n>_DMP_H) and the lower byte of the counter value (SMD<n>_DMP_L) when the corresponding Event_Seq(k) or Event_NoSeq(k) event is triggered. If AND_OR bit = '1' the dump is triggered if the Event_Seq(k) is one of the causes of the transition.

This functionality is controlled by the DMP_EVER bit of the SMD<n>_DMP register.

- DMP_EVER cleared:
  Only the first triggered event dumps the counter time value; the successive events are ignored until the corresponding EX<k>_DUMP bit of SMD<n>_GSTS register is cleared by SW.
- DMP_EVER set:
  Any configured events trigger and override the content of the SMD<n>_DMP_L/H registers with the current run time counter value (the registers contain the counter value of the latest dump event).

Note: When the DMP_EVER bit field of the register SMD<n>_DMP is set, the maximum dump rate supported by dump logic is expressed by the following equation:

f_{DUMP_Max} = 1 / [4 (t_{MASTER} + t_{SMED})]. In case of a fast dump request, the event that exceeded the f_{DUMP_Max} frequency is simply ignored and the SMD<n>DMP_L/H dump registers maintain the time information of the previous dumped event.

21.12 Interrupt functionality

The SMED implements eight dedicated interrupt sources. The following list shows all possible interrupt source requests:

- FSM counter overflow.
- Exit for Event_Seq(I)/Event_NoSeq(I) due to InSig[I] capture with I = 0, 1, 2.
- State S exit for Tmr(S) conditions with S = 0, 1, 2, 3.

The interrupt source requests are individually controllable through the SMD<n>_IER interrupt enable register which enables selectively each interrupt sources and by the SMD<n>_IRQ interrupt request register that contains the interrupt flag requests. Every interrupt flag request is reset by writing '1' to the corresponding bit position.

Note: 1. The interrupt request bits in the SMD<n>_IRQ register are set only if the corresponding bits in the SMD<n>_IER register are set.

2. The reset of the SMED interrupt flag request, asserted by writing '1' into the SMD<n>_IRQ register bits, due to the cross clock logic domains requires the latency time expressed by the next equation: t_{ISR_Rst} = t_{MASTER} + 3 t_{SMED}. This latency has to be considered when the f_{SMED} < f_{MASTER} and the interrupt of SMED logic is triggered by multiple sources. In this case SW has to take care of this aspect when the interrupt flag is evaluated.

3. The interrupt enable/disable sequence performed by writing the SMD<n>_IER register due to the cross clock logic domains requires the following latency time: t_{IER_Lat} = t_{MASTER} + 3 t_{SMED}. Note that an interrupt source is masked only when the corresponding mask reading bit of the SMD<n>_IER register is cleared (after a time t_{IER_Lat}).

4. The minimum time of contiguous write back-to-back cycle to the SMD<n>_IER register, due to the cross clock logic domains is expressed by the next equation: t_{IER_Wb} = 6 t_{MASTER} + 4 t_{SMED}. The subsequent write faster than the t_{IER_Wb} minimum time is ignored.

The interrupt capability is useful when events that are handled in real-time by the FSM, require also an exception handling by the SW application process due to special events or error conditions that must be served in a restricted time limit.

The generations of an interrupt flag, not immediately served by the corresponding interrupt service routine, doesn't affect the behavior of the FSM that is completely autonomous and can continue to run independently from the CPU activity.

The application programmer shall pay attention in configuring the SMED interrupts, especially when SMED logic is running at higher frequency, since the interrupt rate may impact the CPU bandwidth operation.

21.13 Status information

To support a complete state transition handling, in addition to the interrupt functionality, the SMEDs implement dedicated registers containing several status flags that resume the current FSM status.

The status flags are grouped in two distinct registers that collects logically separated conditions.

21.13.1 GSTS (global status register)

This register contains the bit fields that monitor the status of the time stamp functionality; the list of the register fields is shown below:

- The dump event for InSig event transition occurred (EX[2:0]_DUMP).
- The counter reset or overflow when the dump is enabled (CNT_FLAG).
- Dump register lock status (DMP_LK[1:0]): this field controls the logic that preserves the integrity of the 16-bit dump value when the DUMP_EVER option is enabled.
- The dump event overflow (EVENT_OV): set by HW when the DUMP_EVER is cleared and a new event would trigger a dump while the preceding one has not yet been handled by SW.

The SW application program may synchronously poll these flags, resetting them in the time stamp handling routine.

Note: 1. The EX[2:0]_DUMP bits are reset by writing '1' to the corresponding bit (or also by clearing the corresponding EXT[3:1]_INT bit field of the SMD<n>_REQ register if the SMD<n>_DMP[4] bit is set).

2. The CNT_FLAG bit is reset by writing '1' to this field.

3. The DMP_LK field is reset only by reading consecutively both SMD<n>_DMP_L/H registers.

4. The EVENT_OV field is reset by clearing any of the EX[2:0]_DUMP flags (or also by clearing any of the EXT[3:1]_INT flags of the SMD<n>_IRQ register if the SMD<n>_DMP[4]bit is set).

21.13.2 FSM_STS (finite state machine status register)

This register contains the real-time information related to the status of the FSM. The list of the available bit fields is the following:

- Current FSM state (FSM[2:0])
- Current PWM output level (PWM)
- Current InSig levels (EVINP[2:0])

The application program can synchronously poll these flags resetting them in the handling routine.

Table 32 shows the coding value of the FSM field belonging to the SMD<n>_FSM_STS register (refer to *SMD<n>_FSM_STS (FSM status register)* in *Section 21.18.1: SMED core registers on page 199*).

Table 32. FSM states coding

FSM states encoding value	
FSM state	**FSM_STS[2:0] coding**
S0	000
S1	001
S2	010
S3	011
IDLE	100
HOLD	101

21.14 PWM reset output value

The PWM signal reset output value can be configured individually for each SMED unit through a suitable option byte (refer to the product datasheet for details). The initialization value is loaded into the PWM output register during the option byte loading phase which follows the NRST signal release. Each PWM[n] takes its reset value as long as the driving SMED<n> remains or is forced in the IDLE state (by setting the SMD<n>_CTR[1] bit). During the IC configuration process the CPU core is stalled in the reset state.

21.15 PWM pad (pseudo) open drain configuration

Depending on the product, an option byte may be enabled to allow the PWM to be configured in the (pseudo) open drain mode. The option can be applied selectively on each PWM by means of the P1_CR1 register, which controls the similar functionality on the same port configured as the GPIO. A logical high level on a (pseudo) open drain port will result in the Hi-Z state. Refer to the product datasheet to check if this feature is available on your product.

21.16 SMED as ADC trigger source

Depending on the product, the ADC conversion may be triggered by various internal sources, other than issued by a software command. The SMED HW trigger feature is enabled by the option bit SMED_HWtrg of the CLKCTL register set a '1' and requires that the ADC_HWtrg option bit of the AFR_IOMXP2 register is programmed at '0'.

The SMED is able to trigger the ADC conversion start through the Tmr(2) event interrupt. Refer to *Section 27: Analog-to-digital converter (ADC) on page 312* for details on the configuration of the functionality. When the SMED<n> Tmr(2) interrupt is enabled and selected as the ADC SoC source, the interrupt is masked (i.e. does not generate an IRQ on the SMED<n> interrupt channel) and the request is autoreset on the ADC starting conversion sequence.

Note: The HW trigger functionality requires that all SMEDs of the IC device must be configured with $f_{SMED} \geq f_{MASTER}$.

Due to the capability of the IC device to enter low power states by disabling all clock sources, particular caution has to be adopted in order to avoid impact on the SMED logic interface. *Table 33* details the correct procedure to prepare the SMED peripheral to a low power processor state.

Table 33. SMED low power mode

Modes	Description
Wait	No effect on SMED logic.
Halt Active-halt	Before entering the Halt/Active-halt mode the SMED clock must be stopped through the PCKEN2<n> bit of the CLK_PCKENR2 register, then if the PLL is enabled it has to be configured in power-down by clearing the bit PLLON of the CLK_PLLR register. After waking up from Halt/Active-halt, if required by the application, the PLL has to be enabled by setting the PLLON bit of the CLK_PLLR register, as soon as PLL is locked, the SMED clock can be enabled once more by disabling the clock gating features.

The SMED logic does not have the capability to wakeup the device from the Active-halt or Halt mode.

21.17 SMED coupled expansion interface

The expansion interface allows expanding the functionality of a single SMED by grouping other SMED logic together to form a more complex SMED unit capable to generate more complex PWM signals.

The expansion interface logic can be used in case the IC device is provided by several SMED logics (for further information refer to the product datasheet), these can be connected in single coupled or two coupled configurations increasing the performances of a single SMED. The main advantage of coupled configuration is summarized below:

1. To generate complex waveforms on a single PWM channel (involving essentially the eight states belonging to the two SMEDs) by using the SMED<n>_DRVOUT register field.
2. To obtain two interlocked signals on two distinct PWM channels in order e.g.: to permit anti-phase control of high-side and low-side drivers with minimum external logic.

The arbitration between SMED activities is accomplished by the HoldJump/HoldExit logic mechanism and by the Asy_Enable asynchronous signal line in the case of two coupled SMEDs.

21.17.1 SMED coupled interface configuration scheme

The SMED basic operation is defined by the SMDx_GLBCONF and SMDx_DRVOUT fields of the MSC_SMEDCFG<xy> registers; the register coding values is shown in *Table 34* (for further details refer to *Section 21.18.2: SMED environment configuration registers on page 215*).

These register fields select the SMED operating mode that must be configured before enabling the involved SMEDs and cannot be changed until the SMED logics are disabled.

The only exception at this rule is represented by the selection of unlock feature in coupled SMEDs configuration (refer to *MSC_SMULOCK (SMEDs unlock)* in *Section 21.18.1: SMED core registers on page 199*).

Table 34. SMED coupled interface configuration scheme

SMEDs global configuration		Description	
SMD<n>_GLBCONF	SMD<n>_DRVOUT	Main feature	PWM output pulse
000	0	Single SMED	Independent PWMs (fixed by parameters)
	1		RFU (reserved encoding for future use)
001	0	Synchronous coupled SMEDs	Independent PWMs (fixed by parameters)
	1		Combined PWM (driven by active SMEDs)
010	0	Two synchronous coupled SMEDs	Independent PWMs (fixed by parameters)
	1		Combined PWM (driven by active SMEDs)
011	0	Two asynchronous coupled SMEDs	Independent PWMs (fixed by parameters)
	1		Combined PWM (driven by active SMEDs)
100	0	External control SMED	Independent PWMs (fixed by parameters)
	1		RFU (reserved encoding for future use)
101	0	Asynchronous coupled SMEDs	Independent PWMs (fixed by parameters)
	1		Combined PWM (driven by active SMEDs)
11X	X	RFU	RFU (reserved encoding for future use)

Note: The <n> index represents the SMED number present in the product datasheet.

21.17.2 Chapter assumptions:

The following assumptions have been adopted to simplify the discussion of the next paragraphs:

- **Active:** means that the SMED is not in the HOLD state
- **Inactive:** means that the SMED is in the HOLD state
- **Pair active:** means that one SMED of a pair is not in the HOLD state and both SMEDs of the other pair are in the HOLD state
- **Pair inactive:** means that both SMEDs of a pair are in the HOLD state

Note: In any couple configuration the SMED exiting from the HOLD must be programmed to not re-enter the HOLD states at least after 3 TCK.

21.17.3 SMED subsystem configuration

The following sections describe all possible SMED configuration schemes:

Single SMED configuration

This configuration mode is allowed for all SMEDs. The FSM state evolution and the PWM output signal depend only on the configuration of the selected SMED.

Note: If an SMED is configured in the single mode, the related coupled SMED, if available on product devices, must be programmed in the single MODE.

Synchronous coupled SMEDs configuration

Two SMEDs are coupled through an internal dedicated interface to form a single logic unit (pair) from the application point of view. This configuration requires that the two SMEDs must be programmed with the same source clock. This operating mode doubles the resources of a single SMED allowing the synthesis of a more complex PWM output waveform signal.

When configured in this mode, only one of the two SMEDs is active at the same time, while the other one is inactive (frozen in HOLD state) and vice versa.

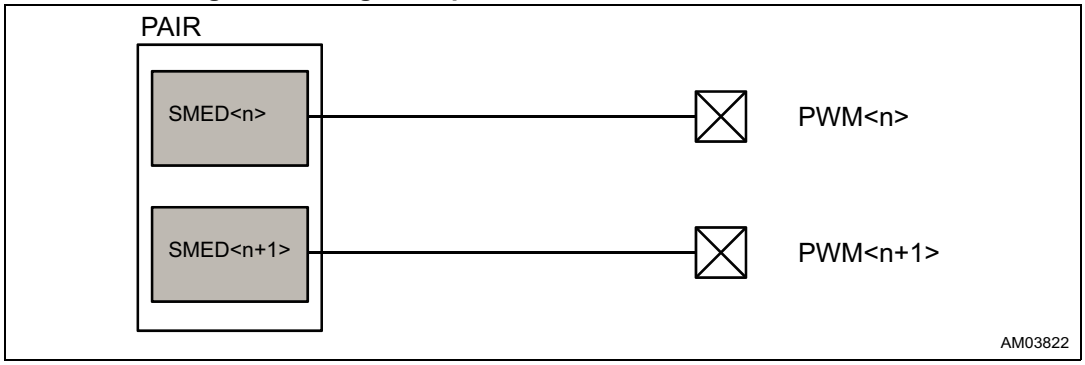
The coupled SMEDs use the HoldJump/HoldExit mechanism to switch the control between them. When one SMED of the pair enters to the HOLD state, the other exits from its HOLD (HOLD_EXIT bit is '1'). Due to signal handshaking, after one SMED of the pair enters the HOLD state, the other will become active at after 2 clock periods.

Figure 44 shows a generic two couple SMEDs configuration scheme where:

SMED<n> is coupled with SMED<n+1>n = 0, 2, 4

The list of the SMED units configurable in the coupled mode is specific for the product device. For details refer to the product datasheet.

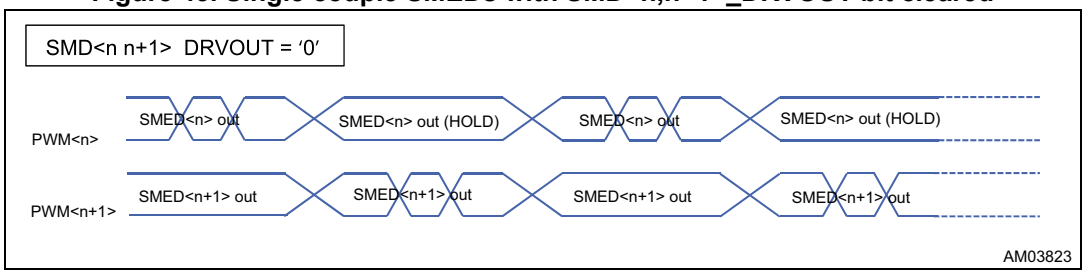
Figure 44. Single couple SMEDs interconnection scheme



The coupled SMEDs may generate two separate PWM output signals or a single more complex PWM signal depending on the value of the SMD<n>_DRVOUT configuration bits.

1. If these register fields are cleared, each SMED generates its own PWM output signal as shown in *Figure 45*.

Figure 45. Single couple SMEDs with SMD<n,n+1>_DRVOUT bit cleared



2. If one of the SMEDs is configured with its SMD<x>_DRVOUT register field set to '1' and is in the active state, it drives its own PWM output signal, else, if it's inactive, hosts

the PWM logical output signal of the other coupled active SMED, as detailed by the next expressions and shown in *Figure 46*:

- If (SMD<n>_DRVOUT = 1 and SMD<n+1>_DRVOUT = 0) then:
  PWM<n> = (Smed<n>_Pwm and [SM_Stat<n> ! = hold]) || (Smed<n+1>_Pwm and [SM_Stat<n+1>! = hold])[k]
- If (SMD<n+1>_DRVOUT = 1 and SMD<n>_DRVOUT = 0) then:
  PWM<n+1> = (Smed<n>_Pwm and [SM_Stat<n> ! = hold]) || (Smed<n+1>_Pwm and[SM_Stat<n+1> ! = hold])[k].

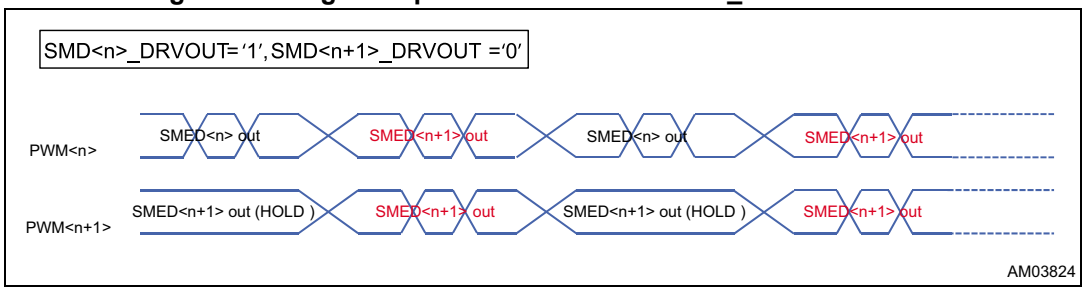
Note: 1. Only one SMD<n,n+1>_DRVOUT bit of the two SMEDs must be active at the same time, selecting which of the two coupled SMEDs physically generates the merged PWM output signal.

2. The unused PWM primary pin may be configured alternatively as GPIO1[x].

3. When using the drive out feature the PWM level of the SMED entering HOLD change, due to the transition, to the level of the SMED currently in HOLD.

Figure 46 shows the behavior of the SMED with the merged PWM output signal.

Figure 46. Single coupled SMED with SMD<n>_DRVOUT bit set



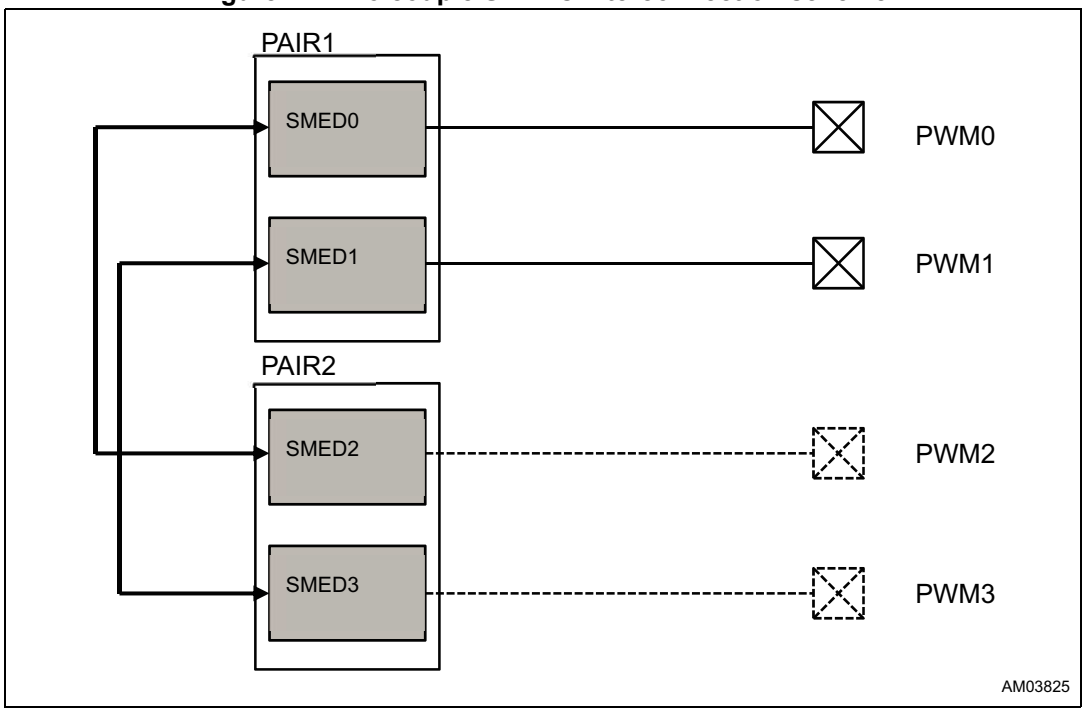
Two synchronous coupled SMEDs configuration

Two pairs of SMEDs are coupled through an internal dedicated interface, to form a single logic unit based on four SMEDs from the application point of view. Synchronous means that all the four SMEDs have to be configured with the same source clock.

The SMEDs pairs used in this configuration mode are SMED<0,1> and SMED<2,3>. The PWM signals are driven by a cross coupled SMED interconnection as shown in *Figure 47*.

k. The PWM<n> signal is driven by the selected SMED when it's not in the hold and the other coupled SMED is in the hold state.

Figure 47. Two couple SMEDs interconnection scheme



Note: 1. PWM0,2 are cross coupled together.

2. PWM1,3 are cross coupled together.

3. Only one of the PWM pairs (PWM[0,2] or PWM[1,3]) are active at a time in this configuration.

In this configuration, only one of the SMED pairs is active at the same time (i.e. inside the pair only one SMED is **active**, the other is in HOLD state), while the other pair is inactive (both SMEDs frozen in HOLD state).

The control switch between the two SMEDs of the same pair is done by using the HoldJump/HoldExit mechanism. Due to signal handshaking, after one SMED of the pair enters the HOLD state, the other will become active at after 2 clock periods

The control switch between the two pairs is done through the SMED Asy_Enable input that has to be programmed in the level sensitive mode, with different polarity for the two pairs (refer to SMD<n>_CTR_INP register description).

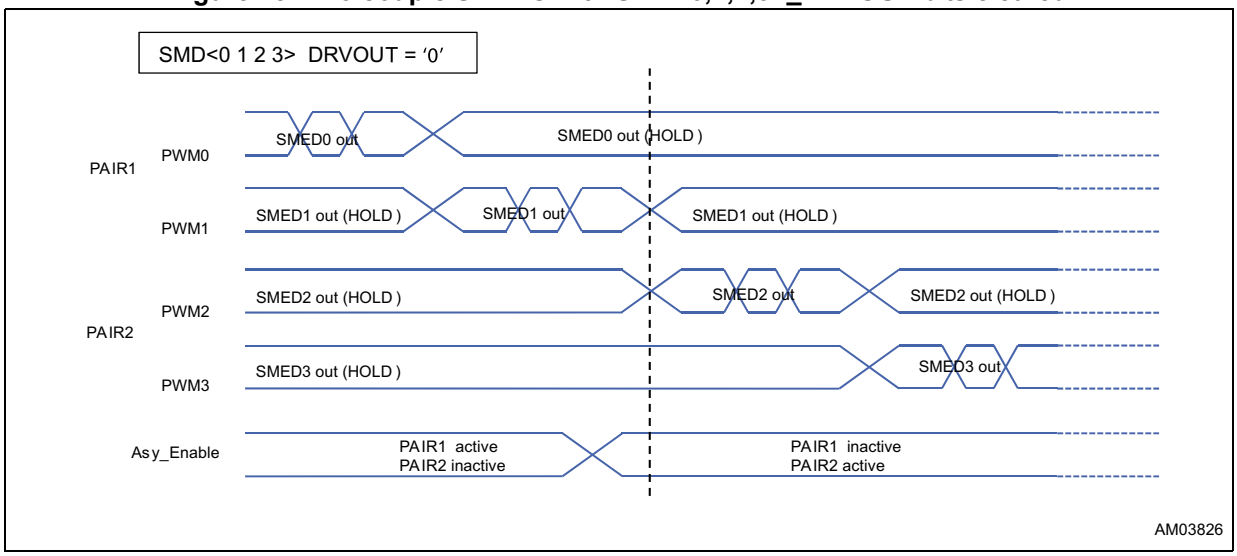
The switch between pairs occur when the Asy_enable input change is detected and the active SMED jumps into the HOLD state (in this timeframe all SMEDs are in HOLD state); then the SMED of the next active pair that becomes active (driving the PWM) is the first that jumped into the HOLD state. At a startup the behavior is defined by the QCOUP_ST bit (see register description).

The Asy_Enable and the InSig[2] input signals are interconnected together to the same CONBOX signal but they have a sensitive capture logic programmable independently via SMD<n>_CTR_INP control registers. As the InSig lines, the Asy_Enable signal is synchronized by a two stage register chain at the input of the SMED. After the Asy_Enable changes the state, the activity switches to the other pair within 3 - 4 clock periods (1 - 2 for synchronization, 2 for signal handshaking).

The two coupled SMEDs may generate four separate PWM output signals or two PWM output signals depending on the value of the SMD<n>_DRVOUT bits.

1. If these register fields are cleared, each SMED generates its own PWM output signal as shown in *Figure 48*.

Figure 48. Two couple SMEDs with SMD<0,1,2,3>_DRVOUT bits cleared



2. If the pair is active and configured with both SMD<x>_DRVOUTs set to '1', it drives its own PWM output signals; else if the pair is inactive, hosts the cross coupled PWM signals of the other SMED active pair as detailed by the next expressions and shown in *Figure 49*:

- 1st cross couple:
  - If (SMD0_DRVOUT = 1 and SMD2_DRVOUT = 0) then:
    PWM0 = (Smed0_Pwm when and [PAIR1 active]) || (Smed2_Pwm and [PAIR2 active])(l)
  - If (SMD2_DRVOUT = 1 and SMD0_DRVOUT = 0) then:
    PWM2 = (Smed0_Pwm and [PAIR1 active]) || (Smed2_Pwm and [PAIR2 active])(l)
- 2nd cross couple:
  - If (SMD1_DRVOUT = 1 and SMD3_DRVOUT = 0) then:
    PWM1 = (Smed1_Pwm and [PAIR1 active]) || (Smed3_Pwm and [PAIR2 active])(l)
  - If (SMD3_DRVOUT = 1 and SMD1_DRVOUT = 0) then:
    PWM3 = (Smed1_Pwm and [PAIR1 active]) || (Smed3_Pwm and [PAIR2 active])(l)

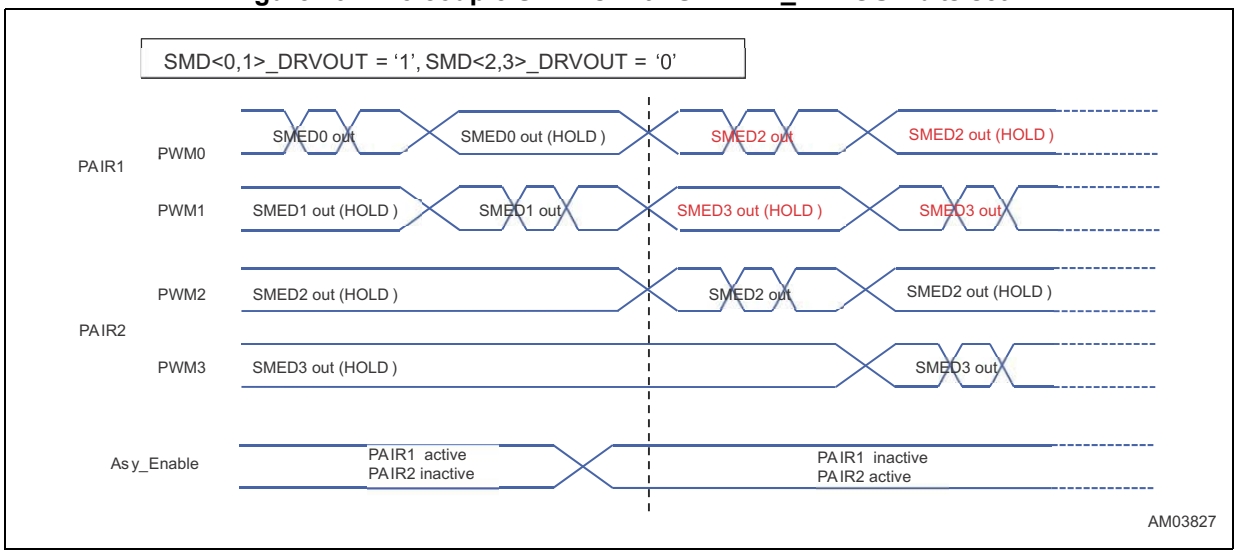
Note: 1. Only the SMED pair responsible to drive the PWM signal has to be configured with both SMD<x>_DRVOUT set at '1'. The other pair must have these register fields cleared.

2. The SMED pairs are interlocked each other avoiding the contemporary timing pulse generation on the two active PWM signals.

3. The unused PWM primary pins may be configured alternatively as the GPIO1[x].

l. The meaning of 'PAIR1 active' and 'PAIR2 active' is defined in *Section 21.17.2: Chapter assumptions: on page 190*.

Figure 49. Two couple SMEDs with SMD<x>_DRVOUT bits set



Two asynchronous coupled SMEDs configuration

This configuration is similar to the two synchronous coupled SMEDs with the only difference that the first pair and the second pair may have different source clocks; while inside each pair the source clock must be the same.

The hold signals exchanged between the two SMED pairs are synchronized by two register stages. After the Asy_Enable changes the state, the activity switches to the other pair within the 1 - 2 clocks of the lowest frequency pair periods (for synchronization) plus 2 clocks of the destination pair (for signal handshaking).

External control SMEDs configuration

In this configuration the SMED units are arranged in the single SMED operating mode and the SMED<m> Tmr(S) event normally driven by timer comparison logic, is now sourced by the SMED<n> NextHold signal, where the SMED number <n> is got from *Table 35* for any state.

Table 35. Tmr(S) NextHold event assignment

SMED<m>	S0	S1	S2	S3
	SMED<n> NextHold	SMED<n> NextHold	SMED<n> NextHold	SMED<n> NextHold
0	2	3	4	5
1	2	3	4	5
2	0	1	4	5
3	0	1	4	5
4	0	1	2	3
5	0	1	2	3

Note that the NextHold is asserted one clock in advance before the HOLD state.

All SMEDs may be configured in the external control mode.

Note: 1. When the external controlled mode is selected, all the involved SMEDs must be configured synchronous with the same source clock.

2. If an SMED is configured in the external controlled mode and some other SMEDs present in the product device are configured in such a way that doesn't provide their NextHold event, the SMED FSM evolution is based only on the Event_Seq(I) and Event_NoSeq(I) events for some states.

Asynchronous coupled SMEDs configuration

This configuration is similar to the synchronous coupled SMEDs with the only difference that the two SMEDs may have different source clocks. Control signals between the two SMEDs are synchronized by two register stages. The Asy_Enable signal has to be configured in this case as a 'rising edge' and enabled through the SMD<n>_CTR_INP and SMD<n>_ISEL registers, in order to allow the synchronization (via a two-stage chain) of the HOLD control signals.

After one SMED of the pair enters the HOLD state, the other will become active after 3 - 4 of its clock period (1 - 2 for synchronization and 2 for signal handshaking).

This configuration is applicable only for the next SMED pair:

- SMED4 with SMED5

21.17.4 Unlock feature

In some applications, it may be necessary to have a startup phase where two PWM output signals are driven alternatively at each cycle, followed by an operative phase where the two PWM outputs are driven alternatively each one for a configurable period. To support this behavior the FSM implements the unlock feature.

The unlock feature is a special extension of the SMED behavior, applicable only to synchronous coupled SMEDs, that switches dynamically between two different handling behavior of the HoldJump/HoldExit mechanism. This feature cannot be reversed; i.e.: it's not possible restore the synchronous coupled mode for an SMED pair that has been unlocked. Once the SMED pair has been unlocked, the corresponding coupled SMEDs start to operate in the single mode.

The selection of different operation is handled automatically by the SMED expansion interface logic that switches the two operating modes without undesired effects on the output signals.

The usage of the unlock feature requires that SW application executes the following step sequences (SMED<n> and SMED<n+1> are considered):

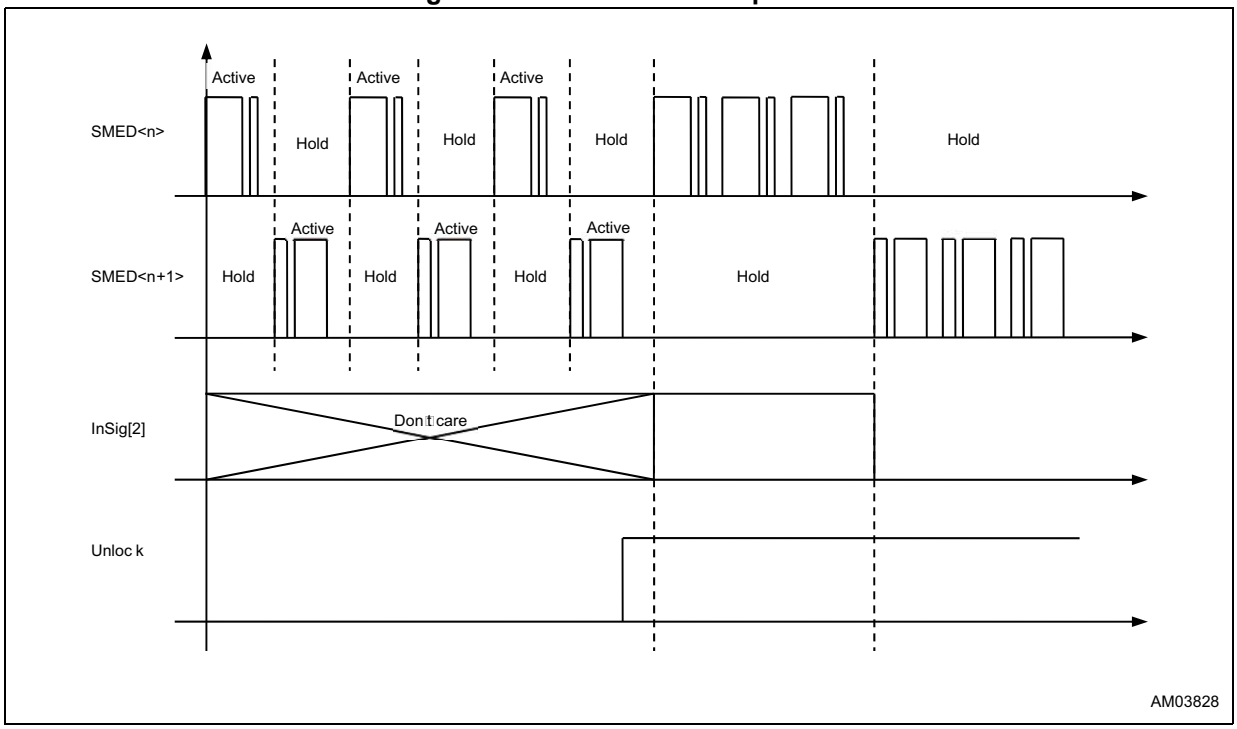
1. Configure the two SMEDs in the synchronous coupled mode for the startup phase. In this configuration the InSig[2] event must be configured to enter the HOLD state on the high level for the SMED<n> and to enter the HOLD state on the low level for the SMED<n+1> (or vice versa: this depends on the hardwired values and has to be considered as an example; see the step 2). The HOLD_EXIT bit of the state causing the transition to the HOLD state has to be set to '1'.
2. Enable the unlock register setting the Use_Unlock bit. This temporarily forces the status of the InSig[2] signal internally to 0 for the SMED<n> and to 1 for the SMED<n+1> independently from the status of the physical lines (the levels of this

unlock-hardwired InSig[2] signal for a pair of the SMEDs can be reversed in the implementation but they are always opposite; check the product datasheet for details).

3. Enable the SMED FSM and counter to proceed with the startup phase. In this phase the control is switched between the two SMEDs depending on the programmed configuration of the Event_NoSeq(I), Event_Seq(I) and Tmr(S) events.
4. To enter the operative phase, set the unlock bit. At the next control switch after the unlock bit is set, the SMED behavior is changed in that the control is switched between the two SMEDs only by the InSig[2] event.

An example of the unlock sequence is shown in *Figure 50*.

Figure 50. SMED unlock sequence



21.17.5 Connection box

The connection box is a switch matrix implemented in order to extend the connectivity of any input coming from external/internal input lines to each input signals of the SMEDs.

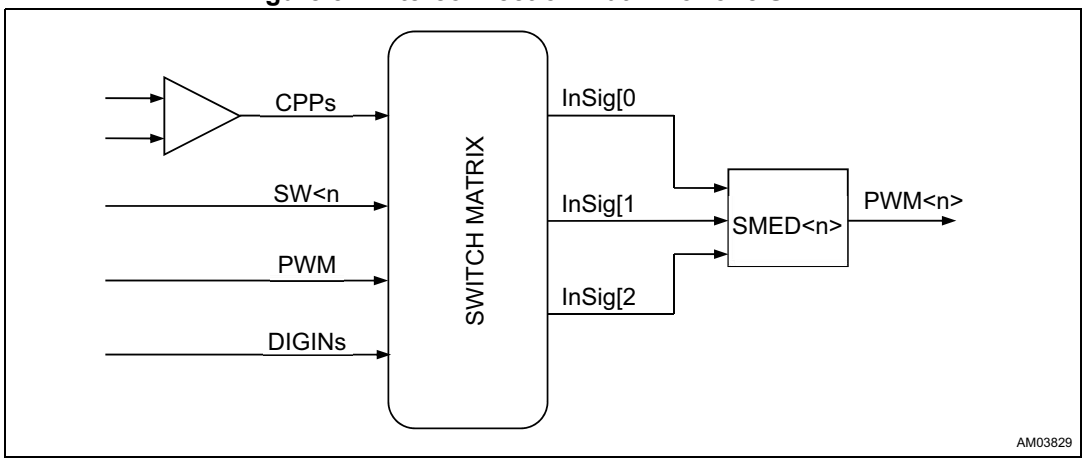
With this logic it is possible to configure more than one driving signal for each one of the InSig[2:0] lines, expanding the FSM capabilities and giving to the application programmer a more flexible solution for the SMED configuration.

The connection box allows multiplexing up to four different input signals for each one of the InSig lines of each SMED. The same input signal is available for multiplexing on more than one InSig line, giving to the application programmer a wide possibility of the triggering event selection.

To simplify the configuration of the multiplexer, each SMED has a corresponding connection box configuration register. Through the configuration registers it is possible to select one out of four input signals for each of the three InSig[2:0] lines.

- The input signals are chosen from four group's families (refer to *Figure 51*):
- DIGINs fast digital input
- CPPs analog comparators
- PWMs logical output signal coming from other SMEDs
- SW<n> software event

Figure 51. Interconnection matrix for one SMED



Switch matrix interconnection

The connection between input signals and SMED InSig lines is based on a programmable associative scheme. A simplified connection matrix has been implemented in order to reduce the amount of logic.

Every SMED unit has three con-box selection lines, one for each input, configurable with the MSC_CBOXS<n> registers (refer to *MSC_CBOXS<n> (connection box selection SMED<n>)* in *Section 21.18.2: SMED environment configuration registers on page 215*). The selection lines choose the interconnection between one of the possible four con-box signals and the SMD<n> input event InSig[<y>]); the general scheme for the connection configuration for an SMED is shown in *Table 36*.

Table 36. Connection box interconnection matrix

Conb_s<n>_<y>					
SMED<n>	**Matrix selection**				
y	**Signal**	**00**	**01**	**10**	**11**
0	InSig[0]	Input<0><0>	Input<0><1>	Input<0><2>	Input<0><3>
1	InSig[1]	Input<1><0>	Input<1><1>	Input<1><2>	Input<1><3>
2	InSig[2]	Input<2><0>	Input<2><1>	Input<2><2>	Input<2><3>

Note: The Asy_Enable and the InSig[2] input signals may be interconnected together at the same CONBOX output signal.

SW events

To increase the flexibility of configuration of the SMEDs FSM, the MSC_SMSWEV register provides up to eight SW event flags.

These SW flags are usually interconnected to the connection box matrix and may be selected as InSig signals for each SMEDs FSM. This gives the application program the capability to modify the SMED finite state machine evolution like any other HW events interconnected to the InSig lines.

21.17.6 FSM diagnostic trace

The MSC_IOMXSMD register offers the capability to trace the SMED FSM variable state during the debug phase of an application program.

By configuring the MSC_IOMXSMD register, it is possible to select the state variable of one of the six SMEDs to be multiplexed on three primary P0 output lines. Refer to *MSC_IOMXSMD (SMED I/O MUX control register)* in *Section 21.18.2: SMED environment configuration registers on page 215* for details about the register configuration.

Note: 1. Selecting the FSM variable state lines as output on the P0 overrides the MSC_IOMXP0 configuration.

2. The trace signals evolution rate cannot exceed the maximum pad toggle frequency (for further information refer to the product datasheet).

21.18 SMED registers description

The SMEDs are accessible to the application programmer through a set of dedicated registers located in the microcontroller registers area. The physical address of this area is product specific.

- The first paragraph contains a detailed description of the SMED core registers used to configure each SMED units.
- The second paragraph describes the SMED environment configuration registers used to configure the SMED subsystem (e.g.: connection box, clock source selection, etc.). These registers allow configuring the SMED external interface, managing the unlock procedure and defining the external connections through the connection box matrix.

21.18.1 SMED core registers

SMED core registers base address

The set of registers for the SMED configuration is replicated at modular addresses in the microcontroller registers space for each one on the instantiated SMEDs.

The base address of each set of registers is obtained from the following formula:

Equation 23

$$\text{SMED<n>\_base\_address} = \text{SMEDs\_physical\_base} + (0x40 * n)$$

where <*n*> is the SMEDs instance number from <0> to <n>, and *SMEDs_physical_base* is the physical address of the first register of the first SMED instantiated. Refer to the specific product datasheet to obtain this address.

In the following registers description for each control register it is indicated the hexadecimal offset of the register with respect to the **SMED<n>_base_address**.

SMD<n>_CTR (control register)

Offset: 0x00

Default value:0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	RFU	RFU	FSM_ENA	START_CNT
r	r	r	r	r	r	r/w	r/w

Bit 0: **START_CNT** start counter (R/W).

When '0' the counter evolution is stopped and put in reset condition, anyway the FSM is still active and can react to the external events if these have been previously enabled.

When '1' and the FSM_ENA bit is set to '1' starts the counter activity.

Note: Initial transition from the IDLE state is handled as all other state transitions with the only exception that the Tmr(S) event is immediately generated when the START_CNT bit is set; all the other actions are controlled by the SMD<n>_PRM_ID0 and SMD<n>_PRM_ID1 register contents.

Bit 1: **FSM_ENA** synchronous FSM enable (R/W).

When '1' enables the FSM to run; otherwise forces the FSM controller in the IDLE state synchronously.

Bit 7-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

SMD<n>_CTR_TMR (control time register)

Offset: 0x01

Default value:0x00

7 6 5	4	3	2	1	0
RFU	DITHER_VAL	TIME_T3_VAL	TIME_T2_VAL	TIME_T1_VAL	TIME_T0_VAL
r	r/w	r/w	r/w	r/w	r/w

Bit 0: **TIME_T0_VAL** validation of the SMD<n>_TMR_T0 register.

When written to '1' validates SMD<n>_TMR_T0 register content enabling shadow register updating.

This bit is cleared by hardware when the corresponding shadow register is updated.

Bit 1: **TIME_T1_VAL** validation of the SMD<n>_TMR_T1 register.

When written to '1' validates SMD<n>_TMR_T1 register content enabling shadow register updating.

This bit is cleared by hardware when the corresponding shadow register is updated.

Bit 2: **TIME_T2_VAL** validation of the SMD<n>_TMR_T2 register.

When written to '1' validates SMD<n>_TMR_T2 register content enabling shadow register updating.

This bit is cleared by hardware when the corresponding shadow register is updated.

Bit 3: **TIME_T3_VAL** validation of the SMD<n>_TMR_T3 register.

When written to '1' validates SMD<n>_TMR_T3 register content enabling shadow register updating.

This bit is cleared by hardware when the corresponding shadow register is updated.

Bit 4: **DITHER_VAL** validation of the SMD<n>_CTR_DTHR register.

When written to '1' validates SMD<n>_CTR_DTHR register content enabling shadow register updating.

This bit is cleared by hardware when the corresponding shadow register is updated.

Bit 7-5: **RFU** reserved; must be kept 0 during register writing for future compatibility.

SMD<n>_CTR_INP (control input register)

Offset: 0x02

Default value:0x00

7	6	5	4	3	2	1	0
EL_EN	EL_INSIG[2]	EL_INSIG[1]	EL_INSIG[0]	RAIS_EN	RS_INSIG[2]	RS_INSIG[1]	RS_INSIG[0]
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 0: **RS_INSIG[0]** external event (0) input polarity level definition (R/W).

This field configures the InSig[0] input signal active mode:

- Edge mode:
  0: input active on the falling edge.
  1: input active on the rising edge.
- Level mode:
  0: input active on the low level.
  1: input active on the high level.

Bit 1: **RS_INSIG[1]** external event (1) input polarity level definition (R/W).

This field configures the InSig [1] input signal active mode:

- Edge mode:
  0: input active on the falling edge.
  1: input active on the rising edge.
- Level mode:
  0: input active on the low level.
  1: input active on the high level.

Bit 2: **RS_INSIG[2]** external event (2) Input polarity level definition (R/W).

This field configures the InSig [2] input signal active mode:

- Edge mode:
  0: input active on the falling edge.
  1: input active on the rising edge.
- Level mode:
  0: input active on the low level.
  1: input active on the high level.

Bit 3: **RAIS_EN** Asy_Enable input polarity level definition (R/W).

This field configures the Asy_Enable input signal active mode:

- Edge mode:
  0: input active on the falling edge.
  1: input active on the rising edge.
- Level mode:
  0: input active on the low level.
  1: input active on the high level.

Bit 4: **EL_INSIG[0]** external InSig[0] input characteristic definition (R/W).
0: input signal configured in the edge mode.
1: input signal configured in the level mode.

Bit 5: **EL_INSIG[1]** external InSig[1] input characteristic definition (R/W).
0: input signal configured in the edge mode.
1: input signal configured in the level mode.

Bit 6: **EL_INSIG[2]** external InSig[2] input characteristic definition (R/W).
0: input signal configured in the edge mode.
1: input signal configured in the level mode.

Bit 7: **EL_EN** Asy_Enable input characteristic definition (R/W).
0: input signal configured in the edge mode.
1: input signal configured in the level mode.

SMD<n>_CTR_DTHR (dithering register)

Offset: 0x03

Default value:0x00

7	6	5	4	3	2	1	0
DITH [7:0]							
r/w							

Bit 7-0: **DITH[7:0]** dithering register (R/W).

SMD<n>_TMR_T<i>L (time T<i> lsb register)

Offset: 0x04+<i*2>

Default value:0x00

7	6	5	4	3	2	1	0
TIM_T<i>L [7:0]							
r/w							

The index <i> ranges from 0 to 3.

Bit 7-0: **TIM_T<i>L[7:0]** time constant LSB register (R/W).
Lower byte of time constant TIM_T<i>(1).

SMD<n>_TMR_T<i>H (time T<i> msb register)

Offset: 0x05+<i*2>

Default value:0x00

7	6	5	4	3	2	1	0
TIM_T<i>H [7:0]							
r/w							

The index <i> ranges from 0 to 3.

Bit 7-0: **TIM_T<i>H[7:0]** time constant MSB register (R/W).

Upper byte of time constant TIM_T<i>(1);

Note: The TIM_T<i>x (TIM_T<i>H and TIM_T<i>L) value 0x0000 is reserved for future use.

SMD<n>_PRM_ID0 (parameter0 register)

Offset: 0x0C

Default value:0x00

7	6	5	4	3	2	1	0
AND_OR	HOLD_JMP	PULS_EDG	CNT_RSTE	EDGE[1:0]		NX_STAT[1:0]	
r/w	r/w	r/w	r/w	r/w		r/w	

Refer to *SMD<n>_PRM_<st>0 (parameter0 register)* register parameter descriptions.

SMD<n>_PRM_ID1 (parameter1 register)

Offset: 0x0D

Default value:0x00

7	6	5	4	3	2	1	0
RFU	HOLD_EXIT	PULS_CMP	CNT_RSTC	CEDGE[1:0]		RFU	RFU
r	r/w	r/w	r/w	r/w		r	r

Refer to *SMD<n>_PRM_<st>1 (parameter1 register)* descriptions.

SMD<n>_PRM_ID2 (parameter2 register)

Offset: 0x0E

Default value:0x00

7	6	5	4	3	2	1	0
QCOUP_ST	RFU	RFU	RFU	RFU	RFU	RFU	LATCH_RS
r/w	r	r	r	r	r	r	r/w

Bit 0: **LATCH_RS** latch reset definition (R/W).

0: no action is done.

1: the latch of InSig[0] capture will be reset when the FSM enters the HOLD state.

Bit 6-1: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 7: **QCOUP_ST** quad couple start mode configuration.

This field is applicable only in quad couple SMED configuration; it is ignored in other modes (refer to the product datasheet to see if this configuration is supported)

0: selects the standard start in the two coupled mode; the SMED masks the PWM output (keeping it at the IDLE value) until it exits from the HOLD state for the first time.

1: selects the fast start in the two coupled mode; the SMED starts to drive the PWM output immediately on exiting from the IDLE state.

Note: 1. If the QCOUP_ST field is cleared, the basic flow used to start the two couple of SMEDs is to program within each couple one SMED active and the other one to enter the HOLD state from IDLE; then select the active couple with the Asy_Enable signal value, and activate the four SMEDs through the SMD<n>_CTR[1:0] register fields. The first SMED which drives its PWM output is the first one of the active coupled that exits from the HOLD state.

2. If the QCOUP_ST is set, the user has to program only one SMED active at the start time, while the other three have to be programmed to enter the HOLD state from IDLE. The Asy_Enable signal must be programmed accordingly, in order to select the SMED couple of the first active SMED as active. Finally the user can activate the four SMEDs through the SMD<n>_CTR[1:0] register fields with the advice to insert a delay of at least $2t_{SMED}$ between the starts of the two SMEDs initially in the HOLD state (i.e.: those of the couple inactive).

SMD<n>_PRM_<st>0 (parameter0 register)

Offset: 0x0F+os(<st>)*3

Default value:0x00

7	6	5	4	3 2	1 0
AND_OR	HOLD_JMP	PULS_EDG	CNT_RSTE	EDGE[1:0]	NX_STAT[1:0]
r/w	r/w	r/w	r/w	r/w	r/w

The index <st> may assume the values S0, S1, S2, S3 corresponding respectively to the states S0, S1, S2 and S3. The address offset is defined as: os(S0) = 0 … os(S3) = 3.

Note: For the IDLE state the Tmr(S) event is to all intents and purposes substituted by the activation of the FSM timer (refer to SMD<n>_CTR (control register) in *Section 21.18.1: SMED core registers on page 199**).*

Bit 1-0: **NX_STAT[1:0]** next event controlled state definition (R/W).

It defines the next state to jump in case of the Event_NoSeq(I) jumping cause or the Tmr(S)/Event_Seq(I) when “AND” function is selected; the coding is shown below:

“00” = S1

“01” = S2

“10” = S3

“11” = S0

Bit 3-2: **EDGE[1:0]** external edge jumping selection (R/W).

The InSig input signal used to jump out the state towards any state S<x> x = 0, 1, 2, 3 as Event_NoSeq(I) transition; the coding is shown below:

“00” = no InSig used

“01” = inSig[0] used

“10” = inSig[1] used

“11” = inSig[2] used

Bit 4: **CNT_RSTE** counter reset definition (R/W).

0: counter reset disable.

1: the counter will be reset in case of the Event_NoSeq(I) jumping cause.

Bit 5: **PULS_EDG** PWM pulse out level definition (R/W).

0: PWM pulse output will be driven low in case of the Event_NoSeq(I) jumping cause.

1: PWM pulse output will be driven high in case of the Event_NoSeq(I) jumping cause.

Bit 6: **HOLD_JMP** hold state definition (R/W).

This bit is used in conjunction with the AND_OR bit to determine SMED transition behavior.

When '0': the next state is determined as discussed under the AND_OR bit description.

When '1': in case the jump to the next state occurs in one of the following conditions:

– The jump is caused by an Event_NoSeq(I)

– A coupled SMED configuration is used for this SMED

– The bit AND_OR of this register is set

then the next state will be the HOLD state and the state following the HOLD state is specified by NX_STAT[1:0] bits.

otherwise: like for '0' setting.

Bit 7: **AND_OR** jumping Boolean function definition (R/W).

When '0' there are two chances to jump out of the state (if the HOLD_JMP bit is set, see the exceptions listed in the description of that bit):

1. The Event_Seq(I) or Tmr(S) event, whichever comes first, to jump into the next sequential state.

2. The Event_NoSeq(I) event to jump into the state selected by NX_STAT[1:0] bits of this register. If EDGE[1:0] selection of this register is “00”, this type of transition is disabled.

When '1' both Event_NoSeq(I) and either Tmr(S) or Event_Seq(I) conditions must be true to jump into the state selected by NX_STAT[1:0] bits of this register or into the HOLD state if the HOLD_JMP bit is set. If EDGE[1:0] selection of this register is “00”, the Event_NoSeq(I) condition is considered always true.

SMD<n>_PRM_<st>1 (parameter1 register)

Offset: 0x10+os(<st>)*3

Default value: 0x00

7	6	5	4	3 2	1	0
RFU	HOLD_EXIT	PULS_CMP	CNT_RSTC	CEDGE[1:0]	RFU	RFU
r	r/w	r/w	r/w	r/w	r	r

The index <st> may assume the values S0, S1, S2, S3 corresponding respectively to the states S0, S1, S2 and S3. The address offset is defined as: os(S0) = 0 … os(S3) = 3.

Note: For the IDLE state the Tmr(S) event is to all intents and purposes substituted by the activation of the FSM timer (refer to SMD<n>_CTR (control register) in *Section 21.18.1: SMED core registers on page 199).*

Bit 1-0: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 3-2: **CEDGE[1:0]** external edge jumping selection (R/W).

The edge input signal used to force jumping out the state sequentially as the Tmr(S) is called Event_Seq(I); the coding is shown below:

“00” = no InSig used

“01” = InSig[0] used

“10” = InSig[1] used

“11” = InSig[2] used

Bit 4: **CNT_RSTC** counter reset definition (R/W).

0: counter reset disable.

1: the counter will be reset in case of the Tmr(S) or Event_Seq(I) jumping cause.

Bit 5: **PULS_CMP** PWM pulse out level definition (R/W).

0: PWM pulse output driven low in case of the Tmr(S) or Event_Seq(I) jumping cause.

1: PWM pulse output driven high in case of the Tmr(S) or Event_Seq(I) jumping cause.

Bit 6: **HOLD_EXIT** HOLD state exit cause definition (R/W).

1: the SMED exits from the HOLD state when the corresponding coupled/paired SMED enters the HOLD state (refer to *Section 21.10.4: Hold exit on page 181*).

0: the SMED exit from the HOLD state depends on the “exit InSig” programming mode (including SMED coupled configuration). In all cases the EDGE[1:0] field relative to the state preceding the HOLD state has to be different from “00” to allow the SMED exit from HOLD state. Refer to *Section 21.10.4: Hold exit*.

– If the level mode is used, the SMED exits the HOLD state when the “exit InSig” becomes false.

– If the edge mode is used, the SMED exits the HOLD when a second edge of the same polarity on the “exit InSig” is detected.

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

SMD<n>_PRM_<st>2 (parameter2 register)

Offset: 0x11+os(<st>)*3

Default value:0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	RFU	RFU	RFU	LATCH_RS
r	r	r	r	r	r	r	r/w

The index <st> may assume the values S0, S1, S2, S3 corresponding respectively to the states S0, S1, S2 and S3. The address offset is defined as: os(S0) = 0 … os(S3) = 3.

Bit 0: **LATCH_RS** latch reset definition (R/W).

When '1' the latch on InSig[0] will be reset when entering the <st> state.

Bit 7-1: **RFU** reserved; must be kept 0 during register writing for future compatibility.

SMD<n>_CFG (timer configuration register)

Offset: 0x1B

Default value:0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	TIM_UPD [1:0]		TIM_NUM [1:0]	
r	r	r	r	r/w		r/w	

Bit 1-0: **TIM_NUM[1:0]** time registers to be temporary incremented selection (R/W).

Time register to be temporary incremented by one; values are:

When “00” => SMD<N>_TMR_T3

When “01” => SMD<N>_TMR_T0

When “10” => SMD<N>_TMR_T1

When “11” => SMD<N>_TMR_T2

Bit 3-2: **TIM_UPD[1:0]** time registers update mode (R/W).

While in the HOLD state: instantly (= at the next SMED clock pulse)

While not in the HOLD state:

“00”: instantly (= at the next SMED clock pulse)

“01”: on the rising edge of the PWM output

“10”: on the falling edge of the PWM output

“11”: on counter reset

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

SMD<n>_DMP_L (dump counter LSB register)

Offset: 0x1C

Default value:0x00

7	6	5	4	3	2	1	0
CNT_LO [7:0]							
r							

Bit 7-0:**CNT_LO[7:0]** counter dump LSB value (read only).

Counter lower byte value, frozen when the selected external event (see SMD<n>_DMP bit [2:0]) occurs. The content is related to the first event if the SMD<n>_DMP bit [3] is low, otherwise is the last frozen.

SMD<n>_DMP_H (dump counter MSB register)

Offset: 0x1D

Default value:0x00

7	6	5	4	3	2	1	0
CNT_HI [7:0]							
r							

Bit 7-0:**CNT_HI[7:0]** counter dump MSB value (read only).

Counter higher byte value, frozen when the selected external event (see SMD<n>_DMP bit [2:0]) occurs. The content is related to the first event if the SMD<n>_DMP bit [3] is low, otherwise is the last frozen.

SMD<n>_GSTS (general status register)

Offset: 0x1E

Default value:0x00

7	6	5 4	3	2	1	0
RFU	EVENT_OV	DMP_LK [1:0]	CNT_FLAG	EX2_DUMP	EX1_DUMP	EX0_DUMP
r	r	r	r_w1	r_w1	r_w1	r_w1

Bit 0: **EX0_DUMP** dumping cause flag (R/W1)[(m)]

0: no pending dump.

1: the dump flag set due to the FSM transition state triggered by the InSig[0] capture condition; the dumped counter value is stored into the SMD<n>_DMP_L/H registers.

m. SMD<n>_GSTS bits are read/write-clear; when the register bits are written, the data are used as bit-clear; so writing '1' the corresponding bit is cleared, writing '0' the bit doesn't change; in reading the register contents is preserved.

Bit 1: **EX1_DUMP** dumping cause flag (R/W1)[m], [n]

0: no pending dump.

1: the dump flag set due to the FSM transition state triggered by the InSig[1] capture condition; the dumped counter value is stored into the SMD<n>_DMP_L/H registers.

Bit 2: **EX2_DUMP** dumping cause flag (R/W1)[m], [n]

0: no pending dump.

1: the dump flag set due to the FSM transition state triggered by the InSig[2] capture condition; the dumped counter value is stored into the SMD<n>_DMP_L/H registers.

Bit 3: **CNT_FLAG** counter reset flag when dump is enabled (R/W1)[m]

0: no pending dump.

1: when at least one SMD<n>_DMP[2:0] bit is active, the SMED internal counter is reset by the FSM or by the counter overflow.

Bit 5-4: **DMP_LK [1:0]** counter dump status (read only)

The counter dump register locked, values are:

“00”: unlocked (free)

“01”: locked, means SMD<n>_DMP_L just read

“10”: locked, means SMD<n>_DMP_H just read

When the SMD<n>_DMP[3] bit is low the dump registers never locks; otherwise when the SMD<n>_DMP[3] bit is high the 1st read access to either the SMD<n>_DMP_L or SMD<n>_DMP_H locks both these registers. To unlock them it is necessary to read the register not yet read.

Bit 6: **EVENT_OV** event overflow flag (read only)[o]

0: no dump overflow event detected.

1: this field is set when an external event requires the updating of SMD<n>_DMP_L/H registers but these resources are already busy for a previous event which generated a counter dump. The SMED is not able to manage two consecutive counters dump values when the SMD<n>_DMP[3] is low; if the DUMP_EVER bit is selected (SMD<n>_DMP[3]='1') the EVENT_OV bit is never set. The SMD<n>GSTS[2:0] bit fields show the input event that froze the counter dump register.

Bit 7: **RFU** reserved; must be kept 0 during register writing for future compatibility.

n. If the SMD<n>_DMP[4] bit is set, the SMD<n>GSTS[2:0] bit fields are also reset by clearing the corresponding bit field of the SMD<n>_IRQ[3:1] register.

o. The EVENT_OV bit is reset each time the system detects a write clear access of any of the SMD<n>_GSTS bit [2:0] (or also when clearing any of the SMD<n>_IRQ[3:1] bit fields if the SMD<n>_DMP bit 4 is set).

SMD<n>_IRQ (interrupt request register)

Offset: 0x1F

Default value:0x00

7	6	5	4	3	2	1	0
STA_S3_IT	STA_S2_IT	STA_S1_IT	STA_S0_IT	EXT2_INT	EXT1_INT	EXT0_INT	CNT_OVER
r_w1	r_w1	r_w1	r_w1	r_w1	r_w1	r_w11	r_w1

Bit 0: **CNT_OVER** interrupt cause flag (R/W1)(p)

0: interrupt request not asserted.

1: counter overflow interrupt request.

Bit 1: **EXT0_INT** interrupt cause flag (R/W1)(p), (q)

0: interrupt request not asserted.

1: interrupt request asserted for a transition occurred on InSig[0] capture condition.

Bit 2: **EXT1_INT** interrupt cause flag (R/W1)(p), (q).

0: interrupt request not asserted.

1: interrupt request asserted for a transition occurred on InSig[1] capture condition.

Bit 3: **EXT2_INT** interrupt cause flag (R/W1)(p), (q).

0: interrupt request not asserted.

1: interrupt request asserted for a transition occurred on InSig[2] capture condition.

Bit 4: **STA_S0_IT** interrupt cause flag (R/W1)(p).

0: interrupt request not asserted.

1: interrupt request asserted for a Tmr(0) transition.

Bit 5: **STA_S1_IT** interrupt cause flag (R/W1)(p).

0: interrupt request not asserted.

1: interrupt request asserted for a Tmr(1) transition.

Bit 6: **STA_S2_IT** interrupt cause flag (R/W1)(p).

0: interrupt request not asserted.

1: interrupt request asserted for a Tmr(2) transition.

Bit 7: **STA_S3_IT** Interrupt cause flag (R/W1)(p).

0: interrupt request not asserted.

1: interrupt request asserted for a Tmr(3) transition.

p. SMD<n>_IRQ bits are read/write-clear; when the register bits are written, the data are used as bit-clear; so writing '1' the corresponding bit is cleared, writing '0' the bit doesn't change; in reading the register contents is preserved.

q. If the SMD<n>_DMP[4] bit is set, the SMD<n>IRQ[3:1] bit fields are also reset by clearing the corresponding bit field of the SMD<n>_GSTS[2:0] register.

SMD<n>_IER (interrupt enable register)

Offset: 0x20

Default value:0x00

7	6	5	4	3	2	1	0
IT_STA_S3	IT_STA_S2	IT_STA_S1	IT_STA_S0	IT_EN_EX2	IT_EN_EX1	IT_EN_EX0	CNT_OV_E
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 0: **CNT_OV_E** mask bit for CNT_OVER flag (R/W)

0: interrupt disable request.

1: internal counter overflow interrupt enabled.

Bit 1: **IT_EN_EX0** mask bit for EXT0_INT flag (R/W)

0: interrupt disable request.

1: interrupt enable for a transition occurred on InSig[0] capture condition.

Bit 2: **IT_EN_EX1** mask bit for EXT1_INT flag (R/W)

0: interrupt disable request.

1: interrupt enable for a transition occurred on InSig[1] capture condition.

Bit 3: **IT_EN_EX2** mask bit for EXT2_INT flag (R/W)

0: interrupt disable request.

1: interrupt enable for a transition occurred on InSig[2] capture condition.

Bit 4: **IT_STA_S0** mask bit for STA_S0_IT flag (R/W)

0: interrupt disable request.

1: interrupt enable for a Tmr(0) transition.

Bit 5: **IT_STA_S1** mask bit for STA_S1_IT flag (R/W)

0: interrupt disable request.

1: interrupt enable for a Tmr(1) transition.

Bit 6: **IT_STA_S2** mask bit for STA_S2_IT flag (R/W)

0: interrupt disable request.

1: interrupt enable for a Tmr(2) transition.

Bit 7: **IT_STA_S3** mask bit for STA_S3_IT flag (R/W)

0: interrupt disable request.

1: interrupt enable for a Tmr(3) transition.

SMD<n>_ISEL (external event control register)

Offset: 0x21

Default value:0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	INPUT_LAT	INPUT2_EN	INPUT1_EN	INPUT0_EN
r	r	r	r	r/w	r/w	r/w	r/w

Bit 0: **INPUT0_EN** InSig[0] input enable (R/W)

0: input signal disable (unused and masked).

1: input signal enable to interact with the SMED internal logic.

Bit 1: **INPUT1_EN** InSig[1] input enable (R/W)

0: input signal disable (unused and masked).

1: input signal enable to interact with the SMED internal logic.

Bit 2: **INPUT2_EN** InSig[2] input enable (R/W)

0: input signal disable (unused and masked).

1: input signal enable to interact with the SMED internal logic.

Bit 3: **INPUT_LAT** enable latch function on InSig[0] input (R/W)

0: disable input signal latch functionality.

1: a latch is added to the InSig[0] input. When this latch is added, and an InSig[0] occurs, it remains stored until the FSM resets it; this feature is useful to maintain an event memorized until it can be managed later on by SW.

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility

SMD<n>_DMP (dump enable register)

Offset: 0x22

Default value:0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	CPL_IT_GE	DMP_EVER	DMPE_EX2	DMPE_EX1	DMPE_EX0
r	r	r	r/w	r/w	r/w	r/w	r/w

Bit 0: **DMPE_EX0** dump update mode (R/W)

0: dump register disable.

1: the dump register update is triggered when a state transition occurs due to the InSig[0] input event.

Bit 1: **DMPE_EX1** dump update mode (R/W)

0: dump register disable.

1: the dump register update is triggered when a state transition occurs due to the InSig[1] input event.

Bit 2: **DMPE_EX2** dump update mode (R/W)

0: dump register disable.

1: the dump register update is triggered when a state transition occurs due to the InSig[2] input event.

Bit 3: **DMP_EVER** dump update mode (R/W)

0: the earlier (first) between the selected events updates the dump register, the following ones do nothing.

1: the dump register is updated any time one of the configured events from the bit field [2:0] is triggered.

Bit 4: **CPL_IT_GE** lock together SMD<n>_GSTS and SMD<n>_IRQ reset signal (R/W)

0: the reset of SMD<n>_GSTS [2:0] bit fields and SMD<n>_IRQ[3:1] register fields is independent.

1: when clearing one of the SMD<n>_GSTS [2:0] bit fields, the corresponding bits of the SMD<n>_IRQ[3:1] register are automatically reset and vice versa. Moreover the EVENT_OV bit of SMD<n>_GSTS register is reset each time the system detects a write clear access of any of the SMD<n>_IRQ bit [3:1].

Bit 7-5: **RFU** reserved; must be kept 0 during register writing for future compatibility.

SMD<n>_FSM_STS (FSM status register)

Offset:0x23

Default value:0x00

7	6	5	4	3	2	1	0
RFU	EVINP [2:0]			PWM	FSM [2:0]		
r	r			r	r		

Bit 2-0: **FSM[2:0]** SMED finite state machine state variable (R)

This field reflects the current status of finite state machine state variable contents. Refer to *Table 34: SMED coupled interface configuration scheme on page 190* for coding value.

Bit 3: **PWM** PWM output signal (R)

This filed reflects the current status of the SMED PWM internal output signal value.

Bit 6-5: **EVINP[2:0]** event input signals (R)

This field reflects the current status of the three SMED input signal lines InSig[2:0].

Bit 7: **RFU** reserved for future use.

SMED core registers map

Table 37 shows the SMEDs internal registers (<n> represents the SMED IPs instance) starting from the base address reported in the corresponding device datasheet; for detailed registers description refer to *Section 21.18.1: SMED core registers on page 199*.

Table 37. SMED core register overview

Name	Description	Offset	Type	Reset value
SMD<n>_CTR	Control register	0x00	R/W	0x00
SMD<n>_CTR_TMR	Control timer register	0x01	R/W	0x00
SMD<n>_CTR_INP	Control input register	0x02	R/W	0x00
SMD<n>_CTR_DTRH	Dithering register	0x03	R/W	0x00
SMD<n>_TMR_T0L	Time T0 LSB register	0x04	R/W	0x00
SMD<n>_TMR_T0H	Time T0 MSB register	0x05	R/W	0x00
SMD<n>_TMR_T1L	Time T1 LSB register	0x06	R/W	0x00
SMD<n>_TMR_T1H	Time T1 MSB register	0x07	R/W	0x00
SMD<n>_TMR_T2L	Time T2 LSB register	0x08	R/W	0x00
SMD<n>_TMR_T2H	Time T2 MSB register	0x09	R/W	0x00
SMD<n>_TMR_T3L	Time T3 LSB register	0x0A	R/W	0x00
SMD<n>_TMR_T3H	Time T3 MSB register	0x0B	R/W	0x00
SMD<n>_PRM_ID0	Parameter 0 IDLE register	0x0C	R/W	0x00
SMD<n>_PRM_ID1	Parameter 1 IDLE register	0x0D	R/W	0x00
SMD<n>_PRM_ID2	Parameter 2 IDLE register	0x0E	R/W	0x00
SMD<n>_PRM_S00	Parameter 0 S0 register	0x0F	R/W	0x00
SMD<n>_PRM_S01	Parameter 1 S0 register	0x10	R/W	0x00
SMD<n>_PRM_S02	Parameter 2 S0 register	0x11	R/W	0x00
SMD<n>_PRM_S10	Parameter 0 S1 register	0x12	R/W	0x00
SMD<n>_PRM_S11	Parameter 1 S1 register	0x13	R/W	0x00
SMD<n>_PRM_S12	Parameter 2 S1 register	0x14	R/W	0x00
SMD<n>_PRM_S20	Parameter 0 S2 register	0x15	R/W	0x00
SMD<n>_PRM_S21	Parameter 1 S2 register	0x16	R/W	0x00
SMD<n>_PRM_S22	Parameter 2 S2 register	0x17	R/W	0x00
SMD<n>_PRM_S30	Parameter 0 S3 register	0x18	R/W	0x00
SMD<n>_PRM_S31	Parameter 1 S3 register	0x19	R/W	0x00
SMD<n>_PRM_S32	Parameter 2 S3 register	0x1A	R/W	0x00
SMD<n>_CFG	Timer configuration register	0x1B	R/W	0x00
SMD<n>_DMP_L	Dump counter LSB register	0x1C	R	0x00
SMD<n>_DMP_H	Dump counter MSB register	0x1D	R	0x00

Table 37. SMED core register overview (continued)

Name	Description	Offset	Type	Reset value
SMD<n>_GSTS	General status register	0x1E	R/W1	0x00
SMD<n>_IRQ	Interrupt request register	0x1F	R/W1	0x00
SMD<n>_IER	Interrupt enable register	0x20	R/W	0x00
SMD<n>_ISEL	External event control register	0x21	R/W	0x00
SMD<n>_DMP	Dump enable register	0x22	R/W	0x00
SMD<n>_FSM_STS	FSM status register	0x23	R	0x00

21.18.2 SMED environment configuration registers

This set of registers defines the SMED environment configuration and are located into a dedicated area register address space. These registers concern the functionalities that allow the user to manage the SMED external interface from/to other coupled SMEDs, to control the unlock mechanism and to define the InSig[2:0] input connection for each FSM. The MSC_SMSWEV register controlling the software events is also placed in this area.

MSC_SMDCFG01 (SMED0 to SMED1 global configuration register)

Offset:0x15

Default value:0x00

7	6	5	4	3	2	1	0
SMD1_GLBCONF [2:0]			SMD1_DRVOUT	SMD0_GLBCONF [2:0]			SMD0_DRVOUT
r/w			r/w	r/w			r/w

Bit 0:**SMD0_DRVOUT** PWM0 control merge output signal in the coupled mode(r), (s), (t)

Bit 3-1:**SMD0_GLBCONF[2:0]** SMED0 global configuration(r), (s), (t)

Bit 4:**SMD1_DRVOUT** PWM1 control merge output signal in the coupled mode(r), (s), (t)

Bit 7-5:**SMD1_GLBCONF[2:0]** SMED1 global configuration(r), (s), (t)

r. For register field description and the configuration values, refer to the SMED global configuration parameter shown in *Table 34: SMED coupled interface configuration scheme on page 190*.

s. This register is backward compatible with STLUX385 devices; in the current product the SMEDx_GLBCONF[3:0] has been split into two separates fields (SMDx_GLBCONF[2:0] and SMDx_DRVOUT) for better configurability.

t. Not all SMED peripherals, PWM signals and coupling schemes may be available depending on the product device. Refer to the product datasheet to find the device features supported. In case the SMED units are not available on the current product, the related register fields have to be considered reserved and must fill with 0 during writing the register.

MSC_SMDCFG23 (SMED2 to SMED3 global configuration register)

Offset:0x16

Default value:0x00

7	6	5	4	3	2	1	0
SMD3_GLBCONF [2:0]			SMD3_DRVOUT	SMD2_GLBCONF [2:0]			SMD2_DRVOUT
r/w			r/w	r/w			r/w

Bit 0:**SMD2_DRVOUT** PWM2 control merge output signal in the coupled mode(r), (s), (t)

Bit 3-1:**SMD2_GLBCONF[2:0]** SMED2 global configuration(r), (s), (t)

Bit 4:**SMD3_DRVOUT** PWM3 control merge output signal in the coupled mode(r), (s), (t)

Bit 7-5:**SMD3_GLBCONF[2:0]** SMED3 global configuration(r), (s), (t)

MSC_SMDCFG45 (SMED4 to SMED5 global configuration register)

Offset:0x17

Default value:0x00

7	6	5	4	3	2	1	0
SMD5_GLBCONF [2:0]			SMD5_DRVOUT	SMD4_GLBCONF [2:0]			SMD4_DRVOUT
r/w			r/w	r/w			r/w

Bit 0: **SMD4_DRVOUT** PWM4 control merge output signal in the coupled mode(u), (v), (w)

Bit 3-1: **SMD4_GLBCONF[2:0]** SMED4 global configuration(u), (v), (w)

Bit 4: **SMD5_DRVOUT** PWM5 control merge output signal in the coupled mode(u), (v), (w)

Bit 7-5: **SMD5_GLBCONF[2:0]** SMED5 global configuration(u), (v), (w)

u. For register field description and the configuration values, refer to the SMED global configuration parameter shown in *Table 34: SMED coupled interface configuration scheme on page 190*.

v. This register is backward compatible with STLUX385 devices; in the current product the SMEDx_GLBCONF[3:0] has been split in two separates fields (SMDx_GLBCONF[2:0] and SMDx_DRVOUT) for better configurability.

w. Not all SMED peripherals, PWM signals and coupling schemes may be available depending on the product device. Refer to the product datasheet to find the device features supported. In case the SMED units are not available on the current product, the related register fields have to be considered reserved and must fill with 0 during writing the register.

MSC_SMSWEV (SMEDs SW events)

Offset: 0x18

Default value: 0x00

7	6	5	4	3	2	1	0
RFU		SW5	SW4	SW3	SW2	SW1	SW0
r		r/w	r/w	r/w	r/w	r/w	r/w

Bit 0: **SW0** software event on SMED0

0: SW0 SMED event cleared.

1: SW0 SMED event asserted.

Bit 1: **SW1** software event on SMED1

0: SW1 SMED event cleared.

1: SW1 SMED event asserted.

Bit 2: **SW2** software event on SMED2

0: SW2 SMED event cleared.

1: SW2 SMED event asserted.

Bit 3: **SW3** software event on SMED3

0: SW3 SMED event cleared.

1: SW3 SMED event asserted.

Bit 4: **SW4** software event on SMED4

0: SW4 SMED event cleared.

1: SW4 SMED event asserted.

Bit 5: **SW5** software event on SMED5

0: SW5 SMED event cleared.

1: SW5 SMED event asserted.

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Note: Not all SMED SW events may be available on current product. Refer to the product datasheet to find the device features supported. In case the SMED units are not available on the current product, the related register fields have to be considered reserved and must fill with 0 during writing the register.

MSC_SMULOCK (SMEDs unlock)

Offset: 0x19

Default value: 0x00

7	6	5	4	3	2	1	0
RFU		Unlock_45	Unlock_23	Unlock_01	Use_Unlock_45	Use_Unlock_23	Use_Unlock_01
r		r/w	r/w	r/w	r/w	r/w	r/w

Bit 0: **Use_Unlock_01**

0: unlock feature disables.

1: enable unlock features for the SMED couple #1 (SMED0, 1). This configuration is applicable only for synchronous coupled SMEDs configuration.

Bit 1: **Use_Unlock_23**

0: unlock feature disables.

1: enable unlock features for the SMED couple #2 (SMED2, 3). This configuration is applicable only for synchronous coupled SMEDs configuration.

Bit 2: **Use_Unlock_45**

0: unlock feature disables.

1: enable unlock features for the SMED couple #3 (SMED4, 5). This configuration is applicable only for synchronous coupled SMEDs configuration.

Bit 3: **Unlock_01**

0: unlock feature command disables.

1: asserted unlock command for the SMED couple #1 (SMED0, 1). This configuration is applicable only for synchronous coupled SMEDs configuration.

Bit 4: **Unlock_23**

0: unlock feature command disables.

1: asserted unlock command for the SMED couple #2 (SMED2, 3). This configuration is applicable only for synchronous coupled SMEDs configuration.

Bit 5: **Unlock_45**

0: unlock feature command disables.

1: asserted unlock command for the SMED couple #3 (SMED4, 5). This configuration is applicable only for synchronous coupled SMEDs configuration.

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Note: Not all SMED couples configuration may be available on the current product. Refer to the product datasheet to find the device features supported. In case the SMED units are not available on the current product, the related register fields have to be considered reserved and must fill with 0 during writing the register.

MSC_CBOXS<n> (connection box selection SMED<n>)

Offset: 0x1A+<n>

Default value: 0x00

7	6	5	4	3	2	1	0
RFU		Conb_s<n>_2 [1:0]		Conb_s<n>_1 [1:0]		Conb_s<n>_0 [1:0]	
r		r/w		r/w		r/w	

<n> ranges from 0 to 5.

For register configuration refer to the layout of the connection box signal definition present in the product datasheet.

Bit 1-0: **Conb_s<n>_0[1:0]** connection box selection line for the SMED<n> input InSig[0].

Bit 3-2: **Conb_s<n>_1[1:0]** connection box selection line for the SMED<n> input InSig[1].

Bit 5-4: **Conb_s<n>_2[1:0]** connection box selection line for the SMED<n> input InSig[2].

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Note: Not all SMED peripherals may be available depending on the product device. Refer to the product datasheet to find the device features supported. In case the SMED units are not available on the current product, the related register fields have to be considered reserved and must fill with 0 during writing the register.

MSC_IOMXSMD (SMED I/O MUX control register)

Offset: 0x20

Default value: 0x00

7	6	5	4	3	2	1	0
Sel_fsmen1	Smd_fsmsl1[2:0]			Sel_fsmen0	Smd_fsmsl0[2:0]		
r/w	r/w			r/w	r/w		

This register selects the SMED finite state machine (FSM) internal status signals to be interconnected for probing on the primary P0s signals.

Bit 2-0: **Smd_fsms0[2:0]** SMED FSM status multiplexing output selection line for P0[2:0]:

000: select SMED0 FSM status signals
001: select SMED1 FSM status signals
010: select SMED2 FSM status signals
011: select SMED3 FSM status signals
100: select SMED4 FSM status signals
101: select SMED5 FSM status signals

Bit 3: **Sel_fsmen0** enable SMED FSM status signals multiplexing on P0[2:0]:

0: disable SMED FSM status signals multiplexing
1: enable SMED FSM status signal multiplexing

Bit 6-4: **Smd_fsms1[2:0]** SMED FSM status multiplexing output selection line for P0[5:3]:

000: select SMED0 FSM status signals

001: select SMED1 FSM status signals

010: select SMED2 FSM status signals

011: select SMED3 FSM status signals

100: select SMED4 FSM status signals

101: select SMED5 FSM status signals

Bit 7: **Sel_fsmen1** enable SMED finite state machine multiplexing output signal on P0[5:3]:

0: disable SMED FSM status signals multiplexing

1: enable SMED FSM status signal multiplexing

Note: Not all SMED peripherals may be available depending on the product device. Refer to the product datasheet to find the device features supported.

SMED environment configuration registers map

Table 38 shows the SMED environment configuration registers starting from the base address specified in the corresponding product datasheet; for detailed register description refer to *Section 21.18.2: SMED environment configuration registers on page 215*.

Table 38. SMED environment configuration register overview

Name	Description	Offset	Type	Reset value
MSC_SMDCFG01	SMED0,1 global config. reg.	0x15	R/W	0x00
MSC_SMDCFG23	SMED2, 3 global config. reg.	0x16	R/W	0x00
MSC_SMDCFG45	SMED4, 5 global config. reg.	0x17	R/W	0x00
MSC_SMSWEV	SMED SW event register	0x18	R/W	0x00
MSC_SMULOCK	SMED unlock register	0x19	R/W	0x00
MSC_CBOXS0	SMED0 ConBox config. reg.	0x1A	R/W	0x00
MSC_CBOXS1	SMED1 ConBox config. reg.	0x1B	R/W	0x00
MSC_CBOXS2	SMED2 ConBox config. reg.	0x1C	R/W	0x00
MSC_CBOXS3	SMED3 ConBox config. reg.	0x1D	R/W	0x00
MSC_CBOXS4	SMED4 ConBox config. reg.	0x1E	R/W	0x00
MSC_CBOXS5	SMED5 ConBox config. reg.	0x1F	R/W	0x00
MSC_IOMXSMD	SMED FSM trace ctr. reg.	0x20	R/W	0x00

22 General purpose I/O port (GPIO)

General purpose input/output ports are used for signaling and generic data transfer between the device and the external world.

Each port pin can be individually programmed as a digital input or a digital output; different optional modes can be selected in both cases. In addition, some ports may host alternative functionality like analog inputs, external interrupts, the input/output for on-chip peripherals. For further details on the specific port configuration pin, refer to the product datasheet.

Basically each port is a group of up to six pins, configured and managed by means of five 8-bit registers which define the status and the behavior of the port pins:

- Output data register
- Input data register
- Data direction register
- Two control registers

A particular port pin will behave as an input or output depending on the value of the relevant bit of the port data direction register (DDR).

22.1 GPIO main features

- Port bits can be configured individually
- Selectable input modes: floating input or input with pull-up
- Selectable output modes: push-pull output or pseudo open drain(x)
- Separate registers for data input and output
- External interrupts can be enabled and disabled individually trough the miscellaneous registers
- Output slope control for reduced EMC noise(y)
- Alternate function I/Os for on-chip peripherals
- Read-modify-write possible on data output latch

22.2 Port configuration and usage

The I/O signals are configured and controlled by the output data register (ODR), input data register (IDR) and data direction register (DDR).

The control register 1 (CR1) and control register 2 (CR2) allow configuring the input/output optional features. The functionality of each I/O pin is programmed through the corresponding bits of the DDR, ODR, CR1 and CR2 registers.

Some port pins have an associated alternate function. To enable the alternate function refer to the product datasheet.

x. Pseudo open drain feature not available for PWM output signals.

y. A feature not available for PWM and GPIO1[5:0] output signals.

Table 39 summarizes all possible GPIO signal configurations.

Table 39. GPIO port signal configuration

Mode	DDR bit	CR2 bit	CR1 bit	Function	Pull-up	P-buffer	Diodes	
							to V_{DD}	to V_{SS}
Input	0	0	0	Floating without interrupt(1)	Off	Off	On	On
	0	0	1	Pull-up without interrupt(1)	On			
	0	1	0	Floating with interrupt(1)	Off			
	0	1	1	Pull-up with interrupt(1)	On			
Output	1	0	0	Open drain output limited to 2 MHz(2)				
	1	0	1	Push pull output limited to 2 MHz	Off	Off		
						On		
	1	1	0	Open drain output limited to 10 MHz(2), (3)		Off		
	1	1	1	Push pull output limited to 10 MHz(3)		On		

1. The interrupt functionality is available for P0 and P2 ports and is configured through the miscellaneous registers; note that the CR2 register controls the interrupt mask feature for the input ports depending on the configuration of MSC_INPP2AUX1 and MSC_INPP2AUX2 registers. Refer to the product datasheet for further details.
2. Pseudo open drain pad configuration.
3. The configuration of I/O buffer speed is applicable only to the port P0.

22.2.1 Input modes

Clearing the DDR<x> bit configures the corresponding port pin as input. In this mode, reading an IDR bit returns the digital value of the pin as driven by external logic.

As shown in *Table 39*, four different input modes can be theoretically configured by software: the floating without interrupt, floating with interrupt, pull-up without interrupt or pull-up with interrupt. However in practice not all ports have the external interrupt capability or pull-ups.

You should refer to the datasheet pinout description for details on the actual hardware capability of each port.

22.2.2 Output modes

Setting the DDR<x> bit selects the output mode. In this mode, writing to the ODR bits applies a digital value to the I/O through the output latch. Reading the IDR bit returns the digital value from the corresponding I/O pin. Using the CR1, CR2 registers different output modes can be configured by software: push-pull output, pseudo open drain output.

22.2.3 Slope control

The maximum output pad toggle frequency is programmable by SW through the P0_CR2 control register. After reset the pad is configured in the low-speed mode (2 MHz) reducing EMC noise injection. Higher frequency (up to 10 MHz) can be selected if needed. This feature can be applied to the output port configured either in pseudo open drain or in the push-pull output mode.

Note that the slope control feature of the I/Os of the port P1 is not programmable.

22.2.4 Reset configuration

After reset almost all port-P0 I/Os are generally floating input signals although few pins may have a different behavior. The port P1 I/Os are configured as output pins. Refer to the product datasheet pinout description for all details.

22.2.5 Unused I/O pins

Unused I/O pins must be connected to fixed voltage levels and configured as input floating. Either connect a pull-up or pull-down to the unused I/O pins, or use the internal weak pull-up if it is available on the pins.

22.2.6 Low power modes

Table 40. Effect of low power modes on GPIO ports

Low power mode	
Mode	**Description**
Wait	No effect on I/O ports. External interrupts cause the device to exit from the wait mode.
Halt	No effect on I/O ports. External interrupts cause the device to wake up from the Halt mode.

Note: When P02/P03 pins are used to connect an external oscillator, to improve the power save in the Halt mode these pins3 should be configured as input with pull-up enabled.

22.3 Alternate output function

Alternate function outputs provide a direct path from a peripheral to an output or to an I/O pad, taking precedence over the output data register (Px_ODR) programming.

An alternate function output can be push-pull or pseudo open drain depending on the peripheral and control register 1 (Px_CR1) and slope can be controlled depending on the control register 2 (Px_CR2) values. For further detail refer to the product datasheet information.

22.4 Interrupt functionality

Every GPIO input signal is capable to generate an interrupt request by configuring the corresponding miscellaneous register; for further detail refer to the product datasheet.

22.5 Interrupt masking

An interrupt polling sequence may be selected on each individual pin by setting the corresponding bit in the configuration register (Px_CR2). When the Px_CR2[y] bit is cleared, the interrupt request asserted by the flag of the miscellaneous status register is masked. After reset, the interrupts are masked. The mask feature of the port-P0 has to be enabled via the miscellaneous register (for further information refer to the product datasheet).

Note: The interrupt capability on the port-P1 is available only on some device model.

22.6 GPIO registers description

Where: <n> is the index port.

<baseaddr>is the port base address reported by the datasheet.

22.6.1 P<n>_ODR (port output data register)

Offset: <baseaddr> + 0x00

Default value: 0x00

7	6	5	4	3	2	1	0
ODR [7:0]							
r/w							

Bit 7-0: **ODR[7:0]** output data register

Writing to the P<n>_ODR register when in the output mode applies a digital value to the I/O through the latch. Reading the P<n>_ODR returns the previously latched value in the register.

In the input mode, writing in the P<n>_ODR register, latches the value in the register but does not change the pin state. The P<n>_ODR register is always cleared after reset; the bit read-modify-write instructions (BSET, BRST) can be used with the P<n>_ODR register to drive an individual pin without affecting the others.

22.6.2 P<n>_IDR (port input data register)

Offset: <baseaddr> + 0x01

Default value: 0xXX

7	6	5	4	3	2	1	0
IDR [7:0]							
r							

Bit 7-0: **IDR[7:0]** input data register

This register (RO) can be used to read the pin value irrespective of whether port is in the input or output mode.

0: input pad low voltage level.

1: input pad high voltage level.

Note: P<n>_IDR reset value depends on the external circuitry.

22.6.3 P<n>_DDR (port data direction register)

Offset: <baseaddr> + 0x02

Default value: 0x00

7	6	5	4	3	2	1	0
DDR [7:0]							
r/w							

Bit 7-0: **DDR[7:0]** data direction register

These bits are set and cleared by software to select the input or output mode for a particular Port<n> pin.

0: configure the GPIO port in input mode.

1: configure the GPIO port in output mode.

22.6.4 P<n>_CR1 (port control register1)

Offset: <baseaddr> + 0x03

Default value: 0x00

7	6	5	4	3	2	1	0
C1 [7:0]							
r/w							

Bits 7:0: **C1[7:0]**: control bits

This register configurable by SW is used in conjunction with the P<n>DDR register to select different input/output port features:

In input mode (DDR = 0):

0: floating input.

1: input with pull-up.

In output mode (DDR = 1):

0: pseudo open drain.

1: push-pull, slope control for the output depends on the corresponding P<n>_CR2 bit.

Note: The pseudo open drain feature is not applicable to the PWMs output signals.

22.6.5 P<n>_CR2 (port control register2)

Offset: <baseaddr> + 0x04

Default value: 0x00

7	6	5	4	3	2	1	0
C2 [7:0]							
r/w							

Bits 7:0: **C2[7:0]:** control bits

These bits are set and cleared by software. They select different functions in the input mode and output mode.

– In the input mode, the P<n>_CR2 bit selects the polling or the interrupt mode. If the I/O does not have the interrupt capability, setting the P<n>_CR2 bit has no effect.

– In the output mode, these bits control the I/O speed.

- In output mode (DDR = 1)[z], [aa]:

  0: output speed up to 2 MHz.

  1: output speed up to 10 MHz.

- In input mode (DDR = 0)[ab], [ac]:

  0: interrupt mask enable (polling mode).

  1: interrupt mask disable.

z. This feature is available only for the Port0 (GPIO0).

aa. The output speed of the Port1 (GPIO1) is not configurable and can sustain a toggle-rate up to 12 MHz.

ab. The interrupt mask feature of the port-P0 is enabled by the miscellaneous register (refer to the product datasheet). If the mask feature is disabled, the interrupt logic is permanently enabled. When the interrupt capability is available on the port-P1, the mask features controllable via the P1_CR2 doesn't require additional enabling.

ac. The interrupt mask feature of the auxiliary/basic timers (when available) is controlled by the bit 0, 1 and 5 of the P1_CR2 while the interrupt detection logic is configured through the miscellaneous registers (refer to *Section 14.12: Miscellaneous interrupt registers on page 127*).

22.7 GPIO registers overview

Table 41 shows the GPIO internal registers (<n> represents the GPIO IPs instance) starting from the base address reported in the corresponding device datasheet; for detailed registers description refer to *Section 21.9: State transition on page 171*.

Table 41. GPIO internal registers overview

Name	Description	Offset	Type	Reset value
P<n>_ODR	Output data register	0x00	R/W	0x00
P<n>_IDR	Input data register	0x01	R	0xXX
P<n>_DDR	Data direction register	0x02	R/W	0x00
P<n>_CR1	Control register 1	0x03	R/W	0x00
P<n>_CR2	Control register 2	0x04	R/W	0x00

23 Inter-integrated circuit interface (I2C)

23.1 I2C introduction

The I2C (inter-integrated circuit) bus interface serves as an interface between the microcontroller and the serial I2C bus. It provides a multi-master capability, and controls all I2C bus-specific sequencing, protocol, arbitration and timing. It supports standard and fast speed modes.

23.2 I2C main features

- Parallel-bus/I2C protocol converter
- Multi-master capability: the same interface can act as a master or slave
- I2C master features:
  - Clock generation
  - Start and stop generation
- I2C slave features:
  - Programmable I2C address detection
  - Stop bit detection
- Generation and detection of 7-bit/10-bit addressing and general call
- Supports different communication speeds:
  - Standard speed (up to 100 kHz)
  - Fast speed (up to 400 kHz)
- Status flags:
  - Transmitter/receiver mode flag
  - End of byte transmission flag
  - I2C busy flag
- Error flags:
  - Arbitration lost condition for master mode
  - Acknowledgment failure after address/ data transmission
  - Detection of misplaced start or stop condition
  - Overrun/underrun if clock stretching is disabled
- 3 types of interrupts:
  - 1 communication interrupt
  - 1 error condition interrupt
  - 1 control interrupt
- Wakeup capability:
  - MCU wakes up from low power mode on address detection in slave mode
- Optional clock stretching

23.3 I2C signal interface

The I2C_scl and I2C_sda signals are multifunction pins configured through the I/O multiplex mechanism described in the product datasheet. The users can program an internal (weak) pull-up on the I2C signal lines in accordance with the selected pins.

23.4 I2C general description

This peripheral converts the data to be transmitted from the parallel to the serial format and vice-versa in case of the data reception. Data transfer and control events may be handled both in the polling mode and in interrupt mode.

This peripheral is connected to the I2C bus by a data pin (SDA) and by a clock pin (SCL); the peripheral supports both the standard (up to 100 kHz), and the fast (up to 400 kHz) I2C line speed.

Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in the slave mode. The interface automatically switches from the slave to the master, after it generates a START condition and from the master to the slave, if an arbitration loss or a STOP generation occurs, allowing the multi-master capability.

23.5 Communication flow

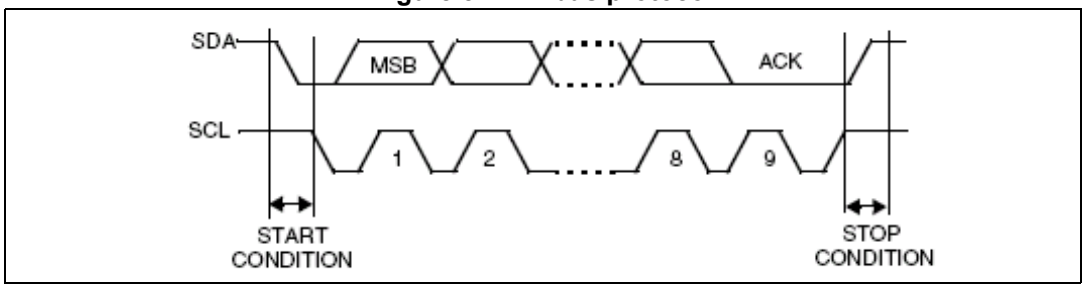
In the master mode, the I2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in the master mode by software.

In the slave mode, the interface is capable of recognizing its own addresses (7- or 10-bit), and the general call address. The general call address detection may be enabled or disabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the start condition contain the address (one byte in 7-bit mode, two bytes in 10-bit mode). The address is always transmitted in the master mode.

A 9th clock pulse follows the 8 clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter. Refer to *Figure 52*.

Figure 52. I2C bus protocol



Acknowledge may be enabled or disabled by software. The I2C interface addresses (7-bit/10-bit and/or general call address) can be selected by software.

23.6 I2C functional description

By default the I2C interface operates in the slave mode. To switch from the default slave mode to the master mode a start condition generation is needed.

23.6.1 I2C slave mode

The peripheral input clock must be programmed in the I2C_FREQR register in order to generate correct timings. The peripheral input clock frequency must be at least:

- 1 MHz in standard mode
- 4 MHz in fast mode

As soon as a start condition is detected, the address is received from the SDA line and sent to the shift register. Then it is compared with the address of the interface (OARL and OARH) or the general call address (if ENGC = 1).

Note: In the 10-bit addressing mode, the comparison includes the header sequence (11110xx0), where xx denotes the two most significant bits of the address.

Header or address not matched: the interface ignores it and waits for another start condition.

Header matched (10-bit mode only): the interface generates an acknowledge pulse if the ACK bit is set and waits for the 8-bit slave address.

Address matched: the interface generates in sequence:

An acknowledge pulse if the ACK bit is set.

- The ADDR bit is set by hardware and an interrupt is generated if the ITEVTEN bit is set.

In the 10-bit mode, after receiving the address sequence the slave is always in the receiver mode. It will enter the transmitter mode on receiving a repeated start condition followed by the header sequence with matching address bits and the least significant bit set (11110xx1).

The TRA bit indicates whether the slave is in the receiver or transmitter mode.

Slave transmitter

Following the address reception and after clearing the ADDR, the slave sends bytes from the DR register to the SDA line via the internal shift register.

The slave stretches the SCL low until the ADDR is cleared and the DR filled with the data to be sent (see transfer sequencing EV1 EV3 in *Figure 54*).

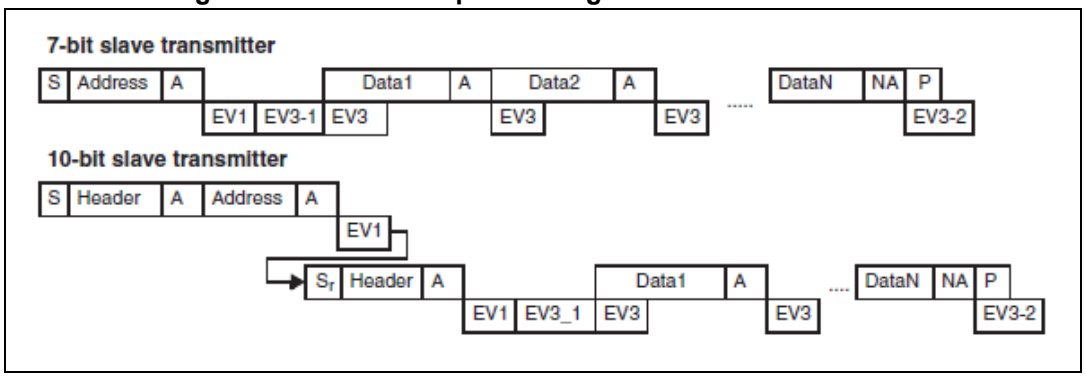
When the acknowledge pulse is received:

- The TxE bit is set by hardware with an interrupt if the ITEVTEN and the ITBUFEN bits are set.

If the TxE is set and a data was not written in the DR register before the end of the next data transmission, the BTF bit is set and the interface waits for a write in the DR register, stretching the SCL low.

In *Figure 54* it is shown the slave transmitter transfer sequence.

Figure 53. Transfer sequence diagram for slave transmitter



Note: 1.Legend:

a) **S** = start, **Sr** = repeated start, **P** = stop, **A** = acknowledge, **NA** = non-acknowledge, **EVx** = event (with interrupt if ITEVTEN = 1).

b) **EV1:** ADDR = 1, cleared by reading the SR1 register followed by reading the SR3.

c) **EV3-1**: TXE = 1, the shift register empty, the data register empty, write Data1 in the DR.

d) **EV3:** TXE = 1, the shift register not empty, the data register empty, cleared by writing the DR.

e) **EV3-2:** AF = 1, the AF is cleared by writing '0' in the AF bit of the SR2 register.

2. EV1 and EV3-1 events stretch the SCL low until the end of the corresponding software sequence.

3. The EV3 software sequence must be performed before the end of the current byte transfer. In case EV3 software sequence cannot be managed before the end of the current byte transfer, it is recommended to use the BTF instead of the TXE with the drawback of slowing the communication.

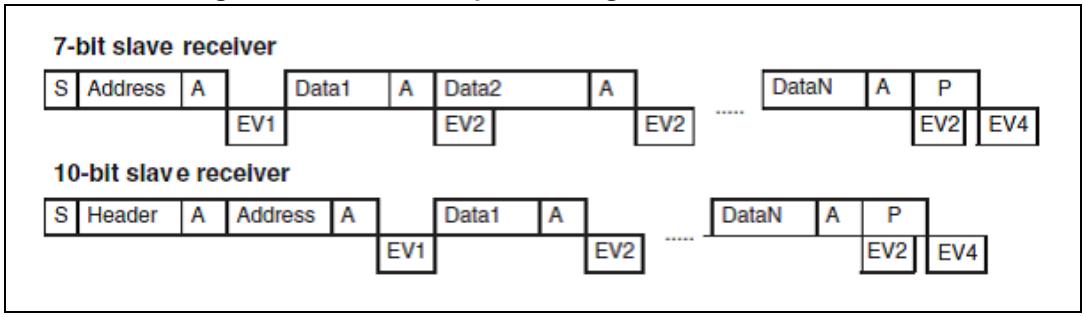
Slave receiver

Following the address reception and after clearing the ADDR, the slave receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set.
- The RxNE bit is set by hardware and an interrupt is generated if the ITEVTEN and TBUFEN bit is set.

If the RxNE is set and the data in the DR register is not read before the end of the next data reception, the BTF bit is set and the interface waits for a read of the DR register, stretching the SCL low (see *Figure 54*).

Figure 54. Transfer sequence diagram for slave receiver



Note: *1. Legend:*

a) ***S** = Start, **Sr** = repeated start, **P** = stop, **A** = acknowledge, **NA** = non-acknowledge, **EVx** = Event (with interrupt if ITEVTEN=1)*

b) **EV1:** ADDR =1, cleared by reading SR1 register followed by reading SR3.

c) **EV2:** RXNE=1, cleared by reading DR register.

d) **EV4:** STOPF=1, cleared by reading SR1 register followed by writing CR2 register

2. The EV1 event stretches the SCL low until the end of the corresponding software sequence.

3. The EV2 software sequence must be performed before the end of the current byte transfer.

4. After checking the SR1 register content, the user should perform the complete clearing sequence for each flag found set. Thus, for the ADDR and STOPF flags, the following sequence is recommended inside the I^2C interrupt routine:

READ SR1

if (ADDR == 1) {READ SR1; READ SR3}

if (STOPF == 1) {READ SR1; WRITE CR2}

The purpose is to make sure that both ADDR and STOPF flags are cleared if both are found set.

Closing slave communication

After the last data byte is transferred, a stop condition is generated by the master. The interface detects this condition and sets, the STOPF bit and generates an interrupt if the ITEVTEN bit is set. Then the interface waits for a read of the SR1 register followed by a write to the CR2 register (refer to transfer sequencing EV4 in *Figure 54*).

23.6.2 I2C master mode

In the master mode, the I2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. The master mode is selected as soon as the start condition is generated on the bus with a START bit.

The following is the required sequence in the master mode.

- Program the peripheral input clock in the I2C_FREQR register in order to generate correct timings
- Configure the clock control registers
- Configure the rise time register
- Program the I2C_CR1 register to enable the peripheral
- Set the START bit in the I2C_CR2 register to generate a start condition

The peripheral input clock frequency must be at least:

- 1 MHz in standard mode
- 4 MHz in fast mode

Start condition

Setting the START bit while the BUSY bit is cleared causes the interface to generate a start condition and switch to the master mode (MSL bit set).

Note: In the master mode, setting the START bit causes the interface to generate a restart condition at the end of the current byte transfer.

Once the start condition is sent:

- The SB bit is set by hardware and an interrupt is generated if the ITEVTEN bit is set.

Then the master waits for a read of the SR1 register followed by a write in the DR register with the slave address (refer to *Figure 55* and *Figure 56* for transfer sequencing EV5).

Slave address transmission

In this mode the slave address is sent to the SDA line via the internal shift register.

- In the 10-bit addressing mode, sending the header sequence causes the following event:
  - The ADD10 bit is set by hardware and an interrupt is generated if the ITEVTEN bit is set.
    Then the master waits for a read of the SR1 register followed by a write in the DR register with the second address byte (refer to the next two figures Figure 20 4 and Figure 20 5 Transfer sequencing EV9).
    The ADDR bit is set by hardware and an interrupt is generated if the ITEVTEN bit is set. Then the master waits for a read of the SR1 register followed by a read in the SR3 register (refer to the *Figure 55* and *Figure 56* for transfer sequencing EV6).
- In the 7-bit addressing mode, one address byte is sent.

As soon as the address byte is sent,

– The ADDR bit is set by hardware and an interrupt is generated if the ITEVTEN bits set.

Then the master waits for a read of the SR1 register followed by a read in the SR3 register (see the next two figures *Figure 55* and *Figure 56* for transfer sequencing EV6).

The master can decide to enter the transmitter or receiver mode depending on the LSB of the slave address sent.

- In the 7-bit addressing mode:
  – To enter the transmitter mode, a master sends the slave address with the LSB reset.
  – To enter the receiver mode, a master sends the slave address with the LSB set.
- In the 10-bit addressing mode:
  – To enter the transmitter mode, a master sends the header (11110xx0) and then the 8 LSB slave address.
  – To enter the receiver mode, a master sends the (11110xx0) header and then the 8 LSB slave address. Then it should send a repeated start condition followed by the header sequence with matching address bits and the least significant bit set (11110xx1).

The TRA bit indicates whether the master is in the receiver or transmitter mode.

Note: The xx denotes the two most significant bits of the 10 bit address.

Master transmitter

Following the address transmission and after clearing the ADDR, the master sends bytes from the DR register to the SDA line via the internal shift register.

The master waits until the first data byte is written in the DR register (TxE is cleared), refer to *Figure 55* for transfer sequencing EV8_1.

When the acknowledge pulse is received:

- The TxE bit is set by hardware and an interrupt is generated if the ITEVTEN and ITBUFEN bits are set.

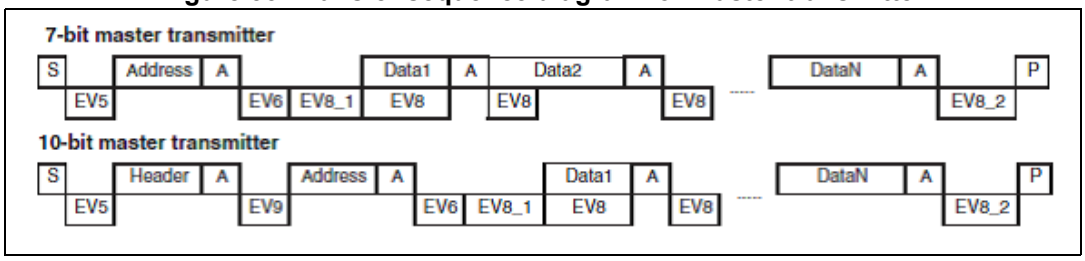
If TxE is set and a data byte was not written in the DR register before the end of the next data transmission, the BTF is set and the interface waits until the BTF is cleared by reading the SR1 register and then writing to the DR register, stretching the SCL low.

Closing the communication

After writing the last byte to the DR register, the STOP bit is set by software to generate a stop condition (refer to *Figure 55* for transfer sequencing EV8_2). The interface goes automatically back to the slave mode (MSL bit cleared).

Note: Stop condition should be programmed during the EV8_2 event, when either the TxE or BTF is set.

Figure 55. Transfer sequence diagram for master transmitter



Note: 1. Legend:

a) ***S** = start, **Sr** = repeated start, **P** = stop, **A** = acknowledge, **NA** = non-acknowledge, **EVx** = event (with interrupt if ITEVTEN = 1).*

b) ***EV5:** SB = 1, cleared by reading the SR1 register followed by writing the DR register with the address.*

c) ***EV6:** ADDR = 1, cleared by reading the SR1 register followed by reading the SR3.*

d) ***EV8_1:** TXE = 1, the shift register empty, the data register empty, write the DR register.*

e) ***EV8:** TXE = 1, the shift register not empty, the data register empty, cleared by writing the DR register.*

f) ***EV8_2:** TXE = 1, BTF = 1, a program STOP request. The TXE and BTF are cleared by HW by stop condition.*

g) ***EV9:** ADD10 = 1, cleared by reading the SR1 register followed by writing the DR register.*

2. EV8 software sequence must be performed before the end of the current byte transfer. In case EV8 software sequence cannot be managed before the end of the current byte transfer, it is recommended to use the BTF instead of the TXE with the drawback of slowing the communication.

Master receiver

Following the address transmission and after clearing the ADDR, the I2C interface enters the master receiver mode. In this mode the interface receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set.
- The RxNE bit is set and an interrupt is generated if the ITEVTEN and ITBUFEN bits are set (refer to *Figure 56* for transfer sequencing EV7).

If the RxNE bit is set and the data were not read in the DR register before the end of the next data reception, the BTF bit is set by hardware and the interface waits for the BTF bit to be cleared by reading the I2C_SR1 and then the I2C_DR, stretching the SCL low.

Closing the communication

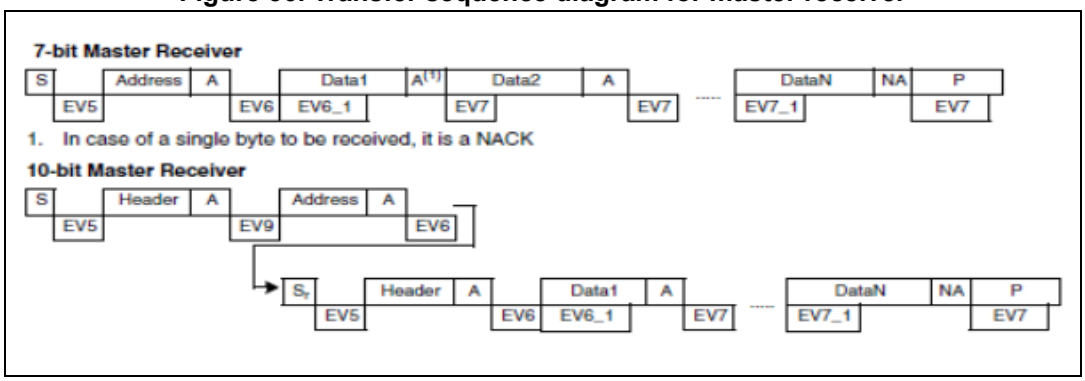
Method 1: This method is for the case when the I2C is used with interrupts that have the highest priority in the application.

The master sends a NACK for the last byte received from the slave. After receiving this NACK, the slave releases the control of the SCL and SDA lines. Then the master can send a stop/restart condition.

- In order to generate the non-acknowledge pulse after the last received data byte, the ACK bit must be cleared just after reading the second last data byte (after second last RxNE event).
- In order to generate the stop/restart condition, software must set the STOP/ START bit just after reading the second last data byte (after the second last RxNE event).
- In case a single byte is to be received, the acknowledge deactivation and the STOP condition generation are made just after the EV6 (in EV6-1 just after ADDR is cleared).

After the stop condition generation, the interface goes automatically back to the slave mode (MSL bit cleared).

Figure 56. Transfer sequence diagram for master receiver



Note: 1. Legend:

a) ***S** = start, **Sr** = repeated start, **P** = stop, **A** = acknowledge, **NA** = non-acknowledge, **EVx** = event (with interrupt if ITEVTEN = 1).*

b) ***EV5:** SB = 1, cleared by reading the SR1 register followed by writing the DR register.*

c) ***EV6:** ADDR = 1, cleared by reading the SR1 register followed by reading the SR3. In the 10-bit master receiver mode, this sequence should be followed by writing the CR2 with START = 1.*

d) ***EV6_1:** no associated flag event, used for 1 byte reception only. Program the ACK = 0 and STOP = 1 after clearing the ADDR.*

e) ***EV7:** RxNE = 1, cleared by reading the DR register.*

f) ***EV7_1:** RxNE = 1, cleared by reading the DR register, program the ACK = 0 and the STOP request.*

g) ***EV9:** ADD10 = 1, cleared by reading the SR1 register followed by writing the DR register.*

2. If the DR and shift registers are full, the next data reception (I²C clock generation for slave) is performed after the EV7 event is cleared. In this case, the EV7 does not overlap with data reception.

3. The EV5, EV6 and EV9 events stretch the SCL low until the end of the corresponding software sequence.

4. The EV7 software sequence must be completed before the end of the current byte transfer. In case the EV7 software sequence cannot be managed before the current byte end of transfer, it is recommended to use the BTF instead of the RXNE with the drawback of slowing the communication.

5. The EV6_1 or EV7_1 software sequence must be completed before the ACK pulse of the current byte transfer.

Method 2: This method is for the case when the I²C is used with interrupts that do not have the highest priority in the application or when the I²C is used with polling.

With this method:

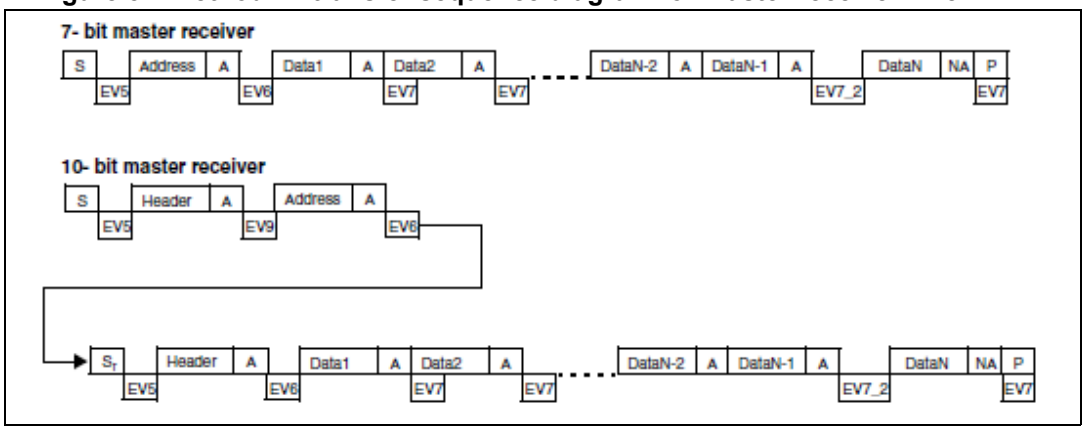
- The DataN-2 is not read, so that after the DataN-1, the communication is stretched (both RxNE and BTF are set).
- Then, the ACK bit must be cleared before reading the DataN-2 in the DR to make sure this bit has been cleared before the DataN acknowledge pulse.
- After that, just after reading the DataN 2, software must set the STOP/START bit and read the DataN 1. After the RxNE is set, read the DataN.

The next session describes the transfer data of different frame data length.

When 3 bytes remain to be read (procedure applicable when N > 2):

- RxNE = 1 => nothing (DataN 2 not read).
- DataN 1 received
- BTF = 1 because both shift and data registers are full: the DataN 2 in the DR and the DataN 1 in the shift register => the SCL tied low: no other data will be received on the bus.
- Clear ACK bit
- Read the DataN 2 in the DR => this launches the DataN reception in the shift register
- DataN received (with a NACK)
- Program START/STOP
- Read DataN 1
- RxNE = 1
- Read DataN

Figure 57. Method 2 - transfer sequence diagram for master receiver when N > 2



Note: 1. Legend:

a) **S** = start, **Sr** = repeated start, **P** = stop, **A** = acknowledge, **NA** = non-acknowledge, **EVx**= event (with interrupt if ITEVTEN = 1).

b) ***EV5:*** *SB = 1, cleared by reading the SR1 register followed by writing the DR register.*

c) ***EV6:*** *ADDR1, cleared by reading the SR1 register followed by reading the SR3. In the 10-bit master receiver mode, this sequence should be followed by writing the CR2 with START = 1.*

d) ***EV7:*** *RxNE=1, cleared by reading the DR register.*

e) ***EV7_2:*** *BTF = 1, the DataN-2 in DR and DataN-1 in the shift register, program ACK = 0, read the DataN-2 in the DR. Program STOP = 1, read the DataN-1.*

f) ***EV9:*** *ADD10= 1, cleared by reading SR1 register followed by writing DR register.*

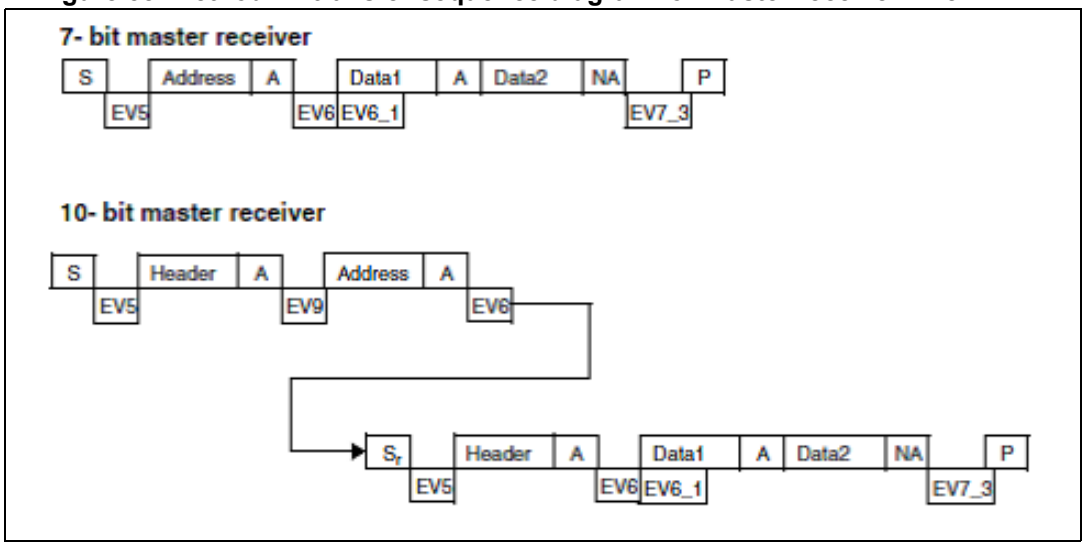
2. The EV5, EV6 and EV9 events stretch the SCL low until the end of the corresponding software sequence.

3. The EV7 software sequence must be completed before the end of the current byte transfer. In case the EV7 software sequence cannot be managed before the current byte end of transfer, it is recommended to use the BTF instead of the RXNE, with the drawback of slowing the communication.

When 2 bytes remain to be read (procedure applicable when N = 2):

- Set POS and ACK
- Wait for the ADDR flag to be set
- Clear ADDR
- Clear ACK
- Wait for BTF to be set
- Program STOP
- Read DR register twice times

Figure 58. Method 2 - transfer sequence diagram for master receiver when N = 2



Note: 1. Legend:

a) ***S** = start, **Sr** = repeated start, **P** = stop, **A** = acknowledge, **NA** = non-acknowledge, **EVx** = event (with interrupt if ITEVTEN = 1).*

b) ***EV5:** SB = 1, cleared by reading the SR1 register followed by writing the DR register.*

c) ***EV6:** ADDR1, cleared by reading the SR1 register followed by reading the SR3. In the 10-bit master receiver mode, this sequence should be followed by writing the CR2 with START = 1.*

d) ***EV6_1:** no associated flag event. The acknowledge should be disabled just after the EV6, that is after the ADDR is cleared.*

e) ***EV7_3:** BTF = 1, program STOP = 1, read the DR twice (read Data1 and Data2) just after programming the STOP.*

f) ***EV9:** ADD10 = 1, cleared by reading the SR1 register followed by writing the DR register.*

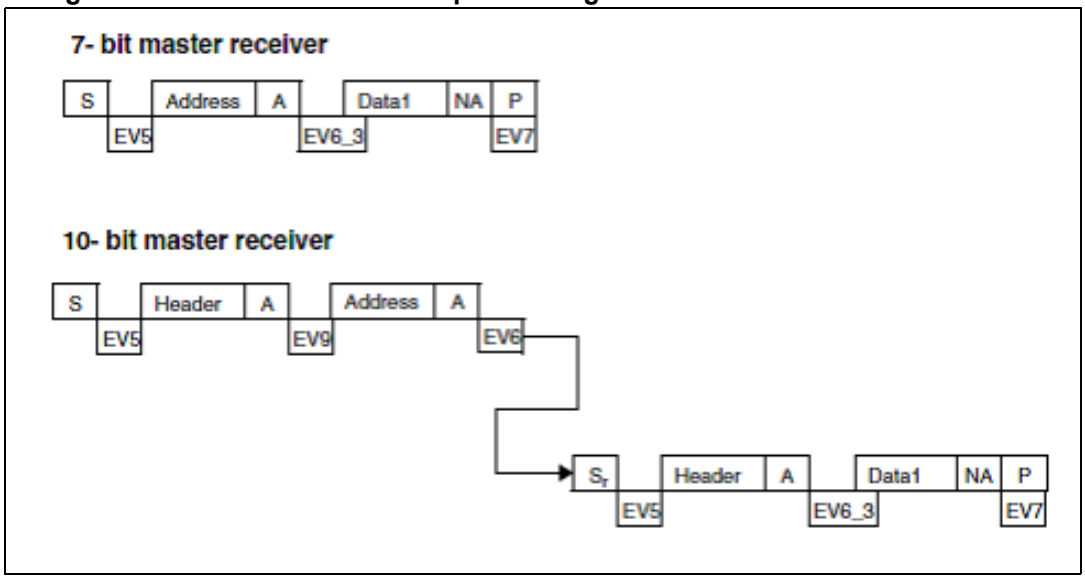
2. The EV5, EV6 and EV9 events stretch the SCL low until the end of the corresponding software sequence.

3. The EV6_1 software sequence must be completed before the ACK pulse of the current byte transfer.

When 1 byte remains to be read (procedure applicable when N = 1):

- In the ADDR event, clear the ACK bit.
- Clear ADDR
- Program the STOP/START bit.
- Read the data after the RxNE flag is set.
-

Figure 59. Method 2- transfer sequence diagram for master receiver when N = 1



Note: 1. Legend:

a) ***S** = start, **Sr** = repeated start, **P** = stop, **A** = acknowledge, **NA** = non-acknowledge, **EVx** = event (with interrupt if ITEVTEN = 1).*

b) ***EV5:** SB = 1, cleared by reading the SR1 register followed by writing the DR register.*

c) ***EV6:** ADDR = 1, cleared by reading the SR1 resister followed by reading the SR3 register.*

d) ***EV6_3:** ADDR = 1, program ACK = 0, clear the ADDR by reading the SR1 register followed by reading the SR3 register, program STOP = 1 just after the ADDR is cleared.*

e) ***EV7:** RxNE = 1, cleared by reading the DR register.*

f) ***EV9:** ADD10 = 1, cleared by reading the SR1 register followed by writing the DR register.*

2. The EV5, EV6 and EV9 events stretch the SCL low until the end of the corresponding software sequence.

3. The EV6_3 software sequence must be completed before the the ACK pulse of the current byte transfer.

23.6.3 Error conditions

The following are the error conditions which may cause communication to fail.

Bus error (BERR)

This error occurs when the I^2C interface detects an external stop or a start condition during an address or data transfer. In this case:

- The BERR bit is set and an interrupt is generated if the ITERREN bit is set.
- In the case of the slave: data are discarded and the lines are released by hardware:
  - In the case of a misplaced start, the slave considers it is a restart and waits for an address or a stop condition.
  - In the case of a misplaced stop, the slave reacts in the same way as for a stop condition and the lines are released by hardware.
- In the case of the master: the lines are not released and there is no effect in the state of the current transmission: software can decide if it wants to abort the current transmission or not.

Acknowledge failure (AF)

This error occurs when the interface detects a non-acknowledge bit. In this case:

- The AF bit is set and an interrupt is generated if the ITERREN bit is set.
- A transmitter which receives a NACK must reset the communication:
  - If slave: lines are released by hardware.
  - If master: a stop condition or repeated start must be generated by software.

Arbitration lost (ARLO)

This error occurs when the I^2C interface detects an arbitration lost condition. In this case:

- The ARLO bit is set by hardware (and an interrupt is generated if the ITERREN bit is set).
- The I^2C interface goes automatically back to the slave mode (the MSL bit is cleared).
- When the I^2C loses the arbitration, it is not able to acknowledge its slave address in the same transfer, but it can acknowledge it after a repeated start from the master.
- Lines are released by hardware.

Overrun/underrun error (OVR)

Overrun error can occur in the slave mode when clock stretching is disabled and the I^2C interface is receiving data. The interface has received a byte (RXNE = 1) and the data in the DR has not been read, before the next byte is received by the interface. In this case:

- The last received byte is lost.
- In case of overrun error, software should clear the RXNE bit and the transmitter should retransmit the last received byte.

Underrun error can occur in the slave mode when clock stretching is disabled and the I^2C interface is transmitting data. The interface has not updated the DR with the next byte (TXE = 1), before the clock comes for the next byte. In this case:

- The same byte in the DR register will be sent again.
- The user should make sure that data received on the receiver side during an underrun error is discarded and that the next bytes are written within the clock low time specified in the I^2C bus standard.
- For the first byte to be transmitted, the DR must be written after the ADDR is cleared and before the first SCL rising edge. If it is not possible, the receiver must discard the first data.

23.6.4 SDA/SCL line control

- If clock stretching is enabled:
  - Transmitter mode: If TXE = 1 and BTF = 1: the interface holds the clock line low before transmission to wait for the microcontroller to read the SR1 and then write the byte in the data register (both buffer and shift register are empty).
  - Receiver mode: If RXNE = 1 and BTF = 1: the interface holds the clock line low after reception to wait for the microcontroller to read the SR1 and then read the byte in the data register or write to the CR2 (both buffer and shift register are full).
- If clock stretching is disabled in the slave mode:
  - Overrun error in case of RXNE = 1 and no read of the DR has been done before the next byte is received. The last received byte is lost.
  - Underrun error in case TXE = 1 and no write into the DR has been done before the next byte must be transmitted. The same byte will be sent again.
  - Write collision not managed.

23.7 Limitations on I^2C peripheral usage

23.7.1 I^2C event management

As described in the previous sections, the application firmware has to manage several software events before the current byte is transferred. If the EV7, EV7_1, EV6_1, EV6_3, EV2, EV8, and EV3 events are not managed before the current byte is transferred, problems may occur such as receiving an extra byte, reading the same data twice, or missing data.

When the EV7, EV7_1, EV6_1, EV6_3, EV2, EV8, and EV3 events cannot be managed before the current byte transfer, and before the acknowledge pulse when the ACK control bit changes, it is recommended to use I^2C interrupts in the nested mode and to make them uninterruptible by increasing their priority to the highest priority in the application.

23.7.2 Corrupted last received data in master receiver mode

In the master receiver mode, when the communication is closed using the method 2, the content of the last read data may be corrupted. The following two sequences are concerned by the limitation:

Sequence 1: transfer sequence for the master receiver when N = 2

a) BTF = 1 (data N-1 in the DR and data N in the shift register)
b) Program STOP = 1
c) Read DR twice (read data N-1 and data N) just after programming the STOP bit.

Sequence 2: transfer sequence for the master receiver when N > 2

a) BTF = 1 (data N-2 in the DR and data N-1 in the shift register)
b) Program ACK = 0
c) Read data N-2 in the DR
d) Program STOP bit to 1
e) Read data N-1.

In this conditions the content of the shift register (data N) is corrupted (data N is shifted 1 bit to the left) if the user software is not able to read data N-1 before the STOP condition is generated on the bus. In this case, reading data N returns a wrong value.

Workaround 1

Sequence 1

When sequence 1 is used to close communication using the method 2, mask all active interrupts between the STOP bit programming and read data N-1.

Sequence 2

When sequence 2 is used to close communication using the method 2, mask all active interrupts between read data N-2, the STOP bit programming and read data N-1.

Workaround 2

Manage the I^2C RxNE and TxE events with interrupts of the highest priority level, so that the condition BTF = 1 never occurs.

23.7.3 Wrong behavior of I^2C peripheral in master mode after misplaced STOP

The I^2C peripheral does not enter the master mode properly if a misplaced STOP is generated on the bus. This can happen in the following conditions:

- If a void message is received (START condition immediately followed by a STOP): the BERR (bus error) flag is not set, and the I^2C peripheral is not able to send a START condition on the bus after writing to the START bit in the I2C_CR2 register.
- In the other cases of a misplaced STOP, the BERR flag is set in the IC2_CR2 register. If the START bit is already set in the I2C_CR2, the START condition is not correctly generated on the bus and can create bus errors.

Workaround

In the I^2C standard, it is not allowed to send a STOP before the full byte is transmitted (8 bits + acknowledge). Other derived protocols like the CBUS allow it, but they are not supported by the I^2C peripheral.

In case of a noisy environment in which unwanted bus errors can occur, it is recommended to implement a timeout to ensure that the SB (start bit) flag is set after the START control bit is set. In case the timeout has elapsed, the peripheral must be reset by setting the SWRST bit in the I2C_CR2 control register. The I^2C peripheral should be reset in the same way if a BERR is detected while the START bit is set in the I2C_CR2.

23.7.4 Violation of I^2C "setup time for repeated START condition" parameter

In case of a repeated start, the "setup time for repeated START condition" parameter (named $t_{SU;STA}$ in the datasheet and $t_{SU;STA}$ in the I^2C specifications) may be slightly violated when the I^2C operates in the master standard mode at a frequency ranging from 88 to 100 kHz. $t_{SU;STA}$ minimum value may be 4 µs instead of 4.7 µs.

The issue occurs under the following conditions:

1. The I^2C peripheral operates in the master standard mode at a frequency ranging from 88 to 100 kHz (no issue in fast mode).
2. And the SCL rise time meets one of the following conditions:
   – The slave does not stretch the clock and the SCL rise time is more than 300 ns (the issue cannot occur when the SCL rise time is less than 300 ns).
   – Or the slave stretches the clock.

Workaround

Reduce the frequency down to 88 kHz or use the I^2C fast mode if it is supported by the slave.

23.7.5 In I^2C slave "NOSTRETCH" mode, underrun errors may not be detected and may generate bus errors

The data valid time ($t_{VD;DAT}$, $t_{VD;ACK}$) described by the I^2C specifications may be violated as well as the maximum current data hold time ($t_{HD;DAT}$) under the conditions described below.

In addition, if the data register is written too late and close to the SCL rising edge, an error may be generated on the bus: the SDA toggles while the SCL is high. These violations cannot be detected because the OVR flag is not set (no transmit buffer underrun is detected).

This issue occurs under the following conditions:

1. The I^2C peripheral operates in the slave transmit mode with clock stretching disabled (NOSTRETCH = 1).
2. The application is late to write the DR data register, but not late enough to set the OVR flag (the data register is written before the SCL rising edge).

Workaround

If the master device supports it, use the clock stretching mechanism by programming the bit NOSTRETCH = 0 in the I2C_CR1 register.

If the master device does not support it, ensure that the write operation to the data register is performed just after the TXE or ADDR events. You can use an interrupt on the TXE or ADDR flag and boost its priority to the higher level.

Using the “NOSTRETCH” mode with a slow I^2C bus speed can prevent the application from being late to write the DR register (second condition).

Note: The first data to be transmitted must be written into the data register after the ADDR flag is cleared, and before the next SCL rising edge, so that the time window to write the first data into the data register is less than t_{LOW}.

If this is not possible, a possible workaround can be the following:

1. *Clear the ADDR flag.*
2. *Wait for the OVR flag to be set.*
3. *Clear the OVR and write the first data.*

The time window for writing the next data is then the time to transfer one byte. In that case, the master must discard the first received data.

23.7.6 I^2C pulse missed

When the I^2C interface is used for long transmit/receive transactions, the MCU may return a NACK somewhere during the transaction instead of returning an ACK for all data. The received data may also be corrupted. In the master mode the I^2C may not detect an incoming ACK. This is due to a weakness in the noise filter of the I/O pad which in certain conditions may cause the I^2C to miss a pulse.

Workaround

Since the data corruption is caused by noise generated by the CPU, the CPU activity should be minimized during data reception and/or transmission. This is done by performing physical data transmission (master mode) and reception (slave mode) in the WFI state (wait for interrupt).

To allow the device to be woken up from the WFI, I^2C transmission and reception routines must be implemented through interrupt routines instead of polling mechanisms. Receive and transmit interrupts (received data processing) must be triggered only by the BTF bit flag (byte transfer finished) in the I2C_SR1 register. This flag indicates that the I^2C is in the stretched state (data transfers are stretched on the bus).

Clock stretching must be enabled to allow data transfers from the slave to be stopped and to allow the CPU to be woken up to read the received byte.

To recover from possible errors, periodically check if the I^2C does not remain in the busy state for too long (BUSY bit set in I2C_SR3 register). If so, it should be reinitialized.

23.8 I²C low power modes

Table 42. I²C low power mode

Mode	Description
Wait	No effects on the I²C interface. I²C interrupts cause the device to exit from the wait mode.
Halt	**In the slave mode:** communication is reset, except for configuration registers. Device is in the slave mode. Wakeup from the Halt interrupt is generated if the ITEVTEN = 1 and the address matched (including allowed headers). The matched address is not acknowledged in the Halt mode so the master has to send it again when the CPU is woken up to receive an acknowledge. If the NOSTRETCH = 0, the SCLH will be stretched after acknowledge pulse in the Halt mode until the WUFH is cleared by software; none of the flags are set by the address which wakes up the CPU. **In the master mode:** communication is frozen until the CPU is woken up. Wakeup from Halt flag and interrupt are generated if the ITEVTEN = 1 and there is a HALT instruction. ***Note:*** *It is forbidden to enter Halt mode while a communication is ongoing.*

23.9 I²C interrupts

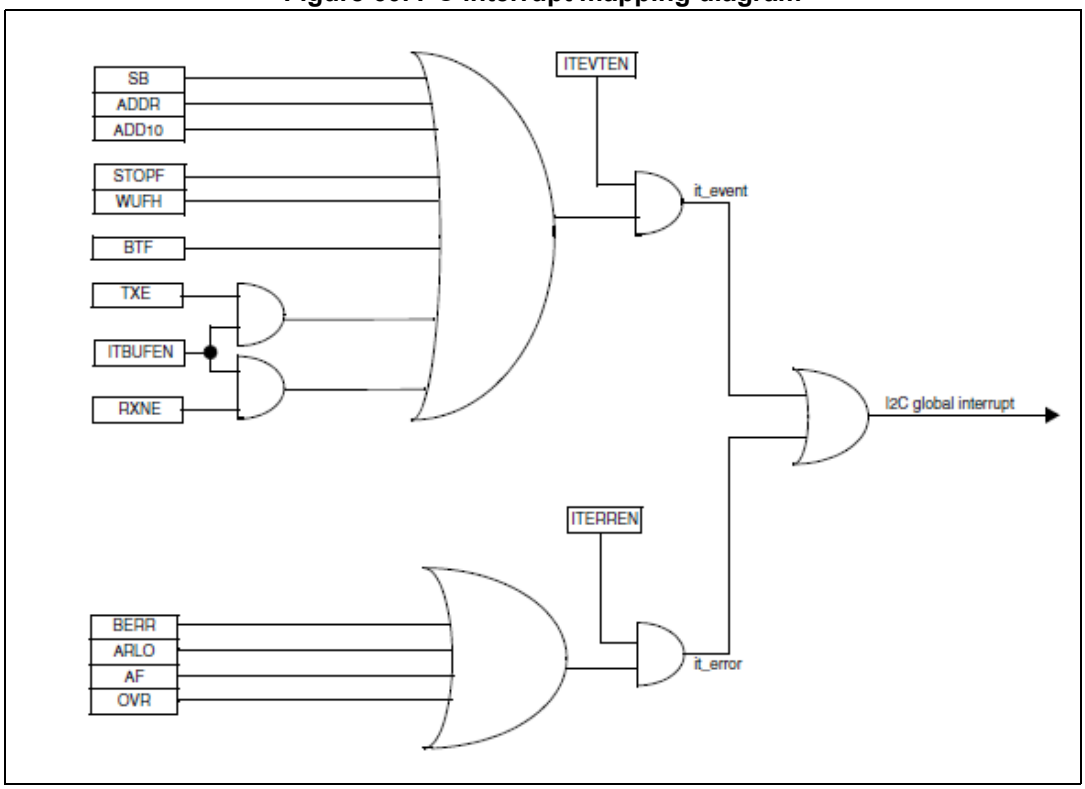
Table 43 summarizes the I²C interrupt events and their capability to exit form the CPU wait or Halt mode.

Table 43. I²C interrupt requests

Interrupt event	Event flag	Enable control bit	Exit from wait	Exit from Halt
Start bit sent (master)	SB	ITEVTEN	Yes	No
Address sent (master) or address matched (slave)	ADDR			
10-bit header sent (master)	ADD10			
Stop received (slave)	STOPF			
Data byte transfer finished	BTF			
Wakeup from Halt	WUFH	ITEVTEN		Yes
Receive buffer not empty	RXNE	ITEVTEN and ITBUFEN		No
Transmit buffer empty	TXE			
Bus error	BERR	ITERREN		
Arbitration loss (master)	ARLO			
Acknowledge failure	AF			
Overrun/underrun	OVR			

Figure 60 shows an outline view of the I2C internal interrupt request.

Figure 60. I2C interrupt mapping diagram



23.10 I2C registers description

23.10.1 I2C_CR1 (control register1)

Offset: 0x00

Default value: 0x00

7	6	5	4	3	2	1	0
NOSTRETCH	ENGC	RFU					PE
r/w	r/w	r					r/w

Bit 0: **PE** peripheral enable

0: peripheral disable.

1: peripheral enable: see device product datasheet for correct configuration of corresponding I/O pins alternate function.

Note: 1. If this bit is reset while a communication is ongoing, the peripheral is disabled at the end of the current communication, when back to the IDLE state.

2. All bit resets due to PE = 0 occur at the end of the communication.

Bit 5-1: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 6: **ENGC** general call enable

0: general call disabled. Address 0x00 is NACKed.

1: general call enabled. Address 0x00 is ACKed.

Bit 7: **NOSTRETCH** clock stretching disable (slave mode)

This bit is used to disable clock stretching in the slave mode when the ADDR or BTF flag is set, until it is reset by software.

0: clock stretch enabled.

1: clock stretch disabled.

23.10.2 I2C_CR2 (control register2)

Offset: 0x01

Default value: 0x00

7	6	5	4	3	2	1	0
SWRST	RFU			POS	ACK	STOP	START
r/w	r			r/w	r/w	r/w	r/w

Bit 0: **START** start generation

This bit is set and cleared by software, cleared by hardware when the START is sent, or PE = 0.

– In master mode:

0: no start generation.

1: repeated start generation.

– In slave mode:

0: no start generation.

1: start generation when the bus is free.

Note: When the STOP or START is set, the user must not perform any write access to the I2C_CR2 before the control bit is cleared by hardware. Otherwise, a second STOP or START request may occur.

Bit 1: **STOP** stop generation.

This bit is set and cleared by software, cleared by hardware when a STOP condition is detected.

– In master mode:

0: no stop generation.

1: stop generation after the current byte transfer or after the current start condition is sent.

– In slave mode:

0: no stop generation.

1: release the SCL and SDA lines after the current byte transfer.

Bit 2: **ACK** acknowledgment enable

This bit is set and cleared by software and cleared by hardware when PE = 0.

0: no acknowledgment returned.

1: acknowledgment returned after a byte is received (matched address or data).

Bit 3: **POS** Acknowledgment position (for data reception).

This bit is set and cleared by software and cleared by hardware when PE = 0.

0: the ACK bit controls the (N)ACK of the current byte being received in the shift register.

1: the ACK bit controls the (N)ACK of the next byte which will be received in the shift register.

Note: The POS bit is used when the procedure for reception of 2 bytes (see Figure 58: Method 2 - transfer sequence diagram for master receiver when N = 2 on page 239) is followed. It must be configured before data reception starts. In this case, to NACK the 2nd byte, the ACK bit must be cleared just after the ADDR is cleared.

Bit 6-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 7: **SWRST** software reset

When active, the I^2C is in the reset state. Before resetting this bit, make sure the I^2C lines are released and the bus is free.

0: the I^2C peripheral not under reset.

1: the I^2C peripheral under the reset state.

Note: This bit can be used in case the BUSY is set to '1' when no stop condition has been detected on the bus.

23.10.3 I2C_FREQR (frequency register)

Offset: 0x02

Default value: 0x00

7	6	5	4	3	2	1	0
RFU		FREQ [5:0]					
r		r/w					

Bit 5-0: **FREQ[5:0]** peripheral clock frequency

Input clock frequency must be programmed to generate correct timings:

The allowed range is between 1 MHz and 50 MHz.

000000: not allowed

000001: 1 MHz

000010: 2 MHz

000011: 3 MHz

…

110010: 50 MHz

Higher values: not allowed

Note: The minimum peripheral frequencies (f_{MASTER}) for respecting the I^2C bus timings are: 1 MHz for the standard mode and 4 MHz for the fast mode.

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

23.10.4 I2C_OARL (own add. LSB register)

Offset: 0x03

Default value: 0x00

7	6	5	4	3	2	1	0
ADD[7:1]							ADD0
r/w							r/w

Bit 0: **ADD0** interface lower address

7-bit addressing mode: don't care.

10-bit addressing mode: bit 0 of the address.

Bit 7-1:**ADD[7:1]** interface address; a common address field for both 7/10-bit addressing modes.

23.10.5 I2C_OARH (own add. MSB register)

Offset: 0x04

Default value: 0x00

7	6	5	4	3	2	1	0
ADDMODE	ADDCONF	RFU			ADD [9:8]		RFU
r/w	r/w	r			r/w		r

Bit 0: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 2-1: **ADD[9:8]** interface upper address

The upper address field in the 10-bit addressing mode.

Bit 5-3: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 6: **ADDCONF** address configuration must be configured and kept at '1'

Bit 7: **ADDMODE** addressing mode (slave mode)

0: the 7 bit slave address (10-bit address not acknowledged).

1: the 10 bit slave address (7-bit address not acknowledged).

23.10.6 I2C_DR (data register)

Offset: 0x06

Default value: 0x00

7	6	5	4	3	2	1	0
DR [7:0]							
r/w							

Bit 7-0: **DR[7:0]** data register

Byte received or to be transmitted to the bus.

– *Transmitter mode:* byte transmission starts automatically when a byte is written in the DR register. A continuous transmit stream can be maintained if the next data to be transmitted is put in the DR once the transmission is started (TxE = 1).

– *Receiver mode:* received byte is copied into the DR (RxNE = 1). A continuous transmit stream can be maintained if the DR is read before the next data are received (RXNE = 1).

Note: 1. In the slave mode, the address is not copied into DR.

2. Write collision is not managed (the DR can be written if the TXE = 0).

3. If an ARLO event occurs on the ACK pulse, the received byte is not copied into the DR and so cannot be read.

23.10.7 I2C_SR1 (status register1)

Offset: 0x07

Default value: 0x00

7	6	5	4	3	2	1	0
TxE	RxNE	RFU	STOPF	ADD10	BTF	ADDR	SB
r	r	r	r	r	r	r	r

Bit 0: **SB** start bit (master mode)(ad)

0: no start condition.

1: start condition generated:

– Set when a start condition is generated.

– Cleared through the software by reading the SR1 register followed by writing the DR register, or by hardware when PE = 0.

Bit 1: **ADDR** address sent (master mode)/matched (slave mode)[ad],[ae]

This bit is cleared by software reading the SR1 register followed reading the SR3, or by hardware when PE = 0.

- Address matched (slave):

0: address mismatched or not received

1: received address matched:

Set by hardware as soon as the receiver slave address matched with the OAR registers content or a general call is recognized (when enabled depending on configuration).

- Address sent (master):

0: no end of address transmission

1: end of address transmission

– For 10-bit addressing, the bit is set after the ACK of the 2nd byte.

– For 7-bit addressing, the bit is set after the ACK of the address byte.

Note: The ADDR is not set after a NACK reception.

Bit 2: **BTF** byte transfer finished[af], [ad]

0: data byte transfer not done

1: data byte transfer succeeded

– Set by hardware when NOSTRETCH = 0.

– In reception when a new byte is received (including the ACK pulse) and the DR has not been read yet (RxNE = 1).

– In transmission when a new byte should be sent and the DR has not been written yet (TxE = 1).

– Cleared by software reading the SR1 followed by either a read or write in the DR register or by hardware after a start or a stop condition in transmission or when PE = 0.

ad. Due to timing constraints, when in the standard mode if the CCR is less than 9 (i.e.: with the peripheral clock below 2 MHz) with $f_{MASTER} = f_{CPU}$ and the event interrupt disabled, the following procedure must be followed: modify the reset sequence in order to insert at least 5 cycles between each operations in the flag clearing sequence. For example, when $f_{MASTER} = f_{CPU}$ = 1 MHz, use the following sequence to poll the SB bit:

```
_label_wait: BTJF I2C_SR1,SB,_label_wait
             NOP
             NOP
             NOP
             NOP
             NOP
             LD A, I2C_SR3; once executed, the SB bit is then cleared.
```

ae. *In the slave mode, it is recommended to perform the complete clearing sequence (READ SR1 then READ SR3) after the ADDR is set.*

af. The BTF bit is not set after a NACK reception, or in case of an ARLO event.

Bit 3: **ADD10** 10-bit header sent (master mode)[ag]

0: no ADD10 event occurred

1: master has sent first the address byte (header)

– Set by hardware when the master has sent the first byte in the 10-bit address mode.

– Cleared by software reading the SR1 register followed by a write in the DR register of the second address byte, or by the hardware when PE = 0.

Bit 4: **STOPF** stop detection (slave mode)[ah], [ai]

0: no stop condition detected

1: stop condition detected

– Set by hardware when a stop condition is detected on the bus by the slave after an acknowledgment (if ACK = 1).

– Cleared by software reading the SR1 register followed by a write in the CR2 register, or by hardware when PE = 0.

Bit 5: **RFU** reserved for future use

Bit 6: **RxNE** data register not empty (receivers)[aj], [ak]

0: data register empty

1: data register not empty

– Set when the data register is not empty in the receiver mode. The RxNE is not set during the address phase.

– Cleared by software reading or writing the DR register or by hardware when PE = 0.

Note: The RXE cannot be cleared by reading data when the BTF bit is set as the DR register is still full in this case.

Bit 7: **TxE** data register empty (transmitters)[al]

0: data register not empty.

1: data register empty.

– Set when the DR is empty in transmission. The TxE is not set during the address phase.

– Cleared by software writing to the DR register or by hardware after a start or a stop condition or when PE = 0.

ag. The ADD10 bit is not set after a NACK reception.

ah. The STOPF bit is not set after a NACK reception.

ai. It is recommended to perform the complete clearing sequence (READ SR1 then WRITE CR2) after the STOPF is set.

aj. The interrupt will be generated when the shift register is copied into the DR after an ACK pulse.

ak. The RXNE is not set in case of an ARLO event.

al. The interrupt will be generated when the DR is copied into the shift register after an ACK pulse. If a NACK is received, copy is not done and the TXE is not set.

23.10.8 I2C_SR2 (status register2)

Offset: 0x08

Default value: 0x00

7	6	5	4	3	2	1	0
RFU		WUFH	RFU	OVR	AF	ARLO	BERR
r		rc/w0	r	rc/w0	rc/w0	rc/w0	rc/w0

Bit 0: **BERR** bus error

0: no misplaced start or stop condition.

1: misplaced start or stop condition.

– Set by hardware when the interface detects a misplaced start or stop condition.

– Cleared by software writing 0, or by hardware when PE = 0.

Bit 1: **ARLO** arbitration lost (master mode)

0: no arbitration lost detected.

1: arbitration lost detected.

– Set by hardware when the interfaces lose the arbitration of the bus to another master on in the SMBUS mode when the ENARP = 1 and the slave loses the arbitration.

– Cleared by software writing 0, or by hardware when PE = 0.

After an ARLO event the interface switches back automatically to the slave mode (M/SL = 0).

Bit 2: **AF** acknowledgment failure

0: no acknowledgment failure.

1: acknowledgment failure.

– Set by hardware when no acknowledgment is returned.

– Cleared by software writing 0, or by hardware when PE = 0.

Bit 3: **OVR** overrun (underrun)

0: no overrun/underrun

1: overrun/underrun

– Set by hardware in the slave mode when the NOSTRETCH = 1 and:

– In reception when a new byte is received (including the ACK pulse) and the DR register has not been read yet. A new received byte is lost.

– In transmission when a new byte should be sent and the DR register has not been written yet. The same byte is sent twice.

– Cleared by software writing 0, or by hardware when PE = 0.

Note: If the DR write occurs very close to the SCL rising edge, the sent data are unspecified and a hold timing error occurs.

Bit 4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 5: **WUFH** wakeup from Halt

0: no wakeup from the HALT mode

1: the 7-bit address or header match in the HALT mode (slave mode) or the HALT entered when in the master mode.

Note: 1. This bit is set asynchronously in the slave mode (during the HALT mode).

2. This bit is set only if the ITEVTEN = 1 and cleared by software, or by hardware when the PE = 0.

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

23.10.9 I2C_SR3 (status register3)

Offset: 0x09

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	GENCALL			RFU	TRA	BUSY	MSL
r	r			r	r	r	r

Bit 0: **MSL** master/slave

0: slave mode

1: master mode

– Set by hardware as soon as the interface is in the master mode (SB = 1).

– Cleared by hardware after detecting a stop condition on the bus or loss arbitration (ARLO = 1), or by hardware when PE = 0.

Note: Reading the I2C_SR3 after reading the I2C_SR1 clears the ADDR flag, even if the ADDR flag was set after reading the I2C_SR1. Consequently, the I2C_SR3 must be read only when the ADDR is found set in the I2C_SR1 or when the STOPF bit is cleared.

Bit 1: **BUSY** bus busy

0: no communication on the bus.

1: communication ongoing on the bus.

Set by hardware on detection of the SDA or the SCL low.

Cleared by hardware on detection of a stop condition.

It indicates a communication in progress on the bus. This information is still updated when the interface is disabled (PE = 0).

Bit 2: **TRA** transmitter/receiver

0: data bytes received.

1: data bytes transmitted.

This bit is set depending on the R/W bit of the address byte, at the end of the total address phase.

It is also cleared by hardware after detection of stop condition (STOPF = 1), repeated start condition, loss of bus arbitration (ARLO = 1), or when the PE = 0.

Bit 3: **RFU** reserved for future use

Bit 4: **GENCALL** general call header (slave mode)

Cleared by hardware after a stop condition or repeated start condition, or when the PE = 0.

0: no general call.

1: general call header received when ENGC = 1.

Bit 7-5: **RFU** reserved; must be kept 0 during register writing for future compatibility.

23.10.10 I2C_ITR (interrupt register)

Offset: 0x0A

Default value: 0x00

7	6	5	4	3	2	1	0
RFU					ITBUFEN	ITEVTEN	ITERREN
r					r/w	r/w	r/w

Bit 0: **ITERREN** error interrupt enable

0: error interrupt disabled.

1: error interrupt enabled.

This interrupt is generated in case of the following events:

- BERR = 1
- ARLO = 1
- AF = 1
- OVR = 1

Bit 1: **ITEVTEN** event interrupt enable

0: event interrupt disabled.

1: event interrupt enabled.

This interrupt is generated in case of the following events:

- SB = 1 (master)
- ADDR = 1 (master/slave)
- ADD10 = 1 (master)
- STOPF = 1(slave)
- BTF = 1 with no TxE or RxNE event
- TxE event to 1 if ITBUFEN = 1
- RxNE event to 1 if ITBUFEN = 1
- WUFH = 1 (asynchronous interrupt to wake up from Halt)

Bit 2: **ITBUFEN** buffer interrupt enable

0: interrupt disable (TxE and RxNE are masked).

1: interrupt enable in case of TxE = 1 or RxNE = 1 events.

Bit 7-3:**RFU** reserved; must be kept 0 during register writing for future compatibility.

23.10.11 I2C_CCRL (clock control low)

Offset: 0x0B

Default value: 0x00

7	6	5	4	3	2	1	0
CCR [7:0]							
r/w							

Bit 7-0: **CCR[7:0]** clock control register (master mode)

Controls the SCLH clock in master mode

• Standard mode:

Period (I^2C) = 2 * CCR * t_{MASTER}

t_{HIGH} = CCR * t_{MASTER}

t_{LOW} = CCR * t_{MASTER}

• Fast mode:

If DUTY = 0:

Period (I^2C) = 3* CCR * t_{MASTER}

t_{HIGH} = CCR * t_{MASTER}

t_{LOW} = 2 * CCR * t_{MASTER}

If DUTY = 1: (to reach 400 kHz)

Period (I^2C) = 25 * CCR * t_{MASTER}

t_{HIGH} = 9 * CCR * t_{MASTER}

t_{LOW} = 16 * CCR * t_{MASTER}

Note: *t_{CK} = 1 / f_{MASTER}: where f_{MASTER} is the input clock to the peripheral configured using the clock control register.*

The minimum allowed value is 0x04, except in the FAST DUTY mode, where the minimum allowed is 0x01.

thigh = tr(SCL) + tw(SCLH). See product datasheet for the definitions of parameters.

tlow = tf(SCL) + tw(SCLL). See product datasheet for the definitions of parameters.

I^2C communication speed, Fscl = 1 / (t_{HIGH} + t_{LOW}).

23.10.12 I2C_CCRH (clock control high)

Offset: 0x0C

Default value: 0x00

7	6	5	4	3	2	1	0
F/S	DUTY	RFU		CCR [11:8]			
r/w	r/w	r		r/w			

Bit 3-0: **CCR[11:8]** clock control register in the fast/standard mode (master mode)

Controls the SCLH clock in the master mode.

- Standard mode:

  Period (I^2C) = 2 * CCR * t_{MASTER}

  t_{HIGH} = CCR * t_{MASTER}

  t_{LOW} = CCR * t_{MASTER}

- Fast mode:

If DUTY = 0:

Period (I^2C) = 3 * CCR * t_{MASTER}

t_{HIGH} = CCR * t_{MASTER}

t_{LOW} = 2 * CCR * t_{MASTER}

If DUTY = 1: (to reach 400 kHz)

Period (I^2C) = 25 * CCR * t_{MASTER}

t_{HIGH} = 9 * CCR * t_{MASTER}

t_{LOW} = 16 * CCR * t_{MASTER}

For instance: in the standard mode, to generate a 100 kHz SCL frequency:

If FREQR = 08, t_{MASTER} = 125 ns, so CCR must be programmed with 0x28

(0x28 → 40d x 125 ns = 5000 ns).

Note: thigh = tr(SCL) + tw(SCLH). See the product datasheet for the definitions of parameters.

tlow = tf(SCL) + tw(SCLL). See the product datasheet for the definitions of parameters.

The CCR registers must be configured only when the I^2C is disabled (PE = 0).

f_{MASTER} = multiple of 10 MHz is required to generate the fast clock at 400 kHz.

$f_{MASTER} \geq$ 1 MHz is required to generate the standard clock at 100 kHz.

Bit 5-4 **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 6: **DUTY** fast mode duty cycle

0: fast mode t_{LOW} / t_{HIGH} = 2

1: fast mode t_{LOW} / t_{HIGH} = 16/9 (see CCR)

Bit 7: **F/S I^2C** master mode selection

0: standard mode I^2C

1: fast mode I^2C

23.10.13 I2C_TRISER (time rise register)

Offset: 0x0D

Default value: 0x02

7	6	5	4	3	2	1	0
RFU		TRISE [5:0]					
r		r/w					

Bit 5-0: **TRISE[5:0]** maximum rise time in the fast/standard mode (master mode)

These bits must be programmed with the maximum SCL rise time given in the I^2C bus specification, incremented by 1.

For instance: in the standard mode, the maximum allowed SCL rise time is 1000 ns. If the value in the I2CFREQR register = 0x08, then t_{MASTER} = 125 ns, therefore the TRISE bits must be programmed with 0x09.

(1000 ns / 125 ns = 8 + 1).

The filter value can also be added to the TRISE[5:0] if the result is not an integer, the TRISE[5:0] must be programmed with the integer part, in order to respect the t_{HIGH} parameter.

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Note: The TRISE[5:0] must be configured only when the I^2C is disabled (PE = 0).

23.11 I2C registers overview

Table 44 details the I2C internal registers starting from the base address specified on the corresponding device datasheet (DS); for detailed registers description refer to *Section 23.10*.

Table 44. I2C internal registers overview

Name	Description	Offset	Type	Reset value
I2C_CR1	Control register1	0x00	R/W	0x00
I2C_CR2	Control register2	0x01	R/W	0x00
I2C_FREQ	Control frequency register	0x02	R/W	0x00
I2C_OARL	Own address register LSB	0x03	R/W	0x00
I2C_OARH	Own address register MSB	0x04	R/W	0x00
RFU	Reserved	0x05		
I2C_DR	Data register	0x06	R/W	0x00
I2C_SR1	Status register1	0x07	R	0x00
I2C_SR2	Status register2	0x08	R/WC	0x00
I2C_SR3	Status register3	0x09	R	0x00
I2C_ITR	Interrupt register	0x0A	R/W	0x00
I2C_CCRL	Clock control register low	0x0B	R/W	0x00
I2C_CCRH	Clock control register high	0x0C	R/W	0x00
I2C_TRISER	Time rise register	0x0D	R/W	0x02

24 Universal asynchronous receiver transmitter (UART)

24.1 UART main features

The following section describes the UART main features:

- SW flow control operating mode
- Full duplex, asynchronous communications
- High precision baud rate generator system
  - Common programmable transmit and receive baud rates up to $f_{MASTER}/16$
- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits
- Separate enable bits for transmitter and receiver
- Transfer detection flags:
  - Receive buffer full
  - Transmit buffer empty
  - End of transmission flags
- Parity control:
  - Transmits parity bit
  - Checks parity of received data byte
- 4 error detection flags:
  - Overrun error
  - Noise error
  - Frame error
  - Parity error
- 6 interrupt sources with flags:
  - Transmit data register empty
  - Transmission complete
  - Receive data register full
  - Idle line received
  - Overrun error
  - Parity error
- 2 interrupt vectors:
  - Transmitter interrupt
  - Receiver interrupt
- Reduced power consumption mode
- Wakeup from the mute mode (by idle line detection or address mark detection)
- 2 receiver wakeup modes:
  - Address bit (MSB)
  - Idle line

24.2 UART functional description

The UART interface is based on two primary signals:

UART_RX is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

UART_TX is the serial data output. When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and there is no transmission, this pin stays at a high level.

The UART_TX and UART_RX lines are alternate multifunction signals assigned to predefined device pins through the I/O multiplex mechanism described in the product datasheet.

Through these pins, serial data are transmitted and received in the normal UART mode as frames comprising:

- An idle line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 1 or 2 stop bits indicating that the frame is complete

The UART logic interface uses the following internal group of registers:

- A status register (UART_SR)
- Control registers (UART_CRx)
- A data register (UART_DR)
- A 16-bit baud rate prescaler (UART_BRR)

24.2.1 UART character description

Word length may be selected as being either 8 or 9 bits by programming the M bit in the UART_CR1 register.

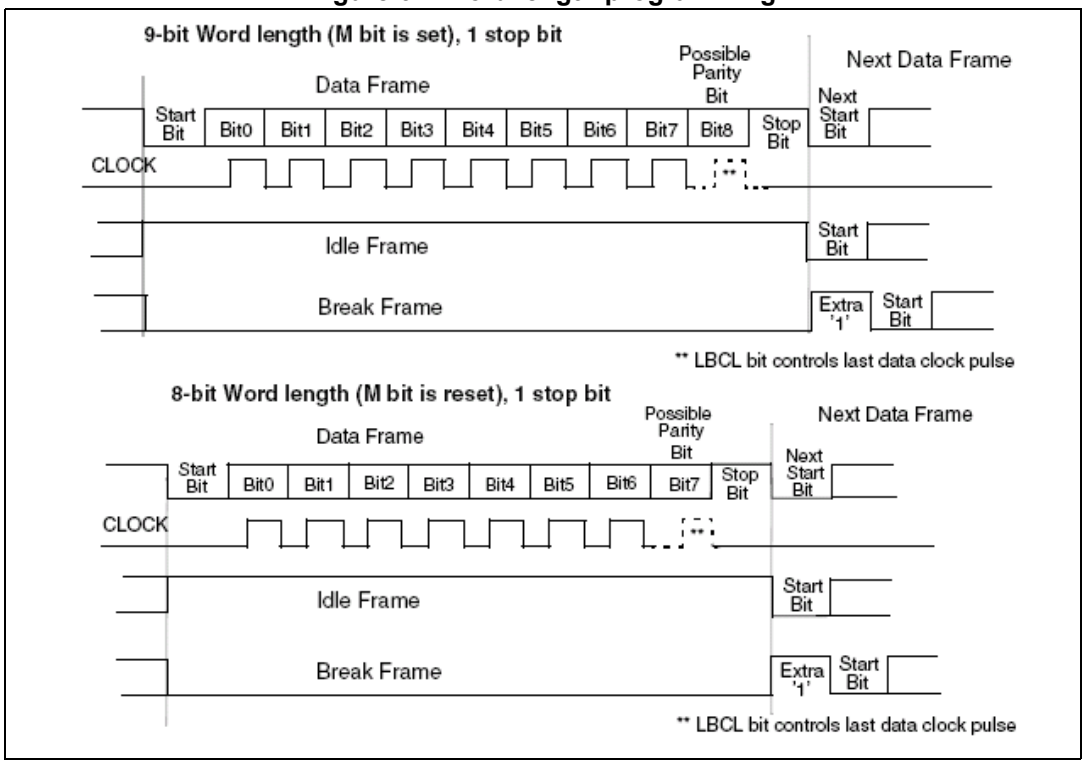
The UART_TX pin is in the low state during the start bit. It is in the high state during the stop bit.

An ***idle character*** is interpreted as an entire frame of 1's (the number of 1's includes the start bit, the number of data bits and the number of stop bits).

A ***break character*** is interpreted on receiving 0's for a frame period. At the end of the break frame the transmitter inserts either 1 or 2 stop bits (logic “1” bit) to acknowledge the start bit.

Transmission and reception are driven by a common baud rate generator; the clock is generated when the enable bit is set respectively for the transmitter and receiver.

Figure 61. Word length programming



24.2.2 UART transmitter

The transmitter can send data words of either 8 or 9 bits depending on the M bit status. When the M bit is set, word length is 9 bits and the 9th bit (the MSB) is stored in the T8 bit in the UART_CR1 register.

When the transmit enable (TEN) bit of the UART_CR2 register is set, the data in the transmit shift register is the output on the UART_TX pin.

Character transmission

During an UART transmission, data shift out the least significant bit first on the UART_TX pin. In this mode, the UART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register.

Every character is preceded by a start bit which is a logic level low for one bit period. The character is terminated by a configurable number of stop bits.

Note: 1. The TEN bit should not be reset during transmission of data. Resetting the TEN bit during the transmission will corrupt the data on the UART_TX pin as the baud rate counters will get frozen. The current data being transmitted will be lost.

2. An idle frame will be sent after the TEN bit is enabled.

Configurable stop bits

The number of stop bits to be transmitted with every character can be programmed in the control register 3, bits 5, 4.

- 1 stop bit: This is the default value of number of stop bits.
- 2 stop bits.

An idle frame transmission will include the stop bits.

A break transmission consists of 10 low bits followed by the configured number of stop bits (when m = 0) and 11 low bits followed by the configured number of stop bits (when m = 1). It is not possible to transmit long breaks (break of length greater than 10/11 low bits).

TX firmware procedure:

1. Program the M bit in the UART_CR1 to define the word length.
2. Program the number of stop bits in the UART_CR3.
3. Select the desired baud rate by programming the baud rate registers in the following order:
   a) UART_BRR2
   b) UART_BRR1
4. Set the TEN bit in the UART_CR2 to enable the transmitter mode.
5. Write the data to send in the UART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted.
6. Once the last data are written to the UART_DR register, wait until the TC bit of the UART_SR register is set to '1', which means that the last data transmission is completed. This last step is required, for instance, to avoid the last data transmission corruption when disabling the UART or entering the Halt mode.

Single byte communication

Clearing the TXE bit is always performed by a write to the data register.

The TXE bit is set by hardware and it indicates:

- The data has been moved from the TDR to the shift register and the data transmission has started.
- The TDR register is empty.
- The next data can be written in the UART_DR register without overwriting the previous data.

This flag generates an interrupt if the TIEN bit is set.

While transmission is taking place, a write instruction to the UART_DR register stores the data in the TDR register. The data are copied in the shift register at the end of the current transmission.

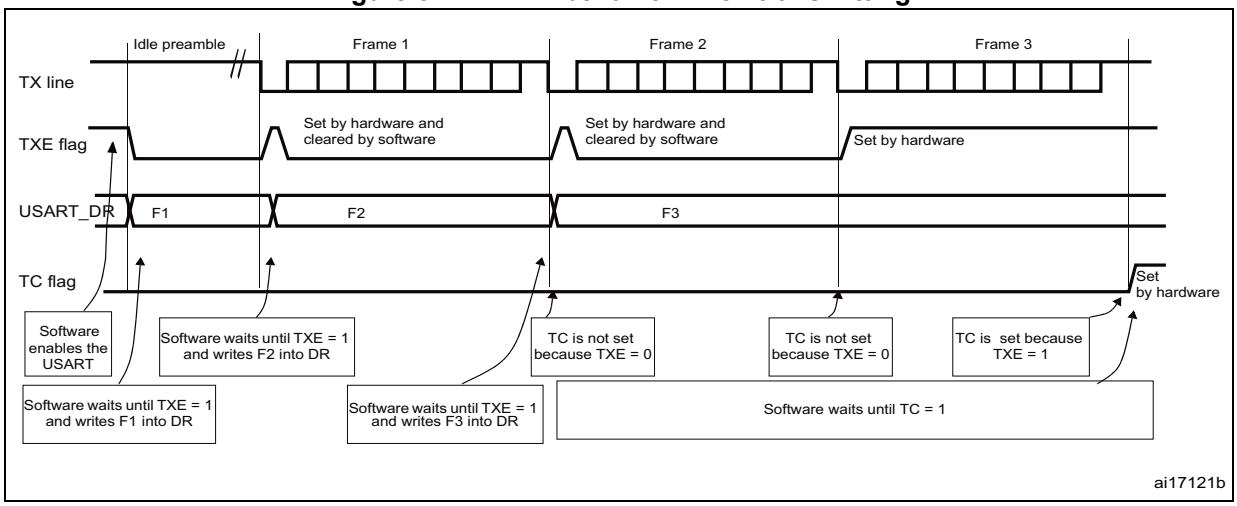
When no transmission is taking place, a write instruction to the UART_DR register places the data directly in the shift register, the data transmission starts, and the TXE bit is immediately set.

If a frame transmission is complete (after the stop bit) and the TXE bit is set, the TC bit is set. An interrupt is generated if the TCIEN is set in the UART_CR2 register. After writing the last data in the UART_DR register, it is mandatory to wait until the TC is set to '1' before entering the Halt mode or disabling the UART (see *Figure 62*).

The following software sequence is required to clear the TC bit:

- Read from the UART_SR register.
- Write to the UART_DR register.

Figure 62. TC/TXE behavior when transmitting



Note: The TC bit can also be cleared by writing a '0' to it.

Break character

Setting the SBK bit transmits a break character. The break frame length depends on the M bit (see *Figure 61*).

If the SBK bit is set to '1' a break character is sent on the UART_TX line after completing the current character transmission. This bit is reset by hardware when the break character is completed (during the stop bit of the break character). The UART inserts a logic 1 bit at the end of the last break frame to guarantee the recognition of the start bit of the next frame.

Note: 1. The break character is sent without taking into account the number of stop bits. If the UART is programmed with 2 stop bits, the TX line is pulled low until the end of the first stop bit only. Then 2 logic 1 bits are inserted before the next character.

2. If the software resets the SBK bit before the start of break transmission, the break character is not transmitted. For two consecutive breaks, the SBK bit should be set after the stop bit of the previous break.

Idle character

Setting the TEN bit drives the UART to send an idle frame before the first data frame.

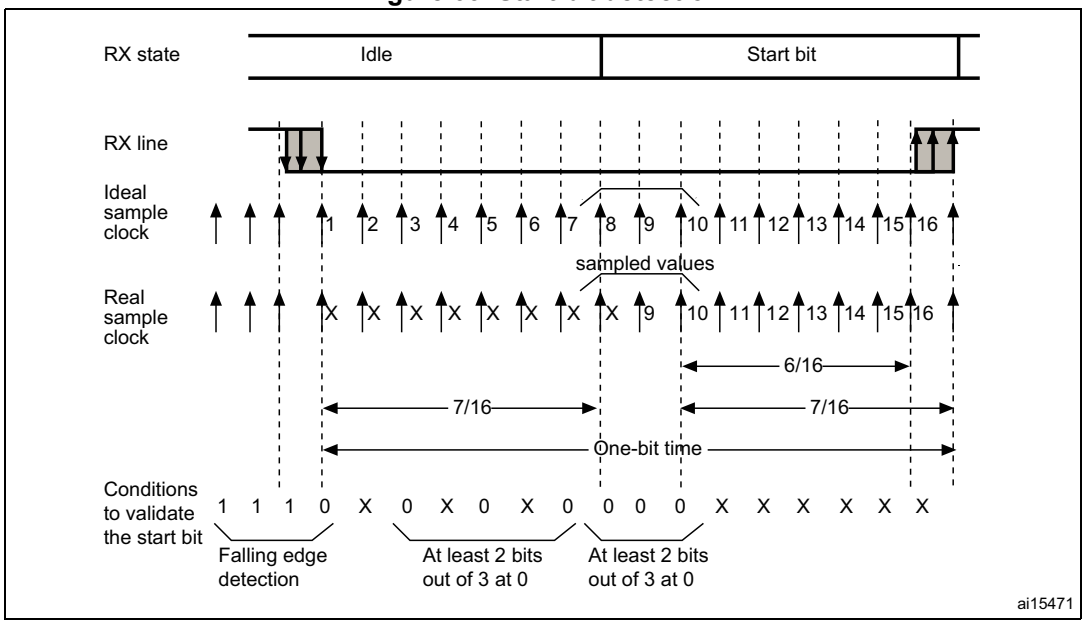
24.2.3 UART receiver

The UART can receive data words of either 8 or 9 bits. When the M bit is set, word length is 9 bits and the MSB is stored in the R8 bit in the UART_CR1 register.

Start bit detection

In the UART, the start bit is detected when a specific sequence of samples is recognized. The start bit detection sequence shown in *Figure 63*.

Figure 63. Start bit detection



Note: If the sequence is not complete, the start bit detection aborts and the receiver returns to the idle state (no flag is set), where it waits for a falling edge.

If only 2 out of the 3 bits are at 0 (sampling on the 3rd, 5th and 7th bits or sampling on the 8th, 9th and 10th bits), the start bit is validated but the NF noise flag bit is set.

The start bit is confirmed if the last 3 samples bits are at 0 (sampling on the 8th, 9th, and 10th bits).

Character reception

During a UART reception, data shift in the least significant bit first through the UART_RX pin. In this mode, the UART_DR register consists of a buffer (RDR) between the internal bus and the received shift register.

RX firmware procedure

1. Program the M bit in the UART_CR1 to define the word length.
2. Program the number of stop bits in the UART_CR3.
3. Select the desired baud rate by programming the baud rate registers in the following order:
   a) UART_BRR2
   b) UART_BRR1
4. Set the REN bit UART_CR2. This enables the receiver which begins searching for a start bit.

When a character is received:

- The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR.
- An interrupt is generated if the RIEN bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
- Clearing the RXNE bit is performed by software reading the UART_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

Note: The REN bit should be not reset while receiving data. If the REN bit is disabled during reception, the reception of the current byte will be aborted.

Break character

When a break character is received, the UART handles it as a framing error.

Idle character

When an idle frame is detected, there is the same procedure as a data received character plus an interrupt if the ILIEN bit is set.

Overrun error

An overrun error occurs when a character is received when the RXNE has not been reset. Data cannot be transferred from the shift register to the RDR register until the RXNE bit is cleared.

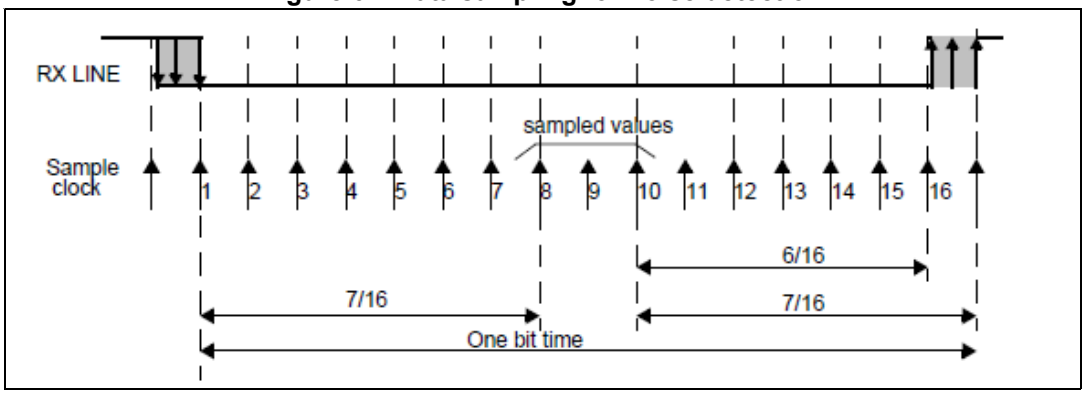
When an overrun error occurs:

- The OR bit is set.
- The RDR content will not be lost. The previous data are available when a read to the UART_DR is performed.
- The shift register will be overwritten. The second data received during the overrun is lost.
- An interrupt is generated if the RIEN bit is set.
- The OR bit is reset by a read to the UART_SR register followed by a UART_DR register read operation.

Noise error

Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

Figure 64. Data sampling for noise detection



Note: The sample clock frequency is 16 x the baud rate.

Table 45. Noise detection from sampled data

Sampled value	NF status	Received bit value	Data validity
000	0	0	Valid
001	1	0	Not valid
010	1	0	Not valid
011	1	1	Not valid
100	1	0	Not valid
101	1	1	Not valid
110	1	1	Not valid
111	0	1	Valid

When noise is detected in a frame:

- The NF is set at the rising edge of the RXNE bit.
- The invalid data are transferred from the shift register to the UART_DR register.

This bit rises at the same time as the RXNE bit which generates an interrupt. The NF bit is reset by a UART_SR register read operation followed by a UART_DR register read operation.

Framing error

A framing error is detected when:

The stop bit is not recognized on reception at the expected time, following either a desynchronization or excessive noise.

When the framing error is detected:

- The FE bit is set by hardware
- The invalid data are transferred from the shift register to the UART_DR register.
- No interrupt is generated in case of single byte communication. However, this bit rises at the same time as the RXNE bit which itself generates an interrupt.

The FE bit is reset by a UART_SR register read operation followed by a UART_DR register read operation.

Configurable stop bits during reception:

The number of stop bits to be received can be configured through the control bits of the UART_CR3 register to the following possible values:

1. Stop bit: default value. Sampling for 1 stop bit is done on the 8th, 9th and 10th samples.
2. Stop bits: sampling for 2 stop bits is done on the 8th, 9th and 10th samples of the first stop bit. If a framing error is detected during the first stop bit, the framing error flag will be set. The second stop bit is not checked for framing error. The RXNE flag will be set at the end of the first stop bit.

Baud rate generator

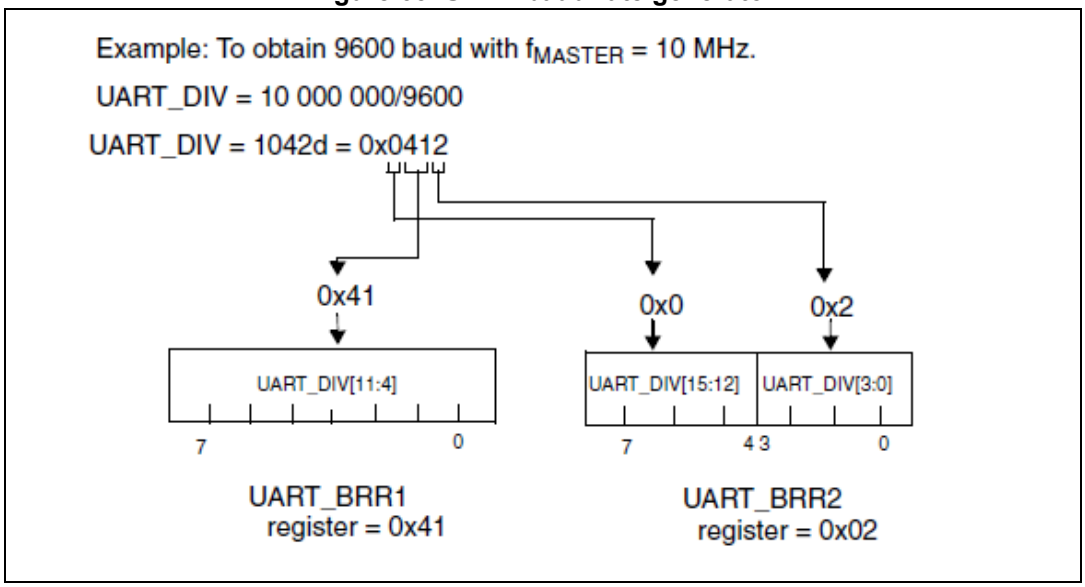
The receiver and transmitter (Rx and Tx) are both set to the same baud rate register, programmed by a 16-bit divider UART_DIV according to the following formula:

Equation 24

$$\text{Tx/ Rx baud rate} = \frac{f_{MASTER}}{UART\_DIV}$$

The UART_DIV baud rate divider is an unsigned integer, coded in the UART_BRR1 and UART_BRR2 registers as shown in *Figure 65*.

Figure 65. UART baud rate generator



Note: The baud counters will be updated with the new value of the baud registers after a write to the BRR1. Hence the baud register value should not be changed during a transaction. The UART_BRR2 should be programmed before the UART_BRR1 register.

The UART_DIV must be greater than or equal to 16d.

Table 46. Baud rate programming and error calculation

Baud rate in kbps	f_{MASTER} = 10 MHz(1)					f_{MASTER} = 16 MHz(1)				
	Actual	%error(2)	UART_DIV	BRR1	BRR2	Actual	%error(2)	UART_DIV	BRR1	BRR2
2.4	2.399	-0.008%	0x1047	0x04	0x17	2.399	-0.005%	0x1A0B	0xA0	0x1B
9.6	9.596	-0.032%	0x0412	0x41	0x02	9.598	-0.020%	0x0693	0x68h	0x03
19.2	19.193	-0.032%	0x0209	0x20	0x09	19.208	-0.040%	0x0341	0x34	0x01
57.6	57.471	-0.224%	0x00AE	0x0A	0x0E	57.554	-0.080%	0x0116	0x11	0x06
115.2	114.942	-0.224%	0x0057	0x05	0x07	115.108	-0.080%	0x008B	0x08	0x0B
230.4	232.558	-0.937%	0x002B	0x02	0x0B	231.884	-0.644%	0x0045	0x04	0x05
460.8	454.545	-1.358%	0x0016	0x01	0x06	457.143	-0.794%	0x0023	0x02	0x03
921.6	NA	NA	NA	NA	NA	941.176	-2.124%	0x0011	0x01	0x01

1. The lower the f_{MASTER} frequency, the lower will be the accuracy for a particular baud rate. The upper limit of the achievable baud rate is shown in the table.
2. Error % = the (calculated - desired) baud rate / the desired baud rate.

24.2.4 Parity control

Parity control (generation of the parity bit in transmission and parity checking in reception) can be enabled by setting the PCEN bit in the UART_CR1 register. Depending on the frame length defined by the M bit, the possible UART frame formats are as listed in *Table 47*.

Table 47. Frame format

M bit	PCEN bit	UART frame
0	0	| SB | 8 bit data | STB |
0	1	| SB | 7-bit data | PB | STB |
1	0	| SB | 9-bit data | STB |
1	1	| SB | 8-bit data PB | STB |

Note: 1. *Legends: SB: start bit, STB: stop bit, PB: parity bit.*

In case of wakeup by an address mark, the MSB bit of the data is taken into account and not the parity bit.

Even parity:

The parity bit is calculated to obtain an even number of “1s” inside the frame made of the 7 or 8 LSB bits (depending on whether the M is equal to 0 or 1) and the parity bit.

Example 1

Data = 00110101; 4 bits set => the parity bit will be 0 if even parity is selected (the PS bit in the UART_CR1 = 0).

Odd parity:

The parity bit is calculated to obtain an odd number of “1s” inside the frame made of the 7 or 8 LSB bits (depending on whether the M is equal to 0 or 1) and the parity bit.

Example 2

Data = 00110101; 4 bits set => the parity bit will be 1 if odd parity is selected (the PS bit in the UART_CR1 = 1).

Transmission:

If the PCEN bit is set in the UART_CR1 then the MSB bit of the data written in the data register is not transmitted but is changed by the parity bit to give an even number of '1's if even parity is selected (PS = 0) or an odd number of '1's if odd parity is selected (PS = 1).

Reception:

If the parity check fails, the PE flag is set in the UART_SR register and an interrupt is generated if the PIEN bit is set in the UART_CR1 register.

24.2.5 Multi-processor communication

It is possible to perform multi-processor communication with the UART (several UARTs connected in a network). For example, one of the UARTs can be the master and its TX output is connected to the RX input of the other UART. The others are slaves and their respective TX outputs are logically ANDed together and connected to the RX input of the master.

In multi-processor configurations it is often desirable that only the intended message recipient should actively receive the full message contents, thus reducing redundant UART service overhead for all non-addressed receivers.

The non-addressed devices may be placed in the mute mode by means of the muting function.

In mute mode:

- None of the reception status bits can be set.
- All the receive interrupts are inhibited.
- The RWU bit in the UART_CR1 register is set to 1. the RWU can be controlled automatically by hardware or written by the software under certain conditions.

The UART can enter or exit from the mute mode using one of two methods, depending on the WAKE bit in the UART_CR1 register:

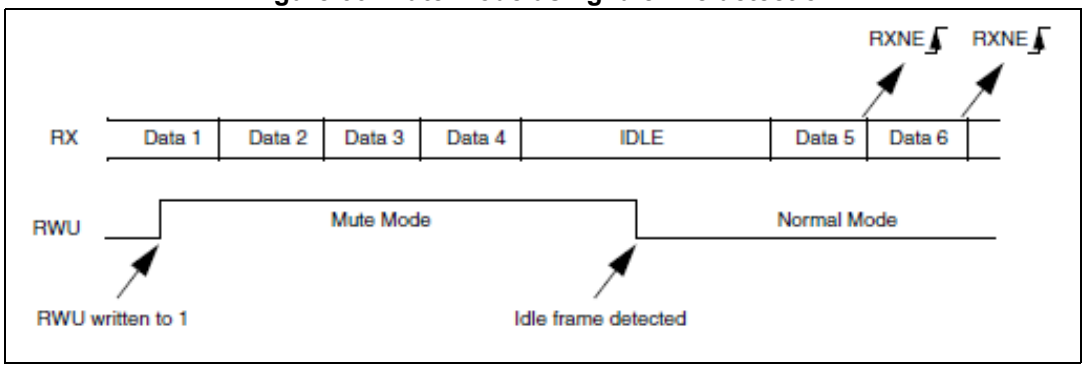
- Idle line detection if the WAKE bit is reset,
- Address mark detection if the WAKE bit is set.

Idle line detection (WAKE = 0)

The UART enters the mute mode when the RWU bit is written to 1. It wakes up when an idle frame is detected. Then the RWU bit is cleared by hardware but the IDLE bit is not set in the UART_SR register. The RWU can also be written to 0 by software.

An example of the mute mode behavior using idle line detection is given in *Figure 66*.

Figure 66. Mute mode using idle line detection



Address mark detection (WAKE = 1)

In this mode, bytes are recognized as addresses if their MSB is a '1', else they are considered as data. In an address byte, the address of the targeted receiver is put on the 4 LSB. This 4-bit word is compared by the receiver with its own address which is programmed in the ADD bits in the UART_CR4 register.

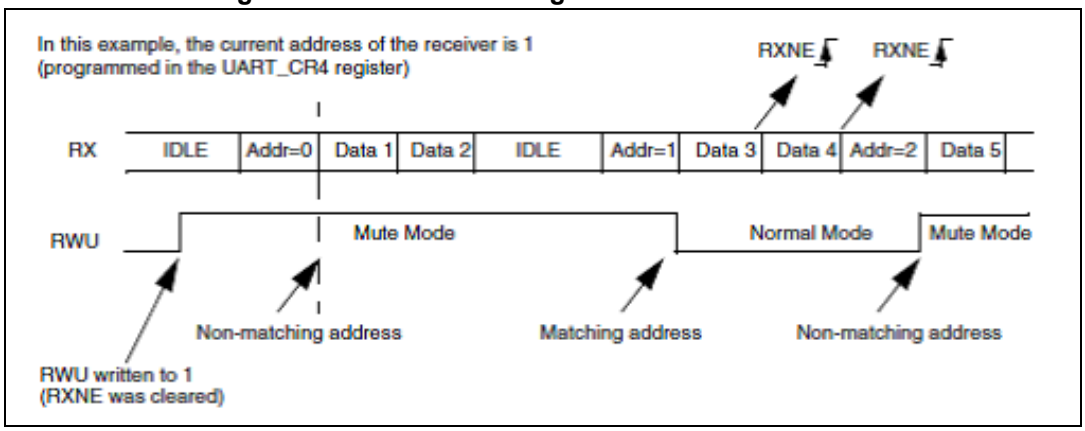
The UART enters the mute mode when an address character is received which does not match its programmed address. The RXNE flag is not set for this address byte and no interrupt request is issued as the UART would have entered the mute mode.

It exits from the mute mode when an address character is received which matches the programmed address. Then the RWU bit is cleared and subsequent bytes are received normally. The RXNE bit is set for the address character since the RWU bit has been cleared.

The RWU bit can be written to 0 or 1 when the receiver buffer contains no data (RXNE = 0 in the UART_SR register). Otherwise the write attempt is ignored.

An example of the mute mode behavior using address mark detection is given in *Figure 67*.

Figure 67. Mute mode using address mark detection



Note: If parity control is enabled, the parity bit remains in the MSB and the address bit is put in the "MSB - 1" bit.

For example, with 7-bit data, the address mode and parity control:

SB I 7-bit data I ADD I PB I STB where:

SB = start bit, STB = stop bit, ADD = address bit, PB = parity bit.

24.3 UART low power mode

Table 48. UART low power mode

UART interface behavior in low power modes	
Mode	**Description**
Wait	No effect on the UART. UART interrupts cause the device to exit from the wait mode.
Halt	UART registers are frozen. In the Halt mode, the UART stops transmitting/receiving until the Halt mode is exited.

24.4 UART interrupts

Table 49. UART interrupt request

Interrupt event	Event flag	Enable control bit	Exit from wait	Exit from Halt
Transmit data register empty	TXE	TIEN	Yes	No
Transmission complete	TC	TCIEN	Yes	No
Received data ready to be read	RXNE	RIEN	Yes	No
Overrun error detected	OR		Yes	No
Idle line detected	IDLE	ILIEN	Yes	No
Parity error	PE	PIEN	Yes	No

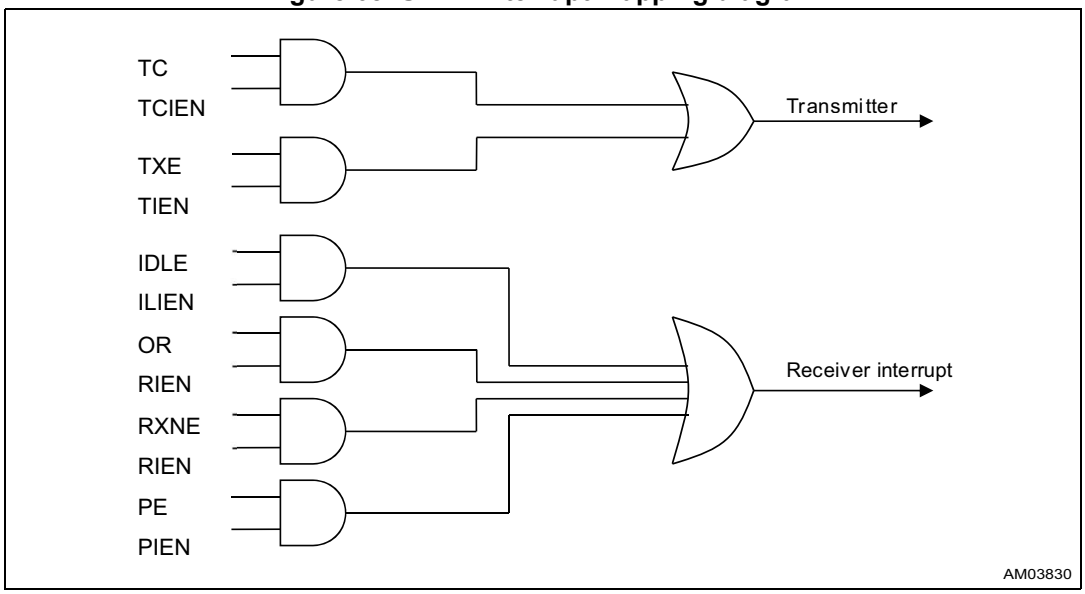
Note: 1. The UART interrupt events are connected to two interrupt vectors (see Figure 68).

a) *Transmission complete or transmit data register empty interrupt.*

b) Idle line detection, overrun error, receive data reg. full, parity error interrupt, and noise flag.

These events generate an interrupt if the corresponding enable control bit is set and the interrupt mask in the CC register is reset (RIM instruction).

Figure 68. UART Interrupt mapping diagram



24.5 UART registers description

24.5.1 UART_SR (status register)

Offset: 0x00

Default value: 0xC0

7	6	5	4	3	2	1	0
TXE	TC	RXNE	IDLE	OR	NF	FE	PE
r	rc_w0	rc_w0	r	r	r	r	r

Bit 0: **PE** parity error

This bit is set by hardware when a parity error occurs in the receiver mode. It is cleared by a software sequence (a read to the status register followed by a read to the UART_DR data register). An interrupt is generated if the PIEN = 1 in the UART_CR1 register.

0: no parity error.

1: parity error.

Bit 1: **FE** framing error[am]

This bit is set by hardware when a desynchronization, excessive noise or a break character is detected. It is cleared by a software sequence (a read to the UART_SR register followed by a read to the UART_DR register).

0: no framing error is detected.

1: framing error or break character is detected.

Bit 2: **NF** noise flag[an]

This bit is set by hardware when noise is detected on a receiver frame. It is cleared by a software sequence (a read to the UART_SR register followed by a read to the UART_DR register).

0: no noise is detected.

1: noise is detected.

Bit 3: **OR** overrun error[ao]

This bit is set by hardware when the word currently being received in the shift register is ready to be transferred into the RDR register while the RXNE = 1. An interrupt is generated if the RIEN = 1 in the UART_CR2 register. It is cleared by a software sequence (a read to the UART_SR register followed by a read to the UART_DR register).

0: no overrun error.

1: overrun error is detected.

Bit 4: **IDLE** IDLE line detected[ap]

This bit is set by hardware when an idle line is detected. An interrupt is generated if the ILIEN = 1 in the UART_CR2 register. It is cleared by a software sequence (a read to the UART_SR register followed by a read to the UART_DR register).

0: no idle line is detected.

1: idle line is detected.

Bit 5: **RXNE** read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the UART_DR register. An interrupt is generated if the RIEN = 1 in the UART_CR2 register. It is cleared by a read to the UART_DR register.

0: data are not received.

1: receiver data are ready to be read.

am. This bit does not generate interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt. If the word currently being transferred causes both frame error and overrun error, it will be transferred and only the OR bit will be set.

an. This bit does not generate interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt.

ao. When this bit is set, the RDR register content will not be lost but the shift register will be overwritten.

ap. The IDLE bit will not be set again until the RXNE bit has been set itself (i.e. a new idle line occurs).

Bit 6: **TC** transmission complete

This bit is set by hardware when transmission of a frame containing data is complete. An interrupt is generated if the TCIEN = 1 in the UART_CR2 register. It is cleared by a software sequence (a read to the UART_SR register followed by a read to the UART_DR register).

0: transmission is not complete.

1: transmission is complete.

Bit 7: **TXE** transmit data register empty

This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TIEN bit = 1 in the UART_CR2 register. It is cleared by a write to the UART_DR register.

0: data are not transferred to the shift register.

1: data are transferred to the shift register.

24.5.2 UART_DR (data register)

Offset: 0x01

Default value: -

7	6	5	4	3	2	1	0
DR [7:0]							
r/w							

Bit 7-0: **DR[7:0]** data value

Contains the received or transmitted data character, depending on whether it is read from or written to.

The data register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR).

The TDR register provides the parallel interface between the internal bus and the output shift register.

The RDR register provides the parallel interface between the input shift register and the internal bus.

24.5.3 UART_BRR1 (baud rate 1 register)

Offset: 0x02

Default value: 0x00

7	6	5	4	3	2	1	0
UART_DIV [11:4]							
r/w							

The baud rate registers are common to both the transmitter and the receiver. The baud rate is programmed using two registers UART_BRR1 and UART_BRR2. Writing of the UART_BRR2 (if required) should precede the UART_BRR1, since a write to the UART_BRR1 will update the baud counters.

Bit 7-0: **UART_DIV[11:4]** LSB mantissa of UART_DIV

These bits define the LSB value of the mantissa of the UART divider (UART_DIV).

The rounded mantissa is:

0x00: UART clock disabled

0x01: 1

0x02: 2

0x03: 3

…

0xFE: 254

0xFF: 255

Note: The baud counters stop counting if the TEN or REN bits are disabled respectively.

24.5.4 UART_BRR2 (baud rate 2 register)

Offset: 0x03

Default value: 0x00

7	6	5	4	3	2	1	0
UART_DIV [15:12]				UART_DIV [3:0]			
r/w				r/w			

Bit 3-0: **UART_DIV[3:0]** fraction bits of UART_DIV

These bits define the value of the fraction of the UART divider (UART_DIV).

The fraction is:

0x0: 0

0x1: 1/16

0x2: 2/16

0x3: 3/16

…

0xE: 14/16

0xF: 15/16

Bit 7-4: **UART_DIV[15:12]** MSB mantissa of UART_DIV

These bits define the MSB value of the mantissa of the UART divider (UART_DIV).

24.5.5 UART_CR1 (control register 1)

Offset: 0x04

Default value: 0x00

7	6	5	4	3	2	1	0
R8	T8	UARTD	M	WAKE	PCEN	PS	PIEN
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 0: **PIEN** parity interrupt enable

This bit is set and cleared by software.

0: parity interrupt disabled.

1: parity interrupt enable; an interrupt is generated whenever the PE = 1 in the UART_SR register.

Bit 1: **PS** parity selection

This bit selects the odd or even parity when the parity generation/detection is enabled (the PCEN bit set). It is set and cleared by software. The parity will be selected after the current byte.

0: even parity.

1: odd parity.

Bit 2: **PCEN** parity control enable

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9^{th} bit if M = 1, 8^{th} bit if M = 0) and parity is checked on the receiver data. This bit is set and cleared by software. Once it is set, the PCEN is active after the current byte (in reception and in transmission).

0: parity control disabled.

1: parity control enabled.

Bit 3: **WAKE** wakeup method

This bit determinates the UART wakeup method; it is set and cleared by software.

0: idle line.

1: address mark.

Bit 4: **M** word length

This bit determinates the word length. It's set and cleared by software.

0: 1 start bit, 8 data bits, n stop bits.

1: 1 start bit, 9 data bits, 1 stop bit.

Note: The M bit must not be modified during a data transfer (both transmission and reception).

Bit 5: **UARTD UART** disable (for low power consumption)

When this bit is set, the UART outputs are stopped at the end of the current byte transfer in order to reduce power consumption. This bit is set and cleared by software.

0: UART enabled.

1: UART prescaler and output disabled.

Bit 6: **T8** transmit data bit 8.

This bit is used to store the 9th bit of the transmitted word when M = 1.

Bit 7: **R8** receive data bit 8.

This bit is used to store the 9th bit of the receiver word when M = 1.

24.5.6 UART_CR2 (control register 2)

Offset: 0x05

Default value: 0x00

7	6	5	4	3	2	1	0
TIEN	TCIEN	RIEN	ILIEN	TEN	REN	RWU	SBK
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 0: **SBK** send break

This bit is used to send break characters. It is set and cleared by software. It should be set by software, and will be reset by hardware during the stop bit of the break.

0: no break character is transmitted.

1: break character is transmitted.

Bit 1: **RWU** receiver wakeup.

This bit determines if the UART is in the mute mode or not. It is set by and cleared by software and can be cleared by hardware when a wakeup sequence is recognized.

0: receiver in active mode.

1: receiver in mute mode.

Note: 1. Before selecting the mute mode (by setting the RWU bit) the UART must first receive a data byte, otherwise it cannot function in the mute mode with wakeup by idle line detection.

2. In address mark detection wakeup configuration (WAKE bit = 1) the RWU bit cannot be modified by software while the RXNE bit is set.

Bit 2: **REN** receiver enable

This bit enables the receiver. It is set and cleared by software.

0: receiver is disabled.

1: receiver is enabled and begins searching for a start bit.

Bit 3: **TEN** transmitter enable

This bit enables the transmitter. It is set and cleared by software.

0: transmitter disable.

1: transmitter is enabled.

Note: 1. During transmission a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word.

2. When the TE is set there is a 1 bit-time delay before the transmission starts.

Bit 4: **ILIEN** IDLE line interrupt enable

This bit is set and cleared by software.

0: interrupt disable.

1: a UART interrupt is generated whenever the IDLE = 1 in the UART_SR register.

Bit 5: **RIEN** receiver interrupt enable

This bit is set and cleared by software.

0: interrupt disable.

1: a UART interrupt is generated whenever the OR = 1 or RXNE = 1 in the UART_SR register.

Bit 6: **TCIEN** transmission complete interrupt enable

This bit is set and cleared by software.

0: interrupt disable.

1: a UART interrupt is generated whenever the TC = 1 in the UART_SR register.

Bit 7: **TIEN** transmitter interrupt enable

This bit is set and cleared by software.

0: interrupt disable.

1: a UART interrupt is generated whenever the TXE = 1 in the UART_SR register.

24.5.7 UART_CR3 (control register 3)

Offset: 0x005236

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	STOP[1:0]		RFU	RFU	RFU	RFU
r	r	r/w		r	r	r	r

Bit 3-0: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 5:4:**STOP[1:0]** clock enable

These bits are used for programming the stop bits.

00: 1 stop bit

01: reserved

10: 2 stop bits

11: reserved

Bit 7-6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

24.5.8 UART_CR4 (control register 4)

Offset: 0x07

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	ADD[3:0]			
r	r	r	r	r/w			

Bit 3-0: **ADD[3:0]** address of UART node

This bit-field gives the address of the UART node.

This is used in multiprocessor communication during the mute mode, for wakeup with address mark detection.

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

24.6 UART registers overview

Table 50 shows the UART internal registers starting from the base address specified on the corresponding device datasheet (DS); for detailed register description refer to *Section 24.5*.

Table 50. UART internal registers overview

Name	Description	Offset	Type	Reset value
UART_SR	Status register	0x00	R_W	0xC0
UART_DR	Data register	0x01	R/W	Undefined
UART_BRR1	Baud rate register1	0x02	R/W	0x00
UART_BRR2	Baud rate register2	0x03	R/W	0x00
UART_CR1	Control register1	0x04	R/W	0x00
UART_CR2	Control register2	0x05	R/W	0x00
UART_CR3	Control register3	0x06	R/W	0x00
UART_CR4	Control register4	0x07	R/W	0x00

25 Digital addressable lighting interface (DALI)

The DALI (“Digital Addressable Lighting Interface”), standardized as IEC 62386, is the standard interface for lighting controls solutions defined by the lighting industry.

The DALI communication module (DCM) is a serial communication circuit designed for controllable electronic ballasts. Ballast is the common name for circuit topologies used to provide the required starting voltage and operating current for fluorescent, mercury or other electronic-discharge lamps.

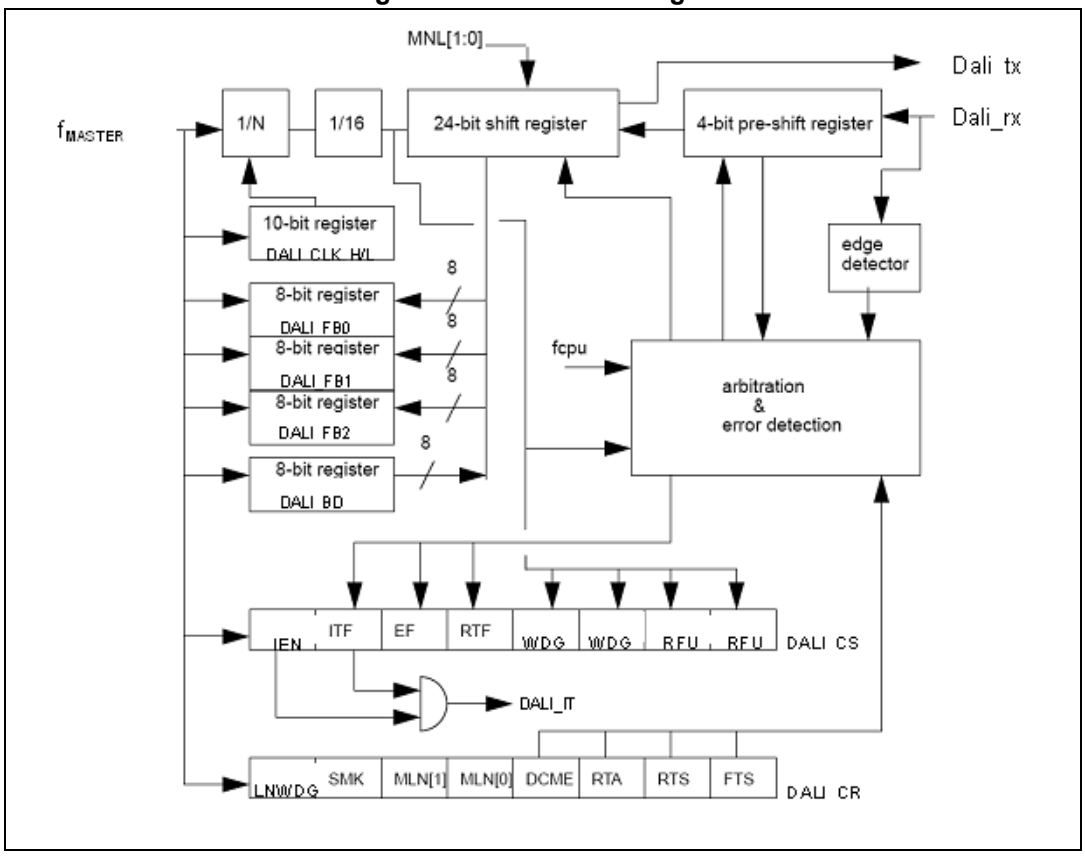
25.1 DALI main features

- 8-bit forward address register for addressing
- Interoperability with different forward message length: 16, 17, 18 and 24 bits
- 8-bit forward and backward data registers
- 1.2, 2.4 and 4.8 kHz speed line rate ± 10%
- Bi-directional communications type
- Monitor receiver line timeout 500 ms ± 10%
- Polarity configurable on DALI_rx, DALI_tx signal lines
- Configurable noise rejection filter on DALI_rx input line
- DALI peripheral clock is slowdown to 153.6 kHz in low-speed mode
- Maskable interrupt

25.2 DALI block diagram

Figure 69 shows the block description of DALI communication module.

Figure 69. DALI block diagram



25.3 DALI functional overview

The DALI protocol uses the bi-phase Manchester asynchronous serial data format. All the bits of the frame are bi-phase encoded except the two stop bits. The default transmission rate is about 1.2 kHz. The bi-phase bit period is 833.33 µs ± 10%.

A forward frame consists of 19, 20, 21 or 27 bi-phase encoded bits, depending on message length configuration:

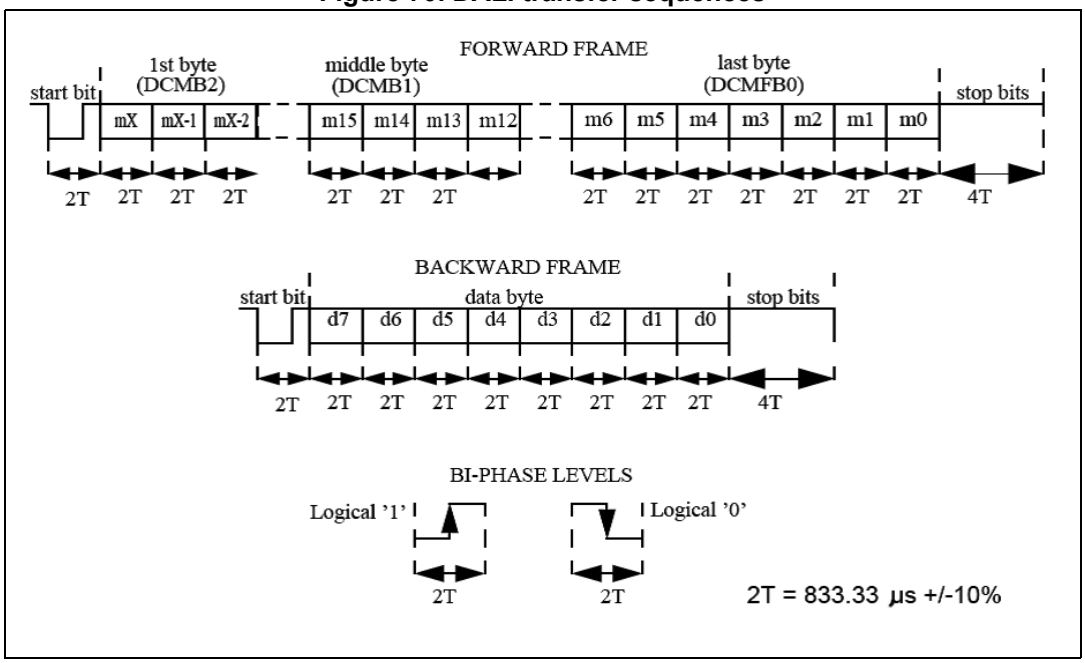
- 1 start bit (0->1: logical '1')
- A message body that takes 16, 17, 18 or 24 bits depending on message length
- 2 high level stop bits (no change of the phase)

A backward frame consists of 11 bi-phase encoded bits:

- 1 start bit (0->1: logical '1')
- 1 data byte (8-bit data)
- 2 high level stop bits (no change of the phase).

An outline picture of the DALI frame format and transfer sequences is shown in *Figure 70*.

Figure 70. DALI transfer sequences



A forward frame consists of a 1 bi-phase encoded start bit (logical '1') and a message body; the message body could be 16, 17, 18 or 24 bi-phase encoded bits depending on the programmed length. The frame is terminated by 2 stop bits (idle). The stop bits do not contain any change of the phase. The configuration of the forward frame length is fixed by register programming. The receiver expects a frame of the exact programmed length. Currently it's not possible to manage a shorter frame as a subset. E.g.: if a 17-bit length is configured, the peripheral cannot manage 16-bit frames and treats the case as an error. Anytime users can reset and reprogram the peripheral in order to change the length of the accepted forward frame.

A backward frame consists of 11 bi-phase encoded bits: a 1 start bit (logical '1') and a 1 data byte. The frame is terminated by 2 stop bits (idle). The stop bits do not contain any change of the phase.

The DALI speed rate may be configured at 1.2, 2.4 or 4.8 kHz for both the forward and backward channels.

The settling time between two subsequent forward frames is 9.17 ms (minimum). The settling time between forward and backward frames is between 2.92 ms and 9.17 ms. If a backward frame has not been started after 9.17 ms, this is interpreted as “no answer”.

In case of code violation, the frame is ignored and the error flag bit EF in the DALI_CSR register is set.

After a code violation has occurred, the system is ready again for a new data reception.

25.4 Monitor receiver line

An internal watchdog timer monitors the activity of the receiver line; if the DALI receiver line is not idle within an elapse time of ~500 ms ± 10%, an interrupt request is asserted to the processor notifying that the DALI receiver line is in the hang state. This functionality is configured through the DALI internal registers (refer to *Section 25.10: DALI registers description on page 291*).

25.5 DALI signal interface

The DALI_rx and DALI_tx signals are multifunction pins configured through the I/O multiplex mechanism described in the product datasheet.

25.5.1 Polarity configurable signal interface

This feature allows programming independently the reverse polarity of the DALI_tx and DALI_rx signal lines in accordance with the DALI standard requirements to extend the interoperability with different external transceivers.

The default configuration after the reset corresponds to the DALI standard, where both transmit and receive lines are set to 1 while in the idle state.

25.5.2 Start bit handling

To correctly support waking up from the Halt mode using the DALI Rx line without losing incoming forward frames, the start bit recognition may be configured in two distinct modes (refer to the SMK bit in *Section 25.10.7: DALI_CR (control register) on page 293*).

In the HALT mode the wakeup time is particularly long and there is the possibility that the DALI device is awakened too late to sample a standard start bit (1 * Te low time). In this case the first frame is ignored and the system must wait for the frame repetition to accept a valid frame.

To avoid this problem when the DALI receive input signal is used to awake the device, the SMK bit in the DALI_CR register must be set to 0; this forces the DALI receiver to accept a valid start bit also if the signal width of the low phase is less than 1 * Te low.

To correctly set this functionality, the system must enter the HALT mode when the DALI device is already in a stable idle condition, that is, the minimum idle time must be already elapsed. This condition can be easily verified testing the RDY_REC bit in the DALI_CSR1 register.

If the system never enters the HALT mode, or if the DALI input line is not used to awake the system, the SMK bit of the DALI_CR register should be set to 1; this configures the DALI device to be compliant to DALI standard specifications, handling the start bit as a bit set to 1 (1 * Te low + 1 * Te high) following the idle condition (4 * Te high).

25.5.3 Digital noise rejection filter

A configurable digital filter may be enabled on the DALI rx line to clean-up noise, glitches, and ring oscillations (optional feature, check the availability on the product datasheet).

The filter has the following main features:

- Configurable source clock
- Clock prescaler 8 bits
- Filter counter 6 bits
- Configurable filter modes

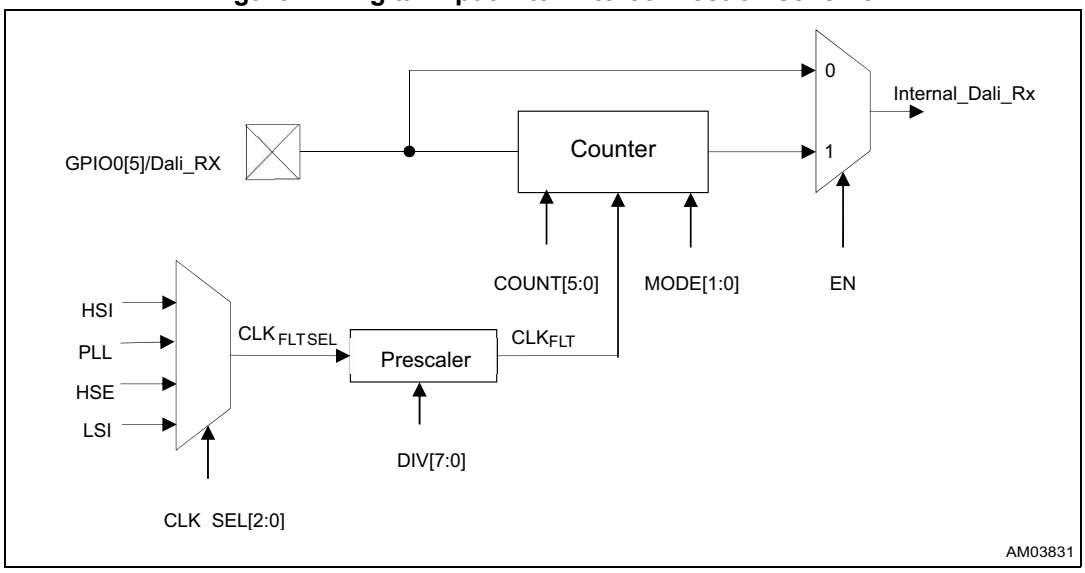
Filter overview

A digital filter may be enabled on the DALI_Rx line when this signal is affected by a noisy, glitch or unstable transitions level.

When the digital filter is enabled, the external input pin (DALI Rx signal) is connected to the input filter, and the output filter is connected to the internal DALI Rx line. In this configuration the DALI device receives an input signal which is glitch-free and with stable levels.

Figure 71 shows an outline view of the DALI noise rejection filter scheme.

Figure 71. Digital input filter interconnection scheme



Note:

1. *The register MSC_DALICKSEL includes the register fields: EN and CLK_SEL[2:0].*
2. *The register MSC_DALICKDIV includes the register field: DIV[7:0].*
3. *The register MSC_DALICONF includes the register fields: MODE[1:0] and COUNT[5:0].*
4. *The DALI noise rejection filter registers description is found in the product datasheet.*

The filter module may be configured to work with different source clocks and in different modes in order to have flexibility on handling different typologies of input noise.

The filter clock may be any of the source clocks available inside the IC device; after being prescaled (if it's necessary), it is used as a base clock for the filter timings operations.

Independently from the selected clock source, the filter logic requires always the activation of the PLL clock since this is used for input sampling and the internal stage synchronization.

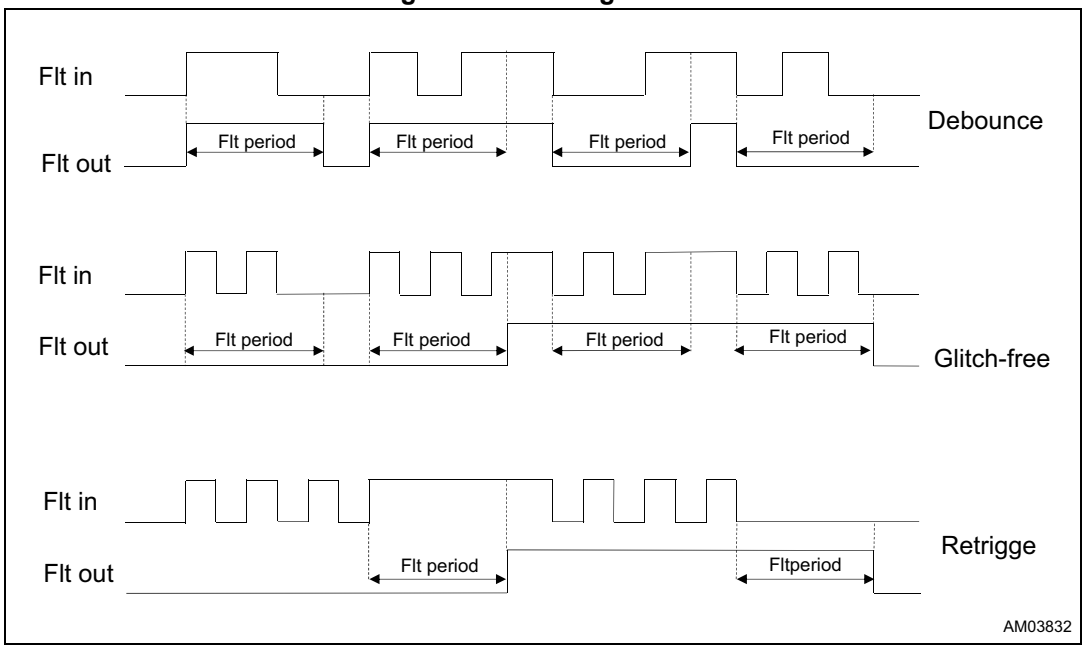
Filter operating mode

The filter can be configured in three different operating modes as detailed in the next paragraph.

- **Debounce:** when the input of the filter changes the level, the output signal is immediately updated, then the input of the filter is masked for the configured period and only at the end of the mask period the filter is again sensible to its input. This mode can be used to remove bouncing input signals (e.g.: electromechanical contacts).
- **Glitch-free:** when the input of the filter changes the level, the starting level is stored inside the filter, the output signal is unchanged and the configured period is started; at the end of the period if the input level is different from the stored level then the output signal is updated with the new level. This mode can be used to ignore short pulses (glitches) from the input signal (noisy lines).
- **Retrigger:** this mode is similar to the glitch-free mode with the difference that the filter period is restarted at every change of the input level. Only when there are no changes on the input signal for the configured filter period, the output signal is updated.

Figure 72 shows an outline view of the filter basic timing operations for the three configuration modes.

Figure 72. Filtering modes



Filter timing

The external input signal is always sampled using the internal high-speed PLL clock (96 MHz). All the other internal filter activities are based on the CLKFLT derived from a prescaled clock source configuration (CLKFLTSEL) refer to *Figure 71*.

In detail the timing of the internal filter depends on the selected clock, clock divider, clock count and internal synchronization periods.

The filter timing model is detailed by the next equations:

Equation 25

$$t_{FLTMIN} = 2 * t_{PLL} + 1 * t_{FLT} + Count * t_{FLT} + 1 * t_{FLT} + 3 * t_{PLL}$$

$$t_{FLTMAX} = 3 * t_{PLL} + 1 * t_{FLT} + Count * t_{FLT} + 1 * t_{FLT} + 4 * t_{PLL}$$

where:

t_{PLL} is the period of the PLL internal clock (1 / 96 MHz)

t_{FLT} is the period of the filter clock (CLKFLT prescaled clock)

Count is the value configured in the filter counter register.

The difference between the t_{FLTMAX} and t_{FLTMIN} value depends on the discrete delay due to the data sampling for internal synchronization stages.

The noise rejection filter configuration registers are accessible through the indirect address space described in the product datasheet.

25.6 DALI data rate

The DALI data rate is programmed by a 10-bits prescale register (DALI_CLK_L and DALI_CLK_H) according to the following formula:

Equation 26

$$f_{DATA} = f_{MASTER} / [(N + 1) * 16]$$

Where *N* is the integer value configured to the registers DALI_CLK_L[7:0] and DALI_CLK_H[9:8].

Note: The f_{DATA} rate must be configured for the double of the desired speed due to the bi-phase Manchester data encoding.

25.7 DALI low power mode

Table 51. DALI low power mode

DALI interface behavior in low power modes	
Mode	**Description**
Wait	No effect on the DALI. DALI interrupts cause the device to exit from the wait mode.
Halt Active-halt	DALI registers are frozen. In the Halt or Active-halt mode, the DALI stops transmitting/receiving until the exit from Halt or Active-halt modes.

25.8 DALI interrupts

The following interrupts can be generated by the DALI controller:

- ITF: the “backward frame” transmission end or the “forward frame” reception interrupt status flag.
- WDGF: the watchdog interrupt line status flag.

Both interrupts are individually maskable (refer to IEN and WDGE bits of the DALI_CSR register in *Section 25.10.8: DALI_CSR (control and status register) on page 295*) and use a single interrupt vector. The ISR checks the ITF or WDGF flags to identify the interrupt type.

25.9 DALI programming sequences

25.9.1 DALI initialization procedure

To use the DALI peripheral, the user should perform the following steps:

- Setup CKC peripheral registers to start the DALI internal clock source (PCKEN1[3]).
- Setup the DALI prescaler register (DALI_CLK_H and DALI_CLK_L).
- Write the DALI_CR message length.
- Enable the interrupt (the IEN bit and/or WDGE bit on the DALI_CSR register) if necessary.
- Setup line polarity as required by the application (the En_Rev bit on the DALI_REVLN register).
- Set DCME bit on the DALI_CR register to start the DALI function.
- Poll the DALI_CSR or DALI_CSR1 register or wait interrupt (if enabled) to handle the DALI data events.

*Note: After the initialization completion (the DCME bit set to enable the device), the DALI peripheral is able to correctly receive an incoming forward frame only after the minimum idle time is elapsed (4 * Te high).*

25.9.2 Data interrupt handling

The interrupt handling is controlled by two different procedures depending on the availability of the backward frame data.

If the backward frame data are immediately available within the ISR (“Interrupt Service Routine”) function, the first procedure is applicable. If the backward message frame construction requires a long elaboration time not compatible with the execution of the ISR function, the second procedure it's preferable.

ISR in line procedure-1

- Initialize the DALI device as described in the previous paragraph with the RTS bit of the DALI_CR register cleared. This switches the peripheral in the reception mode.
- On forward frame receive interrupt (bits ITF and RTF set in the DALI_CSR register) exec the following steps:
  – Read the received data (DALI_FBx); this is not mandatory and depends on the application requirements.
  – Load the transmit data register with the backward frame (DALI_BD).
  – Set the RTS bit in the DALI_CR register to prepare the device to the transmit mode.
  – Set the RTA bit in the DALI_CR register to clear the pending interrupt flag and to start the transmission of the backward frame.
- On backward frame transmit interrupt (the bit ITF set and the bit RTF clear in the DALI_CSR register) exec the following steps:
  – Clear the RTS bit in the DALI CR register to prepare the device to the receive mode.
  – Set the RTA bit in the DALI_CR register to clear the pending interrupt flag and to activate the reception of a new forward frame.

Deferred procedure-2

- Initialize the DALI device as described in the previous paragraph with the RTS bit of the DALI_CR register cleared. This switches the peripheral in the reception mode.
- On forward frame receive interrupt (bits ITF and RTF set in the DALI_CSR register) exec the following steps:
  – Read the received data (DALI_FBx); this is not mandatory and depends on the application requirements.
  – Set the RTA bit in the DALI_CR register to clear the pending interrupt flag.
  – At application level prepare the contents of the backward frame and when ready:
    - Load it in the DALI_BD register.
    - Set the FTS bit in the DALI_CR register to force the transmission mode.
- On backward frame transmit interrupt (the bit ITF set and bit RTF clear in the DALI_CSR register) exec the following steps:
  – Clear the FTS bit in the DALI CR register to prepare the device to the receive mode.
  – Set the RTA bit in the DALI_CR register to clear the pending interrupt flag and to activate the reception of a new forward frame.

*Note: 1. In this second procedure the FTS bit must be mandatory cleared before 1 * Te time from the reception of the interrupt. If this is not feasible (e.g.: the DALI ISR is at low priority and can be interrupted from high priority interrupts), than a different handling of the FTS bit is required. In this case the FTS bit can be set at the application level and then cleared in the following instruction; this grants the FTS signal at the high level for at least one clock cycle and it is enough to force the DALI peripheral to the transmit mode. In the ISR function the clearing of the FTS bit is not required in this alternative handling mode.*

2. The assertion of the RTA interrupt clearing bit must be done only when the ITF interrupt flag is set.

25.10 DALI registers description

25.10.1 DALI_CLK_L (clock prescaler LSB)

Offset: 0x00

Default value: 0x00

7	6	5	4	3	2	1	0
DALI_CLK [7:0]							
r/w							

Bit 7-0: **DALI_CLK[7:0]** clock prescaler lower bit (LSB)

These 8 bits are the lower part of a wider 10-bit prescaler counter used for tuning the DALI data rate $f_{DATA} = f_{MASTER} / [(N + 1) * 16]$,
where *N* is the integer value of the DALI_CLK[9:0] entire prescaler.

25.10.2 DALI_CLK_H (clock prescaler MSB)

Offset: 0x01

Default value: 0x00

7	6	5	4	3	2	1	0
RFU						DALI_CLK [9:8]	
r						rw	

Bit 1-0: **DALI_CLK[9:8]** clock prescaler upper bit (MSB)

The higher 2 bits of the 10-bit prescaler counter.

Bit 7-2: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Note: The division factor, to obtain the transmitting frequency, is made up by combining the DALI_CLK_H and DALI_CLK_L.

25.10.3 DALI_FB0 [forward message register (7:0)]

Offset: 0x02

Default value: 0x00

7	6	5	4	3	2	1	0
DALI_FB0 [7:0]							
r							

Bit 7-0: **DALI_FB0[7:0]** forward message(7:0)

These 8 bits are the lower byte message received from the DALI_rx line.

25.10.4 DALI_FB1 [forward message register(15:8)]

Offset: 0x03

Default value: 0x00

7	6	5	4	3	2	1	0
DALI_FB1 [7:0]							
r							

Bit 7-0: **DALI_FB1[7:0]** forward message(15:8)

These 8 bits are the middle byte message received from the DALI_rx line.

25.10.5 DALI_FB2 [forward message register(23:16)]

Offset: 0x04

Default value: 0x00

7	6	5	4	3	2	1	0
DALI_FB2 [7:0]							
r							

Bit 0: **DALI_FB2[0]** forward message(16)

When the DALI_CR [5:4] is equal to “11”, “10” or “01”, it corresponds to the forward message bit 16, otherwise it is meaningless.

Bit 1: **DALI_FB2[1]** forward message(17)

When the DALI_CR [5:4] is equal to “11” or “10”, it corresponds to the forward message bit 17, otherwise it is meaningless.

Bit 7-2: **DALI_FB2[7:2]** forward message(23:18)

When the DALI_CR [5:4] = “11”, it corresponds to the forward message bits 23 - 18, otherwise it's meaningless.

25.10.6 DALI_BD [backward message register(7:0)]

Offset: 0x05

Default value: 0x00

7	6	5	4	3	2	1	0
DALI_BD [7:0]							
r/w							

Bit 7-0: **DALI_BD[7:0]** backward message(7:0)

These 8 bits are the byte message transmitted to the DALI_tx line.

The software writes to this register before enabling the transmit operation.

25.10.7 DALI_CR (control register)

Offset: 0x06

Default value: 0x00

7	6	5	4	3	2	1	0
LNWDG_EN	SMK	MLN[1:0]		DCME	RTA	RTS	FTS
r/w							

Bit 0: **FTS** force transmit state

When this bit is set, the DALI is forced into the transmit state regardless of the state of the RTS bit. The content of the DALI_BD register is immediately transmitted as backward frame.

Note: Setting of the FTS bit must be used carefully to guarantee the proper DALI transfer frames. The FTS bit must be never asserted during the forward frame receptions.

0: the DALI is not forced to transmit state

1: the DALI is forced to transmit state.

Bit 1: **RTS** receive/transmit state

This bit must be set to '1' after a forward frame is received, if a backward frame is required. This bit must be cleared after a backward frame is transmitted, if a forward frame is required.

0: the DALI is set to receive state.

1: the DALI is set to transmit state.

Bit 2: **RTA** receive/transmit acknowledge

This bit must be set, after a DALI frame reception or transmission, to allow the DALI to perform the next DALI data transfer; this bit is reset by hardware after it has been set by software.

If users miss to set the RTA bit after completion of a reception or transmission the DALI peripheral enters Halt condition.

0: no acknowledge.

1: Interrupt acknowledge bit must be asserted only when the interrupt flag ITF is set.

Bit 3: **DCME** DALI communication enable

When set, it enables DALI communication; it also resets the entire internal finite state machine.

0: DALI disabled (not enabled to receive/transmit, the FSM is put in reset state).

1: DALI enabled to receive/transmit data sequences.

Bit 5-4: **MLN[1:0]** message length

This field defines the forward message length as detailed by the following bit encoding:

00 = 16-bit message

01 = 17-bit message

10 = 18-bit message

11 = 24-bit message

Bit 6: **SMK** mask start bit

0: (default) the start bit low phase (1st transition) is NOT sampled in the middle, but any 0 level sequence from 1 sample and up to the maximum allowed by the Manchester coding for a single phase is accepted as the start bit condition. This is particularly useful when the device is in the HALT condition, and a part of the start bit time is used to awake the device. All other phases including the start bit high phase are sampled in the middle (*Section 25.5.2: Start bit handling on page 285*).

1: the start bit low phase (1st transition) is sampled in the middle as for all other received bits.

Bit 7: **LNWDG_EN** monitor watchdog on receiver line.

It's used to monitor the receiver line activity; when enabled an interrupt is asserted in case the receiver line is not in the idle state within a period of ~ 500 ms .

0: disable monitor watchdog line.

1: enable monitor watchdog line.

Note: Back-to-back writing register requires the insertion of at least three “NOP” instructions.

25.10.8 DALI_CSR (control and status register)

Offset: 0x07

Default value: 0x00

7	6	5	4	3	2	1	0
IEN	ITF	EF	RTF	WDGE	WDGF	RFU	RFU
r/w	r	r_wc	r	r/w	r	r	r

Bit 1-0: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 2: **WDGF** watchdog receiver line interrupt status flag

This field reflects the watchdog monitor output overflow signal. If the WDGE interrupt is enabled and this bit is asserted, the processor interrupt request line is active. The interrupt request is cleared by disabling (resetting) the LNWDG_EN bit field of the DALI_CR register.

0: interrupt deasserted.

1: interrupt pending.

Bit 3: **WDGE** watchdog receiver line interrupt enable

0: interrupt disable.

1: interrupt enable.

Bit 4: **RTF** receive/transmit flag

0: the DALI is in transmit state.

1: the DALI is in receive state.

Bit 5: **EF** error flag

This bit is set when either the DALI data format received is wrong or an interface failure is detected. This bit is set by hardware and is cleared by writing it to “1”.

0: no data format error during reception.

1: data format error during reception.

The EF bit is a pure status bit that does not affect the behavior of the DALI internal state machine. Valid frames can be accepted also when the flag is set. It is up to handling software decides if to ignore the frame due to the error flag or if to ignore the error.

Clearing the EF flag does not clear the internal error condition but simply clears the DALI_CSR status flag. If one of the two above mentioned conditions is still true when clearing the DALI_CSR EF flag, the flag is immediately reasserted again.

Bit 6: **ITF** interrupt flag

This bit is set after the end of the “backward frame” transmission or the “forward frame” reception. It is cleared by setting the RTA bit in the DALI_CR register.

0: interrupt not asserted.

1: end of reception/transmission data transfer.

Bit 7: **IEN** interrupt enable

When set, this bit allows the generation of DALI interrupts.

0: DALI interrupts (ITF) disabled.

1: DALI interrupt (ITF) enabled.

25.10.9 DALI_CSR1 (control and status register1)

Offset: 0x08

Default value: 0x00

7	6	5	4	3	2	1	0
CKS[3:0]				RDY_REC	WDG_PRSC[2:0]		
r				r	r/w		

Bit 2-0: **WDG_PRSC[2:0]** watchdog DALI prescaler timer

000: reserved encoding.

001: select 500 ms constant timer assuming speed line 1.2 Kbps.

010: select 500 ms constant timer assuming speed line 2.4 Kbps.

100: select 500 ms constant timer assuming speed line 4.8 Kbps.

Bit 3: **RDY_REC** ready to receive

0: the DALI receiver is collecting a stream, that's a valid start sequence is received from DALI data input.

1: The DALI is ready to check a valid start sequence from DALI data input.

Bit 7-4: **CKS[3:0]** clock counter value

This is the value of the 4-bit sample clock counter (integer range 0 to 15). The clock counter value is loaded in the DALI_CSR register when a DALI change of the phase signal is detected (edge trigger).

25.10.10 DALI_REVLN (control reverse signal line)

Offset: 0x09

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	RFU	Rev_Dout	Rev_Din	En_Rev
r	r	r	r	r	r/w	r/w	r/w

Bit 0: **En_Rev** reverse DALI reverse signal line:

0: disable reverse functionality.

1: enable reverse functionality.

Bit 1: **Rev_Din** reverse DALI_rx signal line:

0: reverse functionality disable (DALI_rx normal operating).

1: reverse functionality enable [DALI_rx = not (DALI_rx)].

Bit 2: **Rev_Dout** reverse DALI_tx signal line:

0: reverse functionality disable (DALI_tx normal operating).

1: reverse functionality enable [DALI_tx = not (DALI_tx)].

Bit 7-3: **RFU** reserved; must be kept 0 during register writing for future compatibility.

25.11 DALI filter control registers

25.11.1 MSC_DALICKSEL (DALI filter clock selection)

Offset:0x05 (MSC INDIRECT AREA)

Default value:0x00

7	6	5	4	3	2	1	0
RFU				EN	CLK_SEL [2:0]		
r				r/w	r/w		

Bit 2-0: **CLK_SEL[2:0]** DALI filter clock source configuration:

000: HSI source clock

001: HSE source clock

010: LSI source clock

011: PLL source clock

1XX: RFU reserved encoding values.

Bit 3: **EN DALI** filter logic enable:

0: filter logic bypassed; DALI rx line passes through without be filtered.

1: filter logic enabled; the DALI rx line is filtered.

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

25.11.2 MSC_DALICKDIV (DALI filter clock division factor)

Offset: 0x06 MSC (INDIRECT AREA)

Default value:0x00

7	6	5	4	3	2	1	0
DIV [7:0]							
r/w							

Bit 7-0: **DIV[7:0]** filter clock division factor:

This filed is a clock prescale of the clock source selected by the MSC_DALICKSEL.

Equation 27

$$CLKFLT = CLKFLTSEL(CLK\_SEL) / (DIV + 1)$$

25.11.3 MSC_DALICONF (DALI filter mode configuration)

Offset: 0x07 (MSC INDIRECT AREA)

Default value:0x00

7	6	5	4	3	2	1	0
MODE1	MODE0	COUNT [5:0]					
r/w	r/w	r/w					

Bit 5-0: **COUNT[5:0]** filter counter timer value:

This field configures the filter noise masks or delay time referred to the CLKFLT clock; the register field is applicable to any filter operating modes.

This value is used to count a time, the base is the filter's clock selected by the MSC_DALICKSEL register; this time is used as a mask for the two mode filter.

Bit 7-6: **MODE[1:0]** filter mode selection:

00: debounce mode; the filter output is set high/low as soon as a rising/falling edge is detected at the input; with the first edge input triggered the filter is masked from input transitions for an amount of the filter's clock determined by the COUNT.

01: glitch-free mode; the filter output is set high/low only after an amount of time determined by the COUNT and based on filter's clock; the time starts with the first edge triggered in input and when it ends an input level check is done, if the input is high and the trigger was the rise edge or is low and the trigger was falling edge, the output is set high or low respectively, otherwise the output is unchanged.

10: retrigger mode; this configuration is similar to the glitch-free mode with the exception that the filter logic is retriggered upfront to any input change, if there are no changes on the input signal for the configured filter period, the output signal is updated.

11: RFU reserved encoding value.

25.12 DALI registers overview

Table 52 summarizes the DALI internal registers starting from the base address specified in the corresponding device datasheet (DS); for detailed register description refer to *Section 25.10: DALI registers description on page 291* and *Section 25.11: DALI filter control registers on page 297*.

Table 52. DALI internal registers overview

Name	Description	Offset	Type	Reset value
DALI_CLK_L	Data rate control register (LSB).	0x00	R/W	0x00
DALI_CLK_H	Data rate control register (MSB)	0x01	R/W	0x00
DALI_FB0	Message byte 0 register	0x02	R	0x00
DALI_FB1	Message byte 1 register	0x03	R	0x00
DALI_FB2	Message byte 2 register	0x04	R	0x00
DALI_BD	Backward data register	0x05	R/W	0x00
DALI_CR	Control register.	0x06	R/W	0x00
DALI_CSR	Status and control register	0x07	R/W	0x00
DALI_CSR1	Status and control register1	0x08	R/W	0x00
DALI_REVLN	Control reverse signal line	0x09	R/W	0x00

Table 53. DALI filter internal registers overview

Name	Description	Offset	Type	Reset value
MSC_FT2CKSEL	Filter32 clock selection	0x05	R/W	0x00
MSC_DALICKDIV	DALI Filter2 clock division factor	0x06	R/W	0x00
MSC_DALICONF	DALI Filter2 mode configuration	0x07	R/W	0x00

26 Analog comparator unit (ACU)

The analog comparator unit (ACU) consists of up to four DAC comparator subunits. Each subunit is composed of two major blocks:

- One high-speed analog comparator CP[x].
- One 4-bit resistive ladder DAC converter DAC[x].

26.1 Overview description

Each DAC output is connected to the negative input of the respective comparator unit directly or through a dedicated two input multiplexer stage. The DACs provide an internal programmable reference voltage to the associated comparators.

The multiplexer stage allows the user to select an external reference for the relative comparator. The comparator positive input lines are always directly interconnected to analog CPP primary pins.

Depending on the product the reference voltage interconnected to the comparator negative input line may be programmed either to the internal DAC units or to the external CPM pins accordingly to two different configuration schemes.

- The STLUX385, STLUX385A, STLUX383A and STLUX325A devices are configured as shown in *Figure 73* and *Figure 74*.
- The STLUX285A and all the STNRG devices (refer to the proper product datasheet) may be configured as shown in *Figure 75*.

The external reference voltage lines offer the users the ability to define a precise comparison threshold value in order to meet the application target requirement.

The external reference voltage must be comprised in a range from 0 up to 1.25 V. Values exceeding1.25 V up to V_{DDA} are allowed but saturate the comparator maximum threshold voltage.

Figure 73. Comparator unit native logic

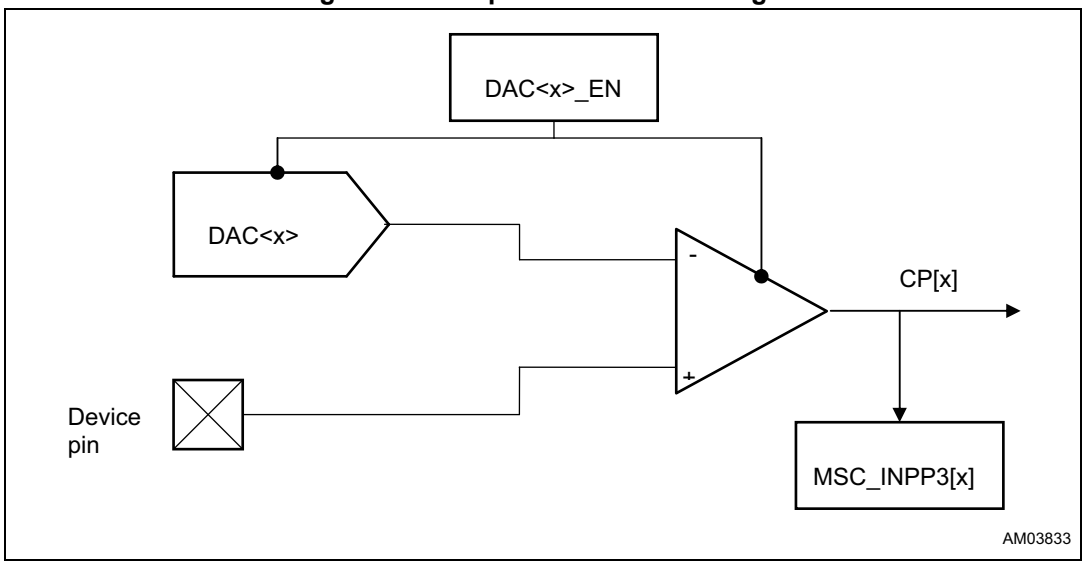


Figure 74. Comparator unit with external reference voltage native logic

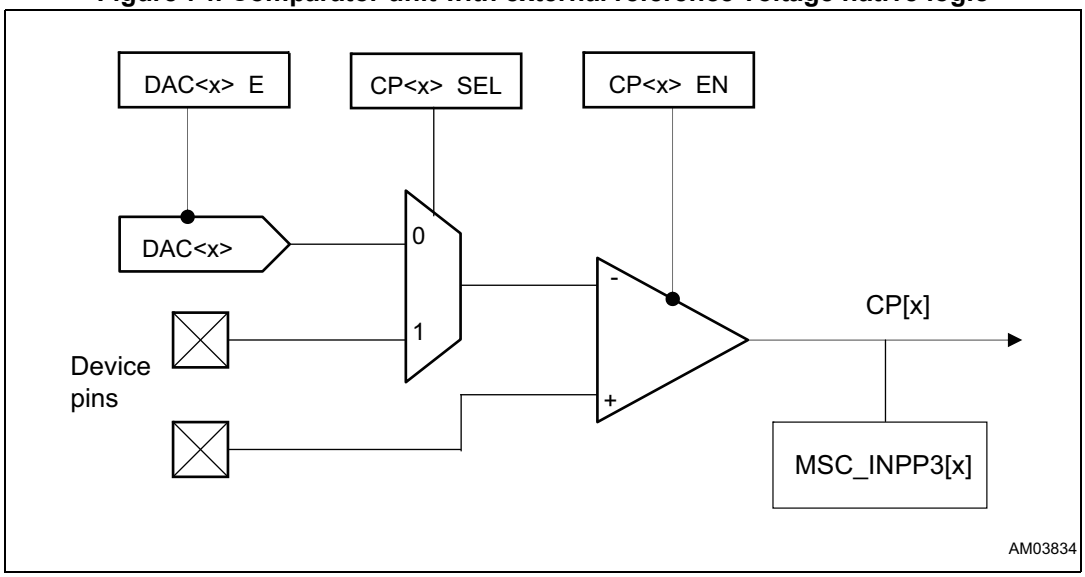
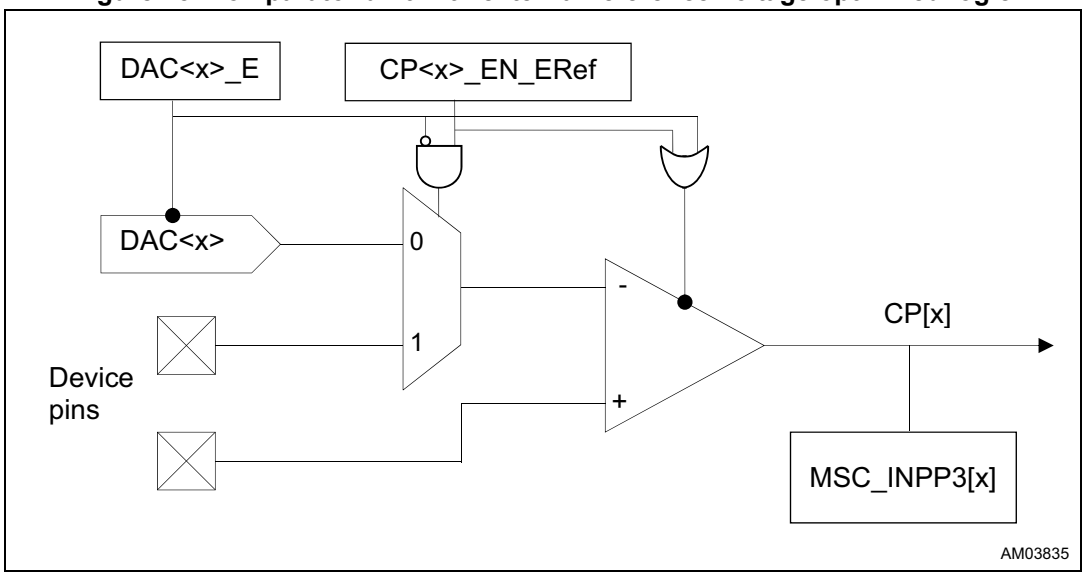


Figure 75. Comparator unit with external reference voltage optimized logic



26.2 Comparator logic

This section describes the detailed features of the comparator blocks.

26.2.1 Main features

- Fast comparison cycle time.
- Direct connection to SMED high-speed logic.
- Comparator output stages readable from processor interface.
- Interrupt capable comparator output signal (please refer to product datasheet for availability).
- Independently programmable up and down hysteresis (please refer to product datasheet for availability).
- ADC trigger functionality (please refer to product datasheet for availability).

The STLUX device family is provided of four independent analog comparator units with fast comparison delay time of maximum 50 ns (Vovd = 3 mV) and the following embedded characteristics:

- Two stages architecture are used to reach a high gain.
- First stage with positive feedback to achieve high-speed time comparison.
- Inverter output stage to drive capacitive load.

The main purpose of the high-speed comparators logic is to supply the SMED peripherals with fast analog input triggered by the application signals and to provide a fast response time to external analog signals through the interrupt functionality.

Refer to the product datasheet and to *Section 21 on page 165* for further details about the comparator interconnection configurability to the SMED units.

26.2.2 DAC and comparator programming

Each comparator stage has the analog positive input port interconnected to the corresponding CPP[x] primary pin, while the negative input ports receive a programmable reference voltage provided by the associated internal 4-bit DAC logic or by an external reference according to the device configurability.

Depending on the product, the negative input port of the comparators may be connected to an external analog input reference voltage signal through multiplexer mechanisms controlled by SW (refer to *Figure 73*, *Figure 74* and *Figure 75* for reference voltage configurability schemes).

The internal reference voltage of each comparator depends on the corresponding DAC configuration register (MSC_DACIN<x>) placed in the MISC addressing area.

The DAC converters, comparator units and the external comparator reference voltage selection are configured by programming the MSC_DACCTR register. Refer to the product datasheet to check the device configurability degree.

- **In all the STLUX devices except the STLUX285A** the comparator3 with external reference is enabled through the CP3_EN bit and the reference voltage selection is done by the CP3_SEL bit field of the MSC_DACCTR register. The DAC units are enabled independently through the DAC<x>_EN fields. The comparators with internal-

only reference voltage are enabled together with the relatives DACs. The bias circuit of both DAC and comparators logics is enabled by the DacBias_en the register bit.

- **In all the STNRG devices and in the STLUX285A** the layout of the MSC_DACCTR register it's different;. the comparator <x> is enabled if either the DAC<x>_EN bit or the CP<x>_EN_ERef bit is set; when set the DAC<x>_EN bit selects the internal reference voltage, otherwise if it's cleared and the CP<x>_EN_ERef bit is set, the external reference voltage is selected. The bias circuitry is enabled by setting at least one of the DAC_EN<x> or CP<x>_EN_ERef bits. *Table 54* summarizes the new layout register encoding bit configuration.

Table 54. DAC and comparator selection

DAC<X>_EN	CP<X>_EN_ERef	Description
0	0	DAC and comparator units power-down
0	1	Enable comparator unit with external reference voltage
1	X	Enable DAC and comparator units with internal reference voltage

Note: 1. Each DAC and comparator unit enabling stages concur to increase the device power consumption.

2. Enabling the bias circuitry increases the device consumption (for details on power consumption see the product datasheet).

26.2.3 Interrupt and wake up capability

STNGR devices offer the possibility of enabling interrupts on the comparator output signals.

The interface used to control the interrupt functionality is present in the MISC indirect addressing area and is similar to the logic that controls the interrupt features of each port (P0, P1 and P2).

The registers MSC_CFGP3<x> and MSC_STSP3 control and configure the interrupt edge detection logic.

In the Halt low-power mode, it's possible leave the ACU unit powered-on and let this to generate the wake-up event through an interrupt assertion. Refer to *Section 14: Interrupt controller (ITC) on page 113* and *Section 13: Power management (PM) on page 107* for further details about the comparator interrupt programming note and the IC device low-power modes.

26.2.4 Hysteresis programming

An optional hysteresis voltage may be programmed independently for rising and falling signals on STNRG devices (refer to the product datasheet for details about the availability of this feature). The hysteresis level is controlled by the MSC_DAC<n>HYS registers, where <n> represent the comparator number; the registers are placed in the MISC indirect memory area.

Once correctly set, Hysteresis applies to the comparator behavior independently whether it works with internal or external reference.

Two register files of three bits (HYSTUP and HYSTDN) control respectively the hysteresis amplitude for rising and falling input comparator signals. The hysteresis voltage values are shown in the product datasheet.

26.2.5 Processor interface to the comparator outputs

The user may read the comparator output values through the processor interface by accessing the memory mapped MSC_INPP3[3:0] register. The register values are double synchronized in the clock master domain before being read by the processor.

26.3 DAC converter logic

The DAC unit consists of up to 4 x 4-bit DAC analog macro used to supply the internal reference voltage to the corresponding comparators logics as shown in *Figure 73*, *Figure 74* and *Figure 75*.

26.4 DAC main features

The DAC architecture is characterized by the following features:

- Four independent 4-bit DAC channels.
- Resistive ladder to achieve high accuracy.
- Digital interface controlled by miscellaneous register.

The input digital value to be converted is programmed independently for each DAC through the miscellaneous registers by configuring the DAC_IN[3:0] field of the MSC_DACIN<x> registers. Power-on and the enable function of DAC circuits are controlled by the DAC_EN[3:0] bit field of the MSC_DACCTR control register.

In STLUX285A and STNRG devices setting the DAC_EN<x> bit selects the internal reference for the comparator x (refer to *Table 54*) and enables the comparator itself, irrespectively of the CP<x>_EN_ERef bit value.

The DAC output voltage conversion scale is shown in *Table 55*.

Table 55. DAC conversion voltage scale

Digital input value (DAC_IN[3:0])	Analog output nominal value	Unit
0x0	0	V
0x1	1 * V_{IREF} / 15	
0x2	2 * V_{IREF} / 15	
0x3	3 * VI_{IREF} / 5	
0x4	4 * V_{IREF} / 5	
0x5	5 * V_{IREF} / 15	
0x6	6 * V_{IREF} / 5	
0x7	7 * V_{IREF} / 15	
0x8	8 * V_{IREF} / 15	
0x9	9 * V_{IREF} / 15	
0xA	10 * V_{IREF} / 15	
0xB	11 * V_{IREF} / 15	
0xC	12 * V_{IREF} / 15	
0xD	13 * V_{IREF} / 15	
0xE	14 * V_{IREF} /15	
0xF	V_{IREF}	

Note: For the V_{IREF} value and PVT ("Process, Voltage and Temperature") variation dependency refer to the product datasheet.

26.5 ACU programming sequence

The DAC analog circuits are enabled and biased by programming the MSC_DACCTR register; in STLUX devices except STLUX285A the DacBias_en bit must be set first before the power-on of the DAC units.

After bias is supplied and DAC/comparator circuits are enabled, the ACU unit requires about 400 ns before being operative and performing correctly the functional comparison sequence.

The ACU unit requires the following programming step sequence:

- Enable the DacBias_en bit field of the MSC_DACCTR register in the STLUX devices except for the STLUX285A. In the STLUX285A and STNRG devices the bias circuitry is enabled as soon as the user enables a comparator or a DAC comparator subunit.
- Configure the references voltage through the MSC_DACCTR register:
  - Internal reference: set the DAC<x>_EN bit. Clear the CP<x>_EN_ERef bit on the STLUX285A and STNRG devices. In STLUX devices except the STLUX285A, reset the CP<x>_SEL bit and set the CP<x>_EN bit if both are available.
  - External reference: clear the DAC<x>_EN bit and the MSC_DACIN<x> register then in the STLUX285A and STNRG devices set the CP<x>_EN_ERef bit, while in STLUX devices except the STLUX285A set both the CP<x>_SEL and CP<x>_EN bits.
- Wait the ACU startup time (at least 400 ns).
- In case of SW polling sequence, check the comparator output values by reading the MSC_INPP3 register.

Note that due to the combinatorial nature of the ACU logics changing the internal or external comparator reference voltage at runtime requires the following rules:

- Disable the interrupt detection logic (if it's enabled).
- Changing internal DAC reference voltage to a new value: the comparator output signal has to be masked for at least 100 ns before being reconsidered by the interconnected logics (SMED and the interrupt detectors).
- Changing external reference voltage to the internal one (supposing the DAC sources already stable) or vice-versa: the comparator output signal has to be masked for at least 50 ns before being reconsidered by the interconnected logics.
- Enable the interrupt detection logic if it's required.

26.6 ACU low power modes

Table 56. ACU low power mode

Modes	Description
Wait	No effect on ACU.
Halt or Active-halt	Before entering in the Halt/Active-halt mode the ACU block must be configured in power-down by SW clearing the MSC_DACCTR register. In STNRG devices, it's possible to keep the ACU unit active, to enable the wake up functionality from the comparator logic. After waking up from Halt/Active-halt, re-enable the peripheral by following the programming sequence described in *Section 26.4: DAC main features*.

The wakeup functionality is implemented by the interrupt control logic connected to the port-P3, for further details refer to *Section 14: Interrupt controller (ITC) on page 113*.

26.7 ACU registers description

The registers listed below are located in the miscellaneous memory area. Please refer to the product datasheet for details on the memory address space.

26.7.1 MSC_DACCTR (DAC comparator control register)

Offset:0x10

A) STLUX devices except the STLUX285A register layout

Default value:0x00

7	6	5	4	3	2	1	0
DacBias_en	RFU	CP3_SEL	CP3_EN	DAC3_EN	DAC2_EN	DAC1_EN	DAC0_EN
r/w	r	r/w	r/w	r/w	r/w	r/w	r/w

Bit 0: **DAC0_EN** enable DAC0 and comparator0 logic unit

0: disable DAC and comp unit (default case).

1: enable DAC and comp unit.

Bit 1: **DAC1_EN** enable DAC1 and comparator1 logic unit

0: disable DAC and comp unit (default case).

1: enable DAC and comp unit.

Bit 2: **DAC2_EN** enable DAC2 and comparator2 logic unit

0: disable DAC and comp unit (default case).

1: enable DAC and comp unit.

Bit 3: **DAC3_EN** enable DAC3 logic unit

0: disable DAC unit (default case).

1: enable DAC unit.

Bit 4: **CP3_EN** enable comparator3 logic unit

0: disable comp unit (default case).

1: enable comp unit.

Bit 5: **CP3_SEL** comparator3 source reference voltage selection.

0: internal DAC3 reference voltage (default case).

1: enable the CPM3 external signal source reference voltage.

Note: The external reference voltage configuration requires the MSC_DACINx contents to be cleared (0x00).

Bit 6: **RFU** reserved; must be kept 0 during register writing for future compatibility.

Bit 7: **DacBias_en** DAC bias enable

The bias is circuit used in conjunction with the external reference voltage.

0: disable DAC bias.

1: enable DAC bias.

B) Register for the STLUX285A and STNRG devices layout

Default value:0x00

7	6	5	4	3	2	1	0
CP3_En_ERef	CP2_En_ERef	CP1_En_ERef	CP0_En_ERef	DAC3_EN	DAC2_EN	DAC1_EN	DAC0_EN
r/w	r	r/w	r/w	r/w	r/w	r/w	r/w

Bit 0: **DAC0_EN** enable the DAC0, Comparator0 logic and selects the internal reference.

0: disable the DAC 0; the Comp0 is disabled if the CP0_En_ERef bit is cleared.

1: enable the DAC0 and Comp0 with the internal DAC reference voltage.

Bit 1: **DAC1_EN** enable the DAC1, Comparator1 logic and selects the internal reference.

0: disable the DAC1; the Comp1 is disabled if the CP1_En_ERef bit is cleared.

1: enable the DAC1 and Comp1 with the internal DAC reference voltage.

Bit 2: **DAC2_EN** enable the DAC2, Comparator2 logic and selects the internal reference.

0: disable the DAC2; the Comp2 is disabled if the CP2_En_ERef bit is cleared.

1: enable the DAC2 and Comp2 with the internal DAC reference voltage.

Bit 3: **DAC3_EN** enable the DAC3, Comparator3 logic and selects the internal reference.

0: disable the DAC3; the Comp3 is disabled if the CP3_En_ERef bit is cleared.

1: enable the DAC3 and Comp3 with the internal DAC reference voltage.

Bit45: **CP0_En_ERef** ERef enable the Comparator0 logic and selects the external reference when the DAC0_EN bit is cleared.

0: the DAC0 and Comp0 are disabled if the DAC0_EN bit is cleared.

1: enable the Comp0 unit and select the external reference if the DAC0_EN bit is cleared.

Bit 5: **CP1_En_ERef** ERef enable the Comparator1 logic and selects the external reference when the DAC1_EN bit is cleared.

0: the DAC1 and Comp1 are disabled if DAC1_EN bit is cleared.

1: enable the Comp1 unit and select the external reference if DAC1_EN bit is cleared.

Bit 6: **CP2_En_ERef** ERef enable the Comparator2 logic and selects the external reference when the DAC2_EN bit is cleared.

0: the DAC2 and Comp2 are disabled if the DAC2_EN bit is cleared.

1: enable the Comp2 unit and select the external reference if DAC2_EN bit is cleared.

Bit 7: **CP3_En_ERef** ERef enable the Comparator3 logic and selects the external reference when the DAC3_EN bit is cleared.

0: the DAC3 and Comp3 are disabled if the DAC3_EN bit is cleared.

1: enable the Comp3 unit and select the external reference if DAC3_EN bit is cleared.

Note: 1. Setting any register fields enabling the bias circuitry.

2. The DAC<x>_EN and CP<x>_EN_ERef coding values is shown in Table 54: DAC and comparator selection on page 303.

26.7.2 MSC_DACIN<n> (Dac<n> input data register)

Offset: 0x11 + <n>

Default value: 0x00

7	6	5	4	3	2	1	0
RFU				DAC_IN<n> [3:0]			
r				r/w			

<n> ranges from 0 to 3.

Bit 3-0: **DAC_IN<n>[3:0]** DAC<n> input digital conversion data.

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

26.7.3 MSC_INPP3 (port P3 input data register)

Offset: 0x29

Default value: undefined

7	6	5	4	3	2	1	0
RFU				COMP [3:0]			
r				r			

This register returns the internal value of the comparator[3:0] signals after two synchronization stages clocked by f_{MASTER}.

Bit 3-0: **COMP[3:0]** comparator logic output signal. This field reflects the comparator output signal values.

Bit 7-4: **RFU** reserved for future use.

26.7.4 MSC_DAC<n>HYST (DAC <n>hysteresis selection)

Indirect address:0x0A + <n>

Default value:0x00

7	6	5	4	3	2	1	0
RFU	HYSTUP [2:0]			RFU	HYSTDN [2:0]		
r	r/w			r	r/w		

<n> ranges from 0 to 3

Bit 2-0: **HYSTDN** comparator hysteresis on falling signals; for the hysteresis voltage values refer to the product datasheet.

Bit 3: **RFU:** reserved for future use

Bit 6-4: **HYSTDP** comparator hysteresis on rising signals; the hysteresis voltage values refer to the product datasheet.

Bit 7: **RFU**: reserved for future use.

26.8 ACU registers overview

Table 57 summarizes the ACU registers which are located in the miscellaneous registers space.

Table 57. ACU register control unit overview

Name	Description	Offset	Type	Reset value
MSC_DACCTR	Enable DACs, and CPs units	0x10	R/W	0x00
MSC_DACIN0	Four bit input value for DAC 0	0x11	R/W	0x00
MSC_DACIN1	Four bit input value for DAC 1	0x12	R/W	0x00
MSC_DACIN2	Four bit input value for DAC 2	0x13	R/W	0x00
MSC_DACIN3	Four bit input value for DAC 3	0x14	R/W	0x00
MSC_INPP3	Port P3 input data register	0x29	R	N. D.

Table 58 lists the MISC indirect address registers related to the ACU peripheral.

Table 58. ACU MISC indirect register overview

Name	Description	Indirect ADD	Type	Reset value
MSC_DAC0HYS	DAC0 hysteresis configuration	0x0A	R/W	0x00
MSC_DAC1HYS	DAC1 hysteresis configuration	0x0B	R/W	0x00
MSC_DAC2HYS	DAC2 hysteresis configuration	0x0C	R/W	0x00
MSC_DAC2HYS	DAC3 hysteresis configuration	0x0D	R/W	0x00
MSC_CFGP30	P3-0 ctrl reg. input line (Cmp0)	0x0E	R/W	0x00
MSC_CFGP31	P3-1 ctrl reg. input line (Cmp1)	0x0F	R/W	0x00
MSC_CFGP32	P3-2 ctrl reg. input line (Cmp2)	0x10	R/W	0x00
MSC_CFGP33	P3-3 ctrl reg. input line (Cmp3)	0x11	R/W	0x00
MSC_STSP3	Port 3 status register (Cmp)	0x12	R/WC	0x00

27 Analog-to-digital converter (ADC)

The 10-bit ADC is a successive approximation analog-to-digital converter unit. This peripheral has up to 8 multiplexed input channels; the analog input signal to be converted is amplified with a selectable gain value by 1 or 4. The analog-to-digital conversion can be done either in single or in continuous modes.

27.1 ADC main features

- 8 ADC input channel
- 10-bit resolution
- Single and continuous conversion mode:
  - Single conversion cycle time 2.416 µs at 6 MHz
  - Continuous conversion cycle time 3 µs at 6 MHz (for channel)
- FIFO auto-flush and auto-reload capability (please refer to product datasheet for availability)
- Hardware triggered start conversion from internal/external sources (please refer to product datasheet for availability)
- Independent conversion gain value x1 or x4
- ADC frequency conversion configurable up to 6 MHz
- Double alignment of conversion data
- Interrupt generations:
  - EOC interrupt asserted on the end of the conversion cycle
  - EOS interrupt asserted on the end of the conversion sequence
  - SEQ_FULL_EN interrupt assert on the sequencer buffer full
- ADC input voltage range: $V_{SSA} \leq V_{IN} \leq V_{DDA}$.

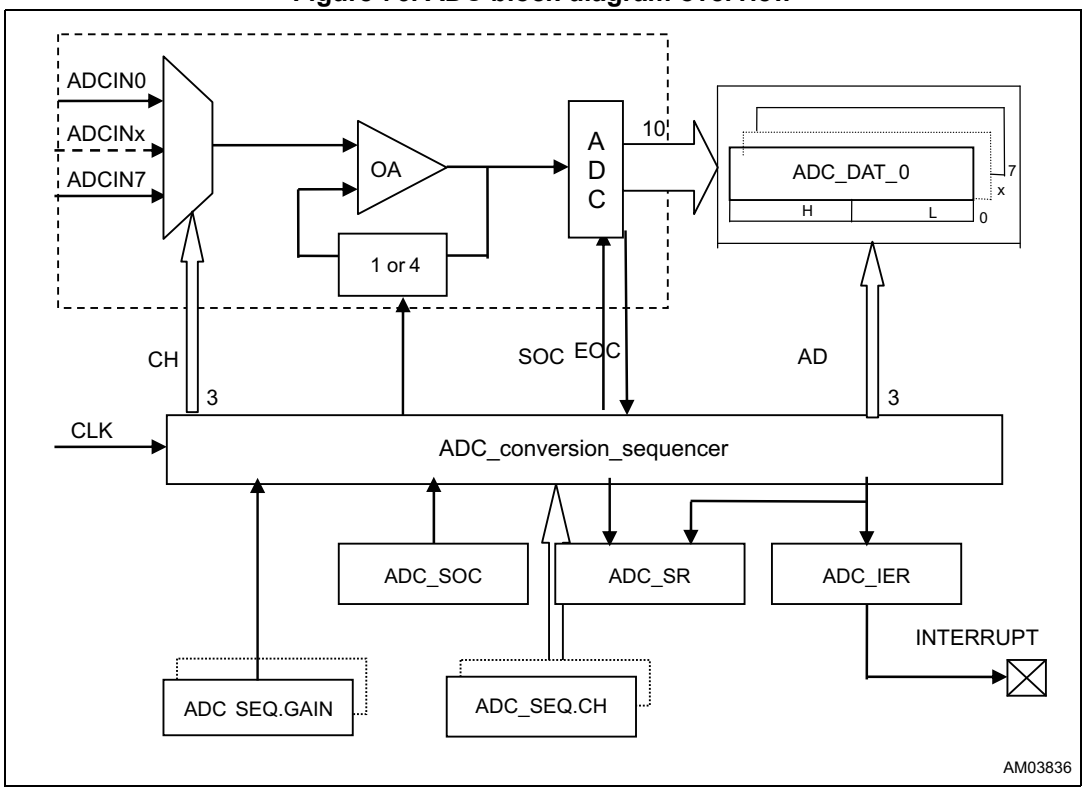
27.2 ADC unit block diagram

The ADC block is composed by two subblocks; a core analog macro responsible for analog-to-digital data conversion and a digital unit that controls the configurability, the conversion sequence and the internal buffer management. The CPU interface controls the following internal buffer registers:

- A write command FIFO 8 x 4 bit data that store eight conversion command parameters; one bit selects the conversion gain configuration (ADC_SEQ.GAIN), three bits (ADC_SEQ.CH) selects the conversion channel number.
- A read buffer 8 x 16 bit that contains the data conversion results.

Figure 76 shows an outline view of the ADC peripheral logic block diagram.

Figure 76. ADC block diagram overview



27.3 ADC analog unit

This block receives the signals from up to eight separate input channels; under the control of the conversion sequencer, the analog unit multiplexes and amplifies the input channel according to the respective content of the FIFO command buffer. The output of the operation amplifier is then sent to the analog-to-digital converter stage and finally the conversion result is written into the read data buffer.

The analog macro includes:

- An analog multiplexer dynamically indexed by the channel values of the FIFO sequencer (up to eight conversions can be programmed on the sequencer and each conversion can be carried out on different or the same channel).
- A programmable operational amplifier to control the input signal amplification (the gain value of each conversion is also programmable by means of the FIFO sequencer).

The operational amplifier allows the 10-bit ADC converter to extend its dynamic range and resolution up to 12-bit equivalent through an amplification factor of x1 or x4.

27.3.1 ADC analog main features

The typical features of the analog front-end block are listed below:

- Settling time for a full dynamic voltage input step: < 500 ns
- Gain bandwidth product of the OA stage: 10 MHz
- High input resistance: > 1 MΩ
- Input parasitic capacitance: 1 pF

The high impedance and the low capacitance offered by the input buffer gives a not negligible advantage when the ADC input signals are driven by a non-ideal source voltage, e.g.: resistor partitions or Zener diodes, since it may reduce the application cost by avoiding external buffers.

After being processed by the front-end stage, the signal is available to the ADC unit, as shown in *Figure 76: ADC block diagram overview*. The signal is then maintained for the conversion time by a RC based “Sample&Hold” mechanism (S&H closed for 3 ADC clock periods). Note that the S&H time can vary depending on the selected ADC clock frequency.

The ADC unit is based on a 10-bit ADC (SAR architecture); the two analog reference voltages V_{refP} and V_{refN}, are generated internally from the bandgap circuit. The ADC input voltage range varies depending on the programmed gain:

- With gain x1 the input voltage range is from 0 up to 1.25 V.
- With gain x4 the input voltage range is from 0 up to 312.5 mV.

When the ADC unit is powered-on by clearing the PD bit of the ADC_CFG register, the internal voltage reference logic that provides V_{refP} and V_{refN} requires some latency time before reaching the nominal value, thus the first conversion sequences cannot be started before 30 µs from the power-up command.

27.4 ADC digital unit

The ADC digital unit:

- Manages the signal handshaking and controls all the timings from/to the ADC analog macrocell.
- Includes the processor interface and the ADC programming registers.
- Stores the conversion data results into an internal buffer area readable from the CPU.
- Generates interrupts on EOC (“End Of single Conversion”), EOS (“End Of conversion Sequence”) and FIFO overflow events.

The ADC interface includes a conversion sequencer, capable of scheduling the selection of the input channels to be converted with the corresponding command FIFO which stores the conversion program parameters. This FIFO has a depth of eight data registers, so the user can program up to 8 conversions that will be executed when the ADC conversion process is started. Each FIFO register is 4-bit wide and stores the gain of the operational amplifier and the input channel number for the respective conversion. Note that the first programmed conversion is the first one carried out; the FIFO is programmable by writing the ADC_SEQ register.

The read buffer is based on eight 8-bit register pairs which store the ADC conversion results as soon as they are available. The logic copies the converted data into the read buffers after any conversion sequence, raising the EoC and the EoS flags when these events occurred; at this point the CPU may read the conversion data.

27.4.1 ADC digital main features

The main features of the ADC digital unit are summarized below:

- Configuration array sequencer.
- Interrupt generation logic:
  - EOC end of conversion cycle.
  - EOS end of conversion sequence.
  - Overflow event on FIFO writing.
- Data conversion buffer 8x2 bytes used to store the 10-bit conversion data (ADC_DATH<k> and ADC_DATL<k>).
- Channel conversion sequencer to program the sequence of the channels to be converted and the conversion gain. Sequences of up to 8 conversions can be programmed.
- Programmable circular buffer for continuous data conversion.
- Completely asynchronous interface between the ADC analog macrocell and digital microcontroller section.
- Configurable data out format: [high (9:2) and low (1:0)] or [high (9:8) and low (7:0)].

27.4.2 The ADC sequencer

The ADC_SEQ register allows configuring the ADC conversion sequence: each conversion can be programmed individually with any of the ADC channels and gain values. The ADC_SEQ register gives access to a FIFO structure of eight registers. The first programmed conversion is the first one executed. There's no restriction on the ADC conversion association in respect to the ADC input lines and gain. Any combination is allowed as follows:

ADC conversion <m> ⟵⟶ ADCIN<n>, <g>

Where:

m is the conversion index ranging from 0 to 7

g is the gain value (0 = x1 - 1 = x4)

n is the number of the ADC input line (ADCIN) varying from 0 to 7.

The same ADC input channel may be assigned to more ADC conversions, also with different gain where is applicable, increasing the conversion resolution on the lower scale and capturing the entire signal dynamics on the higher one.

The conversion command FIFO is loaded by writing the ADC_SEQ register; bit [3] (ADC_SEQ.GAIN) configures the conversion gain, while bits [2:0] (ADC_SEQ.CH) define the ADC channel number. The conversion data results are stored in the ADC read buffer registers ADC_DATL<k> and ADC_DATH<k>; where k ∈ {0, …, 7} is the same index associated with the position of the parameter register in the command FIFO.

27.4.3 ADC conversion modes

The ADC_SEQ is a write-only register (readback is not meaningful). Before starting a conversion, the FIFO has to be programmed by writing the parameters of each desired conversion in the ADC_SEQ register. The ADC conversion sequence is performed strictly respecting the order of the ADC_SEQ writing sequence. The FIFO is indexed by a write pointer, that controls user access, and a read pointer, which is used by the ADC sequencer.

The ADC supports two different conversion modes single or continuous, programmable by configuring the CIRCULAR bit of the ADC_CFG register. In addition the user can select one of the following features, if available in the product (refer to product datasheet):

1. FIFO auto-flush at the end of the conversion sequence
2. FIFO auto-reload after every conversion sequence

Refer to the product datasheet to check the availability of these functionalities on the relevant product. If the product supports the FIFO auto-flush or auto-reload features, these are enabled trough the AReload and AFlush bits of theTM0CONF register placed in the MISC indirect address area (refer to the product datasheet for details). *Table 59* summarizes the possible bit configurations and the corresponding operative modes.

Table 59. ADC operating mode

CIRCULAR	AReload	AFlush	Mode
0	X	0 or N/A	Single (pointer queue emptied; restarting from last position)
0	X	1	Single with auto-flush (pointer queue emptied; restarting from position 0)
1	0 or N/A	X	Continuous (pointer queue not emptied; restarting automatically from pos. 0)
1	1	X	Auto-reload (pointer queue not emptied; restarting from position 0)

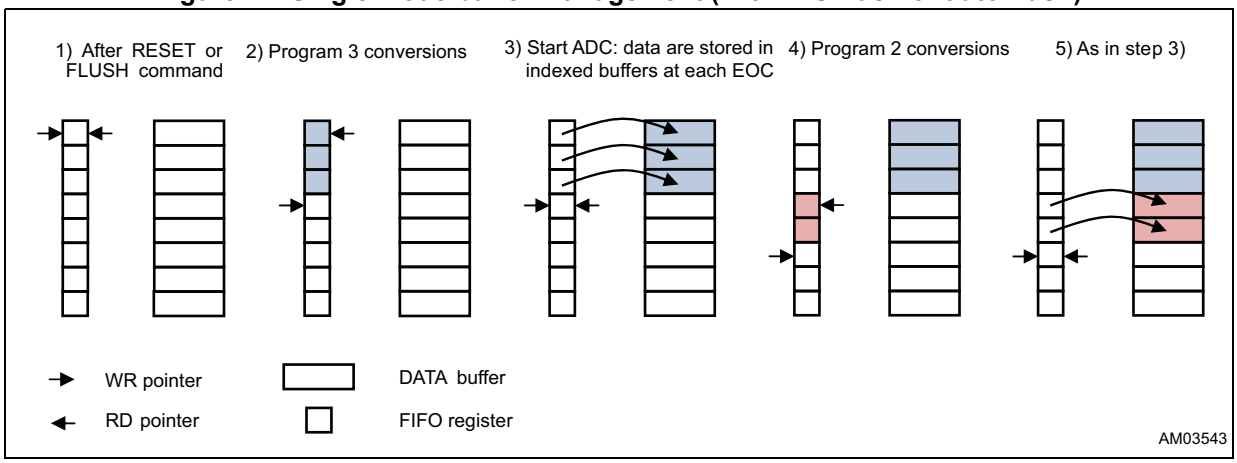
In order to help maintaining coherence between the FIFO programming and data buffer without the user's intervention, an implicit FIFO flush command may be executed by hardware during ADC mode switching. This feature is available only on STNRG devices product revisions (refer to the product datasheet for further details) and may be disabled for compatibility through the option byte. If the functionality is disabled or unavailable, the user has to execute a FIFO flush whenever the ADC operating mode is changed.

Single mode

Single mode: if the CIRCULAR bit is cleared, the ADC performs once the conversion sequence entered in the FIFO, and then it stops its operation.

In case of single mode conversion, the FIFO is considered empty at the end of the ADC conversion list (read and write pointer equals). The user has to write a new sequence (up to 8 consecutive conversions are allowed) before starting the conversion process. If the FIFO auto-flush option is unavailable or disabled, the write and read pointers move circularly around the sequence buffer preserving their position after the conversion sequence has been completed (refer to *Figure 77: Single mode buffer management (w/o FIFO flush or auto-flush)*). The user has to implement an SW circular pointer buffer to index the ADC read buffer and get the correct conversion data; alternatively, before asserting a new conversion command, it's possible to flush the FIFO by setting the register bit ADC_CFG[2]. This command reset both r/w pointers and so that read data of the next conversion sequence are saved starting from the index 0 (sequence 2 - 3 of *Figure 77*). An automatic FIFO flush command is otherwise sent at the end of the conversion sequence if the FIFO auto-flush option is enabled (and supported by the device).

Figure 77. Single mode buffer management (w/o FIFO flush or auto-flush)



Continuous mode

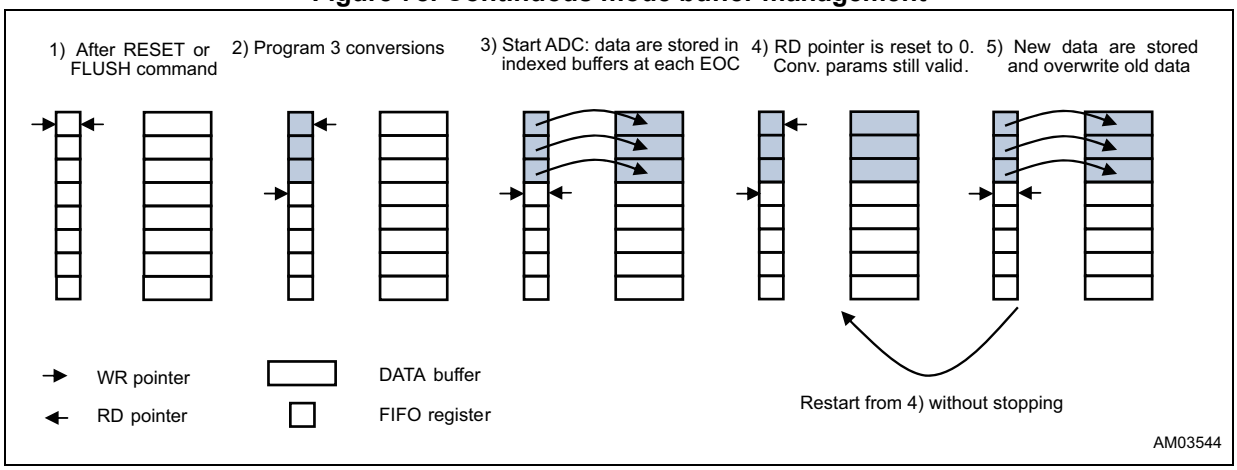
If the CIRCULAR bit is set and the auto-reload option is disabled or unavailable, the ADC repeats continuously the sequence entered in the FIFO, restarting from the first entered conversion every time the EoS event occurs.

In case of continuous mode conversion, after a conversion sequence has been completed, the read pointer restarts from the index 0 (ready to repeat the configured sequence), whatever the programmed conversion count is. The FIFO is never considered empty, except after a reset or a flush command.

The ADC will stop the conversion phase only on the explicit abort request asserted by writing the STOP bit of the ADC_CFG register (for further descriptions about the abort procedure refer to *Section 27.6.3: Abort sequence on page 324*). To program a new conversion sequence in the continuous mode, after the abort sequence has been carried out, the user has to flush the entire sequence from the conversion buffer by writing '1' to the SEQ_DATA_FIFO_FLUSH bit of the ADC_CFG register (this step is product-dependent). The write and read pointers are both cleared by this procedure and the FIFO is flushed. Then a new conversion sequence may be programmed through the ADC_SEQ register.

Figure 78 shows the buffer management in the continuous mode.

Figure 78. Continuous mode buffer management



Auto-reload

When the FIFO auto-reload mode is enabled by setting the CIRCULAR and the AReload bit (if the option available on the product), after the conversion sequence the ADC stops the conversion keeping the conversion command information within the FIFO. The ADC has to be explicitly restarted to repeat once more the conversion sequence, so that the operation can be considered single. The ADC keeps (or auto-reloads) the sequence initially entered in the FIFO, restarting from the conversion programmed in the first FIFO position when a new SoC is requested.

Referring to *Figure 77*, the transition from the step 4) to step 5) [and back to the step 4)] needs in this case an explicit start of the conversion command (either by software or triggered by hardware where available).

27.4.4 Start the ADC conversion

The user may start the ADC conversion from SW through the start of the conversion command SoC. bit of the ADC_SOC register once the FIFO command has been programmed with at least one conversion. The ADC starts the conversion process in accordance with the programmed parameters and the conversion mode configuration (refer to *Section 27.4.3: ADC conversion modes*). Any further SoC command raised during the ADC conversion process is ignored.

Hardware trigger

STNRG devices have the capability to start the ADC conversion sequence from selected internal/external HW trigger sources. This feature is enabled by the ADC_HWtrg option bit of the AFR_IOMXP2 register programmed at '0'.

- The SMED HW trigger capability is enabled by the SMD_HWtrg option bit of the CLKCTL register set at '1'; the enabling feature requires that all SMEDs must be configured with $f_{SMED} \geq f_{MASTER}$.

When this functionality is enabled, the SoC configured by SW and the HW trigger concur together to start the ADC conversion. This has to be considered when the HW trigger conversions are configured. Refer to the product datasheet for further details on the availability of this feature.

Trigger programming

The ADC hardware trigger has to be armed explicitly every time the user needs to start a conversion event. The trigger is armed by setting the ADCTRG_EN bit of the MSC_INPP2AUX1 register placed in the MISC indirect address area (refer to the product datasheet for details). When this bit is set, the ADC waits for the selected event (see *Section 27.9.2: MSC_DALICKSEL (DALI filter clock selection) on page 331* for a description of the programmable trigger sources).

The user should set the ADCTRG_EN bit only when the ADC is stopped; once actives, the ADCTRG_EN bit may be cleared at any time, preventing the hardware triggering of the ADC until the bit is set again. The ADCTRG_EN bit is reset by hardware when a conversion process begins, either by HW or by SW, or if a stop command is issued.

The reiteration of the ADC conversion raised by the HW trigger requires the assertion of the ADCTRG_EN bit after the end of the current conversion sequence or following an abort sequence.

Trigger sources

The HW trigger events are selected by the ADCTRG_SEL[3:0] field of the register MSC_DALICKSEL placed in the MISC indirect address area. The available trigger sources fall in the following categories:

- DIGIN (only some pins)
- CMP (only some pins)
- Timers (please refer to product datasheet for auxiliary/basic timer availability)
- SMED transitions for S2 event (refer to *Section 21: State machine, event driven (SMED) on page 165* for further details on SMED events).

The HW trigger sources list is detailed in the ADCTRG_SEL[3:0] register field description while refer to the product datasheet to know which are the sources relevant for the product device.

The ADC trigger source selection has to be configured before enabling the trigger logic.

The interrupt request of the selected HW trigger source is always masked by HW (except for the timer units).

The external input triggers sources (DIGIN and CMP) are controlled by the P1[4] port. The port has to be enabled and correctly configured as a synchronous edge/level through the MSC_CFGP14 register in order to be registered by the trigger logic.

The internal timers (system timer, auxiliary/basic timers) generate an update event according to its configuration independently by the interrupt generation which can be enabled by SW.

Refer to the product datasheet for details about the availability of this feature.

27.4.5 Reading the conversion result

The ADC_DATL<k> and ADC_DATH<k> are the read buffer registers (where *k* is in the range 0 - 7), used to store the conversion data. When a single conversion data is ready and written to the corresponding buffer registers, the ADC raises the EOC flag; when all conversion data of the sequence are transferred to the buffer registers, the ADC raises also the EOS flag.

It's possible to store the ten bits DATA[9:0] of a single conversion in two distinct formats:

- Shift left data format: ADC_DATL<k>[7,6] contains DATA[1,0] and ADC_DATH<k>[7:0] contains DATA[9:2].
- Shift right data format: ADC_DATL<k>[7:0] contains DATA[7:0] and ADC_DATH<k>[1,0] contains DATA[9,8].

Table 60. Conversion data alignment format

DATA ALIGNMENT	ADC_DATH<k>		ADC_DATL<k>	
RIGHT ALIGNMENT		DATA[9:8]	DATA[7:0]	
LEFT ALIGNMENT	DATA[9:2]		DATA[1:0]	

Note that the data alignment format is controlled by the ADC_CFG DATA_OUT_FORMAT bit of the register ADC_CFG.

27.4.6 DELAY register

The value programmed in the ADC_DLYCNT register is used to insert a delay between the SoC command (either SW, or HW where available) and the effective start of the conversion. The delay affects only the first conversion cycle. If the user writes two or more data to the ADC_SEQ register, only the first conversion is delayed (refer to *Figure 81*). The register content represents the clock unit delay expressed in f_{MASTER} clock.

27.4.7 ADC clock selection

The ADC clock is selected by the clock control unit. For further information refer to *Section 12: Clock control unit (CKC) on page 71* description. The ADC conversion frequency can be configured up to 6 MHz.

27.4.8 ADC timing

From *Figure 79* to *Figure 81* show the ADC single conversion timing diagrams.

Figure 79. ADC single conversion cycle timing diagram

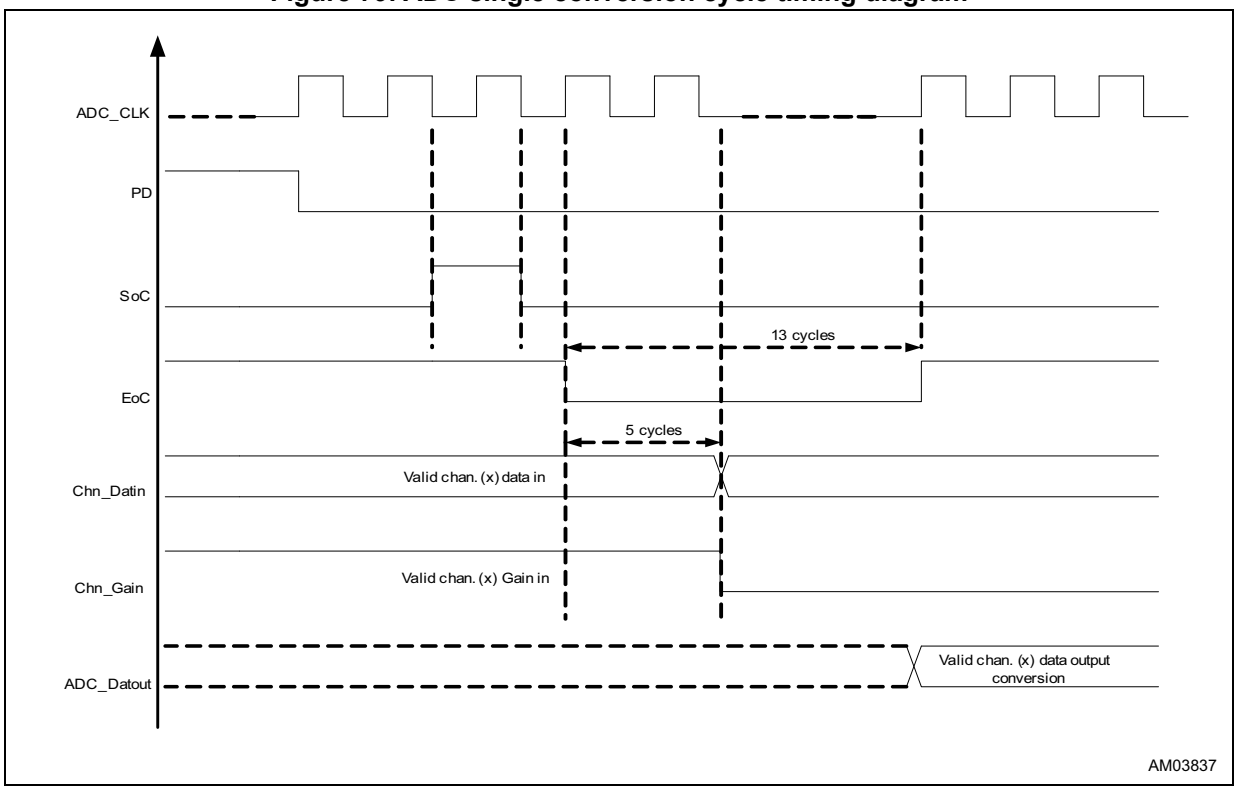


Figure 80 shows the ADC continue conversion cycle timing diagram.

Figure 80. ADC continue conversion cycle timing diagram

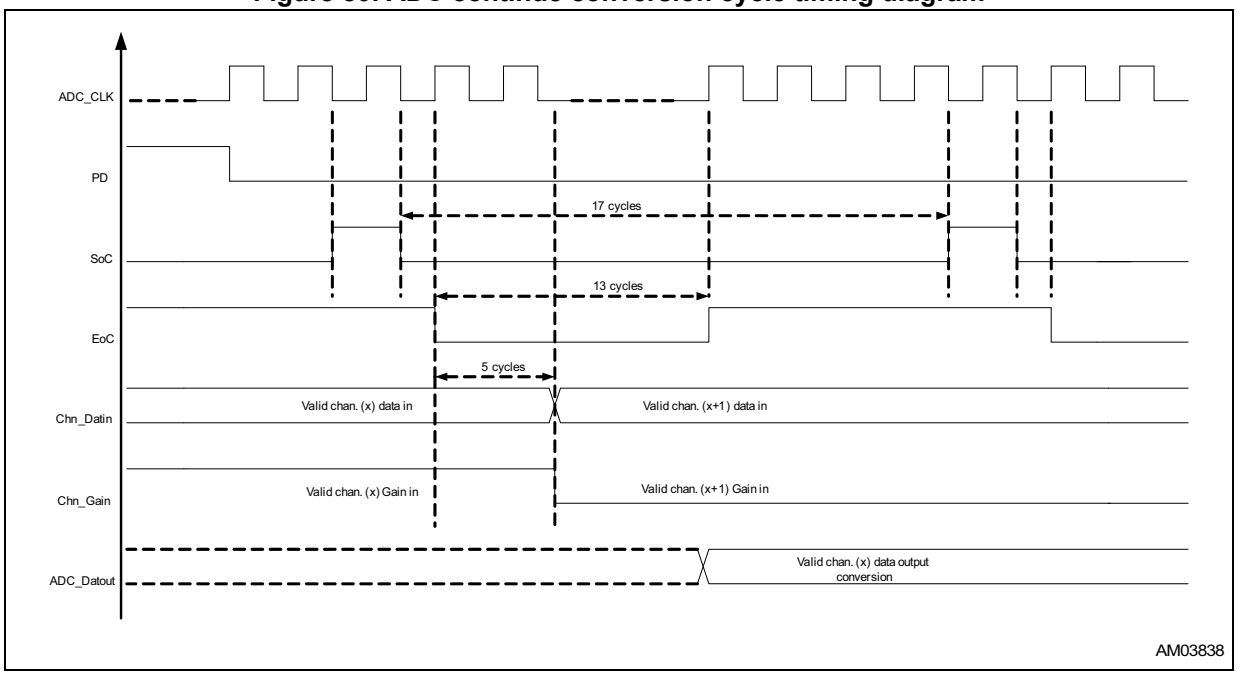
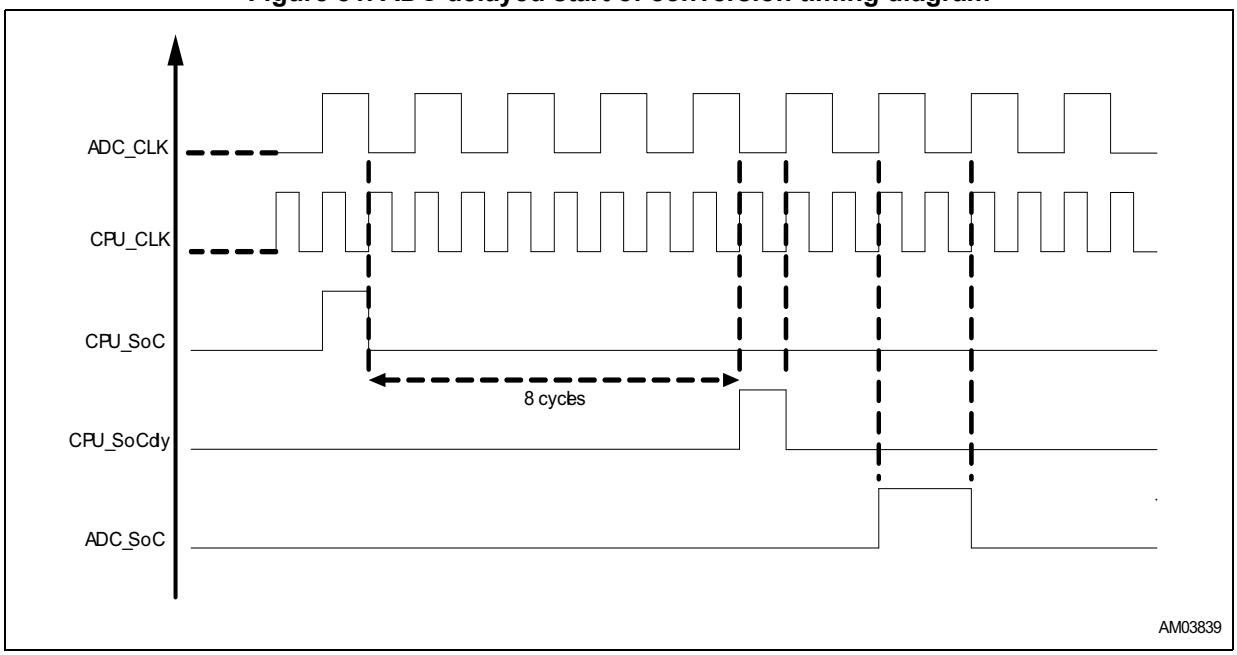


Figure 81 shows the ADC delayed start of conversion timing diagram.

Figure 81. ADC delayed start of conversion timing diagram



27.4.9 ADC HW trigger latency

The latency time between the HW trigger and ADC sample, and the hold (S&H) circuit is shown from the next two equations; for more conversion accuracy users has to consider the latency time trying to compensate this.

1. Internal source delay evaluation (T_D):

Equation 28

$$T_D = T_{syCKM1} + T_{CKM} + T_{DREG} + T_{syCKA} + T_{CKADC}$$

2. External source delay evaluation (T_D):

Equation 29

$$T_D = T_{syCKM2} + 2T_{CKM} + T_{DREG} + T_{syCKA} + T_{CKADC}$$

where:

$T_{CKM} = 1 / f_{MASTER}$

$T_{CKADC} = 1 / f_{ADC}$

$0 \le T_{syCKM1} \le T_{CKM}$

$T_{CKM} \le T_{syCKM2} \le 2T_{CKM}$

T_{DREG} = SOC_DLY_CNT + 1 * T_{CKM}

$T_{CKADC} \le T_{syCKA} \le 2T_{CKADC}$

27.5 Interrupts

The ADC conversion interrupts are associated to the end of the conversion (EOC bit on ADC_SR register) or to the end of the sequence (EOS bit on ADC_SR register).

There is an additional interrupt raised when users attempt to load an additional command by writing the ADC_SEQ register when the command FIFO is full.

The interrupt requests are enabled selectively by the corresponding bits of the ADC_IER register, while the interrupt reset event is done in the following way:

- The EOC and EOS events are reset by writing '1' to the corresponding flags of the ADC_SR register. Both interrupts are capable to wakeup the IC device from the WFI mode (wait for interrupt).
- The SEQ_FULL event is cleared when the FIFO full condition disappears. This happens in the single mode by executing the programmed conversion, so consuming the FIFO entry, or generally by writing '1' to the SEQ_DATA_FIFO_FLUSH bit of the ADC_CFG; this command clears also the entire FIFO contents.

27.6 ADC programming sequences

The following section describes the ADC programming sequence in the single and continuous conversion mode. The ADC conversion sequence can be stopped at any time by writing '1' to the STOP bit of the ADC_CFG register. It is recommended to follow the abort procedures described in *Section 27.6.3: Abort sequence* in case of the reprogramming or power-down sequence.

The switch between different conversion modes has to be done when the ADC is stopped and this should be followed by an SW flush command if it's not performed by HW through the auto-flush feature.

27.6.1 Single mode programming sequence

This procedure has to be used when a single mode is used (single, single with auto-flush, auto-reload modes).

1. Clear the PD bit field of the ADC_CFG register if needed. This turns on the ADC (waiting for the voltage reference stabilization time of 30 µs).
2. Flush the data and sequencer buffers by writing '1' to the SEQ_DATA_FIFO_FLUSH bit of the ADC_CFG register.
3. Reset the CIRCULAR bit and program the OUTPUT_DATA_FORMAT bit of the ADC_CFG register as desired.
4. Enable the conversion interrupt (if used) by setting the EOC_EN or EOS_EN bits.
5. Write optionally the ADC_DLYCNT delay counter register.
6. Write the sequence of *n* conversion into the 4-bit ADC_SEQ register (*n* ranging from 1 to 8).
7. Set the SOC bit of the ADC_SOC register (start of conversion) or select a HW trigger source through the ADCTRG_SEL bit field and enable the trigger functionality by setting the ADCTRG_EN bit (if the HW trigger is available on the device).
8. Waiting for the end of the conversion event by reading in the polling mode the ADC_SR (status) register to check the EOC or EOS flags or waiting for the corresponding interrupt event if enabled by the ADC_IER register.
9. Load the data conversion results by reading the ADC_DATL<k> and ADC_DATH<k> registers.
   a) Single mode (ADC_AFlush = '0' and ADC_AReload = '0' or both N/A): the read pointer buffer has to be tracked by SW, in order to determine where the *n* conversions have been stored. The first value of *k* index is the next (wrapped around the range [0, n-1]) of the last storing index value in the preceding conversion sequence (refer to *Single mode* in *Section 27.4.3: ADC conversion modes on page 315*).
   b) ADC_AFlush = '1' or ADC_AReload = '1': the buffer index *k* ranges from 0 to n-1, where *n* is the number of programmed conversions.
10. If a new conversion is needed:
    a) ADC_AReload = '1': the user has to proceed again from the step 6.
    b) ADC_AReload = '0' or N/A: the user can proceed from the step 7.

    Alternatively, the SW can assert the flush command by writing '1' to the SEQ_DATA_FIFO_FLUSH field of the ADC_CFG register to clear the read and write pointers before restarting the programming sequences from the step 6 (for further details refer to *Section 27.4.3: ADC conversion modes on page 315*).

27.6.2 Continuous (circular buffer) programming sequence

This procedure has to be used when a continuous conversion mode is programmed into the ADC_SEQ register.

1. Clear the PD bit field of the ADC_CFG register if needed. This turns on the ADC (waiting for the voltage reference stabilization time 30 µs).
2. Flush the data and sequencer buffers by writing '1' to the SEQ_DATA_FIFO_FLUSH bit of the ADC_CFG register.
3. Set the CIRCULAR bit and program the OUTPUT_DATA_FORMAT bit of the ADC_CFG register as desired.
4. Enable the conversion interrupt (if used) by setting the EOS_EN bit.
5. Write optionally the ADC_DLYCNT delay counter register.
6. Write the sequence of *n* conversion into the 4-bit ADC_SEQ register (*n* ranging from 1 to 8).
7. Set the SOC bit of the ADC_SOC register (start of conversion) or select a HW trigger source through the ADCTRG_SEL bit field and enable the trigger functionality by setting the ADCTRG_EN bit (if the HW trigger is available on the device).
8. Check the ADC_SR (status) register for the EOS or, if the interrupt is enabled, wait for the interrupt exception event.
9. Load the data conversion by reading the ADC_DATL<k> and ADC_DATH<k> registers with *k* ranging from 0 to n-1, *n* is the count of programmed conversions.
10. Prepare to manage next conversion data. The ADC isn't stopped after it has carried up the preceding conversion sequence.
11. If a new conversion sequence with new parameters is needed, follow the abort sequence explained in *Section 27.6.3: Abort sequence* and execute a flush of the FIFO sequencer by setting the ADC_CFG SEQ_DATA_FIFO_FLUSH bit of the ADC_CFG register. Then restart the programming sequence from the step 6.

27.6.3 Abort sequence

The ADC may be forced from SW to abort and stop the current conversion cycles before a new reconfiguration or to enter in the power-down state by following the next procedures.

1. Clear the ADCTRG_EN bit if the HW trigger functionality is available and enabled.
2. Send a stop command to the ADC by writing '1' in the STOP bit of the ADC_CFG register. If the enhanced abort is available and enabled, the stop command also issues automatically a FIFO flush.
3.
   a) If the enhanced abort functionality is unavailable or disabled by the option byte, wait at least a time equivalent to 20 f_{ADC} clock cycles.
   b) If the new abort features is enable read the STOP bit waiting until it's clear, this ensures that the ADC conversion it's been halted.
4. Clear EOC/EOS flags.
5. Set the SEQ_DATA_FIFO_FLUSH bit to flush the FIFO (not needed with enhanced abort functionality).

27.6.4 ADC power-down sequence

Particular caution must be taken when the ADC peripherals are forced in the low power consumption mode. The CPU can invoke the power-down sequence by executing periodically the HALT instruction. The ADC is then forced in the low power mode irrespectively of the value of the PD bit of the ADC_CFG register. The SW program must assure that the ADC is correctly stopped before forcing the system in the Halt mode. After awaking from the HALT, clear the PD bit and waiting for 30 µs before starting the ADC conversion.

In the single conversion mode (single, single with auto-flush, auto-reload modes) follow the next program sequence:

1. Clear the ADCTRG_EN bit if the HW trigger functionality is available and enabled.
2. If the ADC isn't already stopped, wait for the last programmed conversion interrupt (EOS or also EOC in case of single conversion).
3. Read buffer registers if needed and send the FLUSH command if auto-flush isn't enabled.
4. Set the PD bit of the ADC_CFG register and then execute the HALT instruction.

Alternatively, in either in single or continuous conversion modes, the user can adopt the following procedure:

1. Reset the ADCTRG_EN bit if the HW trigger functionality is available and enabled.
2. Follow the abort procedure as described in *Section 27.6.3: Abort sequence*.
3. Read buffer registers if needed.
4. Set the PD bit of the ADC_CFG register and then execute the HALT instruction.

27.7 ADC low power modes

Table 61. ADC low power mode

Modes	Description
Wait	No effect on ADC.
Halt or Active-halt	Before entering in the Halt/Active-halt mode, the ADC macro must be configured in power-down by SW, following the “ADC power-down sequence described in *Section 27.6.4*. After waking up from Halt/Active-halt, the PD bit must be cleared by software to power-on the ADC, and a delay of 30 µs is needed before starting a new conversion.

The ADC does not have the capability to wake up the device from the Active-halt or Halt mode.

27.8 ADC register description

Where <n> is the ADC_DATL/H register address numbers (0 to 7).

27.8.1 ADC_CFG (configuration register)

Offset: 0x00

Default value: 0x01

7	6	5	4	3	2	1	0
RFU	RFU	RFU	DATA_OUT_FORMAT	CIRCULAR	SEQ_DATA_FIFO_FLUSH	STOP	PD
r	r	r	r/w	r/w	w	w/r(1)	r/w

Bit 0: **PD**: power-down

0: the ADC analog macro is enabled. After clearing this bit wait for the voltage reference stabilization time of 30 µs before starting the first conversion sequence.

1: the ADC analog macro is disabled in power-down.

Bit 1: **STOP**: stop the sequencer of ADC

0: write has no effect.

1: the ADC sequencer is reset and return to the idle state. If the enhanced abort functionality is enabled, the request is deferred while a conversion is in progress.

Note: 1. This bit is always read as 0, if the enhanced abort functionality disabled or not available. Otherwise reading '1' means that the stop request is pending; when '0' the ADC is halted.

2. The ADC stop activity in the two clock domains requires 20 f_{ADC} clock cycles.

Bit 2: **SEQ_DATA_FIFO_FLUSH:** DATA buffer flushing; write only register, when read, returns 0.

0: write has no effect.

1: the FIFO collecting the conversion sequence and the data buffer pointers are flushed.

Note: This command requires at least one f_{MASTER} clock of recovery time (insert NOP instruction) before a new conversion sequence.

Bit 3: **CIRCULAR:** circular mode.

0: the ADC executes one time the programmed conversion sequence (single mode), stopping the activity at the end of sequence without resetting the buffer index pointer values; if the auto-flush feature is enabled, the buffer index pointer values is cleared after the conversion; for further details refer to *Single mode* in *Section 27.4.3: ADC conversion modes on page 315*.

1: the ADC is configured in the continuous conversion mode (the FIFO entries are read circularly).

If the auto-reload functionality is disabled, the sequencer converts continuously the programmed conversion restarting automatically from the first one.

If the auto-reload functionality is enabled, the ADC stops the activity at the end of the conversion sequence, reload the current conversion parameters clearing the buffer index pointer; for details refer to *Continuous mode* in*Section 27.4.3: ADC conversion modes on page 315*.

Bit 4: **DATA_OUT_FORMAT:** data output format.

0: the format of the data out register is configured as left alignment [9:2] and [1,0]. This means that the ADC_DATH contains the high part of 10 bits ADC conversion [9:2], while the ADC_DATL contains the least two bits [1:0] in the high part of the bus (the bit 7 and the bit 6 of the ADC_DATL register).

1: the format of data out register is configured as right alignment [9,8] and [7:0]. This means that the ADC_DATH contains the two MSB of 10 bits ADC conversion [9,8] in the lower part of the bus (the bit 1 and the bit 0 of the ADC_DATH register), while the ADC_DATL contains the seven LSB of ADC conversion [7:0].

Bit 7-5: **RFU** reserved; must be kept 0 during register writing for future compatibility.

27.8.2 ADC_SOC (start of conversion)

Offset: 0x01

Default value: 0x00

7	6	5	4	3	2	1	0
RFU							SOC
r							w

Bit 0: **SOC** start of conversion sequence

0: write has no effect.

1: the ADC will start the conversion phase according to the sequence of channels programmed into the SEQ buffer.

Before starting, the ADC checks always if the PD bit of the ADC_CFG register is zero and if the SEQ buffer is not empty. If these two conditions are not met, the SOC command does not take effect.

Bit 7-1: **RFU** reserved; must be kept 0 during register writing for future compatibility.

27.8.3 ADC_IER (interrupt enable register)

Offset: 0x02

Default value: 0x00

7	6	5	4	3	2	1	0
RFU					SEQ_FULL_EN	EOS_EN	EOC_EN
r					r/w	r/w	r/w

Bit 0: **EOC_EN** end of conversion mode interrupt enable

0: the interrupt is disabled.

1: the interrupt is generated at every end of the conversion cycle (EoC).

Note: The end of the conversion interrupt has to be used in case of a single conversion sequence.

Bit 1: **EOS_EN** end of sequence mode interrupt enable

0: the interrupt is disabled.

1: the interrupt is generated at the end of the conversion sequence, when all the programmed channels have been converted.

Note: The interrupt end of the sequence has to be used in case of multiple conversion sequences, this limits the interrupt events at once in order to avoid increasing the CPU workload due to multiple interrupt exceptions.

Bit 2: **SEQ_FULL_EN** sequencer buffer full interrupt enable

0: the interrupt is disabled.

1: the interrupt is generated if the sequencer buffer is full. Any other writes attempt to the ADC_SEQ register is ignored.

Bit 7-3: **RFU** reserved; must be kept 0 during register writing for future compatibility.

27.8.4 ADC_SEQ (sequencer register)

Offset: 0x03

Default value: 0x00

7	6	5	4	3	2	1	0
RFU	RFU	RFU	RFU	GAIN	CH [2:0]		
r	r	r	r	w	w		

Bit 2-0: **CH[2:0]** channel selection (write only register field); when read it returns “000”.

Analog channel conversion value, in a range from 000b to 111b.

Bit 3: **GAIN** it defines the gain to be applied at the relevant channel CH; write only register field, when read, it returns '0'.

0: the gain operation value is x1.

1: the gain operation value is x4.(aq)

Bit 7-4: **RFU** reserved; must be kept 0 during register writing for future compatibility.

27.8.5 ADC_DATL_<n> (data low register right alignment)

Offset: 0x04 + 2*<n>

Default value: 0x00

7	6	5	4	3	2	1	0
DATA_LOW [7:0]							
r							

Bit 7-0: **DATA_LOW[7:0]** lower data

Low part of data converted, corresponding to the ADC DATA[7:0](1).

aq. Please refer to product datasheet for availability of this option.

27.8.6 ADC_DATH_<n> (data high register right alignment)

Offset: 0x05 + 2*<n>

Default value: 0x00

7	6	5	4	3	2	1	0
RFU						DATA_HIGH [1:0]	
r						r	

Bit 1-0: **DATA_HIGH[1:0]** upper data

A high part of data converted, corresponding to the ADC DATA[9:8](1).

Bit 7-2: **RFU** reserved for future use.

Note: A register layout format when DATA_OUT_FORMAT = '1'.

27.8.7 ADC_DATL_<n> (data low register left alignment)

Offset: 0x04 + 2*<n>

Default value: 0x00

7	6	5	4	3	2	1	0
DATA_LOW [1:0]		RFU					
r		r					

Bit 1-0: **DATA_LOW[7:6]** data low

A low part of data converted, corresponding to the ADC DATA[1:0](1).

Bit 7-2: **RFU** reserved for future use.

27.8.8 ADC_DATH_<n> (data high register left alignment)

Offset: 0x05 + 2*<n>

Default value: 0x00

7	6	5	4	3	2	1	0
DATA_HIGH [7:0]							
r							

Bit 7-0: **DATA_HIGH[7:0]** data high

A high part of data converted, corresponding to the ADC DATA[9:2](1).

Note: A register layout format when DATA_OUT_FORMAT = '0'.

27.8.9 ADC_SR (status register)

Offset: 0x14

Default value: 0x00

7	6	5	4	3	2	1	0
RFU					SEQ_FULL	EOS	EOC
r					r	r/w1	r/w1

Bit 0: **EOC** end of conversion mode.

This bit is set by HW at the end of the conversion cycle and cleared by SW.

0: the EoC is not set[ar].

1: the EoC has been set. This bit is write clear. The SW application must write 1 to clear this bit. If the EOC_EN bit is set, the EoC interrupt is raised.

Bit 1: **EOS** end of sequence mode.

This bit is set by HW at the end of the conversion sequence and cleared by SW.

0: the sequencer has not finished the conversion sequence[ar].

1: the sequencer has finished the programmed conversion sequence. This bit is write clear. The SW application must write 1 to clear this bit. If the EOS_EN bit is set, the EoS interrupt is raised.

Bit 2: **SEQ_FULL** sequencer buffer full

This bit is set and cleared by HW.

0: the sequencer buffer is not full.

1: the sequencer buffer is full. This bit is read only. It is cleared when the condition disappears. If the SEQ_FULL_EN bit is set, the SEQ_FULL interrupt is raised.

Bit 7-3: **RFU** reserved; must be kept 0 during register writing for future compatibility.

27.8.10 ADC_DLYCNT (SOC delay counter register)

Offset: 0x15

Default value: 0x00

7	6	5	4	3	2	1	0
SOC_DLY_CNT [7:0]							
r/w							

Bit 7-0: **SOC_DLY_CNT[7:0]** SOC delay counter

This register permits to delay the start phase of the SOC command adding a setup time between the channel/gain value and the SOC of ADC.

This delay is expressed in number of f_{MASTER} clock cycles.

Note: This delay is applied only at the first conversion cycle and follows the SOC command; the next conversions of the same sequence are never delayed.

ar. Writing 0 has no effect.

27.9 ADC HW trigger configuration registers description

These following registers control the HW trigger functionality which is implemented in recent device versions (refer to product datasheet for further details on availability of the feature).

27.9.1 MSC_FTM0CONF (AddTimer0 mode configuration)

Offset: 0x02 (MSC INDIRECT AREA)

Default value: 0x00

7	6	5	4	3	2	1	0
ADC_AReload(1)	ADC_AFlush(1)	COUNT[5:0]					
r/w	r/w	r/w					

1. This bit is available only on STNRG products.

Bit 5-0: **COUNT[5:0]** AddTimer0 counter timer value:

See *Section 20: Basic timer on page 160* for further explanations.

Bit 6: **ADC_AFlush** FIFO auto-flush for single conversion mode:

0: disable auto-flush.

1: enable auto-flush (if available, it works when the CIRCULAR bit of the ADC_CFG register is '0').

Bit 7: **ADC_AReload** FIFO auto-reload mode:

0: disable auto-reload.

1: enable auto-reload (if available, it works when the CIRCULAR bit of the ADC_CFG register is '1').

27.9.2 MSC_DALICKSEL (DALI filter clock selection)

Offset: 0x05 (MSC INDIRECT AREA)

Default value: 0x00

7	6	5	4	3	2	1	0
ADC_TRGSEL[3:0](1)				EN	CLK_SEL [2:0]		
r				r/w	r/w		

1. This bit is available only on STNRG products.

Bit 2-0: **CLK_SEL[2:0]** DALI filter clock source configuration:

See *Section 25: Digital addressable lighting interface (DALI) on page 282* for further explanations.

Bit 3: **EN** DALI filter logic enable:

See *Section 25: Digital addressable lighting interface (DALI)* for further explanations.

Bit 7-4: **ADCTRG_SEL[3:0]** configures the HW ADC trigger sources[as], if available:

0000: none.
0001: reserved.
0010: DIGIN[0] source signal[at]
0011: comparator[0] source signal[at]
0100: DIGIN[3] source signal[at]
0101: comparator[3] source signal[at]
0110: system timer source signal
0111: auxiliary timer source signal[au]
1000: SMED0[av], [aw]
1001: SMED1[av], [aw]
1010: SMED2[av], [aw]
1011: SMED3[av], [aw]
1100: SMED4[av], [aw]
1101: SMED5[av], [aw]
1110: AddTim0 source signal[at]
1111: AddTim1 source signal[at]

27.9.3 MSC_INPP2AUX1 (INPP2 aux register 1)

Offset: 0x08 (MSC INDIRECT AREA)

Default value: 0x00

7	6	5	4	3	2	1	0
ADCTRG_EN(1)	RFU	INPP2_PULCTR [5:0]					
r/w	r	r/w					

1. This bit is available only on STNRG products.

Bit 5-0: **INPP2_PULCTR[5:0]:** P2 (DIGIN) pull-up control

See the product datasheet for the availability of this field and for further explanations.

Bit 6: **RFU** reserved for future use

Bit 7: **ADCTRG_EN** controls the ADC HW triggered conversion request:

0: disable the ADC HW triggered conversion request

1: enable the ADC HW triggered conversion request. Set/cleared by SW. Cleared by HW when SW/HW conversion starts or when the stop command is issued.

as. The ADC trigger source selection has to be configured before enabling the selected trigger.

at. The INNP1[x] related interrupt channel masked if the AddTim<x> is selected.

au. The main interrupt channel masked.

av. An auto-clear request.

aw. The SMED irq low freq. feature has to be disabled when an SMED is configured as the ADC trigger source.

27.10 ADC registers overview

Table 62 summarizes the ADC internal register starting from the base address reported in the datasheet; for detailed register description refer to *Section 27.8: ADC register description on page 326*.

Table 62. ADC digital interface internal registers overview

Name	Description	Offset	Type	Reset value
ADC_CFG	Configuration register	0x00	R_W	0x01
ADC_SOC	Start of conversion register	0x01	W	0x00
ADC_IER	Interrupt enable register	0x02	R/W	0x00
ADC_SEQ	Sequencer register	0x03	W	0x00
ADC_DATL_0	Low part of data 0 converted	0x04	R	0x00
ADC_DATH_0	High part of data 0 converted	0x05	R	0x00
ADC_DATL_1	Low part of data 1 converted	0x06	R	0x00
ADC_DATH_1	High part of data 1 converted	0x07	R	0x00
ADC_DATL_2	Low part of data 2 converted	0x08	R	0x00
ADC_DATH_2	High part of data 2 converted	0x09	R	0x00
ADC_DATL_3	Low part of data 3 converted	0x0A	R	0x00
ADC_DATH_3	High part of data 3 converted	0x0B	R	0x00
ADC_DATL_4	Low part of data 4 converted	0x0C	R	0x00
ADC_DATH_4	High part of data 4 converted	0x0D	R	0x00
ADC_DATL_5	Low part of data 5 converted	0x0E	R	0x00
ADC_DATH_5	High part of data 5 converted	0x0F	R	0x00
ADC_DATL_6	Low part of data 6 converted	0x10	R	0x00
ADC_DATH_6	High part of data 6 converted	0x11	R	0x00
ADC_DATL_7	Low part of data 7 converted	0x12	R	0x00
ADC_DATH_7	High part of data 7 converted	0x13	R	0x00
ADC_SR	Status register	0x14	R/WC	0x10
ADC_DLYCNT	SOC delay counter register	0x15	R/W	0x00

Table 63 lists the MISC indirect address registers related to the ADC peripheral.

Table 63. ADC MISC register overview

Name	Description	Indirect ADD	Type	Reset value
MSC_FT2CKSEL	DALI filter clock selection	0x02	R/W	0x00
MSC_DALICKDIV	DALI filter clock division factor	0x05	R/W	0x00
MSC_INPP2AUX1	INPP2 aux. register 1	0x08	R/W	0x00

28 Revision history

Table 64. Document revision history

Date	Revision	Changes
24-Jul-2015	1	Initial release.