# RM0013
# Reference manual

## STM8L001xx and STM8L101xx microcontroller families

## Introduction

This reference manual targets application developers. It provides complete information on how to use the STM8L001xx and STM8L101xx microcontrollers memory and peripherals.

The STM8L is a family of microcontrollers with different memory densities, packages and peripherals. The STM8L is designed for low-power applications.

For ordering information, pin description, mechanical and electrical device characteristics, refer to the STM8L datasheets.

For information on the STM8 SWIM communication protocol and debug module, refer to the user manual UM0470.

For information on the STM8 core, refer to the STM8 CPU programming manual (PM0044).

# Contents

# List of tables

# List of figures

# 1 Memory and register map

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

## 1.1 Register description abbreviations

In the register descriptions of each chapter in this reference manual, the following abbreviations are used:

**Table 1. List of abbreviations**

| Abbreviation | Description |
|---|---|
| **read/write (rw)** | Software can read and write to these bits. |
| **read-only (r)** | Software can only read these bits. |
| **write only (w)** | Software can only write to this bit. Reading the bit returns a meaningless value. |
| **read/write once (rwo)** | Software can only write once to this bit but can read it at any time. Only a reset can return this bit to its reset value. |
| **read/clear (rc_w1)** | Software can read and clear this bit by writing 1. Writing '0' has no effect on the bit value. |
| **read/clear (rc_w0)** | Software can read and clear this bit by writing 0. Writing '1' has no effect on the bit value. |
| **read/set (rs)** | Software can read and set this bit. Writing '0' has no effect on the bit value. |
| **read/clear by read (rc_r)** | Software can read this bit. Reading this bit automatically clears it to '0'. Writing '0' has no effect on the bit value. |
| **Reserved (Res.)** | Reserved bit, must be kept at reset value. |

# 2 Central processing unit (CPU)

## 2.1 CPU introduction

The CPU has an 8-bit architecture. Six internal registers allow efficient data manipulations. 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).

## 2.2 CPU registers

The six CPU registers are shown in the programming model in *Figure 1*. Following an interrupt, the registers are pushed onto the stack in the order shown in *Figure 2*. They are popped from stack in the reverse order. The interrupt routine must therefore handle it, if needed, through the POP and PUSH instructions.

### 2.2.1 Description of CPU registers

**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.

**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.

**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 1. Programming model**
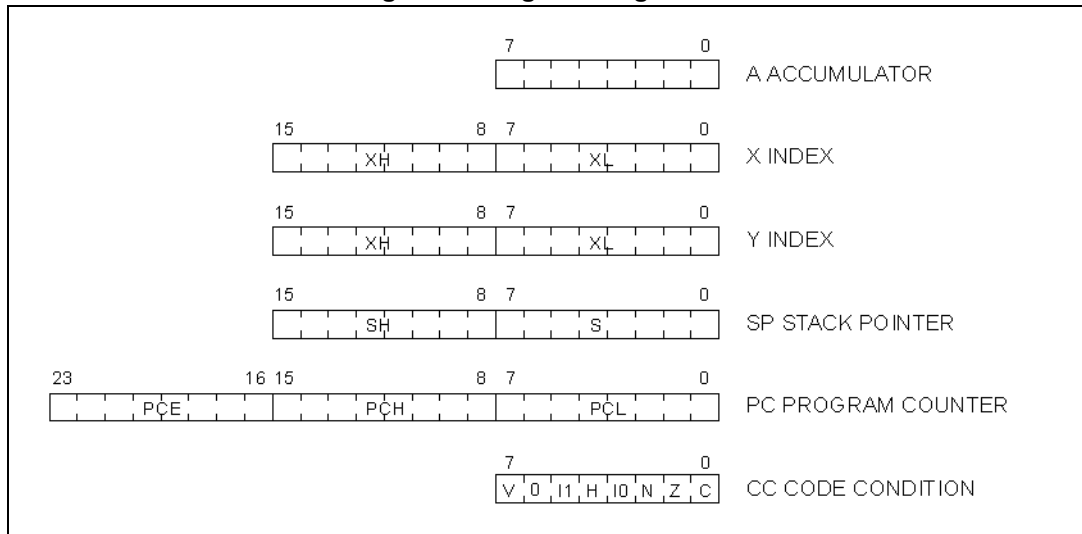


**Stack pointer (SP)**

The stack pointer is a 16-bit register. It contains the address of the next free location of the stack. Depending on the product, the most significant bits can be forced to a preset value.

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.

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 for C users. If you use your own linker file or startup file, make sure the stack pointer is initialized properly (with the address given in the datasheets). For applications written in assembler, you can use either the startup function provided by ST or write your own by initializing the stack pointer with the correct address.

The stack pointer is decremented after data has been pushed onto the stack and incremented after data is popped from the stack. It is up to the application to ensure that the lower limit is not exceeded.

A subroutine call occupies two or three locations. An interrupt occupies nine locations to store all the internal registers (except SP). For more details refer to *Figure 2*.

*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 2. Stacking order**
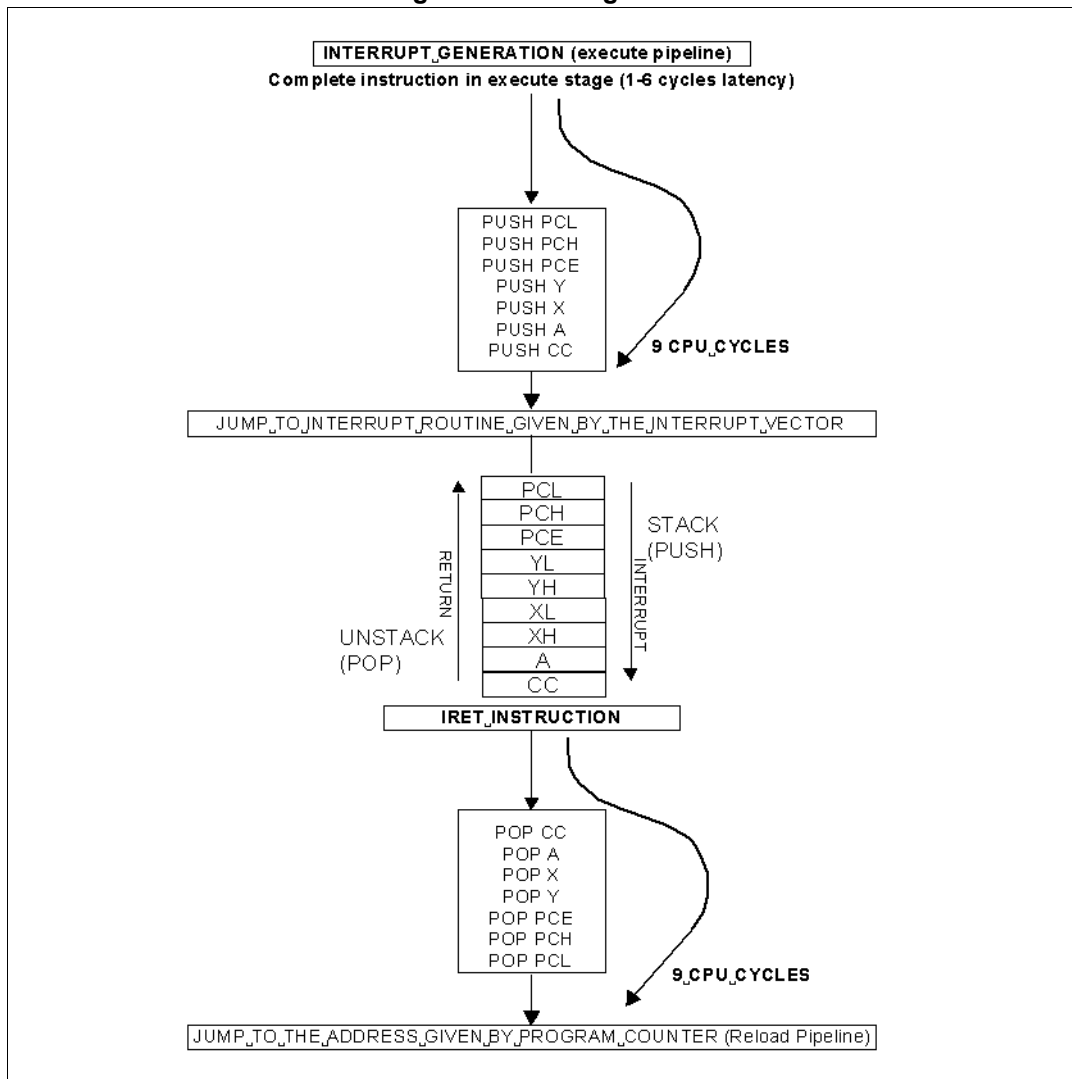


## Condition code register (CC)

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. See 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 interruptability level as shown in *Table 2*. 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 entering an interrupt service routine.

**Table 2. Interrupt levels**

| Interruptability | Priority | I1 | I0 |
|---|---|---|---|
| Interruptable main | Lowest | 1 | 0 |
| Interruptable level 1 | | 0 | 1 |
| Interruptable level 2 | | 0 | 0 |
| Non interruptable | Highest | 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

See Flag I1.

- 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 a 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. See 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. See 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.

Example: Addition

$B5 + $94 = "C" + $49 = $149

```
  C    7                   0
 ┌─┐  ┌─┬─┬─┬─┬─┬─┬─┬─┐
 │0│  │1│0│1│1│0│1│0│1│
 └─┘  └─┴─┴─┴─┴─┴─┴─┴─┘

     C    7                   0
    ┌─┐  ┌─┬─┬─┬─┬─┬─┬─┬─┐
+   │0│  │1│0│0│1│0│1│0│0│
    └─┘  └─┴─┴─┴─┴─┴─┴─┴─┘

     C    7                   0
    ┌─┐  ┌─┬─┬─┬─┬─┬─┬─┬─┐
=   │1│  │0│1│0│0│1│0│0│1│
    └─┘  └─┴─┴─┴─┴─┴─┴─┴─┘
```

### 2.2.2 STM8 CPU register map

The CPU registers are mapped in the STM8 address space as shown in *Table 3*. These registers can only be accessed by the debug module but not by memory access instructions executed in the core.

**Table 3. CPU register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | A | MSB | - | - | - | - | - | - | LSB |
| 0x01 | PCE | MSB | - | - | - | - | - | - | LSB |
| 0x02 | PCH | MSB | - | - | - | - | - | - | LSB |
| 0x03 | PCL | MSB | - | - | - | - | - | - | LSB |
| 0x04 | XH | MSB | - | - | - | - | - | - | LSB |
| 0x05 | XL | MSB | - | - | - | - | - | - | LSB |
| 0x06 | YH | MSB | - | - | - | - | - | - | LSB |
| 0x07 | YL | MSB | - | - | - | - | - | - | LSB |
| 0x08 | SPH | MSB | - | - | - | - | - | - | LSB |
| 0x09 | SPL | MSB | - | - | - | - | - | - | LSB |
| 0x0A | CC | V | 0 | I1 | H | I0 | N | Z | C |

## 2.3 Global configuration register (CFG_GCR)

### 2.3.1 Activation level

The MCU activation level is configured by programming the AL bit in the CFG_GCR register.

For information on the use of this bit refer to *Section 5.4: Activation level/low power mode control on page 43*.

### 2.3.2 SWIM disable

By default, after an MCU reset, the SWIM pin is configured to allow communication with an external tool for debugging or Flash/EEPROM programming. This pin can be configured by the application for use as a general purpose I/O. This is done by setting the SWD bit in the CFG_GCR register.

### 2.3.3 Description of global configuration register (CFG_GCR)

Address offset: 0x00
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | Reserved | | | AL | SWD |
| | | | | | | rw | rw |

Bits 7:2   Reserved

Bit 1   **AL**: Activation level

This bit is set and cleared by software. It configures main or interrupt-only activation.
0: Main activation level. An IRET instruction causes the context to be retrieved from the stack and the main program continues after the WFI instruction.
1: Interrupt-only activation level. An IRET instruction causes the CPU to go back to WFI/Halt mode without restoring the context.

Bit 0   **SWD**: SWIM disable

0: SWIM mode enabled
1: SWIM mode disabled

When SWIM mode is enabled, the SWIM pin cannot be used as general purpose I/O.

### 2.3.4 Global configuration register map and reset values

The CFG_GCR is mapped in the STM8 address space. Refer to the corresponding datasheets for the base address.

**Table 4. CFG_GCR register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CFG_GCR<br>Reset value | -<br>0 | -<br>0 | -<br>0 | -<br>0 | -<br>0 | -<br>0 | AL<br>0 | SWD<br>0 |

# 3 Single wire interface module (SWIM) and debug module (DM)

## 3.1 SWIM and DM introduction

In-circuit debugging mode or 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.

## 3.2 SWIM main features

- Based on an asynchronous, high sink (8 mA), open-drain, bidirectional communication.
- Allows reading or writing any part of memory space.
- Access to CPU registers (A, X, Y, CC, SP). They are memory mapped for read or write access.
- Non intrusive read/write on the fly to the RAM and peripheral registers.
- Device reset capability with status flag in the *Reset status register (RST_SR)*.

SWIM pin can be used as a standard I/O with some restrictions if you also want to use it for debug. The most secure way is to provide on the PCB a strap option.

**Figure 3. SWIM pin connection**
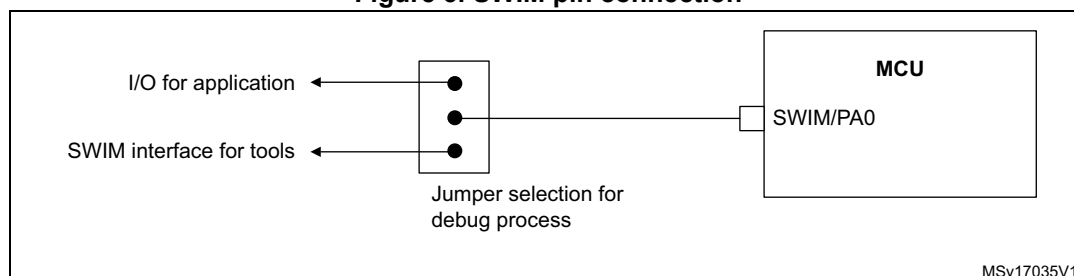


## 3.3 SWIM modes

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

1. **OFF**: Default state after power-on reset. The SWIM pin cannot be used by the application as an I/O.
2. **I/O**: This state is entered by software writing to the SWD bit in the *Global configuration register (CFG_GCR)*. In this state, the SWIM pin can be used by the application as a standard I/O pin. In case of a reset, the SWIM goes back to OFF mode.
3. **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 STM8 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 a description of the SWIM and Debug module (DM) registers. Refer also to the Description of global configuration register (CFG_GCR) on page 23.*

There are two important considerations to highlight for the devices where the NRST pin is not present:

- If the SWIM pin should be used with the I/O pin functionality, it is recommended to add a ~5 seconds delay in the firmware before changing the functionality on the pin with SWIM functions. This action allows the user to set the device into SWIM mode after the device power on and to be able to reprogram the device. If the pin with SWIM functionality is set to I/O mode immediately after the device reset, the device is unable to connect through the SWIM interface and it will be locked forever (if the NRST pin is not available on the package). This initial delay can be removed in the final (locked) code.

- Their program memory must contain a valid program loop. If the device's memory is empty, the program continues into non-existing memory space and executes invalid opcode; this causes the device to reset (reading of non-existing memory is random content). This behavior might lead to periodic device resets and to a difficulty to connect to the device through the SWIM interface.

## 3.4 Debug module (DM)

The debug module (DM) allows the developer to perform certain debugging tasks via SWIM interface without using an emulator. This is useful mainly for in-circuit debugging. The main features of the DM are:

- Two conditional breakpoints (break on instruction fetch, data read or write, stack access)
- Software breakpoint control
- Step mode
- External Stall capability on WOTF command in SWIM mode
- Watchdog and peripherals control
- DM Version identification capability
- Interrupt Vector Table selection

*Note:* *Refer to the STM8 SWIM communication protocol and debug module user manual (UM0470) for a description of the SWIM and DM registers.*

# 4 Flash program memory and data EEPROM

## 4.1 Flash and EEPROM introduction

The embedded Flash program memory and data EEPROM memories are controlled by a common set of registers. Using these registers, the application can program or erase memory contents and set write protection. The application can also program the device option bytes.

## 4.2 Flash and EEPROM glossary

- Block

  A block is a set of bytes that can be programmed or erased in one single programming operation. Operations that are performed at block level are faster than standard programming and erasing. Refer to *Table 5* for the details on block size.

- Page

  A page is a set of blocks.

  Dedicated option bytes can be used to configure, by increments of one page, the size of the user boot code anddata EEPROM.

## 4.3 Main Flash memory features

- **Low density STM8L001xx and STM8L101xx** EEPROM is divided into two memory arrays (see *Section 4.4: Memory organization* for details on low density STM8L001xx and STM8L101xx memory mapping):

  – Up to 8 Kbytes of embedded Flash program including up to 2 Kbytes of data EEPROM. Data EEPROM and Flash program areas can be write protected independently by using the memory access security mechanism (MASS).

  – 64 option bytes (one block) of which 5 bytes are already used for the device.

- 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.

- In-application programming (IAP) and in-circuit programming (ICP) capabilities

- 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).

## 4.4 Memory organization

Low density STM8L001xx and STM8L101xx Flash program memory is divided into 128 pages of 64 bytes each. It is organized in 32-bit words (4 bytes per word).

The memory array is divided into three areas:

- The user boot code area (UBC)
- The data EEPROM (DATA)
- The main program area

The first two pages of the Flash program memory (starting from address 0x00 8000) contain the interrupt vectors.

The devices also feature one block of option bytes (64 bytes) located in the separate memory array.

See *Figure 4* for a description of the memory organization.

**Figure 4. Low density STM8L001xx and STM8L101xx**
**Flash program and data EEPROM organization**
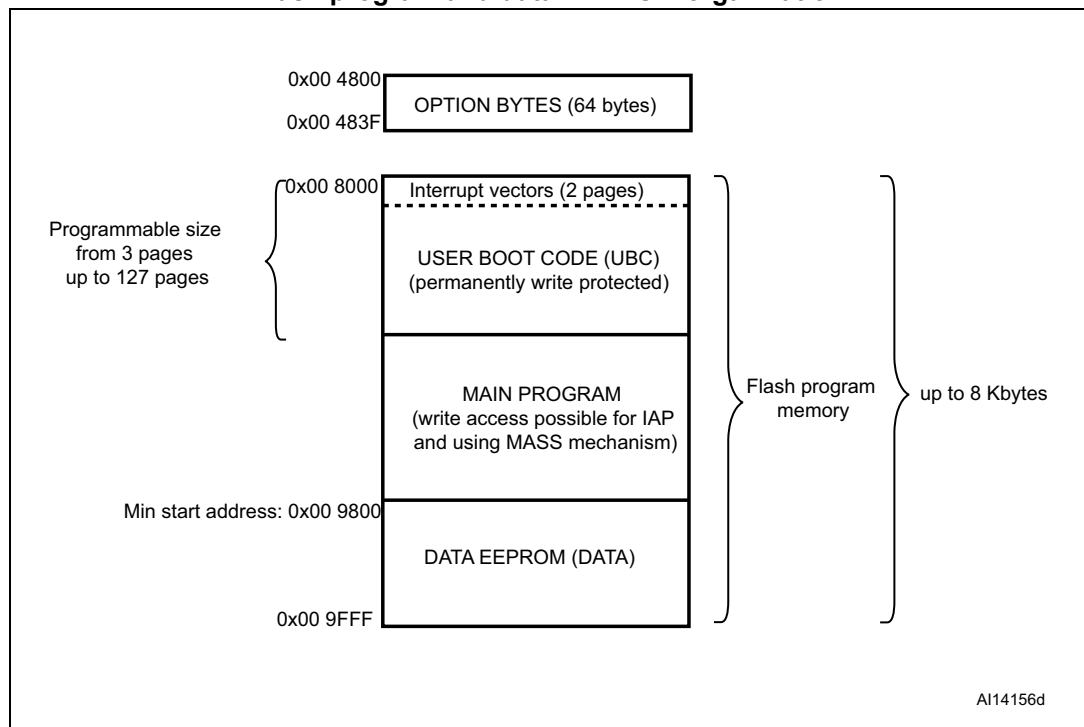


### 4.4.1 User boot area (UBC)

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 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 obtained by reading the UBC option byte.

The size of the UBC area can be configured in 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 address 0x00 8000.

The minimum meaningful size of the UBC area is of 3 pages of which 2 are used to store the interrupt vectors.

The maximum size of the boot area is 127 pages ranging from address 0x00 8000 to 0x00 9FFF(including the interrupt vectors). In this case, no memory space is left  for the main program area .

Refer to *Figure 5* for a description of the UBC area memory mapping and to the option byte section in the datasheets for more details on the UBC option byte.

**Figure 5. UBC area size definition for low density**
**STM8L001xx and STM8L101xx devices[1]**
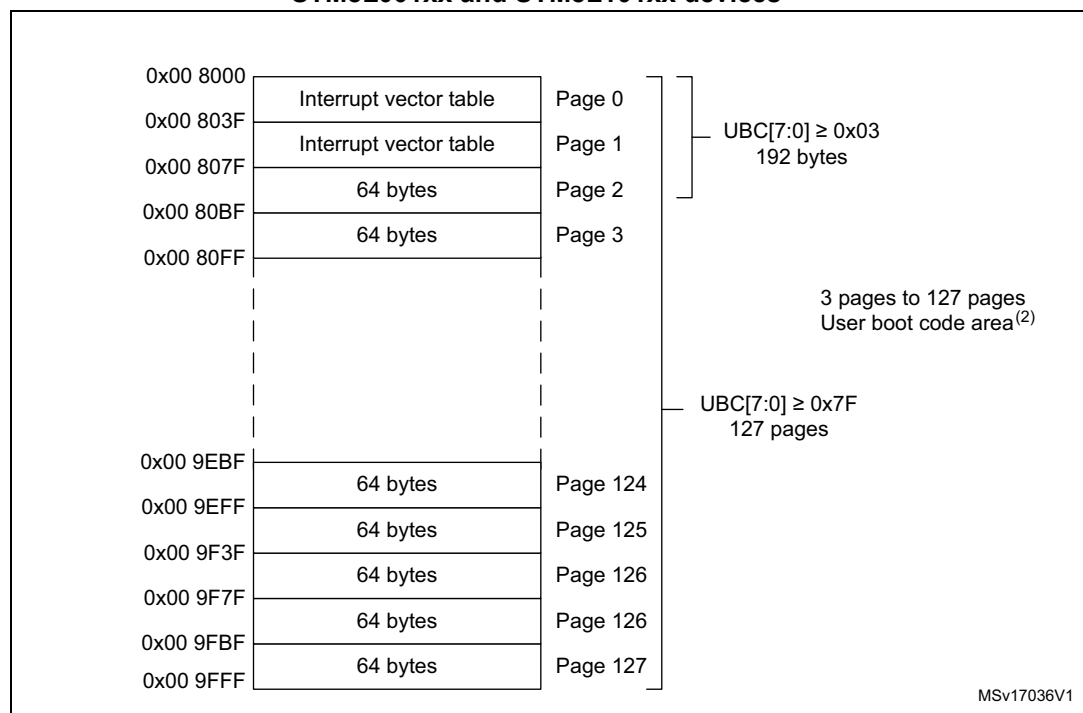


1. UBC[7:0]= 0x00 means no memory space is allocated for the UBC area.

2. Page 127 is reserved for data EEPROM.

### 4.4.2 Data EEPROM (DATA)

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

The size of the DATA area can be configured through the DATA option byte (DATASIZE) in ICP mode. This option byte specifies the number of pages (64 bytes) starting from address 0x00 9FFF.

The maximum size of the DATA area is of 2 Kbytes, corresponding to start address 0x00 9800 (see *Figure 4*) i.e. up to 32 pages of 64 bytes can be defined as data EEPROM.

### 4.4.3 Main program area

The main program is the area which starts at the end of the UBC and ends at address 0x00 9FFF. It is used to store the application code (see *Figure 4*).

### 4.4.4 Option bytes

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 only in ICP/SWIM mode with OPT bit of the FLASH_CR2 register set to 1 (see *Section 4.8.2: Flash control register 2 (FLASH_CR2)*).

Refer to the option byte section in the datasheet for more information on option bytes, and to the STM8 SWIM protocol and debug module user manual (UM0470) for details on how to program them.

## 4.5 Memory protection

### 4.5.1 Readout protection

Readout protection is selected by programming the ROP option byte to 0xAA. When readout protection is enabled, reading or modifying the Flash program memory and DATA area in 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.

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

Refer to *Table 6: Memory access versus programming method* for details on memory access when readout protection is enabled or disabled.

### 4.5.2 Memory access security system (MASS)

After reset, the main program and DATA areas (when they exist) are protected against 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 write protected (see *Section 4.4.1: User boot area (UBC)*).

Once the memory has been modified, it is recommended to enable the write protection again to protect the memory content against 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 (see

*Section 4.8.3: Flash program memory unprotecting key register (FLASH_PUKR)*). 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 of the main program area:

1. Write a 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 key available on the data bus is incorrect, the FLASH_PUKR register remains locked until the next reset. Any new write commands sent to this address are 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 key available on the data bus is incorrect, the write protection on program memory remains locked until the next reset. Any new write commands sent to this address is discarded.

5. If the second hardware key is correct, the main program memory is write unprotected and the PUL bit of the FLASH_IAPSR is set (see *Section 4.8.5: Flash status register (FLASH_IAPSR)*) register.

Before starting programming, the application must verify that PUL bit is effectively set. The application can 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 (see *Section 4.8.6: Flash register map and reset values*). 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 of 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 key available on the data bus 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 key available on the data bus 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 (see *Section 4.8.5: Flash status register (FLASH_IAPSR)*).

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

## 4.6 Memory programming

The main program memory, and the DATA area must be unlocked before attempting to perform any program operation. The unlock mechanism depends on the memory area to be programmed as described in *Section 4.5.2: Memory access security system (MASS)*.

### 4.6.1 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.. The application stops for the duration of the byte program operation.

To erase a byte, simply write 0x00 at 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 are 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. On the contrary if the word is empty, no erase operation is performed and the programming time is shorter (see $t_{PROG}$ in *Table "Flash program memory" in the 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 (see *Section 4.8.1: Flash control register 1 (FLASH_CR1)*). The programming time is consequently fixed and equal to the sum of the erase and write time (see $t_{PROG}$ in *Table "Flash program memory" in the 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 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.*

### 4.6.2 Word programming

A word write operation allows an entire 4-byte word to be programmed in one shot, thus minimizing the programming time.

As for byte programming, word operation is available both for the main program memory and data EEPROM.

To program a word, the WPRG bit in the FLASH_CR2 register must be previously set to enable word programming mode (see *Section 4.8.2: Flash control register 2 (FLASH_CR2)*). 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.

### 4.6.3 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. Refer to *Table 5* for details on the block size according to the devices.

Block operations can be performed both to the main program memory and DATA area. They are 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 in one shot. The block is automatically erase before being programmed.

To program a whole block in standard mode, the PRG bit in the FLASH_CR2 register must be previously set to enable standard block programming (see *Section 4.8.2: Flash control register 2 (FLASH_CR2)*). 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 bytes of data to be latched. To start programming the whole block, all bytes of data 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.

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 fast block programming mode, the FPRG bit of the FLASH_CR2 registers must be previously set.

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 before the fast block program operation.

### Block erasing

A block erase allows a whole block to be erased.

To erase a whole block, the ERASE bit in the FLASH_CR2 register must be previously set to enable block erasing (see *Section 4.8.2: Flash control register 2 (FLASH_CR2)*). 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.

**Table 5. Block size**

| STM8 microcontroller family | Block size |
|---|---|
| Low density STM8L001xx and STM8L101xx | 64 bytes |

## 4.7 ICP (in-circuit programming) and IAP (in-application programming)

The in-circuit programming (ICP) method is used to update the entire content of the memory, using the SWIM interface to load the user application into the microcontroller. ICP offers quick and efficient design iterations and eliminates unnecessary package handling or socketing of devices. The SWIM interface (single wire interface module) uses the SWIM pin to connect to the programming tool.

In contrast to the ICP method, in-application programming (IAP) can use any communication interface supported by the microcontroller (I/Os, I2C, SPI, USART...) to download the data to be programmed in the memory. IAP allows the Flash program memory content to be reprogrammed during application execution. Nevertheless, part of the application must have been previously programmed in the Flash program memory using ICP.

Refer to the STM8L Flash programming manual (PM0054) and STM8 SWIM protocol and debug manual (UM0470) for more information on programming procedures.

## Table 6. Memory access versus programming method[1]

| Mode | ROP | Memory Area | Access from core |
|---|---|---|---|
| User mode, IAP, and bootloader | Readout protection enabled | Interrupt vectors | R/W[2]/E |
| | | User boot code area (UBC) | R/E |
| | | Main program | R/W/E[3] |
| | | Data EEPROM area (DATA) | R/W[4] |
| | | Option bytes | R |
| | Readout protection disabled | Interrupt vectors | R/W[2]/E |
| | | User boot code area (UBC) | R/E[5] |
| | | Main program | R/W/E[3] |
| | | Data EEPROM area (DATA) | R/W[4] |
| | | Option bytes | R |
| SWIM active (ICP mode) | Readout protection enabled | Interrupt vectors | P |
| | | User boot code area (UBC) | P |
| | | Main program | P |
| | | Data EEPROM area (DATA) | P |
| | | Option bytes | P/W$_{ROP}$[6] |
| | Readout protection disabled | Interrupt vectors except for TRAP and TLI | R/W[2]/E |
| | | User boot code area (UBC) | R/E[5] |
| | | Main program | R/W/E[3] |
| | | Data EEPROM area (DATA) | R/W[4] |
| | | Option bytes | R/W |

1. R/W/E = Read, write, and execute
   R/E = Read and execute (write operation forbidden)
   R = Read (write and execute operations forbidden)
   P = The area cannot be accessed (read, execute and write operations forbidden)
   P/W$_{ROP}$ = Protected, write forbidden except for ROP option byte.

2. When no UBC area has been defined, the interrupt vectors can be modified in user/IAP mode.

3. 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. Unlocking can only be done once between two resets. If incorrect keys are provided, the device must be reset and new keys programmed.

4. 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 incorrect keys are provided, another key program sequence can be performed without resetting the device.

5. To program the UBC area the application must first clear the UBC option byte.

6. When ROP is removed, the whole memory is erased, including option bytes.

## 4.8 Flash registers

### 4.8.1 Flash control register 1 (FLASH_CR1)

Address offset: 0x00
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | Reserved | | | IE | FIX |
| | | | | | | rw | rw |

Bits 7:2 Reserved

Bit 1 **IE**: Flash Interrupt enable
This bit is set and cleared by software.
0: Interrupt disabled
1: Interrupt enabled. An interrupt is generated if the EOP or WR_PG_DIS flag in the FLASH_IAPSR register is set.

Bit 0 **FIX**: Fixed Byte programming time
This bit is set and cleared by software.
0: Standard programming time of (1/2 $t_{prog}$) if the memory is already erased and $t_{prog}$ otherwise.
1: Programming time fixed at $t_{prog}$.

### 4.8.2 Flash control register 2 (FLASH_CR2)

Address offset: 0x01
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| OPT | WPRG | ERASE | FPRG | Reserved | | | PRG |
| rw | rw | rw | rw | r | | | rw |

Bit 7  **OPT**: Write option bytes
This bit is set and cleared by software.
0: Write access to option bytes disabled
1: Write access to option bytes enabled

Bit 6  **WPRG**: Word programming
This bit is set by software and cleared by hardware when the operation is completed.
0: Word program operation disabled
1: Word program operation enabled

Bit 5  **ERASE**[(1)]: Block erasing
This bit is set by software and cleared by hardware when the operation is completed.
0: Block erase operation disabled
1: Block erase operation enabled

Bit 4  **FPRG**[(1)]**:** Fast block programming
This bit is set by software and cleared by hardware when the operation is completed.
0: Fast block program operation disabled
1: Fast block program operation enabled

Bits 3:1  Reserved

Bit 0  **PRG**: Standard block programming
This bit is set by software and cleared by hardware when the operation is completed.
0: Standard block programming operation disabled
1: Standard block programming operation enabled (automatically first erasing)

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

### 4.8.3 Flash program memory unprotecting key register (FLASH_PUKR)

Address offset: 0x02
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| MASS_PRG KEYS | | | | | | | |
| rw | | | | | | | |

Bits 7:0  **PUK [7:0]**: Main program memory unlock keys
This byte is written by software (all modes). It returns 0x00 when read.
Refer to *Enabling write access to the main program memory on page 29* for the description of main program area write unprotection mechanism.

### 4.8.4 Data EEPROM unprotection key register (FLASH_DUKR)

Address offset: 0x03
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | MASS_DATA KEYS | | | | |
| | | | rw | | | | |

Bits 7:0 **DUK[7:0]**: Data EEPROM write unlock keys

This byte is written by software (all modes). It returns 0x00 when read.
Refer to *Enabling write access to the DATA area on page 30* for the description of main program area write unprotection mechanism.

### 4.8.5 Flash status register (FLASH_IAPSR)

Address offset: 0x04
Reset value: 0xX0 where X is undefined

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | Reserved | | | DUL | EOP | PUL | WR_PG_DIS |
| | | | | rc_w0 | rc_r | rc_w0 | rc_r |

Bits 7:4 Reserved

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 has been disabled by writing the correct MASS keys

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

This bit is set by hardware. It is 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 1 **PUL**: Flash Program memory unlocked flag

This bit is set by hardware and cleared by software by programming it to 0.
0: Write protection of main Program area enabled
1: Write protection of main Program area has been disabled by writing the correct MASS keys.

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

This bit is set by hardware and cleared by software by reading the register.
0: No WR_PG_DIS event occurred
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.

### 4.8.6 Flash register map and reset values

For details on the Flash register boundary addresses, refer to the general hardware register map in the datasheets.

**Table 7. Flash register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | FLASH_CR1 | - | - | - | - | - | - | IE | FIX |
|  | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x01 | FLASH_CR2 | OPT | WPRG | ERASE | FPRG | - | - | - | PRG |
|  | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x02 | FLASH_PUKR | PUK7 | PUK6 | PUK5 | PUK4 | PUK3 | PUK2 | PUK1 | PUK0 |
|  | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x03 | FLASH_DUKR | DUK7 | DUK6 | DUK5 | DUK4 | DUK3 | DUK2 | DUK1 | DUK0 |
|  | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x04 | FLASH_IAPSR | - | - | - | - | DUL | EOP | PUL | WR_PG_DIS |
|  | Reset value | x | x | x | x | 0 | 0 | 0 | 0 |

# 5 Interrupt controller (ITC)

## 5.1 ITC introduction

- Management of hardware interrupts
  - External interrupt capability on all I/O pins with dedicated interrupt vector per port and dedicated flag per pin
  - Peripheral interrupt capability
- Management of software interrupt (TRAP)
- Nested or concurrent interrupt management with flexible interrupt priority and level management:
  - Up to 4 software programmable nesting levels
  - interrupt vectors fixed by hardware
  - 2 non maskable events: RESET, TRAP

This interrupt management is based on:

- Bit I1 and I0 of the CPU Condition Code register (CCR)
- Software priority registers (ITC_SPRx)
- Reset vector located at 0x00 8000 at the beginning of program memory. The Reset initialization routine is programmed in ROM by STMicroelectronics.
- Fixed interrupt vector addresses located at the high addresses of the memory map (0x00 8004 to 0x00 807C) sorted by hardware priority order.

## 5.2 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 vector (see *Table 8*). The processing flow is shown in *Figure 6*.

When an interrupt request has to be 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 CCR register are set according to the values in the ITC_SPRx registers corresponding to the serviced interrupt vector.
4. The PC is then loaded with the interrupt vector of the interrupt to service and the first instruction of the interrupt service routine is fetched (refer to the datasheet interrupt mapping table for details on vector addresses).

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.

**Table 8. Software priority levels**

| Software priority | Level | I1 | I0 |
|---|---|---|---|
| Level 0 (main) | Low ↓ High | 1 | 0 |
| Level 1 | | 0 | 1 |
| Level 2 | | 0 | 0 |
| Level 3 (= software priority disabled) | | 1 | 1 |

**Figure 6. Interrupt processing flowchart**
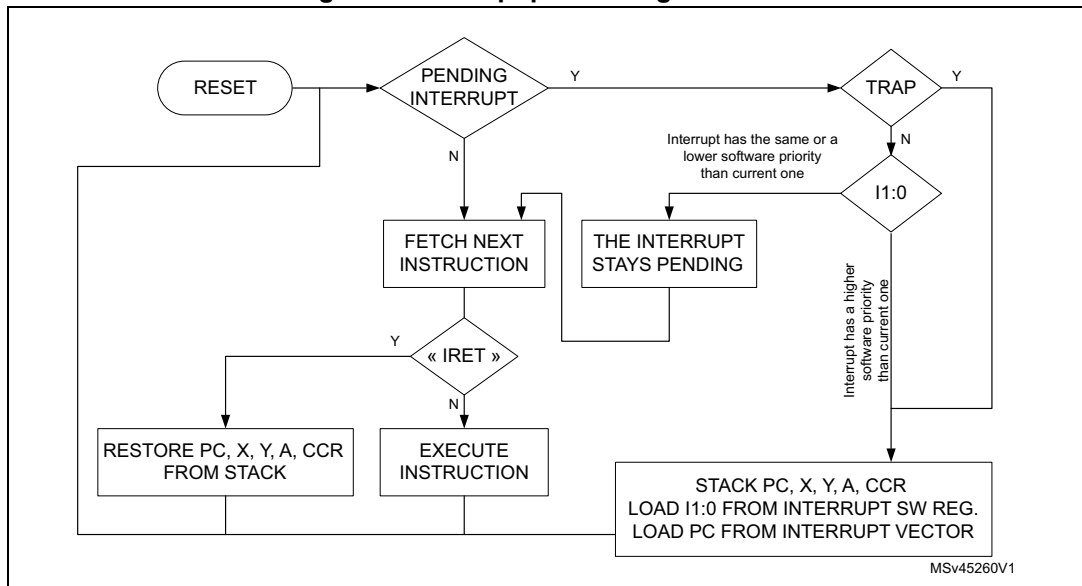


**Caution:** If the interrupt mask bits I0 and I1 are set within an interrupt service routine (ISR) with the instruction SIM, removal of the interrupt mask with RIM causes the software priority to be set to level 0.

To restore the correct priority when disabling and enabling interrupts inside an ISR, follow the procedures presented in *Table 8* for disabling and enabling the interrupts.

### 5.2.1 Servicing pending interrupts

Several interrupts can 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 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 RESET and TRAP are considered as having the highest software priority in the decision process.*

See *Figure 7* for a description of pending interrupt servicing process.

**Figure 7. Priority decision process**
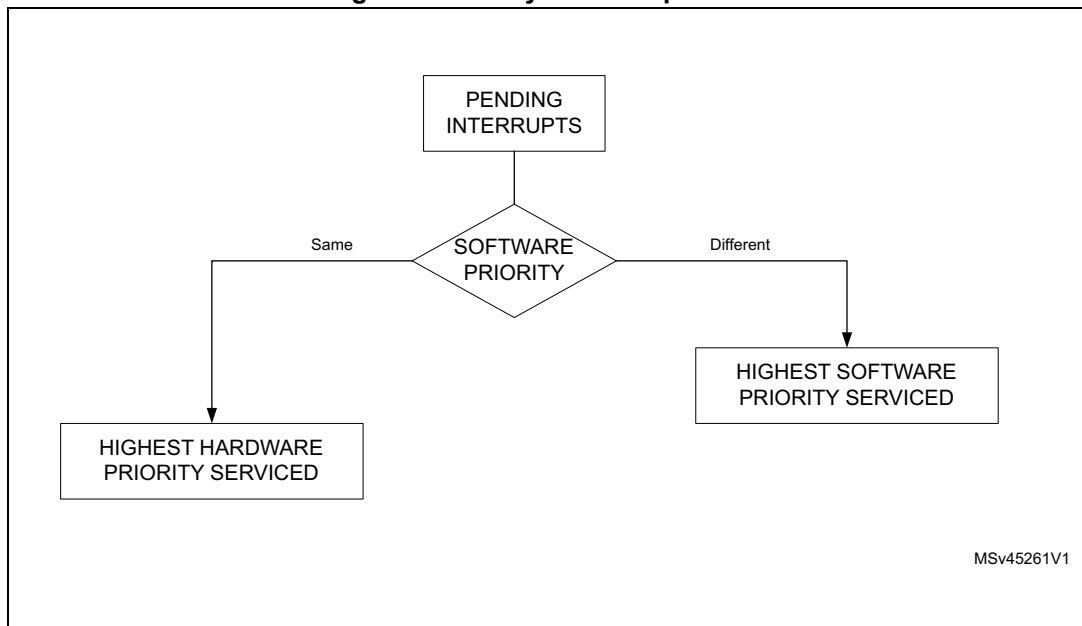


### 5.2.2 Interrupt sources

Two interrupt source types are managed by the STM8 interrupt controller:

- Non-maskable interrupts: RESET and TRAP
- Maskable interrupts: external interrupts or interrupts issued by internal peripherals

**Non-maskable interrupt sources**

Non-maskable interrupt sources are processed regardless of the state of bits I1 and I0 of the CCR register (see *Figure 6*). PC, X, Y, A and CCR registers are stacked only when a TRAP interrupt occurs. 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 (non-maskable software interrupt)

  This software interrupt source is serviced when the TRAP instruction is executed. It is serviced according to the flowchart shown in *Figure 6*.

  A TRAP interrupt does not allow the processor to exit from Halt mode.

- RESET

  The RESET interrupt source has the highest STM8 software and hardware priorities. This means that all the interrupts are disabled at the beginning of the reset routine. They must be re-enabled by the RIM instruction (see *Table 11: Dedicated interrupt instruction set*).

  A RESET interrupt allows the processor to exit from Halt mode.

See RESET chapter for more details on RESET interrupt management.

**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 Halt mode. The device sensitivity to external interrupts can be selected by software through the External Interrupt Control registers (EXTI_CRx).

  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 has been inactivated in the routine.

- Peripheral interrupts

  A few peripheral interrupts cause the MCU to wake up from Halt mode. See the interrupt vector table in the datasheet.

  A peripheral interrupt occurs when a specific flag is set in the peripheral status register and the corresponding enable bit is set in the peripheral control register.

  The standard sequence for clearing a peripheral interrupt performs an access to the status register followed by a read or write to an associated register. 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.

## 5.3 Interrupts and low power modes

All interrupts allow the processor to exit from Wait mode.

Only a Reset or an event allows the processor to exit from Low power wait mode. This mode is entered by executing a WFE instruction in Low power run mode. The wakeup by an event makes the system go back to Low power run mode (refer to *Section 9: Power management* for more details).

Only external and other specific interrupts allow the processor to exit from Halt and Active-halt mode (see wakeup from halt and wakeup from Active-halt).

When several pending interrupts are present while waking up from Halt mode, the first interrupt serviced can only be an interrupt with exit-from-Halt mode capability. It is selected through the decision process shown in *Figure 7*. If the highest priority pending interrupt cannot wake up the device from 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 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 Halt mode to Run mode, with the corresponding delay of $t_{WUH}$ as specified in the datasheet.

## 5.4     Activation level/low power mode control

The MCU activation level is configured by programming the AL bit in the CFG_GCR register (see global configuration register (CFG_GCR)).

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/Halt mode 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 mode (by executing WFI/HALT instruction), then the interrupt routine returns directly to 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 ISR in very simple applications. In more complex ones, an interrupt routine may relaunch 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 time, as these pins are not active, the MCU can return to 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.

## 5.5     Concurrent and nested interrupt management

STM8 devices feature two interrupt management modes:
*   Concurrent mode
*   Nested mode

### 5.5.1 Concurrent interrupt management mode

In this mode, all interrupts are interrupt priority level 3 so that none of them can be interrupted, except by a RESET, or TRAP.

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 (same priority), and RESET.

*Figure 8* shows an example of concurrent interrupt management mode.

**Figure 8. Concurrent interrupt management**
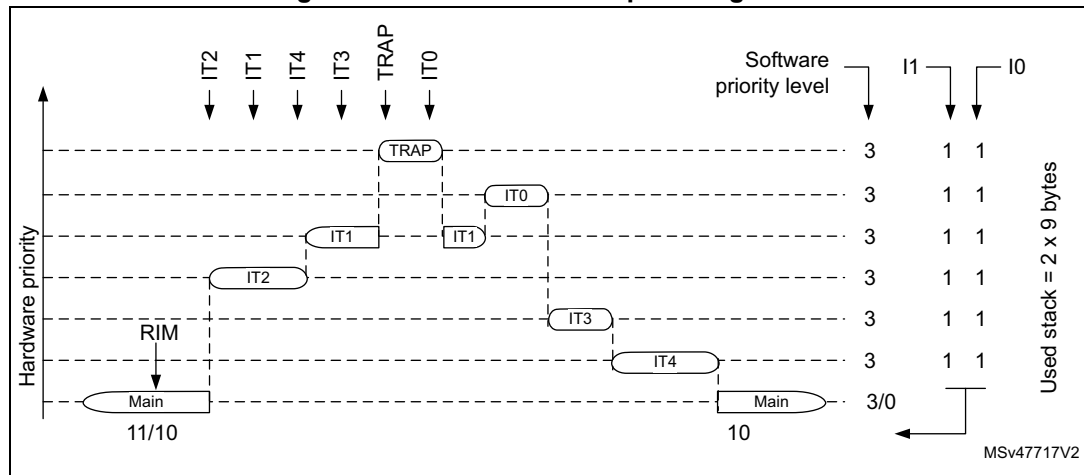
### 5.5.2 Nested interrupt management mode

In this mode, 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, and TRAP.

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 as I1 and I0 bits of the CCR register (see *Table 9*).

Level 0 can not 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 64h, the result is 44h.

The RESET and TRAP vectors have no software priorities. When one 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. See *Section 5.7* for the list of dedicated interrupt instructions.

*Figure 9* shows an example of nested interrupt management mode.

---

**Warning:** **A stack overflow may occur without notifying the software of the failure.**

---

**Table 9. Vector address map versus software priority bits**

| Vector address | ITC_SPRx bits |
|---|---|
| 0x00 8008h | I1_0 and I0_0 bits |
| 0x00 800Ch | I1_1 and I0_1 bits |
| ... | ... |
| 0x00 807Ch | I1_29 and I0_29 bits |

**Figure 9. Nested interrupt management**
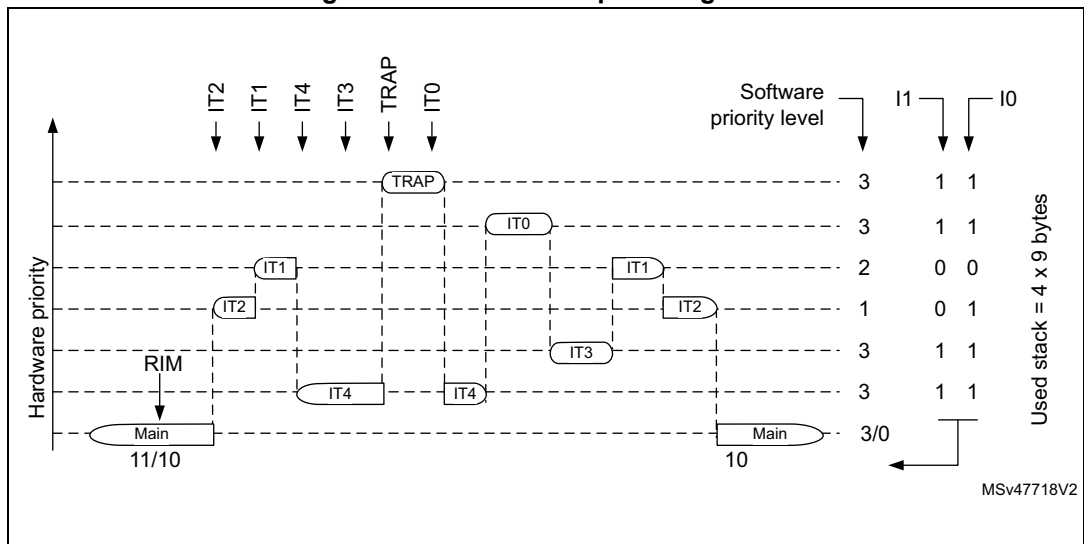
## 5.6 External interrupts

Ten interrupt vectors are dedicated to external Interrupt events:

- EXTIB - 8 lines on Port B: PB[7:0]
- EXTID - 8 lines on Port D: PD[7:0]
- EXTI0 - 4 lines on Port A/B/C/D, bit 0: Px[0]
- EXTI1 - 4 lines on Port A/B/C/D, bit 1: Px[1]
- EXTI2 - 4 lines on Port A/B/C/D, bit 2: Px[2]
- EXTI3 - 4 lines on Port A/B/C/D, bit 3: Px[3]
- EXTI4 - 4 lines on Port A/B/C/D, bit 4: Px[4]
- EXTI5 - 4 lines on Port A/B/C/D, bit 5: Px[5]
- EXTI6 - 4 lines on Port A/B/C/D, bit 6: Px[6]
- EXTI7 - 2 lines on Port B/D, bit 7: Px[7]

To generate an interrupt, the corresponding GPIO port must be configured in input mode with interrupts enabled. Refer to the register description in the GPIO chapter for details.

When an external interrupt occurs, the corresponding bit is set in the EXTI_SRx status register. This indicates a pending interrupt. Clearing this bit, writing a 1 in it, clears the corresponding pending external interrupt.

The interrupt sensitivity must be configured in the external interrupt control register 1 (EXTI_CR1), external interrupt control register 2 (EXTI_CR2), and external interrupt control register 3 (EXTI_CR3) (see *Section 5.9.5*, *Section 5.9.6* and *Section 5.9.7*).

**Table 10. External interrupt sensitivity**

| GPIO port | Interrupt sensitivity | Configuration register |
|---|---|---|
| EXTI0 to EXTI3 on port A, B, C and D | Falling edge and low level  Rising edge only  Falling edge only  Rising and falling edge | EXTI_CR1 |
| EXTI4 to EXTI7 of port A, B, C and D | | EXTI_CR2 |
| EXTIB and EXTID | | EXTI_CR3 |

## 5.7 Interrupt instructions

*Table 11* shows the interrupt instructions.

**Table 11. Dedicated interrupt instruction set**

| Instruction | New description | Function/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) | I1:0=11 ? | - | - | - | - | - | - |
| JRNM | Jump if I1:0<>11 | 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 | - | - | - | - | - | - |

**Table 11. Dedicated interrupt instruction set (continued)**

| Instruction | New description | Function/example | I1 | H | I0 | N | Z | C |
|---|---|---|---|---|---|---|---|---|
| RIM | Enable interrupt (level 0 set) | Load 10 in I1:0 of CCR | 1 | - | 0 | - | - | - |
| SIM | Disable interrupt (level 3 set) | Load 11 in I1:0 of CCR | 1 | - | 1 | - | - | - |
| TRAP | Software trap | Software NMI | 1 | - | 1 | - | - | - |
| WFI | Wait for interrupt | - | 1 | - | 0 | - | - | - |
| WFE | Wait for event | - | 1 | - | 0 | - | - | - |

## 5.8 Interrupt mapping

Refer to the corresponding device datasheet for the table of interrupt vector addresses.

## 5.9 ITC and EXTI registers

### 5.9.1 CPU condition code register interrupt bits (CCR)

Address: refer to the general hardware register map table in the datasheet.
Reset value: 0x28

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

Bits 5, 3[1]  **I[1:0]:** Software interrupt priority bits[2]

These two bits indicate the software priority of the current interrupt request. When an interrupt request occurs, the software priority of the corresponding vector is loaded automatically from the software priority registers (ITC_SPRx).

The I[1:0] bits can be also set/cleared by software using the RIM, SIM, HALT, WFI, IRET or PUSH/POP instructions (see *Figure 9: Nested interrupt management*).

| I1 | I0 | Priority | Level |
|----|----|----------|-------|
| 1 | 0 | Level 0 (main) | |
| 0 | 1 | Level 1 | Low |
| 0 | 0 | Level 2 | |
| 1 | 1 | Level 3 (= software priority disabled*) | High |

1. Refer to the central processing section for details on the other CCR bits.

2. TRAP and RESET events can interrupt a level-3 program.

### 5.9.2 Software priority register x (ITC_SPRx)

Address offset: 0x00 to 0x07
Reset value: 0xFF

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| ITC_SPR1 | Reserved | | Reserved | | VECT1SPR[1:0] | | Reserved | |
| ITC_SPR2 | VECT7SPR[1:0] | | VECT6SPR[1:0] | | Reserved | | VECT4SPR[1:0] | |
| ITC_SPR3 | VECT11SPR[1:0] | | VECT10SPR[1:0] | | VECT9SPR[1:0] | | VECT8SPR[1:0] | |
| ITC_SPR4 | VECT15SPR[1:0] | | VECT14SPR[1:0] | | VECT13SPR[1:0] | | VECT12SPR[1:0] | |
| ITC_SPR5 | VECT19SPR[1:0] | | VECT18SPR[1:0] | | Reserved | | Reserved | |
| ITC_SPR6 | Reserved | | VECT22SPR[1:0] | | VECT21SPR[1:0] | | VECT20SPR[1:0] | |
| ITC_SPR7 | VECT27SPR[1:0] | | VECT26SPR[1:0] | | VECT25SPR[1:0] | | Reserved | |
| ITC_SPR8 | Reserved | | | | VECT29SPR[1:0] | | VECT28SPR[1:0] | |
| | rw | | | | rw | rw | rw | rw |

Bits 7:0 **VECTxSPR[1:0]:** Vector x software priority bits

These eight read/write registers (ITC_SPR1 to ITC_SPR8) are written by software to define the software priority of each interrupt vector.

The list of vectors is given in *Table 9: Vector address map versus software priority bits*.

Refer to *Section 5.9.1: CPU condition code register interrupt bits (CCR)* for the values to be programmed for each priority.

Reserved

ITC_SPR8 bits 7:4 are forced to 1 by hardware.

*Note: It is forbidden to write 10 (priority level 0). If 10 is written, the previous value is kept and the interrupt priority remains unchanged.*

### 5.9.3 External interrupt control register 1 (EXTI_CR1)

Address offset: 0x00

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| P3IS[1:0] | | P2IS[1:0] | | P1IS[1:0] | | P0IS[1:0] | |
| rw | | rw | | rw | | rw | |

Bits 7:6 **P3IS[1:0]:** Portx bit 3 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of bit 3 of Port A, B, C and/or D external interrupts.
00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 5:4 **P2IS[1:0]:** Portx bit 2 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of bit 2 of Port A, B, C and/or D external interrupts.
00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 3:2 **P1IS[1:0]:** Portx bit 1 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of bit 1 of Port A, B, C and/or D external interrupts.
00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 1:0 **P0IS[1:0]:** Portx bit 0 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of bit 0 of Port A, B, C and/or D external interrupts.
00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

### 5.9.4 External interrupt control register 2 (EXTI_CR2)

Address offset: 0x01
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| P7IS[1:0] | | P6IS[1:0] | | P5IS[1:0] | | P4IS[1:0] | |
| rw | | rw | | rw | | rw | |

Bits 7:6 **P7IS[1:0]:** Portx bit 7 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the bit 7 of Port B and/or D external interrupts.
00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 5:4 **P6IS[1:0]:** Portx bit 6 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the bit 6 of Port A, B, C and/or D external interrupts.
00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 3:2 **P5IS[1:0]:** Portx bit 5 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the bit 5 of Port A, B, C and/or D external interrupts.
00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 1:0 **P4IS[1:0]:** Portx bit 4 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the bit 4 of Port A, B, C and/or D external interrupts.
00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

### 5.9.5 External interrupt control register 3 (EXTI_CR3)

Address offset: 0x02
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | Reserved | | | PDIS[1:0] | | PBIS[1:0] | |
| | | | | rw | | rw | |

Bits 7:4  Reserved

Bits 3:2  **PDIS[1:0]:** Port D external interrupt sensitivity bits
These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the Port D external interrupts, when EXTID for Port D[3:0] and/or Port D[7:4] is enabled.
00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 1:0  **PBIS[1:0]:** Port B external interrupt sensitivity bits
These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the Port B external interrupts, when EXTIB for Port B[3:0] and/or PortB[7:4] is enabled.
00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

### 5.9.6 External interrupt status register 1 (EXTI_SR1)

Address offset: 0x03
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| P7F | P6F | P5F | P4F | P3F | P2F | P1F | P0F |
| rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

Bits 7:0  **PxF:** Port A/B/C/D bit x external interrupt flag
These bits are set by hardware when an interrupt event occurs on the corresponding pin.They are cleared by writing a '1' by software.
0: No interrupt
1: External interrupt pending

## 5.9.7 External interrupt status register 2 (EXTI_SR2)

Address offset: 0x04
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | Reserved | | | PDF | PBF |
| | | | | | | rc_w1 | rc_w1 |

Bits 7:2 Reserved

Bit 1 **PDF:** Port D external interrupt flag
This bit is set by hardware when an interrupt event occurs on the corresponding pin. It is cleared by writing a '1' by software.
0: No interrupt
1: External interrupt pending

Bit 0 **PBF:** Port B external interrupt flag
This bit is set by hardware when an interrupt event occurs on the corresponding pin.It is cleared by writing a '1' by software.
0: No interrupt
1: External interrupt pending

## 5.9.8 External interrupt port select register (EXTI_CONF)

Address offset: 0x05
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | Reserved | | PDHIS | PDLIS | PBHIS | PBLIS |
| | | | | rw | rw | rw | rw |

Bits 7:4 Reserved

Bit 3 **PDHIS:** Port D[7:4] external interrupt select
These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). It selects pins PD[7:4] for EXTID interrupt.
0: PD[7:4] are used for EXTI7-EXTI4 interrupt generation
1: PD[7:4] are used for EXTID interrupt generation

Bit 2 **PDLIS:** Port D[3:0] external interrupt select

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). It selects pins PD[3:0] for EXTID interrupt.

0: PD[3:0] are used for EXTI3-EXTI0 interrupt generation

1: PD[3:0] are used for EXTID interrupt generation

Bit 1 **PBHIS:** Port B[7:4] external interrupt select

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). It selects pins PB[7:4] for EXTIB interrupt.

0: PB[7:4] are used for EXTI7-EXTI4 interrupt generation

1: PB[7:4] are used for EXTIB interrupt generation

Bit 0 **PBLIS:** Port B[3:0] external interrupt select

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). It selects pins PB[3:0] for EXTIB interrupt.

0: PB[3:0] are used for EXTI3-EXTI0 interrupt generation

1: PB[3:0] are used for EXTIB interrupt generation

### 5.9.9 ITC and EXTI register map and reset values

**Table 12. ITC and EXTI register map**

| Add. offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| **ITC-SPR block[1]** | | | | | | | | | |
| 0x00 | ITC_SPR1 | - | - | Reserved | Reserved | VECT1 SPR1 | VECT1 SPR0 | - | - |
| | Reset value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0x01 | ITC_SPR2 | VECT7 SPR1 | VECT7 SPR0 | VECT6 SPR1 | VECT6 SPR0 | - | - | VECT4 SPR1 | VECT4 SPR0 |
| | Reset value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0x02 | ITC_SPR3 | VECT11 SPR1 | VECT11 SPR0 | VECT10 SPR1 | VECT10 SPR0 | VECT9 SPR1 | VECT9 SPR0 | VECT8 SPR1 | VECT8 SPR0 |
| | Reset value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0x03 | ITC_SPR4 | VECT15 SPR1 | VECT15 SPR0 | VECT14 SPR1 | VECT14 SPR0 | VECT13 SPR1 | VECT13 SPR0 | VECT12 SPR1 | VECT12 SPR0 |
| | Reset value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0x04 | ITC_SPR5 | VECT19 SPR1 | VECT19 SPR0 | - | - | - | - | - | - |
| | Reset value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0x05 | ITC_SPR6 | - | - | VECT22 SPR1 | VECT22 SPR0 | VECT21 SPR1 | VECT21 SPR0 | VECT20 SPR1 | VECT20 SPR0 |
| | Reset value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0x06 | ITC_SPR7 | VECT27 SPR1 | VECT27 SPR0 | VECT26 SPR1 | VECT26 SPR0 | VECT25 SPR1 | VECT25 SPR0 | - | - |
| | Reset value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0x07 | ITC_SPR8 | - | - | - | - | VECT29 SPR1 | VECT29 SPR0 | VECT28 SPR1 | VECT28 SPR0 |
| | Reset value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **ITC-EXTI block[2]** | | | | | | | | | |
| 0x00 | EXTI_CR1 | P3IS1 | P3IS0 | P2IS1 | P2IS0 | P1IS1 | P1IS0 | P0IS1 | P0IS0 |
| | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x01 | EXTI_CR2 | P7IS1 | P7IS0 | P6IS1 | P6IS0 | P5IS1 | P5IS0 | P4IS1 | P4IS0 |
| | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x02 | EXTI_CR3 | - | - | - | - | PDIS1 | PDIS0 | PBIS1 | PBIS0 |
| | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x03 | EXTI_SR1 | P7F | P6F | P5F | P4F | P3F | P2F | P1F | P0F |
| | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x04 | EXTI_SR2 | - | - | - | - | - | - | PDF | PBF |
| | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x05 | EXTI_CONF | - | - | - | - | PDHIS | PDLIS | PBHIS | PBLIS |
| | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

1. The address offsets are expressed for the ITC-SPR block base address (see Table CPU/SWIM/debug module/interrupt controller registers in the datasheet).

2. The address offsets are expressed for the ITC-EXTI block base address (see Table General hardware register map in the datasheet).

# 6 Power supply

The MCU has one power supply, $V_{DD}/V_{SS}$ (1.65 to 3.6 V), to supply both the I/Os and the internal voltage regulator. The voltage regulator has 2 modes: Main Voltage Regulator mode (MVR) and Low Power Voltage Regulator mode (LPVR).

When entering Halt or Active-halt modes (described in *Section 9.4.2: Halt mode* and *Section 9.4.3: Active-halt mode*), the system automatically switches from the MVR to the LPVR in order to reduce current consumption.

The power-on reset (POR) guarantees a safe reset when the MCU is powered on.

Refer to the electrical characteristics section in the datasheet for details on the operating range.

**Figure 10. Power supply overview**
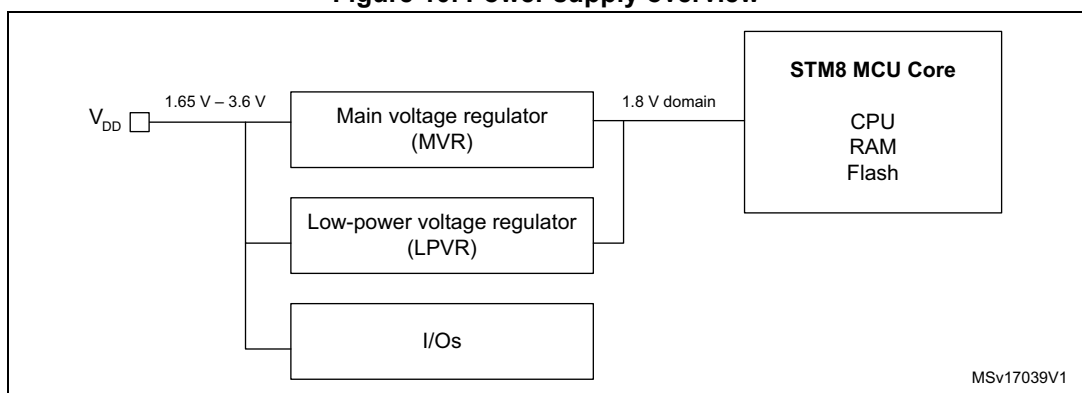


*Note: The different power modes are described in Section 9: Power management on page 66.*

# 7 Reset (RST) and voltage detection

There are five reset sources:

- External reset through the NRST pin (this pin can also be configured as general purpose output)
- Power-on reset (POR)
- Independent watchdog reset (IWDG)
- SWIM reset
- Illegal opcode reset

These sources act on the $\overline{\text{RESET}}$ pin. The RESET service routine vector is fixed at address 0x8000 in the memory map.

**Figure 11. Reset circuit**
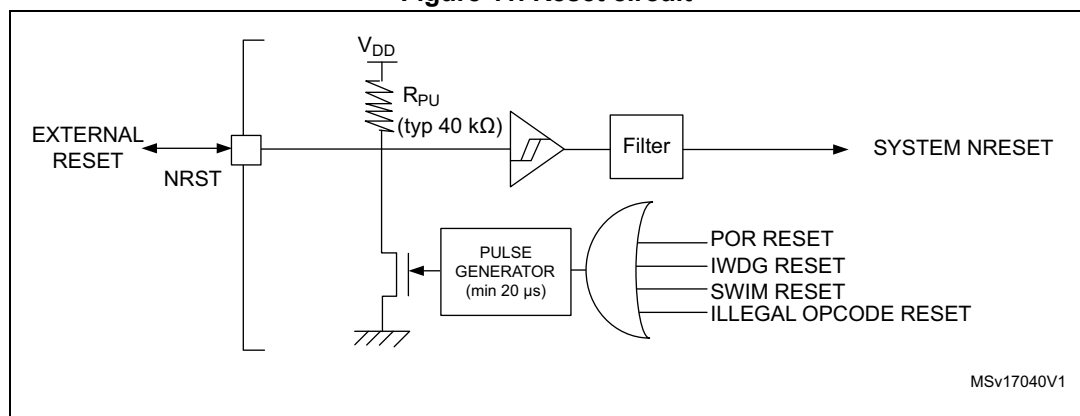


## 7.1 "Reset state" and "under reset" definitions

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 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.

## 7.2 External reset (NRST pin)

### 7.2.1 Asynchronous external reset description

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

A minimum of 300 ns low pulse on the NRST pin is needed to generate an external reset. The reset detection is asynchronous and therefore the MCU can enter reset even in 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 20 µs for every internal reset source. An additional internal weak pull-up ensures a high level on the reset pin when the reset is not forced.

Refer to *Figure 11* and see electrical parameters section in the datasheet for more details.

### 7.2.2 Configuring NRST/PA1 pin as general purpose output

To optimize the number of available pins, the NRST pin (external reset) can be configured as a general purpose push-pull output (PA1).

For security, this configuration can be performed once only after reset, by writing a specified key (D0h) to the *Reset pin configuration register (RST_CR)*.

When the PA1 pin is configured as a general purpose output the MCU can be reset only by a power-on reset (POR) or other internal reset source.

## 7.3 Internal reset

For internal reset sources, the $\overline{\text{RESET}}$ pin is kept low during the delay phase generated by the pulse generator.

Each internal reset source is linked to a specific flag bit in the *Reset status register (RST_SR)*. These flags are set respectively at reset state depending on the given reset source. So they are used to identify the last reset source. They are cleared by software writing the logic value "1".

*Note:* *All flags besides the POR flag are reset at POR.*

### 7.3.1 Power-on reset (POR)

During power-on, the POR keeps the device under reset until the supply voltage ($V_{DD}$) reaches a specified voltage and then holds reset active even longer for a specified time in order to assure that $V_{DD}$ has reached the minimum operating voltage before releasing the $\overline{\text{RESET}}$ pin. See electrical parameters section in the datasheet for more details.

### 7.3.2 Independent watchdog reset

See the independent watchdog section for details.

A reset can be triggered by the application software using the independent watchdog.

### 7.3.3 SWIM reset

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

### 7.3.4 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 value, a reset is generated. This, combined with the watchdog, allows recovery from an unexpected fault or interference.

## 7.4 RST registers

### 7.4.1 Reset pin configuration register (RST_CR)

Address offset: 0x00
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | RSTPIN_KEY | | | | |
| | | | rwo | | | | |

Bits 7:0  **RSTPIN_KEY[7:0]**: Reset pin configuration key
  0x00: NRST/PA1 configured as reset pin (default reset value)
  0xD0: NRST/PA1 configured as general purpose output
These bits are write once only. They can also be read at any time.
*Note: Writing any value beside D0h is equivalent to writing 00h.*

### 7.4.2 Reset status register (RST_SR)

Address offset: 0x01
Reset value after power-on reset: 0x01

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | Reserved | | | SWIMF | ILLOPF | IWDGF | PORF |
| | | | | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

Bits 7:4  Reserved

Bit 3  **SWIMF**: SWIM reset flag
  0: No SWIM reset occurred
  1: A SWIM reset occurred
This bit is set by hardware and cleared by software writing "1".

Bit 2  **ILLOPF**: Illegal opcode reset flag
  0: No ILLOP reset occurred
  1: An ILLOP reset occurred
This bit is set by hardware and cleared by software writing "1".

Bit 1  **IWDGF**: Independent Watchdog reset flag
  0: No IWDG reset occurred
  1: An IWDG reset occurred
This bit is set by hardware and cleared by software writing "1".

Bit 0  **PORF**: Power-on reset (POR) flag
  0: No POR occurred
  1: A POR occurred
This bit is set by hardware and cleared by software writing "1".

*Note:      RST_SR reset value depends on the previous reset source.*

## 7.5 RST register map and reset values

**Table 13. RST register map and reset values**

| Address offset[1] | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | RST_CR<br><br>Reset value | RSTPIN _KEY7<br>0 | RSTPIN _KEY6<br>0 | RSTPIN _KEY5<br>0 | RSTPIN _KEY4<br>0 | RSTPIN _KEY3<br>0 | RSTPIN _KEY2<br>0 | RSTPIN _KEY1<br>0 | RSTPIN _KEY0<br>0 |
| 0x01 | RST_SR<br>Reset value | -<br>0 | -<br>0 | -<br>0 | WWDGF<br>0 | SWIMF<br>0 | ILLOPF<br>0 | IWDGF<br>0 | PORF<br>1 |

1. Please refer to the "general hardware register map" table in the datasheet for details on register addresses.

# 8     Clock control (CLK)

The clock controller is designed to be very robust and at the same time easy to use. Its purpose is to obtain the best performance in the application while at the same time get the full benefit of all the microcontroller power saving capabilities.

You can manage the clock distribution to the CPU and to the various peripherals, in order to optimize the power consumption.

A safe and glitch-free switch mechanism allows you to switch the master clock division factor on the fly, by means of clock prescaler.

**Figure 12. Clock structure**
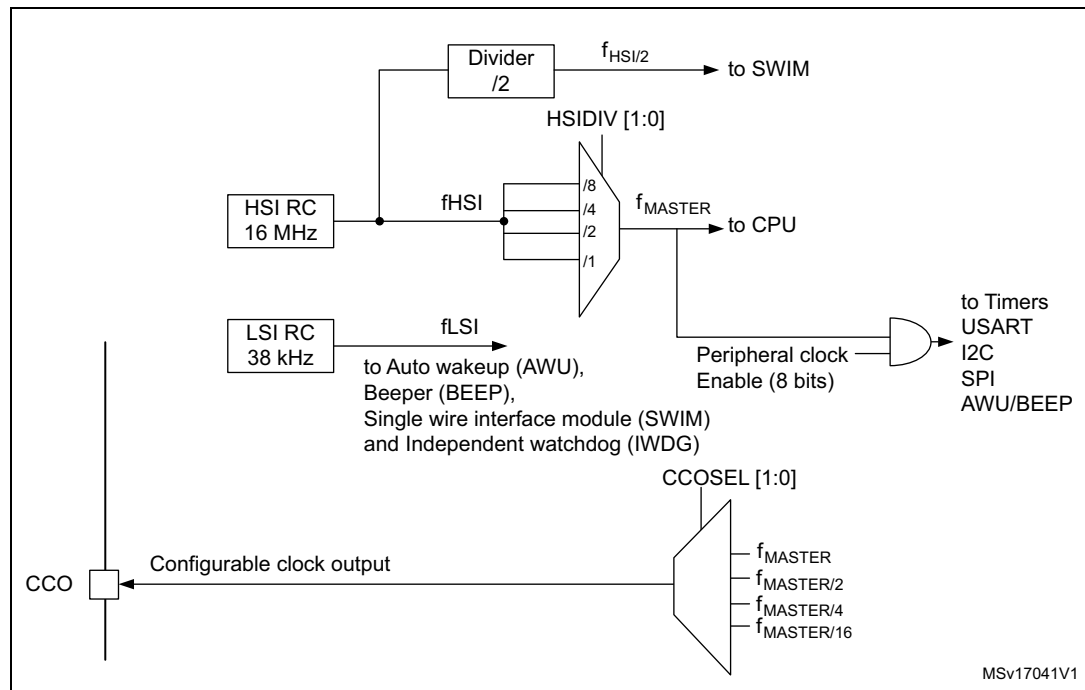


1. HSI = high speed internal clock signal
   LSI = low speed internal clock signal.

## 8.1 Master clock (HSI clock)

The master clock of the system is a 16-MHz high-speed internal RC oscillator (HSI RC), followed by a programmable prescaler.

The $f_{HSI}$ clock signal generated from the internal 16-MHz RC oscillator can then be divided by a programmable divider (factor 1 to 8) to obtain $f_{MASTER}$. This is programmed in the *Clock divider register (CLK_CKDIVR)*.

*Note:* *At startup the master clock source is automatically selected as HSI RC clock output divided by 8 (HSI/8).*

The HSI RC oscillator has the advantage of providing a 16-MHz master clock source with 50% duty cycle at low cost (no external components) and with a fast startup time.

The clock controller configures the master clock source as HSI RC clock output divided by 8 ($f_{HSI}/8$), which helps ensure a safe startup in case of poor $V_{DD}$ conditions.

### 8.1.1 Peripheral clock gating (PCG)

Gating the clock to unused peripherals helps reduce power consumption. Peripheral clock gating (PCG) mode selectively enables or disables the $f_{MASTER}$ clock connection to the following peripherals at any time in run or slow mode:

- TIM2
- TIM3
- TIM4
- I2C
- SPI
- USART
- AWU/BEEP (except LSI clock controlled by AWUEN bit in the *Control/status register (AWU_CSR)* and by BEEPEN bit in the *Beeper control/status register (BEEP_CSR)*).

After a device reset, all peripheral clocks are disabled. You can enable the clock to any peripheral by setting the corresponding PCKEN bit in the *Peripheral clock gating register (CLK_PCKENR)*. To enable a peripheral, first enable the corresponding PCKEN bit in the CLK_PCKENR registers and then set the peripheral enable bit in the peripheral control registers.

To disable properly the peripheral, first disable the appropriate bit in the peripheral control registers and then stop the corresponding clock.

The AWU counter is fed by the low speed internal specific clock ($f_{LSI}$) different from $f_{MASTER}$, so that it continues to run even if the clock gating to this peripheral register is asserted. The same is true for the beeper.

## 8.2 LSI clock

The LSI RC oscillator generates the $f_{LSI}$ clock. This clock is a 38-kHz Low Speed Internal RC oscillator (LSI RC). It is used by the independent watchdog (IWDG), Auto-Wakeup unit (AWU) and beeper (BEEP). The LSI is active in Active-halt mode and can be kept in run mode.

## 8.3 Configurable clock-output capability (CCO)

The configurable clock output (CCO) capability outputs a clock on the external CLK_CCO pin. The master clock is used to drive the clock output with a programmable prescaler which can divide the clock frequency by a factor of 1, 2, 4 or 16.

The selection is controlled by the CCOSEL[1:0] bits in the *Configurable clock output register (CLK_CCOR)*.

The sequence to output the targeted clock starts with CCOEN=1 in *Configurable clock output register (CLK_CCOR)*. When CCOEN=1, the CCOSEL[1:0] bits are write protected.

To disable the clock output the user has to clear the CCOEN bit.

## 8.4 CLK registers

### 8.4.1 Clock divider register (CLK_CKDIVR)

Address offset: 0x00
Reset value: 0x03

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | Reserved | | | HSIDIV[1:0] | |
| | | | | | | rw | |

Bits 7:2  Reserved

Bits 1:0  **HSIDIV[1:0]**: High speed internal clock prescaler

    00: $f_{MASTER}= f_{HSI\ RC\ output}$
    01: $f_{MASTER}= f_{HSI\ RC\ output}/2$
    10: $f_{MASTER}= f_{HSI\ RC\ output}/4$
    11: $f_{MASTER}= f_{HSI\ RC\ output}/8$

These bits are written by software to define the HSIDIV prescaling factor.

### 8.4.2 Peripheral clock gating register (CLK_PCKENR)

Address offset: 0x03

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| PCKEN[7:0] | | | | | | | |
| rw | | | | | | | |

Bits 7:0 **PCKEN[7:0]**: Peripheral clock enable

    0: $f_{MASTER}$ to peripheral disabled

    1: $f_{MASTER}$ to peripheral enabled

These bits are written by software to enable or disable the $f_{MASTER}$ clock to the corresponding peripheral. See *Table 14*

**Table 14. Peripheral clock gating bits**

| Control bit | Peripheral |
|---|---|
| PCKEN7 | Reserved |
| PCKEN6 | AWU+BEEP (except LSI clock controlled by AWUEN bit in the *Control/status register (AWU_CSR)* and BEEPEN bit in the *Beeper control/status register (BEEP_CSR)*) |
| PCKEN5 | USART |
| PCKEN4 | SPI |
| PCKEN3 | I²C |
| PCKEN2 | TIM4 |
| PCKEN1 | TIM3 |
| PCKEN0 | TIM2 |

### 8.4.3 Configurable clock output register (CLK_CCOR)

Address offset: 0x05
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | Reserved | | | CCOSEL[1:0] | | CCOEN |
| | | | | | rw | | rw |

Bit 7:3 Reserved.

Bits 2:1 **CCOSEL[1:0]**: Configurable clock output selection.

These bits are written by software to select the source of the output clock available on the CLK_CCO pin.

    00: $f_{MASTER}$
    01: $f_{MASTER}/2$
    10: $f_{MASTER}/4$
    11: $f_{MASTER}/16$

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

This bit is set and cleared by software.

    0: CCO clock output disabled
    1: CCO clock output enabled

## 8.4.4     CLK register map and reset values

**Table 15. CLK register map and reset values**

| Address offset (1) | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CLK_CKDIVR Reset value | -<br>0 | -<br>0 | -<br>0 | -<br>0 | -<br>0 | -<br>0 | HSIDIV1<br>1 | HSIDIV0<br>1 |
| 0x01 to 0x02 | Reserved area (2 bytes) | | | | | | | | |
| 0x03 | CLK_PCKENR Reset value | PCKEN7<br>0 | PCKEN6<br>0 | PCKEN5<br>0 | PCKEN4<br>0 | PCKEN3<br>0 | PCKEN2<br>0 | PCKEN1<br>0 | PCKEN0<br>0 |
| 0x04 | Reserved area (1 byte) | | | | | | | | |
| 0x05 | CLK_CCOR Reset value | -<br>0 | -<br>0 | -<br>0 | -<br>0 | -<br>0 | CCOSEL1<br>0 | CCOSEL0<br>0 | CCOEN<br>0 |

1.   Please refer to the "general hardware register map" table in the datasheet for details on register addresses.

# 9 Power management

By default, after a system or power reset, the microcontroller is in Run mode. In this mode the CPU is clocked by $f_{MASTER}$ and executes the program code, while the system clock to all peripherals is gated off by the peripheral clock gating system.

While in Run mode, still keeping the CPU running and executing code, the application has several ways to reduce power consumption, such as:

- Slowing down the system clock
- Gating the clocks to individual peripherals when they are unused
- Switching off any unused analog functions

However, when the CPU does not need to be kept running, three dedicated low power modes can be used:

- Wait (wait for interrupt and wait for event)
- Active-halt
- Halt

One of these three modes can be selected and configured to obtain the best compromise between lowest power consumption, fastest startup time and available wakeup sources.

## 9.1 General considerations

Low power consumption features are generally very important for all types of application for energy saving. Ultra low power features are especially important for mobile applications to ensure long battery lifetimes. This is also crucial for environmental protection.

In a silicon chip there are two kind of consumption:

- **Static power consumption** which is due to analog polarization and leakage. Consumption due to leakage is so small that it is only significant in Halt and Active-halt modes (*Section 9.4: Low power modes*).
- **Dynamic power consumption** which comes from running the digital parts of the chip. It depends on $V_{DD}$, clock frequency and load capacitors.

In a microcontroller device the consumption depends on:

- $V_{DD}$ supply voltage
- MCU size or number of digital gates (leakages and load capacitors)
- Clock frequency
- Number of active peripherals
- Available low power modes and low power levels

Device processing performance is also very important, as this allows the application to minimize the time spent in Run mode and maximize the time in Low power mode.

Using the MCU's flexible power management features, you can obtain a range of significant power savings while the system is running or able to resume operations quickly.

## 9.2 Managing the clock for low consumption

### 9.2.1 Slowing the system clocks

In run mode, choosing the clock frequency is very important to ensure the best compromise between performance and consumption. The selection is done by programming the CLK_CKDIVR register. Refer to *8.4.1: Clock divider register (CLK_CKDIVR) on page 63*.

*Note:* *In applications where the MCU can be halted for certain periods, the power consumption can be minimized by keeping a fast clock (high performance execution) during active periods, in order to reduce the ratio between active periods and Halt (that is "zero"-consumption) periods.*

### 9.2.2 Peripheral clock gating

For additional power saving you can use Peripheral Clock Gating (PCG). This can be done at any time by selectively enabling or disabling the $f_{MASTER}$ clock connection to individual peripherals. Refer to *8.4.2: Peripheral clock gating register (CLK_PCKENR) on page 63*.

These settings are effective in both Run and Wait modes. Refer to *8.1.1: Peripheral clock gating (PCG) on page 62*.

Each PCG state represents a specific power or low power level.

## 9.3 Switching peripherals off

Any of the MCU peripherals can be deactivated when they are not used in order to minimize power consumption. This is true for both analog and digital peripherals.

Each ON/OFF combination represents a specific power or low power level.

### 9.3.1 Analog peripherals

The analog comparators can be switched off by dedicated control bits: BIAS_EN and COMPx_EN in the COMP_CR register.

### 9.3.2 Digital peripherals

Each of the digital peripherals can be switched off by a dedicated control bit:

- Single Wire Interface Module (SWIM) by the SWD bit in the SWIM_CSR register
- Auto-Wakeup Unit (AWU) by the AWUEN bit in the AWU_CSR register
- Beeper by the BEEPEN bit in the AWU_CSR register
- Timers by the CEN bits in the TIMx_CR1 registers
- I2C by the PE bit in the I2C_CR1 register
- SPI by the SPE bit in the SPI_CR1
- USART by the USART_CR2 register

*Note:* *These control bits should be written only when the corresponding peripheral clock is enabled*
*Once the IWDG is activated, it cannot be disabled except with a reset.*

## 9.4 Low power modes

By default, the microcontroller is in run mode after a system or power reset. However the device supports three low power modes to achieve the best compromise between low power consumption, short startup time and available wakeup sources:

– **Wait mode**: The CPU clock is stopped, but selected peripherals keep running. An internal or external interrupt or a Reset can be used to exit the microcontroller from Wait mode. Refer to *Section 9.4.1: Wait mode on page 68*

– **Active-halt mode**: The CPU and peripheral clocks are stopped, except RTC. The wakeup can be triggered by RTCinterrupts, external interrupts or reset.

– **Halt mode**: The CPU and peripheral clocks are stopped, the device remains powered on. The wakeup is triggered by an external interrupt or reset. A few peripherals also have wakeup from Halt capability.

The main characteristics of the three low power modes are summarized in *Table 16*.

**Table 16. Low power mode management**

| Mode (consumption level) | | Main voltage regulator (MVR) | Oscillators | CPU | Peripherals | Wakeup trigger event |
|---|---|---|---|---|---|---|
| Wait ( - ) | WFI | ON | ON | OFF | ON[1] | All internal or external interrupts, reset |
| | WFE | ON | ON | OFF | ON[1] | All internal or external interrupts[2], WFE events, reset |
| Active-halt ( - - ) | - | OFF (LPVR ON) | HSI OFF, LSI ON | OFF | Only AWU, BEEP and IWDG if activated and if "no watchdog in Halt" option disabled | AWU or external[3] interrupts, reset |
| Halt ( - - - ) | - | OFF (LPVR ON) | LSI ON if IWDG activated and if "no watchdog in Halt" option is disabled. | OFF | Only BEEP and IWDG if activated and if "no watchdog in Halt" option disabled | External interrupts[3], reset |

1. If the peripheral clock is not disabled by Peripheral Clock Gating function.

2. Refer to*: Wait for event (WFE) mode on page 69*

3. Including communication peripheral interrupts (see interrupt vector table).

In addition, the power consumption in Run mode can be reduced by one of the following means:

– Slowing down the system clocks

– Gating the peripherals clocks when they are unused

Refer to *Section 9.2: Managing the clock for low consumption on page 67*.

### 9.4.1 Wait mode

Wait mode is entered from run mode by executing a WFI (Wait For Interrupt) or WFE (Wait For Event) instruction: this stops the CPU but allows the other peripherals and interrupt controller to continue running. The consumption decreases accordingly. Wait mode can be

combined with PCG to further reduce power consumption of the device. Refer to *Section 8.1.1: Peripheral clock gating (PCG) on page 62*.

In Wait mode, all the registers and RAM contents are preserved and the clock configuration selected through the *Clock divider register (CLK_CKDIVR)* remains unchanged.

### Wait for interrupt (WFI) mode

When an internal or external interrupt request occurs, the CPU wakes up from WFI mode, services the interrupt and resumes processing.

*Note:* *In an interrupt based application, where most of the processing is done through the interrupt routines, the main program may be suspended by setting the activation level bit (AL) in the CFG_GCR register. Setting this bit causes the CPU to return to WFI mode without restoring the main execution context. This saves power by removing both the save/restore context activity and the need for a main software loop execution for power management (in order to return to WFI mode).*

### Wait for event (WFE) mode

There are two ways to wake up the CPU from WFE mode (see *Table 16.: Low power mode management on page 68*):

- When an interrupt occurs:
  - the CPU wakes up from WFE mode and services the interrupt. After processing the interrupt, the processor goes back to WFE mode.
- When a WFE event occurs:
  - the CPU wakes up and resumes processing. As the processing resumes directly after the WFE instruction, there is no context save/restore activity (this saves time and power consumption).

*Note:* *In WFE mode, the interrupt sources are configured as external interrupts only if the corresponding status flags are cleared in the WFE_CR1 or WFE_CR2 register. Otherwise, they generate WFE events (no interrupt serviced and the user has to properly clear the corresponding status flags in the EXTI_SR1 or EXTI_SR2 register). Refer to Section 9.5: WFE registers on page 70, External interrupt status register 1 (EXTI_SR1) on page 52 and External interrupt status register 2 (EXTI_SR2) on page 53.*

Further power consumption reduction may be achieved using this mode together with execution from RAM. In some very low power applications, when the main software routine is short and has a low execution time, this routine can be moved to RAM and executed from RAM. As the Flash program memory is not used at wakeup, the power consumption is then reduced during run time.

At any moment, another routine (stored in the Flash program memory) can be executed by software by simply calling/jumping to this routine.

### 9.4.2 Halt mode

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

The Main Voltage Regulator (MVR) is switched off automatically when the MCU enters Halt mode. The MCU core is powered only by the Low Power Voltage Regulator (LPVR) and all the registers and RAM contents are preserved.

The MCU enters Halt mode when a HALT instruction is executed. Wakeup from Halt mode is triggered by an external interrupt, sourced by a general purpose I/O port configured as interrupt input or by an alternate function pin capable of triggering a peripheral interrupt.

In an interrupt based application, where most of the processing is done through the interrupt routines, the main program may be suspended by setting the activation level bit (AL) in the CPU configuration register. Setting this bit causes the CPU to return to Halt mode when executing the return from interrupt, without restoring the main execution context.

This saves power by removing the save/restore context activity and by removing the need to execute a main level software loop for power management (in order to return to WFI mode).

**Caution:** Before entering Halt mode, the ProxSense peripheral must be disabled or in Low power mode (PXS_EN = '0' or LOW_POWER = '1' in register PXS_CR1).

### 9.4.3 Active-halt mode

Active-halt mode is similar to Halt mode except that it does not require an external interrupt for wake up. It uses the AWU to generate a wakeup event internally after a programmable delay.

To enter Active-halt mode, first enable the AWU as described in the AWU section. Then execute a HALT instruction.

In Active-halt mode, the main oscillator, the CPU and almost all peripherals are stopped. Only the LSI RC oscillator is running to drive the AWU counters, BEEP and IWDG counter if enabled.

The Main Voltage Regulator (MVR) is switched off automatically when the MCU enters Active-halt mode. The MCU core is powered only by the Low Power Voltage Regulator (LPVR) and all registers and RAM contents are preserved.

## 9.5 WFE registers

These registers are used to configure different interrupt sources as external interrupts or WFE events. Refer to *Section : Wait for event (WFE) mode on page 69*

### 9.5.1 WFE control register 1 (WFE_CR1)

Address Offset: 0x00

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EXTI_EV3 | EXTI_EV2 | EXTI_EV1 | EXTI_EV0 | Reserved | | TIM2_EV1 | TIM2_EV0 |
| rw | rw | rw | rw | rw | | rw | rw |

Bit 7 **EXTI_EV3**: External interrupt event 3

0: Interrupt sources from pin 3 of all ports generate external interrupts.

1: Interrupt sources from pin 3 of all ports generate WFE events (no interrupt serviced).

This bit is written by software.

Bit 6  **EXTI_EV2**: External interrupt event 2

    0: Interrupt sources from pin 2 of all ports generate external interrupts.

    1: Interrupt sources from pin 2 of all ports generate WFE events (no interrupt serviced).

    This bit is written by software.

Bit 5  **EXTI_EV1**: External interrupt event 1

    0: Interrupt sources from pin 1 of all ports generate external interrupts.

    1: Interrupt sources from pin 1 of all ports generate WFE events (no interrupt serviced).

    This bit is written by software.

Bit 4  **EXTI_EV0**: External interrupt event 0

    0: Interrupt sources from pin 0 of all ports generate external interrupts.

    1: Interrupt sources from pin 0 of all ports generate WFE events (no interrupt serviced).

    This bit is written by software.

Bits 3:2  Reserved.

Bit 1  **TIM2_EV1**: TIM2 event 1

    0: TIM2 capture and compare interrupt sources generate external interrupts.

    1: TIM2 capture and compare interrupt sources logically ORed and configured to generate WFE events (no interrupt serviced).

    This bit is written by software.

Bit 0  **TIM2_EV0**: TIM2 event 0

    0: TIM2 update, trigger and break interrupt sources generate external interrupts.

    1: TIM2 update, trigger and break interrupt sources logically ORed and configured to generate WFE events (no interrupt serviced).

    This bit is written by software.

## 9.5.2   WFE control register 2 (WFE_CR2)

Address offset: 0x01

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | EXTI_EVD | EXTI_EVB | EXTI_EV7 | EXTI_EV6 | EXTI_EV5 | EXTI_EV4 |
| | | rw | rw | rw | rw | rw | rw |

Bits 7:6  Reserved

Bit 5  **EXTI_EVD**: External interrupt event on Port D

    0: Interrupt sources from Port D generate external interrupts.

    1: Interrupt sources from Port D generate WFE events (no interrupt serviced).

    This bit is written by software.

Bit 4  **EXTI_EVB**: External interrupt event on Port B

    0: Interrupt sources from Port B generate external interrupts.

    1: Interrupt sources from Port B generate WFE events (no interrupt serviced).

    This bit is written by software

Bit 3  **EXTI_EV7**: External interrupt event 7

    0: Interrupt sources from pin 7 of all ports generate external interrupts.

    1: Interrupt sources from pin 7 of all ports generate WFE events (no interrupt serviced).

    This bit is written by software.

Bit 2 **EXTI_EV6**: External interrupt event 6

> 0: Interrupt sources from pin 6 of all ports generate external interrupts.
> 1: Interrupt sources from pin 6 of all ports generate WFE events (no interrupt serviced).

This bit is written by software.

Bit 1 **EXTI_EV5**: External interrupt event 5

> 0: Interrupt sources from pin 5 of all ports generate external interrupts.
> 1: Interrupt sources from pin 5 of all ports generate WFE events (no interrupt serviced).

This bit is written by software.

Bit 0 **EXTI_EV4**: External interrupt event 4

> 0: Interrupt sources from pin 4 of all ports generate external interrupts.
> 1: Interrupt sources from pin 4 of all ports generate WFE events (no interrupt serviced).

This bit is written by software.

# 9.6 WFE register map and reset values

**Table 17. WFE register map**

| Address offset (1) | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | WFE_CR1 Reset value | EXTI_EV3 0 | EXTI_EV2 0 | EXTI_EV1 0 | EXTI_EV0 0 | PXS_EV 0 | - 0 | TIM2_EV1 1 | TIM2_EV0 1 |
| 0x01 | WFE_CR2 Reset value | - 00 | | EXTI_EVD 0 | EXTI_EVB 0 | EXTI_EV7 0 | EXTI_EV6 0 | EXTI_EV5 0 | EXTI_EV4 0 |

1. Please refer to the "general hardware register map" table in the datasheet for details on register addresses.

# 10 General purpose I/O ports (GPIO)

## 10.1 Introduction

General purpose input/output ports are used for data transfers between the chip and the external world. An I/O port can contain up to eight pins. Each pin can be individually programmed as a digital input or digital output. In addition, some ports may have alternate functions like analog inputs, external interrupts, input/output for on-chip peripherals. Only one alternate function can be mapped to a pin at a time.

An output data register, input data register, data direction register and two configuration registers are associated with each port. A particular port will behave as an input or output depending on the status of the data direction register of the port.

## 10.2 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.
- Separate registers for data input and output
- External interrupts can be enabled and disabled individually
- Output slope control for reduced EMC noise
- Alternate function I/Os for on-chip peripherals
- Read-modify-write possible on data output latch
- I/O state guaranteed in voltage range 1.6 V to $V_{DDIOmax}$

**Figure 13. GPIO block diagram**
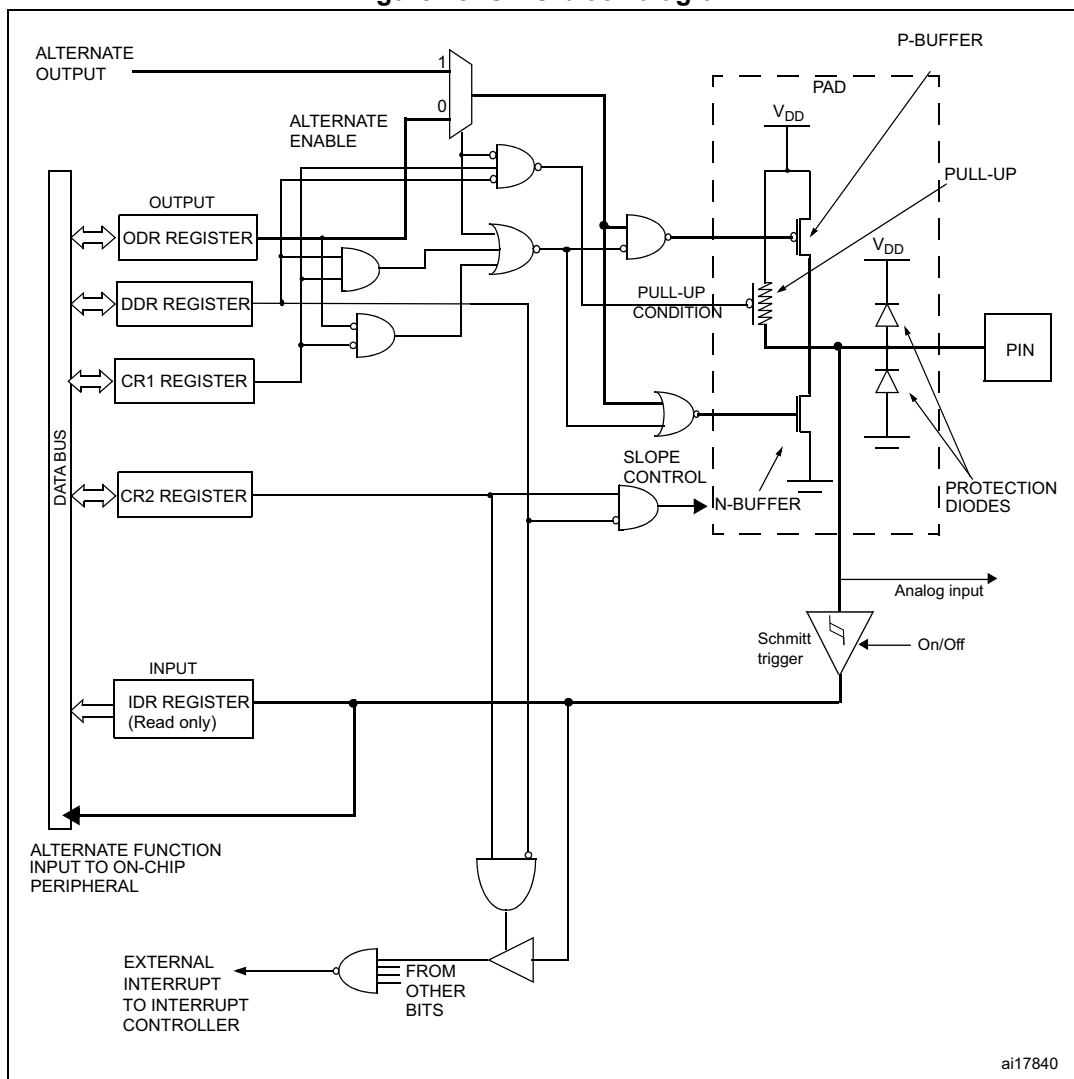


## 10.3 Port configuration and usage

An output data register (ODR), pin input register (IDR), data direction register (DDR) are always associated with each port.

The control register 1 (CR1) and control register 2 (CR2) allow input/output options. An I/O pin is programmed using the corresponding bits in the DDR, ODR, CR1 and CR2 registers.

Bit *n* in the registers corresponds to pin *n* of the Port.

The various configurations are summarized in *Table 18*.

**Table 18. I/O port configuration summary**

| Mode | DDR bit | CR1 bit | CR2 bit | Function | Pull-up | P-buffer | Diodes | |
|------|---------|---------|---------|----------|---------|----------|--------|--------|
| | | | | | | | to $V_{DD}$ | to $V_{SS}$ |
| Input | 0 | 0 | 0 | Floating without interrupt | Off | Off | On | On |
| | 0 | 1 | 0 | Pull-up without interrupt | On | | | |
| | 0 | 0 | 1 | Floating with interrupt | Off | | | |
| | 0 | 1 | 1 | Pull-up with interrupt | On | | | |
| Output | 1 | 0 | 0 | Open drain output | Off | Off | | |
| | 1 | 1 | 0 | Push-pull output | | On | | |
| | 1 | 0 | 1 | Open drain output, fast mode | | Off | | |
| | 1 | 1 | 1 | Push-pull, fast mode | Off | On | | |
| | 1 | x | x | True open drain (on specific pins) | Not implemented | | Not implemented (1) | |

1. The diode connected to $V_{DD}$ is not implemented in true open drain pads. A local protection between the pad and $V_{OL}$ is implemented to protect the device against positive stress.

> **Warning:** On some packages, some ports must be considered as active even if they do not exist on the package. To avoid spurious effects, configure them as pull-up inputs without interrupt at startup, and keep them in this state when changing the port configuration. Refer to the datasheet for additional information.

### 10.3.1 Input modes

Clearing the DDRx bit selects input mode. In this mode, reading a IDR bit returns the digital value of the corresponding I/O pin.

Refer to *Section 10.7: Input mode details on page 77* for information on analog input, external interrupts and Schmitt trigger enable/disable.

As shown in , four different input modes can be theoretically be configured by software: floating without interrupt, floating with interrupt, pull-up without interrupt or pull-up with interrupt. However in practice, not all ports have external interrupt capability or pull-ups. You should refer to the datasheet pin-out description for details on the actual hardware capability of each port.

### 10.3.2 Output modes

Setting the DDRx bit selects output mode. In this mode, writing to the ODR bits applies a digital value to the I/O through the latch. Reading 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, Open-drain output.

Refer to *Section 10.8: Output mode details on page 78* for more information.

## 10.4 Reset configuration

All I/O pins are generally input floating under reset (i.e. during the reset phase) and at reset state (i.e. after reset release). However, a few pins may have a different behavior. Refer to the datasheet pinout description for all details.

## 10.5 Unused I/O pins

Unused I/O pins must not be left floating to avoid extra current consumption. They must be put into one of the following configurations:

- connected to $V_{DD}$ or $V_{SS}$ by external pull-up or pull-down resistor and kept as input floating (reset state),
- configured as input with internal pull-up/down resistor,
- configured as output push-pull low.

The I/O ports not present on smaller packages are automatically configured by a factory setting (unless otherwise specified in the datasheet). As a consequence, no configuration is required on these I/O ports. The bits corresponding to these ports in the configuration registers Px_ODR, PxDDR, PxCR1 and PxCR2 can be written, but this will have no effect. The value read in the corresponding bits of the PxIDR register will be '0'.

## 10.6 Low power modes

**Table 19. Effect of low power modes on GPIO ports**

| Mode | Description |
|------|-------------|
| Wait | No effect on I/O ports. External interrupts cause the device to exit from Wait mode. |
| Halt | No effect on I/O ports. External interrupts cause the device to wakeup from Halt mode. |

## 10.7 Input mode details

### 10.7.1 Alternate function input

Some I/Os can be used as alternate function input. For example as the port may be used as the input capture input to a timer. Alternate function inputs are not selected automatically, you select them by writing to a control bit in the registers of the corresponding peripheral. For Alternate Function input, you should select floating or pull-up input configuration in the DDR and CR1 registers.

### 10.7.2 Interrupt capability

Each I/O can be configured as an input with interrupt capability by setting the CR2x bit while the I/O is in input mode. In this configuration, a signal edge or level input on the I/O generates an interrupt request.

Falling or rising edge sensitivity is programmed independently for each interrupt vector in the EXTI_CR[2:1] registers.

External interrupt capability is only available if the port is configured in input mode.

#### Interrupt masking

Interrupts can be enabled/disabled individually by programming the corresponding bit in the configuration register (Px_CR2). At reset state, the interrupts are disabled.

If a pin alternate function is TLI, use the Px_CR2 bit to enable/disable the TLI interrupt. The TLI interrupt is associated to a dedicated interrupt vector.

On all I/Os with an analog input, it is possible to disable the Schmitt trigger, even if the corresponding ADC channel is not enabled. The two registers ADC_TDRH and ADC_TDRL allow to disable the Schmitt trigger.

Setting one bit in these registers leads to disabling the corresponding Schmitt trigger input buffer.

In case an I/O is used as analog input, and the corresponding ADC channel is enabled (CH[3:0] bits in ADC_CSR register), the Schmitt trigger is disabled, whatever the status of the corresponding bit in ADC_TDRH or ADC_TDRL registers.

### 10.7.3 Analog function

Selected I/Os can be used to deliver analog signal to ADC, Comparators or DAC periphery. The GPIO pin have to be configured in the input floating configuration without interrupt (default state) to use it for analog function. The current consumption of the IO with enabled analog function can be reduced by disabling unused Schmitt trigger in IO input section either by ADC_TRIGRx register in ADC interface (see *Section 14.3.15: Schmitt trigger disabling*) or by switching on a corresponding analog switch in RI by setting corresponding CHxE bit in RI_IOSRx (see *Section 11.2.2: I/O groups*). See the product datasheet for pins with analog functions.

## 10.8 Output mode details

### 10.8.1 Alternate function output

Alternate function outputs provide a direct path from a peripheral to an output or to an I/O pad, taking precedence over the port bit in the data output latch register (Px_ODR) and forcing the Px_DDR corresponding bit to 1.

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.

*Examples:*

SPI outputs must be set-up as push-pull. The slope of SPI outputs is controlled by hardware and configured in fast mode to enable an optimal operation. The user must then keep the CR2 slope control bit cleared to avoid spurious interrupts.

### 10.8.2 Slope control

The maximum frequency that can be applied to an I/O can be controlled by software using the CR2 bit. Low frequency operation with improved EMC behavior is selected at reset. Higher frequency (up to 10 MHz) can be selected if needed. This feature can be applied in either open drain or push-pull output mode on I/O ports of output type O3 or O4. Refer to the pin description tables in the datasheets for the specific output type information for each pin.

## 10.9      GPIO registers

The bit of each port register drives the corresponding pin of the port.

### 10.9.1    Port x output data register (Px_ODR)

Address offset: 0x00
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ODR7 | ODR6 | ODR5 | ODR4 | ODR3 | ODR2 | ODR1 | ODR0 |
| rw | rw | rw | rw | rw | rw | rw | rw |

Bits 7:0  **ODR[7:0]**: Output data register bits

Writing to the ODR register when in output mode applies a digital value to the I/O through the latch. Reading the ODR returns the previously latched value in the register.

In Input mode, writing in the ODR register, latches the value in the register but does not change the pin state. The ODR register is always cleared after reset. Bit read-modify-write instructions (BSET, BRST) can be used on the DR register to drive an individual pin without affecting the others.

### 10.9.2    Port x pin input register (Px_IDR)

Address offset: 0x01
Reset value: 0xXX

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IDR7 | IDR6 | IDR5 | IDR4 | IDR3 | IDR2 | IDR1 | IDR0 |
| r | r | r | r | r | r | r | r |

Bits 7:0  **IDR[7:0]**: Pin input values

The pin register can be used to read the pin value irrespective of whether port is in input or output mode. This register is read-only.
0: Low logic level
1: High logic level

*Note:*         *Px_IDR reset value depends on the external circuitry.*

### 10.9.3 Port x data direction register (Px_DDR)

Address offset: 0x02
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DDR7 | DDR6 | DDR5 | DDR4 | DDR3 | DDR2 | DDR1 | DDR0 |
| rw | rw | rw | rw | rw | rw | rw | rw |

Bits 7:0 **DDR[7:0]**: Data direction bits
These bits are set and cleared by software to select input or output mode for a particular pin of a port.
0: Input mode
1: Output mode

### 10.9.4 Port x control register 1 (Px_CR1)

Address offset: 0x03
Reset value: 0x00 except for PA_CR1 which reset value is 0x01.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| C17 | C16 | C15 | C14 | C13 | C12 | C11 | C10 |
| rw | rw | rw | rw | rw | rw | rw | rw |

Bits 7:0 **C1[7:0]**: Control bits
These bits are set and cleared by software. They select different functions in input mode and output mode (*see .*
– **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 CR2 bit
*Note: This bit has no effect on true open drain ports (refer to pin marked "T" in datasheet pin description table).*

### 10.9.5 Port x control register 2 (Px_CR2)

Address offset: 0x04
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| C27 | C26 | C25 | C24 | C23 | C22 | C21 | C20 |
| rw | rw | rw | rw | rw | rw | rw | rw |

Bits 7:0 **C2[7:0]**: Control bits

These bits are set and cleared by software. They select different functions in input mode and output mode. In input mode, the CR2 bit enables the interrupt capability if available. If the I/O does not have interrupt capability, setting the CR2 bit has no effect. In output mode, setting the bit increases the speed of the I/O. This applies to ports with O3 and O4 output types (see pin description table).

– **In input mode (DDR = 0):**
*0: External* interrupt disabled
*1: External* interrupt enabled
– **In output mode (DDR = 1):**
0: Output speed up to 2 MHz
1: Output speed up to 10 MHz

### 10.9.6 GPIO register map and reset values

Each GPIO port has five registers mapped as shown in *Table 20*. Refer to the register map in the corresponding datasheet for the base address for each port.

*Note:*      *At reset state, all ports are input floating. Exceptions are indicated in the pin description table of the corresponding datasheet.*

**Table 20. GPIO register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | Px_ODR<br>Reset value | ODR7<br>0 | ODR6<br>0 | ODR5<br>0 | ODR4<br>0 | ODR3<br>0 | ODR2<br>0 | ODR1<br>0 | ODR0<br>0 |
| 0x01 | Px_IDR<br>Reset value | IDR7<br>x | IDR6<br>x | IDR5<br>x | IDR4<br>x | IDR3<br>x | IDR2<br>x | IDR1<br>x | IDR0<br>x |
| 0x02 | Px_DDR<br>Reset value | DDR7<br>0 | DDR6<br>0 | DDR5<br>0 | DDR4<br>0 | DDR3<br>0 | DDR2<br>0 | DDR1<br>0 | DDR0<br>0 |
| 0x03 | Px_CR1[1]<br>Reset value | C17<br>0 | C16<br>0 | C15<br>0 | C14<br>0 | C13<br>0 | C12<br>0 | C11<br>0 | C10<br>0 |
| 0x04 | Px_CR2<br>Reset value | C27<br>0 | C26<br>0 | C25<br>0 | C24<br>0 | C23<br>0 | C22<br>0 | C21<br>0 | C20<br>0 |

1. PA_CR1 reset value is 0x01.

# 11 Auto-wakeup (AWU)

## 11.1 AWU introduction

The AWU is used to provide an internal wakeup time base that is used when the MCU goes into Active-halt power saving mode. This time base is clocked by the low speed internal (LSI) RC oscillator clock.

## 11.2 LSI clock measurement

To ensure the best possible accuracy when using the LSI clock, its frequency can be measured with TIM2 timer input capture 1.

**Figure 14. AWU block diagram**
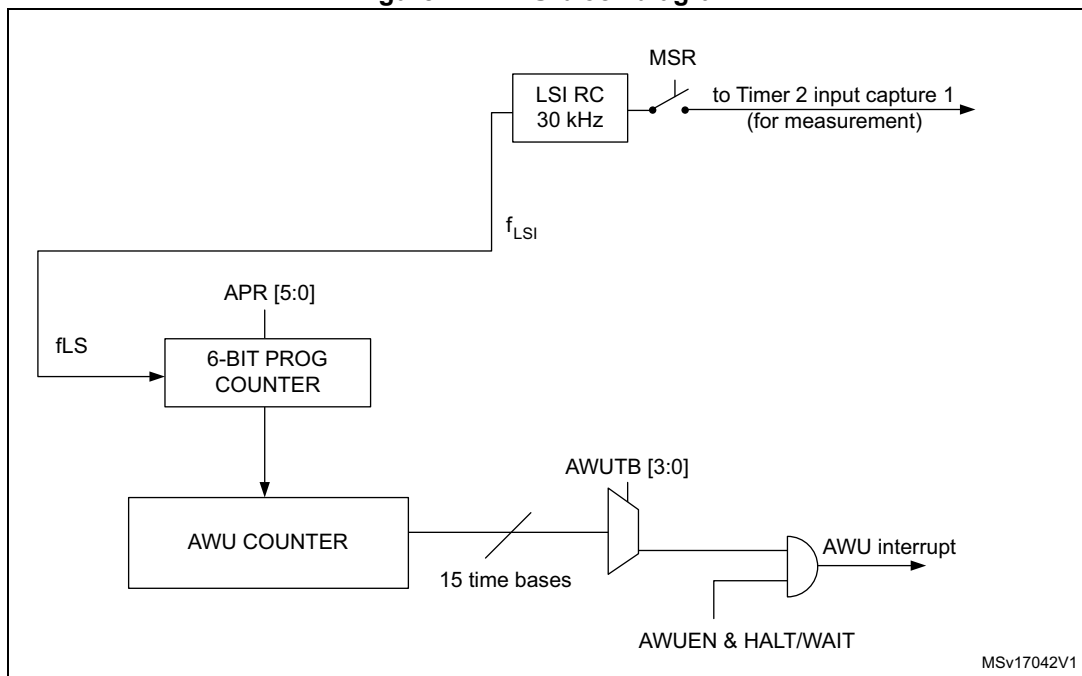
## 11.3 AWU functional description

### 11.3.1 AWU operation

To use the AWU, perform the following steps in order:

1. Measure the LSI clock frequency using the MSR bit in AWU_CSR register and TIM2 input capture 1.

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 *Timebase 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 time interval.

*Note:* *The counters only start when the MCU enters Active-halt mode after a HALT instruction (refer to the Active-halt mode section in the power management chapter). The AWU interrupt is then enabled at the same time.*

*The prescaler counter starts to count only if APR[5:0] value is different from its reset value, 0x3F.*

**Idle mode**

If the AWU is not in use, then the AWUTB[3:0] bits the *Timebase selection register (AWU_TBR)* should be loaded with 0b0000 to reduce power consumption.

### 11.3.2 Time base selection

Please refer to the *Asynchronous prescaler register (AWU_APR)* and *Timebase selection register (AWU_TBR)* descriptions.

The AWU time intervals depend on the values of:

- AWUTB[3:0] bits. This gives the counter output rank.
- APR[5:0] bits. This gives the prescaler division factor ($APR_{DIV}$).

15 non-overlapped ranges of time intervals can be defined as follows:

**Table 21. Time base calculation table**

| Interval range | | AWUTB[3:0] | $APR_{DIV}$ formula for time interval calculation | $APR_{DIV}$ range |
|---|---|---|---|---|
| $f_{LS}$ = f | $f_{LS}$ = 38 kHz | | | |
| 2/f - 64/f | 0.052632 ms - 1.68 ms | 0001 | $APR_{DIV}/f_{LS}$ | 2 to 64 |
| 2x32/f - 2x2x32/f | 1.68 ms - 3.37 ms | 0010 | $2 \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| 2x64/f - 2x2x64/f | 3.37 ms - 6.74 ms | 0011 | $2^2 \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^2$x64/f - $2^2$x128/f | 6.74 ms - 13.47 ms | 0100 | $2^3 \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^3$x64/f - $2^3$x128/f | 13.47 ms - 26.95 ms | 0101 | $2^4 \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^4$x64/f - $2^4$x128/f | 26.95 ms - 53.89 ms | 0110 | $2^5 \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^5$x64/f - $2^5$x128/f | 53.89 ms - 107.8 ms | 0111 | $2^6 \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^6$x64/f - $2^6$x128/f | 107.8 ms - 215.6 ms | 1000 | $2^7 \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^7$x64/f - $2^7$x128/f | 215.6 ms - 431.2 ms | 1001 | $2^8 \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^8$x64/f - $2^8$x128/f | 431.2 ms - 862.3 ms | 1010 | $2^9 \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^9$x64/f - $2^9$x128/f | 0.86 s - 1.7 s | 1011 | $2^{10} \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^{10}$x64/f - $2^{10}$x128/f | 1.7 s - 3.4 s | 1100 | $2^{11} \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^{11}$x64/f - $2^{11}$x128/f | 3.4 s - 6.9 s | 1101 | $2^{12} \times APR_{DIV}/f_{LS}$ | 32 to 64 |
| $2^{11}$x130/f - $2^{11}$x320/f | 7.01 s - 17.24 s | 1110 | $5 \times 2^{11} \times APR_{DIV}/f_{LS}$ | 26 to 64 |
| $2^{11}$x330/f - $2^{12}$x960/f | 17.79 s - 103.5 s | 1111 | $30 \times 2^{11} \times APR_{DIV}/f_{LS}$ | 11 to 64 |

In order to obtain the right values for AWUTB[3:0] and $APR_{DIV}$, you have to:

- First, search the interval range corresponding to the desired time interval. This gives the AWUTB[3:0] value.
- Then $APR_{DIV}$ can be chosen to obtain a time interval value as close as possible to the desired one. This can be done using the formulas listed in the table above.

*Note:* *If the target value is between $2^{11}$x128/$f_{LS}$ and $2^{11}$x130/$f_{LS}$ or between $2^{11}$x320/$f_{LS}$ and $2^{11}$x330/$f_{LS}$, the value closer to the target one must be chosen.*

Example 1

- $f_{LS}$ = 128 kHz
- Target time interval = 6 ms

The appropriate interval range is: 4 ms - 8 ms
so the AWUTB[3:0] value is 0x5.

The $APR_{DIV}$ value is:
6 ms = $2^4$ x $APR_{DIV}$ / $f_{LS}$ => $APR_{DIV}$ = ($6*10^{-3}$ x $f_{LS}$ ) / $2^4$ = 48
so the APR[5:0] value is 48 (0x30)

Example 2

- $f_{LS}$ = 128 kHz
- Target time interval = 3 s

The appropriate interval range is: 2.080 s - 5.120 s
So the AWUTB[3:0] value is 0xE.

The $APR_{DIV}$ value is:
3 s = 5 x $2^{11}$ x $APR_{DIV}$ / $f_{LS}$ => $APR_{DIV}$ = (3 x $f_{LS}$ ) / 5 x $2^{11}$ = 37.5
So the AWUTB[3:0] can be either 37 or 38 which gives a time base of 2.96s or 3.04s
respectively. This is not exactly 3s.

### 11.3.3     LSI clock frequency measurement

Due to the oscillator frequency dispersion, to obtain a precise AWU time interval or beeper
output, the exact LSI frequency has to be measured.

Use the following procedure:

1. Set the MSR bit in the *Control/status register (AWU_CSR)* to connect the LSI clock
   internally to a timer input capture.
2. Measure the frequency of the LSI clock using the Timer input capture interrupt.
3. Write the appropriate value in the APR [5:0] bits in the *Asynchronous prescaler register
   (AWU_APR)* to adjust the AWU time interval to the desired length. The AWUTB[3:0]
   bits can be modified to select different time intervals.

LSI clock frequency measurement can also be used to calibrate the beeper frequency (see
*Section 12.2.2*).

# 11.4 AWU registers

## 11.4.1 Control/status register (AWU_CSR)

Address offset: 0x00

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | AWUF | AWUEN | Reserved | | | MSR |
| r | | rc_r | rw | r | | | rw |

Bits 7:6 Reserved

Bit 5 **AWUF**: Auto-wakeup flag

This bit is set by hardware when the AWU module generates an interrupt and cleared by reading the AWU_CSR register. Writing to this bit does not change its value.

  0: No AWU interrupt occurred
  1: AWU interrupt occurred

Bit 4 **AWUEN**: Auto-wakeup enable

This bit is set and cleared by software. It enables the auto-wakeup feature. If the microcontroller enters Active-halt or Wait mode, the AWU feature wakes up the microcontroller after a programmable time delay.

  0: AWU (Auto-wakeup) disabled
  1: AWU (Auto-wakeup) enabled

Bits 3:1 Reserved

Bit 0 **MSR**: Measurement enable

This bit connects the $f_{LS}$ clock to a timer input capture. This allows the timer to be used to measure the LSI frequency ($f_{LS}I$).

  0: Measurement disabled
  1: Measurement enabled

*Note: Refer to the datasheet for information on which timer input capture can be connected to the LSI clock in the specific product).*

### 11.4.2 Asynchronous prescaler register (AWU_APR)

Address offset: 0x01

Reset value: 0x3F

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | APR[5:0] | | | | | |
| r | | rw | | | | | |

Bits 7:6 Reserved

Bits 5:0 **APR[5:0]**: Asynchronous prescaler divider

These bits are written by software to select the prescaler divider ($APR_{DIV}$) feeding the counter clock.

0x00: $APR_{DIV}$ = 2  0x0E: $APR_{DIV}$ = 16

0x01: $APR_{DIV}$ = 3  0x0F: $APR_{DIV}$ = 17

... ...

0x06: $APR_{DIV}$ = 8  0x3E: $APR_{DIV}$ = 64

*Note: This register must not be kept at its reset value (0x3F)*

### 11.4.3 Timebase selection register (AWU_TBR)

Address offset: 0x02

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | AWUTB[3:0] | | | |
| r | | | | rw | | | |

Bits 7:4 Reserved

Bits 3:0 **AWUTB[3:0]**: Auto-wakeup timebase selection

These bits are written by software to define the time interval between AWU interrupts. AWU interrupts are enabled when AWUEN = 1.

0000: No interrupt

0001: $APR_{DIV}/f_{LS}$  0010: $2 \times APR_{DIV}/f_{LS}$  0011: $2^2 APR_{DIV}/f_{LS}$

0100: $2^3 APR_{DIV}/f_{LS}$  0101: $2^4 APR_{DIV}/f_{LS}$  0110: $2^5 APR_{DIV}/f_{LS}$

0111: $2^6 APR_{DIV}/f_{LS}$  1000: $2^7 APR_{DIV}/f_{LS}$  1001: $2^8 APR_{DIV}/f_{LS}$

1010: $2^9 APR_{DIV}/f_{LS}$  1011: $2^{10} APR_{DIV}/f_{LS}$  1100: $2^{11} APR_{DIV}/f_{LS}$

1101: $2^{12} APR_{DIV}/f_{LS}$  1110: $5 \times 2^{11} APR_{DIV}/f_{LS}$  1111: $30 \times 2^{11} APR_{DIV}/f_{LS}$

### 11.4.4 AWU register map and reset values

**Table 22. AWU register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | AWU_CSR<br>Reset value | -<br>0 | -<br>0 | AWUF<br>0 | AWUEN<br>0 | -<br>0 | -<br>0 | -<br>0 | MSR<br>0 |
| 0x01 | AWU_APR<br>Reset value | -<br>0 | -<br>0 | APR5<br>1 | APR4<br>1 | APR3<br>1 | APR2<br>1 | APR1<br>1 | APR0<br>1 |
| 0x02 | AWU_TBR<br>Reset value | -<br>0 | -<br>0 | -<br>0 | -<br>0 | AWUTB3<br>0 | AWUTB2<br>0 | AWUTB1<br>0 | AWUTB0<br>0 |

# 12 Beeper (BEEP)

## 12.1 Beeper introduction

This function generates a beep signal in the range of 1, 2 or 4 kHz when the LSI clock is operating at a frequency of 38 kHz.

**Figure 15. Beep block diagram**
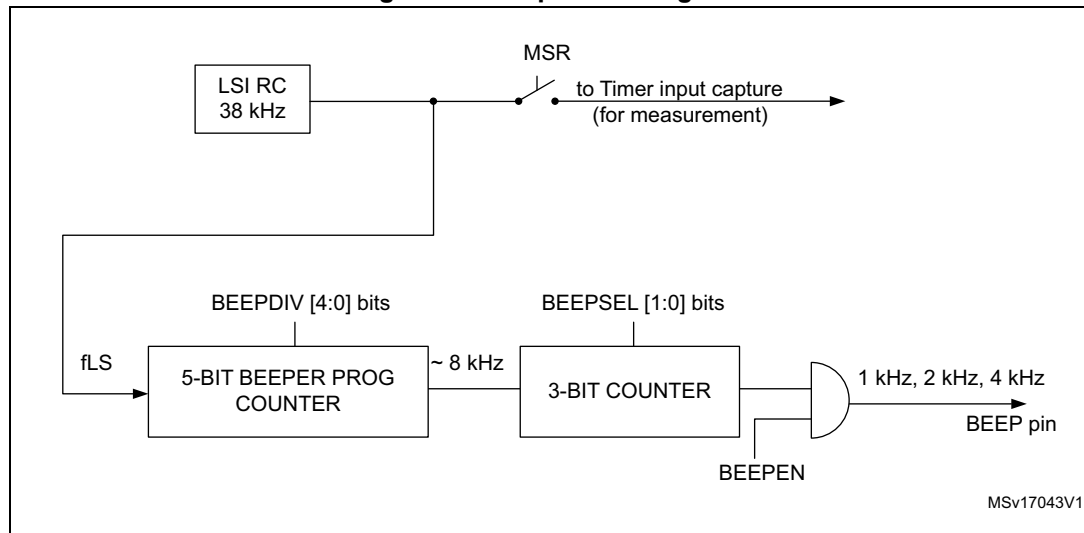


## 12.2 Beeper functional description

### 12.2.1 Beeper operation

To use the beep function, perform the following steps in order:

1. Calibrate the LSI clock frequency as described in *Section 12.2.2: Beeper calibration* to define BEEPDIV[4:0] value.

2. Select 1 kHz, 2 kHz or 4 kHz output frequency by writing to the BEEPSEL[1:0] bits in the *Beeper control/status register (BEEP_CSR)*.

3. Set the BEEPEN bit in the *Beeper control/status register (BEEP_CSR)* to enable the LSI clock source.

*Note:* *The prescaler counter starts to count only if BEEPDIV[4:0] value is different from its reset value, 0x1F.*

### 12.2.2 Beeper calibration

This procedure can be used to calibrate the LSI 38 kHz clock in order to reach the standard frequency output, 1 kHz, 2 kHz or 4 kHz.

Use the following procedure:

1.  Measure the LSI clock frequency (refer to *Section 11.3.3: LSI clock frequency measurement* above).

2.  Calculate the $BEEP_{DIV}$ value as follows, where A and x are the integer and fractional part of $f_{LS}/8$ (in kHz):

    $BEEP_{DIV}$ = A-2 when x is less than or equal to A/(1+2*A), else

    $BEEP_{DIV}$ = A-1

3.  Write the resulting $BEEP_{DIV}$ value in the BEEPDIV[4:0] bits in the *Beeper control/status register (BEEP_CSR)*.

## 12.3 Beeper registers

### 12.3.1 Beeper control/status register (BEEP_CSR)

Address offset: 0x00
Reset value: 0x1F

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BEEPSEL[1:0] | | BEEPEN | BEEPDIV[4:0] | | | | |
| rw | | rw | rw | | | | |

Bits 7:6 **BEEPSEL[1:0]**: Beep selection

These bits are set and cleared by software to select 1, 2 or 4 kHz beep output when calibration is done.
00: $f_{LS}/(8 \times BEEP_{DIV})$ kHz output
01: $f_{LS}/(4 \times BEEP_{DIV})$ kHz output
1x: $f_{LS}/(2 \times BEEP_{DIV})$ kHz output

Bit 5 **BEEPEN**: Beep enable

This bit is set and cleared by software to enable the beep feature.
0: Beep disabled
1: Beep enabled

Bits 4:0 **BEEPDIV[4:0]**: Beep prescaler divider

These bits are set and cleared by software to define the Beeper prescaler dividing factor $BEEP_{DIV}$.
0x00: $BEEP_{DIV}$ = 2
0x01: $BEEP_{DIV}$ = 3
...
0x0E: $BEEP_{DIV}$ = 16
0x0F: $BEEP_{DIV}$ = 17
0x1E: $BEEP_{DIV}$ = 32
*Note: This register must not be kept at its reset value (0x1F)*

## 12.3.2    Beeper register map and reset values

**Table 23. Beeper register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | BEEP_CSR Reset value | BEEPSEL[2:0] 00 | | BEEPEN 0 | BEEPDIV[4:0] 11111 | | | | |

# 13 Independent watchdog (IWDG)

## 13.1 IWDG introduction

The independent watchdog peripheral can be used to resolve processor malfunctions due to hardware or software failures. It is clocked by the 38 kHz LSI internal RC clock source, and thus stays active even if the main clock fails.

## 13.2 IWDG functional description

*Figure 16* shows the functional blocks of the independent watchdog module.

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 13.3: IWDG registers* for details on the IWDG registers.

**Figure 16. Independent watchdog block diagram**
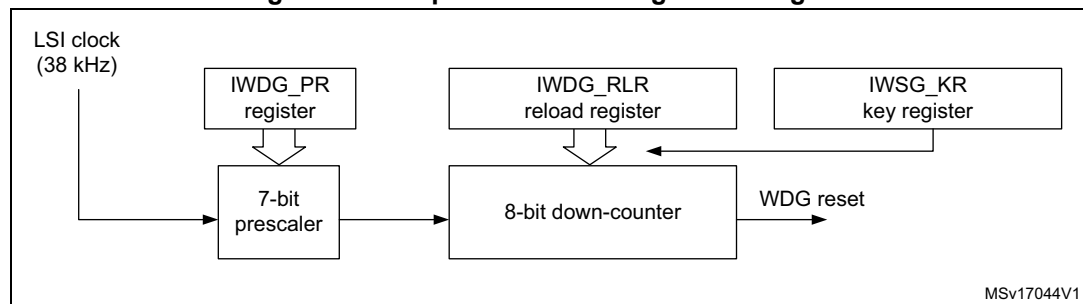


### Hardware watchdog feature

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 the software before the counter reaches end of count. Refer to the option byte description in the datasheet.

**Timeout period**

The maximum timeout period can be configured through the IWDG_PR and IWDG_RLR registers. It is determined by the following equation:

$$T = T_{LSI} \times P \times R$$

where:

T = Maximum 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:

D = T + 3 x $T_{LSI}$

where D= delay between the last refresh operation and the IWDG reset.

**Table 24. Min/Max IWDG timeout (LSI clock frequency = 38 kHz)**

| Prescaler divider | PR[2:0] bits | Timeout (ms) | |
|---|---|---|---|
| | | RL[7:0]= 0x00 | RL[7:0]= 0xFF |
| /4 | 0 | 0.11 | 26.95 |
| /8 | 1 | 0.21 | 53.89 |
| /16 | 2 | 0.42 | 107.79 |
| /32 | 3 | 0.84 | 215.58 |
| /64 | 4 | 1.68 | 431.16 |
| /128 | 5 | 3.37 | 862.32 |
| /256 | 6 | 6.74 | 1724.63 |

**Using the IWDG in Halt/Active-halt mode**

The IWDG can continue to work in Halt or Active-halt mode, depending on the configuration of the IWDG_HALT option byte. In this case, it can wake up the device from one of these modes. For more details, please refer to the Option Byte description in the datasheet.

*Note:*       *The application must configure correctly the IWDG timeout and refresh the IWDG counter before executing the HALT instruction, to avoid unexpected IWDG reset.*

## 13.3 IWDG registers

### 13.3.1 Key register (IWDG_KR)

Address offset: 0x00

Reset value: 0xXX

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | KEY[7:0] | | | | |
| | | | w | | | | |

Bits 7:0  **KEY[7:0]**: Key value

The KEY_REFRESH value must be written by software at regular intervals, otherwise the watchdog generates an MCU reset when the counter reaches 0.

If the IWDG is not enabled by option byte (see datasheet for option byte description), the KEY_ENABLE value is the first value to be written in this register.

**KEY_ENABLE** value = 0xCC
Writing the KEY_ENABLE value starts the IWDG.
**KEY_REFRESH** value = 0xAA
Writing the KEY_REFRESH value refreshes the IWDG.
**KEY_ACCESS** value = 0x55
Writing the KEY_ACCESS value enables the access to the protected IWDG_PR and IWDG_RLR registers (see *Section 13.2*).

### 13.3.2 Prescaler register (IWDG_PR)

Address offset: 0x01

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | Reserved | | | | PR[2:0] | |
| | | r | | | | rw | |

Bits 7:3  Reserved

Bits 2:0  **PR[2:0]:** Prescaler divider

These bits are write access protected (see *Section 13.2*). They can be written by software to select the prescaler divider feeding the counter clock.

   000: divider /4
   001: divider /8
   010: divider /16
   011: divider /32
   100: divider /64
   101: divider /128
   110: divider /256
   111: Reserved

### 13.3.3 Reload register (IWDG_RLR)

Address offset: 0x02

Reset value: 0xFF

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | RL[7:0] | | | | |
| | | | rw | | | | |
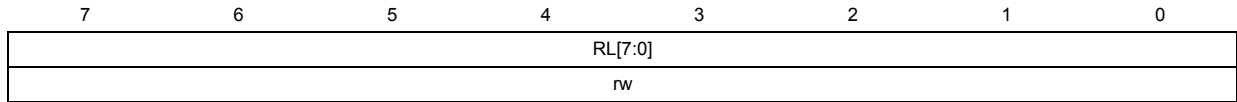
Bits 7:0 **RL[7:0]**: Watchdog counter reload value

These bits are write access protected (see *Section 13.2*). They are 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. Refer to *Table 24*.

### 13.3.4 IWDG register map and reset values

**Table 25. IWDG register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | IWDG_KR | | | | KEY[7:0] | | | | |
| | Reset value | | | | xxxxxxxx | | | | |
| 0x01 | IWDG_PR | - | - | - | - | - | | PR2[2:0] | |
| | Reset value | 0 | 0 | 0 | 0 | 0 | | 000 | |
| 0x02 | IWDG_RLR | | | | RL7[7:0] | | | | |
| | Reset value | | | | 11111111 | | | | |

# 14 Inter-integrated circuit (I2C) interface

## 14.1 I2C introduction

I2C (inter-integrated circuit) bus interface serves as an interface between the microcontroller and the serial I2C bus. It provides multi-master capability, and controls all I2C bus-specific sequencing, protocol, arbitration and timing. It supports standard and fast speed modes.

## 14.2 I2C main features

- Parallel-bus/I2C protocol converter
- Multi-master capability: the same interface can act as 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
    - Acknowledgement 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 wakeup from Halt interrupt
- Wakeup capability:
    - MCU wakes up from Low power mode on address detection in slave mode.
- Optional clock stretching

## 14.3      I2C general description

In addition to receiving and transmitting data, this interface converts it from serial to parallel format and vice versa. The interrupts are enabled or disabled by software. The interface is connected to the I2C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz), or fast (up to 400 kHz) I2C bus.

**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 slave mode. The interface automatically switches from slave to master, after it generates a START condition and from master to slave, if an arbitration loss or a STOP generation occurs, allowing Multi-Master capability.

Communication flow

In 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 master mode by software.

In 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 in 7-bit mode, two in 10-bit mode). The address is always transmitted in 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 the following figure.

**Figure 17. I2Cbus protocol**
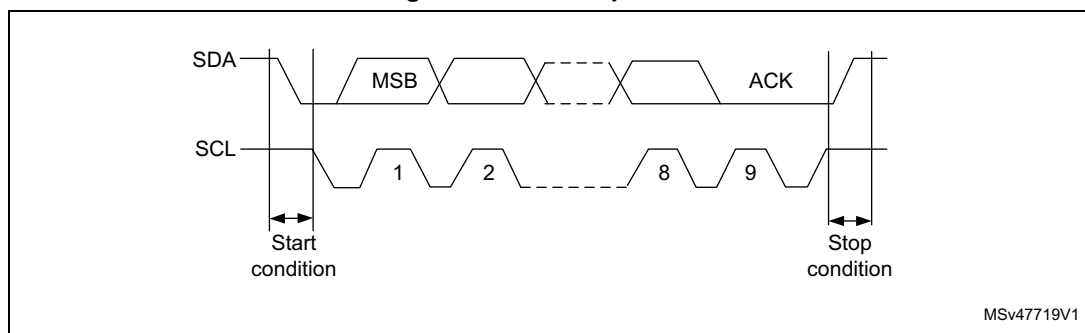


Acknowledge may be enabled or disabled by software. The I2C interface addresses (7-/10-bit and/or general call address) can be selected by software.

The block diagram of the I2C interface is shown in *Figure 18*.

**Figure 18. I2C block diagram**
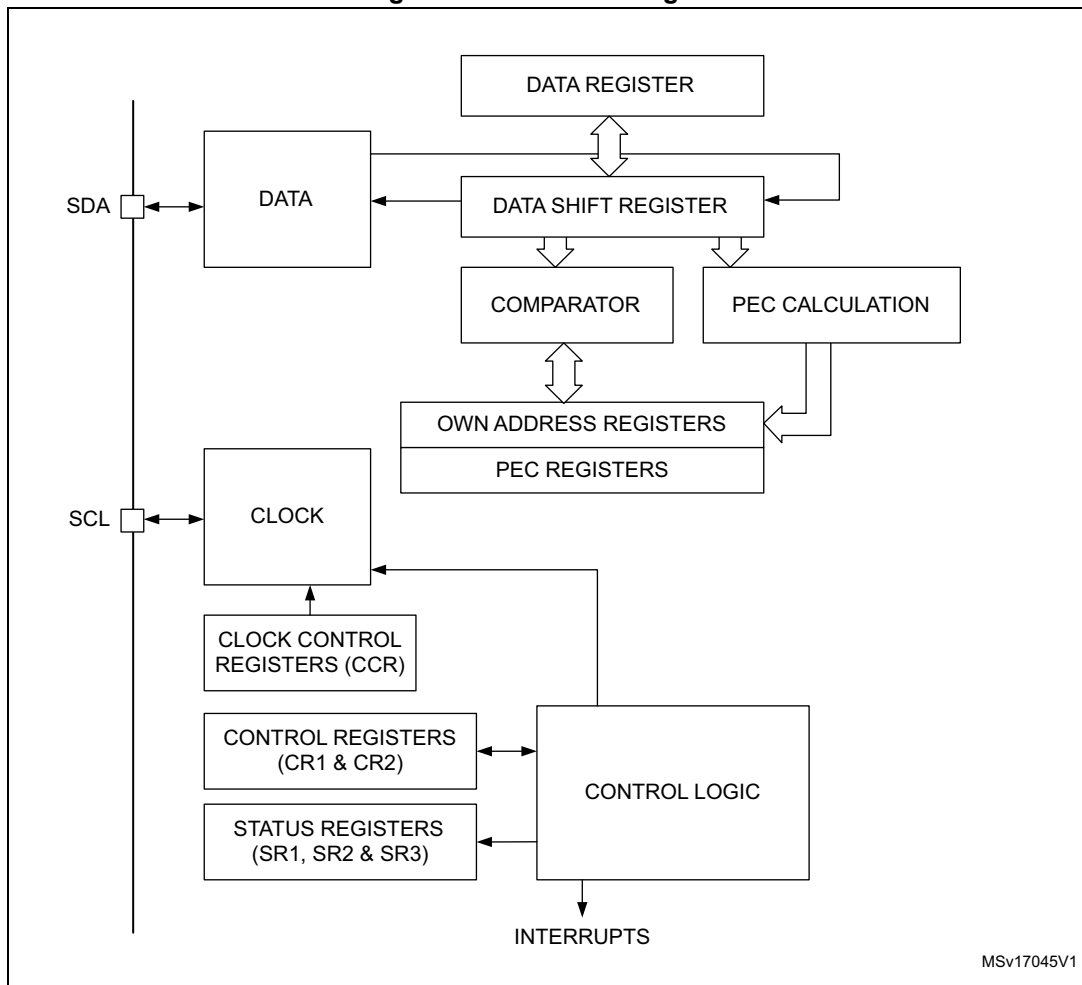
## 14.4 I2C functional description

By default the I2C interface operates in Slave mode. To switch from default Slave mode to Master mode a Start condition generation is needed.

### 14.4.1 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 (OAR1L and OAR2 if ENDUAL = 1) or the General Call address (if ENGC = 1).

*Note:* *In 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 10-bit mode, after receiving the address sequence the slave is always in Receiver mode. It will enter 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 Receiver or Transmitter mode.

### Slave transmitter

Following the address reception and after clearing ADDR, the slave sends bytes from the DR register to the SDA line via the internal shift register.

The slave stretches SCL low until ADDR is cleared and DR filled with the data to be sent (see Transfer sequencing EV1 EV3 in the following figure).

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 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 until BTF is cleared, by reading the SR1 register and then writing to the DR register, stretching SCL low.

#### Figure 19. Transfer sequence diagram for slave transmitter
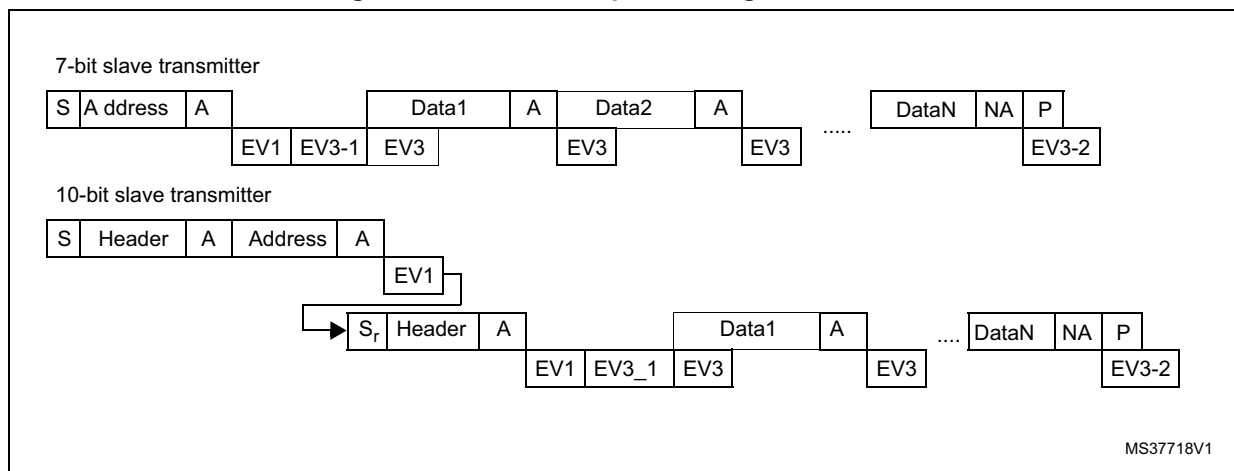


1. Legend:
   **S**= Start, **S$_r$** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge, **EVx** = Event (with interrupt if ITEVTEN=1)
   **EV1:** ADDR =1, cleared by reading SR1 register followed by reading SR3.
   **EV3-1:** TXE=1, shift register empty, data register empty, write Data1 in DR.
   **EV3**: TXE=1, shift register not empty, data register empty, cleared by writing DR.
   **EV3-2:** AF=1, AF is cleared by writing '0' in AF bit of SR2 register.

2. EV1 and EV3-1 events stretch SCL low until the end of the corresponding software sequence.

3. EV3 software sequence must be performed before the end of the current byte transfer. In case EV3 software sequence can not be managed before the end of the current byte transfer, it is recommended to use BTF instead of TXE with the drawback of slowing the communication.

### Slave receiver

Following the address reception and after clearing 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 ITBUFEN bit is set.

If 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 until BTF is cleared, by reading the SR1 register and then reading the DR register, stretching SCL low (see *Figure 20*).

**Figure 20. Transfer sequence diagram for slave receiver**
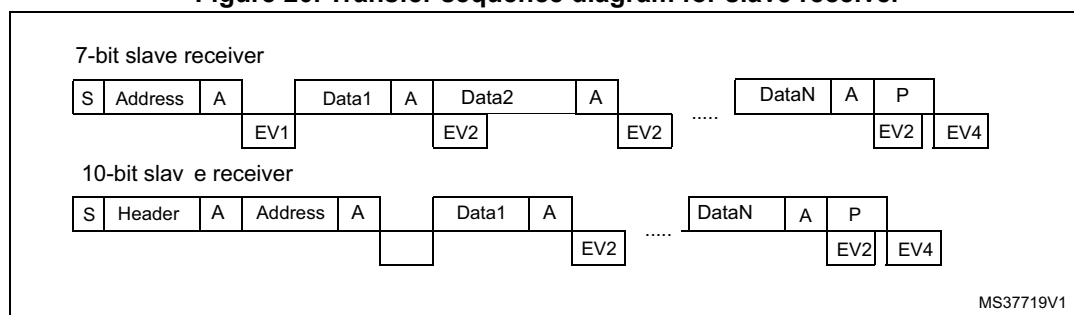


1. Legend:
   **S**= Start, **S<sub>r</sub>** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge, **EVx**= Event (with interrupt if ITEVTEN=1)
   **EV1:** ADDR =1, cleared by reading SR1 register followed by reading SR3.
   **EV2:** RXNE=1, cleared by reading DR register.
   **EV4:** STOPF=1, cleared by reading SR1 register followed by writing CR2 register

2. EV1 event stretches SCL low until the end of the corresponding software sequence.

3. 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 I2C 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.

5. See also: *Note 8 on page 119*.

**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.

STOPF is cleared by a read of the SR1 register followed by a write to the CR2 register (see *Figure 21*).

### 14.4.2 I2C master mode

In 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. 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 master mode.

- Program the peripheral input clock in 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

### SCL master clock generation

The CCR bits are used to generate the high and low level of the SCL clock, starting from the generation of the rising and falling edge (respectively). As a slave may stretch the SCL line, the peripheral checks the SCL input from the bus at the end of the time programmed in TRISE bits after the rising edge generation.

- If the SCL line is low, it means that a slave is stretching the bus, and the high level counter stops until the SCL line is detected high. This allows to guarantee the minimum HIGH period of the SCL clock parameter.
- If the SCL line is high, the high level counter keeps on counting.

Indeed, the feedback loop from the SCL rising edge generation by the peripheral to the SCL rising edge detection by the peripheral takes time even if no slave stretches the clock. This loopback duration is linked to SCL rising time (impacting SCL $V_{IH}$ input detection), plus delay due to the analog noise filter present on SCL input path, plus delay due to internal SCL input synchronization with I2C Peripheral clock. The maximum time used by the feedback loop is programmed in TRISE bits, so that the SCL frequency remains stable whatever the SCL rising time.

### Start condition

Setting the START bit causes the interface to generate a Start condition and to switch to Master mode (MSL bit set) when the BUSY bit is cleared.

*Note:* *In master mode, setting the START bit causes the interface to generate a Re-Start 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 ().

**Slave address transmission**

Then the slave address is sent to the SDA line via the internal shift register.

- In 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 (see *Figure 21* & *Figure 22*  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 (see *Figure 21* & *Figure 22*  Transfer sequencing EV6).

- In 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 bit is set.

  Then the master waits for a read of the SR1 register followed by a read in the SR3 register (see *Figure 21* & *Figure 22*  Transfer sequencing EV6).

  The master can decide to enter Transmitter or Receiver mode depending on the LSB of the slave address sent.

- In 7-bit addressing mode,
  - To enter Transmitter mode, a master sends the slave address with LSB reset.
  - To enter Receiver mode, a master sends the slave address with LSB set.

- In 10-bit addressing mode,
  - To enter Transmitter mode, a master sends the header (11110xx0) and then the slave address, (where xx denotes the two most significant bits of the address).
  - To enter Receiver mode, a master sends the header (11110xx0) and then the slave address. Then it should send a repeated Start condition followed by the header (11110xx1), (where xx denotes the two most significant bits of the address).

  The TRA bit indicates whether the master is in Receiver or Transmitter mode.

### Master transmitter

Following the address transmission and after clearing 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, (see *Figure 21*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, BTF is set and the interface waits until BTF is cleared, by reading the SR1 register and then writing to the DR register, stretching 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 (see *Figure 21* Transfer sequencing EV8_2). The interface goes automatically back to slave mode (MSL bit cleared).

*Note:* *Stop condition should be programmed during EV8_2 event, when either TXE or BTF is set.*

**Figure 21. Transfer sequence diagram for master transmitter**
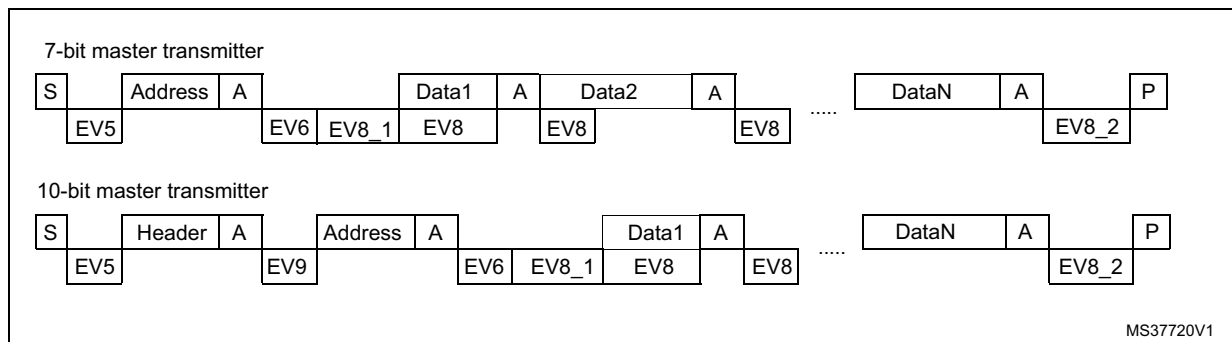


1. Legend:
   **S**= Start, **S$_r$** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge,
   EVx= Event (with interrupt if ITEVTEN=1)
   **EV5:** SB=1, cleared by reading SR1 register followed by writing DR register with Address.
   **EV6:** ADDR=1, cleared by reading SR1 register followed by reading SR3.
   **EV8_1:** TXE=1, shift register empty, data register empty, write DR register.
   **EV8:** TXE=1, shift register not empty, data register empty, cleared by writing DR register.
   **EV8_2:** TXE=1, BTF = 1, Program STOP request. TXE and BTF are cleared by HW by stop condition
   **EV9:** ADD10=1, cleared by reading SR1 register followed by writing DR register. See also:*Note 8 on page 119*

2. EV8 software sequence must be performed before the end of the current byte transfer. In case EV8 software sequence can not be managed before the end of the current byte transfer, it is recommended to use BTF instead of TXE with the drawback of slowing the communication.

**Master receiver**

Following the address transmission and after clearing ADDR, the I$^2$C interface enters 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 ().

If the RXNE bit is set and the data in the DR register was not read 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 I2C_SR1 and then I2C_DR, stretching 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/Re-Start 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/Re-Start 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 EV6 (in *EV6-1* just after ADDR is cleared).

After the Stop condition generation, the interface goes automatically back to slave mode (MSL bit cleared).

**Method 1:** This method is for the case when the I2C is used with interrupts that have the highest priority in the application.

#### Figure 22. Method 1: transfer sequence diagram for master receiver
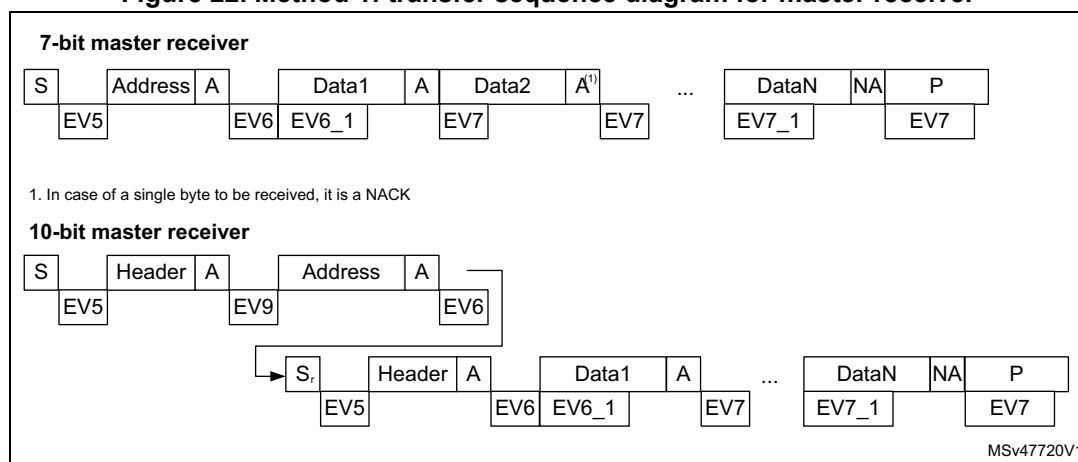


1. In case of a single byte to be received, it is a NACK

1. Legend:
   **S**= Start, **S$_r$** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge,
   EVx= Event (with interrupt if ITEVTEN=1)

**EV5:** SB=1, cleared by reading SR1 register followed by writing DR register.
**EV6:** ADDR=1, cleared by reading SR1 register followed by reading SR3. In 10-bit master receiver mode, this sequence should be followed by writing CR2 with START = 1.
**EV6_1:** no associated flag event, used for 1 byte reception only. Program ACK=0 and STOP=1 after clearing ADDR.
**EV7:** RxNE=1, cleared by reading DR register.
**EV7_1:** RxNE=1, cleared by reading DR register, program ACK=0 and STOP request
**EV9:** ADD10=1, cleared by reading SR1 register followed by writing DR register.

2.  If the DR and shift registers are full, the next data reception ($I^2$C clock generation for slave) is performed after the EV7 event is cleared. In this case, EV7 does not overlap with data reception.

3.  If a single byte is received, it is NA.

4.  EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.

5.  EV7 software sequence must be completed before the end of the current byte transfer.In case EV7 software sequence can not be managed before the current byte end of transfer, it is recommended to use BTF instead of RXNE with the drawback of slowing the communication.

6.  The EV6_1 or EV7_1 software sequence must be completed before the ACK pulse of the current byte transfer.

7.  See also: *Note 8 on page 119*.

**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:

*   DataN_2 is not read, so that after DataN_1, the communication is stretched (both RxNE and BTF are set).

*   Then, the ACK bit must be cleared before reading DataN-2 in DR to make sure this bit has been cleared before the DataN Acknowledge pulse.

*   After that, just after reading DataN_2, software must set the STOP/ START bit and read DataN_1. After RxNE is set, read DataN.

This is illustrated in the following figure:

**Figure 23. Method 2: transfer sequence diagram for master receiver when N >2**
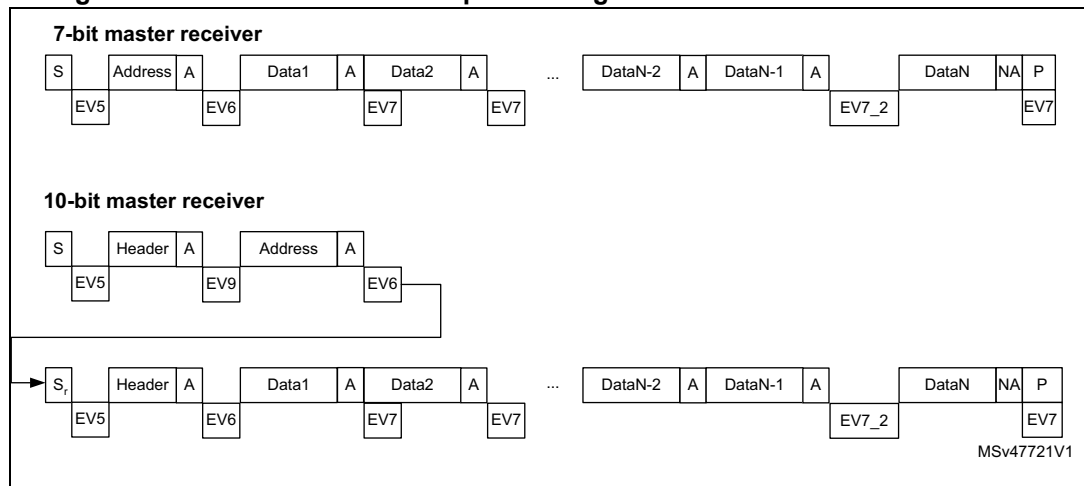


1.  Legend:
    **S**= Start, **S$_r$** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge,
    EVx= Event (with interrupt if ITEVTEN=1)
    **EV5:** SB=1, cleared by reading SR1 register followed by writing the DR register.
    **EV6:** ADDR1, cleared by reading SR1 register followed by reading SR3.
    In 10-bit master receiver mode, this sequence should be followed by writing CR2 with START = 1.
    **EV7:** RxNE=1, cleared by reading DR register.
    **EV7_2:** BTF = 1, DataN-2 in DR and DataN-1 in shift register, program ACK = 0, Read DataN-2 in DR.
    Program STOP = 1, read DataN-1.

**EV9:** ADD10= 1, cleared by reading SR1 register followed by writing DR register.

2. EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.

3. EV7 software sequence must be completed before the end of the current byte transfer. In case EV7 software sequence can not be managed before the current byte end of transfer, it is recommended to use BTF instead of RXNE, with the drawback of slowing the communication.

When 3 bytes remain to be read:

- RxNE = 1 => Nothing (DataN-2 not read).

- DataN-1 received

- BTF = 1 because both shift and data registers are full: DataN-2 in DR and DataN-1 in the shift register => SCL tied low: no other data will be received on the bus.

- Clear ACK bit

- Read DataN-2 in 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

The procedure described above is valid for N>2. The cases where a single byte or two bytes are to be received should be handled differently, as described below:

- Case of a single byte to be received:
    - In the ADDR event, clear the ACK bit.
    - Clear ADDR
    - Program the STOP/START bit.
    - Read the data after the RxNE flag is set.

- Case of two bytes to be received:
    - 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 twice

**Figure 24. Method 2: transfer sequence diagram for master receiver when N=2**
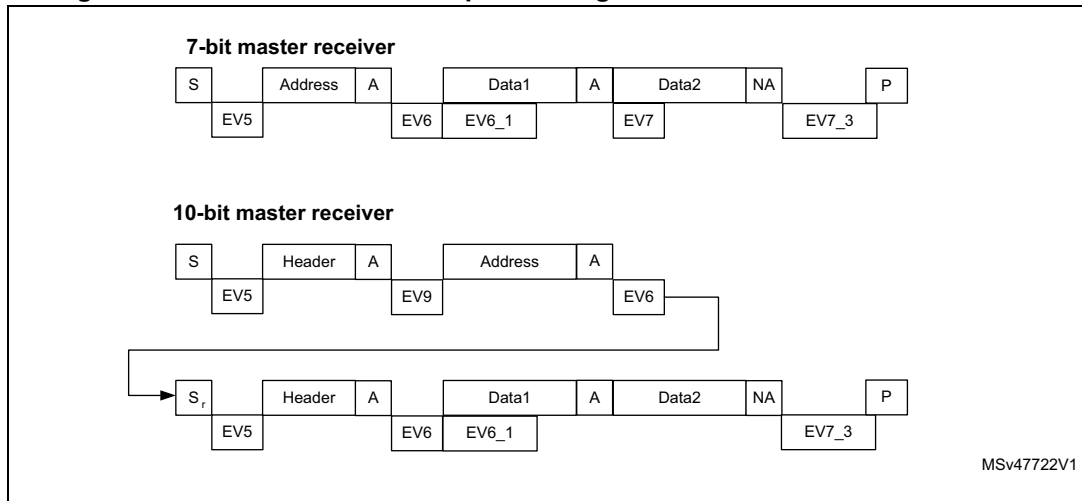


1. Legend:
   **S**= Start, **S$_r$** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge,
   EVx= Event (with interrupt if ITEVTEN=1).
   **EV5:** SB=1, cleared by reading SR1 register followed by writing the DR register.
   **EV6:** ADDR1, cleared by reading SR1 register followed by reading SR3.
   In 10-bit master receiver mode, this sequence should be followed by writing CR2 with START = 1.
   **EV6_1:** No associated flag event. The acknowledge should be disabled just after EV6, that is after ADDR is cleared
   **EV7_3:** BTF = 1, program STOP = 1, read DR twice (Read Data1 and Data2) just after programming the STOP.
   **EV9:** ADD10= 1, cleared by reading SR1 register followed by writing DR register.

2. EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.

3. EV6_1 software sequence must be completed before the ACK pulse of the current byte transfer.

**Figure 25. Method 2: transfer sequence diagram for master receiver when N=1**
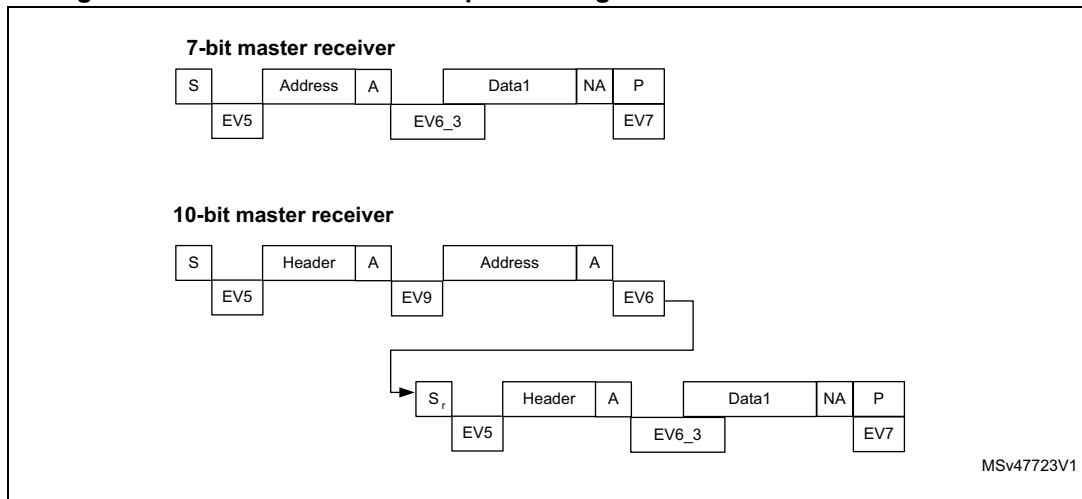


1. Legend:
   **S**= Start, **S$_r$** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge,
   EVx= Event (with interrupt if ITEVTEN=1).
   **EV5:** SB=1, cleared by reading SR1 register followed by writing the DR register.
   **EV6:** ADDR =1, cleared by reading SR1 resister followed by reading SR3 register.
   **EV6_3:** ADDR = 1, program ACK = 0, clear ADDR by reading SR1 register followed by reading SR3 register, program STOP =1 just after ADDR is cleared.
   **EV7:** RxNE =1, cleared by reading DR register.

**EV9:** ADD10= 1, cleared by reading SR1 register followed by writing DR register.

2. EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.

3. EV6_3 software sequence must be completed before the ACK pulse of the current byte transfer.

## 14.4.3 Error conditions

The following are the error conditions which may cause communication to fail.

### Bus error (BERR)

This error occurs when the I2C 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 I2C 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 I2C interface goes automatically back to slave mode (the MSL bit is cleared)
- When the I²C 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)**

An Overrun error can occur in slave mode when clock stretching is disabled and the I2C interface is receiving data. The interface has received a byte (RXNE = 1) and the data in 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 re-transmit the last received byte.

Underrun error can occur in slave mode when clock stretching is disabled and the I2C 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$^2$C bus standard.
- For the first byte to be transmitted, the DR must be written after ADDR is cleared and before the first SCL rising edge. If it is not possible, the receiver must discard the first data.

### 14.4.4 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 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 SR1 and then read the byte in the Data Register or write to CR2 (both buffer and shift register are full).
- If clock stretching is disabled in Slave mode:
  - Overrun error in case of RXNE = 1 and no read of 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 DR has been done before the next byte must be transmitted. The same byte will be sent again.
  - Write Collision not managed.

## 14.5 I2C low power modes

**Table 26. I2C interface behavior in low power modes[1]**

| Mode | Description |
|------|-------------|
| Wait | No effect on I2C interface.<br>I2C interrupts cause the device to exit from Wait mode. |
| Halt | **In slave mode**: Communication is reset, except for configuration registers. Device is in slave mode.<br>Wakeup from Halt interrupt is generated if ITEVTEN = 1 and address matched (including allowed headers).<br>The matched address is not acknowledged in Halt mode so the master has to send it again when the CPU is woken up to receive an acknowledge.<br>If NOSTRETCH = 0, SCLH will be stretched after acknowledge pulse in Halt mode until WUFH is cleared by software;<br>None of the flags are set by the address which wakes up the CPU.<br>**In master mode**: Communication is frozen until the CPU is woken up. Wakeup from Halt flag and interrupt are generated if ITEVTEN=1 and there is a HALT instruction.<br>*Note:       It is forbidden to enter Halt mode while a communication is on going.* |

1. I2C cannot operate at low power run and low power wait modes due to low PCLK frequency.

## 14.6 I2C interrupts

**Table 27. 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 | | No |
| Arbitration loss (Master) | ARLO | | | |
| Acknowledge failure | AF | | | |
| Overrun/underrun | OVR | | | |

**Figure 26. I2C interrupt mapping diagram**
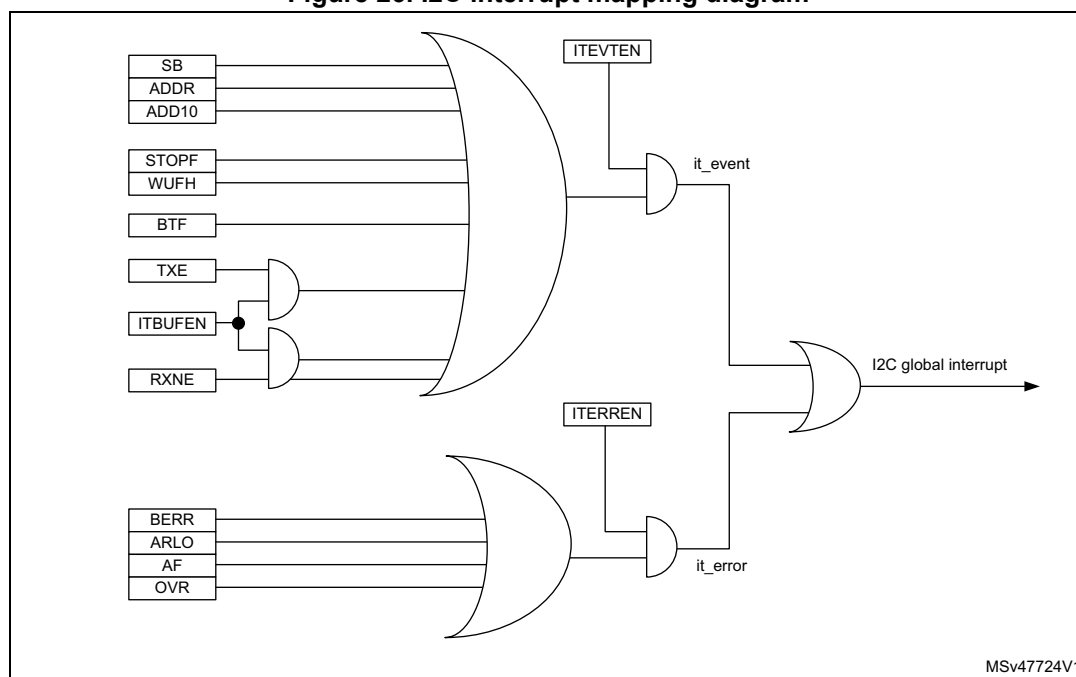
## 14.7 I2C registers

### 14.7.1 Control register 1 (I2C_CR1)

Address offset: 0x00
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| NOSTRETCH | ENGC | Reserved | | | | | PE |
| rw | rw | r | | | | | rw |

Bit 7 **NOSTRETCH**: Clock stretching disable (Slave mode)
This bit is used to disable clock stretching in slave mode when ADDR or BTF flag is set, until it is reset by software.
0: Clock stretching enabled
1: Clock stretching disabled

Bit 6 **ENGC**: General call enable
0: General call disabled. Address 0x00 is NACKed.
1: General call enabled. Address 0x00 is ACKed.

Bits 5:1 Reserved

Bit 0 **PE**: Peripheral enable
0: Peripheral disable
1: Peripheral enable: the corresponding I/Os are selected as alternate functions.

*Note:* *If this bit is reset while a communication is on going, the peripheral is disabled at the end of the current communication, when back to IDLE state.*
*All bit resets due to PE=0 occur at the end of the communication.*

### 14.7.2 Control register 2 (I2C_CR2)

Address offset: 0x01

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SWRST | | Reserved | | POS | ACK | STOP | START |
| rw | | r | | rw | rw | rw | rw |

Bit 7 **SWRST**: Software reset

When set, the I2C is at reset state. Before resetting this bit, make sure the I2C lines are released and the bus is free.

0: I2C Peripheral not at reset state
1: I2C Peripheral at reset state

*Note:* *This bit can be used in case the BUSY bit is set to '1' when no stop condition has been detected on the bus.*

Bits 6::4 Reserved

Bit 3 **POS**: Acknowledge position (for data reception).

This bit is set and cleared by software and cleared by hardware when PE=0.

0: ACK bit controls the (N)ACK of the current byte being received in the shift register.
1: 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 Method 2: transfer sequence diagram for master receiver when N=2) 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 ADDR is cleared.*

Bit 2 **ACK**: Acknowledge enable

This bit is set and cleared by software and cleared by hardware when PE=0.

0: No acknowledge returned
1: Acknowledge returned after a byte is received (matched address or data)

Bit 1 **STOP**: Stop generation

The bit is set and cleared by software, cleared by hardware when a Stop condition is detected, set by hardware when a timeout error 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 0 **START**: Start generation

This bit is set and cleared by software and cleared by hardware when 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 STOP or START is set, the user must not perform any write access to I2C_CR2 before the control bit is cleared by hardware. Otherwise, a second STOP or START request may occur.*

### 14.7.3 Frequency register (I2C_FREQR)

Address offset: 0x02
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | FREQ[5:0] | | | | | |
| r | | rw | | | | | |

Bits 7:6 Reserved

Bits 5:0 **FREQ[5:0]** Peripheral clock frequency. [1]

The FREQ field is used by the peripheral to generate data setup and hold times compliant with the I2C specifications. The FREQ bits must be programmed with the peripheral input clock frequency value:

The allowed range is between 1 MHz and 16 MHz

000000: not allowed
000001: 1 MHz
000010: 2 MHz
...
010000: 16 MHz
Higher values: not allowed

1. The minimum peripheral clock frequencies for respecting the $I^2C$ bus timings are:
   1 MHz for standard mode and 4 MHz for fast mode

### 14.7.4 Own address register LSB (I2C_OARL)

Address offset: 0x03
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ADD[7:1] | | | | | | | ADD0 |
| rw | | | | | | | rw |

Bits 7:1 **ADD[7:1]** Interface address
     bits 7:1 of address

Bit 0 **ADD[0]** Interface address
     7-bit addressing mode: don't care
     10-bit addressing mode: bit 0 of address

### 14.7.5 Own address register MSB (I2C_OARH)

Address offset: 0x04
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ADDMODE | ADDCONF | Reserved | | | ADD[9:8] | | Reserved |
| rw | rw | r | | | rw | | r |

Bit 7 **ADDMODE** Addressing mode (Slave mode)
     0: 7-bit slave address (10-bit address not acknowledged)
     1: 10-bit slave address (7-bit address not acknowledged)

Bit 6 **ADDCONF** Address mode configuration
     This bit must set by software (must always be written as '1').

Bits 5:3 Reserved

Bits 2:1 **ADD[9:8]** Interface address
     10-bit addressing mode: bits 9:8 of address.

Bit 0 Reserved

### 14.7.6 Data register (I2C_DR)

Address offset: 0x06
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DR[7:0] | | | | | | | |
| rw | | | | | | | |

Bits 7:0 **DR[7:0]**: Data register [1][2][3]

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 DR once the transmission is started (TXE=1)

– Receiver mode: Received byte is copied into DR (RXNE=1). A continuous transmit stream can be maintained if DR is read before the next data is received (RXNE=1).

1. In slave mode, the address is not copied into DR.

2. Write collision is not managed (DR can be written if TXE=0).

3. If an ARLO event occurs on ACK pulse, the received byte is not copied into DR and so cannot be read.

## 14.7.7 Status register 1 (I2C_SR1)

Address offset: 0x07
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TXE | RXNE | Reserved | STOPF | ADD10 | BTF | ADDR | SB |
| r | r | r | r | r | r | r | r |

Bit 7 **TXE**: Data register empty (transmitters) [1]

    0: Data register not empty
    1: Data register empty

– Set when DR is empty in transmission. TXE is not set during address phase.

– Cleared by software writing to the DR register or by hardware after a start or a stop condition or when PE=0.

*Note: TXE cannot be cleared by writing the first data in transmission or by writing a data when the BTF bit is set as in both cases, the DR register is still empty.*

Bit 6 **RXNE**: Data register not empty (receivers) [2] [3]

    0: Data register empty
    1: Data register not empty

– Set when data register is not empty in receiver mode. RXNE is not set during address phase.

– Cleared by software reading or writing the DR register or by hardware when PE=0.

*Note: RXE cannot be cleared by reading a data when the BTF bit is set as the DR register is still full in this case.*

Bit 5 Reserved

Bit 4 **STOPF**: Stop detection (Slave mode) [4][5]

    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 acknowledge (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 3 **ADD10**: 10-bit header sent (Master mode) [6]

    0: No ADD10 event occurred.
    1: Master has sent first address byte (header).

– Set by hardware when the master has sent the first byte in 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 hardware when PE=0.

Bit 2 **BTF**: Byte transfer finished [7][8]

0: Data byte transfer not done
1: Data byte transfer succeeded

– Set by hardware when NOSTRETCH=0 and:

– In reception when a new byte is received (including ACK pulse) and DR has not been read yet (RXNE=1).

– In transmission when a new byte should be sent and DR has not been written yet (TXE=1).

– Cleared by software reading 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.

Bit 1 **ADDR**: Address sent (master mode)/matched (slave mode) [8][9]

This bit is cleared by software reading SR1 register followed reading 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 received slave address matched with the OAR registers content or a general call or a SMBus 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 byte.

*Note: ADDR is not set after a NACK reception*

Bit 0 **SB**: Start bit (Master mode) [8]

0: No Start condition
1: Start condition generated.

– Set when a Start condition generated.

– Cleared by software by reading the SR1 register followed by writing the DR register, or by hardware when PE=0

1. The interrupt will be generated when DR is copied into shift register after an ACK pulse. If a NACK is received, copy is not done and TXE is not set.

2. The interrupt will be generated when Shift register is copied into DR after an ACK pulse.

3. RXNE is not set in case of ARLO event.

4. The STOPF bit is not set after a NACK reception.

5. It is recommended to perform the complete clearing sequence (READ SR1 then WRITE CR2) after STOPF is set. Refer to *Figure 20: Transfer sequence diagram for slave receiver on page 101*

6. The ADD10 bit is not set after a NACK reception.

7. The BTF bit is not set after a NACK reception, or in case of an ARLO event.

8. Due to timing constraints, when in standard mode if CCR is less than 9 (i.e. with 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 I2C_DR, A ; once executed, the SB bit is then cleared.

9. In slave mode, it is recommended to perform the complete clearing sequence (READ SR1 then READ SR3) after ADDR is set. Refer to *Figure 20: Transfer sequence diagram for slave receiver on page 101*.

### 14.7.8 Status register 2 (I2C_SR2)

Address offset: 0x08
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | WUFH | Reserved | OVR | AF | ARLO | BERR |
| r | | rc_w0 | r | rc_w0 | rc_w0 | rc_w0 | rc_w0 |

Bits 7:6 Reserved

Bit 5 **WUFH**: Wakeup from Halt

0: no wakeup from Halt mode
1: 7-bit address or header match in Halt mode (slave mode) or Halt entered when in master mode.

*Note: This bit is set asynchronously in slave mode (during HALT mode). It is set only if ITEVTEN = 1.*
– cleared by software writing 0, or by hardware when PE=0.

Bit 4 Reserved

Bit 3 **OVR**: Overrun/underrun

0: No overrun/underrun
1: Overrun or underrun

– Set by hardware in slave mode when NOSTRETCH=1 and:
– In reception when a new byte is received (including ACK pulse) and the DR register has not been read yet. 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 is unspecified and a hold timing error occurs.*

Bit 2 **AF**: Acknowledge failure.

0: No acknowledge failure
1: Acknowledge failure

– Set by hardware when no acknowledge is returned.
– 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 interface loses the arbitration of the bus to another master.
– Cleared by software writing 0, or by hardware when PE=0.

After an ARLO event the interface switches back automatically to Slave mode (MSL=0).

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 SDA rising or falling edge while SCL is high, occurring in a non-valid position during a byte transfer.
– Cleared by software writing 0, or by hardware when PE=0.

### 14.7.9 Status register 3 (I2C_SR3)

Address offset: 0x09
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DUALF | Reserved | | GENCALL | Reserved | TRA | BUSY | MSL |
| r | r | | r | r | r | r | r |

Bit 7 **DUALF**: Dual flag (Slave mode)

0: Received address matched with OAR1
1: Received address matched with OAR2

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bits 6:5 Reserved

Bit 4 **GENCALL**: General call header (Slave mode)

0: No general call
1: General call header received when ENGC=1

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 3 Reserved

Bit 2 **TRA**: Transmitter/Receiver

0: Data bytes received
1: Data bytes transmitted
This bit is set depending on R/W bit of address byte, at the end of 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 PE=0.

Bit 1 **BUSY**: Bus busy

0: No communication on the bus
1: Communication ongoing on the bus

– Set by hardware on detection of SDA or 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 0 **MSL**: Master/Slave

0: Slave mode
1: Master mode

– Set by hardware as soon as the interface is in Master mode (SB=1).

– Cleared by hardware after detecting a Stop condition on the bus or a loss of arbitration (ARLO=1), or by hardware when PE=0.

*Note: Reading I2C_SR3 after reading I2C_SR1 clears the ADDR flag, even if the ADDR flag was set after reading I2C_SR1. Consequently, I2C_SR3 must be read only when ADDR is found set in I2C_SR1 or when the STOPF bit is cleared.*

### 14.7.10 Interrupt register (I2C_ITR)

Address offset: 0x0A
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | Reserved | | | ITBUFEN | ITEVTEN | ITERREN |
| | | r | | | rw | rw | rw |

Bits 7:3 Reserved

Bit 2 **ITBUFEN**: Buffer interrupt enable

0: TXE = 1 or RXNE = 1 does not generate any interrupt.
1:TXE = 1 or RXNE = 1 generates Event interrupt.

Bit 1 **ITEVTEN**: Event interrupt enable

0: Event interrupt disabled
1: Event interrupt enabled

This interrupt is generated when:

– 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 1if ITBUFEN = 1
– WUFH = 1 (asynchronous interrupt to wakeup from Halt)

Bit 0 **ITERREN**: Error interrupt enable

0: Error interrupt disabled
1: Error interrupt enabled

– This interrupt is generated when:
– BERR = 1
– ARLO = 1
– AF = 1
– OVR = 1

### 14.7.11 Clock control register low (I2C_CCRL)

Address offset: 0x02
Reset value: 0x0B

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | CCR[7:0] | | | | |
| | | | rw | | | | |

Bits 7:0   **CCR[7:0]** Clock control register (Master mode)

Controls the SCLH clock in Master mode.

– Standard mode:

Period(I2C) = $2 * CCR * t_{MASTER}$

$t_{high}$ = $CCR * t_{MASTER}$

$t_{low}$ = $CCR * t_{MASTER}$

– Fast mode:

If DUTY = 0:

Period(I2C) = $3* CCR * t_{MASTER}$

$t_{high}$ = $CCR * t_{MASTER}$

$t_{low}$ = $2 * CCR * t_{MASTER}$

If DUTY = 1: (to reach 400 kHz)

Period(I2C) = $25 * CCR * t_{MASTER}$

$t_{high}$ = $9 * CCR * t_{MASTER}$

$t_{low}$ = $16 * CCR * t_{MASTER}$

Note:   $t_{CK}$ = $1/ f_{MASTER}$. $f_{MASTER}$ is the input clock to the peripheral configured using clock control register.

The minimum allowed value is 04h, except in FAST DUTY mode where the minimum allowed value is 0x01.

$t_{high}$ = $t_{r(SCL)} + t_{w(SCLH)}$. See device datasheet for the definitions of parameters.

$t_{low}$ = $t_{f(SCL)} + t_{w(SCLL)}$. See device datasheet for the definitions of parameters.

I2C communication speed, $f_{SCL}$ = $1/(t_{high} + t_{low})$

The real frequency may differ due to the analog noise filter input delay.

### 14.7.12 Clock control register high (I2C_CCRH)

Address offset: 0x0C
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| F/S | DUTY | Reserved | | CCR[11:8] | | | |
| rw | rw | r | | rw | | | |

Bit 7 **F/S**: I2C master mode selection
    0: Standard mode I2C
    1: Fast mode I2C

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)

Bits 5:4 Reserved

Bits 3:0 **CCR[11:8]**: Clock control register in Fast/Standard mode (Master mode)[1]
    Controls the SCLH clock in master mode.
    – Standard mode:
    Period(I2C) = 2 * CCR * $t_{MASTER}$
    $t_{high}$ = CCR * $t_{MASTER}$
    $t_{low}$ = CCR * $t_{MASTER}$
    – Fast mode:
    If DUTY = 0:
    Period(I2C) = 3 * CCR * $t_{MASTER}$
    $t_{high}$ = CCR * $t_{MASTER}$
    $t_{low}$ = 2 * CCR * $t_{MASTER}$
    If DUTY = 1: (to reach 400 kHz)
    Period(I2C) = 25 * CCR * $t_{MASTER}$
    $t_{high}$ = 9 * CCR * $t_{MASTER}$
    $t_{low}$ = 16 * CCR * $t_{MASTER}$
    For instance: in standard mode, to generate a 100 kHz SCL frequency:
    If FREQR = 08, $t_{MASTER}$ = 125 ns so CCR must be programmed with 0x28
    (0x28 <=> 40 x 125 ns = 5000 ns.)
    *Note:  $t_{high}$ = $t_{r(SCL)}$ + $t_{w(SCLH)}$. See device datasheet for the definitions of parameters*
          *$t_{low}$ = $t_{f(SCL)}$ + $t_{w(SCLL)}$. See device datasheet for the definitions of parameters*
          *The real frequency may differ due to the analog noise filter input delay.*

1. Refer to *Table 28: I2C_CCR values for SCL frequency table (fMASTER = 10 MHz or 16 MHz) on page 125*

*Note:     The CCR registers must be configured only when the I²C is disabled (PE=0).*

        *$f_{MASTER}$ = multiple of 10 MHz is required to generate Fast clock at 400 kHz. This cannot be reached with STM8L001xx and STM8L101xx.*

        *$f_{MASTER}$ ≥ 1 MHz is required to generate Standard clock at 100 kHz.*

**Table 28. I2C_CCR values for SCL frequency table (fMASTER = 10 MHz or 16 MHz[1])**

| I2C Speed | I2C frequency (f$_{SCL}$) | fMASTER = 10 MHz | | | | fMASTER = 16 MHz | | | |
|---|---|---|---|---|---|---|---|---|---|
| | in Hz | Actual (Hz) | % Error (%) | I2C_CCR (h) | Duty cycle bit | Actual (Hz) | % Error (%) | I2C_CCR (h) | Duty cycle bit |
| Fast speed | 400000 | 400000 | 0 | 1 | 1 | 410256.41 | 2.56 | D | 0 |
| | 370000 | 370370.37 | 0.10 | 9 | 0 | 380952.38 | 2.96 | E | 0 |
| | 350000 | 370370.37 | 5.82 | 9 | 0 | 355555.56 | 1.59 | F | 0 |
| | 320000 | 333333.33 | 4.17 | A | 0 | 320000 | 0 | 2 | 1 |
| | 300000 | 303030.30 | 1.01 | B | 0 | 313725.49 | 4.57 | 11 | 0 |
| | 270000 | 277777.78 | 2.88 | C | 0 | 280701.75 | 3.96 | 13 | 0 |
| | 250000 | 256410.26 | 2.56 | D | 0 | 253968.25 | 1.59 | 15 | 0 |
| | 220000 | 222222.22 | 1.01 | F | 0 | 222222.22 | 1.01 | 18 | 0 |
| | 200000 | 200000 | 0 | 2 | 1 | 205128.20 | 2.56 | 1A | 0 |
| | 170000 | 175438.60 | 3.20 | 13 | 0 | 172043.01 | 1.20 | 1F | 0 |
| | 150000 | 151515.15 | 1.01 | 16 | 0 | 152380.95 | 1.59 | 23 | 0 |
| | 120000 | 123456.79 | 2.88 | 1B | 0 | 121212.12 | 1.01 | 2C | 0 |
| Standard speed | 100000 | 100000 | 0 | 32 | No impact | 100000 | 0 | 50 | No impact |
| | 50000 | 50000 | 0 | 64 | | 50000 | 0 | A0 | |
| | 30000 | 30120.48 | 0.40 | A6 | | 30075.19 | 0.25 | 10A | |
| | 20000 | 20000 | 0 | FA | | 20000 | 0 | 190 | |

1. The following table gives the values to be written in the I2C_CCR register to obtain the required I²C SCL line frequency

### 14.7.13 TRISE register (I2C_TRISER)

Address offset: 0x0D

Reset value: 0x02

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | TRISE[5:0] | | | | | |
| r | | rw | | | | | |

Bits 7:6  Reserved

Bits 5:0  **TRISE[5:0]** Maximum rise time in Fast/Standard mode (Master mode)

These bits should provide the maximum duration of the SCL feedback loop in master mode. The purpose is to keep a stable SCL frequency whatever the SCL rising edge duration.

These bits must be programmed with the maximum SCL rise time given in the I2C bus specification, incremented by 1.

For instance: in standard mode, the maximum allowed SCL rise time is 1000 ns.

If the value in the I2C_FREQR register = 08h, then $t_{MASTER}$ = 125 ns therefore the TRISE[5:0] bits must be programmed with 0x09.

(1000 ns / 125 ns = 8 + 1)

The filter value can also be added to TRISE[5:0].

If the result is not an integer, TRISE[5:0] must be programmed with the integer part, in order to respect the $t_{HIGH}$ parameter.

*Note:   TRISE[5:0] must be configured only when the I2C is disabled (PE = 0).*

## 14.7.14    I2C register map and reset values

**Table 29. I2C register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | I2C_CR1 Reset value | NO STRETCH 0 | ENGC 0 | - 0 | - 0 | - 0 | - 0 | - 0 | PE 0 |
| 0x01 | I2C_CR2 Reset value | SWRST 0 | - 0 | - 0 | - 0 | POS 0 | ACK 0 | STOP 0 | START 0 |
| 0x02 | I2C_FREQR Reset value | - 0 | - 0 | FREQ[5:0] 000000 | | | | | |
| 0x03 | I2C_OARL Reset value | ADD[7:1] 0000000 | | | | | | | ADD0 0 |
| 0x04 | I2C_OARH Reset value | ADDMODE 0 | ADDCONF 0 | - 0 | - 0 | - 0 | ADD[9:8] 00 | | - 0 |
| 0x05 | | Reserved | | | | | | | |
| 0x06 | I2C_DR Reset value | DR[7:0] 0 | | | | | | | |
| 0x07 | I2C_SR1 Reset value | TXE 0 | RXNE 0 | - 0 | STOPF 0 | ADD10 0 | BTF 0 | ADDR 0 | SB 0 |
| 0x08 | I2C_SR2 Reset value | - 0 | - 0 | WUFH 0 | - 0 | OVR 0 | AF 0 | ARLO 0 | BERR 0 |
| 0x09 | I2C_SR3 Reset value | - 0 | - 0 | - 0 | GENCALL 0 | - 0 | TRA 0 | BUSY 0 | MSL 0 |
| 0x0A | I2C_ITR Reset value | - 0 | - 0 | - 0 | -- 0 | -- 0 | ITBUFEN 0 | ITEVTEN 0 | ITERREN 0 |
| 0x0B | I2C_CCRL Reset value | CCR[7:0] 00000000 | | | | | | | |
| 0x0C | I2C_CCRH Reset value | FS 0 | DUTY 0 | - 0 | - 0 | CCR[11:8] 0000 | | | |
| 0x0D | I2C_TRISER Reset value | - 0 | - 0 | TRISE[5:0] 000010 | | | | | |

# 15 Infrared (IRTIM) interface

## 15.1 IRTIM introduction

An infrared interface (IRTIM) can be used with an IR LED to perform remote control functions.

To generate the infrared remote control signals, the IR interface must be enabled and TIM2 channel 1 (TIM2_OC1) and TIM3 channel 1 (TIM3_OC1) must be properly configured to generate correct waveforms.

**Figure 27. IR internal hardware connections with TIM2 and TIM3**
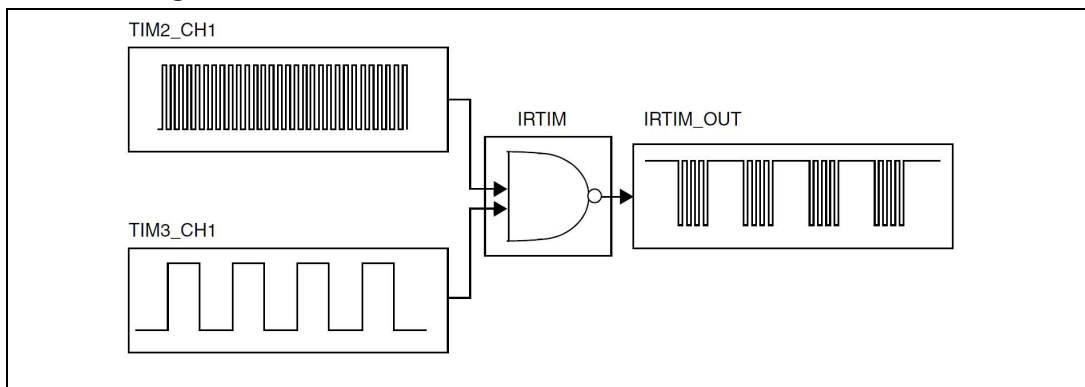


## 15.2 Main features

All standard IR pulse modulation modes can be obtained by programming the two timer output compare channels.
TIM 2 is used to generate the high frequency carrier signal, while TIM3 generates the modulation envelope.

The infrared function is output on the TIM_IR pin. The activation of this function is done through the IR_CR register. When the IR function is enabled by setting the IR_EN bit, the standard TIM2_CC1 and TIM3_CC1 become automatically inactive (these pins may be used as general purpose I/O pins or for other alternate functions).

The high sink LED driver capability (only available on the TIM_IR pin) can be activated through the HS_EN bit in the IR_CR register and used to sink the high current needed to directly control an infrared LED. When the pin is driving the LED in this mode, the other pin input/output levels cannot be guaranteed. It is therefore recommended to program all other device I/Os in input mode without interrupt before sending any infrared signal. The previous function can be restored immediately after the infrared communication is completed. When the high sink capability of the pin is not used (or the current is limited to the standard I/O capabilities) all other pins of the device can be used normally.

## 15.3 IRTIM register

### 15.3.1 Control register (IR_CR)

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | HS_EN | IR_EN |
| rw | rw | rw | rw | rw | rw | rw | rw |

Bits 7:2 Reserved. Must be kept cleared

Bit 1 **HS_EN**: High Sink LED driver capability enable.

0: High Sink LED driver capability disabled.
1: High Sink LED driver capability enabled.

When activated, this pin can sink 20 mA min. with a power supply down to 2 V. Refer to "Output driving current" table in the datasheet.

Bit 0 **IR_EN**: Infrared output enable.

This bit enables the IR output.

0: IR_TIM output disabled.
1: IR_TIM output enabled and provided to PA0 (TIM2 and TIM3 must have been previously configured properly by software)

### 15.3.2 IRTIM register map and reset values

**Table 30. IR register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | IR_CR Reset value | -<br>0 | -<br>0 | -<br>0 | -<br>0 | -<br>0 | -<br>0 | HS_EN<br>0 | IR_EN<br>0 |

.

# 16 Timer overview

The microcontroller has two types of TIM timers, two general purpose (TIM2/ TIM3), and one basic timer (TIM4). They have different features but are based on a common architecture. This makes it easier to design applications using the various timers (identical register mapping, common basic features).

Although the timers do not share any resources, they can be linked together and synchronized.

This section gives a comparison of the different timer features and glossary of internal timer signal names.

*Section 17: 16-bit general purpose timer (TIM2/TIM3)* contains a full description of all the various timer modes.

**Table 31. Timer characteristics**

| Symbol | Parameter | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|
| $t_{w(ICAP)in}$ | Input capture pulse time | 2 | - | - | $t_{MASTER}$ |
| $t_{res(TIM)}$ | Timer resolution time | 1 | - | - | $t_{MASTER}$ |
| $Res_{TIM}$ | Timer resolution with 16-bit counter | - | 16 | - | bit |
| | Timer resolution with 8-bit counter | - | 8 | - | bit |
| $t_{COUNTER}$ | Counter clock period when internal clock is selected | - | 1 | - | $t_{MASTER}$ |
| $t_{MAX\_COUNT}$ | Maximum possible count with 16-bit counter | - | - | 65,536 | $t_{MASTER}$ |
| | Maximum possible count with 8-bit counter | - | - | 256 | $t_{MASTER}$ |

## 16.1 Timer feature comparison

**Table 32. Timer feature comparison**

| Timer | Counter resol- ution | Counter type | Prescaler factor | Capture/ compare chan- nels | Comple- mentary outputs | Repet- ition counter | External trigger input | External break input | Timer synchr- onization/ chaining |
|---|---|---|---|---|---|---|---|---|---|
| TIM2 & TIM3 (general purpose timers) | 16-bit | Up/down | Any power of 2 from 1 to 128 | 2 | None | No | 1 | 1 | Yes |
| TIM4 (basic timer) | 8-bit | Up | Any power of 2 from 1 to 32768 | 0 | | | 0 | 0 | |

## 16.2 Glossary of timer signal names

**Table 33. Glossary of internal timer signals**

| Internal signal name | Description | Related figures |
|---|---|---|
| BI | Break interrupt | *Figure 28: TIMx general block diagram on page 134* |
| CC*i*I, CC1I, CC2I, CC3I, CC4I | Capture/compare interrupt | |
| CK_CNT | Counter clock | *Figure 32: Counter update when ARPE=0 (ARR not preloaded) with prescaler = 2 on page 138* |
| CK_PSC | Prescaler clock | |
| CNT_EN | Counter enable | |
| CNT_INIT | Counter initialize | *Figure 41: TI2 external clock connection example on page 144* |
| ETR | External trigger from TIMx_ETR pin | *Figure 43: External trigger input block on page 145* |
| ETRF | External trigger filtered | |
| ETRP | External trigger prescaled | |
| $f_{MASTER}$ | Timer peripheral clock from clock controller (CLK) | *Figure 12: Clock structure on page 61* |
| IC*i*, IC1, IC2 | Input capture | *Figure 60: Input stage of TIM 1 channel 1 on page 158* |
| IC*i*PS, IC1PS, IC2PS | Input capture prescaled | |
| ITRx, ITR1, ITR2, ITR3 | Internal trigger input tied to TRGO of other TIM timers | *Figure 28: TIMx general block diagram on page 134* |
| MATCH1 | Compare match | *Figure 50: Trigger/master mode selection blocks on page 150* and *Section 17.7.2: Control register 2 (TIMx_CR2) on page 173* |
| OC*i*, OC1, OC2 | Timer output channel | *Figure 17.5.5: Forced output mode on page 161* |
| OC*i*REF, OC1REF, OC2REF | Output compare reference signal | |
| TGI | Trigger interrupt | *Figure 39: Clock/trigger controller block diagram on page 143* |
| TI*i*, TI1, TI2 | Timer input | *Figure 60: Input stage of TIM 1 channel 1 on page 158* |
| TI*i*F, TI1F, TI2F | Timer input filtered | |
| TI1F_ED | Timer input filtered edge detector | |
| TI*i*FPx, TI1FP1, TI1FP2, TI2FP1, TI2FP2 | Timer input filtered prescaled | |
| TRC | Trigger capture | |
| TRGI | Trigger input to clock/trigger/slave mode controller | *Figure 40: Control circuit in normal mode, fMASTER divided by 1 on page 144* |
| TRGO | Trigger output tied to trigger input ITRx of other timers | *Figure 28: TIMx general block diagram on page 134* |

**Table 33. Glossary of internal timer signals  (continued)**

| Internal signal name | Description | Related figures |
|---|---|---|
| UEV | Update event | *Figure 32: Counter update when ARPE=0 (ARR not preloaded) with prescaler = 2 on page 138* |
| UIF | Update interrupt | |

# 17      16-bit general purpose timer (TIM2/TIM3)

## 17.1      Introduction

This chapter describes TIM2 and TIM3 which are identical timers and referred to as TIMx.

Each general purpose timer (TIMx) has a 16-bit up-down auto-reload counter driven by a programmable prescaler.

In this section, the index *i*, may be 1 or 2 referring to the two capture/compare channels.

The timers may be used for a variety of purposes, including:

*   Time base generation
*   Measuring the pulse lengths of input signals (input capture)
*   Generating output waveforms (output compare, PWM and One Pulse Mode)
*   Interrupt capability on various events (capture, compare, overflow, break, trigger)
*   Synchronization with other timers or external signals (external clock, reset, trigger and enable)

These timers are ideally suited for a wide range of control applications, including those requiring center-aligned PWM capability.

The timer clock can be sourced from internal clocks or from an external source selectable through a configuration register.

## 17.2      TIMx main features

TIMx features include:

*   16-bit up, down, up/down counter auto-reload counter.
*   3-bit programmable prescaler allowing the counter clock frequency to be divided "on the fly" by any power of 2 between 1 and 128.
*   Synchronization circuit to control the timer with external signals and to interconnect several timers.
*   2 independent channels that can alternately be configured as:
    –   Input capture
    –   Output compare
    –   PWM generation (edge and center-aligned mode)
    –   One Pulse Mode output
*   Break input to put the timer output signals in reset state or in a known state.
*   Interrupt generation on the following events:
    –   Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
    –   Trigger event (counter start, stop, initialization or count by internal/external trigger)
    –   Input capture
    –   Output compare
    –   Break input

**Figure 28. TIMx general block diagram**
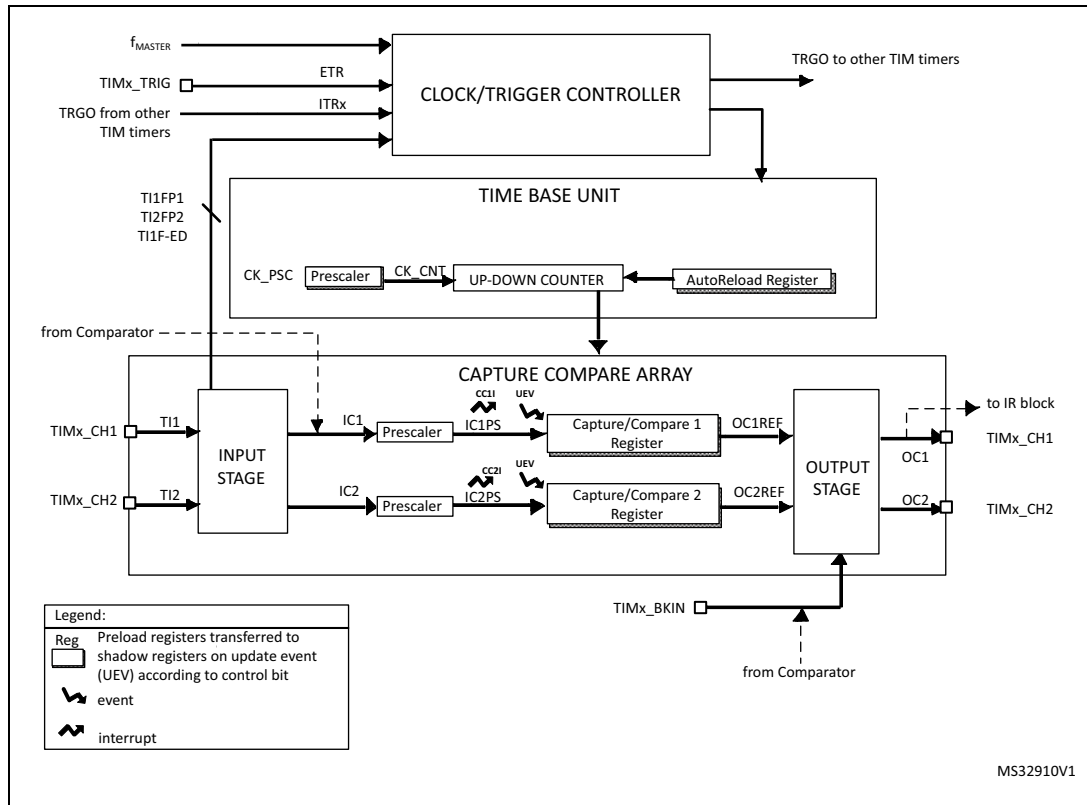
## 17.3    TIMx time base unit

The timer has a *Time base unit* that includes:

*   16-bit up/down counter
*   16-bit auto-reload register
*   Prescaler

**Figure 29. Time base unit**
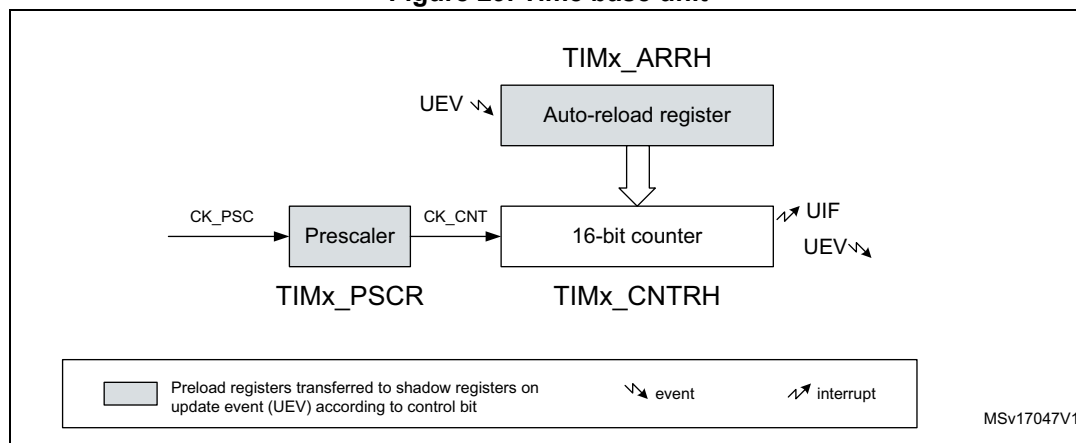


The 16-bit counter, the prescaler and the auto-reload register can be written or read by software.

The auto-reload register is composed of a preload register plus a shadow register.

Writing to the auto-reload register can be done in two modes:

*   **Auto-reload preload enabled** (ARPE bit set in the TIMx_CR1 register). In this mode, when data is written to the autoreload register, it is kept in the preload register and transferred into the shadow register at the next update event (UEV).

*   **Auto-reload preload disabled** (ARPE bit cleared in the TIMx_CR1 register). In this mode, when data is written to the autoreload register it is transferred into the shadow register immediately.

An update event is generated:

*   On a counter overflow or underflow.
*   By software, setting the UG bit in the TIMx_EGR register.
*   By an trigger event from the clock/trigger controller.

With preload enabled (ARPE=1), when an update event occurs: the auto-reload shadow register is updated with the preload value (TIMx_ARR) and the buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSCR register).

The update event (UEV) can be disabled by setting the UDIS bit in the TIMx_CR1

The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in TIMx_CR1 register is set.

*Note:*      *Note that the actual counter enable signal CNT_EN is set 1 clock cycle after CEN.*

### 17.3.1 Reading and writing to the 16-bit counter

There is no buffering when writing the counter. Both TIMx_CNTRH and TIMx_CNTRL can be written at any time, so it is suggested not to write a new value into the counter while it is running to avoid loading a wrong intermediate content.

An 8-bit buffer is implemented for the read. The user must read the MS byte first, then the LS byte value is buffered automatically, as described in *Figure 30*. 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, because it reads the LS byte first, and would return a wrong result.*

**Figure 30. 16-bit read sequence for the counter (TIMx_CNTR)**
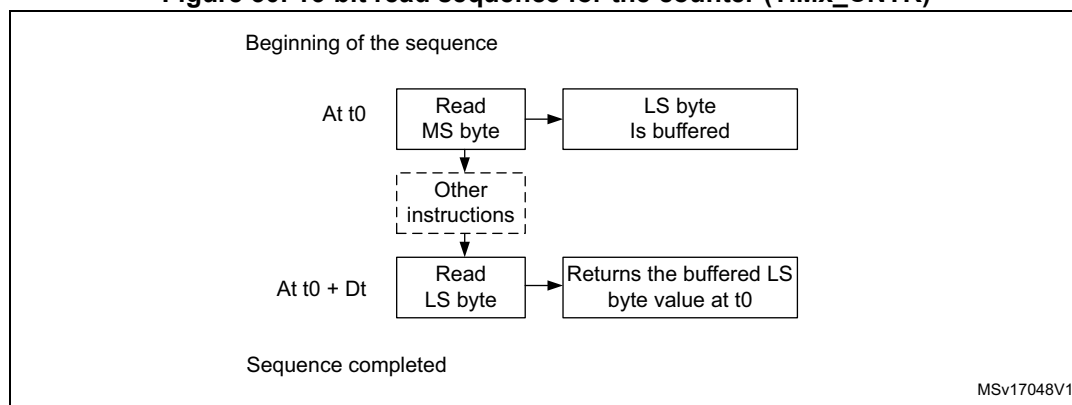


### 17.3.2 Write sequence for 16-bit TIMx_ARR register

16-bit values are loaded in the TIMx_ARR register through preload registers. This must be performed by two write instructions, one for each byte. The MS byte must be written first.

The shadow register update is blocked as soon as the MS byte has been written, and stays blocked until the LS byte has been written. Do not use the LDW instruction, as this writes the LS byte first, and would produce wrong results in this case.

### 17.3.3 Prescaler

The prescaler is based on a 7-bit counter controlled through a 3-bit register (in TIMx_PSCR register). It can be changed on the fly as this control register is buffered. It can divide the counter clock frequency by 1, 2, 4, 8, 16, 32, 64 or 128.

The counter clock frequency is calculated as follows:

$$f_{CK\_CNT} = f_{CK\_PSC}/2^{(PSCR[2:0])}$$

The new prescaler value is taken into account in the following period (after the next counter update event).

## 17.3.4     Up-counting mode

In up-counting mode, the counter counts from 0 to a user-defined compare value (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event, and an update event (UEV) if the UDIS bit is 0 in the TIMx_CR1 register.

*Figure 31* shows an example of this counting mode.

**Figure 31. Counter in up-counting mode**
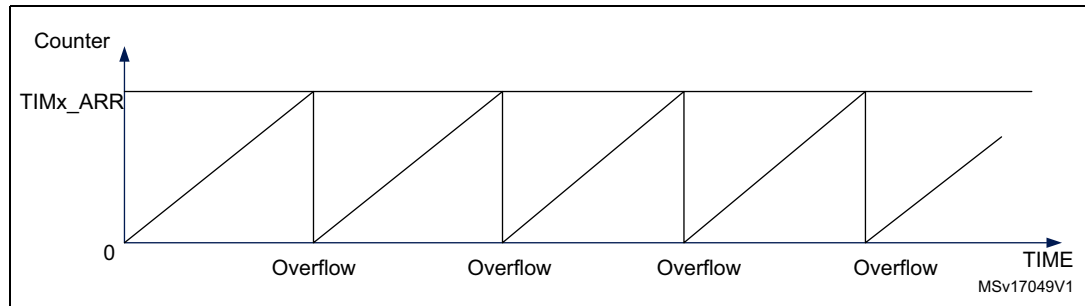


An update event can also be generated by setting the bit UG in the TIMx_EGR register (by software or by using the trigger controller).

The UEV event can be disabled by software by setting the UDIS bit in the TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event will occur until UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescaler division factor does not change). In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR1 register) is set (depending on the URS bit):

The auto-reload shadow register is updated with the preload value (TIMx_ARR),

The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSCR register).

The following figures show two examples of the counter behavior for different clock frequencies when TIMx_ARR=36h.

In *Figure 32* the prescaler divider is set to 2, so the counter clock (CK_CNT) frequency is at half the frequency of the the prescaler clock source (CK_PSC).

In *Figure 32* the autoreload preload is disabled (ARPE=0), so the shadow register is changed immediately and counter overflow occurs when upcounting reaches 36h. This generates an update event.

**Figure 32. Counter update when ARPE=0 (ARR not preloaded) with prescaler = 2**
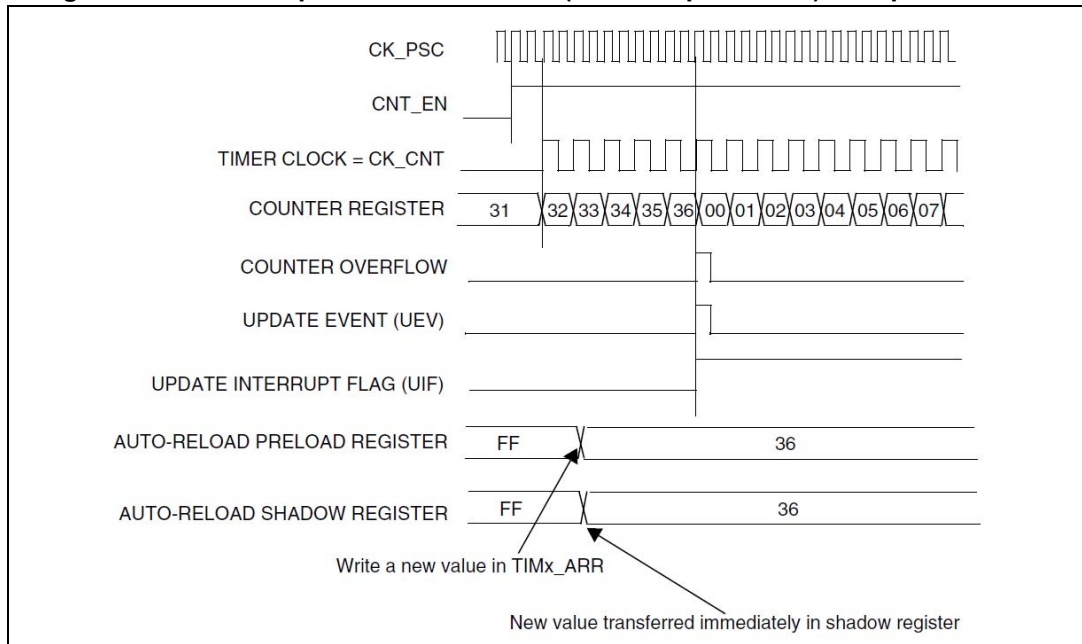


In *Figure 33* the prescaler divider is set to 1, so CK_CNT has the same frequency as CK_PSC.

In *Figure 33* autoreload preload is enabled (ARPE=1), so the next counter overflow occurs at FFh. The new autoreload value register value of 36h is taken into account after the overflow which generates an update event.

**Figure 33. Counter update event when ARPE=1 (TIMx_ARR preloaded)**
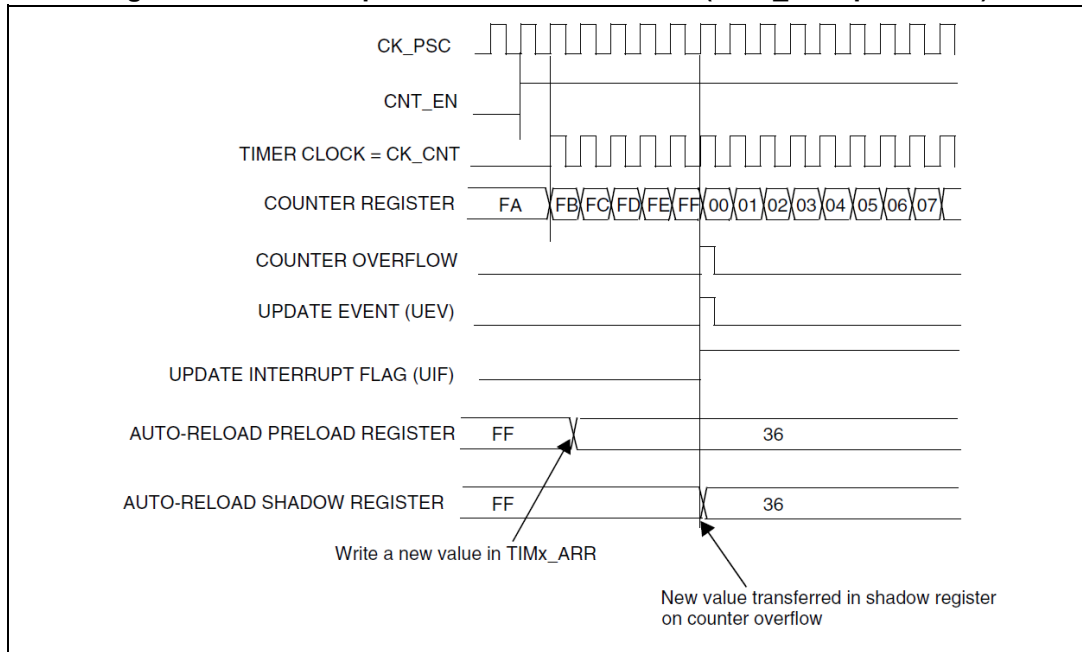
## 17.3.5 Down-counting mode

In down-counting mode, the counter counts from the auto-reload value (content of the TIMx_ARR register) down to 0, then restarts from the auto-reload value and generates a counter underflow and an update event (UEV) if the UDIS bit is 0 in the TIMx_CR1 register.

*Figure 34* shows an example of this counting mode.

**Figure 34. Counter in down-counting mode**
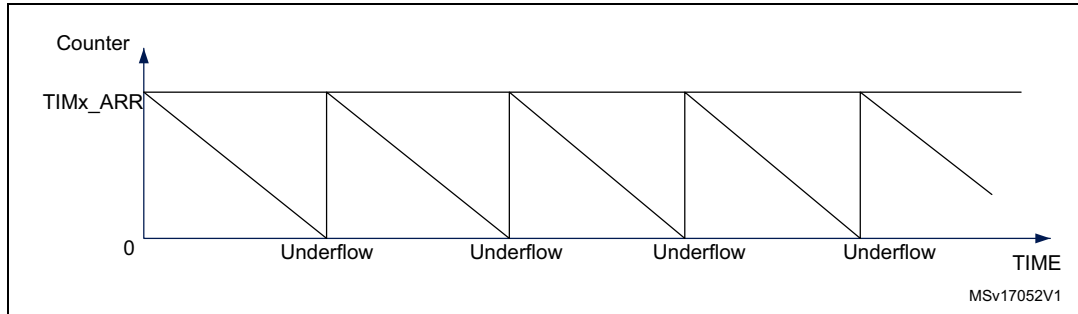


An update event can also be generated by setting the bit UG in the TIMx_EGR register (by software or by using the clock/trigger mode controller).

The UEV update event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event will occur until UDIS bit has been written to 0. However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR1 register) is set (depending on the URS bit):

The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSCR register),

The auto-reload shadow register is updated with the preload value (content of the TIMx_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one.

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=36h.

In downcounting mode, preload is normally not used so that the new value is taken into account in the next period (see *Figure 35*).

**Figure 35. Counter update when ARPE=0 (ARR not preloaded) with prescaler = 2**
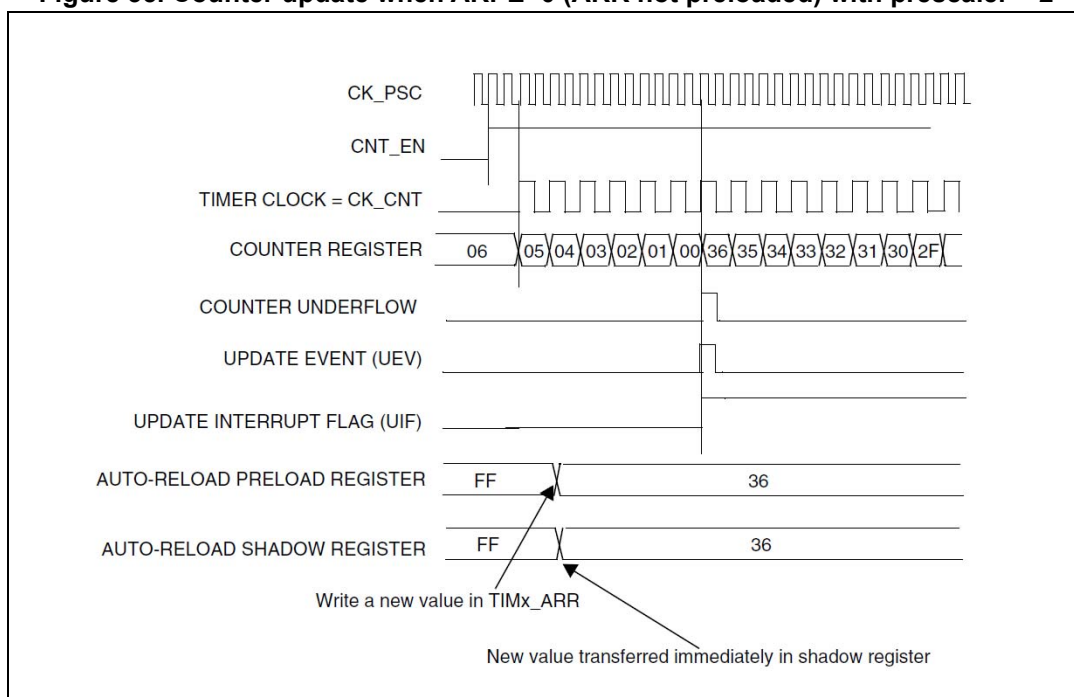


**Figure 36. Counter update when ARPE=1 (ARR preloaded), with prescaler = 1**
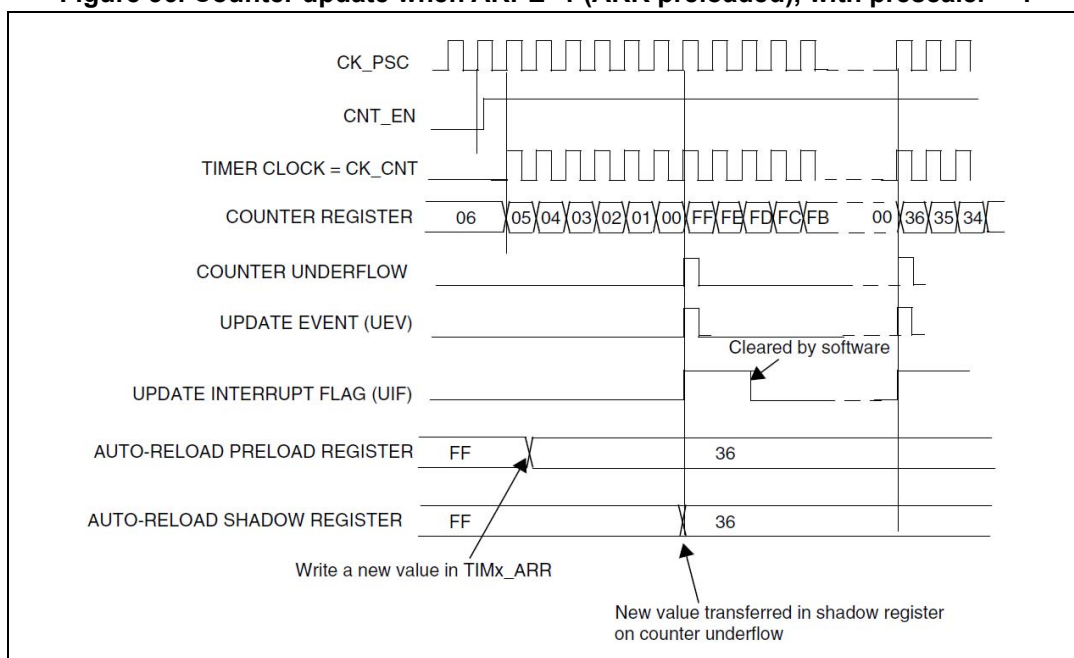
## 17.3.6 Center-aligned mode (up/down counting)

In center-aligned mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register) -1, generates a counter overflow event, then counts down to 0 and generates a counter underflow event. Then it restarts counting from 0.

In this mode, the DIR direction bit in the TIMx_CR1 register cannot be written. It is updated by hardware and gives the current direction of the counter.

The *Figure 37* shows an example of this counting mode.

**Figure 37. Counter in center-aligned mode**
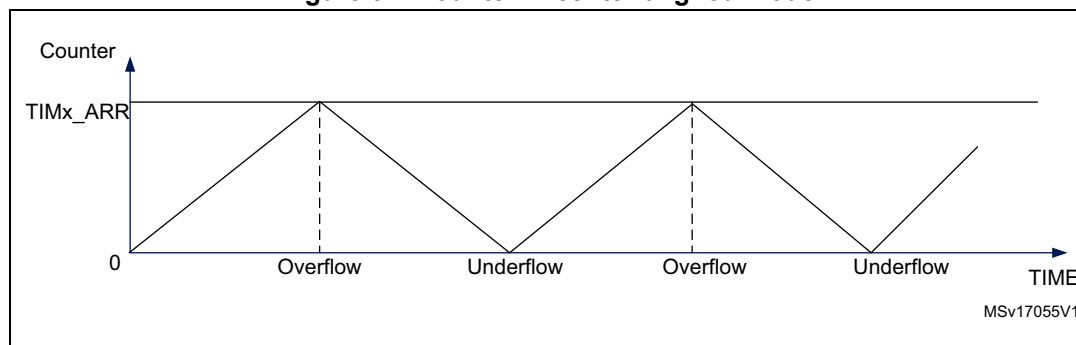


The update event is generated at each counter overflow and at each counter underflow.

Setting the bit UG in the TIMx_EGR register (by software or by using the clock/trigger mode controller) also generates an update event. In this case, the counter restarts counting from 0, as well as the counter of the prescaler.

The update event (UEV) can be disabled by software setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event will occur until UDIS bit has been written to 0. However, the counter continues counting up and down, based on the current auto-reload value.

If the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt request will be sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR1 register) is set (depending on the URS bit).

The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSCR register).

The auto-reload shadow register is updated with the preload value (content of the TIMx_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter is reloaded, so that the next period is the expected one (the counter is loaded with the new value).

Hereafter are some examples of the counter behavior for different clock frequencies.

**Figure 38. Counter timing diagram, CK_PSC divided by 1, TIMx_ARR=06h, ARPE=1**
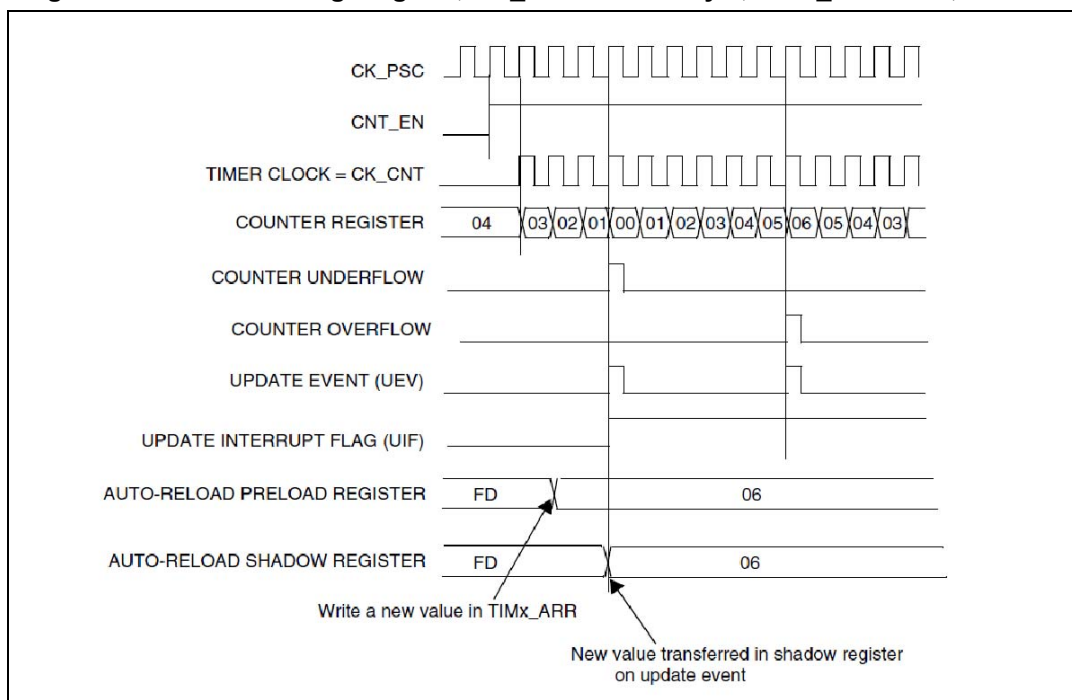


Hints on using center-aligned mode:

- When starting in center-aligned mode, the current up-down configuration is used. It means that the counter will start counting up or down depending on the value written in the DIR bit in the TIMx_CR1 register. Moreover, the DIR and CMS bits must not be changed at the same time by the software.

- Writing to the counter while running in center-aligned mode is not recommended as it can lead to unexpected results. In particular:

   - The direction is not updated if you write a value in the counter that is greater than the auto-reload value (TIMx_CNT>TIMx_ARR). For example, if the counter was counting up, it will continue to count up.

   - The direction is updated if you write 0 or write the TIMx_ARR value in the counter but no Update Event UEV is generated.

- The safest way to use center-aligned mode is to generate an update by software (setting the UG bit in the TIMx_EGR register) just before starting the counter and not to write the counter while it is running.

## 17.4 TIMx clock/trigger controller

The clock/trigger controller allows you to configure the timer clock sources, input triggers and output triggers. The block diagram is shown in *Figure 39.*

**Figure 39. Clock/trigger controller block diagram**
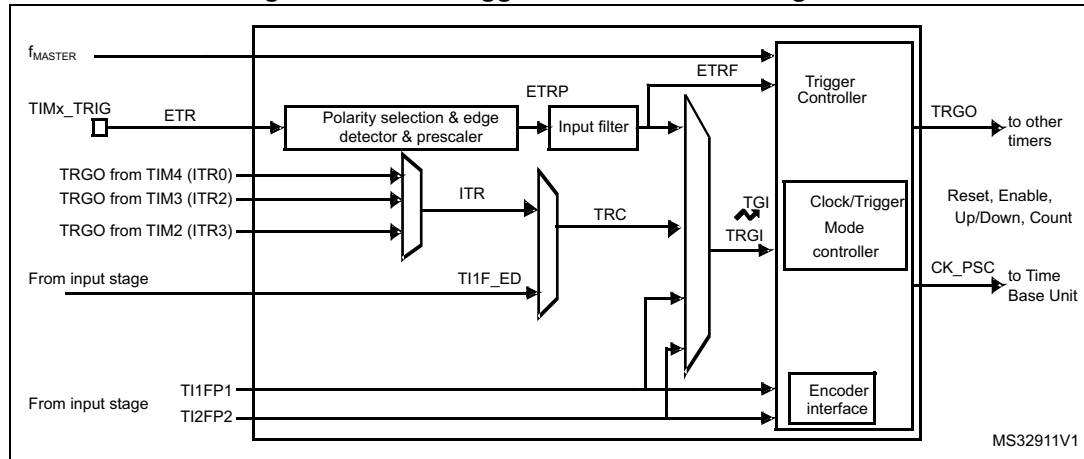


### 17.4.1 Prescaler clock (CK_PSC)

The Time base unit prescaler clock (CK_PSC) can be provided by the following clock sources:

- Internal clock ($f_{MASTER}$)
- External clock mode 1: external timer input (TIx)
- External clock mode 2: external trigger input ETR
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer. Refer to *Using one timer as prescaler for another timer on page 151* for more details.

### 17.4.2 Internal clock source

If both the clock/trigger mode controller and the external trigger input are disabled (SMS=0b000 in TIMx_SMCR and ECE=0 in the TIMx_ETR register), then the CEN, DIR and UG bits are actual control bits and can be changed only by software (except UG which remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock.

*Figure 40* shows the behavior of the control circuit and the up-counter in normal mode, without prescaler.

**Figure 40. Control circuit in normal mode, f$_{MASTER}$ divided by 1**
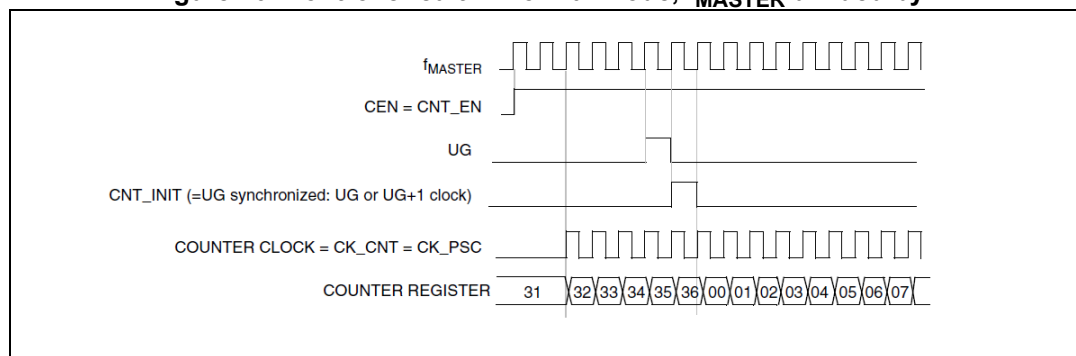


### 17.4.3 External clock source mode 1

The counter can count at each rising or falling edge on a selected timer input. This mode is selected when SMS=0b111 in the TIMx_SMCR register.

**Figure 41. TI2 external clock connection example**
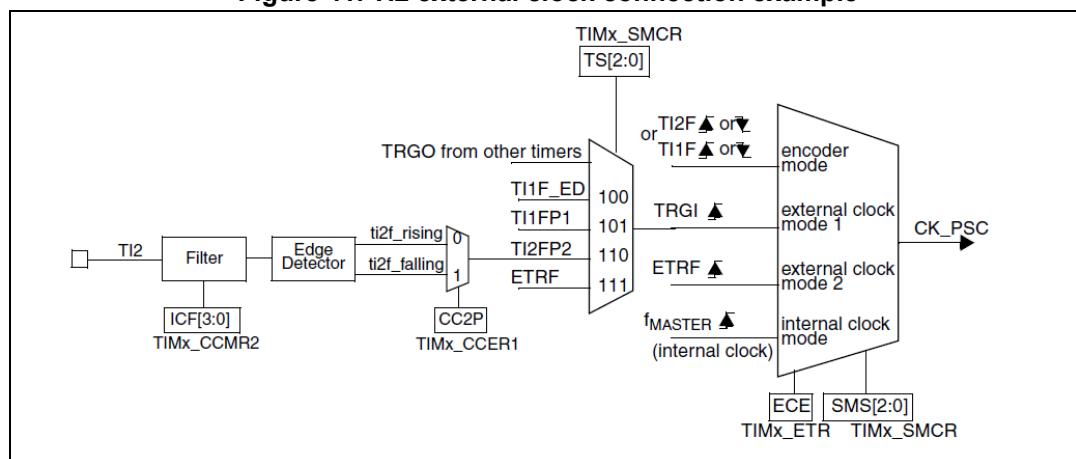


For example, to configure the up-counter to count in response to a rising edge on the TI2 input, use the following procedure:

1.  Configure channel 2 to detect rising edges on the TI2 input by writing CC2S= '01' in the TIMx_CCMR2 register.

2.  Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx_CCMR2 register (if no filter is needed , keep IC2F=0000).

*Note:*     *The capture prescaler is not used for triggering, so you don't need to configure it. Also you don't need to configure the TI2S bits, they only select the input capture source.*

3.  Select rising edge polarity by writing CC2P=0 in the TIMx_CCER1 register.

4.  Configure the timer in external clock mode 1 by writing SMS=0b111 in the TIMx_SMCR register.

5.  Select TI2 as the input source by writing TS=110 in the TIMx_SMCR register.

6.  Enable the counter by writing CEN=1 in the TIMx_CR1 register.

When a rising edge occurs on TI2, the counter counts once and the trigger flag is set (TIF bit in the TIMx_SR1 register) and an interrupt request can be sent if enabled (depending on the TIE in TIMx_IER register).

The delay between the rising edge on TI2 and the actual reset of the counter is due to the resynchronization circuit on TI2 input.

**Figure 42. Control circuit in external clock mode 1**
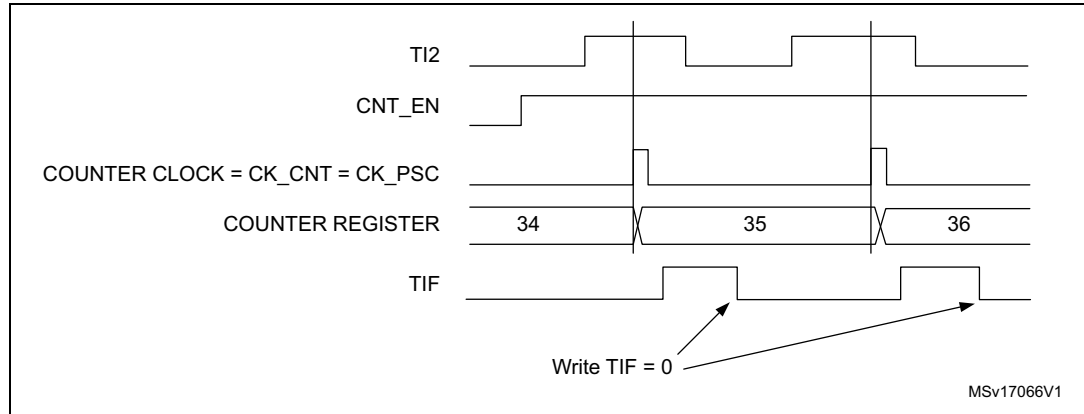


### 17.4.4 External clock source mode 2

The counter can count at each rising or falling edge on the external trigger input ETR. This mode is selected by writing ECE=1 in the TIMx_ETR register.

The *Figure 43* gives an overview of the external trigger input block.

**Figure 43. External trigger input block**
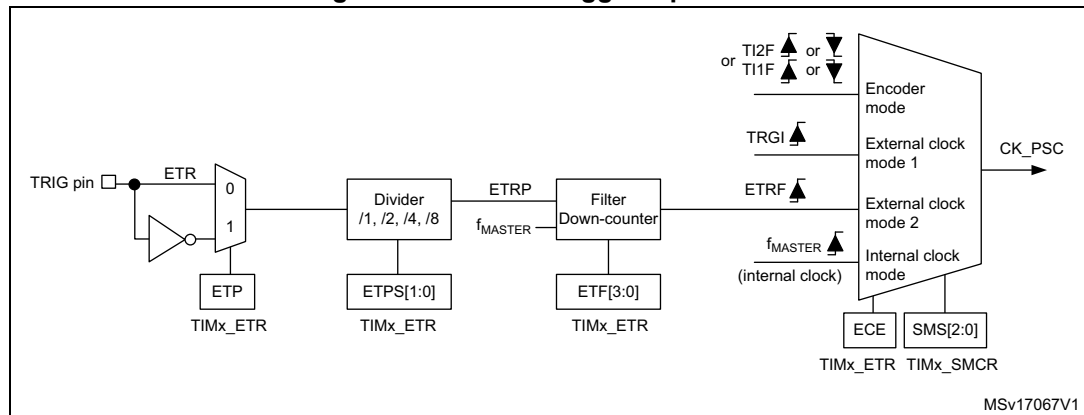


For example, to configure the up-counter to count each 2 rising edges on ETR, use the following procedure:

1. As no filter is needed in this example, write ETF[3:0]=0b0000 in the TIMx_ETR register.
2. Set the prescaler by writing ETPS[1:0]=0b01 in the TIMx_ETR register
3. Select rising edge detection on the TRIG pin by writing ETP=0 in the TIMx_ETR register
4. Enable external clock mode 2 by writing ECE=1 in the TIMx_ETR register.
5. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

The counter counts once each 2 ETR rising edges.

The delay between the rising edge on ETR and the actual reset of the counter is due to the resynchronization circuit on the ETRP signal.

**Figure 44. Control circuit in external clock mode 2**
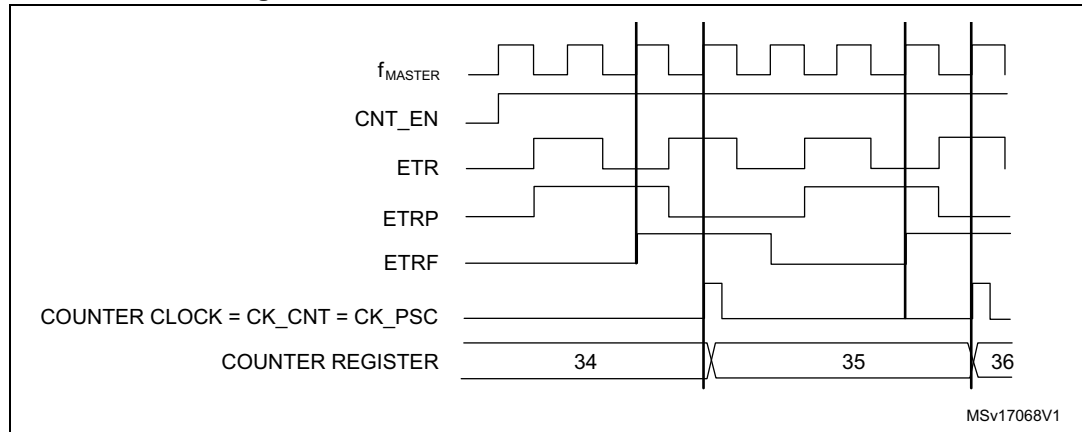


### 17.4.5 Trigger synchronization

There are four trigger inputs (refer to *Table 33: Glossary of internal timer signals on page 131*):

- ETR
- TI1
- TI2
- TRGO from other timers

The  TIMx  timer can be synchronized with an external trigger in three modes: trigger standard mode, trigger reset mode and trigger gated mode.

**Trigger standard mode**

The counter can start in response to an event on a selected input.

In the following example, the up-counter starts in response to a rising edge on TI2 input:

- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we don't need any filter, so we keep IC2F=0b0000). The capture prescaler is not used for triggering, so you don't need to configure it. TI2S bits are selecting the input capture source only, and don't need to be configured too. Write CC2P=0 in TIMx_CCER1 register to select rising edge polarity.
- Configure the timer in trigger mode by writing SMS=0b110 in the TIMx_SMCR register. Select TI2 as the input source by writing TS=0b110 in the TIMx_SMCR register.

When a rising edge occurs on TI2, the counter starts counting on the internal clock and the TIF flag is set.

The delay between the rising edge on TI2 and the actual reset of the counter is due to the resynchronization circuit on TI2 input.

**Figure 45. Control circuit in trigger mode**
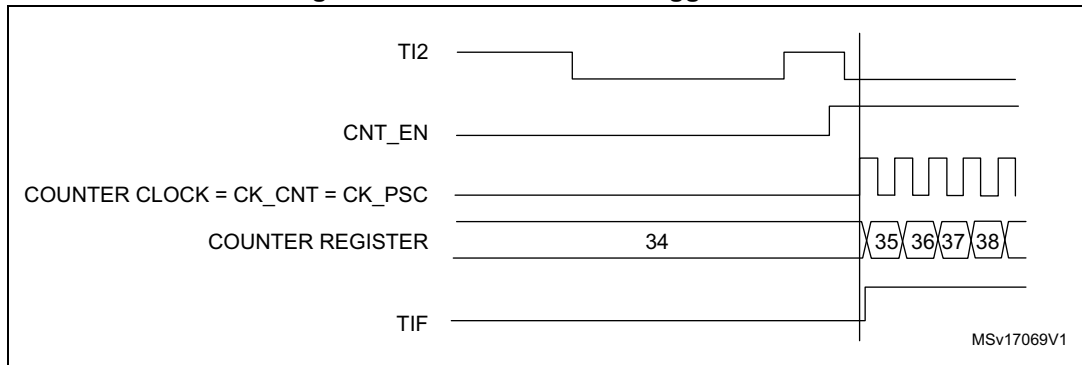


**Trigger reset mode**

The counter and its prescaler can be re-initialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx_ARR, TIMx_CCRx) are updated.

In the following example, the up-counter is cleared in response to a rising edge on TI1 input:

- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0b0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, and do not need to be configured either. Write CC1P=0 in TIMx_CCER1 register to validate the polarity (and detect rising edges only).
- Configure the timer in reset mode by writing SMS=100 in TIMx_SMCR register. Select TI1 as the input source by writing TS=0b101 in TIMx_SMCR register.
- Start the counter by writing CEN=1 in the TIMx_CR1 register.

The counter starts counting on the internal clock, then behaves normally until TI1 rising edge. When TI1 rises, the counter is cleared and restarts from 0. In the meantime, the trigger flag is set (TIF bit in the TIMx_SR1 register) and an interrupt request can be sent if enabled (depending on the TIE in TIMx_IER register).

The following figure shows this behaviour when the auto-reload register TIMx_ARR=36h. The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input.

**Figure 46. Control circuit in trigger reset mode**
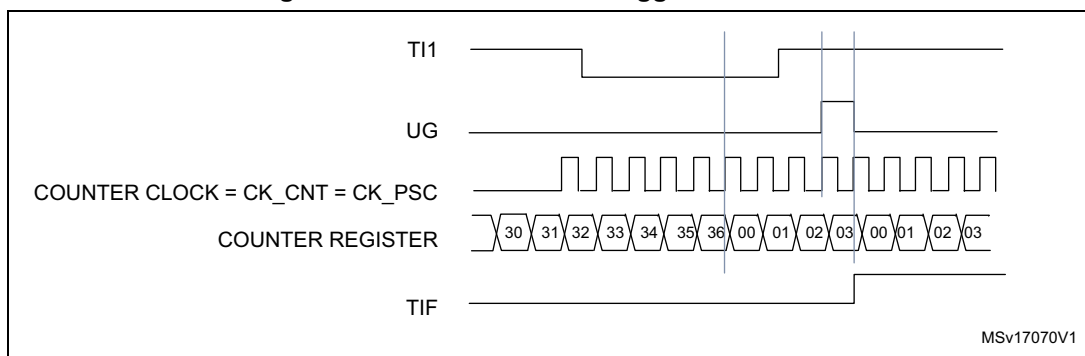
**Trigger gated mode**

The counter can be enabled depending on the level of a selected input.

In the following example, the up-counter counts only when TI1 input is low:

1. Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0b0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, and do not need to be configured either. Write CC1P=1 in TIMx_CCER1 register to validate the polarity (and detect low level only).

2. Configure the timer in trigger gated mode by writing SMS=0b101 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.

3. Enable the counter by writing CEN=1 in the TIMx_CR1 register (in trigger gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

The counter starts counting on the internal clock as long as TI1 is low and stops as soon as TI1 becomes high. The TIF flag is set both when the counter starts or stops.

The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input.

**Figure 47. Control circuit in trigger gated mode**
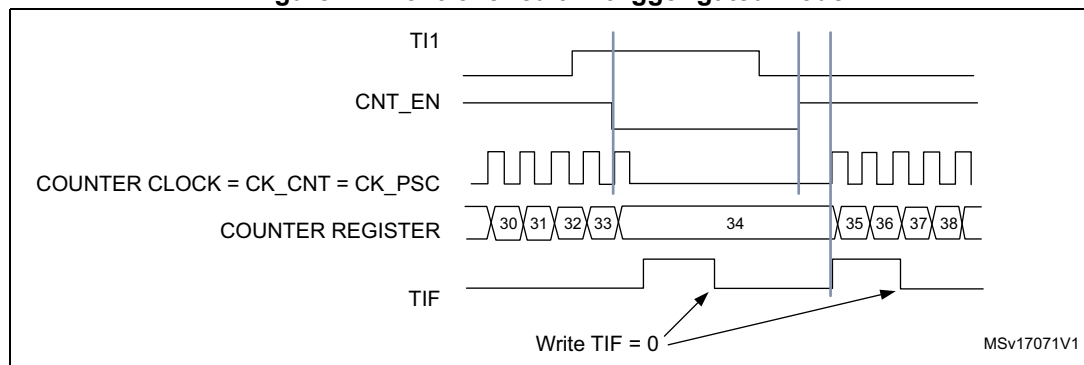


**Combining trigger modes with external clock mode 2**

The external clock mode 2 can be used in addition to another trigger mode. In this case, ETR is used as external clock input, and another input can be selected as trigger input (in trigger standard mode, trigger reset mode or trigger gated mode). Take care that you must not select ETR as TRGI (through the TS bits in TIMx_SMCR register).

In the following example, the up-counter counts at each rising edge on ETR as soon as a rising edge has occured on TI1 (standard trigger mode with external ETR clock):

•  Configure the external trigger input circuit by writing the TIMx_ETR register. In this example, we don't need any filter and write ETF=0b0000. Write ETPS=00 to disable the prescaler, ETP=0 to detect rising edges on ETR and ECE=1 to enable the external clock mode 2.

•  Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0b0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, and do not need to be configured either. Write CC1P=0 in TIMx_CCER1 register to select rising edge polarity.

- • Configure the timer in trigger mode by writing SMS=0b110 in TIMx_SMCR register. Select TI1 as the input source by writing TS=0b101 in TIMx_SMCR register.

A rising edge on TI1 enables the counter and sets the TIF flag. Then the counter counts on ETR rising edges.

The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input. The delay between the rising edge on ETR and the actual reset of the counter is due to the resynchronization circuit on the ETRP signal.

**Figure 48. Control circuit in external clock mode 2 + trigger mode**
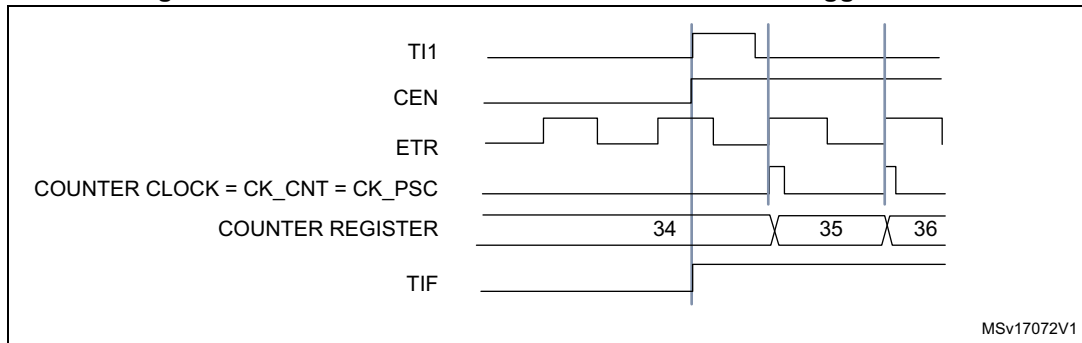
## 17.4.6 Synchronization from other timers

The timers are linked together internally for timer synchronization or chaining. When one timer is configured in master mode, it can output a trigger (TRGO) to reset, start, stop or clock the counter of any other Timer configured in slave mode.

**Figure 49. Timer chaining system implementation example**
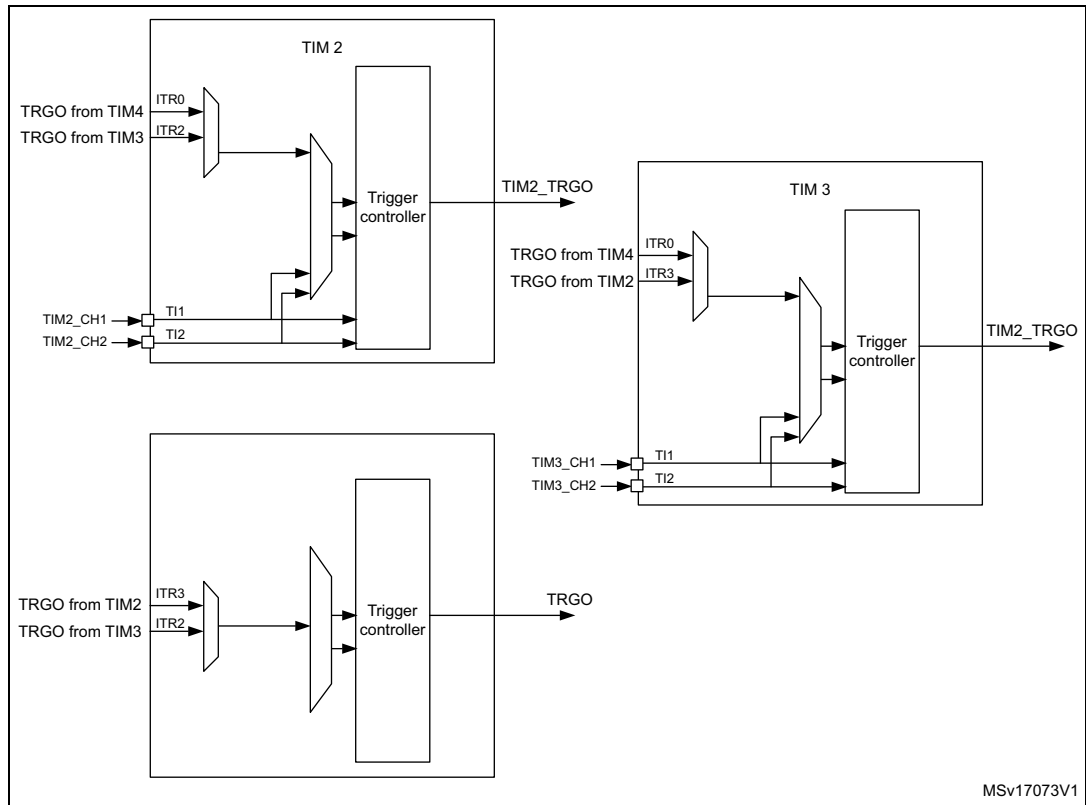


The following figure presents an overview of the trigger selection and the master mode selection blocks.

**Figure 50. Trigger/master mode selection blocks**
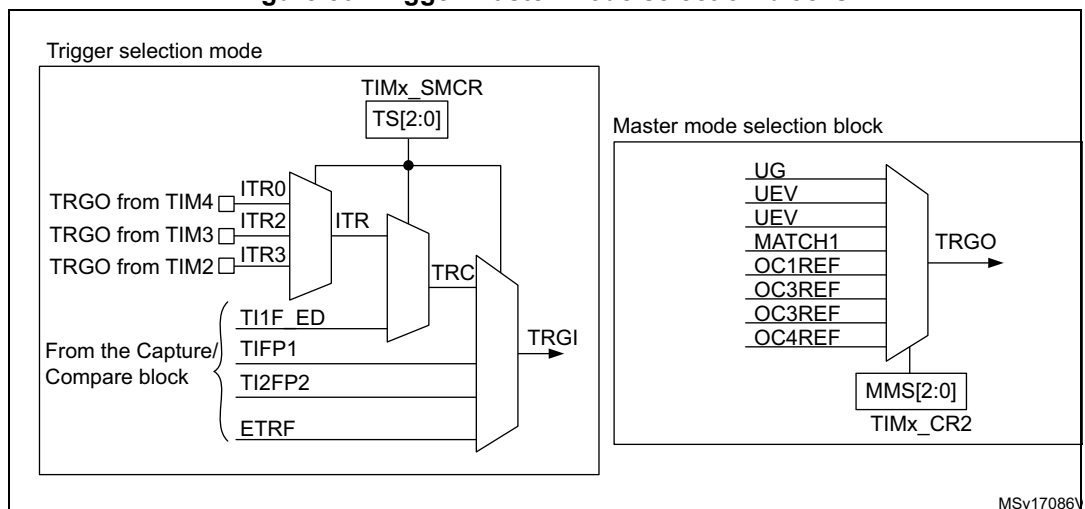
**Using one timer as prescaler for another timer**

**Figure 51. Master/slave timer example**
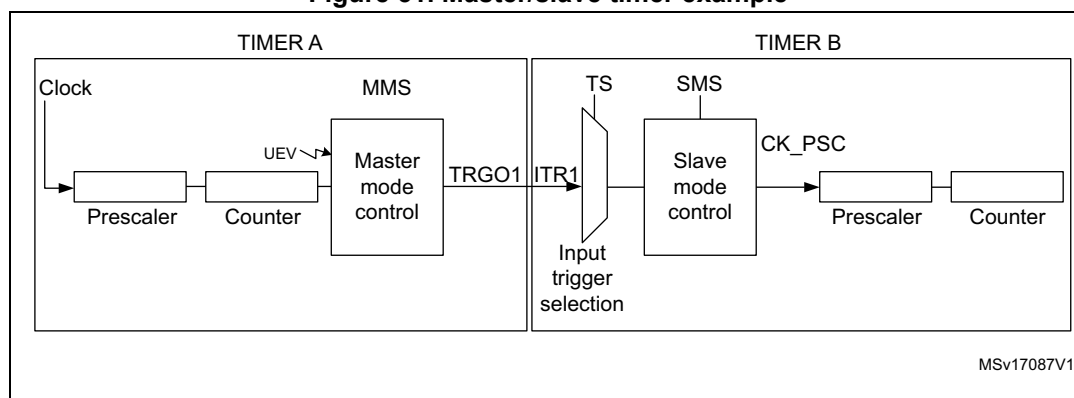


For example, you can configure Timer A to act as a prescaler for Timer B. Refer to *Figure 52*. To do this:

1. Configure Timer A in master mode so that it outputs a periodic trigger signal on each update event UEV. To configure that a rising edge is output on TRGO1 each time an update event is generated, write MMS=010 in the TIMx_CR2 register,.

2. Connect the TRGO1 output of Timer A to Timer B, Timer B must be configured in slave mode using ITR1 as internal trigger. Select this through the TS bits in the TIMx_SMCR register (writing TS=001).

3. Put the clock/trigger controller in external clock mode 1, by writing SMS=111 in the TIMx_SMCR register. This causes Timer B to be clocked by the rising edge of the periodic Timer A trigger signal (which corresponds to the Timer A counter overflow).

4. Finally enable both timers by setting their respective CEN bits (TIMx_CR1 register).

*Note:* *If OCi is selected on Timer A as trigger output (MMS=1xx), its rising edge is used to clock the counter of Timer B.*

**Using one timer to enable another timer**

In this example, we control the enable of Timer B with the output compare 1 of Timer A. Refer to *Figure 52* for connections. Timer B counts on the divided internal clock only when OC1REF of Timer A is high. Both counter clock frequencies are divided by 4 by the prescaler compared to $f_{MASTER}$ ($f_{CK\_CNT} = f_{MASTER}/4$).

1. Configure Timer A master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIMx_CR2 register).

2. Configure the Timer A OC1REF waveform (TIMx_CCMR1 register).

3. Configure Timer B to get the input trigger from Timer A (TS=001 in the TIMx_SMCR register).

4. Configure Timer B in trigger gated mode (SMS=101 in TIMx_SMCR register).

5. Enable Timer B by writing '1' in the CEN bit (TIMx_CR1 register).

6. Start Timer A by writing '1' in the CEN bit (TIMx_CR1 register).

*Note:* *The counter 2 clock is not synchronized with counter 1, this mode only affects the Timer B counter enable signal.*

**Figure 52. Gating Timer B with OC1REF of Timer A**
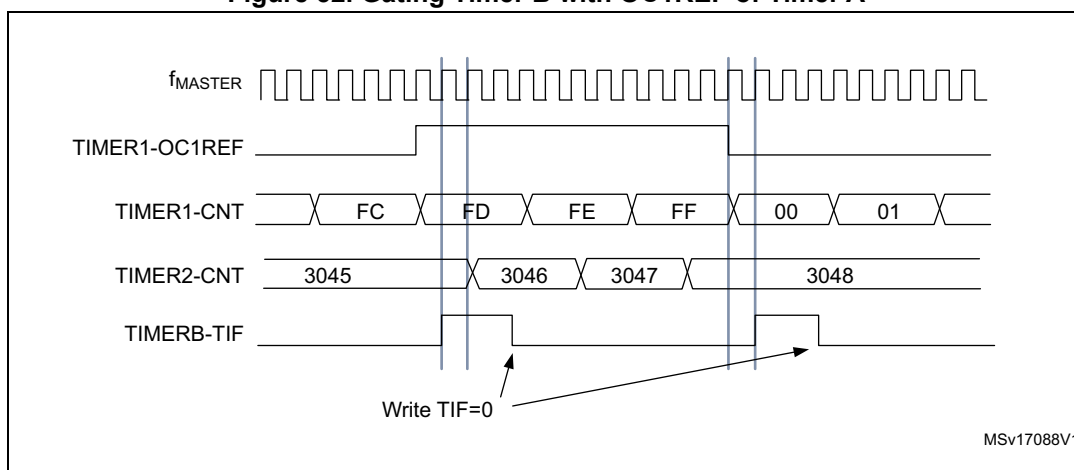


In the example in *Figure 52*, the Timer B counter and prescaler are not initialized before being started. So they start counting from their current value. It is possible to start from a given value by resetting both timers before starting Timer A. You can then write any value you want in the timer counters. The timers can easily be reset by software using the UG bit in the TIMx_EGR registers.

In the next example, we synchronize Timer A and Timer B. Timer A is the master and starts from 0. Timer B is the slave and starts from E7h. The prescaler ratio is the same for both timers. Timer B stops when Timer A is disabled by writing '0' to the CEN bit in the TIMx_CR1 register:

1.  Configure Timer A master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIMx_CR2 register).

2.  Configure the Timer A OC1REF waveform (TIMx_CCMR1 register).

3.  Configure Timer B to get the input trigger from Timer A (TS=001 in the TIMx_SMCR register).

4.  Configure Timer B in trigger gated mode (SMS=101 in TIMx_SMCR register).

5.  Reset Timer A by writing '1' in UG bit (TIMx_EGR register).

6.  Reset Timer B by writing '1' in UG bit (TIMx_EGR register).

7.  Initialize Timer B to 0xE7 by writing 'E7h' in the Timer B counter (TIMx_CNTRL).

8.  Enable Timer B by writing '1' in the CEN bit (TIMx_CR1 register).

9.  Start Timer A by writing '1' in the CEN bit (TIMx_CR1 register).

10. Stop Timer A by writing '0' in the CEN bit (TIMx_CR1 register).

**Figure 53. Gating Timer B with the counter enable signal of Timer A (CNT_EN)**
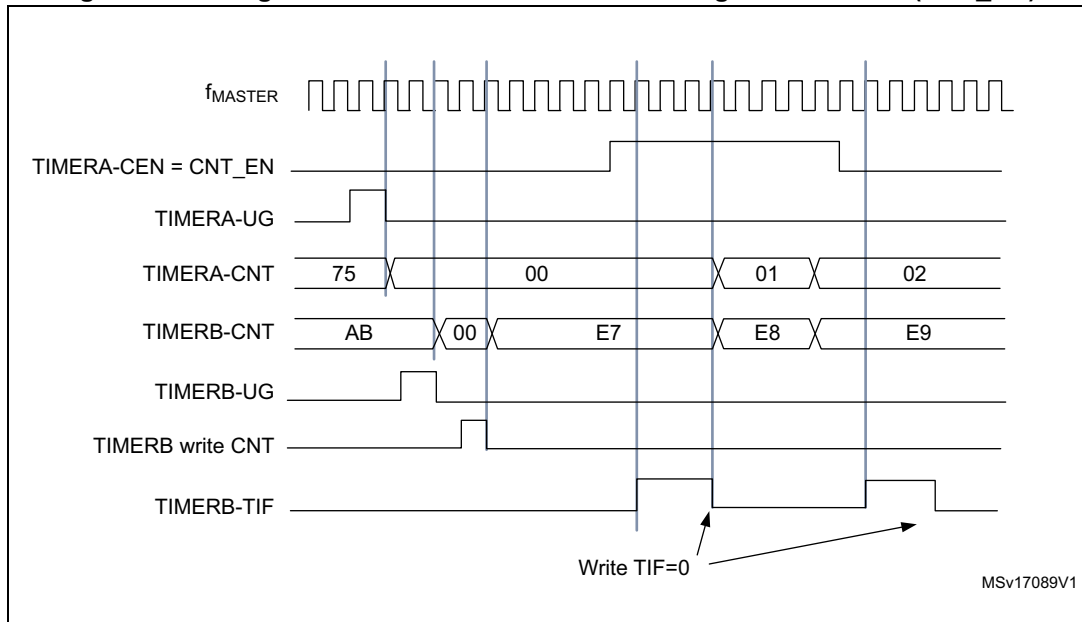


**Using one timer to start another timer**

In this example, we set the enable of Timer B with the update event of Timer A. Refer to *Figure 51* for connections. Timer B starts counting from its current value (which can be non-zero) on the divided internal clock as soon as the update event is generated by Timer A. When Timer B receives the trigger signal its CEN bit is automatically set and the counter counts until we write '0' to the CEN bit in the TIMx_CR1 register. Both counter clock frequencies are divided by 4 by the prescaler compared to $f_{MASTER}$ ($f_{CK\_CNT}$ = $f_{MASTER}/4$).

1. Configure Timer A master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIMx_CR2 register).
2. Configure the Timer A period (TIMx_ARR registers).
3. Configure Timer B to get the input trigger from Timer A (TS=001 in the TIMx_SMCR register).
4. Configure Timer B in trigger mode (SMS=110 in TIMx_SMCR register).
5. Start Timer A by writing '1' in the CEN bit (TIMx_CR1 register).

**Figure 54. Triggering Timer B with update event of Timer A (TIMERA-UEV)**
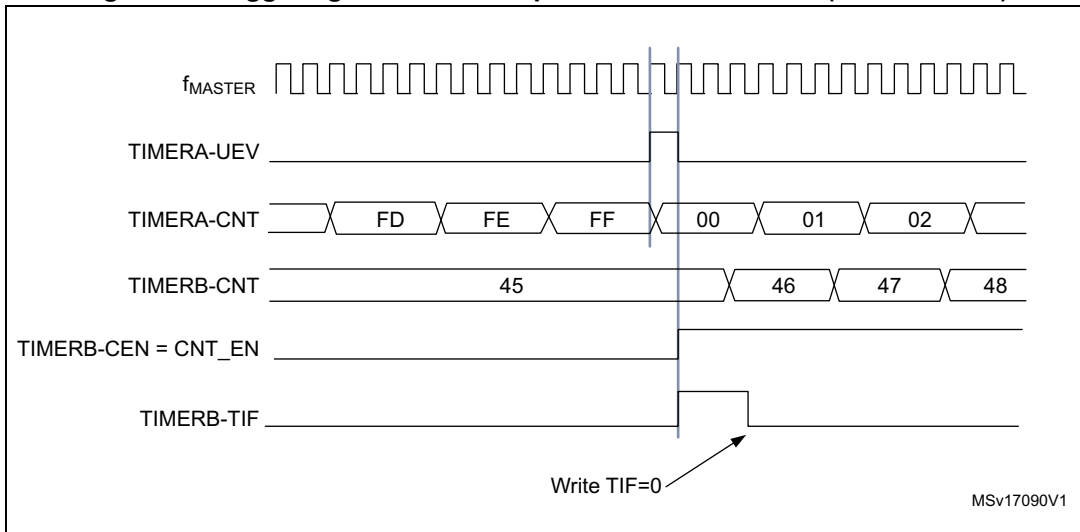


As in the previous example, you can initialize both counters before starting counting. *Figure 55* shows the behaviour with the same configuration as in the *Figure 53* but in trigger standard mode instead of trigger gated mode (SMS=110 in the TIMx_SMCR register).

**Figure 55. Triggering Timer B with counter enable CNT_EN of Timer A**
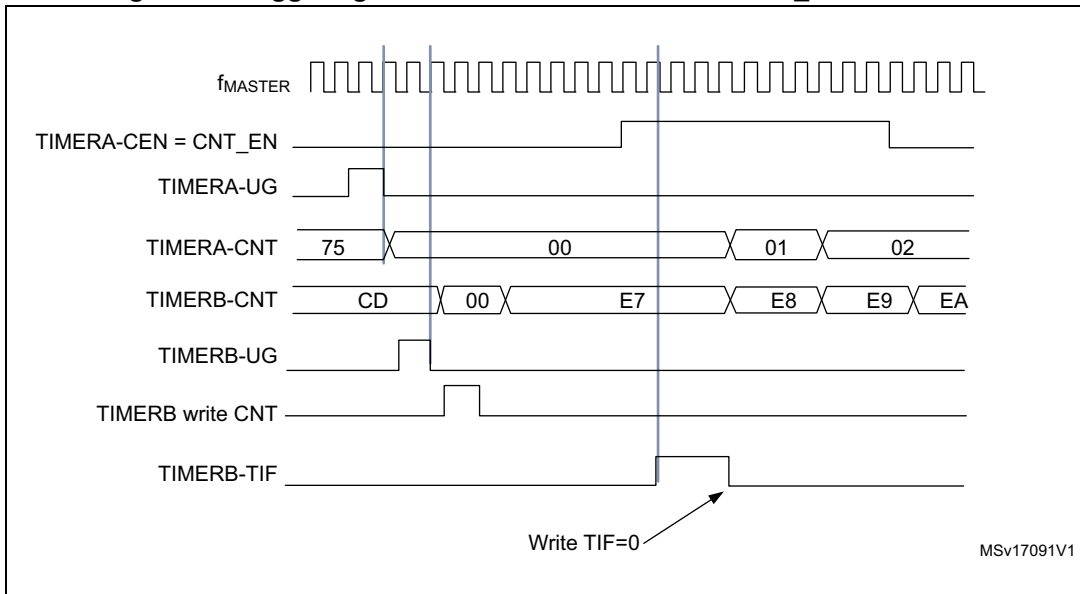
### Starting 2 timers synchronously in response to an external trigger

In this example, we set the enable of Timer A when its TI1 input rises, and the enable of Timer B with the enable of Timer A. Refer to *Figure 51* for connections. To ensure the counters alignment, Timer A must be configured in master/slave mode (slave with respect to TI1, master with respect to Timer B).

1. Configure Timer A master mode to send its Enable as trigger output (MMS=001 in the TIMx_CR2 register).
2. Configure Timer A slave mode to get the input trigger from TI1 (TS=100 in the TIMx_SMCR register).
3. Configure Timer A in trigger mode (SMS=110 in the TIMx_SMCR register).
4. Configure the Timer A in Master/Slave mode by writing MSM='1' (TIMx_SMCR register).
5. Configure Timer B to get the input trigger from Timer A (TS=001 in the TIMx_SMCR register).
6. Configure Timer B in trigger mode (SMS=110 in the TIMx_SMCR register).

When a rising edge occurs on TI1 (Timer A), both counters starts counting synchronously on the internal clock and both TIF flags are set.

*Note:* *In this example both timers are initialized before starting (by setting their respective UG bits). Both counters starts from 0, but you can easily insert an offset between them by writing any of the counter registers (TIMx_CNT). You can see that the master/slave mode insert a delay between CNT_EN and CK_PSC on Timer A.*

**Figure 56. Triggering Timer A and B with Timer A TI1 input**



MSv17092V1
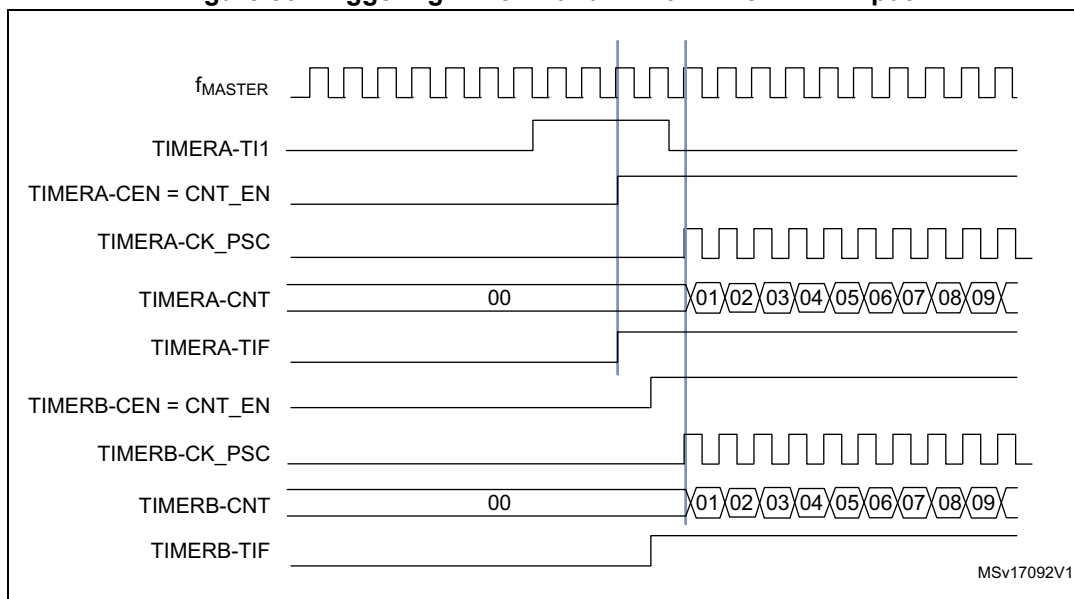
## 17.5 TIMx capture/compare channels

The timer I/O pins (TIMx_CH*i*) can be configured either for input capture or output compare functions. The choice made by configuring the CC*i*S channel selection bits in the capture/compare channel mode register (TIMx_CCMR*i*), where *i* is the channel number.

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), an input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

**Figure 57. Capture/compare channel 1 main circuit**
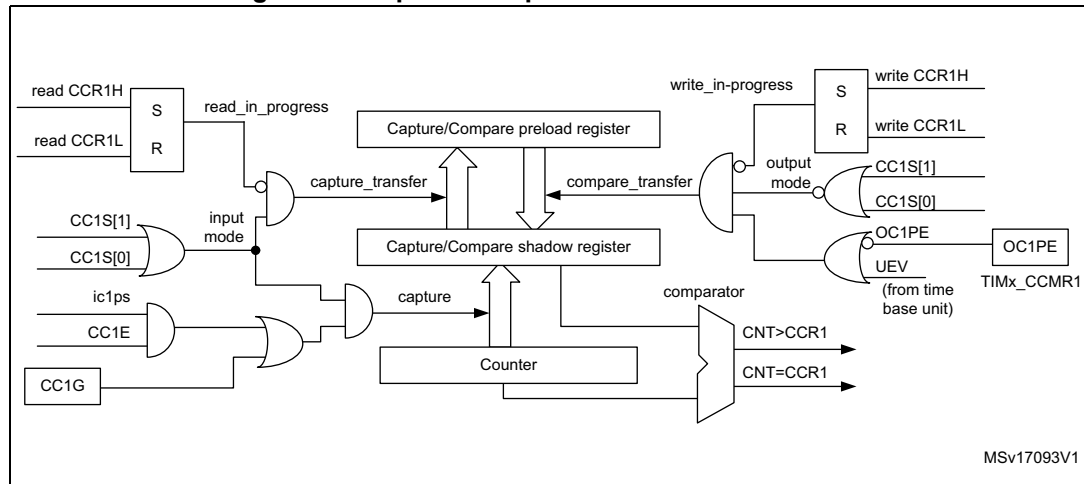


The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register. In capture mode, captures are actually done in the shadow register, which is copied into the preload register. In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

When the channel is configured in output mode (CC*i*S=0b00 in the TIMx_CCMR*i* register) where *i* is the channel number, the TIMx_CCR*i* register can be accessed without any restriction.

When the channel is configured in input mode, the sequence for reading the TIMx_CCR*i* register is the same as for the counter. See *Figure 58*. When a capture occurs, the content of the counter is captured into the TIMx_CCR*i* shadow register. Then this value is loaded into the preload register, except during the read sequence, when the preload register is frozen.

**Figure 58. 16-bit read sequence for the TIMx_CCR*i* register in capture mode**
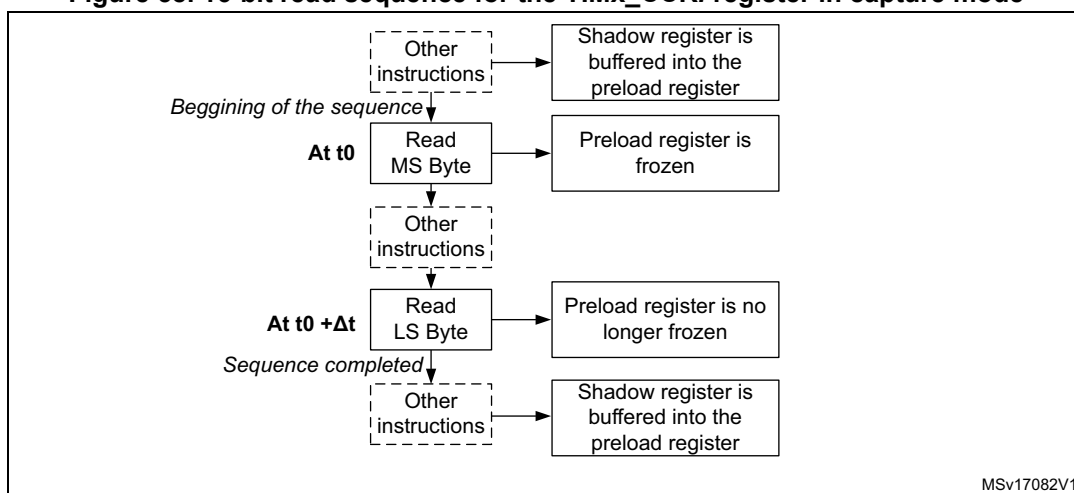


*Figure 58* shows the sequence for reading the CCR*i* registers in the 16-bit timers. This buffered value remains unchanged until the 16-bit read sequence is completed.

After a complete reading sequence, if only the TIMx_CCR*i*L register is read, it returns the LS Byte of the count value at the time of the read.

If the MS byte is read after the LS byte, it no longer corresponds to the same captured value as the LS byte.

### 17.5.1 Write sequence for 16-bit TIMx_CCR*i* registers

16-bit values are loaded in the TIMx_CCR*i* registers through preload registers. This must be performed by two write instructions, one for each byte. The MS byte must be written first.

The shadow register update is blocked as soon as the MS byte has been written, and stays blocked until the LS byte has been written. Do not use the LDW instruction, as this writes the LS byte first, and would produce wrong results in this case.

### 17.5.2 Input stage

**Figure 59. Channel input stage block diagram**
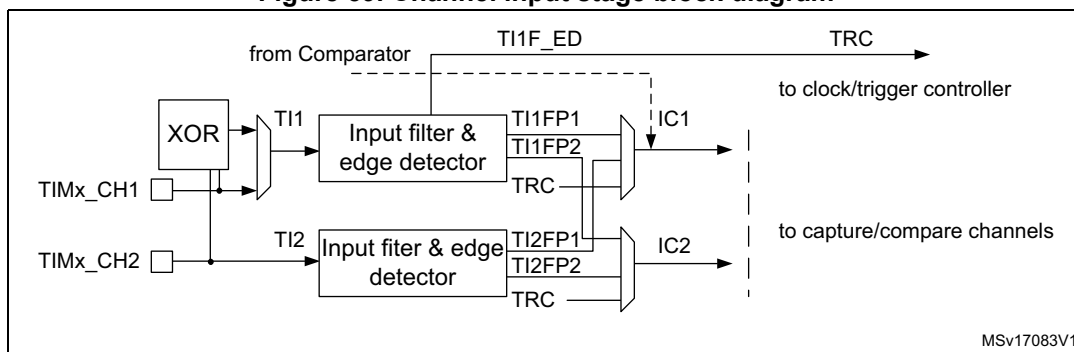


As shown in *Figure 60*, the input stage samples the corresponding TI*i* input to generate a filtered signal TI*i*F. Then, an edge detector with polarity selection generates a signal (TI*i*FPx) which can be used as trigger input by the clock/trigger controller or as the capture command. It is prescaled before the capture register (IC*i*PS).

**Figure 60. Input stage of TIM 1 channel 1**



MSv17084V1
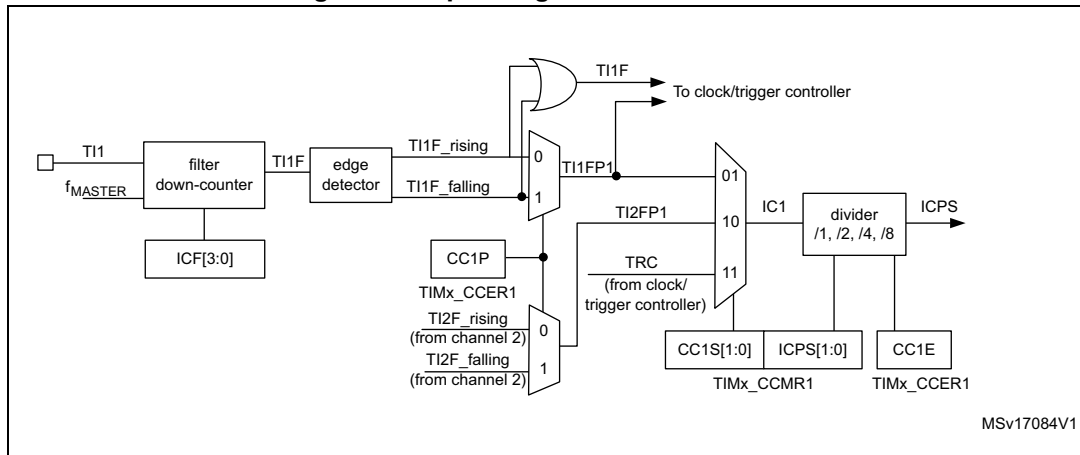
### 17.5.3 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx_CCR*i*) are used to latch the value of the counter after a transition detected on the corresponding IC*i* signal. When a capture occurs, the corresponding CC*i*IF flag (TIMx_SR1 register) is set.

An interrupt can be sent if it is enabled by setting the CC*i*IE bit in the TIMx_IER register. If a capture occurs while the CC*i*IF flag was already high, then the over-capture flag CC*i*OF (TIMx_SR2 register) is set. CC*i*IF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIMx_CCR*i*L register. CC*i*OF is cleared when you write it to '0'.

The following example shows how to capture the counter value in TIMx_CCR1 when TI1 input rises. To do this, use the following procedure:

1.  Select the active input: For example, to link the TIMx_CCR1 register to the TI1 input, write the CC1S bits to 0b01 in the TIMx_CCMR1 register. This configures the channel in input mode and the TIMx_CCR1 register becomes read-only.

2.  Program the input filter duration that is needed for the type of the signal to be conntected to the timer. This is done for each TI*i* input using the IC*i*F bits in the TIMx_CCMR*i* register. For example, if you know that when, the input signal toggles, it is unstable for up to 5 $f_{MASTER}$ cycles, you must program the filter duration longer than 5 clock cycles. The filter bits allow you to select a duration of 8 cycles by writing the value 0b0011 in in these bits the TIMx_CCMR1 register. With this filter setting, a transition on TI1 is valid only when 8 consecutive samples with the new level have been detected (sampled at $f_{MASTER}$ frequency).

3.  Select the edge of the active transition on the TI1 channel by writing CC1P bit to '0' in the TIMx_CCER1 register (rising edge in this case).

4.  Program the input prescaler. In our example, we want the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to 0b00 in the TIMx_CCMR1 register).

5.  Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx_CCER1 register.

6.  If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx_IER register.

When an input capture occurs:

- The TIMx_CCR1 register gets the value of the counter on the active transition.
- The input capture flag (CC1IF) is set (interrupt flag). The overcapture flag CC1OF is also set if at least two consecutive captures occured while the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.

To handle the overcapture event (CC1OF flag), it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

*Note:*      *It is not possible to send an IC interrupt without actually capturing the counter value in the TIMx_CCRx register. Nevertheless, it is possible to generate the capture event by software by setting the corresponding CCiG bit in the TIMx_EGR register.*

### PWM input signal measurement

This mode is a particular case of input capture mode. The procedure is the same except:

- Two ICi are mapped on the same TIi input.
- These 2 ICi are active on edges with opposite polarity.
- One of the two TIiFP is selected as trigger input and the clock/trigger controller is configured in trigger reset mode.

**Figure 61. PWM input signal measurement**
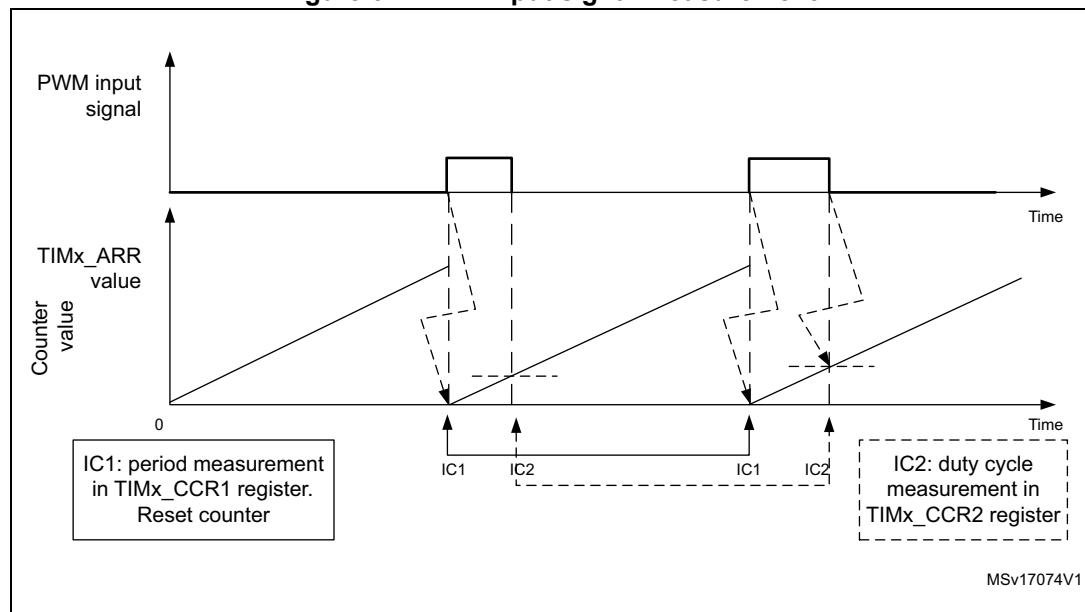
For example, you can measure the period (in the TIMx_CCR1 register) and the duty cycle (in the TIMx_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on $f_{MASTER}$ frequency and prescaler value):

1. Select the active input capture or trigger input for TIMx_CCR1: write the CC1S bits to 0b01 in the TIMx_CCMR1 register (TI1FP1 selected).

2. Select the active polarity for TI1FP1 (used both for capture in TIMx_CCR1 and counter clear): write the CC1P bit to '0' (active on rising edge).

3. Select the active input for TIMx_CCR2: write the CC2S bits to 0b10 in the TIMx_CCMR2 register (TI1FP2 selected).

4. Select the active polarity for TI1FP2 (used for capture in TIMx_CCR2): write the CC2P bit to '1' (active on falling edge).

5. Select the valid trigger input: write the TS bits to 0b101 in the TIMx_SMCR register (TI1FP1 selected).

6. Configure the clock/trigger controller in reset mode: write the SMS bits to '100' in the TIMx_SMCR register.

7. Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx_CCER1 register.

**Figure 62. PWM input signal measurement example**
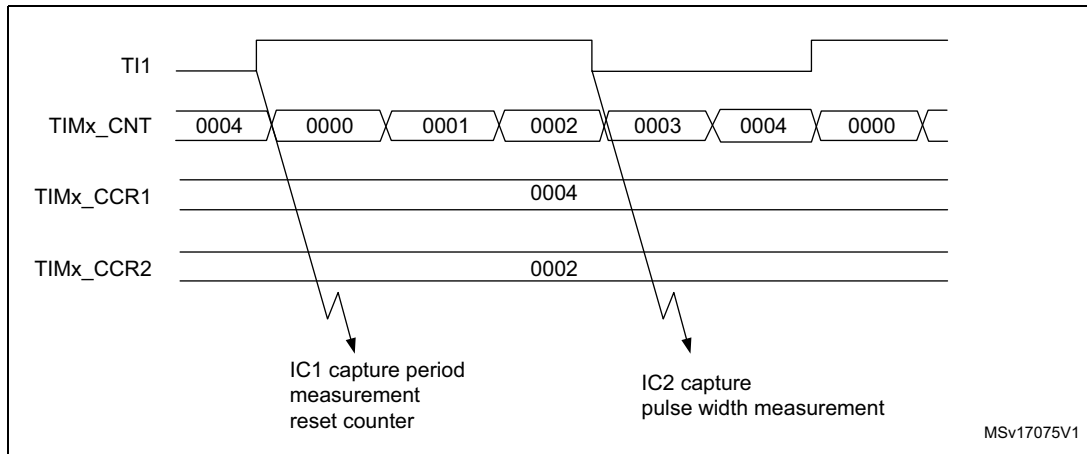


**17.5.4    Output stage**

The output stage generates an intermediate waveform called OC*i*REF (active high) which is then used for reference. Break functions and polarity act at the end of the chain.

**Figure 63. Channel output stage block diagram**
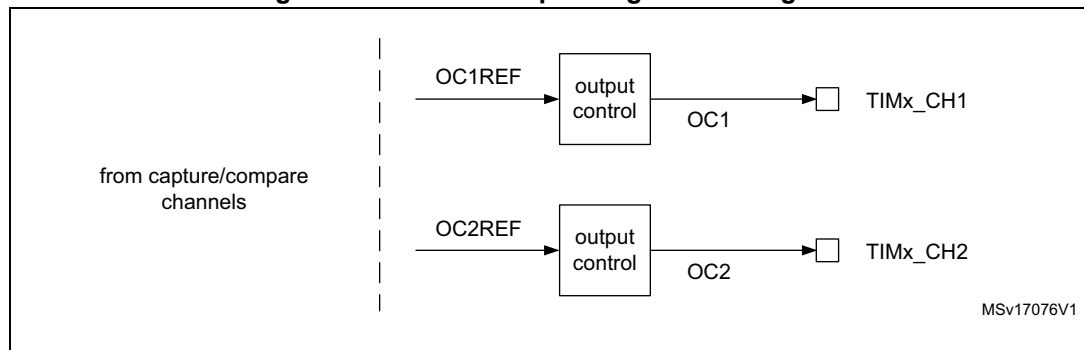
**Figure 64. Output stage of channel 1**
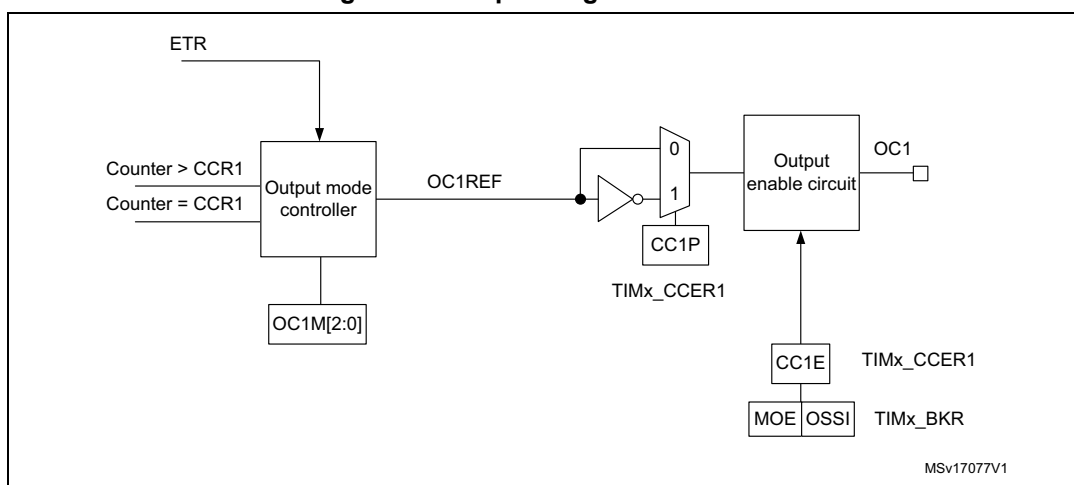


### 17.5.5 Forced output mode

In output mode (CC*i*S bits = 0b00 in the TIMx_CCMR*i* register) where *i* is the channel number, each output compare signal can be forced to high or low level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal to its active level, you just need to write '101' in the OC*i*M bits in the corresponding TIMx_CCMR*i* register. Thus OC*i*REF is forced high (OC*i*REF is always active high) and the OC*i* output is forced to high or low level depending on the CC*i*P polarity bit.

For example: CC*i*P=0 (OC*i* active high) => OC*i* is forced to high level.

The OC*i*REF signal can be forced low by writing the OC*i*M bits to 0b100 in the TIMx_CCMRx register.

Anyway, the comparison between the TIMx_CCR*i* shadow register and the counter is still performed and allows the flag to be set. Interrupt requests can be sent accordingly. This is described in the Output Compare Mode section below.

### 17.5.6 Output compare mode

This function is used to control an output waveform or indicate when a period of time has elapsed.

When a match is found between the capture/compare register and the counter:

- Depending on the output compare mode, the corresponding OC*i* output pin:
  - keeps its level (OC*i*M=0b000),
  - is set active (OC*i*M=0b001),
  - is set inactive (OC*i*M=0b010)
  - or toggles (OC*i*M=0b011)
- Sets a flag in the interrupt status register (CC*i*IF bit in the TIMx_SR1 register).
- Generates an interrupt if the corresponding interrupt mask is set (CC*i*IE bit in the TIMx_IER register).

The output compare mode is defined by the OC*i*M bits in the TIMx_CCMR*i* register. The active or inactive level polarity is defined by the CC*i*P bit in the TIMx_CCER*i* register.

The TIMx_CCR*i* registers can be programmed with or without preload registers using the OC*i*PE bit in the TIMx_CCMR*i* register.

In output compare mode, the update event UEV has no effect on the OC*i*REF and OC*i* output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse.

Procedure:

1.  Select the counter clock (internal, external, prescaler).
2.  Write the desired data in the TIMx_ARR and TIMx_CCR*i* registers.
3.  Set the CC*i*IE bit if an interrupt request is to be generated.
4.  Select the output mode as follwos:
    –   Write OC*i*M = 0b011 to toggle OC*i* output pin when CNT matches CCR*i*
    –   Write OC*i*PE = 0 to disable preload register
    –   Write CC*i*P = 0 to select active high polarity
    –   Write CC*i*E = 1 to enable the output
5.  Enable the counter by setting the CEN bit in the TIMx_CR1 register.

The TIMx_CCR*i* register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OC*i*PE='0', else TIMx_CCR*i* shadow register will be updated only at the next update event UEV). An example is given in *Figure 65*.

**Figure 65. Output compare mode, toggle on OC1**
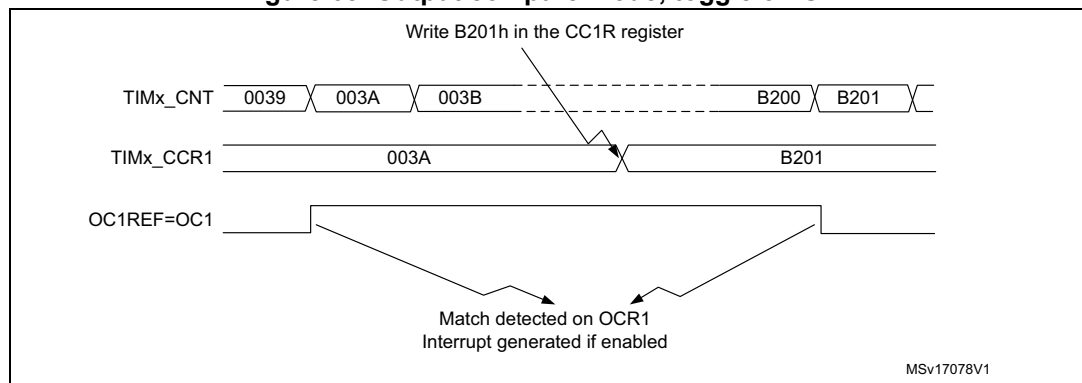


### 17.5.7 PWM mode

Pulse Width Modulation mode allows you to generate a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCR*i* register.

The PWM mode can be selected independently on each channel (one PWM per OC*i* output) by writing 0b110 (PWM mode 1) or 0b111 (PWM mode 2) in the OC*i*M bits in the TIMx_CCMR*i* register. You must enable the corresponding preload register by setting the OC*i*PE bit in the TIMx_CCMR*i* register, and optionally enable the auto-reload preload register (in up-counting or center-aligned modes) by setting the ARPE bit in the TIMx_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the UG bit in the TIMx_EGR register.

OC*i* polarity is software programmable using the CC*i*P bit in the TIMx_CCER*i* register. It can be programmed as active high or active low. OC*i* output is enabled by a combination of CC*i*E, MOE, OIS*i*, OSSR and OSSI bits (TIMx_CCER*i* and TIMx_BKR registers). Refer to the TIMx_CCER*i* register description for more details.

In PWM mode (1 or 2), TIMx_CNT and TIMx_CCR*i* are always compared to determine whether TIMx_CCR*i*≤TIMx_CNT or TIMx_CNT≤TIMx_CCR*i* (depending on the direction of the counter).

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

### PWM edge-aligned mode

Up-counting configuration

Up-counting is active when the DIR bit in the TIMx_CR1 register is low.

In the following example, we consider the PWM mode 1. The reference PWM signal OC*i*REF is high as long as TIMx_CNT <TIMx_CCR*i* else it becomes low. If the compare value in TIMx_CCR*i* is greater than the auto-reload value (in TIMx_ARR) then OC*i*REF will be held at '1'. If the compare value is 0 then OC*i*REF will be held at '0'. *Figure 66* shows some edge-aligned PWM waveforms in an example where TIMx_ARR=8.

**Figure 66. Edge-aligned counting mode PWM mode 1 waveforms (ARR=8)**
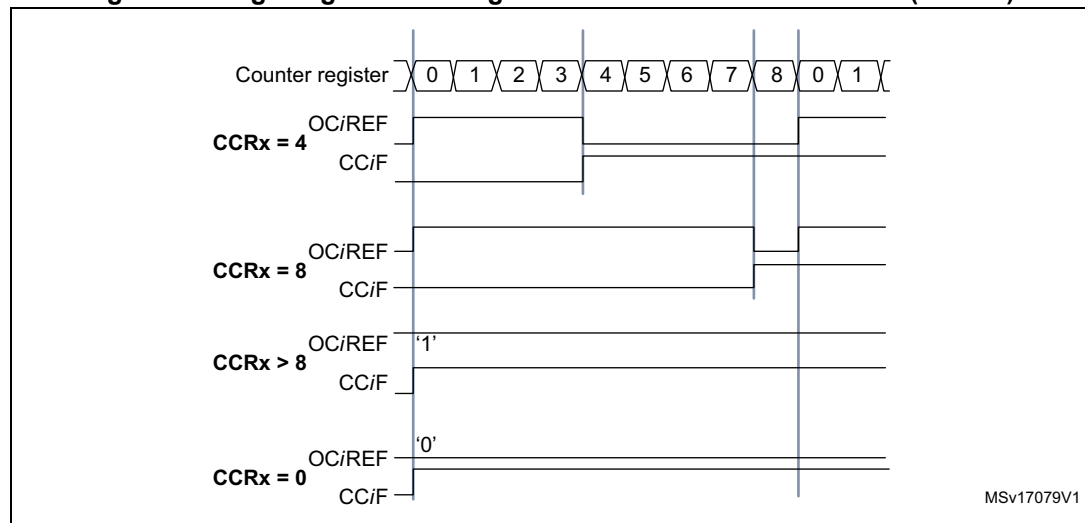


Down-counting configuration

Down-counting is active when DIR bit in TIMx_CR1 register is high. Refer to *Down-counting mode on page 139*

In PWM mode 1, the reference signal OC*i*REF is low as long as TIMx_CNT> TIMx_CCR*i* else it becomes high. If the compare value in TIMx_CCR*i* is greater than the auto-reload value in TIMx_ARR, then OC*i*REF will be held at '1'. 0% PWM is not possible in this mode.

### PWM center-aligned mode

Center-aligned mode is active when the CMS bits in TIMx_CR1 register are different from '00' (all the remaining configurations having the same effect on the OC*i*REF/OC*i* signals).

The compare flag is set when the counter counts up, when it counts down or both when it counts up and down depending on the CMS bits configuration. The direction bit (DIR) in the TIMx_CR1 register is updated by hardware and is read-only in this mode. Refer to *Center-aligned mode (up/down counting) on page 141*.

*Figure 67* shows some center-aligned PWM waveforms in an example where:

- The TIMx_ARR=8,
- PWM mode is PWM mode 1,
- the flag is set (arrow symbol in *Figure 67*) in three different cases:
  - only when the counter counts down (CMS=0b01)
  - only when the counter counts up (CMS=0b10) .
  - when the counter counts up and down (CMS=0b11) .

**Figure 67. Center-aligned PWM waveforms (ARR=8)**
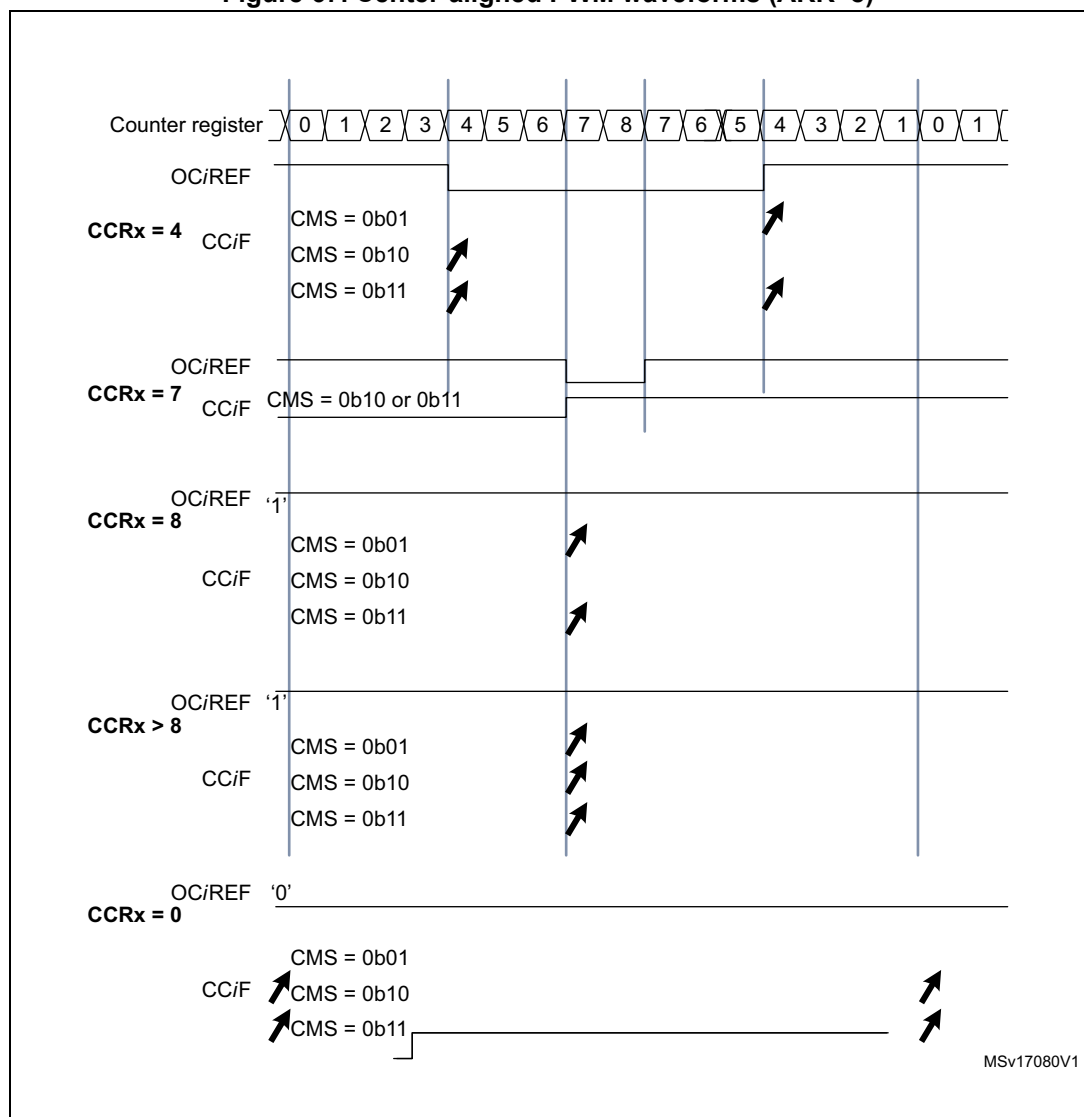
**One pulse mode**

One Pulse Mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the clock/trigger controller. Generating the waveform can be done in output compare mode or PWM mode. You select One Pulse Mode by setting the OPM bit in the TIMx_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be:

In up-counting: $CNT < CCRi \leq ARR$ (in particular, $0 < CCRi$),

In down-counting: $CNT > CCRi$.

**Figure 68. Example of one pulse mode**
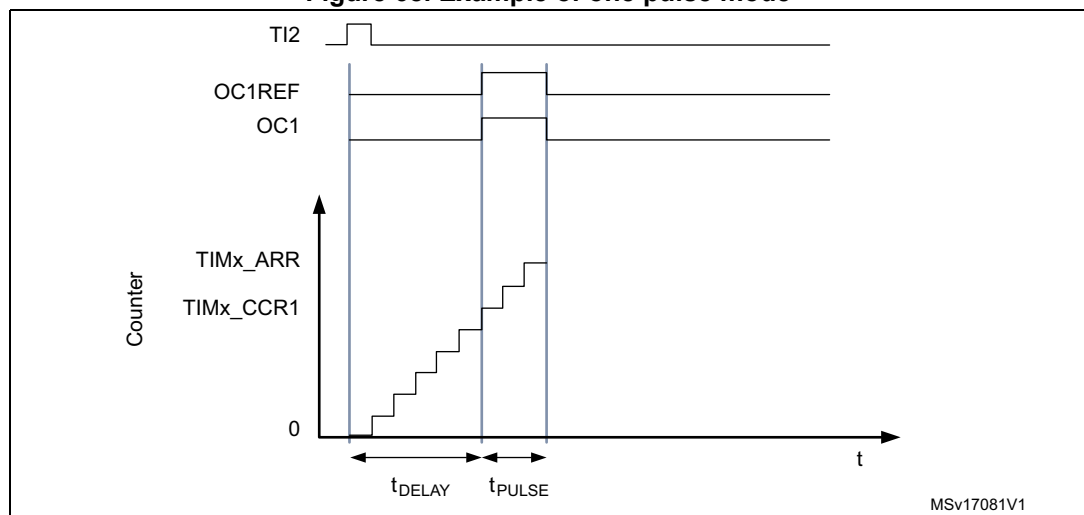


For example you may want to generate a positive pulse on OC1 with a length of $t_{PULSE}$ and after a delay of $t_{DELAY}$ as soon as a positive edge is detected on the TI2 input pin.

Let's use IC2 as trigger 1:

- Map IC2 on TI2 by writing CC2S=0b01 in the TIMx_CCMR2 register.
- IC2 must detect a rising edge, write CC2P='0' in the TIMx_CCER1 register.
- Configure IC2 as trigger for the clock/trigger controller (TRGI) by writing TS=0b110 in the TIMx_SMCR register.
- IC2 is used to start the counter by writing SMS to 0b110 in the TIMx_SMCR register (trigger mode).

The OPM waveform is defined by writing the compare registers (taking into account the clock frequency and the counter prescaler).

- The $t_{DELAY}$ is defined by the value written in the TIMx_CCR1 register.
- The $t_{PULSE}$ is defined by the difference between the auto-reload value and the compare value (TIMx_ARR - TIMx_CCR1).
- Let's say you want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload value. To do this you enable PWM mode 2 by writing OC*i*M=0b111 in the TIMx_CCMR1 register. You can optionally enable the preload registers by writing OC1PE='1' in the TIMx_CCMR1 register and ARPE in the TIMx_CR1 register. In this case you have to write the compare value in the TIMx_CCR1 register, the auto-reload value in the TIMx_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0' in this example.

In our example, the DIR and CMS bits in the TIMx_CR1 register should be low.

You only want 1 pulse, so you write '1' in the OPM bit in the TIMx_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0).

Particular case: OC*i* fast enable:

In One Pulse Mode, the edge detection on TI*i* input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay $t_{DELAY}$ min we can get.

If you want to output a waveform with the minimum delay, you can set the OC*i*FE bit in the TIMx_CCMR*i* register. Then OC*i*REF (and OC*i*) will be forced in response to the stimulus, without taking in account the comparison. Its new level will be the same as if a compare match had occured. OC*i*FE acts only if the channel is configured in PWM1 or PWM2 mode.

## 17.5.8 Using the break function

The break function is often used in motor control. When using the break function, the output enable signals and inactive levels are modified according to additional control bits (MOE, OSSR and OSSI bits in the TIMx_BKR register).

When exiting from reset, the break circuit is disabled and the MOE bit is low. You can enable the break function by setting the BKE bit in the TIMx_BKR register. The break input polarity can be selected by configuring the BKP bit in the same register. BKE and BKP can be modified at the same time.

Because MOE falling edge can be asynchronous, a resynchronization circuit has been inserted between the actual signal (acting on the outputs) and the synchronous control bit (accessed in the TIMx_BKR register). It results in some delays between the asynchronous and the synchronous signals. In particular, if you write MOE to 1 whereas it was low, you must insert a delay (dummy instruction) before reading it correctly.

When a break occurs (selected level on the break input):

- The MOE bit is cleared asynchronously, putting the outputs in inactive state, idle state or in reset state (selected by the OSSI bit). This feature functions even if the MCU oscillator is off.

- Each output channel is driven with the level programmed in the OIS*i* bit in the TIMx_OISR register as soon as MOE=0. If OSSI=0 then the timer releases the enable output else the enable output remains high.

- The break status flag (BIF bit in the TIMx_SR1 register) is set. An interrupt can be generated if the BIE bit in the TIMx_IER register is set.

- If the AOE bit in the TIMx_BKR register is set, the MOE bit is automatically set again at the next update event UEV. This can be used to perform a regulation, for instance. Else, MOE remains low until you write it to '1' again. In this case, it can be used for security and you can connect the break input to an alarm from power drivers, thermal sensors or any security components.

*Note:*     *The break inputs are acting on level. Thus, the MOE cannot be set while the break input is active (neither automatically nor by software). In the meantime, the status flag BIF cannot be cleared.*

The break can be generated by the break input (BKIN) which has a programmable polarity and can be enabled or disabled by setting or resetting the BKE bit in TIMx_BKR register.

In addition to the break inputs and the output management, a write protection has been implemented inside the break circuit to safeguard the application. It allows you to freeze the configuration of several parameters (OC*i* polarities and state when disabled, OC*i*M configurations, break enable and polarity). You can choose from 3 levels of protection selected by the LOCK bits in the TIMx_BKR register. The LOCK bits can be written only once after an MCU reset.

*Figure 69* shows an example of behavior of the outputs in response to a break.

**Figure 69. Behavior of outputs in response to a break**
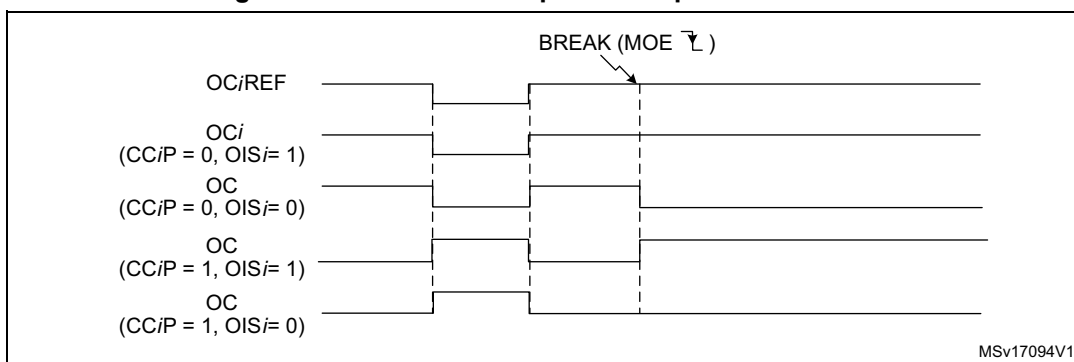


### 17.5.9    Clearing the OC*i*REF signal on an external event

The OC*i*REF signal of a given channel can be cleared when a high level is detected on ETRF (if OC*i*CE='1' in the TIMx_CCMR*i* register, one enable bit per channel). The OC*i*REF signal remains low until the next UEV update event occurs. This function can be used in output compare mode and PWM mode only, it does not work in forced mode.

It can be connected to the output of a comparator and be used for current handling, for instance.

For example, the OC*i*REF signal can be connected to the output of a comparator to be used for current handling. In this case, the external trigger must be configured as follows:

1. The External Trigger Prescaler should be kept off: bits ETPS[1:0] in the TIMx_ETR register set to '00'.

2. The external clock mode 2 must be disabled: bit ECE in the TIMx_ETR register set to '0'.

3. The External Trigger Polarity (ETP) and the External Trigger Filter (ETF) can be configured as desired.

Refer to the external trigger input block diagram *Figure 43 on page 145*

*Figure 70* shows the behavior of the OC*i*REF signal when the ETRF input becomes high, for both values of the enable bit OC*i*CE. In this example, the timer is programmed in PWM mode.

**Figure 70. ETR activation**
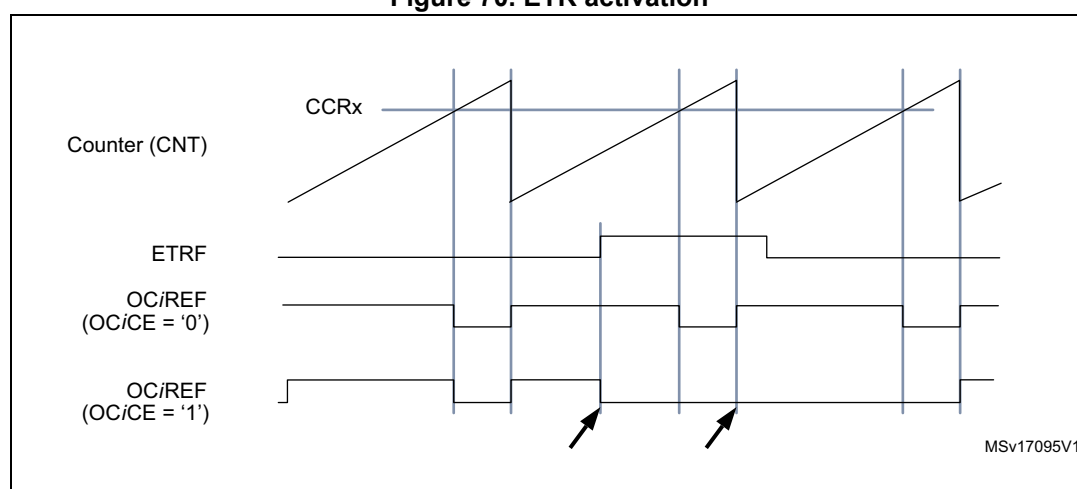


### 17.5.10 Encoder interface mode

This mode is typically used for motor control. To select Encoder Interface mode write SMS=0b001 in the TIMx_SMCR register if the counter is counting on TI2 edges only, SMS=0b010 if it is counting on TI1 edges only and SMS=0b011 if it is counting on both TI1 and TI2 edges.

Select the TI1 and TI2 polarity by programming the CC1P and CC2P bits in the TIMx_CCER1 register. When needed, you can program the input filter as well.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to *Table 34*. The counter is clocked by each valid transition on TI1FP1 or TI2FP2 (TI1 and TI2 after input filter and polarity selection, TI1FP1=TI1 if not filtered and not inverted, TI2FP2=TI2 if not filtered and not inverted) assuming that it is enabled (CEN bit in TIMx_CR1 register written to '1'). The sequence of transitions of the two inputs is evaluated and generates count pulses as well as the direction signal. Depending on the sequence the counter counts up or down, the DIR bit in the TIMx_CR1 register is modified by hardware accordingly. The DIR bit is calculated at each transition on any input (TI1 or TI2), whatever the counter is counting on TI1 only, TI2 only or both TI1 and TI2.

Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value in the TIMx_ARR register (0 to ARR or ARR down to 0 depending on the direction). So you must configure TIMx_ARR before starting. In the same way, the capture, compare, prescaler, trigger output features continue to work as normal. Encoder mode and External clock mode 2 are not compatible and must not be selected together.

In this mode, the counter is modified automatically following the speed and the direction of the incremental encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 don't switch at the same time.

**Table 34. Counting direction versus encoder signals**

| Active edge | Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1) | TI1FP1 signal | | TI2FP2 signal | |
|---|---|---|---|---|---|
| | | Rising | Falling | Rising | Falling |
| Counting on TI1 only | High | Down | Up | No count | No count |
| | Low | Up | Down | No count | No count |
| Counting on TI2 only | High | No count | No count | Up | Down |
| | Low | No count | No count | Down | Up |
| Counting on TI1 and TI2 | High | Down | Up | Up | Down |
| | Low | Up | Down | Down | Up |

An external incremental encoder can be connected directly to the MCU without external interface logic. However, comparators will normally be used to convert the encoder's differential outputs to digital signals. This greatly increases noise immunity. The third encoder output which indicate the mechanical zero position, may be connected to an external interrupt input and trigger a counter reset.

The *Figure 71* gives an example of counter operation, showing count signal generation and direction control. It also shows how input jitter is compensated where both edges are selected. This might occur if the sensor is positioned near to one of the switching points. For this example we assume that the configuration is the following:

- CC1S = 0b01 (TIMx_CCMR1 register, IC1 mapped on TI1).
- CC2S = 0b01 (TIMx_CCMR2 register, IC2 mapped on TI2).
- CC1P = 0 (TIMx_CCER1 register, IC1 non-inverted, IC1=TI1).
- CC2P = 0 (TIMx_CCER2 register, IC2 non-inverted, IC2=TI2).
- SMS = 0b011 (TIMx_SMCR register, both inputs are active on both rising and falling edges).
- CEN = 1 (TIMx_CR1 register, Counter is enabled).

**Figure 71. Example of counter operation in encoder interface mode**
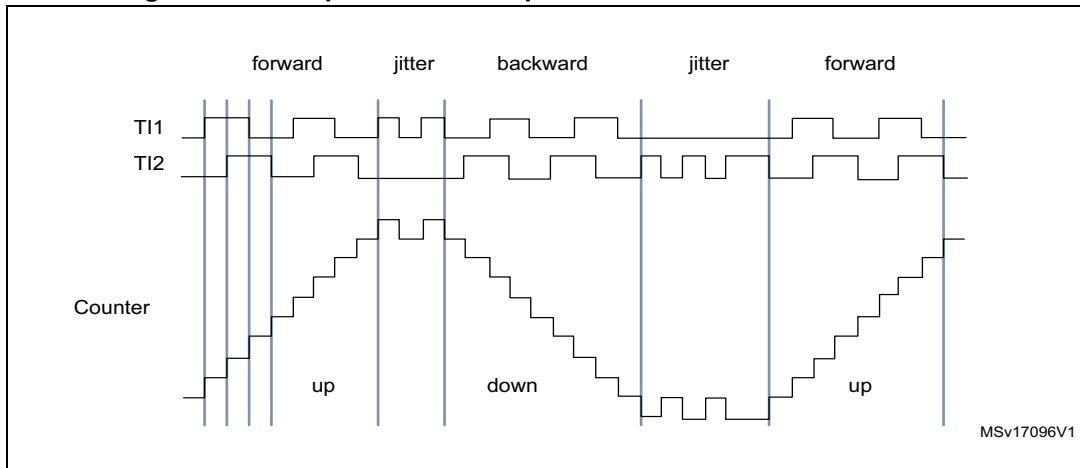


*Figure 72* gives an example of counter behaviour when IC1 polarity is inverted (same configuration as above except CC1P='1').

**Figure 72. Example of encoder interface mode with IC1 polarity inverted**
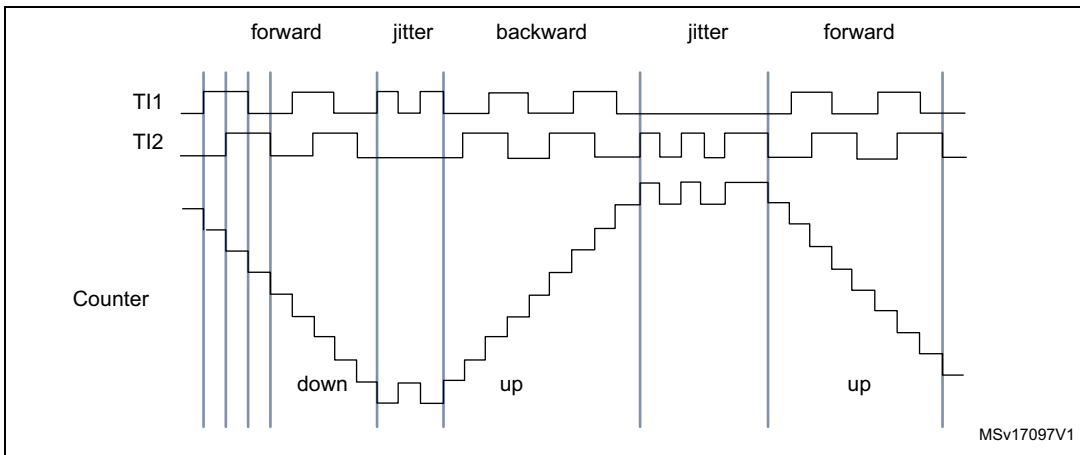


The timer, when configured in Encoder Interface mode provides information on the sensor's current position. You can obtain dynamic information (speed, acceleration, decceleration) by measuring the period between two encoder events using a second timer configured in capture mode. The output of the encoder which indicates the mechanical zero can be used for this purpose. Depending on the time between two events, the counter can also be read at regular times. You can do this by latching the counter value into a third input capture register if available (then the capture signal must be periodic and can be generated by another timer).

## 17.6 TIMx interrupts

TIMx has 6 interrupt request sources, mapped on 2 interrupt vectors:

- Break interrupt
- Trigger interrupt
- Commutation interrupt
- Capture/Compare 2 interrupt
- Capture/Compare 1 interrupt
- Update Interrupt (ex: overflow, underflow, counter initialization)

To use the interrupt features, for each interrupt channel used, set the desired "Interrupt Enable" bit: BIE, TIE, COMIE, CC*i*IE, UIE bits in the TIMx_IER register to enable interrupt requests.

The different interrupt sources can be also generated by software using the corresponding bits in the TIMx_EGR register.

### 17.6.1 TIMx wait-for-event capability

In wait-for-event mode (WFE), TIMx Capture/Compare, break, trigger and update interrupts can be used to wake up the device. The interrupt event must have been previously configured through bits TIMx_EV0 and TIMx_EV1 in the WFE_CR1 register (see *Section 9.5: WFE registers*).

## 17.7 TIMx registers

### 17.7.1 Control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ARPE | CMS[1:0] | | DIR | OPM | URS | UDIS | CEN |
| rw | rw | | rw | rw | rw | rw | rw |

Bit 7 **ARPE**: Auto-Reload Preload Enable

0: TIMx_ARR register is not buffered through a preload register. It can be written directly.
1: TIMx_ARR register is buffered through a preload register.

Bits 6:5 **CMS**: Center-aligned Mode Selection

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).
01: Center-aligned mode 1. The counter counts up and down alternately. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting down.
10: Center-aligned mode 2. The counter counts up and down alternately. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting up.
11: Center-aligned mode 3. The counter counts up and down alternately. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set both when the counter is counting up or down.

*Note: It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)*

*Encoder mode (SMS=001, 010 or 011 in GPT_SMCR register) must be disabled in center-aligned mode.*

Bit 4 **DIR**: Direction

0: Counter used as up-counter.
1: Counter used as down-counter.

*Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.*

Bit 3 **OPM**: One Pulse Mode

0: Counter is not stopped at update event
1: Counter stops counting at the next update event (clearing the CEN bit )

Bit 2 **URS**: Update Request Source

0: When enabled, an update interrupt request is sent as soon as registers are updated (counter overflow)
1: When enabled, an update interrupt request is sent only when the counter reaches the overflow.

Bit 1 **UDIS**: Update Disable

0: An Update event is generated as soon as a counter overflow occurs or a software update is generated or an hardware reset is generated by the clock/trigger mode controller. Buffered registers are then loaded with their preload values
1: An Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). The counter and the prescaler are re-initialized if the UG bit is set .

Bit 0 **CEN**: Counter Enable

0: Counter disabled
1: Counter enabled

## 17.7.2 Control register 2 (TIMx_CR2)

*Address offset: 0x01*

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | MMS[2:0] | | | Reserved | | | |
|  | rw | | |  | | | |

Bit 7 Reserved

Bits 6:4 **MMS**: Master mode selection

These bits select the information to be sent in master mode to other timers for synchronization (TRGO). The combination is as follows:

000: **Reset** - the UG bit from the TIM3_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (clock/trigger mode controller configured in trigger reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter Enable signal is used as trigger output (TRGO). It is used to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode. When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in TIM3_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO).

011: Reserved

100: Reserved

101: Reserved

111: Reserved

Bits 3:0 Reserved

### 17.7.3 Slave mode control register (TIMx_SMCR)

*Address offset: 0x02*

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| MSM | TS[2:0] | | | Reserved | SMS[2:0] | | |
| rw | rw | | | | rw | | |

Bit 7 **MSM**: Master/slave mode

0: No action

1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between timers (through TRGO).

Bits 6:4 **TS[2:0]**: Trigger selection

This bit-field selects the trigger input to be used to synchronize the counter.

　000: Internal trigger ITR0 connected to TIM4 TRGO

　001: Reserved

　010: Internal trigger ITR2 connected to TIM3 TRGO

　011: Internal trigger ITR3 connected to TIM2 TRGO

　100: TI1 Edge Detector (TI1F_ED)

　101: Filtered Timer Input 1 (TI1FP1)

　110: Filtered Timer Input 2 (TI2FP2)

　111: External Trigger input (ETRF)

*Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.*

Bit 3 Reserved

Bits 2:0 **SMS[2:0]**: Clock/trigger/slave mode selection

When external signals are selected, the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control register description).

　000: Clock/trigger controler disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.

　001: Reserved

　010: Reserved

　011: Reserved

　100: Trigger reset mode - Rising edge of the selected trigger signal (TRGI) reinitializes the counter and generates an update of the registers.

　101: Gated Mode - The counter clock is enabled when the trigger signal (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

　110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

　111: External clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

## 17.7.4 External trigger register (TIMx_ETR)

Address offset: 0x03

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ETP | ECE | ETPS[1:0] | | ETF[3:0] | | | |
| rw | rw | rw | | rw | | | |

Bit 7 **ETP**: External trigger polarity

This bit selects whether ETR or $\overline{ETR}$ is used for trigger operations

0: ETR is non-inverted, active at high level or rising edge.

1: ETR is inverted, active at low level or falling edge.

Bit 6 **ECE**: External clock enable

This bit enables External clock mode 2.

0: External clock mode 2 disabled.

1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.

*Note: Setting the ECE bit has the same effect as selecting the external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111 in the TIMx_SMCR register).*

*It is possible to use simultaneously the external clock mode 2 with the following modes: trigger standard mode, trigger reset mode and trigger gated mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111 in TIMx_SMCR register).*

*If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input will be ETRF.*

Bits 5:4 **ETPS**: External trigger prescaler

External trigger signal ETRP frequency must be at must 1/4 of $f_{MASTER}$ frequency. A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.

00: Prescaler OFF.

01: ETRP frequency divided by 2.

10: ETRP frequency divided by 4.

11: ETRP frequency divided by 8.

Bits 3:0 **ETF**: External trigger filter

This bit-field defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

0000: No filter, sampling is done at $f_{MASTER}$

0001: $f_{SAMPLING}=f_{MASTER}$, N=2

0010: $f_{SAMPLING}=f_{MASTER}$, N=4

0011: $f_{SAMPLING}=f_{MASTER}$, N=8

0100: $f_{SAMPLING}=f_{MASTER}/2$, N=6

0101: $f_{SAMPLING}=f_{MASTER}/2$, N=8

0110: $f_{SAMPLING}=f_{MASTER}/4$, N=6

0111: $f_{SAMPLING}=f_{MASTER}/4$, N=8

1000: $f_{SAMPLING}=f_{MASTER}/8$, N=6

1001: $f_{SAMPLING}=f_{MASTER}/8$, N=8

1010: $f_{SAMPLING}=f_{MASTER}/16$, N=5

1011: $f_{SAMPLING}=f_{MASTER}/16$, N=6

1100: $f_{SAMPLING}=f_{MASTER}/16$, N=8

1101: $f_{SAMPLING}=f_{MASTER}/32$, N=5

1110: $f_{SAMPLING}=f_{MASTER}/32$, N=6

1111: $f_{SAMPLING}=f_{MASTER}/32$, N=8

### 17.7.5 Interrupt enable register (TIMx_IER)

Address offset: 0x04

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BIE | TIE | | Reserved | | CC2IE | CC1IE | UIE |
| rw | rw | | | | rw | rw | rw |

Bit 7 **BIE**: Break interrupt enable
> 0: Break Interrupt disabled.
> 1: Break Interrupt enabled.

Bit 6 **TIE**: Trigger Interrupt Enable.
> 0: Trigger Interrupt disabled.
> 1: Trigger Interrupt enabled.

Bits 5:3 Reserved

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable
> 0: CC2 Interrupt disabled
> 1: CC2 Interrupt enabled

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable
> 0: CC1 Interrupt disabled
> 1: CC1 Interrupt enabled

Bit 0 **UIE**: Update interrupt enable
> 0: Update Interrupt disabled
> 1: Update Interrupt enabled

### 17.7.6 Status register 1 (TIMx_SR1)

Address offset: 0x05

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BIF | TIF | | Reserved | | CC2IF | CC1IF | UIF |
| rc_w0 | rc_w0 | | | | rc_w0 | rc_w0 | rc_w0 |

Bit 7 **BIF**: Break interrupt flag
> This flag is set by hardware as soon as the break input goes active. It can be cleared by software if the break input is not active.
> 0: No break event occurred.
> 1: An active level has been detected on the break input.

Bit 6 **TIF**: Trigger interrupt flag
> This flag is set by hardware on trigger event (active edge detected on TRGI signal, both edges in case gated mode is selected). It is cleared by software.
> 0: No trigger event occurred.
> 1: Trigger interrupt pending.

Bits 5:3 Reserved

Bit 2 **CC2IF**: Capture/Compare 2 interrupt flag
> Refer to CC1IF description

Bit 1 **CC1IF**: Capture/Compare 1 interrupt flag
  – If channel CC1 is configured as output:
  This flag is set by hardware when the counter matches the compare value. It is cleared by software.
  0: No match.
  1: The content of the counter TIMx_CNT has matched the content of the TIMx_CCR1 register.
  – If channel CC1 is configured as input:
  This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx_CCR1L register.
  0: No input capture occurred.
  1: The counter value has been captured in TIMx_CCR1 register (An edge has been detected on IC1 which matches the selected polarity).

Bit 0 **UIF**: Update interrupt flag
  This bit is set by hardware on an update event. It is cleared by software.
  0: No update occurred.
  1: Update interrupt pending. This bit is set by hardware when the registers are updated:
  – At overflow if UDIS=0 in the TIMx_CR1 register.
  – When CNT is re-initialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.

## 17.7.7 Status register 2 (TIMx_SR2)

Address offset: 0x06

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | Reserved | | CC2OF | CC1OF | Reserved |
| | | | | | rc_w0 | rc_w0 | |

Bits 7:3 Reserved

Bit 2 **CC2OF**: Capture/Compare 2 overcapture flag
  Refer to CC1OF description

Bit 1 **CC1OF**: Capture/Compare 1 overcapture flag
  This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.
  0: No overcapture has been detected.
  1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set

Bit 0 Reserved

### 17.7.8 Event generation register (TIMx_EGR)

Address offset: 0x07

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BG | TG | | Reserved | | CC2G | CC1G | UG |
| w | w | | | | w | w | w |

Bit 7 **BG**: Break generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action.

1: A break event is generated. MOE bit is cleared and BIF flag is set. An interrupt is generated if enabled by the BIE bit.

Bit 6 **TG**: Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action.

1: The TIF flag is set in TIMx_SR1 register. An interrupt is generated if enabled by the TIE bit.

Bits 5:3 Reserved

Bit 2 **CC2G**: Capture/Compare 2 generation

Refer to CC1G description

Bit 1 **CC1G**: Capture/Compare 1 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A capture/compare event is generated on channel 1:

– **If the CC1 channel is configured in output mode**:

CC1IF flag is set, and the corresponding interrupt request is sent if enabled.

– **If the CC1 channel configured in input mode**:

The current value of the counter is captured in the TIMx_CCR1 register. The CC1IF flag is set, and the corresponding interrupt request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

Bit 0 **UG**: Update generation

This bit can be set by software, it is 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.

### 17.7.9 Capture/compare mode register 1 (TIMx_CCMR1)

The channel can be used in input (capture mode) or in output (compare mode). The direction of the channel is defined by configuring the CC1S bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So be aware that the same bit can have a different meaning for the input stage and for the output stage.

Address offset: 0x08
Reset value: 0x00

#### Channel configured in output

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | OC1M[2:0] | | | OC1PE | OC1FE | CC1S[1:0] | |
| | rw | | | rw | rw | rw | |

Bit 7 Reserved

Bits 6:4 **OC1M**: Output compare 1 mode

These bits defines the behavior of the output reference signal OC1REF from which OC1 is derived. OC1REF is active high whereas OC1 active level depends on the CC1P bit.

000 : Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.

001 : Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx_CNT matchs the capture/compare register 1 (TIMx_CCR1).

010 : Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx_CNT matchs the capture/compare register 1 (TIMx_CCR1).

011 : Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.

100 : Force inactive level - OC1REF is forced low.

101 : Force active level - OC1REF is forced high.

110 : PWM mode 1 - In up-counting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In down-counting, channel 1 is inactive (OC1REF='0') as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF='1').

111 : PWM mode 2 - In up-counting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active.

*Note: In PWM mode 1 or 2, the OCiREF level changes only when the result of the comparison changes or when the output compare mode switches from "frozen" mode to "PWM" mode. Refer to Section 17.5.7 on page 162 for more details.*

Bit 3 **OC1PE**: Output compare 1 preload enable

0: Preload register on TIMx_CCR1 disabled. TIMx_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx_CCR1 enabled. Read/Write operations access the preload register. TIMx_CCR1 preload value is loaded in the shadow register at each update event.

*Note: For correct operation, preload registers must be enabled when the timer is in PWM mode. This is not mandatory in one pulse mode (OPM bit set in TIMx_CR1 register).*

Bit 2 **OC1FE**: Output compare 1 fast enable.

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently from the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OCFE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1FP1.

10: CC1 channel is configured as input, IC1 is mapped on TI2FP1.

11: Reserved

*Note: CC1S bits are writable only when the channel is OFF (CC1E= '0' in TIMx_CCER1 and updated).*

### Channel configured in input

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | IC1F[3:0] | | | IC1PSC[1:0] | | CC1S[1:0] | |
| | rw | | | rw | | rw | |

Bits 7:4 **IC1F**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

0000: No filter, sampling is done at $f_{MASTER}$

0001: $f_{SAMPLING}=f_{MASTER}$, N=2

0010: $f_{SAMPLING}=f_{MASTER}$, N=4

0011: $f_{SAMPLING}=f_{MASTER}$, N=8

0100: $f_{SAMPLING}=f_{MASTER}/2$, N=6

0101: $f_{SAMPLING}=f_{MASTER}/2$, N=8

0110: $f_{SAMPLING}=f_{MASTER}/4$, N=6

0111: $f_{SAMPLING}=f_{MASTER}/4$, N=8

1000: $f_{SAMPLING}=f_{MASTER}/8$, N=6

1001: $f_{SAMPLING}=f_{MASTER}/8$, N=8

1010: $f_{SAMPLING}=f_{MASTER}/16$, N=5

1011: $f_{SAMPLING}=f_{MASTER}/16$, N=6

1100: $f_{SAMPLING}=f_{MASTER}/16$, N=8

1101: $f_{SAMPLING}=f_{MASTER}/32$, N=5

1110: $f_{SAMPLING}=f_{MASTER}/32$, N=6

1111: $f_{SAMPLING}=f_{MASTER}/32$, N=8

Bits 3:2 **IC1PSC**: Input capture 1 prescaler

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).

The prescaler is reset as soon as CC1E='0' (TIMx_CCER register).

00: no prescaler, capture is done each time an edge is detected on the capture input.

01: Capture is done once every 2 events.

10: Capture is done once every 4 events.

11: Capture is done once every 8 events.

*Note: The internal event counter is not reset when IC1PSC is changed on the fly. In this case the old value is used until the next capture occurs. To force a new value to be taken in account immediately, you can clear the CC1E bit and set it again.*

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1FP1.

10: CC1 channel is configured as input, IC1 is mapped on TI2FP1.

11: Reserved

*Note: CC1S bits are writable only when the channel is OFF (CC1E= '0' in TIMx_CCER1 and updated).*

### 17.7.10 Capture/compare mode register 2 (TIMx_CCMR2)

*Note:* *Refer to Capture/compare mode register 1 (TIMx_CCMR1) for details on using these bits.*

Address offset: 0x09
Reset value: 0x00

#### Channel configured in output

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | OC2M[2:0] | | | OC2PE | OC2FE | CC2S[1:0] | |
| | rw | | | rw | rw | rw | |

Bit 7 Reserved

Bits 6:4 **OC2M**: Output compare 2 mode

Bit 3 **OC2PE**: Output compare 2 preload enable

Bit 2 **OC2FE**: Output compare 2 fast enable

Bits 1:0 **CC2S**: Capture/Compare 2 Selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output.

01: CC2 channel is configured as input, IC2 is mapped on TI2FP2.

10: CC2 channel is configured as input, IC2 is mapped on TI1FP2.

11: Reserved

*Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx_CCER1).*

### Channel configured in input

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IC2F[3:0] | | | | IC2PSC[1:0] | | CC2S[1:0] | |
| rw | | | | rw | | rw | |

Bits 7:4 **IC2F**: Input capture 2 filter

Bits 3:2 **IC2PSC**: Input capture 2 prescaler

Bits 1:0 **CC2S**: Capture/Compare 2 selection
This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC2 channel is configured as output.
01: CC2 channel is configured as input, IC2 is mapped on TI2FP2.
10: CC2 channel is configured as input, IC2 is mapped on TI1FP2.
11: Reserved
*Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx_CCER1).*

## 17.7.11 Capture/compare enable register 1 (TIMx_CCER1)

Address offset: 0x0A

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | CC2P | CC2E | Reserved | | CC1P | CC1E |
| | | rw | rw | | | rw | rw |

Bits 6:7 Reserved

Bit 5 **CC2P**: Capture/Compare 2 output polarity
refer to CC1P description

Bit 4 **CC2E**: Capture/Compare 2 output enable
refer to CC1E description

Bits 2:3 Reserved

Bit 1 **CC1P**: Capture/Compare 1 output polarity
CC1 channel configured as output:
0 : OC1 active high
1 : OC1 active low
**CC1 channel configured as input for capture function** (see *Figure 60*):
0 : Capture is done on a rising edge of TI1F or TI2F
1 : Capture is done on a falling edge of TI1F or TI2F

Bit 0 **CC1E**: Capture/Compare 1 output enable
CC1 channel configured as output:
0 : Off - OC1 is not active.
1 : On - OC1 signal is output on the corresponding output pin.
**CC1 channel configured as input**:
In this case this bit determines if a capture of the counter value can actually be done into the input
capture/compare register 1 (TIMx_CCR1) or not.
0 : Capture disabled.
1 : Capture enabled.

### 17.7.12    Counter high (TIMx_CNTRH)

Address offset: 0x0B

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | CNT[15:8] | | | | |
| | | | rw | | | | |
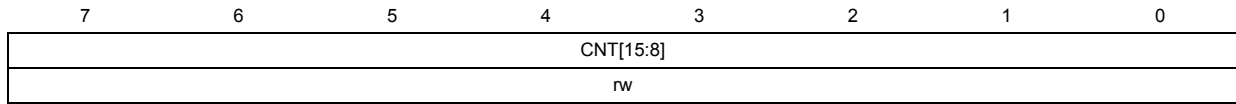
Bits 7:0  **CNT[15:8]**: Counter value (MSB)

### 17.7.13    Counter low (TIMx_CNTRL)

Address offset: 0x0C

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | CNT[7:0] | | | | |
| | | | rw | | | | |
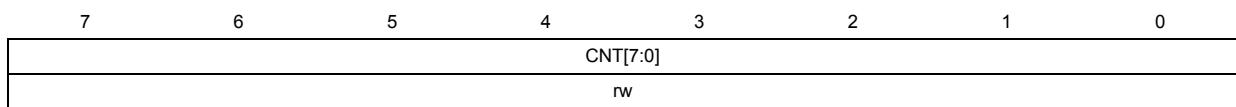
Bits 7:0  **CNT[7:0]**: Counter value (LSB)

### 17.7.14    Prescaler register (TIMx_PSCR)

Address offset: 0x0D

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | Reserved | | | | PSC[2:0] | |
| | | | | | | rw | |
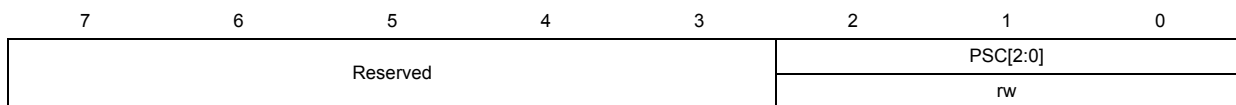
Bits 7:3  Reserved

Bits 2:0  **PSC[2:0]**: Prescaler value

The prescaler value divides the CK_PSC clock frequency.
The counter clock frequency $f_{CK\_CNT}$ is equal to $f_{CK\_PSC}$ / $2^{(PSC[2:0])}$. PSC[7:3] are forced to 0 by hardware.
PSCR contains the value which will be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register).
This means that an update event must be generated in order that a new prescaler value can be taken into account.

### 17.7.15 Auto-reload register high (TIMx_ARRH)

Address offset: 0x0E

Reset value: 0xFF

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | ARR[15:8] | | | | |
| | | | rw | | | | |
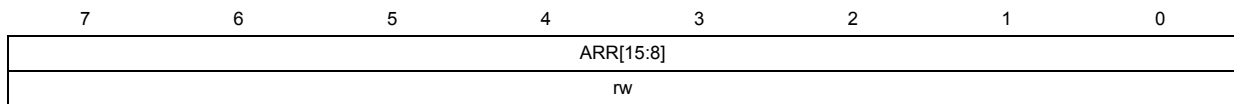
Bits 7:0 **ARR[15:8]**: Autoreload value (MSB)

ARR is the value to be loaded in the actual auto-reload register.

Refer to the *Section 17.3: TIMx time base unit on page 135* for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is 0.

### 17.7.16 Auto-reload register low (TIMx_ARRL)

Address offset: 0x0F

Reset value: 0xFF

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | ARR[7:0] | | | | |
| | | | rw | | | | |
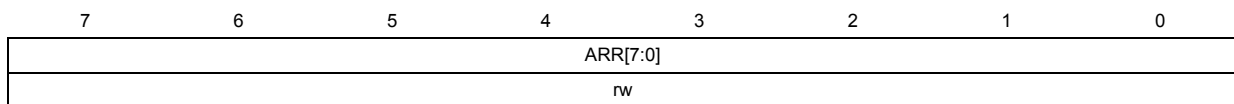
Bits 7:0 **ARR[7:0]**: Autoreload value (LSB)

### 17.7.17 Capture/compare register 1 high (TIMx_CCR1H)

Address offset: 0x10

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | CCR1[15:8] | | | | |
| | | | rw | | | | |
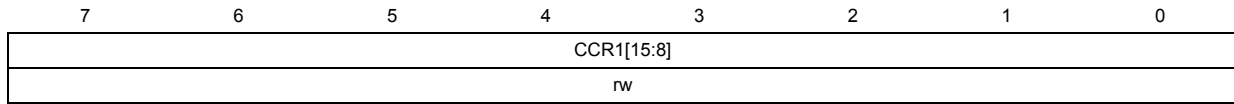
Bits 7:0   **CCR1[15:8]**: Capture/Compare 1 value (MSB)

         **If the CC1 channel is configured as output (CC1S bits in TIMx_CCMR1 register)**:

         CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

         It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

         The active Capture/Compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC1 output.

         **If the CC1 channel is configured as input (CC1S bits in TIMx_CCMR1 register)**:

         CCR1 is the counter value transferred by the last input capture 1 event (IC1). It is read-only in this case.

### 17.7.18 Capture/compare register 1 low (TIMx_CCR1L)

Address offset: 0x11
Reset value: 0x00
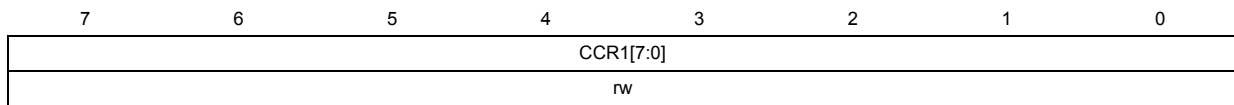
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | CCR1[7:0] | | | | |
| | | | rw | | | | |

Bits 7:0   **CCR1[7:0]**: Capture/Compare 1 value (LSB)

### 17.7.19 Capture/compare register 2 high (TIMx_CCR2H)

Address offset: 0x12

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | CCR2[15:8] | | | | |
| | | | rw | | | | |

Bits 7:0 **CCR2[15:8]**: Capture/Compare 2 value (MSB)

**If the CC2 channel is configured as output (CC2S bits in TIMx_CCMR2 register)**:

CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR2 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC2 output.

**If the CC2 channel is configured as input (CC2S bits in TIMx_CCMR2 register)**:

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

### 17.7.20 Capture/compare register 2 low (TIMx_CCR2L)

Address offset: 0x13

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | CCR2[7:0] | | | | |
| | | | rw | | | | |

Bits 7:0 **CCR2[7:0]**: Capture/Compare value (LSB)

## 17.7.21 Break register (TIMx_BKR)

Address offset: 0x14

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| MOE | AOE | BKP | BKE | Reserved | OSSI | LOCK | |
| rw | rw | rw | rw | | rw | rw | |

Bit 7 **MOE**: Main output enable

This bit is cleared asynchronously by hardware as soon as the break input is active. It is set by software or automatically depending on the AOE bit. It is acting only on the channels which are configured in output.

0: OC outputs are disabled or forced to idle state.

1: OC outputs are enabled if their respective enable bits are set (CCxE in TIMx_CCERx registers).

See OC enable description for more details (*Table 35: Output control bit for OCx channels with break feature on page 188*).

Bit 6 **AOE**: Automatic output enable

0: MOE can be set only by software

1: MOE can be set by software or automatically at the next update event (if the break input is not be active)

*Note: This bit can no longer be modified as long as LOCK level 1 has been programmed (LOCK bits in the TIMx_BKR register).*

Bit 5 **BKP**: Break polarity

0: Break input BKIN is active low

1: Break input BKIN is active high

*Note: This bit can no longer be modified as long as LOCK level 1 has been programmed (LOCK bits in the TIMx_BKR register).*

Bit 4 **BKE**: Break enable

0: Break input (BKIN) disabled

1: Break input (BKIN) enabled

*Note: This bit can no longer be modified as long as LOCK level 1 has been programmed (LOCK bits in the TIMx_BKR register).*

Bit 3 Reserved

Bit 2 **OSSI**: Off-State selection for idle mode

This bit is used when MOE=0 on channels configured as outputs.

See OC enable description for more details (*Table 35: Output control bit for OCx channels with break feature on page 188*).

0: When inactive, OCx outputs are disabled (OCx enable output signal=0).

1: When inactive, OCx outputs are forced first with their idle level as soon as CCxE=1. OC enable output signal=1).

*Note: This bit can no longer be modified as soon as the LOCK level 2 has been programmed (LOCK bits in the TIMx_BKR register).*

Bits 1:0 **LOCK**: Lock configuration

These bits offer a write protection against software errors.

00: LOCK OFF - No bits are write protected.

01: LOCK Level 1 = OISx bit in TIMx_OISR register and BKE/BKP/AOE bits in TIMx_BKR register can no longer be written.

10: LOCK Level 2 = LOCK Level 1 + CC Polarity bits (CCxP bits in TIMx_CCERx registers, as long as the related channel is configured in output through the CCxS bits) as well as the OSSR and OSSI bits can no longer be written.

11: LOCK Level 3 = LOCK Level 2 + CC Control bits (OCxM and OCxPE bits in TIMx_CCMRx registers, as long as the related channel is configured in output through the CCxS bits) can no longer be written.

*Note: The LOCK bits can be written only once after the reset. Once the TIMx_BKR register has been written, their content is frozen until the next reset.*

*Note:* *As the bits AOE, BKP, BKE and OSSI can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BKR register.*

**Table 35. Output control bit for OCx channels with break feature**

| Control bits | | | OCx/OCx_EN output state |
|---|---|---|---|
| **MOE bit** | **OSSI bit** | **CCxE bit** | |
| 1 | X | 0 | Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0 |
| | | 1 | OCx= OCxREF + Polarity ( OCxREF xor CCxP) OCx_EN=1 |
| 0 | 0 | 0 | Output Disabled (not driven by the timer) OCx=OISx, OCx_EN=0 |
| | 0 | 1 | |
| | 1 | 0 | |
| | 1 | 1 | Off-State (output enabled with inactive state) OCx=OISx, OCx_EN=1 |

*Note:* *The state of the external I/O pins connected to the OCx channels depends on the OCx channel state and the GPIO registers.*

## 17.7.22 Output idle state register (TIMx_OISR)

Address offset: 0x15

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | Reserved | | OIS2 | Reserved | OIS1 |
| | | | | | rw | | rw |

Bits 7:3 Reserved

Bit 2 **OIS2**: Output idle state 2 (OC2 output).
Refer to OIS1 bit

Bit 1 Reserved

Bit 0 **OIS1**: Output idle state 1 (OC1 output).
0: OC1=0 when MOE=0
1: OC1=1 when MOE=0

*Note: This bit can no longer be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in the TIMx_BKR register).*

## 17.7.23 TIMx register map and reset values

Refer to the datasheet for the base addresses of TIM2 and TIM3.

**Table 36. TIMx register map**

| Address offset | Register Name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | TIMx_CR1<br>Reset value | ARPE<br>0 | CMS1<br>0 | CMS0<br>0 | DIR<br>0 | OPM<br>0 | URS<br>0 | UDIS<br>0 | CEN<br>0 |
| 0x01 | TIMx_CR2<br>Reset value | TI1S<br>0 | MMS2<br>0 | MMS1<br>0 | MMS0<br>0 | -<br>0 | -<br>0 | -<br>0 | -<br>0 |
| 0x02 | TIMx_SMCR<br>Reset value | MSM<br>0 | TS2<br>0 | TS1<br>0 | TS0<br>0 | -<br>0 | SMS2<br>0 | SMS1<br>0 | SMS0<br>0 |
| 0x03 | TIMx_ETR<br>Reset value | ETP<br>0 | ECE<br>0 | ETPS1<br>0 | ETPS0<br>0 | EFT3<br>0 | EFT2<br>0 | EFT1<br>0 | EFT0<br>0 |
| 0x04 | TIMx_IER<br>Reset value | BIE<br>0 | TIE<br>0 | -<br>0 | -<br>0 | -<br>0 | CC2IE<br>0 | CC1IE<br>0 | UIE<br>0 |
| 0x05 | TIMx_SR1<br>Reset value | BIF<br>0 | TIF<br>0 | -<br>0 | -<br>0 | -<br>0 | CC2IF<br>0 | CC1IF<br>0 | UIF<br>0 |
| 0x06 | TIMx_SR2<br>Reset value | -<br>0 | -<br>0 | -<br>0 | -<br>0 | -<br>0 | CC2OF<br>0 | CC1OF<br>0 | -<br>0 |
| 0x07 | TIMx_EGR<br>Reset value | BG<br>0 | TG<br>0 | -<br>0 | -<br>0 | -<br>0 | CC2G<br>0 | CC1G<br>0 | UG<br>0 |

**Table 36. TIMx register map (continued)**

| Address offset | Register Name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x08 | TIMx_CCMR1 (output mode) Reset Value | - 0 | OC1M2 0 | OC1M1 0 | OC1M0 0 | OC1PE 0 | OC1FE 0 | CC1S1 0 | CC1S0 0 |
| | TIMx_CCMR1 (input mode) Reset value | IC1F3 0 | IC1F2 0 | IC1F1 0 | IC1F0 0 | IC1PSC1 0 | IC1PSC0 0 | CC1S1 0 | CC1S0 0 |
| 0x09 | TIMx_ CCMR2 (output mode) Reset value | - 0 | OC2M2 0 | OC2M1 0 | OC2M0 0 | OC2PE 0 | OC2FE 0 | CC2S1 0 | CC2S0 0 |
| | TIMx_CCMR2 (input mode) Reset value | IC2F3 0 | IC2F2 0 | IC2F1 0 | IC2F0 0 | IC2PSC1 0 | IC2PSC0 0 | CC2S1 0 | CC2S0 0 |
| 0x0A | TIMx_CCER1 Reset value | - 0 | - 0 | CC2P 0 | CC2E 0 | - 0 | - 0 | CC1P 0 | CC1E 0 |
| 0x0B | TIMx_CNTRH Reset value | CNT15 0 | CNT14 0 | CNT13 0 | CNT12 0 | CNT11 0 | CNT10 0 | CNT9 0 | CNT8 0 |
| 0x0C | TIMx_CNTRL Reset value | CNT7 0 | CNT6 0 | CNT5 0 | CNT4 0 | CNT3 0 | CNT2 0 | CNT1 0 | CNT0 0 |
| 0x0D | TIMx_PSCR Reset value | - 0 | - 0 | - 0 | - 0 | - 0 | PSC2 0 | PSC1 0 | PSC0 0 |
| 0x0E | TIMx_ARRH Reset value | ARR15 1 | ARR14 1 | ARR13 1 | ARR12 1 | ARR11 1 | ARR10 1 | ARR9 1 | ARR8 1 |
| 0x0F | TIMx_ARRL Reset value | ARR7 1 | ARR6 1 | ARR5 1 | ARR4 1 | ARR3 1 | ARR2 1 | ARR1 1 | ARR0 1 |
| 0x10 | TIMx_CCR1H Reset value | CCR115 0 | CCR114 0 | CCR113 0 | CCR112 0 | CCR111 0 | CCR110 0 | CCR19 0 | CCR18 0 |
| 0x11 | TIMx_CCR1L Reset value | CCR17 0 | CCR16 0 | CCR15 0 | CCR14 0 | CCR13 0 | CCR12 0 | CCR11 0 | CCR10 0 |
| 0x12 | TIMx_CCR2H Reset value | CCR215 0 | CCR214 0 | CCR213 0 | CCR212 0 | CCR211 0 | CCR210 0 | CCR29 0 | CCR28 0 |
| 0x13 | TIMx_CCR2L Reset value | CCR27 0 | CCR26 0 | CCR25 0 | CCR24 0 | CCR23 0 | CCR22 0 | CCR21 0 | CCR20 0 |
| 0x14 | TIMx_BKR Reset value | MOE 0 | AOE 0 | BKP 0 | BKE 0 | OSSR 0 | OSSI 0 | LOCK 0 | LOCK 0 |
| 0x15 | TIMx_OISR Reset value | - 0 | - 0 | - 0 | - 0 | - 0 | OIS2 0 | - 0 | OIS1 0 |

# 18 8-bit basic timer (TIM4)

## 18.1 Introduction

The timer consists of an 8-bit auto-reload up-counter driven by a programmable prescaler. It can be used for time base generation, with interrupt generation on timer overflow.

Refer to *Section 17.3 on page 135* for the general description of the timer features.

**Figure 73. TIM4 block diagram**
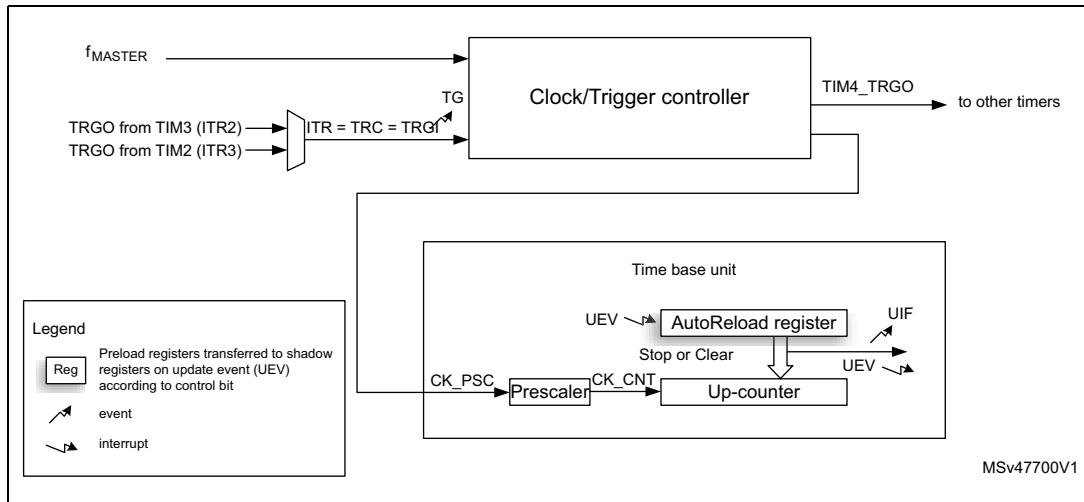


## 18.2 TIM4 main features

The main features include:

- 8-bit up counter auto-reload counter
- 4-bit programmable prescaler allowing dividing (also "on the fly") the counter clock frequency by any power of 2 from 1 to 32768.
- Interrupt generation
  - On counter update: counter overflow
  - On trigger input

## 18.3 TIM4 interrupts

The timer has 2 interrupt request sources:

- Update Interrupt (overflow, counter initialization)
- Trigger input

## 18.4 TIM4 clock selection

The clock source for the timer is the internal clock ($f_{MASTER}$). It is connected directly to the CK_PSC clock that feeds the prescaler driving the counter clock CK_CNT.

### Prescaler

The prescaler implementation is as follows:

- The TIM4 prescaler is based on a 16-bit counter controlled through a 4-bit register (in TIM4_PSCR register). It can be changed on the fly as this control register is buffered. It can divide the counter clock frequency by any power of 2 from 1 to 32768.

  The counter clock frequency is calculated as follows:

  $f_{CK\_CNT} = f_{CK\_PSC}/2^{(PSCR[3:0])}$

The prescaler value is loaded through a preload register. The shadow register, which contains the current value to be used is loaded as soon as the LS Byte has been written.

Read operations to the TIM4_PSCR registers access the preload registers, so no special care needs to be taken to read them.

# 18.5 TIM4 registers

## 18.5.1 Control register 1 (TIM4_CR1)

Address offset: 0x00

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ARPE | | Reserved | | OPM | URS | UDIS | CEN |
| rw | | | | rw | rw | rw | rw |

Bit 7　**ARPE**: Auto-Reload Preload Enable.

　　0: TIM4_ARR register is not buffered through a preload register. It can be written directly.
　　1: TIM4_ARR register is buffered through a preload register.

Bits 6:4　Reserved

Bit 3　**OPM**: One Pulse Mode.

　　0: Counter is not stopped at update event
　　1: Counter stops counting at the next update event (clearing the CEN bit ).

Bit 2　**URS**: Update Request Source.

　　0: When enabled, an update interrupt request is sent as soon as registers are updated (counter overflow)
　　1: When enabled, an update interrupt request is sent only when the counter reaches the overflow/underflow.

Bit 1　**UDIS**: Update disable.

　　0: An Update event is generated as soon as a counter overflow occurs or a software update is generated. Buffered registers are then loaded with their preload values
　　1: An Update event is not generated, shadow registers keep their value (ARR, PSC). The counter and the prescaler are re-initialized if the UG bit is set.

Bit 0　**CEN**: Counter enable.

　　0: Counter disable.
　　1: Counter enable.

## 18.5.2 Control register 2 (TIM4_CR2)

Address offset: 0x01

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | MMS[2:0] | | | Reserved | | | |
| | rw | | | | | | |

Bit 7 Reserved

Bits 6:4 **MMS**: Master mode selection

These bits select the information to be sent in master mode to other timers for synchronization (TRGO). The combination is as follows:

000: **Reset** - the UG bit in the TIM4_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (clock/trigger mode controller configured in trigger reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter Enable signal is used as trigger output (TRGO). It is used to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode. When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in the TIM4_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO).

011: Reserved

100: Reserved

101: Reserved

110: Reserved

111: Reserved

Bits 3:0 Reserved

## 18.5.3    Slave mode control register (TIM4_SMCR)

Address offset: 0x02

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| MSM | TS[2:0] | | | Reserved | SMS[2:0] | | |
| rw | rw | | | | rw | | |

Bit 7 **MSM**: Master/Slave mode

0: No action

1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between timers (through TRGO).

Bits 6:4 **TS[2:0]**: Trigger selection

This bit-field selects the trigger input to be used to synchronize the counter.

000: Reserved

001: Reserved

010: Internal trigger ITR2 connected to TIM3 TRGO

011: Internal trigger ITR3 connected to TIM2 TRGO

100: Reserved

101: Reserved

110: Reserved

111: Reserved

*Note:   These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.*

Bit 3 Reserved

Bits 2:0 **SMS[2:0]**: Clock/trigger/slave mode selection

When external signals are selected, the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

000: Clock/trigger controller disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.

001: Reserved

010: Reserved

011: Reserved

100: Trigger reset mode - Rising edge of the selected trigger signal (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger signal (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

### 18.5.4 Interrupt enable register (TIM4_IER)

Address offset: 0x03

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | TIE | Reserved | | | | | UIE |
| | rw | | | | | | rw |

Bit 7 Reserved

Bit 6 **TIE**: Trigger interrupt enable
   0: Trigger interrupt disabled
   1: Trigger interrupt enabled

Bits 5:1 Reserved

Bit 0 **UIE**: Update Interrupt Enable
   0: Update Interrupt disabled
   1: Update Interrupt enabled

### 18.5.5 Status register 1 (TIM4_SR1)

Address offset: 0x04

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | TIF | Reserved | | | | | UIF |
| | rc_w0 | | | | | | rc_w0 |

Bit 7 Reserved

Bit 6 **TIF**: Trigger interrupt flag
   This flag is set by hardware on trigger event (active edge detected on TRGI signal, both edges in case gated mode is selected). It is cleared by software.
   0: No trigger event occurred.
   1: Trigger interrupt pending.

Bits 5:1 Reserved

Bit 0 **UIF**: Update interrupt flag
   This bit is set by hardware on an update event. It is cleared by software.
   0: No update occurred.
   1: Update interrupt pending. This bit is set by hardware when the registers are updated:
   –   at overflow if UDIS=0 in the TIM4_CR1 register.
   –   when CNT is re-initialized by software using the UG bit in the TIM4_EGR register, if URS=0 and UDIS=0 in the TIM4_CR1 register.

### 18.5.6 Event generation register (TIM4_EGR)

Address offset: 0x05

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | TG | Reserved | | | | | UG |
|  | w |  |  |  |  |  | w |

Bit 7  Reserved

Bit 6  **TG**: Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.
0: No action
1: The TIF flag is set in TIM4_SR1 register. An interrupt is generated if enabled by the TIE bit.

Bits 5:1  Reserved

Bit 0  **UG**: Update generation

This bit can be set by software, it is 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.

### 18.5.7 Counter (TIM4_CNTR)

Address offset: 0x06

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CNT[7:0] | | | | | | | |
| rw | | | | | | | |

Bits 7:0  CNT[7:0]: Counter value

### 18.5.8 Prescaler register (TIM4_PSCR)

Address offset: 0x07

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | PSC[3:0] | | | |
|  |  |  |  | rw | | | |

Bits 7:4  Reserved

Bits 3:0  **PSC[3:0]**: Prescaler value

The prescaler value divides the CK_PSC clock frequency.
The counter clock frequency $f_{CK\_CNT}$ is equal to $f_{CK\_PSC} / 2^{(PSC[3:0])}$.
PSC contains the value which will be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIM4_EGR register).
This means that an update event must be generated in order that a new prescaler value can be taken into account.

### 18.5.9 Auto-reload register (TIM4_ARR)

Address offset: 0x08

Reset value: 0xFF
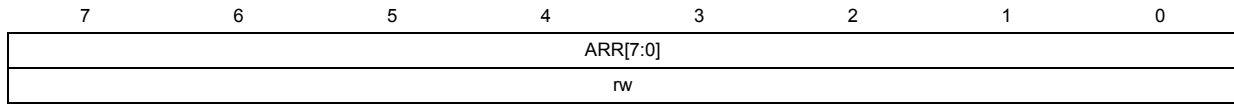
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | ARR[7:0] | | | | |
| | | | rw | | | | |

Bits 7:0 ARR[7:0]: Autoreload Value

### 18.5.10 TIM4 register map and reset values

Refer to the datasheet for the base address.

**Table 37. TIM4 register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | TIM4_CR1 Reset value | ARPE 0 | - 0 | - 0 | - 0 | OPM 0 | URS 0 | UDIS 0 | CEN 0 |
| 0x01 | TIM4_CR2 Reset value | - 0 | MMS2 0 | MMS1 0 | MMS0 0 | - 0 | - 0 | - 0 | - 0 |
| 0x02 | TIM4_SMCR Reset value | MSM 0 | TS2 0 | TS1 0 | TS0 0 | - 0 | SMS2 0 | SMS1 0 | SMS0 0 |
| 0x03 | TIM4_IER Reset value | - 0 | TIE 0 | - 0 | - 0 | - 0 | - 0 | - 0 | UIE 0 |
| 0x04 | TIM4_SR1 Reset value | - 0 | TIF 0 | - 0 | - 0 | - 0 | - 0 | - 0 | UIF 0 |
| 0x05 | TIM4_EGR Reset value | - 0 | TG 0 | - 0 | - 0 | - 0 | - 0 | - 0 | UG 0 |
| 0x06 | TIM4_CNTR Reset value | CNT7 0 | CNT6 0 | CNT5 0 | CNT4 0 | CNT3 0 | CNT2 0 | CNT1 0 | CNT0 0 |
| 0x07 | TIM4_PSCR Reset value | - 0 | - 0 | - 0 | - 0 | PSC3 0 | PSC2 0 | PSC1 0 | PSC0 0 |
| 0x08 | TIM4_ARR Reset value | ARR7 1 | ARR6 1 | ARR5 1 | ARR4 1 | ARR3 1 | ARR2 1 | ARR1 1 | ARR0 1 |

# 19      Serial peripheral interface (SPI)

## 19.1      SPI introduction

The serial peripheral interface (SPI) allows half/ full duplex, synchronous, serial communication with external devices. The interface can be configured as the master and in this case it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multi-master configuration.

## 19.2      SPI main features

- Full duplex synchronous transfers (on 3 lines)
- Simplex synchronous transfers on 2 lines with or without a bidirectional data line
- Master or slave operation
- 8 Master mode frequencies ($f_{MASTER}/2$ max.)
- Slave mode frequency ($f_{MASTER}/2$ max.)
- Faster communication - Maximum SPI speed: 10 MHz
- NSS management by hardware or software for both master and slave
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- Master mode fault and overrun flags with interrupt capability
- Wakeup capability:
  The MCU wakes up from Low power mode in full or half duplex transmit-only modes

## 19.3　SPI functional description

### 19.3.1　SPI general description

The block diagram of the SPI is shown in *Figure 74*.

**Figure 74. SPI block diagram**
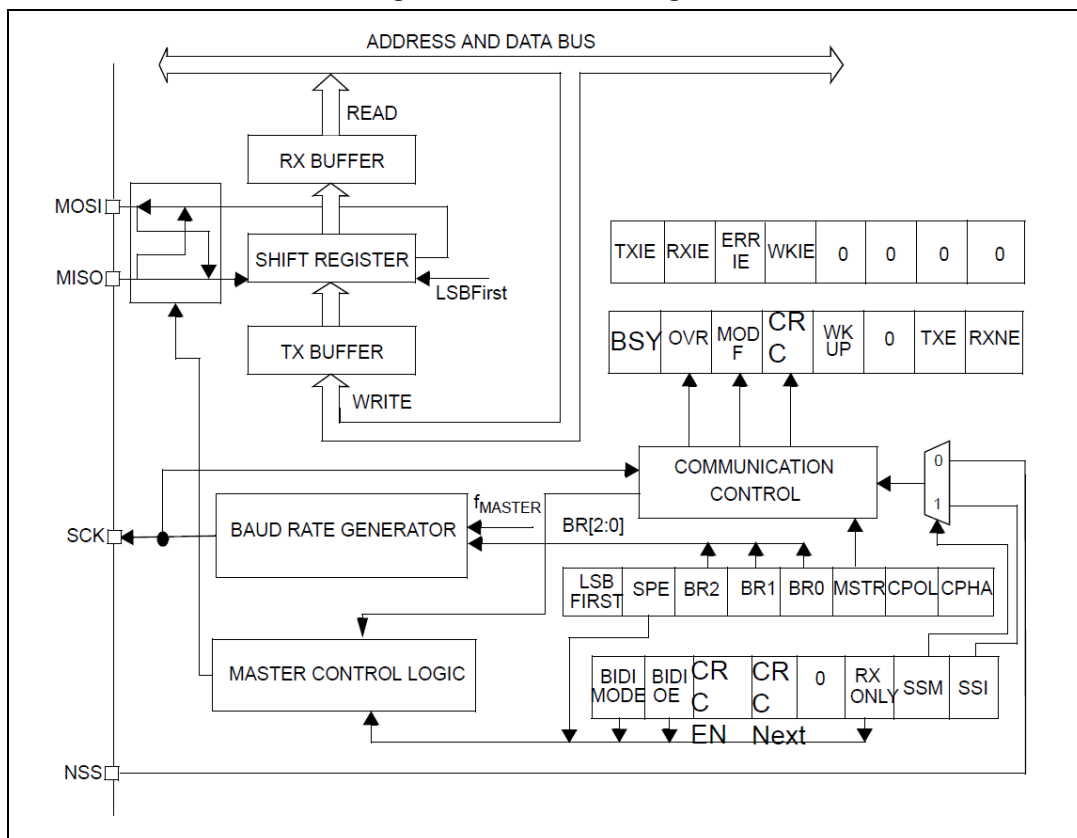


The SPI is connected to external devices through four pins:

- MISO: Master In / Slave Out data (port C7). This pin can be used to transmit data in slave mode and receive data in master mode.

- MOSI: Master Out / Slave In data (port C6). This pin can be used to transmit data in master mode and receive data in slave mode.

- SCK: Serial Clock output (port C5) for SPI masters and Serial Clock input for SPI slaves.

- NSS: Slave select (port E5). This is a optional pin to select a slave device. This pin acts as a 'chip select' to let the SPI master communicate with slaves individually and to avoid contention on the data lines. Slave NSS inputs can be driven by standard I/O ports on the master device. When configured in master mode (MSTR bit =1) and if NSS is pulled low, the SPI enters master mode fault state: the MSTR bit is automatically reset and the device is configured in slave mode (refer to *Section 19.3.8: Error flags on page 215*).

A basic example of interconnections between a single master and a single slave is illustrated in *Figure 75*.

*Note:*        *When using the SPI in High-speed mode, the I/Os where SPI outputs are connected should be programmed as fast slope outputs in order to be able to reach the expected bus speed.*

**Figure 75. Single master/ single slave application**
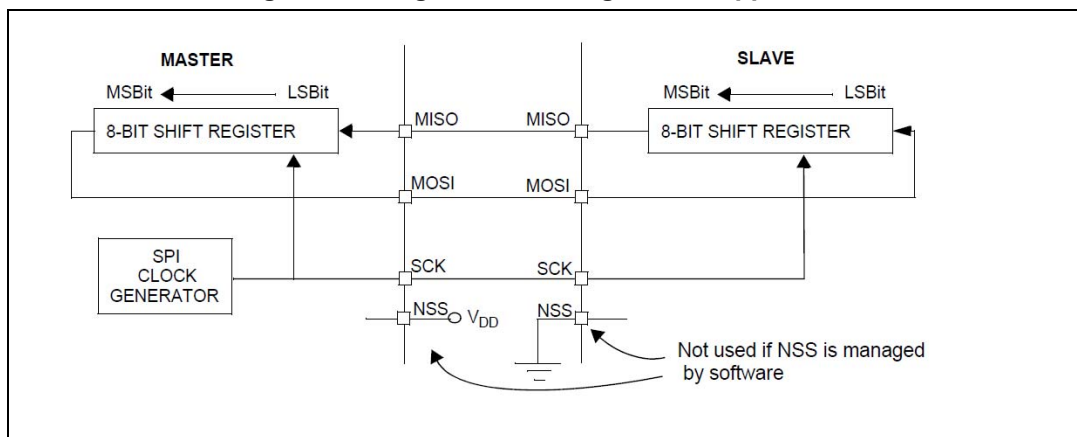


The MOSI pins are connected together and the MISO pins are connected together. In this way data is transferred serially between master and slave (most significant bit first).

The communication is always initiated by the master. When the master device transmits data to a slave device via MOSI pin, the slave device responds the MISO pin. This implies full duplex communication with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).

### Slave select (NSS) pin management

A hardware or software slave select management configuration can be set using the Software slave select management (SSM) bit from the SPI_CR2 register.

- **Software NSS management (SSM = 1):** with this configuration, slave select information is driven internally by the Internal slave select (SSI) bit value in the SPI_CR2 register.The external NSS pin remains free for other application uses.

- **Hardware NSS management (SSM = 0):** For devices set as master, this configuration allows multimaster capability. For devices set as slave, the NSS pin works as a classical NSS input. The slave is selected when the NSS line is in low level and is not selected if the NSS line is in high level.

*Note:*        *When the master is communicating with SPI slaves which need to be deselected between transmissions, the NSS pin must be configured as a GPIO.*

### Clock phase and clock polarity

Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits. The CPOL (clock polarity) bit controls the steady state value of the clock when no data is being transferred. This bit affects both master and slave modes. If CPOL is reset, SCK pin has a low level idle state. If CPOL is set, SCK pin has a high level idle state.

*Note:* *Make sure the SPI pin is configured at the idle state level of the SPI in order to avoid generating an edge on the SPI clock pin when enabling or disabling the SPI cell.*

If CPHA (clock phase) bit is set, the second edge on the SCK pin (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set) is the MSBit capture strobe. Data is latched on the occurrence of the first clock transition. If CPHA bit is reset, the first edge on the SCK pin (falling edge if CPOL bit is set, rising edge if CPOL bit is reset) is the MSBit capture strobe. Data is latched on the occurrence of the second clock transition.

The combination of the CPOL clock polarity and CPHA (clock phase) bits selects the data capture clock edge.

*Figure 76* shows an SPI transfer with the four combinations of the CPHA and CPOL bits. The diagram may be interpreted as a master or slave timing diagram where the SCK pin, the MISO pin, the MOSI pin are directly connected between the master and the slave device.

*Note:* *1 Prior to changing the CPOL/CPHA bits the SPI must be disabled by resetting the SPE bit.*

*2 Master and slave must be programmed with the same timing mode.*

*3 The idle state of SCK must correspond to the polarity selected in the SPI_CR1 register (by pulling up SCK if CPOL=1 or pulling down SCK if CPOL=0).*

**Figure 76. Data clock timing diagram**
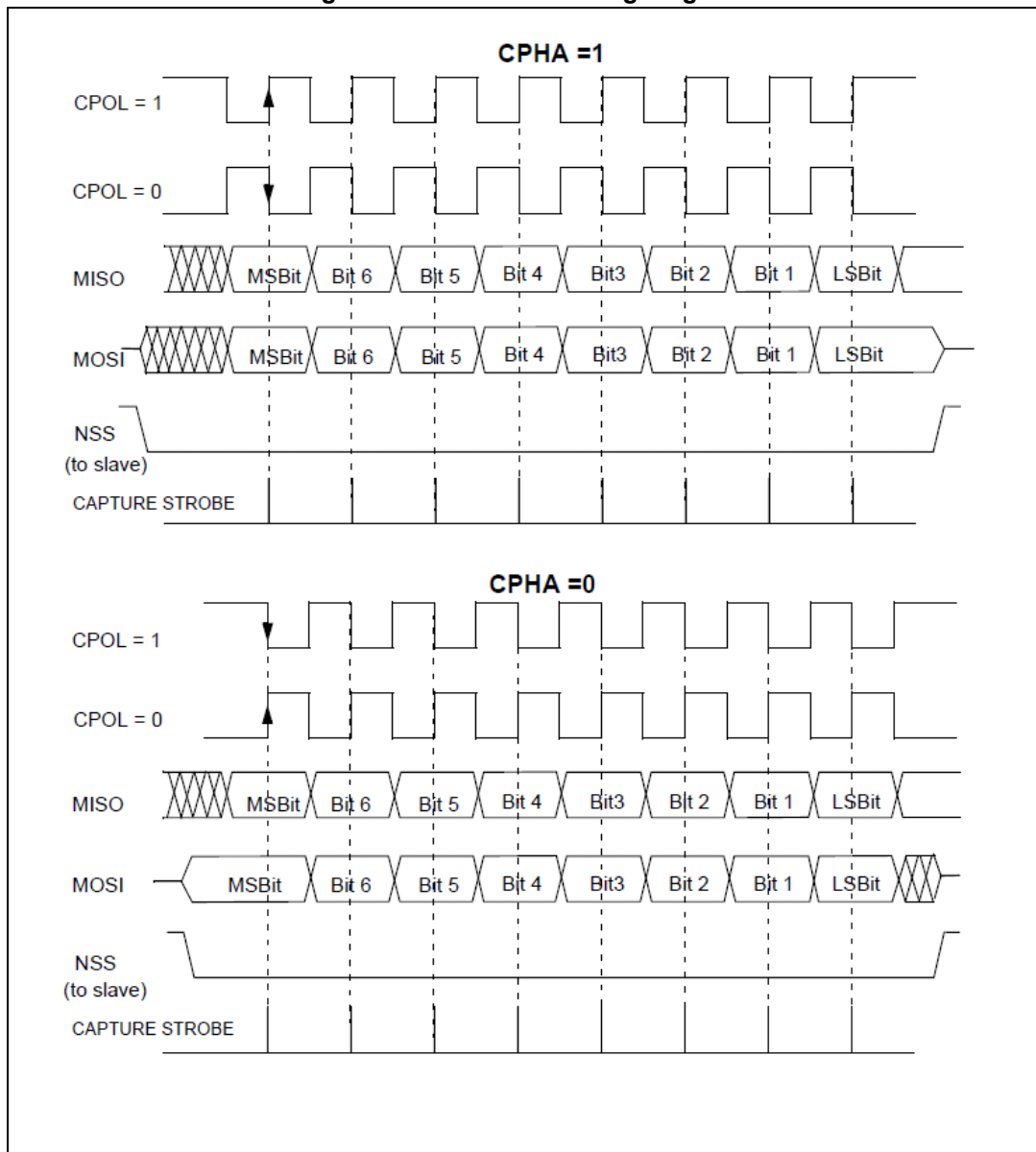


1.   These timings are shown with the LSBFIRST bit reset in the SPI_CR1 register.

## Frame format

Data can be shifted out either MSB-first or LSB-first depending on the value of the
LSBFIRST bit in the SPI_CR1 Register.

### 19.3.2 Configuring the SPI in slave mode

In slave configuration, the serial clock is received on the SCK pin from the master device. The value set in the BR[2:0] bits in the SPI_CR1 register, does not affect the data transfer rate.

Follow the procedure below to configure the SPI in slave mode:

1. Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock (see *Figure 76*). For correct data transfer, the CPOL and CPHA bits must be configured in the same way in the slave device and the master device.

2. The frame format (MSB-first or LSB-first depending on the value of the LSBFIRST bit in the SPI_CR1 register) must be the same as the master device.

3. In Hardware mode (refer to *Slave select (NSS) pin management on page 201*), the NSS pin must be connected to a low level signal during the complete data transmit sequence. In NSS Software mode, set the SSM bit and clear the SSI bit in the SPI_CR2 register.

4. Clear the MSTR bit and set the SPE bit to assign the pins to alternate functions.

In this configuration the MOSI pin is a data input and the MISO pin is a data output.

*Note:* *In applications with a parallel multi-slave structure, with separate NSS signals and the slave MISO outputs connected together, the corresponding GPIO registers must be configured correctly. The SPI_MISO pin is controlled by the SPI peripheral only when the NSS signal is active and the device is selected as slave. When the NSS signal is released, the pin is driven by GPIO register settings only. To function correctly, the GPIO has to be configured in input pull-up mode with no interrupt. This configuration is done using the GPIO_DDR, GPIO_CR1 and GPIO_CR2 registers - see Section 10.8.1: Alternate function output.*

### 19.3.3 Configuring the SPI master mode

In a master configuration, the serial clock is generated on the SCK pin.

Follow the procedure below to configure the SPI in master mode:

1. Select the BR[2:0] bits to define the serial clock baud rate (see SPI_CR1 register).

2. Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock (see *Figure 76*).

3. Configure the LSBFIRST bit in the SPI_CR1 register to define the frame format.

4. In Hardware mode, connect the NSS pin to a high-level signal during the complete data transmit sequence. In software mode, set the SSM and SSI bits in the SPI_CR2 register.

5. Set the MSTR and SPE bits (they remain set only if the NSS pin is connected to a high-level signal).

In this configuration the MOSI pin is a data output and to the MISO pin is a data input.

### 19.3.4 Configuring the SPI for simplex communications

The SPI is capable of operating in simplex mode in 2 configurations.

- 1 clock and 1 bidirectional data wire
- 1 clock and 1 data wire (Receive-only or Transmit-only)

**1 clock and 1 bidirectional data wire**

This mode is enabled by setting the BDM bit in the SPI_CR2 register. In this mode SCK is used for the clock, and MOSI in master or MISO in slave mode is used for data communication. The transfer direction (Input/output) is selected by the BDOE bit in the SPI_CR2 register. When this bit is set to 1, the data line is output, otherwise it is input.

**1 clock and 1 unidirectional data wire (BDM = 0)**

In this mode, the application can use the SPI either in transmit-only mode or in receive-only mode:

- Transmit-only mode is similar to full-duplex mode (BDM = 0, RXONLY = 0): the data is transmitted to the transmit pin (MOSI in master mode or MISO in slave mode) and the receive pin (MISO in master mode or MOSI in slave mode) can be used as general purpose I/O. In this case, the application just needs to ignore the Rx buffer (if the data register is read, it does not contain the received value).
- In receive-only mode, the application can disable the SPI output function by setting the RXONLY bit in the SPI_CR2 register. In this case, it frees the transmit I/O pin (MOSI in master mode or MISO in slave mode) so it can be used for other purposes.

To start the communication in receive-only mode, configure and enable the SPI:

- In master mode, the communication starts immediately and stops when the SPE bit is reset and the current reception stops. There is no need to read the BSY flag in this mode. It is always set when an SPI communication is ongoing.
- In slave mode, the SPI continues to receive as long as the NSS is pulled down (or the SSI bit is reset in NSS software mode) and the SCK is running.

### 19.3.5 Data transmission and reception procedures

**Rx and Tx buffer**

In reception, data are received and then stored into an internal Rx buffer while In transmission, data are first stored into an internal Tx buffer before being transmitted.

A read access of the SPI_DR register returns the Rx buffered value whereas a write access of the SPI_DR stores the written data into the Tx buffer.

**Start sequence in master mode**

- In full-duplex (BDM = 0 and RXONLY = 0)
  - The sequence begins when data is written into the SPI_DR register (Tx buffer).
  - The data is then parallel loaded from the Tx buffer into the 8-bit shift register during the first bit transmission and then shifted out serially to the MOSI pin.
  - At the same time, the received data on MISO pin is shifted in serially to the 8-bit shift register and then parallel loaded into the SPI_DR register (Rx Buffer).

- In unidirectional receive-only mode (BDM = 0 and RXONLY = 1)
    - The sequence begins as soon as the bit SPE = 1
    - Only the receiver is activated and the received data on MISO pin is shifted in serially to the 8-bit shift register and then parallel loaded into the SPI_DR register (Rx Buffer).
- In bidirectional mode, when transmitting (BDM = 1 and BDOE = 1)
    - The sequence begins when a data is written into the SPI_DR register (Tx buffer).
    - The data is then parallel loaded from the Tx buffer into the 8-bit shift register during the first bit transmission and then shifted out serially to the MOSI pin.
    - No data is received.
- In bidirectional mode, when receiving (BDM = 1 and BDOE = 0)
    - The sequence begins as soon as SPE = 1 and BDOE = 0.
    - The received data on MOSI pin is shifted in serially to the 8-bit shift register and then parallel loaded into the SPI_DR register (Rx Buffer).
    - The transmitter is not activated and no data is shifted out serially to the MOSI pin.

### Start sequence in slave mode

- In full-duplex (BDM=0 and RXONLY=0)
    - The sequence begins when the slave device receives the clock signal and the first bit of the data on its MOSI pin. The remaining 7 bits are loaded into the shift register.
    - At the same time, the data is parallel loaded from the Tx buffer into the 8-bit shift register during the first bit transmission and then shifted out serially to the MISO pin. The software must have written the data to be sent before the SPI master device initiates the transfer.
- In unidirectional receive-only mode (BDM = 0 and RXONLY = 1)
    - The sequence begins when the slave device receives the clock signal and the first bit of the data on its MOSI pin. The remaining 7 bits are loaded into the shift register.
    - The transmitter is not activated and no data is shifted out serially to the MISO pin.
- In bidirectional mode, when transmitting (BDM = 1 and BDOE = 1)
    - The sequence begins when the slave device receives the clock signal and the first bit of the Tx buffer is transmitted to the MISO pin.
    - The data is then parallel loaded from the Tx buffer into the 8-bit shift register during the first bit transmission and then shifted out serially to the MISO pin. The software must have written the data to be sent before the SPI master device starts the transfer.
    - no data is received.
- In bidirectional mode, when receiving (BDM = 1 and BDOE = 0)
    - The sequence starts when the slave device receives the clock signal and the first bit of the data to its MISO pin.
    - The data received on MISO pin is shifted in serially to the 8-bit shift register and then parallel loaded into the SPI_DR register (Rx Buffer).
    - The transmitter is not activated and no data is shifted out serially to the MISO pin.

**Handling data transmission and reception**

The TXE flag (Tx buffer empty) is set when the data is transferred from the Tx buffer to the shift register. It indicates that the internal Tx buffer is ready to be loaded with the next data. An interrupt can be generated if TXIE bit in the SPI_ICR register is set.

*Note:*     *The software must ensure that TXE flag is set to 1 before attempting to write into the Tx buffer. Otherwise, it will overwrite the data which was previously written in the Tx buffer.*

The RXNE flag (Rx buffer not empty) is set on the last sampling clock edge, when the data is transferred from the shift register to the Rx buffer. It indicates that a data is ready to be read from the SPI_DR register. An interrupt can be generated if RXIE bit in the SPI_ICR register is set. Clearing the RXNE bit is performed by reading the SPI_DR register.

In some configurations, the BSY flag can be used during the last data transfer to wait until the completion of the transfer.

**Full Duplex Transmit and receive procedure in master or slave mode (BDM=0 and RXONLY = 0)**

1.  Enable the SPI by setting the SPE bit

2.  Write the first data to be transmitted in the SPI_DR register (this clears the TXE flag).

3.  Wait until TXE = 1 and write the second data to be transmitted. Then wait until RXNE = 1 and read the SPI_DR to get the first received data (this clears the RXNE bit). Repeat this operation for each data to be transmitted/received until the n-1 received data.

4.  Wait until RXNE = 1 and read the last received data.

5.  Wait until TXE = 1 and then wait until BSY = 0 before disabling the SPI.

This procedure can also be implemented using dedicated interrupt subroutines launched at each rising edge of RXNE or TXE flags.

## Figure 77. TXE/RXNE/BSY behavior in full duplex mode (RXONLY = 0). Case of continuous transfers
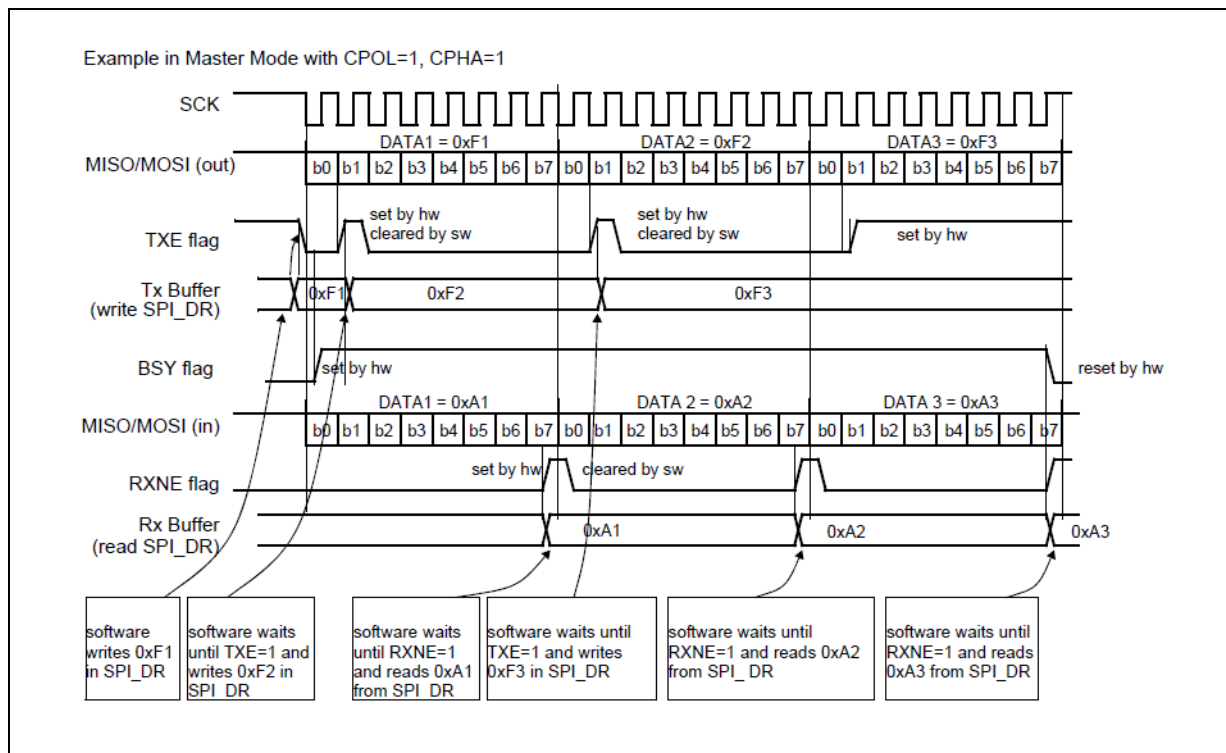


## Figure 78. TXE/RXNE/BSY behavior in slave / full duplex mode (BDM = 0, RXONLY = 0). Case of continuous transfers
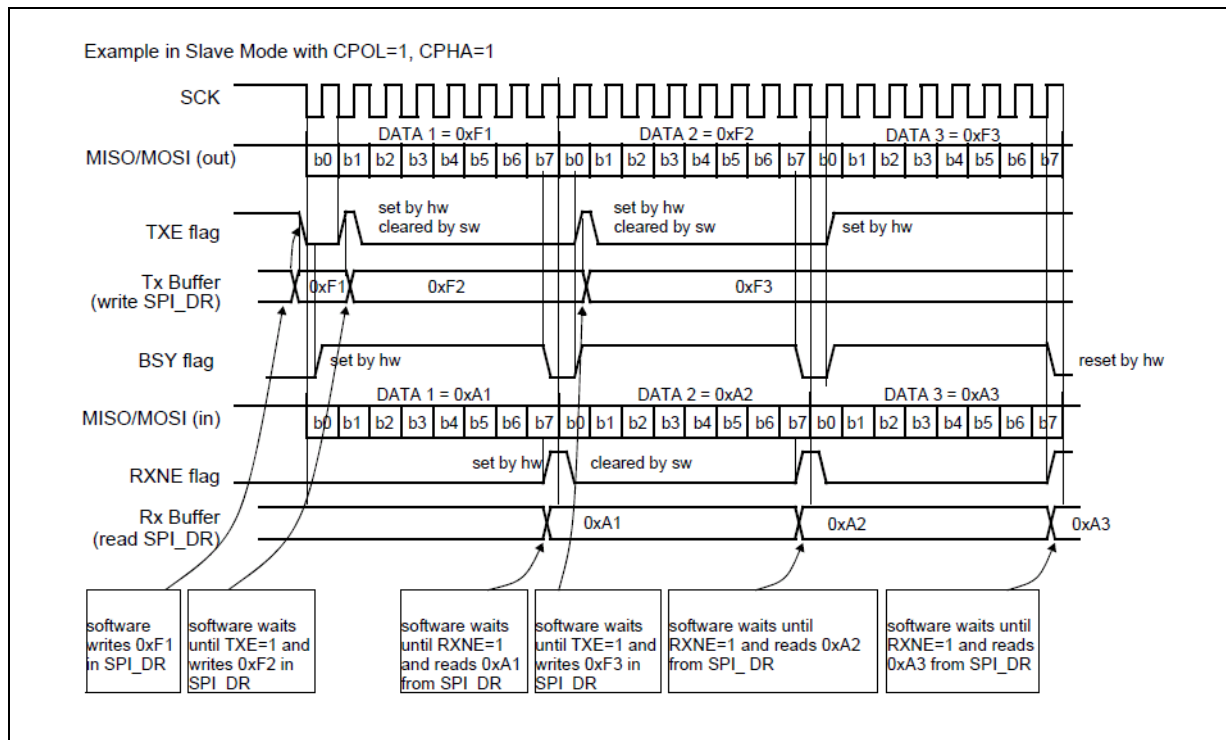
**Transmit-only procedure (BDM = 0 RXONLY = 0)**

In this mode, the procedure can be reduced as described below and the BSY bit can be used to wait until the effective completion of the transmission (see *Figure 77* and *Figure 78*):

1. Enable the SPI by setting the SPE bit
2. Write the first data to send in the SPI_DR register (this clears the TXE bit).
3. Wait until TXE = 1 and write the next data to be transmitted. Repeat this step for each data to be transmitted.
4. After writing the last data in the SPI_DR register, wait until TXE = 1 and then wait until BSY=0 which indicates that the transmission of the last data is complete.

This procedure can be also implemented using dedicated interrupt subroutines launched at each rising edge of TXE flag.

*Note:* 1 *In master mode, during discontinuous communications, there is a 2 CPU clock period delay between the write operation to SPI_DR and the BSY bit setting. As a consequence, in transmit-only mode, it is mandatory to wait first until TXE is set and then until BSY is reset after having written the last data.*

2 *After transmitting two data in transmit-only mode, the OVR flag is set in the SPI_SR register since the received data are never read.*

**Figure 79. TXE/BSY in master transmit-only mode
(BDM = 0 and RXONLY = 0). Case of continuous transfers**
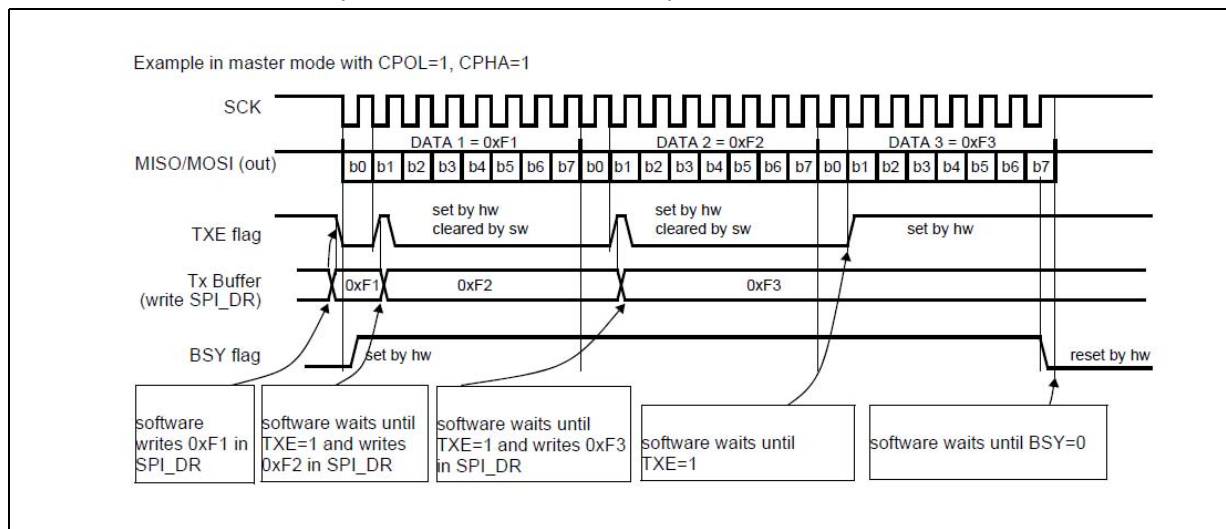
**Figure 80. TXE/BSY in slave transmit-only mode (BDM = 0 and RXONLY = 0).**
**Case of continuous transfers**



### Bidirectional transmit procedure (BDM = 1 and BDOE = 1)

In this mode, the procedure is similar to the Transmit-only procedure except that the BDM and BDOE bits must both be set in the SPI_CR2 register before enabling the SPI.

### Unidirectional receive-only procedure (BDM = 0 and RXONLY = 1)

In this mode, the procedure can be reduced as described below (see *Figure 81*):

1. Set the RXONLY bit in the SPI_CR2 register
2. Enable the SPI by setting bit SPE to 1:
   a) In master mode, this immediately activates the generation of the SCK clock, and data is received serially until the SPI is disabled (SPE = 0).
   b) In slave mode, data are received when the SPI master device drives NSS low and generates the SCK clock.
3. Wait until RXNE =1 and read the SPI_DR register to get the received data (this clears the RXNE bit). Repeat this operation for each data to be received.

This procedure can be also implemented using dedicated interrupt subroutines launched at each rising edge of the RXNE flag.

*Note:* *If it is required to disable the SPI after the last transfer, follow the recommendation described in* Section 19.3.7: Disabling the SPI on page 213.

**Figure 81. RXNE behavior in receive-only mode (BDM = 0 and RXONLY = 1).**
**Case of continuous transfers**
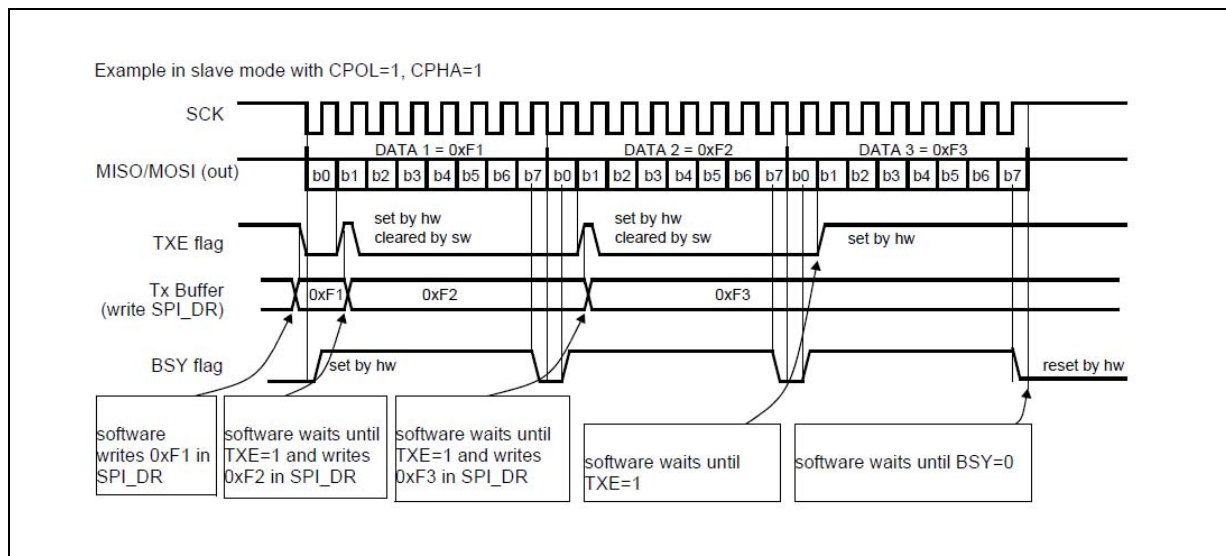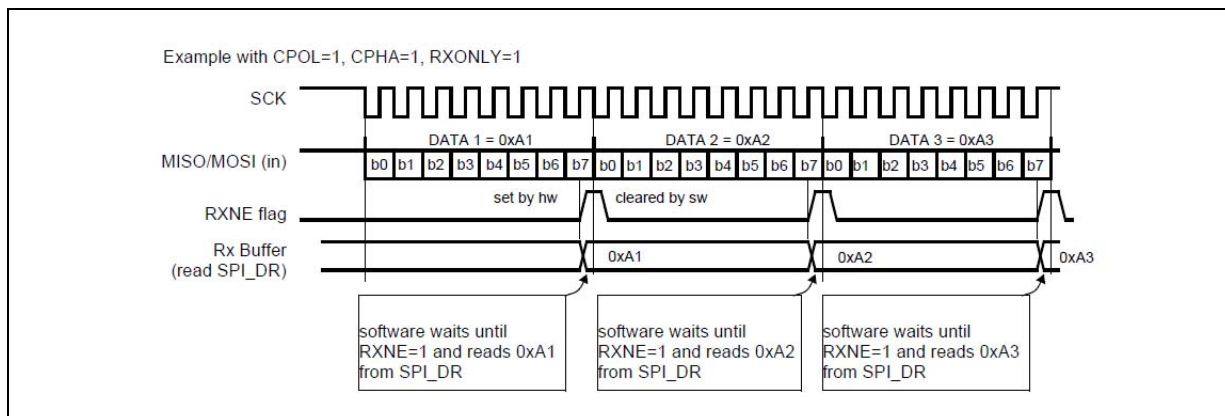


### Bidirectional receive procedure (BDM = 1 and BDOE = 0)

In this mode, the procedure is similar to the Receive-only procedure except that the BDM bit must be set and the BDOE bit must be reset in the SPI_CR2 register before enabling the SPI.

### Continuous and discontinuous transfers

When transmitting data in master mode, if the software is fast enough to detect each TXE rising edge (or TXE interrupt) and to immediately write the SPI_DR register before the ongoing data transfer is complete, the communication is said to be continuous. In this case, there is no discontinuity in the generation of the SPI clock between each data and the BSY bit will never be reset between each data transfer.

On the contrary, if the software is not fast enough, this can lead to some discontinuities in the communication. In this case, the BSY bit is reset between each data transmission (see *Figure 82*).

In master receive-only mode (BDM = 0 and RXONLY = 1) or in bidirectional receive mode (BDM = 1 and BDOE = 0), the communication is always continuous and the BSY flag is always read at 1.

In slave mode, the continuity of the communication is decided by the SPI master device. But even if the communication is continuous, the BSY flag goes low between each transfer for a minimum duration of one SPI clock cycle (see *Figure 78*).

**Figure 82. TXE/BSY behavior when transmitting (BDM = 0 and RXLONY = 0).**
**Case of discontinuous transfers**

Example with CPOL=1, CPHA = 1

SCK

MOSI (out)

DATA 1 = 0xF1    DATA 2 = 0xF2    DATA 3 = 0xF3
b0 b1 b2 b3 b4 b5 b6 b7    b0 b1 b2 b3 b4 b5 b6 b7    b0 b1 b2 b3 b4 b5 b6 b7

TXE flag

Tx buffer
(write SPI_DR)    0xF1    0xF2    0xF3

BSY flag

software writes 0xF1 into SPI_DR

software waits until TXE=1 but is late to write 0xF2 into SPI_DR

software waits until TXE=1 but is late to writes 0xF3 into SPI_DR

software waits until TXE=1

software waits until BSY=0

## 19.3.6 Status flags

There are three status flags to allow the application to completely monitor the state of the SPI bus.

### Tx buffer empty flag (TXE)

When set, this flag indicates that the Tx buffer is empty and that the next data to be transmitted can be loaded into the buffer. The TXE flag is reset when writing the SPI_DR register.

### Rx buffer not empty (RXNE)

When set, this flag indicates that there is a valid received data in the Rx buffer. This flag is reset when SPI_DR is read.

### Busy flag (BSY)

This BSY flag is set and reset by hardware (writing to this flag has no effect). The BSY flag indicates the state of the communication layer of the SPI.

When BSY is set, it indicates that the SPI is busy communicating. There is one exception in master mode / bidirectional receive mode (MSTR=1 and BDM=1 and BDOE=0) where the BSY flag is kept low during the reception.

The BSY flag is useful to detect the end of a transfer if the software wants to disable the SPI and enters Halt mode (or disable the peripheral clock). This will avoid corrupting the last transfer. For this, the procedure described below must be strictly respected.

The BSY flag is also useful to avoid write collisions in a multimaster system.

The BSY flag is set when a transfer starts with the exception of master mode / bidirectional receive mode (MSTR = 1 and BDM = 1 and BDOE = 0).

It is reset:
- when a transfer is finished (except in master mode if the communication is continuous)
- when the SPI is disabled
- when a master mode fault occurs (MODF = 1)

When communication is not continuous, the BSY flag is low between each communication.

When communication is continuous, in master mode, the BSY flag is kept high during the whole transfers.

When communication is continuous, in slave mode, the BSY flag goes back to low state for one SPI clock cycle between each transfer.

*Note:* *Do not use the BSY flag to handle each data transmission or reception. It is better to use TXE and RXNE flags instead.*

## 19.3.7 Disabling the SPI

When a transfer is terminated, the application can stop the communication by disabling the SPI peripheral. This is done by resetting the SPE bit.

For some configurations, disabling the SPI and entering Halt mode while a transfer is on-going, can cause the current transfer to be corrupted and/or it can happen that the BSY flag becomes unreliable.

To avoid any of these effects, it is recommended to respect the following procedure when disabling the SPI:

**In master or slave full duplex mode (BDM = 0, RXONLY = 0):**

1.  Wait until RXNE = 1 to receive the last data
2.  Wait until TXE = 1
3.  Then wait until BSY = 0
4.  Disable the SPI (SPE = 0) and eventually enter Halt mode (or disable the peripheral clock).

**In master or slave unidirectional transmit-only mode (BDM = 0, RXONLY = 0) or bidirectional transmit mode (BDM = 1, BDOE = 1):**

After the last data is written in the SPI_DR register:

1.  Wait until TXE = 1
2.  Then wait until BSY = 0
3.  Disable the SPI (SPE = 0) and, if desired, enter Halt mode (or disable the peripheral clock).

**In master unidirectional receive-only mode (MSTR = 1, BDM = 0, RXONLY = 1) or bidirectional receive mode (MSTR = 1, BDM = 1, BDOE = 0):**

This case must be managed in a particular way to ensure that the SPI does not initiate a new transfer:

1.  Wait for the second to last occurrence of RXNE = 1 (n-1)
2.  Then wait for one SPI clock cycle (using a software loop) before disabling the SPI (SPE = 0)
3.  Then wait for the last RXNE=1 before entering Halt mode (or disabling the peripheral clock).

*Note:*  *In master bidirectional receive mode (MSTR=1 and BDM=1 and BDOE=0), the BSY flag is kept low during a transfer.*

**In slave receive-only mode (MSTR = 0, BDM = 0, RXONLY = 1) or bidirectional receive mode (MSTR = 0, BDM = 1, BDOE = 0):**

1.  You can disable the SPI (write SPE = 1) whenever you want: the current transfer will complete before being effectively disabled.
2.  Then, if you want to enter Halt mode, you must first wait until BSY = 0 before entering Halt mode (or disabling the peripheral clock).

## 19.3.8    Error flags

**Master mode fault (MODF)**

Master mode fault occurs when the master device has its NSS pin pulled low (in NSS hardware mode) or SSI bit low (in NSS software mode), this automatically sets the MODF bit. Master mode fault affects the SPI peripheral in the following ways:

- The MODF bit is set and an SPI interrupt is generated if the ERRIE bit is set.
- The SPE bit is reset. This blocks all output from the device and disables the SPI interface.
- The MSTR bit is reset, thus forcing the device into slave mode.

Use the following software sequence to clear the MODF bit:

1.   Make a read or write access to the SPI_SR register while the MODF bit is set.
2.   Then write to the SPI_CR1 register.

To avoid any multiple slave conflicts in a system comprising several MCUs, the NSS pin must be pulled high during the MODF bit clearing sequence. The SPE and MSTR bits can be restored to their original state after this clearing sequence.

As a security, hardware does not allow you to set the SPE and MSTR bits while the MODF bit is set.

In a slave device the MODF bit cannot be set. However, in a multi-master configuration, the device can be in slave mode with this MODF bit set. In this case, the MODF bit indicates that there might have been a multimaster conflict for system control. You can use an interrupt routine to recover cleanly from this state by performing a reset or returning to a default state.

**Overrun condition**

An overrun condition occurs, when the master device has sent data bytes and the slave device has not cleared the RXNE bit resulting from the previous data byte transmitted. When an overrun condition occurs:

- OVR bit is set and an interrupt is generated if the ERRIE bit is set.

In this case, the receiver buffer contents will not be updated with the newly received data from the master device. A read to the SPI_DR register returns this byte. All other subsequently transmitted bytes are lost.

Clearing the OVR bit is done by a read access to the SPI_DR register followed by a read access to the SPI_SR register.

### 19.3.9 SPI low power modes

**Table 38. SPI behavior in low power modes**

| Mode | Description |
|---|---|
| Wait | No effect on SPI.<br>SPI interrupt events cause the device to exit from Wait mode. |
| Halt | SPI registers are frozen.<br>In Halt mode, the SPI is inactive. If the SPI is in master mode, then communication resumes when the device is woken up by an interrupt with "wakeup from Halt mode" capability.<br>If the SPI is in slave mode, then it can wake up the MCU from Halt mode after detecting the first sampling edge of data. |

#### Using the SPI to wake up the device from Halt mode

When the microcontroller is in Halt mode, the SPI is still capable of responding as a slave provided the NSS pin is tied low or the SSI bit is reset before entering Halt mode.

When the first sampling edge of data (as defined by the CPHA bit) is detected:

- The WKUP bit is set in the SPI_SR register
- An interrupt is generated if the WKIE bit in the SPI_ICR register is set.
- This interrupt wakes up the device from Halt mode.
- Due to the time needed to restore the system clock, the SPI slave sends or receives a few data before being able to communicate correctly. It is then mandatory to use the following protocol:
  - A specific value is written into the SPI_DR before entering Halt mode. This value indicates to the external master that the SPI is in Halt mode
  - The external master sends the same byte continuously until it receives from the SPI slave device a new value other than the unique value indicating the SPI is in Halt mode. This new value indicates the SPI slave has woken-up and can correctly communicate.

#### Restrictions in receive-only modes

The wakeup functionality is not guaranteed in receive-only modes (BDM = 0 and RXONLY = 1 or BDM = 1 and BDOE = 0) since the time needed to restore the system clock can be greater than the data reception time. A loss of data in reception would then be induced and the slave device can not indicate to the master which data has been properly received.

### 19.3.10 SPI interrupts

**Table 39. SPI interrupt requests**

| Interrupt event | Event flag | Enable control bit | Exit from Wait | Exit from Halt |
|---|---|---|---|---|
| Transmit buffer empty flag | TXE | TXIE | Yes | No |
| Receive buffer not empty flag | RXNE | RXIE | Yes | No |

**Table 39. SPI interrupt requests (continued)**

| Interrupt event | Event flag | Enable control bit | Exit from Wait | Exit from Halt |
|---|---|---|---|---|
| Wakeup event flag | WKUP | WKIE | Yes | Yes |
| Master mode fault event | MODF | ERRIE | Yes | No |
| Overrun error | OVR | | Yes | No |

## 19.4 SPI registers

### 19.4.1 SPI control register 1 (SPI_CR1)

Address offset: 0x00
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| LSBFIRST | SPE | | BR [2:0] | | MSTR | CPOL | CPHA |
| rw | rw | | rw | | rw | rw | rw |

Bit 7 **LSBFIRST**: Frame format [1]

  0: MSB is transmitted first
  1: LSB is transmitted first

Bit 6 **SPE**: SPI enable [2]

  0: Peripheral disabled
  1: Peripheral enabled

Bits 5:3 **BR[2:0]**: Baud rate control

  000: $f_{MASTER}/2$
  001: $f_{MASTER}/4$
  010: $f_{MASTER}/8$
  011: $f_{MASTER}/16$
  100: $f_{MASTER}/32$
  101: $f_{MASTER}/64$
  110: $f_{MASTER}/128$
  111: $f_{MASTER}/256$
  *Note:   These bits should not be changed when the communication is ongoing.*

Bit 2 **MSTR**: Master selection [1]

  0: Slave configuration
  1: Master configuration

Bit1 **CPOL**: Clock polarity [1]

  0: SCK to 0 when idle
  1: SCK to 1 when idle

Bit 0 **CPHA**: Clock phase [1]

  0: The first clock transition is the first data capture edge
  1: The second clock transition is the first data capture edge

1. This bit should not be changed when the communication is ongoing.

2. When disabling the SPI, follow the procedure described in *Section 19.3.7: Disabling the SPI on page 213*

## 19.4.2     SPI control register 2 (SPI_CR2)

Address offset: 0x01
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BDM | BDOE | | Reserved | | RXOnly | SSM | SSI |
| rw | rw | | | | rw | rw | rw |

Bit 7    **BDM**: Bidirectional data mode enable

0: 2-line unidirectional data mode selected
1: 1-line bidirectional data mode selected

Bit 6    **BDOE**: Input/Output enable in bidirectional mode

This bit selects the direction of transfer in bidirectional mode when BDM is set to 1.

0: Input enabled (receive-only mode)
1: Output enabled (transmit-only mode)
In master mode, the MOSI pin is used and in slave mode, the MISO pin is used.

Bits 5:3    Reserved

Bit 2    **RXONLY**: Receive only

0: Full duplex (Transmit and receive)
1: Output disabled (Receive only mode)

This bit combined with BDM bit selects the direction of transfer in 2 line uni-directional mode

This bit is also useful in a multi-slave system in which this particular slave is not accessed, the output from the accessed slave is not corrupted.

Bit 1    **SSM**: Software slave management

0: Software slave management disabled
1: Software slave management enabled

When the SSM bit is set, the NSS pin input is replaced with the value coming from the SSI bit

Bit 0    **SSI**: Internal slave select

This bit has effect only when SSM bit is set. The value of this bit is forced onto the NSS pin and the I/O value of the NSS pin is ignored.

0: Slave mode
1: Master mode

### 19.4.3 SPI interrupt control register (SPI_ICR)

Address offset: 0x02
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TXIE | RXIE | ERRIE | WKIE | Reserved | | | |
| rw | rw | rw | rw | r | | | |

Bit 7 **TXIE**: Tx buffer empty interrupt enable
0: TXE interrupt masked
1: TXE interrupt not masked. This allows an interrupt request to be generated when the TXE flag is set.

Bit 6 **RXIE**: RX buffer not empty interrupt enable
0: RXNE interrupt masked
1: RXNE interrupt not masked. This allows an interrupt request to be generated when the RXNE flag is set.

Bit 5 **ERRIE**: Error interrupt enable
0: Error interrupt is masked
1: Error interrupt is enabled. This allows an interrupt request to be generated when an error condition occurs (OVR, MODF).

Bit 4 **WKIE**: Wakeup interrupt enable
0: Wakeup interrupt masked
1: Wakeup interrupt enabled. This allows an interrupt request to be generated when the WKUP flag is set.

Bits 3:0 Reserved

### 19.4.4  SPI status register (SPI_SR)

Address offset: 0x03
Reset value: 0x02

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BSY | OVR | MODF | Reserved | WKUP | Reserved | TXE | RXNE |
| r | rc_w0 | rc_w0 | | rc_w0 | | r | r |

Bit 7 **BSY**: Busy flag
> 0: SPI not busy
> 1: SPI is busy in communication

This flag is set and reset by hardware.

*Note:  BSY flag must be used with cautious: refer to Section 19.3.6: Status flags on page 213 and Section 19.3.7: Disabling the SPI on page 213*

Bit 6 **OVR**: Overrun flag
> 0: No Overrun occurred
> 1: Overrun occurred

This flag is set by hardware and reset by a software sequence.

Bit 5 **MODF**: Mode fault
> 0: No Mode fault occurred
> 1: Mode fault occurred

This flag is set by hardware and reset by a software sequence.

Bit 4 Reserved

Bit 3 **WKUP**: Wakeup flag
> 0: No wakeup event occurred
> 1: Wakeup event occurred

This flag is set on the first sampling edge on SCK when the STM8 is in Halt mode and the SPI is configured as slave.

This flag is reset by software writing 0.

Bit 2 Reserved

Bit 1 **TXE**: Transmit buffer empty
> 0: Tx buffer not empty
> 1: Tx buffer empty

Bit 0 **RXNE**: Receive buffer not empty
> 0: Rx buffer empty
> 1: Rx buffer not empty

### 19.4.5 SPI data register (SPI_DR)

Address offset: 0x04
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | DR[7:0] | | | | |
| | | | rw | | | | |
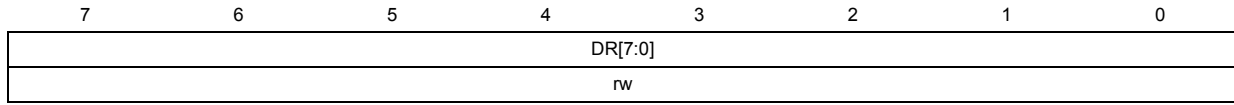
Bits 7:0 **DR[7:0]**: Data register

Byte received or to be transmitted.

The data register is split into 2 buffers - one for writing (Transmit buffer) and another one for reading (Receive buffer). A write to the data register will write into the Tx buffer and a read from the data register will return the value held in the Rx buffer.

## 19.5 SPI register map and reset values

**Table 40. SPI register map and reset values**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | SPI_CR1 Reset value | LSB FIRST 0 | SPE 0 | BR2 0 | BR1 0 | BR0 0 | MSTR 0 | CPOL 0 | CPHA 0 |
| 0x01 | SPI_CR2 Reset value | BDM 0 | BDOE 0 | - 000 | | | RXONLY 0 | SSM 0 | SSI 0 |
| 0x02 | SPI_ICR Reset value | TXIE 0 | RXIE 0 | ERRIE 0 | WKIE 0 | - 0000 | | | |
| 0x03 | SPI_SR Reset value | BSY 0 | OVR 0 | MODF 0 | - 0 | WKUP 0 | - 0 | TXE 1 | RXNE 0 |
| 0x04 | SPI_DR Reset value | DR[7:0] 0 | | | | | | | |

# 20 Universal synchronous/asynchronous receiver transmitter (USART)

## 20.1 USART introduction

The USART (universal synchronous asynchronous receiver transmitter) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. It offers a very wide range of baud rates.

The USART supports synchronous one-way communication. The USART can also be used for multiprocessor communication.

## 20.2 USART main features

- Full duplex, asynchronous communications
- NRZ standard format (Mark/Space)
- 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
- Transmitter clock output for synchronous communication
- 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
- 5 interrupt sources with flags:
    - Transmit data register empty
    - Transmission complete
    - Receive data register full
    - Idle line received
    - Parity error
- 2 interrupt vectors:
    - Transmitter interrupt
    - Receiver interrupt
- Reduced power consumption mode
- Multi-Processor communication - enter into mute mode if address match does not occur
- Wakeup from mute mode (by idle line detection or address mark detection)
- 2 receiver wakeup modes:
    - Address bit (MSB)
    - Idle line

## 20.3 USART functional description

The interface is externally connected to another device by three pins (see *Figure 83*). Any USART bidirectional communication requires a minimum of two pins: USART Receive data input (USART_RX) and USART transmit data output (USART_TX):

**USART_RX** is the serial data input. Over-sampling techniques are used for data recovery by discriminating between valid incoming data and noise.

**USART_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 nothing is to be transmitted, the pin is at high level.

Through these pins, serial data is transmitted and received in normal USART mode as frames including:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 1 and 2 STOP bits indicating that the frame is complete
- A status register (USART_SR)
- Data register (USART_DR)
- 16-bit baud rate prescaler (USART_BRR)

Refer to the register description for the definitions of each bit.

The following pin is required to interface in synchronous mode:

**USART_CK**: Transmitter clock output. This pin outputs the transmitter data clock for synchronous transmission (no clock pulses on start bit and STOP bit, and a software option to send a clock pulse on the last data bit). This can be used to control peripherals that have shift registers (e.g. LCD drivers). The clock phase and polarity are software programmable.

**Figure 83. USART block diagram**
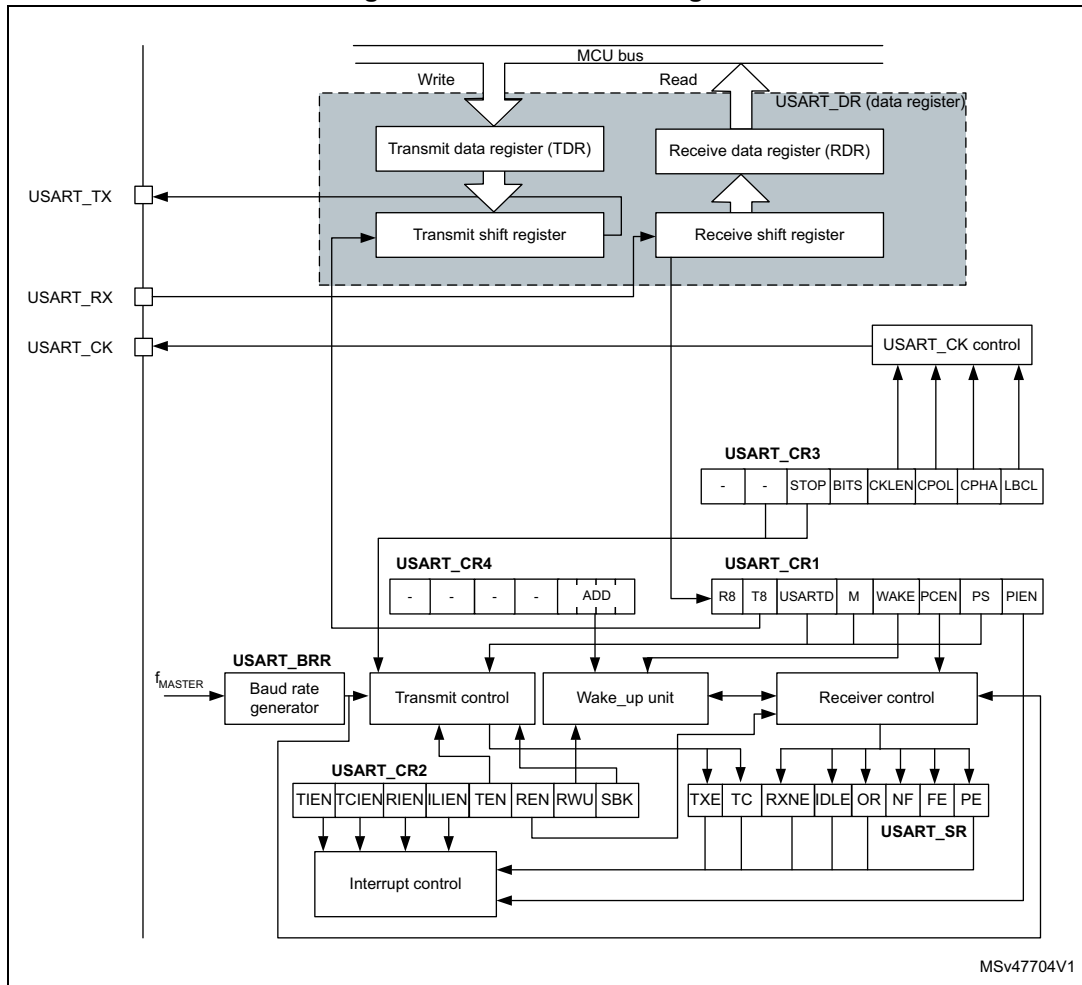
## 20.3.1     USART character description

Word length may be selected as being either 8 or 9 bits by programming the M bit in the USART_CR1 register (see *Figure 84*).

The USART_TX pin is in low state during the start bit. It is in 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 for each is generated when the enable bit is set respectively for the transmitter and receiver.

The details of each block is given below.

**Figure 84. Word length programming**
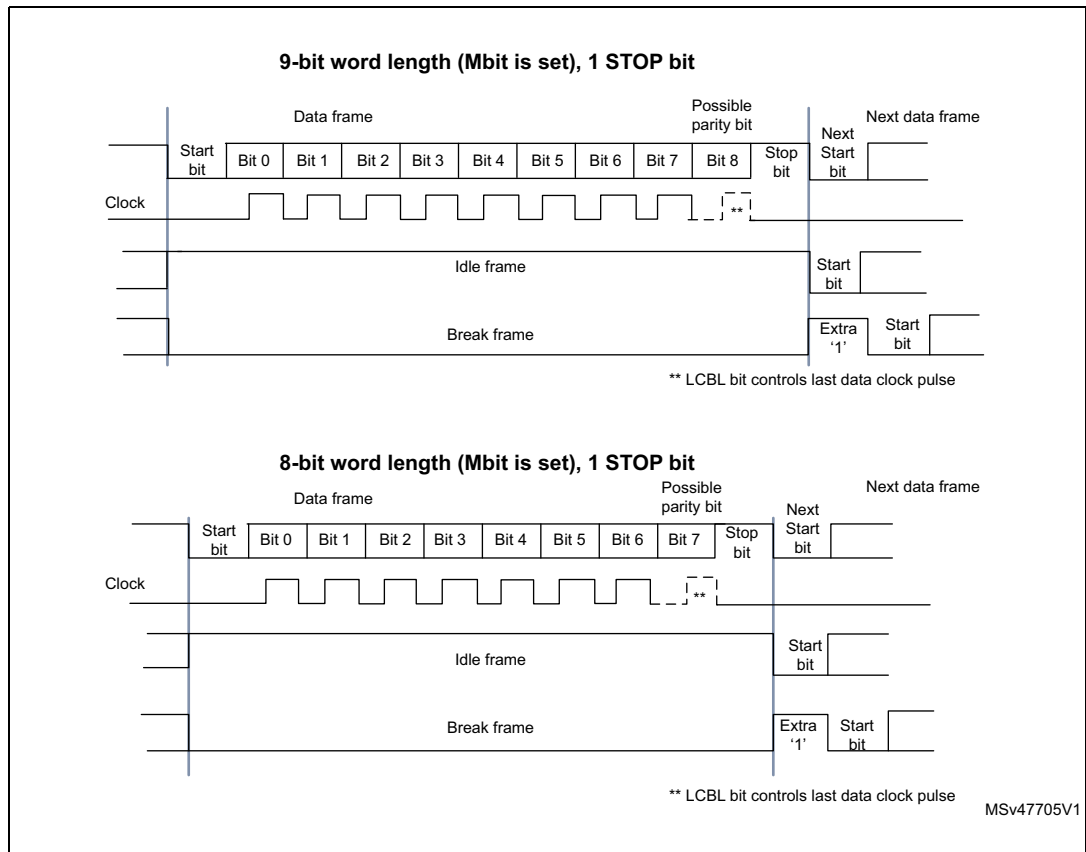
### 20.3.2 Transmitter

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) has to be stored in the T8 bit in the USART_CR1 register.

When the transmit enable bit (TEN) is set, the data in the transmit shift register is output on the USART_TX pin and the corresponding clock pulses are output on the USART_CK pin.

#### Character transmission

During a USART transmission, data shifts out least significant bit first on the USART_TX pin. In this mode, the USART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see *Figure 83*).

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.

The following STOP bits are supported by USART.

*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 USART_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 during transmission

The number of STOP bits to be transmitted with every character can be programmed in Control register 3, bits 5,4.

- *1 STOP bit*: This is the default value of number of STOP bits.
- *2 STOP bits*: This will be supported by normal mode USART.

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).

**Figure 85. Configurable STOP bits**
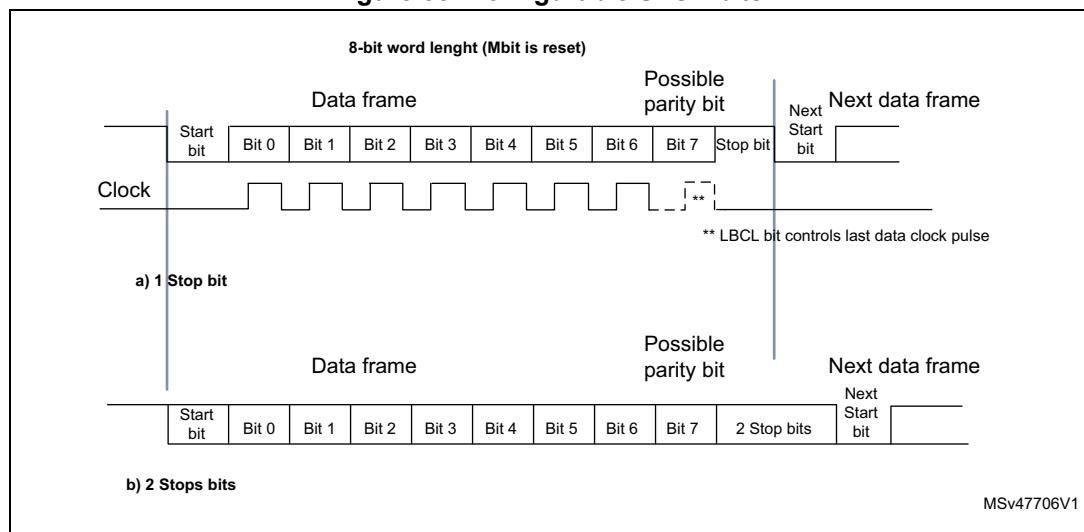
**Procedure**

1. Program the M bit in USART_CR1 to define the word length.
2. Program the number of STOP bits in USART_CR3.
3. Select the desired baud rate by programming the baud rate registers in the following order:
    a) USART_BRR2
    b) USART_BRR1
4. Set the TEN bit in USART_CR2 to enable transmitter mode.
5. Write the data to send in the USART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
6. Once the last data is written to the USART_DR register, wait until TC is set to '1', which indicates that the last data transmission is complete. This last step is required, for instance, to avoid last data transmission corruption when disabling the USART or entering 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 TDR to the shift register and the data transmission has started.
- The TDR register is empty.
- The next data can be written in the USART_DR register without overwriting the previous data.

This flag generates an interrupt if the TIEN bit is set.

When a transmission is taking place, a write instruction to the USART_DR register stores the data in the TDR register. The data is copied in the shift register at the end of the current transmission.

When no transmission is taking place, a write instruction to the USART_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 USART_CR2 register. After writing the last data into the USART_DR register, it is mandatory to wait until TC is set to '1' before entering Halt mode or disabling the USART (see *Figure 86: TC/TXE behavior when transmitting*).

Clearing the TC bit is performed by the following software sequence:

1. A read to the USART_SR register
2. A write to the USART_DR register

*Note:* *The TC bit can also be cleared by writing a '0' to it. This clearing sequence is recommended only for multibuffer communication.*

**Figure 86. TC/TXE behavior when transmitting**
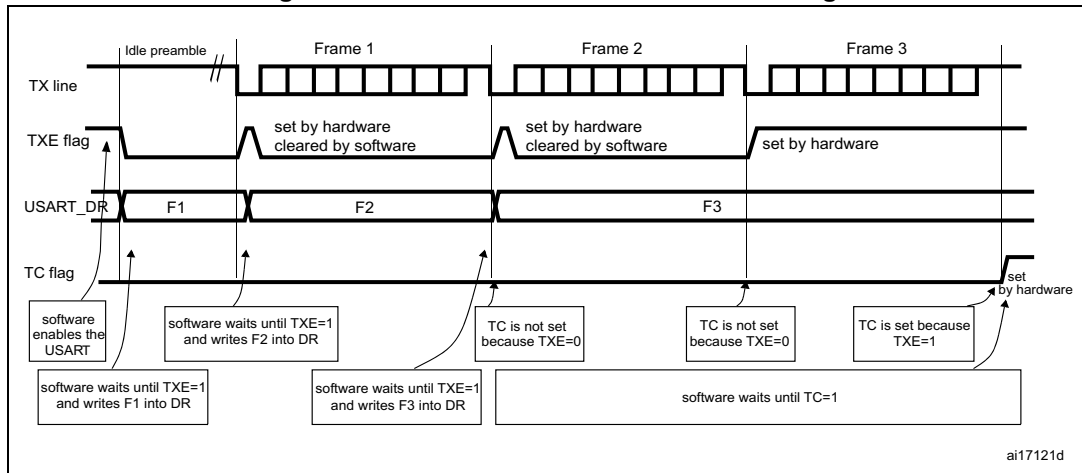


1. This example assumes that several other transmissions occurred after TE has been set. Otherwise an IDLE preamble would be transmitted first when writing to USART_DR for the first time.

### Break character

Setting the SBK bit transmits a break character. The break frame length depends on the M bit (see *Figure 84*).

If the SBK bit is set to '1' a break character is sent on the USART_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 USART 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:* *The break character is sent without taking into account the number of STOP bits. If the USART 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.*

*Note:* *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 USART to send an idle frame before the first data frame.

## 20.3.3 Receiver

The USART 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 USART_CR1 register.

### Start bit detection

In the USART, the start bit is detected when a specific sequence of samples is recognized. This sequence is: 1 1 1 0 X 0 X 0X 0X 0 X 0X 0. The start bit detection sequence shown in *Figure 87*.

**Figure 87. Start bit detection**
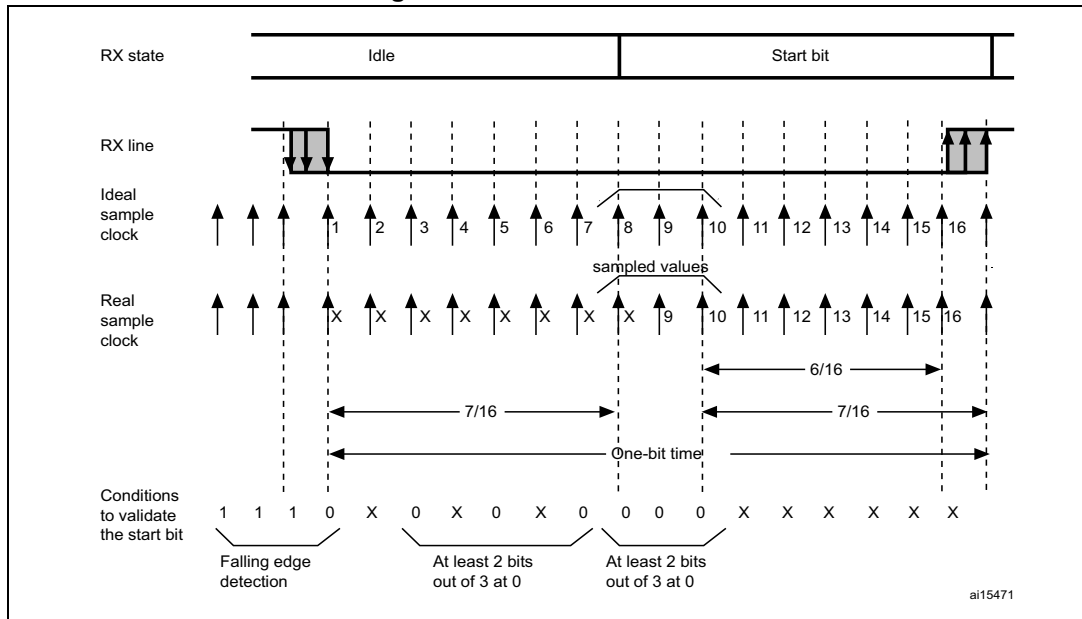


*Note:* *If the sequence is not complete, the start bit detection aborts and the receiver returns to the idle state (no flag is set), where it waits for a falling edge.*

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 are at 0 (sampling on the 8th, 9th, and 10th bits.

**Character reception**

During a USART reception, data shifts in least significant bit first through the USART_RX pin. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register (see *Figure 2*).

Procedure:

1.  Program the M bit in USART_CR1 to define the word length.
2.  Program the number of STOP bits in USART_CR3.
3.  Select the desired baud rate by programming the baud rate registers in the following order:
    a)  USART_BRR2
    b)  USART_BRR1
4.  Set the REN bit USART_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 a software read to the USART_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 not be 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 USART handles it as a framing error.

### Idle character

When an idle frame is detected, there is the same procedure as a received data character plus an interrupt if the ILIEN bit is set.

### Overrun error

An overrun error occurs when a character is received when RXNE has not been reset. Data can not 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 is available when a read to USART_DR is performed.
- The shift register will be overwritten. The second data received during overrun is lost.
- An interrupt is generated if the RIEN bit is set.
- The OR bit is reset by a read to the USART_SR register followed by a USART_DR register read operation.

### Noise error

Over-sampling techniques are used for data recovery by discriminating between valid incoming data and noise.

**Figure 88. Data sampling for noise detection**
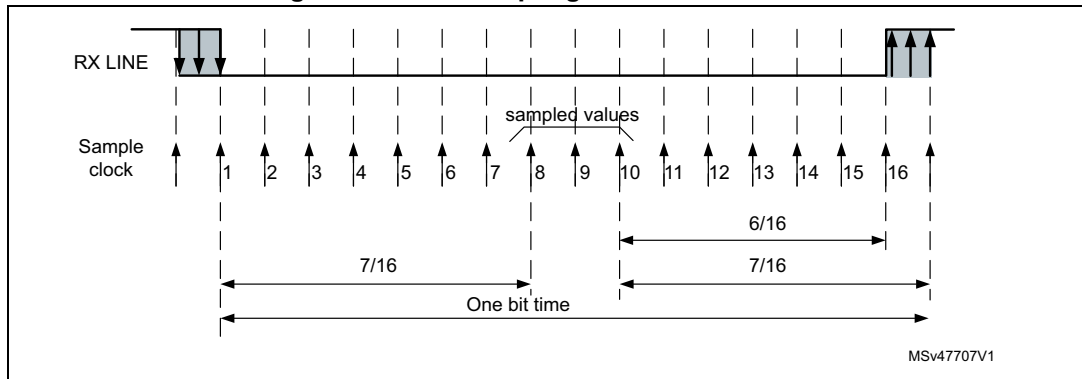


*Note:* *The sample clock frequency is 16x baud rate.*

**Table 41. 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 is transferred from the Shift register to the USART_DR register.

This bit rises at the same time as the RXNE bit which generates an interrupt. The NF bit is reset by a USART_SR register read operation followed by a USART_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 de-synchronization or excessive noise.

When the framing error is detected:

- The FE bit is set by hardware
- The invalid data is transferred from the Shift register to the USART_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 USART_SR register read operation followed by a USART_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 Control Register 3 - it can be either 1 or 2.

1. *1 STOP bit*: Sampling for 1 STOP bit is done on the 8th, 9th and 10th samples.

2. *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.

## 20.3.4 High precision baud rate generator

The receiver and transmitter (Rx and Tx) are both set to the same baud rate programmed by a 16-bit divider USART_DIV according to the following formula:

$$\text{Tx/ Rx baud rate} = \frac{f_{MASTER}}{USART\_DIV}$$

The USART_DIV baud rate divider is an unsigned integer, coded in the BRR1 and BRR2 registers as shown in *Figure 89*.

Refer to *Table 42* for typical baud rate programming examples.

**Figure 89. How to code USART_DIV in the BRR registers**
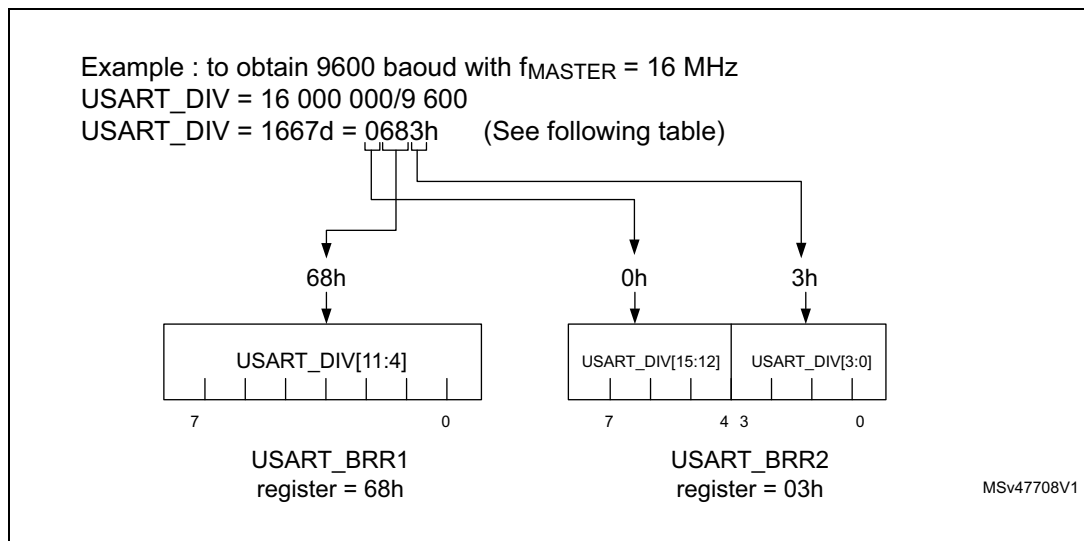


*Note:* *The Baud Counters will be updated with the new value of the Baud Registers after a write to BRR1. Hence the Baud Register value should not be changed during a transaction. The BRR2 should be programmed before BRR1.*

*Note:* *USART_DIV must be greater than or equal to 16d.*

**Table 42. Baud rate programming and error calculation**

| Baud rate | fMASTER= 10 MHz | | | | | fMASTER = 16 MHz | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| In bps | Actual (bps) | % Error (%)[1] | USART_DIV (h) | BRR1 (h) | BRR2 (h) | Actual | % Error[1] | USART_DIV | BRR1 | BRR2 |
| 2400 | 2399.81 | -0.008 | 1047 | 4 | 17 | 2399.88 | -0.005 | 1A0B | A0 | 1B |
| 9600 | 9596.93 | -0.03 | 412 | 41 | 2 | 9598.08 | -0.02 | 683 | 68 | 3 |
| 19200 | 19193.86 | -0.03 | 209 | 20 | 9 | 19207.68 | 0.04 | 341 | 34 | 1 |
| 57600 | 57471.26 | -0.22 | AE | A | E | 57553.96 | -0.08 | 116 | 11 | 6 |
| 115200 | 114942.53 | -0.22 | 57 | 5 | 7 | 115107.91 | -0.08 | 8B | 8 | B |
| 230400 | 232558.14 | 0.94 | 2B | 2 | B | 231884.06 | 0.64 | 45 | 4 | 5 |
| 460800 | 454545.45 | -1.36 | 16 | 1 | 6 | 457142.86 | -0.79 | 23 | 2 | 3 |
| 921600 | NA | NA | NA | NA | NA | 941176.47 | 2.12 | 11 | 1 | 1 |

1. Error % = (Calculated - Desired) Baud Rate / Desired Baud Rate

*Note:*      *The lower the fMASTER frequency, the lower will be the accuracy for a particular baud rate.The upper limit of the achievable baud rate can be fixed with this data.*

## 20.3.5 USART receiver's tolerance to clock deviation

The USART's asynchronous receiver works correctly only if the total clock system deviation is smaller than the USART receiver's tolerance. The causes which contribute to the total deviation are:

- DTRA: Deviation due to the transmitter error (which also includes the deviation of the transmitter's local oscillator)
- DQUANT: Error due to the baud rate quantization of the receiver
- DREC: Deviation of the receiver's local oscillator
- DTCL: Deviation due to the transmission line (generally due to the transceivers which can introduce an asymmetry between the low-to-high transition timing and the high-to-low transition timing)

DTRA + DQUANT + DREC + DTCL < USART receiver's tolerance

The USART receiver's tolerance to properly receive data is equal to the maximum tolerated deviation and depends on the following choices:

- 10- or 11-bit character length defined by the M bit in the USART_CR1 register
- Use of fractional baud rate or not

**Table 43. USART receiver's tolerance when USART_DIV[3:0] is 0**

| M bit | NF is an error | NF is don't care |
|---|---|---|
| 0 | 3.75% | 4.375% |
| 1 | 3.41% | 3.97% |

**Table 44. USART receiver's tolerance when USART_DIV[3:0] is different from 0**

| M bit | NF is an error | NF is don't care |
|-------|----------------|------------------|
| 0 | 3.33% | 3.88% |
| 1 | 3.03% | 3.53% |

*Note:* *The figures specified in Table 43 and Table 44 may slightly differ in the special case when the received frames contain some Idle frames of exactly 10-bit times when M=0 (11-bit times when M=1).*

### 20.3.6 Parity control

Parity control (generation of parity bit in transmission and parity checking in reception) can be enabled by setting the PCEN bit in the USART_CR1 register. Depending on the frame length defined by the M bit, the possible USART frame formats are as listed in *Table 45*.

**Table 45. Frame formats**

| M bit | PCEN bit | USART 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 \| |

Legends: SB: Start Bit, STB: STOP bit, PB: Parity bit

*Note:* *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 M is equal to 0 or 1) and the parity bit.

Ex: data=00110101; 4 bits set => parity bit will be 0 if even parity is selected (PS bit in USART_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 M is equal to 0 or 1) and the parity bit.

Example: data=00110101; 4 bits set => parity bit will be 1 if odd parity is selected (PS bit in USART_CR1 = 1).

**Transmission:** If the PCEN bit is set in USART_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 USART_SR register and an interrupt is generated if the PIEN bit is set in the USART_CR1 register.

## 20.3.7    Multi-processor communication

It is possible to perform multiprocessor communication with the USART (several USARTs connected in a network). For example, one of the USARTs can be the master, its Tx output is connected to the Rx input of the other USART. The others are slaves, their respective Tx outputs are logically ANDed together and connected to the Rx input of the master.

In multiprocessor configurations it is often desirable that only the intended message recipient should actively receive the full message contents, thus reducing redundant USART service overhead for all non addressed receivers.

The non addressed devices may be placed in 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 USART_CR2 register is set to 1. RWU can be controlled automatically by hardware or written by the software under certain conditions.

The USART can enter or exit from mute mode using one of two methods, depending on the WAKE bit in the USART_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 USART enters 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 USART_SR register. RWU can also be written to 0 by software.

An example of mute mode behavior using idle line detection is given in *Figure 90*.

**Figure 90. Mute mode using Idle line detection**
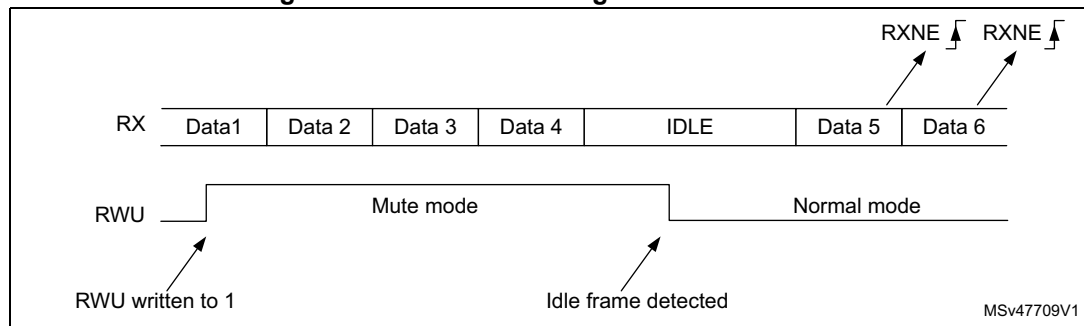


### Address mark detection (WAKE=1)

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 USART_CR4 register.

The USART enters 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 USART would have entered mute mode.

It exits from 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 USART_SR register). Otherwise the write attempt is ignored.

An example of mute mode behavior using address mark detection is given in *Figure 91*.

**Figure 91. Mute mode using address mark detection**
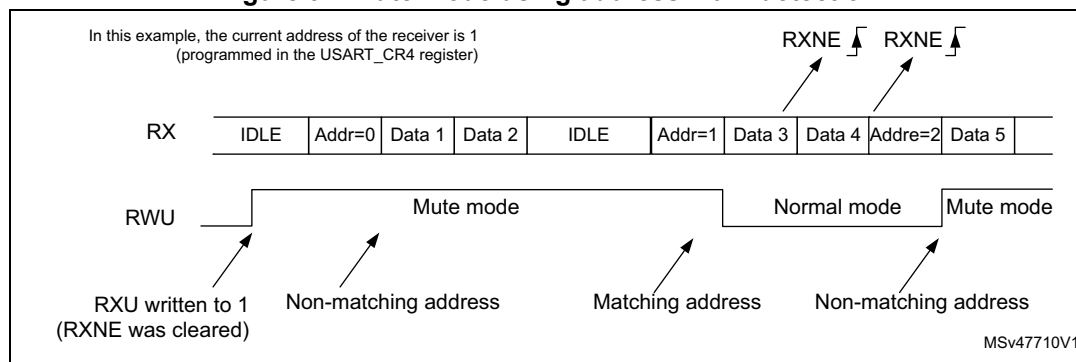


*Note:* *If parity control is enabled, the parity bit remains in the MSB and the address bit is put in the "MSB - 1" bit.*

*For example, with 7-bit data, 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

### 20.3.8 USART synchronous communication

The USART transmitter allows the user to control bidirectional synchronous serial communications in master mode.

*Note:* *This feature is only available for devices with USART_CK pin. Check the device pinout for availability.*

The USART_CK pin is the output of the USART transmitter clock. No clock pulses are sent to the USART_CK pin during start bit and STOP bit. Depending on the state of the LBCL bit in the USART_CR3 register clock pulses will or will not be generated during the last valid data bit (address mark). The CPOL bit in the USART_CR3 register allows the user to select the clock polarity, and the CPHA bit in the USART_CR3 register allows the user to select the phase of the external clock (see *Figure 92*, *Figure 93* & *Figure 94*). During idle and break frames, the external CK clock is not activated.

In synchronous mode, the USART receiver works differently compared to asynchronous mode. If REN=1, the data is sampled on USART_CK (rising or falling edge, depending on CPOL and CPHA), without any oversampling. A setup and a hold time (even if the hold time is not relevant due to the SPI protocol) must be respected (which depends on the baud rate: 1/16 bit time for an integer baud rate).

During the idle state, preamble phase and break transmission, the external USART_CK clock is not activated. In synchronous mode, the USART transmitter works exactly like in asynchronous mode. But as USART_CK is synchronized with USART_TX (depending on CPOL and CPHA), the data on USART_TX is synchronous. In this mode, the USART receiver works slightly differently compared to the asynchronous mode: if REN=1, the data is still sampled using the internal oversampling clock and the baud rate clock is output on the USART_CK pin (rising or falling edge is aligned with the data sampling event depending on CPOL and CPHA). But contrary to asynchronous mode, the data is evaluated using one sample and not the majority of 3 samples, meaning that the NF bit will never be set.

Setup and hold times must be respected (depending on the baud rate: 1/16 bit time for an integer baud rate).

*Note:*    *1*    *The USART_CK pin works in conjunction with the USART_TX pin. When the USART transmitter is disabled (TEN and REN= 0), the USART_CK and USART_TX pins go into high impedance state.*

       *2*    *The LBCL, CPOL and CPHA bits in USART_CR3 have to be selected when both the transmitter and the receiver are disabled (TEN=REN=0) to ensure that the clock pulses function correctly. These bits should not be changed while the transmitter or the receiver is enabled.*

       *3*    *It is recommended to set TEN and REN are set in the same instruction in order to minimize the setup and the hold time of the receiver.*

       *4*    *The USART supports master mode only: it cannot receive or send data related to an input clock (USART_CK is always an output).*

       *5*    *The data given in this section apply only when the USART_DIV[3:0] bits in the USART_BRR2 register are kept at 0. Else the setup and hold times are not 1/16 of a bit time but 4/16 of a bit time.*

This option allows to serially control peripherals which consist of shift registers, without losing any functions of the asynchronous communication which can still talk to other asynchronous transmitters and receivers.

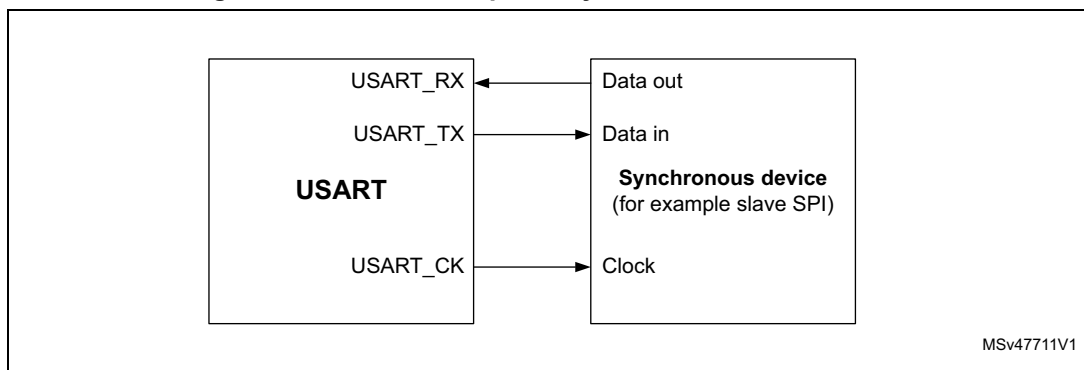**Figure 92. USART example of synchronous transmission**
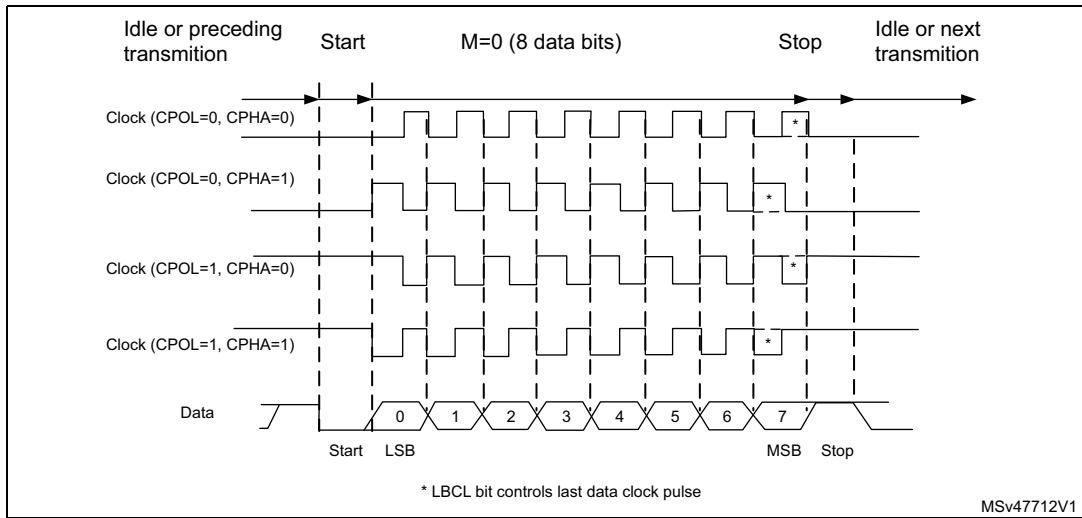
**Figure 93. USART data clock timing diagram (M=0)**
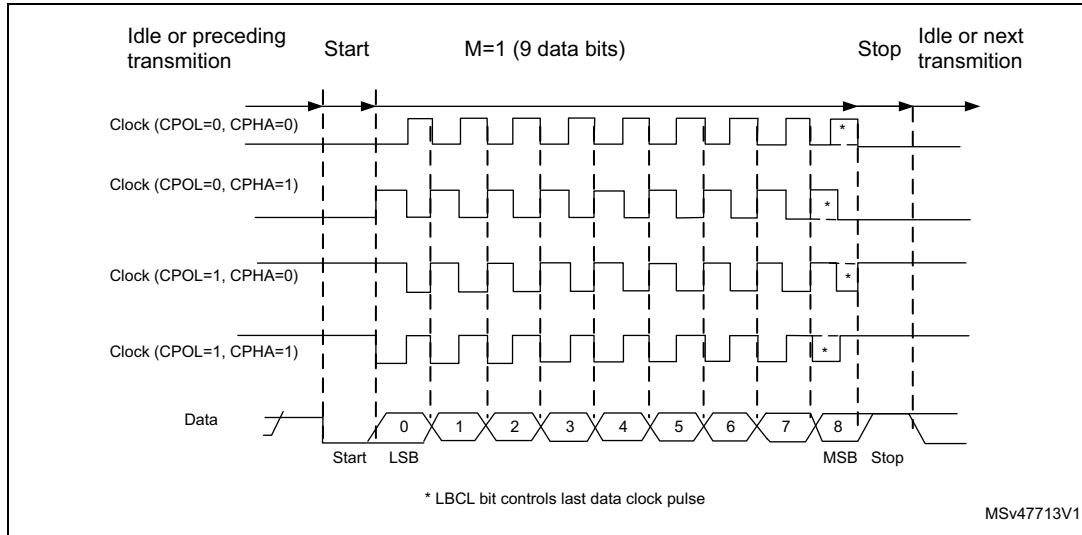


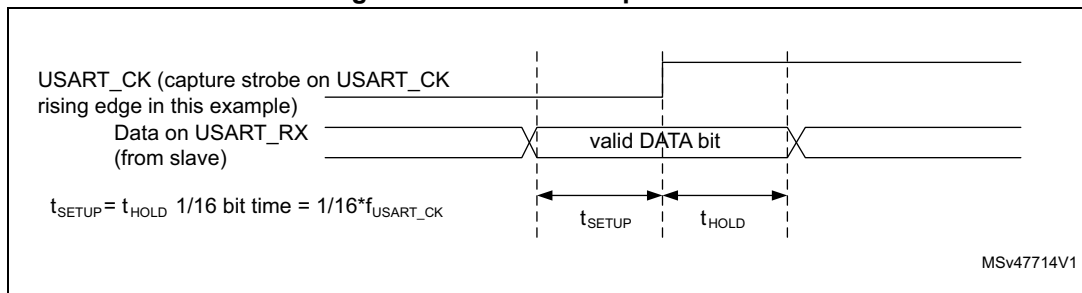**Figure 94. USART data clock timing diagram (M=1)**



**Figure 95. RX data setup/hold time**

## 20.4 USART low power modes

**Table 46. USART interface behavior in low power modes**

| Mode | Description |
|------|-------------|
| Wait | No effect on USART.<br>USART interrupts cause the device to exit from Wait mode. |
| Halt | USART registers are frozen.<br>In Halt mode, the USART stops transmitting/receiving until Halt mode is exited. |

## 20.5 USART interrupts

**Table 47. USART interrupt requests**

| 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 USART interrupt events are connected to two interrupt vectors (see Figure 96).*

   a)   Transmission Complete or Transmit Data Register empty interrupt.

   b)   Idle line detection, Overrun error, Receive data register full, Parity error interrupt, and Noise flag.

   *2   These events generate an interrupt if the corresponding enable control bit is set and the interrupt mask in the CCR register is reset (RIM instruction).*

**Figure 96. USART interrupt mapping diagram**
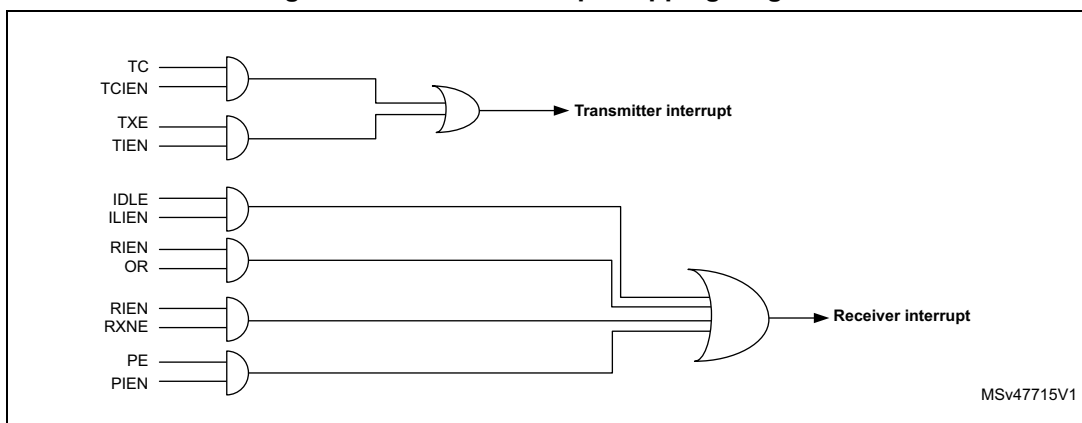
## 20.6 USART registers

### 20.6.1 Status register (USART_SR)

Address offset: 0x00
Reset value: 0xC0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TXE | TC | RXNE | IDLE | OR | NF | FE | PE |
| r | rc_w0 | r | r | r | r | r | r |

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 USART_CR2 register. It is cleared by a write to the USART_DR register.

0: Data is not transferred to the shift register
1: Data is transferred to the shift register

Bit 6 **TC**: Transmission complete

TC bit is set by hardware if the transmission of a frame containing data is complete and TXE bit is set. An interrupt is generated if TCIEN=1 in the USART_CR2 register.

TC bit is cleared either by a software sequence (a read to the USART_SR register followed by a write to the USART_DR register), or by programming the bit to '0'. This clear sequence is recommended only for multibuffer communications.

0: Transmission is not complete
1: Transmission is complete

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 USART_DR register. An interrupt is generated if RIEN=1 in the USART_CR2 register. It is cleared by a read to the USART_DR register.

0: Data is not received
1: Received data is ready to be read.

Bit 4 **IDLE**: IDLE line detected [1]

This bit is set by hardware when an Idle Line is detected. An interrupt is generated if the ILIEN=1 in the USART_CR2 register. It is cleared by a software sequence (a read to the USART_SR register followed by a read to the USART_DR register).

0: No Idle Line is detected
1: Idle Line is detected

Bit 3 **OR**: Overrun error[2]

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 RXNE=1. An interrupt is generated if RIEN=1 in the USART_CR2 register. It is cleared by a software sequence (a read to the USART_SR register followed by a read to the USART_DR register).

0: No Overrun error
1: Overrun error is detected

Bit 2 **NF**: Noise flag [3]

This bit is set by hardware when noise is detected on a received frame. It is cleared by a software sequence (a read to the USART_SR register followed by a read to the USART_DR register).

0: No noise is detected

1: Noise is detected

Bit 1 **FE**: Framing error [4]

This bit is set by hardware when a de-synchronization, excessive noise or a break character is detected. It is cleared by a software sequence (a read to the USART_SR register followed by a read to the USART_DR register).

0: No framing error is detected

1: Framing error or break character is detected

Bit 0 **PE**: Parity error

This bit is set by hardware when a parity error occurs in receiver mode. It is cleared by a software sequence (a read to the status register followed by a read to the USART_DR data register). You have to wait for the RXNE flag to be set before clearing it. An interrupt is generated if PIEN=1 in the USART_CR1 register.

0: No parity error

1: Parity error

1. The IDLE bit is not set again until the RXNE bit has been set itself (i.e. a new idle line occurs)

2. When this bit is set, the RDR register content is not lost but, the shift register is overwritten.

3. This bit does not generate an interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt.

4. This bit does not generate an 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 a frame error and an overrun error, it is transferred and only the OR bit is set.

### 20.6.2 Data register (USART_DR)

Address offset: 0x01
Reset value: 0xXX

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | DR[7:0] | | | | |
| | | | rw | | | | |

Bits 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.

### 20.6.3 Baud rate register 1 (USART_BRR1)

The baud rate registers are common to both the transmitter and the receiver. The baud rate is programmed using two registers BRR1 and BRR2. Writing of BRR2 (if required) should precede BRR1, since a write to BRR1 will update the baud counters.

See *Figure 89: How to code USART_DIV in the BRR registers on page 234* and *Table 42: Baud rate programming and error calculation on page 235*.

*Note:* *The baud counters stop counting if the TEN or REN bits are disabled respectively.*

Address offset: 0x02
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | USART_DIV[11:4] | | | | |
| rw | rw | rw | rw | - | rw | rw | rw |

Bits 7:0 **USART_DIV[11:4]**: USART_DIV bits [1]

These 8 bits define the 2nd and 3rd nibbles of the 16-bit USART divider (USART_DIV).

1. BRR1 = 0x00 means USART clock is disabled.

### 20.6.4 Baud rate register 2 (USART_BRR2)

Address offset: 0x03

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | USART_DIV[15:12] | | | | USART_DIV[3:0] | | |
| | rw | | | | rw | | |

Bits 7:4 **USART_DIV[15:12]**: MSB of USART_DIV
These 4 bits define the MSB of the USART Divider (USART_DIV)

Bits 3:0 **USART_DIV[3:0]**: LSB of USART_DIV
These 4 bits define the LSB of the USART Divider (USART_DIV)

### 20.6.5 Control register 1 (USART_CR1)

Address offset: 0x04
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| R8 | T8 | USARTD | M | WAKE | PCEN | PS | PIEN |
| rw | rw | rw | rw | rw | rw | rw | rw |

Bit 7 **R8**: Receive data bit 8

This bit is used to store the 9th bit of the received word when M=1

Bit 6 **T8**: Transmit data bit 8

This bit is used to store the 9th bit of the transmitted word when M=1

Bit 5 **USARTD**: USART disable (for low power consumption)

When this bit is set the USART prescaler and 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: USART enabled
   1: USART prescaler and outputs disabled

Bit 4 **M**: word length

This bit determines the word length. It is set or cleared by software.

   0: 1 Start bit, 8 Data bits, 'n' STOP bit (n depending on STOP[1:0] bits in the USART_CR3 register)
   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 3 **WAKE**: Wakeup method

This bit determines the USART wakeup method, it is set or cleared by software.

   0: Idle line
   1: Address mark

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 (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCEN is active after the current byte (in reception and in transmission).

   0: Parity control disabled
   1: Parity control enabled

Bit 1 **PS**: Parity selection

This bit selects the odd or even parity when the parity generation/detection is enabled (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 0 **PIEN**: Parity interrupt enable

This bit is set and cleared by software.

   0: Parity interrupt disabled
   1: Parity interrupt is generated whenever PE=1 in the USART_SR register

## 20.6.6 Control register 2 (USART_CR2)

Address offset: 0x05
Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TIEN | TCIEN | RIEN | ILIEN | TEN | REN | RWU | SBK |
| rw | rw | rw | rw | rw | rw | rw | rw |

Bit 7 **TIEN**: Transmitter interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever TXE=1 in the USART_SR register

Bit 6 **TCIEN**: Transmission complete interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever TC=1 in the USART_SR register

Bit 5 **RIEN**: Receiver interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever OR=1 or RXNE=1 in the USART_SR register

Bit 4 **ILIEN**: IDLE Line interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever IDLE=1 in the USART_SR register

Bit 3 **TEN**: Transmitter enable [1] [2]

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

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 1 **RWU**: Receiver wakeup[3] [4]

This bit determines if the USART is in mute mode or not. It is set 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

Bit 0 **SBK**: Send break

This bit set is used to send break characters. It can be set and cleared by software. It should be set by software, and will be reset by hardware during the STOP bit of break.

0: No break character is transmitted

1: Break character will be transmitted

1. During transmission, a "0" pulse on the TEN bit ("0" followed by "1") sends a preamble (idle line) after the current word.

2. When TEN is set there is a 1 bit-time delay before the transmission starts.

3. Before selecting Mute mode (by setting the RWU bit) the USART must first receive a data byte, otherwise it cannot function in Mute mode with wakeup by Idle line detection.

4. In address mark detection wakeup configuration (WAKE bit=1) the RWU bit cannot be modified by software while the RXNE bit is set.

## 20.6.7 Control register 3 (USART_CR3)

Address offset: 0x06

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | STOP[1:0] | | CLKEN | CPOL | CPHA | LBCL |
| | | rw | rw | rw | rw | rw | rw |

Bit 7:6 Reserved

Bits 5:4 **STOP**: STOP bits

These bits are used for programming the STOP bits.

00: 1 STOP bit
01: Reserved
10: 2 STOP bits
11: Reserved

Bit 3 **CLKEN**: Clock enable

This bit allows the user to enable the SCLK pin.

0: SCLK pin disabled
1: SCLK pin enabled

Bit 2 **CPOL**: Clock polarity[1]

This bit allows the user to select the polarity of the clock output on the SCLK pin. It works in conjunction with the CPHA bit to produce the desired clock/data relationship

0: SCLK to 0 when idle
1: SCLK to 1 when idle.

Bit 1 **CPHA**: Clock phase [1]

This bit allows the user to select the phase of the clock output on the SCLK pin. It works in conjunction with the CPOL bit to produce the desired clock/data relationship

0: The first clock transition is the first data capture edge
1: The second clock transition is the first data capture edge

Bit 0 **LBCL**: Last bit clock pulse[1][2]

This bit allows the user to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the SCLK pin.

0: The clock pulse of the last data bit is not output to the SCLK pin.
1: The clock pulse of the last data bit is output to the SCLK pin.

1. These 3 bits (CPOL, CPHA, LBCL) should not be written while the transmitter is enabled.

2. The last bit is the 8th or 9th data bit transmitted depending on the 8 or 9 bit format selected by the M bit in the USART_CR1 register.

## 20.6.8 Control register 4 (USART_CR4)

Address offset: 0x07

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | Reserved | | ADD[3:0] | | | |
| | | | | rw | | | |
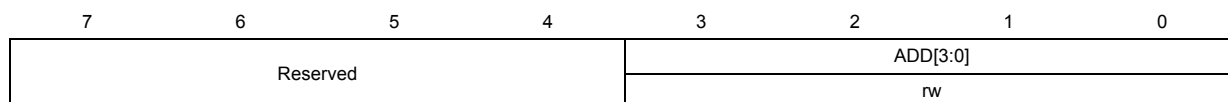
Bit 7:4  Reserved

Bits 3:0  **ADD[3:0]**: Address of the USART node

This bit-field gives the address of the USART node.

This is used in multiprocessor communication during mute mode, for wakeup with address mark detection.

## 20.6.9 USART register map and reset values

**Table 48. USART register map**

| Address offset | Register name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | USART_SR Reset value | TXE 1 | TC 1 | RXNE 0 | IDLE 0 | OR 0 | NF 0 | FE 0 | PE 0 |
| 0x01 | USART_DR Reset value | DR[7:0] xxxxxxxx | | | | | | | |
| 0x02 | USART_BRR1 Reset value | USART_DIV[11:4] 00000000 | | | | | | | |
| 0x03 | USART_BRR2 Reset value | USART_DIV[15:12] 0000 | | | | USART_DIV[3:0] 0000 | | | |
| 0x04 | USART_CR1 Reset value | R8 0 | T8 0 | USARTD 0 | M 0 | WAKE 0 | PCEN 0 | PS 0 | PIEN 0 |
| 0x05 | USART_CR2 Reset value | TIEN 0 | TCIEN 0 | RIEN 0 | ILIEN 0 | TEN 0 | REN 0 | RWU 0 | SBK 0 |
| 0x06 | USART_CR3 Reset value | - 00 | | STOP 00 | | CKEN 0 | CPOL 0 | CPHA 0 | LBCL 0 |
| 0x07 | USART_CR4 Reset value | - 0000 | | | | ADD[3:0] 0000 | | | |

# 21 Comparators (COMP)

## 21.1 COMP introduction

The STM8L features two zero-crossing comparators. They share the same current bias and the same voltage reference. The voltage reference can be internal (comparison with ground) or external (comparison to a reference pin voltage).

Each comparator is connected to 4 channels, which can be used to generate interrupt, timer input capture or timer break. Their polarity can be inverted.

## 21.2 COMP main features

- 2 zero-crossing comparators
- Switches which can either
    - connect any I/O to the comparator input
    - or connect several I/Os to each other and to the comparator input.
- Interrupt generation: one single interrupt is generated for both comparators. It is enabled through flags ITEN1 and ITEN2 of COMP_CSR register (see *Section 21.6.2: Comparator control status register (COMP_CSR)*).
- Input captures or break generation on TIM2
- Input capture generation on TIM3
- A zero-crossing can generate a rising edge or a falling edge on comparator outputs depending on a bit of polarity

The block diagram is shown on *Figure 1*.

**Figure 97. Comparator block diagram**
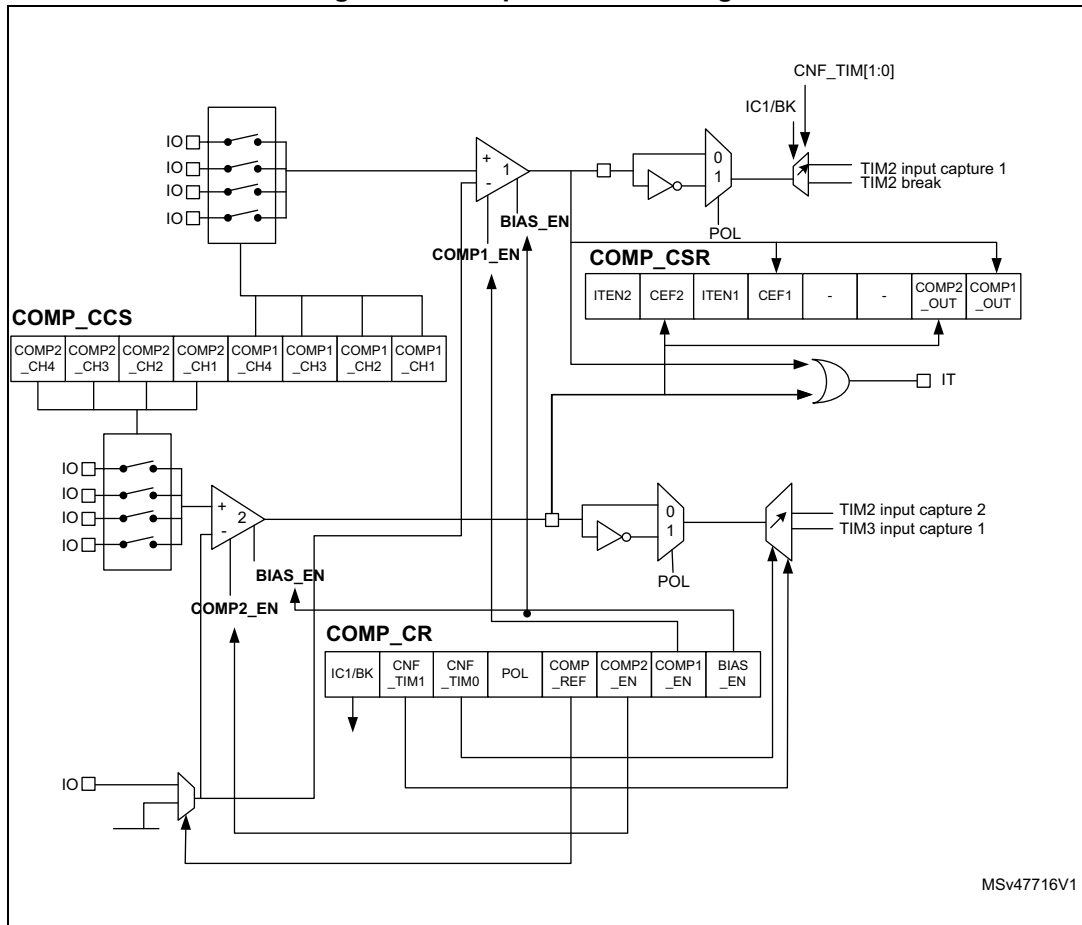
## 21.3 COMP functional description

In order to use the comparators, the application must perform the following steps:

1. Enable the bias of both comparators using the BIAS_EN bit in the COMP_CR register. It must be enabled several microseconds before enabling comparator 1 or 2 through the COMP1_EN or COMP2_EN bits (comparator characteristics).

2. Configure the timers to use the comparators with timer 2 input capture/break and/or timer 3 input capture.

3. If required, perform the following procedures:
   – Select the comparator polarity using the POL bit in the COMP_CR.
   – Select the external reference using the COMP_REF bit in the COMP_CR register (by default, the reference is internal).
   – Configure the interconnection with timer 2 and timer 3 using CNF_TIM[1:0] and IC1/BK in the COMP_CR register.
   – Enable the comparator interrupt using ITEN1 or ITEN2 in the COMP_CSR register.

4. Enable one channel for each comparator using the COMP_CSS register.

5. Enable comparator 1 and/or comparator 2 using COMP1_EN and COMP2_EN bits in the COMP_CR register.

*Note:* *Several switches can be closed at the same time, so as to enable, for instance, charge transfers or resistor bridges.*

## 21.4 Low power modes

**Table 49. Comparator behavior in low power modes**

| Mode | Description |
|------|-------------|
| WAIT | No effect on comparator.<br>Comparator interrupts cause the device to exit from Wait mode. |
| HALT | Comparator registers are frozen. |

## 21.5 Interrupts

**Table 50. Comparator interrupt requests**

| Interrupt event | Event flag | Enable control bit | Exit from Wait | Exit from Halt |
|-----------------|------------|--------------------|----------------|----------------|
| Comparator 1 event flag | CEF1 | ITEN1 | Yes | No |
| Comparator 2 event flag | CEF2 | ITEN2 | Yes | No |

*Note:* *The 2 comparator event interrupts are connected to the same interrupt vector (see the interrupt mapping table in the datasheet ). These events generate an interrupt if the corresponding ITEN bit is set and the interrupt mask in the CC register is reset (RIM instruction).*

## 21.6      COMP registers

### 21.6.1      Comparator control register (COMP_CR)

Address offset: 0x00

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IC1/BK | CNF_TIM1 | CNF_ TIM0 | POL | COMPREF | COMP2_EN | COMP1_EN | BIAS_ EN |
| rw | rw | rw | rw | rw | rw | rw | rw |

Bit 7 **IC1/BK**: Input capture 1/ break selection.

When CNF_TIM1 or CNF_TIM0 is set, if the IC1/BK bit is set, the first comparator output is connected to Timer 2 break. Otherwise it is connected to Timer 2 input capture 1.

0: first comparator output connected to timer 2 input capture 1.
1: first comparator output connected to timer 2 break.

Bits 6:5 **CNF_TIM[1:0]**: Comparator 1/2 output connected to Timer 2 and 3 input capture or break

These bits are used to connect comparator outputs to Timer 2 input capture 1, Timer 2 input capture 2, Timer 2 break or Timer 3 input capture 1.

00: No connection.

01: Comparator 1 sent to timer 2 input capture 1/ break.

10: Comparator 1 sent to timer 2 input capture 1/ break and comparator 2 sent to Timer 2 input capture 2.

11: Comparator 1 sent to timer 2 input capture 1/ break and comparator 2 sent to Timer 3 Input capture 1.

Bit 4 **POL**: Comparator polarity.

This bit configures the comparator polarity. It is common to comparator 1 and 2.

0: Event detected when output comparator is 1.
1: Event detected when output comparator is 0.

Bit 3 **COMP_REF**: Comparator reference.

This bit selects internal or external reference of both comparators. If the reference is internal, the switch of the reference I/O is open.

0: the comparator reference is internal.
1: the comparator reference is external.

Bit 2 **COMP2_EN**: Second comparator enable.

0: comparator 2 disabled.
1: comparator 2 enabled.

Bit 1 **COMP1_EN**: First comparator enable.

0: comparator 1 disabled.
1: comparator 1 enabled.

Bit 0 **BIAS_EN**: Bias enable.

This bit enables the bias used by both comparators.

0: bias disabled.
1: bias enabled.

*Note:      This bit must be enabled several microseconds before setting COMP1_EN or COMP2_EN bits (comparator characteristics).*

## 21.6.2 Comparator control status register (COMP_CSR)

Address offset: 0x01

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ITEN2 | CEF2 | ITEN1 | CEF1 | Reserved | Reserved | COMP2_OUT | COMP1_OUT |
| rw | rc_w1 | rw | rc_w1 | - | - | r | r |

Bit 7 **ITEN2**: Second comparator interrupt enable.

    0: Interrupt is inhibited.
    1: A timer interrupt is generated whenever the CEF2 bit is set.

*Note: It is forbidden to set ITEN2 (interrupt enabled) and clear CEF2 simultaneously.*

Bit 6 **CEF2**: Second comparator event flag.

When a positive edge on the second comparator output occurs, this bit is set. It is cleared writing 1.

    0: No event detected.
    1: Event detected

*Note: It is forbidden to clear CEF2 and set ITEN2 (interrupt enabled) simultaneously.*

Bit 5 **ITEN1**: First comparator interrupt enable.

    0: Interrupt is inhibited.
    1: A timer interrupt is generated whenever the CEF1 is set.

*Note: It is forbidden to set ITEN1 (interrupt enabled) and clear CEF1 simultaneously.*

Bit 4 **CEF1**: First comparator event flag.

When a positive edge on first comparator output occurs, this bit is set. It is cleared writing 1.

    0: No event detected.
    1: Event detected.

*Note: It is forbidden to clear CEF1 and set ITEN1 (interrupt enabled) simultaneously.*

Bits 3:2 Reserved, forced by hardware to 0.

Bit 1 **COMP2_OUT**: Second comparator output.

This bit is the exact copy of second comparator output.

    0: comparator 2 output is 0.
    1: comparator 2 output is 1.

Bit 0 **COMP1_OUT**: First comparator output.

This bit is the image copy of first comparator output.

    0: comparator 2 output is 0.
    1: comparator 2 output is 1.

### 21.6.3 Comparator channel selection (COMP_CCS)

Address offset: 0x02

Reset value: 0x00

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| COMP2_CH4 | COMP2_CH3 | COMP2_CH2 | COMP2_CH1 | COMP1_CH4 | COMP1_CH3 | COMP1_CH2 | COMP1_CH1 |
| rw | rw | rw | rw | rw | rw | rw | rw |

Bits 7:0 **COMPx_CHn**: *Comparator switch enable.*

When COMPx_CHn is set, the switch n (n:1->4) of the comparator x (x: 1 or 2) is enabled

0: Switch open.

1: Switch closed.

### 21.6.4 COMP register map and reset values

**Table 51. Comparator register map**

| Address offset | Register Name | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | COMP_CR Reset value | IC1/BK 0 | CNF_TIM1 0 | CNF_TIM0 0 | POL 0 | COMPREF 0 | COMP1_EN 0 | COMP2_EN 0 | BIAS_EN 0 |
| 0x01 | COMP_CSR Reset value | ITEN2 0 | CEF2 0 | ITEN2 0 | CEF1 0 | - 0 | - 0 | COMP2_OUT 0 | COMP2_OUT 0 |
| 0x02 | COMP_CCS Reset value | COMP2_CH4 0 | COMP2_CH3 0 | COMP2_CH2 0 | COMP2_CH1 0 | COMP1_CH4 0 | COMP1_CH3 0 | COMP1_CH2 0 | COMP1_CH1 0 |

# 22 Revision history

**Table 52. Document revision history**

| Date | Revision | Changes |
|------|----------|---------|
| 09-Apr-2009 | 1 | Initial release. |
| 19-Jun-2009 | 2 | Updated comparator IRQ18 in *Table 30: Interrupt mapping on page 126*.<br>Updated pin connected to the SWIM in *Figure 3: SWIM pin connection*.<br>Added warning in *Section 10.3: Port configuration and usage*.<br>*Section 20: Universal synchronous/asynchronous receiver transmitter (USART)*: 7th step added in *Character transmission* prodedure; updated *Single byte communication* and *Figure 86: TC/TXE behavior when transmitting* added; updated TC bit description in *Section 20.6.1: Status register (USART_SR)*; added *Start bit detection* and *Section 20.3.5: USART receiver's tolerance to clock deviation* in *Section 20.3.3: Receiver* .<br>Updated comparator 2 negative input in *Figure 97: Comparator block diagram*.<br>Rename Infra-red (IR) interface IRTIM in *Section 15*. |
| 10-Sep-2009 | 3 | Modified *Figure 39: Clock/trigger controller block diagram on page 143*, *Figure 49: Timer chaining system implementation example on page 150* and *Figure 50: Trigger/master mode selection blocks on page 150*<br>Modified *Section 13.1: IWDG introduction on page 92*, *Section : Timeout period on page 93*, *Figure 16: Independent watchdog block diagram on page 92*, *Table 24: Min/Max IWDG timeout (LSI clock frequency = 38 kHz) on page 93* and *Section : Using the IWDG in Halt/Active-halt mode on page 93*<br>38 kHz LSI RC instead of 37 kHz LSI RC<br>Modified *Section 17.7.3: Slave mode control register (TIMx_SMCR) on page 174*<br>Modified *Section 18.5.3: Slave mode control register (TIM4_SMCR) on page 195*<br>*Section 19.3.5: Data transmission and reception procedures*: timing diagrams revised and description of receive-only mode expanded.<br>Added *Section 19.3.7: Disabling the SPI*<br>Modified *Section 21.2: Main features on page 268*, *Figure 97: Comparator block diagram on page 251* and *Section 21.3: COMP functional description on page 252*<br>Modified *Section 21.6.1: Comparator control register (COMP_CR) on page 253* (bits 1 and 2)<br>Modified COMPx_CHn bit description in *Section 21.6.3: Comparator channel selection (COMP_CCS) on page 255* |

**Table 52. Document revision history (continued)**

| Date | Revision | Changes |
|---|---|---|
| 25-Jun-2010 | 4 | Added paragraph to *Section 5.6: External interrupts on page 46*<br><br>Added BEEP clock to *Section 8.1.1: Peripheral clock gating (PCG) on page 62* and *Section 8.2: LSI clock on page 63*<br><br>Added reference to BEEP in *Section Table 16.: Low power mode management on page 68*<br><br>Updated AWU *Table 21: Time base calculation table on page 84*<br><br>Added note in *Section 19.3.2: Configuring the SPI in slave mode on page 204*<br><br>Updated I2C *Bus error (BERR) on page 109*<br><br>Updated *I2C_CCR values for SCL frequency table (fMASTER = 10 MHz or 16 MHz) on page 125*<br><br>Updated SPI *Figure 76: Data clock timing diagram on page 203*<br><br>Updated MODF clearing sequence in SPI *Error flags on page 215*<br><br>Updated USART *Baud rate programming and error calculation on page 235*<br><br>Updated *Figure 97: Comparator block diagram on page 251* |
| 22-Jul-2010 | 5 | Modified *Section 7: Reset (RST) and voltage detection on page 57*<br>Added *Section 7.1: "Reset state" and "under reset" definitions on page 57*<br><br>Modified *Section 10.4: Reset configuration on page 76*<br>Modified reset value in *Section 10.9.2: Port x pin input register (Px_IDR) on page 79*, in *Section 10.9.4: Port x control register 1 (Px_CR1) on page 80* and in *Table 20: GPIO register map on page 81*<br>Modified Step 5 in *Section 11.3.1: AWU operation on page 83* |

**Table 52. Document revision history (continued)**

| Date | Revision | Changes |
|------|----------|---------|
| 03-Oct-2017 | 6 | – Updated the scope to the document to include new STM8L001xx devices.<br>– All figures formatting was updated, content of the figures was updated for below items:<br>*Figure 8: Concurrent interrupt management*<br>*Figure 9: Nested interrupt management*<br>– Section's naming updated not impacting the section's content.<br>– Content updated in below sections and tables:<br>*Section 3.3: SWIM modes*<br>*Section 14.7.2: Control register 2 (I2C_CR2)*<br>*Section 2.3.3: Description of global configuration register (CFG_GCR)*<br>*Section 3.2: SWIM main features*<br>*Section 3.3: SWIM modes*<br>*Section 4.4.1: User boot area (UBC)*<br>*Section 4.4.3: Main program area*<br>*Section 4.8.3: Flash program memory unprotecting key register (FLASH_PUKR)*<br>*Section 5.1: ITC introduction*<br>*Section 5.2: Interrupt masking and processing flow*<br>*Section 7.4.1: Reset pin configuration register (RST_CR)*<br>*Section 7.4.2: Reset status register (RST_SR)*<br>*Section 8.3: Configurable clock-output capability (CCO)*<br>*Section 8.4.1: Clock divider register (CLK_CKDIVR)*<br>*Section 9.4.2: Halt mode*<br>*Section 9.5.1: WFE control register 1 (WFE_CR1)*<br>*Section 10.1: Introduction*<br>*Section 10.5: Unused I/O pins*<br>*Section 10.7.2: Interrupt capability*<br>*Section 10.8.2: Slope control*<br>*Section 10.9.2: Port x pin input register (Px_IDR)*<br>*Section 10.9.4: Port x control register 1 (Px_CR1)*<br>*Section 11.4.1: Control/status register (AWU_CSR)*<br>*Section 11.4.2: Asynchronous prescaler register (AWU_APR)*<br>*Section 11.4.3: Timebase selection register (AWU_TBR)*<br>*Section 13.2: IWDG functional description*<br>*Section 13.3.1: Key register (IWDG_KR)*<br>*Section 13.3.2: Prescaler register (IWDG_PR)*<br>*Section 14.4.1: I2C slave mode*<br>*Section 14.4.2: I2C master mode*<br>*Section 14.7.3: Frequency register (I2C_FREQR)*<br>*Section 14.7.5: Own address register MSB (I2C_OARH)*<br>*Section 14.7.7: Status register 1 (I2C_SR1)*<br>*Section 14.7.8: Status register 2 (I2C_SR2)*<br>*Section 14.7.9: Status register 3 (I2C_SR3)* |

**Table 52. Document revision history (continued)**

| Date | Revision | Changes |
|---|---|---|
| 03-Oct-2017 | 6 | – Content updated in below sections and tables (continued)<br>*Section 14.7.11: Clock control register low (I2C_CCRL)*<br>*Section 14.7.12: Clock control register high (I2C_CCRH)*<br>*Section 14.7.13: TRISE register (I2C_TRISER)*<br>*Section 17.7.5: Interrupt enable register (TIMx_IER)*<br>*Section 17.7.6: Status register 1 (TIMx_SR1)*<br>*Section 17.7.7: Status register 2 (TIMx_SR2)*<br>*Section 17.7.22: Output idle state register (TIMx_OISR)*<br>*Section 19.3.1: SPI general description*<br>*Section 19.4.2: SPI control register 2 (SPI_CR2)*<br>*Section 19.4.3: SPI interrupt control register (SPI_ICR)*<br>*Section 19.4.4: SPI status register (SPI_SR)*<br>*Section 20.6.2: Data register (USART_DR)*<br>*Section 20.6.7: Control register 3 (USART_CR3)*<br>*Section 20.6.8: Control register 4 (USART_CR4)*<br>*Table 6: Memory access versus programming method*<br>*Table 13: RST register map and reset values*<br>*Table 14: Peripheral clock gating bits*<br>*Table 18: I/O port configuration summary*<br>*Table 20: GPIO register map*<br>*Table 22: AWU register map*<br>*Table 23: Beeper register map*<br>*Table 24: Min/Max IWDG timeout (LSI clock frequency = 38 kHz)*<br>*Table 33: Glossary of internal timer signals*<br>*Table 36: TIMx register map*<br>*Table 40: SPI register map and reset values*<br>Added *Table 1: List of abbreviations* |

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

Doc ID14400 Rev 6