RM0003
Reference manual

STR750 ARM7TDMI-S®-based microcontroller family

Introduction

This Reference Manual provides complete information for application developers on how to use the STR750 Microcontroller memory and peripherals.

The STR750 is a family of microcontrollers with different memory sizes, packages and peripherals.

For Ordering Information, Mechanical and Electrical Device Characteristics please refer to the STR750 Datasheet.

For information on programming, erasing and protection of the internal Flash memory please refer to the STR7 Flash Programming Reference Manual

For information on the ARM7TDMI-S core, please refer to the ARM7TDMI-S Technical Reference Manual.

Related documents

Available from www.arm.com:

ARM7TDMI-S Rev. 4 Technical Reference Manual ARM DDI 0234A

Available from www.st.com:

STR750 Datasheet

STR7 Flash Programming Manual

Contents

1 Memory and bus architecture

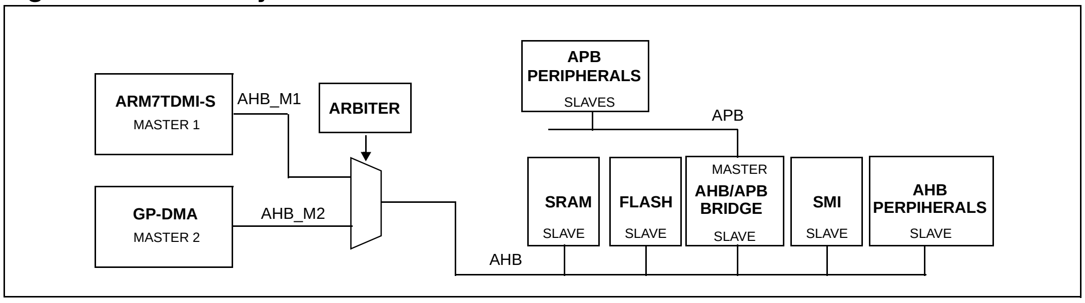
1.1 Multi-Layer AHB bus architecture

The main system consists of:

- Two masters:
  - ARM7TDMI-S CPU
  - GP-DMA (General Purpose DMA)
- Five slaves:
  - Internal SRAM
  - Internal FLASH
  - External SMI (Serial Memory Interface)
  - AHB peripheral register interface
  - AHB to APB bridge which connects all the APB peripherals

These are interconnected using a Multi-Layer AHB bus architecture as shown in *Figure 1*:

Figure 1. Multi-Layer AHB bus architecture



The arbitration is implemented in the following way:

- A round robin algorithm is used to arbitrate the two masters
- The arbitration is evaluated after each bus BURST or SEQUENTIAL AHB transfer (a BURST transfer is never cut).

This Bus Matrix achieves "zero idle" cycle access from ARM CPU or DMA to SRAM or FLASH even for nonsequential accesses.

However, the following frequency limitations apply:

- When accessing the flash:
  - when not configured in BURST mode, the Flash induces no wait states, but the maximum frequency is limited to 32 MHz
  - when configured in BURST mode, the Flash induces 1 wait state for each non sequential access and the maximum frequency is 60 MHz
- The SRAM can be accessed up to 64 MHz with no wait states.

Note: It is up to the user application to ensure that the conditions listed above are respected.

Behavior in ARM DEBUG mode

When entering DEBUG state (CPU is halted), the Arbiter locks the bus to the ARM7TDMI-S. In the case where the DMA is owner of the bus, then the Arbiter waits for the current transfer before locking the bus to the ARM7TDMI-S.

1.2 Memory organization

Program memory, data memory, registers and I/O ports are organized within the same linear address space of 4 GBytes.

The bytes are treated in memory as being in Little Endian format. The lowest numbered byte in a word is considered the word’s least significant byte and the highest numbered byte the most significant.

Figure 2 on page 18 shows the STR750 Memory Map. For the detailed mapping of peripheral registers, please refer to the related chapters.

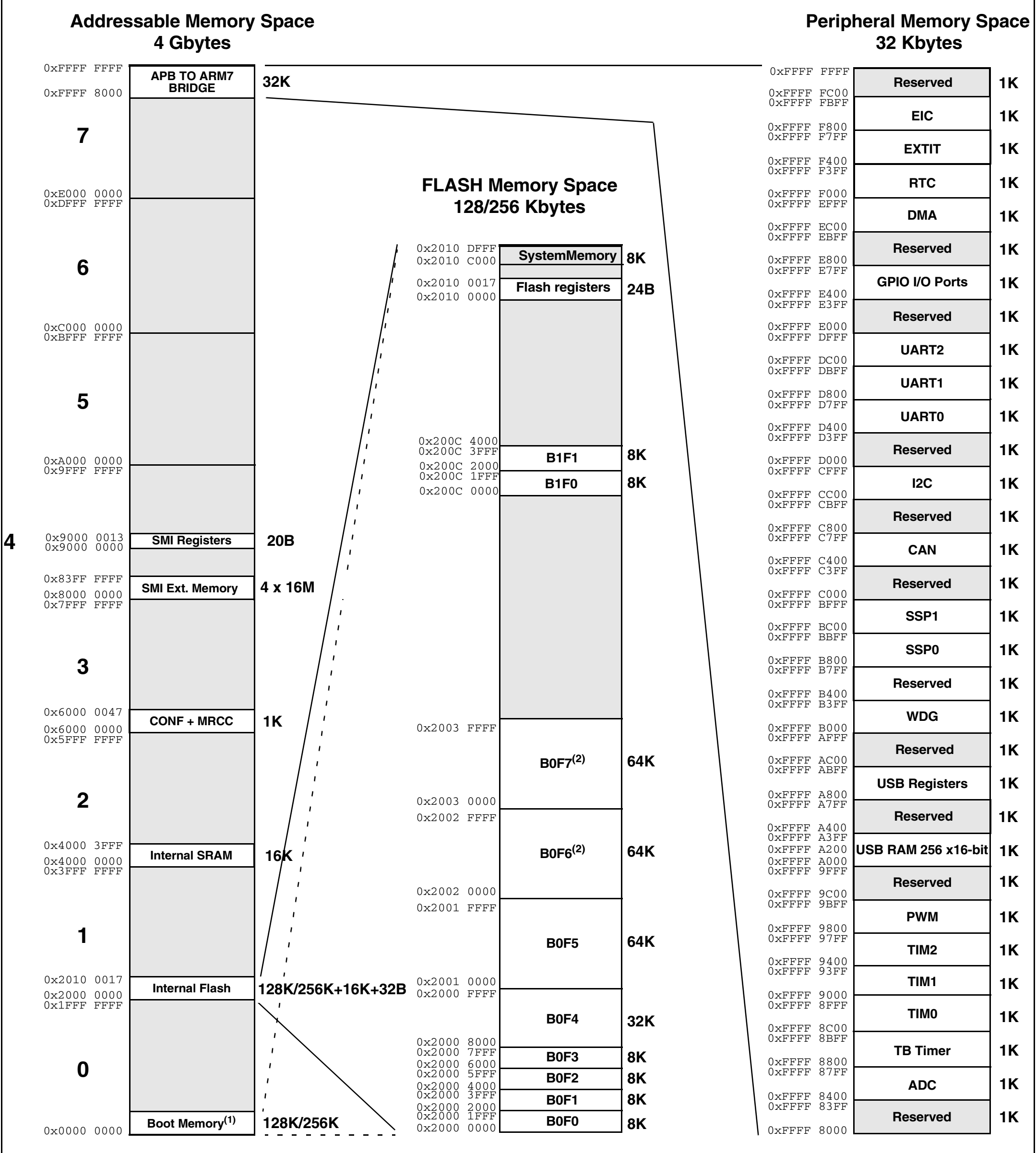
The addressable memory space is divided into 8 main blocks, selected by the three most significant bits of the memory address bus A[31:0]:

- 000 = Boot Memory: depending on the Boot configuration, it can be aliased to Embedded Flash Memory, embedded Boot loader or Embedded SRAM Memory or External Serial Memory
- 001 = Embedded Flash Memory
- 010 = Embedded SRAM Memory
- 011 = Global Configuration Register and MCU Reset and Clock Control (MRCC)
- 100 = External Serial Memory (SMI)
- 101 = Reserved
- 110 = Reserved
- 111 = Peripheral Registers

All memory spaces that are not allocated to on-chip memories and peripherals are considered “reserved” (gray shaded areas in the *Figure 2 on page 18*).

1.3 Memory map

Figure 2. Memory map



(1) In internal Flash Boot Mode, internal FLASH is aliased at 0x0000 0000h

(2) Only available in STR750Fx2

▭ Reserved

1.3.1 Register base addresses

Table 1. Main Block Register base addresses

STR750 Main Blocks	Bus	Base Address	Block Register Map
Flash Registers	AHB	0x2010 0000	See STR7 Flash Programming Reference Manual
Configuration Register (CFG)	AHB	0x6000 0000	*Section 1.5.2*
MCU Reset and Clock Control (MRCC)	AHB	0x6000 0020	*Section 2.10.8*
Serial Memory Interface Registers (SMI)	AHB	0x9000 0000	*Section 6.6*

1.3.2 Peripheral memory map

Table 2. Peripheral memory map

Sub Page	SubPage Boundary Addresses	Peripheral	Bus	Peripheral Boundary Addresses	Bus Access Width	Register Map
0	0xFFFF 8000	reserved	-			
	0xFFFF 83FF					
1	0x FFFF 8400	ADC	APB	0x FFFF 8400	16-bit	*Section 17.5*
	0x FFFF 87FF			0xFFFF 848F		
2	0x FFFF 8800	TB Timer	APB	0x FFFF 8800	16-bit	*Section 9.7*
	0x FFFF 8BFF			0xFFFF 88BF		
3	0x FFFF 8C00	TIM Timer 0	APB	0x FFFF 8C00	16-bit	*Section 10.11*
	0x FFFF 8FFF			0xFFFF 8C5F		
4	0x FFFF 9000	TIM Timer 1	APB	0x FFFF 9000	16-bit	*Section 10.11*
	0x FFFF 93FF			0xFFFF 905F		
5	0x FFFF 9400	TIM Timer 2	APB	0x FFFF 9400	16-bit	*Section 10.11*
	0x FFFF 97FF			0x FFFF 945F		
6	0x FFFF 9800	PWM	APB	0x FFFF 9800	16-bit	*Section 11.11*
	0x FFFF 9BFF			0x FFFF 985F		
7	0x FFFF 9C00	reserved	-			
	0x FFFF 9FFF					
8	0x FFFF A000	USB RAM	APB		16-bit	
	0x FFFF A3FF					
9	0x FFFF A400	reserved	-			
	0x FFFF A7FF					
10	0x FFFF A800	USB Registers	APB	0x FFFF A800	16-bit	*Section 16.7*
	0x FFFF ABFF			0x FFFF A853		

Table 2. Peripheral memory map (continued)

Sub Page	SubPage Boundary Addresses	Peripheral	Bus	Peripheral Boundary Addresses	Bus Access Width	Register Map
11	0x FFFF AC00 0x FFFF AFFF	reserved	-	0x FFFF AC00 0x FFFF AFFF	16-bit	
12	0x FFFF B000 0x FFFF B3FF	WDG	APB	0x FFFF B000 0x FFFF B01B	16-bit	*Section 8.5*
13	0x FFFF B400 0x FFFF B7FF	reserved	-			
14	0x FFFF B800 0x FFFF BBFF	SSP0	APB	0x FFFF B800 0x FFFF B827	16-bit	*Section 14.4*
15	0x FFFF BC00 0x FFFF BFFF	SSP1	APB	0x FFFF BC00 0x FFFF BC27	16-bit	*Section 14.4*
16	0x FFFF C000 0x FFFF C3FF	reserved	-			
17	0x FFFF C400 0x FFFF C7FF	CAN	APB	0x FFFF C400 0x FFFF C57F	16 bit	*Section 12.6*
18	0x FFFF C800 0x FFFF CBFF	reserved	-			
19	0x FFFF CC00 0x FFFF CFFF	I2C	APB	0x FFFF CC00 0x FFFF C01F	8-bit	*Section 13.6*
20	0x FFFF D000 0x FFFF D3FF	reserved	-			
21	0x FFFF D400 0x FFFF D7FF	UART0	APB	0x FFFF D400 0x FFFF D44B	16 bit	*Section 15.5*
22	0x FFFF D800 0x FFFF DBFF	UART1	APB	0x FFFF D800 0x FFFF D84B	16 bit	*Section 15.5*
23	0x FFFF DC00 0x FFFF DFFF	UART2	APB	0x FFFF DC00 0x FFFF DC4B	16 bit	*Section 15.5*
24	0x FFFF E000 0x FFFF E3FF	reserved	-			

Table 2. Peripheral memory map (continued)

Sub Page	SubPage Boundary Addresses	Peripheral	Bus	Peripheral Boundary Addresses	Bus Access Width	Register Map
25	0x FFFF E400 0x FFFF E7FF	GP I/O - Port 0	APB	0x FFFF E400	32 bit	*Section 3.2.8*
				0x FFFF E413		
		GP I/O Remap	APB	0x FFFF E420		
				0x FFFF E427		
		GP I/O - Port 1	APB	0x FFFF E440		
				0x FFFF E453		
		GP I/O - Port 2	APB	0x FFFF E480		
				0x FFFF E493		
26	0x FFFF EC00	DMA	AHB	0x FFFF EC00	16 bit	*Section 5.6*
	0x FFFF EFFF			0x FFFF ECFF		
27	0x FFFF F000	RTC	APB	0x FFFF F000	16 bit	*Section 7.5*
	0x FFFF F3FF			0x FFFF F027		
26	0x FFFF F400	EXTIT	APB	0x FFFF F400	32 bit	*Section 4.8.7*
	0x FFFF F7FF			0x FFFF F40F		
27	0x FFFF F800	EIC	AHB	0x FFFF F800	32 bit	*Section 4.7*
	0x FFFF FFFF			0x FFFF F8DF		

1.3.3 Embedded SRAM

The STR750 features 16 KBytes of static SRAM. It can be accessed as bytes, half-words (16 bits) or full words (32 bits). The SRAM start address is 0x4000 0000.

You can remap the SRAM on-the-fly to Block 000, using the SW_BOOT bits in the CFG_GLCONF register. In SRAM boot mode, the RAM start address is mapped both at 0x0000 0000h and at 0x4000 0000h.

This is particularly useful for managing interrupt vectors and routines, you can copy them to SRAM, modify and access them even when Flash is not available (i.e. during Flash programming or erasing).

1.3.4 Embedded Flash

The high performance Flash Module has the following key features:

- 256 KB Single Voltage Flash Program Memory + 16 KB parameters
- Dual bank architecture for Read-While-Write (RWW) capability
  - Bank 0 : up to 256KB organized in 8 sectors for program memory + SystemMemory sector for ICP
  - Bank 1 : 16 KB for parameters (reprogrammable by the software stored in Bank 0)
- Access at up to 32 MHz with zero wait states when burst mode not used
- Access at up to 60 MHz using burst mode (60 MHz with zero wait states for sequential access and with one wait state for random accesses)
- Access up to up 16 MHz for Read While Write (RWW) accesses (when code executed from Bank 0 reprograms Bank 1).
- Readout protection (program code can only be executed but cannot be dumped out)
- Independent Write protection for each sector of the two banks
- In Circuit Programming (ICP) for programming the embedded flash without using JTAG & Debug features. This is performed by using the embedded Boot Loader located in the Flash SystemMemory sector
- In Application Programming for reprogramming any flash sectors except the boot sector by the user program itself.

The embedded Flash Module consists of two banks (Bank 0 and Bank 1) and is organized in sectors as shown in *Table 3*.

Table 3. Flash module organisation

Bank	Sector	Addresses	Size (bytes)
Bank 0 256 Kbytes Program Memory	Bank 0 Flash Sector 0 (B0F0)	0x00 0000 - 0x00 1FFF	8K
	Bank 0 Flash Sector 1 (B0F1)	0x00 2000 - 0x00 3FFF	8K
	Bank 0 Flash Sector 2 (B0F2)	0x00 4000 - 0x00 5FFF	8K
	Bank 0 Flash Sector 3 (B0F3)	0x00 6000 - 0x00 7FFF	8K
	Bank 0 Flash Sector 4 (B0F4)	0x00 8000 - 0x00 FFFF	32K
	Bank 0 Flash Sector 5 (B0F5)	0x01 0000 - 0x01 FFFF	64K
	Bank 0 Flash Sector 6 (B0F6)	0x02 0000 - 0x02 FFFF	64K(1)
	Bank 0 Flash Sector 7 (B0F7)	0x03 0000 - 0x03 FFFF	64K(1)
Bank 1 16 Kbytes Data Memory	Bank 1 Flash Sector 0 (B1F0)	0x0C 0000 - 0x0C 1FFF	8K
	Bank 1 Flash Sector 1 (B1F1)	0x0C 2000 - 0x0C 3FFF	8K
Flash Control Registers	Flash Control/Data Registers	0x10 0000 - 0x0010 0017	24
System Flash Memory	Boot Loader	0x10 C000 - 0x0010 C7FF	2K
	Reserved	0x10 C800 - 0x0010 DFFF	6K

1. Not available in 128K versions.

RWW operation

The user must respect the following conditions for proper RWW operation:

- The CPU clock frequency is maximum 16 MHz and burst mode must be disabled.

Flash programming

You can program Flash memory using In-Circuit Programming (ICP) and In-Application programming (IAP).

ICP can be performed in two ways :

- using the JTAG interface : the debugger tool can re-program the Flash though the JTAG interface by:
  - Halting the CPU
  - Downloading the SRAM with the flash program/erase routines
  - Jump to the SRAM to execute these routines which reprogram the flash.

    This can be done whatever the selected Boot mode.
- Using a UART : it is possible to re-program the flash using a simple serial interface (UART) when SystemMemory Boot mode is selected.

IAP can be performed directly by the user program : the user is free to choose any protocols using any available communication peripherals to provide the datas to reprogram some sectors of the flash. For this, the flash program/erase routines must be embedded in the user code.

Caution: In IAP mode it is highly recommended to write protect the boot sector (B0F0) to avoid any unrecoverable situations (like erasure of the boot sector)

Flash protection

The embedded Flash memory can be protected against unwanted access (read/write/erase).

The protection bits are stored in non-volatile Flash cells.

Two kind of protections are available: sector *Write protection* to prevent unwanted write access and *Readout Protection* to avoid software piracy:

- Write protection: Each sectors can be independently write protected by writing the corresponding bit of a non-volatile register (FLASH_FNVWPAR).

The write protection will be permanent after each Reset, but can be temporarily unprotected by a dedicated sequence.

- Readout protection: This is done by writing to the READOUT protection bit (bit 0 in the non-volatile FLASH_NVAPR0 register).
- If READOUT protection is enabled, the Debug features and JTAG pins are disabled, any ICE connection will be lost.
  In addition, depending on the BOOT mode:
  - In **Embedded Flash Boot Mode**: the FLASH is kept enabled and the user program will execute normally. But the debugger host can not take the control over the CPU. In addition, if a JTAG sequence is initiated (by simply releasing the NJTRST) then the Flash is automatically disabled and the user program execution

will fail. The user has to ensure that the software does not provide a way for a hacker to download the Flash memory content.

– In **Embedded SRAM** or **external Serial Memory (SMI) Boot Mode** then the embedded Flash is automatically disabled and thus it is not possible to download its content (it is also not possible to execute it).

– In **SystemMemory Boot Mode**, then the program executed from SystemMemory ensures that it is not possible to output the embedded Flash memory contents.

Readout protection can be disabled by programming non-volatile register PDSx bits executed by a Flash user routine, and re-enabled again by programming the PENx non-volatile register bits. This operation can be performed up to 16 times.

Note: Refer to the STR7 Flash programming manual for more details. For STR750 Flash Protection, the READOUT bit (reserved bit 0) must be programmed instead of the DEBUG bit (bit 1) in the FLASH_FNVWPAR register.

Flash Burst mode

By default, the CPU accesses the FLASH without any wait states (Flash Burst mode disabled).

However, to achieve higher frequency, it is possible to enable the Flash Burst mode using the FLASH_BURST bit in the CFG_GLCONF register.

In Burst mode, sequential accesses are performed with zero wait states at speeds of up to the maximum device frequency while non sequential accesses are performed with 1 wait state.

Flash in low power modes

In **STOP mode** (See *Section 2.9.5*), the Flash automatically reduces its power consumption.

If you need lower power consumption, you can disable the Flash when entering STOP or WFI Mode. You do this by configuring the LP_PARAM14 bit in the MRCC_PWRCTL register. The consumption is significantly reduced, but after wake-up from low power, a delay is inserted automatically to ensure the Flash is operational before the CPU restarts.

In **WFI mode** (See *Section 2.9.4*) the Flash remains enabled. If the Flash is used in non-burst mode, you can choose to disable the Flash in WFI mode by configuring the LP_PARAM14 bit in the MRCC_PWRCTL register.

1.4 Boot configuration

In the STR750, 5 different boot modes can be selected by means of the BOOT[1:0] pins as shown in *Table 4*.

Table 4. Boot modes

BOOT Mode Selection Pins		Boot Mode	Aliasing	Note
BOOT1	**BOOT0**			
0	0	Embedded Flash	Embedded FLASH sector B0F0 mapped at 0h	All FLASH sectors accessible except SystemMemory sector
1	0	Embedded SRAM	Embedded SRAM mapped at 0h	
0	1	SystemMemory	SystemMemory mapped at 0h	-
1	1	External SMI	SMI Bank 0 mapped at 0h	-

This aliases the physical memory associated with each boot mode to Block 000 (boot memory). The value of the BOOT pin is latched on the 4th rising edge of CK_SYS after Reset. It is up to the user to manage the BOOT1 and BOOT0 pins at reset release to select the required boot mode.

Note: The BOOT pins are also re-sampled when exiting from STANDBY mode. Consequently they must be kept in the required Boot mode configuration in STANDBY mode.

Even when aliased in the boot memory space, the related memory (FLASH, SRAM or SMI) is still accessible at its original memory space.

After this start-up delay has elapsed, the ARM CPU will start code execution from the boot memory space, located at the bottom of the memory space starting from 0x0000_0000h.

The application can read the status of the boot pins that was latched at start-up and change the memory aliasing on-the-fly by modifying the SW_BOOT bits in the CFG_GLCONF register.

1.4.1 Embedded boot loader mode

Embedded Boot Loader Mode is used to re-program the FLASH using one of the serial interfaces (typically a UART). This program is located in the SystemMemory is called ICP boot loader” and is programmed by ST during production.

Refer to the STR7 Family Flash Programming Reference Manual for details.

1.4.2 External memory (SMI) boot mode

When SMI boot mode is selected the Serial Memory Interface is automatically configured as follows:

- Chip Select Polarity = low
- SMI bank 0 is selected and the associated I/O alternate functions are enabled.
- Boot Space (0000_0000h to 00FF_FFFFh -16MB) is aliased to SMI bank 0.
- The SMI is configured in “NORMAL READ MODE” (reset value)
- The SMI_PRESCALER is set to “2” (reset value)

Programming considerations when booting from SMI

After RESET, the PLL is disabled and CK_SYS and HCLK are clocked by the internal FREEOSC oscillator (~5MHz). Consequently, the SMI clock output is ~2.5MHz (SMI_PRESCALER reset state is 2).

To use a higher frequency, software has to configure the clock & PLLs as described in *Section 2.8.9: System startup on page 48*.

Care is needed if the program performing the PLL & clock configuration is executed directly from serial memory. The software must ensure that a proper clock frequency is provided to the serial memory when changing the SMI_PRESCALER and switching the system clock. That’s why the SMI_PRESCALER must be changed first before switching the system clock to the PLL output clock.

For example, to use the SMI in "NORMAL READ MODE" with a 60 MHz HCLK frequency: Set the SMI_PRESCALER to 4 before switching HCLK to 60 MHz. The SMI clock frequency is then 60 MHz / 4 = 15 MHz.

It is possible to obtain the highest SMI frequency, using "FAST READ MODE" if the serial memory supports this mode. For instance, f_{HCLK} can be set to 48 MHz and the SMI_PRESCALER can be be loaded to "1" to address a high speed Serial Memory at 48 MHz (for read only).

Note: Make sure that the SMI clock frequency (resulting from of your AHB clock and SMI_PRESCALER settings) does not exceed the maximum allowed value. The maximum frequency of the SMI is limited by the I/O speed.

Take care to load the SMI_PRESCALER and the FAST_READ mode with the same write transaction. This can be done by a program which is executed from the serial Flash. In this case, the SMI will change the clock and the READ MODE only at the end of this access.

For more details refer to *Section 6.4.6 on page 155*.

Note: The SMI only supports execution in ARM 32-bit mode (Thumb mode not supported).

1.5 Configuration Register (CFG)

1.5.1 Global Configuration Register (CFG_GLCONF)

Address offset: 0x10
Access : 2 wait states
Reset value: depends on BOOT configuration

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DEVICEID[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						USB_FILTEN	FLASH_BURST	FLASH_BUSY	Reserved					SW_BOOT [1:0]	
						rw	rw	r						rw	rw

Bits 31:16	**DEVICEID[15:0]:** *Device identifier code* These bits contain the STR750 device identfier code (0750h).
Bits 15:10	Reserved, always read as 0.
Bit 9	**USB_FILTEN:** *USB Filter Enable* 0: Filter disabled 1: Filter enabled in USB SUSPEND mode, glitches on USB pins are filtered. Refer to *Section 16.5.5 on page 390* for more details.
Bit 8	**FLASH_BURST:** *Flash Burst Mode enable* 0: Burst mode disabled 1: Embedded Flash is in Burst mode, with 1-cycle access for sequential accesses and 2-cycle access for random access.
Bit 7	**FLASH_BUSY:** *Flash Busy* 0: Flash not busy 1: The Flash is being programmed or erased and is not ready to be used.
Bit 6:3	Reserved, always read as 0.

Bit 2	Reserved.
Bits 1:0	**SW_BOOT[1:0]**: *Boot Space Status/Control* When read, these bits reflect the boot configuration selected at start-up by the external BOOT[1:0] pins as described in *Section 1.4* and the Boot Configuration column of *Table 5*. They can be written to change the aliasing of the boot space as described in the Boot Space Aliasing column of *Table 5* **Note:** When the application selects SMI boot mode (by configuring the BOOT pins at start-up): – SMI Bank 0 is enabled and selected, and – SMI_CS0, SMI_CK, SMI_DIN, and SMI_DOUT alternate functions are automatically enabled on I/O ports P0.04, P0.05, P0.06 and P0.07 respectively. However, if you change the aliasing of the boot space by writing to the SW_BOOT bits while the SMI_EN bit in the GPIO_REMAP0 register and BE[0] bit in SMI_CR1 register are still ‘0’, this disables SMI Bank 0 and the SMI alternate functions. To keep SMI Bank 0 and the SMI alternate functions active, use the following sequence: – Write '1' in the BE[0] bit in the SMI_CR1 register – Write '1' in the SMI_EN bit in the GPIO_REMAP0 register – Modify the SW_BOOT bits as required.

Table 5. **Boot space configuration/aliasing**

Bit Value		**Meaning**	
SW_BOOT		**Boot Configuration**	**Boot Space Aliasing (0000 0000 - 1FFF FFFF)**
1	**0**		
0	0	Embedded Flash	0000 2000 -1FFF FFFFh = not used 0000 0000 - 0000 1FFFh = Embedded Flash Bank 0 Sector 0 (8 KB)
0	1	SystemMemory	0000 2000 - 1FFF FFFFh = not used 0000 0000- 0000 1FFFh = SystemMemory Bank 0 Sector 0 (8 KB)
1	0	Embedded SRAM	0000 4000 - 1FFF FFFFh = not used 0000 0000 - 0000 3FFFh = internal SRAM (16 KB)
1	1	External SMI	0100 0000 - 1FFF FFFFh = not used 0000 0000 - 00FF FFFFh = SMI Bank (16 MB)

1.5.2 CFG Register map

Table 6. **CFG Register map**

Addr. Offset	**Register Name**	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
10h	CFG_GLCONF	Global Configuration Register																															

See *Table 1* for base address.

2 Power, Reset and Clocks

2.1 Introduction

You can connect the device in any of the following ways depending on your application:

- Power Scheme 1: Single external 3.3V power source
- Power Scheme 2: Dual external 3.3V and 1.8V power sources
- Power Scheme 3: Single external 5.0V power source
- Power Scheme 4: Dual external 5.0V and 1.8V power sources

2.2 Power supplies

The device has five power pins:

- V_{DD_IO}: power supply for I/Os (3.3V ±0.3V or 5V ±0.5V). Must be kept on, even in STANDBY mode.

Two embedded regulators are available to supply the internal 1.8V digital power:

- V_{18} (pins V_{18REG} and V_{18} which are internally shorted): Power Supply for Digital, SRAM and Flash: 1.8V ± 0.15V.
- V_{18_BKP}: Backup Power Supply for STANDBY or STOP Mode

V_{18} and V_{18_BKP} are normally generated internally by the embedded regulators:

The Main Voltage Regulator (MVREG) supplies V_{18} and $V_{18\_BKP.}$ It delivers a power supply of 1.8V.

The Low Power Voltage Regulator (LPVREG) can supply V18_BKP or V18 in STOP or STANDBY mode (see section Figure 2.9: Low Power modes on page 57). It delivers a power supply of approximately 1.4V

The Low Power Voltage Regulator (LPVREG) can supply V_{18_BKP} or V_{18} in STOP or STANDBY mode (see section *Figure 2.9: Low Power modes on page 57*). It delivers a power supply of 1.6V ± 0.2V

When the embedded regulators are used, the Main Digital part of the chip (V_{18}) can be powered-off (meaning V_{18} left hi-Z) while keeping the backup circuitry powered on (V_{18_BKP}).

It is also possible to supply V_{18} and V_{18_BKP} externally.

Two sensitive analog blocks have dedicated power pins:

- V_{DDA_PLL}: Analog Power supply for PLL (must have the same voltage level as V_{DD_IO})
- V_{DDA_ADC}: Analog Power supply for ADC (must have the same voltage level as V_{DD_IO})

2.3 Power supply schemes

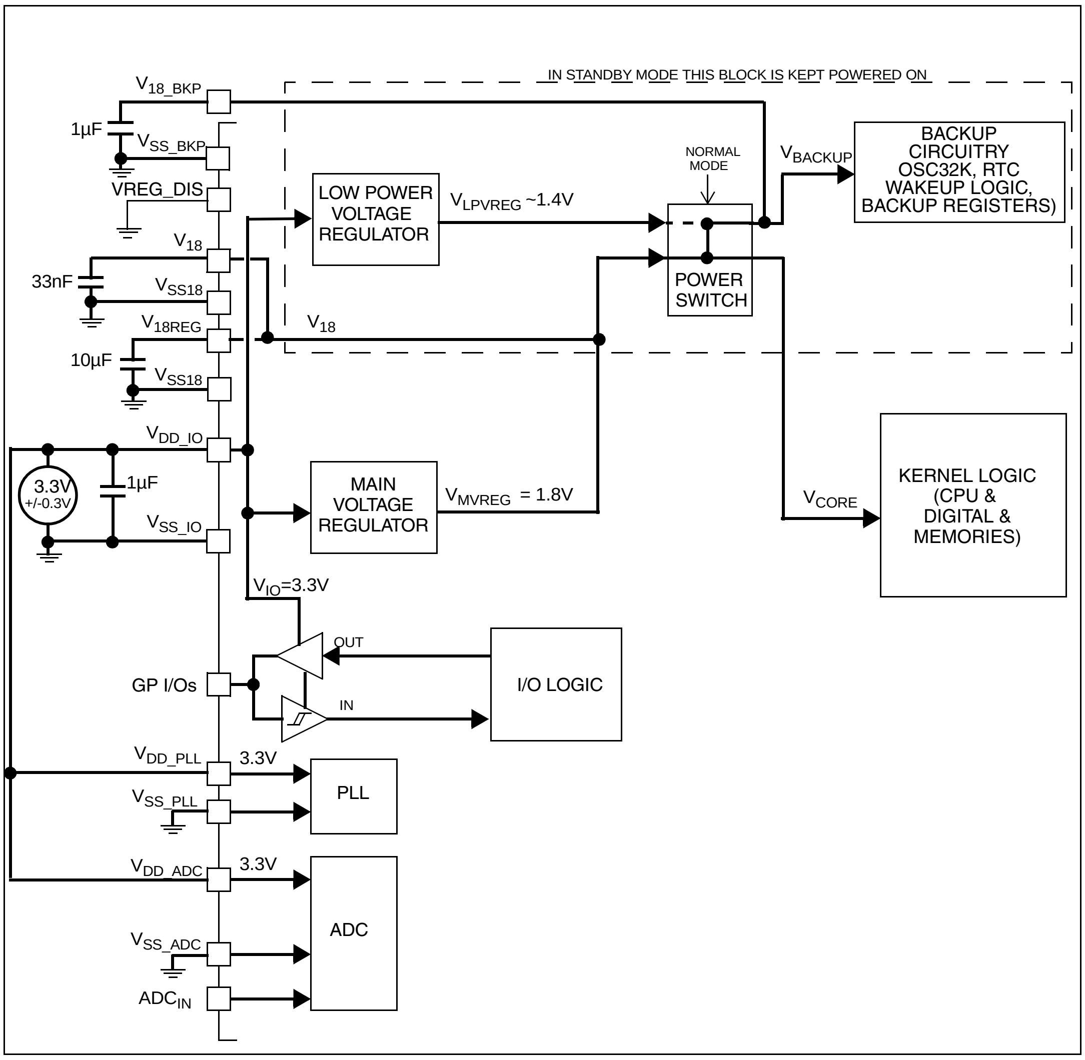
2.3.1 Power scheme 1: Single external 3.3V power source

In this configuration, the internal voltage regulators are switched on by forcing the VREG_DIS pin to low level. The V_{CORE} supply required for the kernel logic and the V_{BACKUP} supply required for the backup circuitry are generated internally by the Main Voltage Regulator or the Low Power Voltage Regulator (depending on the selected low power mode). This scheme has the advantage of requiring only one power source. Refer to *Figure 3*.

At power-up and during Normal Mode (all operating modes except STOP and STANDBY Modes):

- The Main Voltage Regulator powers both V_{CORE} supply required for the kernel logic and the V_{BACKUP} supply required for the backup circuitry.
- The Low Power Regulator is not used

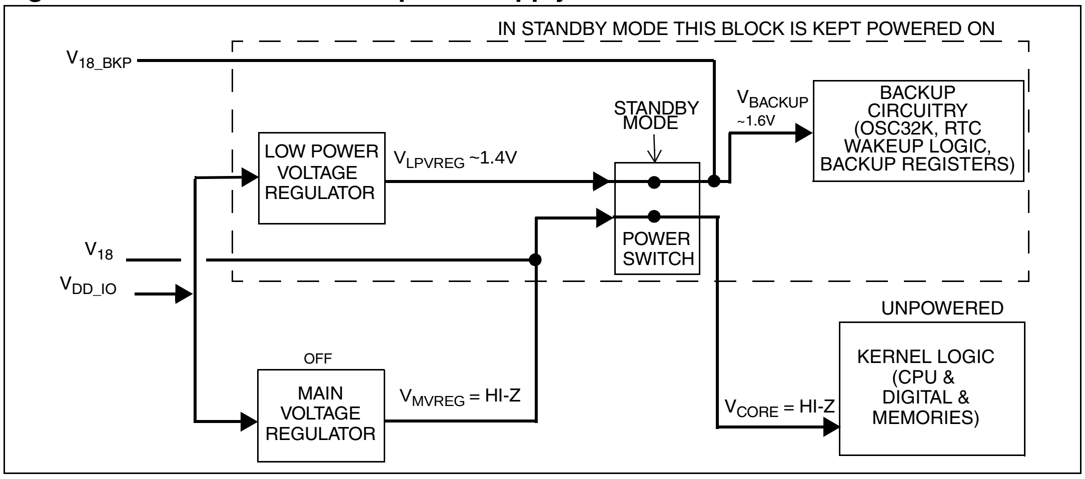
Figure 3. Power supply scheme 1 (single 3.3V supply VREGDIS=0) in NORMAL mode



In STANDBY mode (see *Section 2.9.6*):

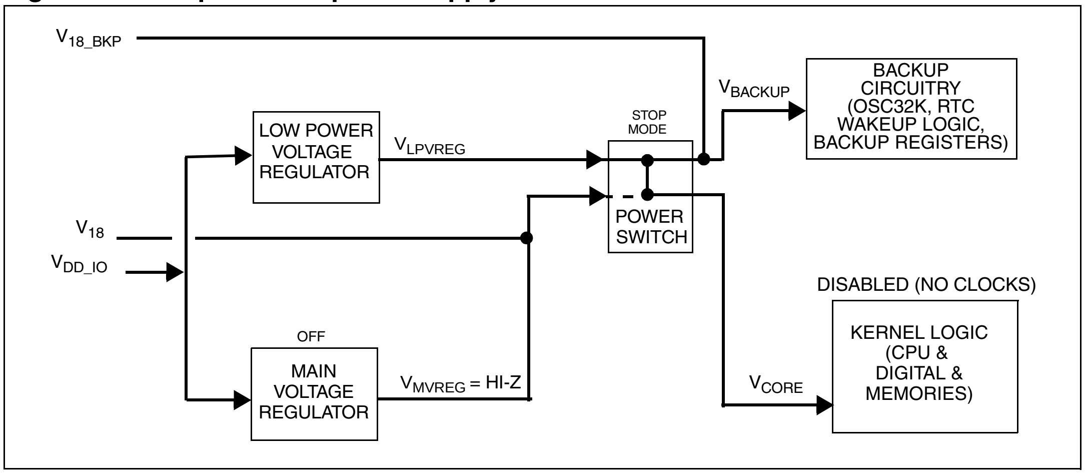
- the Main Voltage Regulator is disabled (output V_{CORE} is hi-Z), the kernel is powered-off
- the Low Power Voltage Regulator powers the backup circuitry.

Figure 4. STANDBY mode in power supply scheme 1


IN STANDBY MODE THIS BLOCK IS KEPT POWERED ON
V18_BKP
STANDBY MODE
VBACKUP ~1.6V
BACKUP CIRCUITRY (OSC32K, RTC WAKEUP LOGIC, BACKUP REGISTERS)
LOW POWER VOLTAGE REGULATOR
VLPVREG ~1.4V
POWER SWITCH
V18
VDD_IO
UNPOWERED
OFF
MAIN VOLTAGE REGULATOR
VMVREG = HI-Z
VCORE = HI-Z
KERNEL LOGIC (CPU & DIGITAL & MEMORIES)

In STOP mode (where all clocks are disabled), it is possible to disable the Main Voltage Regulator and to power both the backup circuitry and the kernel logic with the Low Power Voltage Regulator (see *Section 2.9.5*)

Figure 5. Stop mode in power supply scheme 1 when MVREG is off


V18_BKP
VBACKUP
BACKUP CIRCUITRY (OSC32K, RTC WAKEUP LOGIC, BACKUP REGISTERS)
STOP MODE
LOW POWER VOLTAGE REGULATOR
VLPVREG
POWER SWITCH
V18
VDD_IO
DISABLED (NO CLOCKS)
OFF
MAIN VOLTAGE REGULATOR
VMVREG = HI-Z
VCORE
KERNEL LOGIC (CPU & DIGITAL & MEMORIES)

2.3.2 Power scheme 2: dual external 3.3V and 1.8V power sources

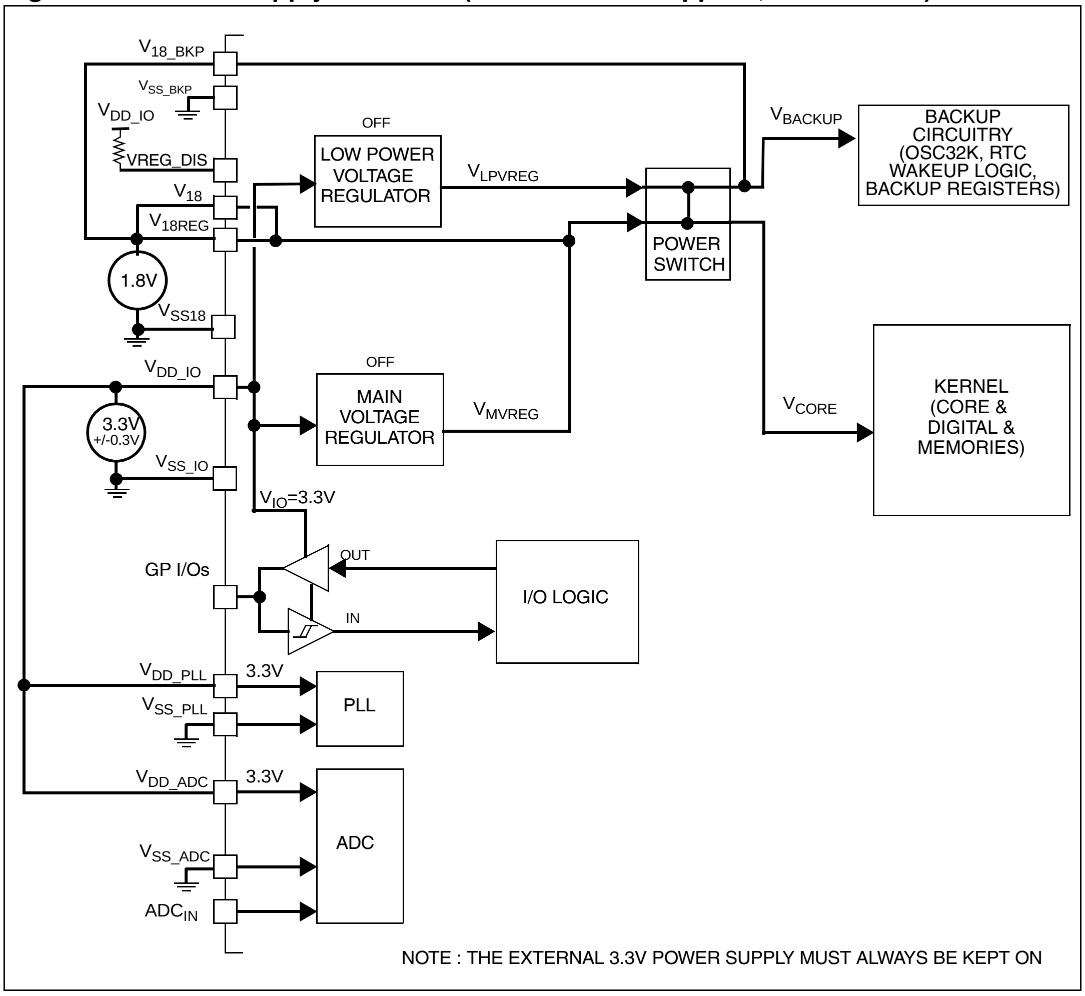
In this configuration, the internal voltage regulators are switched off by forcing the VREG_DIS pin to high level. This scheme has the advantage of saving power consumption when the 1.8V power supply is already available in the application. V_{18} and V_{18_BKP} are provided externally through the V_{18REG}, V_{18} and V_{18_BKP} power pins.

VREG_DIS pin is tied to high level which disables the Main Voltage Regulator and the Low Power Voltage Regulator.

All digital power pins (V_{18REG}, V_{18} and V_{18_BKP}) **must be externally shorted** to the same 1.8V power supply source. Internally, V_{CORE} and V_{BACKUP} are shorted by the Power Switch.

In this scheme, STANDBY Mode is not available.

Figure 6. Power supply scheme 2 (3.3V and 1.8V supplies, VREGDIS=1)



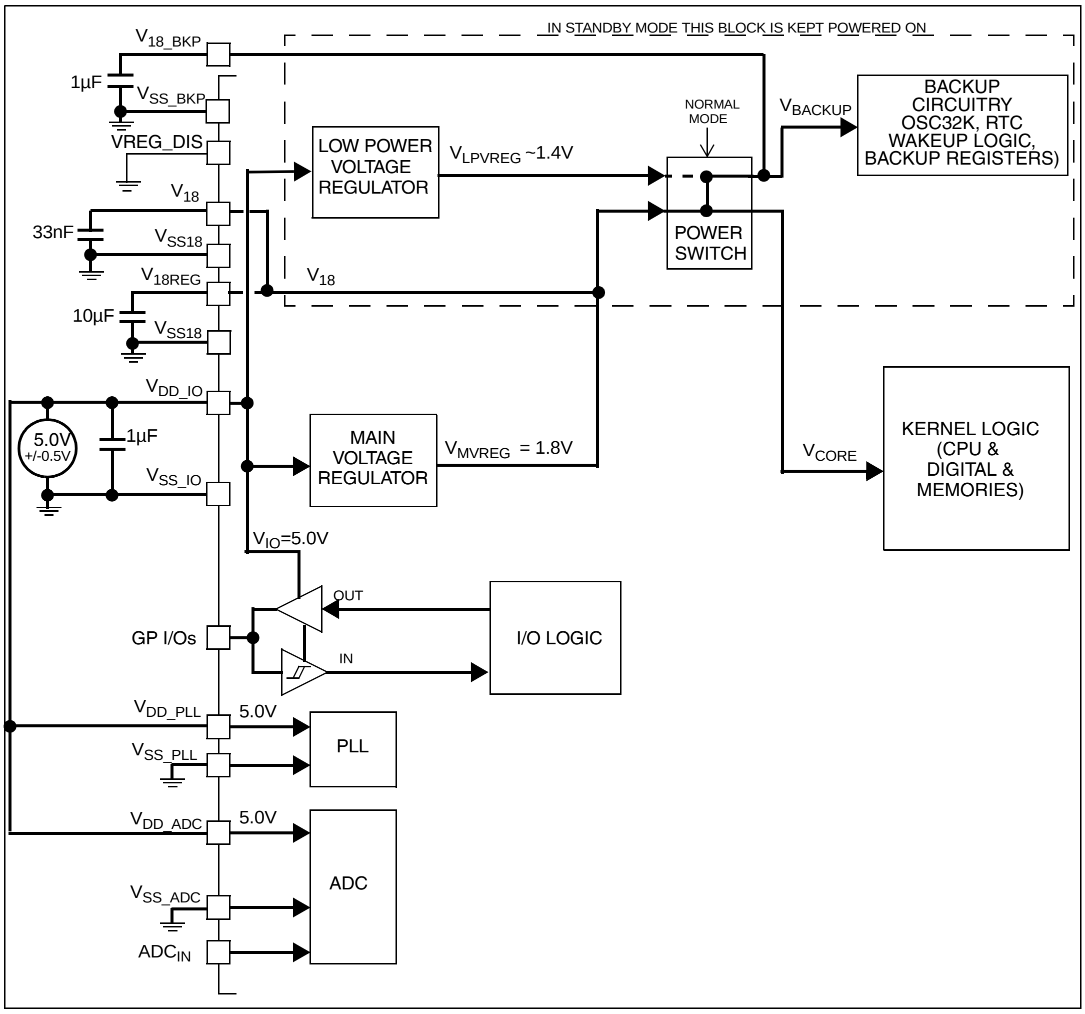
2.3.3 Power scheme 3: single external 5.0V power source

This power scheme is equivalent to Power Scheme 1 with the exception that:

- The external power supply is 5.0V +/-0.5V instead of 3.3V
- USB functionality is not available

STANDBY mode is supported in this scheme.

Figure 7. Power supply scheme 3 (single 5.0V supply VREGDIS=0) in NORMAL mode



2.3.4 Power scheme 4: dual external 5.0V and 1.8V power sources

In this configuration, the internal voltage regulators are switched off, by forcing the VREG_DIS pin to high level. This scheme has the advantage of saving power consumption when the 1.8V power supply is already available in the application and providing 5V I/O capability. V_{18} and V_{18_BKP} are provided externally through the V_{18REG}, V_{18} and V_{18_BKP} power pins.

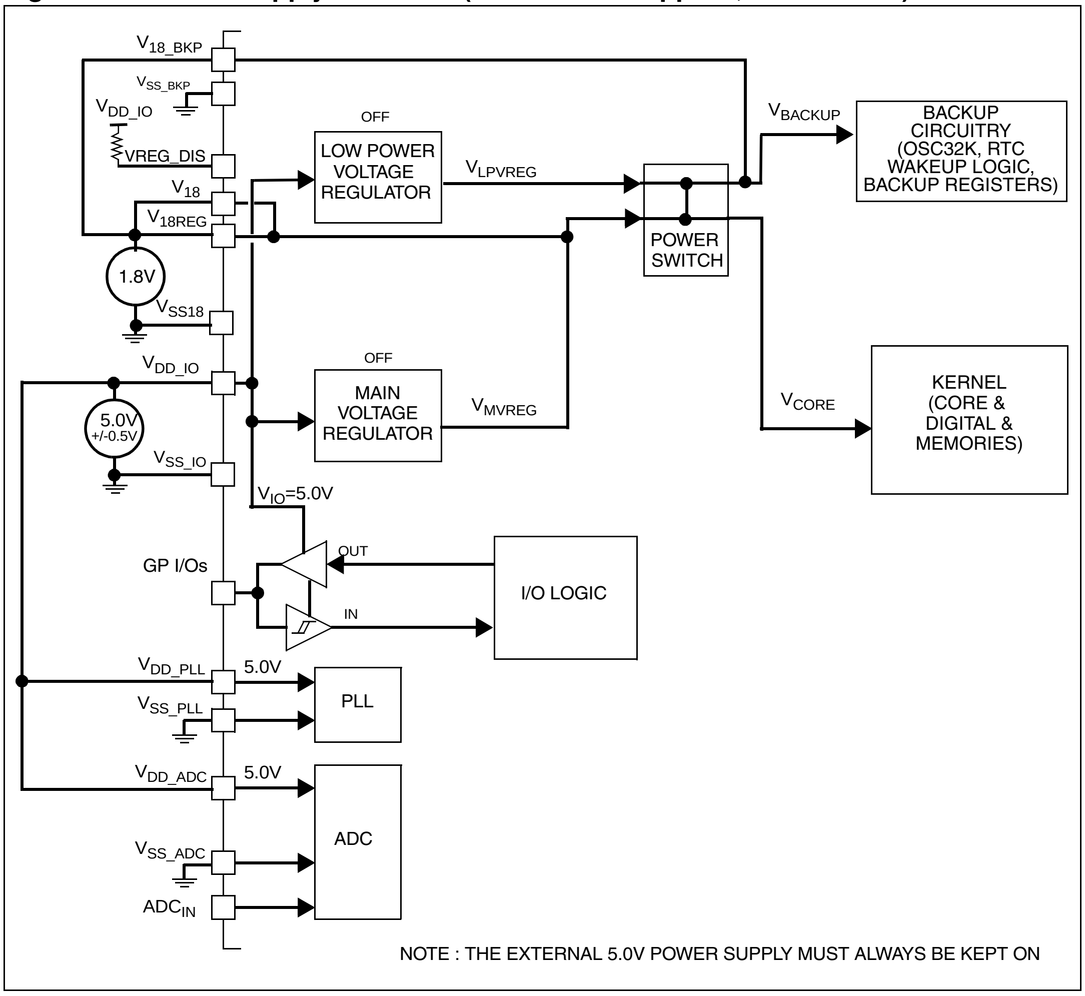
VREG_DIS pin is tied to high level which disables the Main Voltage Regulator and the Low Power Voltage Regulator.

All digital power pins (V_{18REG}, V_{18} and V_{18_BKP}) **must be externally shorted** to the same 1.8V power supply source. Internally, V_{CORE} and V_{BACKUP} are also shorted by the power switch shown in *Figure 8*.

In this scheme:

- STANDBY Mode is not available
- USB functionality is not available

Figure 8. Power supply scheme 4 (5V and 1.8V supplies, VREGDIS=1)



2.4 Main voltage regulator

In Power Scheme 1 or 3 (see *Section 2.3*) the Main Voltage Regulator provides the 1.8V power supply starting from V_{DD_IO}. The V_{18REG} pin must be connected to external stabilization capacitors (min. 10 uF Tantalum, low series resistance, and 33nF ceramic). The V_{18} pin must be left unconnected. A decoupling capacitor of 1µF must be added on the V_{DD_IO} pin which is closest to the V_{18REG} pin. Refer to the diagram in the datasheet for the pin description.

2.5 Low power voltage regulator

In Power Scheme 1 or 3 (see *Section 2.3*) the Low Power Voltage Regulator is used in STANDBY Mode and in STOP Mode (when MVREG is OFF in STOP mode).

The V_{18_BKP} pin must be connected to an external stabilization capacitor of 1µF. It generates a non-stabilized and non-thermally-compensated voltage of approximately 1.4V.

The output current is enough to power the backup circuitry (RTC and Wakeup Logic) or the V_{CORE} supply required for the kernel logic in STOP mode (with the low power control parameters FLASH and OSC4M OFF, see *Section 2.9.5: STOP mode on page 61*).

- In STANDBY mode, it provides the power supply starting from V_{DD_IO} to the backup circuitry while the Kernel Logic is un-powered.
- In STOP mode, if you program the LP_PARAM13 control bit (MVREG OFF) to disable the Main Voltage Regulator, the Low Power Voltage Regulator powers the whole digital circuitry, saving the static consumption of the Main Voltage Regulator (~100µA typical).

2.6 Regulator Startup Monitor (RSM)

Regulator Startup Monitor

The Main Voltage Regulator and the Low Power Voltage Regulator have an internal Regulator Startup Monitor (RSM) which monitors the regulated power supply.

At power-up, the RSM extends the assertion of the RESET until the regulators are operating (until V_{18_BKP} and V_{18} are at the right level).

If there is a drop in the regulated power supply, the RSM automatically generates a RESET. This enhances the security of the system by preventing the MCU from going into an unpredictable state.

Note: An external reset circuit must be used to provide the RESET at V_{DD_IO} power-up. It is not sufficient to rely on the RESET generated by the RSM in this case. This is because RSM operation is guaranteed only when V_{DD_IO} is within the specification (minimum 3.0V).

When regulators are not used (VREG_DIS pin is tied to high level), the RSM is disabled.

2.7 Reset & power startup

2.7.1 Power Startup specifications

To ensure the MCU starts-up cleanly, the rise time of the V_{DD_IO} power supply must be comprised between 20µs/V and 20ms/V.

In addition, you must provide an external RESET for at least 20 µs after the V_{DD_IO} power supply has reached its minimum working value (3.0V or 4.5V depending on power scheme).

It is recommended to use an external Power-On-Reset circuit monitoring V_{DD_IO} to assert the RESET at power-up because it is not possible to rely on the RSM to generate a RESET at power-up.

During V_{DD_IO} power-up (from 0V to 3.3V or 5.0V), all I/Os are guaranteed to be in HiZ state, assuming external RESET is asserted.

If you are using an external 1.8V power supply, the rise time of power supply V_{18} must be comprised between 20µs/V and 20ms/V.

2.7.2 External Reset Input

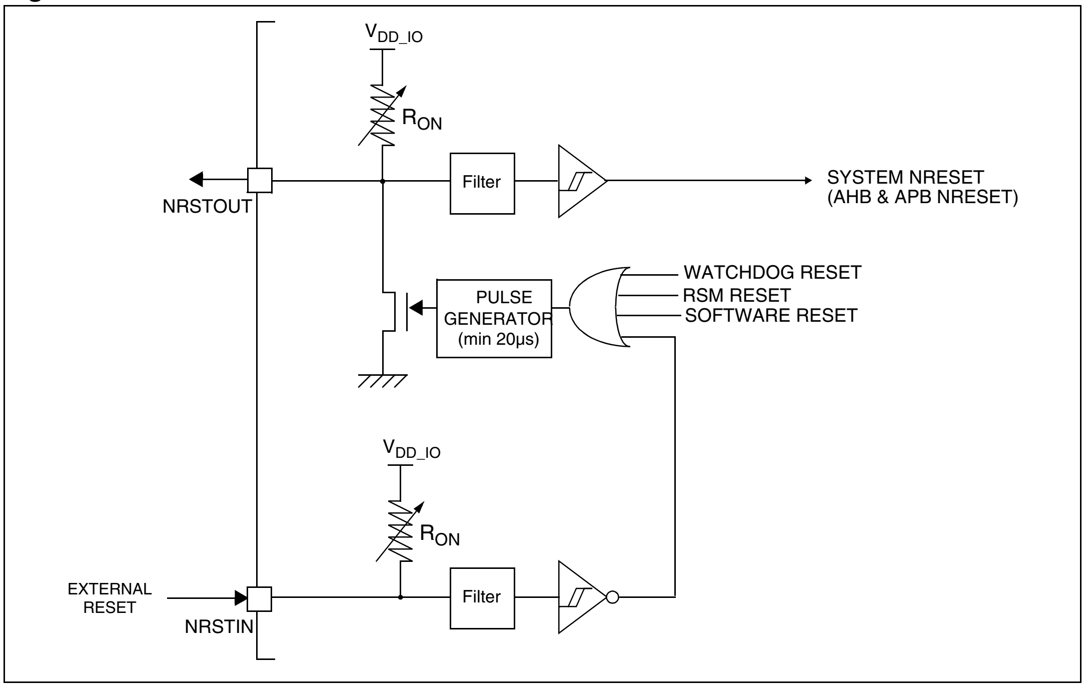
The NRSTIN pin acts as an asynchronous RESET active low.

The NRSTIN pad input is a Schmitt Trigger input pin. A filter is added to ignore all incoming pulses with short duration:

- All negative spikes with a duration less than 150 ns are filtered.
- All trains of negative spikes with a ratio of 1/2 are filtered. This means that all spikes with a maximum duration of 150 ns with minimum interval between spikes of 75 ns are filtered.

2.7.3 RESET sources

Figure 9. Reset circuit



There are 4 RESET sources:

- External RESET through NRSTIN pin
- Internal Watchdog reset
- Software reset
- RSM (Regulator Startup monitor) reset (when regulators are enabled)

All the internal RESET sources (Watchdog, Software Reset, RSM Reset) are logically ORed with the external RESET source. The resulting signal (System Reset) is processed by an analog stretch circuitry which guarantees a minimum reset pulse duration of 20µs for each reset source.

The NRSTOUT pin is an exact image of the system Reset signal (active low) provided to the device which is used to generate the reset of the AHB System and the reset of each APB peripherals.

Some peripheral registers, like the RTC or the BACKUP registers are not reset by a System Reset but only by an RSM Reset: this means that these registers are only reset at 3.3V power up.

Note: When the embedded regulators are disabled, (pin VREG_DIS tied to 1), the RTC is reset by the NRSTIN pin and the BACKUP registers are never reset.

RESET flags are available in the *Reset Flag and Status register (MRCC_RFSR)* to determine the internal reset source. This feature is available only when the embedded regulators are enabled.

Note: In STANDBY mode, the NRSTOUT pin is at high level (no RESET). However, when waking-up from STANDBY (on RTC ALARM or WKP_STDBY pin event), the NRSTOUT pin will be asserted low for 20µs (due to the analog stretcher) providing a RESET to the MCU. This way, the MCU restarts in the same way as after an external RESET; the BACKUP registers and the RTC contents are kept.

2.7.4 Software Reset

To force a software reset, first select software reset using the LPMC[1:0] bits in the MRCC_PWRCTRL register and then execute the Low Power Bit Writing Sequence described in *Section 2.9.1 on page 57*.

2.7.5 Resetting peripherals individually

You can reset the APB peripherals individually using the MRCC_PSWRES control register (see *Section 2.10.5 on page 80*).

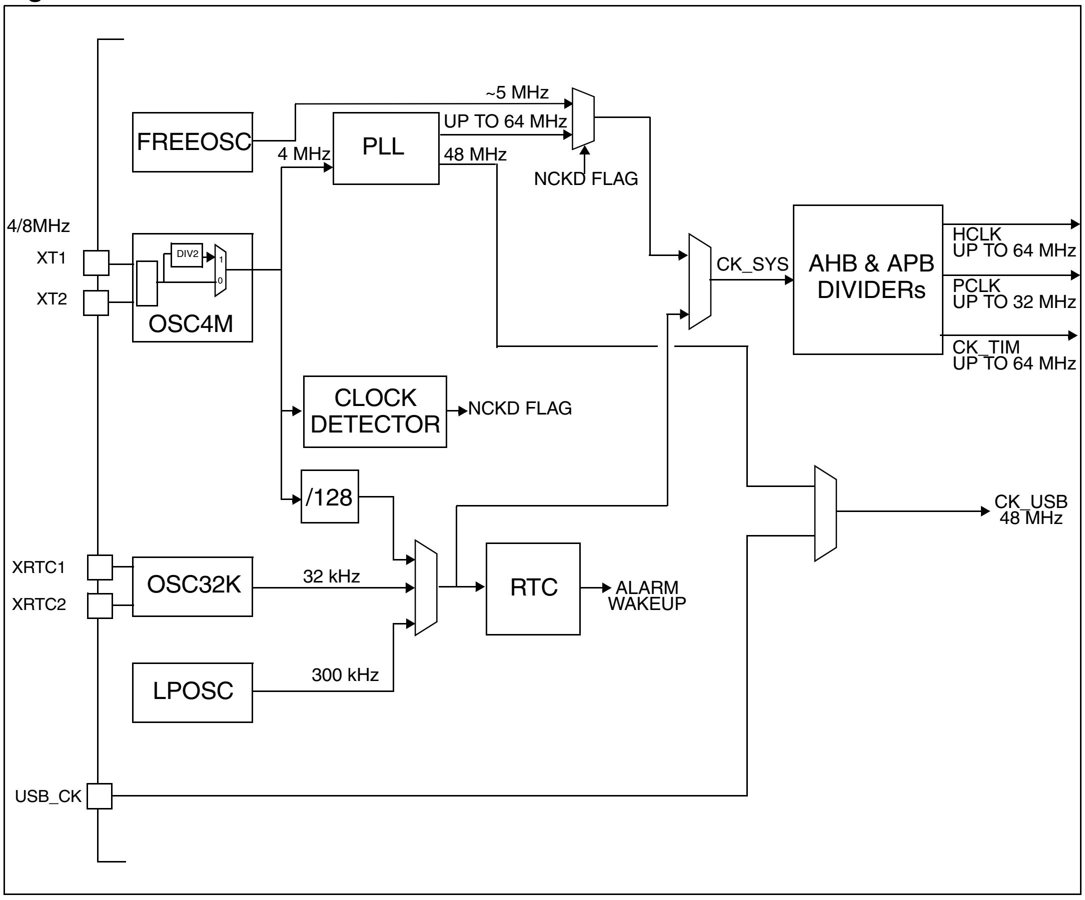
2.8 Clocks

The Clock controller provides the internal clocks needed by the different parts of the MCU:

- HCLK clocks all the peripherals mapped on the AHB bus (refer *Figure 1: Multi-Layer AHB bus architecture on page 16*.)
- PCLK clocks all the peripherals mapped on the APB bus.
- CK_TIM clocks several timer counters independently from the APB clock.
- CK_USB clocks the USB interface kernel.

2.8.1 Clock overview

Figure 10. Clock overview



Several on-chip oscillators can feed the MCU system clock (CK_SYS) from which the HCLK and PCLK derive:

- FREEOSC: Internal Free Running Oscillator providing a clock of approximately 5 MHz, also used as emergency clock. It consists of the internal VCO of the PLL configured in free running mode
- OSC4M: 4 MHz Main Oscillator, based on:
  - a 4 MHz Crystal/Ceramic oscillator connected to XT1/XT2
  - or an 8 MHz Crystal/Ceramic oscillator to XT1/XT2 followed by an embedded divider by 2
  - or external clock connected to XT1

  which can be multiplied by the PLL to provide a wide range of frequencies (up to 64 MHz)
- OSC32K: 32.768kHz Oscillator (Crystal or Ceramic oscillator) which can drive either the system clock and/or the RTC.
- LPOSC: Internal Low Power RC Oscillator providing a clock around 300 kHz which can drive either the system clock and/or the RTC.

Several configurable dividers provide a high degree of flexibility to the application in the choice of the APB or AHB frequency, while keeping a fixed frequency value for the USB clock (48 MHz).

The Clock Detector (CKD) protects the Microcontroller against OSC4M or external clock failures.

The RTC provides calendar, alarm and wake-up functions and can be clocked by any of the oscillators other than FREEOSC.

2.8.2 Main 4MHz oscillator (OSC4M)

XT1 and XT2 pins are used to connect the Main Oscillator source, which can be a resonator (crystal or ceramic) or an external source. Both sources can be used as the input clock to PLL frequency multiplier (PLL) which requires a 4 MHz input clock.

Crystal or ceramic resonator

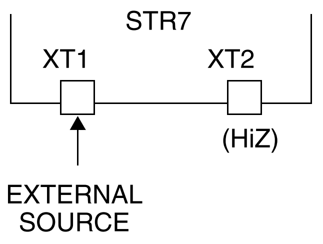
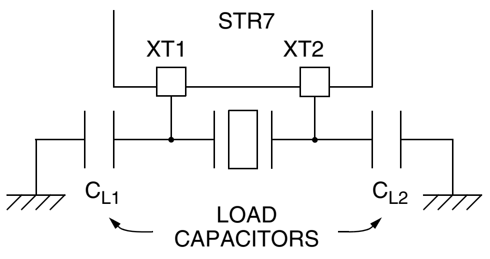
This Oscillator (OSC4M) has the advantage of producing a very accurate rate on the main clock. This oscillator can be directly connected to

- a 4 MHz Oscillator
- or a 8 MHz Oscillator followed by a divider by 2.

If an 8 MHz Crystal or Ceramic is connected, You must select the divider by 2 by setting the XTDIV2 control bit in the *Clock Control Register (MRCC_CLKCTL)* . You can do this, for example, during the initialization phase while the system is clocked by FREEOSC.

The associated hardware configuration is shown in *Figure 11*. Refer to the electrical characteristics section of the datasheet for more details.

Figure 11. Clock sources

	Hardware Configuration
External Clock	
Crystal/Ceramic Resonators	 STR7 XT1 XT2 C_{L1} C_{L2} LOAD CAPACITORS

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

External Source (Bypass Mode)

In this mode, an external clock source must be provided. You select Bypass mode by setting the OSC4MBYP control bit in the *Clock Control Register (MRCC_CLKCTL)* . You can do this, for example, during the initialization phase while the system is clocked by FREEOSC. The external clock signal (square, sinus or triangle) with ~50% duty cycle has to drive the XT1 pin while the XT2 pin should be left hi-Z. See *Figure 11*.

- **If the PLL is used in the application**, the external clock source must be 4 MHz (or 8 MHz divided by 2 if the XTDIV2 bit is set in the *Clock Control Register (MRCC_CLKCTL)*).
- **If the PLL is not used**, the external clock source can have any frequency within the range specified in the datasheet. The divider by two is not mandatory in this case.

Oscillator power down

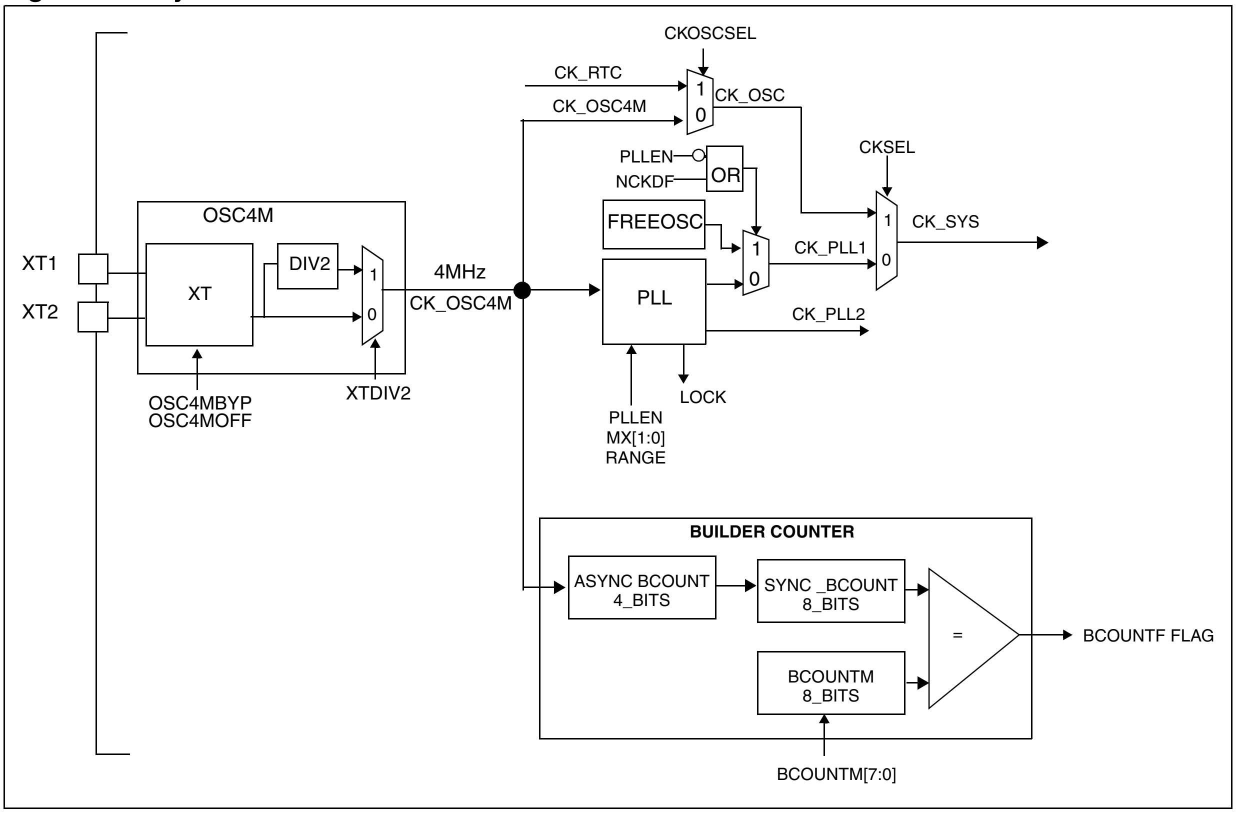
At any time, it is possible to temporarily clock the MCU using another clock source (CK_RTC) and to switch off the OSC4M (setting the OSC4MOFF control bit in the MRCC_CLKCTL register) to reduce power consumption.

2.8.3 Managing the Oscillator start-up time (builder counter)

This clock scheme is designed to allow the microcontroller to become operational almost immediately after power-on or reset, and the application can execute code without waiting for the main oscillator startup: the CPU is initially clocked by the FREEOSC after RESET release or after exiting from Low Power Mode. This allows the STR750 to respond to external events or perform any other type of operation like initializing the hardware registers.

In parallel, the main oscillator (OSC4M) is automatically enabled and software can monitor the startup time of the main oscillator using the builder counter (BCOUNT) and No Clock Detected (NCKDF) flag. This allows the microcontroller to adapt to the start-up time of any oscillator type.

Figure 12. System Clock scheme



The Builder Counter is a 12-Bit counter clocked by CK_OSC4M. It is based on 4-bit asynchronous counter (in order to filter glitches) and an 8-bit synchronous counter. The current value of this 8-bit counter is compared with the BCOUNTM maximum value, previously programmed by software: when they are equal, the BCOUNTF flag is set and the counter is stopped.

The builder counter is reset and enabled each time OSC4M has been disabled, and flagged by the NCKDF bit. To start the builder counter the NCKDF flag must be reset by software. This occurs:

- After RESET
- After STANDBY
- When software re-enables the OSC4M oscillator (using OSC4MOFF bit of the *Clock Control Register (MRCC_CLKCTL) on page 70*).
- When waking-up from STOP mode with LP_PARAM15 bit set (OSC4M OFF) (See *Section 2.9.5: STOP mode on page 61*)
- When the OSC4M clock recovers after an oscillator failure detection (NCKDF bit)

If you write to BCOUNTM[7:0] with a value lower than the current builder counter value, the BCOUNTF flag is immediately set and the counter is stopped.

Note: The default value of BCOUNTM[7:0] is FFh (4096 oscillator cycles of OSC4M)

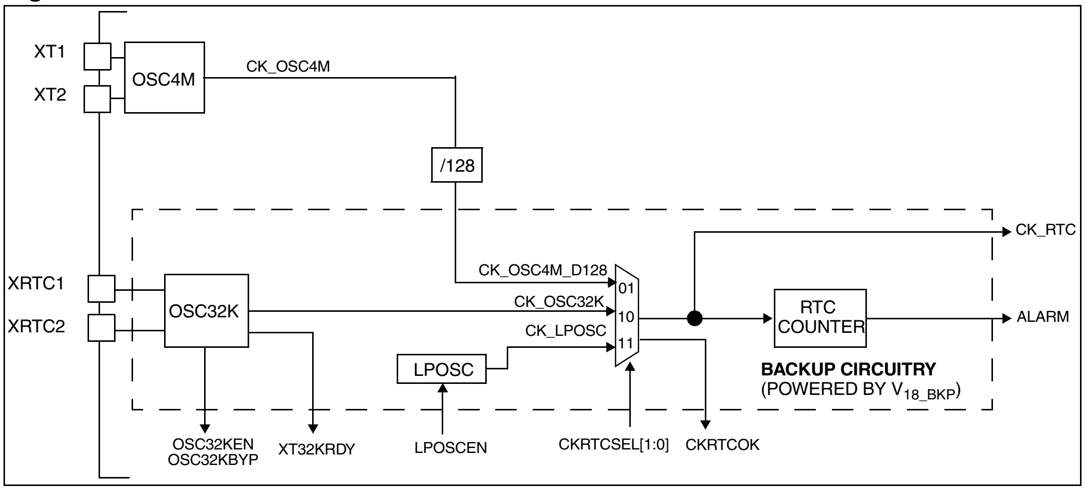
As soon as the main oscillator is running, you can switch the system clock (CK_SYS) to the clock source configured in the *Clock Control Register (MRCC_CLKCTL) on page 70*.

In initialisation part of your code, you can configure the oscillator selection, PLL configuration and dividers for AHB and APB clocks.

The clock multiplexers are designed so that all the above clock switching mechanisms are guaranteed to be glitch-free. In addition, built-in protection hardware prevents you from switching to an inactive clock. This considerably simplifies the driver code.

2.8.4 RTC Clock Source (LPOSC, OSC32K)

Figure 13. RTC Clock scheme



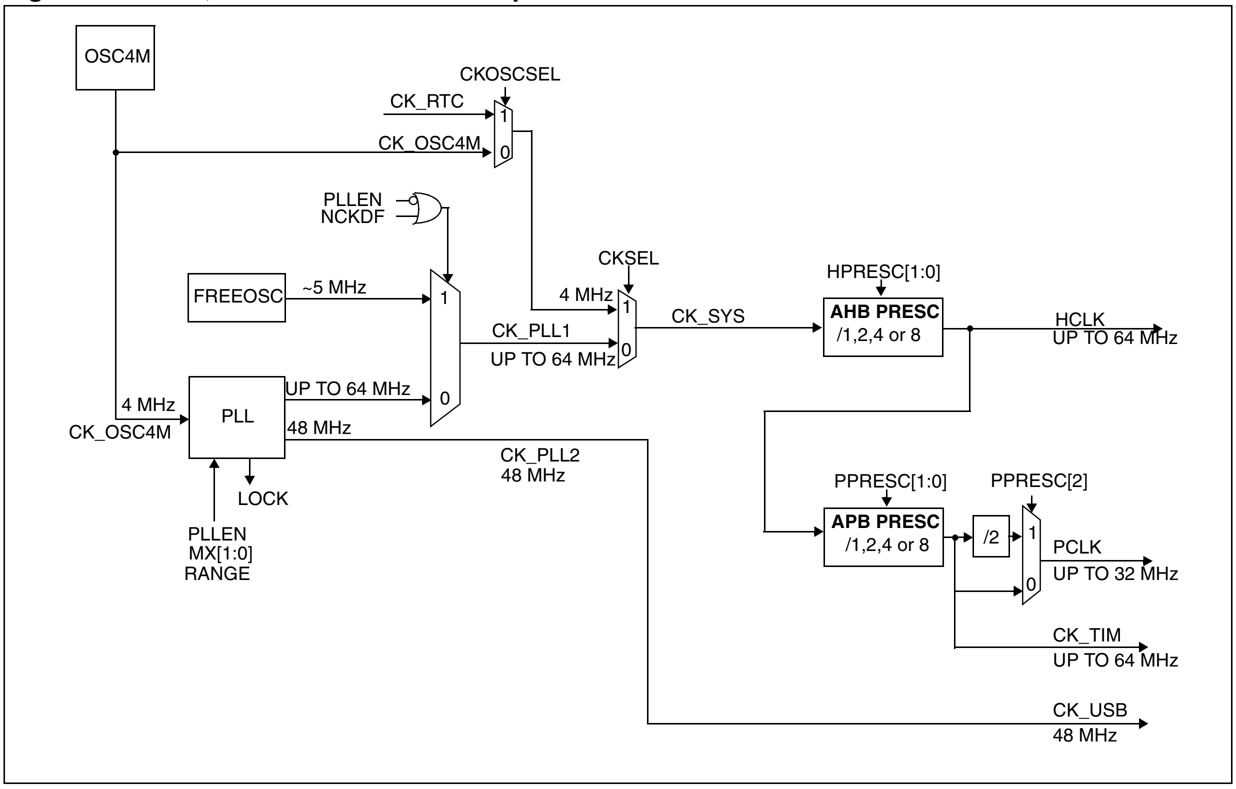
There are 3 clock sources available for the RTC:

- Low Power 32.768 kHz Crystal Oscillator (OSC32K)
- Low Power RC Oscillator (LPOSC)
- Main Oscillator (OSC4M) divided by a fixed prescaler of 128

The RTC clock can be used as a low power system clock, selected by the CKRTCSEL in the MRCC_PWRCTRL register (*Section 2.10.3 on page 76*) and the CKSEL and CKOSCSEL bits in the MCRCC_CLKCTL register (*Section 2.10.1 on page 70*). If LPOSC or OSC32K are selected by CKRTCSEL, the Main Oscillator (OSC4M) can be stopped in this mode.

2.8.5 PLL, FREEOSC, & AHB/APB prescalers

Figure 14. PLL, FREEOSC & AHB/APB prescaler scheme



The PLL provides a Frequency Multiplier starting from a single input clock (OSC4M source) and providing the 2 following independent output clocks:

– CK_PLL1 output with 4 programmable multiplication factors (up to 64 MHz) used for generating CK_SYS
– CK_PLL2 output with fixed 48 MHz frequency when input clock is 4 MHz used for generating CK_USB

FREEOSC provides a Free Running Oscillator for the system clock: this clock is selected when

– The PLL is disabled: FREEOSC is acting as an oscillator source
– The clock failure flag is active: FREEOSC is acting as an emergency clock source.

For security, the hardware prevents the software from making certain unrecoverable errors:

– Software cannot switch CK_SYS to CK_PLL1 (multiplied clock output) until the PLL is locked.
– CK_USB is gated until the software enables the PLL2EN bit in the MRCC_CLKCTL register (*Section 2.10.1 on page 70*).

The AHB and APB prescalers allow you to choose the AHB and APB frequencies from a wide range of possibilities (see *Section 2.10.1: Clock Control Register (MRCC_CLKCTL) on page 70*):

– HCLK (AHB Clock) can be generated from CK_SYS divided by 1, 2, 4 or 8
– PCLK (APB Clock) can be generated from HCLK divided by 1, 2, 4, 8 or 16

The clock provided to the Timers (CK_TIM) can have twice or the same frequency as PCLK . This allows the timers to count at high frequency (up to 64 MHz)

Table 7 gives some typical clock configurations:

Table 7. Typical prescaler uses

f_{OSC4M}	PLL Factor	f_{CK_SYS}	HPRESC [1:0]	f_{HCLK}	PPRESC [1:0]	f_{CK_TIM}	PPRESC 2	f_{PCLK}	f_{CK_USB}
4 MHz	x16	64 MHz	00	64 MHz	00	64 MHz	1	32 MHz	48 MHz
	x16	64 MHz	00	64 MHz	01	32 MHz		16 MHz	48 MHz
	x15	60 MHz	00	60 MHz	00	60 MHz		30 MHz	48 MHz
	x14	56 MHz	00	56 MHz	00	56 MHz		28 MHz	48 MHz
	x12	48 MHz	00	48 MHz	00	48 MHz		24 MHz	48 MHz
	x16	64 MHz	01	32 MHz	00	32 MHz		16 MHz	48 MHz

The software must respect the configuration constraints of the PLL (refer to *Section 2.8.9: System startup on page 48*).

The FREEOSC and PLL are reset during the whole assertion of System RESET. After reset release, the PLL is disabled and FREEOSC supplies the system clock (~5 MHz).

2.8.6 Clock-out capability: MCO (Main Clock Output)

The Main Clock Output (MCO) capability allows you to output a clock on the external MCO pin. The configuration registers of corresponding GPIO port must programmed in alternate function mode. You can select one of 4 clock signals as MCO clock.

- CK_PLL2
- HCLK
- PCLK
- CK_OSC4M

The selection is controlled by the MCOS[1:0] bits of CLKCTL register. A dedicated prescaler (divide by 1 or 2, selected by MCOP bit) can be applied to this clock before outputting it to the MCO pin. Care must be taken when switching MCO clock selection, the alternate function should be disabled to avoid any glitches on the MCO pin.

2.8.7 Clock Detector (CKD)

A CKD (Clock Detector) is implemented to:

- detect if no clock is present on OSC4M (broken or disconnected resonator) and prevent the software from selecting it.
- automatically feeds the MCU with the FREEOSC used as emergency clock if no clock is detected.
- generate an interrupt if enabled, allowing the MCU to perform some rescue operations

Emergency Oscillator Control

The CKD is always enabled. This way, the MCU can start even if there is no oscillator clock source.

In some Low Power Modes, the CKD is active and can wake up the MCU. See *Section 2.9: Low Power modes on page 57*

Emergency Oscillator Activation

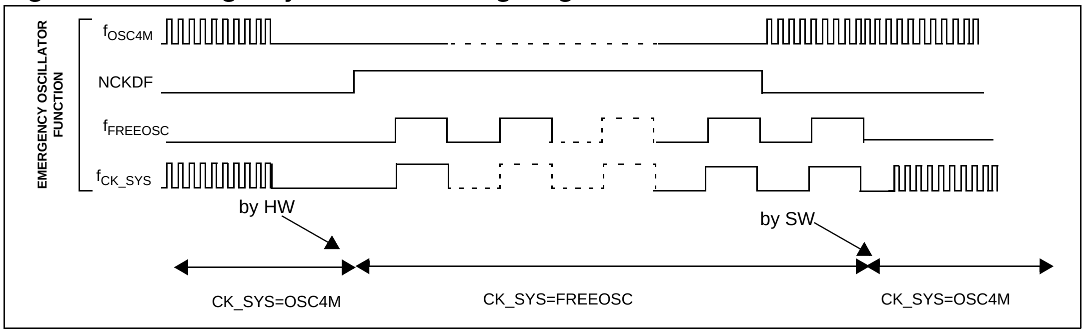
The activation of the emergency oscillator is notified by hardware by setting the NCKDF (No Clock Detected Flag) bit in the in the MRCC_CLKCTL register. An interrupt can be generated if the NCKDIE bit has been previously set.

These two bits are described in *Section 2.10.1 on page 70*.

Emergency oscillator clock recovery

When the clock source switches to the emergency clock, the user program code is still able to run. As soon as the Main Oscillator clock recovers, the software is notified by the NCKDF flag coming back to ‘0’. Then software can switch back to the original clock configuration.

Figure 15. Emergency oscillator timing diagram

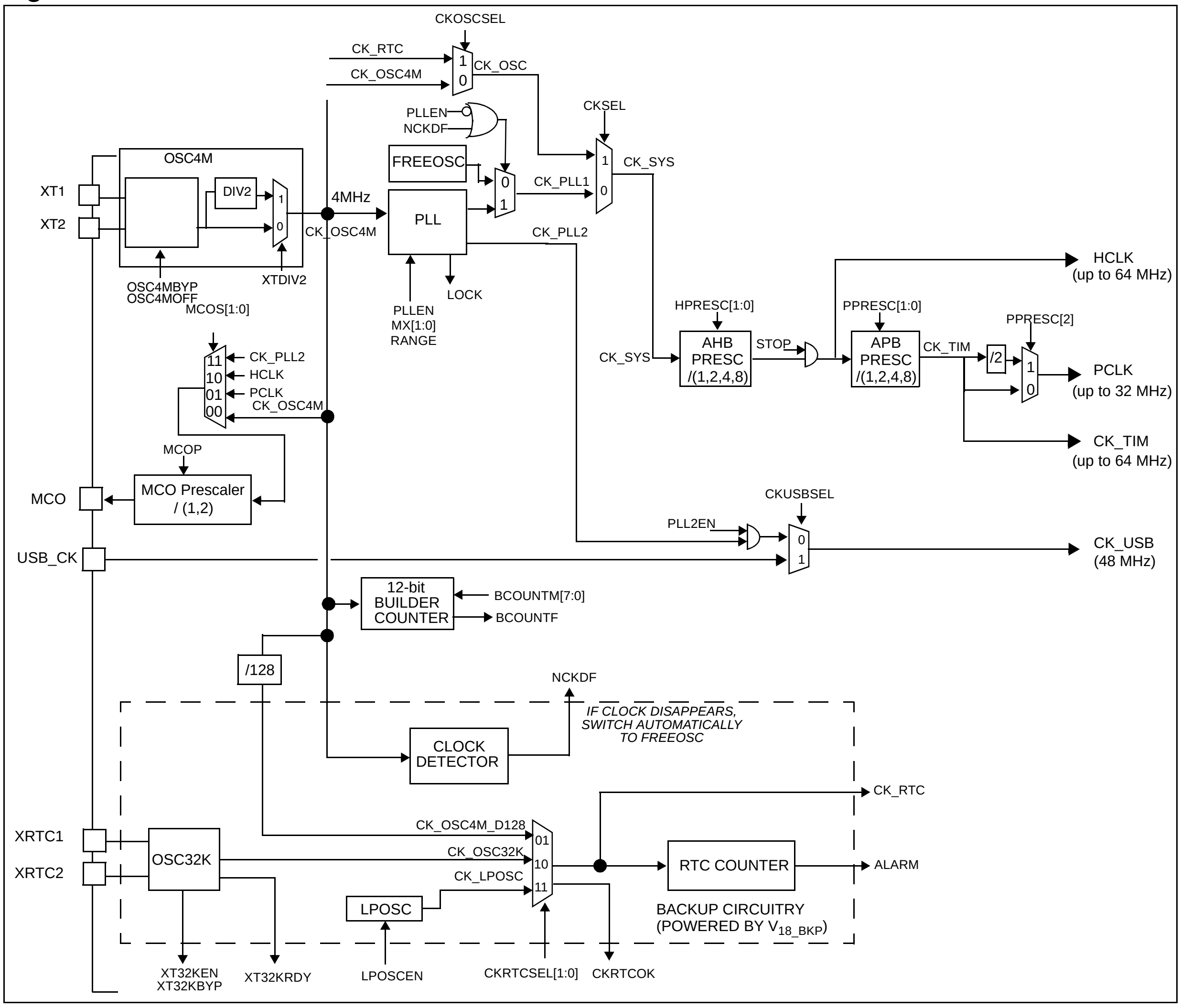


Note: The CKD is also enabled when Oscillator is bypassed by external clock. This means that it is not possible to clock the MCU with a discontinued clock (like clock pulses followed by blanks) because the CKD will provide internal clock cycles.

2.8.8 Detailed Clock scheme

Figure 16 shows an overview of the main clocks. Refer to Figure 24 for details on the peripheral clocks.

Figure 16. Detailed Clock scheme



2.8.9 System startup

This following sections describe the state of the processor just after reset and how to switch from one source clock to another one. *Figure 17* and *Figure 18* show a waveform diagram of a system startup.

Figure 17. Startup waveform diagram

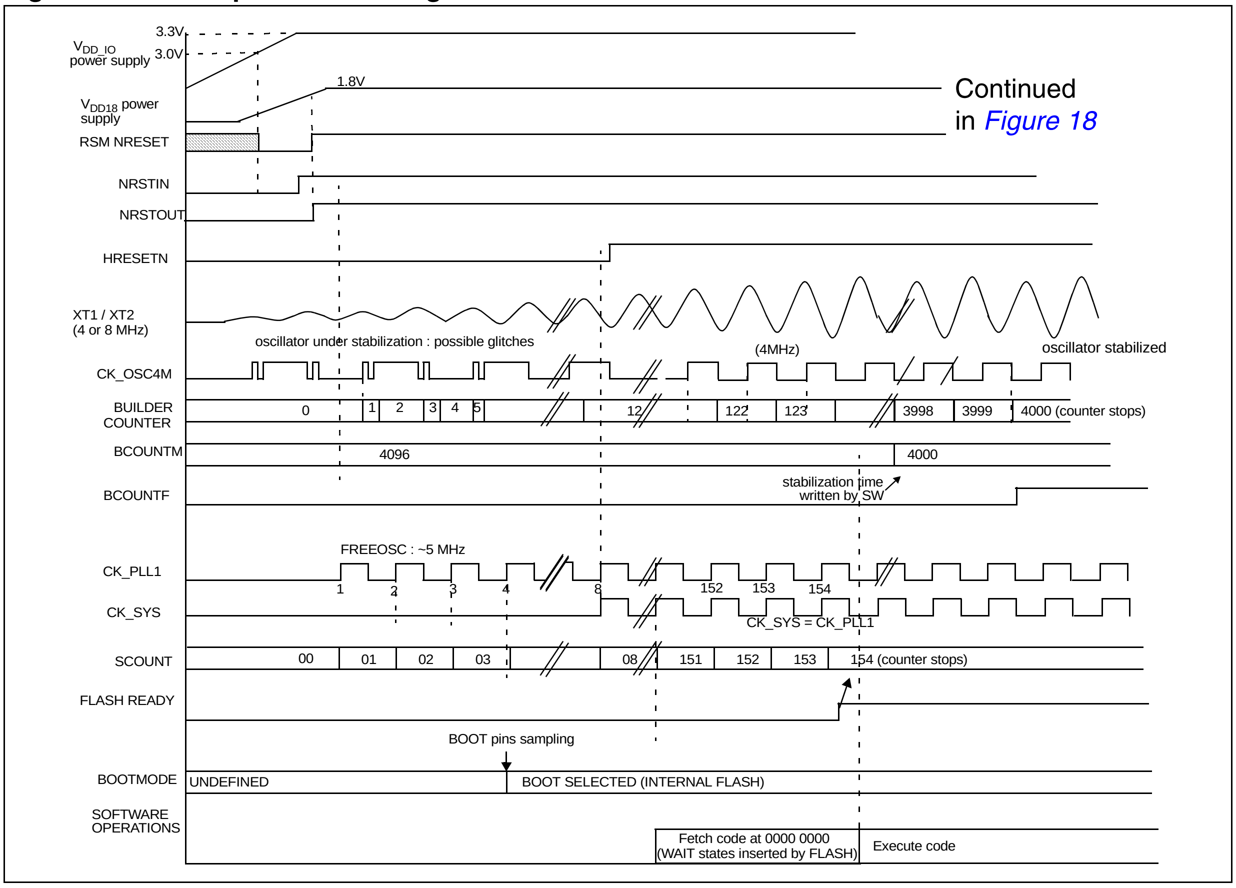
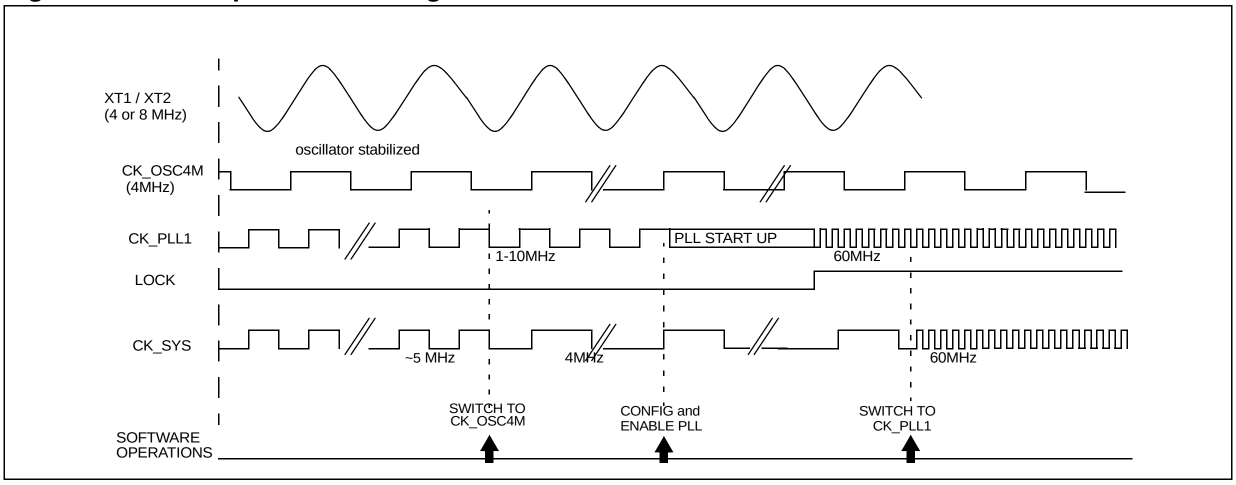


Figure 18. Startup waveform diagram continued



2.8.10 Clock state after Reset

After any reset:

- CK_SYS is driven by the FREEOSC: the reset state of CKSEL bit selects CK_PLL1 (driven by the FREEOSC) as input clock of CK_SYS.
- The AHB prescaler is in “divide by 1” mode. This means that $f_{HCLK} = f_{FREEOSC}$
- The APB prescaler is in “divide by 1” mode. This means that $f_{PCLK} = f_{FREEOSC}$

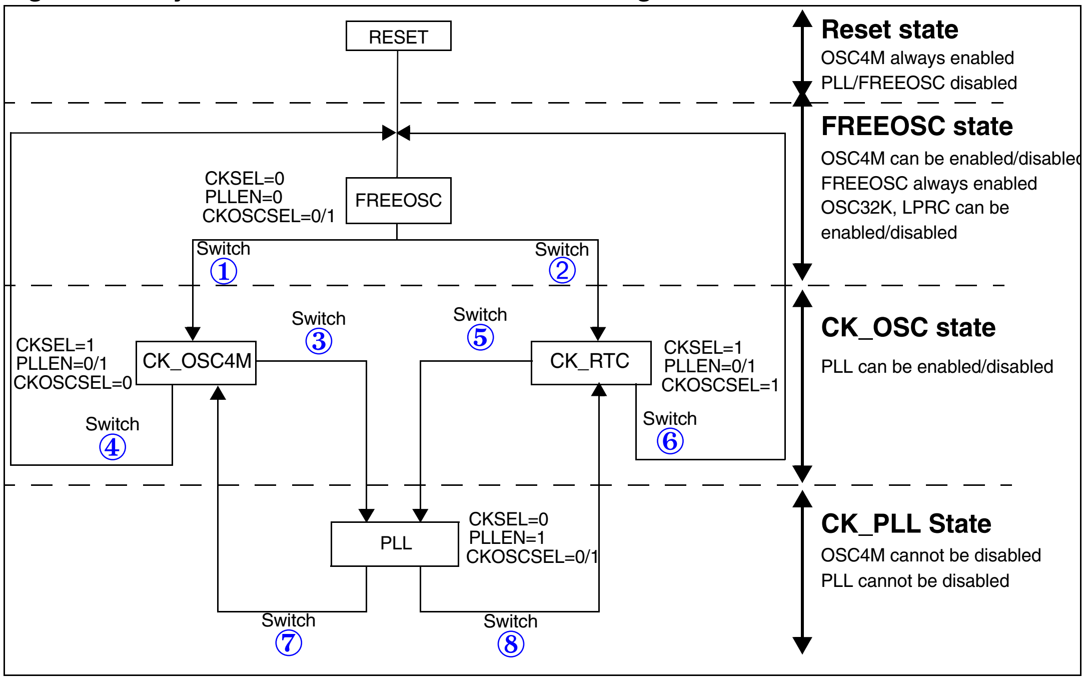
After reset release, CK_SYS is first delayed by an internal temporization.

During this temporization, on the 4th rising edge, the state of the BOOT pins are sampled which defines the boot configuration (Refer to *Figure 17: Startup waveform diagram on page 48*)

- If internal **Flash boot mode** is selected, this temporization is extended and the code execution will wait until the Flash Ready Response.
  This response is not deterministic and may significantly vary depending on the Flash embedded algorithms, process and temperature. The result of this delay count can subsequently be read in the SCOUNT[11:0] bits in the MRCC_RFSR register and the elapsed time can be estimated based on CK_SYS clock frequency.
- If a **boot mode other than internal Flash** is selected, this temporization is limited to 8 clock cycles. The CPU will start executing code immediately after this delay of 8 clock cycles. Then if software wants to access the internal Flash, it should wait for the Flash to become ready to be accessed and monitor the FLASH_BUSY bit in the CFG_GLCONF register. When FLASH_BUSY=0, this means that the Flash is ready and that it can be accessed immediately. However, any access made before the Flash is ready will be performed but will result in WAIT states being inserted.

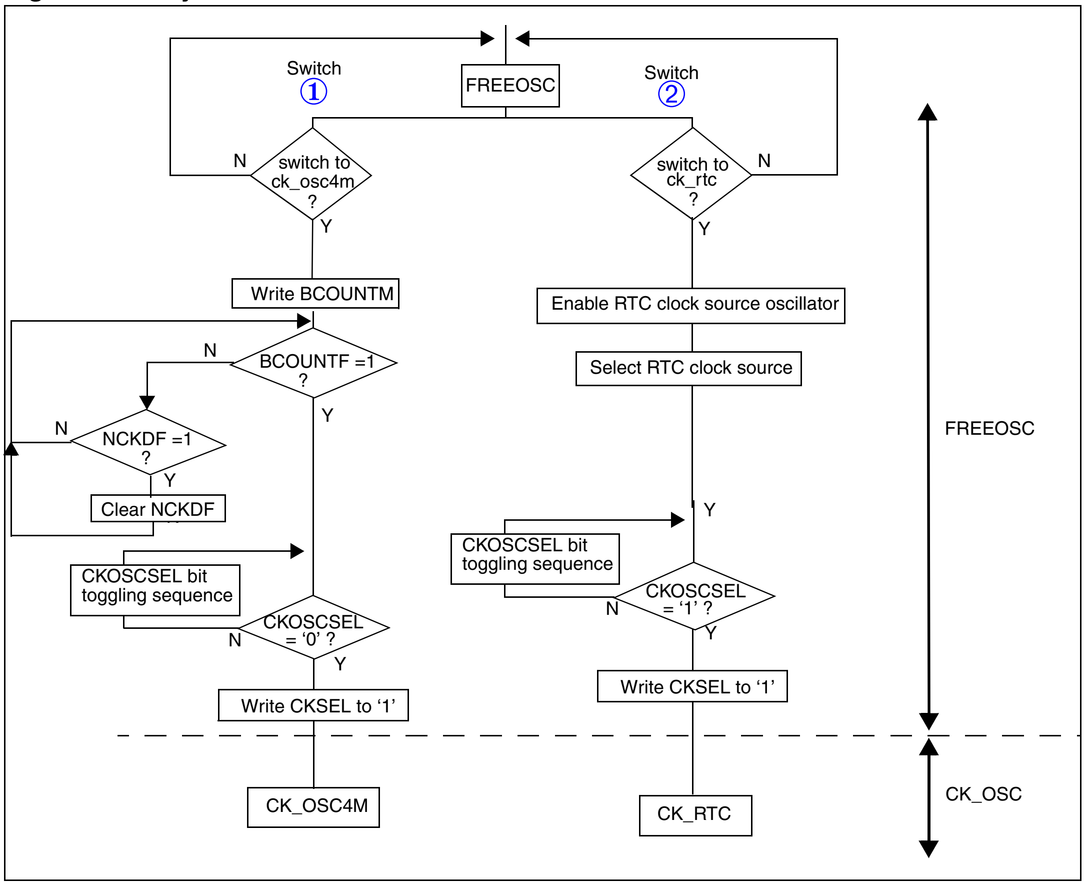
Once the boot mode is defined and temporization has elapsed, the CPU fetches the first instruction. Software must then configure the clocks following the steps outlined in *Figure 19* and detailed in *Figure 20*, *Figure 21*, *Figure 22* and *Figure 23*.

Figure 19. System Clock source control state diagram



2.8.11 Switching CK_SYS from FREEOSC to CK_OSC (CK_OSC4M or CK_RTC)

Figure 20. System Clock source control: switches 1 and 2



After reset, the first software instructions (clocked by the FREEOSC) have to select the clock source and the PLL multiplier factor. You select the clock source using the CKSEL bit in the *Clock Control Register (MRCC_CLKCTL)* register. It is mandatory to proceed in the following way:

Switch 1: switching CK_SYS from FREEOSC to CK_OSC4M

1. If an 8 MHz Ceramic/crystal is connected to the oscillator, the oscillator divider by 2 must be enabled by setting the XTDIV2 bit in the *Clock Control Register (MRCC_CLKCTL)*.
2. Check that an oscillator clock is present (CK_OSC4M) using the NCKDF bit in the MRCC_CLKCTL register. After reset, the NCKDF status flag is set (default value): software must reset the flag, read it back and repeat this until it goes to 0.
3. Define the delay for the oscillator clock by writing this value in the builder counter maximum value (BCOUNTM[7:0] bits in the *Reset Flag and Status register (MRCC_RFSR)*), which depends on the stabilization time required by the main oscillator being used. Even when using an oscillator with a quick start-up time, you have to allow a minimum delay to ensure that a clock is present, and ensure a clean, glitch-free clock.

4. Wait until the BCOUNTF flag is set, indicating that a stabilized clock input is present and verify in the meantime that NCKDF is still reset. If NCKDF has been set, which can occur in a case of oscillator startup, you should reset the flag which will restart the builder counter, and wait until the BCOUNTF flag is set.
5. Switch CK_SYS from CK_PLL1 to CK_OSC4M, using the CKSEL bit in the MRCC_CLKCTL register. To safeguard the system, the selection is possible only if an oscillator clock is present (NCKDF=0). If CKSEL is at 1, the clock system has successfully switched to CK_OSC4M. Otherwise the CKSEL bit remains at ‘0’, CK_SYS remains clocked by the FREEOSC and software must repeat step 1.

Switch 2: switching the CK_SYS from FREEOSC to CK_RTC

1. Configure the clock source that you want to use for the RTC:
   a) LPOSC: Write the LPOSCEN control bit to enable the Low Power RC Oscillator
   b) OSC32K: Write the OSC32KEN control bit to enable the 32.768 kHz Oscillator, and wait for the OSC32KRDY flag to be set to ‘1’ indicating that the oscillator clock is stable.
   c) CK_OSC4M divided by 128: The OSC4M must be active.
2. Select the clock source to be used, by setting the CKRTCSEL[1:0] bits in the MRCC_PWRCTRL register.
3. Execute the CKOSCSEL Bit Toggling Sequence to set it to ‘1’. This sequence is similar to the Low Power Bit Writing Sequence:
   a) Write CKOSCSEL to ‘1’
   b) Write CKOSCSEL to ‘1’
   c) Write CKOSCSEL to ‘0’
   d) Write CKOSCSEL to ‘1’
   e) Read CKOSCSEL until CKOSCSEL=1

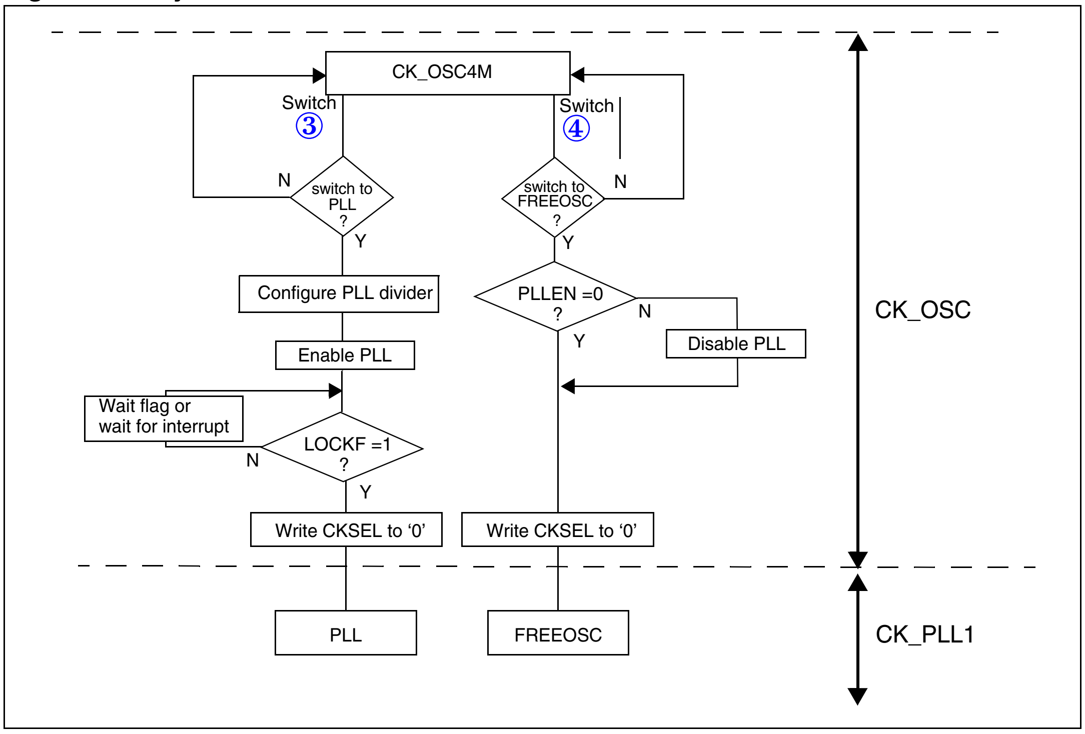
The CKOSCSEL bit is now written to ‘1’. CK_OSC is now driven by CK_RTC (see *Figure 12: System Clock scheme on page 42*).

Note: *It is recommended to mask interrupts when executing the above sequence.*

4. Switch CK_SYS from CK_PLL1 to CK_RTC, by writing CKSEL to ‘1’.
   In the case where OSC4M is selected as RTC clock source, the Clock Detector function prevents the CKSEL bit from going to ‘1’ if the CK_OSC4M clock is not present (NCKDF flag=1).
5. Read both CKOSCSEL and CKSEL. If both are at 1, the clock system has successfully switched to CK_RTC.
6. To further reduce power consumption, you can first disable the PLL by writing the PLLEN bit to ’0’ and in a second step disable the OSC4M Oscillator by writing the OSC4MOFF bit to ‘1’.

2.8.12 Switching CK_SYS from CK_OSC4M to CK_PLL1 (PLL or FREEOSC)

Figure 21. System Clock source control: switches 3 and 4



Switch 3: switching CK_SYS from CK_OSC4M to PLL

1. Configure the PLL multiplier factor respecting the PLL constraints regarding input and output frequencies to obtain the required multiplier factor. (see *Section 2.10.1: Clock Control Register (MRCC_CLKCTL) on page 70*).
2. Enable the PLL by setting the PLLEN bit (see *Section 2.10.1*).
3. Wait for the LOCK status bit by polling or waiting for the interrupt (assuming it has been previously enabled) (see *Section 2.10.1*).
4. Switch CK_SYS from CK_OSC4M to CK_PLL1, using the CKSEL control bit.

Switch 4: switching CK_SYS from CK_OSC4M to FREEOSC

1. Check that PLL is disabled, and write PLLEN bit to '0' if necessary.
2. Switch CKSYS to the FREEOSC clock (CK_PLL1), by writing CKSEL bit to 0.

2.8.13 Switching CK_SYS from CK_RTC to CK_PLL1 (PLL or FREEOSC)

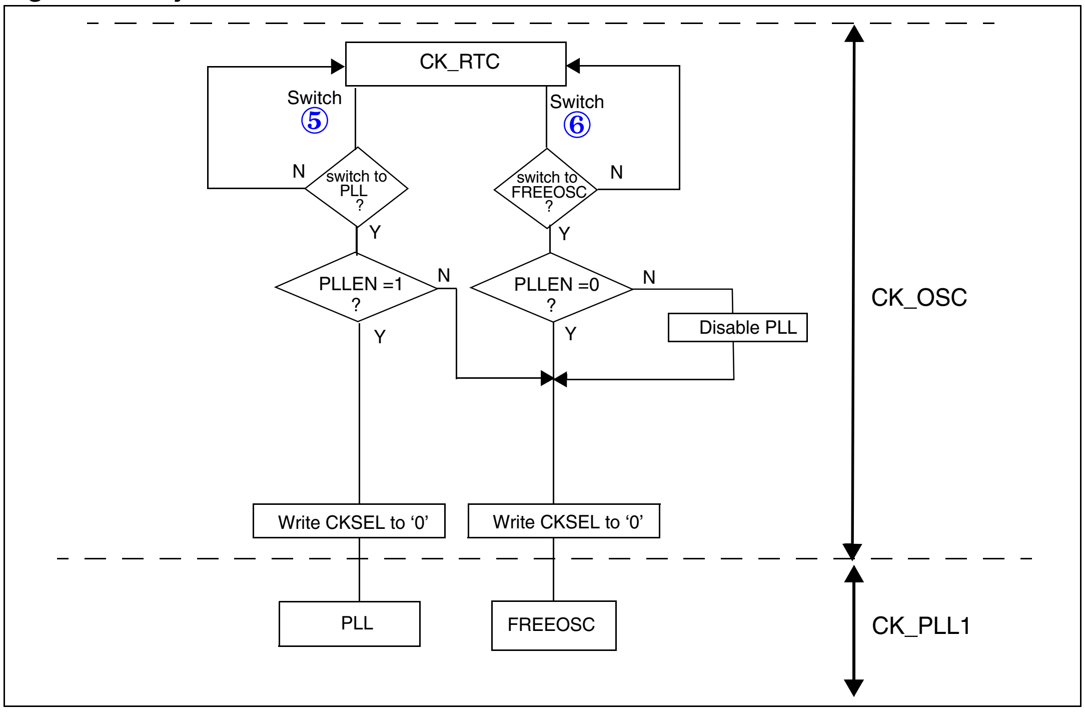


Figure 22. System Clock source control: switches 5 and 6

Switch 5: switching CK_SYS from CK_RTC to PLL

If the PLL is already enabled and locked, you can simply switch CK_SYS to CK_PLL1 by resetting CKSEL bit.

Otherwise, if the PLL is not enabled, use the following procedure:

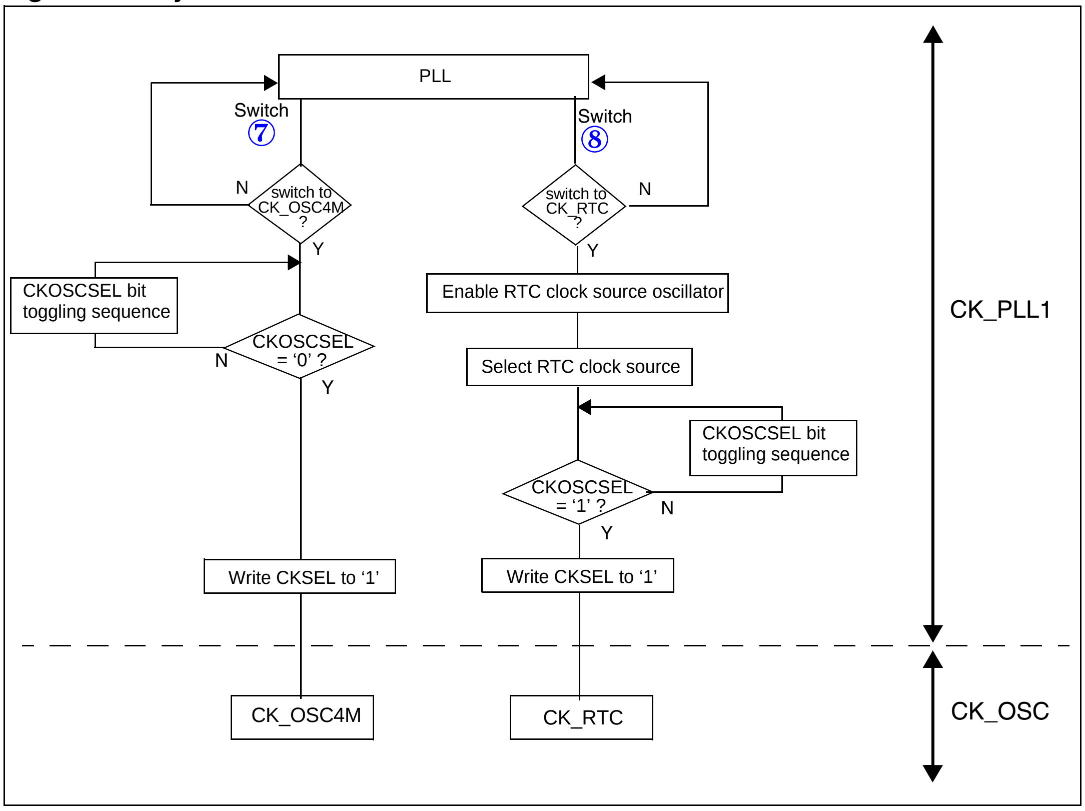
1. Switch CK_SYS to the FREEOSC clock (CK_PLL1), by writing PLLEN bit to 0 and CKSEL bit to 0.
2. Switch CK_SYS from FREEOSC to CK_OSC4M using the procedure in *Section 2.8.11 on page 50*).
3. Switch CK_SYS from CK_OSC4M to PLL using the procedure in *Section 2.8.12 on page 52*).

Switch 6: switching CK_SYS from RTC to FREEOSC

1. Check if the PLL is disabled, and write 0 to the PLLEN bit if the PLL not disabled.
2. Switch CK_SYS to the FREEOSC clock (CK_PLL1), by writing PLLEN bit to 0 and CKSEL bit to 0.

2.8.14 Switching the CK_SYS from PLL to CK_OSC (CK_OSC4M or CK_RTC)

Figure 23. System Clock source control: switches 7 and 8



Switch 7: switching CK_SYS from PLL to CK_OSC4M

1. Check that CKOSCSEL is 0, otherwise apply the CKOSCSEL bit toggling sequence to clear CKOSCSEL.
2. Switch CK_SYS from CK_PLL1 to CK_OSC4M, by setting the CKSEL bit.

Switch 8: switching CK_SYS from PLL to CK_RTC

If the CK_RTC clock source has already been selected, you can simply switch CK_SYS to CK_PLL1 by setting the CKSEL bit.

Otherwise, if a CK_RTC is not selected, use the following procedure (identical to the Switch 2 procedure):

1. Configure the clock source that you want to use for the RTC:
   a) LPOSC: Write the LPOSCEN control bit to enable the Low Power RC Oscillator
   b) OSC32K: Write the OSC32KEN control bit to enable the 32.768 kHz Oscillator, and wait for the OSC32KRDY flag to be set to ‘1’ indicating that the oscillator clock is stable.
   c) CK_OSC4M divided by 128: The OSC4M must be active.
2. Select the clock source to be used, by setting the CKRTCSEL[1:0] bits in the *Power Management Control register (MRCC_PWRCTRL)* register.
3. Execute the CKOSCSEL Bit Toggling Sequence to set it to ‘1’. This sequence is similar to the Low Power Bit Writing Sequence:
   a) Write CKOSCSEL to ‘1’
   b) Write CKOSCSEL to ‘1’
   c) Write CKOSCSEL to ‘0’
   d) Write CKOSCSEL to ‘1’
   e) Read CKOSCSEL until CKOSCSEL=1

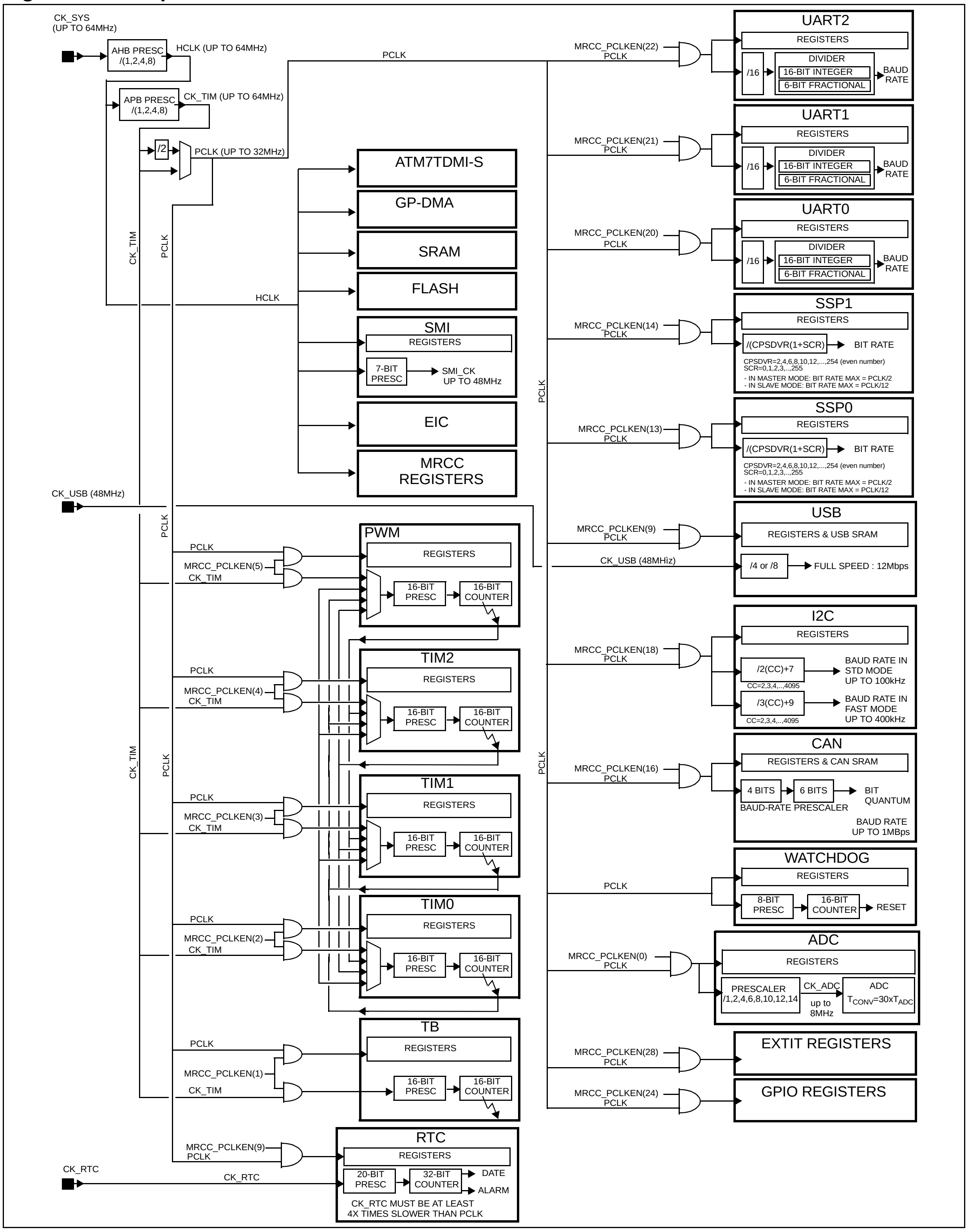
The CKOSCSEL bit is now written to ‘1’. CK_OSC is now driven by CK_RTC (see *Figure 12: System Clock scheme on page 42*).

Note: It is recommended to mask interrupts when executing this sequence.

4. Switch CK_SYS from CK_PLL1 to CK_RTC, by writing CKSEL to ‘1’.
   If OSC4M is selected as RTC clock source, the Clock Detector function prevents the CKSEL bit from going to ‘1’ if CK_OSC4M clock is not present (NCKDF flag=1).
5. Read both CKOSCSEL and CKSEL. If both are at 1, the clock system has successfully switched to CK_RTC.
6. To further reduce power consumption, you can first disable the PLL by clearing the PLLEN bit PLLEN and in a second step disable the main oscillator by writing ‘1’ in the OSC4MOFF bit.

2.8.15 Peripheral Clock scheme

Figure 24. Peripheral Clock scheme



2.9 Low Power modes

Several Low Power Modes are implemented to reach the best compromise between lowest power consumption, fastest start-up time and available wake-up sources:

- SLOW MODE (system clock speed is reduced)
- PCG MODE (APB Peripherals Clock Gating)
- WFI MODE (Wait For Interrupts)
- STOP MODE (all clocks are disabled)
- STANDBY MODE (part of the chip is un-powered)

Software has to execute the “Low Power Bit Writing Sequence” to enter WFI, STOP or STANDBY modes. This sequence protects the application from entering in Low Power mode unintentionally.

2.9.1 Low Power Bit Writing sequence

WFI, STOP or STANDBY Modes are entered by software writing the following specific sequence to the LP bit in the MRCC_PWRCTRL register.

Before executing the sequence, LP_DONE bit status must be previously cleared.

- Write LP bit to ‘1’
- Write LP bit to ‘1’
- Write LP bit to ‘0’
- Write LP bit to ‘1’
- Read LP: if successful, LP is first read at 1 and then cleared by hardware when entering in Low Power Mode.

If this sequence is not performed correctly or if LP_DONE was not cleared, the sequence is automatically reset and must be re-executed from the beginning.

To abort the sequence, simply write twice the LP bit to ‘0’.

The LP_DONE status flag is set after a successful LP Bit Writing Sequence (meaning that the system has entered and exit Low Power Mode). This information is useful to know if Low Power Mode has been effectively entered or not.

As soon as the LP Bit Writing Sequence is initiated:

- the MRCC_PWRCTRL register is locked (any write access other than to the LP bit will be discarded) until the sequence is completed or aborted.
- interrupts are masked (IRQ and FIQ signals to the ARM CPU are gated) until the sequence is completed or aborted.

Any interrupts which occur during this sequence will not be lost: they will be served after the sequence is completed or aborted.

Before performing the sequence, all pending bit interrupts used to wake-up from STOP mode should be cleared.

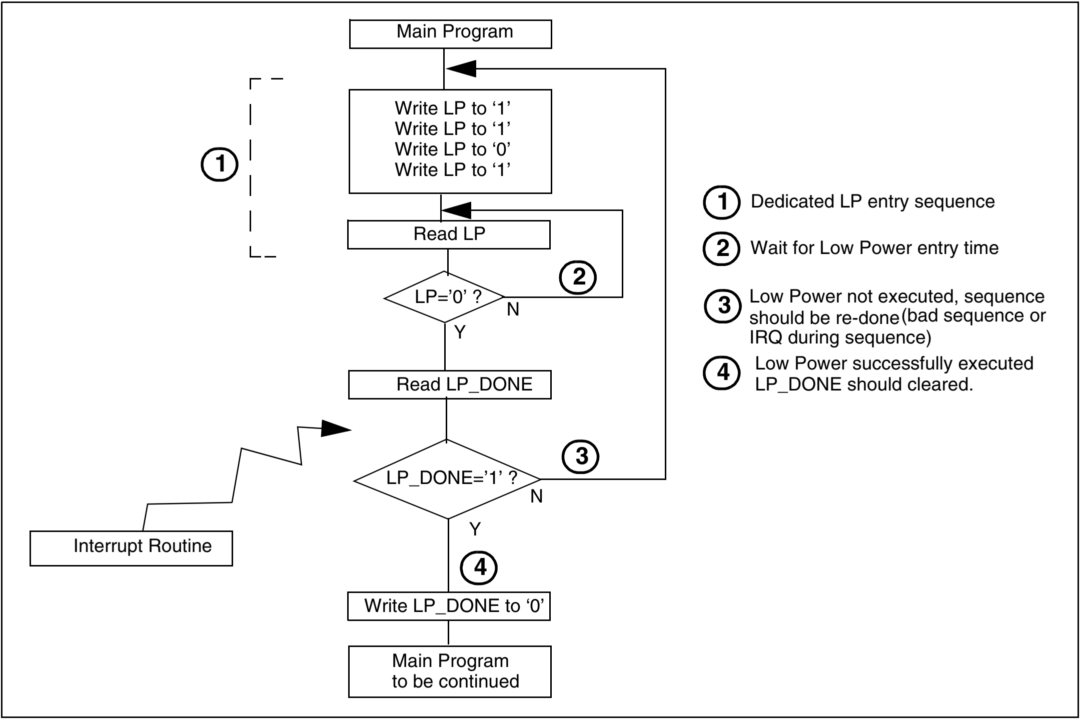
If any interrupts from an external interrupt line, an RTC alarm or NCKDF (no clock detected) event are still pending, the sequence will not performed correctly.

If an interrupt occurs during this sequence, the MCU will not enter Low Power Mode after completing the sequence, but will serve the interrupt instead.

In both cases, software must then reexecute the Low Power Bit Writing Sequence to effectively go into low power mode. It is possible to determine if Low Power Mode was entered by reading the LP and LP_DONE bits status after wake-up.

It is mandatory to follow this flow chart to manage the Low Power Mode:

Figure 25. Mandatory software flow for entering low power mode


Main Program
Write LP to '1'
Write LP to '1'
Write LP to '0'
Write LP to '1'
1
Read LP
LP='0' ?
2
N
Y
Read LP_DONE
Interrupt Routine
LP_DONE='1' ?
3
N
Y
4
Write LP_DONE to '0'
Main Program to be continued
1 Dedicated LP entry sequence
2 Wait for Low Power entry time
3 Low Power not executed, sequence should be re-done (bad sequence or IRQ during sequence)
4 Low Power successfully executed LP_DONE should cleared.

The choice of low power mode entered with this sequence depends on the state of the LPMC bits described in the *Power Management Control register (MRCC_PWRCTRL) on page 76*:

Table 8. Low Power mode selection

Control bits				Low Power Mode Selected
LPMC[1:0]		BURST	WFI_FLASH_EN	
0	0	-	-	STOP Mode
0	1	-	-	Software Reset
1	0	0	0	WFI, Flash disabled (DMA not allowed in WFI)
		0	1	WFI, Flash enabled (DMA allowed in WFI)
		1	0	Forbidden
		1	1	WFI, Flash enabled (DMA allowed in WFI)
1	1	-	-	STANDBY Mode

2.9.2 SLOW mode

In SLOW mode, power consumption is reduced by slowing down the main clocks. It is possible to use all the device functions of the chip, but at reduced speed.

To enter Slow Mode, the HPRESC[1:0] and PPRESC[2:0] bits prescaler in the MRCC_CLKCTL register must be programmed to reduce the CPU and/or peripheral clock frequency.

It is also possible to use the RTC clock as system clock. This is done by programming the CKOSCSEL control bit with a special sequence of write instructions (see *Section 2.8.12: Switching CK_SYS from CK_OSC4M to CK_PLL1 (PLL or FREEOSC) on page 52*). In this configuration, the PLL can be disabled by software to reduce power consumption. When exiting from this mode, if PLL has been disabled, the software must re-enable the PLL in the same way as after a RESET (see *Section 2.8.13: Switching CK_SYS from CK_RTC to CK_PLL1 (PLL or FREEOSC) on page 53*).

2.9.3 PCG mode: Peripherals Clocks Gated mode

It is possible to gate or ungate each clock feeding the APB peripherals independently. This is done by writing to the MRCC_PCLKEN register. This can be done at any time, allowing you to save power whenever these peripherals are not used, adapting dynamically to the needs of the application. Please refer to *Section 2.10.4: Peripheral Clock Enable register (MRCC_PCLKEN) on page 80* for the register description.

By default, after reset, all APB peripherals are in PCG Mode (their clock is disabled).

The clock to the Watchdog is an exception, it cannot be gated in PCG mode, it is always clocked by PCLK.

Some peripherals (USB, TB, TIM, PWM) use a clock source other than PCLK which can be gated using specific control bits:

- CK_USB 48 MHz clock gated by the PLL2EN bit in the *Clock Control Register (MRCC_CLKCTL)*
- CK_TIM: clock to TB, TIM, PWM timers gated by specific bits in the *Peripheral Clock Enable register (MRCC_PCLKEN)*

2.9.4 WFI mode: Wait For Interrupt mode

In WFI mode you reduce power consumption by stopping the CPU. The program stops executing but peripherals are kept running and the register contents are preserved. Execution restarts when an interrupt request is sent to the EIC.

WFI mode entry procedure

Use the following procedure to enter WFI mode:

1. First select WFI Mode by programming the LPMC[1:0] and the WFI_FLASH_EN bits in the *Power Management Control register (MRCC_PWRCTRL)*.
2. Program the LP_PARAM bits in the MRCC_PWRCTRL register to enable or disable selected parts of the MCU circuitry in WFI mode
3. Execute the Low Power Bit Writing Sequence as described in *Section 2.9.1: Low Power Bit Writing sequence on page 57*.

LP mode control parameters

If the Flash is used in **Burst mode**, it must be kept enabled in WFI mode (WFI_FLASH_EN bit = 1).

- If the WFI_FLASH_EN bit is set, the Flash memory is kept enabled in WFI mode and the system can perform DMA transfers.
- The LP_PARAM bits in the MRCC_PWRCTRL register have no effect in this case.

If the Flash is used in **Non-Burst mode**, the Flash may be disabled in WFI mode (WFI_FLASH_EN bit = 0) and the LP_PARAM bits in the MRCC_PWRCTRL register used to further reduce power consumption in WFI mode as follows:

- If WFI_FLASH_EN is reset, DMA must be disabled when entering WFI mode.
- FLASH OFF (LP_PARAM14). The Flash can be put in low power mode (LP_PARAM14=0) or disabled (LP_PARAM14=1). In the second case, the static consumption of the Flash is removed, but latency is incurred when exiting WFI mode.
- Other LP_PARAM bits have no effect in WFI mode.

Wake-up events

Wake-up from WFI mode can be performed either by an external interrupt from an I/O pin or an internal interrupt generated by one of the internal peripherals (assuming that its clock has not been disabled by software) or by NCKDF (no clock detected) interrupt.

DMA interrupts can also be used, but only if the WFI_FLASH_EN bit in the *Power Management Control register (MRCC_PWRCTRL)* has been previously set.

You can also combine WFI mode with PCG mode. However in this case, the peripherals that are disabled cannot generate the interrupt used to exit from WFI mode and another interrupt source has to be available.

It is also possible to wake-up from WFI triggered by an external interrupt line (configured in the EXTIT registers) without having enabled the corresponding IRQ in the EIC. In this case, the external interrupt event will cause a wake-up from WFI, but no IRQ routine will be invoked. The program will simply resume execution.

The RTC can wake up the MCU from WFI in two ways:

- either using the global interrupt of the RTC (Alarm, Second or Overflow)
- or through external interrupt line #15 which is connected only to the RTC alarm. In this case, this external interrupt must be properly configured. (see *Section 4.8.5: External interrupt line mapping on page 126*)

However, it is recommended to use the global interrupt of the RTC to wake-up from WFI.

Wake-up latency

If the Flash memory is kept enabled (WFI_FLASH_EN) or in low power mode (LP_PARAM14=0), there will be no latency when resuming after wake-up from WFI mode.

If Flash memory is disabled in WFI Mode, using the LP_PARAM14 bit in the *Power Management Control register (MRCC_PWRCTRL)*, latency occurs when restarting due to the Flash start-up time. Wait states are inserted until the Flash is ready. This latency is not deterministic and may vary, depending on temperature or process dispersion.

Note: This latency does not occur if the software fetches from SRAM when it restarts. Any access to Flash will be performed correctly, but will result in wait states being inserted if the Flash is not ready.

The Flash must be configured all the above control parameters, even if SRAM is used to fetch parts of the software.

Table 9. WFI mode functionality

Control bits		Functionality in WFI mode											
							Wake-up resources						
LP_PARAM14: Flash off	**RTC enabled**	**RTC clock source**	**Voltage Regulators**	**OSC4M Oscillator**	**PLL + Clock Detector (CKD)**	**Flash**	**NRSTIN**	**Internal interrupts**	**External interrupts**	**RTC Alarm**	**Clock Detector**	**Wake-up latency due to:**	**Clock configuration state after wake-up**
1	No	All	Main VREG	On	On	Off (in non Burst mode)	Yes	Yes if enbl ed	All	No	Yes	Flash	not changed
	Yes									Yes			
0	No			On	On	On (enabled or low power mode				No		No latency	
	Yes									Yes			

2.9.5 STOP mode

In this mode CK_SYS is stopped. This means that the CPU and all the peripherals clocked by CK_SYS or derived clocks are disabled. As a result the AHB and APB peripherals are not clocked, so the digital part of the MCU consumes no power other than the leakage current due to the process technology.

In STOP mode, all the registers and SRAM contents are preserved, the PLL and clock configuration can remain unchanged.

STOP mode entry procedure

Use the following procedure to enter STOP mode:

1. First select STOP Mode by programming the LPMC[1:0] bits in the MRCC_PWRCTRL register.
2. Program the LP_PARAM bits in the MRCC_PWRCTRL register to enable or disable some peripherals in STOP mode
3. Execute the Low Power Bit Writing Sequence as described in *Section 2.9.1: Low Power Bit Writing sequence on page 57*.

LP mode control parameters

To further reduce power consumption in STOP mode, you can disable additional parts of the MCU when it enters STOP Mode. You configure this by programming the LP_PARAM[15:13] bits in the MRCC_PWRCTRL register:

- **OSC4M and PLL OFF (LP_PARAM15)**: This removes the power consumption of the PLL and the OSC4M oscillator, however the clock configuration returns to its reset state. So when resuming from Stop mode after a wake-up event, you have to manage the start-up time for oscillator stabilization and re-enable the PLL.
- **FLASH OFF (LP_PARAM14)**: This saves the static power consumption of the Flash, but some latency for Flash start-up is incurred when waking up from STOP mode.
- **MVREG OFF (LP_PARAM13)**: When this parameter is set, the **LPVREG** (Low Power Voltage Regulator) powers the whole circuit, saving the static consumption of the MVREG (Main Voltage Regulator). In addition, the OSC4M oscillator, PLL and Flash are also switched off (all three LP_PARAM bits 13, 14 and 14 are set) because the Low Power Regulator does not have enough power to drive them. When resuming from Stop mode after a wake-up event, start-up latency is incurred.

Note: 1 *When using the LP_PARAM15 bit to switch off OSC4M in STOP mode, it is recommended to clear the NCKDF bit just before entering STOP mode. Otherwise, if NCKDF=1 when entering STOP mode, OSC4M will not be disabled and will continue to use power.*

2 *If OSC4M is not used as the system clock source, it is strongly recommended to switch it off by setting the OSC4MOFF bin STOP mode even if LP_PARAM 15 is set and NCKDF is cleared as recommended in note 1 above.*

Table 10. STOP mode functionality

Control bits				Functionality in STOP mode											
									Wake-up resources						
LP_PARAM13: MVREG off	LP_PARAM15: OSC4M off	LP_PARAM14: Flash off	RTC enabled	RTC clock source	Voltage Regulators	OSC4M Oscillator	PLL + Clock Detector (CKD)	Flash	NRSTIN	Internal interrupts	External interrupts	RTC Alarm	Clock Detector interrupt	Wake-up latency due to:	Clock configuration state after wake-up
1	1	1	No	None	LP VREG	Off	Off	Off	Yes	No	All	No	No	MVREG + Flash + OSC4M	Reset State
			Yes	LPOSC, OSC32K								Yes			
0	1	1	No	None	Main VREG	Off	Off	Off				No		FLASH + OSC4M	
			Yes	LPOSC, OSC32K								Yes			
	1	0	No	None		Off	Off	On				No		OSC4M	
			Yes	LPOSC, OSC32K								Yes			
	0	1	No	All		On	On	Off				No	Yes	Flash	not changed
			Yes									Yes			
	0	0	No			On	On	On				No		No latency	
			Yes									Yes			

Power Consumption in STOP mode

Table 11 gives an approximate indication of the power savings that can be gained using the LP_PARAM15:13 bits in the *Power Management Control register (MRCC_PWRCTRL)*.

Table 11. Typ. power consumption of circuits controlled by LP mode parameters

Control bit	Circuit	Power Consumption Saving (under typical conditions with 3.3V single power scheme)
LP_PARAM15	OSC4M oscillator and PLL	1.8 mA
LP_PARAM14	Flash memory	515 µA
LP_PARAM13	MVREG Main Voltage Regulator	130 µA

When all the LP_PARAM bits are enabled, the MCU power consumption consists only of the static consumption of the Low Power Regulator (and the RSM) plus the leakage currents of the digital logic (a total of 20µA typical at T_A= 25° C for 3.3 V supply).

In dual supply mode (VREG_DIS pin=1), the power consumption consists only of the leakage current (10 µA typical at T_A = 25° C on V_{18} supply).

Note: The Low Power RC Oscillator (LPOSC) and the 32.768kHz crystal oscillator (OSC32K) are still active if previously programmed (RTC is still working if clocked by one of these two clocks).

The ADC or USB can also consume power unless they are disabled before entering STOP mode.

Wake-up events

Wake-up from STOP mode can be triggered by a reset, an external interrupt event or an RTC alarm or NCKDF (no clock detected) interrupt. Refer to *Table 10: STOP mode functionality on page 63*.

It is also possible to wake-up from STOP mode triggered by an external interrupt line (configured in the EXTIT registers) without having enabled the corresponding IRQ in the EIC. In this case, the External Interrupt Event will exit from STOP, but no IRQ routine will be invoked. The program will simply resume execution.

The user program must clear the external interrupt line pending bit in the Pending Register (EXTIT_PR) which caused the wake-up from STOP mode. If this pending bit is not cleared, the next STOP mode entry procedure will be ignored and program execution will continue.

The RTC can wake up the MCU from STOP mode only through external interrupt line #15 which is internally connected to the RTC alarm. In this case, this external interrupt must be properly configured (see *Section 4.8.5: External interrupt line mapping on page 126*). Note that only the RTC alarm is connected to external interrupt line #15 (not the RTC second or overflow interrupt).

Wake-up latency

When the OSC4M oscillator, the Flash memory and the MVREG voltage regulator are kept active, the application is able to restart immediately with almost no latency.

If Flash memory is disabled in STOP Mode, using the LP_PARAM14 bit in the *Power Management Control register (MRCC_PWRCTRL)*, latency occurs when restarting due to the Flash start-up time. Wait states are inserted until the Flash is ready. This latency is not deterministic and may vary, depending on temperature or process dispersion.

Note: This latency does not occur if the software fetches from SRAM when it restarts. Any access to Flash will be performed correctly, but will result in wait states being inserted if the Flash is not ready.

If you disable the OSC4M oscillator in STOP Mode, using the LP_PARAM15 bit in the *Power Management Control register (MRCC_PWRCTRL)*, the clock configuration is lost and returns to reset state. Software must re-configure the clocks when the MCU wakes-up from STOP mode.

Debug mode

If you are using the debug features of the ARM7TDMI-S, and the application you are running puts the MCU in STOP mode, the debug connection will be lost, because the ARM7TDMI-S core is no longer clocked.

However, using Low Power Debug mode, you can debug your software even if you make extensive use of STOP mode. To enable this feature, set the LPMC_DBG bit after RESET. In this configuration, whenever a STOP mode entry sequence is successfully executed, the MCU goes into WFI Mode instead of STOP mode, so the core keeps running and its debug features remain active.

As WFI is performed instead of STOP mode, the WFI_FLASH_EN must be set (see *Section 2.9.4*);

2.9.6 STANDBY mode

This mode is only available when using the embedded regulators (pin VREG_DIS tied to 0).

In STANDBY Mode, the MVREG (Main Voltage Regulator) is disabled and the internal power switch opened (see *Figure 4: STANDBY mode in power supply scheme 1 on page 31*). This powers off the kernel circuitry, reducing power consumption to almost nil. Only the backup circuitry remains powered by the LPVREG (Low Power Voltage Regulator).

This mode is particularly important for applications which require very low consumption even when a rise in temperature occurs: there is almost no dispersion in temperature because the leakage currents are very low (most of the digital part is not powered).

The contents of the registers and SRAM are lost. The backup circuitry can be used to wake-up the microcontroller. Some backup registers also remain powered and can be used to store some parameters. This backup circuitry consists of:

- RTC Counter and Alarm mechanism
- Wake-up Logic
- Backup Registers (8 bytes)

The Wake-up logic switches the power to the kernel back on. The kernel is kept under RESET until the internal voltage is correctly regulated; at this point the interface between the kernel and Backup block is reconnected, and the CPU restarts from its RESET sequence.

STANDBY mode entry procedure

Use the following procedure to enter STANDBY mode:

1. First select STANDBY Mode by programming the LPMC[1:0] bits in the MRCC_PWRCTRL register.
2. Execute the Low Power Bit Writing Sequence as described in *Section 2.9.1: Low Power Bit Writing sequence on page 57*.

The Low Power Mode Control Parameters (LP_PARAM bits in the MRCC_PWRCTRL) have no effect in STANDBY mode.

To enter STANDBY mode, the WKPF bit must be reset. Otherwise, the STANDBY mode entry procedure will be ignored and program execution will continue.

The user program should also take into account the case of a wake-up event occurring during the STANDBY mode entry procedure: in this case the STANDBY mode entry procedure will be ignored and program execution will continue.

Consumption in STANDBY mode

All the peripherals are disabled except the LPVREG (and its associated RSM). The RTC and its clock source (OSC32K or LPOSC) can be either kept disabled or activated.

If the RTC is disabled, the consumption is reduced to the static consumption of the Low Power Regulator (and the RSM) which is 20µA typical. The leakage currents due to the technology are negligible and there is almost no dispersion in temperature (most of the digital part is not powered)

The RTC clocked by the LPOSC adds around 2µA typical consumption.

The RTC clocked by the OSC32K adds less than 1µA typical consumption.

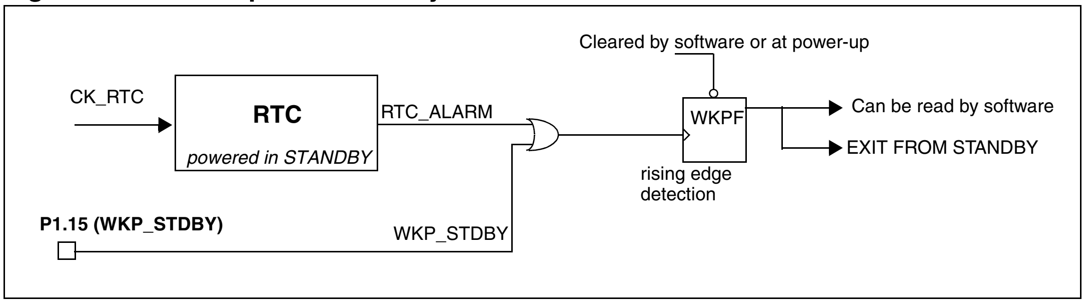
Wake-up events

Wake-up from STANDBY mode can only be performed by:

- a reset
- a rising edge RTC alarm event (see *Figure 44: RTC simplified block diagram* and *Figure 45 on page 170*
- a rising edge on the WKP_STDBY pin

Refer to *Table 12: STANDBY mode functionality on page 68*.

Figure 26. Wake-up from Standby mode



After wake-up from STANDBY mode, program execution will restart in the same way as after a RESET (the vector reset will be fetched). However, if a wake-up event occurs during the STANDBY entry procedure, the STANDBY mode entry procedure will be ignored and the program execution will continue. The user program should take into account the case of a wake-up event occurring during the STANDBY mode entry procedure.

Some status flags helps software to determine if it is resuming from RESET or from STANDBY mode and if STANDBY is exit because of a wake-up event or because of an external reset.

Refers to status bits STDBF and WKPF in *Section 2.10.2: Reset Flag and Status register (MRCC_RFSR) on page 74*).

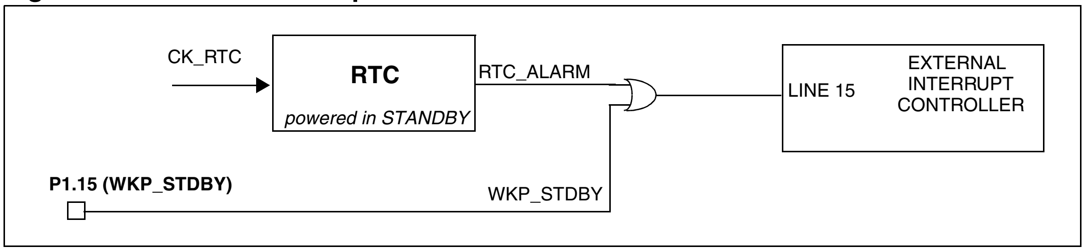
The RTC can wake-up the MCU from STANDBY mode at each RTC alarm event, independantly from external interrupt line #15 (no need to configure it). Note that the RTC Second or Overflow interrupts do not wake-up the MCU from STANDBY.

The condition which wakes-up the MCU from STANDBY is a rising edge of the output of an OR gate between the P1.15 input pin and the RTC alarm event, see *Figure 27*.

Consequently, to wake-up from STANDBY by an RTC alarm, you must:

- Apply low level to P1.15 pin.
- Configure the RTC to generate the RTC alarm

Figure 27. External Interrupt Line 15 internal connexions



Wake-up latency

The latency is the same as when resuming from reset.

I/O states in STANDBY mode

In STANDBY mode, all GPIOs are configured in high impedance except the WKP_STDBY pin which is kept in input mode.

All other pins are disabled (XT1, XT2, USB_CK, etc.) except the NRSTIN, NRSTOUT and XRTC1/2 pins which are still functional.

Debug mode

If you are using the debug features of the ARM7TDMI-S, and the application you are running puts the MCU in STANDBY mode, the debug connection will be lost, because the ARM7TDMI-S core is no longer powered.

Table 12. STANDBY mode functionality

	Feature	Functionality in STANDBY mode	
		RTC enabled	**RTC disabled**
	RTC clock source	LPOSC or OSC32K	None
	Voltage Regulators	LPVREG	
	OSC4M oscillator	Not powered	
	PLL + Clock Detector (CKD)	Not powered	
	Flash memory	Not powered	
Wakeup resources	**NRSTIN**	Yes	
	External Interrupts	WKP_STDBY pin	
	RTC Alarm	Yes	No
	Clock Detector	No	
	Wake-up latency due to:	MVREG + FLASH + OSC4M	
	Clock configuration state after wake-up	Reset state	

2.9.7 Auto-Wake-Up (AWU) from Low Power mode

The RTC can be used to wake-up the MCU from Low Power Mode without depending on an external interrupt (Auto-Wake-Up mode). The RTC provides a programmable time base for waking-up from STOP or STANDBY mode at selectable intervals (using the alarm event). For this purpose, two alternative RTC clock sources can be selected by programming the CKRTCSEL[1:0] bits in the MRCC_PWRCTRL register:

- Low Power 32.768kHz crystal oscillator (OSC32K).
  This clock source provides a precise time base with very low power consumption (less than 1µA added consumption in typical conditions)
- Internal Low Power RC Oscillator (LPOSC)
  This clock source has the advantage of saving the cost of the 32.768kHz crystal. This internal RC Oscillator is designed to add minimum power consumption (2µA added consumption in typical conditions).
  Because its frequency dispersion varies significantly with temperature and process, the LPOSC should be calibrated if a precise time base is needed. Refer to *Section 2.9.8*.

2.9.8 LPOSC oscillator calibration

Because the frequency of the RC oscillator is around 300 kHz with a strong deviation in process and temperature (frequency can be between 150 kHz and 500 kHz), it can be necessary to calibrate it by software. To perform this calibration, software can use the RTCM control bit (RTC Measurement) in the MRCC_PWRCTRL register to connect the RTC clock input to the input capture of the TB timer. Then, the software can measure the RTC clock period using the input capture “1” of the TB timer, assuming TB is clocked by CK_SYS. According to this measurement done at the precision of the main oscillator (OSC4M), the software can adjust the programmable 20-bit prescaler of the RTC to get an accurate time base.

Use the following procedure to get a typical time base of 20ms with an acceptable accuracy:

- Starting from a typical RTC clock of 300kHz (3.33 µs), to obtain 20 ms as RTC time base, set the RTC prescaler to the typical value of “6000” (RTC_PRL=0x176F) with a precision relative to the RTC time base of 1/6000 = 0.016%.
- If the RTC clock speed deviates up to 500 kHz (2 µs), adjust the prescaler to “10000” (RTC_PRL=0x270F) to get 20 ms with a precision on the RTC time base of 1/10000 = 0.01%
- If the RTC clock speed deviates down to 150 kHz (6.66 µs), adjust the prescaler to “3000” (RTC_PRL=0xBB7) to get 20ms with a precision on the RTC time base of 1/3000 = 0.033%

For this reason, software should measure and re-calibrate the LPOSC whenever frequency dispersion can potentially occur (after RESET and when the temperature changes).

Note: *It is important to bear in mind that LPOSC is powered by V_{18BKP} and that it has a frequency dispersion subject to voltage level variations.*

Consequently, special care must be taken if using the LPOSC timebase to wake-up from STANDBY or STOP mode with bit LP_PARAM13 set (MVREG OFF).

In both cases, V_{18BKP} will drop from 1.8 V in RUN mode (where it has been calibrated) to around 1.4 V. When powered by 1.4 V, the period of LPOSC shifts around +30%.

Consequently, this shift can significantly impact the precision of the wake-up time, especially if the wake-up time is long.

For short wake-up times, V_{18BKP} decreases only slightly due to the external capacitors: for instance, 20 ms after MVREG has been disabled, V_{18BKP} is still around 1.72 V and will fall to 1.4V after a typical duration of around 600ms.

2.10 Register description

2.10.1 Clock Control Register (MRCC_CLKCTL)

Address offset: 00h
Access: 2 Wait States
Reset value: 0004 0000h
This Register is Word, Half-Word and Byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LOCK	LOCK IE	LOCK IF	MX[1:0]		Res.	PLL2 EN	PLLEN	CK SEL	CKUSB SEL	CKOSC SEL	Res.	NCKDIE	NCKDF	OSC4M OFF	OSC4MBYP
r	rw	rc	rw	rw		rw	rw	rw	rw	rw		rw	rc	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XTDIV 2	Reserved							MCOS[1:0]		MCOP	HPRESC[1:0]		PPRES[2:0]		
								rw	rw	rw	rw	rw	rw	rw	rw

Bit 31	**LOCK**: *PLL Locked* This bit is read only and is set and cleared by hardware. 0: PLL is disabled or not locked and can not be selected as CK_SYS source. 1: PLL is locked
Bit 30	**LOCKIE**: *PLL Lock interrupt enable* Set and cleared by software. This bit enables an interrupt to be generated when the lock signal of the PLL changes (toggles). 0: Lock interrupt disabled 1: Lock interrupt enabled
Bit 29	**LOCKIF:** *PLL Lock interrupt Flag* Set by hardware and cleared by software (writing 0). This bit is the interrupt pending bit, indicating that an interrupt has been generated due to a change of the Lock Flag of the PLL. It can be used to detect either a lock or an unlock of the PLL.
Bits 28:27	**MX[1:0]:** *PLL Multiplication factors* These bits are set and cleared by software, writable only when the system clock is the Main Oscillator (CKSEL=‘1’ and PLLEN=’0’). 00: Multiplication by 16 01: Multiplication by 15 10: Multiplication by 14 11: Multiplication by 12
Bit 26	Reserved, must be kept at 0.

Bit 25	**PLL2EN:** *PLL output 2 enable* This bit can be set and cleared by software, writable only when the PLL is enabled and locked. (bits PLLEN='1' and LOCK='1'). It can also be cleared by hardware when: –Entering STOP mode when the LP_PARAM15 bit in the MRCC_PWRCTRL register is set (PLL disabled in STOP mode). –No clock is detected on OSC4M (NCKDF=1). In that case, the emergency oscillator (FREEOSC) has been automatically activated. –The PLLEN bit is reset (PLL disabled): this guarantees that no clock other than 48 MHz is applied to the USB peripheral. 0: PLL2 Output disabled. 1: PLL2 Output enabled (CK_USB can be used as CK_PLL2).
Bit 24	**PLLEN:** *PLL enable* This bit is set and cleared by software: It can be set only if CK_SYS is driven by CK_OSC4M (CKSEL='1' and CKOSC='0'). It can be cleared only if CK_SYS is CK_OSC4M or CK_RTC (CKSEL=1). It is cleared by hardware when: –Entering STOP mode when the LP_PARAM15 bit is set (PLL disabled in STOP mode). –No clock is detected on OSC4M (NCKDF=1). In that case, the emergency oscillator (FREEOSC) has been automatically activated and CK_SYS is using this clock. 0: PLL disabled 1: PLL enabled
Bit 23	**CKSEL:** *CK_SYS Clock Source selection/Status* This bit is both a control and status bit. It is set and cleared by hardware or software. Write access: 0: Request to select CK_PLL1 clock as CK_SYS source. 1: Request to select CK_OSC clock as CK_SYS source Read access: 0: CK_PLL1 clock is used as CK_SYS source 1: CK_OSC clock is used as CK_SYS source. It is cleared by hardware when: –Entering STOP mode when the LP_PARAM15 bit is set (OSC4M disabled in STOP mode). –No clock is detected on OSC4M (NCKDF=1). In that case, the emergency oscillator (FREEOSC) has been automatically activated and CK_SYS is using this clock. **Note:** This bit is maintained at '1' by hardware when the PLL is enabled but not yet locked. It can be modified by software after the PLL is locked (LOCK=1), see *Figure 21*.
Bit 22	**CKUSBSEL:** *USB clock selection* This bit is set and cleared by software to select internal or external 48 MHz USB clock. 0: Internal USB clock used (CK_PLL2 output from PLL when PLL2EN is set). 1: External Alternate Function USB clock.

Bit 21	**CKOSCSEL:** CK_OSC *source selection* This bit can be only toggled by software by executing a dedicated bit writing sequence. Toggling this bit can be done only when the system clock is the PLL clock output (CKSEL=‘0’), The sequence consists of: –Writing in sequence ‘1’,‘1’, ‘0’, ‘1’ to the CKOSCSEL bit. –Reading the CKOSCSEL bit. If successful, the CKOSCSEL value toggles (changes from 1 to 0 or from 0 to 1). The sequence can fail only if the sequence is not respected. Reading this bit indicates the source of CK_OSC: 0: CK_OSC4M is used as CK_OSC source. 1: CK_RTC is used as CK_OSC source.
Bit 20	Reserved, forced by hardware to 0.
Bit 19	**NCKDIE:** *No Clock detected interrupt enable* This bit is set and cleared by software. It enables an interrupt to be generated when NCKDF bit is set (no clock is detected by the Clock Detector on OSC4M) 0: Clock Detector interrupt disabled 1: Clock Detector interrupt enabled
Bit 18	**NCKDF:** *No Clock Detected Flag* It is set by hardware when no clock is detected on CK_OSC4M (value after Reset). It is reset by software if a clock is detected on CK_OSC4M. When set, this bit indicates that a failure of the main clock signal (CK_OSC4M) has occurred and that the Emergency oscillator (FREEOSC) has been automatically selected as system clock. 0: OSC4M or external clock is detected. 1: No OSC4M or external clock is detected. This flag is always ‘1’ when OSC4M is disabled by software with bit OSC4MOFF=’1’).
Bit 17	**OSC4MOFF:** *Disable OSC4M Oscillator* This bit is set and cleared by software, This bit is not writable when the system clock is driven by OSC4M (even if driven by OSC4M through the RTC clock), or if the PLL is enabled. 0: OSC4M enabled 1: OSC4M disabled **Note:** The OSC4M goes off as soon as this bit is set: This is different from the LP_PARAM bit 15 bit which disables the OSC4M only in STOP mode
Bit 16	**OSC4MBYP:** *OSC4M Oscillator Bypass* This bit is set and cleared by software, This bit is not writable when the system clock is driven by OSC4M (even if driven by OSC4M through the RTC clock), or if the PLL is enabled. When set, the OSC4M is bypassed. In this case, you can apply an external clock to XT1 pin as main clock. 0: Oscillator selected (and enabled if OSC4MOFF=0) 1: Oscillator bypassed (external clock can be applied to XT1 pin) **Note:** Writing ‘1’ to the OSC4MOFF and OSC4MBYP bits simultaneously is not possible: the priority is given to the OSC4MOFF bit. OSC4MBYP will not be set.

Bit 15	**XTDIV2:** *Oscillator divider by 2* This bit is set and cleared by software, This bit is not writable when the PLL is enabled. It is used to provide 4 MHz clock to PLL when using 8 MHz Ceramic or Crystal. 0: Divider by 2 disabled, when 4 MHz Ceramic or Crystal is connected. 1: Divider by 2 enabled, when 8 MHz Ceramic or Crystal is connected.
Bits 14:8	Reserved, forced by hardware to 0.
Bits 7:6	**MCOS[1:0]** *Main Clock Out selection* These bits are set and cleared by software. They select the clock source for the MCO alternate function. To output the clock on the MCO pin, the associated alternate function must be enabled in the I/O port controller. 00: HCLK 01: PCLK 10: CK_OSC4M 11: CK_PLL2 **Note:** Care must be taken when changing the MCO clock selection: To avoid any glitches on the MCO pin, the alternate function must be disabled during the change. Refer to the GPIO configuration register description. **Note:** In STOP mode when LP_PARAM15 bit is 0 (OSC4M enabled), MCO is still active if CK_OSC4M or CK_PLL2 have been selected. **Note:** When using AHB/APB prescalers (when HCLK or PLCK is selected as the MCO clock source) the output will not be with 50% duty cycle,. However it’s possible to reach 50% duty cycle by setting the MCOP bit.
Bit 5	**MCOP** *Main Clock Out Prescaler* This bit is set and cleared by software. 0: MCO is applied on MCO pin without prescaling. 1: An additional divider by 2 is applied, reducing output frequency on MCO pin.
Bits 4:3	**HPRESC[1:0]** : *Prescaler selection for HCLK* These bits are set and cleared by software. See *Figure 16: Detailed Clock scheme on page 47* 00: HCLK = CK_SYS 01: HCLK = CK_SYS/2 10: HCLK = CK_SYS/4 11: HCLK = CK_SYS/8
Bits 2	**PPRESC2** *Prescaler selection for PCLK* This bit is set and cleared by software. It selects the divider by 2 between CK_TIM and PCLK See *Figure 16: Detailed Clock scheme on page 47* 0: PCLK=CK_TIM 1: PCLK=CK_TIM/2
Bits 1:0	**PPRESC[1:0]** *Prescaler selection for CK_TIM* PRESC[1:0] select the prescaler between HCLK and CK_TIM. See *Figure 16: Detailed Clock scheme on page 47* 00: CK_TIM=HCLK 01: CK_TIM=HCLK/2 10: CK_TIM=HCLK/4 11: CK_TIM=HCLK/8

2.10.2 Reset Flag and Status register (MRCC_RFSR)

Address offset: 04h
Access: 2 Wait States
Reset value: xxFF xxxxh
This Register is Word, Half-Word and Byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SWRF	WDG RF	EXTRF	WKPF	STDBF	BCOU NTF	BCOUNTM[7:0]							
r	r	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				SCOUNT[11:0]											
				r	r	r	r	r	r	r	r	r	r	r	r

Note: This is a READ/CLEAR register, Reset Flags are set by Hardware and can be cleared only by writing '0'.

Bits 31:30	Reserved, must be kept at reset value.
Bit 29	**SWRF** *Software reset flag* This bit is cleared by hardware at V_{DD_IO} power-up only if the embedded regulators are used (VREG_DIS pin=0) It is set by hardware when a software reset occurs. It can be cleared by software (writing zero). 0: No software reset occurred 1: The last reset was generated by a software reset. **Note**: This bit is valid only if the embedded regulators are used (VREG_DIS pin=0)
Bit 28	**WDGRF** *Watchdog reset flag* This bit is cleared by hardware at power-up and set by hardware when a watchdog reset occurs. It can be cleared by software (writing zero). 0: No watchdog reset occurred 1: The last reset was generated by a watchdog reset. **Note**: This bit is valid only if the embedded regulators are used (VREG_DIS pin=0)
Bit 27	**EXTRF** *External Reset flag* This bit is cleared by hardware at V_{DD_IO} power-up only if the embedded regulators are used (VREG_DIS pin=0). It is set by hardware when an external reset occurs (low level on NRSTIN pin). It can be cleared by software (writing zero). 0: No external reset occurred 1: The last reset was generated by an external reset. **Note**: This bit is valid only if the embedded regulators are used (VREG_DIS pin=0)

Bit 26	**WKPF** *Wake-Up Flag* This bit is set by hardware when a Wake-up event occurs (RTC Alarm or WKP_STDBY pin event) and cleared by hardware when the NRSTIN pin is low. It can be cleared by software (writing zero). It can be used to determine if the microcontroller has resumed from STANDBY mode due to a wake-up event (RTC Alarm or rising edge detected on the WKP_STDBY pin). 0: No Wake-Up event occurred 1: Wake-up triggered by RTC Alarm or WKP_STDBY pin occurred **Note**: This bit is valid whether embedded regulators are used or not (VREG_DIS pin=0 or 1).
Bit 25	**STDBF** *STANDBY Flag* This bit is cleared by hardware at power-up and set by hardware when the microcontroller enters STANDBY mode. It can be cleared by software (writing zero). It can be used to determine if the microcontroller has resumed from STANDBY mode. 0: No Wake-Up from STANDBY 1: Wake-up from STANDBY triggered by RTC Alarm, WKP_STDBY pin or NRSTIN occurred **Note**: This bit is valid only if the embedded regulators are used (VREG_DIS pin=0)
Bit 24	**BCOUNTF:** *Builder Counter Flag* This bit is Read Only. It indicates that the 8 Most Significant Bits of the Builder Counter are higher than the value programmed in the BCOUNTM[7:0] bits.
Bits 23:16	**BCOUNTM[7:0]:** *Builder Counter Maximum Value* These bits are set and cleared by software. At start-up, the Builder Counter starts counting up. When the 8 Most Significant Bits of the 12-bit Builder Counter reach the value written in the BCOUNTM[7:0] bit field, the Builder Counter is stopped and the BCOUNTF flag is set (the OSC4M oscillator is considered to be stabilized). See *Figure 17 on page 48* 00: Temporization of 0 CK_OSC4M cycles 01: Temporization of 16 CK_OSC4M cycles ... FF: Temporization of 4096 CK_OSC4M cycles The default value of BCOUNTM[7:0] is FFh (corresponding to 4096 OSC4M cycles) **Note**: If the NCKDF bit is set, it must be cleared by software to start the builder counter.
Bits 15:12	Reserved, must be kept at reset value.
Bits 11:0	**SCOUNT[11:0]:** *CK_SYS Counter* These bits are read only. SCOUNT[11:0] represents the number of CK_SYS clock cycles elapsed during the Flash response time (WAIT states are inserted) after reset has been de-asserted. This delay depends on the Flash response time and is not deterministic.

2.10.3 Power Management Control register (MRCC_PWRCTRL)

Address offset: 08h
Access: 2 Wait States
Reset value: Bits 31:24 xxxx x0xxb
Bits 23:0: 00 0000h
This Register is Word, Half-Word and Byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSC32K RDY	OSC32K BYP	OSC32K EN	LPOSCEN	CKRTCOK	RTCM	CKRTCSEL[1:0]		Reserved							EN33V
r	rw	rw	rw	r	rw	rw	rw								rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LP_PARAM[15:13]			Reserved					LP_ DONE	Reserved		WFI_FLASH_EN	LPMC_DBG	LPMC[1:0]		LP
rw	rw	rw						rc			rw	rw	rw	rw	rw

Bit 31	**OSC32KRDY**: *32k oscillator ready* This bit is read only. 0: OSC32K not ready 1: OSC32K ready (stable and ready to use for very low power consumption). **Reset state:** – When the embedded regulators are enabled (pin VREG_DIS tied to ground), this bit is only reset after a V_{DD_IO} power-up. Then any internal or external reset will have no effect on this bit. – When the embedded regulators are disabled (pin VREG_DIS tied to '1'), this bit is reset when applying an external RESET on NRSTIN pin.
Bit 30	**OSC32KBYP**: *32k oscillator bypass* This bit is set and cleared by software. It is not writable when CK_SYS is driven by CK_RTC (CKSEL='1' & CKOSCSEL='1') 0: OSC32K oscillator can be enabled using OSC32KEN bit. 1: OSC32K oscillator bypassed (external clock can be applied on XRTC1 pin). **Note:** When bypassed, the CK_OSC32K clock is inverted compared to XRTC1 input pin. **Reset state:** – When the embedded regulators are enabled (pin VREG_DIS tied to ground), this bit is only reset after a V_{DD_IO} power-up. Then any internal or external reset will have no effect on this bit. – When the embedded regulators are disabled (pin VREG_DIS tied to '1'), this bit is reset when applying an external RESET on NRSTIN pin. the embedded regulators are disabled (pin VREG_DIS tied to '1'), this bit is reset when applying an external RESET on NRSTIN pin.

Bit 29	**OSC32KEN**: *32k oscillator enable* This bit is set and cleared by software. It is not writable when CK_SYS clock driven by CK_RTC (CKSEL='1' & CKOSCSEL='1') 0: OSC32K disabled 1: OSC32K enabled **Reset state:** – When the embedded regulators are enabled (pin VREG_DIS tied to ground), this bit is only reset after a V_{DD_IO} power-up. Then any internal or external reset will have no effect on this bit. – When the embedded regulators are disabled (pin VREG_DIS tied to '1'), this bit is reset when applying an external RESET on NRSTIN pin. applying an external RESET on NRSTIN pin.
Bit 28	**LPOSCEN**: *Low Power RC oscillator enable* This bit is set and cleared by software. It is not writable when CK_SYS is driven by CK_RTC (CKSEL='1' & CKOSCSEL='1') 0: LPOSC disabled 1: LPOSC enabled **Reset state:** – When the embedded regulators are enabled (pin VREG_DIS tied to ground), this bit is only reset after a V_{DD_IO} power-up. Then any internal or external reset will have no effect on this bit. – When the embedded regulators are disabled (pin VREG_DIS tied to '1'), this bit is reset when applying an external RESET on NRSTIN pin.
Bit 27	**CKRTCOK**: *CK_RTC OK* This bit is read only. It indicates that the RTC clock is present and ready to be used. It is reset each time CKRTCSEL bit is modified and is set on the second rising edge of the new CK_RTC. 0: CK_RTC not ready 1: CK_RTC OK **Note 1:** When you change CK_RTC from one active source to another (using CKRTCSEL), it is mandatory to wait for the CKRTCOK bit to be set (CKRTCOK='1') before disabling the old CK_RTC source. **Note 2:** When resuming from STANDBY mode, this flag is reset. It will be set on the second rising edge of CK_RTC.
Bit 26	**RTCM**: *RTC Measurement* This bit is set and cleared by software. 0: CK_RTC not connected to TB timer IC1 1: CK_RTC is connected to Input Capture 1 (IC1) of the TB timer for measurement and software calibration using the RTC prescalers (see *Section 2.9.8: LPOSC oscillator calibration on page 68* for more details)

Bits 25:24	**CKRTCSEL[1:0]**: *CK_RTC clock source selection* These bits are set and cleared by software. They are not writable when CK_SYS is driven by CK_RTC (CKSEL='1' & CKOSCSEL='1') 01: CK_RTC = CK_OSC4M divided by 128 10: CK_RTC = CK_OSC32K 11: CK_RTC = CK_LPOSC **Note:** When you change CK_RTC from one active source to another (using CKRTCSEL), you must wait for the CKRTCOK bit to be set (CKRTCOK='1') before disabling the old CK_RTC source. **Caution:** The reset state is '00' (CK_RTC disabled). For proper clock management, software must switch directly from one source to another (writing, 01,10,11). Writing '00' is forbidden. **Reset state:** – When the embedded regulators are enabled (pin VREG_DIS tied to ground), these bits are only reset after a V_{DD_IO} power-up. Then any internal or external reset will have no effect on these bits. – When the embedded regulators are disabled (pin VREG_DIS tied to '1'), these bits are reset when applying an external RESET on NRSTIN pin.
Bits 23:17	Reserved
Bit 16	**EN33V**: *I/O voltage range* This bit is set and cleared by software. 0: I/O Pins are optimized for 5V operation, I/O can be used at 3.3V but with downgraded timing characteristics. 1: I/O Pins are optimized for 3.3V, but 5V operation is forbidden.
Bits 15:13	**LP_PARAM[15:13]**: *Low Power mode parameters* Set and cleared by software. When the corresponding bit is set, the corresponding circuitry is disabled when entering STOP or WFI mode: – LP_PARAM15 = Disable OSC4M oscillator and PLL This parameter is only available in STOP mode – LP_PARAM14 = Disable Flash memory This parameter is only available in STOP or WFI mode – LP_PARAM13 = Disable MVREG voltage regulator This parameter is only available in STOP mode When writing this bit, the other LP_PARAM bits (14 and 15) are also set in order to disable the OSC4M oscillator and the Flash memory.
Bits 12:8	Reserved, must be kept at reset value.
Bit 7	**LP_DONE**: *Low Power Bit Writing Sequence has been performed* This bit is set by hardware when entering STOP or WFI Low Power Mode. Refer to *Section 2.9.1: Low Power Bit Writing sequence on page 57*. It must be cleared by software, once Low Power Mode is exited. 0: System has not resumed from STOP or WFI Low Power Mode. 1: System has resumed from STOP or WFI Low Power Mode. **Note:** In STANDBY mode, this bit is not kept powered-on. Consequently, when restarting from STANDBY mode, this bit is in reset state. The WKPF and STDBF flags in the MRCC_RFSR register can be read to determine if the system has resumed from STANDBY mode.
Bits 6:5	Reserved

Bit 4	**WFI_FLASH_EN**: *WFI mode with Flash enabled* This bit is set and cleared by software. You must set this bit if the Flash is configured in Burst mode. You should also set this bit if the Flash is configured in non-Burst mode if you want to perform DMA transfers while the microcontroller is in WFI mode. When this bit is set, the LP_PARAM14 bit is don’t care and the Flash memory is always enabled when entering WFI mode. 0: WFI Mode with Flash in low power mode (DMA not allowed during WFI). 1: WFI Mode with Flash enabled (DMA allowed during WFI). **Note:** This bit must be set, even if Flash memory is not involved in the DMA transfers performed in WFI mode.
Bit 3	**LPMC_DBG**: *Low Power Debug Mode* This bit is set and cleared by software. This mode is useful when using the ARM7TDMI-S Debug features, because entering STOP or STANDBY mode would cause the loss of the debug connection. 0: Low Power Debug Mode disabled 1: Low Power Debug Mode enabled: successful execution of the Low Power Bit Writing Sequence puts the microcontroller in WFI mode, overriding the LPMC[1:0] bit setting. See also *Table 89 on page 432*
Bits 2:1	**LPMC[1:0]**: *Low Power Mode Configuration* These bits are set by software and cleared by hardware. They select the Low Power mode entry or Software Reset performed after a successful LP Bit Writing Sequence. Refer to *Section 2.9.1 on page 57* 00: STOP mode (LP_PARAM bits will be used) 01: Software Reset 10: WFI mode (LP_PARAM and WFI_FLASH_EN bits will be used) 11: STANDBY mode **Notes:** –If the LPMC_DB bit is set, WFI Mode will be entered even if LPMC=00b or 11b. –If the WFI_FLASH_EN bit is cleared, DMA must be disabled before entering WFI mode (LPMC=10b). –If the WFI_FLASH_EN bit is set, DMA does not have to be disabled before entering WFI mode (LPMC=10b) and DMA interrupts can wake-up from WFI. In addition, the LP_PARAM14 bit is don’t care and the Flash memory is always enabled.
Bit 0	**LP**: *Low Power Mode Entry* A successful Low Power Mode entry sequence consists of: –Writing in sequence ‘1’,‘1’, ‘0’,‘1’ to the LP bit (any other accesses during this sequence will abort LP mode entry). Once this sequence is successfully executed, the LP bit must be read once, returning the value ‘1’. –When the LP bit is read with a return value of ‘1’, this puts the device in WFI, STOP, SRESET or STANDBY mode depending on the state of the LPMC[1:0] bits. The LP bit is then automatically cleared as soon as Low Power Mode is entered. A read access to LP bit means: 0: System is in RUN mode. 1: System is going to enter Low Power Mode during the next cycle. **Note:** In STANDBY mode, this bit is not kept powered-on. Consequently, when restarting from STANDBY mode, the LP bit is in reset state.

2.10.4 Peripheral Clock Enable register (MRCC_PCLKEN)

Address offset: 10h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCLKEN[31:0]																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**PCLKEN[31:0]**: *Peripheral Clock enable* Each of these bits gates the clock of the corresponding APB peripheral listed in *Table 13: APB peripherals on page 80*. 0: Peripheral clock disabled 1: Peripheral clock enabled

2.10.5 Peripheral Software Reset register (MRCC_PSWRES)

Address offset: 14h
Access: 2 Wait States
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSWRES[31:0]																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**PSWRES[31:0]**: *Peripheral Software Reset* Each of these bits controls the reset of the corresponding APB peripheral listed in *Table 13* below. 0: Peripheral is reset by a system reset 1: Peripheral is kept under reset independently from the system reset

Table 13. APB peripherals

Bit No	Peripheral	Bit No	Peripheral	Bit No	Peripheral	Bit No	Peripheral
31		23		15		7	
30		22	UART2	14	SSP1	6	
29		21	UART1	13	SSP0	5	PWM
28	EXTIT	20	UART0	12		4	TIM2
27	RTC	19		11		3	TIM1
26		18	I2C	10		2	TIM0
25		17		9	USB	1	TB
24	GPIO	16	CAN	8		0	ADC

2.10.6 Backup register 0 (MRCC_BKP0)

Address offset: 20h
Access: 2 Wait States
Reset value: xxxx xxxxh

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BKP0[31:0]																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**BKP0[31:0]**: *Backup Register 0* This register can be used to store 32 bits of user data. Its content is retained even in STANDBY mode. When embedded regulators are used (pin VREG_DIS tied to ground), this register is reset only after a V_{DD_IO} power-up. Otherwise it is never reset.

2.10.7 Backup register 1 (MRCC_BKP1)

Address offset: 24h
Access: 2 Wait States
Reset value: xxxx xxxxh

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BKP1[31:0]																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**BKP1[31:0]**: *Backup Register 1* This register can be used to store 32 bits of user data. Its content is retained even in STANDBY mode. When embedded regulators are used (pin VREG_DIS tied to ground), this register is reset only after a V_{DD_IO} power-up. Otherwise it is never reset.

2.10.8 MRCC Register map

Table 14. MRCC Register map

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00	MRCC_CLKCTL	Clock Control Register																															
04	MRCC_RFSR	Reset Flag Status Register																															
08	MRCC_PWRCTRL	Power Control Register																															
10	MRCC_PCLKEN	Peripheral Clock Enable Register																															
14	MRCC_PSWRES	Peripheral Software Reset Register																															
20	MRCC_BKP0	Backup Register 0																															
24	MRCC_BKP1	Backup Register 1																															

See *Table 1* for base address

3 General Purpose I/O ports (GPIO)

3.1 Functional description

Each of the General Purpose I/O Ports has three 32-bit configuration registers (PC0, PC1, PC2), a 32-bit Data register (PD) and a 32-bit mask register (PM).

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the General Purpose IO (GPIO) Ports, can be individually configured by software in several modes:

– Input floating
– Input Pull-up
– Input-Pull-down
– Analog Input
– Output Open-Drain
– Output Push-Pull
– Alternate Function

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words (Half-word or Byte accesses are not allowed). The purpose of the mask register is to allow atomic read/modify accesses (or bitwise write accesses) to any of the GPIO registers. This way, there is no risk that an IRQ occurs between the read and the modify access.

Figure 28 shows the basic structure of an I/O Port bit.

Figure 28. Basic structure of an I/O port bit

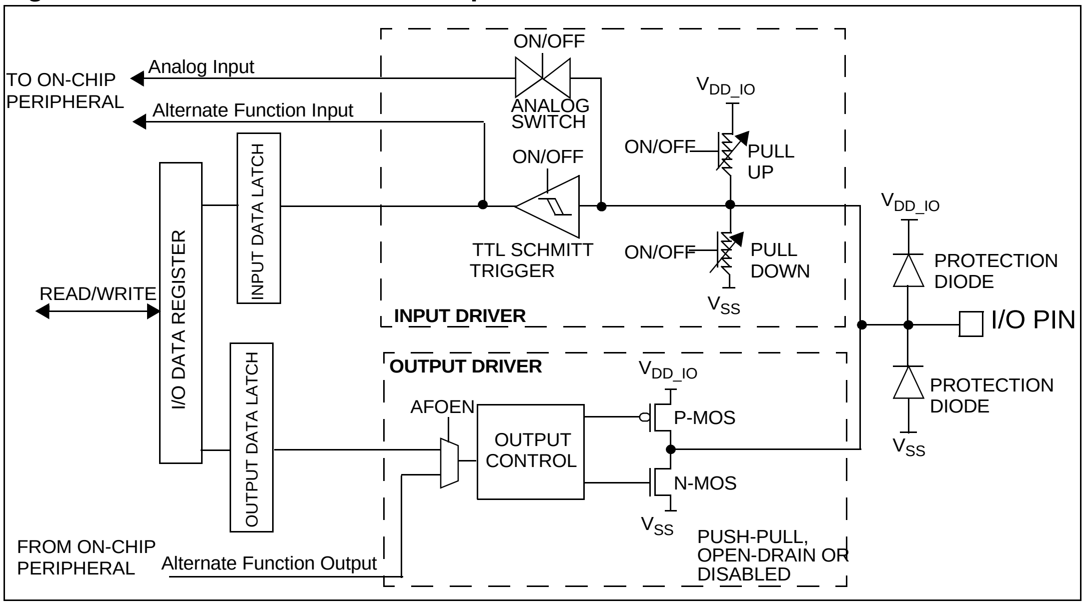


Table 15. **Port bit configuration table**

Configuration mode		Input buffer	PxD Register		PxC2 Register	PxC1 Register	PxC0 Register
			Read access	Write access			
INPUT	Input Floating (reset state)	Input floating	I/O pin	don't care	0	0	1
	Input Floating	Input Floating	I/O pin	don't care	0	1	0
	Input Pull-Down	TTL Pull-Down	I/O pin	0	0	1	1
	Input Pull-Up	TTL Pull-Up	I/O pin	1	0	1	1
	Analog input	AIN	0	don't care	0	0	0
OUTPUT	Output Open-Drain	TTL floating	I/O pin	0 or 1	1	0	0
	Output Push-Pull	not used	last value written	0 or 1	1	0	1
	Alternate Function Open-Drain	TTL floating	I/O pin	don't care	1	1	0
	Alternate Function Push-Pull	TTL floating	I/O pin	don't care	1	1	1

3.1.1 General Purpose I/O (GPIO)

During and just after reset the alternate functions are not active and the I/O ports are configured in Input Floating mode (PxC2=0, PxC1=0, PxC0=1).

When configured as output, the value written to the I/O Data register is loaded in the Output Latch. The Output Latch holds the data to be output. It is possible to use the output driver in Push-Pull mode or Open-Drain mode (only the N-MOS is activated when outputting 0).

The Input Latch captures the data present on the I/O pin at every APB clock cycle.

A read access to the I/O Data register reads the Input Latch or the Output Latch depending on whether the Port bit is configured as input or output Open-Drain or Push-Pull.

All GPIO pins have a internal weak pull-up and weak pull-down which can be activated or not when configured as input.

In all low power modes except STANDBY mode, GPIO states are preserved. In STANDBY mode, all GPIOs are put in high impedance with the exception of the WKP_STDBY pin which is kept in input mode.

Note: *Care must be taken when configuring an I/O port from one mode to another, because an unexpected intermediate state could perturb the application. Program the registers using only intermediate states that do perturb your application. For instance, it is important to be aware that in "analog input" mode, the output of the Schmitt trigger is forced to '0'.*

3.1.2 Atomic Bit Set or Bit Reset (Bit-wise write operations)

The purpose of the Port Mask Register is to allow atomic read/modify accesses (or bit-wise write operations) to any of the GPIO registers.

The bit-wise instructions in the ARM7 Instruction Set only apply on the internal ARM7 Ri registers.

Consequently, it is not possible to perform bit-wise write operations (like a bit set or a bit clear) directly on an I/O Port register. This has to be done in three operations :

- Load the whole 32-bit Port Data register into an Ri register
- Modify the Ri register using the bitwise ARM7 instruction.
- Store back the whole 32-bit result into the Port Data Register.

Because this is not an atomic operation, it is possible for an Interrupt Subroutine (ISR) to be served between the Load and the Store access. If this ISR sets or clears some other bits of the Port register, storing back the Port register can corrupt the Port.

One solution is to disable the interrupts. However, using the Port Mask Register, this is not needed. You can simply:

- First, program the Port Mask Register (PxM) to mask the bits which you don't want to modify.
- Then, program the port registers (PxC2, PxC1, PxC0 and PxD).

The masked bits will not be modified.

This Mask applies to all the configuration and Data registers (PxC2, PxC1, PxC0 and PxD).

Note: It is recommended that each interrupt subroutine which accesses the port registers stacks the Port Mask Register. Otherwise, an interrupt occurring between the modification of the PxM register and a bit manipulation on the PxD registers might lead to a corruption of the port bits.

3.1.3 Alternate Functions (AF)

It is necessary to program the Port Bit Configuration Register before using a default alternate function.

- For alternate function inputs, the port must be configured in Input mode (floating, pull-up or pull-down) and the input pin must be driven externally

Note: It is also possible to emulate the AFI input pin by software by programming the GPIO controller. In this case, the port should be configured in Alternate Function Output mode. And obviously, the corresponding port should not be driven externally as it will be driven by the software using the GPIO controller.

- For Alternate Function Outputs, the port must be configured in Alternate Function Output mode (Push-Pull or Open-Drain).
- For bi-directional Alternate Functions, the port bit must be configured in Alternate Function Output mode (Push-Pull or Open-Drain). In this case the input driver is configured in input floating mode

If you configure a port bit as Alternate Function Output, this disconnects the output latch and connects the pin to the output signal of an on-chip peripheral.

If software configures a GPIO pin as Alternate Function Output, but no peripheral Output Alternate Function exists for that pin (refer to the datasheet pin description table), its output is not specified.

Special case of SSP bi-directional alternate functions

When using the SSP, the Alternate functions MISO, MOSI, NSS and SCK consist of bi-directional alternate functions. They must be configured as Alternate Function Output through the Port Configuration register:

- When configuring the SSP in master mode, the MISO pin is automatically used as alternate function input and the output driver is automatically disabled (even if still programmed as alternated function output in the Port Configuration registers).
  In addition, when configured in master mode, the MOSI pin is always driven (never left Hi-Z) even if the SSP is in idle mode (no transmission)
- When configuring the SSP in slave mode, the MOSI, SCK and NSS pins are automatically configured as alternate functions inputs and the output drivers are automatically disabled (even if still programmed as alternated function output in the Port Configuration registers).
  In addition, when configured in slave mode, the MISO pin is left Hi-Z when the NSS pin is high or when the SOD control bit (Slave Output Disable) is set.

Configuring I2C alternate functions

After reset release, the I2C is able to detect START condition on SDA and SCL lines even if the I2C is not configured. (see *Section 13.2.3: SDA/SCL line control on page 316*).

Consequently, care must be taken when configuring SDA and SCL as Alternate Function Open-Drain in order not to create parasitic falling edges.

The states to avoid are:

- output 0
- input pull-down
- analog input (because the output of the schmitt trigger goes to 0)

Consequently, the configuration must be done in the following order:

1. Reset state: PC2,1,0=001 PD=0: input floating
   -> SDA=SCL = '1' due to external pull-up
2. Write PD=1: PC2,1,0=001 PD=1: input floating
   -> SDA=SCL = '1' due to external pull-up
3. Write PC1=1: PC2,1,0=011 PD=1: input pull-up
   -> SDA=SCL = '1' due to internal and external pup
4. Write PC0=0: PC2,1,0=010 PD=1: input floating
   -> SDA=SCL = '1' due to external pull-up
5. Write PC2=1: PC2,1,0=110 PD=1: AF Open Drain
   -> SDA=SCL = '1' because the I2C does not drive the line when disabled (I2C PE=0)

Software remapping of I/O alternate functions

To optimize the number of peripherals available for the 64-pin or the 100-pin package, it is possible to remap some alternate functions to some other pins. This is achieved by software, by programming the GPIO_REMAP0 register. In that case, the alternate functions are no more mapped to their original assignations.

It is possible to remap:

- TIM0 OC2/TI2 from P1.00/P1.01 to P0.04/P0.05.
  As a result, these alternate functions become available into the 64-pin package, but the SMI and SSP0 are no more available.
- TIM2 OC2/TI2 from P1.02/1.03 to P0.06/P0.07.
  As a result, TIM2/OC2 becomes available in the 64-pin package, but the SMI and SSP0 are no longer available.
- SSP1 SCLK/MISO/MOSI/NSS from P0.16/P0.17/P0.18/P0.19 to P0.08/P0.09/P0.10/P0.11.
  As a result, the analog channels 4 and 5 become available simultaneously with the SSP1, but the I2C and UART0 are no more available.
- UART1 RX/TX from P0.20/P0.21 to P0.14/P0.15
  When remapped, CAN is no more available and the external interrupt/event 5 is mapped on UART1_RX adding the possibility for the UART1 to exit from STOP mode.
- UART2 RX/TX from P0.24/P0.25 to P0.12/P0.13
  As a result, UART2 becomes available in the 64-pin package, but the hardware modem control of UART0 (RTS, CTS) and the CS1 of the SMI are no more available.

The 100-pin package provides another mean for optimizing alternate functions availability: the following alternate function outputs are double mapped to pins to PORT2 to free-up some analog channels and external event/interrupt lines. The user can choose between the two mapping by programming only the desired port bit as alternate function output.

- UART1_RTS is also mapped on P2.03, which frees the analog channel 6
- TIM2_OC1 is also mapped on P2.04, which frees the analog channel 0 or the external event/interrupt line 0
- PWM3N/PWM3/PWM2N/PWM2/PWM1N are also mapped on P2.05/P2.06/P2.07/P2.08/P2.09, which frees analog channel 9,10 and 11 and the external event/interrupt lines 7 and 8

Activating SMI alternate functions

Two cases occur:

- the MCU is not booting from the SMI. In this case, the SMI alternate functions are not mapped by default after RESET. However, software can map them by programming the bit SMI_EN of the GPIO_REMAP0 register. When selected, the default alternate functions dedicated to these pins are no more mapped.
  Software has also the flexibility to choose the number of SMI_CS to be activated by programming the bits SMI_CS1, SMI_CS2 and SMI_CS3 of the GPIO_REMAP0 register
- the MCU is booting from the SMI. In this case, the SMI alternate functions are mapped by default after RESET. But only CS0 is activated. If more banks are required, software must program the bits SMI_CS1, SMI_CS2 and SMI_CS3 of the GPIO_REMAP0 register.

The software remapping of the alternate functions (register GPIO_REMAP1) has priority over the activation of the SMI alternate functions (register GPIO_REMAP0).

Example: if software activates the SMI alternate functions by programming the bit SMI_EN and then remap by software some alternate functions which share the same pin (like TIM0_OC2/TI2), then the SMI alternate functions are no more available.

JTAG and GPIO Multiplexing

Five of the six JTAG I/O pins (JTDI, JTCK, JTMS, JTDO and RTCK) are multiplexed with GPIOs. After a RESET all the JTAG pins are automatically assigned as JTAG alternate functions.

To use the JTAG pins as GPIOs, the application must set the DBGOFF bit in the GPIO_REMAP1R register. For more details, refer to *Section 18.9.2: JTAG and GPIO multiplexing on page 434*

Using BOOT pins as GPIOs

After reset, the two BOOT pins are configured as input floating and they are internally sampled just after the de-assertion of the RESET and before the ARM CPU fetches the RESET vector.

If the application wants to use these BOOT pins as GPI/Os, it is mandatory that each time a reset occurs, the BOOT pins are externally driven to their proper states up to the RESET vector fetch by the ARM CPU. The application can achieve this, by connecting external Pull-up or Pull-down to the BOOT pins.

3.1.4 Alternate Functions table

The alternate functions for each pin are listed below:

Table 16. Alternate Function mapping(1)

Pin Name	100-pin package	64-pin package	Analog inputs	External event/ interrupt	Default Alternate Function		Software Alternate Function Remapping
					when not BOOTING from SMI	when BOOTING from SMI	
P0.00	X	X			TIM0_OC1 / **BOOT0**		
P0.01	X	X			TIM0_TI1		**MCO**
P0.02	X	X	AIN0	EXT0	TIM2_OC1 (also mapped on P2.04)		
P0.03	X	X	AIN1		TIM2_TI1		
P0.04	X	X			SSP0_NSS	***SMI_CS0***	***TIM0_OC2***
P0.05	X	X		EXT1	SSP0_SCLK	***SMI_CK***	***TIM0_TI2***
P0.06	X	X			SSP0_MISO	***SMI_DIN***	***TIM2_OC2***
P0.07	X	X		EXT2	SSP0_MOSI	***SMI_DOUT***	***TIM2_TI2***
P0.08	X	X		EXT3	I2C_SCL		***SSP1_SCLK***
P0.09	X	X			I2C_SDA		***SSP1_MISO***
P0.10	X	X		EXT4	UART0_RX	***SMI_CS3***	***SSP1_MOSI or SMI_CS3***
P0.11	X	X			UART0_TX / **BOOT1**	SMI_CS2 / **BOOT1**	***SSP1_NSS or SMI_CS2***
P0.12	X	X	AIN2		UART0_CTS	SMI_CS1	***UART2_RX or SMI_CS1***
P0.13	X	X			UART0_RTS **/ RTCK**		***UART2_TX***
P0.14	X	X		EXT5	CAN_RX		***UART1_RX***
P0.15	X	X			CAN_X		***UART1_TX***
P0.16	X	X			***SSP1_SCLK***		
P0.17	X	X	AIN3		***SSP1_MISO***		
P0.18	X	X			***SSP1_MOSI***		
P0.19	X	X	AIN4	EXT6	***SSP1_NSS*** / USB_CK		
P0.20	X	X			***UART1_RX***		
P0.21	X	X			***UART1_TX***		
P0.22	X		AIN5		UART1_CTS		
P0.23	X		AIN6		UART1_RTS (also mapped on P2.03)		
P0.24	X				***UART2_RX***		
P0.25	X				***UART2_TX***		
P0.26	X				UART2_CTS		
P0.27	X		AIN7		UART2_RTS (also mapped on P2.17)		
P0.28	X	X			TIM1_OC1		
P0.29	X	X	AIN8		TIM1_IC1		
P0.30	X				TIM1_OC2		
P0.31	X				TIM1_TI2		
P1.00	X				***TIM0_OC2***		
P1.01	X				***TIM0_TI2***		
P1.02	X				***TIM2_OC2***		

Table 16. Alternate Function mapping[(1)] (continued)

Pin Name	100-pin package	64-pin package	Analog inputs	External event/ interrupt	Default Alternate Function: when not BOOTING from SMI	Default Alternate Function: when BOOTING from SMI	Software Alternate Function Remapping
P1.03	X	X			***TIM2_TI2***		
P1.04	X	X	AIN9		PWM3N (also mapped on P2.05)		
P1.05	X	X		EXT7	PWM3 (also mapped on P2.06)		
P1.06	X	X	AIN10		PWM2N (also mapped on P2.07)		
P1.07	X	X		EXT8	PWM2 (also mapped on P2.08)		
P1.08	X	X	AIN11		PWM1N (also mapped on P2.09)		
P1.09	X	X		EXT9	PWM1		
P1.10	X	X		EXT10	PWM_EMGCY		
P1.11	X	X	AIN12	EXT11	UART0_RTS		
P1.12	X	X	AIN13	EXT12	-		
P1.13	X		AIN14	EXT13	-		
P1.14	X		AIN15		-		
P1.15	X			EXT15	WKP_STDBY		
P1.16	X	X			**JTDI (activated after RESET)**		
P1.17	X	X			**JTDO (activated after RESET)**		
P1.18	X	X			**JTCK (activated after RESET)**		
P1.19	X	X			**JTMS (activated after RESET)**		
P2.00	X				-		
P2.01	X				-		
P2.02	X				-		
P2.03	X				UART1_RTS (also mapped on P0.23)		
P2.04	X				TIM2_OC1 (also mapped on P0.02)		
P2.05	X				PWM3N (also mapped on P1.04)		
P2.06	X				PWM3 (also mapped on P1.05)		
P2.07	X				PWM2N (also mapped on P1.06)		
P2.08	X				PWM2 (also mapped on P1.07)		
P2.09	X				PWM1N (also mapped on P1.08)		
P2.10	X				-		
P2.11	X				-		
P2.12	X				-		
P2.13	X				-		
P2.14	X				-		
P2.15	X				-		
P2.16	X				-		
P2.17	X				UART2_RTS (also mapped on P0.27)		
P2.18	X				-		
P2.19	X				-		

1. BOOT and JTAG pins are highlighted in bold.
Alternate Functions which can be remapped to other pins by programming a register are highlighted in italic bold. When remapped to another pin, they are no longer available on the original pin.

3.1.5 External Interrupt / wake-up lines

Some ports have external interrupt/wake-up capability (refer to the datasheet). To use external interrupt / wake-up lines, the port must be configured in input mode. For more information on external interrupts, refer to *Section 4.8*.

3.1.6 Special case of WKP_STDBY pin (P1.15)

The WKP_STDBY pin (port P1.15) is always configured in input floating mode (whatever the GPIO register configuration) and is the only wake-up pin which can exit from STANDBY mode.

It can only be used as:

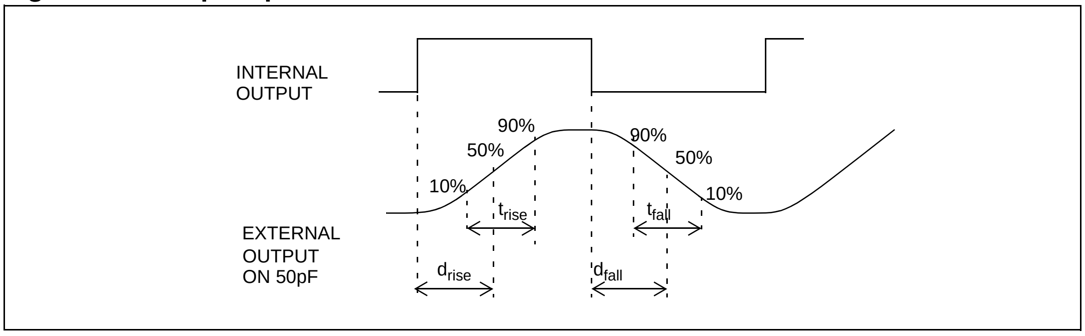
- an input floating GPIO
- an external interrupt/wake-up line (refer to *Section 4.8*)
- a wake-up event (rising edge) to exit STANDBY mode

3.1.7 Output drive & speed

Speed Definitions (refer to *Figure 29*):

Speed is defined on a 50pF output load.
Rise and Fall Transition time are noted as t_r and t_f
Rise and Fall Delay time are noted as d_r and d_f

Figure 29. Output speed definitions



Maximum operating frequency is defined when the stable high and low part of the sinusoidal inside one period is more than $(t_r + t_f)/2$, resulting in max frequency of $f_{max} = 2 / [3(t_r + t_f)]$

Maximum Skew between rise and fall delay must provide 45%-55% duty cycle.

3.1.8 Configuring Outputs for 5V or 3.3V Supply

There are three types of outputs, IO2, IO4, IO8 as indicated in the datasheet Pin Description table. The output speed can be adjusted according to the V_{DD_IO} power supply. This is done by software using a control register bit (EN33). Default value is 1 (for 3.3V operation). Refer to *Section 2.10.3: Power Management Control register (MRCC_PWRCTRL) on page 76*.

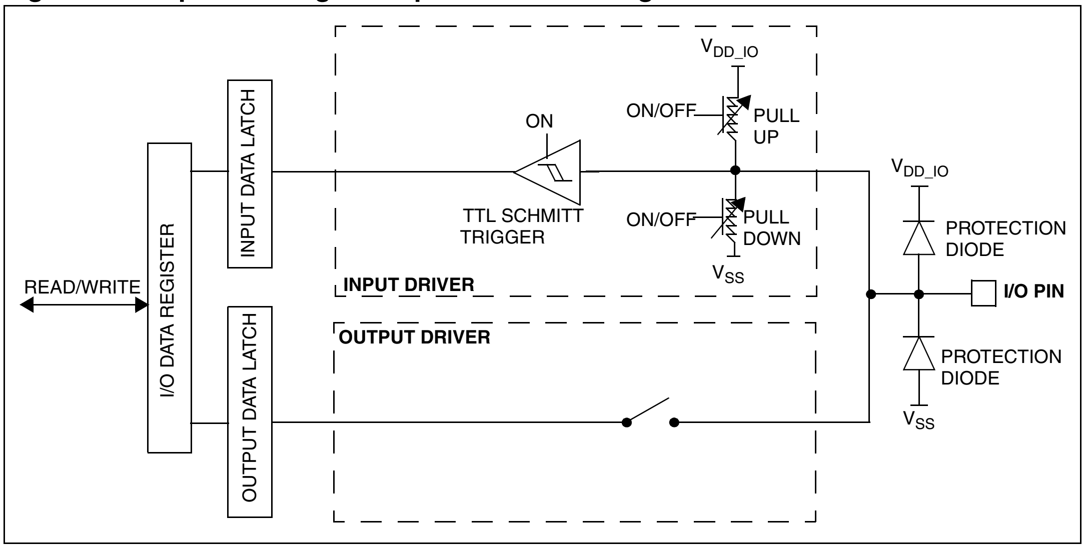
3.1.9 Input configuration

When the I/O Port is programmed as Input:

- The Output Buffer is disabled
- The Schmitt Trigger Input is activated
- The Analog Switch is disabled
- The weak pull-up and pull-down resistors are activated or not depending on input configuration (pull-up, pull-down or floating):
- The data present on the I/O pin is sampled into the Input Latch every APB clock cycle
- A read access to the Data register gets the value in the Input Latch.

The *Figure 30 on page 92* shows the Input Configuration of the I/O Port bit.

Figure 30. Input Floating/Pull Up/Pull Down configurations



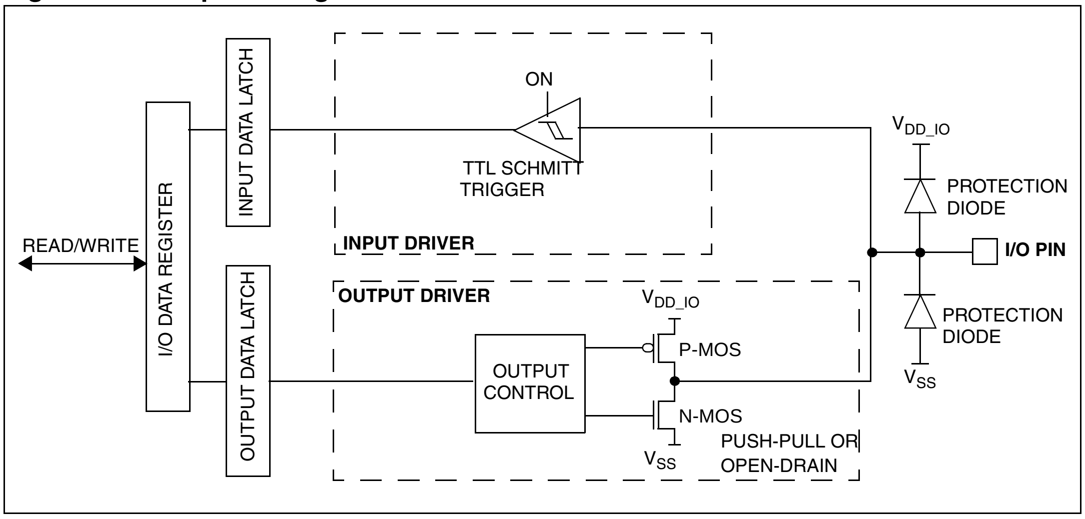
3.1.10 Output configuration

When the I/O Port is programmed as Output:

- The Output Buffer is enabled:
  - Open Drain Mode: A “0” in the Output Latch activates the N-MOS while a “1” in the Output Latch leaves the port in Hi-Z (the P-MOS is never activated)
  - Push-Pull Mode: A “0” in the Output Latch activates the N-MOS while a “1” in the Output Latch activates the P-MOS
- The Schmitt Trigger Input is activated.
- The Analog Switch is disabled
- The weak pull-up and pull-down resistors are disabled.
- The data present on the I/O pin is sampled into the Input Latch every APB clock cycle
- A read access to the I/O Data register gets:
  - the Output Latch value in Push-Pull mode (which corresponds to the last data written).
  - the Input Latch value in Open-Drain mode

The *Figure 31 on page 93* shows the Output configuration of the I/O Port bit.

Figure 31. Output configuration



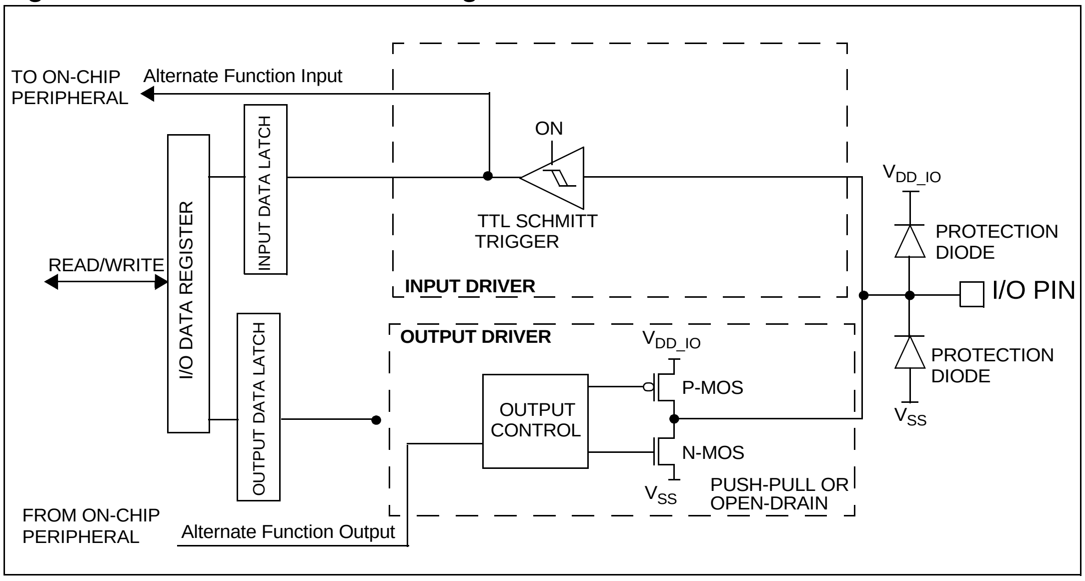
3.1.11 Alternate Function configuration

When the I/O Port is programmed as Alternate Function:

- The Output Buffer is turned on in Open Drain or Push-Pull configuration
- The Output Buffer is driven by the signal coming from the peripheral (alternate function out)
- The Schmitt Trigger Input is activated
- The Analog Switch is disabled
- The weak pull-up and pull-down resistors are disabled.
- The data present on the I/O pin is sampled into the Input Latch every APB clock cycle
- A read access to the I/O Data register gets:
  - the Output Latch value in Push-Pull mode (which corresponds to the last data written).
  - the Input Latch value in Open-Drain mode

The *Figure 32 on page 94* shows the Alternate Function Configuration of the I/O Port bit.

Figure 32. Alternate Function configuration



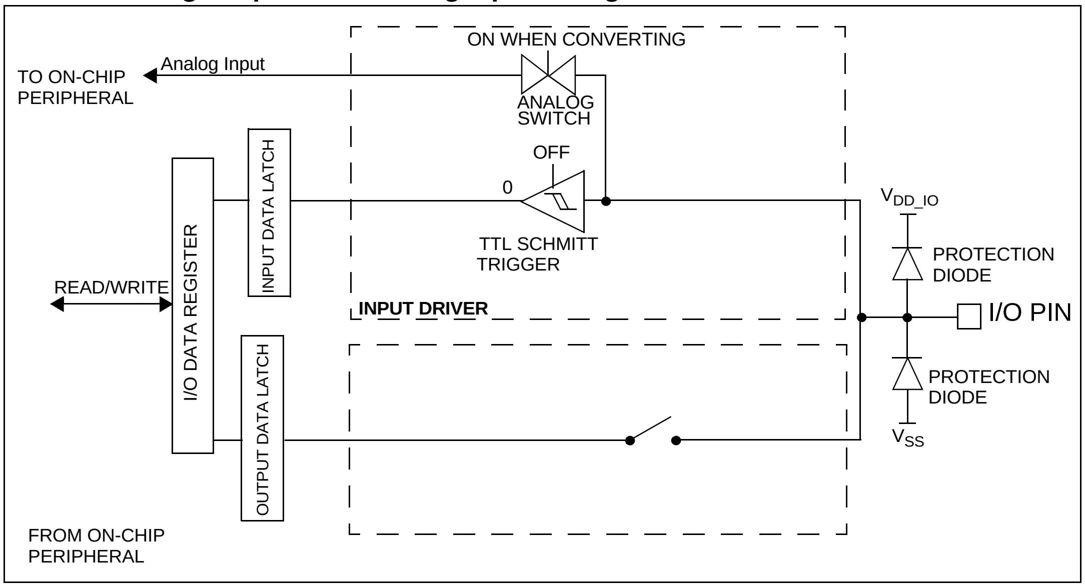
3.1.12 Analog Input configuration

When the I/O Port is programmed as Analog Input Configuration:

- The Output Buffer is disabled.
- The Schmitt Trigger Input is de-activated providing zero consumption for every analog value of the I/O pin. The output of the Schmitt Trigger is forced to a constant value (0).
- The weak pull-up and pull-down resistors are disabled.
- The analog switch is enabled by the ADC each time a conversion is progressing.
- read access to the I/O Data register gets the Input Latch (0).

The *Figure 17 on page 94* shows the High impedance-Analog Input Configuration of the I/O Port bit.

Table 17. High impedance-Analog Input configuration



3.2 Register description

The I/O port registers cannot be accessed by byte.

3.2.1 Port Configuration Register 0 (GPIO_PxC0)

Address Offset: 00h

Reset value: FFFF FFFFh

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PC0[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC0[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31:0	**PC0[31:0]:** *Port Configuration bits* See *Table 15 on page 84* to configure the I/O Port.

3.2.2 Port Configuration Register 1 (GPIO_PxC1)

Address Offset: 04h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PC1[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**PC1[31:0]:** *Port Configuration bits* See *Table 15 on page 84* to configure the I/O Port.

3.2.3 Port Configuration Register 2 (GPIO_PxC2)

Address Offset: 08h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PC2[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**PC2[31:0]:** *Port Configuration bits* See *Table 15 on page 84* to configure the I/O Port.

3.2.4 I/O Data Register (GPIO_PxD)

Address Offset: 0Ch

Reset value: xxxx xxxxh

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PD[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PD[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**PD[31:0]:** *I/O Data bits* A write access to this register always writes the data in the Output Latch. A read access reads the data from the Input Latch in Input and Alternate function configurations or from the Output Latch in Output and High impedance configurations. (*) Refer to the device I/O pin list

3.2.5 I/O Mask Register (GPIO_PxM)

Address Offset: 10h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PM[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PM[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**PM[31:0]:** *I/O Port Mask bits* This register allows you to write to selected bits in the I/O port registers (PxC2, PxC1, PxC0 and PxD). Each bit is written only if the corresponding PM bit is 0. If a PM bit is high, the corresponding bit in I/O port bit location is left unchanged. This can be used for fast BIT SET and BIT RESET without software masking and for easy I/O port configuration for different pin-outs. 0: Selected bit in I/O port registers writable 1: Selected bit in I/O port registers masked **Note:** The PM register has no effect on read access

3.2.6 I/O Remapping Register 0 (GPIO_REMAP0R)

Address offset: 20h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												SMI_CS3_EN	SMI_CS2_EN	SMI_CS1_EN	SMI_EN
												rw	rw	rw	rw

Bits 31:4	Reserved, always read as 0.
Bit 3	**SMI_CS3_EN** *SMI CS3 Enable* Set and cleared by software. This bit enables the SMI_CS3 Alternate Function on P0.10. 0: SMI_CS3 not mapped 1: SMI_CS3 mapped on P0.10
Bit 2	**SMI_CS2_EN** *SMI CS2 Enable* Set and cleared by software. This bit enables the SMI_CS2 Alternate Function on P0.11. 0: SMI_CS2 not mapped 1: SMI_CS2 mapped on P0.11
Bit 1	**SMI_CS1_EN** *SMI CS1 Enable* Set and cleared by software. This bit enables the SMI_CS1 Alternate Function on P0.12. 0: SMI_CS1 not mapped 1: SMI_CS1 mapped on P0.12
Bit 0	**SMI_EN** *SMI Enable* Set and cleared by software. This bit enables the following SMI Alternate Functions: 0: SMI_CS0, SMI_CK, SMI_DIN,SMI_DOUT not mapped 1: SMI_CS0 on P0.04, SMI_CK on P0.05, SMI_DIN on P0.06, SMI_DOUT on P0.07 **Note:** When booting from SMI, the alternate functions SMI_CS0, SMI_CK, SMI_DIN, and SMI_DOUT are automatically enabled on pins P0.04, P0.05, P0.06 and P0.07 respectively but the SMI_EN bit will not reflect this.

3.2.7 I/O Remapping Register 1 (GPIO_REMAP1R)

Address offset: 24h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										DBGOFF	UART1_ REMAP	UART2_ REMAP	SSP1_ REMAP	TIM2_ REMAP	TIM0_ REMAP
-										rw	rw	rw	rw	rw	rw

Bits 31:7	Reserved, always read as 0
Bit 6	Reserved, must be kept cleared.
Bit 5	**DBGOFF** *DEBUG and JTAG disable* Set and cleared by software. When set, the Alternate functions associated with the JTAG are disabled and the corresponding I/Os become free for General Purpose I/O. 0: JTDI on P1.16, JTDO on P1.17, JTCK on P1.18, JTMS on P1.19, RTCK on P0.13 1: JTAG Alternate functions not mapped. ARM7TDMI-S DEBUG mode disabled
Bit 4	**UART1_REMAP** *UART1 Alternate Function Mapping* 0: UART1_RX on P0.20, UART1_TX on P0.21 1: UART1_RX on P0.14, UART1_TX on P0.15
Bit 3	**UART2_REMAP** *UART2 Alternate Function Mapping* 0: UART2_RX on P0.24, UART2_TX on P0.25 1: UART2_RX on P0.12, UART2_TX on P0.13
Bit 2	**SSP1_REMAP** *SSP1 Alternate Function Mapping* 0: *SSP1*_SCLK on P0.16, *SSP1*_MISO on P0.17, *SSP1*_MOSI on P0.18, *SSP1*_NSS on P0.19 1: *SSP1*_SCLK on P0.08, *SSP1*_MISO on P0.09, *SSP1*_MOSI on P0.10, *SSP1*_NSS on P0.11
Bit 1	**TIM2_REMAP** *TIM2 Timer Alternate Function Mapping* 0: TIM2_OC2 on P1.02, TIM2_TI2 on P1.03 1: TIM2_OC2 on P0.06, TIM2_TI2 on P0.07
Bit 0	**TIM0_REMAP** *TIM0 Timer Alternate Function Mapping* 0: TIM0_OC2 on P1.00, TIM0_TI2 on P1.01 1: TIM0_OC2 on P0.04, TIM0_TI2 on P0.05

3.2.8 I/O Port Register map

The following table summarizes the registers implemented in each I/O port.

Table 18. GPIO register map

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00	GPIO_P0C0	P0C0[31:0]																															
04	GPIO_P0C1	P0C1[31:0]																															
08	GPIO_P0C2	P0C2[31:0]																															
0C	GPIO_P0D	P0D[31:0]																															
10	GPIO_P0M	P0M[31:0]																															
20	GPIO_REMAP0R	reserved																												SMI_CS3_EN	SMI_CS2_EN	SMI_CS1_EN	SMI_EN
24	GPIO_REMAP1R	reserved																										DBGOFF	UART1_Remap	UART2_Remap	SSP1_Remap	TIM2_Remap	TIM0_Remap
40	GPIO_P1C0	reserved												P1C0[19:0]																			
44	GPIO_P1C1	reserved												P1C1[19:0]																			
48	GPIO_P1C2	reserved												P1C2[19:0]																			
4C	GPIO_P1D	reserved												P1D[19:0]																			
50	GPIO_P1M	reserved												P1M[19:0]																			
80	GPIO_P2C0	reserved												P2C0[19:0]																			
84	GPIO_P2C1	reserved												P2C1[19:0]																			
88	GPIO_P2C2	reserved												P2C2[19:0]																			
8C	GPIO_P2D	reserved												P2D[19:0]																			
90	GPIO_P2M	reserved												P2M[19:0]																			

See *Table 2* for base address.

4 Interrupts

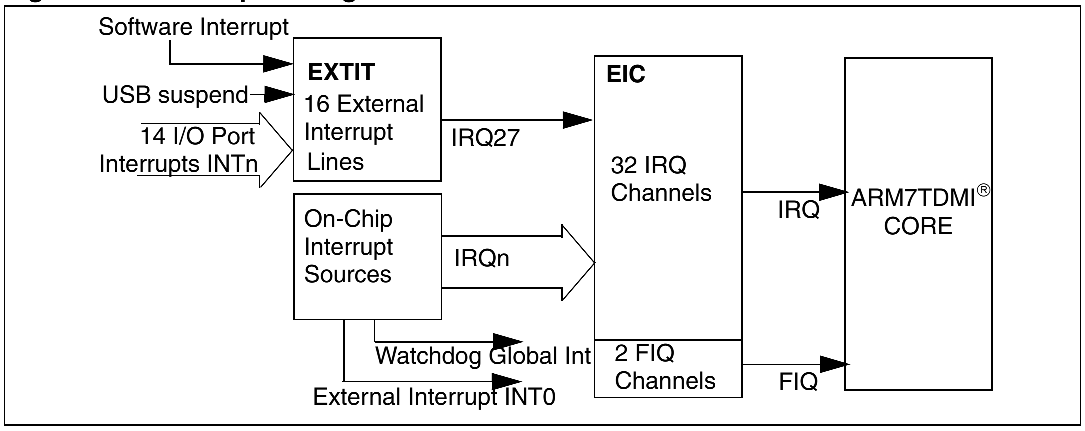
The ARM7 CPU provides two levels of interrupt, FIQ (Fast Interrupt Request) for fast, low latency interrupt handling and IRQ (Interrupt Request) for more general interrupts.

Table 19. ARM exception vector locations

Address	Exception
0x0000_0000	Reset
0x0000_0004	Undefined Instruction
0x0000_0008	Software Interrupt
0x0000_000C	Prefetch Abort (instruction fetch memory fault)
0x0000_0010	Data Abort (data access memory fault)
0x0000_0014	Reserved
0x0000_0018	IRQ
0x0000_001C	FIQ

The STR75x interrupt management system provides two interrupt management blocks: the EIC and EXTIT. Refer to *Figure 33*.

Figure 33. Interrupt management overview

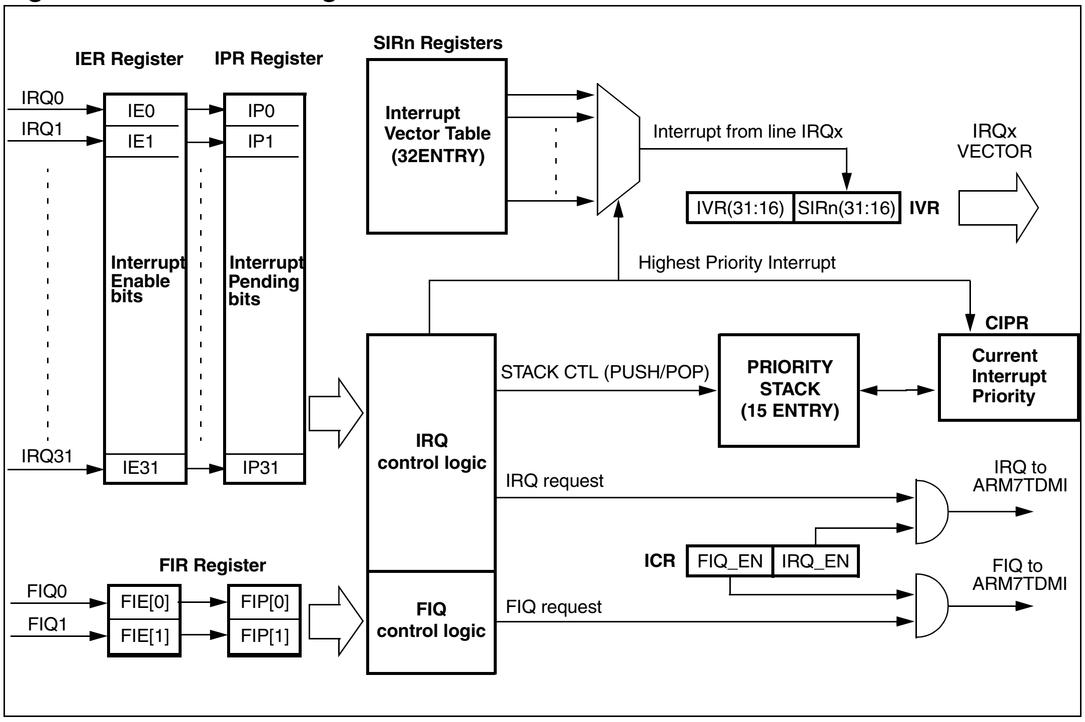


4.1 Enhanced Interrupt Controller (EIC)

The ARM7TDMI CPU provides two levels of interrupt, FIQ (Fast Interrupt Request) for fast, low latency interrupt handling and IRQ (Interrupt Request) for more general interrupts.

Hardware handling of multiple interrupt channels, interrupt priority and automatic vectorization require therefore a separate Enhanced Interrupt Controller (EIC).

Figure 34. EIC block diagram



The Enhanced Interrupt Controller (EIC) performs hardware handling of multiple interrupt channels, interrupt priority arbitration and vectorization. It provides:

- 32 maskable interrupt channels, mapped on the IRQ interrupt request line of the ARM CPU
- 16 programmable priority levels for each interrupt channel mapped on IRQ (15 = highest priority, 1= lowest priority, 0 = never served)
- Hardware support for interrupt nesting (up to 15 interrupt requests can be nested), with internal hardware nesting stack
- 2 maskable interrupt channels, mapped on FIQ interrupt request line of the ARM CPU, with neither priority nor vectorization

The EIC performs the following operations without software intervention:

- Rejects/accepts an interrupt request according to the related channel mask bit
- Compares all pending IRQ requests with the current priority level. The IRQ is asserted to the ARM7 if the priority of the current interrupt request is higher than the stored current priority in CIPR.
- Loads the address vector of the highest priority IRQ to the Interrupt Vector Register
- Saves the previous interrupt priority in the hardware priority stack whenever a new IRQ is accepted
- Updates the Current Interrupt Priority Register with the new priority whenever a new interrupt is accepted

4.1.1 IRQ Interrupt Vector table

Up to 32 interrupt channels are mapped on low priority ARM7TDMI interrupt request pin (IRQ). If multiple interrupt sources are mapped on the same interrupt vector, software has to read the peripheral interrupt flag register to determine the exact source of interrupt (see Interrupt Flags column in *Table 20*).

Table 20. IRQ Interrupt vector table

Vector	Acronym	Peripheral Interrupt
IRQ0	WAKEUP	External WKP_STDBY pin
IRQ1	TIM2	TIM2 Timer Output Compare 2
IRQ2	TIM2	TIM2 Timer Output Compare 1
IRQ3	TIM2	TIM2 Timer Input Capture 1 & 2
IRQ4	TIM2	TIM2 Timer Update
IRQ5	TIM1	TIM1 Timer Output Compare 2
IRQ6	TIM1	TIM1 Timer Output Compare 1
IRQ7	TIM1	TIM1 Timer Input Capture 1 & 2
IRQ8	TIM1	TIM1 Timer Update
IRQ9	TIM0	TIM0 Timer Output Compare 2
IRQ10	TIM0	TIM0 Timer Output Compare 1
IRQ11	TIM0	TIM0 Timer Input Capture 1 & 2
IRQ12	TIM0	TIM0 Timer Update
IRQ13	PWM	PWM Timer Output Compare 1, 2 & 3
IRQ14	PWM	PWM Timer Emergency
IRQ15	PWM	PWM Timer Update
IRQ16	I2C	I2C Global interrupt
IRQ17	SSP1	SSP1 Interrupt
IRQ18	SSP0	SSP0 Interrupt
IRQ19	UART2	UART2 Global interrupt
IRQ20	UART1	UART1 Global interrupt
IRQ21	UART0	UART0 Global interrupt
IRQ22	CAN	CAN Global interrupt
IRQ23	USB-LP	USB Low Priority Event Interrupt
IRQ24	USB-HP	USB High Priority Event Interrupt
IRQ25	ADC	ADC Global interrupt
IRQ26	DMA	DMA Global Interrupt
IRQ27	EXTIT	External Interrupt lines INT14 to INT1
IRQ28	MRCC	MRCC Clock Controller Global interrupt
IRQ29	Flash and SMI	Flash Global interrupt

Table 20. IRQ Interrupt vector table (continued)

Vector	Acronym	Peripheral Interrupt
IRQ30	RTC	RTC Global interrupt
IRQ31	TB	TB Timer Global Interrupt

4.1.2 FIQ Interrupt Vector table

Two maskable interrupt sources are mapped on FIQ vectors, as shown in <Blue HT>Table 21:

Table 21. FIQ Vector table

Vector	Acronym	Interrupt Source
FIQ0	EXTIT	External interrupt line INT0
FIQ1	WATCHDOG	Watchdog global interrupt

In most cases, you should only enable one FIQ source in your application. If you enable both FIQ sources, then you can determine the source of the interrupt by reading the FIQ pending bits in the EIC register. Bear in mind that FIQ has no priority mechanism, so if simultaneous FIQ events occur, software has to manage the priority by polling the pending bits and managing the concurrency.

4.1.3 IRQ Interrupt Structure

The EIC (Enhanced Interrupt Controller) implements an interrupt structure pointing the processor to the first instruction location of the channel-specific Interrupt Service Routine (ISR).

IVR (Interrupt Vector Register) is the EIC's 32-bit register at address 0xFFFF F818h acting as pointer. It is composed of two main fields: the upper half word (16 bit) is directly programmable, while the lower half word is the mirrored entry of a register table (named SIR) indexed by the interrupt channel.

The absolute address 0x0000 0018h is where the ARM7TDMI CPU jumps as consequence of an interrupt request on its IRQ line.

The STR750 considers the ARM's 0x0000 0018h location and the EIC's IVR location (0xFFFF F818h) as distinct addresses. The EIC IVR register must contain the absolute address of the interrupt service routine. The absolute 0x0000 0018h location must contain an instruction which jumps to the location pointed to by the IVR register.

For instance, the absolute location 0x0000 0018h should contain the following instruction:

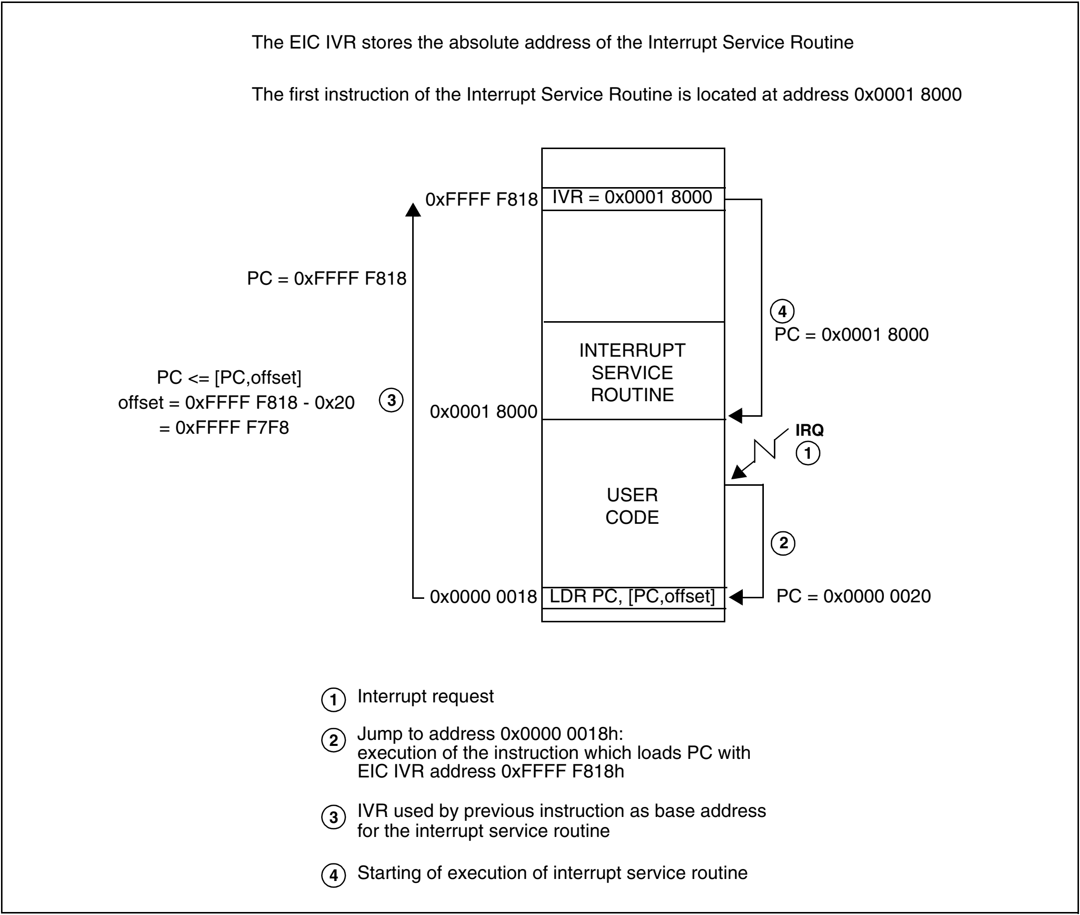
LDR PC, [PC, offset]

where "offset" is what to add to the PC to obtain the EIC IVR address. The instruction above allows the CPU to load the Program Counter register with the address kept in the EIC IVR and so to jump to the location pointed to by the EIC_IVR register.

The user has to consider that when the absolute address 0x0000 0018h is being fetched, the PC is equal to 0x0000 0020h (18h + 8h). So, the offset is equal to:

offset = IVR address – 0x0000 0020h = 0xFFFF F818h – 0x0000 0020h = 0xFFFF F7F8h

Figure 35. IRQ Interrupt Structure


The EIC IVR stores the absolute address of the Interrupt Service Routine
The first instruction of the Interrupt Service Routine is located at address 0x0001 8000
0xFFFF F818
IVR = 0x0001 8000
PC = 0xFFFF F818
4
PC = 0x0001 8000
INTERRUPT SERVICE ROUTINE
PC <= [PC,offset]
offset = 0xFFFF F818 - 0x20
= 0xFFFF F7F8
3
0x0001 8000
IRQ
1
USER CODE
2
0x0000 0018
LDR PC, [PC,offset]
PC = 0x0000 0020
1 Interrupt request
2 Jump to address 0x0000 0018h: execution of the instruction which loads PC with EIC IVR address 0xFFFF F818h
3 IVR used by previous instruction as base address for the interrupt service routine
4 Starting of execution of interrupt service routine

This mechanism allows a jump to virtually any location in the 4GB memory space. However, since the channel dependent portion of the IVR is the lower 16 bits, once the base address (in the upper part) has been fixed, the interrupt handler routines can only be within a 64Kbyte range from that base address.

In case interrupt service routines need to be available during FLASH content updating, RAM must be used to store the related code. In this case, the RAM remapping feature can be used: in this way, the address 0x0000 0018h becomes available in RAM memory space, and the above mechanism works similarly.

4.1.4 Priority Decoder

The priority decoder is a combinational block continuously calculating the pending channel with the highest IRQ priority. If there is a winner, it updates the IVR (Interrupt Vector Register) with the address of the IRQ interrupt routine that has won the arbitration, and asserts the nIRQ internal signal low. The nIRQ internal signal ORed with the inverted EIC IRQ enable bit (IRQ_EN) corresponds to the ARM7TDMI nIRQ signal.

Each channel has a 4-bit field, the SIPL (Source Interrupt Priority Level) in SIRn (Source Interrupt Register 0-31) defining the channel priority level in the range of 0 (lowest priority) to 15 (highest).

If several channels are assigned with the same priority level, an internal hardware daisy chain fixes the priority between them. The higher the channel address, the higher the priority. If channel 2 and channel 6 are assigned to the same software priority level, and if they are both pending at the same time, channel 6 will be served first.

In order to declare a channel as a winner, the channel must:

- Be pending (EIC_IPR - Interrupt Pending Register, 32 pending bits, one per channel). In order to be pending, a channel has to be enabled (EIC_IER - Interrupt Enable Register, 32 enable bits, one per channel). A pending bit will not be set if the corresponding enable bit is not set. If the enable bit is reset while the pending bit is still set, the channel will still be part of the priority decoder until the pending bit is cleared.
- Have the highest priority level, higher than the current one (EIC_CIPR - Current Interrupt Priority Register) and higher than any other pending interrupt channel.
- Have the highest position in the interrupt channel chain if there are multiple pending interrupt channels with the same priority level.

The EIC_CIPR provides the priority of the main program or the priority of the interrupt routine currently being served. At reset, the EIC_CIPR is at 0, it can be modified by software from 0 (lowest) to 15 (highest) for the main program. For an interrupt routine, it can be modified by software from the initialized priority value stored in the EIC_SIRn (Source Interrupt Register 0-31) up to 15. Attempting to write a lower value than the one in EIC_SIRn will have no effect.

For safe operation, it is recommended to disable the global IRQ before modifying the EIC_CIPR, SIR, or IPR registers, to avoid dangerous race conditions. Moreover, if IRQ_EN is cleared in an Interrupt Subroutine, the pending bit related to the IRQ currently being served must not be cleared, otherwise it will no longer be possible to recover the EIC status before popping the stack.

4.1.5 Finite State Machine

The Finite State Machine (FSM) has two states, READY and WAIT. The two states correspond to the ARM7TDMI nIRQ line state masked by the IRQ global enable bit (IRQ_EN). After reset, the FSM is in READY state (EIC nIRQ line is high). When the priority decoder elects a new winner, the FSM moves from READY to WAIT state and the EIC nIRQ line is asserted low.

To move the FSM back to the ready state, it is mandatory to read the IVR register (or to reset the entire device).

Reading the IVR always moves the FSM from WAIT to READY state, assuming that the FSM was in WAIT state, and automatically releases the EIC nIRQ line. IVR must not be read if it is not in order to acknowledge the interrupt, as it releases nIRQ line.

Reading the IVR has no effect on the state machine and nIRQ line if the device is in debug state.

There is no flag indicating the FSM state.

4.1.6 Stack

The stack is up to 15 events deep corresponding to the maximum number of nested interrupts. It is used to push and pop the previous EIC state. The data pushed onto the stack are EIC_CICR (Current Interrupt Channel Register) and EIC_CIPR (Current Interrupt Priority Register).

When the FSM is in WAIT state, reading the IVR raises an internal flag. This pushes the previous CICR and CIPR onto the EIC stack. This happens on the next internal clock cycle after reading the IVR. In the meantime the internal flag is cleared. The EIC_CICR and EIC_CIPR are updated with the value corresponding to the interrupt channel read in IVR.

If IVR is read while the FSM is in READY state, the internal flag is not raised and no operation is performed on the EIC internal stack. IVR must not be read if it is not in order to acknowledge the interrupt, otherwise EIC internal stack would be corrupted.

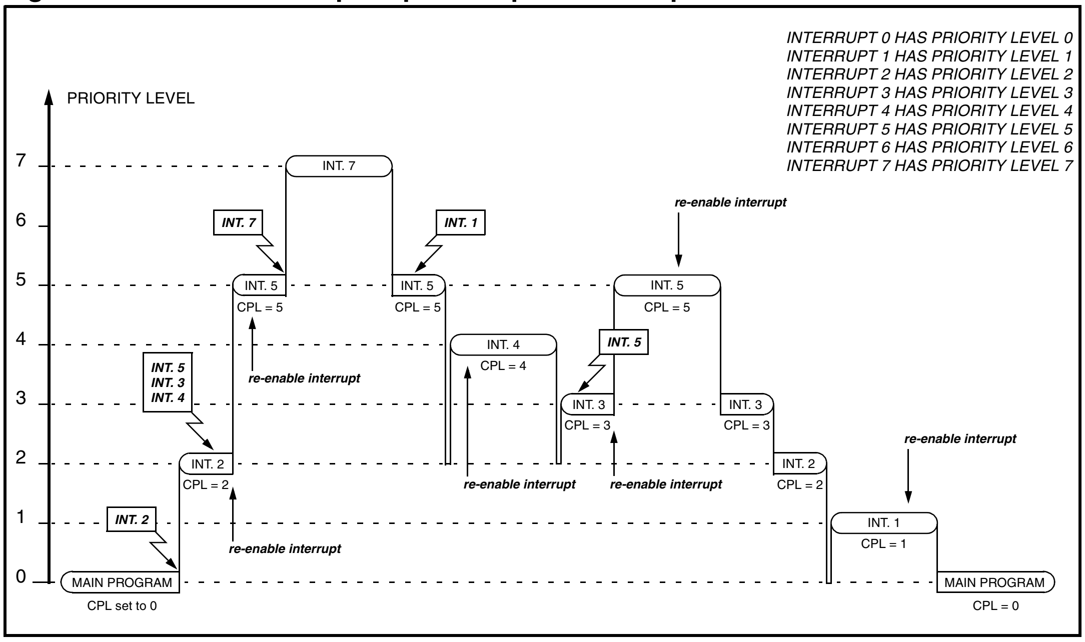
Reading the IVR has no effect on the EIC internal stack if the device is in debug state.

A routine can only be interrupted by an event having a higher priority. Consequently, the maximum number of nested interrupts is 15, corresponding to the 15 priority levels, from 1 to 15. An interrupt with priority level 0 can never be executed. In order for the stack to be full, up to 15 interrupts must be nested and each interrupt event must appear sequentially from priority level 1 up to 15. The main program must have priority level 0.

Having all interrupt sources with a priority level 0 could be useful in applications that only use polling.

To pop the stack, the EIC pending bit corresponding to the interrupt in the EIC_CICR register has to be cleared. Clearing any other pending bits will not pop the stack. Take care not to clear a pending bit corresponding to an event still in the stack, otherwise it will not be possible to pop the stack when reaching this stack stage. When the stack is popped, the EIC_CICR and EIC_CIPR are restored with the values corresponding to the previous interrupt event.

Figure 36. Nested interrupt request sequence example



4.1.7 EIC Interrupt Vectoring

When the ARM7TDMI decodes an IRQ interrupt request, the instruction at the address 0x18 is executed. By this time the IVR register is updated with the address of the highest pending interrupt routine. In order to get the maximum advantage from the EIC mechanism, the instruction at address 0x18 can load the program counter with the address located in the IVR. In this way, the CPU vector points directly to the right interrupt routine without any software overhead.

As the priority decoder is always active, the arbitration is never stopped. It may happen that an interrupt event asserts low the ARM7TDMI nIRQ line and if between the nIRQ line asserted low and the IVR read operation a new highest priority event appears, the IVR will have the value corresponding to the highest priority pending interrupt when the IVR is read.

It is not mandatory to read the IVR and to branch directly to the right interrupt routine with the instruction at address 0x18. An alternative solution could be to branch to a single interrupt entry point and to read the IVR register later on. The only mandatory operations are to first read the IVR once only, then to clear the corresponding pending bit. From an EIC stand point, the interrupt is acknowledged when the IVR is read and is completed when the corresponding pending bit is cleared. From an ARM7TDMI core stand point the interrupt is acknowledged when the ARM7TDMI nIRQ line is asserted low and is completed when the exception return sequence is executed.

The EIC nIRQ line is equivalent to the ARM7TMDI nIRQ line but in addition it is masked by the EIC global enable bit (IRQ_EN). The EIC nIRQ line can be asserted low but if the global EIC enable bit is not set the ARM7TDMI nIRQ line will not be asserted low.

The IPR is a read/clear register, so writing a ‘0’ has no effect, while writing a ‘1’ resets the related bit. Therefore, refrain from using read-modify instructions to avoid corruption of the IPR status. Most of the EIC pending bits are related to a peripheral pending bit. The peripheral pending bit must be cleared prior to clearing the EIC pending bit. Otherwise the EIC pending bit will be set again and the interrupt routine will be executed twice.

4.1.8 EIC IRQ notes

Reading the IVR while the FSM is in READY state will have no effect. The value read will be unpredictable. Actually, the EIC assigns the IVR value to one of the IRQ routine addresses by default.

The IVR can also be unpredictable while the FSM is in WAIT state. This has to be avoided as the CPU would execute one interrupt routine while the value in the IVR register is not relevant. It would result in an EIC stack corruption as the corresponding pending bit could be reset.

There are several cases where it may happen:

- When the main program raises the main program priority level to a value equal to or higher than the current highest priority pending channel. Interrupts have to be disabled globally during this operation.
- When lowering the priority of a pending channel priority by modifying the SIRn register to a value equal or lower than the main program priority.
- When the software clears some pending bits without taking care to execute the standard interrupt routine sequence.

In such cases the priority decoder loses the winner while the EIC nIRQ line is still being asserted. Only reading the IVR will release the EIC nIRQ line.

The CPU will execute the interrupt routine without having a relevant value in the IVR register, possibly corrupting the stack. If the corresponding pending bit is reset, it will not be possible to execute the EIC stack pop operation.

The normal way to process an interrupt event is to read the IVR register only one time from the interrupt routine, otherwise it may corrupt the EIC stack. Before exiting the interrupt routine the corresponding peripheral and the EIC pending bits must be cleared. As soon as the IVR is read the application software can read the CICR register to know which interrupt routine is currently executed, as long as the IVR register is not used as a routine pointer.

If the IVR is not read in the interrupt routine the nIRQ line will not be released and the interrupt will be executed twice. If the pending bit is already cleared the EIC stack will be corrupted as it will not be able to perform the pop operation.

Inside a routine it is not an issue to clear pending bits having a lower priority level than the current one, as the nIRQ line is not asserted low in that case. It can be an issue to clear a pending bit having a higher priority level as the EIC nIRQ line is already asserted low, and when interrupts are re-enabled, the EIC stack will be corrupted.

Clearing a pending bit of an interrupt already in the stack will corrupt the stack.

In the main program, if the global interrupts are disabled, all interrupt sources are disabled and all pending bits are cleared, if the nIRQ was already asserted low as soon as the global interrupts are enabled the CPU executes an interrupt routine. In this situation the IVR read will have an unpredictable value, corrupting the EIC stack.

There are two safe ways to clear pending bits without executing the interrupt routine. The first one is to clear them from an IRQ routine having the higher priority level. By this way, the EIC nIRQ line is guaranteed to be released. The second one applies only from the main routine when there is no nested interrupt (EIC stack empty). It consists in following these steps:

1. The global interrupts are disabled
2. Disable all the interrupt channel bits (IER = 0)
3. IVR is read
4. Read CICR
5. Clear all IPR pending bits (IPR=0), it depends on CICR read. The software has to clear first the register not related to CICR read.
6. Clear the second EIC pending bit register.
7. Enable IER according to the user application.
8. Enable global interrupts.

If the FSM is in READY state, reading IVR in operation 3) will not do anything. Clearing pending bits will not pop anything as this code is executed from the main routine with no nested interrupts (this is the reason why this method is not recommended from an interrupt routine as clearing all the pending bits will abort the current EIC interrupt routine).

If the FSM is in WAIT state the IVR read will correspond to the highest priority level channel, the main CIPR register will be pushed into the stack, and the main program will have the priority of the highest pending interrupt. The EIC nIRQ is released and will not be asserted again if the right IPR register is cleared first.

All EIC pending bits can be cleared including the ones that the user application wants to address later on. The user code needs to make sure that, for those interrupts, the peripheral pending bit is not cleared. By this way, the corresponding EIC pending bits will be set again. As all EIC pending bits are cleared, the EIC stack is guaranteed to pop properly. An alternative solution is to make sure that the EIC pending bit corresponding to the IVR read is cleared.

The operation 3), reading the IVR, is done from the main program and not from an interrupt. It is the only exception where IVR should be read outside an interrupt routine.

4.1.9 FIQ mechanism

Compare to the EIC IRQ mechanism the EIC FIQ mechanism does not provide any automatic vectoring and software priority level to each FIQ interrupt source. It provides a global FIQ enable bit, an enable and a pending bit per FIQ channel.

There are few differences between the global F bit from the CPSR ARM core register and the global FIQ enable bit from the EIC. The F bit can not be modified in user mode while the EIC global FIQ enable bit is always accessible in any modes. In addition the F bit does not modify the nFIQ internal signal level, it just masks the signal to the core while the EIC global FIQ enable bit act on the nFIQ signal level. The nFIQ signal is always inactive as soon as the global EIC FIQ enable bit is reset, and is active as soon as the global EIC FIQ enable bit is set along with at least one FIQ pending bit.

In order for a FIQ channel source to be pending, the corresponding FIQ enable bit must be set. Clearing the FIQ channel enable bit while the corresponding pending bit is set will not clear the pending bit and the channel will stay active until the pending bit is cleared.

In order for the CPU to vector at the address 0x1C (FIQ exception vector address) the F bit and the global EIC FIQ enable bit must be enabled and at least one FIQ pending bit must be active. Otherwise the CPU will not enter the FIQ exception routine.

4.2 Register Programming

Here are a few guidelines on programming the EIC registers in order to get the controller up and running quickly. The following explanation is based on an example using standard (IRQ) interrupts to detect an interrupt on channel #12, which is to be assigned a priority level of 5.

4.2.1 IRQ Example

1. First of all, assign the priority of interrupt channel #12 and the jump address to the related Interrupt Service Routine. To do this:
   – Write the binary value “0101” to the SIPL[3:0] bits in the EIC_SIR12 register , i.e. priority level 5. It must be nonzero to be able to generate an IRQ.
2. Next program the registers involved with supplying the channel interrupt vector to the EIC controller (IVR[31:16] and SIR12[31:16]):
   – Write the memory address offset (or the jump offset) of the start of the Interrupt Service Routine for channel #12 in the SIR[31:16] bits in the EIC_SIR12 register.
   – Write the base jump address (or the jump opcode) in the IVR[31:16] bits in the EIC_IVR register.
3. Finally, enable the response to the interrupt both at the global level and at the single interrupt channel level. To do this:
   – Set the IRQ_EN bit in the EIC_ICR register to 1
   – Set bit # 12 in the EIC_IER register to 1

4.2.2 FIQ Example

FIQ interrupts do not have any vectorization or priority, so only the two part of step 3 above are required. For example, to enable FIQ channel #1:

1. Set the FIQ_EN bit of ICR to 1.
2. Set the FIE1 bit in the EIC_FIR register to 1.

4.3 Application note

Every Interrupt Service Routine (ISR) allowing nested interrupts must be composed of the following blocks of code:

A **Header routine** for entering the ISR. It must be:

```
1) STMFD sp!,{r5,lr}The workspace r5 plus the current return address lr_irq is
pushed into the system stack.

2) MRS r5,spsr       Save the spsr into r5

3) STMFD sp!,{r5}    Save r5

4) MSR cpsr_c,#0x1F  Re-enable IRQ, go into system mode

5) STMFD sp!,{lr}    Save lr_sys for the system mode
```

r5 is a generic register, and could be any available register from r0 to r12. Since there is no way to save SPSR directly into the ARM's stack, the operation is executed in two steps using r5 as a work register.

The ISR **Body routines**.

A **Footer routine** for exiting the ISR. It must be:

```
1) LDMFD sp!,{lr}    Restore lr_sys for the system mode

2) MSR cpsr_c,#0xD2  Disable IRQ, move to IRQ mode

3) Clear pending bit in EIC (using the proper IPRx)

4) LDMFD sp!,{r5}    Restore r5

5) MSR spsr,r5       Restore Status register spsr

6) LDMFD sp!,{r5,lr} Restore status lr_irq and workspace

7) SUBS pc,lr,#4     Return from IRQ and interrupt re-enabled
```

The following two paragraphs provide comments on the code lines shown above and give useful hints on developing subroutines to be invoked from within an ISR.

4.3.1 Avoiding LR_sys and r5 registers content loss

This case refers to a LR_sys content loss problem. Let's assume that an ISR without instruction 5) in the header routine (and consequently without instruction 1) in the footer routine) has just started; the following happens:

- Instruction 4) is executed (so system mode is entered)
- If before the interrupt event the main program was in a leaf routine (no function call in the routine) meaning that the system link register is not pushed into the stack, as soon as system mode is entered in the ISR, the link register could be corrupted if a function call is made in the ISR. It will not be possible to return from the leaf routine in the main program.

To avoid such a dangerous situation, the workaround is to insert line 5) at the end of header routine and consequently line 1) at the beginning of footer routine.

Similar reasons could lead register **r5** to be corrupted. To fix this problem, lines 3) in header and 4) in footer should be added.

4.3.2 Hints about subroutines used inside ISRs

This section discusses the case where a subroutine is called by an ISR.

Supposing that a procedure starts with an instruction like:

```
STMFD    SP!, {..., LR}
```

probably it will end with:

```
LDMFD    SP!, {..., LR}
MOV      PC, LR
```

If a higher priority IRQ occurs between the last two instructions, and the new ISR calls in turn another subroutine, the LR content will be lost: so, when the last IRQ ends, the previously interrupted subroutine will not return to the correct address.

To avoid this, the previous two instructions must be replaced with the single instruction:

```
LDMFD    SP!, {..., PC}
```

which automatically moves the stored link register directly into the program counter, making the subroutine return correctly.

4.4 Interrupt latency

As soon as an interrupt request is generated, the request itself must go through three different stages before the interrupt handler routine can start. The interrupt latency can be seen as the sum of three different factors:

- Latency due to the synchronization of the input stage. This logic can be present (e.g. synchronization stage on external interrupt input lines) or not (e.g. on-chip interrupt request), depending on the interrupt source. Either zero or two clock cycles delay are related to this stage.
- Latency due to the EIC itself. Two clock cycles are related to this stage.
- Latency due to the ARM7TDMI interrupt handling logic (refer to the documentation available on www.arm.com).

Table 22. EIC Interrupt latency (in clock cycles)

	SYNCH. STAGE		EIC STAGE
	min	max	
FIQ	0	2	2
IRQ			

4.5 Register description

In this section, the following abbreviations are used:

Table 23. Abbreviations

read/write (rw)	Software can read and write to these bits.
read-only (r)	Software can only read these bits.
read/clear (rc_w1)	Software can read as well as clear this bit by writing 1. Writing 0 has no effect.

4.6 Register description

Reading from a “Reserved” field in any register will return ‘0’ as the result. Attempting to write in a “Reserved” field has no effect.

4.6.1 Interrupt Control Register (EIC_ICR)

Address Offset: 00h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														FIQ_EN	IRQ_EN
-														rw	rw

Bits 31:2	Reserved, must be kept at reset value (0).
Bit 1	**FIQ_EN:** *Global FIQ output Enable bit* This bit enables FIQ output request to the CPU. 0: Enhanced Interrupt Controller FIQ output request to CPU is disabled, even if the EIC logic has detected valid and enabled fast interrupt requests at its inputs 1: Enhanced Interrupt Controller FIQ output request to CPU is enabled
Bit 0	**IRQ_EN:** *Global IRQ output Enable bit* This bit enables IRQ output request to the CPU. 0: Enhanced Interrupt Controller IRQ output request to CPU is disabled, even if the EIC logic has detected valid and enabled interrupt requests at its inputs 1: Enhanced Interrupt Controller IRQ output request to CPU is enabled

The ICR register is a global enable register.

Note: *While resetting to ‘0’ FIQ_EN bit simply masks out the FIQ output request to the CPU. Resetting to ‘0’ IRQ_EN bit keeps the FSM which controls the IRQ output request to the*

CPU under reset and freezes the FSM that manages the EIC Stack Pointer (Push/Pop control).

This makes it safe to perform operations such as clearing unwanted pending IRQs, raising the priority value in the CIPR register, or modifying the priority of an IRQ channel by re-writing the SIPL field in the related SIR register.

4.6.2 Current Interrupt Channel Register (EIC_CICR)

Address Offset: 04h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved											CIC[4:0]				
-											r	r	r	r	r

Bits 31:5	Reserved, must be kept at reset value (0).
Bits 4:0	**CIC[4:0]:** *Current Interrupt Channel.* Number of the interrupt whose service routine is currently in **execution** phase.

The CICR Register reports the channel ID of the interrupt channel currently serviced. There are 32 possible channel IDs (0 to 31), so only five register bits are significant (4 to 0).

After reset, the CICR value is set to ‘0’ and is updated by the EIC logic only after the processor has started servicing a valid IRQ interrupt request e.g. one clock cycle after having read IVR.

To make this happen, some EIC registers must be set as follows:

- ICR IRQ_EN =1 (to have the nIRQ signal to ARM7TDMI active)
- IER not all 0 (at least one interrupt channel must be enabled)
- Out of all the the interrupt channels enabled in the IER register, at least one must have the SIPL field, of the related SIR register, not set to 0, because the EIC generates a processor interrupt request (asserting the nIRQ line) **ONLY IF** it detects an enabled interrupt request whose **priority value is bigger than the CIPR (Current Interrupt Priority Register) value.**

When the nIRQ signal to the ARM7TDMI is activated, the CPU will read the EIC IVR (Interrupt Vector Register), and this read operation will advise the EIC logic that the ISR (Interrupt Service Routine) has been initiated and the CICR register can be properly updated.

The CICR value can’t be modified by software (read only register).

4.6.3 Current Interrupt Priority Register (EIC_CIPR)

Address Offset: 08h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												CIP[3:0]			
-												rw	rw	rw	rw

Bits 31:4	Reserved, must be kept at reset value (0).
Bits 3:0	**CIP[3:0]:** *Current Interrupt Priority.* Priority value of the interrupt whose service routine is currently in **execution** phase.

The CIPR Register provides the priority value of the main program or the priority value of the interrupt routine currently serviced. There are 16 possible priority values (0 to 15), so only four register bits are significant (3 to 0).

After reset, the CIPR value is set to '0' and is updated by the EIC logic only after the processor has started servicing a valid IRQ interrupt request e.g. one clock cycle after having read IVR.

To make this happen, some EIC registers must be set as follows:

- ICR IRQ_EN = 1 (to have the nIRQ signal to ARM7TDMI active)
- IER not all 0 (at least one interrupt channel must be enabled)
- Out of all the interrupt channels enabled by the IER register, at least one must have the SIPL field, of the related SIR register, not set to 0, because the EIC generates a processor interrupt request (asserting the nIRQ line) **ONLY IF** it detects an enabled interrupt request whose **priority value is bigger than the CIPR (Current Interrupt Priority Register) value.**

When the nIRQ signal to ARM7TDMI is activated, the CPU will read the EIC IVR (Interrupt Vector Register), and this read operation will advise the EIC logic that the ISR (Interrupt Service Routine) has been initiated and the CIPR register can be properly updated.

The CIPR value can be modified by the user code, to either raise the priority of the main code (in order to avoid serving incoming interrupt requests with a priority value lower than a desired one) or to promote a running ISR to a higher level: in this last case, the EIC logic will allow a write to the CIP field of any value higher, or equal, to the priority value associated with the interrupt channel currently serviced.

For example: suppose the nIRQ signal to ARM7TDMI is activated because of an enabled interrupt request on channel #4, whose priority value is 7 (i.e. SIPL of SIR7 is 7); after the processor reads the IVR register, the EIC will load the CIP field with 7. Until the interrupt service procedure is completed, writes of values 7 up to 15 will be allowed, while attempts to modify the CIP content with a priority lower than 7 will have no effect.

In the user code, care should be taken to avoid a situation where the FSM is in WAIT state and the IVR has an unpredictable value.

Note: In order to safely raise the current CIPR (both from main code and from an ISR), the IRQ_EN bit in the ICR register should be cleared first, otherwise the EIC FSM could be put into an unrecoverable state.

4.6.4 Fast Interrupt Enable Register (EIC_FIER)

Address Offset: 10h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														FIE[1:0]	
														rw	rw

Bits 31:2	Reserved, must be kept at reset value (0).
Bits 1:0	**FIE[1:0]:** *FIQ Channel 1:0 Fast Interrupt Enable bit* These bit s are set an cleared by software. In order for the controller to respond to a request on the specified channel (1 or 0), the corresponding bit in the FIER register must be set to 1. A ‘0’ value prevents the corresponding pending bit being set. 0: Fast Interrupt request issued on the specified channel is disabled 1: Fast Interrupt request issued on the specified channel is enabled **Note:** The FIER Register is just an alias of the FIE[1:0] bits in the FIR register.

4.6.5 Fast Interrupt Pending Register (EIC_FIPR)

Address Offset: 14h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														FIP[1:0]	
														rc_w1	rc_w1

Bits 31:2	Reserved, must be kept at reset value (0).
Bits 1:0	**FIP[1:0]:** *FIQ Channel 1:0 Fast Interrupt Pending bit* These bits are set by hardware by a Fast interrupt request on the corresponding channel (1 or 0). These bits are cleared only by software, i.e. writing a '0' has no effect, whereas writing a '1' clears the bit (forces it to '0') 0: No Fast Interrupt pending on the specified channel 1: Fast Interrupt pending on the specified channel **Note:** The FIPR Register is just an alias of the FIP[1:0] bits in the FIR register.

4.6.6 Interrupt Vector Register (EIC_IVR)

Address Offset: 18h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IVR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IVR[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

The IVR is the EIC register the CPU has to read after detecting the nIRQ signal assertion.

The IVR read operation tells the EIC logic that the interrupt service routine (ISR), related to the pending request, has been initiated. The IVR read acknowledges the IRQ, and so this should not be done for any other reason as it could cause an unwanted IRQ acknowledge.

This means that:

- the nIRQ signal can be de-asserted
- the CIPR and CICR can be updated
- no interrupt requests, whose priority is lower, or equal, than the current one can be processed

Bits 31:16(1)	**IVR[31:16]**: *Interrupt Vector (High portion).* This part of the vector does not depend on the interrupt to be serviced, but has to be programmed by the user at initialization time (see Note). It is common to all the interrupt channels. These bits are Read/Write.
Bits 15:0(1)	**IVR[15:0]**: *Interrupt Vector (Low portion).* This part of the vector depends on the interrupt to be serviced (i.e. out of all the enabled interrupts, the one with the highest priority). It is a copy of the SIV (Source Interrupt Vector) value of the SIR (Source Interrupt Register) related to the channel to be serviced. These bits are Read only.

1. The EIC logic does not care about the IVR content: from the controller point of view it's a simple concatenation of two 16-bit fields

   IVR = IVR(31:16) & SIRn(31:16)

   What has to be written in the IVR(31:16) is the higher part of the address pointing to the memory location where the interrupt service routines begin; the single SIRn(31:16) will have to contain the lower 16 bits (offset) of the memory address related to the channel specific ISR.

Reading IVR acknowledges the IRQ only when the CPU is not in debug mode.

4.6.7 Fast Interrupt Register (EIC_FIR)

Address Offset: 1Ch
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												FIP[1:0]		FIE[1:0]	
-												rc_w1	rc_w1	rw	rw

Bits 31:4	Reserved, must be kept at reset value (0).
Bits 3:2	**FIP[1:0]:** *FIQ Channel 1:0 Fast Interrupt Pending bit* These bits are set by hardware by a Fast interrupt request on the corresponding channel (1 or 0). These bits are cleared only by software, i.e. writing a '0' has no effect, whereas writing a '1' clears the bit (forces it to '0') 0: No Fast Interrupt pending on the specified channel 1: Fast Interrupt pending on the specified channel **Note:** The FIPR Register is just an alias of the FIP field in the FIR register.
Bits 1:0	**FIE[1:0]:** *FIQ Channel 1 Fast Interrupt Enable bit* These bits are set and cleared by software. In order for the controller to respond to a request on a specific channel, the corresponding bit in the FIER register must be set to 1. A '0' value prevents the corresponding pending bit being set. 0: Fast Interrupt request issued on the specified channel is disabled 1: Fast Interrupt request issued on the specified channel is enabled **Note:** The FIER Register is just an alias of field FIE in FIR register.

4.6.8 Interrupt Enable Register (EIC_IER)

Address Offset: 20h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IER[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
IER[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**IER[31:0]:** *Channel 31:0 Interrupt Enable bits* These bits are set and cleared by software. To enable the interrupt response to the specified interrupt input channel the corresponding bit in the IER register must be set to ‘1’. A ‘0’ value prevents the corresponding pending bit being set. 0: Input channel disabled 1: Input channel enabled

4.6.9 Interrupt Pending Register (EIC_IPR)

Address Offset: 40h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IPR[31:16]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
IPR[15:0]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:0	**IPR[31:0]:** *Channel 31:0 Interrupt Pending bits* These bits are set by hardware and cleared by software writing 1. They provide the information about the interrupt status of the corresponding channel. If the corresponding bit in the enable register IER has been set, the IPR bit set high (active) means that the related channel has asserted an interrupt request, that has not been serviced yet. These bits are Read/Clear, i.e. writing a ‘0’ has no effect, whereas writing a ‘1’ clears the bit (forces it to ‘0’). 0: No interrupt pending 1: Interrupt pending

Note: 1 *Before exiting the ISR (Interrupt Service Routine), the software must be sure to have cleared the EIC IPR bit related to the executed routine: this bit clear will be read by the EIC logic as the End of Interrupt (EOI) sequence, allowing the interrupt stack pop and new interrupt processing.*

2 *The Interrupt Pending bits must be handled with care because the EIC State Machine, and its internal priority hardware stack, could be forced to an unrecoverable condition if unexpected pending bit clear operations are performed.*

Example 1
Suppose that one or more interrupt channels are enabled, with a priority higher than zero; as soon as an interrupt request occurs, the EIC FSM processes the new input and asserts the nIRQ signal. If, before reading the EIC_IVR (0x18) register, the software clears the pending bits, for any reason, the nIRQ signal will remain asserted, even if no more interrupts are pending.
The only way to reset the nIRQ line logic is to read the EIC_IVR register, or to reset the IRQ_EN bit in the EIC_ICR register.

Example 2

Suppose that one or more interrupt channels are enabled, with a priority higher than zero; as soon as an interrupt request occurs, the EIC FSM processes the new input and asserts the nIRQ signal. If, after reading the EIC_IVR (0x18) register, the software clears the pending bit related to the serviced channel, for any reason, before completing the Interrupt Service Routine, the EIC logic will detect an End Of Interrupt command, will send a pop request to the priority stack and a new interrupt, even of lower priority, will be processed.
To close an interrupt handling section (EOI), the interrupt pending clear operation must be performed, at the end of the related Interrupt Service Routine, on the pending bit related to the serviced channel; on the other hand, as soon as the pending bit of the serviced channel is cleared (even by mistake) by the software, the EOI sequence is entered by the EIC logic.

3 *To safely clear the pending bit of an IRQ that is not currently being serviced, the IRQ_EN bit in the EIC_ICR register should be cleared first. If this is not done, the EIC FSMs could go into an unrecoverable state.*
In any case, when performed in the main program this operation has no drawbacks. When performed inside an IRQ routine it is very important not to mistakenly clear the IPRn bit related to the currently served IRQ . Because the IRQ_EN bit freezes the Stack, the pop operation for the current IRQ will not be done and won't even be possible later on when the IRQs are re-enabled.

4.6.10 Source Interrupt Registers - Channel n (EIC_SIRn)

Address Offset: 60h to 15Ch
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SIV[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												SIPL[3:0]			
-												rw	rw	rw	rw

There are 32 different SIRn registers, one for each interrupt channel.

Bits 31:16	**SIV[31:16]:** *Source Interrupt Vector for interrupt channel n (n = 0... 31).* These bits are written by software to define the interrupt channel dependent part of the interrupt vector, that is provided to the processor when the Interrupt Vector Register (Address 0x18h) is read. The SIV has to be loaded with the lower 16 bits (offset) of the memory address of the first ISR instruction.
Bits 15:4	Reserved, must be kept at reset value (0).
Bits 3:0	**SIPL[3:0]:** *Source Interrupt Priority Level for interrupt channel n (n = 0... 31).* These bits are written by software to define the interrupt channel priority level between 0 and 15 (15 = highest priority, 1= lowest priority, 0 = never served). The reset value is 0. **Note:** To be processed by the EIC priority logic, an interrupt channel must have a priority level higher than the current interrupt priority (CIP); the lowest value CIP can have is 0, so all the interrupt sources that have a priority level equal to 0 will never generate an IRQ request, even if properly enabled.

4.7 EIC Register map

Table 24. EIC register map

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0h	ICR	reserved																														FIQ_EN	IRQ_EN
04h	CICR	reserved																											CIC[4:0]				
08h	CIPR	reserved																												CIP[3:0]			
0Ch	reserved																																
10h	FIER	reserved																														FIE[1:0]	
14h	FIPR	reserved																														FIP[1:0]	
18h	IVR	Jump Instruction Opcode or Jump Base Address																Jump Offset															
1Ch	FIR	reserved																												FIP[1:0]		FIE[1:0]	
20h	IER	IER[31:0]																															
40h	IPR	IPR[31:0]																															
60h	SIR0	SIV0[31:16]																reserved												SIPL0[3:0]			
64h	SIR1	SIV1[31:16]																reserved												SIPL1[3:0]			
68h	SIR2	SIV2[31:16]																reserved												SIPL2[3:0]			
6Ch	SIR3	SIV3[31:16]																reserved												SIPL3[3:0]			
70h	SIR4	SIV4[31:16]																reserved												SIPL4[3:0]			
74h	SIR5	SIV5[31:16]																reserved												SIPL5[3:0]			
78h	SIR6	SIV6[31:16]																reserved												SIPL6[3:0]			
7Ch	SIR7	SIV7[31:16]																reserved												SIPL7[3:0]			
80h	SIR8	SIV8[31:16]																reserved												SIPL8[3:0]			
84h	SIR9	SIV9[31:16]																reserved												SIPL9[3:0]			
88h	SIR10	SIV10[31:16]																reserved												SIPL10[3:0]			
8Ch	SIR11	SIV11[31:16]																reserved												SIPL11[3:0]			
90h	SIR12	SIV12[31:16]																reserved												SIPL12[3:0]			
94h	SIR13	SIV13[31:16]																reserved												SIPL13[3:0]			
98h	SIR14	SIV14[31:16]																reserved												SIPL14[3:0]			
9Ch	SIR15	SIV15[31:16]																reserved												SIPL15[3:0]			
A0h	SIR16	SIV16[31:16]																reserved												SIPL16[3:0]			
A4h	SIR17	SIV17[31:16]																reserved												SIPL17[3:0]			
A8h	SIR18	SIV18[31:16]																reserved												SIPL18[3:0]			
ACh	SIR19	SIV19[31:16]																reserved												SIPL19[3:0]			
B0h	SIR20	SIV20[31:16]																reserved												SIPL20[3:0]			
B4h	SIR21	SIV21[31:16]																reserved												SIPL21[3:0]			
B8h	SIR22	SIV22[31:16]																reserved												SIPL22[3:0]			
BCh	SIR23	SIV23[31:16]																reserved												SIPL23[3:0]			
C0h	SIR24	SIV24[31:16]																reserved												SIPL24[3:0]			
C4h	SIR25	SIV25[31:16]																reserved												SIPL25[3:0]			
C8h	SIR26	SIV26[31:16]																reserved												SIPL26[3:0]			

Table 24. EIC register map (continued)

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCh	SIR27	SIV27[31:16]																reserved												SIPL27[3:0]			
D0h	SIR28	SIV28[31:16]																reserved												SIPL28[3:0]			
D4h	SIR29	SIV29[31:16]																reserved												SIPL29[3:0]			
D8h	SIR30	SIV30[31:16]																reserved												SIPL30[3:0]			
DCh	SIR31	SIV31[31:16]																reserved												SIPL31[3:0]			

See *Table 2* for the base address.

4.8 External Interrupt Controller (EXTIT)

4.8.1 Introduction

The External interrupt controller consists of up to 16 edge detectors for generating interrupt requests. Each interrupt line can be independently configured to select the trigger event (rising or falling) and can be masked independently. A pending register maintains the status of the interrupt requests.

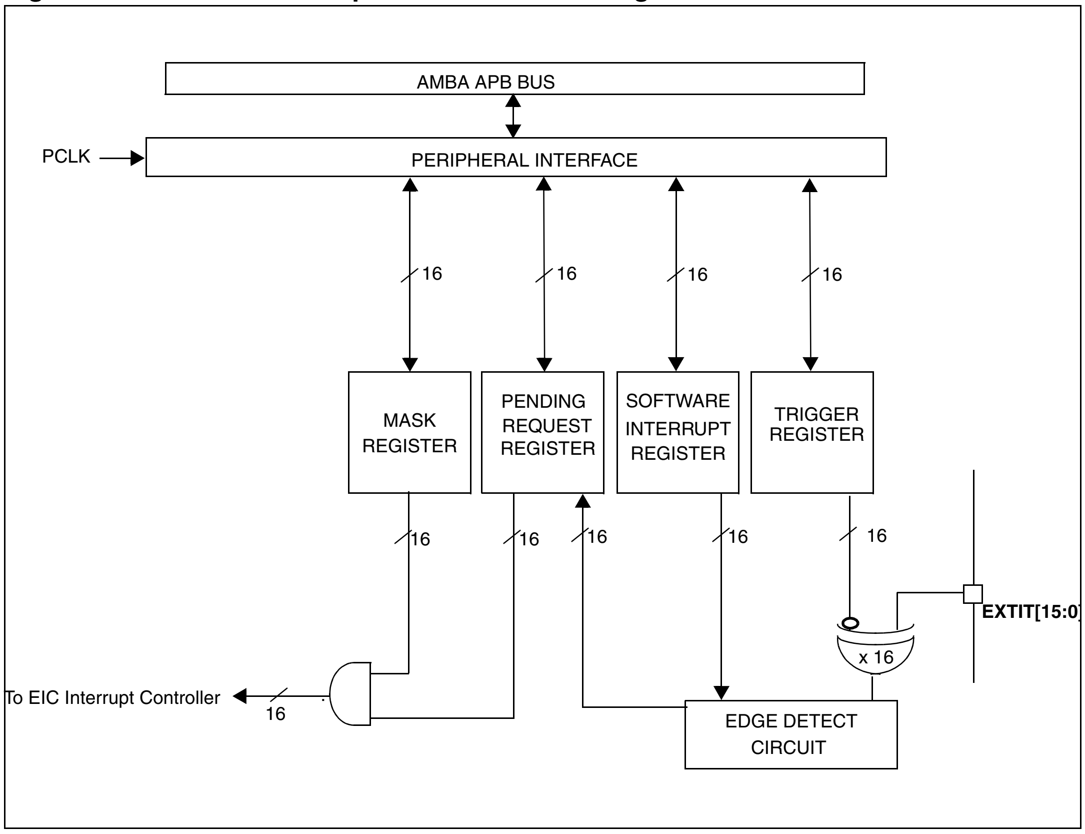
4.8.2 Main features

- Supports generation of up to 16 Interrupt requests
- Independent trigger and mask on each interrupt line
- Dedicated status bit for each interrupt line
- Generation of up to 16 software interrupt requests

4.8.3 Block diagram

The Block Diagram is shown in *Figure 37*.

Figure 37. External Interrupt Controller block diagram



4.8.4 Functional description

To generate the interrupt, the interrupt line should be configured and enabled. This is done by programming the trigger register with the desired polarity and enabling the interrupt request by writing a ‘1’ to the corresponding bit in the mask register. When the selected edge occurs on the external interrupt line, an interrupt request is generated. The pending bit corresponding to the interrupt line is also set. This request is reset by writing a ‘1’ in the pending register.

An interrupt request can also be generated by software by writing a ‘1’ in the software interrupt register.

Hardware Interrupt selection

To configure the 16 lines as interrupt sources, use the following procedure:

- Configure the mask bits of the 16 Interrupt lines (EXTIT_MR)
- Configure the Trigger Selection bits of the Interrupt lines (EXTIT_TSR)
- Configure the enable and mask bits that control the EIC IRQ channel mapped to the External Interrupt Controller (EXTIT) so that an interrupt coming from one of the 16 lines can be correctly acknowledged.

Software Interrupt selection

The 16 lines can be configured as software interrupt lines. The following is the procedure to generate a software interrupt.

- Configure the mask bits of the 16 Interrupt lines (EXTIT_MR)
- Set the required bit of the software interrupt register (EXTIT_SWIR)

Restrictions applying to external interrupt signal inputs

To generate an external interrupt, a minimum pulse width of 2 x t_{PCLK} has to be applied on the external line. This restriction applies in RUN mode, SLOW mode or WFI mode.

However, in STOP mode, this restriction does not apply and the external interrupts can wake-up the MCU while there is no clock running.

In STANDBY mode, only the RTC alarm or the WKP_STDBY pin or the RESET can wake-up the MCU.

Note: In modes other than STOP mode or STANDBY mode, care must be taken if the external line is the RTC alarm: the restriction still applies and the pulse generated by the RTC alarm must be larger than 2 t_{PCLK} clocks : Refer to theFigure 45: RTC second and alarm waveform example with PR=0003, ALARM=00004 on page 170, which explains how the RTC alarm signal is generated.

4.8.5 External interrupt line mapping

Table 25. External interrupt line mapping

EXTIT line#	EXTIT line source
0	P0.02 (TIM2_OC1)
1	P0.05 (SSP0_SCLK)
2	P0.07 (SSP0_MOSI)
3	P0.08 (I2C_SCL)
4	P0.10 (UART0_RX)
5	P0.14 (CAN_RX)
6	P0.19 (SSP1_NSS)
7	P1.05
8	P1.07
9	P1.09
10	P1.10 (PWM_EMERGENCY)
11	P1.11
12	P1.12
13	P1.13
14(1)	USB Wake-up from SUSPEND
15(2) (3)	P1.15 (WKP_STDBY PIN) or RTC_ALARM

1. External interrupt line #14 is internally connected to the USB Wake-up from SUSPEND event. This can be used to wake-up from STOP mode (if the USB has been put in Suspend mode) on any USB activity. For this you need to:
   - configure external interrupt line #14 to be sensitive to rising edge
   - configure the USB
   - put the USB in Suspend mode and enter STOP mode
2. External interrupt line #15 is connected to the output of an OR gate between the P1.15 input pin and the RTC alarm event.
   This RTC connection can be used to wake-up from STOP mode on an RTC alarm event. To do this, you must:
   - Apply a low level on the P1.15 pin
   - Configure the external interrupt line #15 to be sensitive to rising edge.
   - Configure the RTC to generate the RTC alarm
3. To wake-up from STANDBY mode, there is no need to configure external interrupt 15.

4.8.6 Register description

Every register can have only 32-bit access. A byte or half-word cannot be read or written. In this section, the following abbreviations are used:

read/write (rw)	The software can read and write to these bits.
read-only (r)	The software can only read these bits.
read/clear (rc_w1)	The software can read as well as clear this bit by writing 1. Writing '0' has no effect on the bit value.

Interrupt Mask Register (EXTIT_MR)

Address Offset: 00h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, must be kept at reset value (0).
Bits 15:0	**MRx:** *Interrupt Mask on line x* 0 : Interrupt request from Line x is masked 1 : Interrupt request from Line x is not masked

Trigger Selection Register (EXTIT_TSR)

Address Offset: 04h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, must be kept at reset value (0).
Bits 15:0	**TRx :** *Trigger event configuration bit of line x* 0 : Interrupt request configured on falling edge of the input line 1 : Interrupt request configured on rising edge of the input line

Note: The external wake-up lines are edge triggered, no glitches must be generated on these lines.

If either a rising or a falling edge on external interrupt line occurs during writing of EXTIT_TR register, the pending bit will not be set

Software Interrupt Register (EXTIT_SWIR)

Address Offset: 08h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWIR15	SWIR14	SWIR13	SWIR12	SWIR11	SWIR10	SWIR9	SWIR8	SWIR7	SWIR6	SWIR5	SWIR4	SWIR3	SWIR2	SWIR1	SWIR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, must be kept at reset value (0).
Bits 15:0	**SWIx:** *Software Interrupt on line x* Writing a 1 to this bit when it is at 0 sets the corresponding pending bit in EXTIT_PR. If the interrupt is enabled on this line on the EXTIT_MR, an interrupt request is generated. This bit is cleared by clearing the corresponding bit of EXTIT_PR (by writing a 1 into the bit).

Pending Register (EXTIT_PR)

Address Offset: 0Ch
Reset value: xxxx xxxxh

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:16	Reserved, must be kept at reset value (0).
Bits 15:0	**PRx:** *Pending bit* 0: No trigger event occurred 1: selected trigger event occurred This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a 1 into the bit or by changing the sensitivity of the edge detector. **Note**: If an interrupt event occurs one cycle before entering Stop mode, then the EXTIT_PR register will be updated only after exit from Stop mode, generating an interrupt request if the corresponding bit in the EXTIT_MR register is set

4.8.7 EXTIT Register map

Table 26. External Interrupt Controller Register Map

Addr Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	EXTIT_MR	Interrupt Mask Register																															
04h	EXTIT_TSR	Trigger Selection Register																															
08h	EXTIT_SWIR	Software Interrupt Register																															
0Ch	EXTIT_PR	Pending Register																															

Refer to *Table 2* for the register base address.

5 DMA controller (DMA)

5.1 Introduction

The DMA controller provides access to 4 data streams. Since peripherals are memory mapped, data transfers from/to peripherals are managed like memory/memory data transfers.

Each data stream is associated with a particular peripheral (see *Figure 38*). The peripheral, triggers a DMA request, starting the data transfer between the corresponding buffers defined in the stream descriptor. Data stream 3 can alternatively be used as a memory/memory data transfer triggered by a software DMA request, independently from any peripheral activity.

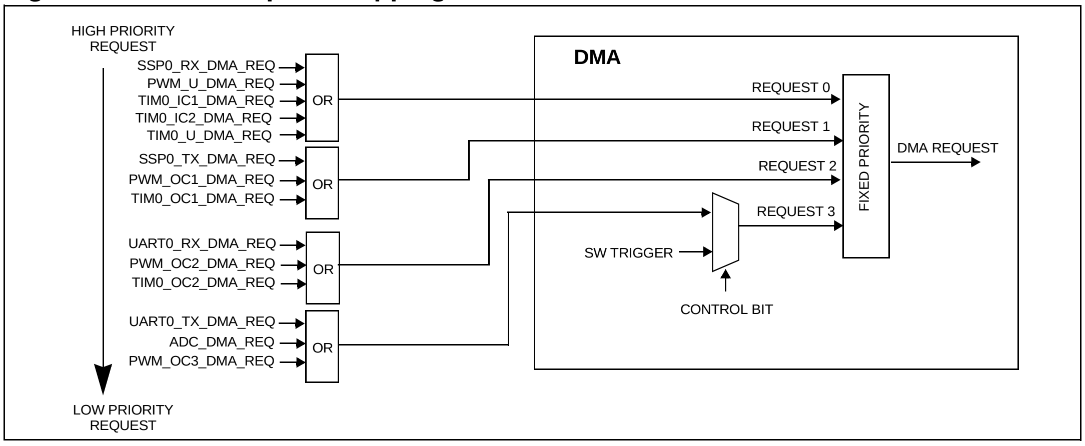
5.2 DMA controller priority

The priority between the different DMA triggering sources is defined by hardware, source 0 being the highest priority request and source 3 being the lowest one.

Note: A typical "single data" DMA transfer will include the following phases:
- Peripherals to DMA interrupt triggering (~2 cycles)
- DMA bus request, arbitration and memory/peripheral to DMA transfer (~5 cycles).
- Internal DMA latency (1 cycle)
- DMA bus request, arbitration and DMA to memory/peripheral transfer (~5 cycles)
Note that in the case of burst DMA transfer, arbitration and internal DMA latency are seen only once, each supplementary data transfer needing 1 cycle.
Example: burst transfer of 16 bytes organized in 4 words in memory and directed to a 16-bit peripheral (8 half-words).
Transfer_cycles = 2+(5+3)+1+(5+7) = 23 cycles (16 bytes)

5.3 DMA Request mapping

Figure 38. DMA Request mapping



5.3.1 SSP DMA Requests

Receive Requests:

1. Single character DMA transfer request when the receive FIFO contains at least one character
2. Burst DMA transfer request when the receive FIFO contains four or more characters

These two DMA requests are ORed to generate the Receive DMA request

Transmit Requests:

1. Single character DMA transfer request when there is at least one empty location in the transmit FIFO
2. Burst DMA transfer request when the transmit FIFO contains four or less characters

These two DMA requests are ORed to generate the Transmit DMA request

The Transmit DMA Request and the Receive DMA Request can be independantly activated/de-activated by programming control bits inside the SSP.

5.3.2 UART DMA Requests

Receive Requests:

1. Single character DMA transfer request when the receive FIFO contains at least one character.
2. Burst DMA transfer request when the receive FIFO contains more than the programmed watermark level.

These two DMA requests are ORed to generate the Receive DMA request

Transmit Requests:

1. Single character DMA transfer request when there is at least one empty location in the transmit FIFO
2. Burst DMA transfer request when when the transmit FIFO contains less than the programmed watermark level.

These two DMA requests are ORed to generate the Transmit DMA request

The Transmit DMA Request and the Receive DMA Request can be independantly activated/de-activated by programming control bits inside the UART.

5.3.3 ADC DMA Request

DMA Request is asserted after the conversion of a selected channel.

5.3.4 TIM0 DMA Requests

- At each Update Event
- At each Output Compare 1 Event
- At Each Input Capture 1 Event
- At each Output Compare 2 Event
- At Each Input Capture 2 Event

5.3.5 PWM Timer DMA Requests

- At each Update Event
- At each Output Compare 1 Event
- At each Output Compare 2 Event
- At each Output Compare 3 Event

The DMA controller treats bytes in memory as being in Little Endian format: the lowest numbered byte in a word is considered as the least significant byte and the highest numbered byte is the most significant.

5.4 Functional description

The DMA Controller is a single channel DMA Controller capable of servicing up to 4 data streams. It has a a single FIFO which is shared by the 4 data streams via the use of priority selection logic as shown in *Table 27*.

Table 27. DMA Request priority

DMA Request	Priority
External0	1 (Highest)
External1	2
External2	3
External3 - Internal0/1	4 (Lowest)

DMA Request3 is multiplexed with 2 internal request lines which are selected when a memory to memory transfer is required. The DMA block diagram is shown in *Figure 39*.

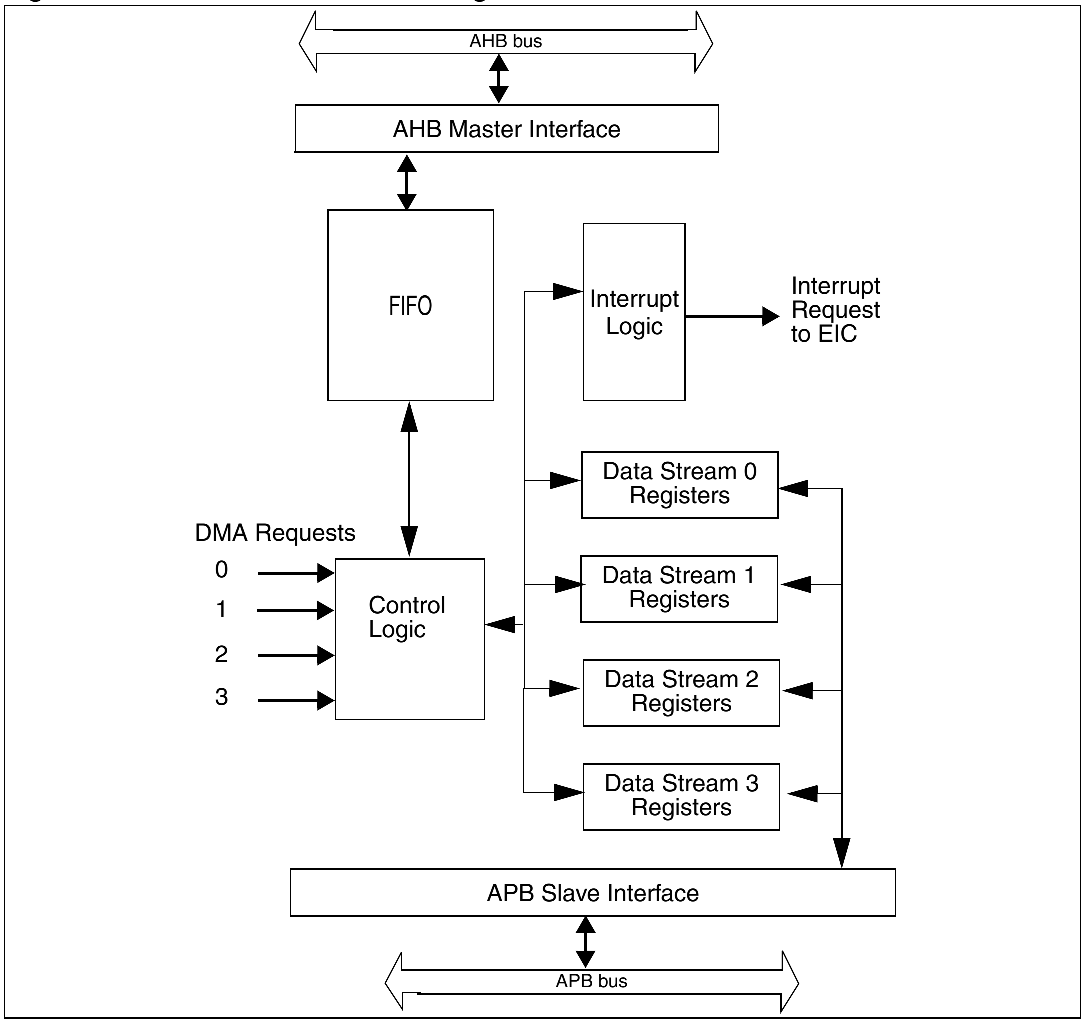
The DMA Controller can be used for a memory to memory transfer, a peripheral to memory transfer or a memory to peripheral transfer.

A DMA data transfer consists of a sequence of DMA data burst transfers. There are two types of burst transfers, the first one is from the source address to the DMA Controller and the second one is from the DMA Controller to the destination address. Each data burst transfer is characterized by the burst length (1, 4, 8, or 16 words) and by the word width (1 byte, 2 bytes or 4 bytes). The DMA data transfer is complete when the programmed total number of bytes has been transferred from the source address to the destination address (Terminal Count register going to zero).

The Control Logic is used to arbitrate between the data streams. It selects the request from the highest priority active data stream, passing the data from the chosen stream to and from the FIFO as required. Each stream has a set of data stream registers which are used by the Control Logic to determine source and destination addresses and the amount and format of data to be transferred. They also provide various other control and status information.

The DMA data stream registers are all 16-bit wide and are accessed via the APB Bus. The control logic can read all of these registers and can write some of them.

Figure 39. DMA Controller block diagram



There is a single dedicated interrupt line from each DMA Controller to the EIC. It is driven by 4 internal interrupt flags (one per data stream) which are ORed together. They work as follows:

Once a transfer for a specific stream is complete (this condition being detected by its Terminal Count register going to zero) the interrupt flag for the data stream is set. The interrupt logic then generates an interrupt to the EIC telling it that a request for one of the data streams has been completed. At the same time this data stream is disabled (i.e. the DMA Controller clears the enable bit of its control register). The status register must be read to determine which data stream caused the interrupt to be raised. The DMA Controller can now safely reconfigure this data stream (if required) for future transfers. A data stream can be re-enabled via a write to the enable bit of the corresponding Control Register.

The DMA FIFO block consists of a 16 32-bit words deep FIFO plus Data Pack and Data Unpack units. The purpose of the FIFO block is to accommodate bus latency and burst length, and to perform any packing or unpacking operations on the data that may be necessary to accommodate different data-out/data-in width ratios.

DMA Request 3 is multiplexed with 2 internal request signals which can be used for a memory to memory data transfer. To allow their use, the ‘Mem2Mem’ bit in the control register must be set. This tells the DMA Controller that data stream 3 is now configured for an internal, rather than an external, transfer request. An internal data transfer consists of two phases: phase 0 (where internal request 0 is asserted) is used for the memory to FIFO data transfer and phase 1 (where internal request 1 is asserted) is used for the FIFO to memory data transfer. An internal transfer will begin once the 'Mem2Mem' bit in the control register is set and data stream 3 is enabled, provided there are no pending requests for any of the other data streams, which all have higher priority. If there are pending requests, then these will be serviced first. When no other requests are pending, phase 0 will start as soon as the channel FIFO can accept the next data burst from the source address. The data packet size is defined in the data stream configuration registers. Phase 1 is asserted if the channel FIFO contains enough data for a next data burst to be sent to the destination address.

Phase 1 will be also started in order to flush the FIFO contents if any of the following conditions occur:

- If the terminal count reaches zero this indicates that the FIFO has received the total number of bytes to be transferred to the destination address. Since this number may be not a integer multiple of the selected burst size, bytes remaining in the FIFO after terminal count reaches zero must be flushed to the destination address.
- If stream 3 is disabled (via a write to the enable bit of the control register for this data stream).

In circular buffer mode, the DMA controller reloads the start address when the word counter (Terminal Count register) reaches the end of count and continues the transfer until application software sets the LAST bit, writing in the DMAn_LUBuff register the buffer location where the last data to be transferred is located. The stream configured in circular mode is controlled by a CIRCULAR flag in the DMAn_Ctrl register ('0'=normal mode, '1'=circular mode), a LAST flag in the DMAn_Last register ('0'=infinite mode, '1'=last buffer sweep), and a DMAn_LUBuff register (read and write).

When a circular buffer is the source of the transfer, the application software do the following:

1. Set the DMA stream configuration registers writing CIRCULAR bit to'1' and LAST bit to '0'.
2. Start feeding the circular buffer using an index to keep a trace of the last buffer location used.
3. When the end of transfer condition occurs, write DMAn_LUBuff with the value of the index and set the LAST bit to '1'.
4. The DMA interrupt line will be activated as soon as the DMAn_LUBuff location has been correctly transferred.

When a circular buffer is the destination of the transfer, the application sofware should proceed in a similar way:

1. Set the DMA stream configuration registers writing CIRCULAR bit to'1' and LAST bit to '0'.
2. Start fetching the circular buffer using an index to keep a trace of the last buffer location used.
3. When the end of transfer condition occurs, stop DMA operation writing DMA_EN to '0'.
4. The last buffer location to be used will be indicated by DMAn_DeCurr register pair.

Due to the fact that FIFO is always flushed when the end of buffer is reached, it is not possible to use circular buffer mode when the buffer size is not a multiple of the configured burst size. Circular buffer mode cannot be used in the following situations:

- When buffer size is not a multiple of the configured burst size. This is due to the fact that FIFO is always flushed when the end of buffer is reached, and the resulting burst would not be followed by a transfer moving the remaining locations, located at the beginning of circular buffer, to complete the programmed burst size.
- When memory to memory data transfer is configured on stream 3.

5.5 Register description

The DMA registers are accessed via the APB bus and the register data path is 16 bits wide.

5.5.1 Source Base Address Low (DMA_SOURCELx) (x=0,...,3)

Address Offset: 00h - 40h - 80h - C0h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMASOURCELx[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**DMASOURCELx[15:0]** DMAn_SOURCELx contains the low base address for stream x source DMA buffer.

5.5.2 Source Base Address High (DMA_SOURCEHx) (x=0,...,3)

Address Offset: 04h - 44h - 84h - C4h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMASOURCEHx[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**DMASOURCEHx[15:0]** DMASOURCEHx contains the high base address for stream x source DMA transfer.

5.5.3 Destination Base Address Low (DMA_DESTLx) (x=0,...,3)

Address Offset: 08h - 48h - 58h - 88h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMADESTLx[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**DMADESTLx[15:0]** DMADESTLx contains the low base address for stream x destination DMA buffer.

5.5.4 Destination Base Address High (DMA_DESTHx) (x=0,...,3)

Address Offset: 0Ch - 4Ch - 8Ch - CCh

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMADESTHx[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**DMA_DESTHxHx[15:0]** DMADESTHx contains the high base address for stream x destination DMA buffer.

5.5.5 Maximum Count Register (DMA_MAXx) (x=0,...,3)

Address Offset: 10h - 50h - 8Ch - D0h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAMAXx[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**DMAMAXx[15:0]** This register is programmed with stream x maximum data unit count, defining the buffer size. The data unit is equal to the configured source DMA data with (byte, half-word or word). Upon enabling DMA Controller, the content of the Maximum Count Register is loaded in the Terminal Count Register.

5.5.6 Control Register (DMA_CTRLx) (x=0, 1, 2)

Address Offset: 14h - 54h - 94h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved		Dir	reserved			Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
-		rw	-			rw	rw		rw		rw		rw	rw	rw

The DMAn_CTRLx register is used to configure Stream x operations.

Bits 15:14	Reserved, forced by hardware to 0.
Bit 13	**DIR** *Direction transfer* This bit is used to indicate if the peripheral is the source or the destination. 0: Peripheral is the source 1: Peripheral is the destination

Bits 12:10	Reserved, forced by hardware to 0.
Bit 9	**CIRCULAR** *Circular mode* This bit is used to enable the DMA to operate in the circular buffer mode. 0: Normal buffer mode 1: Circular buffer mode
Bits 8:7	**DESIZE** DMA to destination data width These bits are used to select the data width for DMA to destination data transfer. 00: 1 byte 01: 1 half-word 10: 1 word 11: reserved
Bits 6:5	**SOBURST** *DMA peripheral burst size* These bits are used to define the number of words in the peripheral burst. When the peripheral is the source, the number of (SoWidth) words read in to the FIFO before writing FIFO contents to destination. When the peripheral is the destination, the DMA interface will automatically read the correct number of source words to compile an SoBurst of the DeWith data. 00: Single 01: 4 incrementing 10: 8 incrementing 11: 16 incrementing
Bits 4:3	**SOSIZE** *Source to DMA data width* These bits are used to select the data width for source to DMA data transfer. 00: 1 byte 01: 1 half-word 10: 1 word 11: reserved
Bit 2	**DEINC** *Increment Current Destination Register* This bit is used to enable the Current Destination Register after each DMA to destination data transfer. 0: Current Destination Register unchanged 1: Current Destination Register incremented
Bit 1	**SOINC** *Increment Current Source Register* This bit is used to enable the Current Source Register after each source to DMA data transfer. 0: Current Source Register unchanged 1: Current Source Register incremented
Bit 0	**ENABLE** *DMA enable* 0: DMA disabled 1: DMA enabled

5.5.7 Control Register 3 (DMA_CTRL3)

Address Offset: D4h

Reset value: 0000h

15 14	13	12	11	10	9	8 7	6 5	4 3	2	1	0
reserved	Dir	reserved	Mem2Mem	Res.	Circular	DeSize	SoBurst	SoSize	DeInc	SoInc	Enable
-	rw	-	rw	-	rw	rw	rw	rw	rw	rw	rw

The DMAn_CTRL3 register is used to configure Stream 3 operations.

Bits 15:14	Reserved, forced by hardware to 0.
Bit 13	**DIR** *Direction transfer* This bit is used to indicate if the peripheral is the source or the destination. 0: Peripheral is the source 1: Peripheral is the destination
Bit 12	Reserved, forced by hardware to 0.
Bit 11	**MEM2MEM** *Selects memory to memory transfer* This configures Stream 3 to operate a memory to memory transfer. When MEM2MEM is set, the DMA will disregard the DMA request connected to Stream 3, and transfer data from source to destination as fast possible until DMAn_MAX3 expires. 0: Stream3 not configured for mem to mem transfer 1: Stream3 configured for mem to mem transfer When Stream 3 is configured as a memory-memory transfer, SOBURST relates to the source side burst length.
Bit 10	Reserved, forced by hardware to 0.
Bit 9	**CIRCULAR** *Circular mode* This bit is used to enable the DMA to operate in circular buffer mode. 0: Normal buffer mode 1: Circular buffer mode
Bits 8:7	**DESIZE** *DMA to destination data width* These bits are used to select the data width for DMA to destination data transfer. 00: 1 byte 01: 1 half-word 10: 1 word 11: reserved
Bits 6:5	**SOBURST** *DMA peripheral burst size* These bits are used to define the number of words in the peripheral burst. When the peripheral is the source, the number of (SOWIDTH) words read in to the FIFO before writing FIFO contents to destination. When the peripheral is the destination, the DMA interface will automatically read the correct number of source words to compile an SOBURST of the DEWIDTH data. 00: Single 01: 4 incrementing 10: 8 incrementing 11: 16 incrementing

Bits 4:3	**SOSIZE** *Source to DMA data width* These bits are used to select the data width for source to DMA data transfer. 00: 1 byte 01: 1 half-word 10: 1 word 11: reserved
Bit 2	**DEINC:** *Increment Current Destination Register* This bit is used to enable the Current Destination Register after each DMA to destination data transfer. 0: Current Destination Register unchanged 1: Current Destination Register incremented
Bit 1	**SOINC:** *Increment Current Source Register* This bit is used to enable the Current Source Register after each source to DMA data transfer. 0: Current Source Register unchanged 1: Current Source Register incremented
Bit 0	**ENABLE:** *DMA enable* 0: DMA disabled 1: DMA enabled

5.5.8 Current Source Address High (DMA_SOCURRHx) (x=0,...,3)

Address Offset: 18h - 58h - 98h - D8h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMASOCURRHx[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0	**DMASOCURRHx[15:0]** The DMAn_SOCURRHx register holds the current value of the high source address pointer related to Stream x. This register is read only.

5.5.9 Current Source Address Low (DMA_SOCURRLx) (x=0,...,3)

Address Offset: 1Ch - 5Ch - 9Ch - DCh

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMASOCURRLx[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0	**DMASOCURRLx[15:0]** Then DMAn_SOCURRLx register holds the current value of the low source address pointer related to Stream x. This register is read only.

The value in the registers (DMAn_SOCURRLx and DMAn_SOCURRHx) is used as an AHB address in a source to DMA data transfer over the AHB bus. If the SOINC bit in the Control Register is set to ‘1’, the value in the Current Source Registers will be incremented as data are transferred from a source to the DMA. The value will be incremented at the end of the address phase of the AHB bus transfer by the transferred size value. If the SOINC bit is ‘0’, the Current Source Register will hold a same value during the whole DMA data transfer.

5.5.10 Current Destination Address High (DMA_DECURRHx) (x=0,...,3)

Address Offset: 20h - 60h - A0h - E0h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMADeCurrHiX															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0	**DMADeCurrHiX[15:0]** DMADeCurrHiX holds the current value of the high destination address pointer related to stream X. This register is read only.

5.5.11 Current Destination Address Low (DMA_DECURRLx) (x=0,...,3)

Address Offset: 24h - 64h - A4h - E4h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMADECURRLx[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0	**DMADECURRLx[15:0]** The DMAn_DECURRLx register holds the current value of the low destination address pointer related to Stream x. This register is read only. The value in the registers (DMAn_DECURRLx and DMAn_DECURRHx) is used as an AHB address in a DMA to destination data transfer over the AHB bus. If the DEINC bit in the Control Register is set to ‘1’, the value in the Current Destination Registers will be incremented as data are transferred from DMA to destination. The value will be incremented at the end of the address phase of the AHB bus transfer by the transferred size value. If DEINC bit is ‘0’, the Current Destination Register will hold a same value during the whole DMA data transfer.

5.5.12 Terminal Counter Register (DMA_TCNTx) (x=0,...,3)

Address Offset: 28h - 68h - A8h - E8h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMATCNTx[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0	**DMATCNTx[15:0]** The DMAn_TCNTx register contains the number of data units remaining in the current DMA transfer. The data unit is equal to the source to DMA data width (byte, half-word or word). The register value is decremented every time data is transferred to the DMA FIFO. When the Terminal Count reaches zero, the FIFO content is transferred to the Destination and the DMA transfer is finished. This is a read only register. **Note :** DMATCntX register can be used also in Circular buffer mode, with the exception of the last buffer sweep. Once LAST flag is set, the value of DMATCntX register becomes not meaningful and should be ignored.

5.5.13 Last Used Buffer location X (DMA_LUBuffX) (X=0,...,3)

Address Offset: 2Ch - 6Ch - ACh - ECh

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMALUBuff															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	DMALUBuffX[15:0] DMALUBuffX is used in circular buffer mode during last buffer sweep, and it contains the circular buffer position where the last data to be used by stream X is located. The first buffer location is indicated writing 0x0000 into this register, the second with 0x0001 and so on, up to the last location which is indicated setting this register with (DMAMaxX - 1).

5.5.14 Interrupt Mask Register (DMA_MASK)

Address Offset: F0h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								SEM3	SEM2	SEM1	SEM0	SIM3	SIM2	SIM1	SIM0
-								rw	rw	rw	rw	rw	rw	rw	rw

The DMA Mask Register is user to select the status flag that can generate an interrupt.

Bits 15:8	Reserved, forced by hardware to 0.
Bit 7	**SEM3** *Stream 3 Error Mask* This bit controls the generation of DMA interrupts triggered by stream 3 transfer errors events. 0: Stream 3 transfer error interrupt is masked 1: Stream 3 transfer error interrupt is enabled
Bit 6	**SEM2** *Stream 2 Error Mask* This bit controls the generation of DMA interrupts triggered by stream 2 transfer errors events. 0: Stream 2 transfer error interrupt is masked 1: Stream 2 transfer error interrupt is enabled
Bit 5	**SEM1** *Stream 1 Error Mask* This bit controls the generation of DMA interrupts triggered by stream 1transfer errors events. 0: Stream 1 transfer error interrupt is masked 1: Stream 1 transfer error interrupt is enabled

Bit 4	**SEM0** *Stream 0 Error Mask* This bit controls the generation of DMA interrupts triggered by stream 0 transfer errors events. 0: Stream 0 transfer error interrupt is masked 1: Stream 0 transfer error interrupt is enabled
Bit 3	**SIM3** *Stream 3 Interrupt Mask* This bit controls the generation of DMA interrupts triggered by stream 3 transfer end events. 0: Stream 3 transfer end interrupt is masked 1: Stream 3 transfer end interrupt is enabled
Bit 2	**SIM2** *Stream 2 Interrupt Mask* This bit controls the generation of DMA interrupts triggered by stream 2 transfer end events. 0: Stream 2 transfer end interrupt is masked 1: Stream 2 transfer end interrupt is enabled
Bit 1	**SIM1** *Stream 1 Interrupt Mask* This bit controls the generation of DMA interrupts triggered by stream 1 transfer end events. 0: Stream 1 transfer end interrupt is masked 1: Stream 1 transfer end interrupt is enabled
Bit 0	**SIM0** *Stream 0 Interrupt Mask* This bit controls the generation of DMA interrupts triggered by stream 0 transfer end events. 0: Stream 0 transfer end interrupt is masked 1: Stream 0 transfer end interrupt is enabled

5.5.15 Interrupt Clear Register (DMA_CLR)

Address Offset: F4h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								SEC3	SEC2	SEC1	SEC0	SIC3	SIC2	SIC1	SIC0
-								w	w	w	w	w	w	w	w

The DMA Clear Register is used to clear the status flags. This is a write-only register.

Bits 15:8	Reserved, forced by hardware to 0.
Bit 7	**SEC3** *Stream 3 Error Clear* This bit allows clearing the pending interrupt flag corresponding to stream 3 transfer error event. 0: No effect 1: Clear SEF3 flag in DMAn_Status register
Bit 6	**SEC2** *Stream 2 Error Clear* This bit allows clearing the pending interrupt flag corresponding to stream 2 transfer error event. 0: No effect 1: Clear SEF2 flag in DMAn_Status register
Bit 5	**SEC1** *Stream 1 Error Clear* This bit allows clearing the pending interrupt flag corresponding to stream 1 transfer error event. 0: No effect 1: Clear SEF1 flag in DMAn_Status register
Bit 4	**SEC0:** *Stream 0 Error Clear* This bit allows clearing the pending interrupt flag corresponding to stream 0 transfer error event. 0: No effect 1: Clear SEF0 flag in DMAn_Status register
Bit 3	**SIC3** *Stream 3 Interrupt Clear* This bit allows clearing the pending interrupt flag corresponding to stream 3 transfer end event. 0: No effect 1: Clear SIF3 flag in DMAn_Status register
Bit 2	**SIC2** *Stream 2 Interrupt Clear* This bit allows clearing the pending interrupt flag corresponding to stream 2 transfer end event. 0: No effect 1: Clear SIF2 flag in DMAn_Status register
Bit 1	**SIC1** *Stream 1 Interrupt Clear* This bit allows clearing the pending interrupt flag corresponding to stream 1 transfer end event. 0: No effect 1: Clear SIF1 flag in DMAn_Status register
Bit 0	**SIC0** *Stream 0 Interrupt Clear* This bit allows clearing the pending interrupt flag corresponding to stream 0 transfer end event. 0: No effect 1: Clear SIF0 flag in DMAn_Status register

5.5.16 Interrupt Status Register (DMA_STATUS)

Address Offset: F8h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				ACT3	ACT2	ACT1	ACT0	SEF3	SEF2	SEF1	SEF0	SIF3	SIF2	SIF1	SIF0
-				r	r	r	r	r	r	r	r	r	r	r	r

The DMAn_Status provides status information regarding the DMA Controller. This is a read-only register.

Bits 15:12	Reserved, forced by hardware to 0
Bit 11	**ACT3:** *Data stream 3 status* 0: Data Stream 3 is not active 1: Data Stream 3 is active
Bit 10	**ACT2:** *Data stream 2 status* 0: Data Stream 2 is not active 1: Data Stream 2 is active
Bit 9	**ACT1:** *Data stream 1 status* 0: Data Stream 1 is not active 1: Data Stream 1 is active
Bit 8	**ACT0** *Data stream 0 status* 0: Data Stream 0 is not active 1: Data Stream 0 is active
Bit 7	**SEF3** *Data stream 3 error flag* When a transfer error event occurs on Stream 3, this bit will be set to '1' and if the SEM3 bit the DMAn_MASK register has also set to '1' by software then a DMA interrupt request will be generated. This flag is cleared by writing '1' in the SEC3 bit in the DMAn_CLEAR register.
Bit 6	**SEF2** *Data stream 2 error flag* When a transfer error event occurs on Stream 2, this bit will be set to '1' and if the SEM2 bit the DMAn_MASK register has also set to '1' by software then a DMA interrupt request will be generated. This flag is cleared by writing '1' in the SEC2 bit in the DMAn_CLEAR register.
Bit 5	**SEF1** *Data stream 1 error flag* When a transfer error event occurs on Stream 1, this bit will be set to '1' and if the SEM1 bit the DMAn_MASK register has also set to '1' by software then a DMA interrupt request will be generated. This flag is cleared by writing '1' in the SEC1 bit in the DMAn_CLEAR register.
Bit 4	**SEF0** *Data stream 0 error flag* When a transfer error event occurs on Stream 0, this bit will be set to '1' and if the SEM0 bit the DMAn_MASK register has also set to '1' by software then a DMA interrupt request will be generated. This flag is cleared by writing '1' in the SEC0 bit in the DMAn_CLEAR register.

Bit 3	**SIF3** *Data stream 3 interrupt flag* When a transfer end event occurs on Stream 3, this bit will be set to ‘1’ and if the SIM3 bit in the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SIC3 bit in the DMAn_CLEAR register.
Bit 2	**SIF2** *Data stream 2 interrupt flag* When a transfer end event occurs on Stream 2, this bit will be set to ‘1’ and if the SIM2 bit in the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SIC2 bit in the DMAn_CLEAR register.
Bit 1	**SIF1** *Data stream 1 interrupt flag* When a transfer end event occurs on Stream 1, this bit will be set to ‘1’ and if the SIM1 bit in the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SIC1 bit in the DMAn_CLEAR register.
Bit 0	**SIF0** *Data stream 0 interrupt flag* When a transfer end event occurs on Stream 0, this bit will be set to ‘1’ and if the SIM0 bit in the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SIC0 bit in the DMAn_CLEAR register.

5.5.17 Last Flag Register (DMA_LAST)

Address Offset: FCh

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												LAST3	LAST2	LAST1	LAST0
-												rw	rw	rw	rw

DMAn_Last controls the activation of last buffer sweep mode for the streams configured in circular buffer mode.

Bits 15:4	Reserved, forced by hardware to 0.
Bit 3	**LAST3** *LAST buffer sweep stream 3* This bit is used to notify DMA that last circular buffer sweep started. If this bit is set while stream 3 is configured in circular mode, the corresponding data stream interrupt flag will get set when DMA uses the circular buffer location contained in DMAn_LUBuff3 register. 0: Continuous circular buffer mode 1: Last circular buffer sweep started.
Bit 2	**LAST2** *LAST buffer sweep stream 2* This bit is used to notify DMA that last circular buffer sweep started. If this bit is set while stream 2 is configured in circular mode, the corresponding data stream interrupt flag will get set when DMA uses the circular buffer location contained in DMAn_LUBuff2 register. 0: Continuous circular buffer mode 1: Last circular buffer sweep started.
Bit 1	**LAST1** *LAST buffer sweep stream 1* This bit is used to notify DMA that last circular buffer sweep started. If this bit is set while stream 1 is configured in circular mode, the corresponding data stream interrupt flag will get set when DMA uses the circular buffer location contained in DMAn_LUBuff1 register. 0: Continuous circular buffer mode 1: Last circular buffer sweep started.
Bit 0	**LAST0** *LAST buffer sweep stream 0* This bit is used to notify DMA that last circular buffer sweep started. If this bit is set while stream 0 is configured in circular mode, the corresponding data stream interrupt flag will get set when DMA uses the circular buffer location contained in DMAn_LUBuff0 register. 0: Continuous circular buffer mode 1: Last circular buffer sweep started.

5.6 DMA Register map

The following table summarizes the DMA registers:

Table 28. DMA Register map

Addr. Off set	Reg. Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00	DMA_SOURCEL0	DMASOL0															
04	DMA_SOURCEH0	DMASOH0															
08	DMA_DESTL0	DMADEL0															
0C	DMAn_DESTH0	DMADEH0															
10	DMA_MAX0	DMAMAX0															
14	DMA_CTRL0	reserved		Dir	reserved			Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
18	DMA_SOCURRH0	DMASOCURRH0															
1C	DMA_SOCURRL0	DMASOCURRL0															
20	DMA_DECURRH0	DMADECURRH0															
24	DMA_DECURRL0	DMADECURRL0															
28	DMA_TCNT0	DMATCNT0															
2C	DMAnLUBUFF0	DMALUBuff0															
30	-	Reserved															
40	DMA_SOURCEL1	DMASoLo1															
44	DMA_SOURCEH1	DMASoHi1															
48	DMA_DESTL1	DMADeLo1															
4C	DMA_DESTH1	DMADeHi1															
50	DMA_MAX11	DMAMax1															
54	DMA_CTRL1	reserved		Dir	reserved			Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
58	DMA_SOCURRH1	DMASoCurrHi1															
5C	DMA_SOCURRL1	DMASoCurrLo1															
60	DMA_DECURRH1	DMADeCurrHi1															
64	DMA_DECURRL1	DMADeCurrLo1															
68	DMA_TCNT1	DMATCnt1															
6C	DMA_LUBUff1	DMALUBuff1															
70	-	Reserved															
80	DMA_SOURCEL2	DMASOL2															
84	DMA_SOURCEH2	DMASOH2															
88	DMA_DESTL2	DMADEL2															
8C	DMA_DESTH2	DMADEH2															
90	DMA_MAX2	DMAMax2															
94	DMA_CTRL2	reserved		Dir	reserved			Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
98	DMA_SOCURRH2	DMASOCURRH2															
9C	DMA_SOCURRL2	DMASOCURRL2															
A0	DMA_DECURRH2	DMADECURRH2															

Table 28. DMA Register map (continued)

Addr. Off set	Reg. Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A4	DMA_DECURRL2	DMADECURRL2															
A8	DMA_TCNT2	DMATCNT2															
AC	DMA_LUBUFF2	DMALUBuff2															
B0	-	Reserved															
C0	DMA_SOURCEL3	DMASOL3															
C4	DMA_SOURCEH3	DMASOH3															
C8	DMA_DESTL3	DMADEL3															
CC	DMA_DESTH3	DMADEH3															
D0	DMA_MAX3	DMAMAX3															
D4	DMA_CTRL3	reserved		Dir	res.	Mem 2Me m	res	Cir-cular	DeSize		SoBurst		SoSize		DeIn c	SoIn c	Ena-ble
D8	DMA_SOCURRH3	DMASOCURRH3															
DC	DMA_SOCURRL3	DMASOCURRL3															
E0	DMA_DECURRH3	DMADECURRH3															
E4	DMA_DECURRL3	DMADECURRL3															
E8	DMA_TCNT3	DMATCNT3															
EC	DMA_LUBUFF3	DMALUBuff3															
F0	DMA_MASK	Reserved								SEM 3	SEM 2	SEM 1	SEM 0	SIM3	SIM2	SIM1	SIM0
F4	DMA_CLR	Reserved								SEC 3	SEC 2	SEC 1	SEC 0	SIC3	SIC2	SIC1	SIC0
F8	DMA_STATUS	Reserved				ACT 3	ACT 2	ACT 1	ACT 0	SEF3	SEF2	SEF1	SEF0	SIF3	SIF2	SIF1	SIF0
FC	DMA_LAST	reserved												LAST 3	LAST 2	LAST 1	LAST 0

Refer to *Table 2* for the base addresses.

6 Serial memory interface (SMI)

6.1 Introduction

The Serial Memory Interface provides an AHB slave interface to external SPI memory devices. The CPU can use this memory as data or program memory.

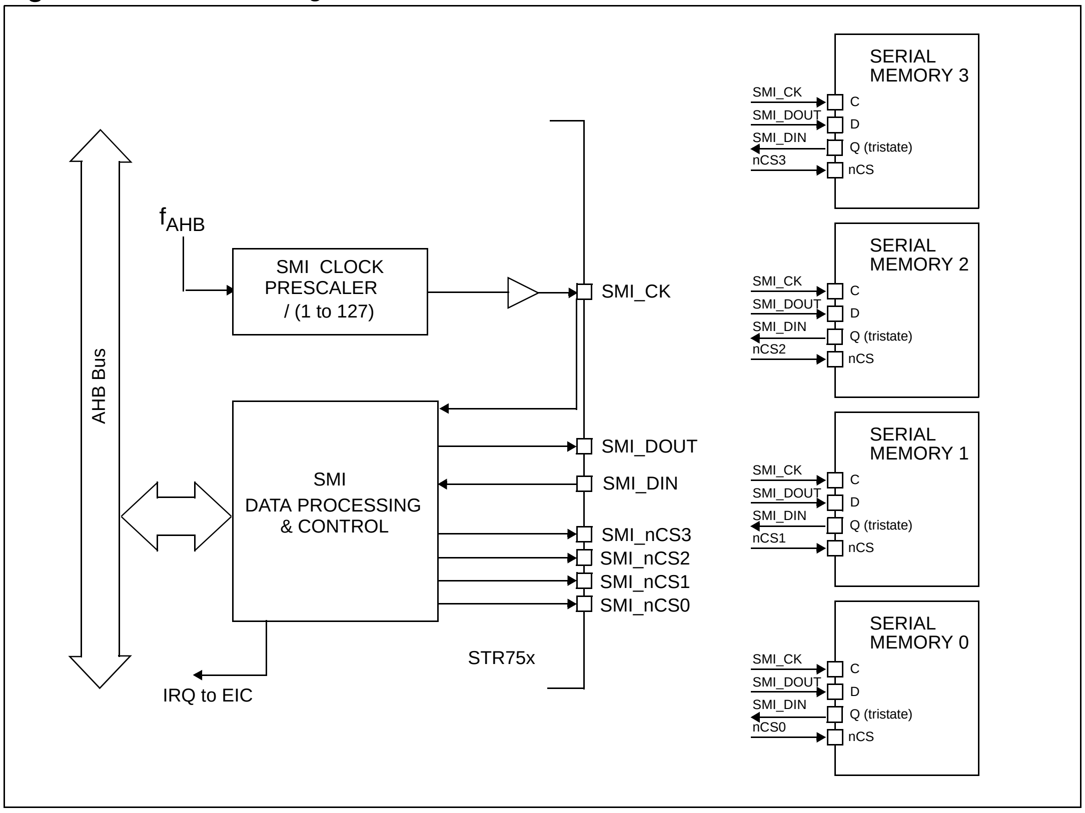
6.2 Main features

- 32, 16 or 8-bit AHB interface
- Programmable clock prescaler
- Up to 20 MHz clock speed in normal read mode and up to 48 MHz in fast read mode
- Up to 30 MHz in write mode
- 4 Chip Select signals (for addressing 4 banks)
- Up to 16 MBytes address space per bank
- Direct Exection by the ARM7 in ARM 32-bit mode (Thumb mode not supported)
- External memory boot mode capability
- Compatible with 3-byte addressing devices:
  - STMicroelectronics M25Pxxx, M45Pxxx
- Interrupt Request on Write Complete or Software Transfer Complete

6.3 Block diagram

The Block Diagram is shown in *Figure 40*.

Figure 40. SMI block diagram



6.4 Functional description

6.4.1 Clock configuration when using SMI

When using the SMI, some combinations of the AHB clock prescalers and SMI prescaler are forbidden.

They are described below:

Table 29. Clock configuration rules when using SMI

AHB prescaler	SMI prescaler	Rule
/1	any setting	authorized
/2, /4, or /8	/1	forbidden
/2, /4, or /8	/2 or more	authorized

Table 30. Clock configuration / memory access rules when using SMI

SMI prescaler	Memory access mode	Rule
/1	read	authorized
	write	forbidden
/2 or more	read	authorized
	write	

It is up to the user to take care to avoid setting a forbidden configuration. Otherwise the resulting SMI clock may not have a 50% duty cycle.

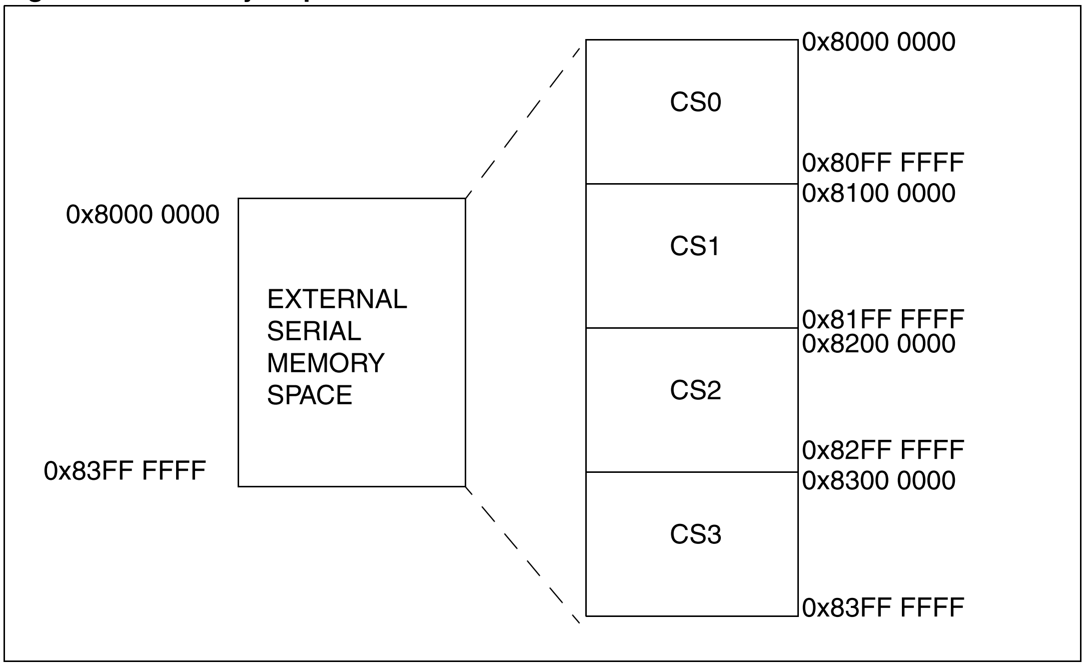
There are some restrictions in the choice of the SMI prescaler depending on the type of access (Read or a Write) performed to the external serial memory. They are described below :

6.4.2 AHB interface

External memory is mapped in AHB address space as shown in *Figure 41*. Up to four banks of 16 Mbytes each can be selected by address bits [25:24]. The following rules apply to the access from the AHB to the SMI.

- Endianness is fixed to little-endian.
- Bursts must not cross Bank boundaries.
- Size of data transfers for memories can be byte/half-word/word.
- Size of data transfers for registers must be 32-bit wide.
- Read Requests: all types of BURST are supported. Wrapping bursts take more time than incrementing bursts, as there is a break in the address increment.
- Write Requests: wrapping bursts are not supported.
- When BUSY transfer: the SMI transfer is held until busy is inactive.
- Address must be aligned to the size of the transfer.

Figure 41. Memory map



The External Memory Address space can be directly executed by the ARM7 processor in ARM 32-bit mode. This feature is only supported in ARM 32-bit mode (it is forbidden to execute from the SMI in Thumb mode).

6.4.3 Memory device compatibility

The communication protocol used is SPI in CPOL = 1 and CPHA = 1 mode.
The instructions supported are listed in *Table 31*.

Table 31. Supported instruction set

Opcode	Description
03h	Read data bytes
0Bh	Read data high speed
05h	Read status register
06h	Write enable
02h	Page program
ABh	Release from deep power-down

6.4.4 Hardware mode

At reset, the SMI operates in hardware mode. In this mode, the SMI Transmit Register (SMI_TR) and SMI Receive Register (SMI_RR) must not be accessed. They are managed by the SMI hardware and used to communicate with the external memory devices whenever an AHB master reads or writes to an address in external memory.

6.4.5 Software mode

In Software Mode, SMI Transmit Register (SMI_TR) and SMI Receive Register (SMI_RR) are accessible. Direct AHB transfers to/from external memories are not allowed. To enable Software Mode, the SW bit in the SMI_CR1 register has to be set to 1.

Software Mode is used to transfer any data or commands from the SMI_TR register to external memory and to read data directly in the SMI_RR register. The transfer is started using the SEND bit in the SMI_CR2 register.

For example Software Mode is used to erase flash memory before writing. Erase cannot be managed in Hardware Mode due to incompatibilites that exist between Flash devices from different vendors.

In Software Mode, application code being executed by the core cannot be fetched from external memory. It must either reside in internal memory, or be previously loaded from external memory while the SMI is in Hardware Mode.

6.4.6 Booting from external memory

If External boot mode is enabled via the microcontroller BOOT pins, Bank 0 is enabled at power-on and the following command sequence is sent to Bank 0.

- Release from deep power-down
- 30 µs delay
- Read Status Register
- Read Data Bytes at memory start location

All other banks are disabled at reset and must be enabled by setting the BE[3:1] bits in the SMI_CR1 register before thay can be accessed.

If external memory boot mode is not selected, then all banks are disabled at reset.

6.4.7 External memory read request

A read request to external memory is served only if the SMI is in Hardware Mode (SMI_CR1 SW = 0), and Write Burst Mode is not selected (SMI_CR1 WBM = 0). Otherwise the ERF1 flag in the SMI_SR register is set.

When a read request occurs in normal mode (SMI_CR1 FAST=0), the following sequence is sent to the bank selected by the AHB address bits [25-24]:

- Read Data Bytes opcode (03h).
- 3-byte address from the most to the least significant bit.
- Then clock is sent until the end of burst request from master.

When a read request occurs in high speed mode (SMI_CR1 FAST = 1), the following sequence is sent to the bank that is addressed:

- Read Data Bytes at High Speed opcode (0Bh).
- 3-byte address from the most to the least significant bit.
- 1 dummy byte (00h).
- Then clock is sent until the end of burst request from master.

The external memory bank remains selected as long as there is no external memory address jump, and as long as no new commands are sent to the SMI (such as WEN, RSR, SW mode or WBM mode, Write Request, Bank disable, Prescaler configuration change (different value written into SMICR1(15:4)) or memory access Error).

It also remains selected when the address rolls over from FFFFFFh to 000000h in same bank.

6.4.8 External memory write request

A write request from AHB is served only if the SMI is in Hardware Mode (SMI_CR1 SW = 0). Otherwise the ERF1 flag in the SMI_SR register is set.

Write capability must be used only if Write in Progress/Busy bit (WIP) of the external memory Status Register is located in bit 0. Otherwise the system will become locked.

Wrapping bursts are not allowed as serial memories don’t support them.

When a write request occurs, it is sent to external memory if the following conditions are met:

- **Bank in Write mode:** When a bank is in Write Mode, the corresponding WM flag is set in the SMI_SR register. If this condition is not met when a write request occurs, the ERF2 flag in the SMI_SR is set. To enable Write Mode, select the Bank using the BS bits in the SMI_CR2 register and then set the WEN bit in the SMI_CR2 register.
- **No Write in Progress**: The WIP bit in the SMI_SR register must be cleared. If this condition is not met, AHB is stalled until WIP = 0.

When these 2 conditions are met, the following sequence is sent to the bank selected by AHB address bits [25-24]:

- Page Program opcode (02h).
- 3-byte address from the most to the least significant bit.
- Then transfer all the data bytes from bit 7 to bit 0, starting with address given previously and incrementing it to the last depending on the size of the write request.

After a write request is sent to external memory, Write Mode bit is reset and the Read Status Register instruction is automatically sent to this bank until WIP = 0.

Bits 7:0 of the SMI_ SR are refreshed every 8 SMI_CK periods with the contents of the status register read from the selected external memory.

When memory programming is finished, the WCF in the SMI_SR is set and an interrupt is generated if the WCIE bit in the SMI_CR2 register is set.

In order to send a write request to another bank than the one under programmation, the software must wait for WIP = 1, otherwise the error ERF2 would be generated due to non incrementing address. The bank under programming phase must not be disable in order to write to another one.

Write burst mode

Write Burst Mode is available to keep the external memory selected after the AHB write request (SMI_CR1 WBM = 1). In that case, the next AHB to external memory write request has to be to the next incremented address and it has to be the same size (byte, halfword or word). Otherwise the ERF2 flag in the SMI_SR register is set. The external memory selection is released by resetting WBM or disabling the bank, and the external memory page program cycle starts. If Bank is enable, Read Status Register instruction is automatically sent to this bank until WIP = 0.

A memory access error (ERF1 or ERF2) generates the CS release and the start of the external memory page program.

If Write Burst Mode is not selected, the next incrementing AHB write request will be sent to external memory if it occurs before the end of the previous serial transfer. Otherwise the ERF2 flag in the SMI_SR register is set. Consequently it is mandatory to set WBM bit in order to perform several write requests which are not sent in the same AHB incrementing burst. If WBM = 0 and no other write request occurs, the external memory selection is released after sending the data, and the external memory page program cycle starts.

Read requests to external memory are forbidden when WBM = 1. Otherwise the ERF1 flag in the SMI_SR register is set.

6.4.9 Read while write

If a read to the same bank which is in progranning phase occurs, the AHB is stalled until WIP = 0.

If a read to another bank occurs, the Read Status Register sequence is stopped, the read request is served and then the Read Status Register sequence is re-sent to the memory being programmed. So during a Read While Write, the external memory select is released after the read command, in order to send the Read Status Register sequence.

6.4.10 Erasing and write status register

In case of Serial Flash an Erase may be necessary before writing. Due to uncompatibility between different serial flash vendors, Erase and Write Status Register can be done only in Software Mode.

It is mandatory to send previously the Write Enable Instruction through Software Mode only, in order not to corrupt the WM bit in the SMI_SR register as the end of internal flash ERASE or Write Status Register cannot be checked by hardware (and consequently Write Complete interrupt is not generated). WIP bit can be checked by sending RSR command continuously.

6.4.11 Timings

SMI_CK is generated by f_{AHB} divided by the value given by the PRESC bits in the SMI_CR1 register. It is held at '1' between transfers.

When f_{AHB} is divided by an odd value:

SMI_CK high time = (presc-1)/2

SMI_CK low time = (presc+1)/2

The AHB bus is stalled until the complete data is read or written.

The normal latency for a 32-bit single read to a non_incrementing serial flash address, assuming the SMI is not already busy is:

- 73 t_{AHB} maximum for presc = 1
- 68 t_{SMI_CK} + 5 t_{AHB} maximum for presc > 1.(refer to *Figure 43*)

The normal latency for a 32-bit single write to a non-incrementing serial flash address, assuming the SMI is not already busy is:

- 5 t_{AHB} maximum for presc = 1
- 2 t_{SMI_CK} + 3 t_{AHB} maximum otherwise for presc > 1.

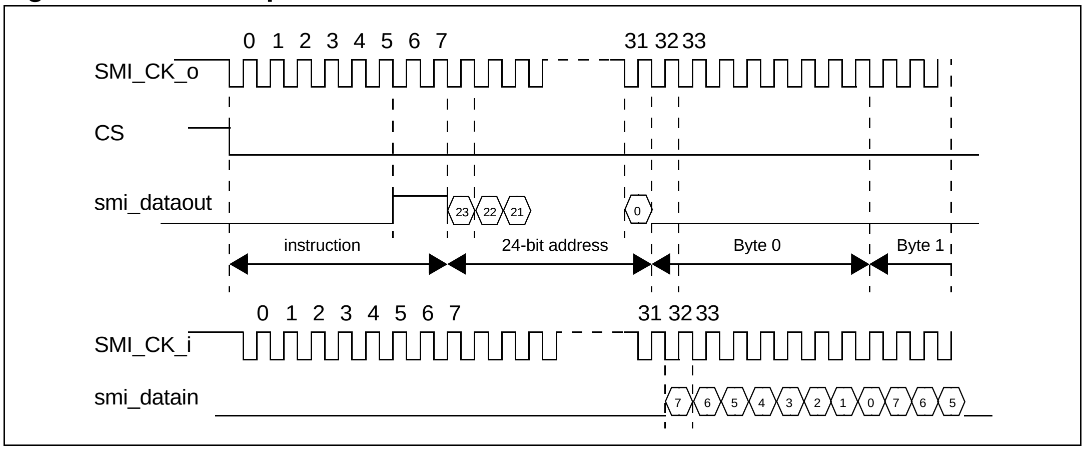
Latency is increased by:

- SMI transfer on going (read, write, RSR or WEN)
- Tcs programming (adds Tcs +1 SMI_CK periods)
- Busy/Idle transfer on AHB
- Fast read which adds 1 dummy byte
- Hold programming
- Boot delay time
- Frequency change
- Programming on-going

For AHB burst read transfers, the maximum latency after the first transfer is the same as the data size: 32 SMI_CK for a word transfer, 16 SMI_CK for a halfword and 8 SMI_CK for a byte.

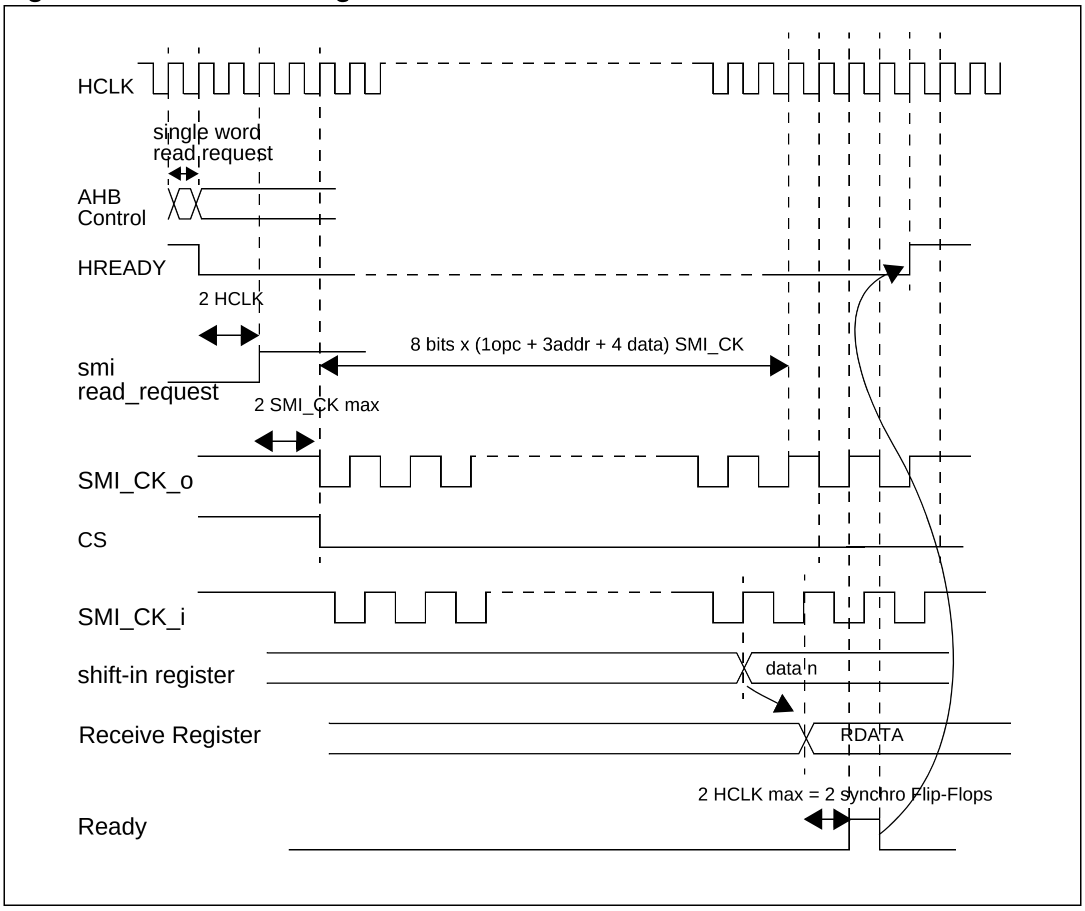
For AHB burst write transfers, the maximum latency for the 2nd transfer will be the data size + opcode + address bytes. The subsequent maximum latency will be the data size.

Figure 42. Read sequence



External memories chip select are generated on the falling edge of SMI_CK.

Figure 43. AHB Word Single Read Transfer



6.4.12 Error management

Several requests can generate an error:

- memory transfer request size different from byte, halfword or word.
- register transfer request size different from word.
- memory read or write request while SMI_CR1 Bank Enable is not set or SMI_CR1 SW bit is set.
- memory write wrapping burst
- memory write request while SMI_SR WM bit is not set.
- memory write request while SMI_CR1 WBM is set and address is not incremented or size request differs.
- memory read request while SMI_CR1 WBM bit is set.

Error flags

There are 2 error flags in SMI_SR:

- ERF1: forbidden H/W request (SW = 1 or Bank not enable or Read with WBM = 1)

- ERF2: forbidden H/W write request (WM = 0 or non-incrementing address or different size)

6.4.13 Interrupts

Table 32. Interrupts

Interrupt Event	Event Flag SMI_SR	Enable Control Bit SMI_CR2
Write Complete	WCF	WCIE
SW Transfer Complete	TFF	TFIE

6.5 Register description

The SMI registers can only be accessed by a 32-bit operation. A halfword or byte cannot be read or written. In this section, the following abbreviations are used:

read/write (rw)	Software can read and write to these bits.
read-only (r)	Software can only read these bits.
read/clear (rc0)	Software can read as well as clear this bit by writing 0. Writing ‘1’ has no effect on the bit value.
read/set (rs)	Software can read as well as set this bit. Writing ‘0’ has no effect on the bit value.

6.5.1 SMI control register 1 (SMI_CR1)

Address Offset: 00h

Reset value: depends on hardware configuration: 00 00 02 51h in External Memory Boot Mode, else 00 00 02 50h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved		WBM	SW	reserved				HOLD							
-		rw	rw					rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FAST	PRESC							TCS				BE			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30	Reserved, forced by hardware to 0.
Bit 29	**WBM:** *Write Burst Mode* 0: When a AHB write request to external memory occurs, chip select is released at the end of the AHB request and page programming cycle starts. 1: After the AHB write request, the external memory chip select remains active, until this bit is reset. Then the page programming cycle starts. After being cleared, this bit must remain reset until the beginning of the programming sequence (WIP=1).
Bit 28	**SW:** *Software Mode* 0: Hardware mode: External memory is accessible with AHB transfers, transmit and receive registers are not accessible. (reset value) 1: Software mode : Transmit and receive registers are accessible. AHB read or write transfers to external memory is not allowed. Refer to *Section 6.4.4 on page 155* for more details. This bit must not be changed if a transfer is ongoing
Bits 27:24	Reserved, must be kept cleared.
Bits 23:16	**HOLD[7:0]:** *Clock Hold Period Selection* When programmed, this register stops the clock between each byte, while CS remains active (hold). 00h: SMI_CK is sent continuously (reset value) 01h: 1 SMI_CK period of hold between each byte ... FFh: 255 SMI_CK periods of hold between each byte
Bit 15	**FAST:** *Fast Read Mode Selection* 0: Normal read: opcode 03h + address + reception (reset value) 1: Fast read: opcode 0Bh + address + dummy byte + reception
Bits 14:8	**PRESC[6:0]:** *Prescaler value* 00h: f_{AHB} divided by 1 01h: f_{AHB} divided by 1 02h: f_{AHB} divided by 2 (reset value) .. 7Fh: f_{AHB} divided by 127 **Note**: The frequency is changed after the completion of the current transfer.

Bits 7:4	**TCS[3:0]:** *Deselect Time* When CS is deselected, it remains deselected for at least (DESELECT_TIME + 1) SMI_CK periods. The reset value (5h) corresponds to SMI_CK frequency limited to 20MHz at reset. So Tcs=6*50ns = 300 ns. **Note:** FAST and TCS have to be written at the same time as PRESC. They are all taken into account after the completion of the current transfer. Any check of the consistency between these three values has to be done by software.
Bits 3:0	**BE[3:0]:** *Bank Enable Bits* Reset value: 0001b in external memory boot mode or 0000 in other boot modes. 0 : Bank disabled 1 : Bank enabled **Notes:** At reset, if External Memory Boot mode is selected by the microcontrollers BOOT[1:0] pins, all banks are disabled except Bank 0. WEN, RSR and SEND commands are not sent if the bank selected by BS is disabled. These bits must not be changed if a transfer is ongoing.

6.5.2 SMI control register 2 (SMI_CR2)

Address Offset: 04h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved		BS		WEN	RSR	WCIE	TFIE	SEND	REC_LENGTH			reserved	TRA_LENGTH		
r	r	rw	rw	rs	rs	rw	rw	rs	rw	rw	rw	r	rw	rw	rw

Bits 31:14	Reserved, forced by hardware to 0.
Bits 13:12	**BS:** *Bank Select* Selects the Bank to be accessed. Only one bank can be accessed at a time. This value is latched at the beginning of the transfer. These bits must not be changed if SEND=1, WEN=1 or RSR=1. 00: Bank 0 (reset value) 01: Bank 1 10: Bank 2 11: Bank 3
Bit 11	**WEN:** *Write Enable Command* 0: No effect 1: Sends a Write Enable command to the selected memory bank (selected by BS bits). Then a write command can be sent to the memory. **Note:** This bit is cleared by hardware when the Write Enable command has been sent. This bit must not be used in order to send a write or an erase through SW mode.
Bit 10	**RSR:** *Read Status Register Command* 0: No effect 1: Send the READ STATUS REGISTER command to the selected bank (selected by BS[1:0] bits) and load the result in STATUS_REGISTER[7:0] **Note:** This bit is cleared by hardware when the Read Status command has been completed.
Bit 9	**WCIE:** *Write Complete Interrupt Enable* 0: Interrupt disabled 1: Generate an interrupt request when *Write Complete Flag* is set.
Bit 8	**TFIE:** *Transfer Finished Interrupt Enable* 0: Interrupt disabled 1: Generate an interrupt request when *Transfer Finished Flag* is set.
Bits 7	**SEND:** *Send Command* 0: No effect 1: Start a transfer in the format defined by TRANSMISSION_LENGTH[2:0] + RECEPTION_LENGTH[2:0]. **Note:** This bit is cleared by hardware only and can be set by software only if the SW bit in the SMI_CR1 register is set.

Bits 6:4	**RECEPTION LENGTH[2:0]:** *Reception Length* This value must be written by software to define the number of bytes to be received from external memory. This value is latched at the beginning of the software transfer. 000: 0 bytes (reset value) 001: 1 byte 010: 2 bytes 011: 3 bytes 1xx: 4 bytes
Bits 2:0	**TRANSMISSION LENGTH[2:0]:** *Transmission Length* This value must be written by software to define the number of bytes to be transmitted to external memory. This value is latched at the beginning of the software transfer. 000: 0 bytes (reset value) 001: 1 byte 010: 2 bytes 011: 3 bytes 1xx : 4 bytes

6.5.3 SMI status register (SMI_SR)

Address Offset: 08h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WM				ERF2	ERF1	WCF	TFF	Serial Memory Status Register							
r	r	r	r	rc0	rc0	rc0	rc0	r	r	r	r	r	r	r	r

Bits 31:16	Reserved, forced by hardware to 0.
Bits 15:12	**WM[3:0]:** *Write Mode for CS[3:0] banks* 0: Bank x not in Write Mode (Write request to this bank is finished. 1: Bank x in Write Mode (SMI_CR1 WEN bit has been sent to this bank). **Note:** This bit is not cleared by the instructions sent in SW mode.
Bit 11	**ERF2**: *Error Flag 2: Forbidden Write Request* 0: No error 1: Write request while corresponding WM bit is reset, or when size is changed or address is not incremented.
Bit 10	**ERF1:** *Error Flag 1: Forbidden Access* 0: No error 1: Read or write access requested on disabled bank, or while SW bit is set, or Read request while WBM is set.

Bit 9	**WCF:** *Write Complete Flag* 0: No Write Complete event 1: Write Complete. After a write instruction, a Read Status Register command is performed by hardware, this flag is set when the WIP bit of the SMI_SR Status Register is reset (meaning the end of programming). The WIP bit of the memory device Status Register has to be bit 0. **Note:** WCF is not set during a Boot phase or by the instructions sent in SW mode..
Bit 8	**TFF:** *Transfer Finished Flag* 0: No Transfer Finished event 1: The software transfer defined by TRANSMISSION-LENGTH + RECEPTION_LENGTH has completed, or the SMI_CR2 register RSR or WEN commands are finished.
Bits 7:0	**STATUS_REGISTER [7:0]:** *Memory Device Status Register* These bits are used to store a copy of the external memory status register. This register is updated in 2 ways: When the SMI_CR2 RSR bit is set, STATUS_REGISTER [7:0] is updated after the RSR sequence. After a write request to a memory bank, STATUS_REGISTER [7:0] is updated until the write cycle is finished (Bit 0 of SMI_SR is cleared)

6.5.4 SMI transmit register (SMI_TR)

Address Offset: 0Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Byte3								Byte2							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte1								Byte0							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**TR[31:0]:** *Transmit Register* This register is a barrel shifter. TR[7:0] is sent first and then 8 bits are shifted. **Notes:** This register can be written only in software mode when SW=1 and SEND=0. SMI_TR is also used in hardware mode. If the SMI is put in hardware mode the value is not kept

6.5.5 SMI receive register (SMI_RR)

Address Offset: 10h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Byte3								Byte2							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte1								Byte0							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0	**RR[31:0]:** *Receive Register* 1st received byte is placed in RR[7:0], then in RR[15:8], then in RR[23:16], then in RR[31:24]. **Notes:** In software mode, this register must be read after the TFF bit is set (otherwise the value is not valid). SMI_RR is also used in hardware mode. If the SMI is put in hardware mode the value is not kept

6.6 Register map

Table 33. SMI register map

Addr. Offset	Register Name	31																															0
00	SMI_CR1	Control Register 1																															
04	SMI_CR2	Control Register 2																															
08	SMI_SR	Status Register																															
0C	SMI_TR	Transmit Register																															
10	SMI_RR	Receive Register																															

Refer to *Table 1* for the register base address.

7 Real Time Clock (RTC)

7.1 Introduction

The Real Time Clock is an independent timer. The RTC provides a set of continuously-running counters which can be used, with suitable software, to provide a clock-calendar function. The counter values can be written to set the current time/date of the system.

7.2 Main features

- Programmable prescaler: division factor up to 2^{20}
- 32-bit programmable counter for long-term measurement
- Two separate clocks : PCLK for the APB interface and RTC clock (must be at least four times slower than the PCLK clock)
- Two separate reset types:
  - The APB interface is reset by system reset
  - The RTC kernel is reset only by a power-on if embedded regulators are enabled or by the NRSTIN pin if embedded regulators are disabled.
- Three dedicated maskable interrupt lines:
  - Alarm interrupt, for generating a software programmable alarm interrupt.
  - Seconds interrupt, for generating a periodic interrupt signal with a programmable period length (up to 1 second).
  - Overflow interrupt, to detect when the internal programmable counter rolls over to zero.

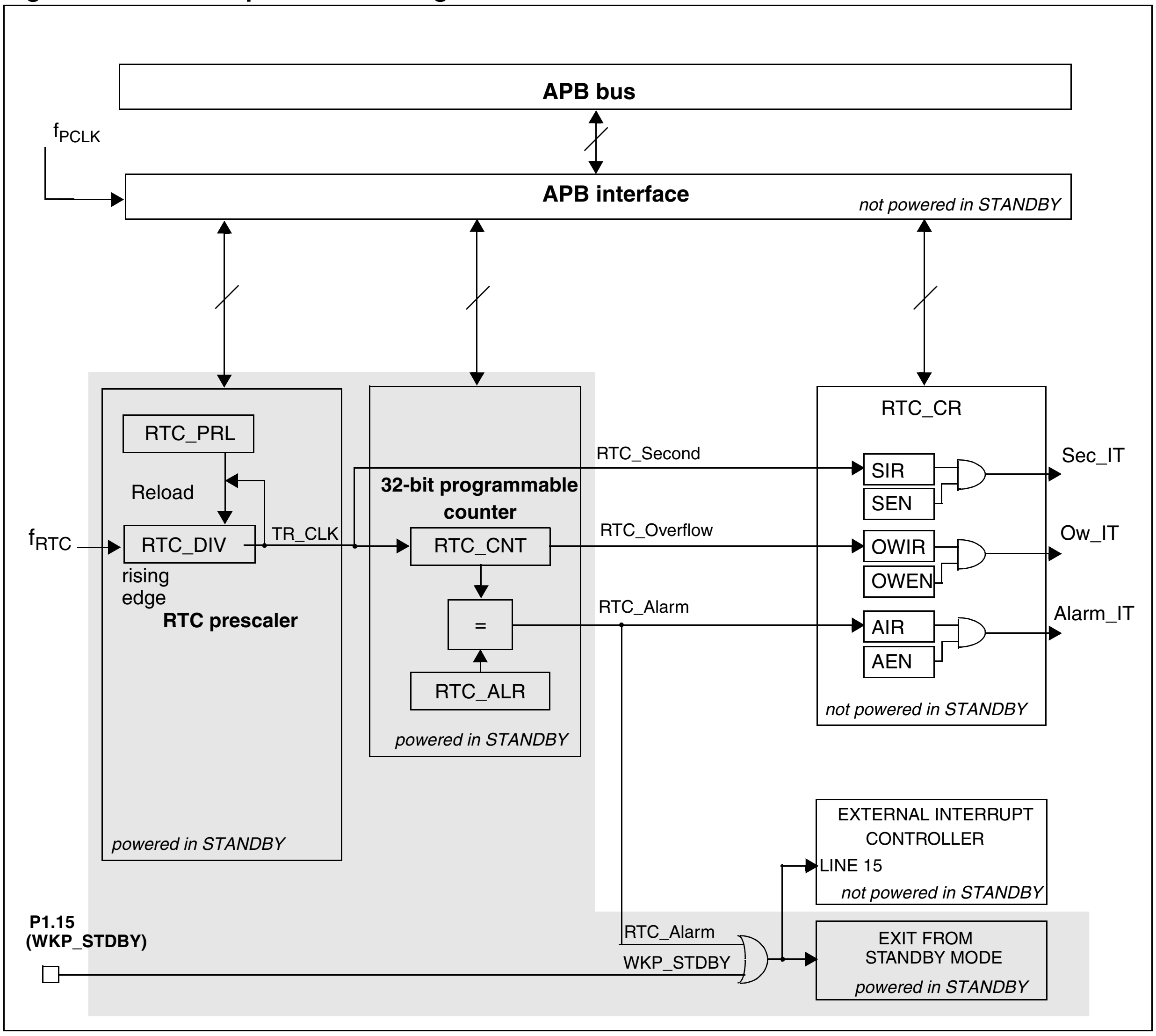
7.3 Functional description

7.3.1 Overview

The RTC consists of two main units (see *Figure 44 on page 168*). The first one (APB Interface) is used to interface with the APB bus. This unit also contains a set of 16-bit registers accessible from the APB bus in read or write mode (for more information refer to *Section 7.4: Register description on page 171*). The APB interface is clocked by the APB bus clock in order to interface with the APB bus.

The other unit (RTC Core) consists of a chain of programmable counters made of two main blocks. The first block is the RTC prescaler block, which generates the RTC time base TR_CLK that can be programmed to have a period of up to 1 second. It includes a 20-bit programmable divider (RTC Prescaler). Every TR_CLK period, the RTC generates an interrupt (SecInt) if it is enabled in the RTC_CR register. The second block is a 32-bit programmable counter that can be initialized to the current system time. The system time is incremented at the TR_CLK rate and compared with a programmable date (stored in the RTC_ALR register) in order to generate an alarm interrupt, if enabled in the RTC_CR control register.

Figure 44. RTC simplified block diagram



7.3.2 Reset procedure

All system registers are asynchronously reset by a System Reset, except RTC_ALR, RTC_CNT, RTC_DIV.

- **When the embedded regulators are enabled (pin VREG_DIS tied to ground)**, these registers and the Real Time Clock counter are reset only by a power on or RSM reset. This makes it possible to wake-up the microcontroller from STANDBY mode by means of an external reset, without resetting the RTC.
- **When the embedded regulators are disabled (pin VREG_DIS tied to 1)**, these registers and the Real Time Clock counter are reset by the NRSTIN pin.

7.3.3 Reading RTC registers

The RTC core is completely independant from the RTC APB interface.

Software accesses the RTC prescaler, counter and alarm values through the APB interface but the associated readable registers are internally updated at each rising edge of the RTC clock resynchronized by the RTC APB clock. This is also true for the RTC flags.

This means that the first read to the RTC APB registers may be corrupted (generally read as 0) if the APB interface has previously been disabled and the read occurs immediately after the APB interface is enabled but before the first internal update of the registers. This can occur if:

- A system reset has occurred
- The MCU has just woken up from STANDBY mode (see *Section 2.9.6: STANDBY mode*)
- The MCU has just woken up from STOP mode (see *Section 2.9.5: STOP mode*)
- The PCLK clock of the RTC has just been enabled using PCG mode (see *Section 2.9.3: PCG mode: Peripherals Clocks Gated mode*)

In all the above cases, the RTC core has been kept running while the APB interface was disabled (reset, not clocked or unpowered).

Consequently when reading the RTC registers, after having disabled the RTC APB interface, the software must first wait for the RSF bit (Register Synchronized Flag) in the RTC_CRL register to be set by hardware.

Note that the RTC APB interface is not affected by WFI low power mode.

7.3.4 Free-running mode

After power-on reset, the peripheral enters free-running mode. In this operating mode, the RTC Prescaler and the programmable counter start counting. Interrupt flags are activated too, but since interrupt signals are masked, there is no interrupt generation. Interrupt signals must be enabled by setting the appropriate bits in the RTC_CR register. In order to avoid spurious interrupt generation it is recommended to clear old interrupt requests before enabling them.

7.3.5 RTC flag assertion

The RTC Second Interrupt Request (SIR) is asserted at each RTC Core clock cycle before the update of the RTC Counter.

The RTC Overflow Interrupt Request (OWIR) is asserted at the last RTC Core clock cycle before the counter reaches the 0x0000 value.

The RTC_Alarm and RTC Alarm Interrupt Request (AIR) , see *Figure 45*, are asserted at the last RTC Core clock cycle before the counter reaches the RTC Alarm value stored in the Alarm register plus one (RTC_ALR + 1). To set the RTC Alarm value, you must be sure that this write is synchronized with the RTC Second flag. For this purpose two alternative methods can be used:

- Use RTC Alarm interrupt and and update the RTC Alarm and/or RTC Counter registers in the RTC interrupt service routine.
- Wait for SIR until it is set and then update the RTC Alarm and/or RTC Counter registers.

Note: If RTC interrupts are used during Run, Slow, or WFI modes the RTC clock must be at least 4 times slower than PCLK clock. However, it still possible to use the RTC alarm but through the EXTI interrupt (EXTI line 15).

Figure 45. RTC second and alarm waveform example with PR=0003, ALARM=00004

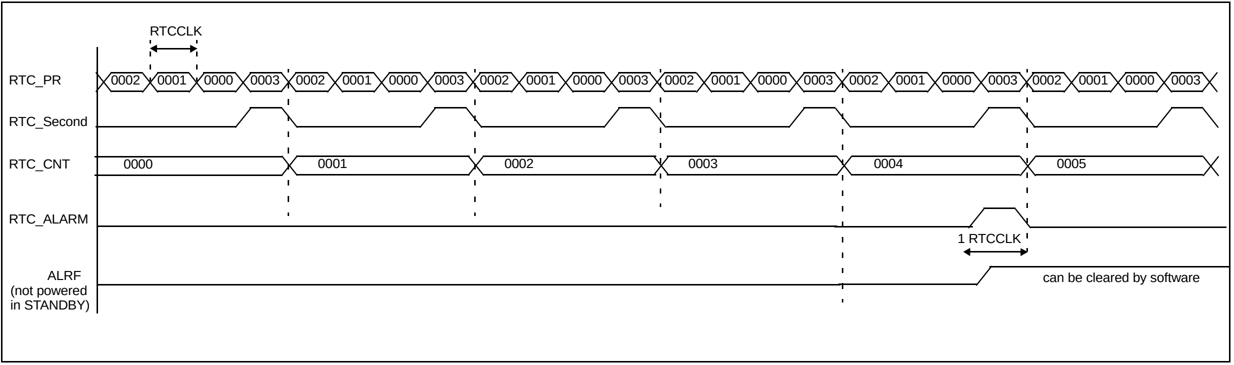
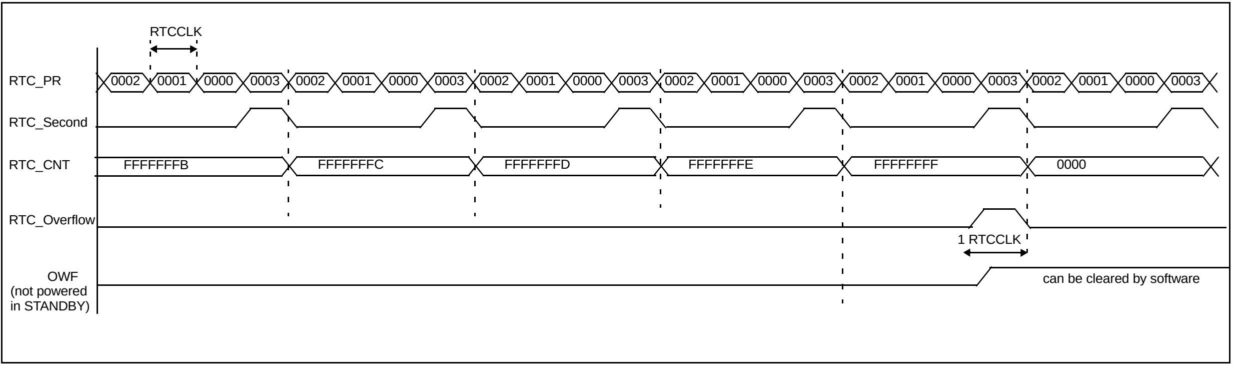


Figure 46. RTC Overflow waveform example with PR=0003



7.3.6 Configuration mode

To write in the RTC_PRL, RTC_CNT, RTC_ALR registers, the peripheral must enter Configuration Mode. This is done by setting the CNF bit in the RTC_CRL register.

In addition, writing to any RTC register is only enabled if the previous write operation is finished. To enable the software to detect this situation, the RTOFF status bit is provided in the RTC_CR register to indicate that an update of the registers is in progress. A new value can be written to the RTC registers only when the RTOFF status bit value is ’1’.

Configuration procedure:

1. Poll RTOFF, wait until its value goes to ‘1’
2. Set the CNF bit to enter configuration mode
3. Write to one or more RTC registers
4. Clear the CNF bit to exit configuration mode
5. Poll RTOFF, wait until its value goes to ‘1’ to check the end of the write operation.

The write operation only executes when the CNF bit is cleared; it takes at least three f_{RTC} cycles to complete.

7.4 Register description

The RTC registers cannot be accessed by bytes, but only by half-words or words. The reserved bits cannot be written and they are always read as ‘0’.

7.4.1 RTC Control Register High (RTC_CRH)

Address Offset: 00h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved													OWEN	AEN	SEN
-	-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw

Bits 15:3	Reserved, forced by hardware to 0.
Bit 2	**OWEN:** *Overflow Interrupt Enable* 0: Overflow interrupt is masked. 1: Overflow interrupt is enabled.
Bit 1	**AEN**: *Alarm Interrupt Enable* 0: Alarm interrupt is masked. 1: Alarm interrupt is enabled.
Bit 0	**SEN:** *Second Interrupt Enable* 0: Second interrupt is masked. 1: Second interrupt is enabled.

These bits are used to mask interrupt requests. Note that at reset all interrupts are disabled, so it is possible to write to the RTC registers to ensure that no interrupt requests are pending after initialization. It is not possible to write to the RTC_CRH register when the peripheral is completing a previous write operation (flagged by RTOFF=0, see *Section 7.3.6 on page 171*).

The RTC functions are controlled by this control register. Some bits must be written using a specific configuration procedure (see *Configuration procedure:*).

7.4.2 RTC Control Register Low (RTC_CRL)

Address Offset: 04h
Reset value: 0020h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										RTOFF	CNF	RSF	OWIR	AIR	SIR
-	-	-	-	-	-	-	-	-	-	r	rw	rc	rc	rc	rc

Bits 15:6	Reserved, forced by hardware to 0.
Bit 5	**RTOFF:** *RTC operation OFF* With this bit the RTC reports the status of the last write operation performed on its registers, indicating if it has been completed or not. If its value is ‘0’ then it is not possible to write to any of the RTC registers. This bit is read only. 0: Last write operation on RTC registers is still ongoing. 1: Last write operation on RTC registers terminated.
Bit 4	**CNF**: *Configuration Flag* This bit must be set by software to enter in configuration mode so as to allow new values to be written in the RTC_CNT, RTC_ALR or RTC_PRL registers. The write operation is only executed when the CNF bit is reset by software after has been set. 0: Exit configuration mode (start update of RTC registers). 1: Enter configuration mode.
Bit 3	**RSF:** *Registers Synchronized Flag* This bit is set by hardware at each time the RTC_CNT and RTC_DIV registers are updated and cleared by software. Before any read operation after an APB reset or an APB clock stop, this bit must be cleared by software, and the user application must wait until it is setted to be sure that the RTC_CNT, RTC_ALR or RTC_PRL registers are synchronized. 0: Registers not yet synchronized. 1: Registers synchronized.
Bit 2	**OWIR:** *Overflow Interrupt Request* This bit stores the status of periodic interrupt request signal (*Ow_IT*) generated by the overflow of the 32-bit programmable counter. When this bit is at ‘1’, the corresponding interrupt will be generated only if OWEN bit is set to ‘1’. The OWIR bit can be set at ‘1’ only by hardware and can be cleared only by software, while writing ‘1’ will leave it unchanged. 0: Overflow interrupt condition not met. 1: Overflow interrupt request pending.

Bit 1	**AIR:** *Alarm Interrupt Request* This bit contains the status of periodic interrupt request signal (*Alarm*) generated by the 32 bit programmable counter when the threshold set in RTC_ALR register is reached. When this bit is at ‘1’, the corresponding interrupt will be generated only if the AEN bit is set to ‘1’. The AIR bit can only be set to‘1’ by hardware and can be cleared by software only, while writing ‘1’ will leave it unchanged. 0: Alarm interrupt condition not met. 1: Alarm interrupt request pending.
Bit 0	**SIR:** *Second Interrupt Request* This bit contains the status of second interrupt request signal (*Sec_It*) generated by the overflow of the 20-bit programmable prescaler which increments the RTC counter. Hence this interrupt provides a periodic signal with a period corresponding to the resolution programmed for the RTC counter (usually one second). When this bit is at ‘1’, the corresponding interrupt will be generated only if the SEN bit is set to ‘1’. The SIR bit can be set to ‘1’ by hardware only and can be cleared by software only, while writing ‘1’ ill leave it unchanged. 0: Second’ interrupt condition not met. 1: Second interrupt request pending.

The functions of the RTC are controlled by this control register. It is not possible to write to the RTC_CR register while the peripheral is completing a previous write operation (flagged by RTOFF=0, see *Section 7.3.6 on page 171*).

Note: 1 Any interrupt request remains pending until the appropriate RTC_CR request bit is reset by software, indicating that the interrupt request has been granted.

2 At reset the interrupts are disabled, no interrupt requests are pending and it is possible to write to the RTC registers.

3 The OWIR, AIR, SIR and RSF bits are not updated when the APB clock is not running.

4 The OWIR, AIR, SIR and RSF bits can only be set by hardware and only cleared by software.

7.4.3 RTC Prescaler Load Register (RTC_PRLH / RTC_PRLL)

The Prescaler Load registers keep the period counting value of the RTC prescaler. They are write-protected by the RTOFF bit in the RTC_CR register, and a write operation is allowed if the RTOFF value is ‘1’.

RTC Prescaler Load Register High (RTC_PRLH)

Address Offset: 08h
Write only (see *Section 7.3.6 on page 171*)
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												PRL[19:16]			
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bits 15:4	Reserved, forced by hardware to 0.
Bits 3:0	**PRL[19:16]:** *RTC Prescaler Reload Value High* These bits are used to define the counter clock frequency according to the following formula: $f_{TR\_CLK} = f_{RTC}/(PRL[19:0]+1)$ **Caution:** The zero value is not recommended. RTC interrupts and flags cannot be asserted correctly.

RTC Prescaler Load Register Low (RTC_PRLL)

Address Offset: 0Ch
Write only (see *Section 7.3.6 on page 171*)
Reset value: 8000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRL[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**PRL[15:0]:** *RTC Prescaler Reload Value Low* These bits are used to define the counter clock frequency according to the follwing formula: $f_{TR\_CLK} = f_{RTC}/(PRL[19:0]+1)$

Note: If the input clock frequency (f_{RTC}) is 32.768 kHz, write 7FFFh in this register to get a signal period of 1 second.

7.4.4 RTC Prescaler Divider Register (RTC_DIVH / RTC_DIVL)

During each period of TR_CLK, the counter inside the RTC prescaler is reloaded with the value stored in the RTC_PRL register. To get an accurate time measurement it is possible to read the current value of the prescaler counter, stored in the RTC_DIV register, without stopping it. This register is read-only and it is reloaded by hardware after any change in the RTC_PRL or RTC_CNT registers.

RTC Prescaler Divider Register High (RTC_DIVH)

Address Offset: 10h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												RTC_DIV[19:16]			
-	-	-	-	-	-	-	-	-	-	-	-	r	r	r	r

Bits 15:4	Reserved
Bits 3:0	**RTC_DIV[19:16]:** *RTC Clock Divider High*

RTC Prescaler Divider Register Low (RTC_DIVL)

Address Offset: 14h
Reset value: 8000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_DIV[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0	**RTC_DIV[15:0]:** *RTC Clock Divider Low*

7.4.5 RTC Counter Register (RTC_CNTH / RTC_CNTL)

The RTC core has one 32-bit programmable counter, accessed through two 16-bit registers; the count rate is based on the TR_CLK time reference, generated by the prescaler. RTC_CNT registers keep the counting value of this counter. They are write-protected by bit RTOFF in the RTC_CR register, and a write operation is allowed if the RTOFF value is ‘1’. A write operation on the upper (RTC_CNTH) or lower (RTC_CNTL) registers directly loads the corresponding programmable counter and reloads the RTC Prescaler. When reading, the current value in the counter (system date) is returned.

RTC Counter Register High (RTC_CNTH)

Address Offset: 18h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_CNT[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**RTC_CNT[31:16]:** *RTC Counter High* Reading the RTC_CNTH register, the current value of the high part of the RTC Counter register is returned. To write to this register it is necessary to enter configuration mode using the RTOFF bit in the RTC_CR register.

RTC Counter Register Low (RTC_CNTL)

Address Offset: 1Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**RTC_CNT[15:0]:** *RTC Counter Low* Reading the RTC_CNTL register, the current value of the lower part of the RTC Counter register is returned. To write to this register it is necessary to enter configuration mode using the RTOFF bit in the RTC_CR register.

7.4.6 RTC Alarm Register High (RTC_ALRH / RTC_ALRL)

When the programmable counter reaches the 32-bit value stored in the RTC_ALR register, an alarm is triggered and the RTC_alarmIT interrupt request is generated. This register is write-protected by the RTOFF bit in the RTC_CR register, and a write operation is allowed if the RTOFF value is ‘1’.

RTC Alarm Register High (RTC_ALRH)

Address Offset: 20h
Write only (see *Section 7.3.6 on page 171*)
Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_ALR[31:16]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 15:0	**RTC_ALR[31:16]:** *RTC Alarm High* The high part of the alarm time is written by software in this register. To write to this register it is necessary to enter configuration mode using the RTOFF bit in the RTC_CR register.

RTC Alarm Register Low (RTC_ALRL)

Address Offset: 24h
Write only (see *Section 7.3.6 on page 171*)
Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_ALR[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 15:0	**RTC_ALR[15:0]:** *RTC Alarm Low* The low part of the alarm time is written by software in this register. To write to this register it is necessary to enter configuration mode using the RTOFF bit in the RTC_CR register.

7.5 RTC register map

RTC registers are mapped as 16-bit addressable registers as described in the table below:

Table 34. RTC register map

Address offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	RTC_CRH	---													OW EN	AEN	SEN
4	RTC_CRL	---										RT OFF	CNF	RSF	OWIR	AIR	SIR
8	RTC_PRLH	---												PRL			
Ch	RTC_PRLL	PRL															
10h	RTC_DIVH	---												DIV			
14h	RTC_DIVL	DIV															
18h	RTC_CNTH	CNTH															
1Ch	RTC_CNTL	CNTL															
20h	RTC_ALRH	ALARMH															
24h	RTC_ALRL	ALARML															

See *Table 2* for the base address

8 Watchdog Timer (WDG)

8.1 Introduction

The Watchdog Timer peripheral can be used as free-running timer or as Watchdog to resolve processor malfunctions due to hardware or software failures.

8.2 Main features

- 16-bit down Counter
- 8-bit clock Prescaler
- Safe Reload Sequence
- Free-running Timer mode
- End of Counting interrupt generation

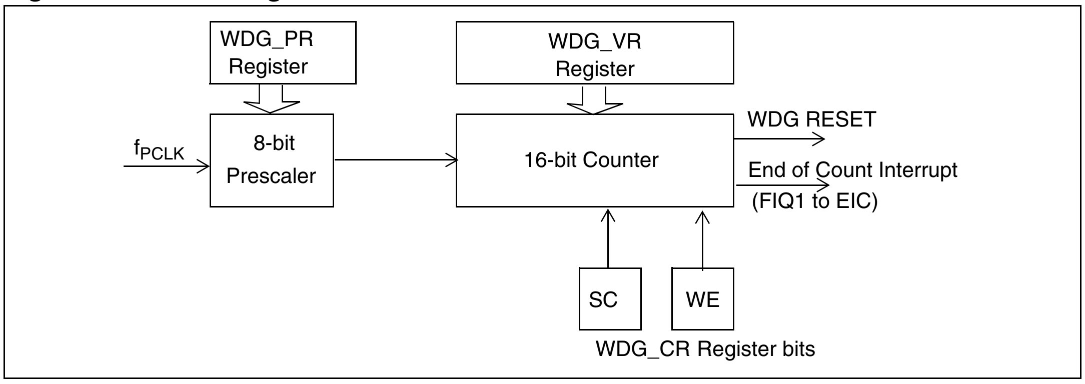
8.3 Functional description

Figure 47 shows the functional blocks of the Watchdog Timer module. The module can work as a Watchdog or as a Free-running Timer. In both modes the 16-bit Counter value can be accessed by reading the WDG_CNT register.

8.3.1 Free-running Timer mode

If the WE bit in the WDG_CR register is not set by software, the peripheral enters free-running timer mode.
In this operating mode, when the SC bit of WDG_CR register is written to ‘1’ the WDG_VR value is loaded in the Counter and the Counter starts counting down.

Figure 47. Watchdog Timer functional block



When it reaches the end of count value (0000h) an End of Count interrupt is generated (EC) and the WDG_VR value is re-loaded. The Counter runs until the SC bit is cleared. If the SC bit is set again, both the Counter and the Prescaler are re-loaded with the values contained in registers WDG_VR and WDG_PR respectively, so it does not restart from where it last stopped, but from a defined state without having to reset and re-program the module. On the other hand, it is not possible to change the prescaler factor on-the-fly since it will only effect the counter after a restart command (setting the SC bit, which generates a re-load operation).

8.3.2 Watchdog mode

If the WE bit of WDG_CR register is written to ‘1’ by software, the peripheral enters Watchdog mode. This operating mode can not be changed by software (the SC bit has no effect and WE bit cannot be cleared).
As the peripheral enters in this operating mode, the WDG_VR value is loaded in the Counter and the Counter starts counting down. When it reaches the end of count value (0000h) a system reset signal is generated (WDG RESET).

If a sequence of two consecutive values, 0xA55A and 0x5AA5, is written in the WDG_KR register see *Section 8.4*, the WDG_VR value is re-loaded in the Counter, the End of count can be prevented.

8.4 Register description

The Watchdog Timer registers can not be accessed by byte.
The reserved bits can not be written and they are always read at ‘0’.

8.4.1 WDG Control Register (WDG_CR)

Address Offset: 00h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														SC	WE
-														rw	rw

Bit 15:2	Reserved. Forced by hardware to 0.
Bit 1	**SC**: *Start Counting bit.* 0: The counter is stopped. 1: The counter loads the Timer pre-load value and starts counting These functions are permitted only in Timer Mode (WE bit = 0).
Bit 0	**WE**: *Watchdog Enable bit.* 0: Timer Mode is enabled 1: Watchdog Mode is enabled This bit can’t be reset by software. When WE bit is high, SC bit has no effect.

8.4.2 WDG Prescaler Register (WDG_PR)

Address Offset: 04h

Reset value: 00FFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
-								rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:8	Reserved. Forced by hardware to 0.
Bit 7:0	**PR[7:0]**: *Prescaler value* The clock to Timer Counter is divided by PR[7:0]+1. This value takes effect when Watchdog mode is enabled (WE bit is set) or the re-load sequence occurs or the Counter starts (SC) bit is set in Timer mode.

8.4.3 WDG Preload Value Register (WDG_VR)

Address Offset: 08h

Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TV15	TV14	TV13	TV12	TV11	TV10	TV9	TV8	TV7	TV6	TV5	TV4	TV3	TV2	TV1	TV0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:0	**TV[15:0]**: *Timer Pre-load Value* This value is loaded in the Timer Counter when it starts counting or a re-load sequence occurs or an End of Count is reached.

The time (µs) need to reach the end of count is given by:

$$\frac{(PR[7:0] + 1) * (TV[15:0] + 1) * t_{CLK}}{1000} \mu s$$

where t_{CLK} is the Clock period measured in ns.
i.e. if CLK = 20MHz the default time-out set after the system reset is:

256*65535*50/1000 = 838861µs.

8.4.4 WDG Counter Register (WDG_CNT)

Address Offset: 0Ch

Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0	**CNT[15:0]:** *Timer Counter Value* The current counting value of the 16-bit Counter is available reading this register.

8.4.5 WDG Status Register (WDG_SR)

Address Offset: 10h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved															EC
-															r-c

Bit 15:1	Reserved. Forced by hardware to 0.
Bit 0	**EC**: *End of Count pending bit* 0: no End of Count has occurred 1: the End of Count has occurred In Watchdog Mode (WE = 1) this bit has no effect. This bit can be set only by hardware and must be reset by software.

8.4.6 WDG Mask Register (WDG_MR)

Address Offset: 14h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved															ECM
-															rw

Bit 15:1	Reserved. Forced by hardware to 0.
Bit 0	**ECM**: *End of Count Mask bit* 0: End of Count interrupt request is disabled 1: End of Count interrupt request is enabled

8.4.7 WDG Key Register (WDG_KR)

Address Offset: 18h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
K15	K14	K13	K12	K11	K10	K9	K8	K7	K6	K5	K4	K3	K2	K1	K0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:0	**K[15:0]:** Key Value When Watchdog Mode is enabled, writing in this register two consecutive values (A55A, 5AA5) the Counter is initialized to TV[15:0] value and the Prescaler value in the WDG_PR register take effect. Any number of instructions can be executed between the two writes. If Watchdog Mode is disabled (WE = 0) a write to this register has no effect. This register returns the value 0000h when read.

8.5 WDG Register map

Table 35. Watchdog Timer Register map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WDG_CR	reserved														SC	WE
4	WDG_PR	reserved								PR(7:0)							
8	WDG_VR	TV(15:0)															
C	WDG_CNT	TV(15:0)															
10	WDG_SR	reserved															EC
14	WDG_MR	reserved															MEC
18	WDG_KR	K[15:0]															

See *Table 2* for the base address.

9 Timebase Timer (TB)

9.1 Introduction

The Timebase Timer can be used as free-running timer to generate a time base.

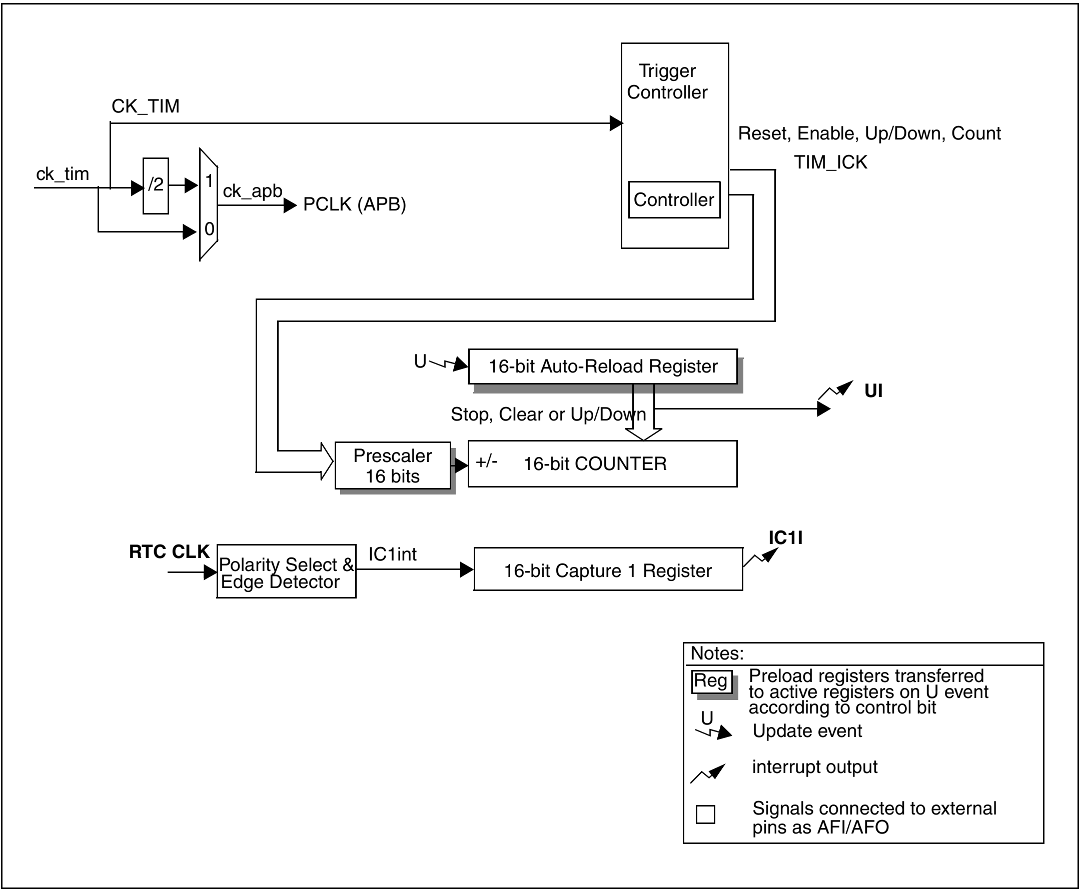
9.2 Synchronizable Programmable Timer (SPT) architecture

The TB Timer forms a family with the PWM timer and the TIM timers which are all based on a common architecture. This makes it easier to design applications using the various timers (identical register mapping, common basic features).

9.3 General block diagram

The general Block Diagram is shown in *Figure 48*.

Figure 48. General block diagram


Trigger Controller
CK_TIM
Reset, Enable, Up/Down, Count
TIM_ICK
ck_tim
/2
1
0
ck_apb
PCLK (APB)
Controller
U
16-bit Auto-Reload Register
UI
Stop, Clear or Up/Down
Prescaler 16 bits
+/-
16-bit COUNTER
IC1I
RTC CLK
Polarity Select & Edge Detector
IC1int
16-bit Capture 1 Register
Notes:
Reg
Preload registers transferred to active registers on U event according to control bit
U
Update event
interrupt output
Signals connected to external pins as AFI/AFO

9.4 General features

16-bit programmable prescaler

- Autoreload 16-bit up/down counter with update status flag and maskable interrupt.
- Timer Interrupts (with enable bits):
- UI: Timer Update interrupt.
- Selectable Timer clock frequency:
  - $f_{CK\_TIM} = f_{PCLK}$ or $f_{PCLK}x2$
  - f_{CK_RTC}

9.4.1 Counter clock selection

The counter clock can be provided by the following signals:

- CK_TIM internal clock
- External Clock (CK_RTC Clock)

The selected signal is fed to the counter through a 16-bit prescaler (TB_PSC). The original clock signal is divided by 1 to 65536.

Internal clock mode

The internal clock (CK_TIM) is selected as counter clock if the Slave mode controller is OFF: SME=0 in the TB_SCR register.

External clock mode (RTC Clock)

To use the RTC as clock source for the TB, proceed as follows:

1. Set the RTCM control bit in the MRCC_PWRCTRL register to connect the RTC clock input to the input capture of the TB timer.
2. Set SMS to ‘01’ and SME to ‘1’ in the TB_SCR register to select external clock mode.
3. The input signal IC1 to be used to clock the counter is selected by setting TS bits, in the same register, to ‘10’. the polarity of the input capture is defined by the IC1P bit in TB_IMCR register.

In any case, the counter is synchronized with the internal clock CK_TIM and it still must be enabled by writing the CNT_EN bit in the TB_CR register to ‘1’. When an external signal is used to clock the counter, at least four rising edges of CK_TIM must occur between two consecutive active edges of the external signal. Thus the external frequency must be less than a quarter of CK_TIM frequency.

9.5 Functional description

9.5.1 Counter

The Timer Base is based on a 16-bit up/down-counter (CNT) and its related control registers.

It can be cleared and enabled/disabled by software. Up-counting or down-counting can be controlled by software or by hardware.

Counter modes

There are three counter modes:

- **Up-counting mode**: The counter counts from 0 to the value in the TB_ARR register then restarts counting from 0 and generates an Update event. To configure the counter in this mode, write U/D = 0 and CMS = 00 in the TB_CR register.
- **Down-counting mode**: The counter counts from the value in the TB_ARR register down to 0 then is re-loaded with the TB_ARR value and generates an Update event. To configure the counter in this mode, write U/D = 1 and CMS = 00 in the TB_CR register.
- **Center-aligned mode**: The counter counts up from 0 to (TB_ARR-1), is re-loaded with the TB_ARR value and then counts down to 0 before restarting counting. An update event can be generated when the counter is re-loaded with TB_ARR and when it reaches 0. To configure the counter in this mode, write CMS = 01, 10 or 11 in the TB_CR register.

Table 36. Counter mode selection

Counter Mode	TB_CR Register CMS bits	TB_CR Register U/D bit
Up-Counting Mode	00	1
Down-Counting Mode	00	0
Center aligned Mode	10 01 11	-

In Center-aligned mode, the U/D bit changes automatically, indicating whether the counter is counting up or down.

Counter Initialization

You can re-initialize the counter (and the prescaler as well) writing a ‘1’ in the CNT_RST bit (which is automatically reset by hardware) in the TB_CR register. Then, depending on the CMS and U/D bits in the TB_CR register, the counter is either cleared (in up-counting or center-aligned mode) or re-loaded with the TB_ARR register value (in down-counting mode). It starts counting as soon as you write a ‘1’ in the CNT_EN bit.

Interrupt Request

When an update event occurs, the UI flag in the TB_ISR register is set depending on UFS bit status (TB_CR register). An interrupt is generated if the URS bit in the TB_RSR register and the URE bit in the TB_RER register are set. If this condition is false, the interrupt request remains pending until it becomes true.

Counter clock selection

The counter clock is provided by the CK_TIM internal clock.

The selected signal is fed to the counter through a 16-bit prescaler (TB_PSC). The original clock signal is divided by 1 to 65536.

The following figures describe the counter timings for different clock ratios.

Figure 49. Counter timing diagram, CK_TIM divided by 2 (TB_ARR=FFFF)

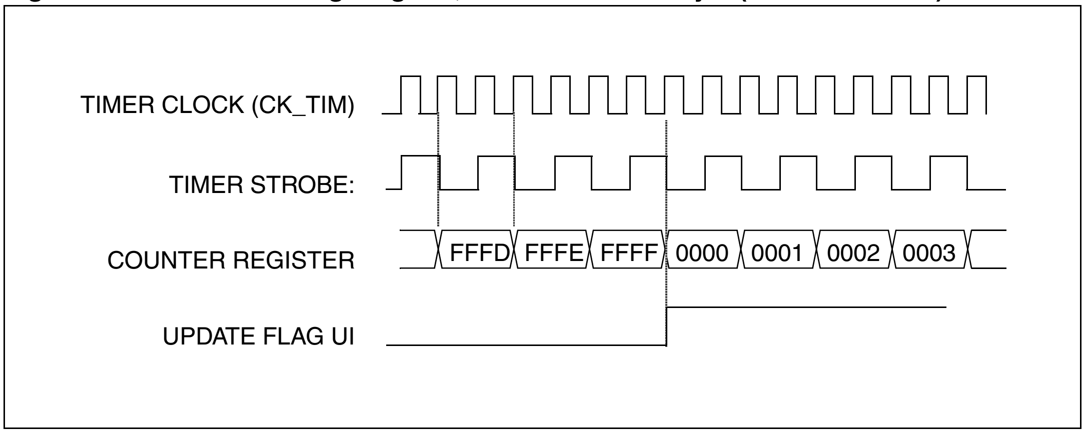


Figure 50. Counter timing diagram, CK_TIM divided by 4

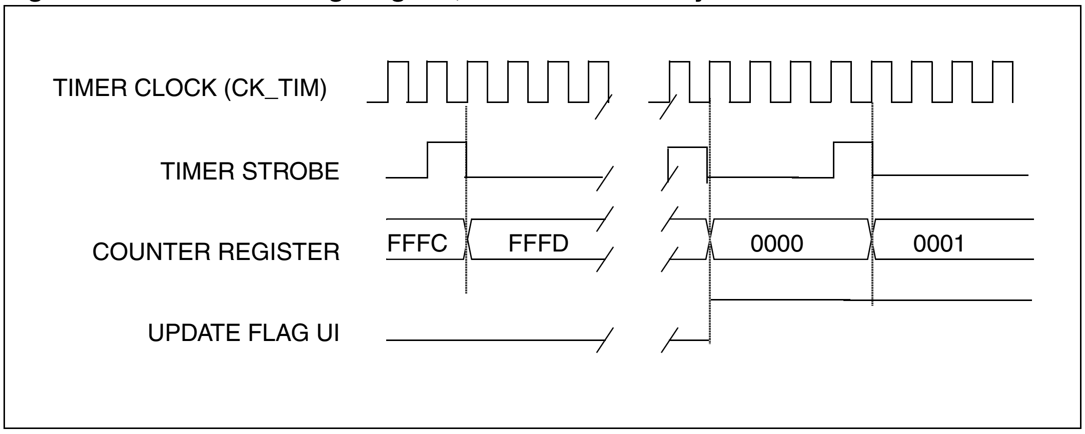
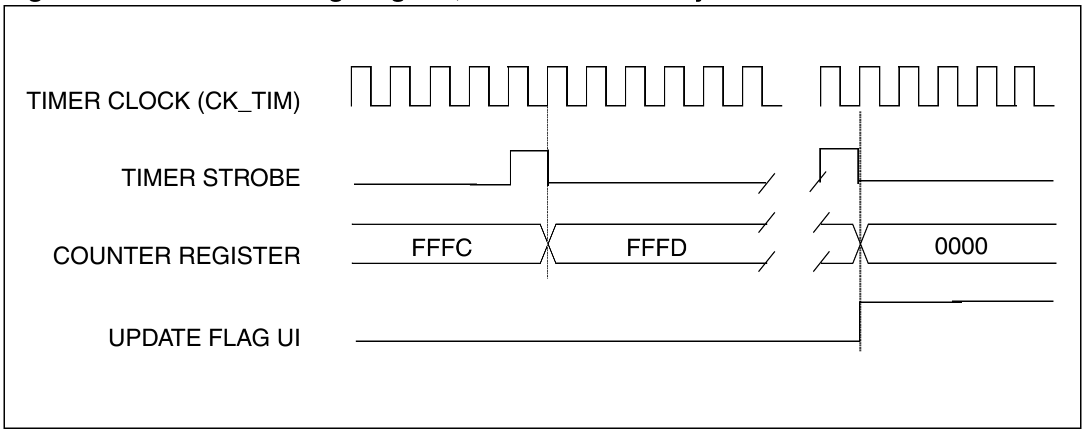


Figure 51. Counter timing diagram, CK_TIM divided by n



9.5.2 Interrupt management

To use the interrupt features, for each interrupt channel used, perform the following sequence:

- Set the desired ICRS and/or URS bits to select interrupt request.
- Set the desired ICRE and/or URE bits in the TB_RER register to enable interrupt requests.

9.5.3 Debug mode

When the microcontroller enters debug mode, the timer either continues to work normally or stops its activity, depending on the DBGC bit in TB_CR register.

If DBGC=0 (reset state), the timer detects any activity on the debug acknowledge (DBGACK) signal which is directly sent by the CPU. Then, as soon as the device enters debug mode:

- The TB_CNT counter stops counting
- The input capture is blocked

Thus, when a breakpoint occurs, the state of all the registers is kept stable.

If DBGC=1, the timer ignores the DBGACK signal.

9.6 Register description

The registers can only accessed as 16-bit data. A byte cannot be read or written. In this section, the following abbreviations are used:

read/write (rw)	Software can read and write to these bits.
read-only (r)	Software can only read these bits.
read/clear (rc_w0)	Software can read as well as clear this bit by writing 0. Writing ‘1’ has no effect on the bit value.
Write once-only (wo)	Software can read these bits. It can write them only once. Once write-accessed, these bits cannot be re-written unless the processor is reset.

9.6.1 Control Register (TB_CR)

Address Offset: 00h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res.					DBGC	reserved			CMS		U/D	OPM	CNT_ EN	CNT_ RST	UFS
					rw				rw	rw	rw	rw	rw	rw	rw

Bits 15:11	Reserved: must be kept cleared.
Bit 10	**DBGC**: *Debug Control* See *Section 9.5.3: Debug mode on page 189* for a detailed description. 0: The timer is stopped in debug mode 1: The timer continues working in debug mode
Bits 9:7	Reserved: must be kept cleared.
Bits 6:5	**CMS**: *Center-Aligned Mode Selection* Do not switch from edge-aligned mode to center-aligned mode while the counter is enabled (CNT_EN=1) 00: Edge-aligned mode 01: Center-aligned mode 10: Center-aligned mode 11: Center-aligned mode
Bit 4	**U/D**: *Up/Down* 0: Counter used as up-counter 1: Counter used as down-counter
Bit 3	Reserved: must be kept cleared.
Bit 2	**CNT_EN**: *Counter Enable* 0: Counter disabled 1: Counter enabled
Bit 1	**CNT_RST**: *Counter Reset* This bit can be written to 1 to reset the counter, it is automatically reset 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 (however the prescaler preload value is not affected). The counter is cleared if center-aligned mode is selected or if U/D=0 (up-counting), else it takes the auto-reload value (TB_ARR) if U/D=1 (down-counting)
Bit 0	**UFS**: *Update Flag Selection* 0: UI flag is set when an update event occurs, whatever its source (counter overflow/underflow, CNT RST bit set by software) 1: UI flag is set only if a counter overflow/underflow is detected. It is not affected by updates generated by software)

9.6.2 Synchro Control Register (TB_SCR)

Address Offset: 04h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved						TS		reserved			SMS		SME	reserved	
						rw	rw				rw	rw	rw		

Bits 15:10	Reserved: must be kept cleared.
Bits 9:8	**TS[1:0]**: *Trigger Selection* Selects the trigger signal (TRG) to be used by the trigger controller. 00: Internal trigger (output of the internal trigger selection) 10: Input Capture 1 (IC1)
Bits 7:5	Reserved: must be kept cleared.
Bits 4:3	**SMS[1:0]**: *Slave Mode Selection* When external signals are selected, the active edge of the trigger signal (TRG) is linked to the polarity selected on the external input (see TB_IMCR and TB_CR Register description) 00: Reset - rising edge of the selected trigger signal (TRG) resets the counter and generates an update of the registers.
Bit 2	**SME**: *Slave Mode Enable* 0: Slave mode disabled. If CNT_EN = ‘1’ then the prescaler is clocked directly by CK_TIM. 1: Slave mode enabled. The counter is controlled by the selected trigger input.
Bits 1:0	Reserved: must be kept cleared.

9.6.3 Input Mode Control Register (TB_IMCR)

Address Offset: 08h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												IC1P	IC1E	reserved	
												rw	rw		

Bits 15:4	Reserved: must be kept cleared.
Bit 3	**IC1P**: *IC1 Polarity* This bit selects whether IC1 or $\overline{\text{IC1}}$ is used for trigger or capture operations. 0: non-inverted: capture is done on a rising edge of IC1. When used as external trigger, IC1 is non-inverted. 1: inverted: capture is done on a falling edge of IC1. When used as external trigger, IC1 is inverted.
Bit 2	**IC1E**: *IC1 Enable* This bit determines if a capture of the counter value into Input Capture Register 1 (TB_ICR1) is enabled or not. 0: Capture disable. 1: Capture enable if IC1I flag (TB_ISR register) is reset.
Bit 1:0	Reserved: must be kept cleared.

9.6.4 Request Selection Register (TB_RSR)

Address Offset: 18h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													IC1RS	res.	URS
													rw		rw

Bit 15:3	Reserved: must be kept cleared.
Bit 2	**IC1RS**: *IC1 Request Selection* 0: IC1 interrupt request not selected. 1: IC1 interrupt request selected.
Bit 1	Reserved: must be kept cleared.
Bit 0	**URS**: *Update Request Selection* 0: Update interrupt request not selected. 1: Update interrupt request selected.

9.6.5 Request Enable Register (TB_RER)

Address Offset: 1Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								reserved					IC1RE	res.	URE
													rw		rw

Bit 15:3	Reserved: must be kept cleared.
Bit 2	**IC1RE**: *IC1 Interrupt Request Enable* 0: IC1 interrupt request disabled. 1: IC1 interrupt request enabled. A request is generated if the IC1I flag is set.
Bit 1	Reserved: must be kept cleared.
Bit 0	**URE**: *Update Interrupt Request Enable* 0: Update interrupt request disabled. 1: Update interrupt request enabled. A request is generated if the UI flag is set.

9.6.6 Interrupt Status Register (TB_ISR)

Address Offset: 20h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													IC1I	res.	UI
													rc_w0		rc_w0

Bit 15:3	Reserved: must be kept cleared.
Bit 2	**IC1I**: *IC1 Interrupt Flag* This bit is set by hardware on a capture. It is cleared by software. 0: No input capture occurred. 1: The counter value has been captured in TB_ICR1 register (An edge has been detected on IC1 which matches the selected polarity).
Bit 1	Reserved: must be kept cleared.
Bit 0	**UI**: *Update Interrupt Flag* This bit is set by hardware when an Update event occurs (depending on UFS bit in TB_CR register, please refer to *Section 9.6.1: Control Register (TB_CR) on page 190*. It is cleared by software. 0: No update occurred. 1: Update interrupt pending. This bit is set by hardware when the counter is updated: – At overflow or underflow – When CNT is cleared by software using the CNT_RST bit in TB_CR register (only if UFS=0 in TB_CR register)

Note: When CK_TIM and PCLK have different frequencies, the events which set the flags are generated on CK_TIM and then re-synchronized with the PCLK clock domain (the flag itself is related to PCLK). This means that there is a delay between the actual event and the setting of the flag. If an event is generated by software, for example writing the CNT_RST bit in the TB_CR register to generate an update event, there will be a delay before the related flag can be read at '1'. Therefore a minimum delay of 4 PCLK periods must be inserted before trying to clear this flag after generating the event (for instance by adding 2 dummy accesses).

9.6.7 Counter Register (TB_CNT)

Address Offset: 24h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the contents of the counter.

Note: 1 The counter doesn't count if the ARR value is 0. No update event is generated in this case.

2 If the counter is written with a value higher than the ARR value while it is up-counting (U/D=0 in TB_CR register), then the counter will count up to FFFF and restart counting from 0 up to ARR.

3 When CK_TIM and PCLK have different frequencies:

- The counter value cannot be accessed directly and is read through a buffer register. The buffer is updated once every 4 CK_TIM clock cycles (max.).
- The CNT_EN bit is not taken into account immediately by the counter. So if the counter is running when you clear the CNT_EN bit, you must insert an additional instruction before reading the counter to get the updated value.
- For the same reason, you must insert a delay before reading the counter after setting the CNT_RST bit.

9.6.8 Prescaler Register (TB_PSC)

Address Offset: 28h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This value divides the frequency of the internal timer clock (TIM_ICK) output by the clock mode controller. The counter clock frequency is f_{TIM_ICK} / (PSC[15:0]+1).
This register contains the preload value which is loaded in the active prescaler register:

- Permanently if the counter is disabled (CNT_EN = 0 in TB_CR register)
- Else at each update event (including when the counter is cleared using the CNT RST bit (TB_CR register) or by the slave mode control block when configured in reset mode - SMS bits = 00 in the TB_SCR register).

9.6.9 Auto-Reload Register (TB_ARR)

Address Offset: 30h
Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the preload value to be loaded in the active auto-reload register at the next update event.

However the active register can be directly written by software in the following cases:

- if the counter is not running (CNT EN=0 in TB_CR register),
- if the counter is running in down-counting mode (CMS='00' and U/D=1 in TB_CR register)

So care must be taken when switching from up-counting to another mode as the preload and the active registers might be different.

The auto-reload register is a 16-bit register which contains the ARR value to be compared to the counter CNT.

If the counter is up-counting in edge-aligned mode, when it reaches the ARR value an Update event is generated. The counter restarts up-counting from 0.

If the counter is up-counting in center-aligned mode, when it reaches the ARR value an update event is generated. The counter restarts down-counting from the ARR preload value.

If the counter is down-counting in edge-aligned mode, when it reaches 0 an update event is generated. The counter restarts down-counting from the ARR preload value.

If the counter is down-counting in center-aligned mode, when it reaches 0 an update event is generated. The counter restarts up-counting from 0.

9.6.10 Input Capture Register 1 (TB_ICR1)

Address Offset: 4Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

This 16-bit register contains the counter value captured when an input capture 1 event (IC1) occurs.

It can be read only when the IC1I flag is set in the TB_ISR register.

9.7 TB Register map

Table 37. TB Timer Register map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	TB_CR	reserved											U/D	OPM	CNT EN	CNT RST	UFS
4	TB_SCR	reserved															
8	TB_IMCR	reserved												IC1P	IC1E	res	IC1S
18	TB_RSR	reserved													IC1RS	res.	URS
1C	TB_RER	reserved													IC1RE	res.	URE
20	TB_ISR	reserved													IC1I	res.	UI
24	TB_CNT	Counter Value (CNT15-CNT0)															
28	TB_PSC	Prescaler (PSC15-PSC0)															
30	TB_ARR	Auto-Reload (ARR15-ARR0)															
4C	TB_ICR1	Input Capture 1															

10 Synchronizable Standard Timer (TIM)

10.1 Introduction

The timer consists of a 16-bit counter driven by a programmable prescaler.

It may be used for a variety of purposes, including pulse length measurement of up of two input signals (input capture) or generation of up to two output waveforms (output compare and PWM). Depending on specific device implementation DMA capability can be available for reduced CPU overload.

Pulse lengths waveform periods can be modulated from a very wide range using the timer prescaler.

10.2 Synchronizable Programmable Timer (SPT) architecture

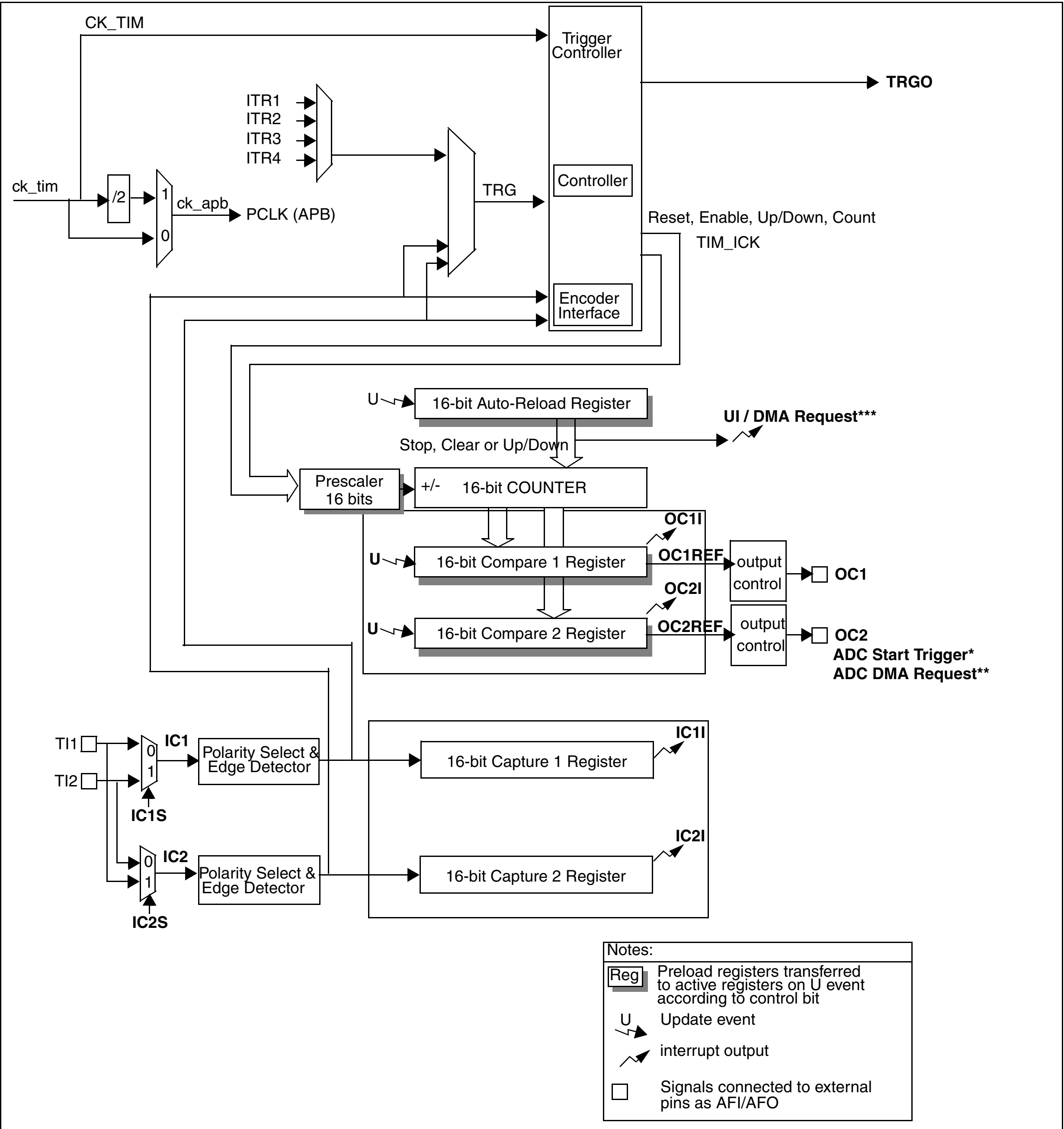
The TIM Timers form a family with the TB timer and the PWM timers which are all based on a common architecture. This makes it easier to design applications using the various timers (identical register mapping, common basic features).

In addition the TIM and PWM timers are linked together and can be synchronized.

10.3 General block diagram

The general Block Diagram is shown in *Figure 52*.

Figure 52. General block diagram


CK_TIM
Trigger Controller
TRGO
ITR1
ITR2
ITR3
ITR4
ck_tim
/2
1
0
ck_apb
PCLK (APB)
TRG
Controller
Reset, Enable, Up/Down, Count
TIM_ICK
Encoder Interface
U
16-bit Auto-Reload Register
UI / DMA Request***
Stop, Clear or Up/Down
Prescaler 16 bits
+/- 16-bit COUNTER
OC1I
U
16-bit Compare 1 Register
OC1REF
output control
OC1
OC2I
U
16-bit Compare 2 Register
OC2REF
output control
OC2
ADC Start Trigger*
ADC DMA Request**
TI1
TI2
IC1
0
1
IC1S
Polarity Select & Edge Detector
IC1I
16-bit Capture 1 Register
IC2
0
1
IC2S
Polarity Select & Edge Detector
IC2I
16-bit Capture 2 Register
Notes:
Reg
Preload registers transferred to active registers on U event according to control bit
U
Update event
interrupt output
Signals connected to external pins as AFI/AFO

***ADC Conversion start triggerable by TIM0 OC2**
****ADC DMA controllable by TIM2 OC2**
*****Independent DMA Request generation on TIM0 asserted on update Event**

10.4 General features

16-bit programmable prescaler : f_{CK_TIM} divided by 1 to 65536.

- Autoreload 16-bit up/down counter with update status flag and maskable interrupt.
- Synchronization using the Timer Link System: the timers can be configured as Master Timer or as Slave Timer. In slave mode operations like reset, clocking, start and stop can be controlled by another timer in master mode or by external events using TI1 or TI2 inputs. A programmable edge detector is associated with each external input.
- PWM input mode: external PWM can be injected on the Input Capture to measure external PWM timings (using input pins TI1 or TI2).
- Input pins TI1 and TI2 are usable as an incremental encoder interface for position sensor
- Timer Interrupts (with enable bits):
- UI: Timer Update interrupt.
- Independent DMA Request Generation (with enable bits) on TIM0, asserted on Update Event
- Selectable Timer clock frequency $f_{CK\_TIM} = f_{PCLK}$ or f_{PCLK}x2
- Input Capture functions
- Output compare/PWM functions
- One pulse mode (OPM)
- PWM Mode:
  - Shadow Registers can be enabled by control bits.
  - Full modulation capability (0 and 100% duty cycle), edge or center-aligned patterns
  - Dedicated interrupt for PWM duty cycles updating (PWM Update Interrupt)
  - The output polarity is programmable channel by channel.
  - Edge-aligned pattern or center-aligned pattern (counter or down-counter mode)

10.5 Functional description

10.5.1 Counter

Each timer is based on a 16-bit up/down-counter (CNT) and its related control registers.

It can be cleared and enabled/disabled by software or by a trigger input signal (refer to *Section 9.5.2 on page 189*). Up-counting or down-counting can be controlled by software or by hardware.

Counter modes

There are three counter modes:

- **Up-counting mode**: The counter counts from 0 to the value in the TIM_ARR register then restarts counting from 0 and generates an Update event. To configure the counter in this mode, write U/D = 0 and CMS = 00 in the TIM_CR register.
- **Down-counting mode**: The counter counts from the value in the TIM_ARR register down to 0 then is re-loaded with the TIM_ARR value and generates an Update event.

To configure the counter in this mode, write U/D = 1 and CMS = 00 in the TIM_CR register.

- **Center-aligned mode**: The counter counts up from 0 to (TIM_ARR-1), is re-loaded with the TIM_ARR value and then counts down to 0 before restarting counting. An update event can be generated when the counter is re-loaded with TIM_ARR and when it reaches 0. To configure the counter in this mode, write CMS = 01, 10 or 11 in the TIM_CR register.

Table 38. Counter Mode selection

Counter Mode	TIM_CR Register CMS bits	TIM_CR Register U/D bit
Up-Counting Mode	00	1
Down-Counting Mode	00	0
Center aligned Mode	10 01 11	-

In Center-aligned mode or in Encoder Interface mode (see *Section 10.5.3 on page 203*), the U/D bit changes automatically, indicating whether the counter is counting up or down.

Counter Initialization

You can re-initialize the counter (and the prescaler as well) writing a ‘1’ in the CNT_RST bit (which is automatically reset by hardware) in the TIM_CR register. Then, depending on the CMS and U/D bits in the TIM_CR register, the counter is either cleared (in up-counting or center-aligned mode) or re-loaded with the TIM_ARR register value (in down-counting mode). It starts counting as soon as you write a ‘1’ in the CNT_EN bit or if CNT_EN goes high under the control of the Slave Mode block (refer to *Section 10.6 on page 213*).

Interrupt/DMA Requests

When an update event occurs, the UI flag in the TIM_ISR register is set depending on UFS bit status (TIM_CR register). An interrupt is generated if the URS bit in the TIM_RSR register and the URE bit in the TIM_RER register are set. If this condition is false, the interrupt request remains pending until it becomes true. A DMA request, on TIM0, is generated if URS bit in TIM_RSR register is reset and URE bit in TIM_RER register is set. If this condition is false, the request remains pending to be issued as soon as it becomes true.

10.5.2 Counter clock selection

The counter clock can be provided by the following signals:

- CK_TIM internal clock
- Internal trigger inputs (ITRx)
- External input pins TI1 or TI2

The selected signal is fed to the counter through a 16-bit prescaler (TIM_PSC). The original clock signal is divided by 1 to 65536.

Figure 53, *Figure 54*, and *Figure 55* describe the counter timings for different clock ratios.

Figure 53. Counter timing diagram, CK_TIM divided by 2 (TIM_ARR=FFFF)

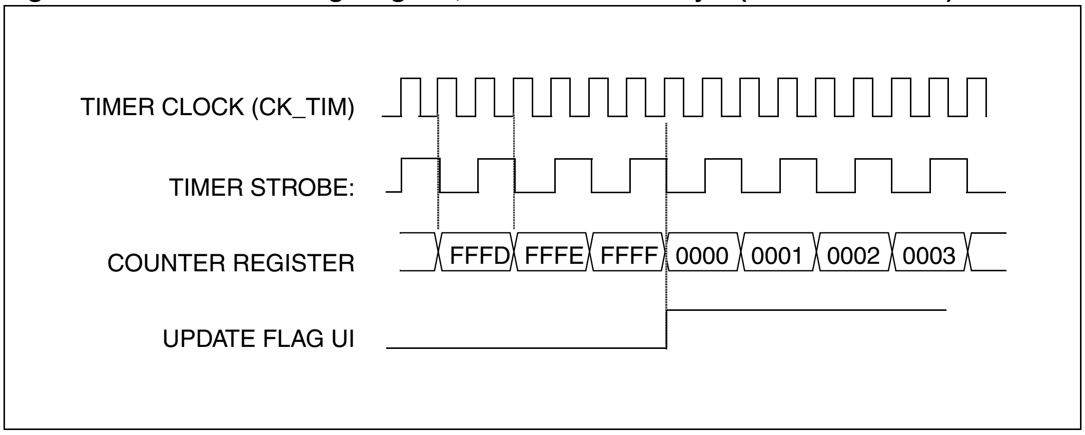


Figure 54. Counter timing diagram, CK_TIM divided by 4

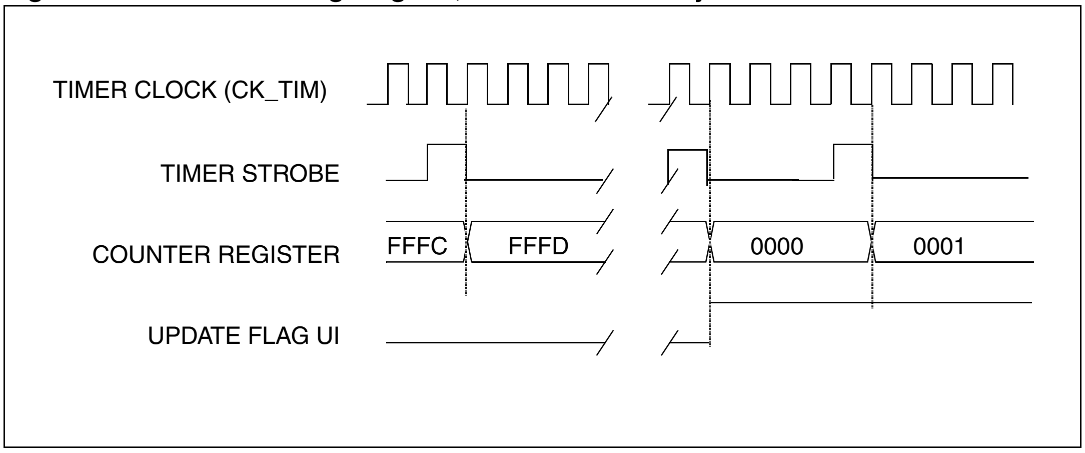
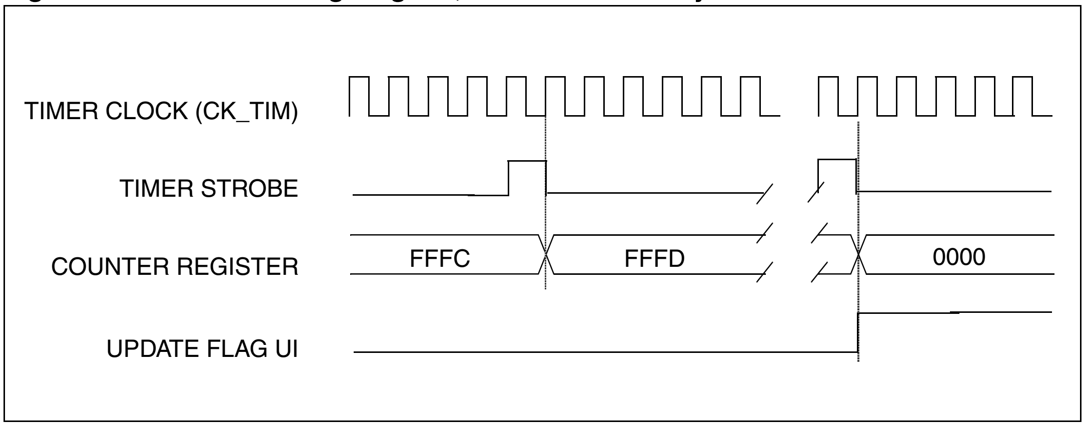


Figure 55. Counter timing diagram, CK_TIM divided by n



Internal Clock mode

The internal clock (CK_TIM) is selected as counter clock if the Slave mode controller is OFF: SME=0 in the TIM_SCR register.

External Clock mode (TI1, TI2)

External clock mode is selected when SMS=01 and SME=1 in the TIM_SCR register.

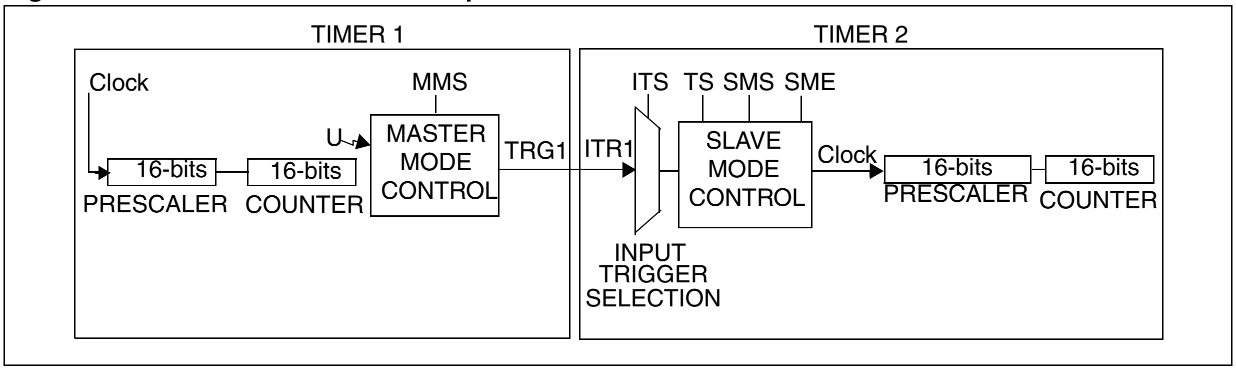
The input signal IC1 or IC2 to be used to clock the counter is selected by the TS bits in the same register.

- TI1: Redirect TI1 to the suitable ICx then select this ICx with the expected active edge.
  - e.g.: IC1 is mapped on TI1 (IC1S=0 in TIM_IMCR), IC1 is selected as external clock input (TS=10 and SMS=01 and SME=1 in TIM_SCR). TI1 is active on rising edge if IC1P=0 in TIM_IMCR, on falling edge if IC1P=1.
- TI2: Same as TI1.

In any case, the counter is synchronized with the internal clock CK_TIM and it still must be enabled by writing the CNT_EN bit in the TIM_CR register to ‘1’. When an external signal is used to clock the counter, at least four rising edges of CK_TIM must occur between two consecutive active edges of the external signal. Thus the external frequency must be less than a quarter of CK_TIM frequency to be properly used.

Using one timer as prescaler for another timer

Figure 56. Master/Slave timer example



For example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to *Figure 56*. To do this:

- Configure Timer 1 in master mode so that it outputs a periodic trigger signal on each Counter Update event. If you write MMS=10 in the TIM1_CR register, a rising edge will be output on TRG1 each time a Counter Update event is generated. Refer to *Figure 56*.
- To connect the TRG1 output of Timer1 to the Timer 2, Timer 2 must be configured in slave mode using ITR1 as internal trigger.
  You select this through the ITS bits in the TIM2_SCR register (writing ITS=001).
  Then you select internal trigger as input to the Slave mode controller (write TS=00 in the TIM2_SCR register).
  Slave mode selection must be put in External clock mode (write SMS = 01, SME = 1 in the TIM2_SCR register). This will cause the Timer 2 to be clocked by the rising edge of the periodic Timer 1 trigger signal.
- Finally both timers must be enabled by setting their respective CNT_EN bits (TIM_CR register).

Note: If OC1 is selected on Timer 1 as trigger output (MMS=11), its rising edge is used to clock the counter of Timer 2.

10.5.3 Encoder Interface Mode

Encoders are used in position and speed detection of rotating motion systems. Encoders enable closed loop control of many motor control applications.

To select Encoder interface mode write:

- ENC='01' in the TIM_SCR register if the counter is counting on IC1 edges only
- ENC='10' if it is counting on IC2 only
- ENC='11' if it is counting on both IC1 and IC2 edges

Select the IC1 and IC2 polarity by programming the IC1P and IC2P bits in the TIM_IMCR register.

The two inputs TI1 and TI2 are used to interface to an incremental encoder.

The counter is clocked by each valid transition on IC1_Int or IC2_Int assuming that it is enabled (CNT_EN bit in TIM_CR register written to '1').

The sequence of transitions of the two input signals is evaluated and generates count pulses as well as the direction signal.

Depending on the sequence, the counter counts up or down, the U/D bit in the TIM_CR register is modified by hardware accordingly.

The U/D direction bit is calculated at each transition on any input (IC1 or IC2), whether the counter is counting on IC1 only, IC2 only or both IC1 and IC2.

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 TIM_ARR register (0 up to TIM_ARR or TIM_ARR down to 0 depending on the direction). So you must configure TIM_ARR before starting. In the same way capture, compare, prescaler, trigger output features continue to work as normal. Encoder interface mode can be used in combination with gated or trigger mode. However, Encoder interface mode and external clock mode 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 current position. The count direction corresponds to the rotation direction of the connected sensor. The table below summarizes the possible combinations, assuming IC1 and IC2 don’t switch at the same time.

Table 39. Counting direction versus encoder signals

Active edges	Level on opposite input	IC1_Int signal		IC2_Int signal	
		Rising	Falling	Rising	Falling
Counting on IC1 only	High	Down	Up	No Count	No count
	Low	Up	Down	No count	No count
Counting on IC2 only	High	No Count	No count	Up	Down
	Low	No count	No count	Down	Up
Counting on IC1 and IC2	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 indicates the mechanical zero position, may be connected to an external interrupt input and trigger a counter reset.

The *Figure 57* 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:

- IC1S='0' (TIM_IMCR register, IC1 mapped on TI1)
- IC2S='0' (TIM_IMCR register, IC2 mapped on TI2)
- IC1P='0' (TIM_IMCR register, IC1 non-inverted. IC1_Int=IC1)
- IC2P='0' (TIM_IMCR register, IC2 non-inverted. IC2_Int=IC2)
- ENC='11' (TIM_SCR register, IC1_Int and IC2_Int are active on both rising and falling edges).
- CNT_EN = '1' (TIM_CR register, Counter is enabled)

Figure 57. Example of counter operation in Encoder Interface mode

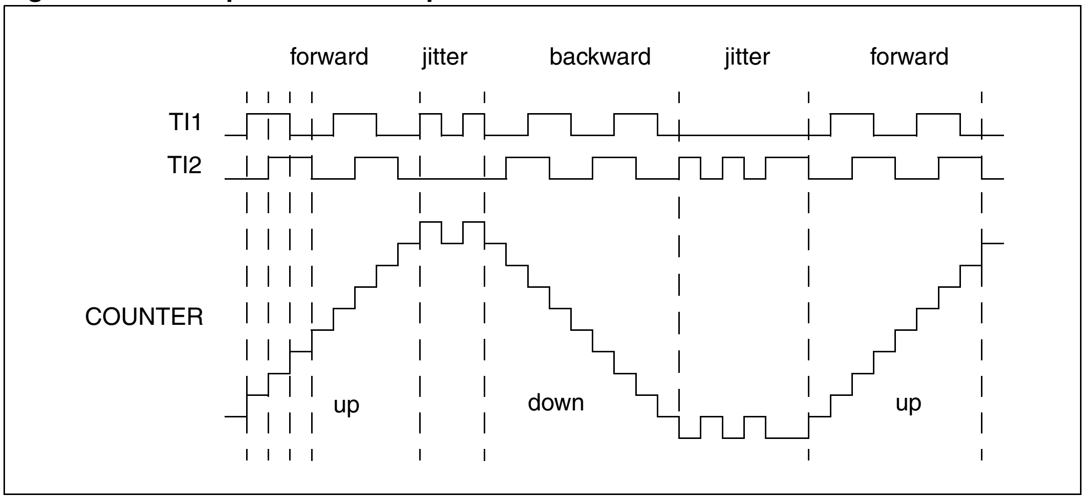
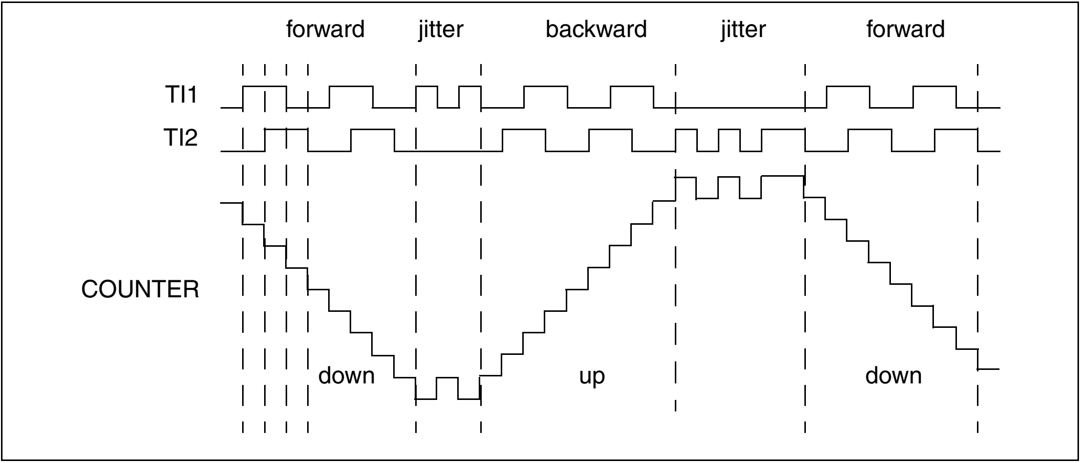


Figure 58 gives an example of counter behavior when IC1 polarity is inverted (same configuration as above except IC1P=1).

Figure 58. 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, deceleration) by measuring the periods between two encoder events using a second timer configured in input 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 value 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 input signal must be periodic). It is also possible to read its value through a DMA request generated by a real time clock.

10.5.4 Input Capture mode

The 16-bit input capture registers (TIM_ICRx) are used to latch the value of the counter after a transition detected by the corresponding ICx input. After a capture detection, data transfer to the TIM_ICRx register is inhibited until the corresponding ICxI flag is reset (by software). This means you always have to reset the ICxI flag between two consecutive captures.

TIM_ICRx are read-only registers. They can be read only when the corresponding ICxI flag is set.

The active transition on ICx is software programmable using the ICxP bit in the TIM_IMCR register.

IC1 and IC2 are specific as they can be independently mapped by software on TI1 or TI2 through the IC1S and IC2S bits.

Procedure:

To configure the TIMx in input capture mode, use the following procedure:

- Select the counter clock source (internal/external, prescaler)
- Select the active input (for example write IC1S='0' to capture the counter value in the TIM_ICR1 register on a transition on the TI1 pin).
- Select the edge of the active transition on the ICx channel by writing ICxP bit in TIM_IMCR
- Enable capture from the counter into the capture register by setting the corresponding ICxE bit in the TIM_IMCR register.
- If needed enable the interrupt request by setting the ICxRS bit in the TIM_RSR register and the ICxRE bit in the TIM_RER register, or, for TIM0, the DMA request by resetting ICxRS in TIM_RSR and setting ICxRE bit in TIM_RER register.

When an input capture occurs:

- The ICRx register gets the value of the counter on the active transition.
- ICxI is set (interrupt flag).
- An interrupt is generated depending on ICxRS and ICxRE.
- A DMA request is generated (for TIM0 only) depending on ICxRS and ICxRE.

Note: *It is not possible to send an IC interrupt or DMA request without actually capturing the counter value in the capture register.*

Clearing the input capture flag ICxI is done by:

- Writing ICxI with '0' by software,
- Automatically by the DMA controller if DMA is enabled on this source (ICxRS='0' and ICxRE= '1').

10.5.5 PWM input mode

This mode is a particular case of input capture mode. The PWM Input functionality enables the measurement of the period and the pulse width of an external waveform.

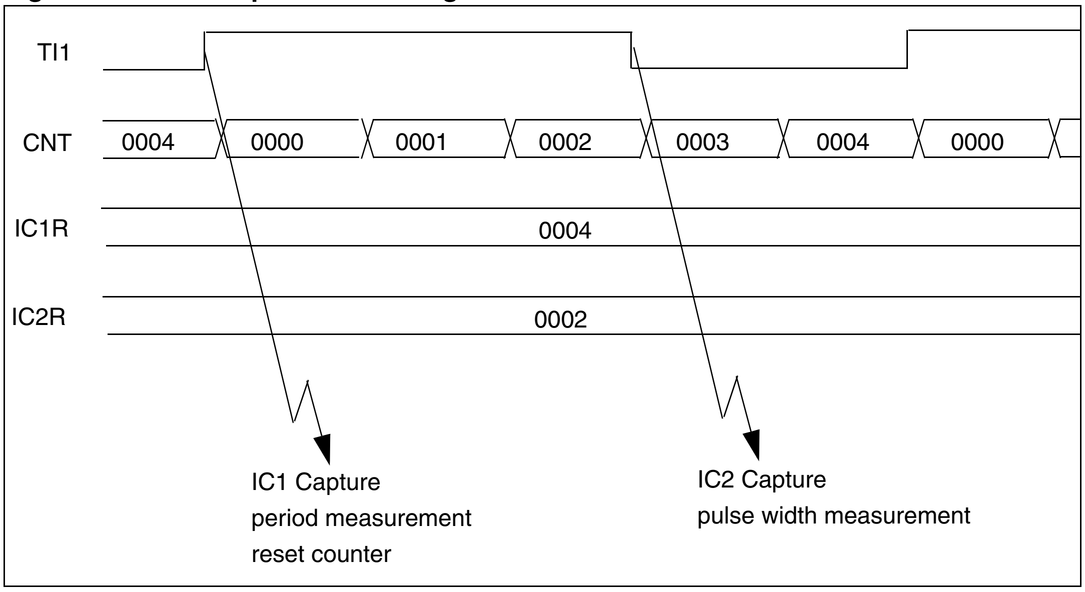
The procedure is the same except with the following conditions:

- IC1 and IC2 are mapped on the same input (eg: TI1)
- IC1 and IC2 active edges have opposite polarity
- IC1 or IC2 is selected as external trigger input and the trigger mode controller is configured in reset mode

For instance you can measure the period (in ICR1) and the duty cycle (in ICR2) of the PWM applied on TI1 with the following configuration (depending on CK_TIM frequency and prescaler value):

- IC1S= '0', IC2S='1', (TI1)
- IC1P= '0', IC2P='1', (IC1 on rising edge, IC2 on falling edge)
- TS='10' (IC1=external trigger), SMS='00', SME='1' (reset mode)

Figure 59. PWM Input mode timing



10.5.6 Forced output mode

Each output compare signal (OCx) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal OCx to its active level, you just need to write “101” in OCxC bits in the corresponding TIM_OMRx register. Thus OCREFx is forced inactive (logic 0 as OCREF is always active high) and OCx get opposite value to OCxP.

e.g.: OCxP=0 (OCx active high) => OCx is forced to high level.

OCx output can be forced to its inactive level by writing OCxC bits to “100”.

Anyway the comparison between the output compare register and the counter is still performed and allows the flag to be set as well as interrupt and DMA request to be sent. This is described in the Output Compare Mode section.

10.5.7 Output compare mode

This function is used to control an output waveform or indicating when a period of time has elapsed.

When a match is found between the output compare register (TIM_OCR1) and the counter (CNT), the output compare function:

- Assigns the corresponding pin with a programmable value defined by the Output Compare Control (OCxC bits). The output pin can keep its level, be set, be reset or can toggle on match.
- Set a flag in the interrupt status register (OCxI in TIM_ISR) if OCRM=0 or OCxRS=1 in TIM_RSR register,
- Generates an interrupt if the corresponding interrupt mask is set (OCxRS in TIM_RSR and OCxRE in TIM_RER),
- Send a DMA request if the corresponding enable bit is set (OCRM and OCxRS in TIM_RSR, OCxRE in TIM_RER).

The OCRx registers can be programmed with or without preload-registers using the PLDx bit in the corresponding TIM_OMRx register.

In output compare mode:

The Update event has no effect on OCx output.

The OCx level is updated each time the counter value matches the output compare register depending on the mode selected:

Table 40. Output compare modes

OCxC bits	OCx pin status
000	Doesn't change
001	Set to active level
010	Set to inactive level
011	Toggles
111	Reserved

Active and inactive levels are defined by OCx polarity. When the OCxP bit in TIM_OMRx register is written to 0, OCx is active high, it is active low when OCxP=1.

Output compare mode can also be used to output a single pulse (See "*Section 10.5.9: One Pulse mode on page 212*.).

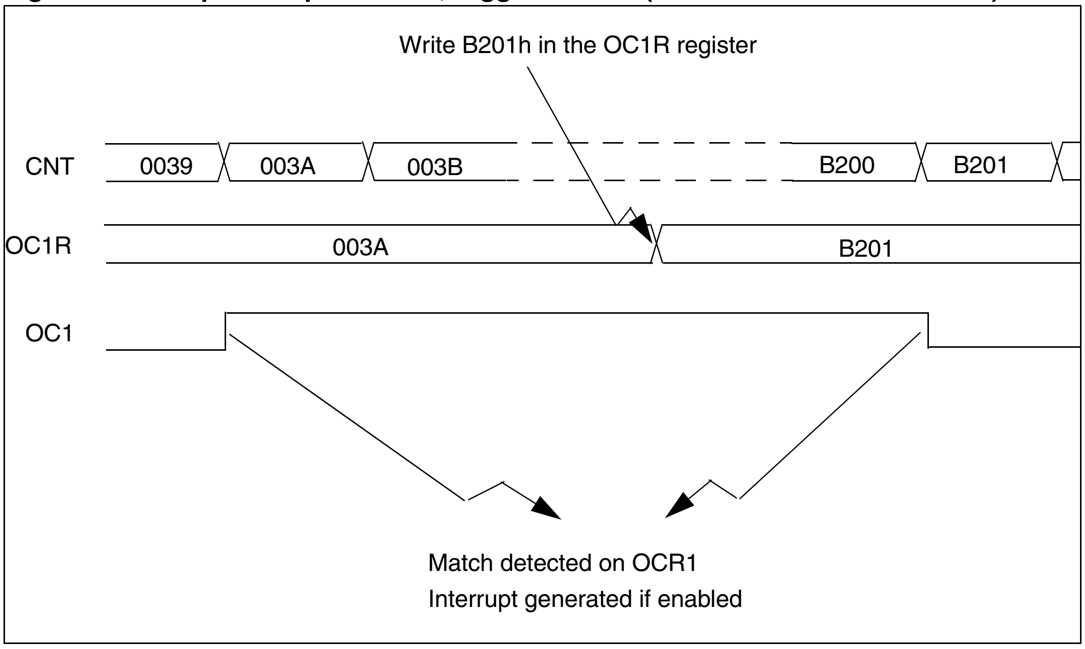
The timing resolution is one count of the counter.

Procedure:

- Select the counter clock (internal/external, prescaler).
- Write the desired data in the TIM_ARR and TIM_OCRx registers.
- Write to the OCxRS bit in the TIM_RSR register to select if an interrupt or a DMA request is to be generated. OCRM must be written to 0 if DMA request is expected.
- Set the OCxRE bit in the TIM_RER enable interrupt or DMA request
- Select the output mode (for example, to select no preload register and to toggle OCx output pin when CNT matches OCRx, write OCxE = '1', OCxP='0', PLDx='0' and OCxC='011').
- Enable Alternate Function on the corresponding I/O port using the GPIO Port Control Registers
- Enable the counter by setting the CNT_EN bit in the TIM_CR register.

TIM_OCRx can be updated at any time by software to control the output waveform provide that the preload register is not enabled (else active OCRx value will change only at the next update event). An example is given in *Figure 9.5.2*.

Figure 60. Output compare mode, toggle on OC1 (OM1='01100' in TIM_OMR1)


Write B201h in the OC1R register
CNT
0039
003A
003B
B200
B201
OC1R
003A
B201
OC1
Match detected on OCR1
Interrupt generated if enabled

10.5.8 PWM mode

Pulse Width Modulation mode allows you to generate a signal with a frequency determined by the value of the TIM_ARR register and a duty cycle determined by the value of the TIM_OCRx register.

The **PWM Mode** can be selected independently on each channel (one PWM per OCx output) by writing '110' in the OCxC control bits in the TIM_OMRx register. You must enable the corresponding preload register by setting the PLDx bit in the TIM_OMRx register.

As a general rule it is mandatory to enable the preload registers to generate PWMs correctly. Preload registers act on the output compare registers (programmable channel by channel), the auto-reload register (in up-counting or center-aligned modes) and the prescaler.

As the preload registers are transferred to the active registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the CNT_RST bit in the TIM_CR register. You can set the CNT_RST and the CNT_EN bits at the same time.

OCx polarity is software programmable using the OCxP bit in TIM_OMRx. It can be programmed as active high or active low.

OCx output enable is controlled by the OCxE bit in the corresponding TIM_OMRx register.

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIM_CR control register. Refer to *Table 41* for the corresponding counting sequence.

Table 41. **Counting sequence in center-aligned and edge-aligned mode**

center-aligned mode	0	1	2	....	15	16	15	....	2	1	0	1

T

edge-aligned mode	0	1	2	.....	15	16	0	1	.....	16	0	1

T

T = PWM period, Value of 12-bit Compare 0 Register= 16

Edge-aligned mode

- Up-counting configuration

Up-counting is active when the U/D bit in the TIM_CR control register is low.

In this mode, the PWM counter counts up to the value loaded in the auto-reload register (TIM_ARR). Then it is cleared and restarts counting.

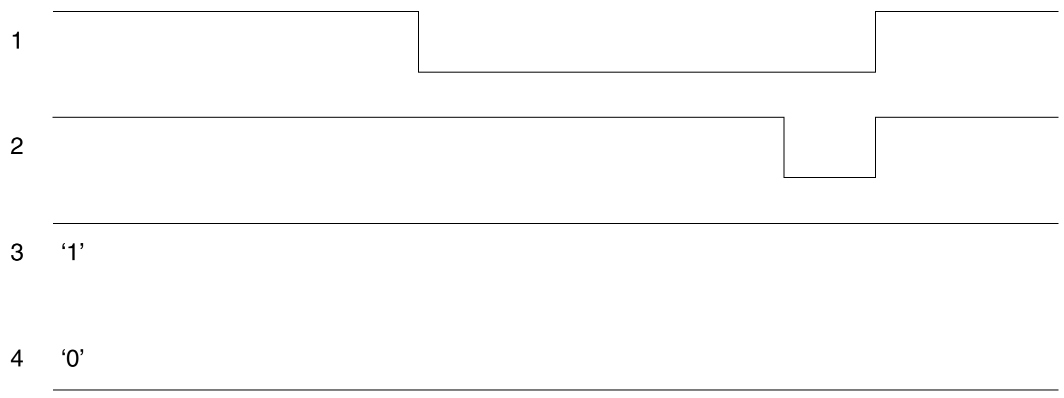
The reference PWM signal OCxREF is high (meaning that the channel is active) as long as TIM_CNT<TIM_OCRx else it becomes low (inactive). OCx follows this pattern taking into account its polarity configured with OCxP bit in TIM_OMRx register.

If the compare value in TIM_OCRx is greater than the auto-reload value (in TIM_ARR) then OCxREF will be held at ‘1’. If the compare value is 0 then OCxREF will be held at ‘0’.

The following figure shows some edge-aligned PWM waveforms in an example where TIM_ARR=8.

Table 42. **Edge-aligned PWM waveforms (Auto-Reload Register = 8)**

0	1	2	3	4	5	6	7	8	0	1



1 Compare Register value = 4
2 Compare Register value = 8
3 Compare Register value > 8
4 Compare Register value = 0

- Down-counting configuration

Down-counting is active when U/D bit in TIM_CR control register is high.

In this operating mode, the PWM counter counts from the value loaded in the auto-reload register (TIM_ARR) down to 0. Then it is reloaded with the auto-reload value and restarts down-counting.

The reference PWM signal OCxREF is low as long as TIM_CNT>TIM_OCRx else it becomes high.

If the compare value in TIM_OCRx is greater than the auto-reload value (in TIM_ARR) then OCxREF will be held at ‘1’.

0% PWM is not possible in this mode.

Center-aligned mode

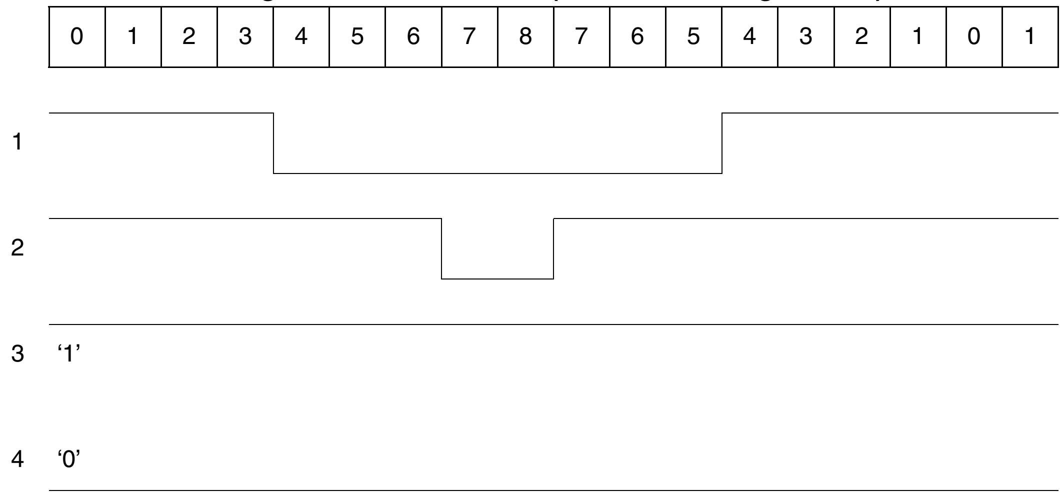
In this mode the counter counts up to the value loaded in the auto-reload register TIM_ARR then counts down until it reaches 0 and restarts counting up. U/D bit in the TIM_CR register is updated by hardware and must not be changed by software.

The reference PWM signal OCxREF is set to ‘0’ (inactive channel) when the counter reaches, in up-counting, the compare value (in TIM_OCRx) and it is set to ‘1’ (active channel) when the counter reaches the compare value again in down-counting. OCx output will follow this pattern taking in account its polarity configured with OCxP bit in TIM_OMRx register.

If the compare value in TIM_OCRx is greater than the auto-reload value (in TIM_ARR) then OCxREF will be held at ‘1’. If the compare value is 0 then OCxREF will be held at ‘0’.

Figure 43 shows some center-aligned PWM waveforms in an example where TIM_ARR=8.

Table 43. Center-aligned PWM waveforms (Auto-Reload Register = 8)



1 Compare Register value = 4
2 Compare Register value = 7
3 Compare Register value > = 8
4 Compare Register value = 0

Depending on the CMS bits in the TIM_CR register, the DMA/Interrupt flags related to output compare channels (OCxI) are set on the OCxREF rising edge, falling edge or both (refer to the TIM_CR register description).

Hints on using center-aligned mode:

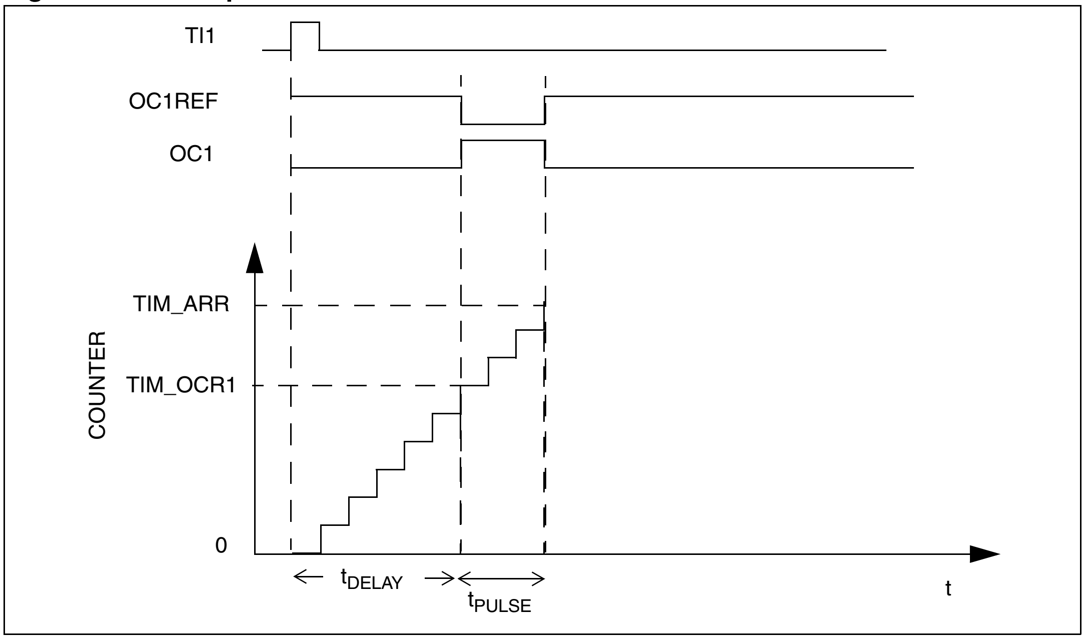
- When starting in center-aligned mode, the current up-down configuration is used. It means that the counter will start counting up or down depending on the value written in the U/D bit in the TIM_CR register. Moreover the U/D and CMS bits must not be changed at the same time by the software.
- Writing to the counter while running in center-aligned mode is not recommended as it can lead to unexpected results. In particular:
  - The Up/Down configuration is not updated if you write a value in the counter that is greater than the auto-reload value (TIM_CNT>TIM_ARR). For example, if the counter was counting up, it will continue to count up.
  - The Up/Down configuration is updated if you write '0' or write the TIM_ARR value in the counter but no Update event is generated (and nor any Update interrupt or DMA request).
- The safest way to use center-aligned mode is to generate an update by software (setting the CNT_RST bit in the TIM_CR register) just before starting the counter and not to write the counter while it is running.

10.5.9 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 trigger controller. Generating the pulse waveform can be done in output compare or PWM mode. You select One Pulse Mode by setting the OPM bit in the TIM_CR control register. The counter stops automatically at the next update event.

Figure 61. 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 TI1 input pin.

Let’s use IC1 as trigger 1:

- Map IC1 on TI1 by writing IC1S = ‘0’ in the TIM_IMCR register.
- IC1 must detect a positive edge, write IC1P = ‘0’ in the TIM_IMCR register.
- Configure IC1 as trigger (TRG) by writing TS=‘10’ in the TIM_SCR register.
- IC1 is used to start the counter by writing SMS to ‘11’ and SME to ‘1’ in the TIM_SCR register (trigger mode).

The OPM waveform is defined by writing the compare registers (taking into account the clock frequency and the counter prescaler).

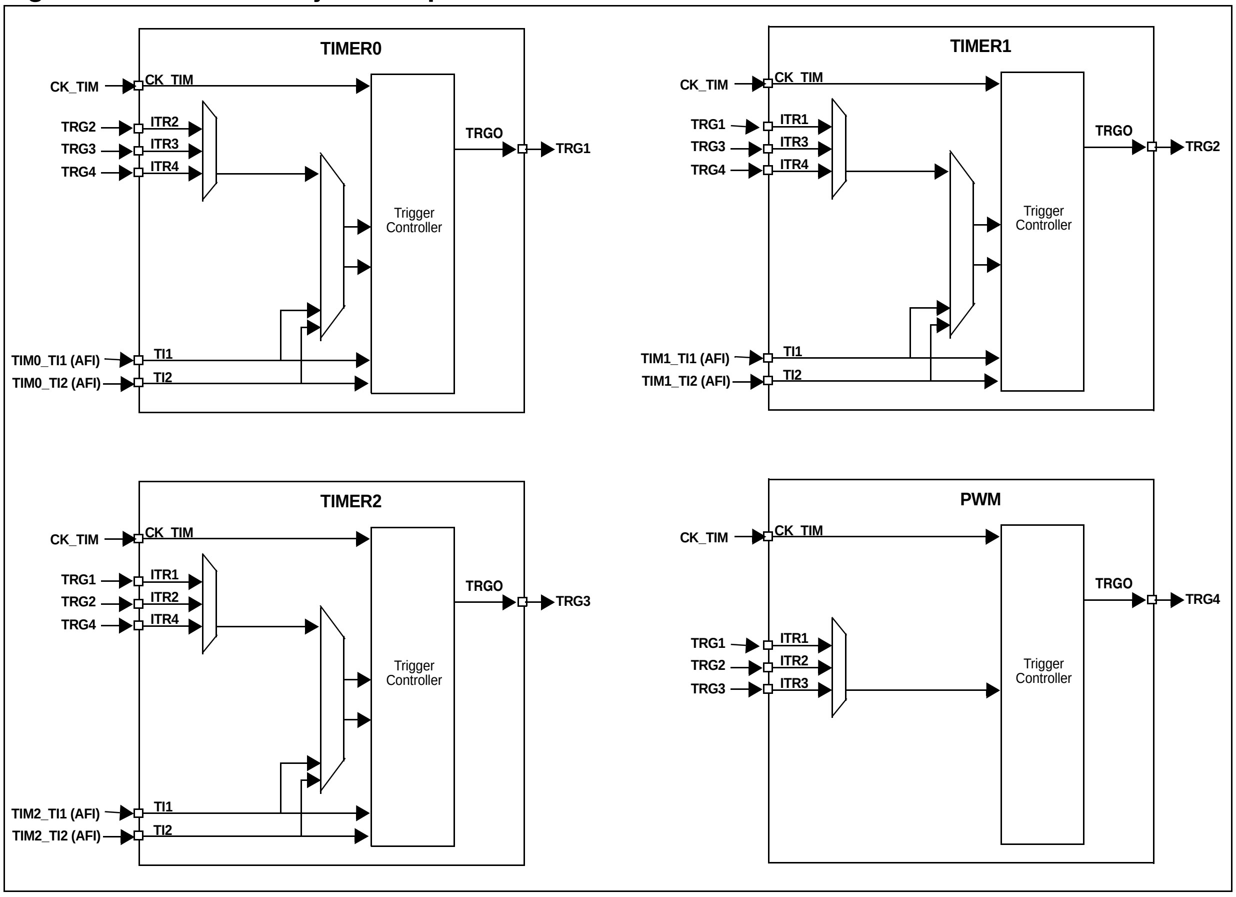
- The t_{DELAY} is defined by the value written in the TIM_OCR1 compare register.
- The t_{PULSE} is defined by the difference between the auto-reload value and the compare value (TIM_ARR - TIM_OCR1).
- Let’s say you want to build a waveform with a transition from ‘0’ to ‘1’ when a compare match occurs and a transition from ‘1’ to ‘0’ when the counter reaches the Autoreload value. To do this you enable PWM mode by writing OC1C=110 in the TIM_OMR1 register.
  You can optionally enable the preload register by writing PLD1=1 in the TIM_OMR1 register. In this case you have to write the compare value in the TIM_OCR1 register, generate an update and wait for the external trigger event on IC1.
  To get a pulse with the correct polarity you must invert the polarity by setting the OC1P bit in the TIM_OMR1 register.
  In our case the U/D and CMS[1:0] bits in the TIM_CR register should be low. Since this is their reset state, there is no need to modify them.
- You only want 1 pulse. So you write ‘1’ to the OPM bit in the TIM_CR register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0).

10.6 Timer synchronization

10.6.1 Timer Link system

The three Timers and the PWM are linked together internally for timer synchronization or chaining. Refer to *Figure 62*. Using the Timer Link system, a Timer or a PWM configured in Master Mode can reset, start, stop or clock the counter of another Timer or the PWM configured in Slave Mode. In the case of Timers TIM2:0, the external Timer Input pins TI1 and TI2 can alternatively be used as trigger inputs.

Figure 62. Timer Link system implementation



10.6.2 Synchronization of 2 Timers

You can synchronize two timers together or synchronize a timer with an external signal using two different modes:

- Gated Mode
- Trigger Mode

The following sections use the example of Timer 2 being enabled by Timer 1 in these two modes.

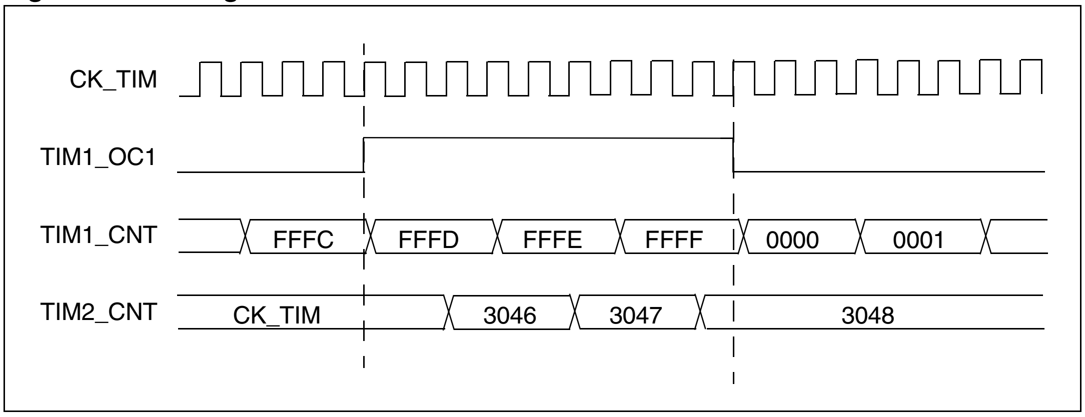
Gated mode

In this example, configure Timer 2 in gated mode by writing SMS=10 and SME=1 in TIM2_SCR register. Then select external clock as the input source by writing the corresponding values in the TS and ITS bits. Once Timer 2 is configured in gated mode, you then enable it by setting the CNT_EN bit in TIM2_CR register. The counter won’t start until the trigger input is at active level.

You can choose to start Timer 2 when the Output Compare 1 (OC1) output of Timer 1 is high. To do this, Timer 2 registers must be programmed with: SMS=10, SME=1, TS=00, ITS=001 and CNT_EN = 1 and Timer 1 must be programmed with MMS=11 and CNT_EN = 1. *Figure 63* shows how this works. In this example the prescaler is 2 for both counters ($f_{CK\_CNT} = f_{CK\_TIM}/3$).

Note: The counter 2 clock is not synchronized with counter 1, this mode only affects the Timer 2 counter enable signal.

Figure 63. Gating timer 2 with OC1 of Timer 1

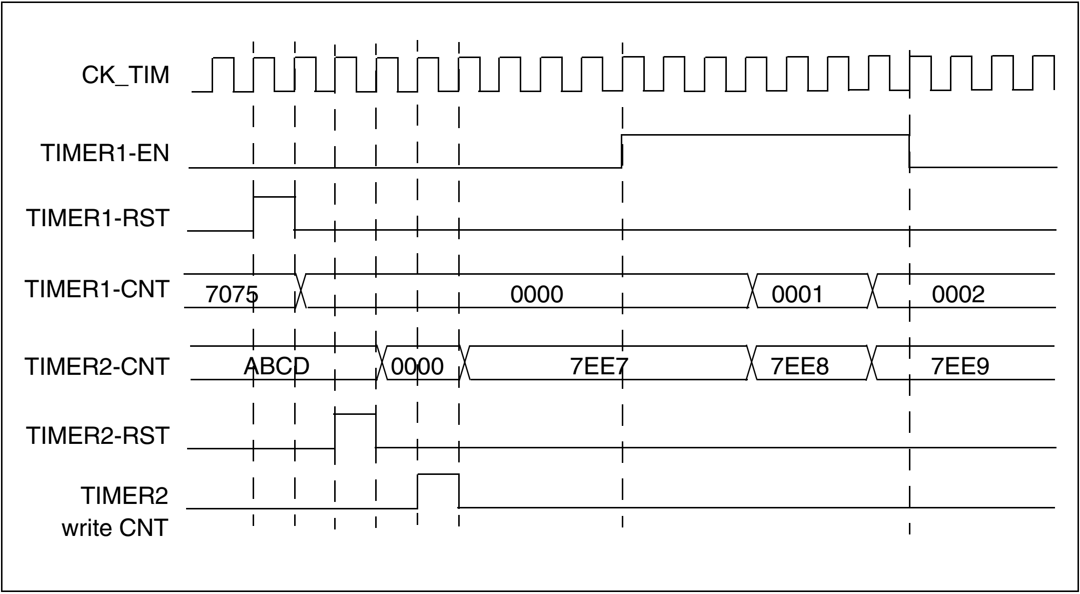


In the example in *Figure 63*, the Timer 2 counter and prescaler are not initialized before being started. So they start counting from their current value. It is possible to start from a given value by resetting both timers before starting Timer 1. You can then write any value you want in the timer counters. The timers can easily be reset by software using the CNT_RST bit in the TIM_CR registers.

In the next example, we synchronize Timer 1 and Timer 2. Timer 1 is the master and starts from 0000. Timer 2 is the slave and starts at the same time from 7EE7h. The prescaler ratio is the same for both timers. Timer 2 stops when Timer 1 is disabled by writing '0' to the CNT_EN bit in the TIM1_CR register:

- Configure Timer 1 master mode to send its enable command as trigger output (MM=01 in the TIM1_CR register).
- Configure Timer 2 to get the input trigger from Timer 1 (ITS = '001' and TS='00' in the TIM2_SCR register).
- Configure Timer 2 in gated mode (SMS='10', SME='1' in the TIM2_SCR).
- Reset Timer 1 by writing '1' in CNT_RST bit (TIM1_CR register).
- Reset Timer 2 by writing '1' in CNT_RST bit (TIM2_ CR register).
- Initialize Timer 2 to 7EE7h by writing '7EE7h' in the Timer 2 counter (TIM2_CNT).
- Enable Timer 2 by writing '1' in the CNT_EN bit (TIM2_CR register).
- Start Timer 1 by writing '1' in the CNT_EN bit (TIM1_CR register).
- Stop Timer 1 by writing '0' in the CNT_EN bit (TIM1_CR register).

Figure 64. Gating Timer 2 with ENABLE of Timer 1

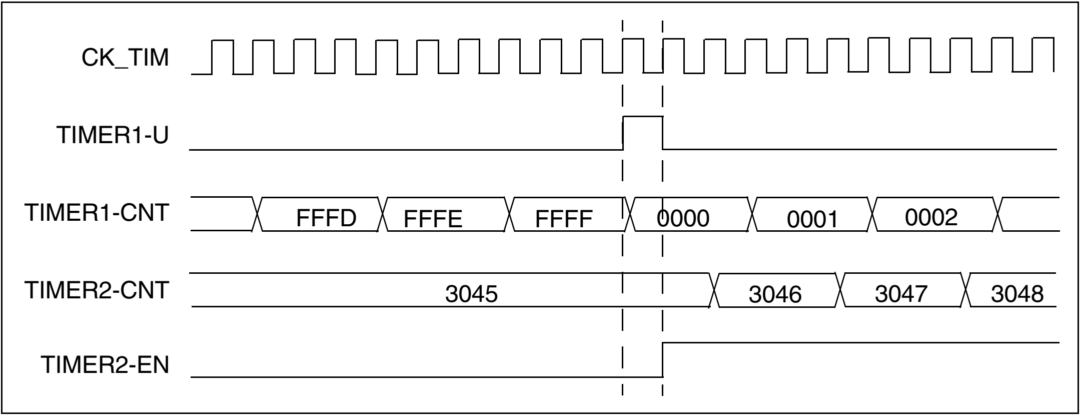


Trigger mode

In this next example, we configure Timer 2 in trigger mode by writing SMS='11' and SME='1' in the TIM2_CR register. As in the previous example, we select the input source using the TS and ITS bits.

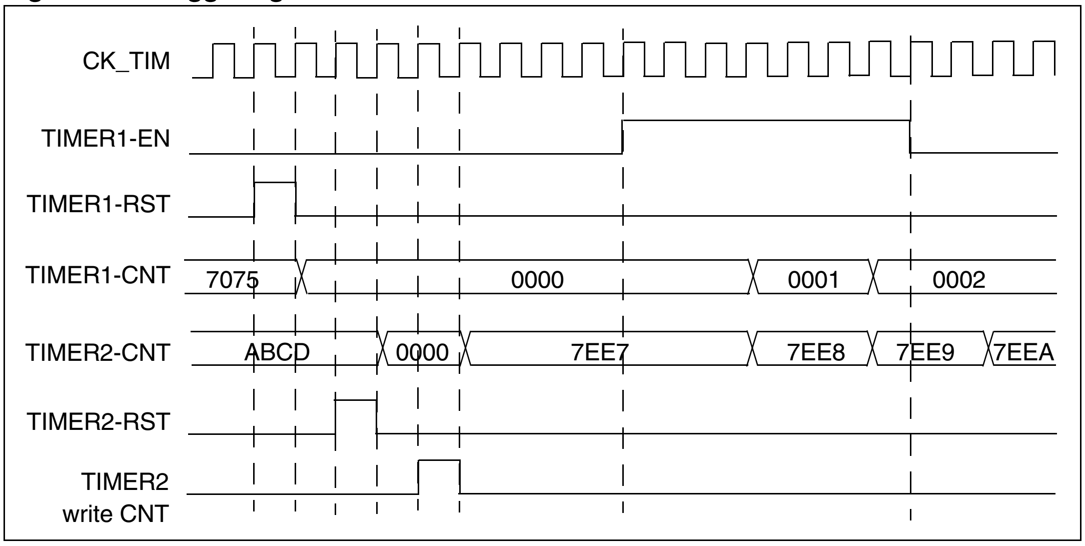
In our example, Timer 2 starts counting from its current value (which can be non-zero) as soon as an Update event is generated by Timer 1. When Timer 2 receives the trigger signal its CNT_EN bit is automatically set and the counter counts until we write '0' to the CNT_EN bit in the TIM2_CR register.

Figure 65. Triggering Timer 2 with UPDATE of Timer 1



As in the previous example, you can initialize both counters before starting counting. *Figure 66* shows the behavior with the same configuration as in the *Figure 64* but in trigger mode instead of gated mode (SMS='11' and SME='1' in the TIM2_SCR register).

Figure 66. Triggering timer 2 with ENABLE of timer 1



10.7 Interrupt management

To use the interrupt features, for each interrupt channel used, perform the following sequence:

- Set the desired OCRSx and/or ICRSx and/or URS bits to select interrupt request rather than DMA request (only for TIM0).
- Set the desired OCREx and/or ICREx and/or URE bits in the TIM_RER register to enable interrupt requests.

10.8 DMA function

10.8.1 Using the DMA feature

- Reset the desired OCRSx and/or ICRSx and/or URS bits to select DMA request rather than interrupt request.
- Set the desired OCREx and/or ICREx and/or URE bits in the TIM_RER register to enable DMA requests.

The DMA request generated as soon as the corresponding flag in the TIM_ISR register) is set. The flag is automatically cleared by hardware by the DMA controller.

A flag must not be written by software when it is controlled by the DMA (when xRS=0 and xRE=1).

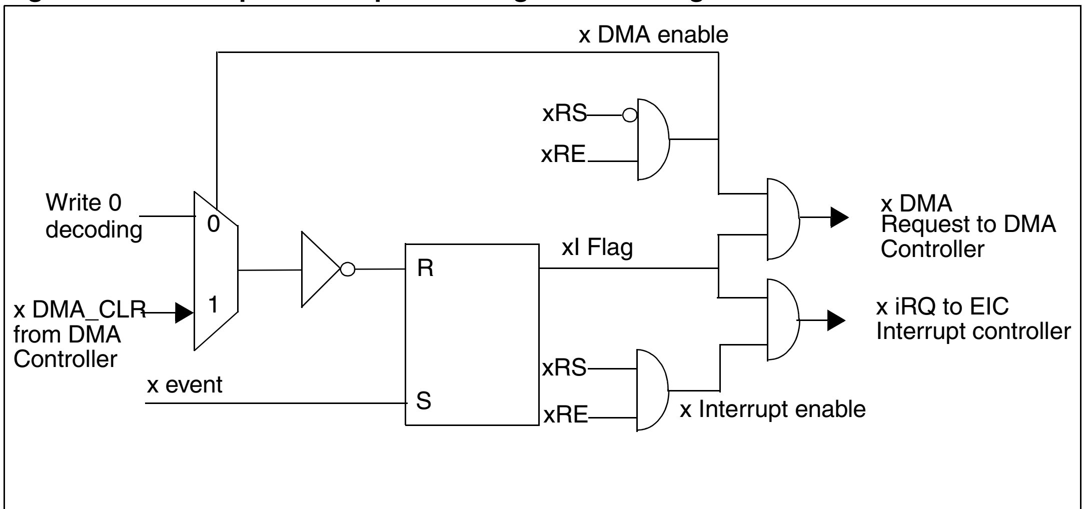
In addition, it is not recommended to modify xRS value while xRE=1.

Moreover, in case of Output Compare requests, the request selection (xRS) and the OC request mode (OCRM) must not be modified while the request is enabled (xRE=1). To change the request configuration when xRE=1:

- Check that no DMA request is pending on the corresponding event
- Clear the enable bit xRE
- Configure the request (xRS and OCRM bits)
- Wait for flag re-synchronization (4 clock cycles)
- Clear the flag by software
- Enable the request by setting the xRE bit

The following figure shows a simplified schematic of flag management

Figure 67. Interrupt/DMA request management for flag "x"


x DMA enable
xRS
xRE
Write 0 decoding
0
1
x DMA_CLR from DMA Controller
R
S
xI Flag
x event
xRS
xRE
x DMA Request to DMA Controller
x iRQ to EIC Interrupt controller
x Interrupt enable

xRS, xRE: control bits in TIM_RSR and TIM_RER registers
xI Flag: status flag in TIM_ISR register

10.8.2 DMA access in burst mode

All the TIM0 registers can be accessed independently by the DMA controller. A specific register address is used for to burst access (TIM0_DMAB: DMA Burst). Any read or write access to this address is redirected to the corresponding register selected by a DMA base-address and a DMA index.

The DMA Base-address is programmed in the DBASE bits in the TIM0_CR register. DBASE is defined as an offset from the timer base address (DBASE=0 means that TIM0_CR is used as DMA base-address. DBASE=1 means that TIM_SCR is used as DMA base-address).

The DMA index adds an offset to the DMA base-address. It is cleared as soon as the DMA request is sent by the timer to prepare the next transfer. Then it is incremented each time the TIM0_DMAB register is accessed by the DMA controller (you program the length of the transfer using the DMA controller registers). This means that several consecutive accesses to the TIM0_DMAB will access an array of consecutive registers.

In the following example, we want to transfer 2 values of a memory table automatically into the TIM0_OCR1 and TIM0_OCR2 registers each time an update event occurs:

- Configure the DMA base-address to point to the TIM0_OCR1 register by writing DBASE to ‘01101’ (13th register in the register map),
- Enable DMA requests on Update event by clearing the URS bit in TIM0_RSR register and setting the URE bit in the TIM0_RER register.
- Program the DMA controller so that an update DMA request generates a transfer of 2 half-words from the memory to the TIM0_DMAB address (in burst mode).

1. When the update occurs, the UI flag is set in the TIM0_ISR register, the request is sent to the DMA controller and the DMA index is cleared (TIM0_DMAB is redirected to TIM0_OCR1 register).
2. The DMA writes the first value to the TIM0_DMAB address and is redirected to write in the TIM_OCR1 register. Then the DMA index is incremented (TIM0_DMAB is redirected to TIM0_OCR2 register).
3. The DMA writes the second value to the TIM0_DMAB address and is redirected to write in the TIM0_OCR2 register. As only 2 values are to be transferred, the DMA controller clears the UI flag in TIM0_ISR register and stops writing to the TIM0_DMAB address.

Note: 1 *Do not write directly in the TIM0_DMAB address as this could corrupt the DMA index (if a burst transfer is interrupted).*

2 *The timer can manage only one burst transfer at a time. This means that you must not program 2 different DMA channels to access TIM0_DMAB at the same time (for instance one in response to an update event and another in response to an output compare event). Else the index could be corrupted and access to TIM0_DMAB would be redirected to unexpected registers.*

10.9 Debug mode

When the microcontroller enters debug mode, the timer either continues to work normally or stops its activity, depending on the DBGC bit in TIM_CR register.

If DBGC=0 (reset state), the timer detects any activity on the debug acknowledge (DBGACK) signal which is directly sent by the CPU. Then, as soon as the device enters debug mode:

- The TIM_CNT counter stops counting
- The input captures are blocked
- The trigger input is blocked

Thus, when a breakpoint occurs, the state of all the registers is kept stable.

If DBGC=1, the timer ignores the DBGACK signal.

10.10 Register description

All can have only be accessed 16-bit access. A byte cannot be read or written. In this section, the following abbreviations are used:

read/write (rw)	Software can read and write to these bits.
read-only (r)	Software can only read these bits.
read/clear (rc_w0)	Software can read as well as clear this bit by writing 0. Writing ‘1’ has no effect on the bit value.
Write once-only (wo)	Software can read these bits. It can write them only once. Once write-accessed, these bits cannot be re-written unless the processor is reset.

10.10.1 Control Register (TIM_CR)

Address Offset: 00h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DBASE					DBGC	MMS		res.	CMS		U/D	OPM	CNT_EN	CNT_RST	UFS
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 15:11	**DBASE:** *DMA Base-address* This 5-bit vector defines the base-address for DMA transfers in burst mode (when read/write access are done through the TIM_DMAB address). DBASE is defined as an offset starting from the first address of the timer (which is the address of TIM_CR). 00000: TIM_CR 00001: TIM_SCR 00010: TIM_IMCR ... 10111: TIM_DTR
Bit 10	**DBGC**: *Debug Control* See *Section 9.5.3: Debug mode on page 189* for a detailed description. 0: The timer is stopped in debug mode 1: The timer continues working in debug mode
Bits 9:8	**MMS[1:0]**: *Master Mode Selection* These bits select the source of the trigger event sent in master mode to slave timers for synchronization (TRGO). 00: Reset - The CNT_RST bit is used as trigger output (TRGO). **Note:** If the CNT_RST bit is set by an input trigger (timer configured in Slave reset mode by the SMS bits in the TIM_SCR register) then there is a delay between the counter reset and the TRGO event. 01: Enable - The Counter Enable signal is used as trigger output (TRGO). This is useful for starting several timers at the same time or controlling a window in which a slave timer is enabled. **Note:** The Counter Enable signal is generated by a logic OR between the CNT_EN 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. 10: Update - The update event is selected as trigger output (TRGO). For example a master timer can then be used as a prescaler for a slave timer (see *Section on page 202*). 11: Compare - The OC1 signal is used as trigger output (TRGO).
Bit 7	Reserved: must be kept cleared.

Bits 6:5	**CMS:** *Center-Aligned Mode Selection* Do not switch from edge-aligned mode to center-aligned mode while the counter is enabled (CNT_EN=1) 00: Edge-aligned mode (used in PWM mode). 01: Center-aligned mode 1 (used in PWM mode). Output compare interrupt flags of channels configured in PWM mode (OCxC=110 in TIM_OMRx register) are set on rising edges of the corresponding OCREF signal. 10: Center-aligned mode 2 (used in PWM mode). Output compare interrupt flags of channels configured in PWM mode (OCxC=110 in TIM_OMRx register) are set on falling edges of the corresponding OCREF signal. 11: Center-aligned mode 3 (used in PWM mode). Output compare interrupt flags of channels configured in PWM mode (OCxC=110 in TIM_OMRx register) will be set on both rising and falling edges of the corresponding OCREF signal.
Bit 4	**U/D**: *Up/Down* 0: Counter used as up-counter 1: Counter used as down-counter
Bit 3	**OPM**: *One Pulse Mode* 0: Counter is not stopped at update event 1: Counter stops counting at the next update event (resetting the bit CNT EN)
Bit 2	**CNT_EN**: *Counter Enable* 0: Counter disabled 1: Counter enabled **Note:** In External clock, gated mode and encoder interface mode the counter works only if CNT_EN is set by software. However in trigger mode CNT_EN is set automatically by hardware.
Bit 1	**CNT_RST**: *Counter Reset* This bit can be written to 1 to reset the counter, it is automatically reset 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 (however the prescaler preload value is not affected). The counter is cleared if center-aligned mode is selected or if U/D=0 (up-counting), else it takes the auto-reload value (TIM_ARR) if U/D=1 (down-counting)
Bit 0	**UFS**: *Update Flag Selection* 0: UI flag is set when an update event occurs, whatever its source (counter overflow/underflow, CNT RST bit set by software or slave mode controller in reset mode) 1: UI flag is set only if a counter overflow/underflow is detected. It is not affected by updates generated by software or by the slave mode controller)

10.10.2 Synchro Control Register (TIM_SCR)

Address Offset: 04h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res.	ITS			reserved		TS		reserved			SMS		SME	ENC	
	rw	rw	rw			rw	rw				rw	rw	rw	rw	rw

Bit 15	Reserved: must be kept cleared.
Bits 14:12	**ITS**: *Internal Trigger Selection* Selects the internal trigger to be used to synchronize the counter. These bits must be changed only when they are not enabled (e.g. when SME is low) to avoid wrong edge detections at the transition. 001: Internal Trigger 1 (ITR1) 010: Internal Trigger 2 (ITR2) 011: Internal Trigger 3 (ITR3) 100: Internal Trigger 4 (ITR4)
Bits 11:10	Reserved: must be kept cleared.
Bits 9:8	**TS**: *Trigger Selection* Selects the trigger signal (TRG) to be used by the trigger controller. 00: Internal trigger (output of the internal trigger selection) 10: Input Capture 1 (IC1) 11: Input Capture 2 (IC2)
Bits 7:5	Reserved: must be kept cleared.
Bits 4:3	**SMS**: *Slave Mode Selection* When external signals are selected the active edge of the trigger signal (TRG) is linked to the polarity selected on the external input (see TIM_IMCR and TIM_CR Register description) 00: Reset - rising edge of the selected trigger signal (TRG) resets the counter and generates an update of the registers. 01: External Clock - rising edge of selected trigger (TRG) clocks the counter. 10: Gated Mode: Counter clock is enabled when trigger signal (TRG) 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. 11: Trigger Mode: The counter starts at a rising edge of the trigger TRG (but it is not reset). Only the start of the counter is controlled.
Bit 2	**SME**: *Slave Mode Enable* 0: Slave mode disabled. If CNT_EN = ‘1’ then the prescaler is clocked directly by CK_TIM. 1: Slave mode enable. The counter is controlled by the selected trigger input. Depending on the slave mode selection (SMS) it can be reset, clocked or enabled.
Bits 1:0	**ENC**: *Encoder Interface Mode Selection* 00: Off - Encoder interface mode disabled. 01: Encoder mode 1 - Counter counts up/down on IC1 edge depending on IC2 level. 10: Encoder mode 2 - Counter counts up/down on IC2 edge depending on IC1 level. 11: Encoder mode 3 - Counter counts up/down on both IC1 and IC2 edges depending on the level of the other input.

10.10.3 Input Mode Control Register (TIM_IMCR)

Address Offset: 08h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved										IC2P	IC2E	IC1P	IC1E	IC2S	IC1S
										rw	rw	rw	rw	rw	rw

Bits 15:6	Reserved: must be kept cleared.
Bit 5	**IC2P**: *IC2 Polarity* This bit selects whether IC2 or $\overline{\text{IC2}}$ is used for trigger or capture operations. 0: non-inverted: capture is done on a rising edge of IC2. When used as external trigger, IC2 is non-inverted. 1: inverted: capture is done on a falling edge of IC2. When used as external trigger, IC2 is inverted.
Bit 4	**IC2E**: *IC2 Enable* This bit determines if capture of the counter value into Input Capture Register 2 (TIM_ICR2) is enabled or not. 0: Capture disable. 1: Capture enable if IC2I flag (TIM_ISR register) is reset.
Bit 3	**IC1P**: *IC1 Polarity* This bit selects whether IC1 or $\overline{\text{IC1}}$ is used for trigger or capture operations. 0: non-inverted: capture is done on a rising edge of IC1. When used as external trigger, IC1 is non-inverted. 1: inverted: capture is done on a falling edge of IC1. When used as external trigger, IC1 is inverted.
Bit 2	**IC1E**: *IC1 Enable* This bit determines if a capture of the counter value into Input Capture Register 1 (TIM_ICR1) is enabled or not. 0: Capture disable. 1: Capture enable if IC1I flag (TIM_ISR register) is reset.
Bit 1	**IC2S**: *IC2 Selection* Determines if IC2 is connected TI1 or TI2. 0: Timer input 2 (TI2) is selected. 1: Timer input 1 (TI1) is selected.
Bit 0	**IC1S**: *IC1 Selection* Determines if IC1 is connected TI1 or TI2. 0: Timer input 1 (TI1) is selected. 1: Timer input 2 (TI2) is selected.

10.10.4 Output Mode Register 1 (TIM_OMR1)

Address Offset: 0Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res.	res.	OC2P	OC2E	OC2C[2:0]			PLD2	res.	res.	OC1P	OC1E	OC1C[2:0]			PLD1
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15	Reserved: must be kept cleared.
Bit 14	Reserved: must be kept cleared.
Bit 13	**OC2P**: *OC2 Polarity* 0: OC2 active high 1: OC2 active low
Bit 12	**OC2E**: *OC2 Enable* 0: Off - OC2 signal is disabled. Refer to *Table 40* for more details. 1: On - OC2 signal is output on the corresponding output pin.
Bits 11:9	**OC2C[2:0]**: *OC2 Control* These bits control the output reference signal OC2REF from which the OC2 output is derived. OC2REF is active high. The active level of the OC2 output depends on the OC2P bit. 000: Frozen - The comparison between the output compare register TIM_OCR2 and the TIM_CNT counter has no effect on the outputs. 001: Set OC2 channel to active level on match. OC2REF signal is forced high when the TIM_CNT counter matches Output Compare Register 2 (TIM_OCR2). 010: Set OC2 channel to inactive level on match. OC2REF signal is forced low when the TIM_CNT counter matches Output Compare Register 2 (TIM_OCR2). 011: Toggle OC2REF signal when TIM_CNT = TIM_OCR2. 100: Force inactive (low) level on OC2REF signal. 101: Force active (high) level on OC2REF signal. 110: PWM mode - In up-counting, OC2 channel is active if TIM_CNT < TIM_OCR2, else inactive. In down-counting, OC2 channel is inactive if TIM_CNT > TIM_OCR2, else active. 111: Reserved
Bit 8	**PLD2**: *Preload Enable 2* 0: Preload register on TIM_OCR2 disabled. TIM_OCR2 can be written at anytime, the new value is taken in account immediately. 1: Preload register on TIM_OCR2 enabled. Read/Write operations access the preload register. TIM_OCR2 preload value is loaded in the active register at each Update event (when the counter reaches the autoreload value or when an update is generated by software or by the slave mode controller). **Note:** It is mandatory to enable the preload register when using PWM mode, otherwise operation is not guaranteed. It is not mandatory in One Pulse Mode (OPM bit set in the TIM_CR register).
Bit 7	Reserved: must be kept cleared.
Bit 6	Reserved: must be kept cleared.

Bit 5	**OC1P**: *OC1 Polarity* 0: OC1 active high 1: OC1 active low
Bit 4	**OC1E**: *OC1 Enable* 0: Off - OC1 signal is disabled 1: On - OC1 signal is output on the corresponding channel. **Note:** Refer to *Table 40* for more details.
Bits 3:1	**OC1C[2:0]**: *OC1 Control* These bits control the output reference signal OC1REF from which the OC1 output is derived. OC1REF is active high. The active level of the OC1 output depends on the OC1P bit. 000: Frozen - The comparison between the output compare register TIM_OCR1 and the TIM_CNT counter has no effect on the outputs. 001: Set OC1 channel to active level on match. OC1REF signal is forced high when the TIM_CNT counter matches Output Compare Register 1 (TIM_OCR1). 010: Set OC1 channel to inactive level on match. OC1REF signal is forced low when the TIM_CNT counter matches Output Compare Register 1 (TIM_OCR1). 011: Toggle OC1REF signal when TIM_CNT = TIM_OCR1. 100: Force inactive (low) level on OC1REF signal. 101: Force active (high) level on OC1REF signal. 110: PWM mode - In up-counting, OC1 channel is active if TIM_CNT < TIM_OCR1, else inactive. In down-counting, OC1 channel is inactive if TIM_CNT > TIM_OCR1, else active. 111: Reserved
Bit 0	**PLD1**: *Preload Enable 1* 0: Preload register on TIM_OCR1 disabled. TIM_OCR1 can be written at anytime, the new value is taken in account immediately. 1: Preload register on TIM_OCR1 enabled. Read/Write operations access the preload register. The TIM_OCR1 preload value is loaded in the active register at each Update event (when the counter reaches the autoreload value or when an update is generated by software or by the slave mode controller). **Note:** It is mandatory to enable the preload register in order to use PWM mode otherwise operation is not guaranteed. The preload register is not required in One Pulse Mode (OPM bit set in the TIM_CR register).

10.10.5 Request Selection Register (TIM_RSR)

Address Offset: 18h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved						OC2RS	OC1RS	OCRM	reserved			IC2RS	IC1RS	res.	URS
						rw	rw					rw	rw		rw

Bit 15:10	Reserved: must be kept cleared.
Bit 9	**OC2RS**: *OC2 Request Selection* 0: OC2 DMA request selected. 1: OC2 Interrupt request selected.
Bit 8	**OC1RS**: *OC1 Request Selection* 0: OC1 DMA request selected. 1: OC1 interrupt request selected.
Bits 7	**OCRM**: *Output Compare Request Mode* 0: when enabled, OCx DMA requests are generated by the result of the comparison between the counter and the corresponding output compare register TIM_OCRx. 1: when enabled, OCx DMA requests are generated by the update event.
Bits 6:4	Reserved: must be kept cleared.
Bit 3	**IC2RS**: *IC2 Request Selection* 0: IC2 DMA request selected. 1: IC2 interrupt request selected.
Bit 2	**IC1RS**: *IC1 Request Selection* 0: IC1 DMA request selected. 1: IC1 interrupt request selected.
Bit 1	Reserved: must be kept cleared.
Bit 0	**URS**: *Update Request Selection* 0: Update DMA request selected. 1: Update interrupt request selected.

10.10.6 Request Enable Register (TIM_RER)

Address Offset: 1Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved						OC2RE	OC1RE	reserved				IC2RE	IC1RE	res.	URE
						rw	rw					rw	rw		rw

Bit 15:10	Reserved: must be kept cleared.
Bit 9	**OC2RE**: *OC2 DMA/Interrupt Request Enable* 0: OC2 DMA/interrupt request disabled. 1: OC2 DMA/interrupt request enabled. A request is generated if the OC2I flag is set.
Bit 8	**OC1RE**: *OC1 DMA/Interrupt Request Enable* 0: OC1 DMA/Interrupt Request disabled. 1: OC1 DMA/interrupt request enabled. A request is generated if the OC1I flag is set.
Bits 7:4	Reserved: must be kept cleared.
Bit 3	**IC2RE**: *IC2 DMA/Interrupt Request Enable* 0: IC2 DMA/interrupt request disabled. 1: IC2 DMA/interrupt request enabled. A request is generated if the IC2I flag is set.
Bit 2	**IC1RE**: *IC1 DMA/Interrupt Request Enable* 0: IC1 DMA/interrupt request disabled. 1: IC1 DMA/interrupt request enabled. A request is generated if the IC1I flag is set.
Bit 1	Reserved: must be kept cleared.
Bit 0	**URE**: *Update DMA/Interrupt Request Enable* 0: Update DMA/interrupt request disabled. 1: Update DMA/interrupt request enabled. A request is generated if the UI flag is set.

10.10.7 Interrupt Status Register (TIM_ISR)

Address Offset: 20h
Reset value: 0300h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved						OC2I	OC1I	reserved				IC2I	IC1I	res.	UI
						rc_w0	rc_w0					rc_w0	rc_w0		rc_w0

Bit 15:10	Reserved: must be kept cleared.
Bit 9	**OC2I**: *OC2 Interrupt Flag* - If OCRM=0 or OC2RS=1: This flag is set by hardware when an OC2 match occurs (when the content of the counter reaches the value in the TIM_OCR2 register). It is cleared by software or by the DMA controller. **In PWM mode** (OC2C = 110b in the TIM_OMR1 register): when upcounting, the match occurs when TIM_CNT ≥ TIM_OCR2. When down-counting, the match occurs when TIM_CNT ≤ TIM_OCR2. In Center-aligned mode, the match depends on the CMS bits in the TIM_CR register. **In other modes** (OC2C < 110b in the TIM_OMR1 register): a match occurs when TIM_CNT=TIM_OCR2 0: No match. 1: An OC2 match occurred. - If OCRM=1 and OC2RS=0: This flag is set by hardware when an update event occurs, in the same conditions as UI flag. It is cleared by hardware by the DMA controller. 0: No update occurred 1: An update event occurred. If enabled (OC2RE=1), a DMA request has been sent to the DMA controller.
Bit 8	**OC1I**: *OC1 Interrupt Flag* - If OCRM=0 or OC1RS=1: This flag is set by hardware when an OC1 match occurs (when the content of the counter reaches the value in the TIM_OCR1 register). It is cleared by software or by the DMA controller. **In PWM mode** (OC1C = 110b in the TIM_OMR1 register): when upcounting, the match occurs when TIM_CNT ≥ TIM_OCR1. When down-counting, the match occurs when TIM_CNT ≤ TIM_OCR1. In Center-aligned mode, the match depends on the CMS bits in the TIM_CR register. **In other modes** (OC1C < 110b in the TIM_OMR1 register): a match occurs when TIM_CNT=TIM_OCR1 0: No match. 1: An OC1 match occurred. - If OCRM=1 and OC1RS=0: This flag is set by hardware when an update event occurs, in the same conditions as UI flag. It is cleared by hardware by the DMA controller. 0: No update occurred 1: An update event occurred. If enabled (OC1RE=1), a DMA request has been sent to the DMA controller.
Bits 7:4	Reserved: must be kept cleared.

Bit 3	**IC2I**: *IC2 Interrupt Flag* This bit is set by hardware on a capture. It is cleared by software or by the DMA controller. 0: No input capture occurred. 1: The counter value has been captured in TIM_ICR2 register (An edge has been detected on IC2 which matches the selected polarity).
Bit 2	**IC1I**: *IC1 Interrupt Flag* This bit is set by hardware on a capture. It is cleared by software or by the DMA controller. 0: No input capture occurred. 1: The counter value has been captured in TIM_ICR1 register (An edge has been detected on IC1 which matches the selected polarity).
Bit 1	Reserved: must be kept cleared.
Bit 0	**UI**: *Update Interrupt Flag* This bit is set by hardware when an Update event occurs (depending on UFS bit in TIM_CR register, please refer to *Section 9.6.1: Control Register (TB_CR) on page 190*. It is cleared by software or by the DMA controller. 0: No update occurred. 1: Update interrupt pending. This bit is set by hardware when the counter is updated: – At overflow or underflow and, in a timer with a repetition counter, if REP_CNT value = 0 – When CNT is cleared by software using the CNT_RST bit in TIM_CR register (only if UFS=0 in TIM_CR register) – When CNT is cleared by a trigger event (refer to the TIM_SCR register description, only if UFS=0 in TIM_CR register)

Note: When CK_TIM and PCLK have different frequencies, the events which set the flags are generated on CK_TIM and then re-synchronized with the PCLK clock domain (the flag itself is related to PCLK). This means that there is a delay between the actual event and the setting of the flag. If an event is generated by software, for example writing an TIM_OCRx register to generate a compare event or writing the CNT_RST bit in TIM_CR register to generate an update event, there will be a delay before the related flag can be read at '1'. Therefore a minimum delay of 4 PCLK periods must be inserted before trying to clear this flag after generating the event (for instance by adding 2 dummy accesses).

10.10.8 Counter Register (TIM_CNT)

Address Offset: 24h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the contents of the counter.

Note: 1 *The counter doesn't count if the ARR value is 0. No update event is generated in this case.*

2 *If the counter is written with a value higher than the ARR value while it is up-counting (U/D=0 in TIM_CR register), then the counter will count up to FFFF and restart counting from 0 up to ARR.*

3 *When CK_TIM and PCLK have different frequencies:*

- The counter value cannot be accessed directly and is read through a buffer register. The buffer is updated once every 4 CK_TIM clock cycles (max.).
- The CNT_EN bit is not taken into account immediately by the counter. So if the counter is running when you clear the CNT_EN bit, you must insert an additional instruction before reading the counter to get the updated value.
- For the same reason, you must insert a delay before reading the counter after setting the CNT_RST bit.

10.10.9 Prescaler Register (TIM_PSC)

Address Offset: 28h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This value divides the frequency of the internal timer clock (TIM_ICK) output by the clock mode controller. The counter clock frequency is f_{TIM_ICK} / (PSC[15:0]+1).

This register contains the preload value which is loaded in the active prescaler register:

- Permanently if the counter is disabled (CNT_EN = 0 in TIM_CR register)
- Else at each update event (including when the counter is cleared using the CNT RST bit (TIM_CR register) or by the slave mode control block when configured in reset mode - SMS bits = 00 in the TIM_SCR register).

10.10.10 Auto-Reload Register (TIM_ARR)

Address Offset: 30h
Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the preload value to be loaded in the active auto-reload register at the next update event.

However the active register can be directly written by software in the following cases:

- if the counter is not running (CNT EN=0 in TIM_CR register),
- if the counter is running in One Pulse Mode (OPM=1 in TIM_CR register),
- if the counter is running in down-counting mode (CMS='00' and U/D=1 in TIM_CR register)

So care must be taken when switching from up-counting to another mode as the preload and the active registers might be different.

The auto-reload register is a 16-bit register which contains the ARR value to be compared to the counter CNT.

If counter is up-counting in edge-aligned mode, when it reaches the ARR value an Update event is generated. The counter restarts up-counting from 0.

If counter is up-counting in center-aligned mode, when it reaches the ARR value an update event is generated. The counter restarts down-counting from the ARR preload value.

If counter is down-counting in edge-aligned mode, when it reaches 0 an update event is generated. The counter restarts down-counting from the ARR preload value.

If counter is down-counting in center-aligned mode, when it reaches 0 an update event is generated. The counter restarts up-counting from 0.

10.10.11 Output Compare Register 1 (TIM_OCR1)

Address Offset: 34h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the preload value to be loaded in the active output compare 1 register.

It is loaded permanently if the preload feature is disabled by the PLD1 bit in the TIM_OMR1 register. Otherwise it is copied in the active output compare 1 register when an update event occurs.

The active output compare register contains the value to be compared to the CNT counter and used to control the OC1 output signal.

10.10.12 Output Compare Register 2 (TIM_OCR2)

Address Offset: 38h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the preload value to be loaded in the active output compare 2 register.

It is loaded permanently if the preload feature is disabled by the PLD2 bit in the TIM_OMR1 register. Otherwise it is copied in the active output compare 2 register when an update event occurs.

The active output compare register contains the value to be compared to the CNT counter and used to control the OC2 output signal.

10.10.13 Input Capture Register 1 (TIM_ICR1)

Address Offset: 4Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

This 16-bit register contains the counter value captured when an input capture 1 event (IC1) occurs.

It can be read only when the IC1I flag is set in the TIM_ISR register.

10.10.14 Input Capture Register 2 (TIM_ICR2)

Address Offset: 50h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

This 16-bit register contains the counter value captured when an input capture 2 event (IC2) occurs.

It can be read only when the IC2I flag is set in the TIM_ISR register.

10.10.15 DMA Burst Address (TIM_DMAB)

Address Offset: 60h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAB															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

A read or write access to this address is re-directed to the register located at:

(TIM_CR address) + (DBASE * 4) + (DMA_index * 4)

where:

- TIM_CR is the address of the first register in the register block of the timer
- DBASE is the user-configured DMA base address written in the TIM_CR register
- DMA_index is the offset automatically controlled by the DMA transfer.

refer to *Section 10.8.2: DMA access in burst mode on page 218* for more details.

10.11 TIM Register map

Table 44. TIM Register map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	TIM_CR	DBASE					res.	MMS		res.	CMS		U/D	OPM	CNT EN	CNT RST	UFS
4	TIM_SCR	res.	ITS			reserved		TS		reserved			SMS		SME	ENC	
8	TIM_IMCR	reserved										IC2P	IC2E	IC1P	IC1E	IC2S	IC1S
C	TIM_OMR1	reserved		OC2P	OC2E	OC2C[2:0]			PLD2	reserved		OC1P	OC1E	OC1C[2:0]			PLD1
18	TIM_RSR	reserved						OC2RS	OC1RS	OCRM	reserved			IC2RS	IC1RS	res.	URS
1C	TIM_RER	reserved						OC2RE	OC1RE	reserved				IC2RE	IC1RE	res.	URE
20	TIM_ISR	reserved						OC2I	OC1I	reserved				IC2I	IC1I	res.	UI
24	TIM_CNT	Counter Value (CNT15-CNT0)															
28	TIM_PSC	Prescaler (PSC15-PSC0)															
30	TIM_ARR	Auto-Reload (ARR15-ARR0)															
34	TIM_OCR1	Output Compare 1															
38	TIM_OCR2	Output Compare 2															
4C	TIM_ICR1	Input Capture 1															
50	TIM_ICR2	Input Capture 2															
60	TIM_DMAB	Virtual register, dedicated address for DMA burst transfers															

See *Table 2* for the register base addresses.

11 Synchronizable-PWM Timer (PWM)

11.1 Introduction

The Synchronizable-PWM Timer consists of an auto-reload 16-bit up/down counter driven by a programmable prescaler.

It may be used for a variety of purposes, including generation of up to two output waveforms (output compare and PWM) with the complementary output, Repetition Counter and Dead Time Generator.

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the CPU clock prescaler.

11.2 Synchronizable Programmable Timer (SPT) architecture

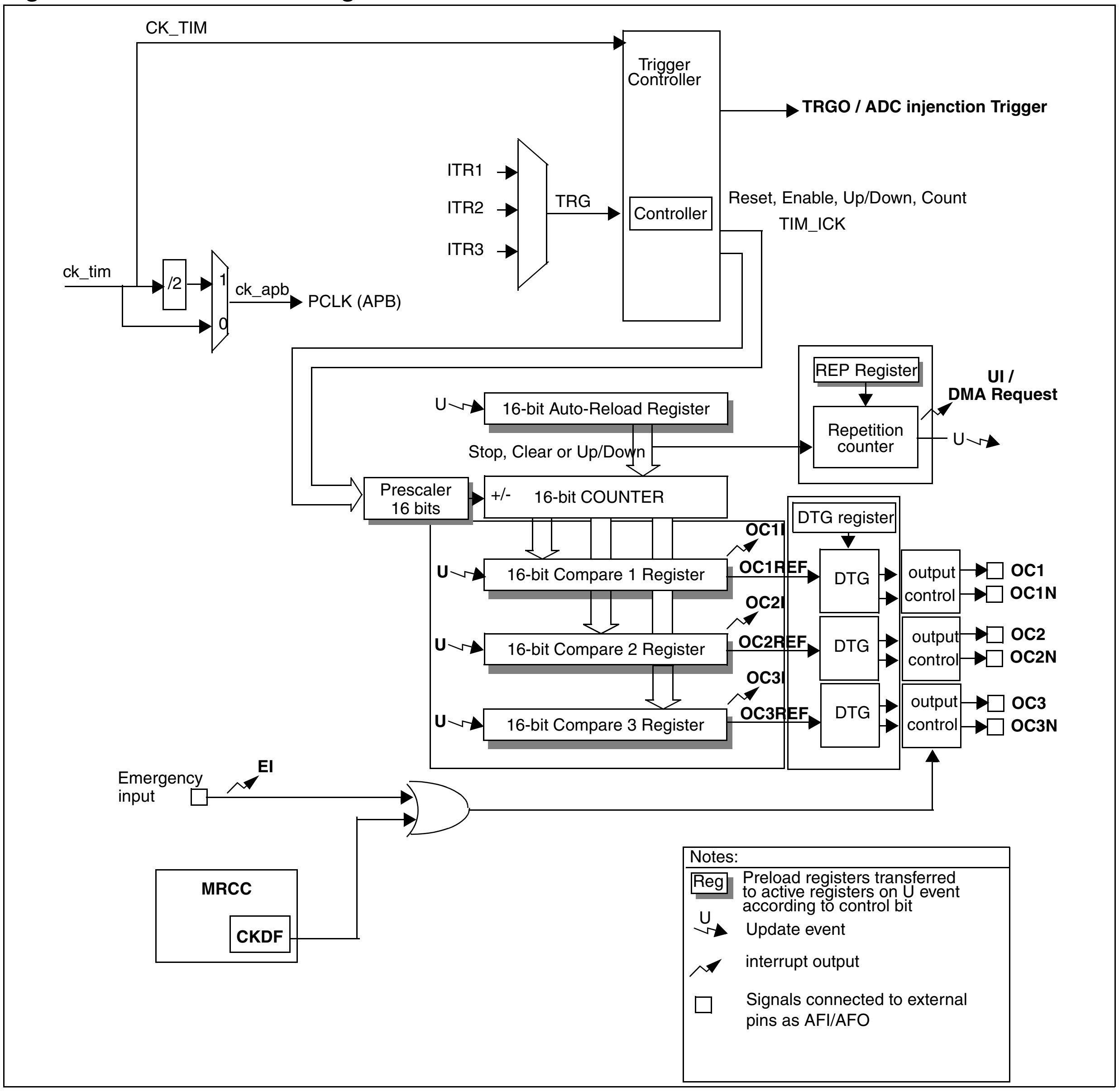
The Synchronizable-PWM Timer forms a family with the TB timer and the TIM timers which are all based on a common architecture. This makes it easier to design applications using the various timers (identical register mapping, common basic features).

In addition the TIM and PWM timers are linked together and can be synchronized.

11.3 General block diagram

The general Block Diagram is shown in *Figure 68*.

Figure 68. General block diagram



11.4 General features

- 16-bit programmable prescaler : f_{CK_TIM} divided by 1 to 65536.
- Autoreload 16-bit up/down counter with update status flag and maskable interrupt.
- Synchronization using the Timer Link System: the timers can be configured as Master Timer or as Slave Timer. In slave mode operations like reset, clocking, start and stop can be controlled by another timer in master mode. A programmable edge detector is associated with each external input.
- Timer Interrupts (with enable bits):
- UI: Timer Update interrupt.
- Independent DMA Request Generation (with enable bits), asserted on Update Event
- Selectable Timer clock frequency $f_{CK\_TIM} = f_{PCLK}$ or f_{PCLK}x2
- Output compare/PWM functions
- One pulse mode (OPM)
- PWM Mode:
  - Shadow Registers can be enabled by control bits.
  - Full modulation capability (0 and 100% duty cycle), edge or center-aligned patterns
  - Dedicated interrupt for PWM duty cycles updating (PWM Update Interrupt)
  - The output polarity is programmable channel by channel.
  - Edge-aligned pattern or center-aligned pattern (counter or down-counter mode)
- Complementary outputs and Dead-time generator
- Programmable Dead-time insertion between each pair of channels.
- The output polarity is programmable channel by channel.
- An “emergency stop” input pin (active low) asynchronously forces OCx and OCxN outputs in Hi-Z. An interrupt or a DMA request can also be generated.
- Repetition Counter
- Programmable PWM repetition counter allows to update PWM registers only after a given number of cycles.

11.5 Functional description

11.5.1 Counter

The timer is based on a 16-bit up/down-counter (CNT) and its related control registers.

It can be cleared and enabled/disabled by software or by a trigger input signal (refer to *Section 9.5.2 on page 189*). Up-counting or down-counting can be controlled by software or by hardware.

Counter modes

There are three counter modes:

- **Up-counting mode**: The counter counts from 0 to the value in the PWM_ARR register then restarts counting from 0 and generates an Update event. To configure the counter in this mode, write U/D = 0 and CMS = 00 in the PWM_CR register.
  Note: If the timer has a repetition counter, the update event is generated after up-counting is repeated for the number of times programmed in the repetition counter (PWM_RCR register).
- **Down-counting mode**: The counter counts from the value in the PWM_ARR register down to 0 then is re-loaded with the PWM_ARR value and generates an Update event. To configure the counter in this mode, write U/D = 1 and CMS = 00 in the PWM_CR register.
  Note: If the timer has a repetition counter, the update event is generated after down-counting is repeated for the number of times programmed in the repetition counter (PWM_RCR register).
- **Center-aligned mode**: The counter counts up from 0 to (PWM_ARR-1), is re-loaded with the PWM_ARR value and then counts down to 0 before restarting counting. An update event can be generated when the counter is re-loaded with PWM_ARR and when it reaches 0. To configure the counter in this mode, write CMS = 01, 10 or 11 in the PWM_CR register.

Table 45. Counter mode selection

Counter Mode	PWM_CR Register CMS bits	PWM_CR Register U/D bit
Up-Counting Mode	00	1
Down-Counting Mode	00	0
Center aligned Mode	10 01 11	-

In Center-aligned mode or in Encoder Interface mode (see *Figure 43 on page 211*), the U/D bit changes automatically, indicating whether the counter is counting up or down.

Counter Initialization

You can re-initialize the counter (and the prescaler as well) writing a ‘1’ in the CNT_RST bit (which is automatically reset by hardware) in the PWM_CR register. Then, depending on the CMS and U/D bits in the PWM_CR register, the counter is either cleared (in up-counting or center-aligned mode) or re-loaded with the PWM_ARR register value (in down-counting mode). It starts counting as soon as you write a ‘1’ in the CNT_EN bit or if CNT_EN goes high under the control of the Slave Mode block (refer to *Section 10.6 on page 213*).

Interrupt/DMA Requests

When an update event occurs, the UI flag in the PWM_ISR register is set depending on UFS bit status (PWM_CR register). An interrupt is generated if the URS bit in the PWM_RSR register and the URE bit in the PWM_RER register are set. If this condition is false, the interrupt request remains pending until it becomes true. A DMA request is generated if URS bit in PWM_RSR register is reset and URE bit in PWM_RER register is set. If this condition is false, the request remains pending to be issued as soon as it becomes true.

11.5.2 Counter clock selection

The counter clock can be provided by the following signals:

- CK_TIM internal clock
- Internal trigger inputs (ITRx)

The selected signal is fed to the counter through a 16-bit prescaler (PWM_PSC). The original clock signal is divided by 1 to 65536.

Figure 69, *Figure 70*, and *Figure 71* describe the counter timings for different clock ratios.

Figure 69. Counter timing diagram, CK_TIM divided by 2 (PWM_ARR=FFFF)

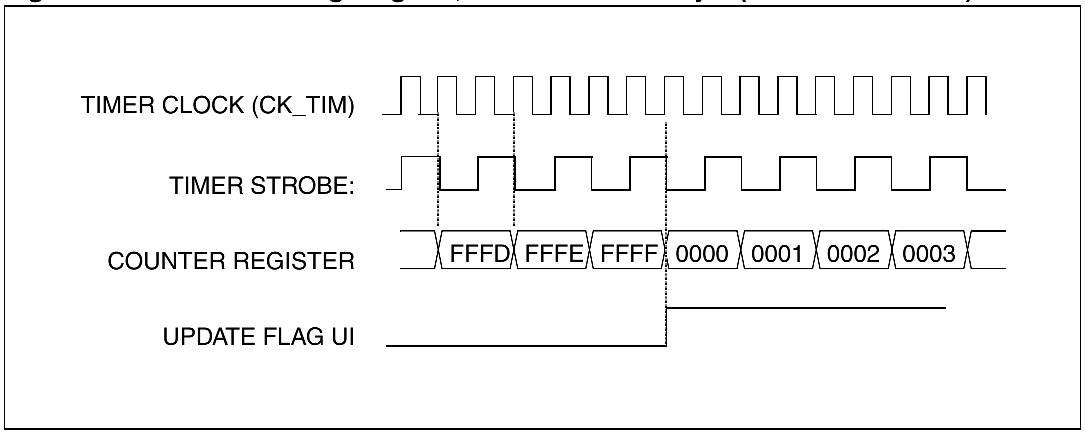


Figure 70. Counter timing diagram, CK_TIM divided by 4

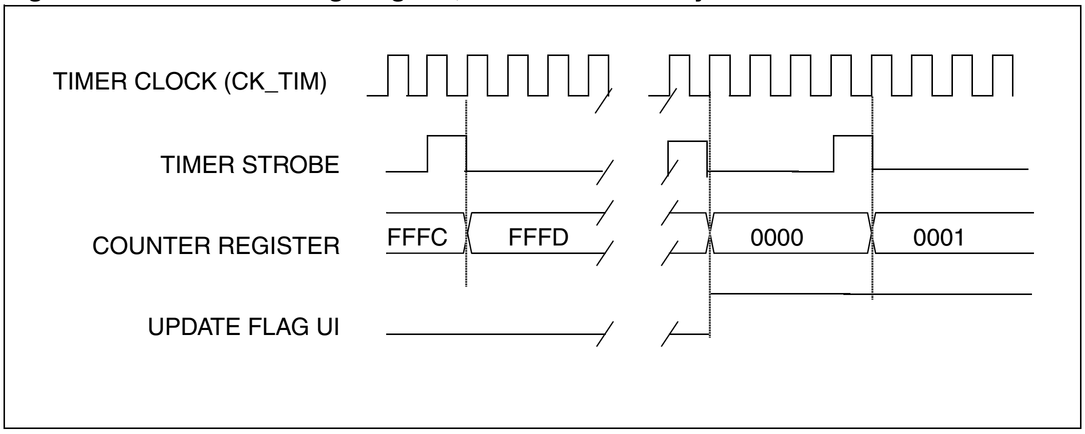
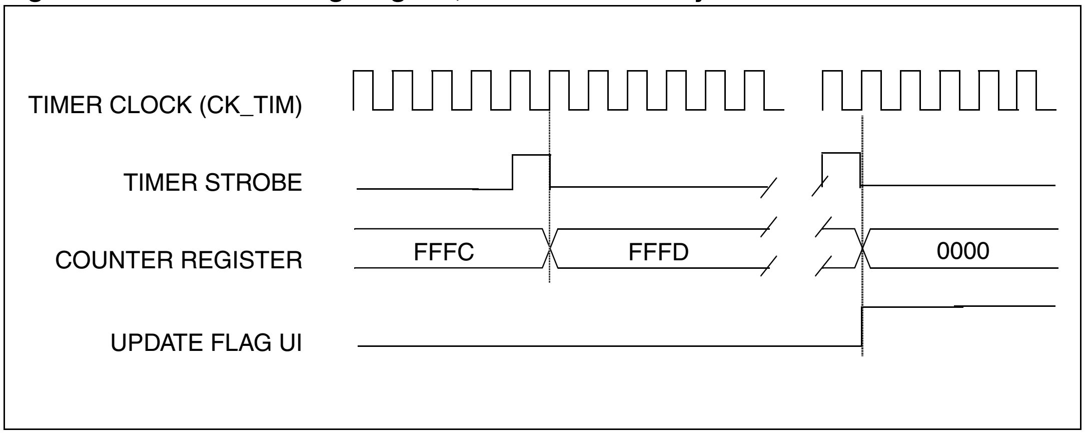


Figure 71. Counter timing diagram, CK_TIM divided by n

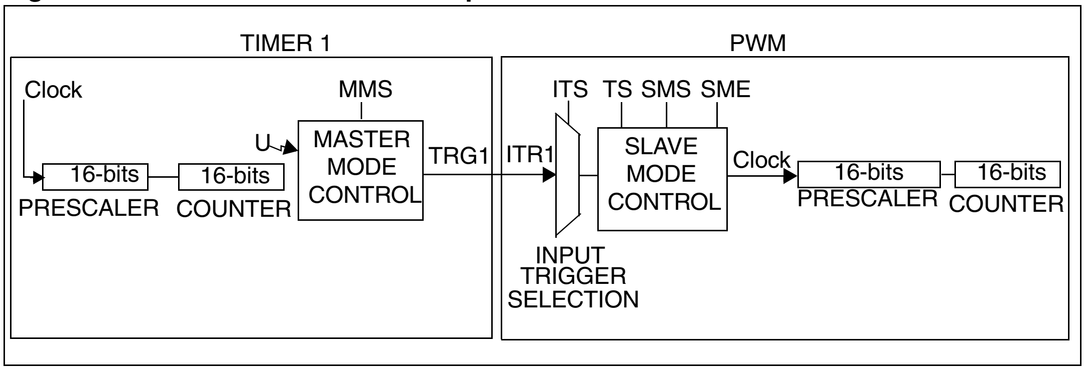


Internal Clock mode

The internal clock (CK_TIM) is selected as counter clock if the Slave mode controller is OFF: SME=0 in the PWM_SCR register.

Using one timer as prescaler for the PWM

Figure 72. Master/Slave timer example



For example, you can configure Timer 1 to act as a prescaler for PWM. Refer to *Figure 56*. To do this:

- Configure Timer 1 in master mode so that it outputs a periodic trigger signal on each Counter Update event. If you write MMS=10 in the TIM1_CR register, a rising edge will be output on TRG1 each time a Counter Update event is generated. Refer to *Figure 56*.
- To connect the TRG1 output of Timer1 to the PWM, PWM must be configured in slave mode using ITR1 as internal trigger.
  You select this through the ITS bits in the PWM_SCR register (writing ITS=001).
  Then you select internal trigger as input to the Slave mode controller (write TS=00 in the PWM_SCR register).
  Slave mode selection must be put in External clock mode (write SMS = 01, SME = 1 in the PWM_SCR register). This will cause the PWM to be clocked by the rising edge of the periodic Timer 1 trigger signal.
- Finally both timers must be enabled by setting their respective CNT_EN bits (TIM1_CR and PWM_CR registers).

Note: If OC1 is selected on Timer 1 as trigger output (MMS=11), its rising edge is used to clock the counter of PWM.

11.5.3 Forced output mode

Each output compare signal (OCx) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal OCx to its active level, you just need to write “101” in OCxC bits in the corresponding PWM_OMRx register. Thus OCREFx is forced inactive (logic 0 as OCREF is always active high) and OCx get opposite value to OCxP.

e.g.: OCxP=0 (OCx active high) => OCx is forced to high level.

OCx output can be forced to its inactive level by writing OCxC bits to “100”.

Anyway the comparison between the output compare register and the counter is still performed and allows the flag to be set as well as interrupt and DMA request to be sent. This is described in the Output Compare Mode section.

11.5.4 Output compare mode

This function is used to control an output waveform or indicating when a period of time has elapsed.

When a match is found between the output compare register (PWM_OCR1) and the counter (CNT), the output compare function:

- Assigns the corresponding pin with a programmable value defined by the Output Compare Control (OCxC bits). The output pin can keep its level, be set, be reset or can toggle on match.
- Set a flag in the interrupt status register (OCxI in PWM_ISR) if OCRM=0 or OCxRS=1 in PWM_RSR register,
- Generates an interrupt if the corresponding interrupt mask is set (OCxRS in PWM_RSR and OCxRE in PWM_RER),
- Send a DMA request if the corresponding enable bit is set (OCRM and OCxRS in PWM_RSR, OCxRE in PWM_RER).

The OCRx registers can be programmed with or without preload-registers using the PLDx bit in the corresponding PWM_OMRx register.

In output compare mode:

The Update event has no effect on OCx output.

The OCx level is updated each time the counter value matches the output compare register depending on the mode selected:

Table 46. Output compare modes

OCxC bits	OCx pin status
000	Doesn’t change
001	Set to active level
010	Set to inactive level
011	Toggles
111	Reserved

Active and inactive levels are defined by OCx polarity. When the OCxP bit in PWM_OMRx register is written to 0, OCx is active high, it is active low when OCxP=1.

Output compare mode can also be used to output a single pulse (See "*Section 10.5.9: One Pulse mode on page 212*.).

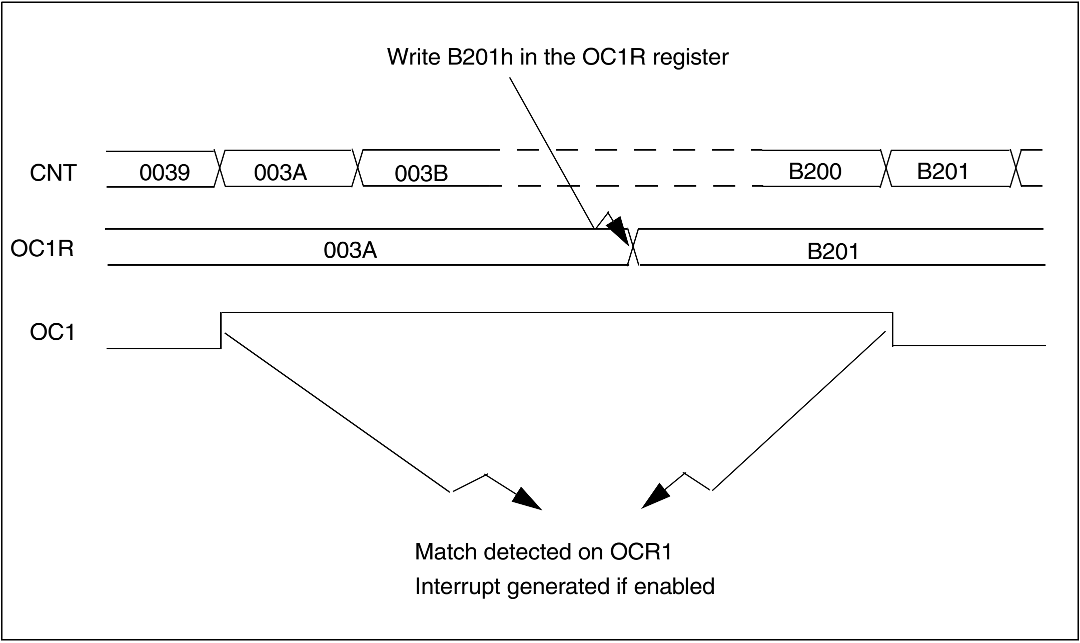
The timing resolution is one count of the counter.

Procedure:

- Select the counter clock (internal/external, prescaler).
- Write the desired data in the PWM_ARR and PWM_OCRx registers.
- Write to the OCxRS bit in the PWM_RSR register to select if an interrupt or a DMA request is to be generated. OCRM must be written to 0 if DMA request is expected.
- Set the OCxRE bit in the PWM_RER enable interrupt or DMA request
- Select the output mode (for example, to select no preload register and to toggle OCx output pin when CNT matches OCRx, write OCxE = '1', OCxP='0', PLDx='0' and OCxC='011').
- Enable Alternate Function on the corresponding I/O port using the GPIO Port Control Registers
- Enable the counter by setting the CNT_EN bit in the PWM_CR register.

PWM_OCRx can be updated at any time by software to control the output waveform provide that the preload register is not enabled (else active OCRx value will change only at the next update event). An example is given in *Figure 9.5.2*.

Figure 73. Output compare mode, toggle on OC1 (OM1='01100' in PWM_OMR1)


Write B201h in the OC1R register
CNT
0039
003A
003B
B200
B201
OC1R
003A
B201
OC1
Match detected on OCR1
Interrupt generated if enabled

11.5.5 PWM mode

Pulse Width Modulation mode allows you to generate a signal with a frequency determined by the value of the PWM_ARR register and a duty cycle determined by the value of the PWM_OCRx register.

The **PWM Mode** can be selected independently on each channel (one PWM per OCx output) by writing ‘110’ in the OCxC control bits in the PWM_OMRx register. You must enable the corresponding preload register by setting the PLDx bit in the PWM_OMRx register.

As a general rule it is mandatory to enable the preload registers to generate PWMs correctly. Preload registers act on the output compare registers (programmable channel by channel), the auto-reload register (in up-counting or center-aligned modes), the repetition register (if available) and the prescaler.

As the preload registers are transferred to the active registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the CNT_RST bit in the PWM_CR register. You can set the CNT_RST and the CNT_EN bits at the same time.

OCx polarity is software programmable using the OCxP bit in PWM_OMRx. It can be programmed as active high or active low.

OCx output enable is controlled by the OCxE bit in the corresponding PWM_OMRx register when the respective complementary output OCxN is not implemented, else by a combination of OCxE, OCxNE, MOE, OSSI and OSSR bits (PWM_OMRx and PWM_DTR registers). Refer to the OMRx register descriptions for more details.

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the PWM_CR control register. Refer to *Figure 47* for the corresponding counting sequence.

Table 47. Counting sequence in center-aligned and edge-aligned mode

center-aligned mode	0	1	2	....	15	16	15	....	2	1	0	1
						T						
edge-aligned mode	0	1	2	.....	15	16	0	1	.....	16	0	1
				T								

T = PWM period, Value of 12-bit Compare 0 Register= 16

Edge-aligned mode

- Up-counting configuration

Up-counting is active when the U/D bit in the PWM_CR control register is low.

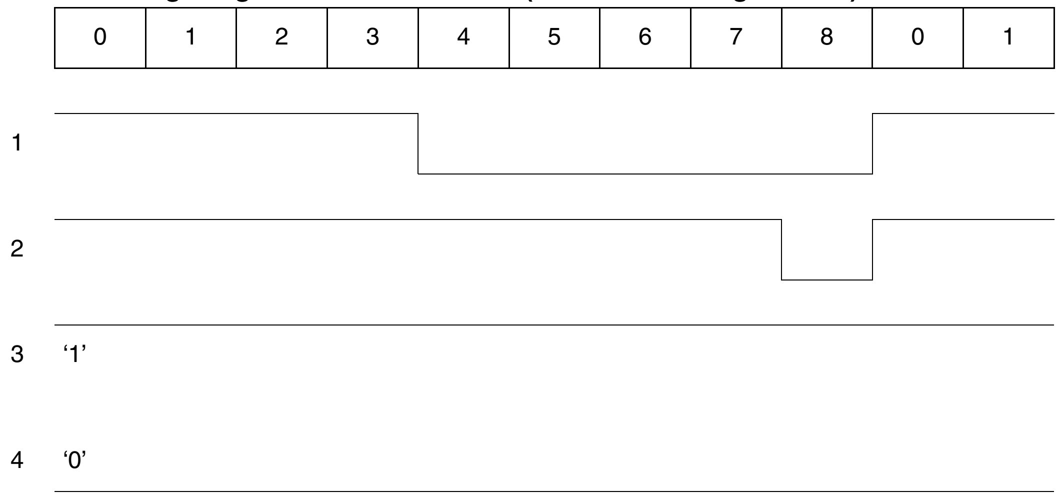
In this mode, the PWM counter counts up to the value loaded in the auto-reload register (PWM_ARR). Then it is cleared and restarts counting.

The reference PWM signal OCxREF is high (meaning that the channel is active) as long as PWM_CNT<PWM_OCRx else it becomes low (inactive). OCx follows this pattern taking into account its polarity configured with OCxP bit in PWM_OMRx register.

If the compare value in PWM_OCRx is greater than the auto-reload value (in PWM_ARR) then OCxREF will be held at ‘1’. If the compare value is 0 then OCxREF will be held at ‘0’.

The following figure shows some edge-aligned PWM waveforms in an example where PWM_ARR=8.

Table 48. Edge-aligned PWM waveforms (Auto-Reload Register = 8)



1 Compare Register value = 4
2 Compare Register value = 8
3 Compare Register value > 8
4 Compare Register value = 0

- Down-counting configuration

Down-counting is active when U/D bit in PWM_CR control register is high.

In this operating mode, the PWM counter counts from the value loaded in the auto-reload register (PWM_ARR) down to 0. Then it is reloaded with the auto-reload value and restarts down-counting.

The reference PWM signal OCxREF is low as long as PWM_CNT>PWM_OCRx else it becomes high.

If the compare value in PWM_OCRx is greater than the auto-reload value (in PWM_ARR) then OCxREF will be held at ‘1’.

0% PWM is not possible in this mode.

Center-aligned mode

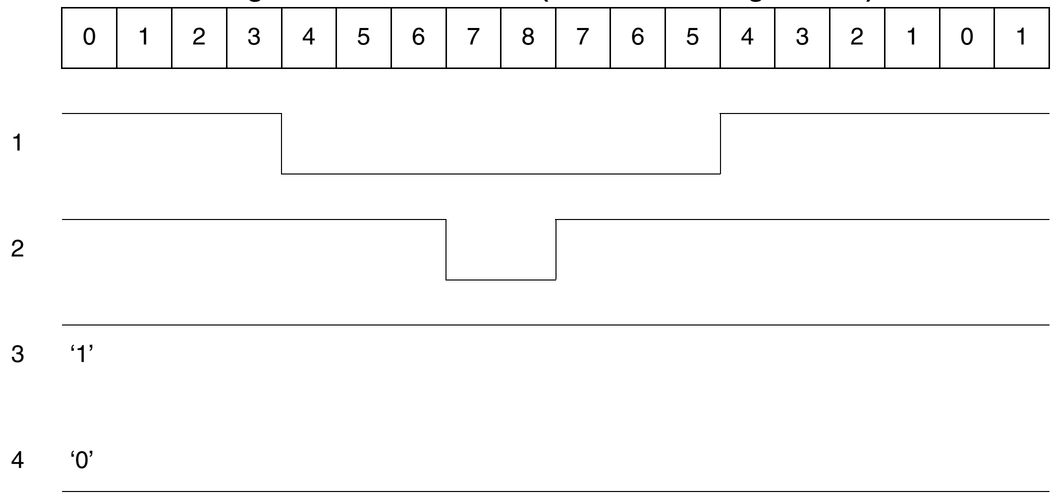
In this mode the counter counts up to the value loaded in the auto-reload register PWM_ARR then counts down until it reaches 0 and restarts counting up. U/D bit in the PWM_CR register is updated by hardware and must not be changed by software.

The reference PWM signal OCxREF is set to ‘0’ (inactive channel) when the counter reaches, in up-counting, the compare value (in PWM_OCRx) and it is set to ‘1’ (active channel) when the counter reaches the compare value again in down-counting. OCx output will follow this pattern taking in account its polarity configured with OCxP bit in PWM_OMRx register.

If the compare value in PWM_OCRx is greater than the auto-reload value (in PWM_ARR) then OCxREF will be held at ‘1’. If the compare value is 0 then OCxREF will be held at ‘0’.

Figure 49 shows some center-aligned PWM waveforms in an example where PWM_ARR=8.

Table 49. Center-aligned PWM Waveforms (Auto-Reload Register = 8)



1 Compare Register value = 4
2 Compare Register value = 7
3 Compare Register value > = 8
4 Compare Register value = 0

Depending on the CMS bits in the PWM_CR register, the DMA/Interrupt flags related to output compare channels (OCxI) are set on the OCxREF rising edge, falling edge or both (refer to the PWM_CR register description).

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 U/D bit in the PWM_CR register. Moreover the U/D 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 Up/Down configuration is not updated if you write a value in the counter that is greater than the auto-reload value (PWM_CNT>PWM_ARR). For example, if the counter was counting up, it will continue to count up.
  - The Up/Down configuration is updated if you write '0' or write the PWM_ARR value in the counter but no Update event is generated (and nor any Update interrupt or DMA request).
- The safest way to use center-aligned mode is to generate an update by software (setting the CNT_RST bit in the PWM_CR register) just before starting the counter and not to write the counter while it is running.

11.5.6 PWM with Complementary Outputs and Dead-Time Insertion

This mode allows the timer to output two complementary signals and to manage the time between the switching-off and the switching-on instants of the outputs.

This time is usually known as dead-time and you have to adjust it depending on the devices you have connected to the PWM outputs and their characteristics (intrinsic delays of level-shifters, delays due to power switches...)

You can select the polarity of the outputs (main output or complementary) independently for each output. This is done by writing to the OCxP and OCxNP bits in the PWM_OMRx register.

The complementary signals OCx and OCxN are activated by a combination of several control bits: the MOE, OSSI and OSSR bits in the PWM_DTR register and the OCxE and OCxNE bits in the PWM_OMRx register. Please refer to *Table 9.6.4: Request Selection Register (TB_RSR) on page 192* for more details.

Dead-time insertion is enabled by setting the MOE bit in the PWM_DTR register and both OCxE and OCxNE bits in the PWM_OMRx register. There is one 10-bit dead-time generator for each channel. From a reference waveform OCxREF, it generates 2 outputs OCx and OCxN where x is the number of the channel. If OCx and OCxN are active high (or non-inverted):

- The OCx output signal is the same as the reference signal except for the rising edge, which is delayed relative to the reference signal rising edge.
- The OCxN output signal is the opposite of the reference signal except for the rising edge, which is delayed relative to the reference signal falling edge.

If the delay is greater than the width of the active output (OCx or OCxN) then the corresponding pulse is not generated.

The following figures show the relationships between the output signals of the dead-time generator and its inputs (the MOE bit in the PWM_DTR register as well as the OCxE and OCxNE bits are high on these examples).

Figure 74: PWM mode, OCxC=110, OCxP=0, OCxNP=0 in PWM_OMR register, OSSI=1 in PWM_DTR register. In this case: if OSSI=0 in PWM_DTR register, OCxN is no longer driven by the timer.

Figure 75: PWM mode, OCxC=110, OCxP=0, OCxNP=0 in PWM_OMR register

Figure 76: PWM mode, OCxC=110, OCxP=0, OCxNP=0 in PWM_OMR register

Figure 77: PWM mode, OCxC=110, OCxP=0, OCxNP=0 in PWM_OMR register

Figure 74. Complementary output waveforms when dead-time is disabled

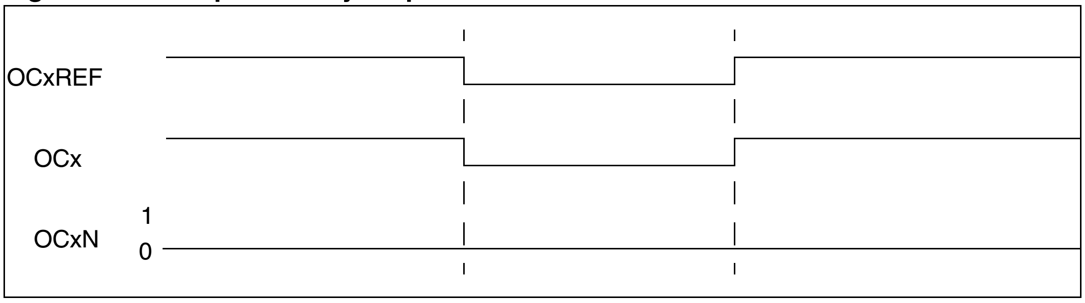


Figure 75. Complementary output waveforms when dead-time is enabled

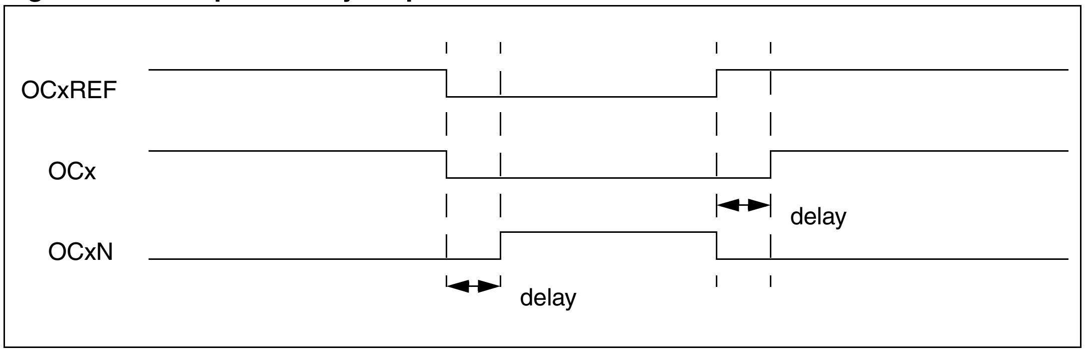


Figure 76. Dead-time waveforms with delay greater than the negative pulse

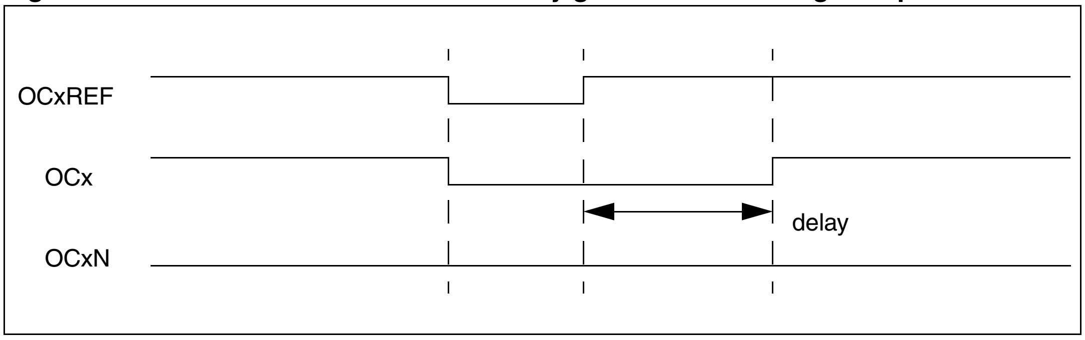
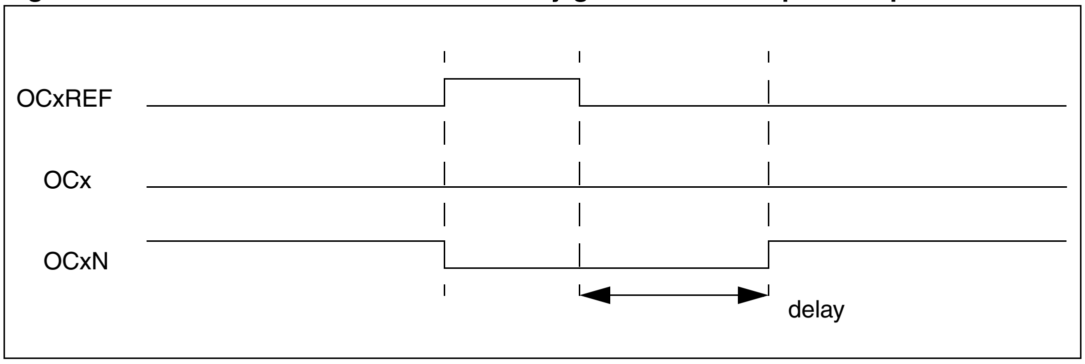


Figure 77. Dead-time waveforms with delay greater than the positive pulse



The dead-time delay is the same for each of the channels and is programmable with the DTG bits in the MDTR register.

The dead-time duration can be calculated using the formula: $DT = \frac{DTG[9,0]}{f_{CKTIM}}$

Parameter locking

To safeguard the application, a write protection has been implemented which allows you to freeze the configuration of several parameters (Dead-time duration, OCx/OCxN polarities, OCx/OCxN configuration when disabled, OCxC configurations). You can choose from 3 levels of protection selected by the LOCK bits. Refer to *Section 11.10.15: Dead-Time Register (PWM_DTR) on page 271*. The LOCK bits can be written only once after an MCU reset.

Emergency input control

An emergency input is also implemented with the dead-time generator (acting only on channels where complementary outputs are implemented). It is active at low level and leads to the following behavior:

- The MOE bit in PWM_DTR register is reset, putting the outputs off or in reset state asynchronously, selected by the OSSI bit in PWM_DTR register. This feature functions even if the MCU oscillator is off.
- An interrupt or a DMA request can be generated if the ERE bit in the PWM_RER register is set. You can select either interrupt or DMA request by configuring the ERS bit in the PWM_RSR register.

When exiting from reset, the emergency circuit is disabled. You must set the EEN bit in the PWM_DTR register to enable it. EEN can no longer be written as soon as LOCK bits are configured at level 1, 2 or 3, so that the emergency feature cannot be disabled by mistake once it has been activated.

You can connect the Emergency input to an alarm signal from power drivers, thermal sensors or any other security components.

11.5.7 Re-directing OCxREF to OCx or OCxN

In forced output mode as well as in output compare or in PWM mode, OCxREF can be re-directed to the OCx output or to OCxN output by configuring the OCxE and OCxNE bits in the corresponding PWM_OMR register.

This allows you to send a specific waveform (such as PWM or static active level) on one output while the complementary output remains at its inactive level. Other alternative possibilities are to have both outputs at inactive level or both outputs active and complementary with dead-time.

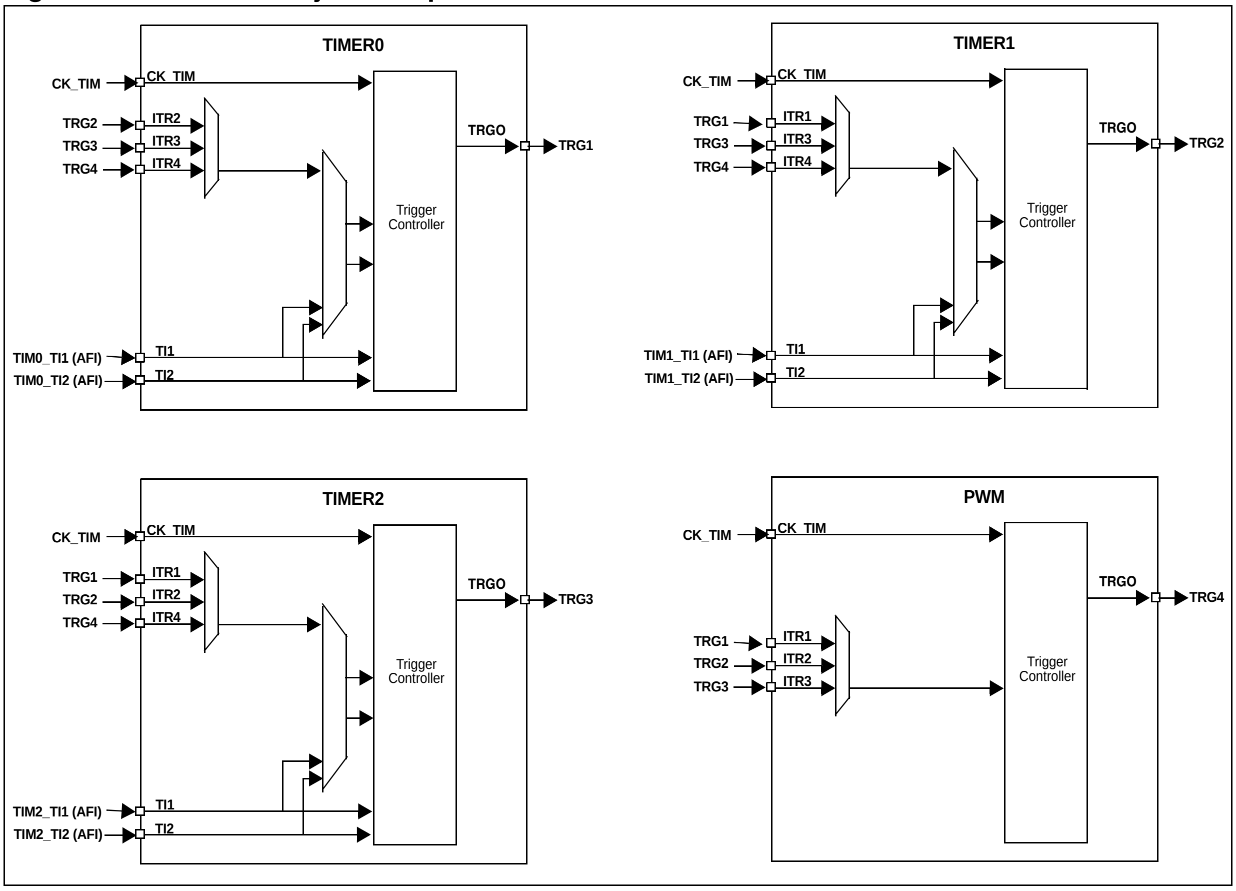
Note: When only OCxN is enabled, it becomes active as soon as OCREF is active. For example, if OCxNP=0 then OCxN=OCREF. On the other hand, when both OCx and OCxN are enabled (OCxE=OCxNE=1, dead-time enabled) OCx becomes active when OCREF is active whereas OCxN becomes active when OCREF is inactive (it is inverted to become complementary to OCx).

11.6 PWM & Timer synchronization

11.6.1 Timer Link system

The PWM and the three Timers are linked together internally for timer synchronization or chaining. Refer to *Figure 78*. Using the Timer Link system, a Timer or a PWM configured in Master Mode can reset, start, stop or clock the counter of another Timer or the PWM configured in Slave Mode.

Figure 78. Timer Link system implementation



11.6.2 Synchronization of Timers and PWM

You can synchronize a timer and the PWM together using two different modes:

- Gated Mode
- Trigger Mode

The following sections use the example of Timer 2 being enabled by PWM in these two modes.

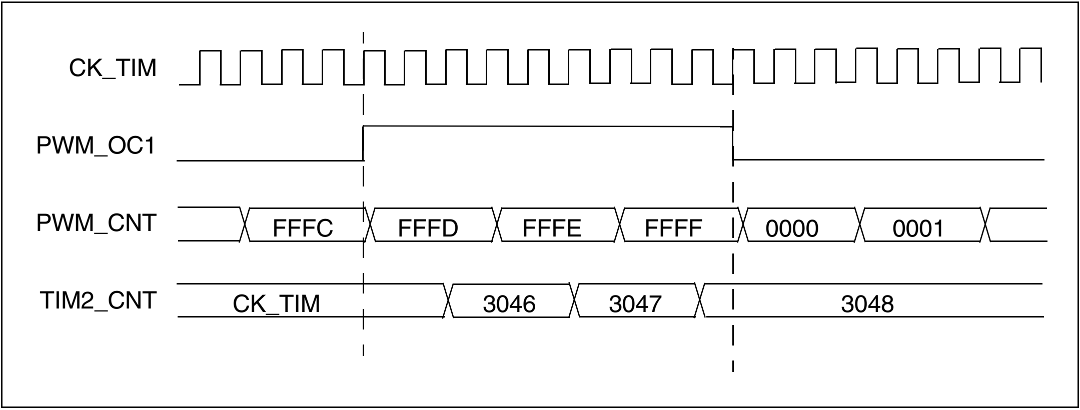
Gated mode

In this example, configure Timer 2 in gated mode by writing SMS=10 and SME=1 in TIM2_SCR register. Then select external clock as the input source by writing the corresponding values in the TS and ITS bits. Once Timer 2 is configured in gated mode, you then enable it by setting the CNT_EN bit in TIM2_CR register. The counter won't start until the trigger input is at active level.

You can choose to start Timer 2 when the Output Compare 1 (OC1) output of PWM is high. To do this, Timer 2 registers must be programmed with: SMS=10, SME=1, TS=00, ITS=001 and CNT_EN = 1 and PWM must be programmed with MMS=11 and CNT_EN = 1. *Figure 63* shows how this works. In this example the prescaler is 2 for both counters ($f_{CK\_CNT} = f_{CK\_TIM}/3$).

Note: The counter 2 clock is not synchronized with counter 1, this mode only affects the Timer 2 counter enable signal.

Figure 79. Gating timer 2 with OC1 of PWM

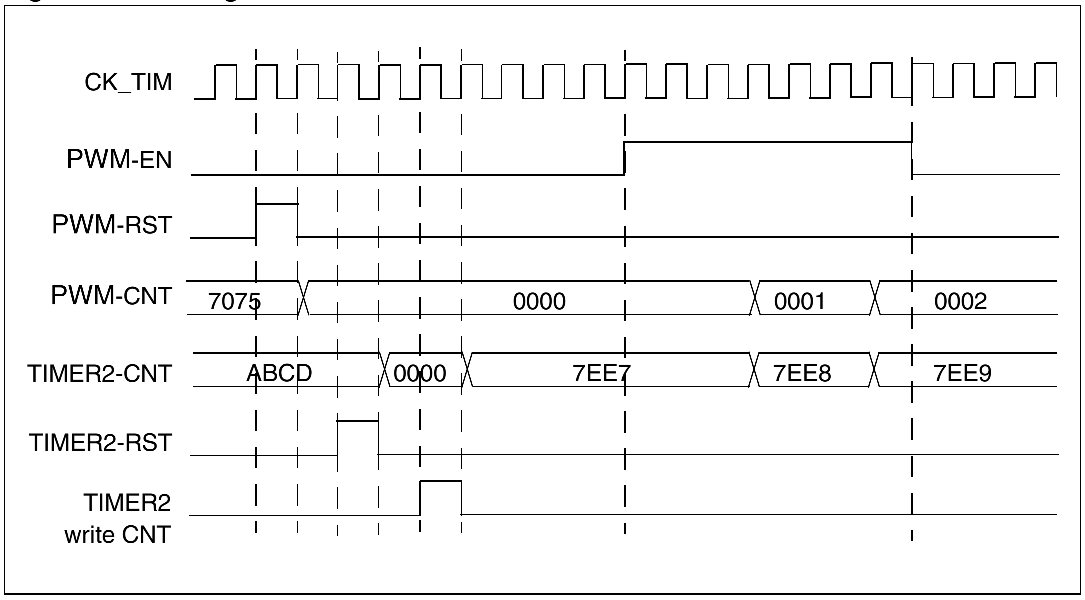


In the example in *Figure 63*, the Timer 2 counter and prescaler are not initialized before being started. So they start counting from their current value. It is possible to start from a given value by resetting both timers before starting PWM. You can then write any value you want in the timer counters. The timers can easily be reset by software using the CNT_RST bit in the TIM_CR registers.

In the next example, we synchronize PWM and Timer 2. PWM is the master and starts from 0000. Timer 2 is the slave and starts at the same time from 7EE7h. The prescaler ratio is the same for both timers. Timer 2 stops when PWM is disabled by writing '0' to the CNT_EN bit in the PWM_CR register:

- Configure PWM master mode to send its enable command as trigger output (MM=01 in the PWM_CR register).
- Configure Timer 2 to get the input trigger from PWM (ITS = '001' and TS='00' in the TIM2_SCR register).
- Configure Timer 2 in gated mode (SMS='10', SME='1' in the TIM2_SCR).
- Reset PWM by writing '1' in CNT_RST bit (PWM_CR register).
- Reset Timer 2 by writing '1' in CNT_RST bit (TIM2_ CR register).
- Initialize Timer 2 to 7EE7h by writing '7EE7h' in the Timer 2 counter (TIM2_CNT).
- Enable Timer 2 by writing '1' in the CNT_EN bit (TIM2_CR register).
- Start PWM by writing '1' in the CNT_EN bit (PWM_CR register).
- Stop PWM by writing '0' in the CNT_EN bit (PWM_CR register).

Figure 80. Gating Timer 2 with ENABLE of PWM

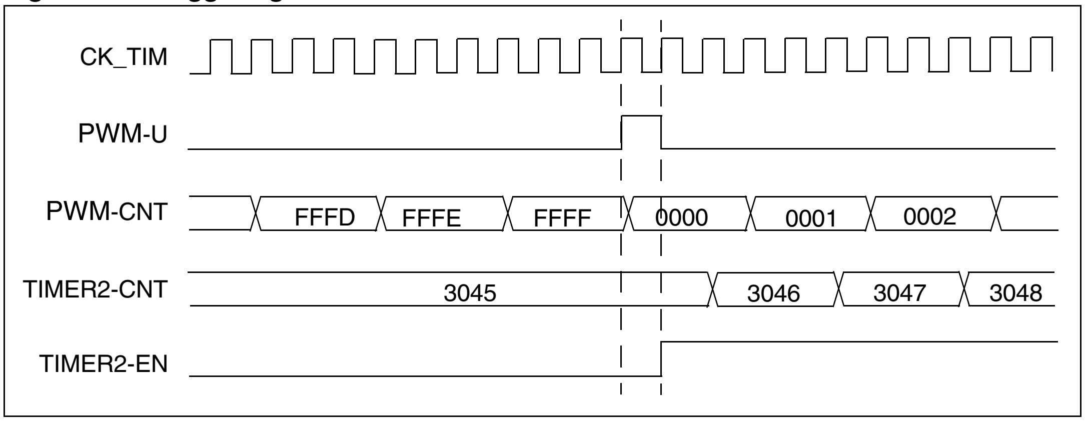


Trigger mode

In this next example, we configure Timer 2 in trigger mode by writing SMS='11' and SME='1' in the TIM2_CR register. As in the previous example, we select the input source using the TS and ITS bits.

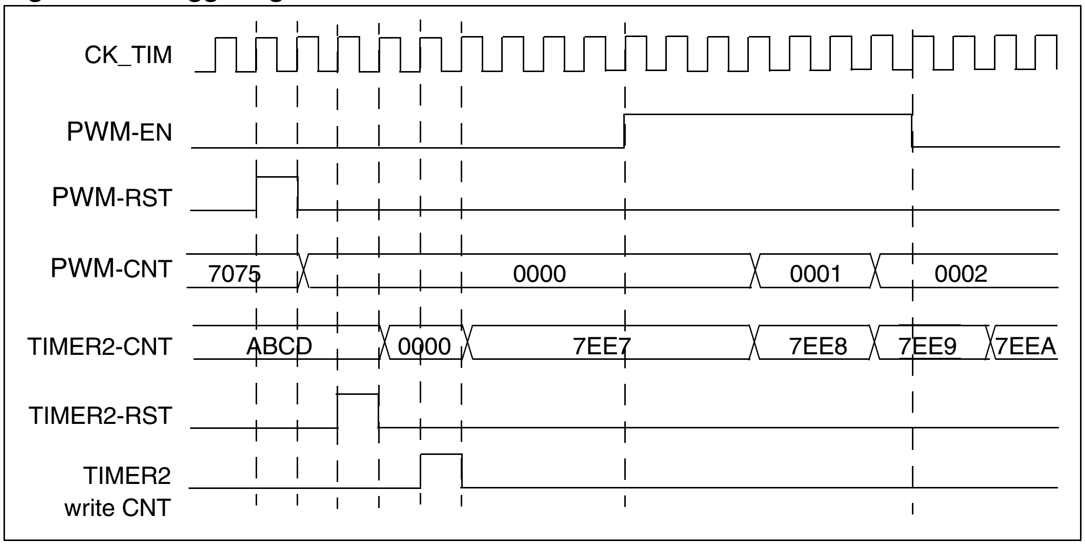
In our example, Timer 2 starts counting from its current value (which can be non-zero) as soon as an Update event is generated by PWM. When Timer 2 receives the trigger signal its CNT_EN bit is automatically set and the counter counts until we write '0' to the CNT_EN bit in the TIM2_CR register.

Figure 81. Triggering Timer 2 with UPDATE of PWM



As in the previous example, you can initialize both counters before starting counting. *Figure 66* shows the behavior with the same configuration as in the *Figure 64* but in trigger mode instead of gated mode (SMS='11' and SME='1' in the TIM2_SCR register).

Figure 82. Triggering Timer 2 with ENABLE of PWM



11.7 Interrupt management

To use the interrupt features, for each interrupt channel used, perform the following sequence:

- Set the desired ERS and/or OCRSx and/or ICRSx and/or URS bits to select interrupt request rather than DMA request.
- Set the desired ERE and/or OCREx and/or ICREx and/or URE bits in the PWM_RER register to enable interrupt requests.

11.8 DMA function

11.8.1 Using the DMA feature

- Reset the desired ERS and/or OCRSx and/or ICRSx and/or URS bits to select DMA request rather than interrupt request.
- Set the desired ERE and/or OCREx and/or ICREx and/or URE bits in the PWM_RER register to enable DMA requests.

The DMA request generated as soon as the corresponding flag in the PWM_ISR register) is set. The flag is automatically cleared by hardware by the DMA controller.

A flag must not be written by software when it is controlled by the DMA (when xRS=0 and xRE=1).

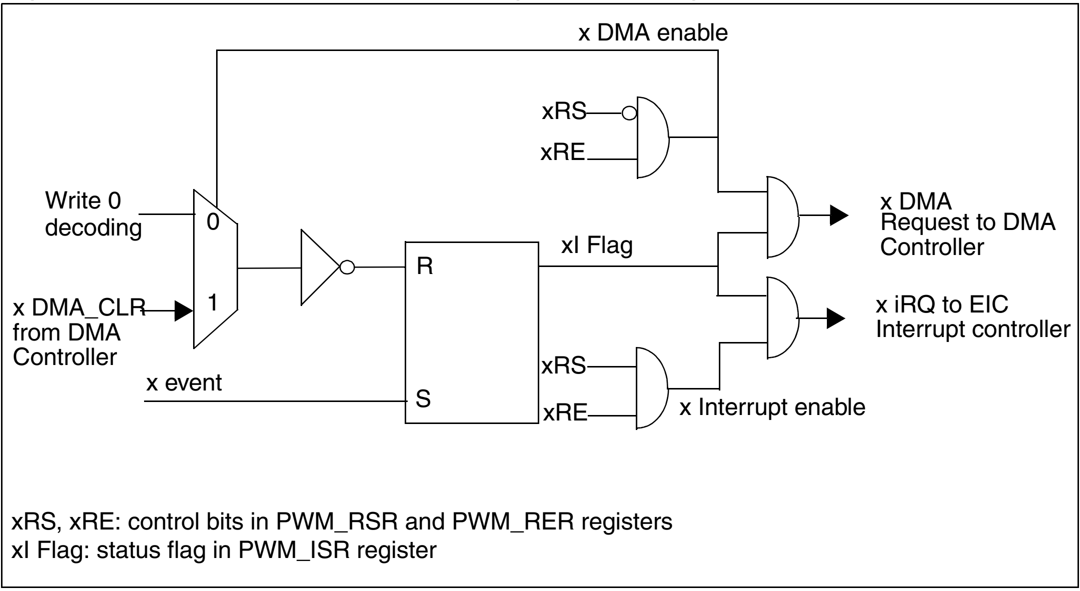
In addition, it is not recommended to modify xRS value while xRE=1.

Moreover, in case of Output Compare requests, the request selection (xRS) and the OC request mode (OCRM) must not be modified while the request is enabled (xRE=1). To change the request configuration when xRE=1:

- Check that no DMA request is pending on the corresponding event
- Clear the enable bit xRE
- Configure the request (xRS and OCRM bits)
- Wait for flag re-synchronization (4 clock cycles)
- Clear the flag by software
- Enable the request by setting the xRE bit

The following figure shows a simplified schematic of flag management

Figure 83. Interrupt/DMA request management for flag "x"


x DMA enable
xRS
xRE
Write 0 decoding
0
1
x DMA_CLR from DMA Controller
x event
R
S
xI Flag
x DMA Request to DMA Controller
x iRQ to EIC Interrupt controller
xRS
xRE
x Interrupt enable
xRS, xRE: control bits in PWM_RSR and PWM_RER registers
xI Flag: status flag in PWM_ISR register

11.8.2 DMA access in burst mode

All the PWM registers can be accessed independently by the DMA controller. A specific register address is used for to burst access (PWM_DMAB: DMA Burst). Any read or write access to this address is redirected to the corresponding register selected by a DMA base-address and a DMA index.

The DMA Base-address is programmed in the DBASE bits in PWM_CR register. DBASE is defined as an offset from the timer base address (DBASE=0 means that PWM_CR is used as DMA base-address. DBASE=1 means that PWM_SCR is used as DMA base-address).

The DMA index adds an offset to the DMA base-address. It is cleared as soon as the DMA request is sent by the timer to prepare the next transfer. Then it is incremented each time the PWM_DMAB register is accessed by the DMA controller (you program the length of the transfer using the DMA controller registers). This means that several consecutive accesses to the PWM_DMAB will access an array of consecutive registers.

In the following example, we want to transfer 2 values of a memory table automatically into the PWM_OCR1 and PWM_OCR2 registers each time an update event occurs:

- Configure the DMA base-address to point to PWM_OCR1 by writing DBASE to ‘01101’ (13th register in the register map),
- Enable DMA requests on Update event by clearing the URS bit in PWM_RSR register and setting the URE bit in PWM_RER register.
- Program the DMA controller so that an update DMA request generates a transfer of 2 half-words from the memory to the PWM_DMAB address (in burst mode).

1. When the update occurs, the UI flag is set in the PWM_ISR register, the request is sent to the DMA controller and the DMA index is cleared (PWM_DMAB is redirected to PWM_OCR1 register).
2. The DMA writes the first value to the PWM_DMAB address and is redirected to write in the PWM_OCR1 register. Then the DMA index is incremented (PWM_DMAB is redirected to PWM_OCR2 register).
3. The DMA writes the second value to the PWM_DMAB address and is redirected to write in the PWM_OCR2 register. As only 2 values are to be transferred, the DMA controller clears the UI flag in PWM_ISR register and stops writing to the PWM_DMAB address.

Note: 1 *Do not write directly in the PWM_DMAB address as this could corrupt the DMA index (if a burst transfer is interrupted).*

2 *The PWM can manage only one burst transfer at a time. This means that you must not program 2 different DMA channels to access PWM_DMAB at the same time (for instance one in response to an update event and another in response to an output compare event). Else the index could be corrupted and access to PWM_DMAB would be redirected to unexpected registers.*

11.9 Debug mode

When the microcontroller enters debug mode, the timer either continues to work normally or stops its activity, depending on the DBGC bit in PWM_CR register.

If DBGC=0 (reset state), the timer detects any activity on the debug acknowledge (DBGACK) signal which is directly sent by the CPU. Then, as soon as the device enters debug mode:

- The PWM_CNT counter stops counting
- The trigger input is blocked
- The MOE bit (if implemented) becomes inactive and the OCx/OCxN outputs behave as if MOE=0 (but the state of MOE doesn’t change). DBGACK has no effect on OCx outputs when the MOE bit and the OCxN complementary outputs are not implemented.

Thus, when a breakpoint occurs, the state of all the registers is kept stable.

If DBGC=1, the timer ignores the DBGACK signal.

11.10 Register description

All can have only be accessed 16-bit access. A byte cannot be read or written. In this section, the following abbreviations are used:

read/write (rw)	Software can read and write to these bits.
read-only (r)	Software can only read these bits.
read/clear (rc_w0)	Software can read as well as clear this bit by writing 0. Writing ‘1’ has no effect on the bit value.
Write once-only (wo)	Software can read these bits. It can write them only once. Once write-accessed, these bits cannot be re-written unless the processor is reset.

11.10.1 Control Register (PWM_CR)

Address Offset: 00h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DBASE					DBGC	MMS		res.	CMS		U/D	OPM	CNT_EN	CNT_RST	UFS
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 15:11	**DBASE:** *DMA Base-address* This 5-bit vector defines the base-address for DMA transfers in burst mode (when read/write access are done through the PWM_DMAB address). DBASE is defined as an offset starting from the first address of the timer (which is the address of PWM_CR). 00000: PWM_CR 00001: PWM_SCR 00010: PWM_IMCR ... 10111: PWM_DTR
Bit 10	**DBGC***: Debug Control* See *Section 9.5.3: Debug mode on page 189* for a detailed description. 0: The timer is stopped in debug mode 1: The timer continues working in debug mode

Bits 9:8	**MMS[1:0]**: *Master Mode Selection* These bits select the source of the trigger event sent in master mode to slave timers for synchronization (TRGO). 00: Reset - The CNT_RST bit is used as trigger output (TRGO). **Note:** If the CNT_RST bit is set by an input trigger (timer configured in Slave reset mode by the SMS bits in the PWM_SCR register) then there is a delay between the counter reset and the TRGO event. 01: Enable - The Counter Enable signal is used as trigger output (TRGO). This is useful for starting several timers at the same time or controlling a window in which a slave timer is enabled. **Note:** The Counter Enable signal is generated by a logic OR between the CNT_EN 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. 10: Update - The update event is selected as trigger output (TRGO). For example a master timer can then be used as a prescaler for a slave timer (see *Using one timer as prescaler for another timer on page 202*). 11: Compare - The OC1 signal is used as trigger output (TRGO).
Bit 7	Reserved: must be kept cleared.
Bits 6:5	**CMS:** *Center-Aligned Mode Selection* Do not switch from edge-aligned mode to center-aligned mode while the counter is enabled (CNT_EN=1) 00: Edge-aligned mode (used in PWM mode). 01: Center-aligned mode 1 (used in PWM mode). Output compare interrupt flags of channels configured in PWM mode (OCxC=110 in PWM_OMRx register) are set on rising edges of the corresponding OCREF signal. 10: Center-aligned mode 2 (used in PWM mode). Output compare interrupt flags of channels configured in PWM mode (OCxC=110 in PWM_OMRx register) are set on falling edges of the corresponding OCREF signal. 11: Center-aligned mode 3 (used in PWM mode). Output compare interrupt flags of channels configured in PWM mode (OCxC=110 in PWM_OMRx register) will be set on both rising and falling edges of the corresponding OCREF signal.
Bit 4	**U/D**: *Up/Down* 0: Counter used as up-counter 1: Counter used as down-counter
Bit 3	**OPM**: *One Pulse Mode* 0: Counter is not stopped at update event 1: Counter stops counting at the next update event (resetting the bit CNT EN)
Bit 2	**CNT_EN**: *Counter Enable* 0: Counter disabled 1: Counter enabled **Note:** In External clock, gated mode and encoder interface mode the counter works only if CNT_EN is set by software. However in trigger mode CNT_EN is set automatically by hardware.

Bit 1	**CNT_RST**: *Counter Reset* This bit can be written to 1 to reset the counter, it is automatically reset 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 (however the prescaler preload value is not affected). The counter is cleared if center-aligned mode is selected or if U/D=0 (up-counting), else it takes the auto-reload value (PWM_ARR) if U/D=1 (down-counting)
Bit 0	**UFS**: *Update Flag Selection* 0: UI flag is set when an update event occurs, whatever its source (counter overflow/underflow, CNT RST bit set by software or slave mode controller in reset mode) 1: UI flag is set only if a counter overflow/underflow is detected. It is not affected by updates generated by software or by the slave mode controller)

11.10.2 Synchro Control Register (PWM_SCR)

Address Offset: 04h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved	ITS			reserved		TS		reserved			SMS		SME	res.	
	rw	rw	rw			rw	rw				rw	rw	rw	rw	rw

Bit 15	Reserved: must be kept cleared.
Bits 14:12	**ITS**: *Internal Trigger Selection* Selects the internal trigger to be used to synchronize the counter. These bits must be changed only when they are not enabled (e.g. when SME is low) to avoid wrong edge detections at the transition. 001: Internal Trigger 1 (ITR1) 010: Internal Trigger 2 (ITR2) 011: Internal Trigger 3 (ITR3) 100: Internal Trigger 4 (ITR4)
Bits 11:10	Reserved: must be kept cleared.
Bits 9:8	**TS**: *Trigger Selection* Selects the trigger signal (TRG) to be used by the trigger controller. 00: Internal trigger (output of the internal trigger selection) 10: Input Capture 1 (IC1) 11: Input Capture 2 (IC2)
Bits 7:5	Reserved: must be kept cleared.

Bits 4:3	**SMS**: *Slave Mode Selection* When external signals are selected the active edge of the trigger signal (TRG) is linked to the polarity selected on the external input (see PWM_IMCR and PWM_CR Register description) 00: Reset - rising edge of the selected trigger signal (TRG) resets the counter and generates an update of the registers. 01: External Clock - rising edge of selected trigger (TRG) clocks the counter. 10: Gated Mode: Counter clock is enabled when trigger signal (TRG) 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. 11: Trigger Mode: The counter starts at a rising edge of the trigger TRG (but it is not reset). Only the start of the counter is controlled.
Bit 2	**SME**: *Slave Mode Enable* 0: Slave mode disabled. If CNT_EN = ‘1’ then the prescaler is clocked directly by CK_TIM. 1: Slave mode enable. The counter is controlled by the selected trigger input. Depending on the slave mode selection (SMS) it can be reset, clocked or enabled.
Bits 1:0	Reserved: must be kept cleared.

11.10.3 Output Mode Register 1 (PWM_OMR1)

Address Offset: 0Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2NP	OC2NE	OC2P	OC2E	OC2C[2:0]			PLD2	OC1NP	OC1NE	OC1P	OC1E	OC1C[2:0]			PLD1
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15	**OC2NP**: *OC2N Polarity* 0: OC2N active high 1: OC2N active low **Note:** This bit cannot be modified if LOCK level 2 or 3 is programmed (LOCK bits in PWM_DTR register).
Bit 14	**OC2NE**: *OC2N Enable* 0: Off - OC2N is not active. Refer to *Table 9.6.4* for more details. 1: On - OC2N signal is output on the corresponding output pin.
Bit 13	**OC2P**: *OC2 Polarity* 0: OC2 active high 1: OC2 active low **Note:** This bit cannot be modified if LOCK level 2 or 3 is programmed (LOCK bits in PWM_DTR register).
Bit 12	**OC2E**: *OC2 Enable* 0: Off - OC2 signal is disabled. Refer to *Table 50* and *Table 9.6.4* for more details. 1: On - OC2 signal is output on the corresponding output pin.
Bits 11:9	**OC2C[2:0]**: *OC2 Control* These bits control the output reference signal OC2REF from which the OC2 and OC2N outputs are derived. OC2REF is active high. The active level of the OC2 and OC2N outputs depends on the OC2P and OC2NP bits. **Note:** These bits cannot be modified if LOCK level 3 is programmed (using the LOCK bits in the PWM_DTR register). 000: Frozen - The comparison between the output compare register PWM_OCR2 and the PWM_CNT counter has no effect on the outputs. 001: Set OC2 channel to active level on match. OC2REF signal is forced high when the PWM_CNT counter matches Output Compare Register 2 (PWM_OCR2). 010: Set OC2 channel to inactive level on match. OC2REF signal is forced low when the PWM_CNT counter matches Output Compare Register 2 (PWM_OCR2). 011: Toggle OC2REF signal when PWM_CNT = PWM_OCR2. 100: Force inactive (low) level on OC2REF signal. 101: Force active (high) level on OC2REF signal. 110: PWM mode - In up-counting, OC2 channel is active if PWM_CNT < PWM_OCR2, else inactive. In down-counting, OC2 channel is inactive if PWM_CNT > PWM_OCR2, else active. 111: Reserved

Bit 8	**PLD2**: *Preload Enable 2* 0: Preload register on PWM_OCR2 disabled. PWM_OCR2 can be written at anytime, the new value is taken in account immediately. 1: Preload register on PWM_OCR2 enabled. Read/Write operations access the preload register. PWM_OCR2 preload value is loaded in the active register at each Update event (when the counter reaches the autoreload value or when an update is generated by software or by the slave mode controller). **Note:** It is mandatory to enable the preload register when using PWM mode, otherwise operation is not guaranteed. It is not mandatory in One Pulse Mode (OPM bit set in the PWM_CR register). **Note:** This bit cannot be modified if LOCK level 3 has been programmed (using the LOCK bits in the PWM_DTR register).
Bit 7	**OC1NP**: *OC1N Polarity* 0: OC1N active high 1: OC1N active low **Note:** This bit cannot be modified if LOCK level 2 or 3 has been programmed (using the LOCK bits in the PWM_DTR register).
Bit 6	**OC1NE**: *OC1N Enable* 0: Off - OC1N signal is disabled. Refer to *Table 9.6.4* for more details. 1: On - OC1N signal is output on the corresponding output pin.
Bit 5	**OC1P**: *OC1 Polarity* 0: OC1 active high 1: OC1 active low **Note:** This bit cannot be modified if LOCK level 2 or 3 is programmed (using the LOCK bits in the PWM_DTR register).
Bit 4	**OC1E**: *OC1 Enable* 0: Off - OC1 signal is disabled 1: On - OC1 signal is output on the corresponding channel. **Note:** Refer to *Table 50* and *Table 9.6.4* for more details.

Bits 3:1	**OC1C[2:0]**: *OC1 Control* These bits control the output reference signal OC1REF from which the OC1 and OC1N outputs are derived. OC1REF is active high. The active level of the OC1 and OC1N outputs depends on the OC1P and OC1NP bits. **Note:** These bits cannot be modified if LOCK level 3 is programmed (using the LOCK bits in the PWM_DTR register). 000: Frozen - The comparison between the output compare register PWM_OCR1 and the PWM_CNT counter has no effect on the outputs. 001: Set OC1 channel to active level on match. OC1REF signal is forced high when the PWM_CNT counter matches Output Compare Register 1 (PWM_OCR1). 010: Set OC1 channel to inactive level on match. OC1REF signal is forced low when the PWM_CNT counter matches Output Compare Register 1 (PWM_OCR1). 011: Toggle OC1REF signal when PWM_CNT = PWM_OCR1. 100: Force inactive (low) level on OC1REF signal. 101: Force active (high) level on OC1REF signal. 110: PWM mode - In up-counting, OC1 channel is active if PWM_CNT < PWM_OCR1, else inactive. In down-counting, OC1 channel is inactive if PWM_CNT > PWM_OCR1, else active. 111: Reserved
Bit 0	**PLD1**: *Preload Enable 1* 0: Preload register on PWM_OCR1 disabled. PWM_OCR1 can be written at anytime, the new value is taken in account immediately. 1: Preload register on PWM_OCR1 enabled. Read/Write operations access the preload register. The PWM_OCR1 preload value is loaded in the active register at each Update event (when the counter reaches the autoreload value or when an update is generated by software or by the slave mode controller). **Note:** It is mandatory to enable the preload register in order to use PWM mode otherwise operation is not guaranteed. The preload register is not required in One Pulse Mode (OPM bit set in the PWM_CR register). **Note:** This bit cannot modified if LOCK level 3 has been programmed (LOCK bits in PWM_DTR register).

11.10.4 Output Mode Register 2 (PWM_OMR2)

Address Offset: 10h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								OC3NP	OC3NE	OC3P	OC3E	OC3C[2:0]			PLD3
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8	Reserved: must be kept cleared.
Bit 7	**OC3NP**: *OC3N Polarity* 0: OC3N active high 1: OC3N active low **Note:** This bit cannot be modified if LOCK level 2 or 3 is programmed (using the LOCK bits in the PWM_DTR register).
Bit 6	**OC3NE**: *OC3N Enable* 0: Off - OC3N is not active. Refer to *Table 9.6.4* for more details. 1: On - OC3N signal is output on the corresponding output pin.

Bit 5	**OC3P**: *OC3 Polarity* 0: OC3 active high 1: OC3 active low **Note:** This bit cannot be modified if LOCK level 2 or 3 is programmed (LOCK bits in PWM_DTR register).
Bit 4	**OC3E**: *OC3 Enable* 0: Off - OC3 signal is disabled. Refer to *Table 50* and *Table 9.6.4* for more details. 1: On - OC3 signal is output on the corresponding output pin.
Bits 3:1	**OC3C[2:0]**: *OC3 Control* These bits control the output reference signal OC3REF from which the OC3 and OC3N outputs are derived. OC3REF is active high. The active level of the OC3 and OC3N outputs depends on the OC3P and OC3NP bits. **Note:** These bits cannot be modified if LOCK level 3 is programmed (using the LOCK bits in the PWM_DTR register). 000: Frozen - The comparison between the output compare register PWM_OCR3 and the PWM_CNT counter has no effect on the outputs. 001: Set OC3 channel to active level on match. OC3REF signal is forced high when the PWM_CNT counter matches Output Compare Register 3 (PWM_OCR3). 010: Set OC3 channel to inactive level on match. OC3REF signal is forced low when the PWM_CNT counter matches Output Compare Register 3 (PWM_OCR3). 011: Toggle OC3REF signal when PWM_CNT = PWM_OCR3. 100: Force inactive (low) level on OC3REF signal. 101: Force active (high) level on OC3REF signal. 110: PWM mode - In up-counting, OC3 channel is active if PWM_CNT < PWM_OCR3, else inactive. In down-counting, OC3 channel is inactive if PWM_CNT > PWM_OCR3, else active. 111: Reserved
Bit 0	**PLD3**: *Preload Enable 3* 0: Preload register on PWM_OCR3 disabled. PWM_OCR3 can be written at anytime, the new value is taken in account immediately. 1: Preload register on PWM_OCR3 enabled. Read/Write operations access the preload register. The PWM_OCR3 preload value is loaded in the active register at each Update event (when the counter reaches the autoreload value or when an update is generated by software or by the slave mode controller). **Note:** It is mandatory to enable the preload register in order to use PWM mode otherwise operation is not guaranteed. The preload register is not required in One Pulse Mode (OPM bit set in the PWM_CR register). **Note:** This bit cannot modified if LOCK level 3 has been programmed (LOCK bits in PWM_DTR register).

Table 50. Output Control bit for standard OCx channels

OCxE bit	OCx Output state
0	Output Disabled
1	OCxREF + Polarity

Note: The state of the external I/O pins connected to the standard OCx channels depends on the OCx channel state and the GPIO Port Configuration Registers.

Table 51. Output control of complementary OCx and OCxN channels

Control Bits					Output State	
MOE bit	**OSSI bit**	**OSSR bit**	**OCxE bit**	**OCxNE bit**	**OCx output state**	**OCxN output state**
1	x	0	0	0	Output disabled	Output disabled
	x	0	0	1	Output disabled	OCREF + Polarity (OCREF xor OCxP)
	x	0	1	0	OCREF + Polarity (OCREF xor OCxP)	Output disabled
	x	0	1	1	OCREF + Polarity + dead-time	Complementary to OCREF (not OCREF) + Polarity + dead-time
	x	1	0	0	Off-State (output enabled with inactive state)	Off-State (output enabled with inactive state)
	x	1	0	1	Off-State (output enabled with inactive state)	OCREF + Polarity (OCREF xor OCxNP)
	x	1	1	0	OCREF + Polarity (OCREF xor OCxP)	Off-State (output enabled with inactive state)
	x	1	1	1	OCREF + Polarity + dead-time	Complementary to OCREF (not OCREF) + Polarity + dead-time
0	0	x	0	0	Output disabled	
	0	x	0	1		
	0	x	1	0		
	0	x	1	1		
	1	x	0	0	Off-State (output enabled with inactive state)	
	1	x	0	1		
	1	x	1	0		
	1	x	1	1		

Note: *The state of the external I/O pins connected to the complementary OCx and OCxN channels depends on the OCx and OCXN channel state and the GPIO Port Configuration Registers.*

11.10.5 Request Selection Register (PWM_RSR)

Address Offset: 18h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERS	reserved				OC3RS	OC2RS	OC1RS	OCRM	reserved			res.	res.	reserved	URS
rw					rw	rw	rw					rw	rw		rw

Bit 15	**ERS**: *Emergency Request Selection* 0: Emergency DMA request selected. 1: Emergency Interrupt request selected.
Bits 14:11	Reserved: must be kept cleared.
Bit 10	**OC3RS**: *OC3 Request Selection* 0: OC3 DMA request selected. 1: OC3 interrupt request selected.
Bit 9	**OC2RS**: *OC2 Request Selection* 0: OC2 DMA request selected. 1: OC2 Interrupt request selected.
Bit 8	**OC1RS**: *OC1 Request Selection* 0: OC1 DMA request selected. 1: OC1 interrupt request selected.
Bits 7	**OCRM**: *Output Compare Request Mode* 0: when enabled, OCx DMA requests are generated by the result of the comparison between the counter and the corresponding output compare register PWM_OCRx. 1: when enabled, OCx DMA requests are generated by the update event.
Bits 6:4	Reserved: must be kept cleared.
Bit 3	Reserved: must be kept cleared.
Bit 2	Reserved: must be kept cleared.
Bit 1	Reserved: must be kept cleared.
Bit 0	**URS**: *Update Request Selection* 0: Update DMA request selected. 1: Update interrupt request selected.

11.10.6 Request Enable Register (PWM_RER)

Address Offset: 1Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERE	reserved				OC3RE	OC2RE	OC1RE	reserved				res.	res.	reserved	URE
rw					rw	rw	rw					rw	rw		rw

Bit 15	**ERE**: *Emergency DMA/Interrupt Request Enable* 0: Emergency DMA/interrupt request disabled. 1: Emergency DMA/interrupt request enabled. A request is generated if the EI flag is set.
Bits 14:11	Reserved: must be kept cleared.
Bit 10	**OC3RE**: *OC3 DMA/Interrupt Request Enable* 0: OC3 DMA/interrupt request disabled. 1: OC3 DMA/interrupt request enabled. A request is generated if the OC3I flag is set.
Bit 9	**OC2RE**: *OC2 DMA/Interrupt Request Enable* 0: OC2 DMA/interrupt request disabled. 1: OC2 DMA/interrupt request enabled. A request is generated if the OC2I flag is set.
Bit 8	**OC1RE**: *OC1 DMA/Interrupt Request Enable* 0: OC1 DMA/Interrupt Request disabled. 1: OC1 DMA/interrupt request enabled. A request is generated if the OC1I flag is set.
Bits 7:4	Reserved: must be kept cleared.
Bit 3	Reserved: must be kept cleared.
Bit 2	Reserved: must be kept cleared.
Bit 1	Reserved: must be kept cleared.
Bit 0	**URE**: *Update DMA/Interrupt Request Enable* 0: Update DMA/interrupt request disabled. 1: Update DMA/interrupt request enabled. A request is generated if the UI flag is set.

11.10.7 Interrupt Status Register (PWM_ISR)

Address Offset: 20h
Reset value: 8700h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EI	reserved				OC3I	OC2I	OC1I	reserved				res.	res.	reserve d	UI
rc_w0					rc_w0	rc_w0	rc_w0					rc_w0	rc_w0		rc_w0

Bit 15	**EI**: *Emergency Status Flag* This flag is set by hardware when the emergency input pin goes low. It is cleared by software or by the DMA controller. 0: No emergency event occurred. 1: A low level has been detected on the emergency input.
Bits 14:11	Reserved: must be kept cleared.
Bit 10	**OC3I**: *OC3 Interrupt Flag* - If OCRM=0 or OC3RS=1: This flag is set by hardware when an OC3 match occurs (when the content of the counter reaches the value in the PWM_OCR3 register). It is cleared by software or by the DMA controller. **In PWM mode** (OC3C = 110b in the PWM_OMR2 register): when upcounting, the match occurs when PWM_CNT ≥ PWM_OCR3. When down-counting, the match occurs when PWM_CNT ≤ PWM_OCR3. In Center-aligned mode, the match depends on the CMS bits in the PWM_CR register. **In other modes** (OC3C < 110b in the PWM_OMR2 register): a match occurs when PWM_CNT=PWM_OCR3 0: No match. 1: An OC3 match occurred. - If OCRM=1 and OC3RS=0: This flag is set by hardware when an update event occurs, in the same conditions as UI flag. It is cleared by hardware by the DMA controller. 0: No update occurred 1: An update event occurred. If enabled (OC3RE=1), a DMA request has been sent to the DMA controller.

Bit 9	**OC2I**: *OC2 Interrupt Flag* - If OCRM=0 or OC2RS=1: This flag is set by hardware when an OC2 match occurs (when the content of the counter reaches the value in the PWM_OCR2 register). It is cleared by software or by the DMA controller. **In PWM mode** (OC2C = 110b in the PWM_OMR1 register): when upcounting, the match occurs when PWM_CNT ≥ PWM_OCR2. When down-counting, the match occurs when PWM_CNT ≤ PWM_OCR2. In Center-aligned mode, the match depends on the CMS bits in the PWM_CR register. **In other modes** (OC2C < 110b in the PWM_OMR1 register): a match occurs when PWM_CNT=PWM_OCR2 0: No match. 1: An OC2 match occurred. - If OCRM=1 and OC2RS=0: This flag is set by hardware when an update event occurs, in the same conditions as UI flag. It is cleared by hardware by the DMA controller. 0: No update occurred 1: An update event occurred. If enabled (OC2RE=1), a DMA request has been sent to the DMA controller.
Bit 8	**OC1I**: *OC1 Interrupt Flag* - If OCRM=0 or OC1RS=1: This flag is set by hardware when an OC1 match occurs (when the content of the counter reaches the value in the PWM_OCR1 register). It is cleared by software or by the DMA controller. **In PWM mode** (OC1C = 110b in the PWM_OMR1 register): when upcounting, the match occurs when PWM_CNT ≥ PWM_OCR1. When down-counting, the match occurs when PWM_CNT ≤ PWM_OCR1. In Center-aligned mode, the match depends on the CMS bits in the PWM_CR register. **In other modes** (OC1C < 110b in the PWM_OMR1 register): a match occurs when PWM_CNT=PWM_OCR1 0: No match. 1: An OC1 match occurred. - If OCRM=1 and OC1RS=0: This flag is set by hardware when an update event occurs, in the same conditions as UI flag. It is cleared by hardware by the DMA controller. 0: No update occurred 1: An update event occurred. If enabled (OC1RE=1), a DMA request has been sent to the DMA controller.
Bits 7:4	Reserved: must be kept cleared.
Bit 3	Reserved: must be kept cleared.
Bit 2	Reserved: must be kept cleared.

Bit 1	Reserved: must be kept cleared.
Bit 0	**UI**: *Update Interrupt Flag* This bit is set by hardware when an Update event occurs (depending on UFS bit in PWM_CR register, please refer to *Section 9.6.1: Control Register (TB_CR) on page 190*. It is cleared by software or by the DMA controller. 0: No update occurred. 1: Update interrupt pending. This bit is set by hardware when the counter is updated: – At overflow or underflow and, in a timer with a repetition counter, if REP_CNT value = 0 – When CNT is cleared by software using the CNT_RST bit in PWM_CR register (only if UFS=0 in PWM_CR register) – When CNT is cleared by a trigger event (refer to the PWM_SCR register description, only if UFS=0 in PWM_CR register)

Note: When CK_TIM and PCLK have different frequencies, the events which set the flags are generated on CK_TIM and then re-synchronized with the PCLK clock domain (the flag itself is related to PCLK). This means that there is a delay between the actual event and the setting of the flag. If an event is generated by software, for example writing an PWM_OCRx register to generate a compare event or writing the CNT_RST bit in PWM_CR register to generate an update event, there will be a delay before the related flag can be read at '1'. Therefore a minimum delay of 4 PCLK periods must be inserted before trying to clear this flag after generating the event (for instance by adding 2 dummy accesses).

11.10.8 Counter Register (PWM_CNT)

Address Offset: 24h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the contents of the counter.

Note: 1 The counter doesn't count if the ARR value is 0. No update event is generated in this case.

2 If the counter is written with a value higher than the ARR value while it is up-counting (U/D=0 in PWM_CR register), then the counter will count up to FFFF and restart counting from 0 up to ARR.

3 When CK_TIM and PCLK have different frequencies:

- The counter value cannot be accessed directly and is read through a buffer register. The buffer is updated once every 4 CK_TIM clock cycles (max.).
- The CNT_EN bit is not taken into account immediately by the counter. So if the counter is running when you clear the CNT_EN bit, you must insert an additional instruction before reading the counter to get the updated value.
- For the same reason, you must insert a delay before reading the counter after setting the CNT_RST bit.

11.10.9 Prescaler Register (PWM_PSC)

Address Offset: 28h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This value divides the frequency of the internal timer clock (TIM_ICK) output by the clock mode controller. The counter clock frequency is f_{TIM_ICK} / (PSC[15:0]+1).

This register contains the preload value which is loaded in the active prescaler register:

- Permanently if the counter is disabled (CNT_EN = 0 in PWM_CR register)
- Else at each update event (including when the counter is cleared using the CNT RST bit (PWM_CR register) or by the slave mode control block when configured in reset mode - SMS bits = 00 in the PWM_SCR register).

11.10.10 Repetition Counter Register (PWM_RCR)

Address Offset: 2Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								REP							
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8	Reserved: must be kept cleared.
Bits 7:0	**REP**: *Repetition Counter Value (N)* These bits allow you to set-up the update rate of the compare registers (i.e. periodic transfers from preload to active registers) when preload registers are enabled, as well as the update interrupt generation rate, if this interrupt is enabled. Each time the REP_CNT down-counter reaches zero, an update event is generated and it restarts counting from REP value (N). As REP_CNT is reloaded with REP value only at the update event, any write to the PWM_RCR register will not taken into account until the next update event. This means in PWM mode (N+1) corresponds to: - the number of PWM periods in edge-aligned mode - the number of half PWM period in center-aligned mode

11.10.11 Auto-Reload Register (PWM_ARR)

Address Offset: 30h
Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the preload value to be loaded in the active auto-reload register at the next update event.

However the active register can be directly written by software in the following cases:

- if the counter is not running (CNT EN=0 in PWM_CR register),
- if the counter is running in One Pulse Mode (OPM=1 in PWM_CR register),
- if the counter is running in down-counting mode (CMS=’00’ and U/D=1 in PWM_CR register)

So care must be taken when switching from up-counting to another mode as the preload and the active registers might be different.

The auto-reload register is a 16-bit register which contains the ARR value to be compared to the counter CNT.

If counter is up-counting in edge-aligned mode, when it reaches the ARR value an Update event is generated (depending on repetition counter). The counter restarts up-counting from 0.

If counter is up-counting in center-aligned mode, when it reaches the ARR value an update event is generated (depending on repetition counter). The counter restarts down-counting from the ARR preload value.

If counter is down-counting in edge-aligned mode, when it reaches 0 an update event is generated (depending on repetition counter). The counter restarts down-counting from the ARR preload value.

If counter is down-counting in center-aligned mode, when it reaches 0 an update event is generated (depending on repetition counter). The counter restarts up-counting from 0.

11.10.12 Output Compare Register 1 (PWM_OCR1)

Address Offset: 34h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the preload value to be loaded in the active output compare 1 register.

It is loaded permanently if the preload feature is disabled by the PLD1 bit in the PWM_OMR1 register. Otherwise it is copied in the active output compare 1 register when an update event occurs.

The active output compare register contains the value to be compared to the CNT counter and used to control the OC1 output signal.

11.10.13 Output Compare Register 2 (PWM_OCR2)

Address Offset: 38h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the preload value to be loaded in the active output compare 2 register.

It is loaded permanently if the preload feature is disabled by the PLD2 bit in the PWM_OMR2 register. Otherwise it is copied in the active output compare 2 register when an update event occurs.

The active output compare register contains the value to be compared to the CNT counter and used to control the OC2 output signal.

11.10.14 Output Compare Register 3 (PWM_OCR3)

Address Offset: 3Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This 16-bit register contains the preload value to be loaded in the active output compare 3 register.

It is loaded permanently if the preload feature is disabled by the PLD3 bit in the PWM_OMR2 register. Otherwise it is copied in the active output compare 3 register when an update event occurs.

The active output compare register contains the value to be compared to the CNT counter and used to control the OC3 output signal.

11.10.15 Dead-Time Register (PWM_DTR)

Address Offset: 5Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOE	OSSR	OSSI	EEN	LOCK		DTG									
rw	rw	rw	wo	wo	wo	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15	**MOE**: *Main Output Enable* This bit is cleared asynchronously by hardware when the emergency input pin goes low (when the emergency is enabled, refer to EEN bit description). 0: OCx and OCxN outputs are disabled. 1: OCx and OCxN outputs are enabled if their respective enable bits are set (OCxE, OCxNE in PWM_OMRx registers). See *Table 9.6.4* for more details.
Bit 14	**OSSR**: *Off-State Selection for RUN mode (used when MOE=1)* **Note:** This bit cannot be modified if LOCK level 2 or 3 is programmed (LOCK bits in PWM_DTR register). 0: In off-state, OCx/OCxN outputs are disabled. 1: In off-state, OCx/OCxN outputs are enabled and forced to their off-state level, defined by the polarity bits (OCxP/OCxNP bits in the PWM_OMRx registers - OCx=OCxP, OCxN=OCxNP
Bit 13	**OSSI**: *Off-State Selection for Idle state (used when MOE=0)* **Note:** This bit cannot be modified if LOCK level 2 or 3 is programmed (LOCK bits in PWM_DTR register). 0: In off-state, OC/OCN outputs are disabled. 1: In off-state, OC/OCN outputs are enabled and forced to their off-state level, defined by the polarity bits (OCxP/OCxNP bits in the PWM_OMRx registers - OCx=OCxP, OCxN=OCxNP.
Bit 12	**EEN**: *Emergency Enable* **Note:** This bit cannot be modified if LOCK level 1, 2 or 3 is programmed (LOCK bits in PWM_DTR register). 0: Emergency disabled. Emergency input has no effect on the outputs nor on the emergency interrupt flag. 1: Emergency enabled. Emergency input asynchronously clears the MOE bit when it becomes low, an interrupt can be generated.

Bits 11:10	**LOCK**: *LOCK configuration* These bits can be written by software once only. They can be used to program three levels of write protection. 00: LOCK OFF - No bits are locked. 01: LOCK Level 1 = DTG and EEN bits in PWM_DTR are locked 10: LOCK Level 2 = LOCK Level 1 + OCxP/OCxNP bits in PWM_OMRx registers and OSSR and OSSI bits are locked 11: LOCK Level 3 = LOCK Level 2 + OCxC and PLDx bits in PWM_OMRx registers) are locked.
Bits 9:0	**DTG**: *Dead-Time Generator Set-up* This bit cannot be modified if LOCK level 1, 2 or 3 is programmed (LOCK bits in PWM_DTR register). These 16 bits define the dead-time duration using the formula: $DT = \frac{DTG[9,0]}{f_{CKTIM}}$

11.10.16 DMA burst address (PWM_DMAB)

Address Offset: 60h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAB															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

A read or write access to this address is re-directed to the register located at:

(PWM_CR address) + (DBASE * 4) + (DMA_index * 4)

where:

- PWM_CR is the address of the first register in the register block of the timer
- DBASE is the user-configured DMA base address written in the PWM_CR register
- DMA_index is the offset automatically controlled by the DMA transfer.

refer to *DMA access in burst mode on page 218* for more details.

11.11 PWM Register map

Table 52. PWM Register map

Addr. Off set	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PWM_CR	DBASE						MMS			CMS		U/D	OPM	CNT EN	CNT RST	UFS
4	PWM_SCR		ITS					TS					SMS		SME	ENC	
8	Reserved																
C	PWM_OMR1	OC2NP	OC2NE	OC2P	OC2E	OC2C[2:0]			PLD2	OC1NP	OC1NE	OC1P	OC1E	OC1C[2:0]			PLD1
10	PWM_OMR2									OC3NP	OC3NE	OC3P	OC3E	OC3C[2:0]			PLD3
14	Reserved																
18	PWM_RSR	ERS					OC3RS	OC2RS	OC1RS	OCRM							URS
1C	PWM_RER	ERE					OC3RE	OC2RE	OC1RE								URE
20	PWM_ISR	EI					OC3I	OC2I	OC1I								UI
24	PWM_CNT	Counter Value (CNT15-CNT0)															
28	PWM_PSC	Prescaler (PSC15-PSC0)															
2C	PWM_RCR									Repetition Counter Value (REP7-REP0)							
30	PWM_ARR	Auto-Reload (ARR15-ARR0)															
34	PWM_OCR1	Output Compare 1															
38	PWM_OCR2	Output Compare 2															
3C	PWM_OCR3	Output Compare 3															
40	Reserved																
44	Reserved																
48	Reserved																
4C	Reserved																
50	Reserved																
54	Reserved																
58	Reserved																
5C	PWM_DTR	MOE	OSSR	OSSI	EEN	LOCK		DTG									
60	PWM_DMAB	Virtual register, dedicated address for DMA burst transfers															

See *Table 2* for the base address

12 Controller area network (CAN)

12.1 Introduction

The CAN peripheral consists of the CAN Core, Message RAM, Message Handler, Control Registers and Module Interface (Refer to *Figure 84*).

The CAN Core performs communication according to the CAN protocol version 2.0 part A and B. The bit rate can be programmed to values up to 1MBit/s. For the connection to the physical layer, additional transceiver hardware is required.

For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks for acceptance filtering of received messages are stored in the Message RAM.

All functions concerning the handling of messages are implemented in the Message Handler. These functions include acceptance filtering, the transfer of messages between the CAN Core and the Message RAM, and the handling of transmission requests as well as the generation of the module interrupt.

The register set of the CAN peripheral can be accessed directly by the CPU through the module interface. These registers are used to control/configure the CAN Core and the Message Handler and to access the Message RAM.

12.2 Main features

- Supports CAN protocol version 2.0 part A and B
- Bit rates up to 1 MBit/s
- 32 Message Objects
- Each Message Object has its own identifier mask
- Programmable FIFO mode (concatenation of Message Objects)
- Maskable interrupt
- Disabled Automatic Re-transmission mode for Time Triggered CAN applications
- Programmable loop-back mode for self-test operation
- Two 16-bit module interfaces to the APB bus

12.3 Block diagram

The CAN peripheral interfaces with the AMBA APB bus. *Figure 84* shows the block diagram of the CAN peripheral.

CAN core

CAN Protocol Controller and Rx/Tx Shift Register.

Message RAM

Stores Message Objects and Identifier Masks.

Registers

All registers used to control and to configure the CAN peripheral.

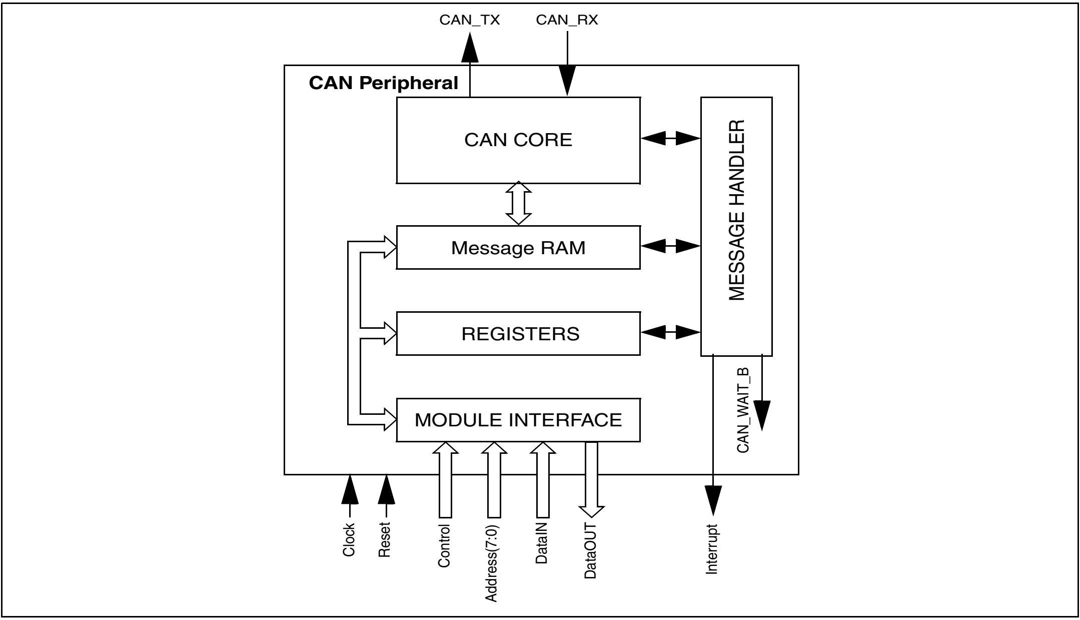
Message handler

State Machine that controls the data transfer between the Rx/Tx Shift Register of the CAN Core and the Message RAM as well as the generation of interrupts as programmed in the Control and Configuration Registers.

Module interface

The module interface provides the interface between the APB 16-bit bus and the CAN peripheral registers.

Figure 84. Block Diagram of the CAN Peripheral



12.4 Functional description

12.4.1 Software initialization

The software initialization is started by setting the Init bit in the CAN Control Register, either by a software or a hardware reset, or by going to Bus_Off state.

While the Init bit is set, all message transfers to and from the CAN bus are stopped and the status of the CAN_TX output pin is recessive (HIGH). The Error Management Logic (EML) counters are unchanged. Setting the Init bit does not change any configuration register.

To initialize the CAN Controller, software has to set up the Bit Timing Register and each Message Object. If a Message Object is not required, the corresponding MsgVal bit should be cleared. Otherwise, the entire Message Object has to be initialized.

Access to the Bit Timing Register and to the Baud Rate Prescaler (BRP) Extension Register for configuring bit timing is enabled when the Init and Configuration Change Enable *(*CCE) bits in the CAN Control Register are both set.

Resetting the Init bit (by CPU only) finishes the software initialization. Later, the Bit Stream Processor (BSP) (see *Section 12.7.10: Configuring the bit timing on page 306*) synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (≡ Bus Idle) before it can take part in bus activities and start the message transfer.

The initialization of the Message Objects is independent of Init and can be done on the fly, but the Message Objects should all be configured to particular identifiers or set to not valid before the BSP starts the message transfer.

To change the configuration of a Message Object during normal operation, software has to start by resetting the corresponding MsgVal bit. When the configuration is completed, MsgVal is set again.

12.4.2 CAN message transfer

Once the CAN peripheral is initialized and Init bit is cleared, the CAN peripheral Core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored in their appropriate Message Objects if they pass the Message Handler's acceptance filtering. The whole message including all arbitration bits, DLC and eight data bytes are stored in the Message Object. If the Identifier Mask is used, the arbitration bits which are masked to “don't care” may be overwritten in the Message Object.

Software can read or write each message any time through the Interface Registers and the Message Handler guarantees data consistency in case of concurrent accesses.

Messages to be transmitted are updated by the application software. If a permanent Message Object (arbitration and control bits are set during configuration) exists for the message, only the data bytes are updated and the TxRqst bit with NewDat bit are set to start the transmission. If several transmit messages are assigned to the same Message Object (when the number of Message Objects is not sufficient), the whole Message Object has to be configured before the transmission of this message is requested.

The transmission of any number of Message Objects may be requested at the same time. Message objects are transmitted subsequently according to their internal priority. Messages may be updated or set to not valid any time, even when their requested transmission is still pending. The old data will be discarded when a message is updated before its pending transmission has started.

Depending on the configuration of the Message Object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

12.4.3 Disabled automatic re-transmission mode

In accordance with the CAN Specification (see ISO11898, 6.3.3 Recovery Management), the CAN peripheral provides means for automatic re-transmission of frames that have lost arbitration or have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. This means that, by default, automatic retransmission is enabled. It can be disabled to enable the CAN peripheral to work within a Time Triggered CAN (TTCAN, see ISO11898-1) environment.

Disabled Automatic Retransmission mode is enabled by setting the Disable Automatic Retransmission (DAR) bit in the CAN Control Register. In this operation mode, the programmer has to consider the different behaviour of bits TxRqst and NewDat in the Control Registers of the Message Buffers:

- When a transmission starts, bit TxRqst of the respective Message Buffer is cleared, while bit NewDat remains set.
- When the transmission completed successfully, bit NewDat is cleared.
- When a transmission fails (lost arbitration or error), bit NewDat remains set.

To restart the transmission, the CPU should set the bit TxRqst again.

12.4.4 Test mode

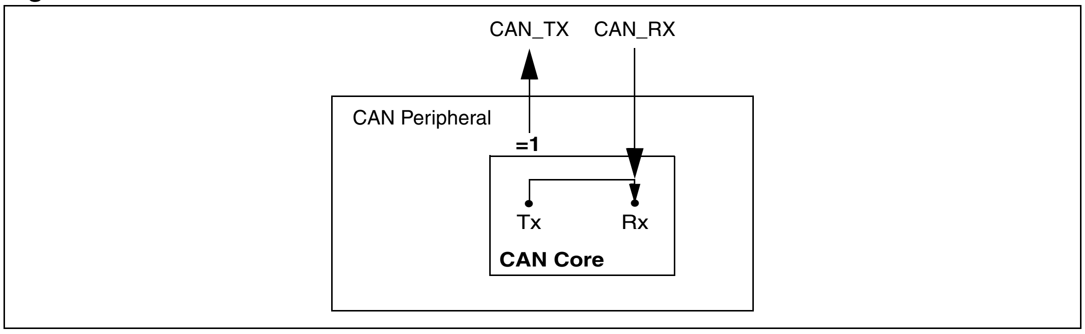
Test Mode is entered by setting the Test bit in the CAN Control Register. In Test Mode, bits Tx1, Tx0, LBack, Silent and Basic in the Test Register are writeable. Bit Rx monitors the state of the CAN_RX pin and therefore is only readable. All Test Register functions are disabled when the Test bit is cleared.

Silent mode

The CAN Core can be set in Silent Mode by programming the Silent bit in the Test Register to one.

In Silent Mode, the CAN peripheral is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the CAN Core is required to send a dominant bit (ACK bit, Error Frames), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. The Silent Mode can be used to analyse the traffic on a CAN bus without affecting it by the transmission of *dominant* bits. *Figure 85* shows the connection of signals CAN_TX and CAN_RX to the CAN Core in Silent Mode.

Figure 85. CAN core in silent mode

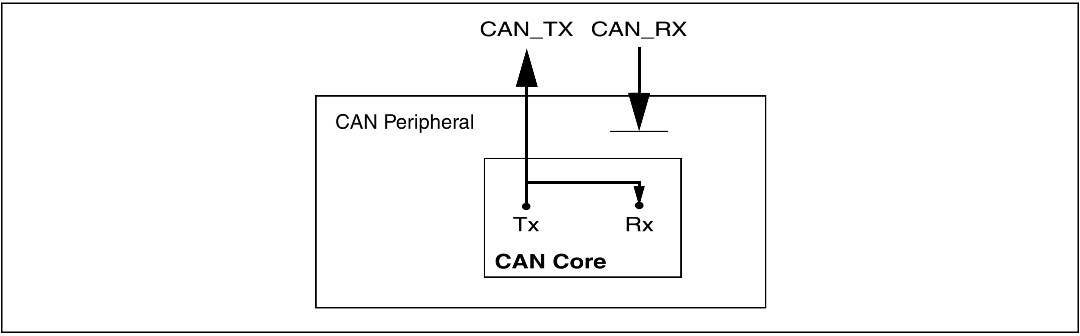


In ISO 11898-1, Silent Mode is called Bus Monitoring Mode.

Loop back mode

The CAN Core can be set in Loop Back Mode by programming the Test Register bit LBack to one. In Loop Back Mode, the CAN Core treats its own transmitted messages as received messages and stores them in a Receive Buffer (if they pass acceptance filtering). *Figure 86* shows the connection of signals, CAN_TX and CAN_RX, to the CAN Core in Loop Back Mode.

Figure 86. CAN core in loop back mode

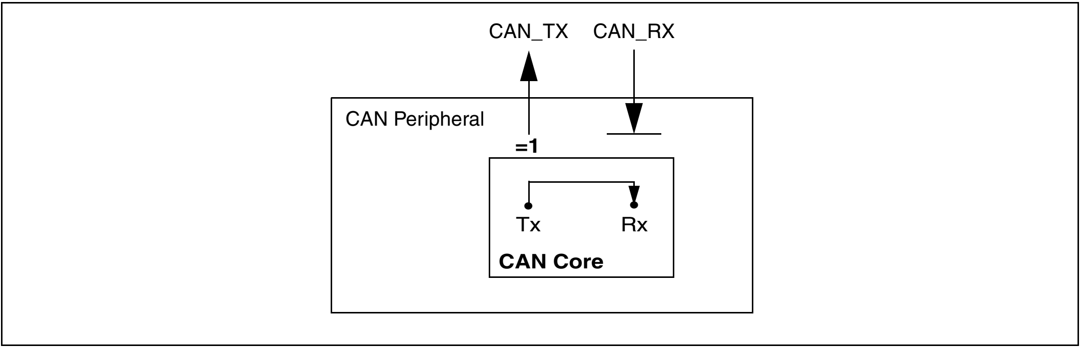


This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop Back Mode. In this mode, the CAN Core performs an internal feedback from its Tx output to its Rx input. The actual value of the CAN_RX input pin is disregarded by the CAN Core. The transmitted messages can be monitored on the CAN_TX pin.

Loop back combined with silent mode

It is also possible to combine Loop Back Mode and Silent Mode by programming bits LBack and Silent to one at the same time. This mode can be used for a “Hot Selftest”, which means that CAN peripheral can be tested without affecting a running CAN system connected to the CAN_TX and CAN_RX pins. In this mode, the CAN_RX pin is disconnected from the CAN Core and the CAN_TX pin is held recessive. *Figure 87* shows the connection of signals CAN_TX and CAN_RX to the CAN Core in case of the combination of Loop Back Mode with Silent Mode.

Figure 87. CAN core in loop back mode combined with silent mode



Basic mode

The CAN Core can be set in Basic Mode by programming the Test Register bit Basic to one. In this mode, the CAN peripheral runs without the Message RAM.

The IF1 Registers are used as Transmit Buffer. The transmission of the contents of the IF1 Registers are requested by writing the Busy bit of the IF1 Command Request Register to one. The IF1 Registers are locked while the Busy bit is set. The Busy bit indicates that the transmission is pending.

As soon the CAN bus is idle, the IF1 Registers are loaded into the shift register of the CAN Core and the transmission is started. When the transmission has been completed, the Busy bit is reset and the locked IF1 Registers are released.

A pending transmission can be aborted at any time by resetting the Busy bit in the IF1 Command Request Register while the IF1 Registers are locked. If the CPU has reset the Busy bit, a possible retransmission in case of lost arbitration or in case of an error is disabled.

The IF2 Registers are used as a Receive Buffer. After the reception of a message the contents of the shift register is stored into the IF2 Registers, without any acceptance filtering.

Additionally, the actual contents of the shift register can be monitored during the message transfer. Each time a read Message Object is initiated by writing the Busy bit of the IF2 Command Request Register to one, the contents of the shift register are stored in the IF2 Registers.

In Basic Mode, the evaluation of all Message Object related control and status bits and the control bits of the IFn Command Mask Registers are turned off. The message number of the Command request registers is not evaluated. The NewDat and MsgLst bits in the IF2 Message Control Register retain their function, DLC3-0 indicate the received DLC, and the other control bits are read as ‘0’.

Software control of CAN_TX pin

Four output functions are available for the CAN transmit pin, CAN_TX. In addition to its default function (serial data output), the CAN transmit pin can drive the CAN Sample Point signal to monitor CAN_Core’s bit timing and it can drive constant dominant or recessive values. The latter two functions, combined with the readable CAN receive pin CAN_RX, can be used to check the physical layer of the CAN bus.

The output mode for the CAN_TX pin is selected by programming the Tx1 and Tx0 bits of the CAN Test Register.

The three test functions of the CAN_TX pin interfere with all CAN protocol functions. CAN_TX must be left in its default function when CAN message transfer or any of the test modes (Loop Back Mode, Silent Mode, or Basic Mode) are selected.

12.5 Register description

The CAN peripheral allocates an address space of 256 bytes. The registers are organized as 16-bit registers.

The two sets of interface registers (IF1 and IF2) control the CPU access to the Message RAM. They buffer the data to be transferred to and from the RAM, avoiding conflicts between CPU accesses and message reception/transmission.

In this section, the following abbreviations are used:

read/write (rw)	The software can read and write to these bits.
read-only (r)	The software can only read these bits.
write-only (w)	The software should only write to these bits.

The CAN registers are listed in *Table 53*.

Table 53. **CAN registers**

Register Name	Address Offset	Reset Value
CAN Control Register (CAN_CR)	00h	0001h
Status Register (CAN_SR)	04h	0000h
Error Counter (CAN_ERR)	08h	0000h
Bit Timing Register (CAN_BTR)	0Ch	2301h
Test Register (CAN_TESTR)	14h	0000 0000 R000 0000 b **Note:** R=current value of the RX pin
BRP Extension Register (CAN_BRPR)	18h	0000h
IFn Command Request Registers (CAN_IFn_CRR)	20h (CAN_IF1_CRR), 80h (CAN_IF2_CRR)	0001h
IFn Command Mask Registers (CAN_IFn_CMR)	24h (CAN_IF1_CMR), 84h (CAN_IF2_CMR)	0000h
IFn Mask 1 Register (CAN_IFn_M1R)	28h (CAN_IF1_M1R), 88h (CAN_IF2_M1R)	FFFFh
IFn Mask 2 Register (CAN_IFn_M2R)	2Ch (CAN_IF1_M2R), 8Ch (CAN_IF2_M2R)	FFFFh
IFn Message Arbitration 1 Register (CAN_IFn_A1R)	30h (CAN_IF1_A1R), 90h (CAN_IF2_A1R)	0000h
IFn Message Arbitration 2 Register (CAN_IFn_A2R)	34h (CAN_IF1_A2R), 94h (CAN_IF2_A2R)	0000h
IFn Message Control Registers (CAN_IFn_MCR)	38h (CAN_IF1_MCR), 98h (CAN_IF2_MCR)	0000h
IFn Data A/B Registers (CAN_IFn_DAnR and CAN_IFn_DBnR)	3Ch (CAN_IF1_DA1R), 40h (CAN_IF1_DA2R), 44h (CAN_IF1_DB1R), 48h (CAN_IF1_DB2R), 9Ch (CAN_IF2_DA1R), A0h (CAN_IF2_DA2R), A4h (CAN_IF2_DB1R), A8h (CAN_IF2_DB2R)	0000h
Interrupt Identifier Register (CAN_IDR)	10h	0000h
Transmission Request Registers 1 & 2 (CAN_TxRnR)	100h (CAN_TxR1R), 104h (CAN_TxR2R)	0000h
New Data Registers 1 & 2 (CAN_NDnR)	120h (CAN_ND1R), 124h (CAN_ND2R)	0000h
Interrupt Pending Registers 1 & 2 (CAN_IPnR)	140h (CAN_IP1R), 144h (CAN_IP2R)	0000h
Message Valid Registers 1 & 2 (CAN_MVnR)	160h (CAN_MV1R), 164h (CAN_MV2R)	0000h

12.5.1 CAN interface reset state

After the hardware reset, the CAN peripheral registers hold the reset values givenin *Table 53* and the register descriptions below.

Additionally the *busoff* state is reset and the output CAN_TX is set to recessive (HIGH). The value 0x0001 (Init = ‘1’) in the CAN Control Register enables the software initialization. The CAN peripheral does not influence the CAN bus until the CPU resets the Init bit to ‘0’.

The data stored in the Message RAM is not affected by a hardware reset. After powering on, the contents of the Message RAM are undefined.

12.5.2 CAN protocol related registers

These registers are related to the CAN protocol controller in the CAN Core. They control the operating modes and the configuration of the CAN bit timing and provide status information.

CAN control register (CAN_CR)

Address Offset: 00h
Reset value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Test	CCE	DAR	res	EIE	SIE	IE	Init
-	-	-	-	-	-	-	-	rw	rw	rw	-	rw	rw	rw	rw

Bits 15:8	Reserved, , forced by hardware to 0.
Bit 7	**Test:** *Test Mode Enable* 0: Normal Operation. 1: Test Mode.
Bit 6	**CCE:** *Configuration Change Enable* 0: No write access to the Bit Timing Register. 1: Write access to the Bit Timing Register allowed (while bit Init=1).
Bit 5	**DAR:** *Disable Automatic Re-transmission* 0: Automatic Retransmission of disturbed messages enabled. 1: Automatic Retransmission disabled.
Bit 4	Reserved, forced by hardware to 0.
Bit 3	**EIE:** *Error Interrupt Enable* 0: Disabled - No Error Status Interrupt will be generated. 1: Enabled - A change in the bits BOff or EWarn in the Status Register will generate an interrupt.
Bit 2	**SIE:** *Status Change Interrupt Enable* 0: Disabled - No Status Change Interrupt will be generated. 1: Enabled - An interrupt will be generated when a message transfer is successfully completed or a CAN bus error is detected.

Bit 1	**IE:** *Module Interrupt Enable* 0: Disabled. 1: Enabled.
Bit 0	**Init:** *Initialization* 0: Normal Operation. 1: Initialization is started.

*Note: The busoff recovery sequence (see CAN Specification Rev. 2.0) cannot be shortened by setting or resetting the Init bit. If the device goes in the busoff state, it will set Init of its own accord, stopping all bus activities. Once Init has been cleared by the CPU, the device will then wait for 129 occurrences of Bus Idle (129 * 11 consecutive recessive bits) before resuming normal operations. At the end of the busoff recovery sequence, the Error Management Counters will be reset.*

During the waiting time after resetting Init, each time a sequence of 11 recessive bits has been monitored, a Bit0Error code is written to the Status Register, enabling the CPU to readily check up whether the CAN bus is stuck at dominant or continuously disturbed and to monitor the proceeding of the busoff recovery sequence.

Status register (CAN_SR)

Address Offset: 04h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BOff	EWarn	EPass	RxOk	TxOk	LEC		
-	-	-	-	-	-	-	-	r	r	r	rw	rw	rw	rw	rw

Bit 15:8	Reserved, forced by hardware to 0.
Bit 7	**BOff:** *Busoff Status* 0: The CAN module is not in busoff state. 1: The CAN module is in busoff state.
Bit 6	**EWarn:** *Warning Status* 0: Both error counters are below the error warning limit of 96. 1: At least one of the error counters in the EML has reached the error warning limit of 96.
Bit 5	**EPass:** *Error Passive* 0: The CAN Core is error active. 1: The CAN Core is in the error passive state as defined in the CAN Specification.
Bit 4	**RxOk:** *Received a Message Successfully* 0: No message has been successfully received since this bit was last reset by the CPU. This bit is never reset by the CAN Core. 1: A message has been successfully received since this bit was last reset by the CPU (independent of the result of acceptance filtering).

Bit 3	**TxOk:** *Transmitted a Message Successfully* 0: Since this bit was reset by the CPU, no message has been successfully transmitted. This bit is never reset by the CAN Core. 1: Since this bit was last reset by the CPU, a message has been successfully (error free and acknowledged by at least one other node) transmitted.
Bits 2:0	**LEC[2:0]:** *Last Error Code (Type of the last error to occur on the CAN bus)* The LEC field holds a code, which indicates the type of the last error to occur on the CAN bus. This field will be cleared to '0' when a message has been transferred (reception or transmission) without error. The unused code '7' may be written by the CPU to check for updates. Table 54 describes the error codes.

Table 54. Error codes

Error Code	Meaning
0	No Error
1	Stuff Error: More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
2	Form Error: A fixed format part of a received frame has the wrong format.
3	AckError: The message this CAN Core transmitted was not acknowledged by another node.
4	Bit1Error: During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value '1'), but the monitored bus value was dominant.
5	Bit0Error: During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), though the device wanted to send a dominant level (data or identifier bit logical value '0'), but the monitored Bus value was recessive. During busoff recovery, this status is set each time a sequence of 11 recessive bits has been monitored. This enables the CPU to monitor the proceedings of the busoff recovery sequence (indicating the bus is not stuck at *dominant* or continuously disturbed).
6	CRCError: The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data.
7	Unused: When the LEC shows the value '7', no CAN bus event was detected since the CPU wrote this value to the LEC.

Status interrupts

A Status Interrupt is generated by bits BOff and EWarn (Error Interrupt) or by RxOk, TxOk, and LEC (Status Change Interrupt) assuming that the corresponding enable bits in the CAN Control Register are set. A change of bit EPass or a write to RxOk, TxOk, or LEC will never generate a Status Interrupt.

Reading the Status Register will clear the Status Interrupt value (8000h) in the Interrupt Register, if it is pending.

Error counter (CAN_ERR)

Address Offset: 08h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RP	REC[6:0]							TEC[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bit 15	**RP:** *Receive Error Passive* 0: The Receive Error Counter is below the error passive level. 1: The Receive Error Counter has reached the error passive level as defined in the CAN Specification.
Bits 14:8	**REC[6:0]:** *Receive Error Counter* Actual state of the Receive Error Counter. Values between 0 and 127.
Bits 7:0	**TEC[7:0]:** *Transmit Error Counter* Actual state of the Transmit Error Counter. Values between 0 and 255.

Bit timing register (CAN_BTR)

Address Offset: 0Ch

Reset value: 2301h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res	TSeg2			TSeg1				SJW		BRP					
-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15	Reserved, forced by hardware to 0.
Bits 14:12	**TSeg2:** *Time segment after sample point* 0x0-0x7: Valid values for TSeg2 are [0 … 7]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.
Bits 11:8	**TSeg1:** *Time segment before the sample point minus Sync_Seg* 0x01-0x0F: valid values for TSeg1 are [1 … 15]. The actual interpretation by the hardware of this value is such that one more than the value programmed is used.
Bits 7:6	**SJW***:* (*Re)Synchronization Jump Width* 0x0-0x3: Valid programmed values are [0 … 3]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.
Bits 5:0	**BRP:** *Baud Rate Prescaler* 0x01-0x3F: The value by which the oscillator frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quanta. Valid values for the Baud Rate Prescaler are [0 … 63]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Note: With a module clock APB_CLK of 8 MHz, the reset value of 0x2301 configures the CAN peripheral for a bit rate of 500 kBit/s. The registers are only writable if bits CCE and Init in the CAN Control Register are set.

Test register (CAN_TESTR)

Address Offset: 14h
Reset value: 0000 0000 R000 0000 b (R:current value of RX pin)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Rx	Tx[1:0]		LBack	Silent	Basic	Res	
-	-	-	-	-	-	-	-	r	rw	rw	rw	rw	rw	-	-

Bits 15:8	Reserved, forced by hardware to 0.
Bit 7	**Rx:** *Current value of* CAN_RX Pin 0: The CAN bus is dominant (CAN_RX = ‘0’). 1: The CAN bus is recessive (CAN_RX = ‘1’).
Bit 6:5	**Tx[1:0]:** CAN_TX pin control 00: Reset value, CAN_TX is controlled by the CAN Core 01: Sample Point can be monitored at CAN_TX pin 10: CAN_TX pin drives a dominant (‘0’) value. 11: CAN_TX pin drives a recessive (‘1’) value.
Bit 4	**LBack:** *Loop Back Mode* 0: Loop Back Mode is disabled. 1: Loop Back Mode is enabled.
Bit 3	**Silent:** *Silent Mode* 0: Normal operation. 1: The module is in Silent Mode.
Bit 2	**Basic:** *Basic Mode* 0: Basic Mode disabled. 1: IF1 Registers used as Tx Buffer, IF2 Registers used as Rx Buffer.
Bits 1:0	Reserved, forced by hardware to 0.

Write access to the Test Register is enabled by setting the Test bit in the CAN Control Register. The different test functions may be combined, but Tx1-0 ≠ “00” disturbs message transfer.

BRP extension register (CAN_BRPR)

Address Offset: 18h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												BRPE			
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bits 15:4	Reserved, forced by hardware to 0.
Bits 3:0	**BRPE:** *Baud Rate Prescaler Extension* 0x00-0x0F: By programming BRPE, the Baud Rate Prescaler can be extended to values up to 1023. The actual interpretation by the hardware is that one more than the value programmed by BRPE (MSBs) and BRP (LSBs) is used.

12.5.3 Message interface register sets

There are two sets of Interface Registers, which are used to control the CPU access to the Message RAM. The Interface Registers avoid conflict between the CPU access to the Message RAM and CAN message reception and transmission by buffering the data to be transferred. A complete Message Object (see *Section : Message object in the message memory*) or parts of the Message Object may be transferred between the Message RAM and the IF*n* Message Buffer registers (see *Section : IFn message buffer registers*) in one single transfer.

The function of the two interface register sets is identical except for the Basic test mode. They can be used the way one set of registers is used for data transfer to the Message RAM while the other set of registers is used for the data transfer from the Message RAM, allowing both processes to be interrupted by each other. *Table 55: IF1 and IF2 message interface register set on page 286* provides an overview of the two Interface Register sets.

Each set of Interface Registers consists of Message Buffer Registers controlled by their own Command Registers. The Command Mask Register specifies the direction of the data transfer and which parts of a Message Object will be transferred. The Command Request Register is used to select a Message Object in the Message RAM as target or source for the transfer and to start the action specified in the Command Mask Register.

Table 55. IF1 and IF2 message interface register set

Address	IF1 Register Set	Address	IF2 Register Set
CAN Base + 0x20	IF1 Command Request	CAN Base + 0x80	IF2 Command Request
CAN Base + 0x24	IF1 Command Mask	CAN Base + 0x84	IF2 Command Mask
CAN Base + 0x28	IF1 Mask 1	CAN Base + 0x88	IF2 Mask 1
CAN Base + 0x2C	IF1 Mask 2	CAN Base + 0x8C	IF2 Mask 2
CAN Base + 0x30	IF1 Arbitration 1	CAN Base + 0x90	IF2 Arbitration 1
CAN Base + 0x34	IF1 Arbitration 2	CAN Base + 0x94	IF2 Arbitration 2
CAN Base + 0x38	IF1 Message Control	CAN Base + 0x98	IF2 Message Control
CAN Base + 0x3C	IF1 Data A 1	CAN Base + 0x9C	IF2 Data A 1
CAN Base + 0x40	IF1 Data A 2	CAN Base + 0xA0	IF2 Data A 2
CAN Base + 0x44	IF1 Data B 1	CAN Base + 0xA4	IF2 Data B 1
CAN Base + 0x48	IF1 Data B 2	CAN Base + 0xA8	IF2 Data B 2

IF*n* command request registers (CAN_IF*n*_CRR)

Address offset: 20h (CAN_IF1_CRR), 80h (CAN_IF2_CRR)
Reset Value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Busy	Reserved									Message Number					
r	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw

A message transfer is started as soon as the application software has written the message number to the Command Request Register. With this write operation, the Busy bit is automatically set to notify the CPU that a transfer is in progress. After a waiting time of 3 to 6 APB_CLK periods, the transfer between the Interface Register and the Message RAM is completed. The Busy bit is cleared.

Bit 15	**Busy:** *Busy Flag* 0: Read/write action has finished. 1: Writing to the IF*n* Command Request Register is in progress. This bit can only be read by the software.
Bits 14:6	Reserved, forced by hardware to 0.
Bits 5:0	**Message Number:** 0x01-0x20: Valid Message Number, the Message Object in the Message RAM is selected for data transfer. 0x00: Not a valid Message Number, interpreted as 0x20. 0x21-0x3F: Not a valid Message Number, interpreted as *0x01-0x1F*.

Note: When a Message Number that is not valid is written into the Command Request Register, the Message Number will be transformed into a valid value and that Message Object will be transferred.

IF*n* command mask registers (CAN_IF*n*_CMR)

Address offset: 24h (CAN_IF1_CMR), 84h (CAN_IF2_CMR)
Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/ NewDat	Data A	Data B
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

The control bits of the IF*n* Command Mask Register specify the transfer direction and select which of the IF*n* Message Buffer Registers are source or target of the data transfer.

Bits 15:8	Reserved, forced by hardware to 0.
Bit 7	**WR/RD:** *Write / Read* 0: Read: Transfer data from the Message Object addressed by the Command Request Register into the selected Message Buffer Registers. 1: Write: Transfer data from the selected Message Buffer Registers to the Message Object addressed by the Command Request Register.

Bits 6:0	These bits of IFn Command Mask Register have different functions depending on the transfer direction: Direction = Write Bit 6 = Mask Access Mask Bits 0: Mask bits unchanged. 1: transfer Identifier Mask + MDir + MXtd to Message Object. Bit 5 = Arb Access Arbitration Bits 0: Arbitration bits unchanged. 1: Transfer Identifier + Dir + Xtd + MsgVal to Message Object. Bit 4 = Control Access Control Bits 0: Control Bits unchanged. 1: Transfer Control Bits to Message Object. Bit 3 = ClrIntPnd Clear Interrupt Pending Bit When writing to a Message Object, this bit is ignored. Bit 2 = TxRqst/NewDat Access Transmission Request Bit 0: TxRqst bit unchanged. 1: Set TxRqst bit. If a transmission is requested by programming bit TxRqst/NewDat in the IFn Command Mask Register, bit TxRqst in the IFn Message Control Register will be ignored. Bit 1 = Data A Access Data Bytes 3:0 0: Data Bytes 3:0 unchanged. 1: Transfer Data Bytes 3:0 to Message Object. Bit 0 = Data B Access Data Bytes 7:4 0: Data Bytes 7:4 unchanged. 1: Transfer Data Bytes 7:4 to Message Object.
Bits 6:0	Direction = Read Bit 6 = Mask: *Access Mask Bits* 0: Mask bits unchanged. 1: Transfer Identifier Mask + MDir + MXtd to IF*n* Message Buffer Register. Bit 5 = Arb: *Access Arbitration Bits* 0: Arbitration bits unchanged. 1: Transfer Identifier + Dir + Xtd + MsgVal to IF*n* Message Buffer Register. Bit 4 = Control: *Access Control Bits* 0: Control Bits unchanged. 1: Transfer Control Bits to IF*n* Message Buffer Register. Bit 3 = ClrIntPnd: *Clear Interrupt Pending Bit* 0: IntPnd bit remains unchanged. 1: Clear IntPnd bit in the Message Object. Bit 2 = TxRqst/NewDat: *Access Transmission Request Bit* 0: NewDat bit remains unchanged. 1: Clear NewDat bit in the Message Object. A read access to a Message Object can be combined with the reset of the control bits IntPnd and NewDat. The values of these bits transferred to the IFn Message Control Register always reflect the status before resetting these bits. Bit 1 = Data A Access Data Bytes 3:0 0: Data Bytes 3:0 unchanged. 1: Transfer Data Bytes 3:0 to IF*n* Message Buffer Register. Bit 0 = Data B Access Data Bytes 7:4 0: Data Bytes 7:4 unchanged. 1: Transfer Data Bytes 7:4 to IF*n* Message Buffer Register.

IF*n* message buffer registers

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM. The function of the Message Objects bits is described in *Section : Message object in the message memory on page 291*.

IF*n* mask 1 register (CAN_IF*n*_M1R)

Address offset: 28h (CAN_IF1_M1R), 88h (CAN_IF2_M1R)
Reset Value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Msk[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The function of the Msk bits is described in *Section : Message object in the message memory on page 291*.

IF*n* mask 2 register (CAN_IF*n*_M2R)

Address offset: 2Ch (CAN_IF1_M2R), 8Ch (CAN_IF2_M2R)
Reset Value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MXtd	MDir	Res	Msk[28:16]												
rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The function of the Message Objects bits is described in the *Section : Message object in the message memory on page 291*.

IF*n* message arbitration 1 register (CAN_IF*n*_A1R)

Address offset: 30h (CAN_IF1_A1R), 90h (CAN_IF2_A1R)
Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The function of the Message Objects bits is described in the *Section : Message object in the message memory on page 291*.

IF*n* message arbitration 2 register (CAN_IF*n*_A2R)

Address offset: 34h (CAN_IF1_A2R), 94h (CAN_IF2_A2R)
Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MsgVal	Xtd	Dir	ID[28:16]												
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The function of the Message Objects bits is described in the *Section : Message object in the message memory on page 291*.

IF*n* message control registers (CAN_IF*n*_MCR)

Address offset: 38h (CAN_IF1_MCR), 98h (CAN_IF2_MCR)
Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	-	-	-	rw	rw	rw	rw

The function of the Message Objects bits is described in the *Section : Message object in the message memory on page 291*.

IF*n* data A/B registers (CAN_IF*n*_DA*n*R and CAN_IF*n*_DB*n*R)

The data bytes of CAN messages are stored in the IF*n* Message Buffer Registers in the following order:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IF1 Message Data A1 (address 0x3C)	Data(1)								Data(0)							
IF1 Message Data A2 (address 0x40)	Data(3)								Data(2)							
IF1 Message Data B1 (address 0x44)	Data(5)								Data(4)							
IF1 Message Data B2 (address 0x48)	Data(7)								Data(6)							
IF2 Message Data A1 (address 0x9C)	Data(1)								Data(0)							
IF2 Message Data A2 (address 0xA0)	Data(3)								Data(2)							
IF2 Message Data B1 (address 0xA4)	Data(5)								Data(4)							
IF2 Message Data B2 (address 0xA8)	Data(7)								Data(6)							
	rw								rw							

In a CAN Data Frame, Data(0) is the first, Data(7) is the last byte to be transmitted or received. In CAN's serial bit stream, the MSB of each byte will be transmitted first.

Message object in the message memory

There are 32 Message Objects in the Message RAM. To avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission, the CPU cannot directly access the Message Objects, these accesses are handled through the IF*n* Interface Registers.

Table 56 provides an overview of the structures of a Message Object

Table 56. Structure of a message object in the message memory

Message Object												
UMask	Msk 28-0	MXtd	MDir	EoB	NewDat		MsgLst	RxIE	TxIE	Int Pnd	RmtEn	TxRqst
MsgVal	ID28-0	Xtd	Dir	DLC 3-0	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7

The Arbitration Registers ID28-0, Xtd, and Dir are used to define the identifier and type of outgoing messages and are used (together with the mask registers Msk28-0, MXtd, and MDir) for acceptance filtering of incoming messages. A received message is stored in the valid Message Object with matching identifier and direction set to receive (Data Frame) or transmit (Remote Frame). Extended frames can be stored only in Message Objects with Xtd set, standard frames in Message Objects with Xtd clear. If a received message (Data Frame or Remote Frame) matches more than one valid Message Object, it is stored into that with the lowest message number. For details see *Section : Acceptance filtering of received messages*.

MsgVal	Message Valid 1: The Message Object is configured and should be considered by the Message Handler. 0: The Message Object is ignored by the Message Handler. Note: The application software must reset the MsgVal bit of all unused Messages Objects during the initialization before it resets bit Init in the CAN Control Register. This bit must also be reset before the identifier Id28-0, the control bits Xtd, Dir, or the Data Length Code DLC3-0 are modified, or if the Messages Object is no longer required.
UMask	Use Acceptance Mask 1: Use Mask (Msk28-0, MXtd, and MDir) for acceptance filtering. 0: Mask ignored. Note: If the UMask bit is set to one, the Message Object's mask bits have to be programmed during initialization of the Message Object before MsgVal is set to one.
ID28-0	Message Identifier ID28 - ID0, 29-bit Identifier ("Extended Frame") ID28 - ID18, 11-bit Identifier ("Standard Frame")
Msk28-0	Identifier Mask 1: The corresponding identifier bit is used for acceptance filtering. 0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
Xtd	Extended Identifier 1: The 29-bit ("extended") Identifier will be used for this Message Object. 0: The 11-bit ("standard") Identifier will be used for this Message Object.

MXtd	Mask Extended Identifier 1: The extended identifier bit (IDE) is used for acceptance filtering. 0: The extended identifier bit (IDE) has no effect on the acceptance filtering. Note: When 11-bit (“standard”) Identifiers are used for a Message Object, the identifiers of received Data Frames are written into bits ID28 to ID18. For acceptance filtering, only these bits together with mask bits Msk28 to Msk18 are considered.
Dir	Message Direction 1: Direction = transmit: On TxRqst, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the TxRqst bit of this Message Object is set (if RmtEn = one). 0: Direction = *receive*: On TxRqst, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.
MDir	Mask Message Direction 1: The message direction bit (Dir) is used for acceptance filtering. 0: The message direction bit (Dir) has no effect on the acceptance filtering.
EoB	End of Buffer 1: Single Message Object or last Message Object of a FIFO Buffer. 0: Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer. Note: This bit is used to concatenate two or more Message Objects (up to 32) to build a FIFO Buffer. For single Message Objects (not belonging to a FIFO Buffer), this bit must always be set to one. For details on the concatenation of Message Objects see *Section 12.7.7: Configuring a FIFO buffer*.
NewDat	New Data 1: The Message Handler or the application software has written new data into the data portion of this Message Object. 0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the application software.
MsgLst	Message Lost (only valid for Message Objects with direction = *receive*) 1: The Message Handler stored a new message into this object when NewDat was still set, the CPU has lost a message. 0: No message lost since last time this bit was reset by the CPU.
RxIE	Receive Interrupt Enable 1: IntPnd will be set after a successful reception of a frame. 0: IntPnd will be left unchanged after a successful reception of a frame.
TxIE	Transmit Interrupt Enable 1: IntPnd will be set after a successful transmission of a frame. 0: IntPnd will be left unchanged after the successful transmission of a frame.
IntPnd	Interrupt Pending 1: This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority. 0: This message object is not the source of an interrupt.
RmtEn	Remote Enable 1: At the reception of a Remote Frame, TxRqst is set. 0: At the reception of a Remote Frame, TxRqst is left unchanged.

TxRqst	Transmit Request 1: The transmission of this Message Object is requested and is not yet done. 0: This Message Object is not waiting for transmission.
DLC3-0	Data Length Code *0-8*: Data Frame has 0-8 data bytes. *9-15* : Data Frame has 8 data bytes Note: The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message. **Data 0:** 1st data byte of a CAN Data Frame **Data 1**: 2nd data byte of a CAN Data Frame **Data 2**: 3rd data byte of a CAN Data Frame **Data 3**: 4th data byte of a CAN Data Frame **Data 4**: 5th data byte of a CAN Data Frame **Data 5**: 6th data byte of a CAN Data Frame **Data 6**: 7th data byte of a CAN Data Frame **Data 7:** 8th data byte of a CAN Data Frame Note: The Data 0 Byte is the first data byte shifted into the shift register of the CAN Core during a reception while the Data 7 byte is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by unspecified values.

12.5.4 Message handler registers

All Message Handler registers are read-only. Their contents, TxRqst, NewDat, IntPnd, and MsgVal bits of each Message Object and the Interrupt Identifier is status information provided by the Message Handler FSM.

Interrupt identifier register (CAN_IDR)

Address Offset: 10h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IntId[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0	**IntId[15:0]:** Interrupt Identifier (*Table 57* indicates the source of the interrupt) If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the application software has cleared it. If IntId is different from 0x0000 and IE is set, the IRQ interrupt signal to the EIC is active. The interrupt remains active until IntId is back to value 0x0000 (the cause of the interrupt is reset) or until IE is reset. The Status Interrupt has the highest priority. Among the message interrupts, the Message Object' s interrupt priority decreases with increasing message number. A message interrupt is cleared by clearing the Message Object's IntPnd bit. The Status Interrupt is cleared by reading the Status Register.

Table 57. Source of interrupts

Interrupt Identifier	Cause of the Inerrupt
0x0000	No Interrupt is Pending
0x0001-0x0020	Number of Message Object which caused the interrupt.
0x0021-0x7FFF	unused
0x8000	Status Interrupt
0x8001-0xFFFF	unused

Transmission request registers 1 & 2 (CAN_TxR*n*R)

Address Offset: 100h (CAN_TxR1R), 104h (CAN_TxR2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TxRqst[32:17]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TxRqst[16:1]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

These registers hold the TxRqst bits of the 32 Message Objects. By reading the TxRqst bits, the CPU can check which Message Object in a Transmission Request is pending. The TxRqst bit of a specific Message Object can be set/reset by the application software through the IFn Message Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.

Bits 31:16	**TxRqst[32:17]:** *Transmission Request Bits (of all Message Objects)* 0: This Message Object is not waiting for transmission. 1: The transmission of this Message Object is requested and is not yet done. These bits are read only.
Bits 15:0	**TxRqst1[6:1]:** *Transmission Request Bits (of all Message Objects)* 0: This Message Object is not waiting for transmission. 1: The transmission of this Message Object is requested and is not yet done. These bits are read only.

New data registers 1 & 2 (CAN_ND*n*R)

Address Offset: 120h (CAN_ND1R), 124h (CAN_ND2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NewDat[32:17]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NewDat[16:1]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

These registers hold the NewDat bits of the 32 Message Objects. By reading out the NewDat bits, the CPU can check for which Message Object the data portion was updated. The NewDat bit of a specific Message Object can be set/reset by the CPU through the IFn Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

Bits 31:16	**NewDat[32:17]:** *New Data Bits (of all Message Objects)* 0: No new data has been written into the data portion of this Message Object by the Message Handler since the last time this flag was cleared by the application software. 1: The Message Handler or the application software has written new data into the data portion of this Message Object.
Bits 15:0	**NewDat[16:1]:** *New Data Bits (of all Message Objects)* 0: No new data has been written into the data portion of this Message Object by the Message Handler since the last time this flag was cleared by the application software. 1: The Message Handler or the application software has written new data into the data portion of this Message Object.

Interrupt pending registers 1 & 2 (CAN_IP*n*R)

Address Offset: 140h (CAN_IP1R), 144h (CAN_IP2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IntPnd[32:17]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IntPnd[16:1]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

These registers contain the IntPnd bits of the 32 Message Objects. By reading the IntPnd bits, the CPU can check for which Message Object an interrupt is pending. The IntPnd bit of a specific Message Object can be set/reset by the application software through the IFn Message Interface Registers or by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of IntId in the Interrupt Register

Bits 31:16	**IntPnd[32:17]:** *Interrupt Pending Bits (of all Message Objects)* 0: This message object is not the source of an interrupt. 1: This message object is the source of an interrupt.
Bits 15:0	**IntPnd[16:1]:** *Interrupt Pending Bits (of all Message Objects)* 0: This message object is not the source of an interrupt. 1: This message object is the source of an interrupt.

Message valid registers 1 & 2 (CAN_MV*n*R)

Address Offset: 160h (CAN_MV1R), 164h (CAN_MV2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MsgVal[32:17]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MsgVal[16:1]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

These registers hold the MsgVal bits of the 32 Message Objects. By reading the MsgVal bits, the application software can check which Message Object is valid. The MsgVal bit of a specific Message Object can be set/reset by the application software via the IFn Message Interface Registers.

Bits 31:16	**MsgVal[32:17]:** *Message Valid Bits (of all Message Objects)* 0: This Message Object is ignored by the Message Handler. 1: This Message Object is configured and should be considered by the Message Handler.
Bits 15:0	**MsgVal[16:1]:** *Message Valid Bits (of all Message Objects)* 0: This Message Object is ignored by the Message Handler. 1: This Message Object is configured and should be considered by the Message Handler.

12.6 Register map

Table 58. CAN register map

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	CAN_CR	Reserved								Test	CCE	DAR	res	EIE	SIE	IE	Init
04h	CAN_SR	Reserved								BOff	EWarn	EPass	RxOk	TxOk	LEC		
08h	CAN_ERR	RP	REC6-0							TEC7-0							
0Ch	CAN_BTR	res	TSeg2			TSeg1				SJW		BRP					
10h	CAN_IDR	IntId15-8								IntId7-0							
14h	CAN_TESTR	Reserved								Rx	Tx1	Tx0	LBack	Silent	Basic	Reserved	
18h	CAN_BRPR	Reserved												BRPE			
20h	CAN_IF1_CRR	Busy	Reserved									Message Number					
24h	CAN_IF1_CMR	Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/ NewDat	Data A	Data B
28h	CAN_IF1_M1R	Msk15-0															
2Ch	CAN_IF1_M2R	MXtd	MDir	res	Msk28-16												
30h	CAN_IF1_A1R	ID15-0															
34h	CAN_IF1_A2R	MsgVal	Xtd	Dir	ID28-16												
38h	CAN_IF1_MCR	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC3-0			
3Ch	CAN_IF1_DA1R	Data(1)								Data(0)							
40h	CAN_IF1_DA2R	Data(3)								Data(2)							
44h	CAN_IF1_DB1R	Data(5)								Data(4)							
48h	CAN_IF1_DB2R	Data(7)								Data(6)							
80h	CAN_IF2_CRR	Busy	Reserved									Message Number					
84h	CAN_IF2_CMR	Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/ NewDat	Data A	Data B
88h	CAN_IF2_M1R	Msk15-0															
8Ch	CAN_IF2_M2R	MXtd	MDir	res	Msk28-16												
90h	CAN_IF2_A1R	ID15-0															
94h	CAN_IF2_A2R	MsgVal	Xtd	Dir	ID28-16												

Table 58. CAN register map (continued)

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
98h	CAN_IF2_MCR	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC3-0			
9Ch	CAN_IF2_DA1R	Data(1)								Data(0)							
A0h	CAN_IF2_DA2R	Data(3)								Data(2)							
A4h	CAN_IF2_DB1R	Data(5)								Data(4)							
A8h	CAN_IF2_DB2R	Data(7)								Data(6)							
100h	CAN_TxR1R	TxRqst16-1															
104h	CAN_TxR2R	TxRqst32-17															
120h	CAN_ND1R	NewDat16-1															
124h	CAN_ND2R	NewDat32-17															
140h	CAN_IP1R	IntPnd16-1															
144h	CAN_IP2R	IntPnd32-17															
160h	CAN_MV1R	MsgVal16-1															
164h	CAN_MV2R	MsgVal32-17															

Note: Reserved bits are read as 0' except for IFn Mask 2 Register where they are read as '1'.

Refer to *Table 2 on page 19* for the register base addresses.

12.7 CAN communications

12.7.1 Managing message objects

The configuration of the Message Objects in the Message RAM (with the exception of the bits MsgVal, NewDat, IntPnd, and TxRqst) will not be affected by resetting the chip. All the Message Objects must be initialized by the application software or they must be “not valid” (MsgVal = ‘0’) and the bit timing must be configured before the application software clears the Init bit in the CAN Control Register.

The configuration of a Message Object is done by programming Mask, Arbitration, Control and Data fields of one of the two interface registers to the desired values. By writing to the corresponding IFn Command Request Register, the IFn Message Buffer Registers are loaded into the addressed Message Object in the Message RAM.

When the Init bit in the CAN Control Register is cleared, the CAN Protocol Controller state machine of the CAN_Core and state machine of the Message Handler control the internal data flow of the CAN peripheral. Received messages that pass the acceptance filtering are stored in the Message RAM, messages with pending transmission request are loaded into the CAN_Core’s Shift Register and are transmitted through the CAN bus.

The application software reads received messages and updates messages to be transmitted through the IF*n* Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

12.7.2 Message handler state machine

The Message Handler controls the data transfer between the Rx/Tx Shift Register of the CAN Core, the Message RAM and the IF*n* Registers.

The Message Handler FSM controls the following functions:

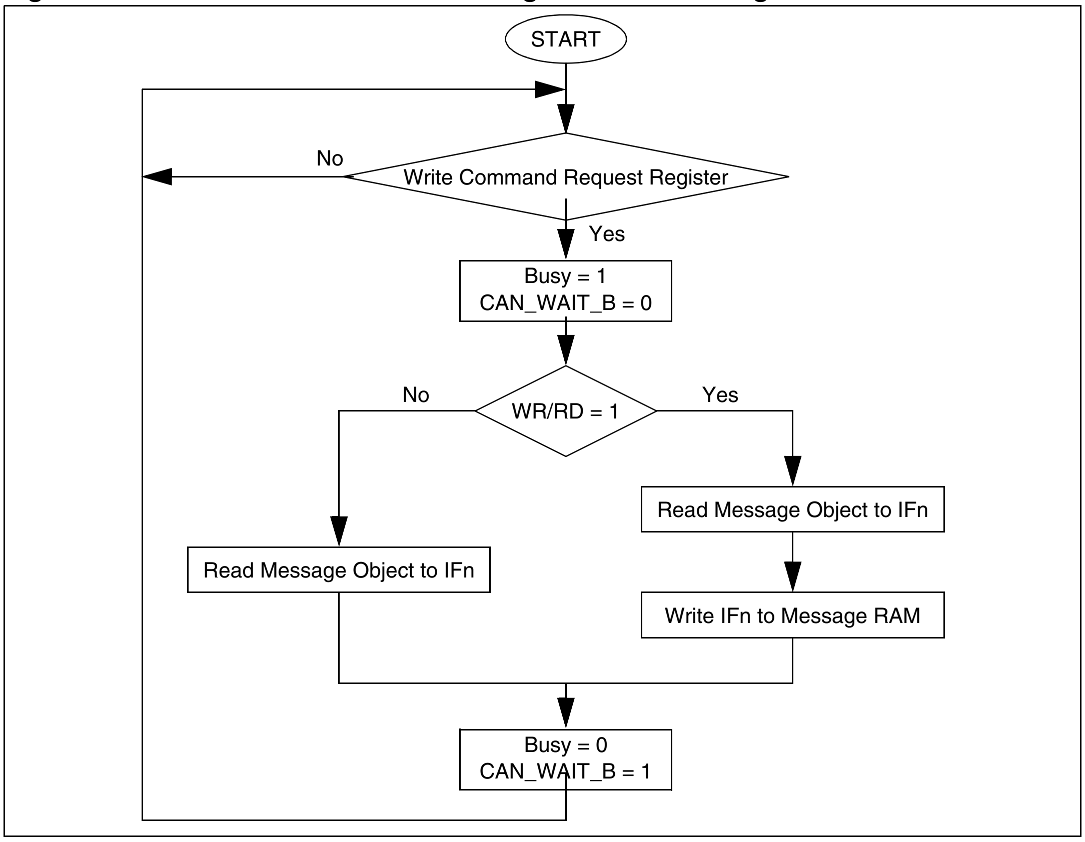
- Data Transfer from IF*n* Registers to the Message RAM
- Data Transfer from Message RAM to the IF*n* Registers
- Data Transfer from Shift Register to the Message RAM
- Data Transfer from Message RAM to Shift Register
- Data Transfer from Shift Register to the Acceptance Filtering unit
- Scanning of Message RAM for a matching Message Object
- Handling of TxRqst flags
- Handling of interrupts.

Data transfer from/to message RAM

When the CPU initiates a data transfer between the IF*n* Registers and Message RAM, the Message Handler sets the Busy bit in the respective Command Request Register (CAN_IFn_CRR). After the transfer has completed, the Busy bit is again cleared (see *Figure 88*).

The respective Command Mask Register specifies whether a complete Message Object or only parts of it will be transferred. Due to the structure of the Message RAM, it is not possible to write single bits/bytes of one Message Object. It is always necessary to write a complete Message Object into the Message RAM. Therefore, the data transfer from the IF*n* Registers to the Message RAM requires a read-modify-write cycle. First, those parts of the Message Object that are not to be changed are read from the Message RAM and then the complete contents of the Message Buffer Registers are written into the Message Object.

Figure 88. Data transfer between IF*n* Registers and Message RAM



After a partial write of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will set the actual contents of the selected Message Object.

After a partial read of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will be left unchanged.

Message transmission

If the shift register of the CAN Core cell is ready for loading and if there is no data transfer between the IFn Registers and Message RAM, the MsgVal bits in the Message Valid Register and TxRqst bits in the Transmission Request Register are evaluated. The valid Message Object with the highest priority pending transmission request is loaded into the shift register by the Message Handler and the transmission is started. The NewDat bit of the Message Object is reset.

After a successful transmission and also if no new data was written to the Message Object (NewDat = ‘0’) since the start of the transmission, the TxRqst bit of the Message Control register (CAN_IFn_MCR) will be reset. If TxIE bit of the Message Control register (CAN_IFn_MCR) is set, IntPnd bit of the Interrupt Identifier register (CAN_IDR) will be set after a successful transmission. If the CAN peripheral has lost the arbitration or if an error occurred during the transmission, the message will be retransmitted as soon as the CAN bus is free again. Meanwhile, if the transmission of a message with higher priority has been requested, the messages will be transmitted in the order of their priority.

Acceptance filtering of received messages

When the arbitration and control field (Identifier + IDE + RTR + DLC) of an incoming message is completely shifted into the Rx/Tx Shift Register of the CAN Core, the Message Handler FSM starts the scanning of the Message RAM for a matching valid Message Object.

To scan the Message RAM for a matching Message Object, the Acceptance Filtering unit is loaded with the arbitration bits from the CAN Core shift register. The arbitration and mask fields (including MsgVal, UMask, NewDat, and EoB) of Message Object 1 are then loaded into the Acceptance Filtering unit and compared with the arbitration field from the shift register. This is repeated with each following Message Object until a matching Message Object is found or until the end of the Message RAM is reached.

If a match occurs, the scan is stopped and the Message Handler FSM proceeds depending on the type of frame (Data Frame or Remote Frame) received.

Reception of data frame

The Message Handler FSM stores the message from the CAN Core shift register into the respective Message Object in the Message RAM. Not only the data bytes, but all arbitration bits and the Data Length Code are stored in the corresponding Message Object. This is done to keep the data bytes connected with the identifier even if arbitration mask registers are used.

The NewDat bit is set to indicate that new data (not yet seen by the CPU) has been received. The application software should reset NewDat bit when the Message Object has been read. If at the time of reception, the NewDat bit was already set, MsgLst is set to indicate that the previous data (supposedly not seen by the CPU) is lost. If the RxIE bit is set, the IntPnd bit is set, causing the Interrupt Register to point to this Message Object.

The TxRqst bit of this Message Object is reset to prevent the transmission of a Remote Frame, while the requested Data Frame has just been received.

Reception of Remote Frame

When a Remote Frame is received, three different configurations of the matching Message Object have to be considered:

1. **Dir** = ‘1’ (direction = *transmit*), **RmtEn** = ‘1’, **UMask** = ‘1’ or’0’
   At the reception of a matching Remote Frame, the TxRqst bit of this Message Object is set. The rest of the Message Object remains unchanged.
2. **Dir** = ‘1’ (direction = *transmit*), **RmtEn** = ‘0’, **UMask** =’0’
   At the reception of a matching Remote Frame, the TxRqst bit of this Message Object remains unchanged; the Remote Frame is ignored.
3. **Dir** = ‘1’ (direction = *transmit*), **RmtEn** = ‘0’, **UMask** =’1’
   At the reception of a matching Remote Frame, the TxRqst bit of this Message Object is reset. The arbitration and control field (Identifier + IDE + RTR + DLC) from the shift register is stored in the Message Object of the Message RAM and the NewDat bit of this Message Object is set. The data field of the Message Object remains unchanged; the Remote Frame is treated similar to a received Data Frame.

Receive/transmit priority

The receive/transmit priority for the Message Objects is attached to the message number. Message Object 1 has the highest priority, while Message Object 32 has the lowest priority. If more than one transmission request is pending, they are serviced due to the priority of the corresponding Message Object.

12.7.3 Configuring a transmit object

Table 59 shows how a Transmit Object should be initialized.

Table 59. Initialization of a Transmit Object

MsgVal	Arb	Data	Mask	EoB	Dir	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
1	appl.	appl.	appl.	1	1	0	0	0	appl.	0	appl.	0

The Arbitration Register values (ID28-0 and Xtd bit) are provided by the application. They define the identifier and type of the outgoing message. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to ID28 - ID18. The ID17 - ID0 can then be disregarded.

If the TxIE bit is set, the IntPnd bit will be set after a successful transmission of the Message Object.

If the RmtEn bit is set, a matching received Remote Frame will cause the TxRqst bit to be set; the Remote Frame will autonomously be answered by a Data Frame.

The Data Register values (DLC3-0, Data0-7) are provided by the application, TxRqst and RmtEn may not be set before the data is valid.

The Mask Registers (Msk28-0, UMask, MXtd, and MDir bits) may be used (UMask=’1’) to allow groups of Remote Frames with similar identifiers to set the TxRqst bit. The Dir bit should not be masked.

12.7.4 Updating a transmit object

The CPU may update the data bytes of a Transmit Object any time through the IFn Interface registers, neither MsgVal nor TxRqst have to be reset before the update. Even if only a part of the data bytes are to be updated, all four bytes of the corresponding IFn Data A Register or IFn Data B Register have to be valid before the contents of that register are transferred to the Message Object. Either the CPU has to write all four bytes into the IFn Data Register or the Message Object is transferred to the IFn Data Register before the CPU writes the new data bytes.

When only the (eight) data bytes are updated, first 0x0087 is written to the Command Mask Register and then the number of the Message Object is written to the Command Request Register, concurrently updating the data bytes and setting TxRqst.

To prevent the reset of TxRqst at the end of a transmission that may already be in progress while the data is updated, NewDat has to be set together with TxRqst. For details see *Section : Message transmission on page 300*.

When NewDat is set together with TxRqst, NewDat will be reset as soon as the new transmission has started.

12.7.5 Configuring a receive object

Table 60 shows how a Receive Object should be initialized.

Table 60. Initialization of a Receive Object

MsgVal	Arb	Data	Mask	EoB	Dir	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
1	appl.	appl.	appl.	1	0	0	0	appl.	0	0	0	0

The Arbitration Registers values (ID28-0 and Xtd bit) are provided by the application. They define the identifier and type of accepted received messages. If an 11-bit Identifier ("Standard Frame") is used, it is programmed to ID28 - ID18. Then ID17 - ID0 can be disregarded. When a Data Frame with an 11-bit Identifier is received, ID17 - ID0 will be set to '0'.

If the RxIE bit is set, the IntPnd bit will be set when a received Data Frame is accepted and stored in the Message Object.

The Data Length Code (DLC3-0) is provided by the application. When the Message Handler stores a Data Frame in the Message Object, it will store the received Data Length Code and eight data bytes. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by unspecified values.

The Mask Registers (Msk28-0, UMask, MXtd, and MDir bits) may be used (UMask='1') to allow groups of Data Frames with similar identifiers to be accepted. The Dir bit should not be masked in typical applications.

12.7.6 Handling received messages

The CPU may read a received message any time via the IF*n* Interface registers. The data consistency is guaranteed by the Message Handler state machine.

Typically, the CPU will write first 0x007F to the Command Mask Register and then the number of the Message Object to the Command Request Register. This combination will transfer the whole received message from the Message RAM into the Message Buffer Register. Additionally, the bits NewDat and IntPnd are cleared in the Message RAM (not in the Message Buffer).

If the Message Object uses masks for acceptance filtering, the arbitration bits shows which of the matching messages have been received.

The actual value of NewDat shows whether a new message has been received since the last time this Message Object was read. The actual value of MsgLst shows whether more than one message has been received since the last time this Message Object was read. MsgLst will not be automatically reset.

By means of a Remote Frame, the CPU may request another CAN node to provide new data for a receive object. Setting the TxRqst bit of a receive object will cause the transmission of a Remote Frame with the receive object's identifier. This Remote Frame triggers the other CAN node to start the transmission of the matching Data Frame. If the matching Data Frame is received before the Remote Frame could be transmitted, the TxRqst bit is automatically reset.

12.7.7 Configuring a FIFO buffer

With the exception of the EoB bit, the configuration of Receive Objects belonging to a FIFO Buffer is the same as the configuration of a (single) Receive Object, see *Section 12.7.5: Configuring a receive object on page 303*.

To concatenate two or more Message Objects into a FIFO Buffer, the identifiers and masks (if used) of these Message Objects have to be programmed to matching values. Due to the implicit priority of the Message Objects, the Message Object with the lowest number will be the first Message Object of the FIFO Buffer. The EoB bit of all Message Objects of a FIFO Buffer except the last have to be programmed to zero. The EoB bits of the last Message Object of a FIFO Buffer is set to one, configuring it as the End of the Block.

12.7.8 Receiving messages with FIFO buffers

Received messages with identifiers matching to a FIFO Buffer are stored in a Message Object of this FIFO Buffer starting with the Message Object with the lowest message number.

When a message is stored in a Message Object of a FIFO Buffer, the NewDat bit of this Message Object is set. By setting NewDat while EoB is zero, the Message Object is locked for further write access by the Message Handler until the application software has written the NewDat bit back to zero.

Messages are stored into a FIFO Buffer until the last Message Object of this FIFO Buffer is reached. If none of the preceding Message Objects is released by writing NewDat to zero, all further messages for this FIFO Buffer will be written into the last Message Object of the FIFO Buffer and therefore overwrite previous messages.

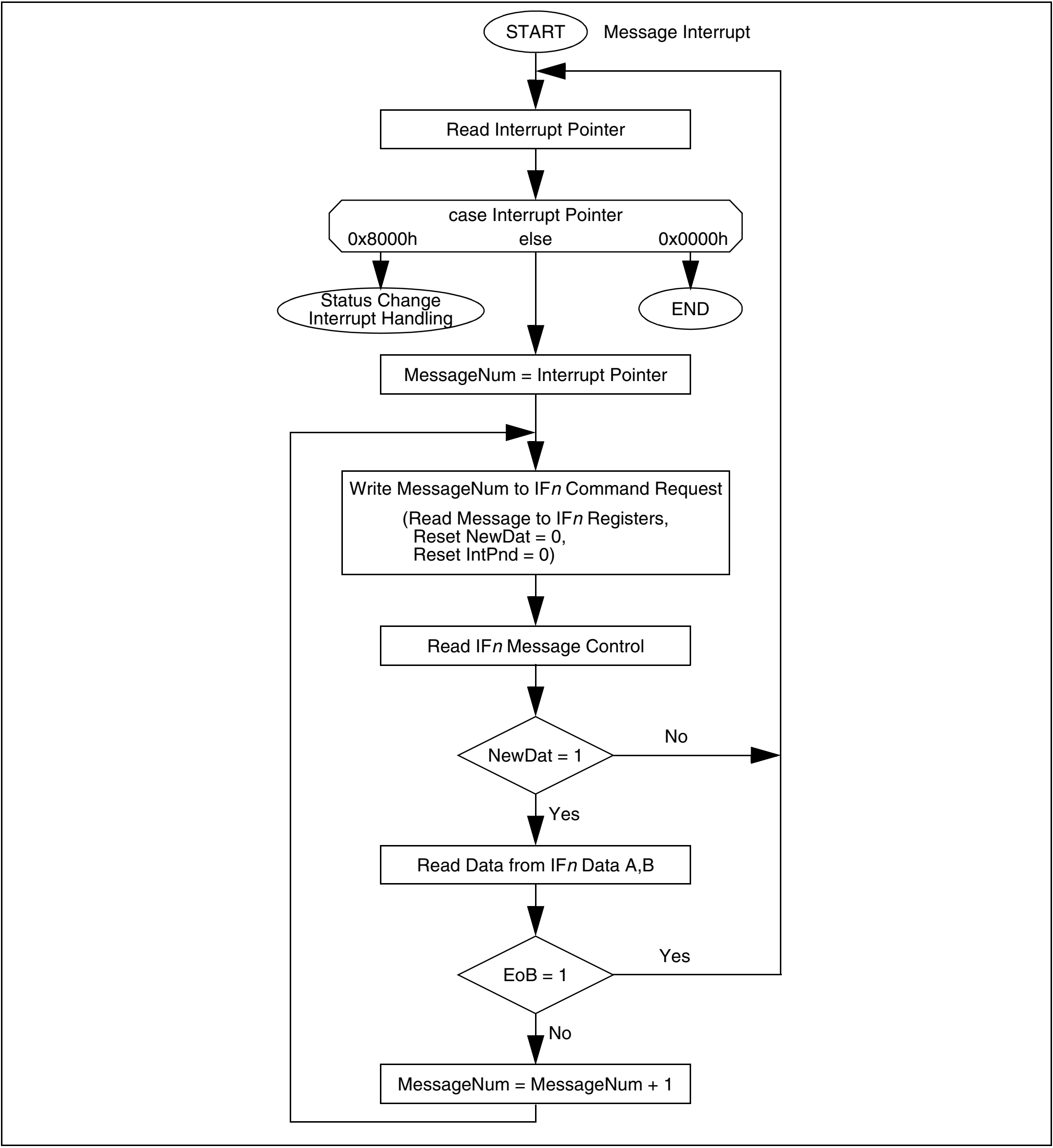
Reading from a FIFO buffer

When the CPU transfers the contents of a Message Object to the IFn Message Buffer register by writing its number to the IFn Command Request Register, the corresponding Command Mask Register should be programmed in such a way that bits NewDat and IntPnd are reset to zero (TxRqst/NewDat = ‘1’ and ClrIntPnd = ‘1’). The values of these bits in the Message Control Register always reflect the status before resetting the bits.

To assure the correct function of a FIFO Buffer, the CPU should read the Message Objects starting at the FIFO Object with the lowest message number.

Figure 89 shows how a set of Message Objects which are concatenated to a FIFO Buffer can be handled by the CPU.

Figure 89. CPU handling of a FIFO buffer


START
Message Interrupt
Read Interrupt Pointer
case Interrupt Pointer
0x8000h
else
0x0000h
Status Change Interrupt Handling
END
MessageNum = Interrupt Pointer
Write MessageNum to IFn Command Request
(Read Message to IFn Registers, Reset NewDat = 0, Reset IntPnd = 0)
Read IFn Message Control
NewDat = 1
No
Yes
Read Data from IFn Data A,B
EoB = 1
Yes
No
MessageNum = MessageNum + 1

12.7.9 Handling interrupts

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the application software has cleared it.

The Status Interrupt has the highest priority. Among the message interrupts, interrupt priority of the Message Object decreases with increasing message number.

A message interrupt is cleared by clearing the IntPnd bit of the Message Object. The Status Interrupt is cleared by reading the Status Register.

The interrupt identifier, IntId, in the Interrupt Register, indicates the cause of the interrupt. When no interrupt is pending, the register will hold the value zero. If the value of the Interrupt Register is different from zero, then there is an interrupt pending and, if IE is set, the IRQ interrupt signal to the EIC is active. The interrupt remains active until the Interrupt Register is back to value zero (the cause of the interrupt is reset) or until IE is reset.

The value 0x8000 indicates that an interrupt is pending because the CAN Core has updated (not necessarily changed) the Status Register (Error Interrupt or Status Interrupt). This interrupt has the highest priority. The CPU can update (reset) the status bits RxOk, TxOk and LEC, but a write access of the CPU to the Status Register can never generate or reset an interrupt.

All other values indicate that the source of the interrupt is one of the Message Objects. IntId points to the pending message interrupt with the highest interrupt priority.

The CPU controls whether a change of the Status Register may cause an interrupt (bits EIE and SIE in the CAN Control Register) and whether the interrupt line becomes active when the Interrupt Register is different from zero (bit IE in the CAN Control Register). The Interrupt Register will be updated even when IE is reset.

The CPU has two possibilities to follow the source of a message interrupt. First, it can follow the IntId in the Interrupt Register and second it can poll the Interrupt Pending Register (see *Section : Interrupt pending registers 1 & 2 (CAN_IPnR) on page 295*).

An interrupt service routine that is reading the message that is the source of the interrupt may read the message and reset the Message Object's IntPnd at the same time (bit ClrIntPnd in the Command Mask Register). When IntPnd is cleared, the Interrupt Register will point to the next Message Object with a pending interrupt.

12.7.10 Configuring the bit timing

Even if minor errors in the configuration of the CAN bit timing do not result in immediate failure, the performance of a CAN network can be reduced significantly.

In many cases, the CAN bit synchronization will amend a faulty configuration of the CAN bit timing to such a degree that only occasionally an error frame is generated. However, in the case of arbitration, when two or more CAN nodes simultaneously try to transmit a frame, a misplaced sample point may cause one of the transmitters to become error passive.

The analysis of such sporadic errors requires a detailed knowledge of the CAN bit synchronization inside a CAN node and interaction of the CAN nodes on the CAN bus.

Bit time and bit rate

CAN supports bit rates in the range of lower than 1 kBit/s up to 1000 kBit/s. Each member of the CAN network has its own clock generator, usually a quartz oscillator. The timing parameter of the bit time (i.e. the reciprocal of the bit rate) can be configured individually for each CAN node, creating a common bit rate even though the oscillator periods of the CAN nodes (f_{osc}) may be different.

The frequencies of these oscillators are not absolutely stable, small variations are caused by changes in temperature or voltage and by deteriorating components. As long as the variations remain inside a specific oscillator tolerance range (df), the CAN nodes are able to compensate for the different bit rates by re-synchronizing to the bit stream.

According to the CAN specification, the bit time is divided into four segments (see *Figure 90*). The Synchronization Segment, the Propagation Time Segment, the Phase Buffer Segment 1 and the Phase Buffer Segment 2. Each segment consists of a specific, programmable number of time quanta (see *Table 61*). The length of the time quantum (t_q), which is the basic time unit of the bit time, is defined by the CAN controller's system clock f_{APB} and the BRP bit of the Bit Timing Register (CAN_BTR): t_q = BRP / f_{APB}.

The Synchronization Segment, Sync_Seg, is that part of the bit time where edges of the CAN bus level are expected to occur. The distance between an edge, that occurs outside of Sync_Seg, and the Sync_Seg is called the phase error of that edge. The Propagation Time Segment, Prop_Seg, is intended to compensate for the physical delay times within the CAN network. The Phase Buffer Segments Phase_Seg1 and Phase_Seg2 surround the Sample Point. The (Re-)Synchronization Jump Width (SJW) defines how far a re-synchronization may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.

Figure 90. Bit timing

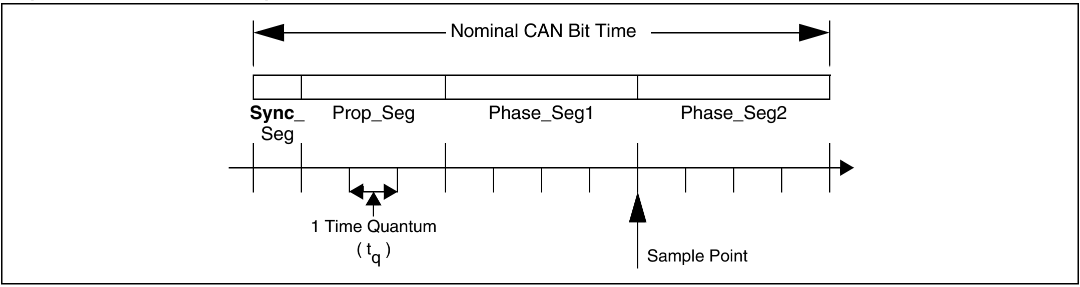


Table 61. CAN bit time parameters

Parameter	Range	Remark
BRP	[1 .. 32]	defines the length of the time quantum t_q
Sync_Seg	1 t_q	fixed length, synchronization of bus input to system clock
Prop_Seg	[1.. 8] t_q	compensates for the physical delay times
Phase_Seg1	[1..8] t_q	may be lengthened temporarily by synchronization
Phase_Seg2	[1.. 8] t_q	may be shortened temporarily by synchronization
SJW	[1 .. 4] t_q	may not be longer than either Phase Buffer Segment
This table describes the minimum programmable ranges required by the CAN protocol		

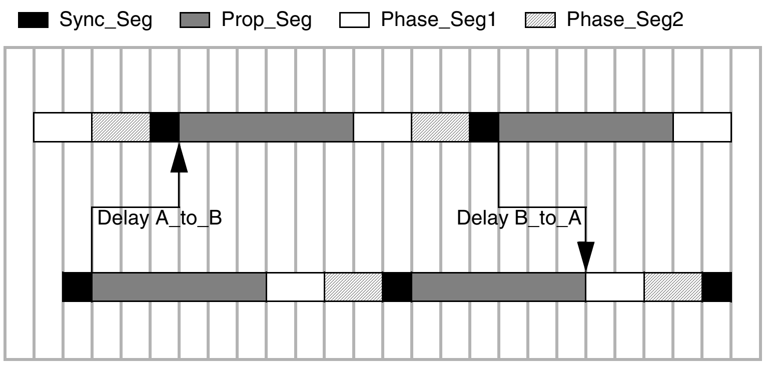
A given bit rate may be met by different bit time configurations, but for the proper function of the CAN network the physical delay times and the oscillator's tolerance range have to be considered.

Propagation time segment

This part of the bit time is used to compensate physical delay times within the network. These delay times consist of the signal propagation time on the bus and the internal delay time of the CAN nodes.

Any CAN node synchronized to the bit stream on the CAN bus will be out of phase with the transmitter of that bit stream, caused by the signal propagation time between the two nodes. The CAN protocol's non-destructive bitwise arbitration and the dominant acknowledge bit provided by receivers of CAN messages requires that a CAN node transmitting a bit stream must also be able to receive dominant bits transmitted by other CAN nodes that are synchronized to that bit stream. The example in *Figure 91* shows the phase shift and propagation times between two CAN nodes.

Figure 91. Propagation time segment



Delay A_to_B >= node output delay(A) + bus line delay(A→B) + node input delay(B)

Prop_Seg >= Delay A_to_B + Delay B_to_A

Prop_Seg >= 2 • [max(node output delay+ bus line delay + node input delay)]

In this example, both nodes A and B are transmitters, performing an arbitration for the CAN bus. Node A has sent its Start of Frame bit less than one bit time earlier than node B, therefore node B has synchronized itself to the received edge from recessive to dominant. Since node B has received this edge delay (A_to_B) after it has been transmitted, B's bit timing segments are shifted with respect to A. Node B sends an identifier with higher priority and so it will win the arbitration at a specific identifier bit when it transmits a dominant bit while node A transmits a recessive bit. The dominant bit transmitted by node B will arrive at node A after the delay (B_to_A).

Due to oscillator tolerances, the actual position of node A's Sample Point can be anywhere inside the nominal range of node A's Phase Buffer Segments, so the bit transmitted by node B must arrive at node A before the start of Phase_Seg1. This condition defines the length of Prop_Seg.

If the edge from recessive to dominant transmitted by node B arrives at node A after the start of Phase_Seg1, it can happen that node A samples a recessive bit instead of a dominant bit, resulting in a bit error and the destruction of the current frame by an error flag.

The error occurs only when two nodes arbitrate for the CAN bus that have oscillators of opposite ends of the tolerance range and that are separated by a long bus line. This is an example of a minor error in the bit timing configuration (Prop_Seg to short) that causes sporadic bus errors.

Some CAN implementations provide an optional 3 Sample Mode but the CAN peripheral does not. In this mode, the CAN bus input signal passes a digital low-pass filter, using three samples and a majority logic to determine the valid bit value. This results in an additional input delay of 1 t_q, requiring a longer Prop_Seg.

Phase buffer segments and synchronization

The Phase Buffer Segments (Phase_Seg1 and Phase_Seg2) and the Synchronization Jump Width (SJW) are used to compensate for the oscillator tolerance. The Phase Buffer Segments may be lengthened or shortened by synchronization.

Synchronizations occur on edges from recessive to dominant, their purpose is to control the distance between edges and Sample Points.

Edges are detected by sampling the actual bus level in each time quantum and comparing it with the bus level at the previous Sample Point. A synchronization may be done only if a recessive bit was sampled at the previous Sample Point and if the bus level at the actual time quantum is dominant.

An edge is synchronous if it occurs inside of Sync_Seg, otherwise the distance between edge and the end of Sync_Seg is the edge phase error, measured in time quanta. If the edge occurs before Sync_Seg, the phase error is negative, else it is positive.

Two types of synchronization exist, Hard Synchronization and Re-synchronization.

A Hard Synchronization is done once at the start of a frame and inside a frame only when Re-synchronizations occur.

- Hard Synchronization
  After a hard synchronization, the bit time is restarted with the end of Sync_Seg, regardless of the edge phase error. Thus hard synchronization forces the edge, which has caused the hard synchronization to lie within the synchronization segment of the restarted bit time.
- Bit Re-synchronization
  Re-synchronization leads to a shortening or lengthening of the bit time such that the position of the sample point is shifted with regard to the edge.
  When the phase error of the edge which causes Re-synchronization is positive, Phase_Seg1 is lengthened. If the magnitude of the phase error is less than SJW, Phase_Seg1 is lengthened by the magnitude of the phase error, else it is lengthened by SJW.
  When the phase error of the edge, which causes Re-synchronization is negative, Phase_Seg2 is shortened. If the magnitude of the phase error is less than SJW, Phase_Seg2 is shortened by the magnitude of the phase error, else it is shortened by SJW.

When the magnitude of the phase error of the edge is less than or equal to the programmed value of SJW, the results of Hard Synchronization and Re-synchronization are the same. If the magnitude of the phase error is larger than SJW, the Re-synchronization cannot compensate the phase error completely, an error (phase error - SJW) remains.

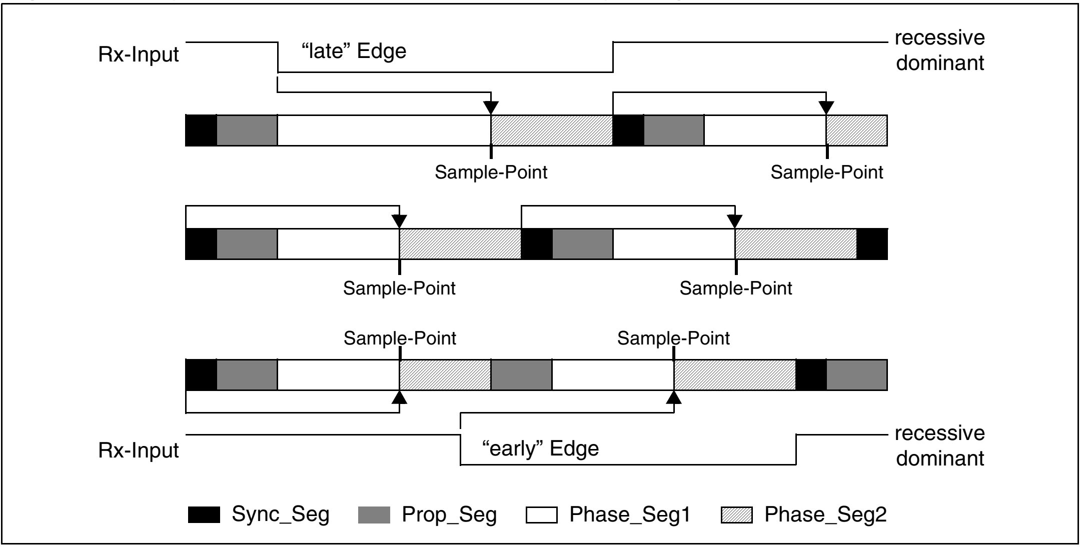
Only one synchronization may be done between two Sample Points. The Synchronizations maintain a minimum distance between edges and Sample Points, giving the bus level time to stabilize and filtering out spikes that are shorter than (Prop_Seg + Phase_Seg1).

Apart from noise spikes, most synchronizations are caused by arbitration. All nodes synchronize “hard” on the edge transmitted by the “leading” transceiver that started transmitting first, but due to propagation delay times, they cannot become ideally synchronized. The “leading” transmitter does not necessarily win the arbitration, therefore the receivers have to synchronize themselves to different transmitters that subsequently “take the lead” and that are differently synchronized to the previously “leading” transmitter. The same happens at the acknowledge field, where the transmitter and some of the receivers will have to synchronize to that receiver that “takes the lead” in the transmission of the dominant acknowledge bit.

Synchronizations after the end of the arbitration will be caused by oscillator tolerance, when the differences in the oscillator’s clock periods of transmitter and receivers sum up during the time between synchronizations (at most ten bits). These summarized differences may not be longer than the SJW, limiting the oscillator’s tolerance range.

The examples in *Figure 92* show how the Phase Buffer Segments are used to compensate for phase errors. There are three drawings of each two consecutive bit timings. The upper drawing shows the synchronization on a “late” edge, the lower drawing shows the synchronization on an “early” edge, and the middle drawing is the reference without synchronization.

Figure 92. Synchronization on “late” and “early” Edges



In the first example, an edge from recessive to dominant occurs at the end of Prop_Seg. The edge is “late” since it occurs after the Sync_Seg. Reacting to the “late” edge, Phase_Seg1 is lengthened so that the distance from the edge to the Sample Point is the same as it would have been from the Sync_Seg to the Sample Point if no edge had occurred. The phase error of this “late” edge is less than SJW, so it is fully compensated and the edge from dominant to recessive at the end of the bit, which is one nominal bit time long, occurs in the Sync_Seg.

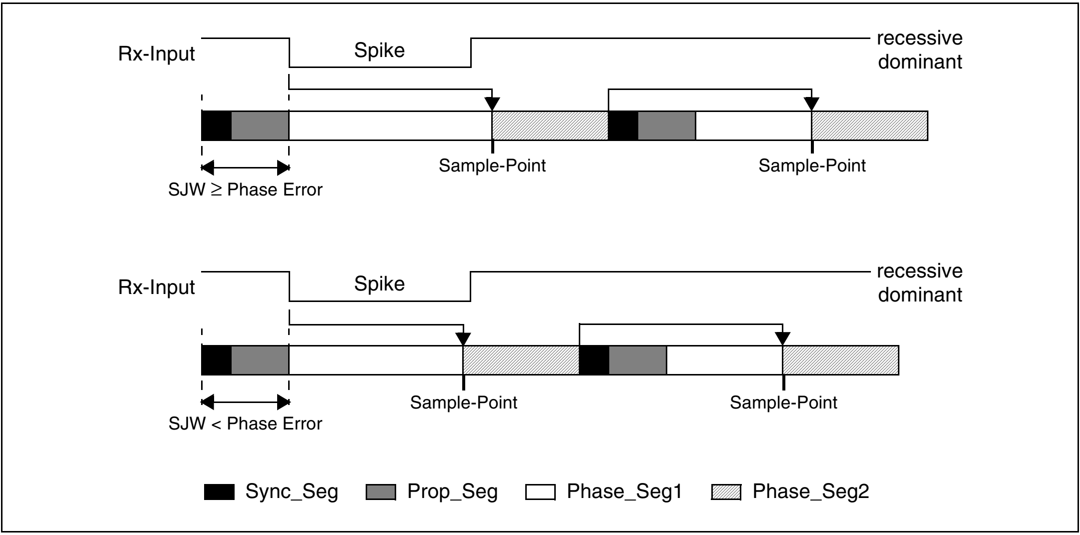
In the second example, an edge from recessive to dominant occurs during Phase_Seg2. The edge is “early” since it occurs before a Sync_Seg. Reacting to the “early” edge, Phase_Seg2 is shortened and Sync_Seg is omitted, so that the distance from the edge to the Sample Point is the same as it would have been from an Sync_Seg to the Sample Point if no edge had occurred. As in the previous example, the magnitude of phase error of this “early” edge’s is less than SJW, so it is fully compensated.

The Phase Buffer Segments are lengthened or shortened temporarily only; at the next bit time, the segments return to their nominal programmed values.

In these examples, the bit timing is seen from the point of view of the CAN state machine, where the bit time starts and ends at the Sample Points. The state machine omits Sync_Seg when synchronizing on an “early” edge, because it cannot subsequently redefine that time quantum of Phase_Seg2 where the edge occurs to be the Sync_Seg.

The examples in *Figure 93* show how short dominant noise spikes are filtered by synchronizations. In both examples the spike starts at the end of Prop_Seg and has the length of “Prop_Seg + Phase_Seg1”.

Figure 93. Filtering of short dominant spikes



In the first example, the Synchronization Jump Width is greater than or equal to the phase error of the spike’s edge from recessive to dominant. Therefore the Sample Point is shifted after the end of the spike; a recessive bus level is sampled.

In the second example, SJW is shorter than the phase error, so the Sample Point cannot be shifted far enough; the dominant spike is sampled as actual bus level.

Oscillator tolerance range

The oscillator tolerance range was increased when the CAN protocol was developed from version 1.1 to version 1.2 (version 1.0 was never implemented in silicon). The option to synchronize on edges from dominant to recessive became obsolete, only edges from recessive to dominant are considered for synchronization. The only CAN controllers to implement protocol version 1.1 have been Intel 82526 and Philips 82C200, both are superseded by successor products. The protocol update to version 2.0 (A and B) had no influence on the oscillator tolerance.

The tolerance range df for an oscillator frequency f_{osc} around the nominal frequency f_{nom} is:

$$(1 - df) \bullet f_{nom} \le f_{osc} \le (1 + df) \bullet f_{nom}$$

It depends on the proportions of Phase_Seg1, Phase_Seg2, SJW, and the bit time. The maximum tolerance df is the defined by two conditions (both shall be met):

$$\text{I:}\quad df \le \frac{\min(\text{Phase\_Seg1, Phase\_Seg2})}{2 \cdot (13 \cdot \text{bit\_time} - \text{Phase\_Seg2})}$$

$$\text{II:}\quad df \le \frac{\text{SJW}}{20 \cdot \text{bit\_time}}$$

It has to be considered that SJW may not be larger than the smaller of the Phase Buffer Segments and that the Propagation Time Segment limits that part of the bit time that may be used for the Phase Buffer Segments.

The combination Prop_Seg = 1 and Phase_Seg1 = Phase_Seg2 = SJW = 4 allows the largest possible oscillator tolerance of 1.58%. This combination with a Propagation Time Segment of only 10% of the bit time is not suitable for short bit times; it can be used for bit rates of up to 125 kBit/s (bit time = 8 µs) with a bus length of 40 m.

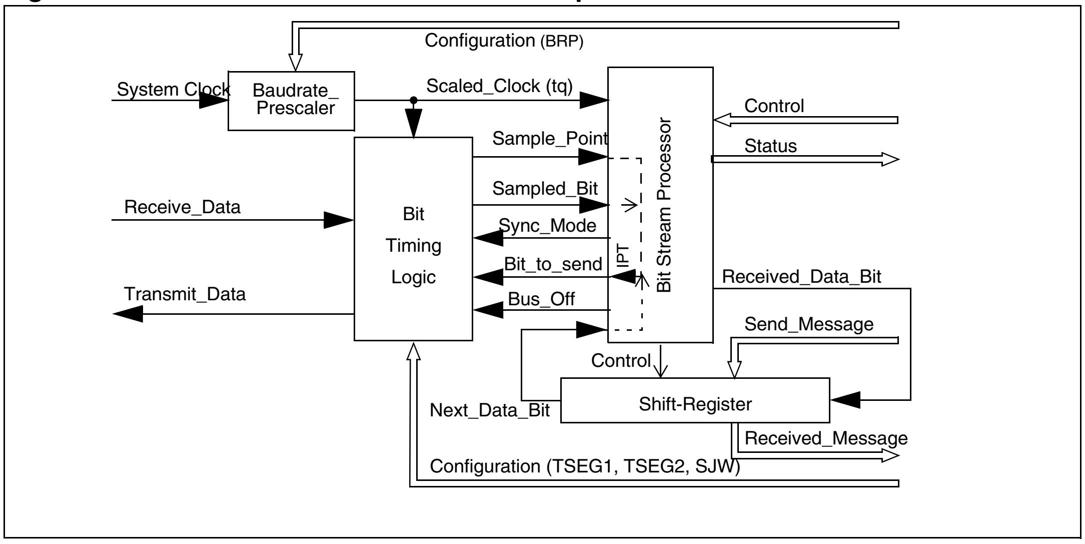
Configuring the CAN protocol controller

In most CAN implementations and also in the CAN peripheral, the bit timing configuration is programmed in two register bytes. The sum of Prop_Seg and Phase_Seg1 (as TSEG1) is combined with Phase_Seg2 (as TSEG2) in one byte, SJW and BRP are combined in the other byte (see *Figure 94 on page 312*).

In these bit timing registers, the four components TSEG1, TSEG2, SJW, and BRP have to be programmed to a numerical value that is one less than its functional value. Therefore, instead of values in the range of [1..n], values in the range of [0..n-1] are programmed. That way, e.g. SJW (functional range of [1..4]) is represented by only two bits.

Therefore the length of the bit time is (programmed values) [TSEG1 + TSEG2 + 3] t_q or (functional values) [Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2] t_q.

Figure 94. Structure of the CAN core's CAN protocol controller



The data in the bit timing registers is the configuration input of the CAN protocol controller. The Baud Rate Prescaler (configured by BRP) defines the length of the time quantum, the basic time unit of the bit time; the Bit Timing Logic (configured by TSEG1, TSEG2, and SJW) defines the number of time quanta in the bit time.

The processing of the bit time, the calculation of the position of the Sample Point, and occasional synchronizations are controlled by the BTL state machine, which is evaluated once each time quantum. The rest of the CAN protocol controller, the BSP state machine is evaluated once each bit time, at the Sample Point.

The Shift Register sends the messages serially and receives the messages parallely. Its loading and shifting is controlled by the BSP.

The BSP translates messages into frames and vice versa. It generates and discards the enclosing fixed format bits, inserts and extracts stuff bits, calculates and checks the CRC code, performs the error management, and decides which type of synchronization is to be used. It is evaluated at the Sample Point and processes the sampled bus input bit. The time that is needed to calculate the next bit to be sent after the Sample point(e.g. data bit, CRC bit, stuff bit, error flag, or idle) is called the Information Processing Time (IPT).

The IPT is application specific but may not be longer than 2 t_q; the IPT for the CAN peripheral is 0 t_q. Its length is the lower limit of the programmed length of Phase_Seg2. In case of a synchronization, Phase_Seg2 may be shortened to a value less than IPT, which does not affect bus timing.

Calculating bit timing parameters

Usually, the calculation of the bit timing configuration starts with a desired bit rate or bit time. The resulting bit time (1/bit rate) must be an integer multiple of the system clock period.

The bit time may consist of 4 to 25 time quanta, the length of the time quantum t_q is defined by the Baud Rate Prescaler with t_q = (Baud Rate Prescaler)/f_{sys}. Several combinations may lead to the desired bit time, allowing iterations of the following steps.

First part of the bit time to be defined is the Prop_Seg. Its length depends on the delay times measured in the system. A maximum bus length as well as a maximum node delay has to be defined for expandible CAN bus systems. The resulting time for Prop_Seg is converted into time quanta (rounded up to the nearest integer multiple of t_q).

The Sync_Seg is 1 t_q long (fixed), leaving (bit time – Prop_Seg – 1) t_q for the two Phase Buffer Segments. If the number of remaining t_q is even, the Phase Buffer Segments have the same length, Phase_Seg2 = Phase_Seg1, else Phase_Seg2 = Phase_Seg1 + 1.

The minimum nominal length of Phase_Seg2 has to be regarded as well. Phase_Seg2 may not be shorter than the IPT of the CAN controller, which, depending on the actual implementation, is in the range of [0..2] t_q.

The length of the Synchronization Jump Width is set to its maximum value, which is the minimum of 4 and Phase_Seg1.

The oscillator tolerance range necessary for the resulting configuration is calculated by the formulas given in *Section : Oscillator tolerance range on page 311*

If more than one configuration is possible, that configuration allowing the highest oscillator tolerance range should be chosen.

CAN nodes with different system clocks require different configurations to come to the same bit rate. The calculation of the propagation time in the CAN network, based on the nodes with the longest delay times, is done once for the whole network.

The oscillator tolerance range of the CAN systems is limited by that node with the lowest tolerance range.

The calculation may show that bus length or bit rate have to be decreased or that the stability of the oscillator frequency has to be increased in order to find a protocol compliant configuration of the CAN bit timing.

The resulting configuration is written into the Bit Timing Register:

(Phase_Seg2-1)&(Phase_Seg1+Prop_Seg-1)&

(SynchronisationJumpWidth-1)&(Prescaler-1)

Example for bit timing at high baudrate

In this example, the frequency of APB_CLK is 10 MHz, BRP is 0, the bit rate is 1 MBit/s.

t_q	100	ns	$= t_{APB\_CLK}$
delay of bus driver	50	ns	
delay of receiver circuit	30	ns	
delay of bus line (40m)	220	ns	
t_{Prop}	600	ns	$= 6 \cdot t_q$
t_{SJW}	100	ns	$= 1 \cdot t_q$
t_{TSeg1}	700	ns	$= t_{Prop} + t_{SJW}$
t_{TSeg2}	200	ns	= Information Processing Time + $1 \cdot t_q$
$t_{Sync\text{-}Seg}$	100	ns	$= 1 \cdot t_q$
bit time	1000	ns	$= t_{Sync\text{-}Seg} + t_{TSeg1} + t_{TSeg2}$
tolerance for APB_CLK	0.39	%	$= \frac{\min(PB1,PB2)}{2x(13xbit\_time-PB2)}$ $= \frac{0.1\mu s}{2x(13x1\mu s-0.2\mu s)}$

In this example, the concatenated bit time parameters are $(2\text{-}1)_3\&(7\text{-}1)_4\&(1\text{-}1)_2\&(1\text{-}1)_6$, the Bit Timing Register is programmed to= 0x1600.

Example for bit timing at low baudrate

In this example, the frequency of APB_CLK is 2 MHz, BRP is 1, the bit rate is 100 KBit/s.

t_q	1	µs	$= 2 \cdot t_{APB\_CLK}$
delay of bus driver	200	ns	
delay of receiver circuit	80	ns	
delay of bus line (40m)	220	ns	
t_{Prop}	1	µs	$= 1 \cdot t_q$
t_{SJW}	4	µs	$= 4 \cdot t_q$
t_{TSeg1}	5	µs	$= t_{Prop} + t_{SJW}$
t_{TSeg2}	4	µs	= Information Processing Time + $3 \cdot t_q$
$t_{Sync\text{-}Seg}$	1	µs	$= 1 \cdot t_q$
bit time	10	µs	$= t_{Sync\text{-}Seg} + t_{TSeg1} + t_{TSeg2}$
tolerance for APB_CLK	1.58	%	$= \frac{\min(PB1,PB2)}{2x(13xbit\_time-PB2)}$ $= \frac{4\mu s}{2x(13x10\mu s-4\mu s)}$

In this example, the concatenated bit time parameters are $(4\text{-}1)_3\&(5\text{-}1)_4\&(4\text{-}1)_2\&(2\text{-}1)_6$, the Bit Timing Register is programmed to= 0x34C1.

13 I^2C Interface Module (I^2C)

A I^2C Bus Interface serves as an interface between the microcontroller and the serial I^2C bus. It provides both multimaster and slave functions, and controls all I^2C bus-specific sequencing, protocol, arbitration and timing. It supports fast I^2C mode (400kHz).

13.1 Main features

- Parallel-bus/I^2C protocol converter
- Multi-master capability
- 7-bit/10-bit Addressing
- Transmitter/Receiver flag
- End-of-byte transmission flag
- Transfer problem detection
- Standard/Fast I^2C mode

I2C Master Features:

- Clock generation
- I^2C bus busy flag
- Arbitration Lost Flag
- End-of-byte transmission flag
- Transmitter/Receiver Flag
- Start bit detection flag
- Start and Stop generation

I2C Slave Features:

- Start bit detection flag
- Stop bit detection
- I^2C bus busy flag
- Detection of misplaced start or stop condition
- Programmable I^2C Address detection
- Transfer problem detection
- End-of-byte transmission flag
- Transmitter/Receiver flag

13.2 General description

In addition to receiving and transmitting data, this interface converts them from serial to parallel format and vice versa, using either an interrupt or polled handshake. The interrupts are enabled or disabled by software. The interface is connected to the I^2C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected both with a standard I^2C bus and a Fast I^2C bus. This selection is made by software.

13.2.1 Mode selection

The interface can operate in the four following modes:

- Slave transmitter/receiver
- Master transmitter/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 in case of arbitration loss or a STOP generation, allowing then Multi-Master capability.

13.2.2 Communication flow

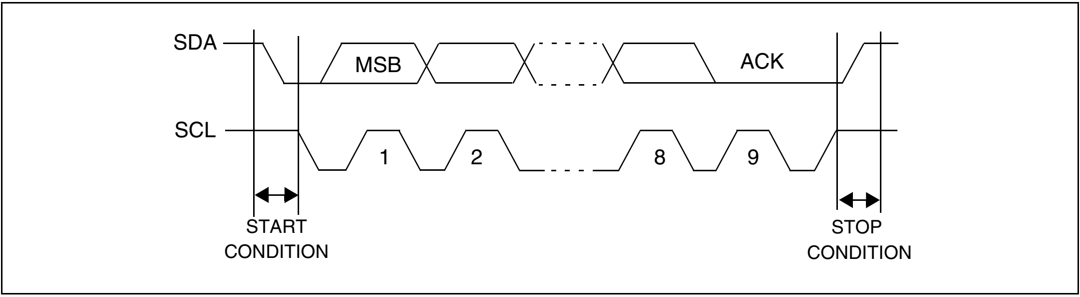
In Master mode, it initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by hardware as soon as the Master mode is selected.

In Slave mode, the interface is capable of recognizing its own address (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 *Figure 95*.

Figure 95. I^2C bus protocol



Acknowledge may be enabled and disabled by software.

The I^2C interface address and/or general call address can be selected by software.

The speed of the I^2C interface may be selected between Standard (0-100KHz) and Fast I^2C (100-400KHz).

13.2.3 SDA/SCL line control

Transmitter mode: the interface holds the clock line low before transmission to wait for the microcontroller to write the byte in the Data Register.

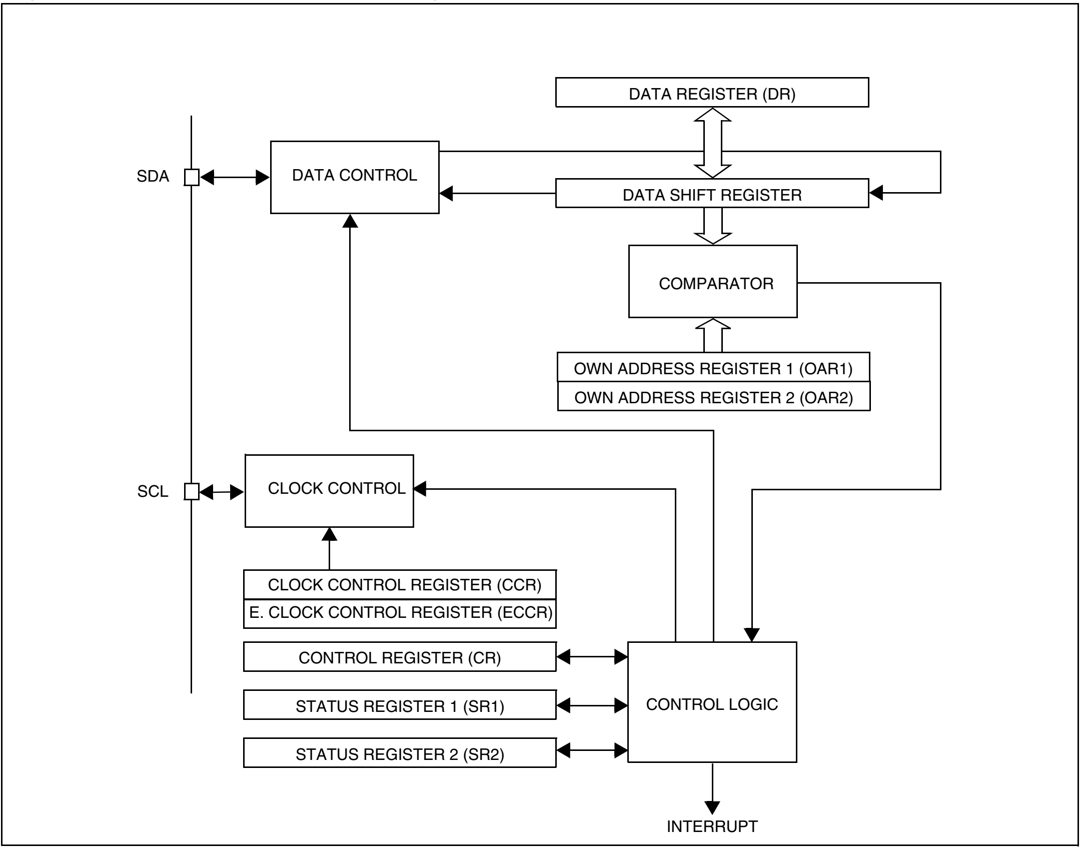
Receiver mode: the interface holds the clock line low after reception to wait for the microcontroller to read the byte in the Data Register.

The SCL frequency (f_{SCL}) is controlled by a programmable clock divider which depends on the I^2C bus mode.

After reset release, the I2C is able to detect a START condition on SDA and SCL lines even if the I2C is disabled (reset state of bit I2C_CR/PE is 0). If a start condition is detected, the BUSY bit of I2C_SR1 can be set. To clear this flag, a STOP condition must be detected.

Consequently, before enabling the I2C, care must be taken when configuring the I2C I/O lines to alternate functions (see the procedure in *Configuring I2C alternate functions on page 86*).

Figure 96. I^2C Interface block diagram



13.3 Functional description

Refer to the I2C_CR, I2C_SR1 and I2C_SR2 registers in *Section 13.5* for the bit definitions.

By default the I^2C interface operates in Slave mode (M/SL bit is cleared) except when it initiates a transmit or receive sequence.

First the interface frequency must be configured using the FRi bits in the I2C_OAR2 register.

13.3.1 Slave 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 or the General Call address (if selected by software).

Note: In 10-bit addressing mode, the comparison includes the header sequence (11110xx0) and the two most significant bits of the address.

Header matched (10-bit mode only): the interface generates an acknowledge pulse if the ACK bit is set.

Address not matched: the interface ignores it and waits for another Start condition.

Address matched: the interface generates in sequence:

- Acknowledge pulse if the ACK bit is set.
- EVF and ADSL bits are set with an interrupt if the ITE bit is set.

Then the interface waits for a read of the I2C_SR1 register, **holding the SCL line low** (see *Figure 97* Transfer sequencing EV1).
Next, in 7-bit mode read the I2C_DR register to determine from the least significant bit (Data Direction Bit) if the slave must enter Receiver or Transmitter mode.

In 10-bit mode, after receiving the address sequence the slave is always in receive mode. It will enter transmit mode on receiving a repeated Start condition followed by the header sequence with matching address bits and the least significant bit set (11110xx1).

Slave receiver

Following the address reception and after I2C_SR1 register has been read, the slave receives bytes from the SDA line into the I2C_DR register via the internal shift register. After each byte the interface generates in sequence:

- Acknowledge pulse if the ACK bit is set
- EVF and BTF bits are set with an interrupt if the ITE bit is set.

Then the interface waits for a read of the I2C_SR1 register followed by a read of the I2C_DR register, **holding the SCL line low** (see *Figure 97* Transfer sequencing EV2).

Slave transmitter

Following the address reception and after I2C_SR1 register has been read, the slave sends bytes from the I2C_DR register to the SDA line via the internal shift register.

The slave waits for a read of the I2C_SR1 register followed by a write in the I2C_DR register, **holding the SCL line low** (see *Figure 97* Transfer sequencing EV3).

When the acknowledge pulse is received:

- The EVF and BTF bits are set by hardware with an interrupt if the ITE bit is set.

Closing slave communication

After the last data byte is transferred a Stop Condition is generated by the master. The interface detects this condition and sets:

- EVF and STOPF bits with an interrupt if the ITE bit is set.

Then the interface waits for a read of the I2C_SR2 register (see *Figure 97* Transfer sequencing EV4).

Error cases

- **BERR**: Detection of a Stop or a Start condition during a byte transfer. In this case, the EVF and the BERR bits are set with an interrupt if the ITE bit is set.
  This detection is performed on the last 8 bits of a byte transfer but not on the first bit, as

a Start or Stop condition is a normal operation at this stage in Slave mode.
If it is a Stop then the interface discards the data, released the lines and waits for another Start condition.
If it is a Start then the interface discards the data and waits for the next slave address on the bus.

- **AF**: Detection of a non-acknowledge bit. In this case, the EVF and AF bits are set with an interrupt if the ITE bit is set.

Note: In both cases, SCL line is not held low; however, SDA line can remain low due to possible «0» bits transmitted last. It is then necessary to release both lines by software.

How to release the SDA / SCL lines

Set and subsequently clear the STOP bit while BTF is set. The SDA/SCL lines are released after the transfer of the current byte.

13.3.2 Master mode

To switch from default Slave mode to Master mode a Start condition generation is needed.

Start condition

Setting the START bit while the BUSY bit is cleared causes the interface to switch to Master mode (M/SL bit set) and generates a Start condition.

Once the Start condition is sent:

- The EVF and SB bits are set by hardware with an interrupt if the ITE bit is set.

Then the master waits for a read of the I2C_SR1 register followed by a write in the I2C_DR register with the Slave address, **holding the SCL line low** (see *Figure 97* Transfer sequencing EV5).

Slave address transmission

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

In 7-bit addressing mode, one address byte is sent.

In 10-bit addressing mode, sending the first byte including the header sequence causes the following event:

- The EVF bit is set by hardware with interrupt generation if the ITE bit is set.

Then the master waits for a read of the I2C_SR1 register followed by a write in the I2C_DR register, **holding the SCL line low** (see *Figure 97* Transfer sequencing EV9).

Then the second address byte is sent by the interface.

After completion of this transfer (and acknowledge from the slave if the ACK bit is set):

- The EVF bit is set by hardware with interrupt generation if the ITE bit is set.

Then the master waits for a read of the I2C_SR1 register followed by a write in the I2C_CR register (for example set PE bit), **holding the SCL line low** (see *Figure 97* Transfer sequencing EV6).

Next the master must enter Receiver or Transmitter mode.

Note: In 10-bit addressing mode, to switch the master to Receiver mode, software must generate a repeated Start condition and re-send the header sequence with the least significant bit set (11110xx1).

Master receiver

Following the address transmission and after I2C_SR1 and I2C_CR registers have been accessed, the master receives bytes from the SDA line into the I2C_DR register via the internal shift register. After each byte the interface generates in sequence:

- Acknowledge pulse if the ACK bit is set
- EVF and BTF bits are set by hardware with an interrupt if the ITE bit is set.

Then the interface waits for a read of the SR1 register followed by a read of the I2C_DR register, **holding the SCL line low** (see *Figure 97* Transfer sequencing EV7).

To close the communication: before reading the last byte from the I2C_DR register, set the STOP bit to generate the Stop condition. The interface goes automatically back to slave mode (M/SL bit cleared).

Note: In order to generate the non-acknowledge pulse after the last received data byte, the ACK bit must be cleared just before reading the second last data byte.

Master transmitter

Following the address transmission and after I2C_SR1 register has been read, the master sends bytes from the I2C_DR register to the SDA line via the internal shift register.

The master waits for a read of the I2C_SR1 register followed by a write in the I2C_DR register, **holding the SCL line low** (see *Figure 97* Transfer sequencing EV8).

When the acknowledge bit is received, the interface sets:

- EVF and BTF bits with an interrupt if the ITE bit is set.

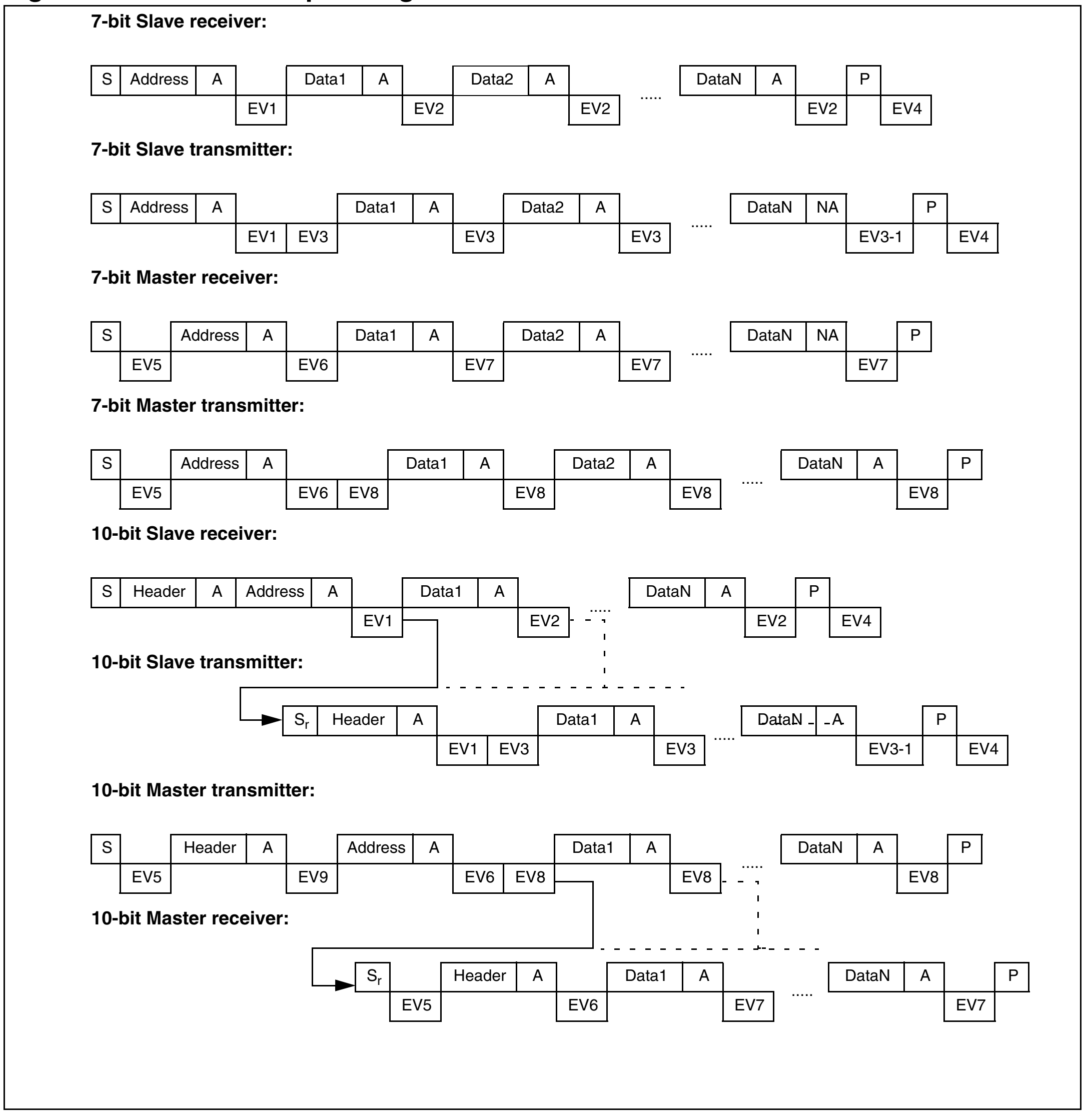
To close the communication: after writing the last byte to the I2C_DR register, set the STOP bit to generate the Stop condition. The interface goes automatically back to slave mode (M/SL bit cleared).

Error cases

- **BERR**: Detection of a Stop or a Start condition during a byte transfer (on all bits). In this case, the EVF and BERR bits are set by hardware with an interrupt if ITE is set.
- **AF**: Detection of a non-acknowledge bit. In this case, the EVF and AF bits are set by hardware with an interrupt if the ITE bit is set. To resume, set the START or STOP bit.
- **ARLO:** Detection of an arbitration lost condition.
  In this case the ARLO bit is set by hardware (with an interrupt if the ITE bit is set and the interface goes automatically back to slave mode (the M/SL bit is cleared).

Note: In all these cases, the SCL line is not held low; however, the SDA line can remain low due to possible «0» bits transmitted last. It is then necessary to release both lines by software.

Figure 97. Transfer sequencing


7-bit Slave receiver:
S Address A Data1 A Data2 A ..... DataN A P
EV1 EV2 EV2 EV2 EV4
7-bit Slave transmitter:
S Address A Data1 A Data2 A ..... DataN NA P
EV1 EV3 EV3 EV3 EV3-1 EV4
7-bit Master receiver:
S Address A Data1 A Data2 A ..... DataN NA P
EV5 EV6 EV7 EV7 EV7
7-bit Master transmitter:
S Address A Data1 A Data2 A ..... DataN A P
EV5 EV6 EV8 EV8 EV8 EV8
10-bit Slave receiver:
S Header A Address A Data1 A ..... DataN A P
EV1 EV2 EV2 EV4
10-bit Slave transmitter:
Sr Header A Data1 A ..... DataN A P
EV1 EV3 EV3 EV3-1 EV4
10-bit Master transmitter:
S Header A Address A Data1 A ..... DataN A P
EV5 EV9 EV6 EV8 EV8 EV8
10-bit Master receiver:
Sr Header A Data1 A ..... DataN A P
EV5 EV6 EV7 EV7

Legend:

S=Start, S_r = Repeated Start, P=Stop, A=Acknowledge, NA=Non-acknowledge, EVx=Event (with interrupt if ITE=1)

EV1: EVF=1, ADSL=1, cleared by reading I2C_SR1 register.

EV2: EVF=1, BTF=1, cleared by reading I2C_DR register.

EV3: EVF=1, BTF=1, cleared by reading I2C_SR1 register followed by writing to the DR register.

EV3-1: EVF=1, AF=1, BTF=1; AF is cleared by reading SR2 register. BTF is cleared by releasing the lines (STOP=1, STOP=0) or by writing I2C_DR register (DR=FFh).

Note: If lines are released bySTOP=1, STOP=0, the subsequent EV4 is not seen.

EV4: EVF=1, STOPF=1, cleared by reading SR2 register.

EV5: EVF=1, SB=1, cleared by reading I2C_SR1 register followed by writing I2C_DR register.

EV6: EVF=1, ENDAD=1 cleared by reading I2C_SR2 register followed by writing I2C_CR register (for example PE=1).

EV7: EVF=1, BTF=1, cleared by reading the I2C_DR register.

EV8: EVF=1, BTF=1, cleared by writing to the I2C_DR register.

EV9: EVF=1, ADD10=1, cleared by reading the I2C_SR1 register followed by writing to the I2C_DR register.

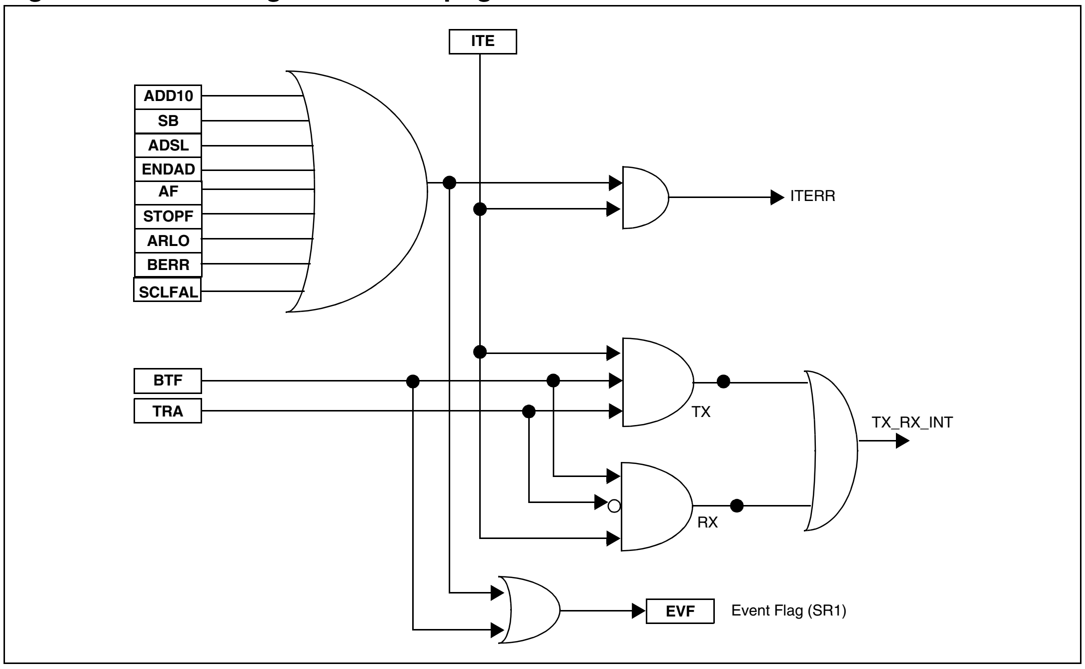
13.4 Interrupts

Several interrupt events can be flagged by the module:

- requests related to bus events, like start or stop events, arbitration lost, etc.;
- requests related to data transmission and/or reception;

These requests are issued to the interrupt controller by two different lines as described in *Figure 98*. The different flags identify the events and can be polled by the software (interrupt service routine).

Figure 98. Event flags and interrupt generation



13.5 Register description

13.5.1 I^2C Control Register (I2C_CR)

Address Offset: 00h
Reset value: 00h

7	6	5	4	3	2	1	0
reserved		PE	ENGC	START	ACK	STOP	ITE
-		rw	rw	rw	rw	rw	rw

Bit 7:6	Reserved, forced by hardware to ‘0’.
Bit 5	**PE:** *Peripheral Enable.* This bit is set and cleared by software. 0: Peripheral disabled 1: Master/Slave capability *Notes:* 0: all the bits of the I2C_CR register and the I2C_SR register except the STOP and BUSY bit are reset. All outputs are released while PE=0. 1: the corresponding I/O pins are selected by hardware as alternate functions. To enable the I^2C interface, write the I2C_CR register **TWICE** with PE=1 as the first write only activates the interface (only PE is set).
Bit 4	**ENGC**: *Enable General Call.* This bit is set and cleared by software. It is also cleared by hardware when the interface is disabled (PE=0). The 00h General Call address is acknowledged (01h ignored). 0: General Call disabled. 1: General Call enabled.
Bit 3	**START**: *Generation of a Start condition.* This bit is set and cleared by software. It is also cleared by hardware when the interface is disabled (PE=0) or when the Start condition is sent (with interrupt generation if ITE=1). **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.
Bit 2	**ACK**: *Acknowledge enable.* This bit is set and cleared by software. It is also cleared by hardware when the interface is disabled (PE=0). 0: No acknowledge returned 1: Acknowledge returned after an address byte or a data byte is received

Bit 1	**STOP**: *Generation of a Stop condition.* This bit is set and cleared by software. It is also cleared by hardware in master mode. Note: This bit is not cleared when the interface is disabled (PE=0). **In master mode:** 0: No stop generation. 1: Stop generation after the current byte transfer or after the current Start condition is sent. The STOP bit is cleared by hardware when the Stop condition is sent. **In slave mode:** 0: No stop generation. 1: Release the SCL and SDA lines after the current byte transfer (BTF=1). In this mode the STOP bit has to be cleared by software.
Bit 0	**ITE**: *Interrupt enable.* This bit is set and cleared by software and cleared by hardware when the interface is disabled (PE=0). 0: Interrupts disabled. 1: Interrupts enabled. Refer to *Figure 98* for the relationship between the events and the interrupts. SCL is held low when the ADD10, SB, BTF or ADSL flags or an EV6 event (See *Figure 97*) is detected.

13.5.2 I^{2}C Status Register 1 (I2C_SR1)

Address Offset: 04h
Reset value: 00h

7	6	5	4	3	2	1	0
EVF	ADD10	TRA	BUSY	BTF	ADSL	M/SL	SB
r	r	r	r	r	r	r	r

Bit 7	**EVF**: *Event flag.* This bit is set by hardware as soon as an event occurs. It is cleared by software reading I2C_SR2 register in case of error event or as described in *Figure 97*. It is also cleared by hardware when the interface is disabled (PE=0). 0: No event 1: One of the following events has occurred: - BTF=1 (Byte received or transmitted) - ADSL=1 (Address matched in Slave mode while ACK=1) - SB=1 (Start condition generated in Master mode) - AF=1 (No acknowledge received after byte transmission) - STOPF=1 (Stop condition detected in Slave mode) - ARLO=1 (Arbitration lost in Master mode) - BERR=1 (Bus error, misplaced Start or Stop condition detected) - ADD10=1 (Master has sent header byte) - ENDAD=1 (Address byte successfully transmitted in Master mode).
Bit 6	**ADD10**: *10-bit addressing in Master mode.* This bit is set by hardware when the master has sent the first byte in 10-bit address mode. It is cleared by software reading I2C_SR2 register followed by a write in the I2C_DR register of the second address byte. It is also cleared by hardware when the peripheral is disabled (PE=0). 0: No ADD10 event occurred. 1: Master has sent first address byte (header).

Bit 5	**TRA**: *Transmitter/Receiver.* When BTF is set, TRA=1 if a data byte has been transmitted. It is cleared automatically when BTF is cleared. It is also cleared by hardware after detection of Stop condition (STOPF=1), loss of bus arbitration (ARLO=1) or when the interface is disabled (PE=0). 0: Data byte received (if BTF=1). 1: Data byte transmitted.
Bit 4	**BUSY**: *Bus busy.* This bit is set by hardware on detection of a Start condition and 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). 0: No communication on the bus 1: Communication ongoing on the bus
Bit 3	**BTF**: *Byte transfer finished.* This bit is set by hardware as soon as a byte is correctly received or transmitted with interrupt generation if ITE=1. It is cleared by software reading I2C_SR1 register followed by a read or write of I2C_DR register. It is also cleared by hardware when the interface is disabled (PE=0). Following a byte reception, this bit is set after transmission of the acknowledge clock pulse if ACK=1. BTF is cleared by reading I2C_SR1 register followed by reading the byte from I2C_DR register. Following a byte transmission, this bit is set after reception of the acknowledge clock pulse. In case an address byte is sent, this bit is set only after the EV6 event (See *Figure 97*). BTF is cleared by writing the next byte in I2C_DR register. The SCL line is held low while BTF=1. 0: Byte transfer not done 1: Byte transfer succeeded
Bit 2	**ADSL**: *Address matched (Slave mode).* This bit is set by hardware as soon as the received slave address matched with the I2C_OAR register content or a general call is recognized. An interrupt is generated if ITE=1. It is cleared by software reading I2C_SR1 register or by hardware when the interface is disabled (PE=0). The SCL line is held low while ADSL=1. 0: Address mismatched or not received. 1: Received address matched.
Bit 1	**M/SL**: *Master/Slave.* This bit is set by hardware as soon as the interface is in Master mode (writing START=1). It is cleared by hardware after detecting a Stop condition on the bus or a loss of arbitration (ARLO=1). It is also cleared when the interface is disabled (PE=0). 0: Slave mode. 1: Master mode.
Bit 0	**SB**: *Start bit (Master mode).* This bit is set by hardware as soon as the Start condition is generated (following a write START=1). An interrupt is generated if ITE=1. It is cleared by software reading I2C_SR1 register followed by writing the address byte in I2C_DR register. It is also cleared by hardware when the interface is disabled (PE=0). 0: No Start condition. 1: Start condition generated.

13.5.3 I^2C Status Register 2 (I2C_SR2)

Address Offset: 08h
Reset value: 00h

7	6	5	4	3	2	1	0
reserved		ENDAD	AF	STOPF	ARLO	BERR	GCAL
-		r	r	r	r	r	r

Bit 7:6	Reserved, forced by hardware to ‘0’.
Bit 5	**ENDAD**: *End of address transmission.* This bit is set by hardware when: - 7-bit addressing mode: the address byte has been transmitted; - 10-bit addressing mode: the MSB and the LSB have been transmitted during the addressing phase. When the master needs to receive data from the slave, it has to send just the MSB of the slave address once again; hence the ENDAD flag is set, without waiting for the LSB of the address.It is cleared by software by reading SR2 and a following write to the CR or by hardware when the interface is disabled (PE=0). 0: No end of address transmission 1: End of address transmission
Bit 4	**AF**: *Acknowledge failure*. This bit is set by hardware when no acknowledge is returned. An interrupt is generated if ITE=1. It is cleared by software by reading I2C_SR2 register or by hardware when the interface is disabled (PE=0). The SCL line is not held low while AF=1. 0: No acknowledge failure 1: Acknowledge failure
Bit 3	**STOPF**: *Stop detection (Slave mode).* This bit is set by hardware when a Stop condition is detected on the bus after an acknowledge (if ACK=1). An interrupt is generated if ITE=1. It is cleared by software reading I2C_SR2 register or by hardware when the interface is disabled (PE=0). The SCL line is not held low while STOPF=1. 0: No Stop condition detected 1: Stop condition detected
Bit 2	**ARLO**: *Arbitration lost*. This bit is set by hardware when the interface loses the arbitration of the bus to another master. An interrupt is generated if ITE=1. It is cleared by software reading I2C_SR2 register or by hardware when the interface is disabled (PE=0). After an ARLO event the interface switches back automatically to Slave mode (M/SL=0). The SCL line is not held low while ARLO=1. 0: No arbitration lost detected 1: Arbitration lost detected

Bit 1	**BERR**: *Bus error.* This bit is set by hardware when the interface detects a misplaced Start or Stop condition on all bits of a byte transfer in master mode and on the last 8 bits of a byte transfer in slave mode. An interrupt is generated if ITE=1. It is cleared by software reading I2C_SR2 register or by hardware when the interface is disabled (PE=0). The SCL line is not held low while BERR=1. 0: No misplaced Start or Stop condition 1: Misplaced Start or Stop condition
Bit 0	**GCAL:** *General Call (Slave mode).* This bit is set by hardware when a general call address is detected on the bus while ENGC=1. It is cleared by hardware detecting a Stop condition (STOPF=1) or when the interface is disabled (PE=0). 0: No general call address detected on bus 1: general call address detected on bus

13.5.4 I^2C Clock Control Register (I2C_CCR)

Address Offset: 0Ch
Reset value: 00h

7	6	5	4	3	2	1	0
FM/SM	CC6	CC5	CC4	CC3	CC2	CC1	CC0
rw	rw	rw	rw	rw	rw	rw	rw

Bit 7	**FM/SM**: *Fast/Standard I^2C mode.* This bit is set and cleared by software. It is not cleared when the interface is disabled (PE=0). 0: Standard I^2C mode 1: Fast I^2C mode
Bit 6:0	**CC[6:0]**: *12-bit clock divider.* These bits along with CC11-CC7 of the Extended Clock Control Register select the speed of the bus (f_{SCL}) depending on the I^2C mode. They are not cleared when the interface is disabled (PE=0). – Standard mode (FM/SM=0): $f_{SCL} \leq 100$ kHz $f_{SCL} = f_{PCLK} / ((2 \times CC[11:0]) + 7)$ Given a certain f_{PCLK} it is easy to obtain the right divider factor: $CC[11:0] = ((f_{PCLK} / f_{SCL}) - 7) / 2 = ((t_{SCL} / t_{PCLK}) - 7) / 2$ – Fast mode (FM/SM=1): $100kHz < f_{SCL} < 400kHz$ $f_{SCL} = f_{PCLK} / ((3 \times [CC11...CC0]) + 9)$ Given a certain f_{PCLK} it is easy to obtain the right divider factor: $CC[11:0] = ((f_{PCLK} / f_{SCL}) - 9) / 3 = ((t_{SCL} / t_{PCLK}) - 9) / 3$

Note: The programmed f_{SCL} assumes no load on SCL and SDA lines.

Note: For a correct usage of the divider, [CC11...CC0] must be equal or greater than 0x002 (000000000010b). [CC11...CC0] equal to 0x001 (000000000001b) is not admitted.

13.5.5 I^{2}C Extended Clock Control Register (I2C_ECCR)

Address Offset: 1Ch
Reset value: 00h

7	6	5	4	3	2	1	0
reserved			CC11	CC10	CC9	CC8	CC7
			rw	rw	rw	rw	rw

Bits 7:5	Reserved, forced by hardware to ‘0’.
Bits 6:0	**CC[11:7]**: *12-bit clock divider.* These bits along with those of the Clock Control Register select the speed of the bus (f_{SCL}) depending on the I^{2}C mode. They are not cleared when the interface is disabled (PE=0)

13.5.6 I^{2}C Own Address Register 1 (I2C_OAR1)

Address Offset: 10h
Reset value: 00h

7	6	5	4	3	2	1	0
ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0
rw	rw	rw	rw	rw	rw	rw	rw

7-bit Addressing Mode

Bits 7:1	**ADD[7:1]**: *Interface address.* These bits define the I^{2}C bus address of the interface. They are not cleared when the interface is disabled (PE=0).
Bit 0	**ADD[0]**: *Address direction bit.* This bit is don’t care, the interface acknowledges either 0 or 1. It is not cleared when the interface is disabled (PE=0). Note: Address 01h is always ignored.

10-bit Addressing Mode

Bits 7:0	**ADD[7:0]**: *Interface address.* These are the least significant bits of the I^{2}C bus address of the interface. They are not cleared when the interface is disabled (PE=0).

13.5.7 I^2C Own Address Register 2 (I2C_OAR2)

Address Offset: 14h
Reset value: 40h

7	6	5	4	3	2	1	0
FR2	FR1	FR0	reserved		ADD9	ADD8	res.
rw	rw	rw	-		rw	rw	-

Bit 7:5	**FR[2:0]**: *Frequency bits.* These bits are set by software only when the interface is disabled (PE=0). To configure the interface to I^2C specified delays select the value corresponding to f_{PCLK}: 000: f_{PCLK} = 5 to 10 MHz 001: f_{PCLK} = 10 to 16.67 MHz 010: f_{PCLK} = 16.67 to 26.67 MHz 011: f_{PCLK} = 26.67 to 40 MHz
Bit 4:3	Reserved, forced by hardware to ‘0’.
Bit 2:1	**ADD[9:8]**: *Interface address.* These are the most significant bits of the I^2C bus address of the interface (10-bit mode only). They are not cleared when the interface is disabled (PE=0).
Bit 0	Reserved, forced by hardware to ‘0’.

13.5.8 I^2C Data Register (I2C_DR)

Address Offset: 18h
Reset value: 00h

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
rw	rw	rw	rw	rw	rw	rw	rw

Bit 7:0	**D[7:0]**: *8-bit Data Register.* These bits contain the byte to be received or transmitted on the bus. **Transmitter mode**: Byte transmission start automatically when the software writes in the I2C_DR register. **Receiver mode**: the first data byte is received automatically in the I2C_DR register using the least significant bit of the address. Then, the following data bytes are received one by one after reading the I2C_DR register.

13.6 I2C Register map

Table 62. I^{2}C Interface register map

Address Offset	Register Name	7	6	5	4	3	2	1	0
00	I2C_CR	reserved		PE	ENGC	START	ACK	STOP	ITE
04	I2C_SR1	EVF	ADD10	TRA	BUSY	BTF	ADSL	M/SL	SB
08	I2C_SR2	reserved		ENDAD	AF	STOPF	ARLO	BERR	GCAL
0C	I2C_CCR	FM/SM	CC6	CC5	CC4	CC3	CC2	CC1	CC0
10	I2C_OAR1	ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0
14	I2C_OAR2	FR2	FR1	FR0	reserved		ADD9	ADD8	res.
18	I2C_DR	D7	D6	D5	D4	D3	D2	D1	D0
1C	I2C_ECCR	reserved			CC11	CC10	CC9	CC8	CC7

See *Table 2* for the base address.

14 Synchronous Serial Peripheral (SSP)

The SSP is a master or slave interface for synchronous serial communication with peripheral devices that have either Motorola SPI, or Texas Instruments SSI synchronous serial interfaces.

The SSP performs serial-to-parallel conversion on data received from a peripheral device. The CPU accesses data, control, and status information through the AMBA APB interface. The transmit and receive paths are buffered with internal FIFO memories allowing up to eight 16-bit values to be stored independently in both transmit and receive modes. The SSP includes a programmable bit rate clock divider and prescaler to generate the serial output clock SCK from the input clock PCLK.

The SSP operating mode, frame format, and size are programmed through the control registers SSP_CR0 and SSP_CR1.

Four individually maskable interrupt events are generated:

- A TX event requests servicing of the transmit buffer
- A RX event requests servicing of the receive buffer
- An ROR event indicates an overrun condition in the receive FIFO
- An RT event indicates that a timeout expired while data was present in the receive FIFO.

In addition to the above interrupts, a set of DMA signals are provided for interfacing with a DMA controller.

14.1 Main features

- Master and Slave modes supported
- Programmable choice of interface operation: Motorola SPI or TI synchronous serial
- Programmable data frame size from 4 to 16 bits
- Programmable bit rate and Internal clock prescaler
- Separate transmit and receive FIFO buffers, 16 bits wide, 8 locations deep
- Independent masking of transmit FIFO, receive FIFO, timeout and overrun interrupts
- Bit rate up to 16Mbits/s for SSP0 and 8Mbits/s for SSP1
- Slave select NSS: Hardware or Software management
- Dynamic change of Master/Slave operations
- DMA interface (SSP0):
  – Receive DMA Single or Burst requests
  – Transmit DMA Single or Burst requests
- Internal loopback test mode

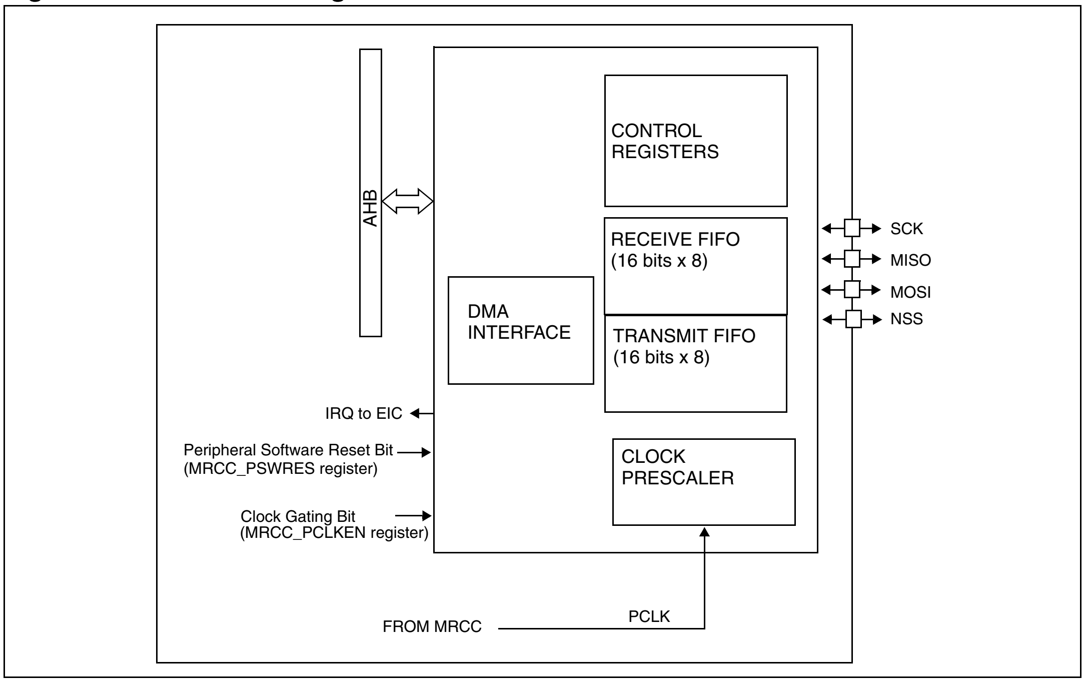
14.2 Functional description

The processor views the SSP as a memory mapped peripheral, which may be used by standard polling, interrupt programming techniques or DMA controlled access.

When an SSP transfer occurs data is transmitted and received simultaneously. A serial clock line synchronizes shifting and sampling of the information on the two serial data lines. A slave select line allows individual selection of a slave device. The central elements in the SSP system are the 16-bit shift register and the two read data buffer which each one is 8 words x 16-bit. An SSP-DMA interface is also present to allow for data to be transfered to/from memory using the DMA

A block diagram of the SSP is shown in *Figure 99.*

Figure 99. SSP block diagram


AHB
CONTROL REGISTERS
RECEIVE FIFO (16 bits x 8)
TRANSMIT FIFO (16 bits x 8)
DMA INTERFACE
CLOCK PRESCALER
SCK
MISO
MOSI
NSS
IRQ to EIC
Peripheral Software Reset Bit (MRCC_PSWRES register)
Clock Gating Bit (MRCC_PCLKEN register)
FROM MRCC
PCLK

14.2.1 Pin description

The SSP is a four wire, bi-directional bus. The data path is determined by the mode of operation selected.

The SSP is connected to external devices through 3 or 4 I/O pins, see table below:

- MISO: Master In Slave Out pin
- MOSI: Master Out Slave In pin
- SCK: Serial Clock pin
- NSS: Slave Select pin (not required if Slave Select signal is managed in software mode)

Table 63. SSP Pins

Pin name	Description
SCK	Serial Clock pin: The bit clock for all data transfers. When the SSP is a master the SCK is output from the chip. When configured as a slave the SCK is input from the external source.
MISO	Master Input/ Slave Output serial data line
MOSI	Master Output/ Slave Input serial data line
NSS	Slave Select pin: In slave mode, NSS is an input pin used to select the slave. It must be pulled low by the master after the SCK is stable and held low for the duration of the data transfer. In master mode, NSS acts as an output pin and can be used to select a slave.

A basic example of interconnections between a single master and a single slave is illustrated in *Figure 100* and *Figure 101*.

The MOSI pins are connected together as are MISO pins. In this way data is transferred serially between master and slave (most significant bit first).

When the master device transmits data to a slave device via MOSI pin, the slave device responds by sending data to the master device via the MISO pin. This implies full duplex transmission with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).

Figure 100. Interconnection example

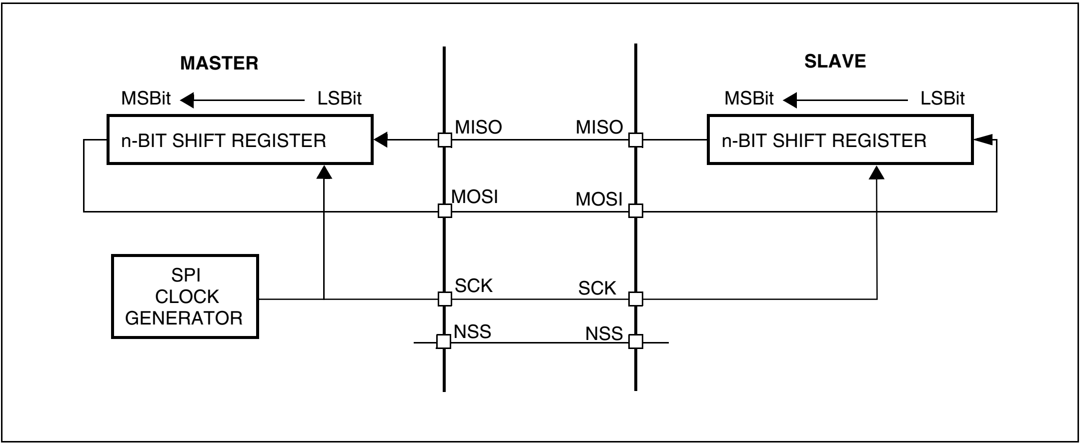
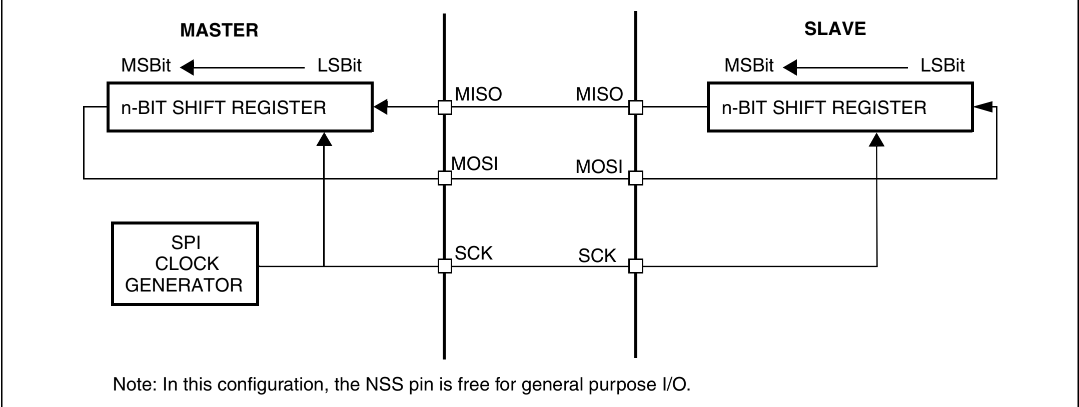


Figure 101. Interconnection example using NSS software management



14.2.2 Master mode

When configured as a master, the clock to the attached slaves is derived from a divided down version of PCLK through the prescaler operations. The master transmit logic successively reads a value from its transmit FIFO and performs parallel to serial conversion on it. Then the serial data stream and frame control signal, synchronized to SCK, are output through the MOSI pin to the attached slaves. The master receive logic performs serial to parallel conversion on the incoming synchronous MISO data stream, extracting and storing values into its receive FIFO, for subsequent reading through the APB interface.

14.2.3 Slave mode

When configured as a slave, the SCK clock is provided by an attached master and used to time its transmission and reception sequences. The slave transmit logic, under control of the master clock, successively reads a value from its transmit FIFO, performs parallel to serial conversion, then output the serial data stream and frame control signal through the slave MISO pin. The slave receive logic performs serial to parallel conversion on the incoming MOSI data stream, extracting and storing values into its receive FIFO, for subsequent reading through the APB interface.

14.2.4 Slave Select Management

As an alternative to using the NSS pin to control the Slave Select signal, the application can choose to manage the Slave Select signal by software. This is configured by the SSM bit in the SSP_CR1 register (see *Figure 102*)

In software management, the external NSS pin is free for other application uses and the NSS_internal signal level is driven by writing to the SSI bit in the SSP_CR1 register.

There are two cases depending on the data/clock timing relationship (see *Figure 102*):

- If CPHA=1 (data latched on 2nd clock edge):

NSS_internal must be held low during the entire transmission. This implies that in single slave applications the NSS pin either can be tied to V_{SS}, or made free for standard I/O by managing the NSS function by software (SSM= 1 and SSI=0 in the in the SSP_CR1 register)

- If CPHA=0 (data latched on 1st clock edge):

NSS internal must be held low during byte transmission and pulled high between each byte to allow the slave to write to the FIFO register.

Figure 102. Generic NSS timing diagram

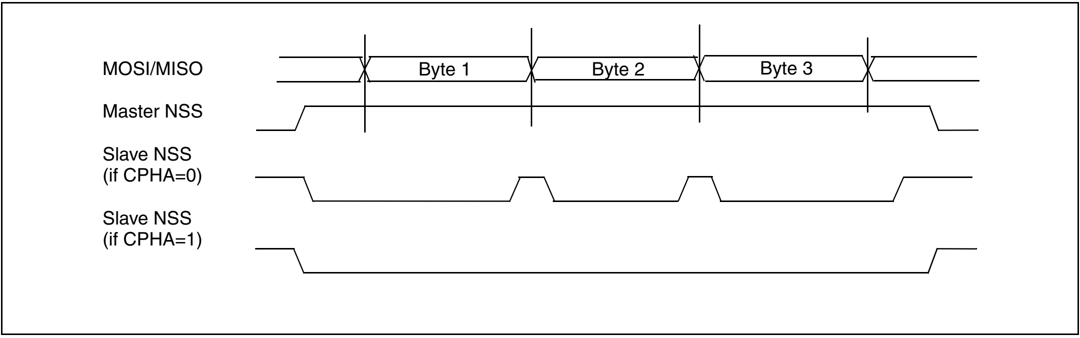
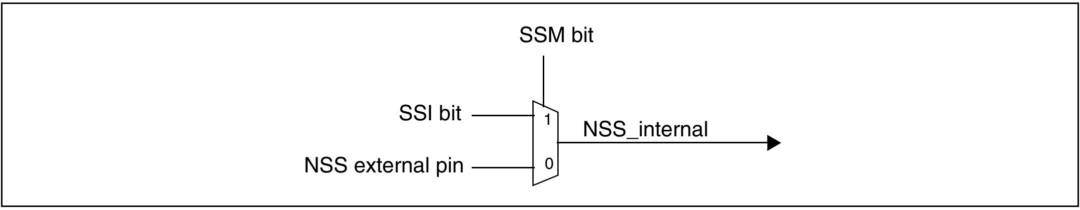


Figure 103. Hardware/Software Slave Select Management



Note: In master mode, NSS internal must be held high continuously

14.2.5 SSP operation

Following reset, the SSP logic is disabled and must be configured when in this state.

Control registers SSP_CR0 and SSP_CR1 need to be programmed to configure the peripheral as a master or slave operating under one of the following protocols:

- Motorola SPI
- Texas Instruments SSI

Enabling SSP operation

You can either prime the transmit FIFO, by writing up to eight 16-bit values when the SSP is disabled, or allow the transmit FIFO service request to interrupt the CPU. Once enabled, transmission or reception of data begins on the transmit and receive pins.

Programming the SSP_CR0 Control Register

The SSP_CR0 register is used to:

- Program the serial clock rate
- Select one of the two protocols
- Select the data word size (where applicable)
- Operating protocol

The Serial Clock Rate (SCR) value, in conjunction with the SSP_PR register clock prescale divisor value (CPSDVSR), is used to derive the SSP transmit and receive bit rate from the external SCK.

The frame format is programmed through the FRF bits and the data word size through the DSS bits.

Bit phase and polarity, applicable to Motorola SPI format only, are programmed through the CPHA and CPOL bits.

Programming the SSP_CR1 Control Register

The SSP_CR1 register is used to:

- Select master or slave mode
- Enable a loop back test feature
- Enable the SSP peripheral

To configure the SSP as a master, clear the SSP_CR1 register master or slave selection bit (MS) to 0, which is the default value on reset.

Setting the SSP_CR1 register MS bit to 1 configures the SSP as a slave. When configured as a slave, enabling or disabling of the MISO output signal is provided through the SSP_CR1 slave mode MISO output disable bit (SOD). This can be used in some multi-slave environments where masters might parallel broadcast.

To enable the operation of the SSP set the Synchronous Serial Port Enable (SSE) bit to 1.

Clock ratios

There is a constraint on the ratio of the frequencies of PCLK to SCK. To ensure correct device operation in slave mode, PCLK must be at least 12 times faster than the maximum expected frequency of SCK.

The frequency selected for PCLK must accommodate the desired range of bit clock rates. The ratio of minimum PCLK frequency to SCK maximum frequency in the case of the slave mode is 12 and for the master mode it is two.

To generate a maximum bit rate of 16 Mbps in Master mode, the frequency of PCLK must be at least 32 MHz. With a PCLK frequency of 32 MHz, the SSP_PR register has to be programmed with a value of two and the SCR[7:0] field in the SSP_CR0 register needs to be programmed as zero.

To work with a maximum bit rate of 2.66 Mbps in slave mode, the frequency of PCLK must be at least 32 MHz. With an PCLK frequency of 32 MHz, the SSP_PR register can be programmed with a value of 12 and the SCR[7:0] field in the SSP_CR0 register can be programmed as zero. Similarly the ratio of PCLK maximum frequency to SCK minimum frequency is 254 x 256.

The minimum frequency of PCLK is governed by the following equations, both of which have to be satisfied:

$f_{PCLK(min)}$ => 2 x $f_{SCK(max)}$ [for master mode]

$f_{PCLK(min)}$ => 12 x $f_{SCK(max)}$ [for slave mode].

The maximum frequency of PCLK is governed by the following equations, both of which have to be satisfied:

$f_{PCLK(max)}$ <= 254 x 256 x $f_{SCK(min)}$ [for master mode]

$f_{PCLK(max)}$ <= 254 x 256 x $f_{SCK(min)}$ [for slave mode]

Bit rate generation

When configured as a master, an internal prescaler, comprising two free-running reloadable serially linked counters, is used to provide the serial output clock SCK.

The serial bit rate is derived by dividing down the input clock PCLK. The clock is first divided by an even prescale value CPSDVSR from 2 to 254 in steps of two, which is programmed in SSP_PR register. The least significant bit of the programmed number is hard coded to zero. If an odd number is written to this register, data read back from this register has the least significant bit as zero.

The output of the first prescaler is further divided by a factor of 1 to 256, which is 1 + SCR, where SCR is the value programmed in SSP_CR0 register, to give the final master output clock SCK.

$$\mathrm{SCK} = \frac{\mathrm{PCLK}}{\mathrm{CPSDVSR} \times (1 + \mathrm{SCR})}$$

The frequency of the output signal bit clock SCK is defined below:

For example, if PCLK is 32 MHz, and CPSDVSR = 2, then SCK has a frequency range from 62.5kHz to 16MHz.

Frame format

Each data frame is between 4 and 16 bits long depending on the size of data programmed, and is transmitted starting with the MSB. There are three basic frame types that can be selected:

- Texas Instruments synchronous serial
- Motorola SPI

For both formats, the serial clock (SCK) is held inactive while the SSP is idle, and transitions at the programmed frequency only during active transmission or reception of data. The idle state of SCK is utilized to provide a receive timeout indication that occurs when the receive FIFO still contains data after a timeout.

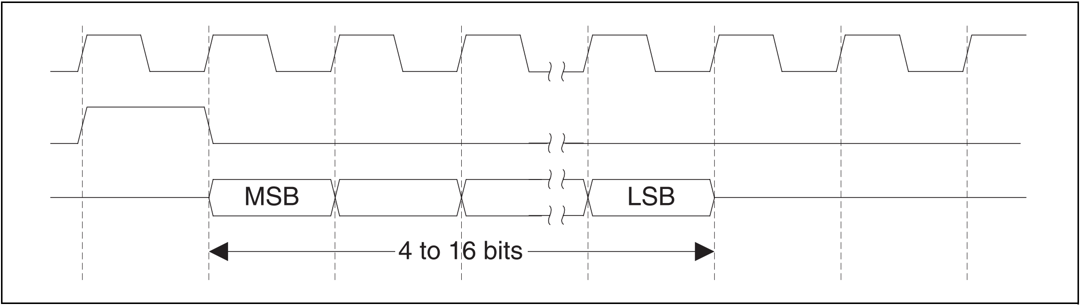
For Motorola SPI, the slave select (NSS) pin is active LOW, and is asserted (pulled down) during the entire transmission of the frame.

For Texas Instruments synchronous serial frame format, the NSS pin is pulsed for one serial clock period starting at its rising edge, prior to the transmission of each frame. For this frame format, both the SSP and the off-chip slave device drive their output data on the rising edge of SCK, and latch data from the other device on the falling edge.

Texas Instruments synchronous serial frame format

(see *Figure 104*) shows the Texas Instruments synchronous serial frame format for a single transmitted frame.

Figure 104. TI synchronous serial frame format (single transfer)



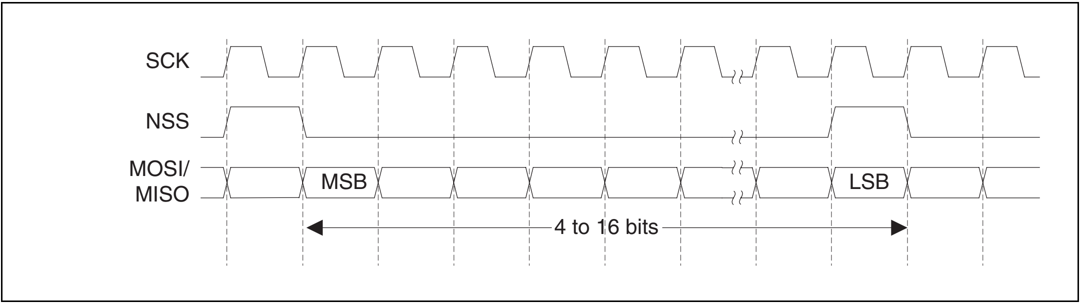
In this mode, SCK and NSS are forced LOW, and the transmit data line is tristated whenever the SSP is idle. Once the bottom entry of the transmit FIFO contains data, NSS is pulsed HIGH for one SCK period. The value to be transmitted is also transferred from the transmit FIFO to the serial shift register of the transmit logic. On the next rising edge of SCK, the MSB of the 4 to 16-bit data frame is shifted out on the Transmit (MOSI) pin. Likewise, the MSB of the received data is shifted onto the Receive (MISO) pin by the off-chip serial slave device.

Both the SSP and the off-chip serial slave device then clock each data bit into their serial shifter on the falling edge of each SCK. The received data is transferred from the serial shifter to the receive FIFO on the first rising edge of SCK after the LSB has been latched.

Note: When configured in TI slave mode , the internal SSPx SCLK clock rate must be configured (using SSPx_CR and SSPx_PR registers) to match the SCLK clock rate of the external master. Note that this does not apply in Motorola SPI slave mode where there is no need to configure the internal slave clock rate: consequently, in this mode, it is possible as a slave to connect to a master without knowing the SPI clock rate.

Figure 105 shows the Texas Instruments synchronous serial frame format when back-to-back frames are transmitted.

Figure 105. TI synchronous serial frame format (continuous transfer)



Motorola SPI frame format

The Motorola SPI interface is a four-wire interface where the NSS signal behaves as a slave select. The main feature of the Motorola SPI format is that the inactive state and phase of the SCK signal are programmable through the CPOL and CPHA bits in the SSP_CR0 control register.

- **CPOL, clock polarity**

  When the CPOL clock polarity control bit is LOW, it produces a steady state low value on the SCK pin. If the CPOL clock polarity control bit is HIGH, a steady state high value is placed on the SCK pin when data is not being transferred.

- **CPHA, clock phase**

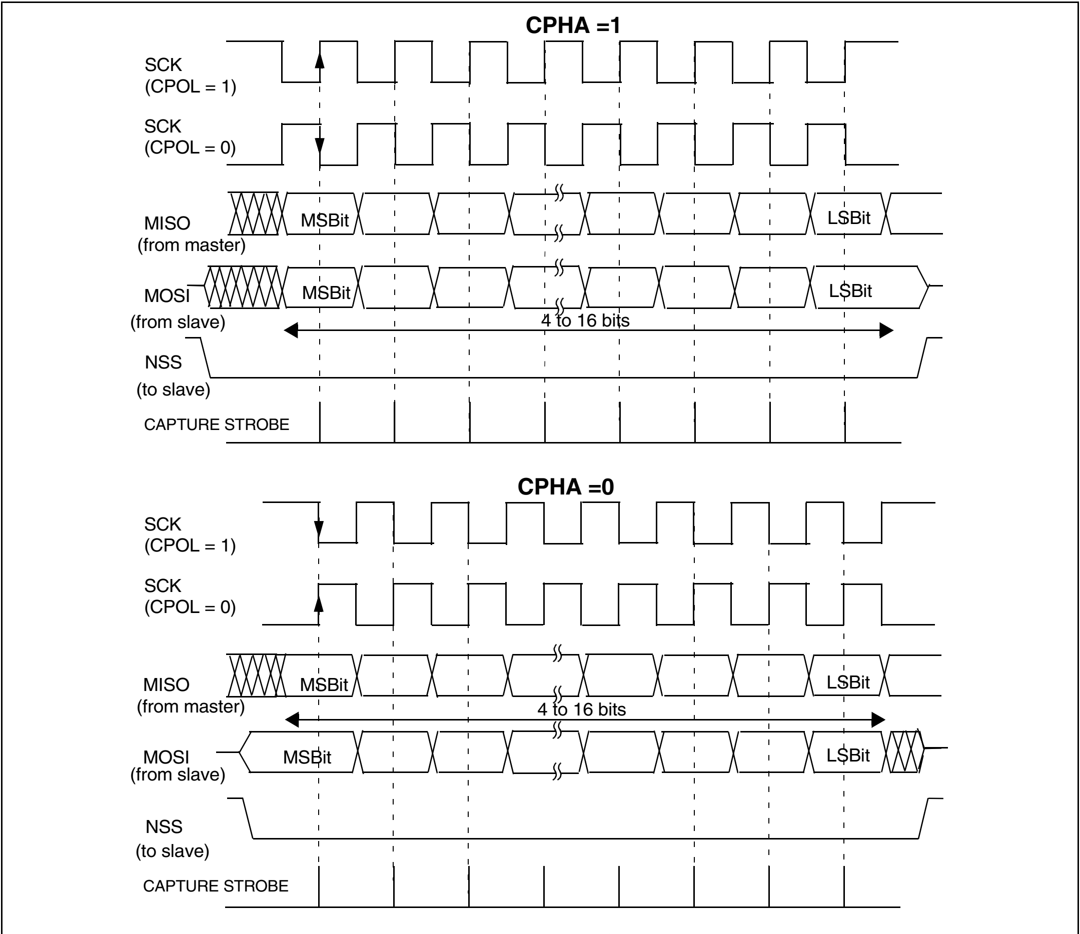
  The CPHA control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge.

  When the CPHA phase control bit is LOW, data is captured on the first clock edge transition. If the CPHA clock phase control bit is HIGH, data is captured on the second clock edge transition.

Note: The idle state of SCK must correspond to the polarity selected in the SSP_CR0 register (by pulling up SCK if CPOL=1 or pulling down SCK if CPOL=0).

Figure 106, 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.

Figure 106. Motorola SPI frame format



Note: This figure should not be used as a replacement for parametric information. Refer to the Electrical Characteristics chapter.

14.2.6 Transmit FIFO

The common transmit FIFO is a 16-bit wide, 8-locations deep, first-in, first-out memory buffer. When software writes data to the SSP Data Register (SSP_DR), it is stored in the buffer until read out by the transmit logic.

When configured as a master or a slave parallel data is written into the transmit FIFO prior to serial conversion and transmission to the attached slave or master respectively, through the transmit pin (MOSI or MISO).

14.2.7 Receive FIFO

The common receive FIFO is a 16-bit wide, 8-locations deep, first-in, first-out memory buffer. Received data from the serial interface are stored in the buffer until software reads the data from the SSP Data Register (SSP_DR).

When configured as a master or slave, serial data received through the receive pin (MOSI or MISO) is registered prior to parallel loading into the attached slave or master receive FIFO respectively.

14.2.8 Interrupt Control

Four individually maskable interrupt events are generated:

- A TX event requests servicing of the transmit buffer:
  - The transmit interrupt is asserted when the number of valid entries in the transmit FIFO is less than or equal to four (when there is space for four or more entries).
  - The transmit interrupt can be enabled so that data can be written to the transmit FIFO by an interrupt service routine.
- An RX event requests servicing of the receive buffer:
  - The receive interrupt is asserted when there are four or more valid entries in the receive FIFO
- An ROR event indicates an overrun condition in the receive FIFO
  - The receive overrun interrupt is asserted when the FIFO is already full and an additional data frame is received, causing an overrun of the FIFO. Data is overwritten in the shift register, but not in the FIFO. The overrun interrupt flag can be cleared by writing to the RORIC bit in the SSP_ICR register if the corresponding mask bit RORIM in the SSP_IMSR register is set.
- An RT event indicates that a timeout expired while data was present in the receive FIFO.
  - The receive timeout interrupt is asserted when the receive FIFO is not empty and the SSP has remained idle for a fixed 32 baud rate period. This ensures that the user is aware that data is still present in the receive FIFO and requires servicing. This interrupt is deasserted if the receive FIFO becomes empty after one or more read accesses, or if new data is received. The timeout interrupt flag can be cleared by writing to the RTIC bit in the SSP_ICR register if the corresponding mask bit RTIM in the SSP_IMSCR register is set.

14.2.9 SSP0 DMA Interface

The SSP0 provides an interface for connecting to the DMA controller. The DMA operation of the SSP0 is controlled through the SSP DMA control register SSP_DMACR. The DMA interface includes the following requests:

Receive Requests

- SSP0 Receive DMA Single request
  - Single character DMA transfer request, asserted by the SSP0. This request is asserted when the receive FIFO contains at least one character.
- SSP0 Receive DMA Burst request
  - Burst DMA transfer request, asserted by the SSP0. This request is asserted when the receive FIFO contains four or more characters.

Transmit Requests

- SSP0 Transmit DMA Single request
  - Single character DMA transfer request, asserted by the SSP0. This request is asserted when there is at least one empty location in the transmit FIFO.
- SSP0 Transmit DMA Burst request:
  - Burst DMA transfer request, asserted by the SSP0. This request is asserted when the transmit FIFO contains four or less characters.

14.3 Register description

The registers can only accessed as 32-bit data. A byte or half-word cannot be read or written.

14.3.1 Control Register 0 (SSP_CR0)

Address Offset: 00h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCR[7:0]								CPHA	CPOL	FRF[1:0]		DSS[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, always read as 0.
Bits 15:8	**SCR[7:0]**: *Serial clock rate* This value is used to configure the transmit and receive bit rate of SCK. The bit rate is $f_{PCLK}/(CPSDVR*(1+SCR))$, where CPSDVR is an even value from 2 to 254, programmed through the SSP_PR register and SCR is a value from 0 to 255.
Bit 7	**CPHA**: *Serial clock phase* This bit is used to select the serial output clock phase (applicable only to Motorola SPI format). 0: Data is captured on first clock edge 1: Data is captured on second clock edge
Bit 6	**CPOL**: *Serial clock polarity* This bit is used to select the SCK clock polarity (applicable only to Motorola SPI format). 0: SCK is held low when no data is being transferred 1: SCK is held high when no data is being transferred
Bits 5:4	**FRF[1:0]**: *Frame format* 00: Motorola SPI frame format 01: TI synchronous serial frame format 1x: Reserved
Bits 3:0	**DSS[3:0]**: *Data Size Select* 000x: Reserved 0010: Reserved 0011: 4-bit data 0100: 5-bit data 0101: 6-bit data 0110: 7-bit data 0111: 8-bit data 1000: 9-bit data 1001: 10-bit data 1010: 11-bit data 1011: 12-bit data 1100: 13-bit data 1101: 14-bit data 1110: 15-bit data 1111: 16-bit data.

14.3.2 Control Register 1 (SSP_CR1)

Address Offset: 04h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved										SSI	SSM	SOD	MS	SSE	LBM
										rw	rw	rw	rw	rw	rw

Bits 31:6	Reserved, must be written with 0.
Bit 5	**SSI**: *$\overline{SS}$ Internal Mode.* This bit is set and cleared by software. It acts as a 'chip select' by controlling the level of the $\overline{SS}$ slave select signal when the SSM bit is set. 0: Slave selected 1: Slave deselected
Bit 4	**SSM**: *$\overline{SS}$ Management.* 0: Hardware management ($\overline{SS}$ managed by external pin) 1: Software management (internal $\overline{SS}$ signal controlled by SSI bit. External $\overline{SS}$ pin free for general-purpose I/O)
Bit 3	**SOD**: *Slave-mode output disable.* This bit is relevant only in the slave mode (MS=1). In multiple-slave systems, it is possible for an SSP master to broadcast a message to all slaves in the system while ensuring that only one slave drives data onto its serial output line. In such systems the data input lines from multiple slaves could be tied together. To operate in such systems, the SOD bit can be set if the SSP slave is not supposed to drive the serial data output line. 0: SSP can drive the serial data output in slave mode. 1: SSP can not drive the serial data output in slave mode.
Bit 2	**MS**: *Master or slave mode select.* This bit can be modified only when the SSP is disabled (SSE=0). 0: Device configured as master (default) 1: Device configured as slave
Bit 1	**SSE**: *SSP enable.* 0: SSP operation disabled 1: SSP operation enabled
Bit 0	**LBM**: *Loop back mode.* 0 = Normal serial port operation enabled 1 = Output of transmit serial shifter is connected to input of receive serial shifter internally

14.3.3 Data Register (SSP_DR)

Address Offset: 08h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, must be written with 0.
Bits 15:0	**DATA[15:0]:** *Transmit/Receive FIFO* When SSP_DR is read, the entry in the receive FIFO (pointed to by the current FIFO read pointer) is accessed. As data values are removed by the SSP receive logic from the incoming data frame, they are placed into the entry in the receive FIFO pointed to by the current FIFO write pointer. When SSP_DR is written to, the entry in the transmit FIFO pointed to by the write pointer is written to. Data values are removed from the transmit FIFO one value at a time by the transmit logic. It is loaded into the transmit serial shifter, then serially shifted out onto the data output pin at the programmed bit rate. When a data size of less than 16 bits is selected, the user must right-justify data written to the transmit FIFO. The transmit logic ignores the unused bits. Received data less than 16 bits is automatically right-justified in the receive buffer. The transmit FIFO and the receive FIFO are not cleared even when SSE is set to zero. This allows the software to fill the transmit FIFO before enabling the SSP.

14.3.4 Status Register (SSP_SR)

Address Offset: 0Ch

Reset value: 0000 0003h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved											BSY	RFF	RNE	TNF	TFE
											r	r	r	r	r

Bits 31:5	Reserved, must be written with 0.
Bit 4	**BSY:** *SSP Busy* 0: SSP is idle 1: SSP is currently transmitting and/or receiving a frame or the transmit FIFO is not empty
Bit 3	**RFF:** *Receive FIFO full* 0: Receive FIFO is not full 1: Receive FIFO is full
Bit 2	**RNE:** *Receive FIFO not empty* 0: Receive FIFO is empty 1: Receive FIFO is not empty
Bit 1	**TNF:** *Transmit FIFO not full* 0: Transmit FIFO is full 1: Transmit FIFO is not full
Bit 0	**TFE:** *Transmit FIFO empty* 0: Transmit FIFO is not empty 1: Transmit FIFO is empty

14.3.5 Clock Prescaler Register (SSP_PR)

Address Offset: 10h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								CPSDVSR[7:0]							
-								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:8	Reserved, must be written with 0.
Bits 7:0	**CPSDVSR[7:0]:** *Clock prescaler divider* These bits specify the division factor by which the input PCLK must be divided for use by the SSP. The value written to this register must be an even number between 2 and 254. The least significant bit of the programmed number is hard-coded to zero. If an odd number is written to this register, data read back from this register has the least significant bit as zero.

14.3.6 Interrupt Mask Set and Clear Register (SSP_IMSCR)

Address Offset: 14h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												TXIM	RXIM	RTIM	RORIM
-												rw	rw	rw	rw

Bits 31:4	Reserved, must written with 0.
Bit 3	**TXIM:** *Transmit FIFO interrupt mask* 0: Tx FIFO half empty or less condition interrupt is masked 1: Tx FIFO half empty or less condition interrupt is not masked
Bit 2	**RXIM:** *Receive FIFO interrupt mask* 0: Rx FIFO half full or more condition interrupt is masked 1: Rx FIFO half full or more condition interrupt is not masked
Bit 1	**RTIM:** *Receive timeout interrupt mask* 0: RxFIFO not empty and no read prior to timeout interrupt is masked. 1: RxFIFO not empty and no read prior to timeout interrupt is not masked
Bit 0	**RORIM:** *Receive overrun interrupt mask* 0: RxFIFO written to while full condition interrupt is masked 1: RxFIFO written to while full condition interrupt is not masked

14.3.7 Raw Interrupt Status Register (SSP_RISR)

Address Offset: 18h

Reset value: 0000 0008h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												TXRIS	RXRIS	RTRIS	RORRIS
-												r	r	r	r

Bits 31:4	Reserved, must be written with 0.
Bit 3	**TXRIS**: *Transmit FIFO raw status flag* 0: No TX FIFO event occurred 1: A TX FIFO event occurred (prior to masking)
Bit 2	**RXRIS**: *Receive FIFO raw status flag* 0: No RX FIFO event occurred 1: A RX FIFO event occurred (prior to masking)
Bit 1	**RTRIS**: *Receive timeout raw status flag* 0: No RX Timeout event occurred 1: A RX Timeout event occurred (prior to masking) **Note**: The timeout period is equal to a 32 baud rate period
Bit 0	**RORRIS**: *Receive overrun raw status flag* 0: No RX Overrun event occurred 1: A RX Overrun event occurred (prior to masking)

14.3.8 Masked Interrupt Status Register (SSP_MISR)

Address Offset: 1Ch
Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												TXMIS	RXMIS	RTMIS	RORMIS
												r	r	r	r

Bits 31:4	Reserved, must be written with 0.
Bit 3	**TXMIS**: *Transmit FIFO masked status flag* 0: No TX FIFO interrupt request (after masking) 1: TX FIFO interrupt request occurred (after masking)
Bit 2	**RXMIS**: *Receive FIFO masked status flag* 0: No RX FIFO interrupt request (after masking) 1: An RX FIFO interrupt request occurred (after masking)
Bit 1	**RTMIS**: *Receive timeout masked status flag* 0: No RX Timeout interrupt request (after masking) 1: RX Timeout interrupt request occurred (after masking) **Note**: The timeout period is equal to a 32-bit period
Bit 0	**RORMIS**: *Receive overrun masked status flag* 0: No RX Overrun interrupt request (after masking) 1: RX Overrun interrupt request occurred (after masking)

14.3.9 Interrupt Clear Register (SSP_ICR)

Address Offset: 20h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														RTIC	RORIC
-														w	w

Bits 31:2	Reserved, must be written with 0.
Bit 1	**RTIC**: *Clear RX Timeout interrupt* 0: No effect 1: Clear RX Timeout interrupt
Bit 0	**RORIC**: *Clear RX Overrun interrupt* 0: No effect 1: Clear RX Overrun interrupt

14.3.10 DMA Control Register (SSP_DMACR)

Address Offset: 24h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												TXDM ASD	RXDM ASD	TXDM AE	RXDM AE
-												rw	rw	rw	rw

Bits 31:4	Reserved, must be written with 0.
Bit 3	**TXDMASD**: *Transmit DMA Single request disable* 0: TXDMA single request enabled. This signal is asserted when there is at least one empty location in the transmit FIFO 1: TXDMA single request disabled. The Burst request is enabled. This signal is asserted when the transmit FIFO contains four or less characters.
Bit 2	**RXDMASD**: *Receive DMA Single request disable* 0: RXDMA single request enabled. This signal is asserted when the receive FIFO contains at least one character. 1: RXDMA single request disabled. The Burst request is enabled. This signal is asserted when the receive FIFO contains four or more characters.
Bit 1	**TXDMAE**: *Transmit DMA enable* 0: DMA for the transmit FIFO disabled 1: DMA for the transmit FIFO enabled
Bit 0	**RXDMAE**: *Receive DMA enable* 0: DMA for the receive FIFO disabled 1: DMA for the receive FIFO enabled

14.4 Register map

Table 64. Register map

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	SSP_CR0	reserved																SCR[7:0]								CPHA	CPOL	FRF[1:0]		DSS[3:0]			
4	SSP_CR1	reserved																										SSI	SSM	SOD	MS	SSE	LBM
8	SSP_DR	reserved																DATA[15:0]															
C	SSP_SR	reserved																											BSY	RFF	RNE	TNF	TFE
10	SSP_PR	reserved																								CPSDVSR[7:0]							
14	SSP_IMSCR	reserved																												TXIM	RXIM	RTIM	ROR IM
18	SSP_RISR	reserved																												TX RIS	RX RIS	RT RIS	ROR RIS
1C	SSP_MISR	reserved																												TX MIS	RX MIS	RT MIS	RORMIS
20	SSP_ICR	reserved																														RTIC	ROR IC
24	SSP_DMACR	reserved																												TX DMA SD	RX DMA SD	TX DMAE	RX DMAE
28 to 7C		reserved																															

See *Table 2* for the base addresses.

15 Universal Asynchronous Receiver Transmitter (UART)

15.1 Introduction

The UART interface provides serial communication between the STR750 and other microcontrollers, microprocessors or external peripherals.

The UART supports full-duplex asynchronous communication. Five to eight bit data transfer, parity generation, and the number of stop bits are programmable. Parity, framing, and overrun error detection are provided to increase the reliability of data transfers. Transmission and reception of data can simply be double-buffered, or 16-deep FIFOs may be used. Testing is supported by a loop-back option. A programmable baud rate generator provides the UART with a separate serial clock signal. The break length is programmable.

15.2 Main features

- Separate 16 x 8 transmit and 16 x 12 receive First-In First-Out memory buffers (FIFOs) to reduce CPU interrupts.
- Programmable FIFO disabling for 1-byte depth.
- Programmable baud rate generator. This enables division of the reference clock by (1 x 16) to (65535 x 16) and generates an internal x16 clock. The divider can be a fractional number enabling you to use a large range of clock frequencies.
- Standard asynchronous communication bits (start, stop and parity). These are added prior to transmission and removed on reception.
- LIN Master compatible with programmable break length from 10 to 20 in transmission and break detection in reception.
- Independent masking of transmit FIFO, receive FIFO, receive timeout, CTS status, and error condition interrupts.
- Support for Direct Memory Access (DMA)
- False start bit detection.
- Programmable hardware flow control CTS and RTS.
- Fully-programmable serial interface characteristics:
  - Data can be 5, 6, 7, or 8 bits
  - Even, odd, stick, or no-parity bit generation and detection
  - 1 or 2 stop bit generation
  - Baud rate up to PCLK_max_freq/16

15.3 Functional description

The UART supports full-duplex asynchronous communication, where both the transmitter and the receiver use the same data frame format and the same baud rate. Data is transmitted on the UART_TX pin and received on the UART_RX pin.

A character frame (see Figure 107 and Figure 108) consists of:

- Five to eight data bits D4:0, D5:0, D6:0 or D7:0 (by setting the WLEN bits in UART_LCR)
- An optional parity bit if enabled by the PEN bit in the UART_LCR register
- One or two stop bits depending on the STP2 bit in the UART_LCR register

Figure 107. 8-bit data frames

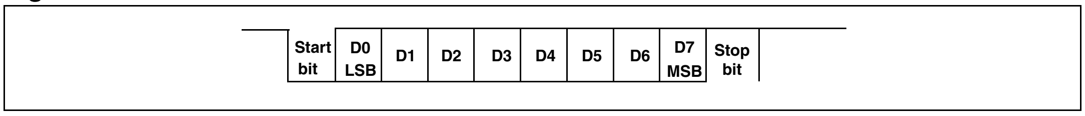
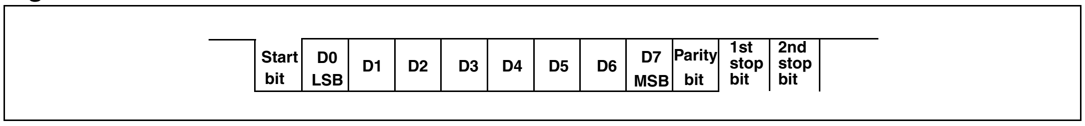


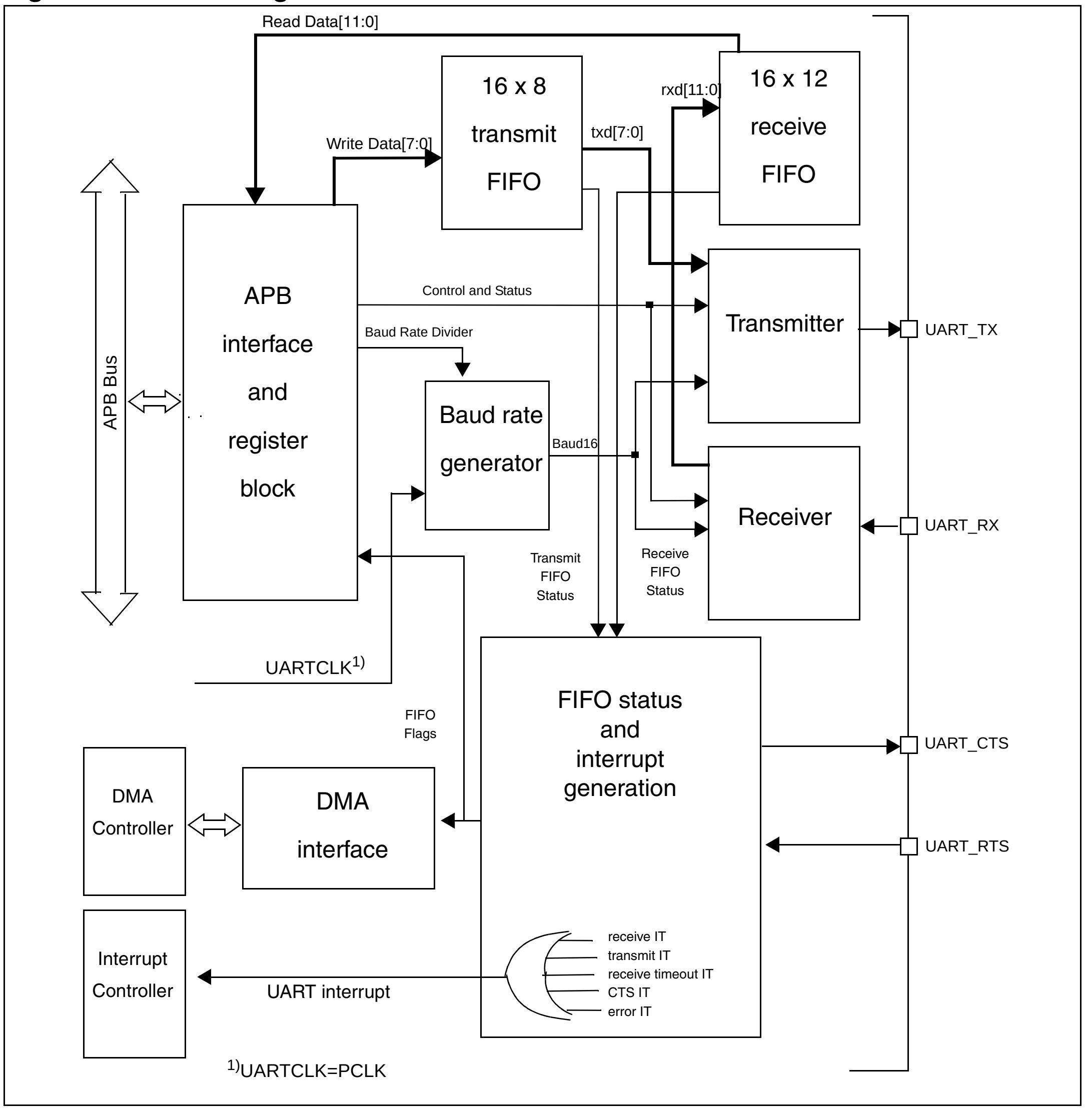
Figure 108. 8-bit data frames with PEN = 1 and STP2 = 1



15.3.1 Functional block diagram

Figure 109 shows the UART functional block diagram.

Figure 109. Block diagram



15.3.2 Fractional baud rate divider

The baud rate divider is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part. It is used by the baud rate generator to determine the bit period. The frequency selected for UARTCLK must accommodate the desired range of baud rates:

$f_{UARTCLK}$ (min) ≥ 16 x baud_rate (max)
$f_{UARTCLK}$(max) ≤ 16 x 65535 x baud_rate (min)

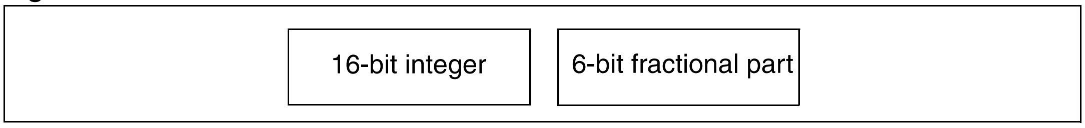
For example, for a range of baud rates from 110 baud to 460800 baud the UARTCLK frequency must be within the range 7.3728 MHz to 115 MHz. The frequency of UARTCLK must also be within the required error limits for all baud rates to be used.

The 16-bit integer is loaded through the UART_IBRD register. The 6-bit fractional part is loaded into the UART_FBRD register. The Baud Rate Divider has the following relationship to the UARTCLK frequency:

Baud Rate Divider BAUDDIV = Freq(UARTCLK**)** / (16 x Baud Rate) = BRD_I + BRD_F

where BRD_I is the integer part and BRD_F is the fractional part separated by a decimal point as shown in Figure 110

Figure 110. Baud rate divider



You can calculate the 6-bit number (m) by taking the fractional part of the required baud rate divider and multiplying it by 64 (that is, 2^n, where n is the width of the UART_FBRD register) and adding 0.5 to account for rounding errors:

m = integer(BRD_F * 2^n + 0.5)

An internal clock signal, Baud16, is generated, and is a stream of one UARTCLK wide pulses with an average frequency of 16 times the desired baud rate. This signal is then divided by 16 to give the transmit clock. A low number in the baud rate divider gives a short bit period, and a high number in the baud rate divider gives a long bit period.

Figure 111 is an example of how to calculate the divider value.

Figure 111. Calculating the divider value

If the required baud rate is 230400 and UARTCLK = 8 MHz then:

Baud Rate Divider = (8 * 10^6)/(16 * 230400) = 2.17

Therefore, BRD_I = 2 and BRD_F = 0.17,

Therefore, fractional part, m = integer((0.17 * 64) + 0.5) = 11

Generated baud rate divider = 2 + 11/64 = 2.171

Generated baud rate = (8 * 10^6)/(16 * 2.171) = 230309

Error = (230400 - 230309)/230400 * 100 = 0.039%

The maximum error using a 6-bit UART_FBRD register = 1/64 * 100 = 1.56%. This occurs when m = 1, and the error is cumulative over 64 clock ticks.

For UARTCLK = 16 MHz, the BRD_I = 4 and BRD_F = 0.34, m = 22, the generated baud rate is 230256 and Error is 0.062%.

Table 65 shows some typical bit rates and their corresponding dividers, given the UART clock frequency of 7.3728MHz. These values do not use the fractional divider so the value in the UART_FBRD register is zero.

Table 65. Typical baud rates and their integer dividers

Programmed integer divider	Bit rate (bps)
0x1	460800
0x2	230400
0x4	115200
0x6	76800
0x8	57600
0xC	38400
0x18	19200
0x20	14400
0x30	9600
0xC0	2400
0x180	1200
0x105D	110

Table 66 shows some required bit rates and their corresponding integer and fractional divider values and generated bit rates with a clock frequency of 8 MHz.

Table 66. Typical baud rates and their corresponding integer and fractional dividers

Programmed divider (integer)	Programmed divider (fraction)	Required bit rate (bps)	Generated bit rate (bps)	Error (%)
0x2	0xB	230400	230309	0.039
0x4	0x16	115200	115128	0.062
0x6	0x21	76800	76746	0.07
0xD	0x1	38400	38417	0.044
0x22	0x2E	14400	14402	0.013
0xD0	0x15	2400	2400	~0
0x11C1	0x1D	110	110	~0

Table 67 shows some required bit rates and their corresponding integer and fractional divider values and generated bit rates given a clock frequency of 16 MHz.

Table 67. Typical baud rates and their corresponding integer and fractional dividers

Programmed divider (integer)	Programmed divider (fraction)	Required bit rate (bps)	Generated bit rate (bps)	Error (%)
0x4	0x16	230400	230256	0.062
0x8	0x2C	115200	115115	0.073
0xD	0x1	76800	76834	0.044
0x1A	0x3	38400	38394	0.015
0x45	0x1C	14400	14402	0.013
0x1A0	0x2B	2400	2400	~0
0x2382	0x3A	110	110	~0

15.3.3 Data transmission or reception

Data received or transmitted is stored in two 16-byte FIFOs, though the receive FIFO has an extra four bits per character for status information.

For transmission, data is written into the transmit FIFO. If the UART is enabled, it causes a data frame to start transmitting with the parameters indicated in the UART_LCR register. Data continues to be transmitted until there is no data left in the transmit FIFO.

The BUSY bit in the UART_FR register goes HIGH as soon as data is written to the transmit FIFO (that is, the FIFO is not empty) and remains asserted HIGH while data is being transmitted. BUSY is negated only when the transmit FIFO is empty, and the last character has been transmitted from the shift register, including the stop bits. BUSY can be asserted HIGH even though the UART might no longer be enabled.

For each sample of data, three readings are taken and the majority value is kept. In the following paragraphs the middle sampling point is defined, and one sample is taken either side of it.

When the receiver is idle (the UART_RX pin continuously 1, in the marking state) and a LOW is detected on the data input (a start bit has been received), the receive counter, with the clock enabled by Baud16, begins running and data is sampled on the eighth cycle of that counter.

The start bit is valid if the UART_RX pin is still LOW on the eighth cycle of Baud16, otherwise a false start bit has been detected and is ignored.

If the start bit is valid, successive data bits are sampled on every 16th cycle of Baud16 (that is, one bit period later) according to the programmed length of the data characters. The parity bit is then checked if parity mode is enabled.

Lastly, a valid stop bit is confirmed if the UART_RX pin is HIGH (otherwise a framing error has occurred). When a full word is received, the data is stored in the receive FIFO, with any error bits associated with that word (see Table 68).

Error bits

Three error bits are stored in bits [10:8] of the receive FIFO, and are associated with a particular character. There is an additional error that indicates an overrun error and this is stored in bit 11 of the receive FIFO.

Overrun bit

The overrun bit is not associated with the character in the receive FIFO. The overrun error is set when the FIFO is full, and the next character is completely received in the shift register. The data in the shift register is overwritten, but it is not written into the FIFO. When an empty location is available in the receive FIFO, and another character is received, the state of the overrun bit is copied into the receive FIFO along with the received character. The overrun state is then cleared. Table 68 shows the bit functions of the receive FIFO.

Table 68. Receive FIFO bit functions

FIFO bit	Function
11	Overrun indicator
10	Break error
9	Parity error
8	Framing error
7:0	Received data

Disabling the FIFOs

Additionally, you can disable the FIFOs. In this case, the transmit and receive sides of the UART have 1-byte holding registers (the bottom entry of the FIFOs). The overrun bit is set when a word has been received and the previous one has not yet been read.

In this implementation, the FIFOs are not physically disabled, but the flags are manipulated to give the illusion of a 1-byte register.

System and diagnostic loop-back testing

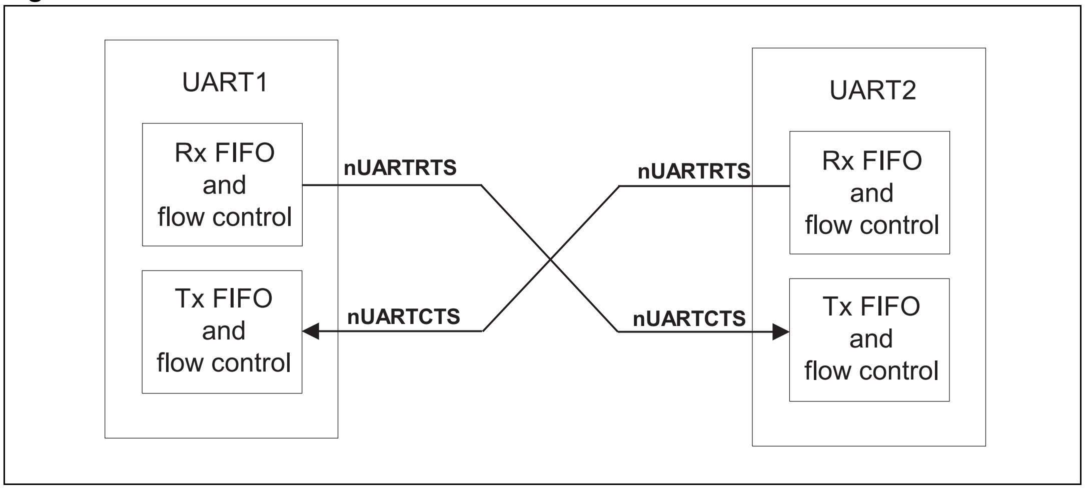
You can perform loop-back testing for UART data by setting the *Loop Back Enable* (LBE) bit to 1 in the control register UART_CR (bit 7).

Data transmitted on the UART_TX pin is received on the UART_RX input.

15.3.4 UART hardware flow control

The hardware flow control feature is fully selectable and enables you to control the serial data flow by using the UART_RTS output and UART_CTS input signals. Figure 112 shows how two devices can communicate with each other using hardware flow control.

Figure 112. Hardware flow control between two similar devices



When the RTS flow control is enabled, the UART_RTS signal is asserted until the receive FIFO is filled up to the programmed watermark level. When the CTS flow control is enabled, the transmitter can only transmit data when the UART_CTS signal is asserted.

The hardware flow control is selectable through bits 14 (RTSEn) and 15 (CTSEn) in the UART_CR register (UART_CR). Table 69 shows how you must set the bits to enable RTS and CTS flow control both simultaneously and independently.

Table 69. Control bits to enable and disable hardware flow control

CTSEn bit 15 in UART_CR	RTSEn bit 14 in UART_CR	Description
1	1	Both RTS and CTS flow control enabled
1	0	Only CTS flow control enabled
0	1	Only RTS flow control enabled
0	0	Both RTS and CTS flow control disabled

When RTS flow control is enabled, the software cannot control the UART_RTS pin through bit 11 of the UART_CR register.

RTS flow control

The RTS flow control logic is linked to the programmable receive FIFO watermark levels. When RTS flow control is enabled, the UART_RTS signal is asserted until the receive FIFO is filled up to the watermark level. When the receive FIFO watermark level is reached, the UART_RTS signal is de-asserted, indicating that there is no more room to receive any more data. The transmission of data is expected to cease after the current character has been transmitted.

The UART_RTS signal is re-asserted when data has been read out of the receive FIFO so that it is filled to less than the watermark level. If RTS flow control is disabled and the UART is still enabled, then data is received until the receive FIFO is full, otherwise no more data is transmitted to it.

CTS flow control

If CTS flow control is enabled, then the transmitter checks the UART_CTS signal before transmitting the next byte. If the UART_CTS signal is asserted, it transmits the byte, otherwise transmission does not occur.

15.3.5 Interrupts

There are eight maskable interrupts generated within the UART. These are combined to one interrupt output, which is the OR of the individual interrupts.

You can enable or disable the individual interrupts by changing the mask bits in the UART_IMSC register. Setting the appropriate mask bit to HIGH enables the interrupt.

The status of the individual interrupt sources can be read from either UART_RIS, for raw interrupt status, or from UART_MIS, for the masked interrupt status.

The individual interrupt is cleared by setting the corresponding bit of UART_ICR.

Table 70. Status of individual interrupt sources

Interrupt event	Event flag UART_RIS	Enable control bit UART_IMSC	Masked interrupt status UART_MIS	Interrupt clear UART_ICR
Receive Interrupt	RXRIS	RXIM	RXMIS	RXIC
Transmit Interrupt	TXRIS	TXIM	TXMIS	TXIC
Receive Timeout Interrupt	RTRIS	RTXIM	RTXMIS	RTXIC
Framing Interrupt	FERIS	FEIM	FEMIS	FEIC
Parity Error Interrupt	PERIS	PEIM	PEMIS	PEIC
Break Error Interrupt	BERIS	BEIM	BEMIS	BEIC
Overrun Error Interrupt	OERIS	OEIM	OEMIS	OEIC
CTS Interrupt	CTSRIS	CTSIM	CTSMIS	CTSIC

15.3.6 UART0 DMA Interface

The UART0 provides an interface for connecting to the DMA controller. The DMA operation of the UART0 is controlled through the UART DMA control register, UART_DMACR. The DMA interface includes the following requests:

For receive

- UART0 Receive DMA Single request

Single character DMA transfer request, asserted by the UART0. For receive, one character consists of up to 12 bits. This request is asserted when the receive FIFO contains at least one character.

- UART0 Receive DMA Burst request

Burst DMA transfer request, asserted by the UART0. This request is asserted when the receive FIFO contains more characters than the programmed watermark level. You can program the watermark level for each FIFO through the UART_IFLS register.

For transmit

- UART0 Transmit DMA Single request

Single character DMA transfer request, asserted by the UART0. For transmit, one character consists of up to eight bits. This request is asserted when there is at least one empty location in the transmit FIFO.

- UART0 Transmit DMA Burst request

Burst DMA transfer request, asserted by the UART0. This request is asserted when the transmit FIFO contains less characters than the watermark level. You can program the watermark level for each FIFO through the UART_IFLS register.

When the UART is in FIFO-disabled mode, only DMA single transfer mode can be used, since only one character can be transferred to, or from the FIFOs at a time.

UART0 Receive DMA Single requests and UART0 Transmit DMA Single requests are the only requests that can be asserted. When the UART is in FIFO-enabled mode, data transfers can be made by either single or burst transfers depending on the programmed watermark level and the amount of data in the FIFO. *Table 71* shows the trigger points for DMA Burst requests depending on the watermark level, for both the transmit and receive FIFOs.

Table 71. Trigger points for DMA burst requests

Watermark level	Burst length	
	Transmit (number of empty locations)	Receive (number of filled locations
1/4	-	4
1/2	8	8
3/4	4	-

In addition to the above, the DMAONERR bit in the DMA control register supports the use of the receive error interrupt. It enables the DMA receive request outputs, UART0 Receive DMA Single request or UART0 Receive DMA Burst request, to be masked out when the UART0 error interrupt is asserted.

The DMA receive request outputs remain inactive until the UART0 error interrupt is cleared.

The DMA transmit request outputs are unaffected.

15.4 Register description

In this section, the following abbreviations are used:

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 these bits.
read/clear (rc)	Software can read as well as clear this bit by writing any value.
read/set (rs)	Software can read as well as set this bit. Writing ‘0’ has no effect on the bit value.

15.4.1 Data Register (UART_DR)

Address Offset: 00h

Reset value: ----h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				OE	BE	PE	FE	DATA							
-	-	-	-	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:12	Reserved, forced by hardware to 0.
Bit 11	**OE:** *Overrun Error* 0: No Overrun Error. 1: Overrun Error: data is received while the receive FIFO is already full.
Bit 10	**BE:** *Break Error* 0: No Break Error. 1: Break Error: the received data input was held LOW for longer than a full-word transmission time.
Bit 9	**PE:** *Parity Error* 0: No Parity Error. 1: Parity Error: the parity of the received data character did not match the parity selected as defined by bits 2 and 7 of the UART_LCR register.
Bit 8	**FE:** *Framing Error* 0: No Framing Error. 1: Framing Error: the received character did not have a valid stop bit.
Bits 7:0	**DATA:** Receive (read) data character. Transmit (write) data character.

Note: *For words to be transmitted:*

- if the FIFOs are enabled, data written to this location is pushed onto the transmit FIFO.

- if the FIFOs are not enabled, data is stored in the transmitting holding register (the bottom word of the transmit FIFO).

The write operation initiates transmission from the UART. The data is prefixed with a start bit, appended with the appropriate parity bit (if parity is enabled), and a stop bit. The resultant word is then transmitted.

For received words:

- if the FIFOs are enabled, the data byte and the 4-bit status (break, frame, parity, and overrun) are *pushed onto the 12-bit-wide receive FIFO.*

- if the FIFOs are not enabled, the data byte and status are stored in the receiving holding register (the bottom word of the receive FIFO).

15.4.2 Receive Status Register (UART_RSR)

Address Offset: 04h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												OE	BF	PE	FE
-	-	-	-	-	-	-	-	-	-	-	-	rc	rc	rc	rc

Bits 15:4	Reserved, forced by hardware to 0.
Bit 3	**OE:** *Overrun Error* 0: No Overrun Error. 1: Overrun Error: data is received while the receive FIFO is already full. This bit is cleared to 0 by a write to UART_RSR. **Note:** The FIFO contents remain valid since no further data is written when the FIFO is full, only the contents of the shift register are overwritten. The CPU must now read the data in order to empty the FIFO.
Bit 2	**BF:** *Break Flag* 0: No Break Error. 1: Break Error: a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity and stop bits). This bit is cleared to 0 by a write to UART_RSR. **Note:** In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state) and the next valid start bit is received.
Bit 1	**PE:** *Parity Error* 0: No Parity Error. 1: Parity Error: the parity of the received data character does not match the parity selected as defined by bits 2 and 7 of the UART_LCR register. This bit is cleared to 0 by a write to UART_RSR. **Note:** In FIFO mode, this error is associated with the character at the top of the FIFO.
Bit 0	**FE:** *Framing Error* 0: No Framing Error. 1: Framing Error: the received character did not have a valid stop bit (a valid stop bit being 1). This bit is cleared to 0 by a write to UART_RSR. **Note:** In FIFO mode, this error is associated with the character at the top of the FIFO.

Note: The received data character must first be read from the UART_DR register before reading the error status associated with that data character from the UART_RSR register. This read sequence cannot be reversed, because the UART_RSR register is updated only when a read occurs from the UART_DR register. However, the status information can also be obtained by reading the UART_DR register. The status information for overrun is set immediately when an overrun condition occurs.

15.4.3 Flag Register (UART_FR)

Address Offset: 18h

Reset value: 009-h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXFE	RXFF	TXFF	RXFE	BUSY	Reserved		CTS
-	-	-	-	-	-	-	-	r	r	r	r	r	-	-	r

Bits 15:8	Reserved, forced by hardware to 0.
Bit 7	**TXFE:** *Transmit FIFO Empty* The meaning of this bit depends on the state of the FEN bit in the UART_LCR register. If the FIFO is disabled, this bit is set when the transmit holding register is empty. If the FIFO is enabled, the TXFE bit is set when the transmit FIFO is empty. **Note:** If a break request occurs during the transmission after which TXFE should be set, it will be set only after the break transmission.
Bit 6	**RXFF:** *Receive FIFO Full* The meaning of this bit depends on the state of the FEN bit in the UART_LCR register. If the FIFO is disabled, this bit is set when the receive holding register is full. If the FIFO is enabled, the RXFF bit is set when the receive FIFO is full.
Bit 5	**TXFF:** *Transmit FIFO Full* The meaning of this bit depends on the state of the FEN bit in the UART_LCR register. If the FIFO is disabled, this bit is set when the transmit holding register is full. If the FIFO is enabled, the TXFF bit is set when the transmit FIFO is full.
Bit 4	**RXFE**: *Receive FIFO Empty* The meaning of this bit depends on the state of the FEN bit in the UART_LCR register. If the FIFO is disabled, this bit is set when the receive holding register is empty. If the FIFO is enabled, the RXFE bit is set when the receive FIFO is empty.
Bit 3	**BUSY:** *UART Busy* If this bit is set to 1, the UART is busy transmitting data. This bit remains set until the complete byte, including all the stop bits, has been sent from the shift register. This bit is set as soon as the transmit FIFO becomes non-empty (regardless of whether the UART is enabled or not). This bit is set while the break frame is being transmitted (including the break stop bit). **Note:** If the transmit FIFO is empty, the *busy* bit gets reset after data transfer and before *break* transfer, which corresponds to an IDLE state.
Bits 2:1	Reserved, forced by hardware to 0.
Bit 0	**CTS:** *Clear To Send* This bit is the complement of the UART clear to send (UART_CTS) status input, that is, the bit is 1 when this status input is 0.

15.4.4 Break Register (UART_BKR)

Address Offset: 1Ch

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															BRK
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	rs

Bits 15:1	Reserved, forced by hardware to 0.
Bit 0	**BRK:** *Send Break* If this bit is set to ‘1’, a break character is sent on the UART_TX output, after completing transmission of the current character. This bit is reset by hardware when the break is completed. The break length (number of low bits) depends on the LINEN/LBKLEN bits and word configuration. The low bits are followed by a high level, the duration of which is at least one bit. **Note:** This bit cannot be written when the STP2 is set. (The *break* feature is not available in “2 stop bits” mode). The break is sent before or after the data depending on whether: - transmission is on-going (in which case the break character will be sent at the end of the current transmission). - transmission has not yet started (in which case the break will be inserted first).

15.4.5 Integer Baud Rate Register (UART_IBRD)

Address Offset: 24h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BAUD DIVINT															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**BAUD DIVINT:** *Integer Baud Rate Divider* **Notes:** In order to internally update the contents of the UART_IBRD register, a write to the UART_LCR register must always be performed at the end. The baud rate must not be changed: - When the UART is enabled. - When completing a transmission or reception when it has been programmed to become disabled. The minimum possible divide ratio is 1 and the maximum is 65535(2^{16} - 1). When this is the case, UART_IBRD = 0 is invalid and UART_FBRD is ignored. Similarly, when UART_IBRD = 65535 (that is 0xFFFF), then UART_FBRD must not be greater than zero. If this is exceeded , the result is an aborted transmission or reception.

15.4.6 Fractional Baud Rate Register (UART_FBRD)

Address Offset: 28h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										BAUD DIVFRAC					
-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw

Bits 15:6	Reserved, forced by hardware to 0.
Bits 5:0	**BAUD DIVFRAC:** *Fractional Baud Rate Divider* **Notes:** In order to internally update the contents of UART_FBRD, a UART_LCR write must always be performed at the end. The baud rate must not be changed: - When the UART is enabled. - When completing a transmission or reception when it has been programmed to become disabled. The minimum divide ratio possible is 1 and the maximum is 65535(2^{16} - 1). When this is the case, UART_IBRD = 0 is invalid and UART_FBRD is ignored. Similarly, when UART_IBRD = 65535 (that is 0xFFFF), then UART_FBRD must not be greater than zero. If this is exceeded, the result is an aborted transmission or reception.

15.4.7 Line Control Register (UART_LCR)

Address Offset: 2Ch

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			LBKLEN				LINEN	SPS	WLEN		FEN	STP2	EPS	PEN	Res.
-	-	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	-

Bits 15:13	Reserved, forced by hardware to 0.
Bits 12:9	**LBKLEN:** *LIN Break Length* 0000: 10 low bits 0001: 11 low bits 0010: 12 low bits 0011: 13 low bits 0100: 14 low bits 0101: 15 low bits 0110: 16 low bits 0111: 17 low bits 1000: 18 low bits 1001: 19 low bits 1010: 20 low bits others: 20 low bits

Bit 8	**LINEN:** *LIN Master Mode Enable* 0: LIN MASTER mode disabled. The break length in transmission depends on the character configuration which is defined by: - Start - Word length defined by WLEN (bit 6:5 of UART_LCR) - Parity enable PEN (bit 1 of UART_LCR) - Stop length defined by STP2 (bit 3 of UART_LCR). STP2 = ‘1’ is not allowed. For example: if WLEN=11 and PEN=1 and STP2=0 then 11 bits at ‘0’ will be sent, followed by 1 bit at ‘1’: 1 start + 8 data + 1 parity + 1 stop = 11. The break character is followed by at least 1 bit at ‘1’. 1: LIN MASTER mode enabled. The break length in transmission is defined by the LBKLEN bits in the UART_LCR register. The break character is followed by at least 1 bit at ‘1’. **Note:** A BREAK ERROR FLAG is set when the received data input is held LOW for longer than a full-word transfer time (defined by start, data, parity and stop bits). Enabling LIN master mode (LINEN=1) affects only transmission and does not prevent you from choosing the correct character configuration for reception: 1 start, 8 data, 1 stop. In this case a break is received as at least 10 low bits.
Bit 7	**SPS:** *Stick Parity Select* When bits 1, 2, and 7 of the UART_LCR register are set, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set, and bit 2 is 0, the parity bit is transmitted and checked as a 1. When this bit is cleared stick parity is disabled. *Table 72 on page 367* is a truth table showing the SPS, EPS and PEN bits.
Bits 6:5	**WLEN:** *Word Length* The selected bits indicate the number of data bits transmitted or received in a frame as follows: 11 = 8 bits 10 = 7 bits 01 = 6 bits 00 = 5 bits
Bit 4	**FEN:** *Enable FIFOs* 0: FIFOs disabled (character mode): the FIFOs become 1-byte-deep holding registers. 1: FIFOs enabled.
Bit 3	**STP2:** *Two Stop Bits Select* 0: One stop bit is transmitted at the end of the frame. 1: Two stop bits are transmitted at the end of the frame. **Note**: The receive logic does not check for two stop bits being received.
Bit 2	**EPS:** *Even Parity Select* 0: Odd parity: checks for an odd number of ‘1s’ in data and parity bits. 1: Even parity: checks for an even number of ‘1s’ in data and parity bits. **Notes:** Generation and checking is performed during transmission and reception. This bit has no effect when parity is disabled by Parity Enable (bit 1) being cleared to 0. *Table 72 on page 367* is a truth table showing the SPS, EPS and PEN bits.
Bit 1	**PEN:** *Parity Enable* 0: Parity checking and generation disabled. 1: Parity checking and generation enabled. Refer to Table 72.
Bit 0	Reserved, forced by hardware to 0.

Note: The line control register must not be changed :
- when the UART is enabled.

- when completing a transmission or a reception when it has been programmed to become disabled.

Table 72 is a truth table for the SPS, EPS and PEN bits of the UART_LCR register

Table 72. SPS, EPS and PEN bits truth table

Parity Enable (PEN)	Even Parity Select (EPS)	Stick Parity Select (SPS)	Parity bit (transmitted or checked)
0	x	x	Not transmitted or checked
1	1	0	Even parity
1	0	0	Odd parity
1	0	1	1
1	1	1	0

The integrity of the FIFO is not guaranteed if the software disables the UART in the middle of a transmission with data in the FIFO, and then re-enables it.

15.4.8 Control Register (UART_CR)

Address Offset: 30h

Reset value: 0300h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTSEn	RSTEn	Reserved		RTS	Res.	RXE	TXE	LBE	Reserved						UART EN
rw	rw	-	-	rw	-	rw	rw	rw	-	-	-	-	-	-	rw

Bit 15	**CTSEn:** *CTS Hardware Flow Control Enable* 0: CTS hardware flow control disabled. 1: CTS hardware flow control enabled: Data is only transmitted when the UART_CTS signal is asserted.
Bit 14	**RTSEn:** *RTS Hardware Flow Control Enable* 0: RTS hardware flow control disabled. 1: RTS hardware flow control enabled: Data is only requested when there is space in the receive FIFO for it to be received.
Bits 13:12	Reserved, forced by hardware to 0.
Bit 11	**RTS:** *Request to Send* This bit is the complement of the UART request to send (UART_RTS) modem status output. That is, when the bit is programmed to a 1, the output is 0.
Bit 10	Reserved, forced by hardware to 0.
Bit 9	**RXE:** *Receive Enable* 0: Receive section of the UART disabled. 1: Receive section of the UART enabled. **Note:** When the UART is disabled in the middle of reception, it completes the current character before stopping.
Bit 8	**TXE**: *Transmit Enable* 0: Transmit section of the UART disabled. 1: Transmit section of the UART enabled. **Note:** When the UART is disabled in the middle of transmission, it completes the current character before stopping.
Bit 7	**LBE:** *Loop Back Enable* 0: Loop Back disabled. 1: Loop Back enabled: UART_TX is fed to UART_RX and UART_RTS is fed to UART_CTS.
Bits 6:1	Reserved, forced by hardware to 0.
Bit 0	**UARTEN:** *UART Enable* 0: UART disabled. 1: UART enabled. **Notes:** When the UART is disabled in the middle of transmission or reception, it completes the current character before stopping. To enable transmission, both TXE, bit 8, and UARTEN, bit 0, must be set. Similarly, to enable reception, RXE, bit 9, and UARTEN, bit 0, must be set.

Note: The control register must be programmed as follows:
- Disable the UART.

- Wait for the end of transmission or reception of the current character.
- Flush the transmit FIFO by disabling bit 4 (FEN) in the line control register (UART_LCR).
- Reprogram the control register.
- Enable the UART.

15.4.9 Interrupt FIFO Level Select Register (UART_IFLS)

Address Offset: 34h

Reset value: 0012h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										RXIFLSEL			TXIFLSEL		
-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw

Bits 15:6	Reserved, forced by hardware to 0.
Bits 5:3	**RXIFLSEL:** *Receive Interrupt FIFO Level Select* The trigger points for the receive interrupt are as follows: 000 = Receive FIFO becomes >= 1/8 full 001 = Receive FIFO becomes >= 1/4 full 010 = Receive FIFO becomes >= 1/2 full 011 = Receive FIFO becomes >= 3/4 full 100 = Receive FIFO becomes >= 7/8 full 101:111 = reserved.
Bits 2:0	**TXIFLSEL:** *Transmit Interrupt FIFO Level Select* The trigger points for the transmit interrupt are as follows: 000 = Transmit FIFO becomes <= 1/8 full 001 = Transmit FIFO becomes <= 1/4 full 010 = Transmit FIFO becomes <= 1/2 full 011 = Transmit FIFO becomes <= 3/4 full 100 = Transmit FIFO becomes <= 7/8 full 101:111 = reserved.

Note: *The UART_IFLS register is the interrupt FIFO level select register. You can use the UART_IFLS register to define the FIFO level at which the TX and RX interrupts are triggered.*

The interrupts are generated based on a transition through a level rather than being based on the level. That is, the interrupts are generated when the fill level progresses through the trigger level.

The *reset value of these bits selects* the *trigger level of FIFOs at the half-way mark.*

15.4.10 Interrupt Mask Set/Clear Register (UART_IMSC)

Address Offset: 38h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					OEIM	BEIM	PEIM	FEIM	RTIM	TXIM	RXIM	Reserved		CTSIM	Res.
-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	-	-	rw	-

Bits 15:11	Reserved, forced by hardware to 0.
Bit 10	**OEIM:** *Overrun Error Interrupt Mask* 0: Overrun Error Interrupt Disabled 1: Overrun Error Interrupt Enable
Bit 9	**BEIM:** *Break Error Interrupt Mask* 0: Break Error Interrupt Disabled 1: Break Error Interrupt Enabled
Bit 8	**PEIM:** *Parity Error Interrupt Mask* 0: Parity Error Interrupt Disabled 1: Parity Error Interrupt Enabled
Bit 7	**FEIM:** *Framing Error Interrupt Mask* 0: Framing Error Interrupt Disabled 1: Framing Error Interrupt EnableD
Bit 6	**RTIM:** *Receive Timeout Interrupt Mask* 0: Receive Timeout Interrupt Disabled 1: Receive Timeout Interrupt Enabled
Bit 5	**TXIM:** *Transmit Interrupt Mask* 0: Transmit Interrupt Disabled 1: Transmit Interrupt Enabled
Bit 4	**RXIM:** *Receive Interrupt Mask* 0: Receive Interrupt Disabled 1: Receive Interrupt Enabled
Bits 3:2	Reserved, forced by hardware to 0.
Bit 1	**CTSIM:** *CTS Interrupt Mask* 0: CTS Interrupt Disable 1: CTS Interrupt Enable
Bit 0	Reserved, forced by hardware to 0.

15.4.11 Raw Interrupt Status Register (UART_RIS)

Address Offset: 3Ch

Reset value: 000-h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					OERIS	BERIS	PERIS	FERIS	RTRIS	TXRIS	RXRIS	Reserved		CTSRIS	Res.
-	-	-	-	-	r	r	r	r	r	r	r	-	-	r	r

Bits 15:11	Reserved, forced by hardware to 0.
Bit 10	**OERIS:** *Overrun Error Interrupt Status* Gives the raw interrupt state (prior to masking) of the OE interrupt. 0: No Overrun Error. 1: An Overrun Error occurs during reception of data.
Bit 9	**BERIS:** *Break Error Interrupt Status* Gives the raw interrupt state (prior to masking) of the BE interrupt. 0: No Break Error. 1: A Break Error occurs during reception of data.
Bit 8	**PERIS:** *Parity Error Interrupt Status* Gives the raw interrupt state (prior to masking) of the PE interrupt. 0: No Parity Error. 1: A Parity Error occurs during reception of data.
Bit 7	**FERIS:** *Framing Error Interrupt Status* Gives the raw interrupt state (prior to masking) of the FE interrupt. 0: No Framing Error. 1: A Framing Error occurs during reception of data.
Bit 6	**RTRIS:** *Receive Timeout Interrupt Status* Gives the masked raw interrupt state of the RT interrupt. 0: No Receive Timeout 1: A Receive Timeout interrupt occurs when the receive FIFO is not empty, and no further data is received over a 32 baud rate period. In this case, the raw interrupt cannot be set unless the mask is set, this being due to the fact that the mask acts as an *enable* for power saving. (RTRIS=RTMIS) **Note:** The receive timeout interrupt is cleared either when the FIFO becomes empty through reading all the data (or by reading the holding register), or when a 1 is written to the corresponding bit of the UART_ICR register.

Bit 5	**TXRIS:** *Transmit Interrupt Status* Gives the raw interrupt state (prior to masking) of the TX interrupt. The transmit interrupt changes state when one of the following events occurs: - If the FIFOs are enabled and the transmit FIFO reaches the programmed trigger level. When this happens, the transmit interrupt is asserted HIGH. The transmit interrupt is cleared by writing data to the transmit FIFO until it becomes greater than the trigger level, or by clearing the interrupt. - If the FIFOs are disabled (have a depth of one location) and there is no data present in the transmitter's single location, the transmit interrupt is asserted HIGH. It is cleared by performing a single write to the transmit FIFO, or by clearing the interrupt. **Note:** If TXRIS is cleared by writing '1' to the UART_ICR TXIC bit, without writing to the Data Register, TXRIS may be set and the TX interrupt may occur again if a break is requested. To update the transmit FIFO you must write data to the transmit FIFO either prior to enabling the UART and the interrupts, or after enabling the UART and interrupts. The transmit interrupt is based on a transition *through* a level, rather than on the level itself. When the interrupt and UART are enabled *before* any data is written to the transmit FIFO, the interrupt is not set. The interrupt is *only* set once written data leaves the single location of the transmit FIFO and it becomes empty. **Note:** If a break request occurs during the transmission after which TXRIS should be set, it will be set only after the break transmission.
Bit 4	**RXRIS:** *Receive Interrupt Status* Gives the raw interrupt state (prior to masking) of the RX interrupt. The receive interrupt changes state when one of the following events occurs: - If the FIFOs are enabled and the receive FIFO reaches the programmed trigger level. When this happens, the receive interrupt is asserted HIGH. The receive interrupt is cleared by reading data from the receive FIFO until it becomes less than the trigger level, or by clearing the interrupt. - If the FIFOs are disabled (have a depth of one location) and data is received thereby filling the location, the receive interrupt is asserted HIGH. The receive interrupt is cleared by performing a single read of the receive FIFO, or by clearing the interrupt.
Bits 3:2	Reserved, forced by hardware to 0.
Bit 1	**CTSRIS:** *CTS Interrupt Status* Gives the raw interrupt state (prior to masking) of the CTS interrupt. The CTS interrupt is asserted if there is a change on UART_CTS. It is cleared by writing a 1 to the corresponding bit in the UART_ICR register.
Bit 0	Reserved, forced by hardware to 0.

15.4.12 Masked Interrupt Status Register (UART_MIS)

Address Offset: 40h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					OEMIS	BEMIS	PEMIS	FEMIS	RTMIS	TXMIS	RXMIS	Reserved		CTSMIS	Res.
-	-	-	-	-	r	r	r	r	r	r	r	-	-	r	-

Bits 15:11	Reserved, forced by hardware to 0.
Bit 10	**OEMIS:** *Overrun Error Masked Interrupt Status* Gives the masked interrupt state (after masking) of the OE interrupt.
Bit 9	**BEMIS:** *Break Error Masked Interrupt Status* Gives the masked interrupt state (after masking) of the BE interrupt.
Bit 8	**PEMIS:** *Parity Error Masked Interrupt Status* Gives the masked interrupt state (after masking) of the PE interrupt.
Bit 7	**FEMIS:** *Frame Error Masked Interrupt Status* Gives the masked interrupt state (after masking) of the FE interrupt.
Bit 6	**RTMIS:** *Receive Timeout Masked Interrupt Status* Gives the masked interrupt state (after masking) of the RT interrupt.
Bit 5	**TXMIS:** *Transmit Masked Interrupt Status* Gives the masked interrupt state (after masking) of the TX interrupt.
Bit 4	**RXMIS:** *Receive Masked Interrupt Status* Gives the masked interrupt state (after masking) of the RX interrupt.
Bits 3:2	Reserved, forced by hardware to 0.
Bit 1	**CTSMIS:** *CTS Masked Interrupt Status* Gives the masked interrupt state (after masking) of the CTS interrupt.
Bit 0	Reserved, forced by hardware to 0.

15.4.13 Interrupt Clear Register (UART_ICR)

Address Offset: 44h

Reset value: -

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					OEIC	BEIC	PEIC	FEIC	RTIC	TXIC	RXIC	Reserved		CTSIC	Res.
-	-	-	-	-	w	w	w	w	w	w	w	-	-	w	-

Bits 15:11	Reserved, forced by hardware to 0.
Bit 10	**OEIC:** *Overrun Error Interrupt Clear* Write ‘1’ clears the OE interrupt. Write ‘0’ has no effect.
Bit 9	**BEIC:** *Break Error Interrupt Clear* Write ‘1’ clears the BE interrupt. Write ‘0’ has no effect.
Bit 8	**PEIC:** *Parity Error Interrupt Clear* Write ‘1’ clears the PE interrupt. Write ‘0’ has no effect.
Bit 7	**FEIC:** *Frame Error Interrupt Clear* Write ‘1’ clears the FE interrupt. Write ‘0’ has no effect.
Bit 6	**RTIC:** *Receive Timeout Interrupt Clear* Write ‘1’ clears the RT interrupt. Write ‘0’ has no effect.
Bit 5	**TXIC:** *Transmit Interrupt Clear* Write ‘1’ clears the TX interrupt. Write ‘0’ has no effect.
Bit 4	**RXIC:** *Receive Interrupt Clear* Write ‘1’ clears the RX interrupt. Write ‘0’ has no effect.
Bits 3:2	Reserved, forced by hardware to 0.
Bit 1	**CTSIC:** *CTS Interrupt Clear* Write ‘1’ clears the CTS interrupt. Write ‘0’ has no effect.
Bit 0	Reserved, forced by hardware to 0.

15.4.14 DMA Control Register (UART_DMACR)

Address Offset: 48h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												DMASD	DMAONERR	TXDMAE	RXDMAE
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bits 15:4	Reserved, forced by hardware to 0.
Bit 3	**DMASD:** *DMA Single Request Disable* 0: DMA Single Request Enabled. 1: DMA Single Request Disabled. **Note:** DMASD must be set in order to generate DMA Burst Request only, otherwise a DMA request would be needed for each transfer. Generating a DMA single request is useful in situations where the number of characters left to be received or sent in the stream is less than a burst.
Bit 2	**DMAONERR:** *DMA on Error* 0:The DMA receive requests are enabled and can be asserted when receiving a character, whatever a UART error is asserted or not. 1:The DMA receive requests are disabled when a UART error is asserted.
Bit 1	**TXDMAE:** *Transmit DMA Enable* 0: DMA for the transmit FIFO disabled. 1: DMA for the transmit FIFO enabled. A DMA is requested: - when there is at least one empty location in the transmit FIFO if DMASD = 0. - when the transmit FIFO contains less characters than the watermark level (programmed in UART_IFLS register).
Bit 0	**RXDMAE:** *Receive DMA Enable* 0: DMA for the receive FIFO disabled. 1: DMA for the receive FIFO enabled. A DMA is requested when the receive FIFO contains at least one character if DMASD = 0, or when the receive FIFO contains more characters than the programmed watermark level (programmed in the UART_IFLS register).

When the UART is in FIFO enabled mode, data transfers can be made by either single or burst transfers, depending on the programmed watermark level and the amount of data in the FIFO. *Table 73 on page 375* shows the trigger points for DMA burst requests depending on the watermark level, for both the transmit and receive FIFOs.

Table 73. Trigger points for DMA burst requests

Watermark level	Burst length	
	Tramsmit (number of empty locations)	**Receive (number of filled locations**
1/4	-	4
1/2	8	8
3/4	4	-

15.5 UART Register map

Table 74. Register map

Address offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00	UART_DR	Data register															
04	UART_RSR	Receive Status register															
18	UART_FR	Flag register															
1C	UART_BKR	Break register															
24	UART_IBRD	Integer Baud Rate Divider register															
28	UART_FBRD	Fractional Baud Rate Divider register															
2C	UART_LCR	Line Control register															
30	UART_CR	Control register															
34	UART_IFLS	Interrupt FIFO Level Select															
38	UART_IMSC	Interrupt Mask Set/Clear register															
3C	UART_RIS	Raw Interrupt Status															
40	UART_MIS	Masked Interrupt Status															
44	UART_ICR	Interrupt Clear register															
48	UART_DMACR	DMA Control register															

See *Table 2* for the base addresses.

16 USB full-speed device interface (USB)

16.1 Introduction

The USB Peripheral implements an interface between a full-speed USB 2.0 bus and the APB bus.

USB suspend/resume are supported which allows to stop the device clocks for low power consumption.

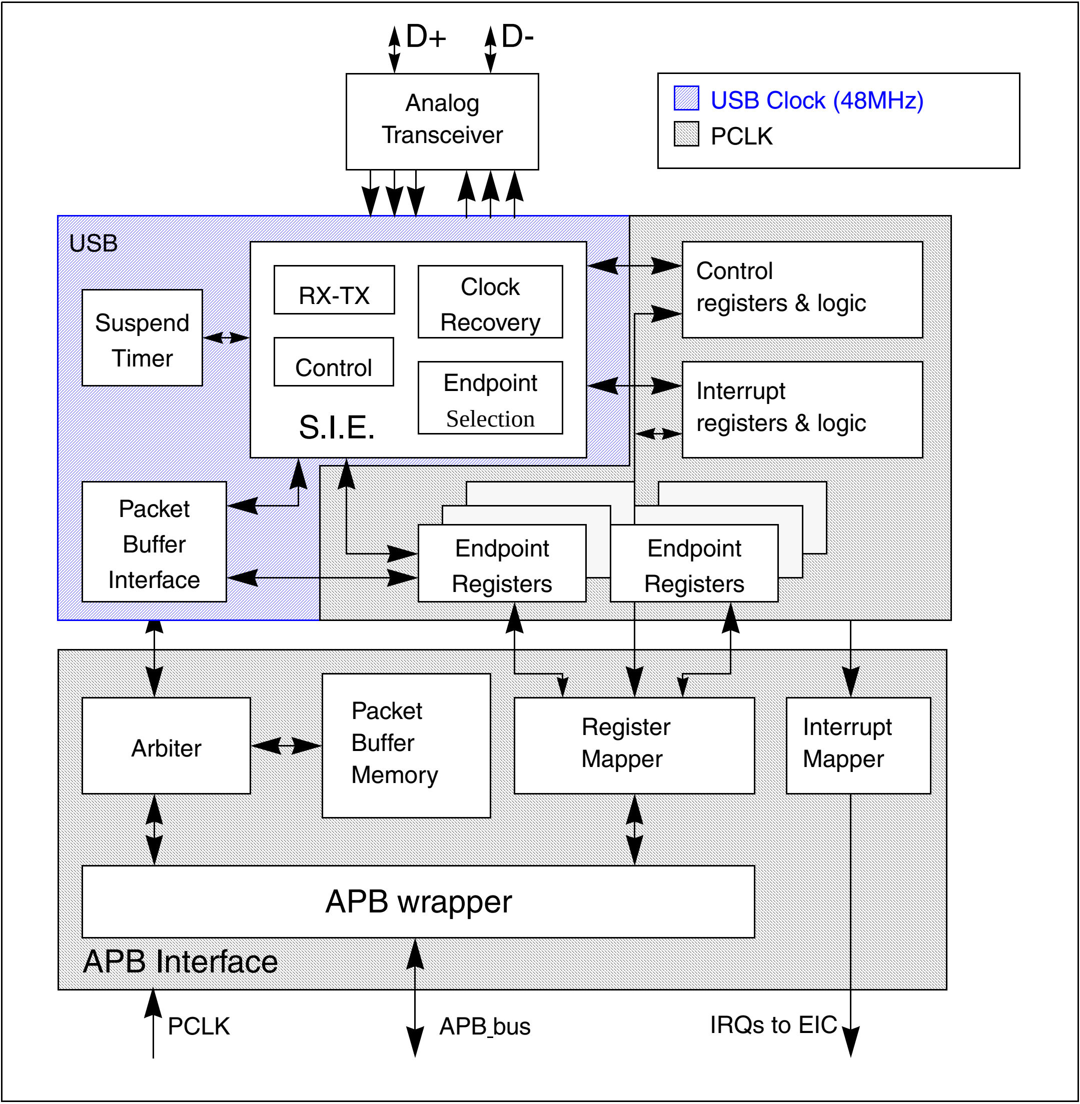
16.2 Main features

- USB specification version 2.0 Full speed compliant
- Configurable number of endpoints from 1 to 8
- Cyclic Redundancy Check (CRC) generation/checking, Non-Return-to-Zero Inverted (NRZI) encoding/decoding and bit-stuffing
- Isochronous transfers support
- Double-buffered bulk endpoint support
- USB Suspend/Resume operations
- Frame locked clock pulse generation

16.3 Block diagram

Figure 113 shows the block diagram of the USB Peripheral.

Figure 113. USB Peripheral block diagram



16.4 Functional description

The USB Peripheral provides an USB compliant connection between the host PC and the function implemented by the microcontroller. Data transfer between the host PC and the system memory occurs through a dedicated packet buffer memory accessed directly by the USB Peripheral. The size of this dedicated buffer memory must be according to the number of endpoints used and the maximum packet size. This dedicated memory is sized to 512 Byte and up to 16 mono-directional/single-buffered endpoints can be used. The USB Peripheral interfaces with the USB host, detecting token packets, handling data transmission/reception, and processing handshake packets as required by the USB standard. Transaction formatting is performed by the hardware, including CRC generation and checking.

Each endpoint is associated with a buffer description block indicating where the endpoint related memory area is located, how large it is or how many bytes must be transmitted.

When a token for a valid function/endpoint pair is recognized by the USB Peripheral, the related data transfer (if required and if the endpoint is configured) takes place. The data buffered by the USB Peripheral is loaded in an internal 16 bit register and memory access to the dedicated buffer is performed. When all the data has been transferred, if needed, the proper handshake packet over the USB is generated or expected according to the direction of the transfer.

At the end of the transaction, an endpoint-specific interrupt is generated, reading status registers and/or using different interrupt response routines. The microcontroller can determine:

- which endpoint has to be served
- which type of transaction took place, if errors occurred (bit stuffing, format, CRC, protocol, missing ACK, over/underrun, etc).

Two interrupt lines are generated by the USB Peripheral : one IRQ collecting high priority endpoint interrupts (isochronous and double-buffered bulk) and another IRQ collecting all other interrupt sources (check the IRQ interrupt vector table for detailed interrupt source mapping).

Special support is offered to Isochronous transfers and high throughput bulk transfers, implementing a double buffer usage, which allows to always have an available buffer for the USB Peripheral while the microcontroller uses the other one.

The unit can be placed in low-power mode (SUSPEND mode), by writing in the control register, whenever required. At this time, all static power dissipation is avoided, and the USB clock can be slowed down or stopped. The detection of activity at the USB inputs, while in low-power mode, wakes the device up asynchronously. A special interrupt source can be connected directly to a wake-up line to allow the system to immediately restart the normal clock generation and/or support direct clock start/stop.

16.4.1 Description of USB blocks

The USB Peripheral implements all the features related to USB interfacing, which include the following blocks:

- Serial Interface Engine (SIE): The functions of this block include: synchronization pattern recognition, bit-stuffing, CRC generation and checking, PID verification/generation, and handshake evaluation. It must interface with the USB transceivers and uses the virtual buffers provided by the packet buffer interface for local data storage,. This unit also generates signals according to USB Peripheral events, such as Start of Frame (SOF), USB_Reset, Data errors etc. and to Endpoint related events like end of transmission or correct reception of a packet; these signals are then used to generate interrupts.
- Suspend Timer: This block generates the frame locked clock pulse for any external device requiring Start-of-Frame synchronization and it detects a global suspend (from the host) when no traffic has been received for 3 mS.
- Packet Buffer Interface: This block manages the local memory implementing a set of buffers in a flexible way, both for transmission and reception. It can choose the proper buffer according to requests coming from the SIE and locate them in the memory addresses pointed by the Endpoint registers. It increments the address after each exchanged word until the end of packet, keeping track of the number of exchanged bytes and preventing the buffer to overrun the maximum capacity.
- Endpoint-Related Registers: Each endpoint has an associated register containing the endpoint type and its current status. For mono-directional/single-buffer endpoints, a

single register can be used to implement two distinct endpoints. The number of registers is 8, allowing up to 8 double-buffer endpoints or up to 16 mono-directional/single-buffer ones in any combination. For example the USB Peripheral can be programmed to have 4 doublebuffer endpoints and 8 single-buffer/mono-directional endpoints.

- Control Registers: These are the registers containing information about the status of the whole USB Peripheral and used to force some USB events, such as resume and power-down.
- Interrupt Registers: These contain the Interrupt masks and a record of the events. They can be used to inquire an interrupt reason, the interrupt status or to clear the status of a pending interrupt.

The USB Peripheral is connected to the APB bus through an APB interface, containing the following blocks:

- Packet Memory: This is the local memory that physically contains the Packet Buffers. It can be used by the Packet Buffer interface, which creates the data structure and can be accessed directly by the application software. The size of the Packet Memory is 512 Bytes, structured as 256 words by 16 bits.
- Arbiter: This block accepts memory requests coming from the APB bus and from the USB interface. It resolves the conflicts by giving priority to APB accesses, while always reserving half of the memory bandwidth to complete all USB transfers. This time-duplex scheme implements a virtual dual-port RAM that allows memory access, while an USB transaction is happening. Multi-word APB transfers of any length are also allowed by this scheme.
- Register Mapper: This block collects the various byte-wide and bit-wide registers of the USB Peripheral in a structured 16-bit wide word set addressed by the APB.
- Interrupt Mapper: This block is used to select how the possible USB events can generate interrupts and map them to IRQ lines of the EIC.
- APB Wrapper: This provides an interface to the APB for the memory and register. It also maps the whole USB Peripheral in the APB address space.

16.5 Programming considerations

In the following sections, the expected interactions between the USB Peripheral and the application program are described, in order to ease application software development.

16.5.1 Generic USB device programming

This part describes the main tasks required of the application software in order to obtain USB compliant behaviour. The actions related to the most general USB events are taken into account and paragraphs are dedicated to the special cases of double-buffered endpoints and Isochronous transfers. Apart from system reset, action is always initiated by the USB Peripheral, driven by one of the USB events described below.

16.5.2 System and power-on reset

Upon system and power-on reset, the first operation the application software should perform is to provide all required clock signals to the USB Peripheral and subsequently de-assert its reset signal so to be able to access its registers. The whole initialization sequence is hereafter described.

As a first step application software needs to activate register macrocell clock and de-assert macrocell specific reset signal using related control bits provided by device clock management logic.

After that the analog part of the device related to the USB transceiver must be switched on using the PDWN bit in CNTR register which requires a special handling. This bit is intended to switch on the internal voltage references supplying the port transceiver . Since this circuit has a defined start-up time ($t_{STARTUP}$) specified in the datasheet, during which the behaviour of USB transceiver is not defined, it is necessary to wait this time, after having set the PDWN bit in CNTR register, then the reset condition on the USB part can be removed (clearing of FRES bit in CNTR register) and the ISTR register can be cleared, removing any spurious pending interrupt, before enabling any other macrocell operation.

As a last step the USB specific 48 MHz clock needs to be activated, using the related control bits provided by device clock management logic.

At system reset, the microcontroller must initialize all required registers and the packet buffer description table, to make the USB Peripheral able to properly generate interrupts and data transfers. All registers not specific to any endpoint must be initialized according to the needs of application software (choice of enabled interrupts, chosen address of packet buffers, etc.). Then the process continues as for the USB reset case (see further paragraph).

USB reset (RESET interrupt)

When this event occurs, the USB Peripheral is put in the same conditions it is left by the system reset after the initialization described in the previous paragraph: communication is disabled in all endpoint registers (the USB Peripheral will not respond to any packet). As a response to the USB reset event, the USB function must be enabled, having as USB address 0, implementing only the default control endpoint (endpoint address is 0 too). This is accomplished by setting the Enable Function (EF) bit of the USB_DADDR register and initializing the EP0R register and its related packet buffers accordingly. During USB enumeration process, the host assigns a unique address to this device, which must be written in the ADD[6:0] bits of the USB_DADDR register, and configures any other necessary endpoint.
When a RESET interrupt is received, the application software is responsible to enable again the default endpoint of USB function 0 within 10mS from the end of reset sequence which triggered the interrupt.

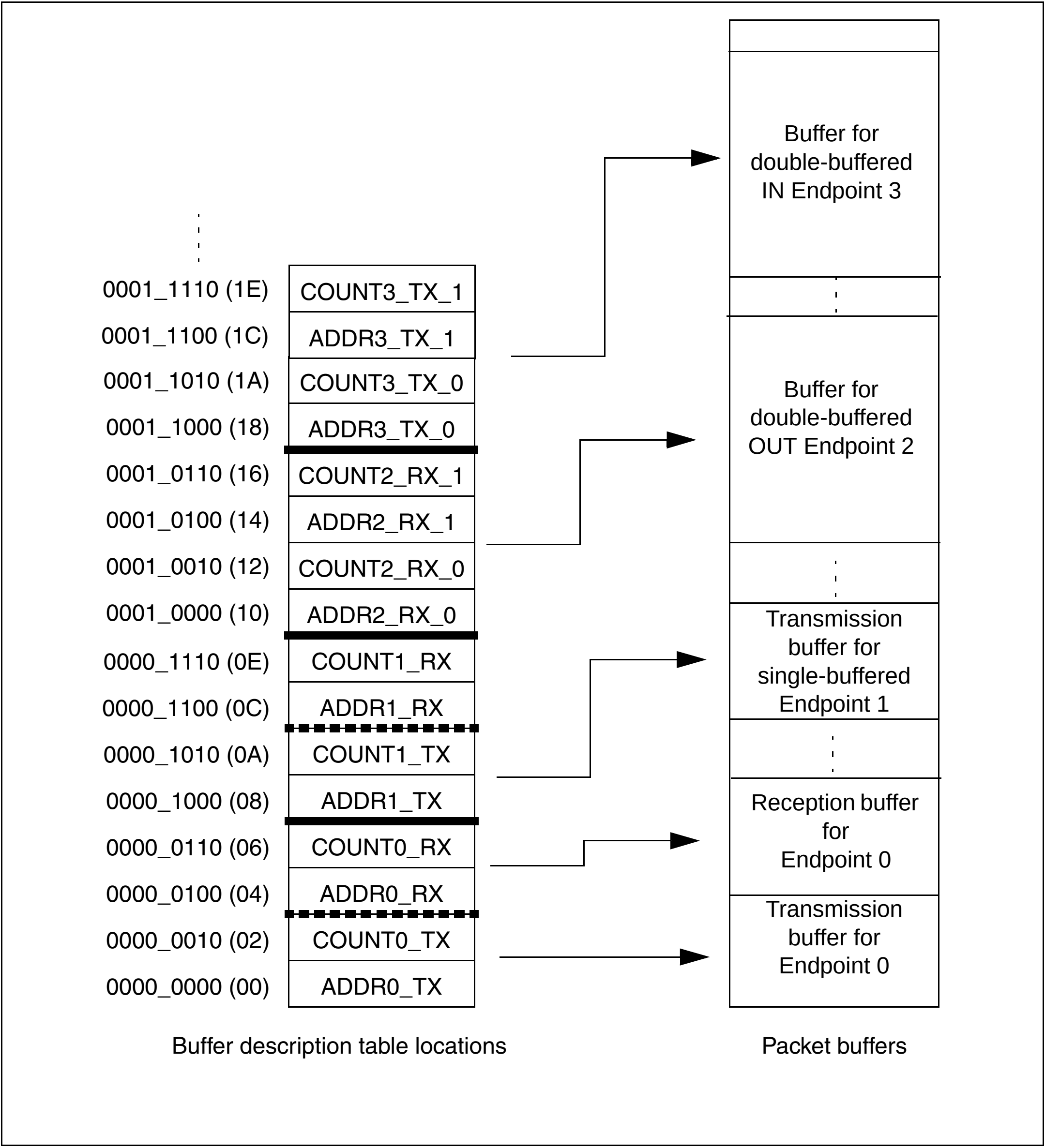
Structure and usage of packet buffers

Each bidirectional endpoint may receive or transmit data from/to the host. The received data is stored in a dedicated memory buffer reserved for that endpoint, while another memory buffer contains the data to be transmitted by the endpoint. Access to this memory is performed by the packet buffer interface block, which delivers a memory access request and waits for its acknowledgement. Since the packet buffer memory has to be accessed by the microcontroller also, an arbitration logic takes care of the access conflicts, using half APB cycle for microcontroller access and the remaining half for the USB Peripheral access. In this way, both the agents can operate as if the packet memory is a dual-port RAM, without being aware of any conflict even when the microcontroller is performing back-to-back accesses. The USB Peripheral logic uses a dedicated clock. The frequency of this dedicated clock is fixed by the requirements of the USB standard at 48 MHz, and this can be different from the clock used for the interface to the APB bus. Different clock configurations are possible where the APB clock frequency can be higher or lower than the USB Peripheral one.

Note: Due to USB data rate and packet memory interface requirements, the APB clock frequency must be greater than 8 MHz to avoid data overrun/underrun problems.

Each endpoint is associated with two packet buffers (usually one for transmission and the other one for reception). The size of the buffer can be upto 512 words each. Buffers can be placed anywhere inside the packet memory because their location and size is specified in a buffer description table, which is also located in the packet memory at the address indicated by the USB_BTABLE register. Each table entry is associated to an endpoint register and it is composed of four 16-bit words so that table start address must always be aligned to an 8-byte boundary (the lowest three bits of USB_BTABLE register are always “000”). Buffer descriptor table entries are described in the *Section 16.6.3: Buffer descriptor table*. If an endpoint is unidirectional and it is neither an Isochronous nor a double-buffered bulk, only one packet buffer is required (the one related to the supported transfer direction). Other table locations related to unsupported transfer directions or unused endpoints, are available to the user. isochronous and double-buffered bulk endpoints have special handling of packet buffers (Refer to *Section 16.5.4: Isochronous transfers* and *Section 16.5.3: Double-buffered endpoints* respectively). The relationship between buffer description table entries and packet buffer areas is depicted in *Figure 114*.

Figure 114. Packet buffer areas with examples of buffer description table locations



Each packet buffer is used either during reception or transmission starting from the bottom. The USB Peripheral will never change the contents of memory locations adjacent to the allocated memory buffers; if a packet bigger than the allocated buffer length is received (buffer overrun condition) the data will be copied to the memory only up to the last available location.

Endpoint initialization

The first step to initialize an endpoint is to write appropriate values to the ADDRn_TX/ADDRn_RX registers so that the USB Peripheral finds the data to be transmitted already available and the data to be received can be buffered. The EP_TYPE bits in the USB_EPnR register must be set according to the endpoint type, eventually using the EP_KIND bit to enable any special required feature. On the transmit side, the endpoint must be enabled using the STAT_TX bits in the USB_EPnR register and COUNTn_TX must be initialized. For reception, STAT_RX bits must be set to enable reception and COUNTn_RX must be written with the allocated buffer size using the BL_SIZE and NUM_BLOCK fields. Unidirectional endpoints, except Isochronous and double-buffered bulk endpoints, need to initialize only bits and registers related to the supported direction. Once the transmission and/or reception are enabled, register USB_EPnR and locations ADDRn_TX/ADDRn_RX, COUNTn_TX/COUNTn_RX (respectively), should not be modified by the application software, as the hardware can change their value on the fly. When the data transfer operation is completed, notified by a CTR interrupt event, they can be accessed again to re-enable a new operation.

IN packets (data transmission)

When receiving an IN token packet, if the received address matches a configured and valid endpoint one, the USB Peripheral accesses the contents of ADDRn_TX and COUNTn_TX locations inside buffer descriptor table entry related to the addressed endpoint. The content of these locations is stored in its internal 16 bit registers ADDR and COUNT (not accessible by software). The packet memory is accessed again to read the first word to be transmitted (Refer to *Section : Structure and usage of packet buffers*) and starts sending a DATA0 or DATA1 PID according to USB_EPnR bit DTOG_TX. When the PID is completed, the first byte from the word, read from buffer memory, is loaded into the output shift register to be transmitted on the USB bus. After the last data byte is transmitted, the computed CRC is sent. If the addressed endpoint is not valid, a NAK or STALL handshake packet is sent instead of the data packet, according to STAT_TX bits in the USB_EPnR register.

The ADDR internal register is used as a pointer to the current buffer memory location while COUNT is used to count the number of remaining bytes to be transmitted. Each word read from the packet buffer memory is transmitted over the USB bus starting from the least significant byte. Transmission buffer memory is read starting from the address pointed by ADDRn_TX for COUNTn_TX/2 words. If a transmitted packet is composed of an odd number of bytes, only the lower half of the last word accessed will be used.

On receiving the ACK receipt by the host, the USB_EPnR register is updated in the following way: DTOG_TX bit is toggled, the endpoint is made invalid by setting STAT_TX=10 (NAK) and bit CTR_TX is set. The application software must first identify the endpoint, which is requesting microcontroller attention by examining the EP_ID and DIR bits in the USB_ISTR register. Servicing of the CTR_TX event starts clearing the interrupt bit; the application software then prepares another buffer full of data to be sent, updates the COUNTn_TX table location with the number of byte to be transmitted during the next transfer, and finally sets STAT_TX to ‘11’ (VALID) to re-enable transmissions. While the STAT_TX bits are equal to ‘10’ (NAK), any IN request addressed to that endpoint is NAKed, indicating a flow control condition: the USB host will retry the transaction until it succeeds. It is mandatory to execute the sequence of operations in the above mentioned order to avoid losing the notification of a second IN transaction addressed to the same endpoint immediately following the one which triggered the CTR interrupt.

OUT and SETUP packets (data reception)

These two tokens are handled by the USB Peripheral more or less in the same way; the differences in the handling of SETUP packets are detailed in the following paragraph about control transfers. When receiving an OUT/SETUP PID, if the address matches a valid endpoint, the USB Peripheral accesses the contents of the ADDRn_RX and COUNTn_RX locations inside the buffer descriptor table entry related to the addressed endpoint. The content of the ADDRn_RX is stored directly in its internal register ADDR. While COUNT is now reset and the values of BL_SIZE and NUM_BLOCK bit fields, which are read within COUNTn_RX content are used to initialize BUF_COUNT, an internal 16 bit counter, which is used to check the buffer overrun condition (all these internal registers are not accessible by software). Data bytes subsequently received by the USB Peripheral are packed in words (the first byte received is stored as least significant byte) and then transferred to the packet buffer starting from the address contained in the internal ADDR register while BUF_COUNT is decremented and COUNT is incremented at each byte transfer. When the end of DATA packet is detected, the correctness of the received CRC is tested and only if no errors occurred during the reception, an ACK handshake packet is sent back to the transmitting host. In case of wrong CRC or other kinds of errors (bit-stuff violations, frame errors, etc.), data bytes are anyways copied in the packet memory buffer, at least until the error detection point, but ACK packet is not sent and the ERR bit in USB_ISTR register is set. However, there is usually no software action required in this case: the USB Peripheral recovers from reception errors and remains ready for the next transaction to come. If the addressed endpoint is not valid, a NAK or STALL handshake packet is sent instead of the ACK, according to bits STAT_RX in the USB_EPnR register and no data is written in the reception memory buffers.

Reception memory buffer locations are written starting from the address contained in the ADDRn_RX for a number of bytes corresponding to the received data packet length, CRC included (i.e. data payload length + 2), or up to the last allocated memory location, as defined by BL_SIZE and NUM_BLOCK, whichever comes first. In this way, the USB Peripheral never writes beyond the end of the allocated reception memory buffer area. If the length of the data packet payload (actual number of bytes used by the application) is greater than the allocated buffer, the USB Peripheral detects a buffer overrun condition. in this case, a STALL handshake is sent instead of the usual ACK to notify the problem to the host, no interrupt is generated and the transaction is considered failed.

When the transaction is completed correctly, by sending the ACK handshake packet, the internal COUNT register is copied back in the COUNTn_RX location inside the buffer description table entry, leaving unaffected BL_SIZE and NUM_BLOCK fields, which normally do not require to be re-written, and the USB_EPnR register is updated in the following way: DTOG_RX bit is toggled, the endpoint is made invalid by setting STAT_RX = ‘10’ (NAK) and bit CTR_RX is set. If the transaction has failed due to errors or buffer overrun condition, none of the previously listed actions take place. The application software must first identify the endpoint, which is requesting microcontroller attention by examining the EP_ID and DIR bits in the USB_ISTR register. The CTR_RX event is serviced by first determining the transaction type (SETUP bit in the USB_EPnR register); the application software must clear the interrupt flag bit and get the number of received bytes reading the COUNTn_RX location inside the buffer description table entry related to the endpoint being processed. After the received data is processed, the application software should set the STAT_RX bits to ‘11’ (Valid) in the USB_EPnR, enabling further transactions. While the STAT_RX bits are equal to ‘10’ (NAK), any OUT request addressed to that endpoint is NAKed, indicating a flow control condition: the USB host will retry the transaction until it succeeds. It is mandatory to execute the sequence of operations in the above mentioned order to avoid losing the notification of a second OUT transaction addressed to the same endpoint following immediately the one which triggered the CTR interrupt.

Control transfers

Control transfers are made of a SETUP transaction, followed by zero or more data stages, all of the same direction, followed by a status stage (a zero-byte transfer in the opposite direction). SETUP transactions are handled by control endpoints only and are very similar to OUT ones (data reception) except that the values of DTOG_TX and DTOG_RX bits of the addressed endpoint registers are set to 1 and 0 respectively, to initialize the control transfer, and both STAT_TX and STAT_RX are set to ‘10’ (NAK) to let software decide if subsequent transactions must be IN or OUT depending on the SETUP contents. A control endpoint must check SETUP bit in the USB_EPnR register at each CTR_RX event to distinguish normal OUT transactions from SETUP ones. A USB device can determine the number and direction of data stages by interpreting the data transferred in the SETUP stage, and is required to STALL the transaction in the case of errors. To do so, at all data stages before the last, the unused direction should be set to STALL, so that, if the host reverses the transfer direction too soon, it gets a STALL as a status stage. While enabling the last data stage, the opposite direction should be set to NAK, so that, if the host reverses the transfer direction (to perform the status stage) immediately, it is kept waiting for the completion of the control operation. If the control operation completes successfully, the software will change NAK to VALID, otherwise to STALL. At the same time, if the status stage will be an OUT, the STATUS_OUT (EP_KIND in the USB_EPnR register) bit should be set, so that an error is generated if a status transaction is performed with not-zero data. When the status transaction is serviced, the application clears the STATUS_OUT bit and sets STAT_RX to VALID (to accept a new command) and STAT_TX to NAK (to delay a possible status stage immediately following the next setup).

Since the USB specification states that a SETUP packet cannot be answered with a handshake different from ACK, eventually aborting a previously issued command to start the new one, the USB logic doesn’t allow a control endpoint to answer with a NAK or STALL packet to a SETUP token received from the host.

When the STAT_RX bits are set to ‘01’ (STALL) or ‘10’ (NAK) and a SETUP token is received, the USB accepts the data, performing the required data transfers and sends back an ACK handshake. If that endpoint has a previously issued CTR_RX request not yet acknowledged by the application (i.e. CTR_RX bit is still set from a previously completed reception), the USB discards the SETUP transaction and does not answer with any handshake packet regardless of its state, simulating a reception error and forcing the host to send the SETUP token again. This is done to avoid losing the notification of a SETUP transaction addressed to the same endpoint immediately following the transaction, which triggered the CTR_RX interrupt.

16.5.3 Double-buffered endpoints

All different endpoint types defined by the USB standard represent different traffic models, and describe the typical requirements of different kind of data transfer operations. When large portions of data are to be transferred between the host PC and the USB function, the bulk endpoint type is the most suited model. This is because the host schedules bulk transactions so as to fill all the available bandwidth in the frame, maximizing the actual transfer rate as long as the USB function is ready to handle a bulk transaction addressed to it. If the USB function is still busy with the previous transaction when the next one arrives, it will answer with a NAK handshake and the host PC will issue the same transaction again until the USB function is ready to handle it, reducing the actual transfer rate due to the bandwidth occupied by re-transmissions. For this reason, a dedicated feature called ‘double-buffering’ can be used with bulk endpoints.

When ‘double-buffering’ is activated, data toggle sequencing is used to select, which buffer is to be used by the USB Peripheral to perform the required data transfers, using both ‘transmission’ and ‘reception’ packet memory areas to manage buffer swapping on each successful transaction in order to always have a complete buffer to be used by the application, while the USB Peripheral fills the other one. For example, during an OUT transaction directed to a ‘reception’ double-buffered bulk endpoint, while one buffer is being filled with new data coming from the USB host, the other one is available for the microcontroller software usage (the same would happen with a ‘transmission’ double-buffered bulk endpoint and an IN transaction).

Since the swapped buffer management requires the usage of all 4 buffer description table locations hosting the address pointer and the length of the allocated memory buffers, the USB_EPnR registers used to implement double-buffered bulk endpoints are forced to be used as uni-directional ones. Therefore, only one STAT bit pair must be set at a value different from ‘00’ (Disabled): STAT_RX if the double-buffered bulk endpoint is enabled for reception, STAT_TX if the double-buffered bulk endpoint is enabled for transmission. In case it is required to have double-buffered bulk endpoints enabled both for reception and transmission, two USB_EPnR registers must be used.

To exploit the double-buffering feature and reach the highest possible transfer rate, the endpoint flow control structure, described in previous chapters, has to be modified, in order to switch the endpoint status to NAK only when a buffer conflict occurs between the USB Peripheral and application software, instead of doing it at the end of each successful transaction. The memory buffer which is currently being used by the USB Peripheral is defined by the DTOG bit related to the endpoint direction: DTOG_RX (bit 14 of USB_EPnR register) for ‘reception’ double-buffered bulk endpoints or DTOG_TX (bit 6 of USB_EPnR register) for ‘transmission’ double-buffered bulk endpoints. To implement the new flow control scheme, the USB Peripheral should know which packet buffer is currently in use by the application software, so to be aware of any conflict. Since in the USB_EPnR register, there are two DTOG bits but only one is used by USB Peripheral for data and buffer sequencing (due to the uni-directional constraint required by double-buffering feature) the other one can be used by the application software to show which buffer it is currently using. This new buffer flag is called SW_BUF. In the following table the correspondence between USB_EPnR register bits and DTOG/SW_BUF definition is explained, for the cases of ‘transmission’ and ‘reception’ double-buffered bulk endpoints.

Table 75. **Double-buffering buffer flag definition**

Buffer flag	‘Transmission’ endpoint	‘Reception’ endpoint
DTOG	DTOG_TX (USB_EPnR bit 6)	DTOG_RX (USB_EPnR bit 14)
SW_BUF	USB_EPnR bit 14	USB_EPnR bit 6

The memory buffer which is currently being used by the USB Peripheral is defined by DTOG buffer flag, while the buffer currently in use by application software is identified by SW_BUF buffer flag. The relationship between the buffer flag value and the used packet buffer is the same in both cases, and it is listed in the following table.

Table 76. **Double-buffering memory buffers usage**

Endpoint Type	DTOG or SW_BUF bit value	Packet buffer used by USB Peripheral (DTOG) or application software (SW_BUF)
IN	0	ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.
	1	ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.
OUT	0	ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.
	1	ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.

Double-buffering feature for a bulk endpoint is activated by:

- writing EP_TYPE bit field at ‘00’ in its USB_EPnR register, to define the endpoint as a bulk, and
- setting EP_KIND bit at ‘1’ (DBL_BUF), in the same register.

The application software is responsible for DTOG and SW_BUF bits initialization according to the first buffer to be used; this has to be done considering the special toggle-only property that these two bits have. The end of the first transaction occurring after having set DBL_BUF, triggers the special flow control of double-buffered bulk endpoints, which is used for all other transactions addressed to this endpoint until DBL_BUF remain set. At the end of each transaction the CTR_RX or CTR_TX bit of the addressed endpoint USB_EPnR register is set, depending on the enabled direction. At the same time, the affected DTOG bit in the USB_EPnR register is hardware toggled making the USB Peripheral buffer swapping completely software independent. Unlike common transactions, and the first one after DBL_BUF setting, STAT bit pair is not affected by the transaction termination and its value remains ‘11’ (Valid). However, as the token packet of a new transaction is received, the actual endpoint status will be masked as ‘10’ (NAK) when a buffer conflict between the USB Peripheral and the application software is detected (this condition is identified by DTOG and SW_BUF having the same value). The application software responds to the CTR event notification by clearing the interrupt flag and starting any required handling of the completed transaction. When the application packet buffer usage is over, the software toggles the SW_BUF bit, writing ‘1’ to it, to notify the USB Peripheral about the availability of that buffer. In this way, the number of NAKed transactions is limited only by the application elaboration time of a transaction data: if the elaboration time is shorter than the time required to complete a transaction on the USB bus, no re-transmissions due to flow control will take place and the actual transfer rate will be limited only by the host PC.

The application software can always override the special flow control implemented for double-buffered bulk endpoints, writing an explicit status different from '11' (Valid) into the STAT bit pair of the related USB_EPnR register. In this case, the USB Peripheral will always use the programmed endpoint status, regardless of the buffer usage condition.

16.5.4 Isochronous transfers

The USB standard supports full speed peripherals requiring a fixed and accurate data production/consume frequency, defining this kind of traffic as 'Isochronous'. Typical examples of this data are: audio samples, compressed video streams, and in general any sort of sampled data having strict requirements for the accuracy of delivered frequency. When an endpoint is defined to be 'isochronous' during the enumeration phase, the host allocates in the frame the required bandwidth and delivers exactly one IN or OUT packet each frame, depending on endpoint direction. To limit the bandwidth requirements, no re-transmission of failed transactions is possible for Isochronous traffic; this leads to the fact that an isochronous transaction does not have a handshake phase and no ACK packet is expected or sent after the data packet. For the same reason, Isochronous transfers do not support data toggle sequencing and always use DATA0 PID to start any data packet.

The Isochronous behaviour for an endpoint is selected by setting the EP_TYPE bits at '10' in its USB_EPnR register; since there is no handshake phase the only legal values for the STAT_RX/STAT_TX bit pairs are '00' (Disabled) and '11' (Valid), any other value will produce results not compliant to USB standard. Isochronous endpoints implement double-buffering to ease application software development, using both 'transmission' and 'reception' packet memory areas to manage buffer swapping on each successful transaction in order to have always a complete buffer to be used by the application, while the USB Peripheral fills the other.

The memory buffer which is currently used by the USB Peripheral is defined by the DTOG bit related to the endpoint direction (DTOG_RX for 'reception' isochronous endpoints, DTOG_TX for 'transmission' isochronous endpoints, both in the related USB_EPnR register) according to *Table 77*.

Table 77. Isochronous memory buffers usage

Endpoint Type	DTOG bit value	DMA buffer used by USB Peripheral	DMA buffer used by application software
IN	0	ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.	ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.
	1	ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.	ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.
OUT	0	ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.	ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.
	1	ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.	ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.

As it happens with double-buffered bulk endpoints, the USB_EPnR registers used to implement Isochronous endpoints are forced to be used as uni-directional ones. In case it is required to have Isochronous endpoints enabled both for reception and transmission, two USB_EPnR registers must be used.

The application software is responsible for the DTOG bit initialization according to the first buffer to be used; this has to be done considering the special toggle-only property that these two bits have. At the end of each transaction, the CTR_RX or CTR_TX bit of the addressed endpoint USB_EPnR register is set, depending on the enabled direction. At the same time, the affected DTOG bit in the USB_EPnR register is hardware toggled making buffer swapping completely software independent. STAT bit pair is not affected by transaction completion; since no flow control is possible for Isochronous transfers due to the lack of handshake phase, the endpoint remains always ‘11’ (Valid). CRC errors or buffer-overrun conditions occurring during Isochronous OUT transfers are anyway considered as correct transactions and they always trigger an CTR_RX event. However, CRC errors will anyway set the ERR bit in the USB_ISTR register to notify the software of the possible data corruption.

16.5.5 Suspend/Resume events

The USB standard defines a special peripheral state, called SUSPEND, in which the average current drawn from the USB bus must not be greater than 500 µA. This requirement is of fundamental importance for bus-powered devices, while self-powered devices are not required to comply to this strict power consumption constraint. In suspend mode, the host PC sends the notification to not send any traffic on the USB bus for more than 3mS: since a SOF packet must be sent every mS during normal operations, the USB Peripheral detects the lack of 3 consecutive SOF packets as a suspend request from the host PC and set the SUSP bit to ‘1’ in USB_ISTR register, causing an interrupt if enabled. Once the device is suspended, its normal operation can be restored by a so called RESUME sequence, which can be started from the host PC or directly from the peripheral itself, but it is always terminated by the host PC. The suspended USB Peripheral must be anyway able to detect a RESET sequence, reacting to this event as a normal USB reset event.

The actual procedure used to suspend the USB peripheral is device dependent since according to the device composition, different actions may be required to reduce the total consumption.

A brief description of a typical suspend procedure is provided below, focused on the USB-related aspects of the application software routine responding to the SUSP notification of the USB Peripheral:

1. Set the FSUSP bit in the USB_CNTR register to 1. This action activates the suspend mode within the USB Peripheral. As soon as the suspend mode is activated, the check on SOF reception is disabled to avoid any further SUSP interrupts being issued while the USB is suspended.
2. Remove or reduce any static power consumption in blocks different from the USB Peripheral.
3. Set LP_MODE bit in USB_CNTR register to 1 to remove static power consumption in the analog USB transceivers but keeping them able to detect resume activity.
4. Optionally turn off external oscillator and device PLL to stop any activity inside the device.

When an USB event occurs while the device is in SUSPEND mode, the RESUME procedure must be invoked to restore nominal clocks and regain normal USB behaviour. Particular care must be taken to insure that this process does not take more than 10mS when the wakening event is an USB reset sequence (See “Universal Serial Bus Specification” for more details). The start of a resume or reset sequence, while the USB Peripheral is suspended, clears the LP_MODE bit in USB_CNTR register asynchronously. Even if this event can trigger an WKUP interrupt if enabled, the use of an interrupt response routine must be carefully evaluated because of the long latency due to system clock restart; to have the shorter latency before re-activating the nominal clock it is suggested to put the resume procedure just after the end of the suspend one, so its code is immediately executed as soon as the system clock restarts. To prevent ESD discharges or any other kind of noise from waking-up the system (the exit from suspend mode is an asynchronous event), a suitable analog filter on data line status is activated during suspend; the filter width is about 70ns.

The following is a list of actions a resume procedure should address:

1. Optionally turn on external oscillator and/or device PLL.
2. Clear FSUSP bit of USB_CNTR register.
3. If the resume triggering event has to be identified, bits RXDP and RXDM in the USB_FNR register can be used according to *Table 78*, which also lists the intended software action in all the cases. If required, the end of resume or reset sequence can be detected monitoring the status of the above mentioned bits by checking when they reach the “10” configuration, which represent the Idle bus state; moreover at the end of a reset sequence the RESET bit in USB_ISTR register is set to 1, issuing an interrupt if enabled, which should be handled as usual.

Table 78. Resume event detection

[RXDP,RXDM] Status	Wake-up event	Required resume software action
“00”	Root reset	None
“10”	None (noise on bus)	Go back in Suspend mode
“01”	Root resume	None
“11”	Not Allowed (noise on bus)	Go back in Suspend mode

A device may require to exit from suspend mode as an answer to particular events not directly related to the USB protocol (e.g. a mouse movement wakes up the whole system). In this case, the resume sequence can be started by setting the RESUME bit in the USB_CNTR register to ‘1’ and resetting it to 0 after an interval between 1mS and 15mS (this interval can be timed using ESOF interrupts, occurring with a 1mS period when the system clock is running at nominal frequency). Once the RESUME bit is clear, the resume sequence will be completed by the host PC and its end can be monitored again using the RXDP and RXDM bits in the USB_FNR register.

Note: The RESUME bit must be anyway used only after the USB Peripheral has been put in suspend mode, setting the FSUSP bit in USB_CNTR register to 1.

16.6 Register description

The USB Peripheral registers can be divided into the following groups:

- Common Registers: Interrupt and Control registers
- Endpoint Registers: Endpoint configuration and status
- Buffer Descriptor Table: Location of packet memory used to locate data buffers

All register addresses are expressed as offsets with respect to the USB Peripheral registers base address 0xC000 8000, except the buffer descriptor table locations, which starts at the address specified by the USB_BTABLE register. Due to the common limitation of APB bridges on word addressability, all register addresses are aligned to 32-bit word boundaries although they are 16-bit wide. The same address alignment is used to access packet buffer memory locations, which are located starting from 0xC000 8800. In this section, the following abbreviations are used:

Read/write (rw)	The software can read and write to these bits.
Read-only (r)	The software can only read these bits.
Write-only (w)	The software can only write to these bits.
Read-clear (rc)	The software can only read or clear this bit.
Toggle (t)	The software can only toggle this bit by writing ‘1’. Writing ‘0’ has no effect.

16.6.1 Common registers

These registers affect the general behaviour of the USB Peripheral defining operating mode, interrupt handling, device address and giving access to the current frame number updated by the host PC.

USB control register (USB_CNTR)

Address Offset: 40h

Reset Value: 0000 0000 0000 0011 (0003h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTRM	DOVRM	ERRM	WKUPM	SUSPM	RESETM	SOFM	ESOFM	Reserved			RESUME	FSUSP	LPMODE	PDWN	FRES
rw	rw	rw	rw	rw	rw	rw	rw	-	-	-	rw	rw	rw	rw	rw

Bit 15	**CTRM:** *Correct Transfer Interrupt Mask* 0: Correct Transfer (CTR) Interrupt disabled. 1: CTR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.
Bit 14	**DOVRM:** *DMA over / underrun Interrupt Mask* 0: DOVR Interrupt disabled. 1: DOVR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 13	**ERRM:** *Error Interrupt Mask* 0: ERR Interrupt disabled. 1: ERR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.
Bit 12	**WKUPM:** *Wake-up Interrupt Mask* 0: WKUP Interrupt disabled. 1: WKUP Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.
Bit 11	**SUSPM:** *Suspend mode Interrupt Mask* 0: Suspend Mode Request (SUSP) Interrupt disabled. 1: SUSP Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.
Bit 10	**RESETM:** *USB Reset Interrupt Mask* 0: RESET Interrupt disabled. 1: RESET Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.
Bit 9	**SOFM:** *Start Of Frame Interrupt Mask* 0: SOF Interrupt disabled. 1: SOF Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.
Bit 8	**ESOFM:** *Expected Start Of Frame Interrupt Mask* 0: Expected Start of Frame (ESOF) Interrupt disabled. 1: ESOF Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.
Bits 7:5	Reserved, forced by hardware to 0.
Bit 4	**RESUME:** *Resume request* The microcontroller can set this bit to send a Resume signal to the host. It must be activated, according to USB specifications, for no less than 1mS and no more than 15mS after which the Host PC is ready to drive the resume sequence up to its end.
Bit 3	**FSUSP:** *Force suspend* Software must set this bit when the SUSP interrupt is received, which is issued when no traffic is received by the USB Peripheral for 3 mS. 0: No effect. 1: Enter suspend mode. Clocks and static power dissipation in the analog transceiver are left unaffected. If suspend power consumption is a requirement (bus-powered device), the application software should set the LP_MODE bit after FSUSP as explained below.
Bit 2	**LP_MODE:** *Low-power mode* This mode is used when the suspend-mode power constraints require that all static power dissipation is avoided, except the one required to supply the external pull-up resistor. This condition should be entered when the application is ready to stop all system clocks, or reduce their frequency in order to meet the power consumption requirements of the USB suspend condition. The USB activity during the suspend mode (WKUP event) asynchronously resets this bit (it can also be reset by software). 0: No Low Power Mode. 1: Enter Low Power mode.

Bit 1	**PDWN:** *Power down* This bit is used to completely switch off all USB-related analog parts if it is required to completely disable the USB Peripheral for any reason. When this bit is set, the USB Peripheral is disconnected from the transceivers and it cannot be used. 0: Exit Power Down. 1: Enter Power down mode.
Bit 0	**FRES:** *Force USB Reset* 0: Clear USB reset. 1: Force a reset of the USB Peripheral, exactly like a RESET signalling on the USB. The USB Peripheral is held in RESET state until software clears this bit. A “USB-RESET” interrupt is generated, if enabled.

USB interrupt status register (USB_ISTR)

Address Offset: 44h

Reset Value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR	DOVR	ERR	WKUP	SUSP	RESET	SOF	ESOF	Reserved			DIR	EP_ID[3:0]			
r	rc	rc	rc	rc	rc	rc	rc	-	-	-	r	r	r	r	r

This register contains the status of all the interrupt sources allowing application software to determine, which events caused an interrupt request.

The upper part of this register contains single bits, each of them representing a specific event. These bits are set by the hardware when the related event occurs; if the corresponding bit in the USB_CNTR register is set, a generic interrupt request is generated. The interrupt routine, examining each bit, will perform all necessary actions, and finally it will clear the serviced bits. If any of them is not cleared, the interrupt is considered to be still pending, and the interrupt line will be kept high again. If several bits are set simultaneously, only a single interrupt will be generated.

Endpoint transaction completion can be handled in a different way to reduce interrupt response latency. The CTR bit is set by the hardware as soon as an endpoint successfully completes a transaction, generating a generic interrupt request if the corresponding bit in USB_CNTR is set. An endpoint dedicated interrupt condition is activated independently from the CTRM bit in the USB_CNTR register. Both interrupt conditions remain active until software clears the pending bit in the corresponding USB_EPnR register (the CTR bit is actually a read only bit). The USB Peripheral has two interrupt request lines:

- Higher priority USB IRQ: The pending requests for endpoints, which have transactions with a higher priority (isochronous and double-buffered bulk) and they cannot be masked.
- Lower priority USB IRQ: All other interrupt conditions, which can either be non-maskable pending requests related to the lower priority transactions and all other maskable events flagged by the USB_ISTR high bytes.

For endpoint-related interrupts, the software can use the Direction of Transaction (DIR) and EP_ID read-only bits to identify, which endpoint made the last interrupt request and called the corresponding interrupt service routine.

The user can choose the relative priority of simultaneously pending USB_ISTR events by specifying the order in which software checks USB_ISTR bits in an interrupt service routine.

Only the bits related to events, which are serviced, are cleared. At the end of the service routine, another interrupt will be requested, to service the remaining conditions.

To avoid spurious clearing of some bits, it is recommended to clear them with a load instruction where all bits which must not be altered are written with 1, and all bits to be cleared are written with ‘0’ (these bits can only be cleared by software). Read-modify-write cycles should be avoided because between the read and the write operations another bit could be set by the hardware and the next write will clear it before the microprocessor has the time to serve the event.

The following describes each bit in detail:

Bit 15	**CTR:** *Correct Transfer* This bit is set by the hardware to indicate that an endpoint has successfully completed a transaction; using DIR and EP_ID bits software can determine which endpoint requested the interrupt. This bit is read-only.
Bit 14	**DOVR:** *DMA over / underrun* This bit is set if the microcontroller has not been able to respond in time to an USB memory request. The USB Peripheral handles this event in the following way: During reception an ACK handshake packet is not sent, during transmission a bit-stuff error is forced on the transmitted stream; in both cases the host will retry the transaction. The DOVR interrupt should never occur during normal operations. Since the failed transaction is retried by the host, the application software has the chance to speed-up device operations during this interrupt handling, to be ready for the next transaction retry; however this does not happen during Isochronous transfers (no isochronous transaction is anyway retried) leading to a loss of data in this case. This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.
Bit 13	**ERR:** *Error* This flag is set whenever one of the errors listed below has occurred: NANS: No ANSwer. The timeout for a host response has expired. CRC: Cyclic Redundancy Check error. One of the received CRCs, either in the token or in the data, was wrong. BST: Bit Stuffing error. A bit stuffing error was detected anywhere in the PID, data, and/or CRC. FVIO: Framing format Violation. A non-standard frame was received (EOP not in the right place, wrong token sequence, etc.). The USB software can usually ignore errors, since the USB Peripheral and the PC host manage retransmission in case of errors in a fully transparent way. This interrupt can be useful during the software development phase, or to monitor the quality of transmission over the USB bus, to flag possible problems to the user (e.g. loose connector, too noisy environment, broken conductor in the USB cable and so on). This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.
Bit 12	**WKUP:** *Wake up* This bit is set to 1 by the hardware when, during suspend mode, activity is detected that wakes up the USB Peripheral. This event asynchronously clears the LP_MODE bit in the CTLR register and activates the USB_WAKEUP line, which can be used to notify the rest of the device (e.g. wake-up unit) about the start of the resume process. This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.

Bit 11	**SUSP:** *Suspend mode request* This bit is set by the hardware when no traffic has been received for 3mS, indicating a suspend mode request from the USB bus. The suspend condition check is enabled immediately after any USB reset and it is disabled by the hardware when the suspend mode is active (FSUSP=1) until the end of resume sequence. This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.
Bit 10	**RESET:** *USB RESET request* Set when the USB Peripheral detects an active USB RESET signal at its inputs. The USB Peripheral, in response to a RESET, just resets its internal protocol state machine, generating an interrupt if RESETM enable bit in the USB_CNTR register is set. Reception and transmission are disabled until the RESET bit is cleared. All configuration registers do not reset: the microcontroller must explicitly clear these registers (this is to ensure that the RESET interrupt can be safely delivered, and any transaction immediately followed by a RESET can be completed). The function address and endpoint registers are reset by an USB reset event. This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.
Bit 9	**SOF:** *Start Of Frame* This bit signals the beginning of a new USB frame and it is set when a SOF packet arrives through the USB bus. The interrupt service routine may monitor the SOF events to have a 1mS synchronization event to the USB host and to safely read the USB_FNR register which is updated at the SOF packet reception (this could be useful for isochronous applications). This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.
Bit 8	**ESOF:** *Expected Start Of Frame* This bit is set by the hardware when an SOF packet is expected but not received. The host sends an SOF packet each mS, but if the hub does not receive it properly, the Suspend Timer issues this interrupt. If three consecutive ESOF interrupts are generated (i.e. three SOF packets are lost) without any traffic occurring in between, a SUSP interrupt is generated. This bit is set even when the missing SOF packets occur while the Suspend Timer is not yet locked. This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.
Bits 7:5	Reserved, forced by hardware to 0.

Bit 4	**DIR:** *Direction of transaction.* This bit is written by the hardware according to the direction of the successful transaction, which generated the interrupt request. If DIR bit=0, CTR_TX bit is set in the USB_EPnR register related to the interrupting endpoint. The interrupting transaction is of IN type (data transmitted by the USB Peripheral to the host PC). If DIR bit=1, CTR_RX bit or both CTR_TX/CTR_RX are set in the USB_EPnR register related to the interrupting endpoint. The interrupting transaction is of OUT type (data received by the USB Peripheral from the host PC) or two pending transactions are waiting to be processed. This information can be used by the application software to access the USB_EPnR bits related to the triggering transaction since it represents the direction having the interrupt pending. This bit is read-only.
Bits 3:0	**EP_ID[3:0]:** *Endpoint Identifier.* These bits are written by the hardware according to the endpoint number, which generated the interrupt request. If several endpoint transactions are pending, the hardware writes the endpoint identifier related to the endpoint having the highest priority defined in the following way: Two endpoint sets are defined, in order of priority: Isochronous and double-buffered bulk endpoints are considered first and then the other endpoints are examined. If more than one endpoint from the same set is requesting an interrupt, the EP_ID bits in USB_ISTR register are assigned according to the lowest requesting endpoint register, EP0R having the highest priority followed by EP1R and so on. The application software can assign a register to each endpoint according to this priority scheme, so as to order the concurring endpoint requests in a suitable way. These bits are read only.

USB frame number register (USB_FNR)

Address Offset: 48h

Reset Value: 0000 0xxx xxxx xxxx (0xxxh)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDP	RXDM	LCK	LSOF[1:0]		FN[10:0]										
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bit 15	**RXDP:** *Receive Data + Line Status* This bit can be used to observe the status of received data plus upstream port data line. It can be used during end-of-suspend routines to help determining the wake-up event.
Bit 14	**RXDM:** *Receive Data - Line Status* This bit can be used to observe the status of received data minus upstream port data line. It can be used during end-of-suspend routines to help determining the wake-up event.
Bit 13	**LCK:** *Locked* This bit is set by the hardware when at least two consecutive SOF packets have been received after the end of an USB reset condition or after the end of an USB resume sequence. Once locked, the frame timer remains in this state until an USB reset or USB suspend event occurs.
Bits12:11	**LSOF[1:0]:** *Lost SOF* These bits are written by the hardware when an ESOF interrupt is generated, counting the number of consecutive SOF packets lost. At the reception of an SOF packet, these bits are cleared.
Bits 10:0	**FN[10:0]:** *Frame Number* This bit field contains the 11-bits frame number contained in the last received SOF packet. The frame number is incremented for every frame sent by the host and it is useful for Isochronous transfers. This bit field is updated on the generation of an SOF interrupt.

USB device address (USB_DADDR)

Address Offset: 4Ch

Reset Value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								EF	ADD[6:0]						
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8	Reserved, forced by hardware to 0.
Bit 7	**EF:** *Enable Function* This bit is set by the software to enable the USB device. The address of this device is contained in the following ADD[6:0] bits. If this bit is at ‘0’ no transactions are handled, irrespective of the settings of USB_EPnR registers.
Bits 6:0	**ADD[6:0]:** *Device Address* These bits contain the USB function address assigned by the host PC during the enumeration process. Both this field and the Endpoint Address (EA) field in the associated USB_EPnR register must match with the information contained in a USB token in order to handle a transaction to the required endpoint.

Buffer table address (USB_BTABLE)

Address Offset: 50h

Reset Value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BTABLE[15:3]													Reserved		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	-	-	-

Bits 15:3	**BTABLE[15:3]:** *Buffer Table.* These bits contain the start address of the buffer allocation table inside the dedicated packet memory. This table describes each endpoint buffer location and size and it must be aligned to an 8 byte boundary (the 3 least significant bits are always ‘0’). At the beginning of every transaction addressed to this device, the USP peripheral reads the element of this table related to the addressed endpoint, to get its buffer start location and the buffer size (Refer to *Section : Structure and usage of packet buffers*).
Bits 2:0	Reserved, forced by hardware to 0.

16.6.2 Endpoint-specific registers

The number of these registers varies according to the number of endpoints that the USB Peripheral is designed to handle. The USB Peripheral supports up to 8 bi-directional endpoints. Each USB device must support a control endpoint whose address (EA bits) must be set to 0. The USB Peripheral behaves in an undefined way if multiple endpoints are enabled having the same endpoint number value. For each endpoint, an USB_EPnR register is available to store the endpoint specific information.

USB endpoint n register (USB_EPnR), n=[0..7]

Address Offset: 00h to 2Ch

Reset value: 0000 0000 0000 0000b (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR_ RX	DTOG _RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_ KIND	CTR TX	DTOG_ TX	STAT TX[1:0]		EA[3:0]			
r-c	t	t	t	r	rw	rw	rw	r-c	t	t	t	rw	rw	rw	rw

They are also reset when an USB reset is received from the USB bus or forced through bit FRES in the CTLR register, except the CTR_RX and CTR_TX bits, which are kept unchanged to avoid missing a correct packet notification immediately followed by an USB reset event. Each endpoint has its USB_EPnR register where *n* is the endpoint identifier.

Read-modify-write cycles on these registers should be avoided because between the read and the write operations some bits could be set by the hardware and the next write would modify them before the CPU has the time to detect the change. For this purpose, all bits affected by this problem have an ‘invariant’ value that must be used whenever their modification is not required. It is recommended to modify these registers with a load instruction where all the bits, which can be modified only by the hardware, are written with their ‘invariant’ value.

Bit 15	**CTR_RX:** *Correct Transfer for reception* This bit is set by the hardware when an OUT/SETUP transaction is successfully completed on this endpoint; the software can only clear this bit. If the CTRM bit in USB_CNTR register is set accordingly, a generic interrupt condition is generated together with the endpoint related interrupt condition, which is always activated. The type of occurred transaction, OUT or SETUP, can be determined from the SETUP bit described below. A transaction ended with a NAK or STALL handshake does not set this bit, since no data is actually transferred, as in the case of protocol errors or data toggle mismatches. This bit is read/write but only ‘0’ can be written, writing 1 has no effect.
Bit 14	**DTOG_RX:** *Data Toggle, for reception transfers* If the endpoint is not Isochronous, this bit contains the expected value of the data toggle bit (0=DATA0, 1=DATA1) for the next data packet to be received. Hardware toggles this bit, when the ACK handshake is sent to the USB host, following a data packet reception having a matching data PID value; if the endpoint is defined as a control one, hardware clears this bit at the reception of a SETUP PID addressed to this endpoint. If the endpoint is using the double-buffering feature this bit is used to support packet buffer swapping too (Refer to *Section 16.5.3: Double-buffered endpoints*). If the endpoint is Isochronous, this bit is used only to support packet buffer swapping since no data toggling is used for this sort of endpoints and only DATA0 packet are transmitted (Refer to *Section 16.5.4: Isochronous transfers*). Hardware toggles this bit just after the end of data packet reception, since no handshake is used for isochronous transfers. This bit can also be toggled by the software to initialize its value (mandatory when the endpoint is not a control one) or to force specific data toggle/packet buffer usage. When the application software writes ‘0’, the value of DTOG_RX remains unchanged, while writing ‘1’ makes the bit value toggle. This bit is read/write but it can be only toggled by writing 1.
Bits 13:12	**STAT_RX [1:0]:** *Status bits, for reception transfers* These bits contain information about the endpoint status, which are listed in *Table 79: Reception status encoding on page 403*.These bits can be toggled by software to initialize their value. When the application software writes ‘0’, the value remains unchanged, while writing ‘1’ makes the bit value toggle. Hardware sets the STAT_RX bits to NAK when a correct transfer has occurred (CTR_RX=1) corresponding to a OUT or SETUP (control only) transaction addressed to this endpoint, so the software has the time to elaborate the received data before it acknowledge a new transaction Double-buffered bulk endpoints implement a special transaction flow control, which control the status based upon buffer availability condition (Refer to *Section 16.5.3: Double-buffered endpoints*). If the endpoint is defined as Isochronous, its status can be only “VALID” or “DISABLED”, so that the hardware cannot change the status of the endpoint after a successful transaction. If the software sets the STAT_RX bits to ‘STALL’ or ‘NAK’ for an Isochronous endpoint, the USB Peripheral behaviour is not defined. These bits are read/write but they can be only toggled by writing ‘1’.
Bit 11	**SETUP:** *Setup transaction completed* This bit is read-only and it is set by the hardware when the last completed transaction is a SETUP. This bit changes its value only for control endpoints. It must be examined, in the case of a successful receive transaction (CTR_RX event), to determine the type of transaction occurred. To protect the interrupt service routine from the changes in SETUP bits due to next incoming tokens, this bit is kept frozen while CTR_RX bit is at 1; its state changes when CTR_RX is at 0. This bit is read-only.

Bits 10:9	**EP_TYPE[1:0]:** *Endpoint type* These bits configure the behaviour of this endpoint as described in *Table 80: Endpoint type encoding on page 404*. Endpoint 0 must always be a control endpoint and each USB function must have at least one control endpoint which has address 0, but there may be other control endpoints if required. Only control endpoints handle SETUP transactions, which are ignored by endpoints of other kinds. SETUP transactions cannot be answered with NAK or STALL. If a control endpoint is defined as NAK, the USB Peripheral will not answer, simulating a receive error, in the receive direction when a SETUP transaction is received. If the control endpoint is defined as STALL in the receive direction, then the SETUP packet will be accepted anyway, transferring data and issuing the CTR interrupt. The reception of OUT transactions is handled in the normal way, even if the endpoint is a control one. Bulk and interrupt endpoints have very similar behaviour and they differ only in the special feature available using the EP_KIND configuration bit. The usage of Isochronous endpoints is explained in *Section 16.5.4: Isochronous transfers*
Bit 8	**EP_KIND:** *Endpoint Kind* The meaning of this bit depends on the endpoint type configured by the EP_TYPE bits. *Table 81* summarizes the different meanings. DBL_BUF: This bit is set by the software to enable the double-buffering feature for this bulk endpoint. The usage of double-buffered bulk endpoints is explained in *Section 16.5.3: Double-buffered endpoints*. STATUS_OUT: This bit is set by the software to indicate that a status out transaction is expected: in this case all OUT transactions containing more than zero data bytes are answered ‘STALL’ instead of ‘ACK’. This bit may be used to improve the robustness of the application to protocol errors during control transfers and its usage is intended for control endpoints only. When STATUS_OUT is reset, OUT transactions can have any number of bytes, as required.
Bit 7	**CTR_TX:** *Correct Transfer for transmission* This bit is set by the hardware when an IN transaction is successfully completed on this endpoint; the software can only clear this bit. If the CTRM bit in the USB_CNTR register is set accordingly, a generic interrupt condition is generated together with the endpoint related interrupt condition, which is always activated. A transaction ended with a NAK or STALL handshake does not set this bit, since no data is actually transferred, as in the case of protocol errors or data toggle mismatches. This bit is read/write but only ‘0’ can be written.

Bit 6	**DTOG_TX:** *Data Toggle, for transmission transfers* If the endpoint is non-isochronous, this bit contains the required value of the data toggle bit (0=DATA0, 1=DATA1) for the next data packet to be transmitted. Hardware toggles this bit when the ACK handshake is received from the USB host, following a data packet transmission. If the endpoint is defined as a control one, hardware sets this bit to 1 at the reception of a SETUP PID addressed to this endpoint. If the endpoint is using the double buffer feature, this bit is used to support packet buffer swapping too (Refer to *Section 16.5.3: Double-buffered endpoints*) If the endpoint is Isochronous, this bit is used to support packet buffer swapping since no data toggling is used for this sort of endpoints and only DATA0 packet are transmitted (Refer to *Section 16.5.4: Isochronous transfers*). Hardware toggles this bit just after the end of data packet transmission, since no handshake is used for Isochronous transfers. This bit can also be toggled by the software to initialize its value (mandatory when the endpoint is not a control one) or to force a specific data toggle/packet buffer usage. When the application software writes ‘0’, the value of DTOG_TX remains unchanged, while writing ‘1’ makes the bit value toggle. This bit is read/write but it can only be toggled by writing 1.
Bit 5:4	**STAT_TX [1:0]:** *Status bits, for transmission transfers* These bits contain the information about the endpoint status, listed in *Table 82*. These bits can be toggled by the software to initialize their value. When the application software writes ‘0’, the value remains unchanged, while writing ‘1’ makes the bit value toggle. Hardware sets the STAT_TX bits to NAK, when a correct transfer has occurred (CTR_TX=1) corresponding to a IN or SETUP (control only) transaction addressed to this endpoint. It then waits for the software to prepare the next set of data to be transmitted. Double-buffered bulk endpoints implement a special transaction flow control, which controls the status based on buffer availability condition (Refer to *Section 16.5.3: Double-buffered endpoints*). If the endpoint is defined as Isochronous, its status can only be “VALID” or “DISABLED”. Therefore, the hardware cannot change the status of the endpoint after a successful transaction. If the software sets the STAT_TX bits to ‘STALL’ or ‘NAK’ for an Isochronous endpoint, the USB Peripheral behaviour is not defined. These bits are read/write but they can be only toggled by writing ‘1’.
Bit 3:0	**EA[3:0]:** *Endpoint Address.* Software must write in this field the 4-bit address used to identify the transactions directed to this endpoint. A value must be written before enabling the corresponding endpoint.

Table 79. Reception status encoding

STAT_RX[1:0]	Meaning
00	**DISABLED:** all reception requests addressed to this endpoint are ignored.
01	**STALL**: the endpoint is stalled and all reception requests result in a STALL handshake.
10	**NAK**: the endpoint is naked and all reception requests result in a NAK handshake.
11	**VALID**: this endpoint is enabled for reception.

Table 80. **Endpoint type encoding**

EP_TYPE[1:0]	Meaning
00	BULK
01	CONTROL
10	ISO
11	INTERRUPT

Table 81. **Endpoint kind meaning**

EP_TYPE[1:0]		EP_KIND Meaning
00	BULK	DBL_BUF
01	CONTROL	STATUS_OUT
10	ISO	Not used
11	INTERRUPT	Not used

Table 82. **Transmission status encoding**

STAT_TX[1:0]	Meaning
00	**DISABLED:** all transmission requests addressed to this endpoint are ignored.
01	**STALL**: the endpoint is stalled and all transmission requests result in a STALL handshake.
10	**NAK**: the endpoint is naked and all transmission requests result in a NAK handshake.
11	**VALID**: this endpoint is enabled for transmission.

16.6.3 Buffer descriptor table

Although this table is located inside packet buffer memory, its entries can be considered as additional registers used to configure the location and size of packet buffers used to exchange data between USB macrocell and the STR750. Due to the common APB bridge limitation on word addressability, all packet memory locations are accessed by the APB using 32-bit aligned addresses, instead of the actual memory location addresses utilized by the USB Peripheral for the USB_BTABLE register and buffer description table locations. In the following pages two location addresses are reported: the one to be used by application software while accessing the packet memory, and the local one relative to USB Peripheral access. To obtain the correct STR750 memory address value to be used in the application software while accessing the packet memory, the actual memory location address must be multiplied by two. The first packet memory location is located at 0xC000 8800.

The buffer description table entry associated with the USB_EPnR registers is described below. A thorough explanation of packet buffers and buffer descriptor table usage can be found in the *Section : Structure and usage of packet buffers*.

Transmission buffer address n (USB_ADDRn_TX)

Address Offset: [USB_BTABLE] + n*16

USB local Address: [USB_BTABLE] + n*8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDRn_TX[15:1]															-
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	-

Bits 15:1	**ADDRn_TX[15:1]:** *Transmission Buffer Address* These bits point to the starting address of the packet buffer containing data to be transmitted by the endpoint associated with the USB_EPnR register at the next IN token addressed to it.
Bit 0	Must always be written as ‘0’ since packet memory is word-wide and all packet buffers must be word-aligned.

Transmission byte count n (USB_COUNTn_TX)

Address Offset: [USB_BTABLE] + n*16 + 4

USB local Address: [USB_BTABLE] + n*8 + 2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-						COUNTn_TX[9:0]									
-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10	These bits are not used since packet size is limited by USB specifications to 1023 bytes. Their value is not considered by the USB Peripheral.
Bits 9:0	**COUNTn_TX[9:0]:** *Transmission Byte Count* These bits contain the number of bytes to be transmitted by the endpoint associated with the USB_EPnR register at the next IN token addressed to it.

Note: Double-buffered and Isochronous IN Endpoints have two USB_COUNTn_TX registers: named USB_COUNTn_TX_1 and USB_COUNTn_TX_0 with the following content

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
-						COUNTn_TX_1[9:0]									
-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
-						COUNTn_TX_0[9:0]									
-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Reception buffer address n (USB_ADDRn_RX)

Address Offset: [USB_BTABLE] + n*16 + 8

USB local Address: [USB_BTABLE] + n*8 + 4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDRn_RX[15:1]															-
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	-

Bits 15:1	**ADDRn_RX[15:1]:** *Reception Buffer Address* These bits point to the starting address of the packet buffer, which will contain the data received by the endpoint associated with the USB_EPnR register at the next OUT/SETUP token addressed to it.
Bit 0	This bit must always be written as ‘0’ since packet memory is word-wide and all packet buffers must be word-aligned.

Reception byte count n (USB_COUNTn_RX)

Address Offset: [USB_BTABLE] + n*16 + 12

USB local Address: [USB_BTABLE] + n*8 + 6

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BLSIZE	NUM_BLOCK[4:0]					COUNTn_RX[9:0]									
rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	r	r

This table location is used to store two different values, both required during packet reception. The most significant bits contains the definition of allocated buffer size, to allow buffer overflow detection, while the least significant part of this location is written back by the USB Peripheral at the end of reception to give the actual number of received bytes. Due to the restrictions on the number of available bits, buffer size is represented using the number of allocated memory blocks, where block size can be selected to choose the trade-off between fine-granularity/small-buffer and coarse-granularity/large-buffer. The size of allocated buffer is a part of the endpoint descriptor and it is normally defined during the enumeration process according to its maxPacketSize parameter value (See “Universal Serial Bus Specification”).

Bit 15	**BL_SIZE:** *BLock SIZE.* This bit selects the size of memory block used to define the allocated buffer area. – If BL_SIZE=0, the memory block is 2 `byte` large, which is the minimum block allowed in a word-wide memory. With this block size the allocated buffer size ranges from 2 to 62 `bytes`. – If BL_SIZE=1, the memory block is 32 `byte` large, which allows to reach the maximum packet length defined by USB specifications. With this block size the allocated buffer size ranges from 32 to 1024 bytes, which is the longest packet size allowed by USB standard specifications.
Bits 14:10	**NUM_BLOCK[4:0]:** *Number of blocks.* These bits define the number of memory blocks allocated to this packet buffer. The actual amount of allocated memory depends on the BL_SIZE value as illustrated in *Table 83*.
Bits 9:0	**COUNTn_RX[9:0]:** *Reception Byte Count* These bits contain the number of bytes received by the endpoint associated with the USB_EPnR register during the last OUT/SETUP transaction addressed to it.

Note: Double-buffered and Isochronous OUT Endpoints have two USB_COUNTn_RX registers: named USB_COUNTn_RX_1 and USB_COUNTn_RX_0 with the following content

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BLSIZE _1	NUM_BLOCK_1[4:0]					COUNTn_RX_1[9:0]									
rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BLSIZE _0	NUM_BLOCK_0[4:0]					COUNTn_RX_0[9:0]									
rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	r	r

Table 83. Definition of allocated buffer memory

Value of NUM_BLOCK[4:0]	Memory allocated when BL_SIZE=0	Memory allocated when BL_SIZE=1
0 (‘00000’)	Not allowed	32 bytes
1 (‘00001’)	2 bytes	64 bytes
2 (‘00010’)	4 bytes	96 bytes
3 (‘00011’)	6 bytes	128 bytes
...	...	...
15 (‘01111’)	30 bytes	512 bytes
16 (‘10000’)	32 bytes	544 bytes
17 (‘10001’)	34 bytes	576 bytes
18 (‘10010’)	36 bytes	608 bytes
...	...	...
30 (‘11110’)	60 bytes	992 bytes
31 (‘11111’)	62 bytes	1024 bytes

16.7 USB Register map

To be able to find the correct offset for each register, *Table 84* shows the mapping of all USB Peripheral registers.

Table 84. USB Peripheral register page mapping

Offset	Register	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USB_EP0R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x04	USB_EP1R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x08	USB_EP2R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x0C	USB_EP3R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x10	USB_EP4R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x14	USB_EP5R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x18	USB_EP6R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x1C	USB_EP7R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x40	USB_CNTR	CTRM	DOVRM	ERRM	WKUPM	SUSPM	RESETM	SOFM	ESOFM	Reserved			RESUME	FSUSP	LP MODE	PDWN	FRES
0x44	USB_ISTR	CTR	DOVR	ERR	WKUP	SUSP	RESET	SOF	ESOF	Reserved			DIR	EP_ID[3:0]			
0x48	USB_FNR	RXDP	RXDM	LCK	LSOF[1:0]		FN[10:0]										
0x4C	USB_DADDR	Reserved								EF	ADD[6:0]						
0x50	USB_BTABLE	BTABLE[15:3]													Reserved		

Refer to *Table 2* for the base address.

17 Analog-to-Digital Converter (ADC)

17.1 Main characteristics

- 3.75µs between 2 conversions at maximum ADC clock frequency (f_{CK_ADC}=8 MHz) comprising :
  - Sampling Time: 11 ADC clock cycles
  - Conversion Time : 19 ADC clock cycles
- ADC clock derives from PCLK through a 4-bit frequency prescaler
- Resolution: 10 bits
- Monotonicity: guaranteed
- No missing codes: guaranteed
- Zero Input reading: 0000h
- Full Scale reading: 03FFh
- 16 x 10-bit data registers (one register for each channel)
- One-Shot/Scan modes (possibility to convert a part or all of 16 channels successively without any software interaction)
- Start of conversion triggerable by timer TIM0 timer OC2 (in One-Shot or Scan mode)
- Chain Injection mode, triggerable by timer PWM timer TRGO
- Power Down mode
- Four selectable analog watchdog channels with interrupt generation (when their converted values are outside a window previously programmed by software)
- Automatic DMA request after the end of conversion. DMA transfers can be enabled by software or by hardware (using the DMA external enable trigger connected to the TIM2 timer OC2)

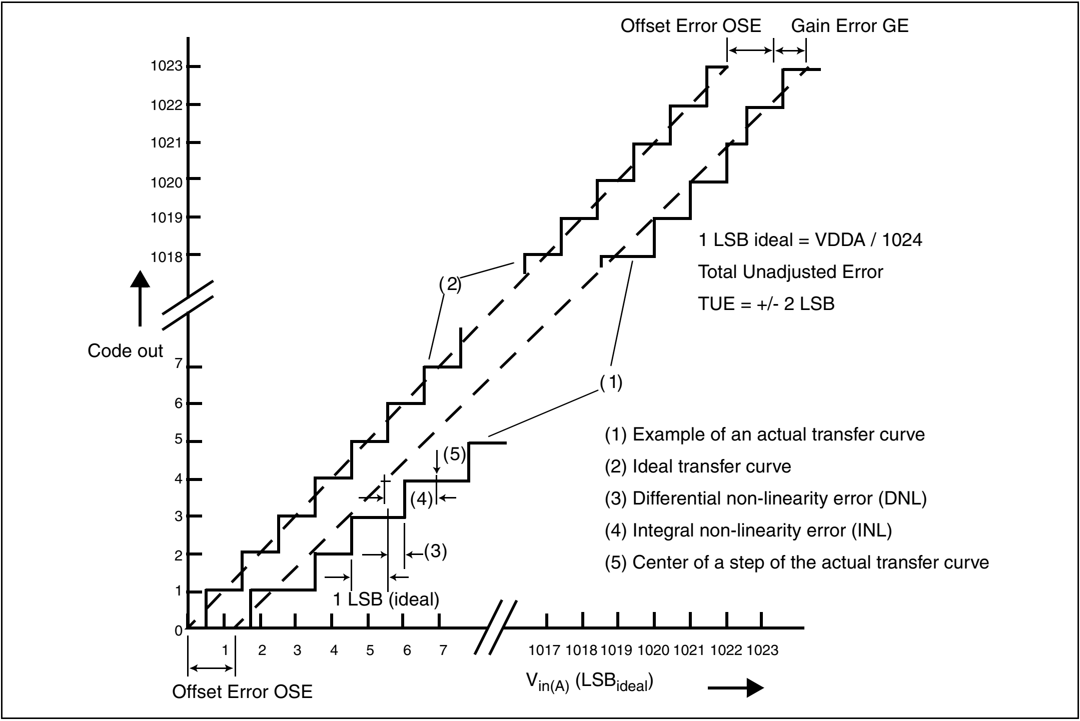
17.2 Introduction

The Analog-to -Digital Converter (ADC) comprises an input multiplexed channel selector feeding a successive approximation converter. The conversion resolution is 10 bits.

The conversion time depends on the ADC clock frequency and the prescaler factors stored in the ADC_CLR1 register. Two prescalers can be used to provide a slow clock during sampling and a fast clock during conversion.

You can increase the conversion time by modifying the prescaling factor, for example if you need to add an external resistor larger than 10 kΩ to the ADC input pin. The minimum conversion time specified in the STR750 datasheet includes the time required by the built-in Sample and Hold circuitry, which minimizes the need for external components and allows quick sampling of the signal to minimize warping and conversion errors.

Figure 115. A/D conversion characteristics



The converter uses a fully differential analog input configuration for the best noise immunity and precision performance. Two separate supply references are provided to ensure the best possible supply noise rejection. In fact, the converted digital value is referred to the analog reference voltage which determines the full-scale converted value. Of course, analog and digital grounds MUST be common (to be tied together externally).

Up to 16 multiplexer analog inputs are available. A group of signals can be converted sequentially by simply programming the starting address of the first analog channel to be converted and the number of channels to convert.

An analog watchdog is provided, allowing continuous hardware monitoring of four input channels. The comparison result is stored in a dedicated register.

Single and continuous conversion modes are available and a power-down programmable bit allows the ADC to be set in low-power idle mode.

17.3 Functional description

17.3.1 Start of calibration

To reach the target accuracy, it is mandatory to calibrate the ADC after each power-up or when resuming from STANDBY mode.

The internal calibration process can be started by setting the CAL bit in the ADC_CLR0 register. This bit remains at level 1 until calibration is over. It is not possible to stop calibration. If a conversion is started before calibration is finished, the effective conversion waits for the end of calibration.

The start of calibration is performed only if the PWDN bit in the ADC_CLR4 register is cleared.

17.3.2 Start of conversion

You can start the programmed conversion in one of three ways:

- If TRGEN (external trigger enable) is reset, the conversion chain starts when the START bit in the ADC_CLR0 register is set.
- If TRGEN is set and EDGLEV (edge/level selection) is reset, the conversion chain starts when the programmed edge is detected on the TIM0 OC2 output or the START bit in the ADC_CLR0 register is set.
- If TRGEN and EDGLEV are set, the conversion chain starts when the START bit in the ADC_CLR0 register is set and the programmed level on the TIM0 OC2 output is detected.

The start of conversion is performed only if the PWDN bit in the ADC_CLR4 register is cleared.

If you clear the START bit by writing 0, the ADC will finish the current chain conversion and clear the START bit when the operation is completed.

17.3.3 Operating modes

Two operating modes are available: One-shot Mode and Scan Mode. These modes are selected using the MODE bit in the Control Logic Register 2 (ADC_CLR2).

In **One-shot Mode** (MODE=0) a sequential conversion of the number of channels specified in the ADC_CLR2 register is performed once only. At the end of each conversion the digital result of the conversion is stored in the corresponding data register.

Clearing the START bit in the ADC_CLR0 register has no effect in One-shot Mode.

In **Scan Mode** (MODE=1) a sequential conversion of the number of channels specified in the ADC_CLR2 register is performed continuously. At the end of each conversion the digital result of the conversion is stored in the corresponding data register. To stop Scan Mode, clear the START bit in the ADC_CLR0 register. The ADC will clear the START bit after the last conversion in the current scan is completed.

You can convert a single channel by setting the number of input channels to zero.

In both modes, at the end of each conversion an End Of Conversion interrupt request is generated (if enabled by the corresponding mask bit). At the end of the sequence, an End Of Chain interrupt request is generated (if enabled by the corresponding mask bit).

17.3.4 Input channel selection

You can select the number of analog inputs to be converted by writing the number of the first channel to be converted (FCH[3:0] bits) and the total number of channels minus one (NCH[3:0] bits) in the ADC_CLR2 register. At the end of each conversion, the channel value is incremented and the number of channels to convert is decremented until the channel number is zero.

If FCH[3:0] or NCH[3:0] are programmed with a value greater than the maximum number of available channels the stored value is reset to zero.

If the sum of FCH[3:0] and NCH[3:0] is greater than the number of available channels then the conversion chain continues from channel zero after the last available channel is converted, in a circular conversion chain.

17.3.5 Timer start of conversion trigger/enable

A timer trigger/enable for the start of the chain conversion signal can be programmed using configuration bits in the ADC_CLR0 register. The timer trigger is generated by the TIM0 timer OC2 output.

If the TRGEN (trigger enable) bit in the ADC_CLR0 register is set, this feature is enabled and two options are available:

- If the EDGLEV bit in the ADC_CLR0 register is reset then a rising/falling edge (depending on the EDGE bit in the ADC_CLR0 register) detected on the TIM0 timer OC2 output sets the START bit in the ADC_CLR0 register and starts the programmed conversion. EDGE = 0 means falling edge; EDGE = 1 means rising edge.
- If the EDGLEV bit in the ADC_CLR0 register is set, the conversion is started if and only if the START bit in the ADC_CLR0 register is set and the TIM0 timer OC2 output is zero or one (depending on the EDGE bit in the ADC_CLR0 register). EDGE = 0 means that the start of conversion is enabled if the TIM0 timer OC2 output is low; if EDGE = 1 the start of conversion is enabled when the TIM0 timer OC2 output is high.

The enable bit is checked only when the conversion is started; the end and the restart of the conversion must be done by software.

Note: *The start of conversion trigger connected to the TIM0 timer OC2 output signal is internally resynchronized with PCLK. Consequently, it is mandatory that the user respects the minimum pulse duration of 2 APB clocks when programming the TIM0 timer to generate this trigger.*

17.3.6 Analog clock prescaler

Two prescalers can be used, one defining the main ADC clock frequency (CNVP), and another (SMPP) that can be optionally enabled to provide the ADC clock only during the sampling phase. This allows you to sample at low speed while converting at full speed.

The frequency of the sampling (or conversion) clock is configured independently using the SMPP[2:0] and CNVP[2:0] bits in the ADC_CLR1 register. The prescalers allow you to configure the ADC clock frequency based on PCLK with the following division factors: 1, 2, 4, 6, 8, 10, 12 or 14. Do not exceed the maximum ADC clock frequency of 8 MHz when programming the prescaler.

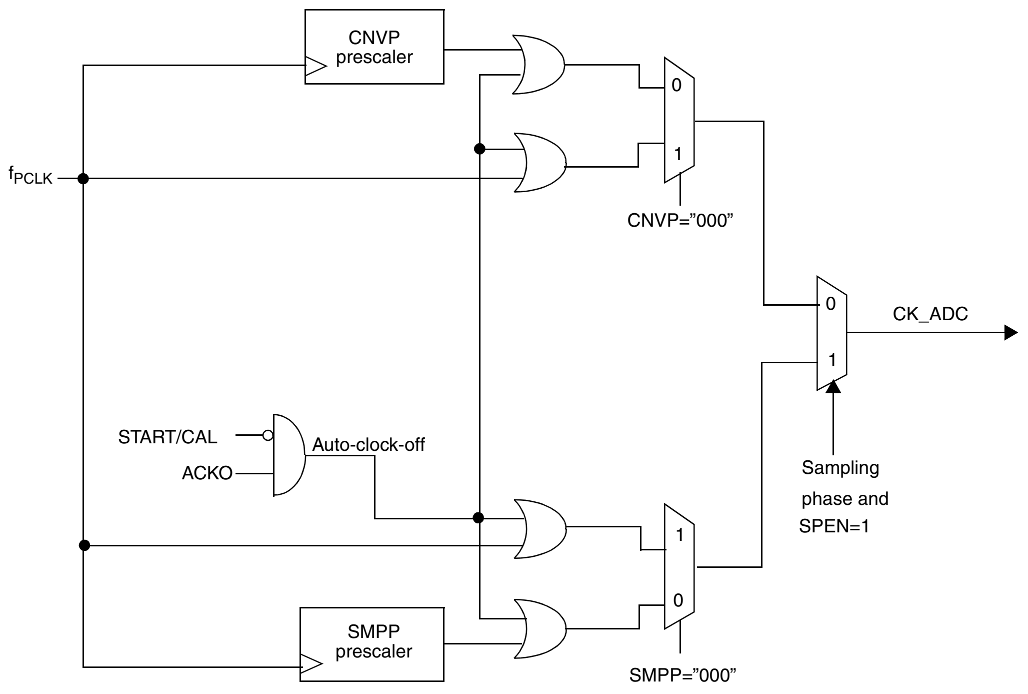
During calibration, the CNVP clock is used.

You can only change the prescalers when no conversion or calibration is ongoing. To ensure this, a write to these registers becomes effective only when the conversion/calibration start bits are inactive.

Setting the SPEN bit in the ADC_CLR1 register allows you to use different clock division factors for sampling and conversion as described above. When SPEN = 0, the same clock division factor, defined by the CNVP[2:0] bits in the ADC_CLR1 register, is used for both sampling and conversion.

A state machine memorizes the state of the ADC in order to select the right clock during the sampling and conversion phase (see *Figure 116 on page 414*).

Figure 116. ADC prescaler block diagram



CAL:	ADC_CLR0(1)
START:	ADC_CLR0(0)
ACKO:	ADC_CLR4(14)
SPEN:	ADC_CLR1(15)

Note: *If SPEN = 0, the sampling phase duration (t_S) and the conversion phase duration (T_C) are defined as stated below:*

$$t_S = 11 * t_{(PCLK)} * CNVP$$
$$t_C = 19 * t_{(PCLK)} * CNVP$$

If SPEN = 1, the sampling phase duration (t_S) and the conversion phase duration (T_C) are defined as stated below:

$$t_S = [11 * t_{(PCLK)} * SMPP] + t_{(PCLK)}$$
$$t_C = 19 * t_{(PCLK)} * CNVP$$

In both cases, the global ADC conversion time (t) is equal to:

$$t = t_S + t_C$$

Table 85. ADC Clock Prescaler programming

CNVP[2:0] / SMPP[2:0] Binary value	CNVP/SMPP Division Factor	f_{ADC}
000	1	f_{PCLK}
001	2	$f_{PCLK}/2$
010	4	$f_{PCLK}/4$
011	6	$f_{PCLK}/6$
100	8	$f_{PCLK}/8$
101	10	$f_{PCLK}/10$
110	12	$f_{PCLK}/12$
111	14	$f_{PCLK}/14$

17.3.7 Injected conversion chain

A conversion chain can be injected by a timer trigger or by setting the JSTART bit in the ADC_CLR3 register. The timer trigger is generated by the internal PWM Timer TRGO signal. This conversion can only be done in one-shot mode and can interrupt a normal conversion; when the injected conversion is finished, normal conversion restarts (if pending) from the first non-converted channel in the programmed chain.

Note: *It is forbidden to start an injected conversion during scan mode.*

At the end of each conversion an End Of Injected Conversion interrupt is issued (if enabled by the corresponding mask bit) and at the end of the sequence an End Of Injected Chain interrupt is issued (if enabled by the corresponding mask bit).

The first channel of the injected chain is programmed by the JFCH field in the ADC_CLR3 register. The number of converted channels is programmed by the JNCH field in the ADC_CLR3 register.

When you set the JSTART bit in the ADC_CLR3 register the current conversion is terminated and the injected chain is converted. At the end of the injected chain the JSTART bit is reset and normal conversion is resumed.

If the JTRGEN bit in the ADC_CLR3 register is set by software then a programmed event (rising/falling edge) on the PWM Timer TRGO output starts the injected conversion by setting the JSTART bit (see *Section 17.4.5: Control Logic Register 3 (ADC_CLR3) on page 421*).

Note: *The start of injected conversion trigger connected to the PWM timer TRGO output signal is internally resynchronized with PCLK. Consequently, it is mandatory that the user respects the minimum pulse duration of 2 APB clocks when programming the PWM timer to generate this trigger.*

17.3.8 Analog watchdogs

Four programmable watchdogs are available for analog threshold detection.

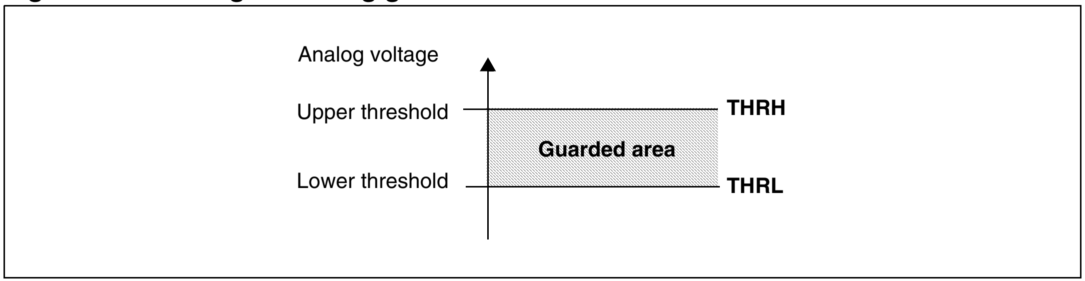
For each analog watchdog, the analog input channel to be guarded is selected by the THRCHn bits in the ADC_TRAx (x=[0,1,2,3]) registers.

The low and high thresholds of the guarded area are selected by the THRL and THRH bits in the ADC_TRBx and ADC_TRAx (x=[0,1,2,3]) registers.

The analog watchdogs are enabled by setting the THREN bits in the ADC_TRBx (x=[0,1,2,3]) registers.

After conversion of the selected channel is finished, a comparison is performed between the current channel and the threshold values in THRL and THRH. The result is stored in the THRx bits in the ADC_PBR register as shown in *Table 87 on page 426*, and, depending on the MSKxL and MSKxH mask bits in the ADC_IMR register, an interrupt request is generated if the converted value is outside the guarded area shown in *Figure 117*.

Figure 117. Analog watchdog guarded area



17.3.9 DMA functionality

A DMA request can be programmed after conversion of each channel, by setting the respective masking bit DMAx (x=[0..15]) in the DMAR register. It is strongly recommended to set/reset the DMA masking bits only when the channel conversion is stopped.

The DMA operates only if the DMAEN bit in the ADC_DMAE register is set by software or if the DMA is enabled by hardware using the DMA external enable trigger. Enabling the DMA by hardware is active if bit DXEN in the ADC_DMAE register as been set to 1. The level sensitivity is controlled by the DEDGE bit in the ADC_DMAE register.

When the DMA is enabled (by software or hardware), the number of the first DMA enabled converted channel is stored in the DENCH[3:0] bits in the ADC_DMAE register, to be able to synchronize the software analyzing the data transferred by DMA with the conversion of the ADC channels (this is especially useful when the DMA is enabled by hardware).

Note: The DMA external enable trigger connected to the TIM2 OC2 outputs signal is internally resynchronized with PCLK. Consequently, it is mandatory that the user respects the minimum pulse duration of 2 APB clocks when programming the TIM2 timer to generate this trigger.

17.3.10 Interrupts

The ADC can generate five interrupt requests:

- EOC (End of Conversion) interrupt request
- ECH (End of Chain) interrupt request
- JEOC (End of Injected Conversion) interrupt request
- JECH (End of Injected Chain) interrupt request
- Analog watchdog interrupt requests (two pending bits for each channel)

The logical OR of all previous requests is provided to the EIC.

Two registers named ADC_PBR (Pending Bit Register) and ADC_IMR (Interrupt Mask Register) are provided to check and enable the interrupt request to the EIC.

The ADC_PBR register contains the interrupt pending request. The watchdog interrupt source sets two pending bits (as indicated in *Table 87 on page 426*) for each of the four guarded channels.

To reset a PBR pending bit, a register write with all bits to zero (except the one to be cleared) should be done by software.

This is the only write operation to the PBR register. Any other write operation is forbidden, therefore the register is accessible in Read/Clear mode *only*.

The ADC_IMR register stores the 12 bits used to enable the interrupt request sources. The register is read/write accessible.

17.3.11 Power-down mode

The analog part of the ADC can be switched to low-power mode by setting the PWDN bit in the ADC_CLR4 register.

After reset, the analog module is in power-down mode by default. This state must therefore be exited before starting any operation by resetting the appropriate bit in ADC_CLR4.

When in power-down mode, no calibration or conversion can be started, and if a calibration or conversion is ongoing, the operation is immediately killed and must be restarted manually (by setting the appropriate start bit) after resuming from power-down.

17.3.12 Auto-clock-off mode

To reduce power consumption during operation (without switching to power-down mode) an “auto-clock-off” feature can be enabled by setting the ACKO bit in the ADC_CLR4 register. When enabled, the analog clock is automatically switched off when no operation is ongoing.

17.4 Register description

17.4.1 ADC Data Register (ADC_Dx)

Address Offset: 50h...8Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CDATA									
-	-	-	-	-	-	r	r	r	r	r	r	r	r	r	r

Bits 15:10	Reserved, forced by hardware to 0.
Bits 9:0	**CDATA:** *Channel Converted Data* The conversion results for the 16 available channels are loaded into the 16 different data registers following conversion of the corresponding analog input.

17.4.2 Control Logic Register 0 (ADC_CLR0)

Address Offset: 00h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TRGEN	EDG LEV	EDGE	Reserved										CAL	START
-	rw	rw	rw	-	-	-	-	-	-	-	-	-	-	rw	rw

Bit 15	Reserved, forced by hardware to 0.
Bit 14	**TRGEN:** *External Trigger Enable* 0 : The start of conversion trigger (connected to the TIM0 timer OC2) is disabled. 1 : The start of conversion trigger (connected to the TIM0 timer OC2) is enabled.
Bit 13	**EDGLEV:** *Edge or Level Selection for start conversion Trigger* Selects edge or level detection for the start of conversion trigger . 0: Level detection: conversion starts if the level is detected on the timer trigger (whatever the START bit state). 1: Edge detection: conversion starts if (and only if) the edge is detected on the timer trigger AND if the START bit is set.
Bit 12	**EDGE:** *Start Trigger Edge/Level Detection Polarity* If TRGEN is set, this bit selects the edge/level detection polarity for the start of conversion trigger. 0: Falling edge or low level detection (depending on EDGLEV bit). 1: Rising edge or high level detection (depending on EDGLEV bit).
Bits 11:2	Reserved, forced by hardware to 0.

Bit 1	**CAL:** *Calibrate* 0: No effect (write) or no calibration ongoing (read). 1: Start calibration (write) or calibration ongoing (read). No other operation (except power-down) is possible until calibration is terminated, that is, when this bit is cleared by hardware.
Bit 0	**START:** *Start Conversion* 0: Stop conversion. Clearing this bit in Scan conversion mode causes the current chain conversion to finish and stops the operation. 1: Start chain or scan conversion. This bit stays at a high value while conversion is ongoing (or pending in injection mode).

17.4.3 Control Logic Register 1 (ADC_CLR1)

Address Offset: 04h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPEN	Reserved							CNVP			Reserved		SMPP		
rw	-	-	-	-	-	-	-	rw	rw	rw	-	-	rw	rw	rw

Bit 15	**SPEN:** *Sample Prescaler Enable* 0: Use clock from conversion prescaler during sampling phase. 1: Use clock from sampling prescaler during sampling phase.
Bits 14:8	Reserved, forced by hardware to 0.
Bits 7:5	**CNVP[2:0]:** *Conversion Prescaler* These bits are written by software to define the prescaling factor used during calibration and conversion. Refer to *Table 86*. This clock is also used for sampling if the SPEN bit is cleared.
Bits 4:3	Reserved, forced by hardware to 0.
Bits 2:0	**SMPP[2:0]:** *Sampling Prescaler* These bits are written by software to define the prescaling factor used during the sampling phase of the conversion. Refer to *Table 86*. This clock is applied only during channel sampling and if the SPEN bit is set. If a conversion or calibration is ongoing, a write to this field becomes effective only at the end of the pending operation.

Table 86. Conversion/sampling prescaler configuration

CNVP[2:0] / SMPP[2:0]	f_{ADC}
000	PCLK
001	PCLK/2
010	PCLK/4
011	PCLK/6
100	PCLK/8
101	PCLK/10
110	PCLK/12
111	PCLK/14

17.4.4 Control Logic Register 2 (ADC_CLR2)

Address Offset: 08h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE	Reserved					NCH				Reserved		FCH			
rw	-	-	-	-	-	rw	rw	rw	rw	-	-	rw	rw	rw	rw

Bit 15	**MODE:** *One-shot Scan* 0: One-shot mode: configure the conversion of one chain (from channel FCH for NCH channels). 1: Scan mode: configure continuous chain conversion mode (from channel FCH for NCH channels). When the programmed chain conversion is finished it restarts immediately.
Bits 14:12	Reserved, forced by hardware to 0.
Bits 11:10	Reserved, must be kept at reset value 0.
Bits 9:6	**NCH[3:0]:** *Number of channels to be converted.* These bits define the number of channels to be converted minus one. When a channel is converted, the number is then decremented for a successive conversion, until the number is 0. When conversion is ongoing, a read of this register will return the number of the channel to be converted before the end of the chain (even in injected mode). At the end of a chain operation, the value written by software is restored.
Bits 5:4	Reserved, must be kept at reset value 0.
Bits 3:0	**FCH[3:0]:** *First channel to be converted.* These bits define the starting analog input channel. The first channel addressed by FCH is converted, the channel number is then incremented for a successive conversion, until the last channel is converted. When conversion is ongoing, a read of this register will return the number of the currently converted channel (or the pending channel in injected mode). When the maximum available channel is reached and NCH is not 0, the next converted channel is set to 0. At the end of the chain operation, the value written by software is restored.

17.4.5 Control Logic Register 3 (ADC_CLR3)

Address Offset: 0Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JSTART	JTRGEN	JEDGE	Reserved			JNCH				Reserved		JFCH			
rw	rw	rw	-	-	-	rw	rw	rw	rw	-	-	rw	rw	rw	rw

Bit 15	**JSTART:** *Injection Start* 0: No effect (injected conversion cannot be interrupted). 1: Start conversion of injected analog channels.
Bit 14	**JTRGEN:** *Injection Trigger Enable* 0: The start of injection trigger (connected to the PWM timer TRGO) is disabled. 1: The start of injection trigger (connected to the PWM timer TRGO) is enabled.
Bit 13	**JEDGE:** *Timer Injection Trigger Edge Selection* If JTRGEN is ‘1’, this bit selects the edge detection polarity for the PWM timer trigger. 0: Falling edge. 1: Rising edge.
Bits 12:10	Reserved, must be kept at reset value 0.
Bits 9:6	**JNCH[2:0]:** *Number of injected channels to convert.* These bits define the number of analog input channels minus one to be used for channel injection.
Bits 5:4	Reserved, must be kept at reset value 0.
Bits 3:0	**JFCH[3:0]:** *First injected channel to convert.* These bits define the starting analog input channel for the injected conversion.

17.4.6 Control Logic Register 4 (ADC_CLR4)

Address Offset: 10h
Reset value: 8000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PWDN	ACKO	Reserved								NOAVRG	Reserved				
rw	rw	-	-	-	-	-	-	-	-	rw	-	-	-	-	-

Bit 15	**PWDN:** *Power Down Enable* 0: Power-down mode disabled (no effect if a conversion or calibration is ongoing). 1: Power-down mode enabled.
Bit 14	**ACKO:** *Auto-clock Off Enable* 0: Auto clock off feature disabled 1: Auto clock off feature enabled (switch off analog clock while not converting or calibrating).
Bits 13:6	Reserved, forced by hardware to 0.
Bit 5	**NOAVRG:** *No Calibration Average Enable* This bit is set and cleared by software. 0: Noise filtering enabled during calibration. 1: Noise filtering disabled during calibration (cuts calibration time by 16).
Bits 4:0	Reserved, must be kept at reset value 0..

17.4.7 Threshold Registers A (ADC_TRA0..3)

Address Offset: 14h, 18h, 1Ch, 20h
Reset value: 0000h

The four TRAx (x=0,1,2,3) registers are used to store the user programmable upper thresholds 10-bit values and the channel linked to thresholds detection.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		THRCH				THRH									
-	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:14	Reserved, must be kept at reset value 0
Bits 13:10	**THRCH:** *Channel linked to threshold detection.* These bits define the analog input channel to be used for threshold detection of channel x (x = 0, 1, 2, 3).
Bits 9:0	**THRH:** *High threshold value for channel x*

17.4.8 Threshold Registers B (ADC_TRB0..3)

Address Offset: 24h, 28h, 2Ch, 30h
Reset value: 0000h

The four TRBx (x=0,1,2,3) registers are used to store the user programmable lower thresholds 10-bit values and to enable/disable the threshold detection.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
THREN	Reserved					THRL									
rw	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15	**THREN:** *Threshold Enable* When set, enables the threshold detection feature for the selected channel.
Bits 14:10	Reserved, must be kept at reset value 0.
Bits 9:0	**THRL:** *Low threshold value for channel x.*

17.4.9 DMA Channel Enable Register (ADC_DMAR)

Address Offset: 34h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMA15	DMA14	DMA13	DMA12	DMA11	DMA10	DMA9	DMA8	DMA7	DMA6	DMA5	DMA4	DMA3	DMA2	DMA1	DMA0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**DMA[15:0]:** *DMA Enable for channel n* 0: DMA capability is disabled for channel n 1: Channel n is enabled to transfer data in DMA mode.

17.4.10 DMA Global Enable Register (ADC_DMAE)

Address Offset: 44h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAEN	DXEN	DLEV	Reserved									DENCH			
rw	rw	rw	-	-	-	-	-	-	-	-	-	r	r	r	r

Bit 15	**DMAEN:** *DMA global enable* This bit must be set by software to globally enable the DMA feature. 0: DMA disabled. 1: DMA enabled.
Bit 14	**DXEN:** *DMA External Enable* This bit must be set if the user wants to enable the DMA though the DMA external enable trigger. 0: the DMA enable trigger will not enable the DMA 1: the DMA external enable trigger can enable the DMA (even if the DMAEN bit is reset)
Bit 13	**DLEV:** *DMA external enable level selection.* This bit selects the active level for the TIM2 OC2 DMA enable signal. 0: Low active level. 1: High active level.
Bits 12:4	Reserved, must be kept at reset value 0.
Bits 3:0	**DENCH:** *DMA first enabled channel.* The number of the first DMA-enabled channel valid when the DMAEN bit was last set (read only).

17.4.11 Pending Bit Register (ADC_PBR)

Address Offset: 48h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				THR3		THR2		THR1		THR0		JEOC	JECH	EOC	ECH
-	-	-	-	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc

Bits 15:12	Reserved, forced by hardware to 0.
Bits 11:10	**THR3[1:0]:** *Threshold Detection Channel 3.* These bits contain the result of the comparison on Analog Watchdog Threshold Channel 3. Refer to *Table 87*. Comparison takes place only if the THREN bit in the ADC_TRA3 register is set. Threshold channel 3 is associated with an analog input channel using the THRCH bits in the ADC_TRA3 register. An interrupt request is generated if enabled by the corresponding MSK bit in the ADC_IMR register.
Bits 9:8	**THR2[1:0]:** *Threshold Detection Channel 2.* These bits contain the result of the comparison on Analog Watchdog Threshold Channel 2. Refer to *Table 87*. Comparison takes place only if the THREN bit in the ADC_TRA2 register is set. Threshold channel 2 is associated with an analog input channel using the THRCH bits in the ADC_TRA2 register. An interrupt request is generated if enabled by the corresponding MSK bit in the ADC_IMR register.
Bits 7:6	**THR1[1:0]:** *Threshold Detection Channel 1.* These bits contain the result of the comparison on Analog Watchdog Threshold Channel 1. Refer to *Table 87*. Comparison takes place only if the THREN bit in the ADC_TRA1 register is set. Threshold channel 1 is associated with an analog input channel using the THRCH bits in the ADC_TRA1 register. An interrupt request is generated if enabled by the corresponding MSK bit in the ADC_IMR register.
Bits 5:4	**THR0[1:0]:** *Threshold Detection Channel 0.* These bits contain the result of the comparison on Analog Watchdog Threshold Channel 0. Refer to *Table 87*. Comparison takes place only if the THREN bit in the ADC_TRA0 register is set. Threshold channel 0 is associated with an analog input channel using the THRCH bits in the ADC_TRA0 register. An interrupt request is generated if enabled by the corresponding MSK bit in the ADC_IMR register.
Bit 3	**JEOC:** *End of injected channel conversion.* This bit is set by hardware at the End Of Injected Conversion started by the JSTART bit. An interrupt request is generated if enabled by the MSKJEOC bit in the ADC_IMR register. 0: No JEOC event. 1: End of injected channel conversion.
Bit 2	**JECH:** *End of injected chain conversion.* This bit is set by hardware at the End of Injected Chain conversion started by the JSTART bit. An interrupt request is generated if enabled by the MSKJECH bit in the ADC_IMR register. 0: No JECH event. 1: End of injected chain conversion.

Bit 1	**EOC:** *End of channel conversion.* This bit is set by hardware at the end of conversion started by the START bit. An interrupt request is generated if enabled by the MSKEOCH bit in the ADC_IMR register. 0: No EOC event. 1: End of channel conversion.
Bit 0	**ECH:** *End of chain conversion.* This bit is set by hardware at the End of Chain conversion started by the START bit. An interrupt request is generated if enabled by the MSKECH bit in the ADC_IMR register. 0: No ECH event. 1: End of Chain Conversion.

Table 87. Significance of analog watchdog THRx[1:0] bits

THRx[1:0] bits	Meaning
10	Converted data >= THRH
01	Converted data < THRL
00	THRL <= onverted data < THRH

17.4.12 Interrupt Mask Register (ADC_IMR)

Address Offset: 4Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				MSK3H	MSK3L	MSK2H	MSK2L	MSK1H	MSK1L	MSK0H	MSK0L	MSK JEOC	MSK JECH	MSK EOC	MSK ECH
-	-	-	-	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc

Bits 15:12	Reserved, forced by hardware to 0.
Bit 11	**MSK3H:** *Analog Watchdog 3 High Threshold Interrupt Enable* This bit is set and cleared by software. 0: THR3H interrupt request disabled. 1: THR3H interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 3 is greater than or equal to the high threshold (THR3[1:0]=10).
Bit 10	**MSK3L:** *Analog Watchdog 3 Low Threshold Interrupt Enable* This bit is set and cleared by software. 0: THR3L interrupt request disabled. 1: THR3L interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 3 is less than the low threshold (THR3[1:0]=01).
Bit 9	**MSK2H:** *Analog Watchdog 2 High Threshold Interrupt Enable* This bit is set and cleared by software. 0: THR2H interrupt request disabled. 1: THR2H interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 2 is greater than or equal to the high threshold (THR2[1:0]=10).
Bit 8	**MSK2L:** *Analog Watchdog 2 Low Threshold Interrupt Enable.* This bit is set and cleared by software. 0: THR2L interrupt request disabled. 1: THR2L interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 2 is less than the low threshold (THR1[1:0]=01).
Bit 7	**MSK1H:** *Analog Watchdog 1 High Threshold Interrupt Enable* This bit is set and cleared by software. 0: THR1H interrupt request disabled. 1: THR1H interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 1 is greater than or equal to the high threshold (THR1[1:0]=10).
Bit 6	**MSK1L:** *Analog Watchdog 1 Low Threshold Interrupt Enable* This bit is set and cleared by software. 0: THR1L interrupt request disabled. 1: THR1L interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 1 is less than the low threshold (THR1[1:0]=01).

Bit 5	**MSK0H:** *Analog Watchdog 0 High Threshold Interrupt Enable* This bit is set and cleared by software. 0: THR0H interrupt request disabled. 1: THR0H interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 0 is greater than or equal to the high threshold (THR0[1:0]=10).
Bit 4	**MSK0L:** Analog Watchdog 0 Low Threshold Interrupt Enable This bit is set and cleared by software. 0: THR0L interrupt request disabled. 1: THR0L interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 0 is less than the low threshold (THR0[1:0]=01).
Bit 3	**MSKJEOC:** Injected End of Conversion Interrupt Enable This bit is set and cleared by software. 0: JEOC interrupt request disabled. 1: JEOC interrupt request enabled. An interrupt request is generated if the JEOC bit in the ADC_PBR register is set.
Bit 2	**MSKJECH:** Injected End of Chain Conversion Interrupt Enable This bit is set and cleared by software. 0: JECH interrupt request disabled. 1: JECH interrupt request enabled. An interrupt request is generated if the JECH bit in the ADC_PBR register is set.
Bit 1	**MSKEOC:** End of Conversion Interrupt Enable This bit is set and cleared by software. 0: EOC interrupt request disabled. 1: EOC interrupt request enabled. An interrupt request is generated if the EOC bit in the ADC_PBR register is set.
Bit 0	**MSKECH:** End of Chain Conversion Interrupt Enable This bit is set and cleared by software. 0: ECH interrupt request disabled. 1: ECH interrupt request enabled. An interrupt request is generated if the ECH bit in the ADC_PBR register is set.

17.5 ADC register map

Table 88. ADC register map

Address offset	Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	ADC_CLR0	res.	TRGEN	EDG LEV	EDG E	reserved										CAL	START
4	ADC_CLR1	SPEN	reserved							CNVP			reserved		SMPP		
8	ADC_CLR2	MODE	reserved					NCH				reserved		FCH			
C	ADC_CLR3	J START	JTRG EN	JEDGE	reserved			JNCH				reserved		JFCH			
10	ADC_CLR4	PWDN	ACKO	reserved								NO AVRG	reserved				
14	ADC_TRA0	THRCH						THRH									
18	ADC_TRA1	THRCH						THRH									
1C	ADC_TRA2	THRCH						THRH									
20	ADC_TRA3	THRCH						THRH									
24	ADC_TRB0	THR EN	reserved					THRL									
28	ADC_TRB1	THR EN	reserved					THRL									
2C	ADC_TRB2	THR EN	reserved					THRL									
30	ADC_TRB3	THR EN	reserved					THRL									
34	ADC_DMAR	DMA 15	DMA 14	DMA 13	DMA 12	DMA 11	DMA 10	DMA9	DMA8	DMA7	DMA6	DMA5	DMA4	DMA3	DMA2	DMA1	DMA0
44	ADC_DMAE	DMAEN	DXEN	DLEV	reserved									DENCH			
48	ADC_PBR	reserved				THR3		THR2		THR1		THR0		JEOC	JECH	EOC	ECH
4c	ADC_IMR	reserved				MSK3H	MSK3L	MSK2H	MSK2L	MSK1H	MSK1L	MSK0H	MSK 0L	MSK JEOC	MSK JECH	MSKE OC	MSKEC H
50	ADC_D0	reserved					CDATA										
...	...	...					...										
8C	ADC_D15	reserved					CDATA										

See *Table 2* for the base address

18 JTAG interface & Debug features

18.1 Overview

The STR750 is built around an ARM7TDMI-S core whose debug interface is Joint Test Action Group (JTAG) based. ARM7TDMI-S contains hardware extensions for advanced debugging features. The debug extensions allow the core to be stopped either on a given instruction fetch (breakpoint) or data access (watchpoint). When this happens, ARM7TDMI-S, is said to be in debug state. At this point, the core's internal state and the system's external state may be examined. Once examination is complete, the core and the system may be restored and program execution resumed.

ARM7TDMI-S is forced into debug state by an internal functional unit known as In-circuit Emulation Unit (ICE). Once in debug state, the core isolates itself from the memory system. The core can then be examined while all other system activity continues as normal.

The internal state of the ARM7TDMI-S is examined via a JTAG-style serial interface, which allows instructions to be serially inserted into the core's pipeline without using the data bus. Thus, when in debug state, a store-multiple (STM) could be inserted into the instruction pipeline and this would dump the contents of the ARM7TDMI-S, registers. This data can be serially shifted out without affecting the rest of the system.

18.2 Debug system

The ARM7TDMI-S is only one component of a more complex debug system. This is typically composed of three parts: the debug host, the protocol converter and the ARM7TDMI-S core.

18.3 Debug Host

The debug host is typically a computer running a software debugger. The debug host allows the user to issue high level commands such as "set breakpoint at location XX" or "examine the contents of memory from address 0x0 to 0x100".

18.4 Protocol Converter

The debug host will be connected to the ARM7TDMI-S development system via an interface (RS232, for example). The messages broadcast over this link must be converted to the interface signals of the core. This function is performed by the protocol converter.

18.5 ARM7TDMI-S

The ARM7TDMI-S is the lowest level of the system. Its debug extensions allow the user to stall the core from program execution, examine its internal state and the state of the memory system and then resume the program execution.

18.6 ARM7TDMI-S Debug Interface

The ARM7TDMI-S debug interface provides hardware extensions for advanced debugging. The main blocks are:

- The CPU core, with hardware support for debug.
- The Embedded ICE-RT unit (In-Circuit Emulation Real-Time) comprises :
  - one debug control register
  - one debug status register
  - two real time watchpoint units
  - an abort status register
- The Debug Communication Channel (DCC) allows to pass information between the target and the host debugger. This is implemented as coprocessor 14 and comprises two registers :
  - The DCC Control register, used for synchronized handshaking
  - The DCC data register used for data transfers
- The Test Access Port (TAP) controller.
- The Scan Chain 1 which provides serial access to the ARM CPU databus and the means to halt the CPU
- The Scan Chain 2 which provides serial access to the test logic
- The device Test Logic.

For more details, refers to the ARM documentation: ARM7TDMI-S (Rev4) Technical Reference Manual: ARM DDI 0234A.

18.7 Debugging applications with low power modes

If the application uses low power modes extensively, this can make the software difficult to debug because the debug connection may be lost. There are different cases:

- If using WFI mode, the debug connection is not lost
- If using STOP mode, the debug connection may be lost (depending on the debugger tool)
- If using STANDBY mode, the debug connection is lost

However, the microcontroller has a feature that allows you to debug your software even if you make extensive use of STOP mode. To enable this feature, the LPMC_DBG and WFI_FLASH_EN bits must be set after RESET (generally this is done by setting an option in the debugger tool).

In this configuration, whenever a STOP mode entry sequence is successfully executed, the MCU goes into WFI Mode instead of STOP mode, so the core keeps running and its debug features remain active.

Caution: Entering WFI mode instead of STOP mode may result in some differences in the behavior the software that controls the Main Clock & Reset Controller (MRCC). This is because some clock sources may have remained active, and the software may not expect this when reprogramming the clock controller after waking-up from STOP mode.

Table 89. **Low Power Debug mode selection**

Control bits				Low Power Mode Selected
LPMC[1:0]		LPMC_DBG	WFI_FLASH_EN	
0	0	0	-	STOP Mode entry
		1	1	WFI Mode (replaces STOP mode because LPMC_DBG=1)
0	1	1	0	Forbidden configuration
0	0	-	-	Software Reset
1	0	-	-	WFI Mode
1	1	-	-	STANDBY Mode

Note: When setting the LPMC_DBG bit, it is mandatory to also set the WFI_FLASH_EN bit.

18.8 Debug & Timers

During debug sessions, it is possible to choose between two options controlling how the timers behave when the ARM7TDMI-S core is halted by a breakpoint. This is controlled by a DBGC "Debug Control" bit in the corresponding CR register of each of the timers:

- see *Section 9.6.1: Control Register (TB_CR) on page 190*) for the TB timer
- see *Section 10.10.1: Control Register (TIM_CR) on page 220* for all TIM timers
- see *Section 11.10.1: Control Register (PWM_CR) on page 254* for the PWM timer

The value of each DBGC bit has the following effect:

0: The timer is stopped when the core is halted

1: The timer continues working when the core is halted

Note: The DBGC bits are reset by a system reset of the MCU.

They are are generally configured after reset by the debugger tool (depending on an option chosen via the tool user interface).

18.9 JTAG Interface signals

According to IEEE 1149.1 standard (JTAG), the physical interface to the TAP controller is based on five signals. Beyond describing how these are used, the standard also gives a set of indications about their reset status and the usage of pull-up resistors. These signals are:

- **nJTRST:** Test Reset. Active low reset signal for the TAP controller finite state machine. When the JTAG interface is used, this pin has to be held low at or after power-on in such a way as to produce an initialization (reset) of the debug controller. When out of reset, the pin must be pulled up to make JTAG communication possible. It can also be grounded if JTAG communication is not required in the application.
- **JTDI:** Test Data Input. To be pulled up by an external resistor.
- **JTMS:** Test Mode Select. To be pulled up by an external resistor. It must be high during '0' to '1' transition of nJTRST.
- **JTCK:** Test Clock. This clock is used to advance the TAP controller finite state machine. The TAP state machine maintains its state indefinitely when JTCK is held low. Optionally, similar behaviour is also obtained by holding the JTCK pin high. The standard does not impose any pull-up or pull-down resistor. A floating input is not recommended, to avoid any static power consumption.
- **JTDO:** Test Data Output. This output is in high impedance when not in use.

A sixth signal is also provided:

- **RTCK:** Return JTAG Clock. This output consists of JTCK resynchronized by a 3-stage resynchronizer based on CK_SYS.

According to the standard, all interface signals should have an external pull-up or pull-down resistor as follows:

- nJTRST Do not connect the pull-up recommended by the standard. In the final application, this pin should be grounded. In cases where debug capability is needed in the final application, implement a jumper to temporarily disconnect JTRST from ground.
- JTDI Pull-up
- JTMS Pull-up
- JTCK Pull-down or Pull up
- RTCK Pull-down or Pull up
- JTDO Floating or Pull-up/down (no static consumption anyway)

In the STR750, none of the previous resistors are implemented internally. Therefore, they need to be implemented on the application board if the JTAG & Debug Interface is used.

The JTAG reset pin (nJTRST) is a dedicated pin:

- When JTAG communication is used on the application board, nJTRST pin must be connected to a pull-up. For example, nJTSRT can be connected to the system reset which is pulled-up. However, it is mandatory to apply one negative pulse on the nJTRST pin after power-up otherwise the MCU may enter a reserved test mode.
- When the JTAG communication will not be used on the application board, it can be connected to the system reset or it can be also indefinitely grounded.

Caution: When the internal Flash is readout protected, the Flash will not execute if nJRST is not continously reset because it is considered as an intrusion. Consequently, in this case, it is mandatory that the nJRST pin sees a continuous low level after reset when the user application is running.

When the JTAG & Debug interface is not implemented and software does not use the JTAG pins as GPIOs, all the inputs must be tied to ground or V_{DD} (JTDI, JTMS, JTCK) while the outputs must be left Hi-Z (JTDO, RTCK).

18.9.1 JTAG ID Code

The JTAG ID Code is fixed to:

0x4F1F_0041 = 0b0100_1111_0001_1111__0000_0000_0100_0001

18.9.2 JTAG and GPIO multiplexing

Five of the six JTAG I/O pins are multiplexed with GPIOs. After a RESET all the JTAG pins are automatically assigned as JTAG alternate functions.

To use the JTAG pins as GPIOs, the application must set the DBGOFF bit in the GPIO_REMAP1R register. In this mode:

- These GPIOs can not be debugged
- The I/O pin states are not controlled by the GPIO registers during the initialization phase. The application hardware must manage these I/O pins during this phase (especially the outputs like JTDO and RTCK).

After RESET, the following ports are configured as:

- P1.16 / JTDI: configured as input floating during start-up phase
- P1.17 / JTDO: configured as configured as output push-pull 0
- P1.18 / JTCK: configured as input floating during start-up phase
- P1.19 / JTMS: configured as input floating during start-up phase
- P0.13 / RTCK: configured as output push-pull 0 during start-up phase

This assumes that the nJTRST pin is continuously tied to 0 (JTAG under reset).

Note: If using the JTAG pins as GPIOs, some restrictions exist when the Debug host is connecting : see Section 18.9.3: Debug mode entry & restrictions on page 434

18.9.3 Debug mode entry & restrictions

Whatever the BOOT Configuration (See *Section 1.4: Boot configuration on page 25*), the Debug Host can put the STR750 in Debug Mode using the JTAG port and halt the CPU. Some restrictions exist, which are described below:

Halting the CPU using the DEBUG features via JTAG takes time and during this time, it is possible for the ARM CPU to execute code which de-activates the JTAG and DEBUG features before the Debugger has put the ARM CPU in Halt Mode. This can happen in the following cases :

- Activating a Low Power Mode which disables the CPU clock (CK_SYS). This will disable the JTAG Clock because the JTAG signals are resynchronized with CK_SYS). This can occur when entering STOP or STANDBY mode.
- Activating the DBGOFF bit in the GPIO_REMAP0 register which kills the JTAG and DEBUG features.
- Activating Flash readout protection feature (READOUT protection bit) which disables the JTAG and DEBUG features. Refer to the *Flash protection on page 23*.

You may encounter difficulties to re-program the user code if you use the Debug Host to program a user application code capable of deactivating the JTAG as described below.

If one of these cases occurs, you can recover this situation by booting in a safe mode (for instance in Embedded Boot Loader mode, see *Section 1.4.1 on page 25*) while the Debug host is connecting.

18.10 Using RTCK for Debug mode entry

In the ARM7TDMI-S, the JTAG inputs signals are resynchronized with the system clock and RTCK (Return JTAG Clock) is provided to allow the debugger tool to adapt the JTAG clock automatically to changes in the STR750 system clock (CK_SYS) frequency.

There are two distinct cases:

- If f_{JTCK} <= (f_{CK_SYS} /6), the debugger does not need to use the returned JTAG clock (RTCK).
- If f_{JTCK} > f_{CK_SYS} /6), RTCK is required so that the debugger reduces the JTAG clock down to 6.

When debugging the STR75x, it is difficult to predict what the CPU frequency will be, as it can vary a lot depending on the current state of the smart Clock Controller (several oscillators with different range of frequencies can be used) and Low Power Modes (CPU clock can be slowed down by software).

This is why it is recommended to use RTCK when debugging, by selecting the adaptive JTAG clock feature in the debugger tool configuration.

19 Revision history

Table 90. Document revision history

Date	Revision	Changes
21-Aug-2007	1	Created new document RM0003 to replace UM0191 and restart revision numbering. Changes compared to the last revision of UM0191 (Rev 3 dated 8-Nov-2006): Removed PCLK related condition from *RWW operation on page 23* Updated *Table 7: Typical prescaler uses on page 45* Added *Section : Restrictions applying to external interrupt signal inputs on page 125* Added *Section 7.3.5: RTC flag assertion on page 169*