RM0006
Reference manual

STR91xFA ARM9®- based microcontroller family

Introduction

This reference manual provides complete information for application developers on how to use the STR91xFA microcontroller memory and peripherals.

The STR91xFA is a family of microcontrollers with different memory sizes, packages and peripherals.

For ordering information, pin description, mechanical and electrical device characteristics please refer to the STR91xFA datasheet.

For information on programming, erasing and protection of the internal Flash memory please refer to the STR9 Flash programming manual.

For information on the ARM966E-S core, please refer to the ARM966E-S Rev. 2 technical reference manual.

Related documents

- Available from www.arm.com: ARM966E-S Rev. 2 technical reference manual
- Available from *www.st.com*:
  - STR91xFA datasheet
  - STR9 Flash programming manual (PM0020)

Contents

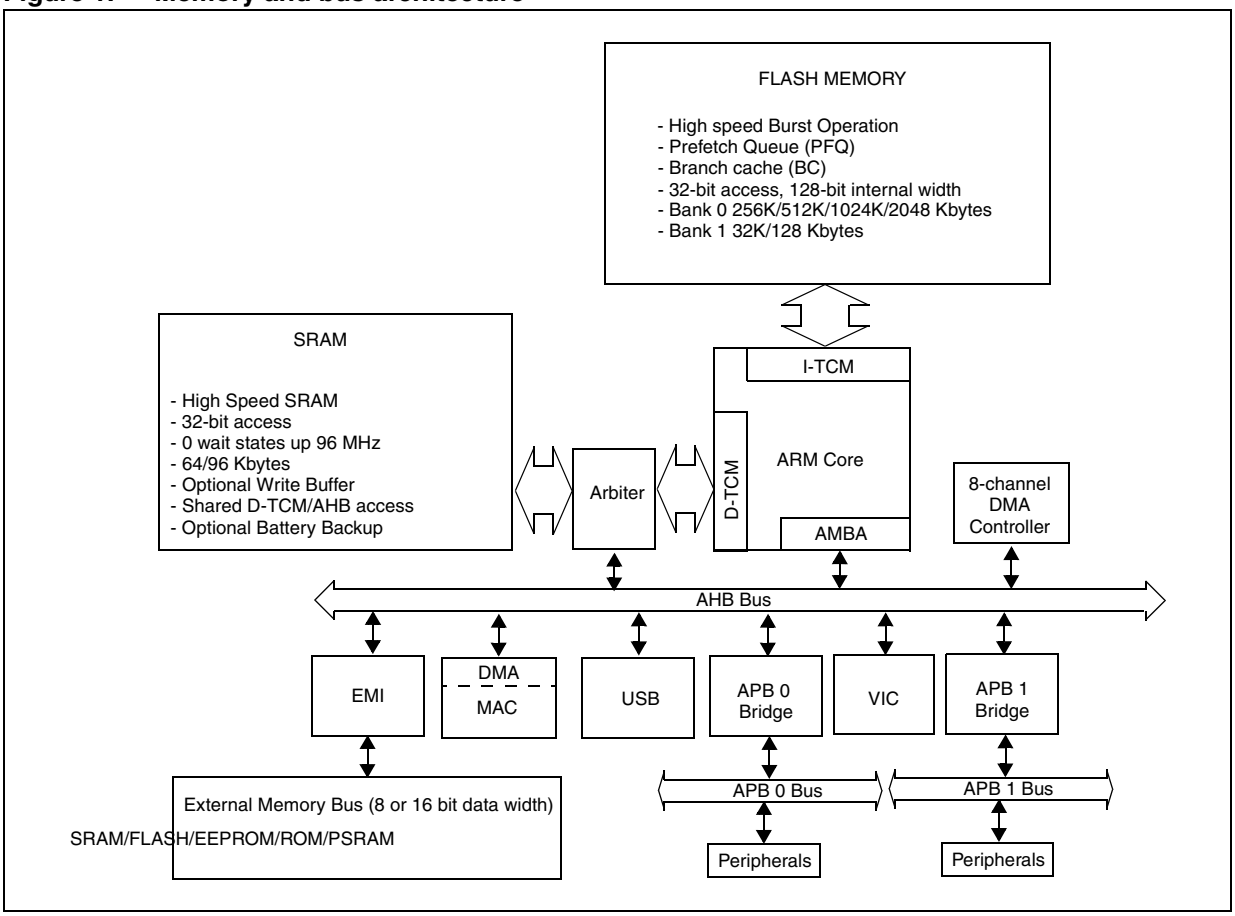
List of tables

List of figures

1 Memory and bus architecture

1.1 Introduction

Figure 1. Memory and bus architecture



1.2 ARM9 TCM memories

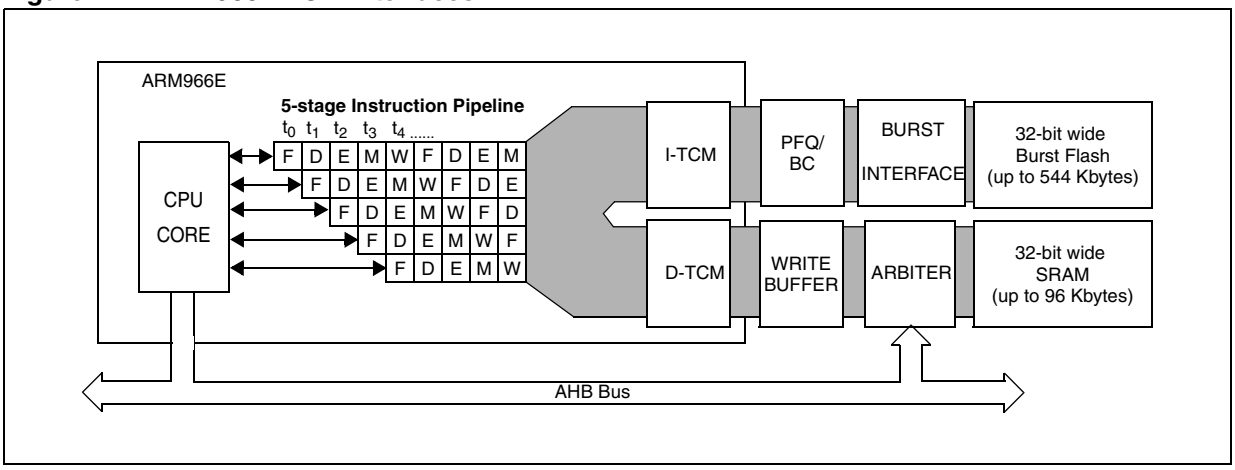
The ARM9 Tightly Coupled Memories are designed to store real-time code and performance critical code/data in dedicated memory blocks close to the processor core for quick access.

In the STR91xFA, the D-TCM and I-TCM are used as the main memory interfaces for data and instruction memory. The TCMs are enabled automatically after power on and contain the SRAM and Burst Flash memory. Refer to *Figure 2*.

The ARM966 TCM interface has the following features:

- Ability to stall the ARM966 core using the wait signal
- Signal to indicate if an access is sequential
- Signal to indicate if TCM access is Instruction or Data

Figure 2. ARM966E TCM interfaces



1. Legend: F = Fetch; D = Decode; E = execute; M = Memory read; W = Register write back

1.2.1 Burst Flash

- Dual Flash memory banks
- MCU can write/erase one while reading the other
- Either Flash bank can reside at boot location (address 0x00000000)
- Bank order is user defined

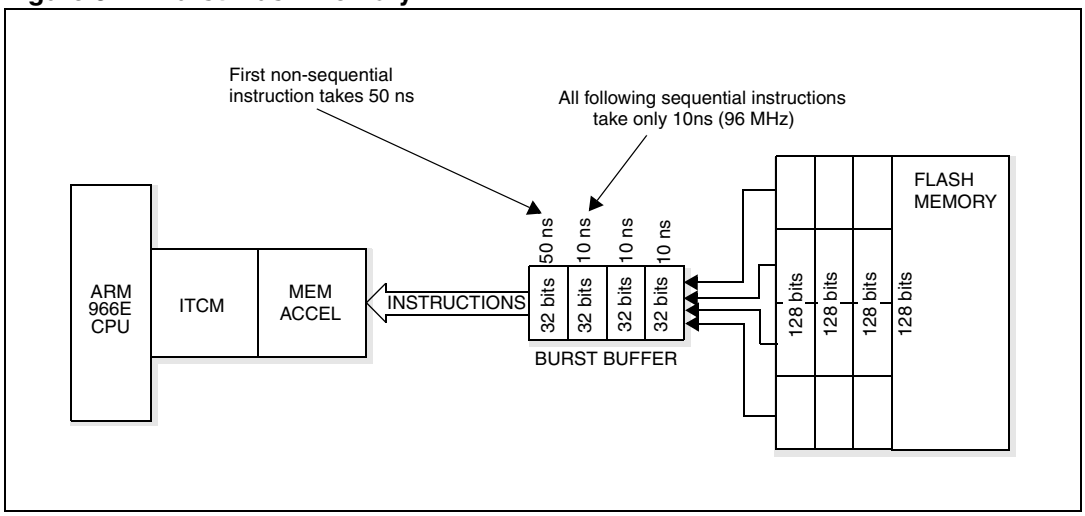
Refer to *Table 1* and *Table 2* for the bank and sector address mapping. Refer to the STR9 Flash Programming Manual for information on how to erase/program/protect the Flash.

The Low Power, Dual Bank, Burst Flash (32 bits wide) is connected to the I-TCM on a private Flash Bus. The two banks contain 256/512/1024/2028 Kb Main Flash and 32 Kb/128 Kb Secondary Flash.

Internally, burst Flash memories are 128-bits wide (4 words), which feed a 4-stage burst buffer capable of pipelining 4 words, as shown in *Figure 3*. The output of the burst buffer feeds a memory accelerator consisting of a Pre-Fetch Queue and a Branch Cache (explained in *Section 1.2.2*). In addition to storing instructions, Flash memory can store data constants, also known as literals.

When the CPU requests to read the burst Flash memory with sequential addresses, the burst buffer can supply a steady stream of 32-bit words to the CPU at a sustained rate of 96 MHz (10.4 ns access time). Anytime the CPU requests to read burst Flash memory with a non-sequential address, the 4-word burst buffer is flushed (emptied) and a new block of 128-bits is accumulated and loaded into the 4-word burst buffer. In this case, when the requested address is non-sequential, the access time for the first word coming out of the burst buffer is 50-ns. However, access time immediately returns to 10.4 ns when subsequent CPU fetches have sequential addresses, and the burst Flash again can sustain a rate of 96 MHz operation.

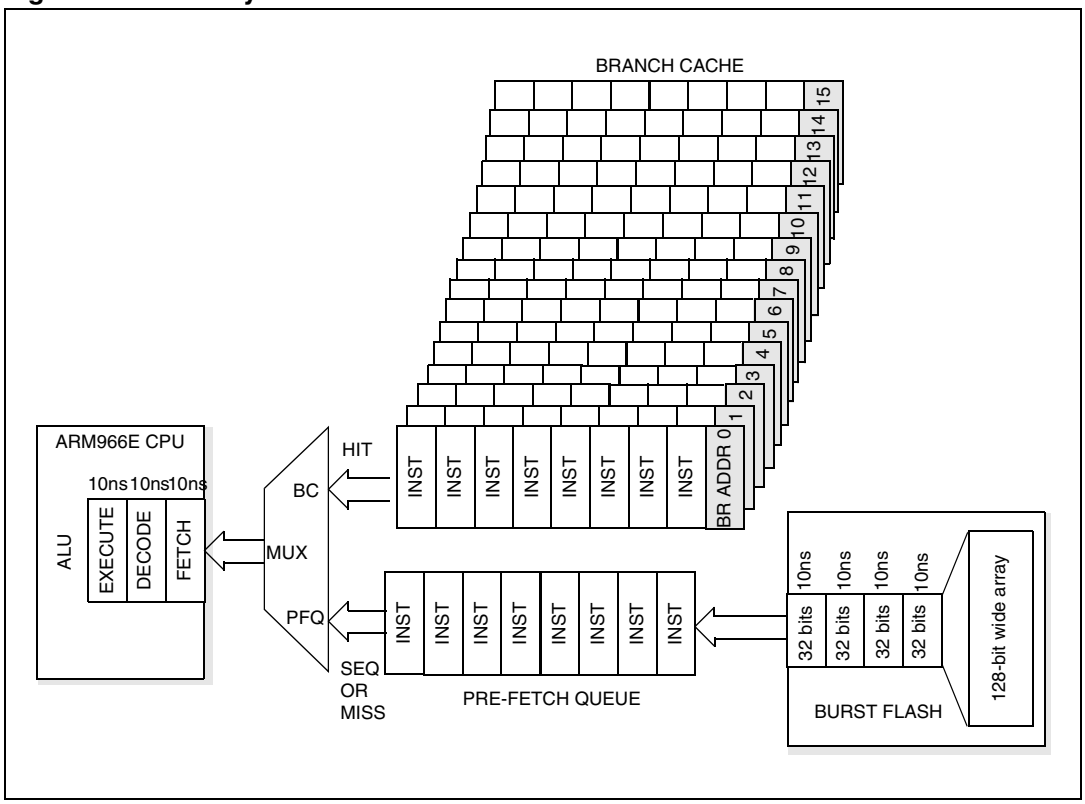
Figure 3. Burst Flash memory



1.2.2 Memory accelerator: Pre-Fetch Queue (PFQ) and Branch Cache (BC)

To minimize the effect of the 50 ns "penalty" in access time of a non-sequential address fetch, a memory accelerator (*Figure 4*) is employed to maintain a steady flow of instructions or literals with minimum "gaps" to the CPU.

Figure 4. Memory accelerator



Simply put, these "gaps" are caused by either idle bus cycles, or non-sequential instructions. The Pre-Fetch Queue (PFQ) minimizes gaps caused by idle bus cycles, and the Branch Cache (BC) minimizes gaps cause by non-sequential addresses resulting from branches in instruction flow.

Pre-fetch queue (PFQ)

Even though the ARM9E is a RISC processor, there are still 2 and 4-cycle instructions in addition to traditional 1-cycle RISC instructions. During 2 and 4-cycle instructions there are idle bus cycles when the CPU core consumes less than one word per clock. The 8-word deep PFQ has a chance to fill, or "catch-up" by prefetching instruction elements during these idle bus cycles. The resulting benefit is that there are minimum gaps in instruction flow to the CPU during sequential access to burst Flash memory, even during multi-cycle instructions.

Special design consideration was given to the PFQ to avoid a PFQ flush when the CPU fetches data constants, or literals, from the burst Flash through the ITCM. The ARM compiler allows the storage of such data literals (a look-up table for example) in the same non-volatile memory as the instructions. The PFQ logic can recognize when data literals are being fetched by the CPU and will preserve the instructions in the PFQ until the literals have been fetched, then instructions will be resumed from the PFQ.

Branch cache (BC)

When instruction addresses requested from burst Flash memory are non-sequential (from a branch in instruction flow), the PFQ must be flushed and there is a time penalty while the PFQ refills itself again from burst Flash memory, causing the CPU to stall.

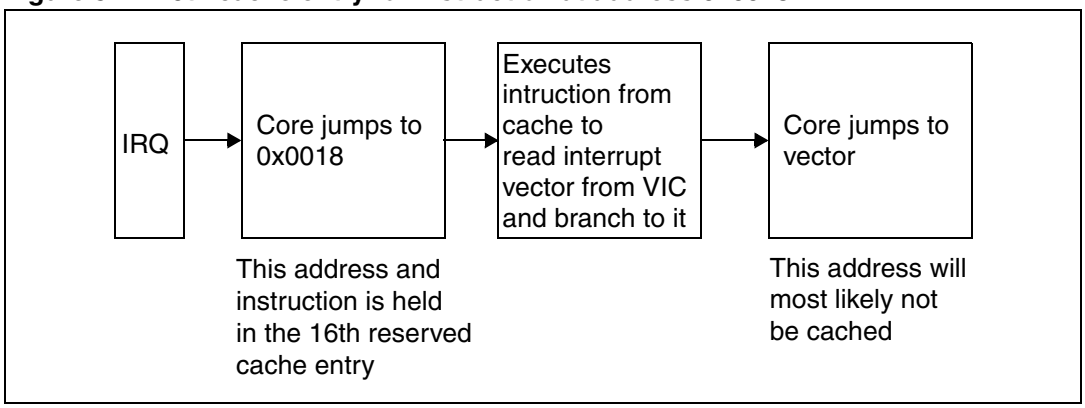
To minimize this situation, the BC remembers the destination address of the most recent previous 15 branches that the CPU has taken, and the BC stores up to 8 instructions associated with each one of those 15 branch addresses.

***Branch cache hit*:** Each time the CPU makes an instruction branch, the BC will immediately compare the current branch destination address to all 15 BC entries simultaneously. If the current branch address matches one of the 15 stored addresses, then the BC will supply up to 8 instructions of that branch immediately, minimizing time penalty or CPU stall. While the CPU is consuming up to 8 instructions provided by the BC, the PFQ has time to load itself. By the time the CPU has consumed the 8th instruction from the BC for this branch, the PFQ is ready to take over and provide the subsequent instructions (9th, 10th, and so on) without delay.

***Branch cache miss*:** If the comparison of the current branch destination address does not match any of the 15 BC entries, then the PFQ must provide the instructions, and the CPU will stall while the PFQ begins to reload itself. However, this new branch destination address and the initial 8 instructions associated with this branch are loaded into the BC for next time. The least recently used branch entry is removed from the BC to make room for this new BC entry.

The STR91xFA also makes use of the 16th entry in the BC to hold the instruction at address 0x0018 when an interrupt (IRQ) occurs. This significantly reduces the stall time when servicing interrupts.

Figure 5. 16th cache entry for instruction at address 0x0018



1.2.3 Main SRAM

The main SRAM is 32-bit wide and supports byte, half word and word data. It has zero wait state access for CPU clock frequency up to 96 MHz. A battery backup supply can optionally be connected to the VBATT pin to preserve the SRAM contents when the main power is switched off.

Shared access

SRAM Access is required by both the ARM966 core and the DMA units located on the AHB bus. A a simple “ping-pong” arbiter is implemented between the two requesters. It arbitrates access to the SRAM from the ARM Core (DTCM) and AHB Bus.

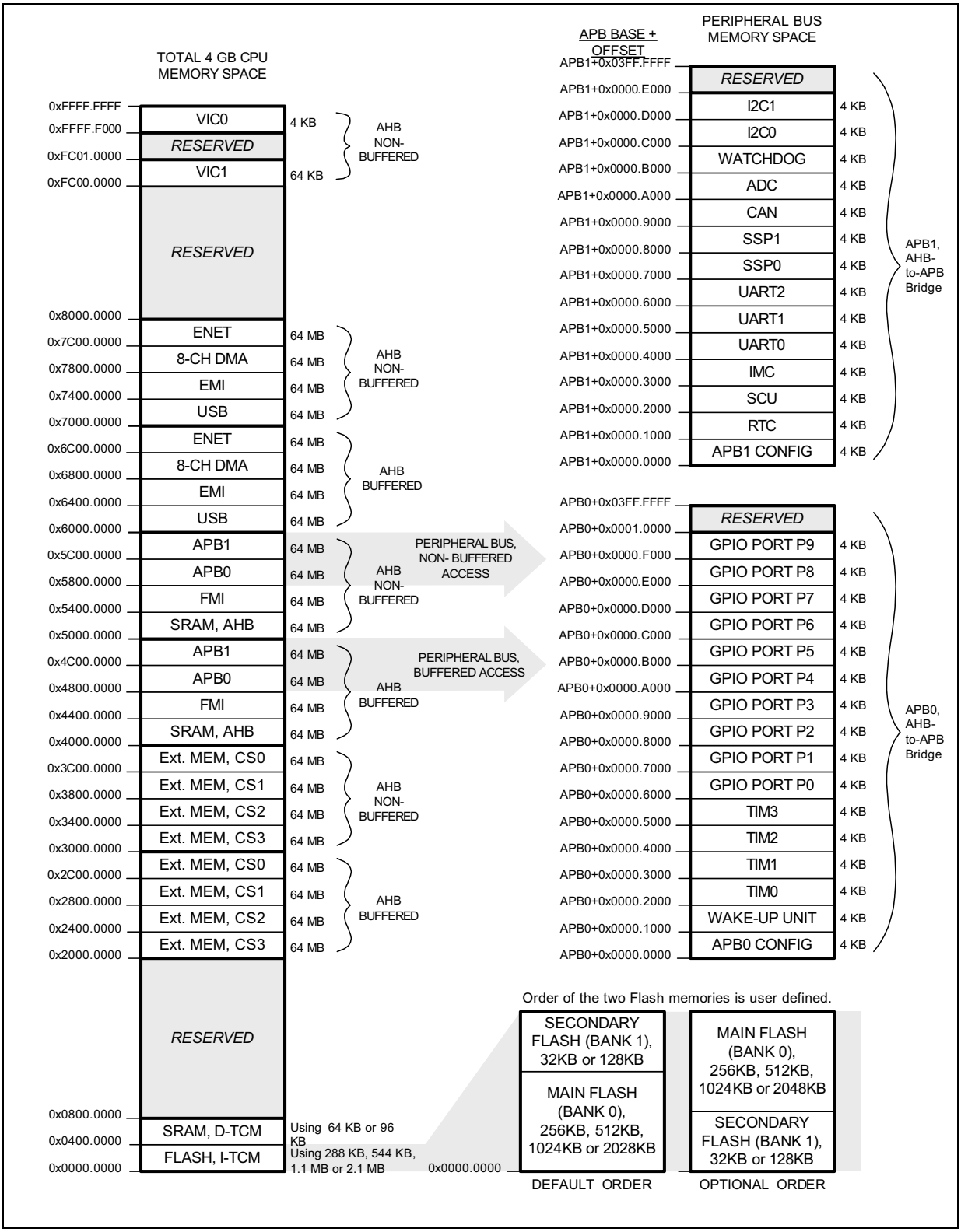
It supports Zero Wait state access to the SRAM when no contention takes place between the ARM966 DTCM and AHB bus.

When both the DTCM and the AHB are requesting access to the SRAM, it interleaves access to the SRAM adding a single wait cycle to each requestor’s data access.

1.3 Memory map

- Single linear address range
- Four Gigabyte range
- Harvard busses transparent to firmware
- Code and data separated in silicon

Figure 6. STR91xFA system memory map



The Flash program memory is organized in 32-bit wide memory cells which can be used for storing both code and data constants.

You can Program Bank 0 and Bank 1 independently, i.e. you can read from one bank while writing to the other.

The on-chip Flash is divided in 2 banks that can mapped independently in the 64 Mbyte address space 0x0000-0000 - 0x03FF.FFF by programming the FMI registers.

The STR91xFAx32 embedded Flash Module is organized as shown in *Table 1*.

Table 1. STR91xFAx32 Flash module organization

Bank	Sector	Address offset	Size (bytes)
Bank 0 256 Kbytes	Bank 0 Sector 0	0x0000.0000 - 0x0000.FFFF	64K
	Bank 0 Sector 1	0x0001.0000 - 0x0001.FFFF	64K
	Bank 0 Sector 2	0x0002.0000 - 0x0002.FFFF	64K
	Bank 0 Sector 3	0x0003.0000 - 0x0003.FFFF	64K
Bank 1 32 Kbytes	Bank 1 Sector 0	0x0000.0000 - 0x0000.1FFF	8K
	Bank 1 Sector 1	0x0000.2000 - 0x0000.3FFF	8K
	Bank 1 Sector 2	0x0000.4000 - 0x0000.5FFF	8K
	Bank 1 Sector 3	0x0000.6000 - 0x0000.7FFF	8K
Bank 1	User Configuration Sector (OTP and Electronic Signature, Configuration and Protection Registers)	Access via CUI or JTAG	32

The STR91xFAx44 embedded Flash Module is organized as shown in *Table 2*.

Table 2. STR91xFAx44 Flash module organization

Bank	Sector	Address offset	Size (bytes)
Bank 0 512 Kbytes	Bank 0 Sector 0	0x0000.0000 - 0x0000.FFFF	64K
	Bank 0 Sector 1	0x0001.0000 - 0x0001.FFFF	64K
	Bank 0 Sector 2	0x0002.0000 - 0x0002.FFFF	64K
	Bank 0 Sector 3	0x0003.0000 - 0x0003.FFFF	64K
	Bank 0 Sector 4	0x0004.0000 - 0x0004.FFFF	64K
	Bank 0 Sector 5	0x0005.0000 - 0x0005.FFFF	64K
	Bank 0 Sector 6	0x0006.0000 - 0x0006.FFFF	64K
	Bank 0 Sector 7	0x0007.0000 - 0x0007.FFFF	64K
Bank 1 32 Kbytes	Bank 1 Sector 0	0x0000.0000 - 0x0000.1FFF	8K
	Bank 1 Sector 1	0x0000.2000 - 0x0000.3FFF	8K
	Bank 1 Sector 2	0x0000.4000 - 0x0000.5FFF	8K
	Bank 1 Sector 3	0x0000.6000 - 0x0000.7FFF	8K
Bank 1	User Configuration Sector (OTP and Electronic Signature, Configuration and Protection Registers)	Access via CUI or JTAG	32

The STR91xFAx46 embedded Flash Module is organized as shown in *Table 3*.

Table 3. STR91xFAx46 Flash module organization

Bank	Sector	Address offset	Size (bytes)
Bank 0 1024 Kbytes	Bank 0 Sector 0	0x0000.0000 - 0x0000.FFFF	64K
	Bank 0 Sector 1	0x0001.0000 - 0x0001.FFFF	64K
	Bank 0 Sector 2	0x0002.0000 - 0x0002.FFFF	64K
	Bank 0 Sector 3	0x0003.0000 - 0x0003.FFFF	64K
	Bank 0 Sector 4	0x0004.0000 - 0x0004.FFFF	64K
	Bank 0 Sector 5	0x0005.0000 - 0x0005.FFFF	64K
	Bank 0 Sector 6	0x0006.0000 - 0x0006.FFFF	64K
	Bank 0 Sector 7	0x0007.0000 - 0x0007.FFFF	64K
	Bank 0 Sector 8	0x0008.0000 - 0x0008.FFFF	64K
	Bank 0 Sector 9	0x0009.0000 - 0x0009.FFFF	64K
	Bank 0 Sector 10	0x000A.0000 - 0x000A.FFFF	64K
	Bank 0 Sector 11	0x000B.0000 - 0x000B.FFFF	64K
	Bank 0 Sector 12	0x000C.0000 - 0x000C.FFFF	64K
	Bank 0 Sector 13	0x000D.0000 - 0x000D.FFFF	64K
	Bank 0 Sector 14	0x000E.0000 - 0x000E.FFFF	64K
	Bank 0 Sector 15	0x000F.0000 - 0x000F.FFFF	64K
Bank 1 128 Kbytes	Bank 1 Sector 0	0x0000.0000 - 0x0000.3FFF	16K
	Bank 1 Sector 1	0x0000.4000 - 0x0000.7FFF	16K
	Bank 1 Sector 2	0x0000.8000 - 0x0000.BFFF	16K
	Bank 1 Sector 3	0x0000.C000 - 0x0000.FFFF	16K
	Bank 1 Sector 4	0x0001.0000 - 0x0001.3FFF	16K
	Bank 1 Sector 5	0x0001.4000 - 0x0000.7FFF	16K
	Bank 1 Sector 6	0x0001.8000 - 0x0001.BFFF	16K
	Bank 1 Sector 7	0x0001.C000 - 0x0001.FFFF	16K
Bank 1	User Configuration Sector (OTP and Electronic Signature, Configuration and Protection Registers)	Access via CUI or JTAG	32

The STR91xFAx47 embedded Flash Module is organized as shown in *Table 3*.

Table 4. STR91xFAx47 Flash module organization

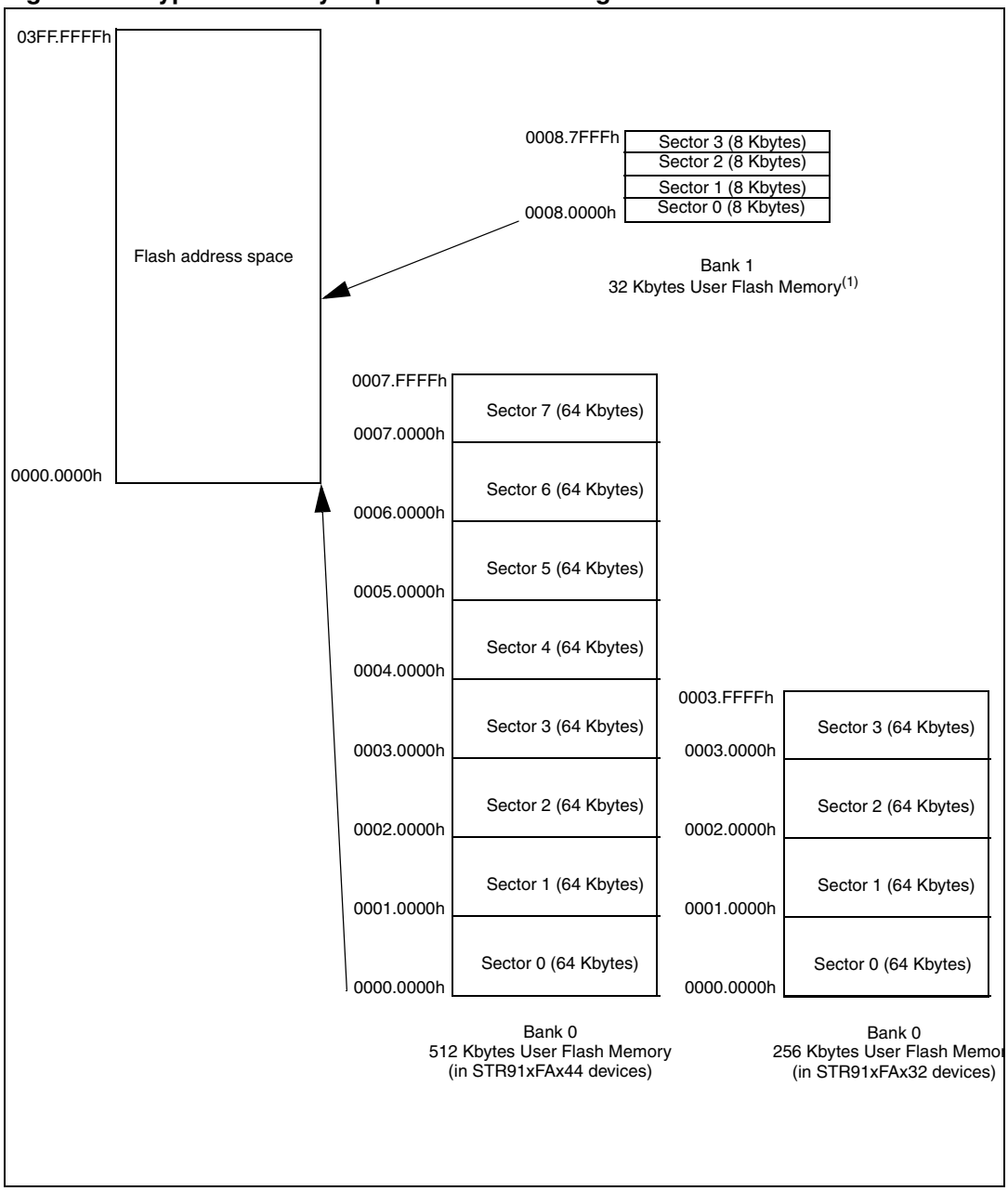
Bank	Sector	Address offset	Size (bytes)
Bank 0 2048 Kbytes	Bank 0 Sector 0	0x0000.0000 - 0x0000.FFFF	64K
	Bank 0 Sector 1	0x0001.0000 - 0x0001.FFFF	64K
	Bank 0 Sector 2	0x0002.0000 - 0x0002.FFFF	64K
	Bank 0 Sector 3	0x0003.0000 - 0x0003.FFFF	64K
	Bank 0 Sector 4	0x0004.0000 - 0x0004.FFFF	64K
	Bank 0 Sector 5	0x0005.0000 - 0x0005.FFFF	64K
	Bank 0 Sector 6	0x0006.0000 - 0x0006.FFFF	64K
	Bank 0 Sector 7	0x0007.0000 - 0x0007.FFFF	64K
	Bank 0 Sector 8	0x0008.0000 - 0x0008.FFFF	64K
	Bank 0 Sector 9	0x0009.0000 - 0x0009.FFFF	64K
	Bank 0 Sector 10	0x000A.0000 - 0x000A.FFFF	64K
	Bank 0 Sector 11	0x000B.0000 - 0x000B.FFFF	64K
	Bank 0 Sector 12	0x000C.0000 - 0x000C.FFFF	64K
	Bank 0 Sector 13	0x000D.0000 - 0x000D.FFFF	64K
	Bank 0 Sector 14	0x000E.0000 - 0x000E.FFFF	64K
	Bank 0 Sector 15	0x000F.0000 - 0x000F.FFFF	64K
	Bank 0 Sector 16	0x0010.0000 - 0x0010.FFFF	64K
	Bank 0 Sector 17	0x0011.0000 - 0x0011.FFFF	64K
	Bank 0 Sector 18	0x0012.0000 - 0x0012.FFFF	64K
	Bank 0 Sector 19	0x0013.0000 - 0x0013.FFFF	64K
	Bank 0 Sector 20	0x0014.0000 - 0x0014.FFFF	64K
	Bank 0 Sector 21	0x0015.0000 - 0x0015.FFFF	64K
	Bank 0 Sector 22	0x0016.0000 - 0x0016.FFFF	64K
	Bank 0 Sector 23	0x0017.0000 - 0x0017.FFFF	64K
	Bank 0 Sector 24	0x0018.0000 - 0x0018.FFFF	64K
	Bank 0 Sector 25	0x0019.0000 - 0x0019.FFFF	64K
	Bank 0 Sector 26	0x001A.0000 - 0x001A.FFFF	64K
	Bank 0 Sector 27	0x001B.0000 - 0x001B.FFFF	64K
	Bank 0 Sector 28	0x001C.0000 - 0x001C.FFFF	64K
	Bank 0 Sector 29	0x001D.0000 - 0x001D.FFFF	64K
	Bank 0 Sector 30	0x001E.0000 - 0x001E.FFFF	64K
	Bank 0 Sector 31	0x001F.0000 - 0x001F.FFFF	64K

Table 4. STR91xFAx47 Flash module organization (continued)

Bank	Sector	Address offset	Size (bytes)
Bank 1 128 Kbytes	Bank 1 Sector 0	0x0000.0000 - 0x0000.3FFF	16K
	Bank 1 Sector 1	0x0000.4000 - 0x0000.7FFF	16K
	Bank 1 Sector 2	0x0000.8000 - 0x0000.BFFF	16K
	Bank 1 Sector 3	0x0000.C000 - 0x0000.FFFF	16K
	Bank 1 Sector 4	0x0001.0000 - 0x0001.3FFF	16K
	Bank 1 Sector 5	0x0001.4000 - 0x0000.7FFF	16K
	Bank 1 Sector 6	0x0001.8000 - 0x0001.BFFF	16K
	Bank 1 Sector 7	0x0001.C000 - 0x0001.FFFF	16K
Bank 1	User Configuration Sector (OTP and Electronic Signature, Configuration and Protection Registers)	Access via CUI or JTAG	32

The write operations of the two banks are managed by an embedded Flash Program/Erase Controller (FPEC). The high voltage needed for Program/Erase operations is internally generated.

Figure 7. Typical memory map with device configured to boot from Bank 0



1. Bank 1 also contains the user configuration sector with OTP memory, Electronic Signature and Protection Registers.

1.4 Initialization

After reset, to define the mapping of the Flash memory banks, the user firmware has to write the start address and memory size of Bank 0 and Bank 1 in the FMI registers (see *Section 1.10*).

You must write the start address and the memory size of the bank configured as boot memory first and then the start address and the memory size of the other (non-boot) bank.

1.5 Boot configuration

The STR91xFA always boots from internal Flash address 0x0000.0000h.

In the default configuration, after reset the first sector of Bank 0 is enabled and resides at 0x0000.0000h so that the device boots from Bank 0, and Bank 1 is disabled.

The application then has to write to the FMI Registers configure the size and base address of Bank 0 and Bank 1. Refer to *Section 1.10: FMI register description on page 38*.

Using the JTAG interface, you can configure the device to boot from Bank 1. The selection of which Flash memory is at the boot location is programmed in a non-volatile Flash-based configuration bit. The firmware cannot change this configuration bit, only the JTAG interface has access. Refer to the STR9 Flash Programming Manual.

1.6 OTP sector

This device provides 30 One Time Programmable (OTP) bytes that can be read or written by the CPU, or the JTAG interface. These bytes can be used to store calibration contents, serial numbers, security codes, Ethernet MAC address, etc.

Each byte can be written only one time, and it is not possible to modify that byte ever again once written. Erasing an OTP byte is never possible. There is a lock bit available that can be set to prevent the writing of OTP bytes. For example, the lock bit can be set after writing 5 OTP bytes, and the remaining 25 bytes cannot be written.

The 31st and 32nd OTP bytes are reserved and are programmed at the factory to contain the revision number of the STR91xFA silicon. This information can be read by the CPU or JTAG interface but can never be modified or erased.

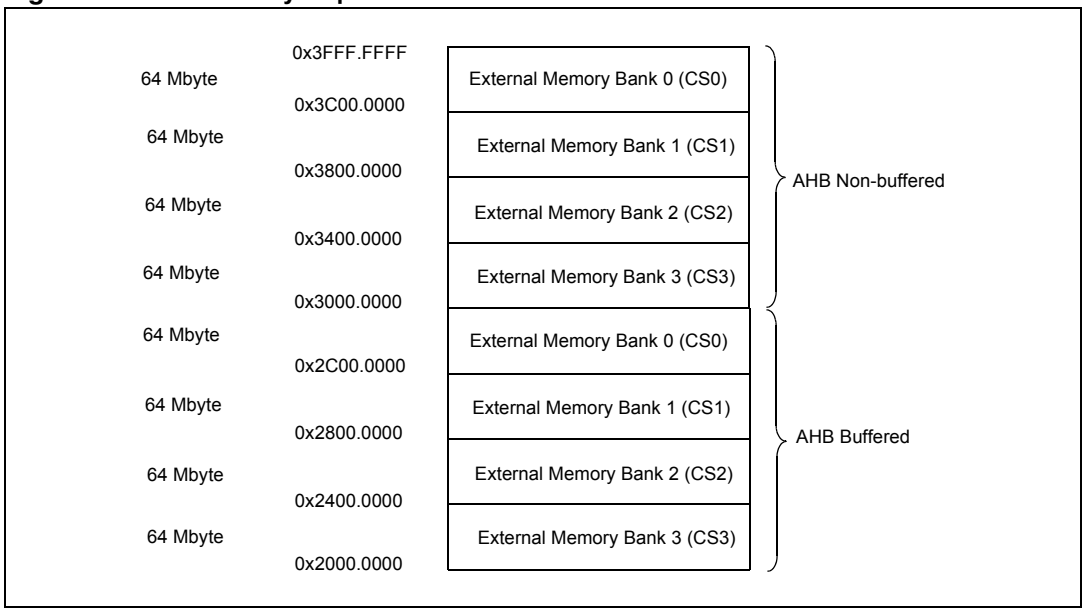
It is recommended to use the 25th through 30th OTP bytes to store an Ethernet MAC address.

Refer to the STR9 Flash Programming manual details on programming the OTP sector.

1.7 External memory

Refer to *Section 1.12* for a description of the external memory interface (EMI).

Figure 8. EMI Memory Map



1.8 Peripheral access

- High Speed Peripherals on AHB
- Lower Speed Peripherals on APB
- Firmware accesses APB through a bridge
- Separate Ranges for AHB Write Buffer
- Peripherals have two address ranges, one for buffered writes and another for non-buffered writes
  - Buffered writes increase overall performance
  - Non-buffered writes guarantee data coherency

The CPU makes use of write buffers on the AHB and the D-TCM to decouple the CPU from any wait states associated with a write operation. The user may choose to use write with buffers on the AHB by setting bit 3 in control register CP15 and selecting the appropriate AHB address range when writing. By default at reset, buffered writes are disabled (bit 3 of CP15 is clear) and all AHB writes are non-buffered until enabled. *Figure 6* shows that most addressable items on the AHB are aliased at two address ranges, one for buffered writes and another for non-buffered writes. A buffered write will allow the CPU to continue program execution while the write-back is performed through a FIFO to the final destination on the AHB. If the FIFO is full, the CPU is stalled until FIFO space is available. A non-buffered write will impose an immediate delay to the CPU, but results in a direct write to the final AHB destination, ensuring data coherency. Read operations from AHB locations are always direct and never buffered.

Note: It is recommended to use non-buffered writes when writing to configuration registers.

1.9 Peripheral memory map

Table 5. Peripheral memory map

Peripheral name	Bus	Peripheral boundary addresses		Peripheral register map
		Buffered	Non-Buffered	
Vectored Interrupt Controller 0 (VIC0)	AHB	N/A	0x FFFF F23F	*Section 4.11 on page 135*
			0xFFFF F000	
Vectored Interrupt Controller 1 (VIC1)	AHB	N/A	0x FC00 023F	
			0xFC00 0000	
802.3 MAC/DMA (ENET)	AHB	0x6C00 042F	0x7C00 042F	*Section 8.5 on page 242*
		0x6C00 0000	0x7C00 0000	
8-Channel DMA Controller (DMAC)	AHB	0x6800 01F3	0x7800 01F3	*Section 9.6 on page 273*
		0x6800 0000	0x7800 0000	
External Memory Interface (EMI)	AHB	0x6400 00F7	0x7400 00F7	*Section 1.12.9 on page 64*
		0x6400 0000	0x7400 0000	
Universal Serial Bus (USB)	AHB	0x6000 0867	0x7000 0867	*Section 15.6.5 on page 461*
		0x6000 0000	0x7000 0000	

Table 5. Peripheral memory map (continued)

Peripheral name	Bus	Peripheral boundary addresses		Peripheral register map
		Buffered	Non-Buffered	
I^2C bus interface 1 (I2C1)	APB1	0x4C00 D01F	0x5C00 D01F	Section 12.6 on page 340
		0x4C00 D000	0x5C00 D000	
I^2C bus interface 0 (I2C0)	APB1	0x4C00 C01F	0x5C00 C01F	
		0x4C00 C000	0x5C00 C000	
Watchdog Timer (WDG)	APB1	0x4C00 B01B	0x5C00 B01B	Section 6.5 on page 161
		0x4C00 B000	0x5C00 B000	
Analog/Digital converter (ADC)	APB1	0x4C00 A037	0x5C00 A037	Section 16.6 on page 478
		0x4C00 A000	0x5C00 A000	
Controller Area Network (CAN)	APB1	0x4C00 9167	0x5C00 9167	Section 14.6 on page 402
		0x4C00 9000	0x5C00 9000	
Synchronous Serial Peripheral (SSP1)	APB1	0x4C00 8027	0x5C00 8027	Section 10.6 on page 294
		0x4C00 8000	0x5C00 8000	
Synchronous Serial Peripheral (SSP0)	APB1	0x4C00 7027	0x5C00 7027	
		0x4C00 7000	0x5C00 7000	
UART 2	APB1	0x4C00 604B	0x5C00 604B	Section 11.5 on page 322
		0x4C00 6000	0x5C00 6000	
UART 1	APB1	0x4C00 504B	0x5C00 504B	
		0x4C00 5000	0x5C00 5000	
UART 0	APB1	0x4C00 404B	0x5C00 404B	
		0x4C00 4000	0x5C00 4000	
Induction Motor Control (MC)	APB1	0x4C00 3047	0x5C00 3047	Section 13.5 on page 372
		0x4C00 3000	0x5C00 3000	
System Control Unit (SCU)	APB1	0x4C00 20BF	0x5C00 20BF	Section 2.6.10 on page 114
		0x4C00 2000	0x5C00 2000	
Real Time Clock (RTC)	APB1	0x4C00 1017	0x5C00 1017	Section 5.10 on page 154
		0x4C00 1000	0x5C00 1000	
AHB/APB1 bridge (APB1)	AHB/APB1	0x4C00 000B	0x5C00 000B	Section 17.5 on page 482
		0x4C00 0000	0x5C00 0000	

Table 5. Peripheral memory map (continued)

Peripheral name	Bus	Peripheral boundary addresses		Peripheral register map
		Buffered	Non-Buffered	
GPIO Port 9	APB0	0x4800 F423	0x5800 F423	Section 3.4.4 on page 121
		0x4800 F000	0x5800 F000	
GPIO Port 8	APB0	0x4800 E423	0x5800 E423	
		0x4800 E000	0x5800 E000	
GPIO Port 7	APB0	0x4800 D423	0x5800 D423	
		0x4800 D000	0x5800 D000	
GPIO Port 6	APB0	0x4800 C423	0x5800 C423	
		0x4800 C000	0x5800 C000	
GPIO Port 5	APB0	0x4800 B423	0x5800 B423	
		0x4800 B000	0x5800 B000	
GPIO Port 4	APB0	0x4800 A423	0x5800 A423	Section 3.4.4 on page 121
		0x4800 A000	0x5800 A000	
GPIO Port 3	APB0	0x4800 9423	0x5800 9423	
		0x4800 9000	0x5800 9000	
GPIO Port 2	APB0	0x4800 8423	0x5800 8423	
		0x4800 8000	0x5800 8000	
GPIO Port 1	APB0	0x4800 7423	0x5800 7423	
		0x4800 7000	0x5800 7000	
GPIO Port 0	APB0	0x4800 6423	0x5800 6423	
		0x4800 6000	0x5800 6000	
Timer 3 (TIM3)	APB0	0x4800 501F	0x5800 501F	Section 7.7 on page 181
		0x4800 5000	0x5800 5000	
Timer 2 (TIM2)	APB0	0x4800 401F	0x5800 401F	
		0x4800 4000	0x5800 4000	
Timer 1 (TIM1)	APB0	0x4800 301F	0x5800 301F	
		0x4800 3000	0x5800 3000	
Timer 0 (TIM0)	APB0	0x4800 201F	0x5800 201F	
		0x4800 2000	0x5800 2000	
Wakeup/Interrupt Unit (WUI)	APB0	0x4800 1013	0x5800 1013	Section 4.12.3 on page 142
		0x4800 1000	0x5800 1000	
AHB/APB0 bridge (APB0)	AHB/APB0	0x4800 000B	0x5800 000B	Section 17.5 on page 482
		0x4800 0000	0x5800 0000	

Table 5. Peripheral memory map (continued)

Peripheral name	Bus	Peripheral boundary addresses		Peripheral register map
		Buffered	Non-Buffered	
Flash Memory Interface (FMI)	AHB	0x4400 0013	0x5400 0013	Section 1.11 on page 44
		0x4400 0000	0x5400 0000	

1.10 FMI register description

The FMI Registers configure the size and base address of the Bank 0 and Bank 1. The address ranges of Bank 0 and Bank 1 must not overlap each other.

The microcontroller boots from Bank 0 by default:

In the default configuration:

- Bank 0 is the Boot Bank, after reset the application program has to write the size and base address of Bank 0 in the FMI_BBSR and FMI_BBADR registers
- Bank 1 is the Non-Boot Bank, after reset the application program has to write the size and base address of Bank 1 in the FMI_NBBSR and FMI_NBBADR registers

Booting from Bank 1

The microcontroller can also boot from Bank 1. The selection of the Boot Bank can be modified using the "CAP" Software Tool.

If Bank 1 is the Boot bank, after reset, the application program has to write the size and start address of Bank 1 in the FMI_BBSR and FMI_BBADR registers and the size and start address of Bank 0 in the FMI_NBBSR and FMI_NBBADR registers.

1.10.1 Boot bank size register (FMI_BBSR)

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												BBSIZE[3:0]			
												rw	rw	rw	rw

Bits 31:4	Reserved, always read as 0
Bits 3:0	**BBSIZE[3:0]**: *Boot bank size* These bits are set and cleared by software. They define the address space for the boot bank. 0000: 32 Kbytes. 0001: 64 Kbytes .... 1011: 64 Mbytes Other values are reserved.

1.10.2 Non-boot bank size register (FMI_NBBSR)

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												NBBSIZE[3:0]			
												rw	rw	rw	rw

Bits 31:4	Reserved, always read as 0
Bits 3:0	**NBBSIZE[3:0]**: *Non-boot bank size* These bits are set and cleared by software. They define the address space for the non booting memory bank. 0000: 8 Kbytes. 0001: 16 Kbytes .... 1101: 64 Mbytes Other values are reserved.

1.10.3 Boot Bank base address register (FMI_BBADR)

Address offset: 0Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								B0ADDR[23:16]							
								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B0ADDR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24	Reserved, always read as 0
Bits 23:0	**BBADDR[23:0]:** *Boot bank base address* These bits are set and cleared by software. They define the base address of the boot bank. The address must be word-aligned.

1.10.4 Non-boot bank base address register (FMI_NBBADR)

Address offset: 10h

Reset value: 0000 0000h

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

Bits 31:24	Reserved, always read as 0
Bits 23:0	**NBBADDR[23:0]**: *Non-boot bank base address* These bits are set and cleared by software. They define the base address of the non-boot bank. The address must be word-aligned.

1.10.5 FMI Control register (FMI_CR)

Address offset: 18h

Reset value: 0000 0009h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							WWS	OMIE	BEIE	Res.	NBB EN	BBEN	Reserved		
							rw	rw	rw	rw	rw	rw			

Bits 31:9	Reserved, always read as 0
Bit 8	**WWS**: *Write Wait States* This bit is set and cleared by software. It defines the number of wait states in Flash write access. 0: Flash write is active for 1 clock cycle (Recommended setting) 1: Flash write is active for 2 clock cycles (Reserved for future use)
Bit 7	**OMIE**: *Out of Memory interrupt enable* This bit is set and cleared by software. It enables/disables the Out of Memory interrupts. 0: Disabled 1: Enabled. An interrupt is generated when the OM bit in the FMI_SR register is set.
Bit 6	**BERRIE**: *Flash Bank Error interrupt enable* This bit is set and cleared by software. It enables/disables Flash bank error interrupts. 0: Disabled 1: Enabled. An interrupt is generated when the B1ERR or B0ERR bit in the FMI_SR register are set.
Bit 5	Reserved, always read as 0
Bit 4	**NBBEN**: *Flash Non Boot Bank enable* This bit is set and cleared by software. It enables/disables the Non Boot Bank. 0: Disabled 1: Enabled.
Bit 3	**BBEN**: *Flash Boot Bank enable* This bit is set and cleared by software. It enables/disables Flash Boot Bank. 0: Disabled 1: Enabled
Bits 2:0	Reserved, always read as 01

1.10.6 FMI Status register (FMI_SR)

Address offset: 1Ch

Reset value: 0000 0010h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved											PFQ BCEN	OM	Res.	B1ERR	B0ERR
											r	rc_w1		rc_w1	rc_w1

Bits 31:5	Reserved, always read as 0
Bit 4	**PFQBCEN**: *PFQBCEN Status* This bit is set and cleared by harware. 0: PFQ/BC disabled (bypassed) 1: PFQ/BC enabled
Bit 3	**OM**: *Out of Memory error* This bit is set by hardware and cleared by software writing 1. It indicates that an access was made outside the configured memory area. An interrupt is generated if the OMIE bit in the FMI_CR register is set. 0: No OM error 1: An Out of Memory error occurred
Bit 2	Reserved, always read as 0
Bit 1	**B1ERR**: *Flash Bank 1 error* This bit is set by hardware and cleared by software writing 1. It indicates that an access was made to Bank 1 while it was disabled. An interrupt is generated if the BERRIE bit in the FMI_CR register is set. 0: No B1ERR error 1: A Flash Bank 1 error occurred
Bit 0	**B0ERR**: *Flash Bank 0 error* This bit is set by hardware and cleared by software writing 1. It indicates that an access was made to Bank 0 while it was disabled. An interrupt is generated if the BERRIE bit in the FMI_CR register is set. 0: No B0ERR error 1: A Flash Bank 0 error occurred

1.10.7 BC 16th Entry Target Address register (FMI_BCE16ADDR)

Address offset: 20h

Reset value: 0000 0006h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								BCE16ADDR[23:16]							
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCE16ADDR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24	Reserved, always read as 0
Bits 23:0	**BCE16ADDR[23:0]**: *Branch Cache 16th Entry Target Address* These bits are set and cleared by software. They define the target address of the BC 16th entry, provided to implement interrupt (IRQ) mode or any “special” branch not subject to the LRU algorithm. The address written to the register is the target address divided by 4. For example, IRQ address at 0x18 divided by 4 = 0x06. Defaults to 0x00000006 at reset i.e. IRQ exception).

1.11 FMI register map

Table 6. FMI 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	FMI_BBSR																													BBSIZE			
04h	FMI_NBBSR																													NBBSIZE			
0Ch	FMI_BBADR									BBADDR[23:0]																							
10h	FMI_NBBADR									NBBADDR[23:0]																							
18h	FMI_CR																								WWS	OMIE	BEIE		NBBEN	BBEN			
1Ch	FMI_SR																													PFQ BCEN	OM		B1ERR
20h	FMI_BCE16ADDR									BCE16ADDR[23:0]																							

Refer to *Table 5 on page 35* for the base addresses.

1.12 External memory interface (EMI)

1.12.1 Functional description

The EMI provides an interface between the AHB system bus and external (off-chip) memory devices, supporting up to four memory banks that you can configure independently. Each memory bank supports:

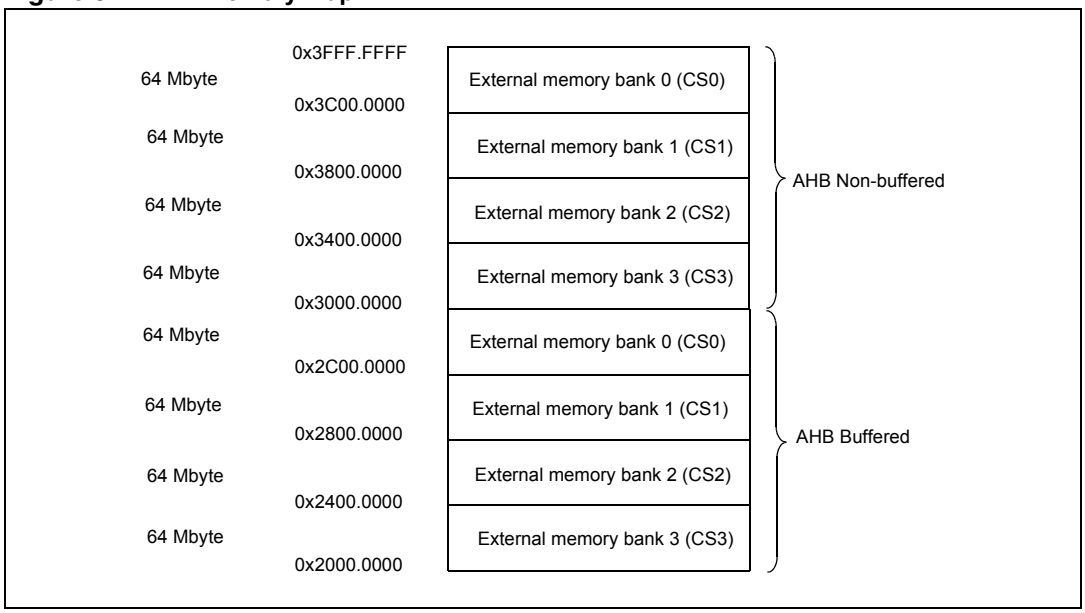
- SRAM
- ROM
- Flash EPROM
- PSRAM

You can configure each memory bank to use 8- or 16-bit data paths.

You can configure the EMI memory banks to support:

- Asynchronous read and write accesses
- Asynchronous page mode read accesses supported in 8-bit non-multiplexed EMI configuration.
- Synchronous burst read and write accesses to PSRAM
- Up to 24 address lines in multiplexed EMI configuration.

Figure 9. EMI memory map



1.12.2 Summary of bus configurations

Non-Mux mode

1. 8-bit only
2. Control Signals: EMI_Rdn, EMI_WRn
3. Port Config:
   a) Port 8 EMI_D[7:0]
   b) Port 7 EMI_A[7:0]
   c) Port 9 EMI_A[15:8]

Mux mode

1. 8- or 16-bit

   When configured as a 16-bit data bus, the address output on the EMI bus is shifted by 1 so as to address 16-bit memory devices. For example, writing a half word to location 0x0042 will generate an EMI address of 0x0021
2. Control Signals: EMI_Rdn, EMI_WRHn, EMI_WRLn, EMI_ALE
3. Port Config:
   a) Port 8 EMI_AD[7:0]
   b) Port 9 EMI_AD[15:8]
   c) Port 7 EMI_A[23:16]

PSRAM mode

1. A subset of 16-bit mux mode (LFBGA package only). This mode allows the EMI bus interface directly to PSRAM for synchronous access (burst read and write).
2. Control Signals: EMI_Rdn, EMI_WEn, EMI_UBn, EMI_LBn, EMI_ALE, EMI_WAITn, CRE (a GPIO pin output, not an EMI bus signal)
3. Port Config:
   a) Port 8 EMI_AD[7:0]
   b) Port 9 EMI_AD[15:8]
   c) Port 7 EMI_A[23:16]

Figure 10. Mux mode with 16-bit data, 20-bit address

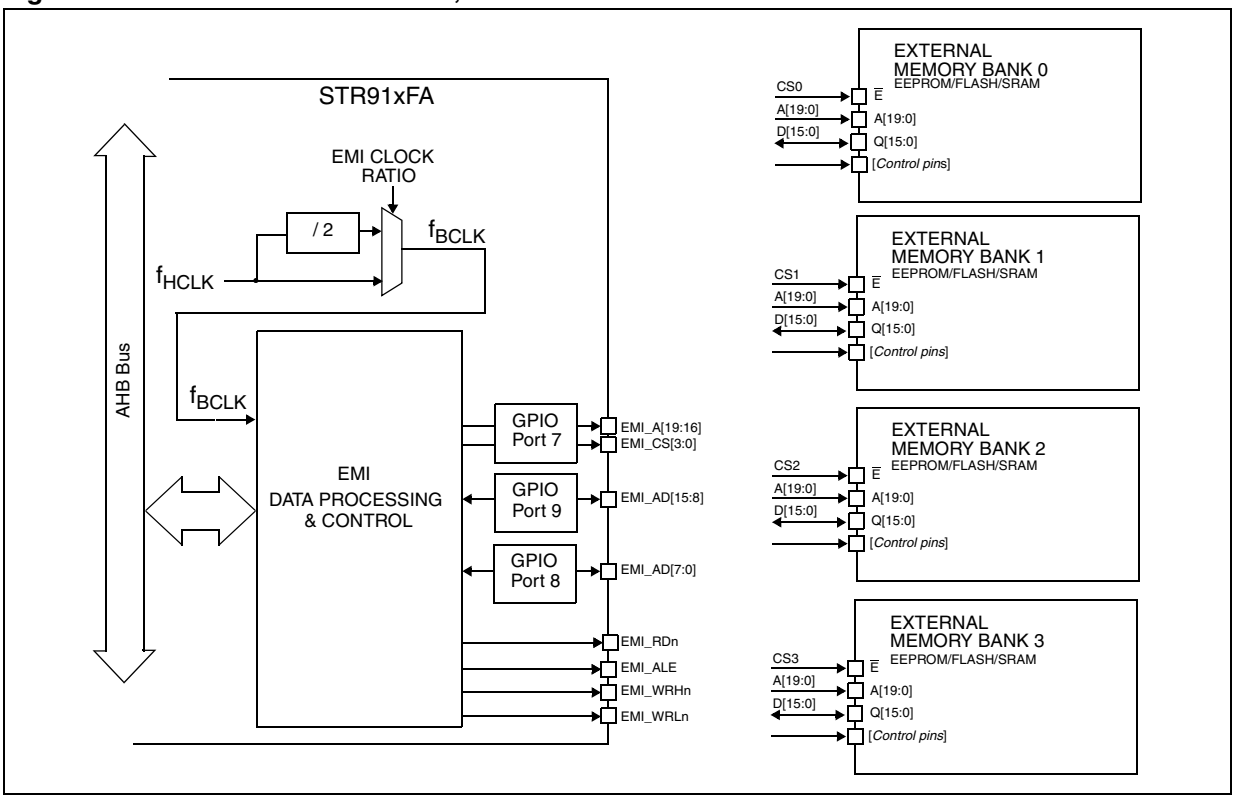


Figure 11. Mux mode with 16-bit data, 24-bit address

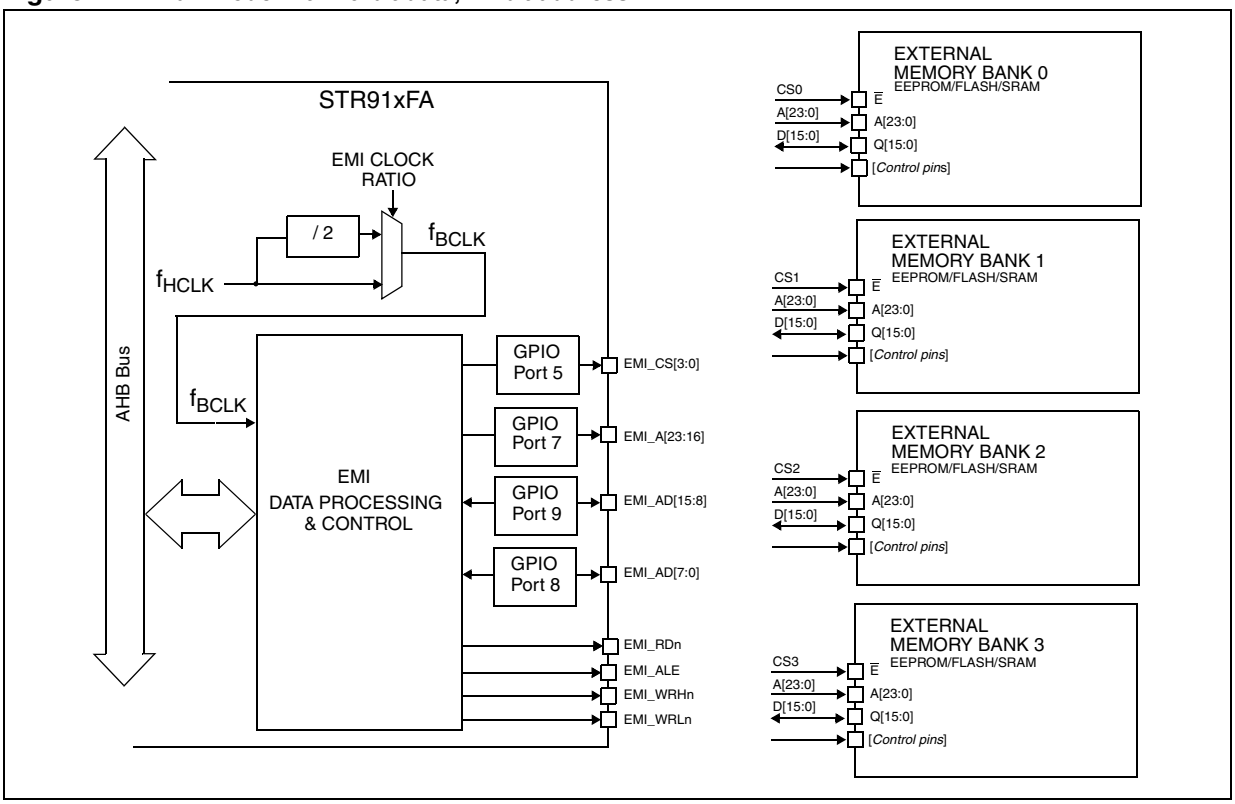


Figure 12. Non-mux mode with 8-bit data, 16-bit address

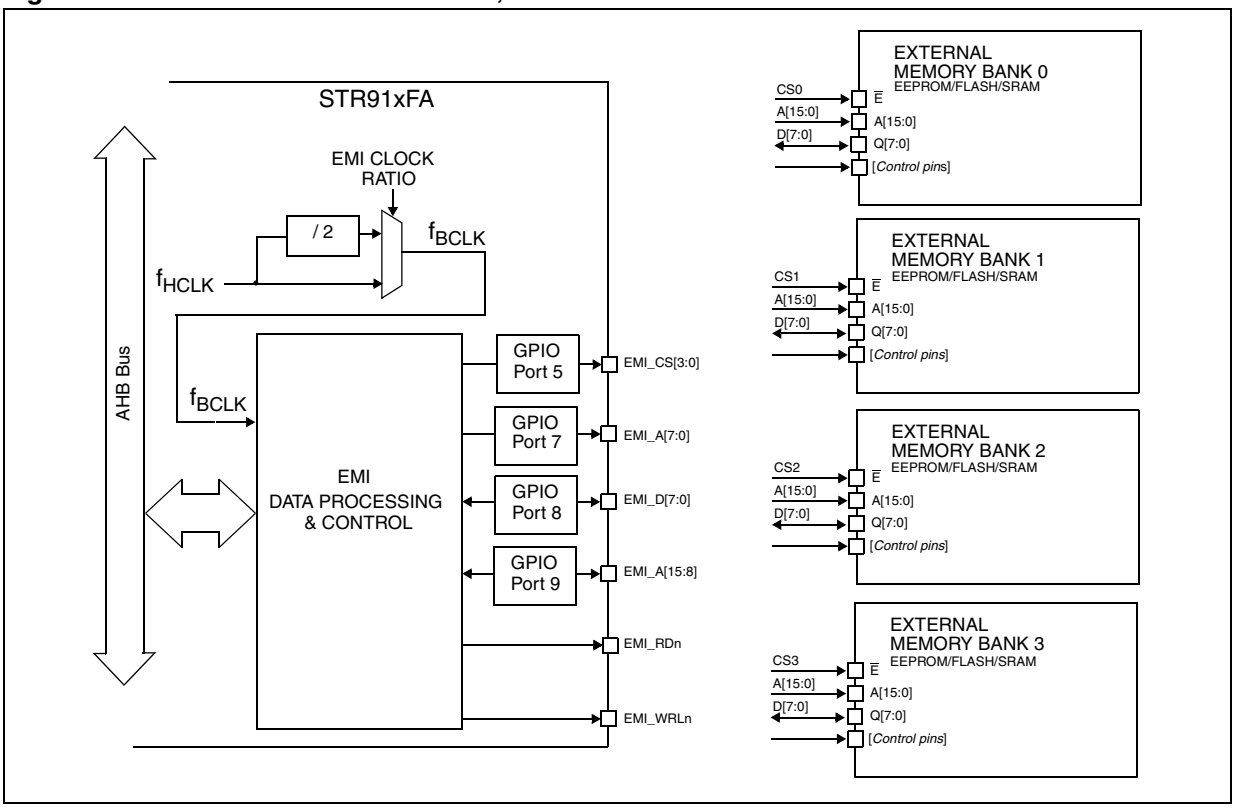
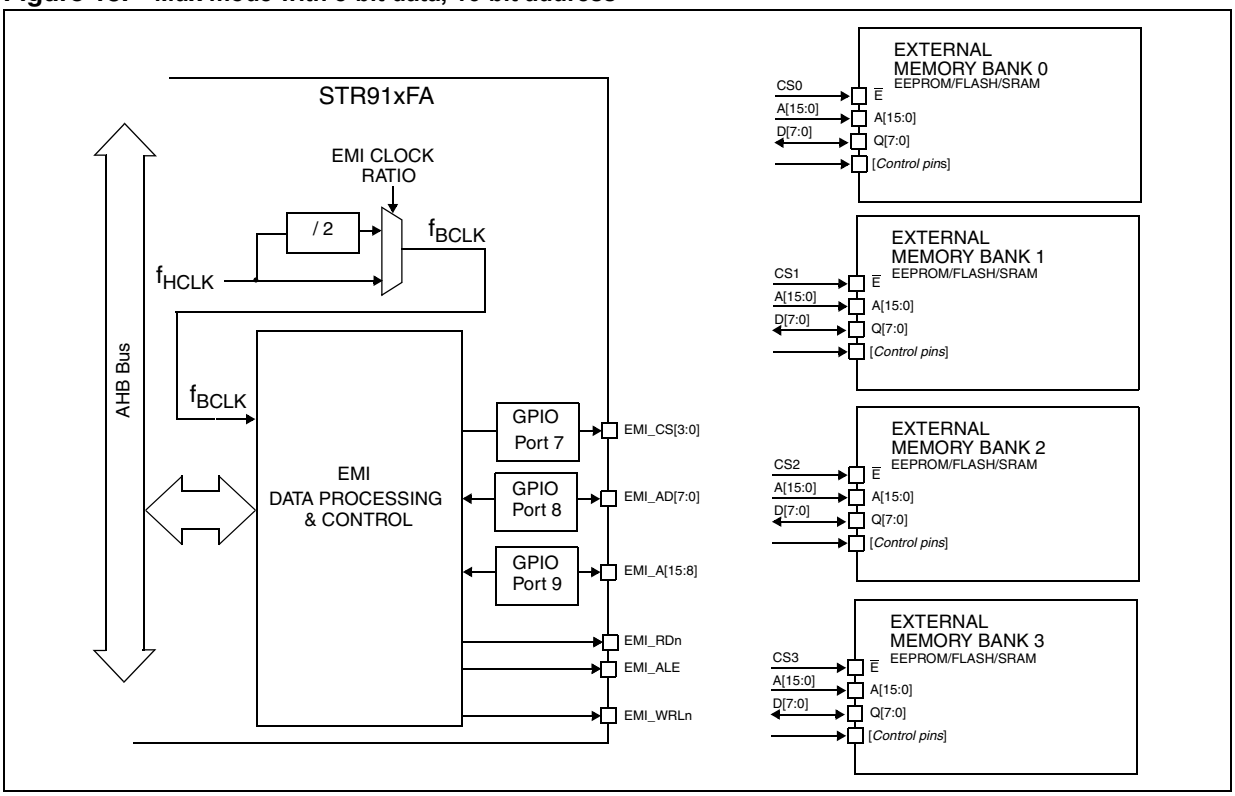


Figure 13. Mux mode with 8-bit data, 16-bit address



1.12.3 External memory interface (EMI) configuration/control

Mux/demux mode

Using the EMI_MUX bit in the *System configuration register 0 (SCU_SCR0) on page 108*, you can select EMI Mux/Demux mode or demux mode.

ALE length

Using the EMI_ALE_LENGTH bit in the *System configuration register 0 (SCU_SCR0) on page 108*, you can select EMI ALE length to be 1 or 2 BCLK periods. (Note: 0.5 or 1.5 BCLK periods when in synchronous mode).

ALE polarity

Using the EMI_ALE_POL bit in the *System configuration register 0 (SCU_SCR0) on page 108*, you can select EMI ALE polarity to be active high or low.

GPIO port 8, 9 and 7

You have to set up set up bits 0 to 1 in the *GPIO external memory interface register (SCU_EMI) on page 111* to enable ports 8 and 9 for the EMI bus function.

Port 7 is the address port and is configured as Alternative 2 output function for pins P7.0 to P7.6 and as Alternative 3 output function for pin P7.7 (refer to GPIO chapter). Address lines on Port 7 are pin selectable, where only the address lines that are needed are enabled.

Chip selects CS0-3

The 4 chip selects are available on Ports 0, 5 or 7. Configure the pins P5.4 to P5.7 and P7.4 to P7.6 as GPIO Alternate function 3 and pins P0.4 to P0.7 and P7.7 as GPIO Alternate function 2 (refer to GPIO chapter) to enable the chip selects. It is recommended that the CS0-3 signals have external pull up resistors as they are not driven by the MCU before the bus is configured.

Byte select signal configuration

In the LFBGA package, the upper byte and lower byte enable signals (EMI_UBn, EMI_LBn) share the same pins as the EMI_WRHn and EMI_WRLn and are user configurable. The byte select signals are enabled by setting bit 2 (PSRAM mode) in the *GPIO external memory interface register (SCU_EMI) on page 111* and bit 1 in the EMI_BCRx register. The EMI_WEn signal is needed to work with the byte enable signals to write to a 16 bit PSRAM memory device.

1.12.4 External memory interface clock (BCLK)

You can select the frequency of the EMI bus clock (BCLK) to be HCLK or HCLK/2 using the EMIRATIO bit in the *Clock control register (SCU_CLKCNTR)*. By default the frequency is HCLK/2.

In the LFBGA package, the BCLK can be brought out to the pin by setting bit 1 to 0 in the *GPIO external memory interface register (SCU_EMI) on page 111* and bit 0 in the EMI_CCR register to 1. All bus timings and parameters are reference to the BCLK.

1.12.5 EMI bus timing configuration

The EMI bus timing is not configured at Power Up. You need to set up the bus timing configuration registers for each of the banks before you enable the EMI bus. The key timing parameters that you have to define to match your external memory device requirements are:

- WSTOEN: Read Enable. It specifies the delay between the assertion of the chip select and the EMI_RDn signal. The delay is defined in terms of BCLK clock periods.
- WSTRD: Read wait state. It specifies the pulse width of the EMI_RDn signal. The pulse width is defined in terms of BLCK periods and is = (WSTRD-WSTOEN+1)
- WSTWEN: Write Enable. It specifies the delay period between the assertion of the chip select and the EMI_WRn signal. The delay is defined in terms of BCLK clock periods and is = (WSTWEN + 1/2) for asynchronous write cycles and is = WSTWEN for synchronous access.
- WSTWR: Write wait state. It specifies the pulse width of the EMI_WRn signal. The pulse width is defined in terms of BLCK periods and is = (WSTWR-WSTWEN+1) for asynchronous write cycles. For synchronized write accesses, the width is= (WSTWR-WSTWEN + 2).

Example: A read bus cycle with WSTRD = 4, WSTOEN = 2. The resulting EMI_RDn signal is asserted 2 BLCK clock periods after CS. The pulse width is = 4 - 2 + 1 = 3 BCLK periods.

1.12.6 Timing rules

It is important to enter the correct read and write wait state values in the configuration registers. Furthermore, the EMI bus wait states must meet the following timing rules to be functional:

1. The number of Read wait states must be greater than or equal to the Output Enable wait states (WSTRD => WSTOEN) (See *Bank x read wait state control register (EMI_RCRx)* and *Bank x output enable control register (EMI_OECRx)*
2. The number of Output Enable wait states must be greater than the Address Latch Enable time in mux mode (WSTOEN> ALE) (See *Bank x output enable control register (EMI_OECRx)* and *System configuration register 0 (SCU_SCR0)*
3. The number of Write wait states must be greater than or equal to the Write Enable wait states (WSTWR=>WSTWEN) (See *Bank x write wait state control register (EMI_WCRx)* and *Bank x write enable control register (EMI_WECRx)*
4. The number of Write Enable wait states must be greater than the Address Latch Enable time in mux mode (WSTWEN> ALE) (see *Bank x output enable control register (EMI_OECRx)* and *System configuration register 0 (SCU_SCR0)*.
   Exception: WSTWEN can have a value of zero in PSRAM mode where the signals EMI_ WEn, EMI_UBn and EMI_LBn are enabled by setting bit 2 in the *GPIO external memory interface register (SCU_EMI) on page 111*.

1.12.7 Bus mode configuration

Standard asynchronous read/write bus mode

The EMI bus uses the same read and write timing to access standard SRAM and Flash devices. Each bus cycle starts with the assertion of the memory bank chip select signal (CS0-3) and memory address. When in mux bus mode, the address stays on the bus for another half BCLK clock after the EMI_ALE signal is terminated. The read or write access time is determined by the number of wait states programmed in the WSTRD or WSTWR fields of the Bank Read/Write Wait State Control Registers (EMI_RCRx, EMI_WCRx). The IDCY field in the Idle Cycle Control Register, EMI_ICRx, determines the number of bus turnaround wait states added between the read and write transfers.

The read and write bus timing diagrams in *Figure 14* and *Figure 15* are referenced to the BCLK clock. Since these are asynchronous bus accesses, the BCLK clock is not required by the memory devices. The basic configuration bits from the EMI registers that are required for the asynchronous bus mode include:

- Read Wait State (WSTRD, EMI_RCRx register)
- Write Wait State (WSTWR, EMI_WCRx register)
- Output Enable Assertion Delay (WSTOEN, EMI_OECRx register)
- Write Enable Assertion Delay (WSTWEN, EMI_WECRx register)
- Memory width (MW, EMI_BCRx register bits 5:4)
- Asynchronous access (EMI_BCRx register bits 17 and 9)

Figure 14. Asynchronous read bus cycle (mux mode, with WSTOE = 2, WSTRD = 3)

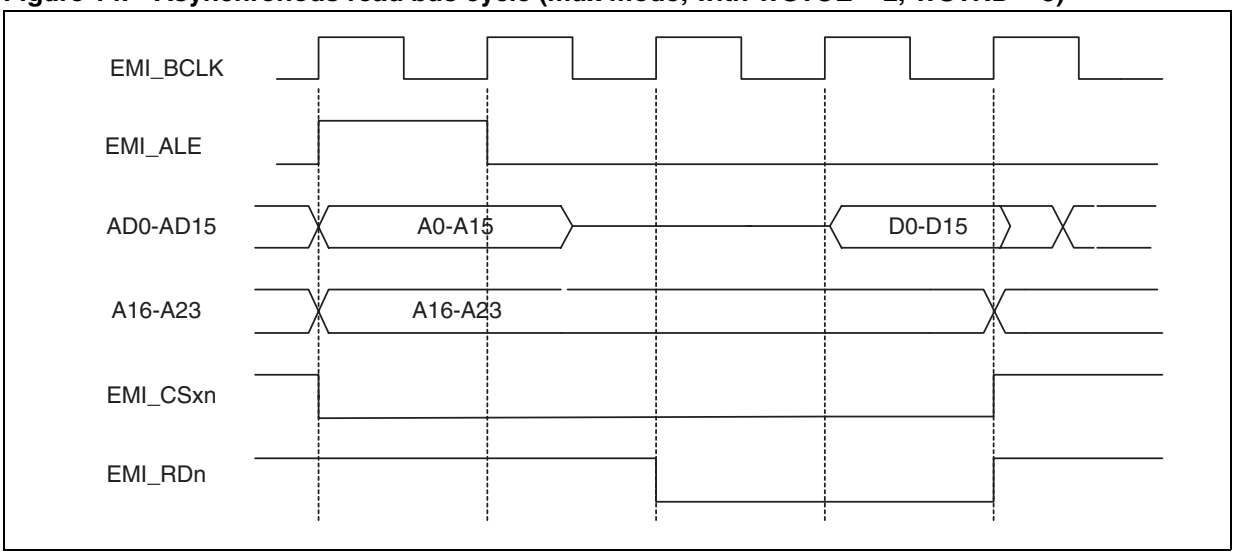
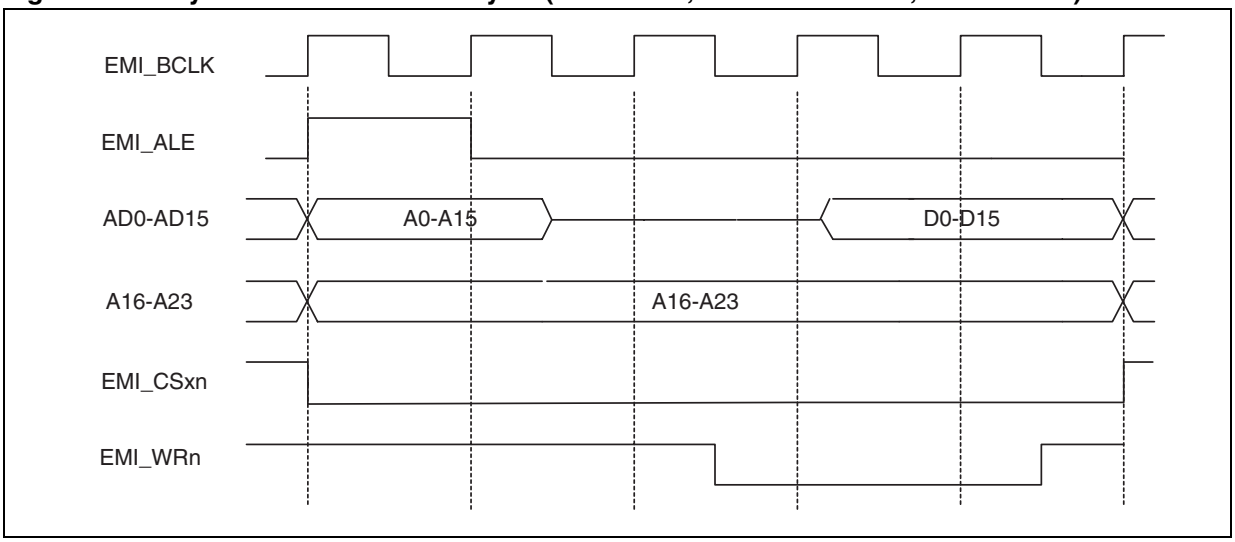


Figure 15. Asynchronous write bus cycle (mux mode, with WSTWE = 2, WSTWR = 3)



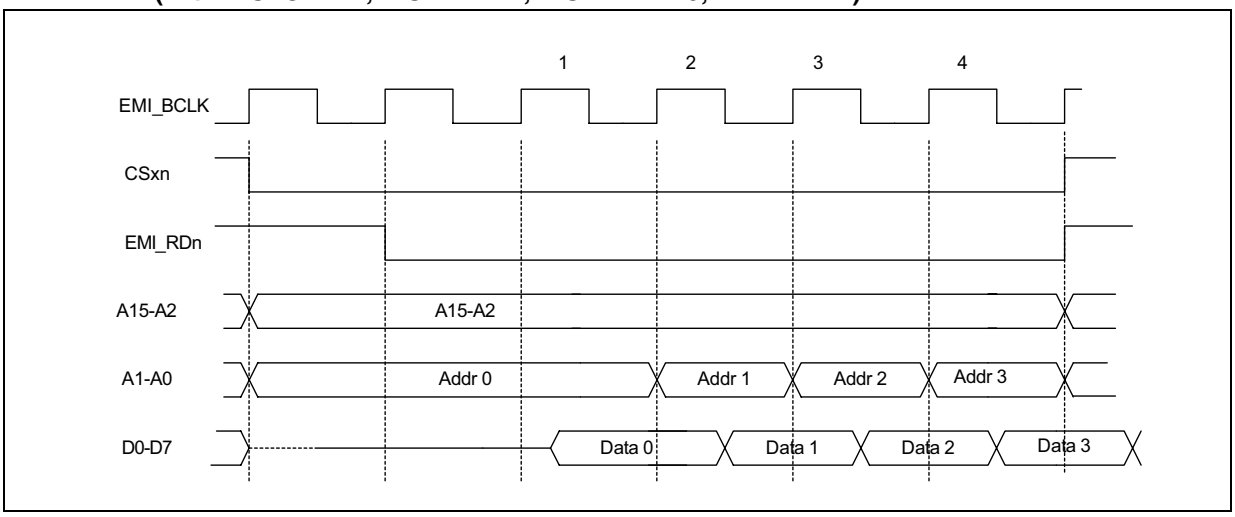
Page read mode for non-mux bus

The Non-mux EMI bus supports asynchronous page reads to four or eight consecutive locations. Page mode is enabled by setting the mode bits in the EMI_BCRx register. This feature increases the bus bandwidth by using a reduced access time for the sequential reads after the initial asynchronous read. The chip select lines CSx and EMI_RDn are held low during the page access, and only the low address changes between subsequent accesses. At the end of the page read, the CSx chip select and EMI_RDn lines are terminated at the same time. A page read takes one BCLK clock period to complete, the BCLK clock frequency must be adjusted so as to meet the page access time of the memory device.

The basic configuration bits that are required for the asynchronous page mode include:

- Read Wait State (WSTRD, EMI_RCRx register)
- Output Enable Assertion Delay (WSTOEN, EMI_OECRx register)
- Burst Wait State (WSTBRD, EMI_BRDCRx register)
- Page mode selection (BPM, EMI_BCRx register bits 8)
- Memory width (MW, 8-bit, EMI_BCRx register bits 5:4)
- Asynchronous Read access (SyncReadDev, EMI_BCRx register bit 9)
- Page Read transfer length (BRLEN, EMI_BCRx register bits 11:10)

Figure 16. Asynchronous page mode read bus cycle (with WSTOE = 1, WSTRD = 2, WSTBRD = 0, BRLEN = 4)

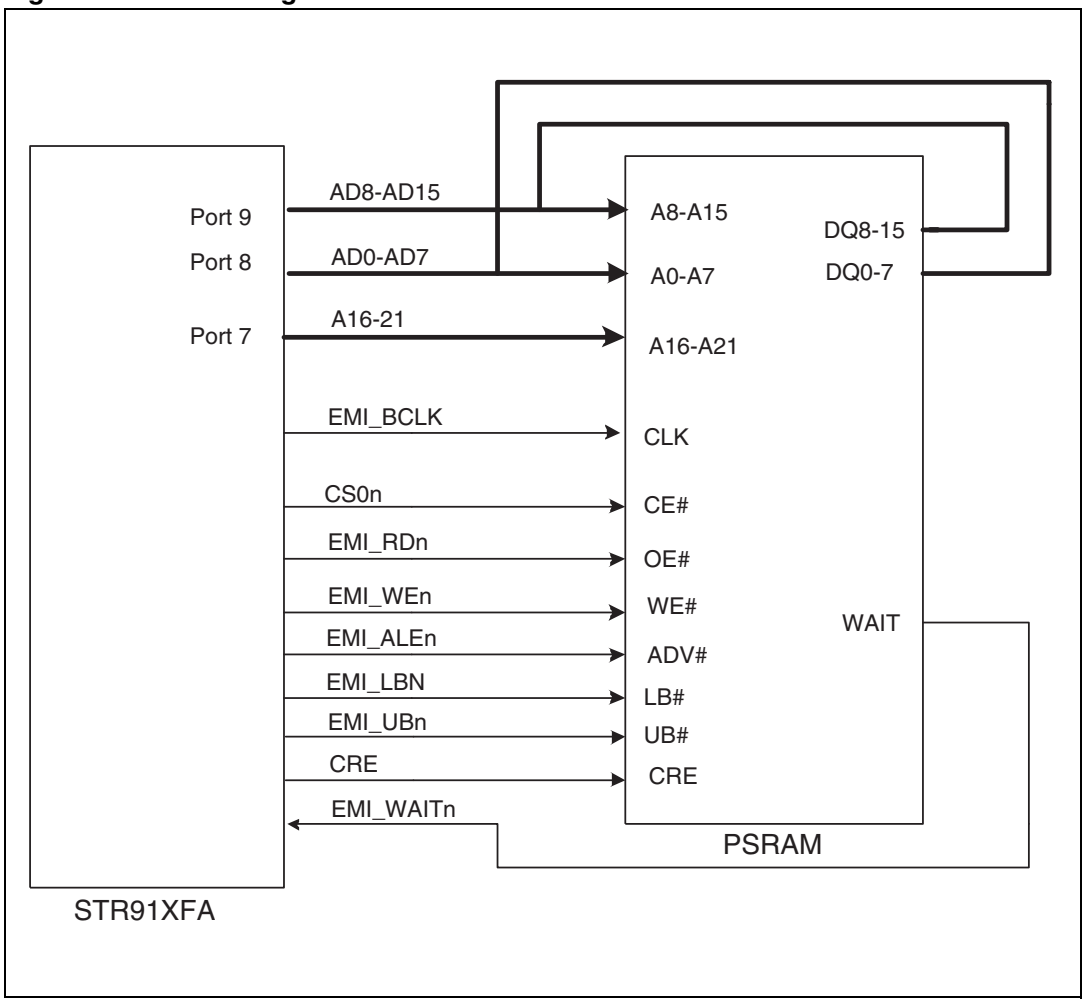


PSRAM mode (LFBGA 144 pin package only)

Figure 17 shows a "glue-less" bus interface between a STR91xFA and a Micron 64 Mb PSRAM (MT45W4MW16BCGB. In PSRAM mode, the EMI bus is configured as a 16 bit, multiplexed bus. The EMI_ALEn signal is programmed with a negative polarity and a 2 clock wide pulse width to meet the PSRAM's ADV# (address valid) timing requirement. The EMI address is latched by the PSRAM at the rising edge of the BCLK while the EMI_ALEn is low. In a read bus cycle, the EMI bus is tri-stated half clock after the trailing edge of the EMI_ALE signal. The PSRAM can then drives the bus when EMI_RDn becomes active. The PSRAM mode bit (bit 2) in the *GPIO external memory interface register (SCU_EMI) on page 111* must be set to 1 to enable the EMI_UBn and EMI_LBn signals.

The EMI bus can access the PSRAM memory array in asynchronous mode or in burst mode. However, the EMI bus must be in asynchronous mode when writing to the PSRAM bus configuration register. The CRE signal, which is required to be high when writing to the PSRAM configuration register, can be connected to any GPIO output pin and the signal logic level is controlled by the firmware.

Figure 17. EMI Bus "glue-less" interface to PSRAM



PSRAM mode control signals

The EMI bus supports synchronized burst read and write bus cycle in "PSRAM mode". The additional EMI signals provided in the LFBGA package that support the burst mode are:

- EMI_BCLK : the bus clock output. The EMI_BCLK has the same frequency or half of that of the HCLK. By default the clock is enabled after an EMI bus cycle but can be disabled by the user.
- EMI_WAITn : the not ready or wait input signal for synchronous access only
- EMI_BAAn : burst address advance or burst enable signal
- EMI_WEn : write enable signal
- EMI_UBn, EMI_LBn : upper byte and lower byte enable signals. These two signals share the same pins as the EMI_WRHn and EMI_WRLn and are user configurable through bit 2 in the *GPIO external memory interface register (SCU_EMI) on page 111*. In typical application, the EMI_WEn signal is needed to work with the byte enable signals to write to a 16 bit memory device.

By defining the bus parameters such as burst length, burst type, read and write wait states in the bus control registers, the multiplexed EMI bus is able to interface directly to standard PSRAM memory device

PSRAM burst read mode

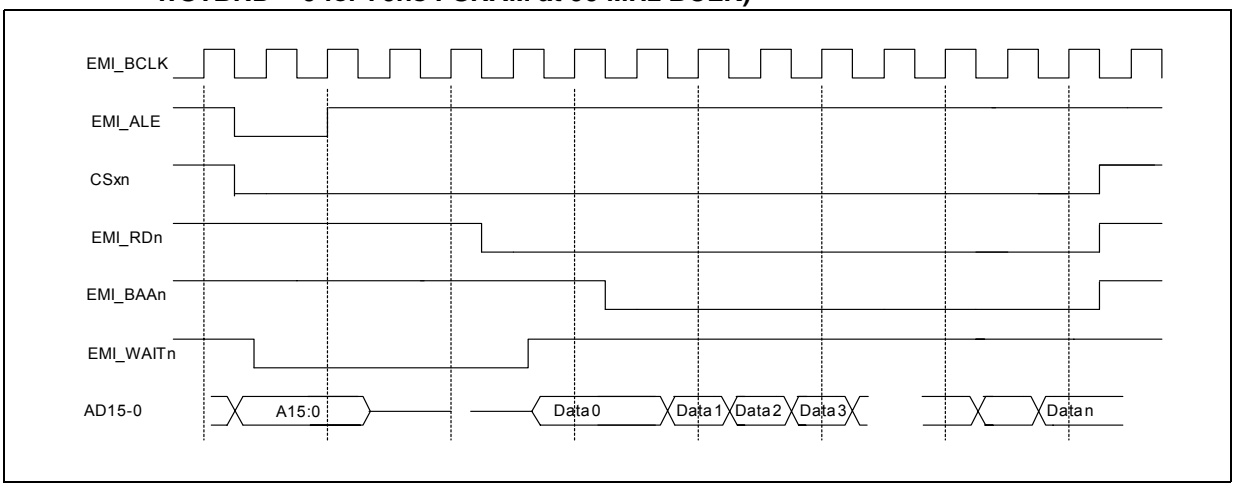
PSRAM mode supports synchronous burst read. The burst length can be specified by the user to be 4, 8, 16 or continuous transfer. Burst Read mode is enabled by setting the mode bits in the EMI_BCRx Register. This burst feature increases the bus bandwidth by reading one memory location per BCLK clock. The burst mode cycle consists of a first read access followed by synchronized burst reads. The read wait state (WSTRD) defines the first read access time and the subsequent burst is defined by the burst wait state (WSTBRD). The EMI bus can achieve one transfer per BCLK clock with WSTBRD set to 0.The chip select CSx and EMI_RDn are held low during the burst access. The burst address advance (EMI_BAA) is asserted once the bus starts burst transfer. For some devices, the EMI_BAA signal is not needed. At the end of the burst read the CSx chip select and EMI_RDn lines are terminated at the same time.

Note: The EMI control signals in the synchronous PSRAM mode are activated on the falling edge of the EMI_BCLK and the EMI_ALE signal width is truncated by half BCLK period.

The basic configuration bits that are required for the PSRAM synchronous burst read mode include:

- Read Wait State (WSTRD, EMI_RCRx register)
- Output Enable Assertion Delay (WSTOEN, EMI_OECRx register)
- Burst Wait State (WSTBRD, EMI_BRDCRx register)
- Burst mode selection (BPM, EMI_BCRx register bits 8)
- Memory width (EMI_BCRx register bits 5:4)
- Synchronous Read access (SyncReadDev, EMI_BCRx register bit 9)
- Burst Read transfer length (BRLEN, EMI_BCRx register bits 11:10)

Figure 18. PSRAM synchronous burst read bus cycle (with WSTOE = 4, WSTRD = 5, WSTBRD = 0 for 70ns PSRAM at 96 MHz BCLK)



PSRAM burst write mode

In PSRAM mode, the EMI bus support synchronous burst writes; burst length can be 4, 8, or continuous transfer. Burst Write mode is enabled by setting the mode bits in the EMI_BCRx Register. This feature increases the bus bandwidth by writing one memory location per BCLK clock. The burst mode access consists of an initial first access followed by synchronized burst writes. The write wait state (WSTWR) defines the first access time and the subsequent write is one per each BCLK clock. The chip select CSx and EMI_WEn are held low during the burst access. The burst address advance (EMI_BAA) is active once the bus starts burst transfer. For some memory devices, the EMI_BAA signal is not needed to advance the address internally. At the end of the burst write the CSx chip select and EMI_WEn lines are terminated at the same time.

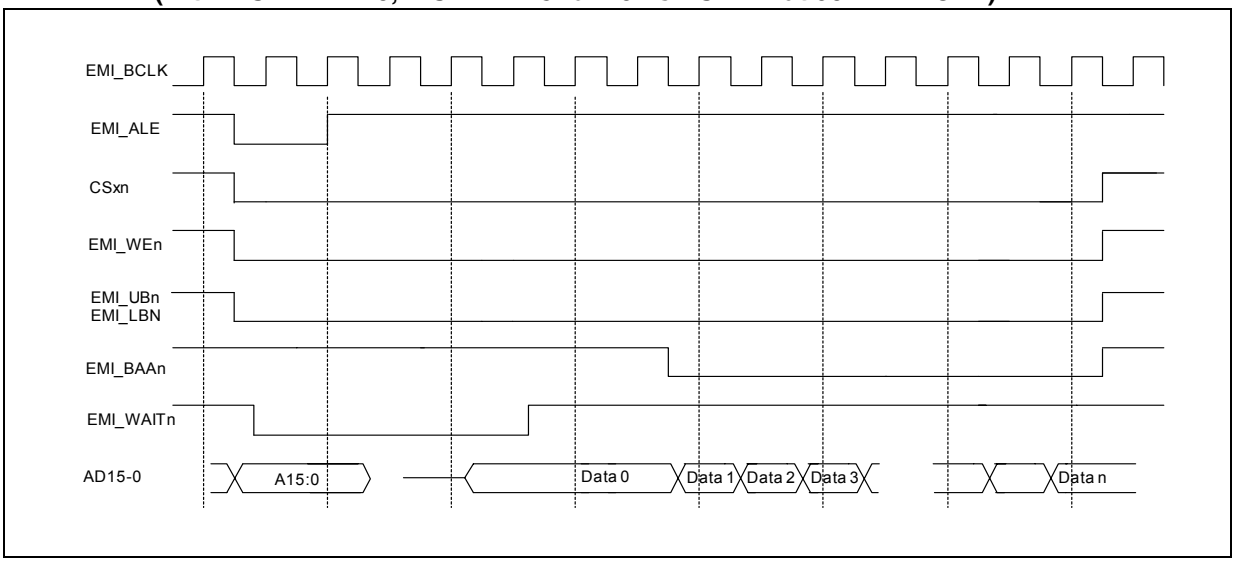
Note: The EMI control signals in the synchronous PSRAM mode are activated on the falling edge of the EMI_BCLK and the EMI_ALE signal width is truncated by half BCLK period.

The EMI_WEn write enable signal timing in burst mode is synchronized with the rising edge of the BCLK clock, while in asynchronous mode it is synchronized with the falling edge of BCLK clock.

The basic configuration bits that are required for the synchronous burst write mode include:

- Write Wait State (WSTWR, EMI_WCRx register)
- Write Enable Assertion Delay (WSTWEN, EMI_WECRx register)
- Burst Write mode selection (BMWrite, EMI_BCRx register bits 16)
- Memory width (EMI_BCRx register bits 5:4)
- Synchronous Write access (SyncWriteDev, EMI_BCRx register bit 17)
- Burst Write transfer length (BWLEN, EMI_BCRx register bits 19:18)

Figure 19. PSRAM synchronous burst write bus cycle (with WSTWEN = 0, WSTWR = 5 for 70 ns PSRAM at 96 MHz BCLK)



Synchronous external wait control

Burst transfers can be delayed by the EMI_Waitn input signal which is connected to the PSRAM device's Wait or Ready pin. The memory device must be programmed such that the EMI_Waitn is asserted in the cycle before the delay is to apply.

The PSRAM device can use the EMI_Waitn signal to indicate that the current burst read transfer is delayed, for example when crossing an address boundary. You can assert the EMI_Waitn input synchronously at any time, but the signal must be de-asserted a cycle before the read data is valid.

1.12.8 Register description

In this section, the following abbreviations are used:

Read/write (rw)	Software can read and write to these bits

Bank x idle cycle control register (EMI_ICRx)

Address offset: 00h (Bank 1), 20h (Bank 2), 40h (Bank 3), E0h (Bank 0)

Reset value: 0000 000Fh

7	6	5	4	3	2	1	0
Reserved				IDCY[3:0]			
				rw	rw	rw	rw

Bits 31:4	Reserved, must be kept at zero
Bits 3:0	**IDCY[3:0]:** *Idle Cycles* The value written in this field defines the number of idle or bus turnaround cycles added between read and write accesses to prevent bus contention on the external memory bus. Turnaround time = 2+IDCY x t_{BCLK} . The valid IDCY range is from 3 to Fh, and the reset value is Fh (15).

Bank x read wait state control register (EMI_RCRx)

Address offset: 04h (Bank 1), 24h (Bank 2), 44h (Bank 3), E4h (Bank 0)

Reset value: 0000 001Fh

7	6	5	4	3	2	1	0
Reserved			WSTRD[4:0]				
			rw	rw	rw	rw	rw

Bits 31:5	Reserved, must be kept at zero
Bits 4:0	**WSTRD[4:0]:** *Read Wait states* The value written in this field defines the number of wait states for read accesses to SRAM and ROM. The reset value is 1Fh (31). Wait state time = WSTRD x t_{BCLK}.

Bank x write wait state control register (EMI_WCRx)

Address offset: 08h (Bank 1), 28h (Bank 2), 48h (Bank 3), E8h (Bank 0)

Reset value: 0000 001Fh

7	6	5	4	3	2	1	0
Reserved			WSTWR[4:0]				
			rw	rw	rw	rw	rw

Bits 31:5	Reserved, must be kept at zero
Bits 4:0	**WSTWR[4:0]:** *Write Wait states* The value written in this field defines the number of wait states for write accesses. The reset value is 1Fh (31). Wait state time = WSTWR x t_{BCLK}.

Bank x output enable control register (EMI_OECRx)

Address offset: 0Ch (Bank 1), 2Ch (Bank 2), 4Ch (Bank 3), ECh (Bank 0)

Reset value: Banks 1, 2 and 3 = 0000 0000h

Reset value: Bank 0 = 0000 0003h

7	6	5	4	3	2	1	0
Reserved				WSTOEN[3:0]			
				rw	rw	rw	rw

Bits 31:4	Reserved, must be kept at zero
Bits 3:0	**WSTOEN[3:0]:** *Output Enable Assertion Delay* The value written in this field defines the Output Enable assertion delay from chip select assertion. The reset value is 1.

Bank x write enable control register (EMI_WECRx)

Address offset: 10h (Bank 1), 30h (Bank 2), 50h (Bank 3), F0h (Bank 0)

Reset value: 0000 0001h

7	6	5	4	3	2	1	0
Reserved				WSTWEN[3:0]			
				rw	rw	rw	rw

Bits 31:4	Reserved, must be kept at zero
Bits 3:0	**WSTWEN[3:0]:** *Write Enable Assertion Delay* The value written in this field defines the Write Enable assertion delay from chip select assertion. The reset value is 1.

Bank x control register (EMI_BCRx)

Address offset: 14h (Bank 1), 34h (Bank 2), 54h (Bank 3), F4h (Bank 0)

Reset value: 0030 3010h - Bank 0
0030 3020h - Bank 1
0030 3000h - Bank 2
0030 3010h - Bank 3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										1	1	BWLEN		SYNC WRITE DEV	BM WRITE
										rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	BRLEN[1:0]		SYNC READ DEV	BPM	Reserved		MW[1:0]		WP	Reserved		BLE
rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw			rw

Bits 31:22	Reserved, must be kept at reset value
Bits 21:20	Reserved, do not modify, read as 1
Bits 19:18	**BWLEN[1:0]:** *Burst write transfer length* These bits are used to set the number of sequential transfers supported by the burst device for a write: 00: 4-transfer burst write (default) 01: 8-transfer burst write 10: Reserved 11: Continuous burst (synchronous only)
Bit 17	**SYNCWRITEDEV**: *Synchronous write access device* This bit must be set to access the device using synchronous accesses for writes: 0: Asynchronous device (default) 1: Synchronous device
Bit 16	**BMWRITE:** *Burst mode write* This bit is set and cleared by software to select burst or non-burst write to memory. 0: Non-burst writes to memory devices (default at reset) 1: Burst mode writes to memory devices
Bit 15:14	Reserved, do not modify, read as 0, write as 0
Bits 13:12	Reserved, do not modify, read as 1

Bits 11:10	**BRLEN[1:0]:** *Burst Read Transfer Length* These bits are written by software to define the transfer length for burst or page mode read cycle. Page mode is limited to 4 or 8 transfer. 00: 4-transfer burst read 01: 8-transfer burst read 10: 16-transfer burst read 11: Continuous (synchronous only)
Bit 9	**SYNCREADDEV:** *Synchronous read access device* Access the device using synchronous accesses for reads: 0: Asynchronous device (default). 1: Synchronous device.
Bit 8	**BPM:** *Burst and Page Mode Read Selection* This bit is set and cleared by software to select/deselect Burst or Page Mode read. 0: Normal mode 1: Burst or Page Mode Read. (Page Mode is supported only when the EMI bus is configured as an 8-bit non-mux bus)
Bits 7:6	Reserved, must be kept at reset value.
Bits 5:4	**MW[1:0]:** *Memory width* These bits are written by software to define the memory width of the bank. The bits must be set to match the EMI data bus width configuration. 00: 8-bit 01: 16-bit 10: Reserved 11: Reserved
Bit 3	**WP:** *Write protect* This bit is set and cleared by software to protect/unprotect the bank from write access. 0: Bank not write protected 1: Bank write protected
Bits 2:1	Reserved, must be kept at reset value
Bit 0	**BLE:** *Byte Lane Enable* This bit enables the byte select signals in 16-bit PSRAM bus mode. 0: Byte Select signals are not enabled 1: Byte Select signals (EMI_UBn and EMI_LBn) are enabled. Bit 2 in the GPIO EMI register (SCU_EMI) must also be set to 1.

Bank x burst read wait delay register (EMI_BRDCRx)

Address offset: 1Ch (Bank 1), 3Ch (Bank 2), 5Ch (Bank 3), FCh (Bank 0)

Reset value: 0000 001Fh - Bank 0

7	6	5	4	3	2	1	0
Reserved			WSTBRD[4:0]				
			rw	rw	rw	rw	rw

Bits 31:5	Reserved, must be kept at reset value
Bits 4:0	**WSTBRD[4:0]:** *Burst read wait states* These bits are written by software to define the number of wait states for burst read accesses after the first read. They do not apply to non-burst devices. The value defaults to 1Fh at reset. Wait state time = WSTBRD x t_{BCLK}.

Clock control register (EMI_CCR)

Address offset: 204h

Reset value: 0000 0001h

7	6	5	4	3	2	1	0
Reserved							BCLKEN
							rw

Bits 31:1	Reserved, do not modify, read as zero, write as zero
Bit 0	**BCLKEN** *BCLK enable* This bit is set and cleared by software to configure the activation of BCLK (available in LBGA package). This setting affects all banks. 0: BCLK clock only active during bus access 1: BCLK clock always running (activated by first bus access). Requires bit 1 in the GPIO EMI register (SCU_EMI) to be 0.

1.12.9 EMI register map

The following table summarizes the EMI registers.

Table 7. EMI register map

Addr. offset	Register name	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	EMI_ICR1	Reserved																				IDCY			
04	EMI_RCR1	Reserved																				WSTRD			
08	EMI_WCR1	Reserved																				WSTWR			
0C	EMI_OECR1	Reserved																				WSTOEN			
10	EMI_WECR1	Reserved																				WSTWEN			
14	EMI_BCR1	Reserved				BWLEN		SYNCWRITEDEV	BM WRITE	0	WRAPREAD	1	1	BRLEN[1:0]		SYNCREADDEV	BPM	Reserved		MW[1:0]		WP	Reserved		BLE
18	Reserved																								
1C	EMI_BRDCR1	Reserved																				WSTBRD			
20	EMI_ICR2	Reserved																				IDCY			
24	EMI_RCR2	Reserved																				WSTRD			
28	EMI_WCR2	Reserved																				WSTWR			
2C	EMI_OECR2	Reserved																				WSTOEN			
30	EMI_WECR2	Reserved																				WSTWEN			
34	EMI_BCR2	Reserved				BWLEN		SYNCWRITEDEV	BM WRITE	0	WRAPREAD	1	1	BRLEN[1:0]		SYNCREADDEV	BPM	Reserved		MW[1:0]		WP	Reserved		BLE
38	Reserved																								
3C	EMI_BRDCR2	Reserved																				WSTBRD			
40	EMI_ICR3	Reserved																				IDCY			
44	EMI_RCR3	Reserved																				WSTRD			
48	EMI_WCR3	Reserved																				WSTWR			
4C	EMI_OECR3	Reserved																				WSTOEN			
50	EMI_WECR3	Reserved																				WSTWEN			
54	EMI_BCR3	Reserved				BWLEN		SYNCWRITEDEV	BM WRITE	0	WRAPREAD	1	1	BRLEN[1:0]		SYNCREADDEV	BPM	Reserved		MW[1:0]		WP	Reserved		BLE
58	Reserved																								
5C	EMI_BRDCR3	Reserved																				WSTBRD			
60-DF	Reserved																								

Table 7. EMI register map (continued)

Addr. offset	Register name	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E0	EMI_ICR0	Reserved																				IDCY			
E4	EMI_RCR0	Reserved																				WSTRD			
E8	EMI_WCR0	Reserved																				WSTWR			
EC	EMI_OECR0	Reserved																				WSTOEN			
F0	EMI_WECR0	Reserved																				WSTWEN			
F4	EMI_BCR0	Reserved				BWLEN		SYNCWRITEDEV	BM WRITE	0	WRAPREAD	1	1	BRLEN[1:0]		SYNCREADDEV	BPM	Reserved		MW[1:0]		WP	Reserved		BLE
F8	Reserved																								
FC	EMI_BRDCR0	Reserved																				WSTBRD			
100-203	Reserved																								
204	EMI_CCR	Reserved																							BCLK EN

Refer to *Table 5 on page 35* for the register base addresses.

Refer to the System Controller Unit chapter for the EMI control bits in the SCU Configuration register description.

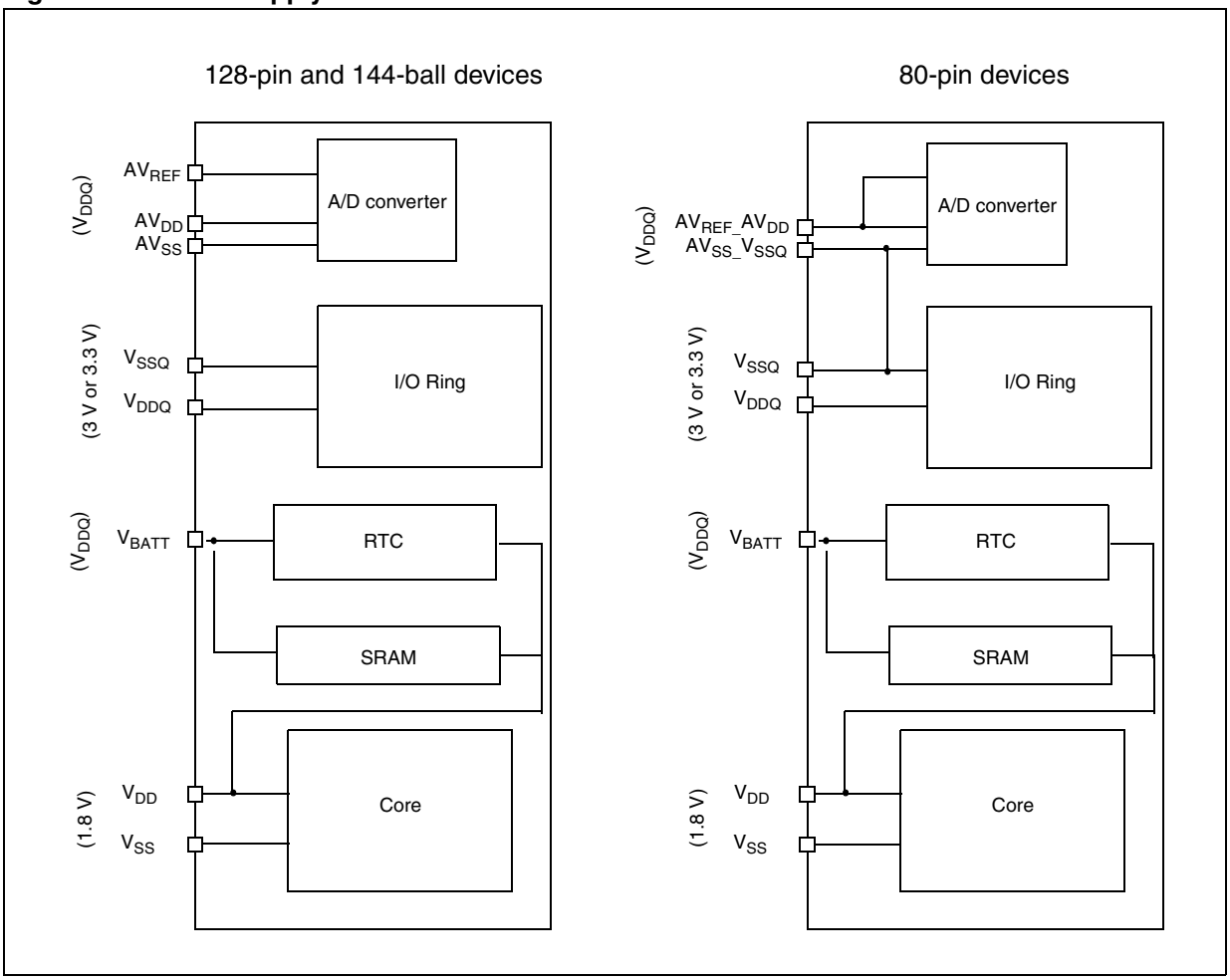
2 Power, reset and clocks

2.1 Power supply

2.1.1 Main operating voltages

The STR91xFA requires two separate operating voltage supplies. The CPU and memories operate from a 1.65 V to 2.0 V on the VDD pins, and the I/O ring operates at 2.7 V to 3.6 V on the VDDQ pins.

Figure 20. Power supply overview



2.1.2 Independent A/D converter supply and reference voltage

To improve conversion accuracy, the ADC has an independent power supply which you can separately filter and shield from noise in the PCB.

On 128-pin, 144-ball packages:

- The ADC voltage supply input is on a separate AVDD pin
- An isolated supply ground connection is provided on pin AVSS
- You can connect a separate external reference voltage input for the ADC on the AVREF pin for better accuracy on low voltage inputs.

On 80-pin packages

The ADC voltage supply is tied internally to the ADC reference voltage pin AVCC_AVREF and the analog ground is shared with the digital ground at a single point, on pin AVSS_VSSQ.

2.1.3 Battery backup

An optional stand-by voltage from a battery or other source may be connected to pin VBATT to retain the contents of SRAM in the event of a loss of the main digital supplies (V_{DD} and V_{DDQ}). The SRAM will automatically switch its supply from the internal VDD source to the VBATT pin when the V_{DD} and V_{DDQ} voltage drops below the LVD threshold (and V_{BAT} remains above the threshold).

Note: In order to use the battery supply, the LVD must be enabled

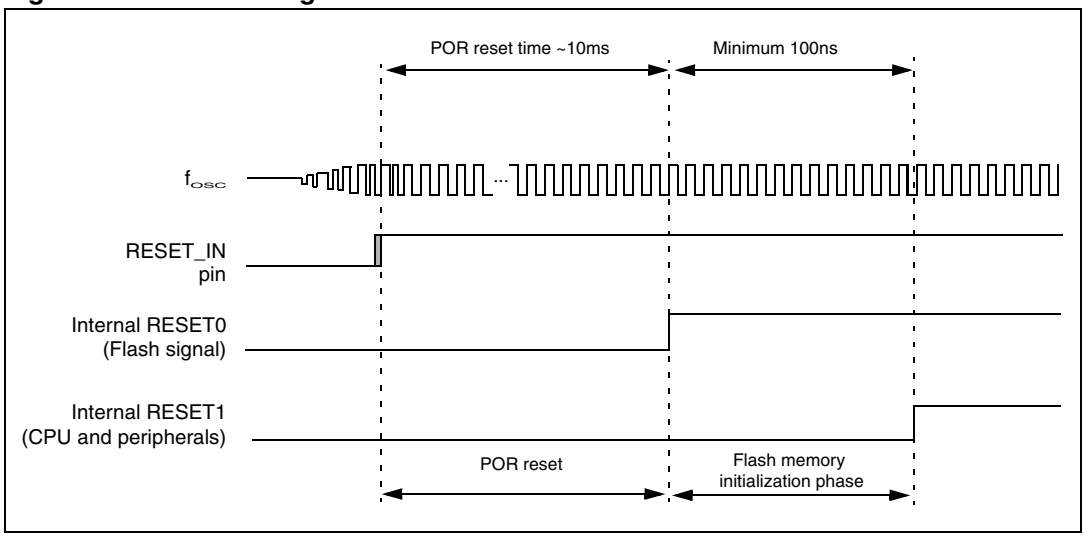
The VBATT pin also supplies power to the RTC unit, allowing the RTC to function even when the main digital supplies (V_{DD} and V_{DDQ}) are switched off. By programming the device configuration via JTAG, you can select to power only the RTC or both the RTC and the SRAM from VBATT.

2.1.4 Power-up

The LVD circuitry will always generate a global reset when the STR91xFA powers up, meaning internal reset is active until V_{DDQ} and V_{DD} are both above the LVD thresholds. This POR condition has a duration of t_{POR}, after which the CPU will fetch its first instruction from address 0x0000.0000.

Figure 21 shows the reset timing.

Figure 21. Reset timing



2.2 Reset

There are two types of reset generated internally, defined as System Reset and Global Reset.

2.2.1 System reset

A system reset is generated when one of the following events occurs:

1. A low level on the RESET_INn pin (External Reset)
2. Watchdog end of count condition (WDG Reset)
3. JTAG Reset Command (JTAG reset).

A system reset sets all registers to their reset values except the *Clock control register (SCU_CLKCNTR)*, *PLL configuration register (SCU_PLLCONF)*, *System status register (SCU_SYSSTATUS)* and some RTC registers.

Note: In earlier silicon revisions, prior to Rev H, the FMI Bank address and Bank size registers are also set by a system reset.

2.2.2 Global reset

A global reset is generated when one of the following events occurs:

1. A voltage drop below internal LVD threshold (LVD Reset)
2. Power On Reset (POR reset), which has the same behavior as the LVD Reset

A global reset sets all the registers to their reset values (except some RTC registers).

2.2.3 Reset flags

An LVD or Watchdog reset is flagged in the *System status register (SCU_SYSSTATUS)* and an interrupt request to the VIC is generated when either flag is set. You can read these flags to determine the source of the last reset as shown in *Table 8.*

Table 8. Reset flags

WDG_RST bit	LVD_RST bit	Meaning
0	0	An External Reset or JTAG Reset occurred (system reset)
0	1	An LVD reset or POR occurred (Global reset)
1	0	A watchdog reset occurred (System reset)

2.2.4 Reset peripherals (software reset)

Through the *Peripheral reset register 0 (SCU_PRR0)* and *Peripheral reset register 1 (SCU_PRR1)*, it is possible to force the reset for each peripheral.

2.2.5 Reset output

The RESET_OUT pin can be used to reset other application components when a system or global reset occurs. It is an ORed output of all reset sources: system resets and global resets. Each of the reset has its own time duration, refer to the data sheet on their timings. The RESET_OUT pin is a push-pull pin with 4mA drive.

2.3 Low voltage detector

Voltage dropout: The LVD circuit monitors V_{DD} (1.8 V), and V_{DDQ} 3.0 V (or 3.3 V) supplies and generates a global reset whenever either voltage drops below the configured V_{DD_LVD} and V_{DDQ_LVD} levels. If the MCU was reset by the LVD, this is flagged in the *System status register (SCU_SYSSTATUS)* and an interrupt request to the VIC is generated if enabled.

Voltage brownout: You can also program the LVD to generate an Early Warning interrupt when either voltage drops below the V_{DD_BRN} and V_{DDQ_BRN} thresholds. The Early Warning event signal is connected to the VIC1.7 interrupt channel. Software can manage the Early Warning interrupt using the VIC1.7 channel bits in the VIC registers.

Configuration

You can configure the LVD by programming the non-volatile configuration bits via JTAG as described in the STR91xFA JTAG/ISP programming specification. There are three bits:

- The LVD_th bit selects the LVD threshold. Configure the 2.4 V threshold for applications with 3 V V_{DDQ} 3 V or a 2.7 V threshold if V_{DDQ} is 3.3 V.
- The LVD_RESET_SELECT bit selects if an LVD reset is triggered on the V_{DD} threshold only or on both V_{DD} and V_{DDQ}.
- The LVD RESET WARNING bit selects if an Early Warning interrupt is triggered on the V_{DD} threshold only or on both V_{DD} and V_{DDQ}.

The LVD circuit consumes current in power down mode. In certain low power applications this may not be desirable. The LVDEN bit in the Flash Configuration Register allows you to turn off the LVD circuit before power down mode and turn it back on later. This is a volatile bit and is cleared (LVD enabled) after reset. You can configure it by software via the Flash Memory CUI (Command user interface). Refer to the Flash memory interface (FMI) section for details of this register.

Note: When the LVD is turned off, the VBAT feature is not supported.

The LVD logic consists of a lower power voltage band gap that provide an accurate voltage reference. This voltage reference is used to create the voltage threshold levels that are compared with the supply voltages.

When either voltage supply falls below the threshold for that supply, the LVD generates a global reset.

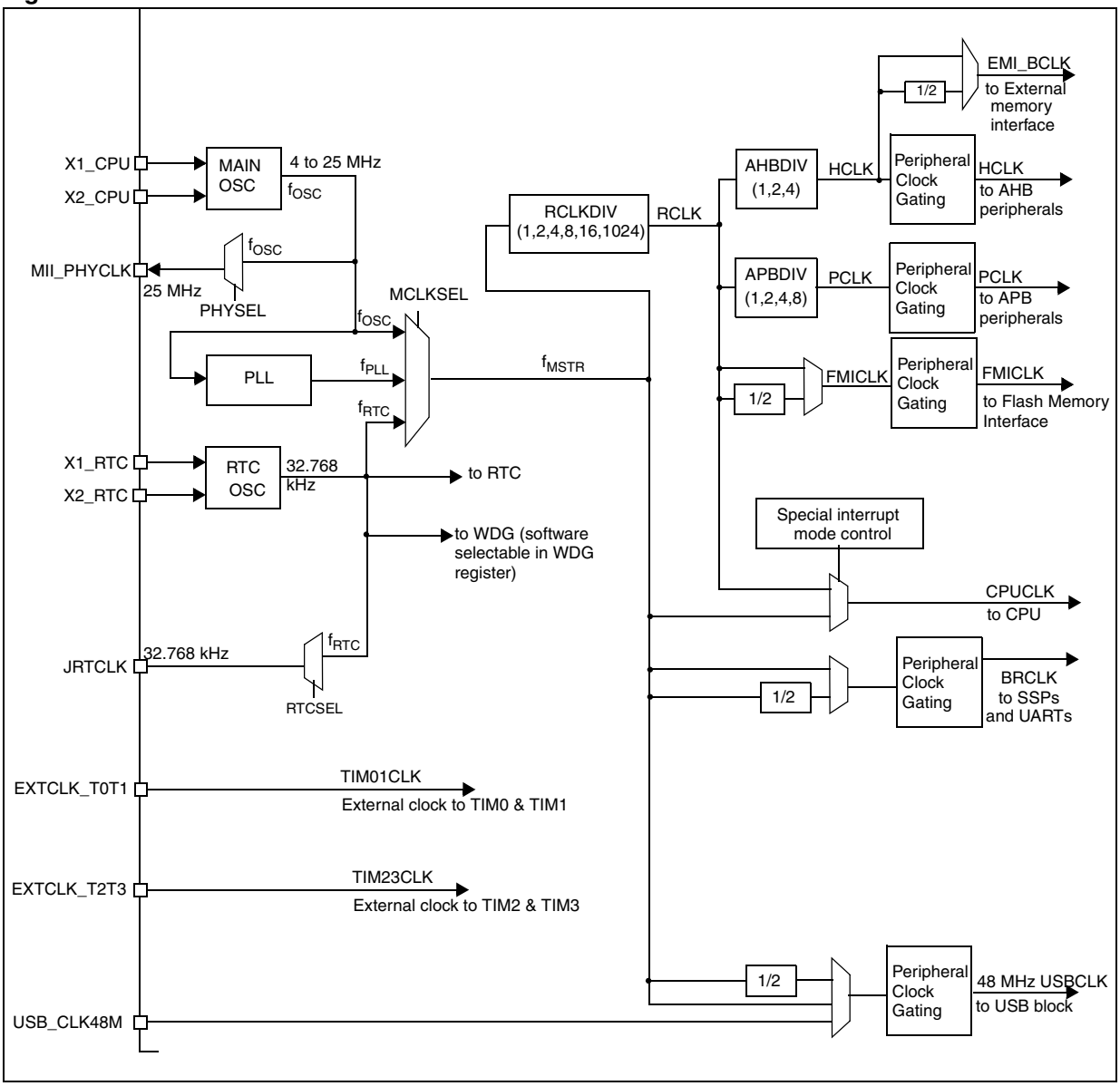
2.4 Clocks

2.4.1 External clock sources

The system controller has the following four external clock sources:

1. **f_{OSC}:** A 4 to 25 MHz oscillator provides the main operating clock for all on-chip functional blocks.
2. **f_{RTC}:** The RTC has an independent 32.768 kHz crystal. The RTC keeps on running even when the CPU is in power down or power off mode. This slow RTC clock can also be used in power management.
3. f_{USB}: You can optionally configure this as a 48 MHz input clock to the USB. It is needed when the PLL is configured to generate a clock that cannot be shared by the USB. The PLL is able to generate a 48 or 96 MHz clock from the 25 MHz input crystal for internal use by selecting the appropriate multiplier and divider.
4. **f_{TIMEXT}:** The TIM Timer/counters can run on the internal peripheral clock or the external input clock. You select this by programming the TIM01SEL and TIM23SEL bits in the *Clock control register (SCU_CLKCNTR)*. These clock can be gated through the Peripheral Clock Gating Registers (see *Section 2.4.12*). When these pins are not used as clock inputs, they can be configured as GPIO.

Figure 22. Clock control



2.4.2 Master clock (f_{MSTR})

The master clock (f_{MSTR}) has three clock sources that you select using the MCLKSEL[1:0] bits in the *Clock control register (SCU_CLKCNTR)*. The clock sources are the PLL output, the oscillator input pin and the RTC clock:

- The f_{PLL} output frequency is programmable, typical frequency is 48 MHz, 66 MHz or 96 MHz (maximum). When power consumption is critical, you can disable the PLL and run the microcontroller directly from the external clock (RTC clock or Oscillator).
- The f_{OSC} oscillator input clock has a frequency of 4 to 25 MHz. This input clock can be sourced by a crystal or an oscillator.
- f_{RTC} is a 32.768 kHz input. You can program the application to run from this slow clock when you want to save power.

You can choose the source to match the CPU performance and the power management requirements of your application. Transitions from one clock to another are glitch-free and do not disrupt any on-going activities.

2.4.3 Flash memory interface clock (FMICLK)

The FMICLK clock is an internal clock derived from RCLK and with the same frequency. You can optionally divide it by 2 by setting the FMI_SEL bit in the *Clock control register (SCU_CLKCNTR)*. FMICLK can be gated through the Peripheral Clock Gating Registers (see *Section 2.4.12*).

2.4.4 UART and SSP clock (BRCLK)

BRCLK is an internal clock derived from f_{MSTR} that is used to drive the two SSP peripherals and to generate the baud rate for the three on-chip UART peripherals. You can optionally divide the frequency by 2 by setting the BR_SEL bit in the *Clock control register (SCU_CLKCNTR)*. BRCLK can be gated through the Peripheral Clock Gating Registers (see *Section 2.4.12*).

2.4.5 External memory interface clock (BCLK)

You can select the frequency of the EMI bus clock (BCLK) to be HCLK or HCLK/2 using the EMIRATIO bit in the *Clock control register (SCU_CLKCNTR)*. By default the frequency is HCLK/2. The BCLK clock is available on the LFBGA package as an output pin. You can disable the BCLK output by setting the BCLK_EN bit in the EMI register (SCU_GPIOEMI).

2.4.6 USBCLK

The USB clock can be derived from f_{MSTR} when the frequency is 48 MHz or 96 MHz. If you use another f_{MSTR} frequency, the 48 MHz USBCLK must be sourced from the external pin (GPIO pin). You select this using the USB_SEL[1:0] bits in the *Clock control register (SCU_CLKCNTR)*. USBCLK can be gated through the Peripheral Clock Gating Registers (see *Section 2.4.12*).

2.4.7 External RTC calibration clock

The RTC_CLK can be enabled as an output on the JRTCK pin by setting the Calibration Clock Output Enable bit in the RTC_CR register. The RTC_CLK is used for RTC oscillator calibration. The RTC_CLK is active in Sleep mode and can be used as a system wakeup control clock.

2.4.8 PHY clock output

MII_PHYCLK: This pin can be configured as a 25 MHz output clock for the Ethernet PHY interface. You enable the output clock using the MAC_SEL bit in the *Clock control register (SCU_CLKCNTR)*. This configuration requires f_{OSC} to be 25 MHz.

2.4.9 PLL

As shown in *Figure 22*, the oscillator input clock (f_{OSC}) is the input clock to the programmable PLL frequency multiplier.

When the PLL is active, it generates an output frequency (f_{PLL})according to the following equation:

$f_{PLL} = (2 \times N \times f_{OSC})/(M \times 2^P)$

Where the values of M, N and P must satisfy the following constraints:

$1 \leq M \leq 255$

$1 \leq N \leq 255$

$0 \leq P \leq 5$

$1\ MHz \leq f_{OSC}/M \leq 2\ MHz$

$200\ MHz \leq (2 \times N \times f_{OSC}) / M \leq 622\ MHz$

$4\ MHz \leq f_{OSC} \leq 25\ MHz$

You program the M, N and P values by writing to the *PLL configuration register (SCU_PLLCONF)*.

Care is required when programming the PLL multiplier and divider factors, not to exceed the maximum allowed operation frequency (96 MHz).

At power up, the CPU defaults to run on the oscillator clock, as the PLL is not ready (locked). The CPU can switch to the PLL clock only after the LOCK bit in the *System status register (SCU_SYSSTATUS)* is set. In Sleep mode, the PLL is turned off. When waking up from sleep mode if the f_{MSTR} is selected to run off the PLL, the CPU will wait until the LOCK bit is set before it starts to run.

The LOCK bit is set when the PLL clock has stabilized (locked status) and maintains this value as long as the PLL is locked. You can select the PLL clock as f_{MSTR} clock source only when the LOCK bit is 1. If the LOCK bit goes to 0 if for any reason, the PLL loses the programmed frequency in which it was locked. In this case, the LOCK_LOST bit is set and f_{MSTR} automatically switches back to f_{OSC}. f_{PLL} is restored as the f_{MSTR} source when the LOCK bit becomes 1 again.

The LOCK and LOCK_LOST events can be configured to generate interrupt requests to the VIC. See *Section 2.6.1: SCU interrupts*.

2.4.10 Changing the PLL configuration

While the CPU is running on the PLL clock, the PLL clock frequency can be changed by updating the SCU_PLLCONF register. You need to follow the steps below to change the clock:

1. Switch the CPU Master clock source to the OSC by setting bits [1:0] in the SCU_CLKCNTR register to “10”.
2. Write the new configuration to the SCU_PLLCONF register (write the new P, N and M values with the PLL_EN enable bit set to “0”).
3. The SCU_PLLCONF register will be updated after the clock has been switched to the OSC.
4. If you need the CPU to run at the new PLL clock frequency, write to the SCU_PLLCONF register again with the new P, N and M values AND the PLL_EN bit set to “1”.
5. Switch the CPU clock source back to the PLL clock by setting bit[1:0] in the SCU_CLKCNTR register to “00”.
6. The CPU Master clock will switch automatically from the OSC to the PLL once the LOCK bit is set. Do not initiate another SCU_PLLCONF register change before the LOCK bit is set.

2.4.11 Clock dividers

The main clock (f_{MSTR}) can be divided to operate at a slower frequency reference clock (RCLK) for the ARM core and all the peripherals. The RCLK provide the divided clock for the ARM core, and feeds the dividers for the AHB, APB, External Memory Interface, and FMI units.

You program the RCLK divider using the RCLKDIV[2:0] bits in the *Clock control register (SCU_CLKCNTR)*.

The AHB clock can be divided by 1, 2 or 4. The APB clock can be divided by 1, 2, 4, or 8. You program the PCLK and HCLK dividers using the APBDIV[1:0] and AHBDIV[1:0]bits in the *Clock control register (SCU_CLKCNTR)*.

2.4.12 Peripheral clock gating

After reset, only the CPU, the Flash memory, the SRAM and a small subset (see default values of *Peripheral clock gating register 0 (SCU_PCGR0)* and *Peripheral clock gating register 1 (SCU_PCGR1)* registers) of the peripherals start operating. The other parts of the system remain stopped. because the related PCGR bits are reset. To start them, you have to write 1 to the related register bit. You can stop the peripheral again, by writing 0 to the related bit.

This allows you to dynamically control the number of peripherals that are running which allows you to optimize the power used in a very flexible way.

The *Idle mode gating mask register 0 (SCU_MGR0) Idle mode gating mask register 1 (SCU_MGR1)* allow you to define a set of peripherals that are kept running when the microcontroller goes into Idle mode. In Sleep mode all peripherals except the RTC are turned off.

Clock gating in emulation mode

During the emulation mode (debug state of the ARM966E-S processor) the System Controller allows gating the clock of a peripheral or a group of peripherals. The software application can choose to stop the desired peripheral when ARM966E-S enters emulation mode. When you clear the related bit in the *Peripheral emulation clock gating register 0 (SCU_PECGR0)*, or *Peripheral emulation clock gating register 1 (SCU_PECGR1)*, the peripheral clock is gated in emulation mode.

2.5 Low power modes

The STR91xFA implements a configurable and flexible power management control that allows you to choose the best power option to fit your application. You can dynamically manage the power consumption or hardware to match the system's requirements.

Power management is provided via clock control for the CPU and individual clock control for the various peripherals.

The STR91xFA supports the following 4 global power control modes:

- Normal Run Mode
- Special Interrupt Run Mode
- Idle Mode
- Sleep Mode

Note: In the application development environment, a special mode (Debug state) is active during in-circuit emulation (ICE). In this mode, the clocks are never switched off when the ICE is in use even if the CPU enters Idle or Sleep Mode. In Idle Mode the CPU stops fetching instructions, but the ICE can override this state in order to run the debugger code.

Using the Flash_PD_DBG bit in the Power management register (SCU_PWRMNG) you can configure the Flash to enter power down mode when debug mode is active.

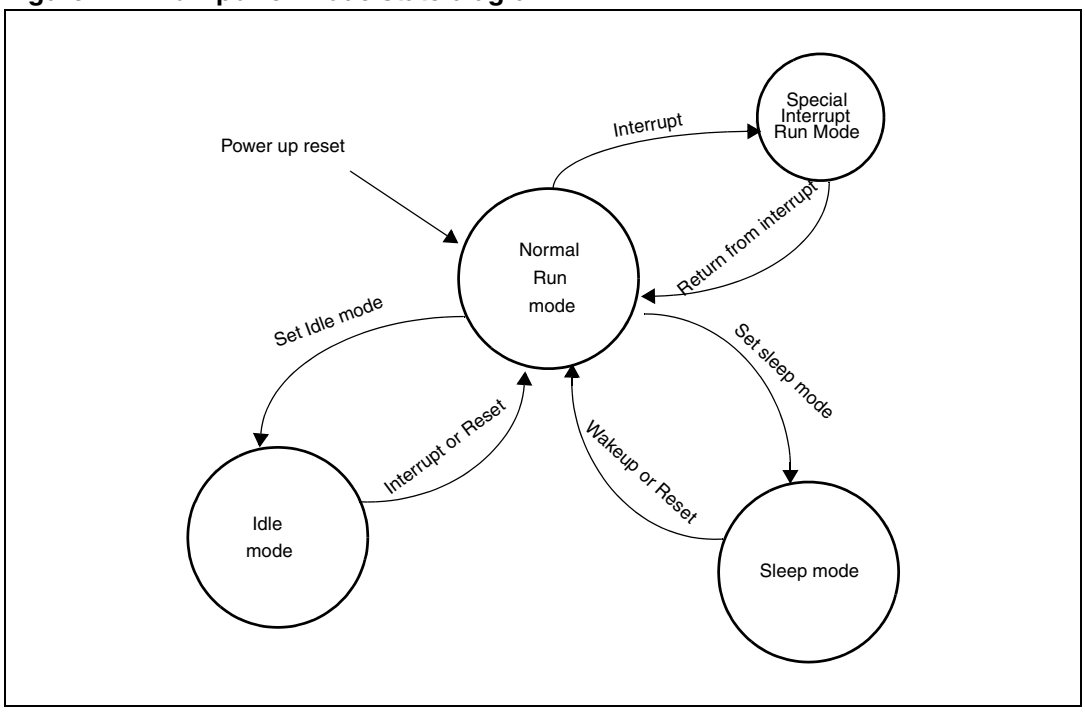
Figure 23. Comparison of power control modes

Power State	Clocks	Wakeup event	Description
Normal Run Mode	– All clocks are on – CPU is clocked by RCLK (divided by RCLKDIV)		– Peripherals active if enabled by the Peripheral Clock Gating Registers
Special Interrupt Run mode	– CPU is clocked by RCLK – While executing interrupt service routines, CPU is clocked by f_{MSTR} (RCLKDIV is bypassed)		
Idle Mode	– ARMCLK = Off – FMICLK = Off(1) – HCLK = On(1) – PCLK = On(1)	– External reset – WDG reset – Interrupts – RTC alarm – External wakeup	– CPU off – Peripherals active if enabled by the Peripheral Clock Gating Registers AND the corresponding bit is set in the Idle Mode Gating Mask Registers.
Sleep Mode	– ARMCLK = Off – FMICLK = Off – HCLK = Off – PCLK = Off	– External reset – External wakeup – RTC Alarm	– All clocks off except RTC – Flash memory in power down mode – PLL off – Oscillator pin (4-25 MHz) off

(1) The OFF and ON state must be configured in *Idle mode gating mask register 0 (SCU_MGR0)* and *Idle mode gating mask register 1 (SCU_MGR1)*

Figure 24 shows the power management state diagram (core not in Debug state).

Figure 24. Low power mode state diagram



2.5.1 Normal run mode

This is the default run mode. The CPU executes instructions and any or all of the on-chip peripherals are in active state. You can turn-on or turn-off the clock of any of the peripherals writing to *Peripheral clock gating register 0 (SCU_PCGR0)* or *Peripheral clock gating register 1 (SCU_PCGR1)*. You can also reduce the frequency (by means of clock dividers) of the various clocks in order to optimize power usage while operating in normal run mode.

2.5.2 Special interrupt run mode

Special Interrupt mode FIQ

The special interrupt mode using FIQ causes the CPU to temporarily operate at full speed (f_{MSTR} as clock frequency) while servicing one or more interrupts and return back to normal run mode with the speed selected by the clock RCLKDIV divider (see *Figure 22*) when the interrupt routine is complete. You enable/disable this mode using the CPU_INTR bit in the *Power management register (SCU_PWRMNG)*.

Special Interrupt mode IRQ

The special interrupt mode using IRQ causes the CPU to operate at full speed (f_{MSTR} as clock frequency) when the IRQ service routine reads the vector address register in the VIC and jumps then to the specified interrupt routine with the speed selected by the RCLKDIV clock divider. You enable/disable this mode using the CPU_INTR bit in the *Power management register (SCU_PWRMNG)*.

2.5.3 Idle mode

Idle Mode is entered under software control, by writing the value '001b' to the PWR_MODE[2:0] bits in the *Power management register (SCU_PWRMNG)*.

In this mode, the CPU suspends code execution. The CPU and FMI clocks are turned off. The various peripherals still continue to operate with their programmed clock rate if they are enabled by the related bits of the SCU_PCGRx and the SCU_MGRx registers. If the SCU_MGRx register bit is 0, when the system enters Idle mode, the related clock will be gated, otherwise the peripheral will continue to receive the clock if the PCGR bit is set.

To exit from Idle Mode, an interrupt must be generated by one of the active peripherals or from an external source:

- External reset or watchdog reset
- External or internal peripheral interrupt
- RTC alarm event
- Input from EXTINT pins (GPIO pins) via wakeup unit (WIU)

Idle Mode entry timing

The time required to enter Idle mode depends both on the oscillator clock, on the CPUCLK clock and on the slowest clock set for the peripherals coming out of the Clock Control Unit (see *Table 10* and *Figure 22*) according to the following equation:

$t_{IDLE} = 17 * (t_{OSC}) + 14 * (t_{SLOWEST\_PERIPH\_CLK}) + 6 * (t_{CPUCLK})$

If the WDG clock is RTC then $t_{SLOWEST\_PERIPH\_CLK} = t_{RTC}$

Idle Mode exit timing

On Idle Mode wake-up, all the clock sources, in particular the oscillator pad (4-25 MHz), automatically turn on. This procedure is controlled by the same State Machine inside the Power Management Unit (PMU). From Idle Mode, an internal temporization is used for Flash recovery (1250 Oscillator clock cycle) before the CPU fetch. When the clock oscillators are enabled, all peripherals have to send an acknowledgement back to the PMU that the clock is enabled. However, as soon as the CPU clock is enabled, the code starts.

For this reason, it is recommend that the PMU state machine is turned back to its READY state, before selecting a new Low Power Mode entry.

To ensure this time recovery, *Equation 1* should be used.

Equation 1

$$t_{RECOVERY} = 17 \times (t_{OSC}) + 14 \times (t_{SLOWEST\_PERIPH\_CLK}) + 6 \times (t_{CPUCLK})$$

2.5.4 Sleep mode

Sleep mode is entered under software control, by writing the value '010b' to the PWR_MODE[2:0] bits in the *Power management register (SCU_PWRMNG)*.

This is the lowest power mode of MCU. In this mode, all clock circuits (except RTC) and the oscillator pin (4-25 MHz) are turned off. In this mode, the CPU does not continue to execute any instructions. All peripherals except the RTC have their clocks stopped. The ARM Flash Memory is put in power down mode at the same time as the ARM MCU. The ARM MCU when enters into the Power Down mode, generates a PD signal to the Flash Memory. The Flash memory take at minimum 50µs of recovery time to resume operation on wakeup from sleep mode. The system clock is switched on only after the recovery time is over.

To exit from Sleep mode, one of the following events must occur:

- External reset (RESET_INn pin
- External interrupt via wakeup unit (WIU)
- RTC alarm

When a wakeup interrupt occurs, the MCU will start up all the clocks, respond to the interrupt and then continue normal execution.

Sleep Mode exit timing

Refer to *Idle Mode exit timing*.

2.5.5 Sleep mode and Idle mode configuration considerations

When enabling Sleep or Idle mode, certain requirements must be met to ensure the proper operation of the low power modes. The following sections describe these requirements when entering or exiting Sleep or Idle mode.

Code execution after entering Sleep mode and Idle mode

Once Idle mode or Sleep Mode are entered by writing the PWR_MODE[2:0] bits in the Power management register (SCU_PWRMNG) it takes about 12 crystal oscillator cycles (X1_CPU input frequency) for the device before stopping the execution.

In order to avoid executing any valid instructions after the Idle or Sleep bit setting and before entering the mode, it is mandatory to execute a certain number of dummy instructions after the Power management register setting.

The number of dummy instructions to be executed depends on the ratio between the CPU clock frequency and the oscillator input frequency according to the following:

N_dummy_Instr = (fcpuclk/fosc_x1)*12 if (fcpuclk/fosc_x1)>=1

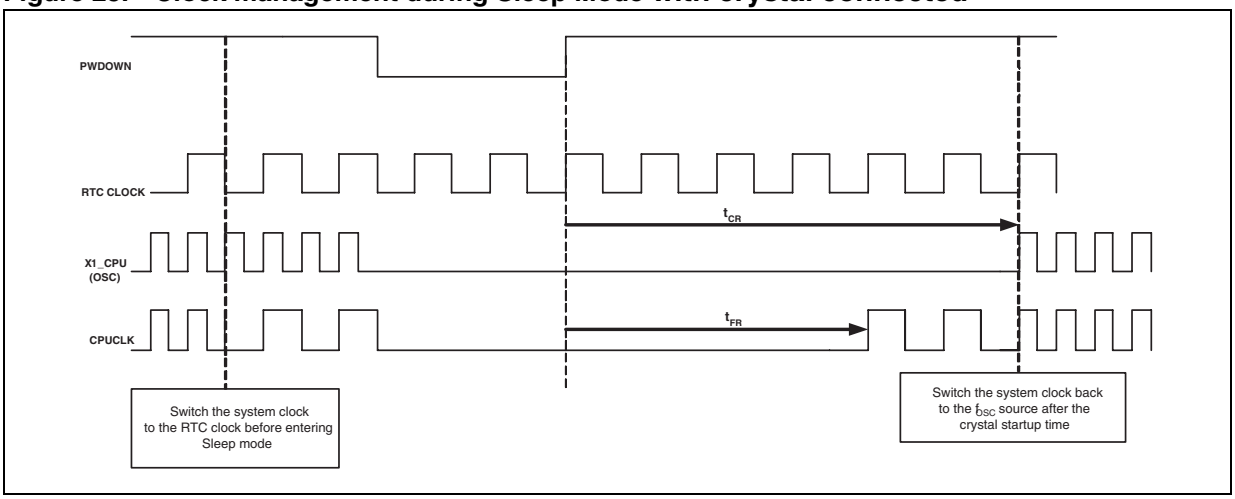
N_dummy_Instr = 3 if(fcpuclk/fosc_x1)<1

The worst case is represented by the core working out of the PLL maximum frequency (96 MHz) with an 4 MHz crystal or oscillator on the X1 inputs. In that case 288 dummy instructions would be needed.

Sleep Mode with a crystal connected to X1_CPU input

In order to cut the power consumption due to oscillation, the crystal inputs are disabled during Sleep Mode. During recovery from Sleep Mode the oscillator will take a start-up time to re-start the oscillation (refer to the data sheet on the start-up time). For this reason, when a crystal is connected to the crystal inputs, the system clock source must be switched to the RTC clock before entering Sleep Mode. After waking up, the CPU runs on the RTC clock and needs to wait until the crystal start-up time elapses before switching back to the oscillator or PLL clock.

Figure 25. Clock management during Sleep Mode with crystal connected



Sleep mode with an oscillator connected to X1_CPU input

In this case the oscillation will restart right away after the X1 inputs are re-enabled and it is not necessary to switch to RTC clock before entering Sleep Mode

Sleep mode with the PLL used as system clock source

If the oscillation on the X1_CPU inputs is generated by a crystal the software has to explicitly switch the system clock source to the RTC clock before entering Sleep Mode and, when exiting Sleep Mode, switch back to the PLL only after the crystal start-up time.

During Sleep Mode the PLL is automatically turned off and the system clock is automatically switched to the oscillator input. On exit from Sleep Mode, the system clock will go back to the PLL clock only after the PLL is locked.

If the oscillation on the X1_CPU input is generated by an oscillator no action needs to be taken since the PLL is automatically turned off, the system clock is automatically connected to the oscillator clock and changed back to the PLL clock after exiting from Sleep Mode once the PLL is locked.

Figure 26. Clock management during Sleep mode with crystal and PLL

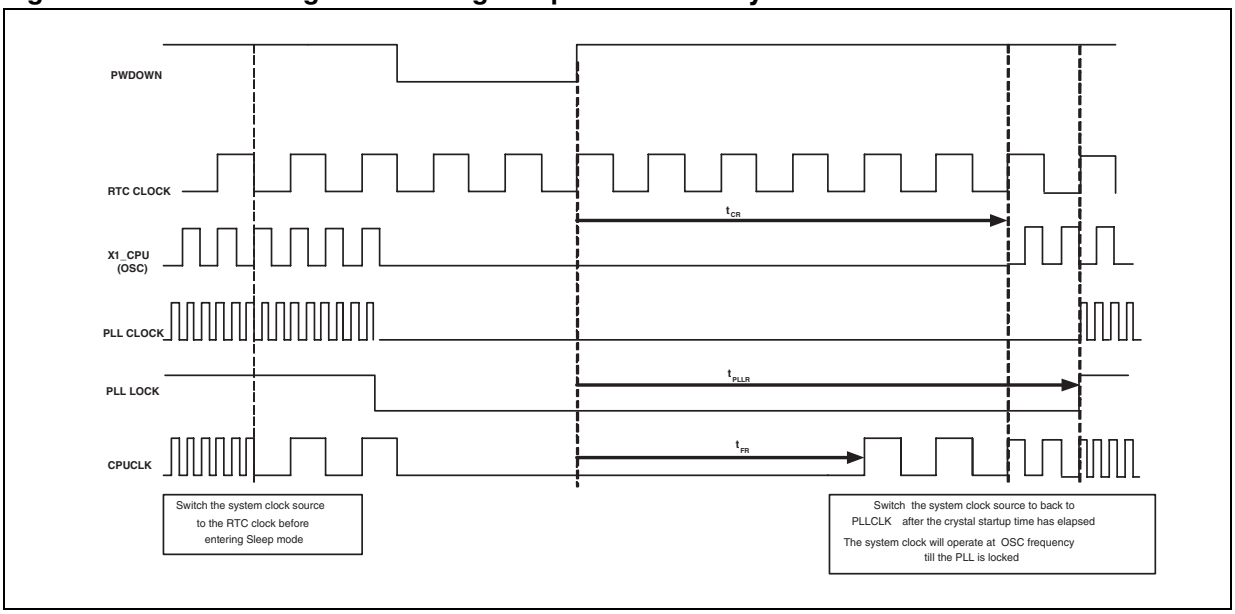


Table 9. Sleep mode wakeup time for PLL, Flash and crystal

Parameter	Description	Min	Typ	Max
t_{CR}	Crystal start-up time		1.5 ms	
t_{FL}	Flash memory recovery time	50 µs		
t_{PLLR}	Lock-in time from PLL enable		0.3 ms	1.5 ms

Sleep Mode entry timing

When Sleep mode is selected, all the clock circuits and the oscillator pad (4-25 MHz) are turned off. This procedure is controlled by a dedicated State Machine inside the Power Management Unit (PMU), in order to switch off the clocks safely.

During the Sleep Mode sequence there are many interactions between the Power Management Unit (PMU) and the Clock Control Unit (CCU). In particular, when the low power mode is set, a signal is asserted to gate the peripheral clocks. As response to this signal, all the peripherals have to send back to the PMU the acknowledgement that the clock was shut-off.

The setting or enabling of the peripheral clocks depends on the Clock Control register (SCU_CLKCNTR) configuration and on the EE bit setting in the the Watchdog clock control register (selecting APB or RTC clock as the Watchdog clock source).

Table 10. CCU output clocks that determine the entry time (t_{SLEEP})

CCU_OUT	Description	Control Register
BRCLK	Baud Rate Clock to the UART	SCU_CLKCNTR
TIMO1CLK	Timer 0-1 clock	SCU_CLKCNTR
TIM23CLK	Timer 2-3 clock	SCU_CLKCNTR
EMICLK	EMI clock	SCU_CLKCNTR
FMICLK	FMI clock	SCU_CLKCNTR
WDG	Watchdog clock	WDG_CR (EE)
HCLK	AHB clock	SCU_CLKCNTR
PCLK	APB clock	SCU_CLKCNTR
CPUCLK	ARM core clock	SCU_CLKCNTR
USBCLK	USB Clock	SCU_CLKCNTR

As a result the time required to enter Sleep mode depends both on the oscillator clock, on the CPUCLK clock and on the slowest clock set for the peripherals coming out of the Clock Control Unit (see *Table 10* and *Figure 22*) according to the following equation:

t_{SLEEP} = 17 * (t_{OSC}) + 14 * ($t_{SLOWEST\_PERIPH\_CLK}$)+ 6 * (t_{CPUCLK})

During this t_{SLEEP} time a wakeup input will be ignored.

If the WDG clock is RTC then $t_{SLOWEST\_PERIPH\_CLK}$= t_{RTC}

Example 1

t_{OSC} =40ns (@ 25 MHz)

t_{CPUCLK} =20.83ns (@48 MHz)

All clock dividers in the default state except APB clock divided by
4 ' ($t_{SLOWEST\_PERIPH\_CLK}$)=4* t_{CPUCLK}

t_{SLEEP} = 17 * 40 ns + 14 * (4 * 20.83 ns)+ 6 * 20.83 ns = 1971.46 ns

Example 2

t_{OSC} = 40 ns (@ 25 MHz)

t_{CPUCLK} = t_{RTC} = 31,250 ns (@ 32 kHz)

All clock dividers in the default state except APB clock divided by

2 ' ($t_{SLOWEST\_PERIPH\_CLK}$) = 2 * t_{CPUCLK}

t_{SLEEP} = 17 * 40 ns + 14 * 2 * 31,250 ns + 6 * 31,250 ns ~ 1.06 ms

Example 3

t_{OSC} = 40 ns (@ 25 MHz)

t_{CPUCLK} =20.83 ns (@ 48 MHz)

All clock dividers in the default state except APB clock divided by 4 'PCLK=4* t_{CPUCLK}

WDG working out of the RTC clock '($t_{SLOWEST\_PERIPH\_CLK}$)=t_{RTC}

t_{SLEEP} = 17 * 40 ns + 14 * (31,250 ns) + 6 * 20.83 ns ~ 0.43 ms

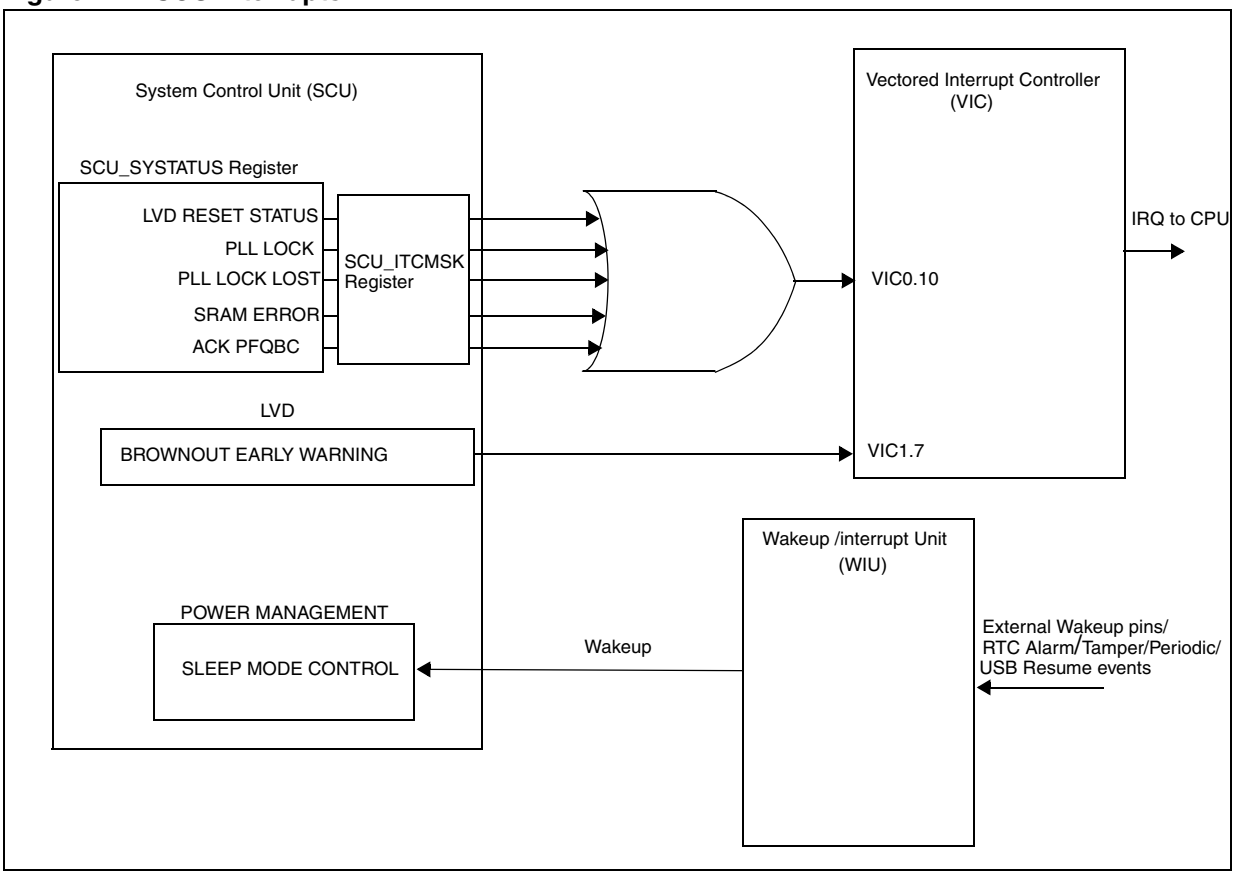
2.6 System control unit (SCU)

The System Control Unit (SCU) provides the control logic for the STR91xFA power, reset and clocks as described in the previous *Section 2*. The SCU also controls a large number of other miscellaneous features described in *Section 2.6.2* to *Section 2.6.7*

2.6.1 SCU interrupts

The SCU interrupt sources are mapped on two channels of the vectored interrupt controller VIC as shown in *Figure 27*.

Figure 27. SCU Interrupts



2.6.2 SRAM configuration/control

AHB/DTCM arbiter wait states

Using the WSR_DTCM bit in the *System configuration register 0 (SCU_SCR0)*, you can select the number of wait states inserted when reading the SRAM through the DTCM.

Using the WSR_AHB bit in the *System configuration register 0 (SCU_SCR0)*, you can select the number of wait states inserted when reading the SRAM through the AHB.

Size selection

Using the SRAM_SIZE bit in the *System configuration register 0 (SCU_SCR0)*, you can select the size of the SRAM (32 Kb, 64 Kb, 96 Kb).

Lock transfer enable

Using the SRAM_LK_EN bit in the *System configuration register 0 (SCU_SCR0)*, you can enable AHB Lock transfer for SRAM arbiter.

2.6.3 PFQ/BC configuration/control

Using the EN_PFQBC bit in the *System configuration register 0 (SCU_SCR0)*, you can enable/disable the PFQ/BC unit

2.6.4 External memory interface (EMI) configuration/control

Mux/Demux mode

Using the EMI_MUX bit in the *System configuration register 0 (SCU_SCR0)*, you can select EMI Mux mode or Demux mode.

ALE length

Using the EMI_ALE_LENGTH bit in the *System configuration register 0 (SCU_SCR0)*, you can select EMI ALE length to be 1 or 2 cycles.

ALE Polarity

Using the EMI_ALE_POL bit in the *System configuration register 0 (SCU_SCR0)*, you can select EMI ALE polarity to be active high or active low.

Refer to *Section 1.12* for more information.

2.6.5 UART configuration/control

Using the UART_IRDA[2:0] bits in the *System configuration register 0 (SCU_SCR0)*, you can configure the three UART peripherals individually in UART or IrDA mode.

2.6.6 Port 3.0 ETM trigger or external debug request selection

Using the P30_SEL_EDBG and the EXT_EMT_EDBGR bits in the *System configuration register 0 (SCU_SCR0)*, you can set up GPIO port 3.0 as the ETM trigger or External debug request input.

2.6.7 System control unit GPIO registers

GPIO Pins on P0 thru P7 have multiple input and output alternate functions. You select these using the System Control Unit (SCU) registers. SCU registers are also used to select open collector or Push-Pull operation and to configure Port 4 pins for use as analog inputs.

GPIO Pins on P8 thru P9 are only multiplexed with EMI and have no SCU output or input control registers.

All ports have SCU_GPIOTYPE registers for selecting Open Collector or Push/Pull configuration.

2.6.8 ADC Fast trigger conversion in single mode

Using the ACG bits in the *GPIO analog mode register (SCU_GPIOANA)*, you can configure the ADC in automatic clock gating mode between each trigger start command.

Refer to *Section 16.4.5* for more information.

Note: This feature is not available in silicon Rev G and earlier revs (see datasheet for silicon revision information).

2.6.9 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.
Read/clear (rc_w1)	Software can read as well as clear this bit by writing 1. Writing ‘0’ has no effect on the bit value.
Read/set (rs)	Software can read as well as set this bit. Writing ‘0’ has no effect on the bit value.
Write only (wo)	Software can only write to this bit. Reading the bit returns the reset value.

Note: In the register description, n = 0 to 7 and m = 0 to 9.

Clock control register (SCU_CLKCNTR)

Address offset: 00h

Reset value: 0002 0002h (see note)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.													EMIRATIO [1:0]		FMI SEL
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM23 SEL	TIM01 SEL	PHY SEL	USBSEL [1:0]		BR SEL	APBDIV [1:0]		AHBDIV [1:0]		RCLKDIV[2:0]			MCLK SEL[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:19	Reserved, always read as 0
Bits 18:17	**EMIRATIO[1:0]:** *External Memory Interface ratio* These bits are written by software to define the ratio of EMI bus clock (f_{BCLK}) and HCLK (default 0h) 00: f_{BCLK}=HCLK 01: f_{BCLK}=HCLK/2 (default) 10: Reserved 11: Reserved
Bit 16	**FMISEL:** *Flash Memory Interface Clock Divider* This bit is set and cleared by software. It enables/disables the FMICLK divider. 0: FMICLK=RCLK 1: FMICLK=RCLK/2
Bit 15	Reserved, always read as 0
Bit 14	**TIM23SEL:** *Timers 2 and 3 external clock enable* This bit is set and cleared by software. It enables the TIM23 external clock source for TIM2 and TIM3. 0: External clock disabled 1: TIM23 external clock enabled (from EXTCLK_T2T3 pin GPIO P2.5)
Bit 13	**TIM01SEL:** *Timers 0 and 1 external clock enable* This bit is set and cleared by software. It enables the TIM01 external clock source for TIM0 and TIM1. 0: External clock disabled 1: TIM01 external clock enabled (from EXTCLK_T0T1 pin GPIO P2.4)
Bit 12	**PHYSEL:** *MII_PHYCLK Enable* This bit is set and cleared by software. It enables the 25 MHz PHY output clock 0: MII_PHYCLK output disabled (default) 1: f_{OSC} output on MII_PHYCLK pin GPIO P5.2

Bits 11:10	**USBSEL[1:0]:** *USB 48 MHz Clock Selection* These bits are written by software to select the source of the 48 MHz clock input to the USB block 00: f_{MSTR} (default) 01: f_{MSTR} divided by 2 10: External Clock from USB_CLK48M pin GPIO P2.7 11: Reserved
Bit 9	**BRSEL:** *Baud Rate Clock Selection* This bit is set and cleared by software. It selects the clock source for BRCLK 0: f_{MSTR} divided by 2 1: f_{MSTR}
Bits 8:7	**APBDIV[1:0]:** *PCLK divider* These bits are written by software to define the PCLK divider 00: PCLK=RCLK (default) 01: PCLK=RCLK divided by 2 10: PCLK=RCLK divided by 4 11: PCLK=RCLK divided by 8
Bits 6:5	**AHBDIV[1:0]:** *HCLK divider* These bits are written by software to define the HCLK divider 00: HCLK=RCLK (default) 01: HCLK=RCLK divided by 2 10: HCLK=RCLK divided by 4 11: Reserved
Bits 4:2	**RCLKDIV[2:0]:** *RCLK divider* These bits are written by software to define the RCLK divider 000: RCLK=f_{MSTR} (default) 001: RCLK=f_{MSTR} divided by 2 010: RCLK=f_{MSTR} divided by 4 011: RCLK=f_{MSTR} divided by 8 100: RCLK=f_{MSTR} divided by 16 101: RCLK=f_{MSTR} divided by 1024 **Note**: Other values are reserved.
Bits 1:0	**MCLKSEL[1:0]:** *Main Clock Source* These bits are written by software to select the source of f_{MSTR}. Refer to 00: f_{MSTR}=f_{PLL} 01: f_{MSTR}=f_{RTC} 10: f_{MSTR}=f_{OSC}(default) 11: Reserved

Note: This register is set to reset value only after a Power on Reset. A system reset does not reset the register.

PLL configuration register (SCU_PLLCONF)

Address offset: 04h

Reset value: 0003 C019h (see note)

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

Bits 31:20	Reserved, always read as 0
Bit 19	**PLL_EN:** *PLL enable* This bit is set and cleared by software. 0: PLL disabled (default) 1: PLL enabled
Bits 18:16	**PLL_PDIV[2:0]:** *PLL Post-divider* These bits are written by software to define the PLL Post-divider (default 3h). Refer to *Section 2.4.9 on page 73* for more details
Bit 15:8	**PLL_NDIV[7:0]:** *PLL Feedback divider* These bits are written by software to define the PLL Feedback divider (default C0h). Refer to *Section 2.4.9 on page 73* for more details.
Bit 7:0	**PLL_MDIV[7:0]:** *PLL Pre-divider* These bits are written by software to define the PLL Pre-divider (default 19h). Refer to *Section 2.4.9 on page 73* for more details

Note: 1 With the default value the PLL generates an output frequency after reset of 48 MHz when the input frequency is 25 MHz.

2 This register is set to the reset value only after a Power on Reset. A system reset does not reset the register.

3 Writing to the SCU_PLLCONF register while the CPU clock is running on the PLL clock will not be updated and will be ignored.

4 The PLL_EN bit in the SCU_PLLCONF register must be cleared (disabled) when the N, M and P dividers are being updated.

System status register (SCU_SYSSTATUS)

Address offset: 08h

Reset value: 0000 0018h (after an LVD reset)
This register is set to reset value only after a power on. A system reset does not reset the register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										SRAM_ ERR	ACK_ PFQBC	LVD_ RST	WDG_ RST	LOCK_ LOST	LOCK
										rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:6	Reserved, always read as 0
Bit 5	**SRAM_ERR:** *SRAM Error event flag* This bit is set by hardware when a write error occurs (i.e. the ARM tries to write to an invalid location). This flag is used for debug purposes. If the related mask bit in the SCU_ITCMSK register is 0, an interrupt request is sent to the interrupt controller (VIC). Write 1 in order to clear this flag. 0: No SRAM_ERR event 1: An SRAM_ERR event occurred
Bit 4	**ACK_PFQBC:** *ACK PFQBC event flag* This bit is set by hardware when an acknowledge is received in response to software setting the EN_PFQBC bit in the SCU_SCR0 register. If the related mask bit in the SCU_ITCMSK register is 0, an interrupt request is sent to the interrupt controller (VIC). Write 1 in order to clear this flag. 0: No ACK_PFQBC event 1: An ACK_PFQBC event occurred
Bit 3	**LVD_RST:** *LVD Reset event flag* This bit is set by hardware when the system comes out of reset after an LVD reset has occurred. Write 1 in order to clear this flag. 0: No event 1: An LVD reset event occurred
Bit 2	**WDG_RST:** *WDG Reset event flag* This bit is set by hardware when the system comes out of reset after an WDG reset event has occurred. Write 1 in order to clear this flag. 0: No event 1: An WDG reset event occurred
Bit 1	**LOCK_LOST:** *LOCK LOST event flag* This bit is set by hardware when the PLL has lost the lock with the reference clock. If the related mask bit in the SCU_ITCMSK register is 0, an interrupt request is sent to the interrupt controller (VIC). Write 1 in order to clear this flag. 0: No LOCK_LOST event 1: An LOCK_LOST event occurred

Bit 0	**LOCK:** *LOCK event flag* This bit is set by hardware when the PLL is locked. If the related mask bit in the SCU_ITCMSK register is 0, an interrupt request is sent to the interrupt controller (VIC). Write 1 in order to clear this flag. 0: PLL not locked 1: PLL locked

Power management register (SCU_PWRMNG)

Address offset: 0Ch

Reset value: 0000 0000h

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

Bits 31:5	Reserved, always read as 0
Bit 4	**FLASH_PD_DBG:** *Flash Power Down in Debug Mode* This bit is set and cleared by software to select if the Flash goes into power down mode in debug mode. 0: The Flash is not to enter power down mode during debug mode (default) 1: The Flash is set to enter power down mode during debug mode
Bit 3	**CPU_INTR:** *Special Interrupt mode* This bit is set and cleared by software to select the clock speed in interrupt mode. 0: Interrupt code executes at the speed defined by the RCLKDIV[2:0] bits in the SCU_CLKCNTR register (default) 1: Interrupt code executes at full speed (bypassing RCLKDIV)
Bit 2:0	**PWR_MODE[2:0]:** *Power Mode control bits* These bits are written by software to put the microcontroller in the selected power mode. 000: Run mode (default) 001: Idle mode 010: Sleep mode ***Note*:** Other values are reserved.

Interrupt mask register (SCU_ITCMSK)

Address offset: 10h

Reset value: 0000 001Fh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved											MSK_LVD_RST	MSK_SRAM_ERR	MSK_ACK_PFQBC	MSK_LOCK_LOST	MSK_LOCK
											rw	rw	rw	rw	rw

Bits 31:5	Reserved, always read as 0
Bit 4	**MSK_LVD_RST:** *LVD Reset interrupt mask* This bit is set and cleared by software to enable or disable the LVD reset interrupt (refer to SCU_SYSSTATUS register). 0: LVD reset not masked 1: LVD reset masked (default)
Bit 3	**MSK_SRAM_ERR:** *SRAM Error interrupt mask* This bit is set and cleared by software to enable or disable the SRAM error interrupt (refer to SCU_SYSSTATUS register). 0: SRAM_ERR interrupt not masked 1: SRAM_ERR interrupt masked (default)
Bit 2	**MSK_ACK_PFQBC:** *ACK PFQBC interrupt mask* This bit is set and cleared by software to enable or disable the ACK_PFQBC interrupt (refer to SCU_SYSSTATUS register). 0: ACK_PFQBC interrupt not masked 1: ACK_PFQBC interrupt masked (default)
Bit 1	**MSK_LOCK_LOST:** *LOCK LOST interrupt mask* This bit is set and cleared by software to enable or disable the LOCK_LOST interrupt (refer to SCU_SYSSTATUS register). 0: LOCK_LOST interrupt not masked 1: LOCK_LOST interrupt masked (default)
Bit 0	**MSK_LOCK:** *LOCK interrupt mask* This bit is set and cleared by software to enable or disable the LOCK interrupt (refer to SCU_SYSSTATUS register). 0: LOCK interrupt not masked 1: LOCK interrupt masked (default)

Peripheral clock gating register 0 (SCU_PCGR0)

Address offset: 14h

Reset value: 0000 00DBh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				MAC	USB 48M	USB	DMA	EXT_MEM _CLK	EMI	VIC	SRAM _ARBITER	SRAM	Res.	PFQ BC	FMI
				rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Bits 31:12	Reserved, always read as 0
Bit 11	**MAC:** *Ethernet peripheral clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the Ethernet peripheral clock. 0: Ethernet peripheral clock stopped 1: Ethernet peripheral clock running
Bit 10	**USB48M:** *USB 48 MHz clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the 48 MHz USB clock. 0: 48 MHz USB clock stopped 1: 48 MHz USB clock running
Bit 9	**USB:** *USB peripheral clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the USB peripheral. 0: USB peripheral clock stopped 1: USB peripheral clock running
Bit 8	**DMA:** *DMA peripheral clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the DMA peripheral. 0: DMA peripheral clock stopped 1: DMA peripheral clock running
Bit 7	**EXT_MEM_CLK:** *External memory clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the external memory clock. 0: External memory clock stopped 1: External memory clock running
Bit 6	**EMI:** *EMI peripheral clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the EMI peripheral. 0: EMI peripheral clock stopped 1: EMI peripheral clock running
Bit 5	**VIC:** *VIC peripheral clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the Vectored Interrupt Controller. 0: VIC peripheral clock stopped 1: VIC peripheral clock running

Bit 4	**SRAM_ARBITER:** *SRAM arbiter clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the SRAM arbiter clock. 0: SRAM arbiter clock stopped 1: SRAM arbiter clock running
Bit 3	**SRAM:** *SRAM clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the SRAM clock. 0: SRAM clock stopped 1: SRAM clock running
Bit 2	Reserved, always read as 0
Bit 1	**PQFBC:** *PQFBC clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the Prefetch Queue/Branch Cache clock. 0: PQFBC clock stopped 1: PQFBC clock running
Bit 0	**FMI:** *FMI clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the Flash memory interface clock. 0: FMI clock stopped 1: FMI clock running

Peripheral clock gating register 1 (SCU_PCGR1)

Address offset: 18h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved							RTC	GPIO 9	GPIO 8	GPIO 7	GPIO 6	GPIO 5	GPIO 4	GPIO 3	GPIO 2
							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
GPIO 1	GPIO 0	WIU	Res.	ADC	CAN	SSP 1	SSP 0	I2C 1	I2C 0	UART2	UART1	UART0	MC	TIM23	TIM01
rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Bits 31:25	Reserved, always read as 0
Bit 24	**RTC:** *RTC clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the RTC clock. 0: RTC clock stopped 1: RTC clock running
Bits 23:14	**GPIO[9:0]:** *GPIO Port clock gating* These bits are set and cleared by software. They allow you to reduce power consumption by turning off the clock to the corresponding GPIO port. 0: GPIO Port clock stopped 1: GPIO Port clock running
Bit 13	**WIU:** *WIU peripheral clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the WIU peripheral. 0: WIU peripheral clock stopped 1: WIU peripheral clock running
Bit 12	Reserved, must be kept at reset value 0
Bit 11	**ADC:** *ADC clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the ADC clock. 0: ADC clock stopped 1: ADC clock running
Bit 10	**CAN:** *CAN peripheral clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the CAN peripheral. 0: CAN peripheral clock stopped 1: CAN peripheral clock running
Bit 9:8	**SSP[1:0]:** *SSP peripheral clock gating* These bits are set and cleared by software. They allow you to reduce power consumption by turning off the corresponding SSP peripheral. 0: SSP peripheral clock stopped 1: SSP peripheral clock running

Bit 7:6	**I2C[1:0]:** *I2C peripheral clock gating* These bits are set and cleared by software. They allow you to reduce power consumption by turning off the corresponding I2C peripheral. 0: I2C peripheral clock stopped 1: I2C peripheral clock running
Bit 5:3	**UART[2:0]:** *UART peripheral clock gating* These bits are set and cleared by software. They allow you to reduce power consumption by turning off the corresponding UART peripheral. 0: UART peripheral clock stopped 1: UART peripheral clock running
Bit 2	**MC:** *Motor Control peripheral clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the MC clock. 0: MC clock stopped 1: MC clock running
Bit 1	**TIM23:** *Timers 2 and 3 clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the Timer 2 and 3 peripherals. 0: TIM23CLK stopped 1: TIM23CLK running
Bit 0	**TIM01:** *Timers 0 and 1 clock gating* This bit is set and cleared by software. It allows you to reduce power consumption by turning off the Timer 0 and 1 peripherals. 0: TIM01CLK stopped 1: TIM01CLK running

Peripheral reset register 0 (SCU_PRR0)

Address offset: 1Ch

Reset value: 0000 1053h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			RST_ PFQBC_ AHB	RST_ MAC	Res.	RST_ USB	RST_ DMA	Res.	RST_ EMI	RST_ VIC	RST_ SRAM_ ARBITER	Res.		RST_ PFQBC	RST_ FMI
			rw	rw		rw	rw		rw	rw	rw			rw	rw

Bits 31:13	Reserved, always read as 0
Bit 12	**RST_PFQBC_AHB:** *PFQBC AHB reset* This bit is set and cleared by software. It allows you to force a reset of the Prefetch Queue/Branch Cache AHB interface. 0: Module held in reset 1: Module not held in reset (default)
Bit 11	**RST_MAC:** *Ethernet peripheral reset* This bit is set and cleared by software. It allows you to force a reset of the Ethernet *peripheral*. 0: Module held in reset (default) 1: Module not held in reset
Bit 10	Reserved, always read as 0
Bit 9	**RST_USB:** *USB peripheral reset* This bit is set and cleared by software. It allows you to force a reset of the USB peripheral. 0: Module held in reset (default) 1: Module not held in reset
Bit 8	**RST_DMA:** *DMA peripheral reset* This bit is set and cleared by software. It allows you to force a reset of the DMA peripheral. 0: Module held in reset (default) 1: Module not held in reset
Bit 7	Reserved, always read as 0
Bit 6	**RST_EMI:** *EMI peripheral reset* This bit is set and cleared by software. It allows you to force a reset of the EMI peripheral. 0: Module held in reset 1: Module not held in reset (default)
Bit 5	**RST_VIC:** *VIC peripheral reset* It allows you to force a reset of the Vectored Interrupt Controller. 0: Module held in reset (default) 1: Module not held in reset

Bit 4	**RST_SRAM_ARBITER:** *SRAM arbiter reset* This bit is set and cleared by software. It allows you to force a reset of the SRAM arbiter. 0: Module held in reset 1: Module not held in reset (default)
Bits 3:2	Reserved, always read as 0
Bit 1	**RST_PQFBC:** *PQFBC reset* This bit is set and cleared by software. It allows you to force a reset of the Prefetch Queue/Branch Cache. 0: Module held in reset 1: Module not held in reset (default)
Bit 0	**RST_FMI:** *FMI reset* This bit is set and cleared by software. It allows you to force a reset of the Flash memory interface. 0: Module held in reset 1: Module not held in reset (default)

Peripheral reset register 1 (SCU_PRR1)

Address offset: 20h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved							RST_RTC	RST_GPIO9	RST_GPIO8	RST_GPIO7	RST_GPIO6	RST_GPIO5	RST_GPIO4	RST_GPIO3	RST_GPIO2
							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
RST_GPIO1	RST_GPIO0	RST_WIU	Reserved	RST_ADC	RST_CAN	RST_SSP1	RST_SSP0	RST_I2C1	RST_I2C0	RST_UART2	RST_UART1	RST_UART0	RST_MC	RST_TIM23	RST_TIM01
rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:25	Reserved, always read as 0
Bit 24	**RST_RTC:** *RTC reset* This bit is set and cleared by software. It allows you to force a reset of the RTC peripheral. 0: Module held in reset (default) 1: Module not held in reset
Bits 23:14	**RST_GPIO[9:0]:** *GPIO Port reset* These bits are set and cleared by software. They allow you to force a reset of the corresponding GPIO port. 0: Module held in reset (default) 1: Module not held in reset
Bit 13	**RST_WIU:** *WIU peripheral reset* This bit is set and cleared by software. It allows you to force a reset of the WIU peripheral. 0: Module held in reset (default) 1: Module not held in reset
Bit 12	Reserved, must be kept at reset value 0
Bit 11	**RST_ADC:** *ADC reset* This bit is set and cleared by software. It allows you to force a reset of the ADC clock. 0: Module held in reset (default) 1: Module not held in reset
Bit 10	**RST_CAN:** *CAN peripheral reset* This bit is set and cleared by software. It allows you to force a reset of the CAN peripheral. 0: Module held in reset (default) 1: Module not held in reset

Bit 9:8	**RST_SSP[1:0]:** *SSP peripheral reset* These bits are set and cleared by software. They allow you to force a reset of the corresponding SSP peripheral. 0: Module held in reset (default) 1: Module not held in reset
Bit 7:6	**RST_I2C[1:0]:** *I2C peripheral reset* These bits are set and cleared by software. They allow you to force a reset of the corresponding I2C peripheral. 0: Module held in reset (default) 1: Module not held in reset
Bit 5:3	**RST_UART[2:0]:** *UART peripheral reset* These bits are set and cleared by software. They allow you to force a reset of the corresponding UART peripheral. 0: Module held in reset (default) 1: Module not held in reset
Bit 2	**RST_MC:** *Motor Control peripheral reset* This bit is set and cleared by software. It allows you to force a reset of the MC clock. 0: Module held in reset (default) 1: Module not held in reset
Bit 1	**RST_TIM23:** *Timers 2 and 3 reset* This bit is set and cleared by software. It allows you to force a reset of the Timer 2 and 3 peripherals. 0: Module held in reset (default) 1: Module not held in reset
Bit 0	**RST_TIM01:** *Timers 0 and 1 reset* This bit is set and cleared by software. It allows you to force a reset of the Timer 0 and 1 peripherals. 0: Module held in reset (default) 1: Module not held in reset

Idle mode gating mask register 0 (SCU_MGR0)

Address offset: 24h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				MSK_MAC	MSK_USB 48M	MSK_USB	MSK_DMA	MSK_EXT_MEM_CLK	MSK_EMI	MSK_VIC	MSK_SRAM_ARBITER	MSK_SRAM	Reserved	MSK_PFQBC	Reserved
				rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	

Bits 31:12	Reserved, always read as 0
Bit 11	**MSK_MAC:** *Ethernet peripheral gating mask* This bit is set and cleared by software. It selects if the Ethernet peripheral is turned off when the MCU enters Idle mode. 0: Ethernet peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: Ethernet peripheral clock running in Idle mode if the corresponding PCGR bit is 1.
Bit 10	**MSK_USB48M:** *USB 48 MHz clock gating mask* This bit is set and cleared by software. It selects if the 48 MHz USB clock is turned off when the MCU enters Idle mode. 0: 48 MHz USB clock stopped in Idle mode (overrides PCGR bit setting) 1: 48 MHz USB clock running in Idle mode if the corresponding PCGR bit is 1.
Bit 9	**MSK_USB:** *USB peripheral clock gating mask* This bit is set and cleared by software. It selects if the USB peripheral is turned off when the MCU enters Idle mode. 0: USB peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: USB peripheral clock running in Idle mode if the corresponding PCGR bit is 1.
Bit 8	**MSK_DMA:** *DMA peripheral clock gating mask* This bit is set and cleared by software. It selects if the DMA peripheral is turned off when the MCU enters Idle mode. 0: DMA peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: DMA peripheral clock running in Idle mode if the corresponding PCGR bit is 1.
Bit 7	**MSK_EXT_MEM_CLK:** *External memory clock gating mask* This bit is set and cleared by software. It selects if the external memory clock is turned off when the MCU enters Idle mode. 0: External memory clock stopped in Idle mode (overrides PCGR bit setting) 1: External memory clock running in Idle mode if the corresponding PCGR bit is 1.

Bit 6	**MSK_EMI:** *EMI peripheral clock gating mask* This bit is set and cleared by software. It selects if the EMI peripheral is turned off when the MCU enters Idle mode. 0: EMI peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: EMI peripheral clock running in Idle mode if the corresponding PCGR bit is 1.
Bit 5	**MSK_VIC:** *VIC peripheral clock gating mask* This bit is set and cleared by software. It selects if the Vectored Interrupt Controller is turned off when the MCU enters Idle mode. 0: VIC peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: VIC peripheral clock running in Idle mode if the corresponding PCGR bit is 1.
Bit 4	**MSK_SRAM_ARBITER:** *SRAM arbiter clock gating mask* This bit is set and cleared by software. It selects if the SRAM arbiter clock is turned off when the MCU enters Idle mode. 0: SRAM arbiter clock stopped in Idle mode (overrides PCGR bit setting) 1: SRAM arbiter clock running in Idle mode if the corresponding PCGR bit is 1.
Bit 3	**MSK_SRAM:** *SRAM clock gating mask* This bit is set and cleared by software. It selects if the SRAM clock is turned off when the MCU enters Idle mode. 0: SRAM clock stopped in Idle mode (overrides PCGR bit setting) 1: SRAM clock running in Idle mode if the corresponding PCGR bit is 1.
Bit 2	Reserved, always read as 0
Bit 1	**MSK_ PQFBC:** *PQFBC clock gating mask* This bit is set and cleared by software. It selects if the Prefetch Queue/Branch Cache clock is turned off when the MCU enters Idle mode. 0: PQFBC clock stopped in Idle mode (overrides PCGR bit setting) 1: PQFBC clock running in Idle mode if the corresponding PCGR bit is 1.
Bit 0	Reserved, always read as 0

Idle mode gating mask register 1 (SCU_MGR1)

Address offset: 28h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved							MSK_RTC	MSK_GPIO9	MSK_GPIO8	MSK_GPIO7	MSK_GPIO6	MSK_GPIO5	MSK_GPIO4	MSK_GPIO3	MSK_GPIO2
							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
MSK_GPIO1	MSK_GPIO0	MSK_WIU	MSK_WDG	MSK_ADC	MSK_CAN	MSK_SSP1	MSK_SSP0	MSK_I2C1	MSK_I2C0	MSK_UART2	MSK_UART1	MSK_UART0	MSK_MC	MSK_TIM23	MSK_TIM01
				rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Bits 31:25	Reserved, always read as 0
Bit 24	**MSK_RTC:** *RTC clock gating mask* This bit is set and cleared by software. It selects if the RTC peripheral is turned off when the MCU enters Idle mode. 0: RTC peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: RTC peripheral clock running in Idle mode if the corresponding PCGR bit is 1
Bits 23:14	**MSK_GPIO[9:0]:** *GPIO Port clock gating mask* These bits are set and cleared by software. They allow you to force a reset of the corresponding GPIO port is turned off when the MCU enters Idle mode. 0: GPIO peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: GPIO peripheral clock running in Idle mode if the corresponding PCGR bit is 1
Bit 13	**MSK_WIU:** *WIU peripheral clock gating mask* This bit is set and cleared by software. It selects if the WIU peripheral is turned off when the MCU enters Idle mode. 0: WIU peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: WIU peripheral clock running in Idle mode if the corresponding PCGR bit is 1
Bit 12	**MSK_WDG:** *WDG peripheral clock gating mask* This bit is set and cleared by software. It selects if the WDG peripheral is turned off when the MCU enters Idle mode. 0: WDG peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: WDG peripheral clock running in Idle mode if the corresponding PCGR bit is 1
Bit 11	**MSK_ADC:** *ADC clock gating mask* This bit is set and cleared by software. It selects if the ADC clock is turned off when the MCU enters Idle mode. 0: ADC peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: ADC peripheral clock running in Idle mode if the corresponding PCGR bit is 1

Bit 10	**MSK_CAN:** *CAN peripheral clock gating mask* This bit is set and cleared by software. It selects if the CAN peripheral is turned off when the MCU enters Idle mode. 0: CAN peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: CAN peripheral clock running in Idle mode if the corresponding PCGR bit is 1
Bit 9:8	**MSK_SSP[1:0]:** *SSP peripheral clock gating mask* These bits are set and cleared by software. They select if the corresponding SSP peripheral is turned off when the MCU enters Idle mode. 0: SSP peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: SSP peripheral clock running in Idle mode if the corresponding PCGR bit is 1
Bit 7:6	**MSK_I2C[1:0]:** *I2C peripheral clock gating mask* These bits are set and cleared by software. They select if the corresponding I2C peripheral is turned off when the MCU enters Idle mode. 0: I2C peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1:I2C peripheral clock running in Idle mode if the corresponding PCGR bit is 1
Bit 5:3	**MSK_UART[2:0]:** *UART peripheral clock gating mask* These bits are set and cleared by software. They select if the corresponding UART peripheral is turned off when the MCU enters Idle mode. 0: UART peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: UART peripheral clock running in Idle mode if the corresponding PCGR bit is 1
Bit 2	**MSK_MC:** *Motor Control peripheral clock gating mask* This bit is set and cleared by software. It selects if the MC clock is turned off when the MCU enters Idle mode. 0: MC peripheral clock stopped in Idle mode (overrides PCGR bit setting) 1: MC peripheral clock running in Idle mode if the corresponding PCGR bit is 1
Bit 1	**MSK_TIM23:** *Timers 2 and 3 clock gating mask* This bit is set and cleared by software. It selects if the Timer 2 and 3 peripherals are turned off when the MCU enters Idle mode. 0: TIM23CLK stopped in Idle mode (overrides PCGR bit setting) 1: TIM23CLK running in Idle mode if the corresponding PCGR bit is 1
Bit 0	**MSK_TIM01:** *Timers 0 and 1 clock gating mask* This bit is set and cleared by software. It selects if the Timer 0 and 1 peripherals are turned off when the MCU enters Idle mode. 0: TIM01CLK stopped in Idle mode (overrides PCGR bit setting) 1: TIM01CLK running in Idle mode if the corresponding PCGR bit is 1

Peripheral emulation clock gating register 0 (SCU_PECGR0)

Address offset: 2Ch

Reset value: 0000 0FFBh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				MAC	USB 48M	USB	DMA	EXT_ MEM_ CLK	EMI	VIC	SRAM_ ARBITER	SRAM	Res.	PFQBC	FMI
				rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Bits 31:12	Reserved, always read as 0
Bit 11	**MAC:** *Ethernet peripheral clock gating* This bit is set and cleared by software. It allows you to turn off the Ethernet peripheral clock when the core enters emulation mode. 0: Ethernet peripheral clock stopped in emulation mode 1: Ethernet peripheral clock running in emulation mode
Bit 10	**USB48M:** *USB 48 MHz clock gating* This bit is set and cleared by software. It allows you to turn off the 48 MHz USB clock when the core enters emulation mode. 0: 48 MHz USB clock stopped in emulation mode 1: 48 MHz USB clock running in emulation mode
Bit 9	**USB:** *USB peripheral clock gating* This bit is set and cleared by software. It allows you to turn off the USB peripheral when the core enters emulation mode. 0: USB peripheral clock stopped in emulation mode 1: USB peripheral clock running in emulation mode
Bit 8	**DMA:** *DMA peripheral clock gating* This bit is set and cleared by software. It allows you to turn off the DMA peripheral when the core enters emulation mode. 0: DMA peripheral clock stopped in emulation mode 1: DMA peripheral clock running in emulation mode
Bit 7	**EXT_MEM_CLK:** *External memory clock gating* This bit is set and cleared by software. It allows you to turn off the external memory clock when the core enters emulation mode. 0: External memory clock stopped in emulation mode 1: External memory clock running in emulation mode
Bit 6	**EMI:** *EMI peripheral clock gating* This bit is set and cleared by software. It allows you to turn off the EMI peripheral when the core enters emulation mode. 0: EMI peripheral clock stopped in emulation mode 1: EMI peripheral clock running in emulation mode
Bit 5	**VIC:** *VIC peripheral clock gating* This bit is set and cleared by software. It allows you to turn off the Vectored Interrupt Controller when the core enters emulation mode. 0: VIC peripheral clock stopped in emulation mode 1: VIC peripheral clock running in emulation mode

Bit 4	**SRAM_ARBITER:** *SRAM arbiter clock gating* This bit is set and cleared by software. It allows you to turn off the SRAM arbiter clock when the core enters emulation mode. 0: SRAM arbiter clock stopped in emulation mode 1: SRAM arbiter clock running in emulation mode
Bit 3	**SRAM:** *SRAM clock gating* This bit is set and cleared by software. It allows you to turn off the SRAM clock when the core enters emulation mode. 0: SRAM clock stopped in emulation mode 1: SRAM clock running in emulation mode
Bit 2	Reserved, always read as 0
Bit 1	**PQFBC:** *PQFBC clock gating* This bit is set and cleared by software. It allows you to turn off the Prefetch Queue/Branch Cache clock when the core enters emulation mode. 0: PQFBC clock stopped in emulation mode 1: PQFBC clock running in emulation mode
Bit 0	**FMI:** *FMI clock gating* This bit is set and cleared by software. It allows you to turn off the Flash memory interface clock when the core enters emulation mode. 0: FMI clock stopped in emulation mode 1: FMI clock running in emulation mode

Peripheral emulation clock gating register 1 (SCU_PECGR1)

Address offset: 30h

Reset value: 01FF FFFFh

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								GPIO 9	GPIO 8	GPIO 7	GPIO 6	GPIO 5	GPIO 4	GPIO 3	GPIO 2
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPIO 1	GPIO 0	WIU	WDG	ADC	CAN	SSP 1	SSP 0	I2C 1	I2C 0	UART2	UART1	UART0	MC	TIM23	TIM01
				rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Bits 31:24	Reserved, always read as 0
Bits 23:14	**GPIO[9:0]:** *GPIO Port clock gating* These bits are set and cleared by software. They allow you to turn off the clock to the corresponding GPIO port when the core enters emulation mode. 0: GPIO Port clock stopped in emulation mode 1: GPIO Port clock running in emulation mode
Bit 13	**WIU:** *WIU peripheral clock gating* This bit is set and cleared by software. It allows you to turn off the WIU peripheral when the core enters emulation mode. 0: WIU peripheral clock stopped in emulation mode 1: WIU peripheral clock running in emulation mode
Bit 12	**WDG:** *WDG Peripheral clock gating* This bit is set and cleared by software. It allows you to turn off the WDG peripheral when the core enters emulation mode. 0: WDG peripheral clock stopped in emulation mode 1: WDG peripheral clock running in emulation mode
Bit 11	**ADC:** *ADC clock gating* This bit is set and cleared by software. It allows you to turn off the ADC clock when the core enters emulation mode. 0: ADC clock stopped in emulation mode 1: ADC clock running in emulation mode
Bit 10	**CAN:** *CAN peripheral clock gating* This bit is set and cleared by software. It allows you to turn off the CAN peripheral when the core enters emulation mode. 0: CAN peripheral clock stopped in emulation mode 1: CAN peripheral clock running in emulation mode
Bit 9:8	**SSP[1:0]:** *SSP peripheral clock gating* These bits are set and cleared by software. They allow you to turn off the corresponding SSP peripheral when the core enters emulation mode. 0: SSP peripheral clock stopped in emulation mode 1: SSP peripheral clock running in emulation mode

Bit 7:6	**I2C[1:0]:** *I2C peripheral clock gating* These bits are set and cleared by software. They allow you to turn off the corresponding I2C peripheral when the core enters emulation mode. 0: I2C peripheral clock stopped in emulation mode 1: I2C peripheral clock running in emulation mode
Bit 5:3	**UART[2:0]:** *UART peripheral clock gating* These bits are set and cleared by software. They allow you to turn off the corresponding UART peripheral when the core enters emulation mode. 0: UART peripheral clock stopped in emulation mode 1: UART peripheral clock running in emulation mode
Bit 2	**MC:** *Motor Control peripheral clock gating* This bit is set and cleared by software. It allows you to turn off the MC clock when the core enters emulation mode. 0: MC clock stopped in emulation mode 1: MC clock running in emulation mode
Bit 1	**TIM23:** *Timers 2 and 3 clock gating* This bit is set and cleared by software. It allows you to turn off the Timer 2 and 3 peripherals when the core enters emulation mode. 0: TIM23CLK clock stopped in emulation mode 1: TIM23CLK clock running in emulation mode
Bit 0	**TIM01:** *Timers 0 and 1 clock gating* This bit is set and cleared by software. It allows you to turn off the Timer 0 and 1 peripherals when the core enters emulation mode. 0: TIM01CLK stopped in emulation mode 1: TIM01CLK clock running in emulation mode

System configuration register 0 (SCU_SCR0)

Address offset: 34h

Reset value: 0000 0187h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	P30_SELEDBG	EXT_ETMT_EDBGR	UART_IRDA[2:0]			Res.	EMI_ALE_LNGT	EMI_ALE_POLR	EMI_MUX	SRAM_LK_EN	SRAM_SIZE[1:0]		WSR_AHB	WSR_DTCM	EN_PFQBC
	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:15	Reserved, always read as 0
Bit 14	**P30_SELEDBG:** *GPIO Port 3 External Debug Configuration* This bit is set and cleared by software to configure GPIO port 3.0 as input for ETM trigger or External Debug Request. 0: Port 3.0 not configured for External Debug Request / ETM trigger 1: Port 3.0 configured for External Debug Request / ETM trigger ***Note*:** Port 3.0 can be used for other functions controlled by the SCU_GPIOIN3 register bit 0. Do not set the P30_SELEDBG bit and GPIOIN3(0) bit to 1 at the same time. This will cause unpredictable behavior.
Bit 13	**EXT_ETMT_EDBGR:** *ETM Trigger/External Debug Selection* This bit is set and cleared by software to select ETM trigger or External Debug Request (refer to description of P30_SELEDBG). 0: External Debug request 1: External ETM trigger
Bits 12:10	**UART_IRDA[2:0]:** *UARTx IrDA mode selection* These bits are set and cleared by software to select the mode of the corresponding UART peripheral. 0: UART mode 1: IrDA mode
Bit 9	Reserved, must be kept at 0
Bit 8	**EMI_ALE_LNGT:** *EMI Active Level Length* This bit is set and cleared by software to define the EMI ALE length (duration of the active level). 0: One clock cycle (half clock cycle in synchronous mode) 1: Two clock cycles (default, one and a half clock cycles in synchronous mode).
Bit 7	**EMI_ALE_POLR:** *EMI Active Level Polarity* This bit is set and cleared by software to define the polarity of the ALE used for EMI in Mux mode. 0: Active low 1: Active high (default)

Bit 6	**EMI_MUX:** *EMI Mux/Demux* This bit is set and cleared by software to select the EMI Mux or Demux Mode. 0: Multiplexed mode 1: Demultiplexed mode (default)
Bit 5	**SRAM_LK_EN:** *SRAM Arbiter lock transfer enable* This bit is set and cleared by software to enable/disable lock transfer 0: AHB Lock transfer disabled (default) 1: AHB Lock transfer enabled
Bits 4:3	**SRAM_SIZE[1:0]**: *SRAM size* These bits are set and cleared by software to define the internal SRAM size. 00: 32 Kb (default) 01: 64 Kb 10: 96 Kb 11: reserved ***Note*:** Each time the size of SRAM is changed, the new size value is active one AHB clock cycle later. In order to perform a safe size change, it is better to wait one AHB clock cycle before accessing the memory.
Bit 2	**WSR_AHB:** *AHB Wait state enable* This bit is set and cleared by software to enable/disable the insertion of a wait state during an SRAM read access performed on the AHB bus 0: No wait state 1: 1 wait state (default)
Bit 1	**WSR_DTCM:** *DTCM Wait state enable* This bit is set and cleared by software to enable/disable the insertion of a wait state during an SRAM read access performed by the DTCM 0: No wait state 1: 1 wait state (default)
Bit 0	**EN_PFQBC:** *PFQBC Unit enable* This bit is set and cleared by software to enable/disable the PFQBC 0: PFQBC is disabled. The Flash memory access is performed with a by-pass. 1: PFQBC enabled (default)

GPIO output register (SCU_GPIOOUT*n*)

Address offset: 44h + *n**4 (*n* = 0 to 7)

Reset value: 0000 0000 0000 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P*n*.7 OUT 1	P*n*.7 OUT 0	P*n*.6 OUT 1	P*n*.6 OUT 0	P*n*.5 OUT 1	P*n*.5 OUT 0	P*n*.4 OUT 1	P*n*.4 OUT 0	P*n*.3 OUT 1	P*n*.3 OUT 0	P*n*.2 OUT 1	P*n*.2 OUT 0	P*n*.1 OUT 1	P*n*.1 OUT 0	P*n*.0 OUT 1	P*n*.0 OUT 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 15:0	**P*n*.[7:0]OUT[1:0]:** *GPIO Port Output Control bits* 00: Input mode 01: Alternate Output 1 (general purpose output) 10: Alternate Output 2 11: Alternate Output 3 See also *Section 2.6.7: System control unit GPIO registers on page 85*

GPIO input register (SCU_GPIOIN*n*)

Address offset: 64h + *n**4 (*n* = 0 to 7)

Reset value: 0000 0000h

7	6	5	4	3	2	1	0
P*n*.7IN	P*n*.6IN	P*n*.5IN	P*n*.4IN	P*n*.3IN	P*n*.2IN	P*n*.1IN	P*n*.0IN
rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, always read as 0
Bits 15:0	**P*n*.[7:0]IN:** *GPIO Port Input Control bits* 0: On-chip peripheral not connected to the input 1: On-chip peripheral connected to the input (Alternate input 1) ***Note*:** If a peripheral has the same input on multiple pins, do not enable more than one pin. See also *Section 2.6.7: System control unit GPIO registers on page 85*

GPIO type register (SCU_GPIOTYPE*m*)

Address offset: 84h + *m**4 (*m* = 0 to 9)

Reset value: 0000 0000

7	6	5	4	3	2	1	0
TYPE7	TYPE6	TYPE5	TYPE4	TYPE3	TYPE2	TYPE1	TYPE0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0	**TYPE[7:0]:** *GPIO output type register* These bits are set and cleared by software to configure the corresponding GPIO pin output type. All bits are cleared by a reset, so the GPIO pins are push-pull by default. 0: Push-pull 1: Open collector See also *Section 2.6.7: System control unit GPIO registers on page 85*

GPIO external memory interface register (SCU_EMI)

Address offset: ACh

Reset value: 0000 0000h

7	6	5	4	3	2	1	0
Reserved					BYTE_EN	BCLK_EN	GPIOEMI
					rw	rw	rw

Bits 31:3	Reserved, always read as 0
Bit 2	**BYTE_EN** *Byte Select control* This bit controls if the EMI_WRL/EMI_WRH pins behave as the write signals or as byte select signals. Available in LFBGA package only. 0: The pins function as the EMI_WRHn and EMI_WRLn signals. (default) 1. The pins function as the EMI_UBn
Bit 1	BCLK_EN *BCLK enable* This bit controls if the EMI_BCLK pin is enabled to drive the BCLK clock out. Available in LFBGA package only. 0: BCLK clock out is enabled (default) if the BCLKEN bit in the EMI_CCR register is also set. 1: BCLK clock is disabled
Bit 0	**GPIOEMI:** *GPIO EMI selection* This bit is set and cleared by software to configure Port 8 and 9 to the EMI block. It is cleared by a reset. Therefore, the external memory bus is disabled by default. 0: Port 8 and 9 used for general purpose I/O 1: Port 8 and 9 connected to EMI block See also *Section 1.12: External memory interface (EMI) on page 45*

Wakeup selection register (SCU_WKUPSEL)

Address offset: B0h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				WKUPSEL7.[2:0]			WKUPSEL6.[2:0]			WKUPSEL5.[2:0]			WKUPSEL3.[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12	Reserved, always read as 0
Bits 11:9	**WKUP_SEL7[2:0]:** *Wakeup/External Interrupt Port 7 selection* These bits are set and cleared by software. They are used to connect one of eight external wakeup/interrupt lines on GPIO Port 7 (EXINT[31:24]) to the VIC1.13 interrupt channel. Refer to the VIC and WIU chapters for more information. 000: Select EXINT24 line as VIC1.13 interrupt source (reset value) 001: Select EXINT25 line as VIC1.13 interrupt source ... 111: Select EXINT31 line as VIC1.13 interrupt source
Bits 8:6	**WKUP_SEL6[2:0]:** *Wakeup/External Interrupt Port 6 selection* These bits are set and cleared by software. They are used to connect one of eight external wakeup/interrupt lines on GPIO Port 6 (EXINT[23:16]) to the VIC1.12 interrupt channel. Refer to the VIC and WIU chapters for more information. 000: Select EXINT16 line as VIC1.12 interrupt source (reset value) 001: Select EXINT17 line as VIC1.12 interrupt source ... 111: Select EXINT23 line as VIC1.12 interrupt source
Bits 5:3	**WKUP_SEL5[2:0]:** *Wakeup/External Interrupt Port 5 selection* These bits are set and cleared by software. They are used to connect one of eight external wakeup/interrupt lines on GPIO Port 5 (EXINT[15:8]) to the VIC1.11 interrupt channel. Refer to the VIC and WIU chapters for more information. 000: Select EXINT8 line as VIC1.11 interrupt source (reset value) 001: Select EXINT9 line as VIC1.11 interrupt source ... 111: Select EXINT15 line as VIC1.11 interrupt source
Bits 2:0	**WKUP_SEL3[2:0]:** *Wakeup/External Interrupt Port 3 selection* These bits are set and cleared by software. They are used to connect the RTC event, USB resume event or one of six external wakeup/interrupt lines on GPIO Port 3 (EXINT[7:2]) to the VIC1.10 interrupt channel. Refer to the VIC and WIU chapters for more information. 000: Select RTC event as VIC1.10 interrupt source (reset value) 001: Select USB resume event as VIC1.10 interrupt source 010: Select EXINT2 line as VIC1.10 interrupt source 011: Select EXINT3 line as VIC1.10 interrupt source ... 111: Select EXINT7 line as VIC1.10 interrupt source

GPIO analog mode register (SCU_GPIOANA)

Address offset: BCh

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							ACG	P4.7A	P4.6A	P4.5A	P4.4A	P4.3A	P4.2A	P4.1A	P4.0A
							rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:9	Reserved, always read as 0
Bit 8	**ACG:** *ADC automatic clock gated mode* This bit is set and cleared by software to configure the ADC in automatic clock gated mode. This mode should be used only if trigger mode is enable to Start ADC conversion. It provide faster conversion time and cycle accurate synchronous data available after each trigger. 0: ADC Automatic clock gated mode off 1: ADC Automatic clock gated mode on See also *Section 16 on page 463*
Bits 7:0	**P4[7:0]A:** *GPIO Port 4 Analog Control bits* These bits are set and cleared by software to configure the corresponding GPIO port 4 pin in analog mode. When you use analog mode, clear the corresponding bits in the SCU_GPIOIN*n* and SCU_GPIOUT*n* registers. 0: Analog mode off 1: Analog mode on See also *Section 16 on page 463*

2.6.10 SCU register map

The following table summarizes the SCU registers.

Table 11. SCU 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	SCU_CLKCNTR	Reserved													Clock Control Register																		
04	SCU_PLLCONF	Reserved													PLL configuration Register																		
08	SCU_SYSSTATUS	Reserved																															System Status
0C	SCU_PWRMNG	Reserved																															Power Mgt
10	SCU_ITCMSK	Reserved																															Interrupt Mask
14	SCU_PCGR0	Reserved																					Peripheral Clock Gating Reg. 0										
18	SCU_PCGR1	Reserved							Peripheral clock Gating Register 1																								
1C	SCU_PRR0	Reserved																							Peripheral Reset Reg. 0								
20	SCU_PRR1	Reserved							Peripheral Reset Register 1																								
24	SCU_MGR0	Reserved																					Mask Gating Reg. 0										
28	SCU_MGR1	Reserved							Mask Gating Register 1																								
2C	SCU_PECGR0	Reserved																					Peripheral Emulation Clock Gating Register 0										
30	SCU_PECGR1	Reserved							Peripheral Emulation Clock Gating Register 1																								
34	SCU_SCR0	Reserved																															System Configuration Reg. 0
38-43	Reserved																																
44	SCU_GPIOOUT0	Reserved																															GPIO Output Register 0
48	SCU_GPIOOUT1	Reserved																															GPIO Output Register 1
4C	SCU_GPIOOUT2	Reserved																															GPIO Output Register 2
50	SCU_GPIOOUT3	Reserved																															GPIO Output Register 3
54	SCU_GPIOOUT4	Reserved																															GPIO Output Register 4
58	SCU_GPIOOUT5	Reserved																															GPIO Output Register 5
5C	SCU_GPIOOUT6	Reserved																															GPIO Output Register 6
60	SCU_GPIOOUT7	Reserved																															GPIO Output Register 7
64	SCU_GPIOIN0	Reserved																															GPIO Input Reg. 0
68	SCU_GPIOIN1	Reserved																															GPIO Input Reg. 1
6C	SCU_GPIOIN2	Reserved																															GPIO Input Reg. 2
70	SCU_GPIOIN3	Reserved																															GPIO Input Reg. 3

Table 11. SCU 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
74	SCU_GPIOIN4	Reserved																								GPIO Input Reg. 4							
78	SCU_GPIOIN5	Reserved																								GPIO Input Reg. 5							
7C	SCU_GPIOIN6	Reserved																								GPIO Input Reg. 6							
80	SCU_GPIOIN7	Reserved																								GPIO Input Reg.; 7							
84	SCU_GPIOTYPE0	Reserved																								GPIO Type Reg. 0							
88	SCU_GPIOTYPE1	Reserved																								GPIO Type Reg. 1							
8C	SCU_GPIOTYPE2	Reserved																								GPIO Type Reg. 2							
90	SCU_GPIOTYPE3	Reserved																								GPIO Type Reg. 3							
94	SCU_GPIOTYPE4	Reserved																								GPIO Type Reg. 4							
98	SCU_GPIOTYPE5	Reserved																								GPIO Type Reg. 5							
9C	SCU_GPIOTYPE6	Reserved																								GPIO Type Reg. 6							
A0	SCU_GPIOTYPE7	Reserved																								GPIO Type Reg. 7							
A4	SCU_GPIOTYPE8	Reserved																								GPIO Type Reg. 8							
A8	SCU_GPIOTYPE9	Reserved																								GPIO Type Reg. 9							
AC	SCU_GPIOEMI	Reserved																															GPIOEMI
B0	SCU_WKUPSEL	Reserved																								Wakeup selection register							
B4-BB	Reserved	Reserved																															
BC	SCU_GPIOANA	Reserved																							ACG	GPIO Analog Mode							

Refer to *Table 5 on page 35* for the register base addresses.

3 General purpose I/O ports (GPIO)

3.1 Functional description

The I/O pins have the following characteristics:

- All GPIO pins are 5V tolerant.
- I/O port drivers may be configured as push-pull or as open collector.
- Some peripheral functions have bi-directional functionality such as I^2C data and clock lines. In these configurations, the I/O driver must be configured as open collector.
- Only Port 4 bits 0..7 have an ADC input.
- All ports when configured for Input mode are in high impedance.
- Alternate Input functions default to open (i.e. the driver output to the IP is disabled) and on P0 - P7 are controlled by SCU_GPIOIN control registers.
- Alternate Output functions on P0 - P7 are configured via SCU_GPIOOUT control registers that select from one of 3 output functions.
- The GPIO ports have no internal or programmable pull-up resistors.

Note: Refer to the STR91xFA datasheet for the device pin description and electrical characteristics.

3.2 I/O operation

Each GPIO port comprises eight programmable input/output lines. Data and control for these lines are provided by a data register and a data direction register. On reads, the data register contains the current status of the GPIO pins, whether they are configured as input or output. Writing to the data register only affects the pins that are configured as outputs.

3.2.1 GPIO_DATA register read/write masking

So that independent software drivers can set their GPIO bits in a single write operation, without affecting any other pins, the address bus is used as a mask on read/write operations. The data register effectively covers 256 locations in the address space. The eight address lines used as for masking are PADDR[9:2].

Write example

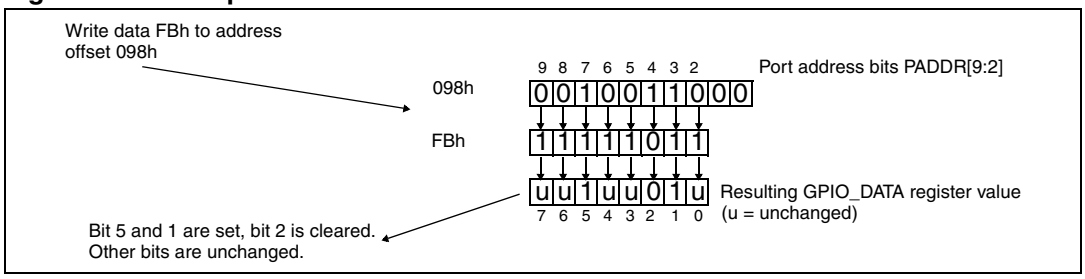
During a write, if the address bit associated with that data bit is HIGH, the value of the GPIO_DATA register is altered. If it is LOW, it is left unchanged. For example:

Writing to address GPIO_DATA base address + 098h = 000010011000b

PADDR[9:2] = 0000100110b. When a value of FBh is written to the address 098h then:

- Bits 5, and 1 of the GPIO pins are set to 1, and bit 2 is set to 0
- The other bits are not changed. *Figure 28.* shows the above effect of the address value of 0x098 operating on the data value of 0xFB.

Figure 28. Example Write to address 098h



So following this principle, to change all the port bits at the same time, write the data to address offset 001111111100b (3FCh), if you want to change just port bit 0, then write the data to address offset 000000000100b (004h) and so on.

Read example

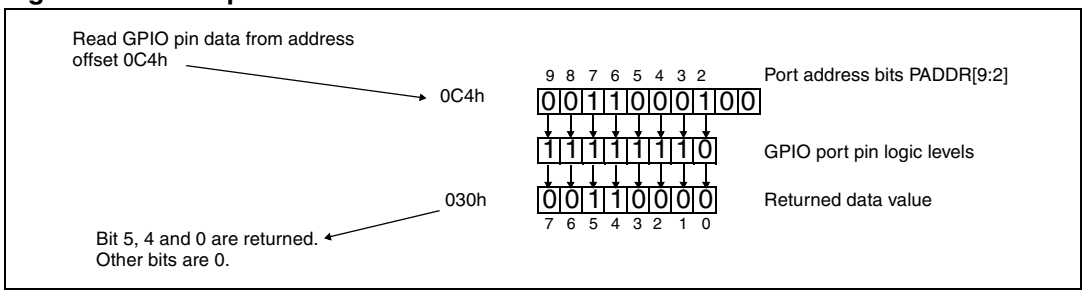
During a read if an address bit associated with data is HIGH the value is read, if it is LOW it is read as zero. For example:

Read from address GPIO_DATA base address + 0x0C4 = 000011000100b

PADDR[9:2] = 0000110001b. When reading from 0x0C4 then:

- Bits 5, 4, and 0 of the GPIO pins are returned
- the value of bits 7, 6, 3, 2, and 1 are returned as zero, regardless of their state.

Figure 29. Example Read from address 0C4h



3.2.2 Reset state

- All registers are cleared to zero
- Input and output pins are configured as inputs

3.3 System control unit GPIO registers

GPIO pins on P0 thru P7 have multiple input and output alternate functions. You select these using the System Control Unit (SCU) registers. SCU registers are also used to select open collector or Push-Pull operation and to configure Port 4 pins for use as analog inputs.

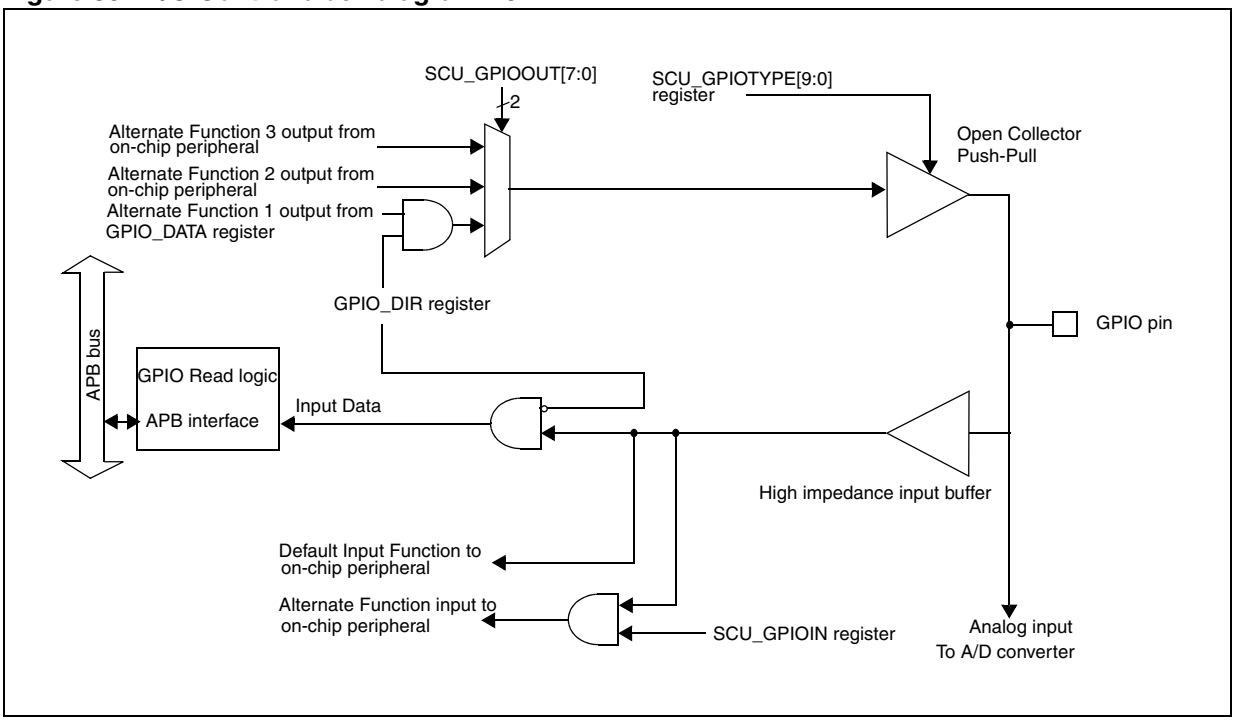
GPIO pins on P8 thru P9 are only multiplexed with EMI and have no SCU output or input control registers.

All ports have SCU_GPIOTYPE registers for selecting Open Collector or Push/Pull configuration.

Refer to following register desriptions for more details:

- *GPIO output register (SCU_GPIOOUTn) on page 110*
- *GPIO input register (SCU_GPIOINn) on page 110*
- *GPIO type register (SCU_GPIOTYPEm) on page 111*

Figure 30. I/O Control block diagram P0 - P7



3.4 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 (wo)	Software can only write to this bit. Reading the bit returns the reset value

3.4.1 GPIO data register (GPIO_DATA)

Address offset: 00h to 3FCh

Reset value: 0000 0000

7	6	5	4	3	2	1	0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0	**DATA[7:0]**: *GPIO data register* Values written in the GPIO_DATA register are transferred onto the GPIO pins if the respective pins have been configured as outputs through the GPIO_DIR register and Alternate Function 1 is configured in the SCU_GPIOOUT register. In order to write to GPIO_DATA, the corresponding bits in the mask, resulting from bits [9:2] of the address offset, must be HIGH. Otherwise the bit values remain unchanged by the write. Similarly, the values read from this register are determined for each bit, by the mask bit derived from bits [9:2] of the address offset used to access the data register. Refer to *Figure 28.* and *Figure 29.* for write and read examples. Bits that are 1 in the address mask cause the corresponding bits in GPIO_DATA to be read, and bits that are 0 in the address mask cause the corresponding bits in GPIO_DATA to be read as 0, regardless of their value. A read from GPIO_DATA returns the last bit value written if the respective pins are configured as output, or it returns the value on the corresponding input pin when these are configured as inputs. All bits are cleared by a reset.

3.4.2 GPIO data direction register (GPIO_DIR)

Address offset: 400h

Reset value: 0000 0000

7	6	5	4	3	2	1	0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0	**DIR[7:0]:** *GPIO data direction register* These bits are set and cleared by software to configure the corresponding GPIO pin to be an input or an output. All bits are cleared by a reset. Therefore, the GPIO pins are input by default. 0: Input 1: Output

3.4.3 GPIO mode control register (GPIO_SEL)

Address offset: 420h

Reset value: 0000 0000

7	6	5	4	3	2	1	0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0	**GPIOSEL[7:0]:** *GPIO mode control register* These bits are set an cleared by software. All bits are cleared by a reset. 0: GPIO mode (for general purpose I/O) 1: Reserved ***Note***: This bit must be '0' when GPIO is used.

3.4.4 GPIO register map

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

Table 12. GPIO register map

Address offset	Register name	7	6	5	4	3	2	1	0
000-03FC	GPIO_DATA	GPIO Data Register							
400	GPIO_DIR	GPIO Data Direction Register							
404-41C	Reserved								
420	GPIO_SEL	GPIO Mode Control Register							

Refer to the *Section 2.6.10 on page 114* for the SCU GPIO register description.

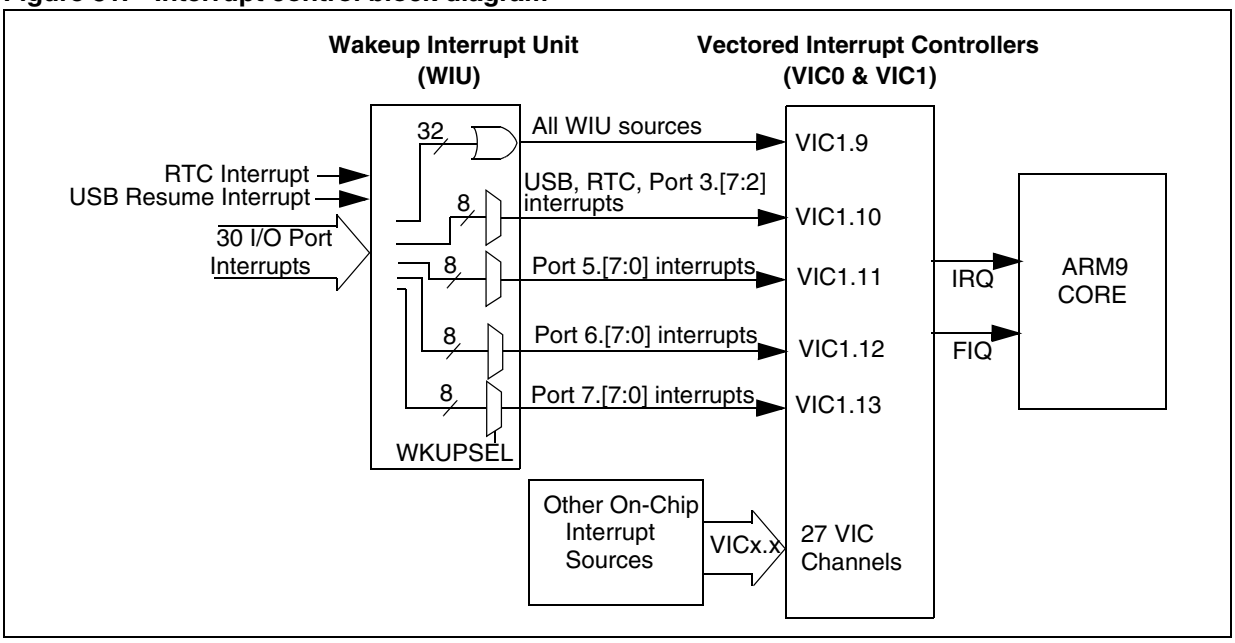
4 Interrupts (VIC and WIU)

4.1 Overview

The STR91xFA has a flexible Interrupt structure that is composed of two interrupt control blocks. The first interrupt control block is the Wakeup/Interrupt Unit (WIU). The wakeup unit monitors up to 30 external interrupt sources that are optionally shared with the GPIO inputs and the interrupt from the RTC and USB. Any of these interrupt sources may be used to wakeup the MCU and/or generate an interrupt to the VIC. The Wakeup unit has 5 interrupt outputs. One is the Wakeup interrupt which is the logical OR of all the 30 external (pin) interrupts. The other four are one of eight groups of interrupt sources (mapped to the VIC interrupt inputs as shown in *Figure 31*. Any of these external interrupt sources can be used to generate a IRQ or FIQ interrupt to the core. Any of the 30 GPIO inputs can be configured either as interrupt or wakeup inputs.

The second interrupt control block consists of two Vectored Interrupt Controllers, based on ARM primecell 190, connected in a daisy chain. The VICs (VIC0 and VIC1) are responsible for accepting up to 32 interrupt sources and assigning them to the IRQ or FIQ interrupt signals in the ARM core. The VICs support up to 32 vectored interrupt addresses. One of the 32 sources may be assigned as a FIQ source. In addition, the RTC and USB interrupts may also be used to wakeup the system and are also connected to the VIC so they may be assigned to vectored interrupts.

Figure 31. Interrupt control block diagram



4.2 Interrupt inputs to the CPU

The VICs are connected to the AHB bus for fast interrupt response. VIC0 generates two interrupt signals to the CPU: the FIQ and IRQ. The fast interrupt FIQ is a non-vectored interrupt and is dedicated to a user specified interrupt source. The CPU can execute the interrupt service routine directly without determining the interrupt source.

The IRQ is an ORed output of up to 32 vectored interrupts. The CPU fetches the current vector address in the VIC0_VAR register and jumps to the specified address. The 32 vectored interrupt sources come from the on-chip peripherals, refer to *Table 13 on page 124*. The VIC interrupt inputs are active high, level sensitive and are held high until the interrupt is cleared in the peripheral registers by the interrupt service routine.

4.3 Vectored interrupt controller (VIC)

The STR91xFA has two daisy-chained VICs supporting up to 32 vectored interrupt sources.

The VIC interrupt request logic is shown in *Figure 32.* Interrupt sources [15:0] are connected to VIC0 and Interrupt sources [31:16] are connected to VIC1. The interrupt requests (IRQ, FIQ) from VIC1 are daisy chained to VIC0. The vector address output from VIC1 is connected to VIC0 to be passed on to the CPU. VIC0 is responsible for generating the FIQ and IRQ to the CPU when any one of the 32 interrupt inputs is active.

When serving an IRQ, the vector address of the interrupt is returned when the CPU reads the VIC0 Current Vector Address Register (VIC0_VAR). If the interrupt comes from VIC0, VIC0 drives the vector address from one of the 16 Vector Address registers (VIC0_VA*i*R) to the CPU. If the interrupt comes from VIC1, VIC0 will pass the vector address that is originated from VIC1.

Reading the VIC0 Current Vector Address Register (VIC0_VAR) also updates the priority hardware that masks out the current and any lower priority interrupt requests. When the interrupt source is from VIC1, the interrupt subroutine has to read the VIC1_VAR register as well, to update the priority hardware. Writing to the VIC0_VAR and VIC1_VAR registers before exiting the interrupt subroutine indicates to the priority hardware that the current interrupt is served, allowing lower priority interrupt to go active.

However, a higher priority can preempt the execution of the current interrupt subroutine immediately without the need to write in the VIC0_VAR

4.4 FIQ handling

FIQ (Fast Interrupt Request) is a non-vectored interrupt, allowing the CPU to execute an interrupt service routine directly without having to determine/prioritize the interrupt source, minimizing interrupt latency. Typically only one interrupt source is assigned to FIQ. An FIQ interrupt has its own set of banked registers to minimize the time to make a context switch. Any of the 32 VIC input channels can be assigned to FIQ using the Interrupt Select register (VICx_INTSR).

4.5 IRQ handling

Any of the 32 VIC interrupt channels can be assigned as an IRQ source using the Interrupt Select registers (VICx_INTSR). The IRQ sent to the CPU is the logical OR of all these channels. The priority of each VIC channel is fixed by hardware. Interrupt sources can be assigned using the Vector Control registers (VICx_VC*i*R).

When an interrupt occurs on an interrupt channel, the VIC hardware will resolve the interrupt priority and generate an IRQ. The global interrupt service routine then reads the VIC0 Current Vector Address register (VIC0_VAR) and jumps to the interrupt service routine for the specific interrupt channel.

The STR91xFA has a feature to reduce ISR response time for IRQ interrupts. Typically, it requires two memory accesses to read the interrupt vector address from the VIC, but the STR91xFA reduces this to a single access by adding a 16th entry in the instruction branch cache, dedicated for interrupts. This 16th cache entry always holds the instruction that reads the interrupt vector address from the VIC, eliminating one of the memory accesses typically required in traditional ARM implementations.

See *Table 13* for the list of VIC interrupt channels.

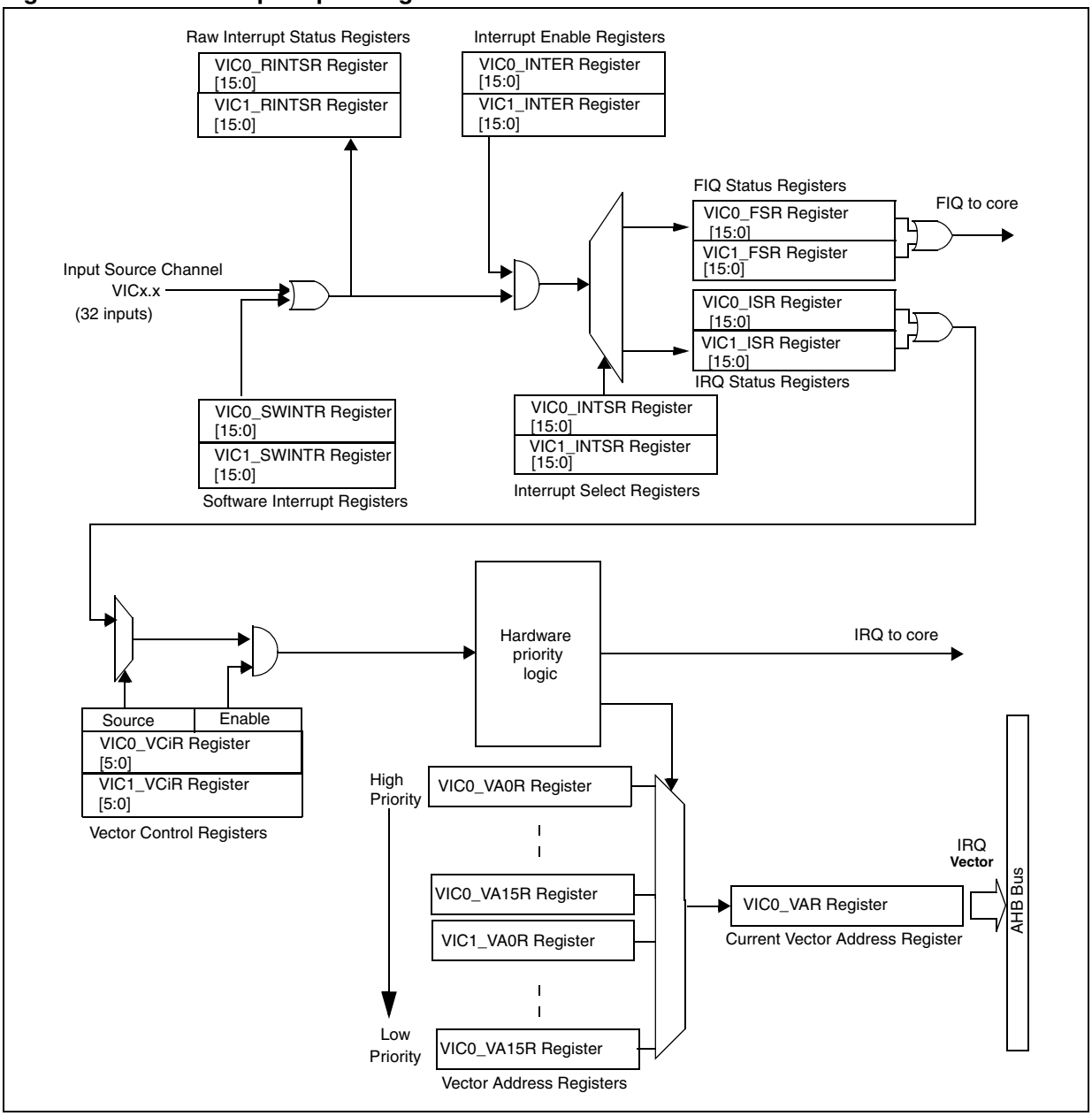
Table 13. VIC interrupt channels

VIC input channel	Logic block	Interrupt source
VIC0.0	Watchdog	Timeout in WDG mode, End of Count in Counter Mode
VIC0.1		Software interrupt
VIC0.2	CPU Core	Debug Receive Command
VIC0.3	CPU Core	Debug Transmit Command
VIC0.4	TIM Timer 0	Logic OR of ICI0_0, ICI0_1, OCI0_0, OCI0_1, Timer overflow
VIC0.5	TIM Timer 1	Logic OR of ICI1_0, ICI1_1, OCI1_0, OCI1_1, Timer overflow
VIC0.6	TIM Timer 2	Logic OR of ICI2_0, ICI2_1, OCI2_0, OCI2_1, Timer overflow
VIC0.7	TIM Timer 3	Logic OR of ICI3_0, ICI3_1, OCI3_0, OCI3_1, Timer overflow
VIC0.8	USB	Logic OR of high priority USB interrupts
VIC0.9	USB	Logic OR of low priority USB interrupts
VIC0.10	SCU	Logic OR of all interrupts from System Control Unit (SCU) except Early warning
VIC0.11	Ethernet MAC	Logic OR of Ethernet MAC interrupts via its own dedicated DMA channel.
VIC0.12	DMA	Logic OR of interrupts from each of the 8 individual DMA channels
VIC0.13	CAN	Logic OR of all CAN interface interrupt sources
VIC0.14	MC	Logic OR of 8 Induction Motor Control Unit interrupts
VIC0.15	ADC	End of A/D conversion or analog watchdog interrupt

Table 13. VIC interrupt channels (continued)

VIC input channel	Logic block	Interrupt source
VIC1.0	UART0	Logic OR of 5 interrupts from UART channel 0
VIC1.1	UART1	Logic OR of 5 interrupts from UART channel 1
VIC1.2	UART2	Logic OR of 5 interrupts from UART channel 2
VIC1.3	I2C0	Logic OR of transmit, receive, and error interrupts of I2C channel 0
VIC1.4	I2C1	Logic OR of transmit, receive, and error interrupts of I2C channel 1
VIC1.5	SSP0	Logic OR of all interrupts from SSP0
VIC1.6	SSP1	Logic OR of all interrupts from SSP1
VIC1.7	SCU	LVD early warning interrupt (Brownout)
VIC1.8	RTC	Logic OR of Alarm, Tamper, or Periodic Timer interrupts
VIC1.9	WIU (all)	Logic OR of all 32 inputs of Wakeup unit (30 pins, RTC, and USB Resume)
VIC1.10	WIU Group 0	One of 8 interrupt sources: RTC, USB Resume, pins P3.2 to P3.7 selected by SCU_WKUPSEL register
VIC1.11	WIU Group 1	One of 8 interrupt sources from pins P5.0 to P5.7 selected by SCU_WKUPSEL register
VIC1.12	WIU Group 2	One of 8 interrupt sources from pins P6.0 to P6.7 selected by SCU_WKUPSEL register
VIC1.13	WIU Group 3	One of 8 interrupt sources from pins P7.0 to P7.7 selected by SCU_WKUPSEL register
VIC1.14	USB	USB Bus Resume Wakeup (also input to wakeup unit)
VIC1.15	PFQ-BC	Special use of interrupts from Prefetch Queue and Branch Cache

Figure 32. VIC interrupt request logic



4.6 VIC register address mapping

The CPU reads or writes to the VIC registers through the AHB bus. For the CPU to read the Vector Address Register in VIC0 in one instruction (LDR PC), VIC0 is located in the upper 4 Kb of the memory (0XFFFFF000).

4.7 Interrupt priority

The FIQ interrupt has the highest priority and is followed by the 32 vectored interrupts. The interrupt priority is based on the position of the vectored interrupt, where VIC0 Vectored Interrupt 0 has the highest priority and VIC1 Vectored Interrupt 15 has the lowest priority. The priority is hardwired and can not be changed.

This means that interrupts mapped on VIC0 will always have higher priority than interrupts mapped on VIC1 (hardwired priority).

Each of the Vectored Interrupts has a Control Register which specifies the input source of the interrupt. Depending on your application requirement, inside the same VIC (VIC0 or VIC1), you can assign an interrupt source/input to a low or high priority Vectored Interrupt by writing to the Control Register. The Vectored Interrupt priority is fixed by hardware, but the interrupt input source to the Vectored Interrupt is software programmable (selectable).

4.8 Software interrupts

VIC0 and VIC1 each have a Software Interrupt Register (VICx_SWINTR). Setting a bit in this register will generate an interrupt to the CPU. The software interrupt can be assigned to any one of the 32 vectored interrupt inputs. Software interrupts are cleared by writing to the Software Interrupt Clear register (VICx_SWINTCR).

4.9 Enabling interrupts

Enabling an interrupt requires the following VIC register configuration:

- Enable interrupt input line by setting the enable bit in the Interrupt Enable register (VICx_INTER).
- Clear the corresponding bit in the Interrupt Select register (VICx_INTSR) to configure the input as vectored interrupt (IRQ) or set the bit to configure the input as Fast Interrupt (FIQ).
- Set the E bit in the Vector Control register (VICx_VC*i*R) to enable the vectored interrupt

4.10 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
Read/set (rs)	Software can read as well as set this bit. Writing ‘0’ has no effect on the bit value
Write only (wo)	Software can only write to this bit. Reading the bit returns the reset value

4.10.1 IRQ status register (VICx_ISR)

Address offset: 000h

Reset value: 0000 0000

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

Bits 15:0	**IRQStatus[15:0]**: *IRQ Status bits* These bits are set by hardware after masking by the VICx_INTER and VICx_INTSR registers. An active bit will remain high until software clears the interrupt in the registers of the peripheral which sourced the interrupt event. Each bit corresponds to an input channel. IRQStatus0 gives the status of channel VICx.0 and IRQStatus15 gives the status of channel VICx.15 (see *Table 13.*). 0: No IRQ interrupt generated by this input channel 1: IRQ interrupt generated by this input channel

4.10.2 FIQ status register (VICx_FSR)

Address offset: 004h

Reset value: 0000 0000 0000 0000

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

Bits 15:0	**FIQStatus[15:0]**: *FIQ Status bits* These bits are set by hardware after masking by the VICx_INTER and VICx_INTSR registers. An active bit will remain high until software clears the interrupt in the registers of the peripheral which sourced the interrupt event. Each bit corresponds to an input channel. FIQStatus0 gives the status of channel VICx.0 and FIQStatus15 gives the status of channel VICx.15 (see *Table 13.*). 0: No FIQ interrupt generated by this input channel 1: FIQ interrupt generated by this input channel

4.10.3 Raw interrupt status register (VICx_RINTSR)

Address offset: 008h

Reset value: undefined

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

Bits 15:0	**RawIntr[15:0]**: *Raw interrupt Status bits* These bits give the status of the interrupt sources (and software interrupts). An active bit will remain high until software clears the interrupt in the registers of the peripheral which sourced the interrupt event. Each bit corresponds to an input channel. RawIntr0 gives the status of channel VICx.0 and RawIntr15 gives the status of channel VICx.15 (see *Table 13.*). 0: Interrupt source inactive 1: Interrupt source active before masking

4.10.4 Interrupt select register (VICx_INTSR)

Address offset: 00Ch

Reset value: 0000 0000 0000 0000

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

Bits 15:0	**IntSelect[15:0]**: *Interrupt Select bits* These bits select whether the corresponding interrupt source generates an FIQ or an IRQ interrupt. 0: IRQ 1: FIQ

4.10.5 Interrupt enable register (VICx_INTER)

Address offset: 010h

Reset value: 0000 0000 0000 0000

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

Bits 15:0	**IntEnable[15:0]**: *Interrupt Enable bits* Software can set these bits to enable interrupts on the corresponding channel. On reset, all interrupt sources are masked. Writing 0 has no effect. 0: Interrupt disabled (masked) (read) 1: Interrupt enabled

4.10.6 Interrupt enable clear register (VICx_INTECR)

Address offset: 014h

Reset value: undefined

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

Bits 15:0	**IntEnClear[15:0]**: *Interrupt enable clear bits* Software can set these bits to disable interrupts on the corresponding channel. Writing 1 clears the corresponding bit in the VICx_INTER register. Writing 0 has no effect. 0: No effect 1: Interrupt disabled

4.10.7 Software interrupt register (VICx_SWINTR)

Address offset: 018h

Reset value: 0000 0000 0000 0000

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

Bits 15:0	**SoftInt[15:0]**: *Software interrupt bits* Software can set these bits to generate a source interrupt event on the corresponding channel before masking. Writing 1 sets the corresponding register bit. Writing 0 has no effect. 0: No effect 1: Interrupt source active

4.10.8 Software interrupt clear register (VICx_SWINTCR)

Address offset: 01Ch

Reset value: undefined

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

Bits 15:0	**SoftIntClear[15:0]**: *Software interrupt clear bits* Software can set these bits to clear an active software interrupt on the corresponding channel. Writing 1 clears the corresponding bit in the VICx_SWINTR register. Writing 0 has no effect. 0: No effect 1: Clear software interrupt

4.10.9 Protection enable register (VICx_PER)

Address offset: 020h

Reset value: 0000 0000 0000 0000

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

Bits 15:1	Reserved, forced by hardware to 0
Bit 0	**PROT**: *Protection bit* This bit is set and cleared by software to enable/disable protected register access. 0: VIC registers can be accessed in user mode and privileged mode 1: VIC registers can be accessed in privileged mode only ***Note*:** If the bus master cannot generate accurate protection information, leave this register in its reset state to allow user mode access.

4.10.10 Current vector address register (VICx_VAR)

Address offset: 030h

Reset value: 0000 0000 0000 0000

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

Bits 31:0	**CurrVectorAddr[31:0]**: *Current Vector Address* This register contains the address of the currently active IRQ interrupt service routine. Reading from the VIC0_VAR register provides the address of the interrupt subroutine (ISR) and indicates to the priority hardware that the interrupt is being serviced. Writing to VIC0_VAR or VIC1_VAR register indicates to the priority hardware that the interrupt has been serviced. The register should be used as follows: – When an IRQ interrupt is generated, read the VIC0_VAR register to fetch the address of the interrupt service routine (ISR). – At the end of the ISR, write to the VIC0_VAR register (or VIC1_VAR if the interrupt source is from VIC1) to update the priority hardware. ***Caution*:** Reading or writing to the register at other times can cause incorrect operation.

4.10.11 Default vector address register (VICx_DVAR)

Address offset: 034

Reset value: 0000 0000 0000 0000

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

Bits 31:0	**DefVectorAddr[31:0]**: *Default Vector Address* This register contains the default interrupt subroutine (ISR) address.

4.10.12 Vector address *i* registers (VICx_VA*i*R)

Address offset: see *Table 14*

Reset value: 0000 0000 0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VectAddr*i*[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
VectAddr*i*[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**VectorAddr*i*[31:0]**: *Vector Address i (i = 0 to 15)* These sixteen registers contain the addresses of the interrupt subroutines (ISR). VecAddr0 has the highest priority, VecAddr15 has the lowest priority.

4.10.13 Vector control *i* registers (VICx_VC*i*R)

Address offset: see *Table 14*

Reset value: 0000 0000 0000 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										E*i*	Res.	VectCntli[3:0]			
										rw		rw	rw	rw	rw

Bits 15:6	Reserved, forced by hardware to 0
Bit 5	**E*i***: *Vector interrupt i enable (i = 0 to 15)* This bit enables vectored interrupt i. It is cleared on reset. 0: Disabled 1: Enabled ***Note*:** Vectored interrupts are only generated if the interrupt is enabled. You enable the specific interrupt in the VICx_INTER register and set the interrupt is set to generate an IRQ interrupt in the VICx_INTSR register. This prevents multiple interrupts being generated from a single request if the controller is incorrectly programmed
Bit 4	Reserved, always read as 0
Bits 3:0	**VectorCntl*i*[3:0]**: *Vector Control i (i = 0 to 15)* These bits select the interrupt source for the vectored interrupt. You can select any of the 16 VIC interrupt sources

4.11 VIC register map

The following table is a summary of the VIC registers. VIC0 and VIC1 have the same set of registers, only the base addresses are different.

Table 14. VICx register map

Addr. offset hex.	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
000h	VICx_ISR	Reserved																IRQ Status Register															
004h	VICx_FSR	Reserved																FIQ Status Register															
008h	VICx_RINTSR	Reserved																Raw Status Register															
00Ch	VICx_INTSR	Reserved																Interrupt Select Register															
010h	VICx_INTER	Reserved																Interrupt Enable Register															
014h	VICx_ INTECR	Reserved																Interrupt Enable Clear Register															
018h	VICx_ SWINTR	Reserved																Software Interrupt Register															
01Ch	VICx_ SWINTCR	Reserved																Software Interrupt Clear Register															
020h	VICx_PER	Reserved																															PROT
030h	VICx_VAR	Current Vector Address																															
034h	VICx_DVAR	Default Vector Address																															
100h	VICx_VA0R	Vector Address 0																															
104h	VICx_VA1R	Vector Address 1																															
108h	VICx_VA2R	Vector Address 2																															
10Ch	VICx_VA3R	Vector Address 3																															
110h	VICx_VA4R	Vector Address 4																															
114h	VICx_VA5R	Vector Address 5																															
118h	VICx_VA6R	Vector Address 6																															
11Ch	VICx_VA7R	Vector Address 7																															
120h	VICx_VA8R	Vector Address 8																															
124h	VICx_VA9R	Vector Address 9																															
128h	VICx_VA10R	Vector Address 10																															
12Ch	VICx_VA11R	Vector Address 11																															
130h	VICx_VA12R	Vector Address 12																															
134h	VICx_VA13R	Vector Address 13																															
138h	VICx_VA14R	Vector Address 14																															
13Ch	VICx_VA15R	Vector Address 15																															
200h	VICx_VC0R	Reserved																										E0				VectCntrl0	
204h	VICx_VC1R	Reserved																										E1				VectCntrl1	
208h	VICx_VC2R	Reserved																										E2				VectCntrl2	
20Ch	VICx_VC3R	Reserved																										E3				VectCntrl3	
210h	VICx_VC4R	Reserved																										E4				VectCntrl4	
214h	VICx_VC5R	Reserved																										E5				VectCntrl5	
218h	VICx_VC6R	Reserved																										E6				VectCntrl6	
21Ch	VICx_VC7R	Reserved																										E7				VectCntrl7	

Table 14. VICx register map (continued)

Addr. offset hex.	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
220h	VICx_VC8R	Reserved																										E8		VectCntrl8			
224h	VICx_VC9R	Reserved																										E9		VectCntrl9			
228h	VICx_VC10R	Reserved																										E10		VectCntrl10			
22Ch	VICx_VC11R	Reserved																										E11		VectCntr11			
230h	VICx_VC12R	Reserved																										E12		VectCntr12			
234h	VICx_VC13R	Reserved																										E13		VectCntrl13			
238h	VICx_VC14R	Reserved																										E14		VectCntrl14			
23Ch	VICx_VC15R	Reserved																										E15		VectCntrl15			

Refer to *Table 5 on page 35* for the register base addresses.

4.12 Wakeup/Interrupt Unit (WIU)

The main function of the Wakeup Interrupt Unit (WIU) is to manage the external wakeup/interrupt pins (EXINT[31:2]). The WIU is connected to five of the 16 input channels of the VIC1 vectored interrupt controller (see *Table 31 on page 122*)

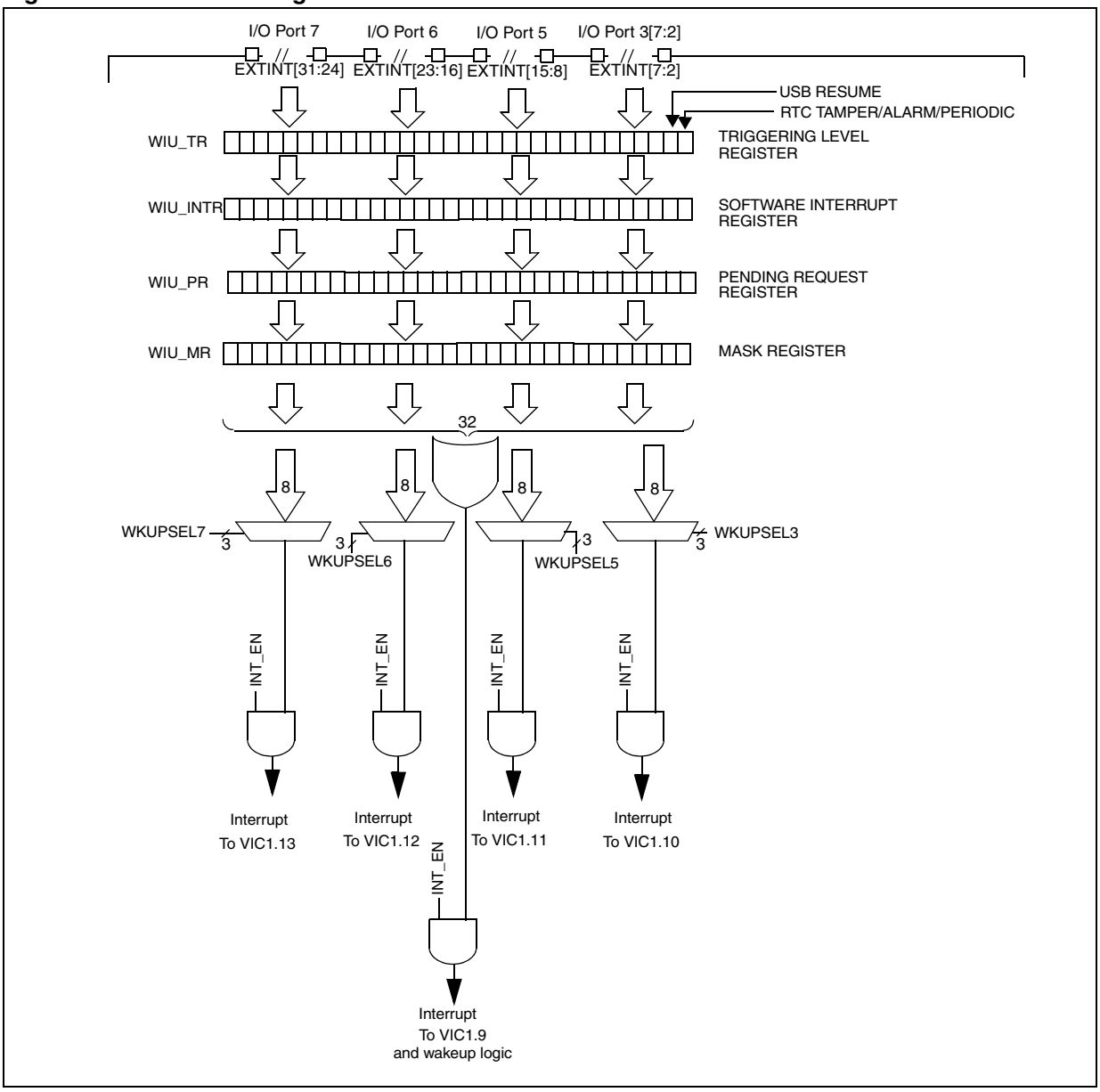
Using the WIU registers, 30 I/O port interrupts and the USB resume, RTC Alarm, Tamper or Periodic interrupts can be programmed as external interrupt lines or as wakeup lines, able to wakeup the STR91xFA from Sleep mode.

The Wakeup signal is an ORed function of the 30 I/O port interrupts, the USB and RTC interrupts (refer to *Figure 33: WIU block diagram*). The Wakeup signal is routed as an input to the VIC1 channel 9 as well as to the Wakeup Logic. When the Wakeup signal is asserted, an interrupt and a wakeup event is generated. You can disable the VIC 1 channel 9 if you do not want an interrupt when the CPU is waking up.

4.12.1 Features

- 30 I/O Port EXINT interrupt pins can be used to wakeup the MCU from Sleep mode or generate an IRQ or FIQ to the core via the Vectored Interrupt Controller (VIC1)
- Wakeup from Sleep Mode can be triggered by RTC Alarm/Tamper/Periodic or USB Resume events
- Programmable trigger edge polarity on EXINT pins
- All EXINT pins individually maskable
- Software interrupt register can be used to generate wakeup/interrupt events

Figure 33. WIU block diagram



1. Refer to the System Controller chapter (SCU) for a description of the WKUPSELx bits.

4.12.2 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
Read/clear (rc_w1)	Software can read as well as clear this bit by writing 1. Writing ‘0’ has no effect on the bit value
Read/set (rs)	Software can read as well as set this bit. Writing ‘0’ has no effect on the bit value

WIU control register (WIU_CTRL)

Address offset: 00h

Reset value: 0000 0000 0000 0000

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														INT_ EN	Res.
														rw	

Bits 31:2	Reserved, must be kept at reset value
Bit 1	**INT_EN:** *Global WIU Interrupt Enable.* This bit is set and cleared by software. It provides a global mask for WIU interrupts to the VIC. 0: WIU interrupts disabled 1: WIU interrupts enabled ***Caution*:** To avoid spurious interrupt requests to the VIC due to change of interrupt source, it is recommended to clear the corresponding enable bit in the VIC1_IntEnable register before modifying the INT_EN bit.
Bit 0	Reserved, must be kept at reset value

WIU mask register (WIU_MR)

Address offset: 04h

Reset value: 0000 0000 0000 0000

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

Bits 31:0	**WUM[31:0]**: *WIU Mask bits* These bits are set and cleared by software. They provide an individual mask for each of the 30 EXTINT lines, RTC/USB interrupts or software interrupts. The EXTINT lines are controlled by WUM[31:2], the USB interrupt is controlled by WUM1, and the RTC interrupt is controlled by WUM0. If WUMx is set, an interrupt and/or a wakeup event are generated if the corresponding WUPx pending bit is set. 0: WIU interrupt disabled (masked) 1: WIU interrupt enabled ***Note*:** If WUMx and WUPx are 1 and INT_EN is set to 1, then an interrupt is generated. It will also generate a wakeup event if the CPU is in low power mode. You can disable the VIC1 channel 9 if you do not want an interrupt when the CPU is waking up.

WIU trigger register (WIU_TR)

Address offset: 08h

Reset value: 0000 0000 0000 0000

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

Bits 31:0	**WUT[31:0]**: *Wakeup Trigger Polarity bits* These bits are set and cleared by software. They select whether the corresponding WUPx pending bit will be set on the falling or rising edge of the EXTINT line/RTC/USB interrupt. The EXTINT lines are controlled by WUT[31:2], the USB interrupt is controlled by WUT1, and the RTC interrupt is controlled by WUT0. 0: Falling edge 1: Rising edge ***Caution***: - As the external EXTINT lines are edge-triggered, no glitches must be generated on these lines. **-** If either a rising or a falling edge on an external EXTINT line occurs when writing to the WIU_TR register, the pending bit will not be set.

WIU software interrupt register (WIU_INTR)

Address offset: 10h

Reset value: 0000 0000 0000 0000

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

Bits 31:0	**WUINT[31:0]**: *WIU Software Interrupt bits* The WUINTx bits are set by software to generate a software interrupt. Setting one of these bits sets the corresponding bit in the Pending register (WIU_PR). The WUINTx bits are reset when the pending bits are cleared by writing a ‘1’ in the corresponding WUPx bit. 0: No effect 1: Software interrupt

WIU pending register (WIU_PR)

Address offset: 0Ch

Reset value: 0000 0000h

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

Bits 31:0	**WUP[31:0]**: *WIU Pending bits* The WUPx bits are Read/Clear, they are set by hardware on occurrence of the trigger event. They can be reset by software writing a ‘1’; writing a ‘0’ is ignored. 0: No wakeup trigger event occurred 1: Wakeup Trigger event occurred ***Note*:** The WUPx bits can be set by software, setting the corresponding bits in the Software Interrupt register (WIU_INTR) and choosing the trigger level high (WUTx set to ‘1’).

4.12.3 WIU register map

Table 15. WIU 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	WIU_CTRL	Reserved																														INT _EN	Res.
04h	WIU_MR	WUMR[31:0]																															
08h	WIU_TR	WUTR[31:0]																															
0Ch	WIU_PR	WUPR[31:0]																															
10h	WIU_INTR	WUINTR[31:0]																															

Refer to *Table 5 on page 35* for the register base addresses.

5 Real time clock (RTC)

5.1 Introduction

The RTC block combines a complete time of day clock with alarm, periodic interrupt, tamper detection and a 9999-year calendar. The time is in 24 hour mode, and time/calendar values are stored in binary-coded decimal format.

The RTC has separate power and clock connections that are activated during power down: this feature allows the RTC to continue working when the rest of the MCU is powered off.

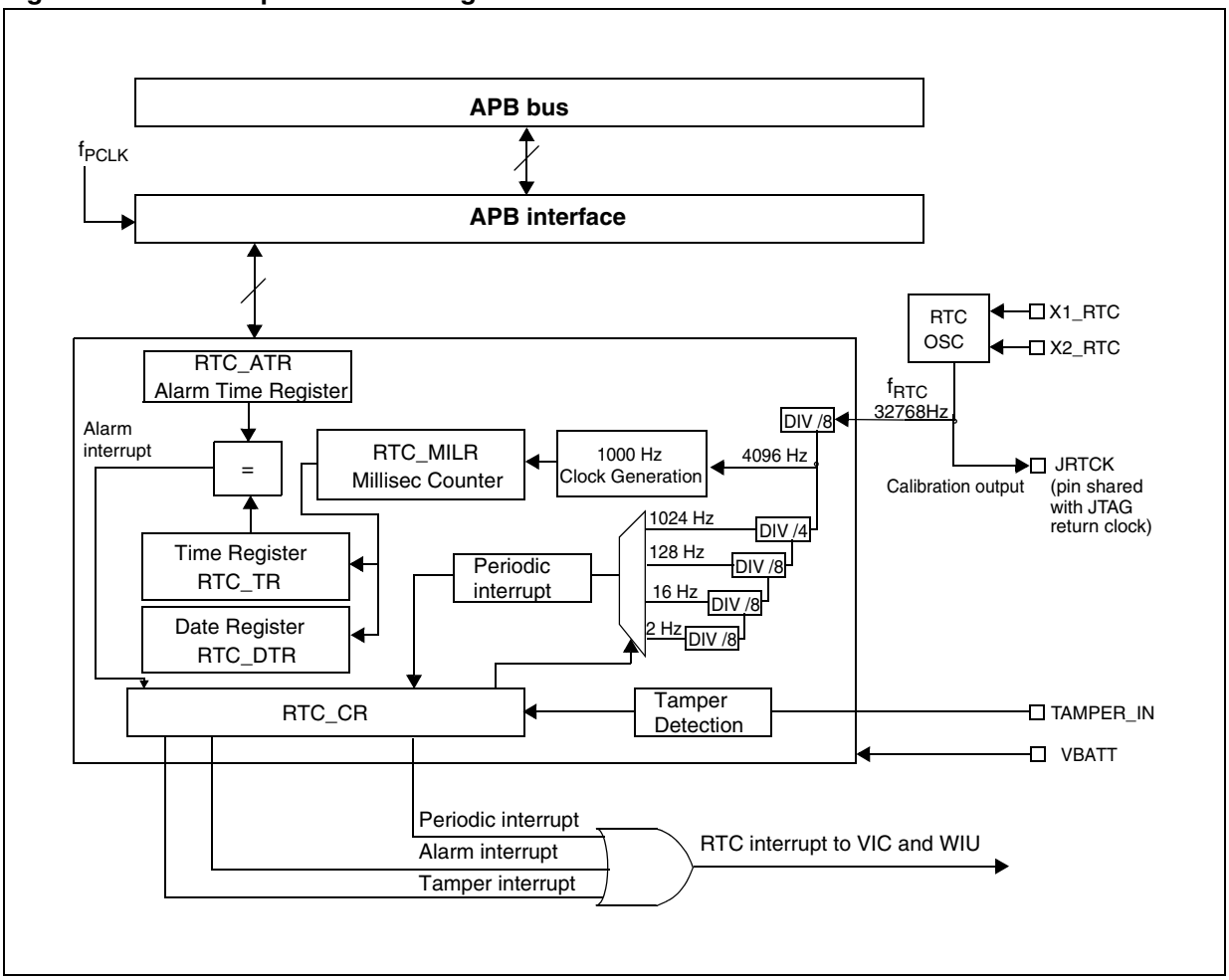
5.2 Main features

- Time of Day in 24 hour mode
- 9999-year calendar
- Leap year support
- Programmable alarm interrupt, supporting up to one month range with wakeup from Sleep mode capability
- Tamper detection with Time Stamp and Wakeup from Sleep mode capability
- Programmable periodic interrupt (1024 Hz / 128 Hz /16 Hz /2 Hz)
- Millisecond Real Time Clock
- Clock Calibration output
- Remains active during power down

Note: 1 Daylight saving time and 12 hour clock are not supported

2 The TAMPER_IN pin is not available on all packages, refer to datasheet for details)

Figure 34. RTC simplified block diagram



5.2.1 RTC clock control

The RTC is clocked by f_{RTC} which allows it to continue operating even when the microcontroller is in Sleep mode. The APB interface to the RTC registers is clocked by PLCK. Refer to *Figure 34.*

5.2.2 Battery backup

The VBATT pin can supply power to the RTC unit, allowing the RTC to function even when the main digital supplies (V_{DD} and V_{DDQ}) are switched off. By configuring the PWR bit in the RTC_CR register, you can select to power only the RTC or both the RTC and the SRAM from VBATT.

5.3 Reset

The contents of the following registers are not affected by a system reset.

- Time register
- Date Register

5.4 Clock calibration output

When the C bit in the *RTC control register (RTC_CR)* is set, the JTRTC outputs a 32.768 kHz clock. This can be used to calibrate an external clock source. When enabled, this clock output is active during Sleep Mode and can be used to control the wakeup logic.

The RTC clock can be manually calibrated by adjusting the Trim Capacitors on the RTC crystal circuit while monitoring the output clock on the JRTCK pin.

5.5 Time of day clock / calendar

The *RTC time register (RTC_TR)* and *RTC date register (RTC_DTR)* can be read at any time, they provide the current date and time updated by the RTC millisecond timer.

To set up the Time of day Clock/Calendar:

1. Set the W bit in the *RTC control register (RTC_CR)* to enable time register write mode
2. Write the Time /day of month in the *RTC time register (RTC_TR)*, the date in the *RTC date register (RTC_DTR)* and the millisecond in the *RTC millisecond register (RTC_MILR)*.
3. Clear the W bit in the *RTC control register (RTC_CR)* to disable time register write mode and update the internal registers

5.6 Tamper detection

The tamper detect feature monitors the level of the TAMPER_IN input pin, logs when a tamper event occurs (time stamp) and cuts off the SRAM standby voltage source to invalidate all SRAM contents.

To set up Tamper detection:

1. Select the operating mode by writing to the TM bit in the *RTC control register (RTC_CR)*. Writing 1, has no effect. By writing 0, the “Driven Low/High” scheme is activated.
2. If TM=0, select the trigger polarity by writing to the TIS bit in the *RTC control register (RTC_CR)*. For example, when set to 1, the tamper detect logic will register a tamper when the input goes high; and when set to 0, the tamper detect logic will register a tamper when the input goes low.
3. Enable the tamper detection logic by setting the TE bit *RTC control register (RTC_CR)*.

When a tamper event occurs:

- The RTC Time, Date and Millisecond counters are latched into the Time, Date and Millisecond registers to record the tamper event
- The SRAM standby voltage is cut off
- The TSF flag is set in the *RTC status register (RTC_SR)*
- An interrupt request and a wakeup signal is sent to the Vectored Interrupt Controller (VIC) and Wakeup Interrupt Unit (WIU) if the TDIE bit in the *RTC control register (RTC_CR)* is set

The application can read the Time, Date, and Millisecond registers for the time stamp information.

Software must then clear the Tamper status flag (TSF) and Time stamp information by resetting the TE bit in the *RTC control register (RTC_CR)*.

To resume Tamper detection, set the TE bit.

5.7 Alarm

The Alarm feature compares the *RTC alarm time register (RTC_ATR)* with the *RTC time register (RTC_TR)* and when their contents match, it generates an Alarm event. Both registers include a date field, allowing an Alarm event to be programmed up to 31 days in advance.

To set up the Alarm:

1. Set the W bit in the *RTC control register (RTC_CR)* to enable time register write mode
2. Write the Alarm time /day of month in the *RTC alarm time register (RTC_ATR)*
3. Clear the W bit in the *RTC control register (RTC_CR)* to disable time register write mode and update the internal registers
4. Enable the Alarm by setting the AFE bit in the *RTC control register (RTC_CR)*
5. You can optionally enable interrupts by setting the AFIE bit in the *RTC control register (RTC_CR)*

When an Alarm event occurs:

- The AF flag is set in the *RTC status register (RTC_SR)*
- An interrupt request and a wakeup signal is sent to the Vectored Interrupt Controller (VIC) and Wakeup Interrupt Unit (WIU) if the AFIE bit in the *RTC control register (RTC_CR)* is set

Software must then clear the Alarm status flag (AF) by resetting the AFE bit in the *RTC control register (RTC_CR)*.

5.8 Periodic interrupt

The RTC can be configured to generate a periodic event, setting a flag and optionally generating an interrupt at regular intervals. Selectable rates are 1024 Hz, 128Hz, 16 Hz and 2 Hz. This feature does not operate in Sleep mode.

To set up the Periodic Interrupt:

1. Enable the feature and select the rate by setting one of the PISEL[3:0] bits *RTC control register (RTC_CR)*.

At the end of every selected period:

- The PISF flag is set in the *RTC status register (RTC_SR)*.
- An interrupt request is sent to the Vectored Interrupt Controller (VIC) if the PIE bit in the *RTC control register (RTC_CR)* is set.

Note: Do not poll the PISF flag before the interrupt request is generated.

Software must then clear the Periodic Interrupt Status flag (PISF) by reading the *RTC status register (RTC_SR)*.

To disable periodic interrupts, clear the PIE bit and/or clear all the PISEL[3:0] bits

5.9 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’.

5.9.1 RTC time register (RTC_TR)

Address offset: 00h

Reset value: N/A (not affected by reset)

Note: The time register is updated automatically by the RTC. It can be read to obtain the current date and time in days, hours, minutes and seconds. It can only be written when the W bit in the RTC_CR register is set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DT[1:0]		DU[3:0]				Reserved		HT[1:0]		HU[3:0]			
		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
Res.	MIT[2:0]			MIU[3:0]				Res.	ST[2:0]			SU[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:30	Reserved, forced by hardware to 0
Bits 29:28	**DT[1:0]:** *Date Tens in BCD Format*
Bits 27:24	**DU[3:0]:** *Date Units in BCD Format* Range DT+DU: 01-31
Bits 23:22	Reserved, forced by hardware to 0
Bits 21:20	**HT[1:0]:** *Hour Tens in BCD Format*
Bits 19:16	**HU[3:0]:** *Hour Units in BCD Format* Range HT+HU: 00-23
Bit 15	Reserved, forced by hardware to 0
Bits 14:12	**MIT[2:0]:** *Minute Tens in BCD Format*
Bits 11:8	**MIU[3:0]:** *Minute Units in BCD Format* Range MIT+MIU: 00-59
Bit 7	Reserved, forced by hardware to 0
Bits 6:4	**ST[2:0]:** *Second Tens in BCD Format*
Bits 3:0	**SU[3:0]:** *Second Units in BCD Format* Range ST+SU: 00-59

5.9.2 RTC date register (RTC_DTR)

Address offset: 04h

Reset value: N/A (not affected by reset)

The date register is updated automatically by the RTC. It can be read to obtain the weekday, month, year and century. It can only be written when the W bit in the RTC_CR register is set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CT[3:0]				CU[3:0]				YT[3:0]				YU[3:0]			
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			MT	MU[3:0]				Reserved				WDU[3:0]			
			rw	rw	rw	rw	rw					rw	rw	rw	rw

Bits 31:28	**CT[3:0]:** *Century Tens in BCD Format*
Bits 27:24	**CU[3:0]:** *Century Units in BCD Format* Range CT+CU: 00-99
Bits 23:20	**YT[3:0]:** *Year Tens in BCD Format*
Bits 19:16	**YU[3:0]:** *Year Units in BCD Format* Range YT+YU: 00-99
Bits 15:13	Reserved, forced by hardware to 0
Bit 12	**MT:** *Month Tens in BCD Format*
Bits 11:8	**MU[3:0]:** *Month Units in BCD Format* Range MT+MU: 01-12
Bits 7:4	Reserved, forced by hardware to 0
Bits 3:0	**WDU[3:0]:** *Weekday Units in BCD Format* Range 01-07

5.9.3 RTC alarm time register (RTC_ATR)

Address offset: 08h

Reset value: Not affected by reset

Note: The alarm time register contains the alarm date and time in hours, minutes and seconds.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		ADT[1:0]		ADU[3:0]				Reserved		AHT[1:0]		AHU[3:0]			
		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
Res.	AMIT[2:0]			AMIU[3:0]				Res.	AST[2:0]			ASU[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:30	Reserved, forced by hardware to 0
Bits 29:28	**ADT[1:0]:** *Alarm Date Tens in BCD Format*
Bits 27:24	**ADU[3:0]:** *Alarm Date Units in BCD Format* Range ADT+ADU: 01-31
Bits 23:22	Reserved, forced by hardware to 0
Bits 21:20	**AHT[1:0]:** *Alarm Hour Tens in BCD Format*
Bits 19:16	**AHU[3:0]:** *Alarm Hour Units in BCD Format* Range AHT+AHU: 00-23
Bit 15	Reserved, forced by hardware to 0.
Bits 14:12	**AMIT[2:0]:** *Alarm Minute Tens in BCD Format*
Bits 11:8	**AMIU[3:0]:** *Alarm Minute Units in BCD Format* Range AMIT+AMIU: 00-59
Bits 7	Reserved, forced by hardware to 0
Bits 6:4	**AST[2:0]:** *Alarm Second Tens in BCD Format*
Bits 3:0	**ASU[3:0]:** *Alarm Second Units in BCD Format* Range AST+ASU: 00-59

5.9.4 RTC control register (RTC_CR)

Address offset: 0Ch

Reset value: Bits [31:16] and [1:0] are not affected by system and global reset. Bits 15:2 are reset to 0 by system reset but, are not affected by LVD reset in V_{BAT} mode.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								AIE	TIE	PIE	AE	PISEL[3:0]			
								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								W	RTCSEL	Res.	TM	PWR	TIS	Res.	TE
								rw	rw		rw	rw	rw		rw

Bits 31:24	Reserved, forced by hardware to 0
Bit 23	**AIE:** *Alarm Interrupt Enable* 0: Alarm interrupt disabled 1: Alarm interrupt enabled. An interrupt is generated when the AF bit in the RTC_SR register is set.
Bit 22	**TIE:** *Tamper Interrupt Enable* 0: Tamper interrupt disabled 1: Tamper interrupt enabled. A Tamper interrupt is generated when the TS bit in the RTC_SR register is set.
Bit 21	**PIE:** *Periodic Interrupt Enable* 0: Periodic interrupt disabled 1: Periodic interrupt enabled. An interrupt is generated when the PI bit in the RTC_SR register is set.
Bit 20	**AE:** *Alarm Enable* 0: Alarm disabled 1: Alarm enabled. The AF bit in the RTC_SR register is set when the value in the Time register (RTC_TR) matches the Alarm Time register (RTC_ATR).
Bits 19:16	**PISEL[3:0]:** *Periodic Interrupt select* 0001: 2 Hz period selected 0010: 16 Hz period selected 0100: 128Hz period selected 1000: 1024 Hz period selected
Bits 15:8	Reserved, forced by hardware to 0
Bit 7	**W:** *Write Enable* This bit is set and cleared by software. It must be set to enable write access to the RTC_TR, RTC_DTR and RTC_MILR registers. The W bit must be cleared after the write operation for the registers to be updated. 0: Write access to RTC_TR, RTC_DTR and RTC_MILR disabled 1: Write access to RTC_TR, RTC_DTR and RTC_MILR disabled

Bit 6	**RTCSEL:** *Calibration Clock Output Enable* This bit is set and cleared by software. 0: Calibration clock output disabled 1: Calibration clock output enabled, clock remains active during Sleep Mode
Bit 5	Reserved, forced by hardware to 0.
Bit 4	**TM:** *Tamper Mode* 0: Driven Low/High Scheme. If TE = 1, a tamper event is triggered when the TAMPER_IN input is driven high or low depending on the TIS bit. 1: No effect.
Bit 3	**PWR:** *SRAM V_{BATT} power control* This bit is set and cleared by software. It is cleared by hardware when a Tamper event occurs. Once this bit is set and SRAM is connected to V_{BATT}, it will not be cleared by a subsequent reset. 0: SRAM *disconnected from V_{BATT}* 1: *SRAM connected to V_{BATT}*
Bit 2	**TIS:** *Trigger selection* This bit is set and cleared by software. 0: Tamper event triggered when Tamper input goes low 1: Tamper event triggered when Tamper input goes high
Bit 1	Reserved, forced by hardware to 0
Bit 0	**TE:** *Tamper input enable* This bit is set and cleared by software. 0: Tamper input pin disabled 1: Tamper input pin enabled After a tamper event occurs the Tamper time stamp can be read in the Time and Date registers. The TE bit must then be cleared by software.

Note: After enabling the RTC clock, the Control register (RTC_CR) and the alarm time register (RTC_ATR) must be cleared by software.

5.9.5 RTC status register (RTC_SR)

Address offset: 10h

Reset value: 0000h (reset by system reset but, not affected by LVD reset in V_{BAT} mode mode).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PISF	AF	Res.	TSF	Reserved											
r	r	r	r												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															

Bit 31	**PISF:** *Periodic Interrupt Flag* This bit is set by hardware. It does not set when the RTC is running on battery. It is cleared by reading the register. 0: No periodic event 1: A periodic event occurred as configured by the PISEL bits in the RTC_CR register. An interrupt is generated if the PIE bit in the RTC_CR register is set.
Bit 30	**AF:** *Alarm flag* This bit is set by hardware. It is cleared by clearing the AE bit in the RTC_CR register. 0: No Alarm event 1: An alarm event occurred. An interrupt is generated if the AIE bit in the RTC_CR register is set.
Bit 29	Reserved, forced by hardware to 0
Bit 28	**TSF:** *Tamper flag* This bit is set by hardware. It is cleared by clearing the TE bit in RTC_CR register. 0: No Tamper event 1: An Tamper event occurred. An interrupt is generated if the TIE bit in the RTC_CR register is set.
Bits 27:0	Reserved, forced by hardware to 0

5.9.6 RTC millisecond register (RTC_MILR)

This register contains current value of the RTC millisecond counter. When it is read, it returns the last updated value of the millisecond counter. Before writing to this register, the W bit must be set to '1' in the RTC_CR register.

Note: After all the clock registers (Time, Date, and Millisecond) are written, you must reset the W bit in the RTC_CR to update the clock registers. As long as W bit remains high, the clock registers will stay in write mode and the millisecond clock will not update these registers.

Address offset: 14h

Read/Write

Reset value: Not affected by reset

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				MSH[3:0]				MST[3:0]				MSU[3:0]			
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12	Reserved, forced by hardware to 0
Bits 11:8	**MSH[3:0]:** *Millisecond Hundreds in BCD Format*
Bits 7:4	**MST[3:0]:** *Millisecond Tens in BCD Format*
Bits 3:0	**MSU[3:0]:** *Millisecond Units in BCD Format* Range MSH+MST+MSU: 000-999

5.10 RTC register map

RTC registers are mapped as 32-bit addressable registers as described in the table below:

Table 16. RTC register map

Addr. off set	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	RTC_TR	Time register																															
04h	RTC_DTR	Date register																															
08h	RTC_ATR	Alarm Time register																															
0Ch	RTC_CR	Control register																															
10h	RTC_SR	Status				Reserved																											
14h	RTC_MILR	Reserved																				Millisecond Register											

Refer to *Table 5 on page 35* for the register base addresses.

6 Watchdog timer (WDG)

6.1 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.

6.2 Main features

- 16-bit down Counter
- 8-bit clock Prescaler
- Safe Reload Sequence
- Free-running Timer mode
- End of Counting interrupt generation

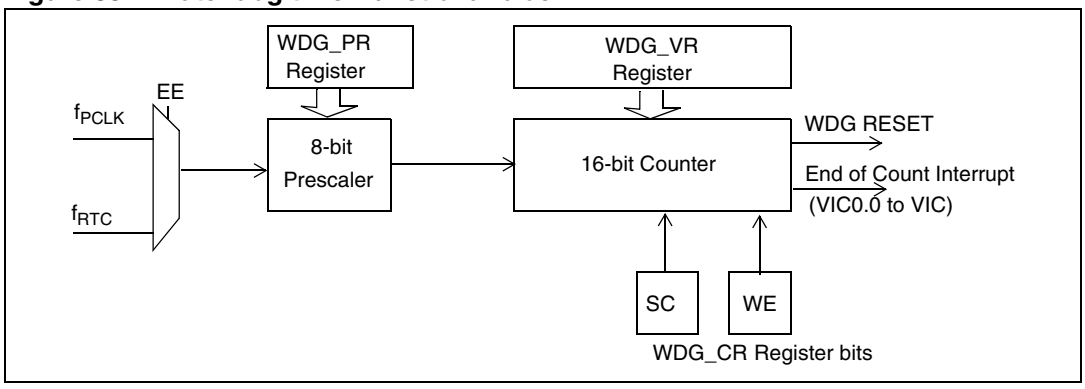
6.3 Functional description

Figure 35 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.

6.3.1 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 35. 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).

6.3.2 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 6.4*, the WDG_VR value is re-loaded in the Counter, the End of count can be prevented.

6.3.3 Programming considerations

The following method must be used when attempting to change the WDG clock source (by changing the EE bit in the WGD register) when the counters are running:

- Set the SC bit to '0' to stop the counters
- Write the new clock value to the EE bit
- Set the SC bit to '1' to restart the counters with the new clock value

The following method must be used when writing to the WDG prescaler register (WDG_PR) or the WDG preload value register (WDG_VR):

- Set the SC bit to '0' to stop the counters
- Write to the WDG_PR or WDG_VR register
- Set the SC bit to '1' to restart the counters with the new value

The above ensures that the counters operate correctly.

6.4 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’.

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

6.4.1 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													EE	SC	WE
-													rw	rw	rw

Bits 15:2	Reserved. Forced by hardware to 0
Bit 2	**EE**: *External Clock Enable* This bit is set and cleared by software. It selects the WDG clock source. This bit can be modified only when the Watchdog Timer is not in Watchdog mode. 0: PCLK 1: 32 kHz RTC clock
Bit 1	**SC**: *Start Counting bit* 0: The counter is stopped 1: The prescaler and counter are load with the preload values in the WDG_PR and WDG_TVR registers and the counter starts counting. This bit is effective 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.

6.4.2 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

Bits 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.

6.4.3 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

Bits 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 = 20 MHz the default time-out set after the system reset is:

256*65535*50/1000 = 838861µs.

6.4.4 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
CNT 15	CNT 14	CNT 13	CNT 12	CNT 11	CNT 10	CNT 9	CNT 8	CNT 7	CNT 6	CNT 5	CNT 4	CNT 3	CNT 2	CNT 1	CNT 0
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 value of the 16-bit Counter can be obtained by reading this register.

6.4.5 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
-															rc_w0

Bits 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.

6.4.6 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

Bits 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

6.4.7 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 WTDPR 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.

6.5 Watchdog timer register map

Table 17. Watchdog timer register map

Address offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WDG_CR	Reserved													EE	SC	WE
4	WDG_PR	Reserved								PR[7:0]							
8	WDG_VR	TV[15:0]															
C	WDG_CNT	CNT[15:0]															
10	WDG_SR	Reserved															EC
14	WDG_MR	Reserved															ECM
18	WDG_KR	K[15:0]															

Refer to *Table 5 on page 35* for the register base addresses.

7 16-bit timer (TIM)

7.1 Introduction

The timer consists of a 16-bit counter driven by a programmable prescaler.

It may be used for a variety of purposes, including measuring the pulse lengths of up to two input signals (input capture) or generating up to two output waveforms (output compare and PWM).

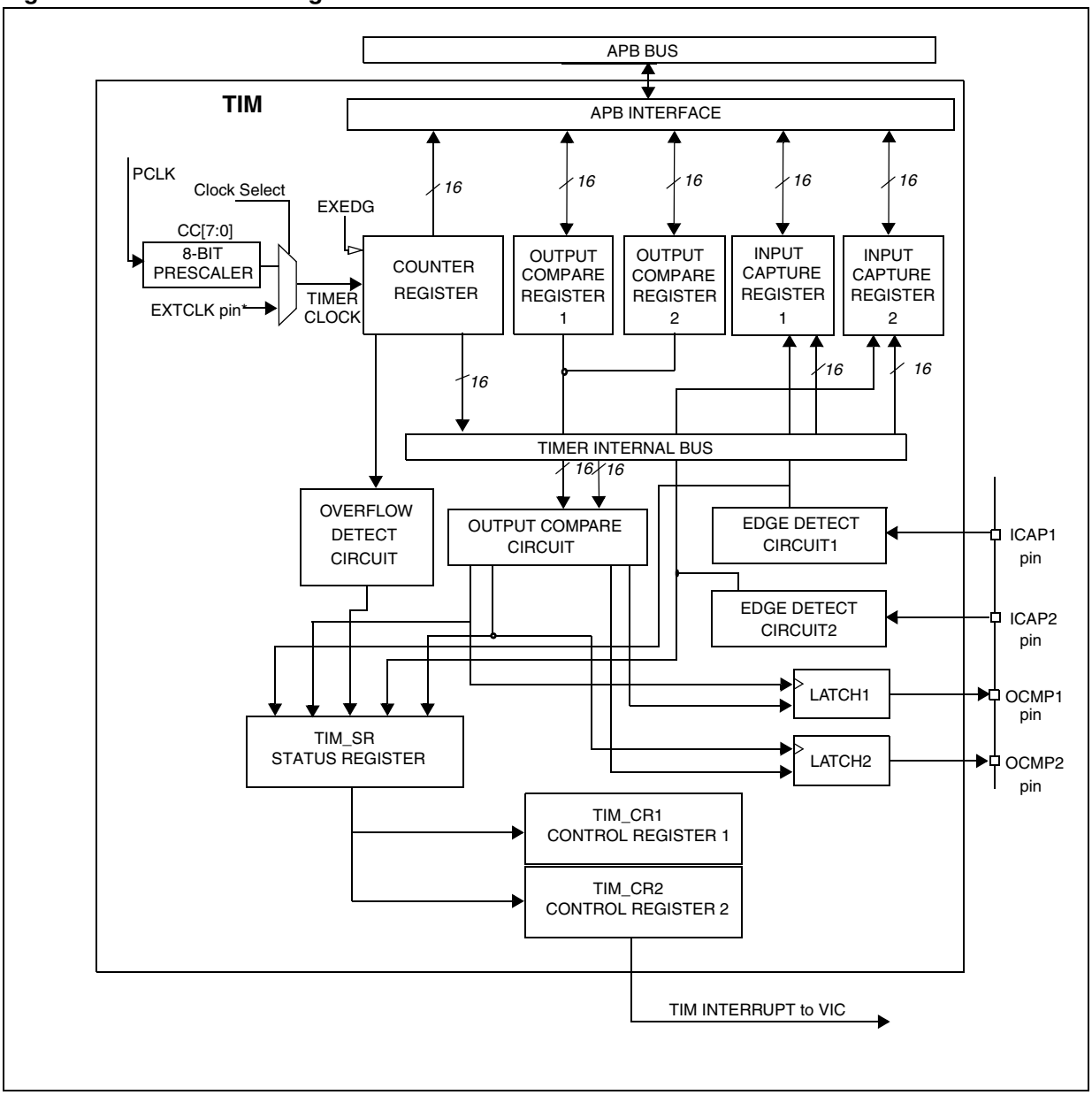
Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds.

7.2 Main features

- Programmable prescaler: f_{PCLK} divided by 1 to 256 in steps of 1
- Overflow status flag and maskable interrupt
- External clock inputs with the choice of active edge (see External Clock section on frequency limitation).
- Output compare functions with:
  - 2 dedicated 16-bit registers
  - 2 dedicated programmable signals
  - 2 dedicated status flags
  - 1 maskable interrupt
- Input capture functions with:
  - 2 dedicated 16-bit registers
  - 2 dedicated active edge selection signals
  - 2 dedicated status flags
  - 1 maskable interrupt
- Pulse Width Modulation output mode (PWM)
- One Pulse mode (OPM)
- PWM input mode (PWMI)
- 5 alternate functions on I/O ports (ICAP1, ICAP2, OCMP1, OCMP2, EXTCLK)
- DMA support (TIM0 and TIM1)

The block diagram is shown in *Figure 36*.

Figure 36. Timer block diagram



1. To select the external clock pin, set bit 13 or 14 in the SCU_CLKCNTR register

7.3 Functional description

7.3.1 Counter

The main block of the Programmable Timer is a 16-bit upcounter and its associated 16-bit registers.

Counter Register (TIM_CNTR)

Writing in the TIM_CNTR register resets the free running counter to the FFFCh value.

The timer clock source can be either internal or external as selected by the ECKEN bit in the TIM_CR1 register and the TIMxSEL bit in the SCU_CLKCNTR register. When internal clock (PCLK) is selected, the frequency depends on the value programmed inthe CC[7:0] bits in the TIM_CR2 register.

An overflow occurs when the counter rolls over from FFFFh to 0000h then:

- The TOF bit in the TIM_SR register is set.
- A timer interrupt is generated if the TOIE bit in the TIM_CR2 register is set

To clear the interrupt, write 0 to the TOF bit in the TIM_SR register.

7.3.2 External clock

The external clock from external clock input pin is selected if ECKEN=1 in the TIM_CR1 register and TIMxSEL=1 in the SCU_CLKCNTR register.

The status of the EXEDG bit in the TIM_CR1 register determines the type of level transition on the external clock EXTCLK that will trigger the free running counter.

The counter is synchronized with the rising edge of PCLK.

A minimum of four falling edges of the PCLK must occur between two consecutive active edges of the external clock; thus the external clock frequency must be less than a quarter of the PCLK frequency.

Figure 37. Counter timing diagram, internal clock divided by 2

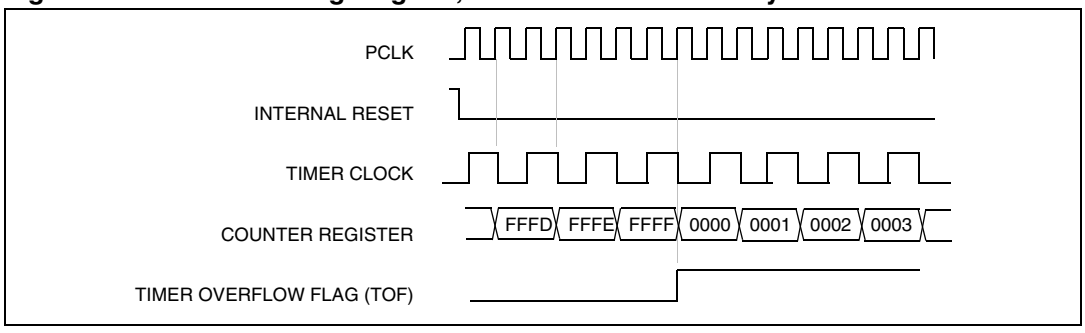


Figure 38. Counter timing diagram, internal clock divided by 4

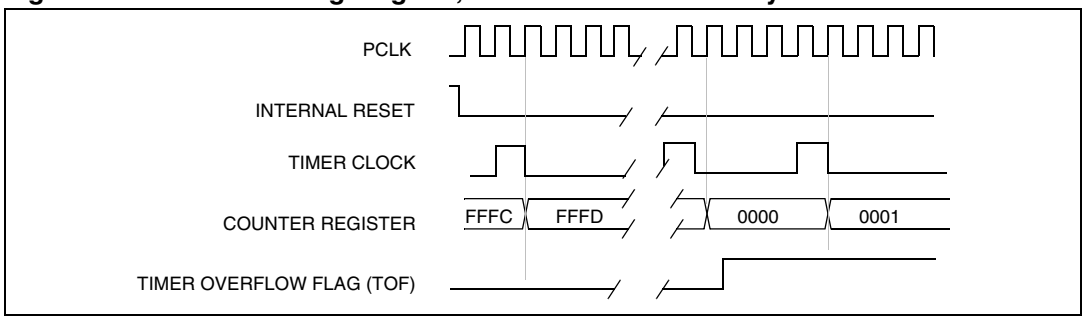
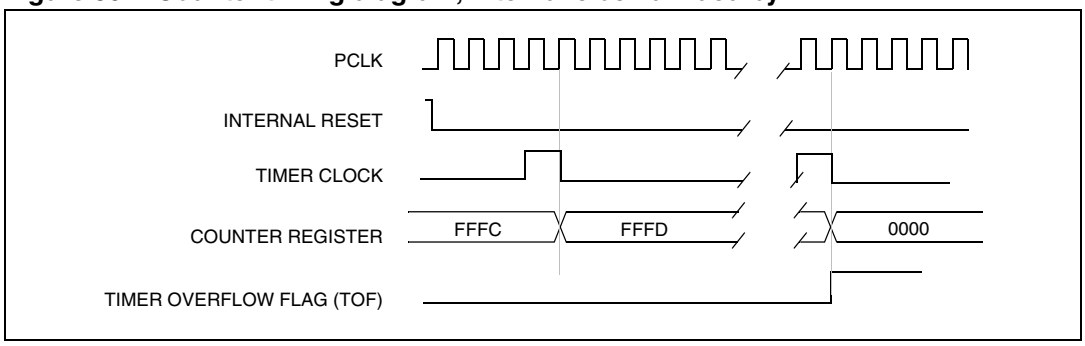


Figure 39. Counter timing diagram, internal clock divided by n



Note: The Timer's internal reset signal is controlled by bits [1:0] in the SCU_PRR1 register.

7.3.3 Input capture

In this section, the index, *i*, may be 1 or 2.

The two input capture 16-bit registers (TIM_IC1R and TIM_IC2R) are used to latch the value of the counter after a transition is detected by the ICAP*i* pin (see *Figure 39*).

The TIM_IC*i*R register is a read-only register.

The active transition is software programmable through the IEDG*i* bit in Control Register 1 (TIM_CR1).

Timing resolution is one/two counts of the counter: (f_{PCLK}/CC[7:0]+1).

Procedure

To use the input capture function, select the following in the TIM_CR1 and TIM_CR2 registers:

- Select the timer clock source (ECKEN bit) and the TIMxSEL bit in the SCU_CLKCNTR register.
- Program the timer clock prescaler CC[7:0] bits if PCLK is used (ECKEN = 0)
- Select the edge of the active transition on the ICAP1 pin with the IEDG1 bit if ICAP1 is active.
- Select the edge of the active transition on the ICAP2 pin with the IEDG2 bit if ICAP2 is active.
- Set the IC*i*IE bits to generate an interrupt after an input capture coming from the corresponding ICAP1 pin or ICAP2 pins.

When an input capture occurs:

- The ICF*i* bit is set.
- The TIM_IC*i*R register contains the value of the counter on the active transition on the ICAP*i* pin (see *Figure 41*).
- A timer interrupt is generated if the corresponding IC*i*E bit is set.

To clear the interrupt, write 0 to the ICF*i* bit in the TIM_SR register.

Figure 40. Input capture block diagram

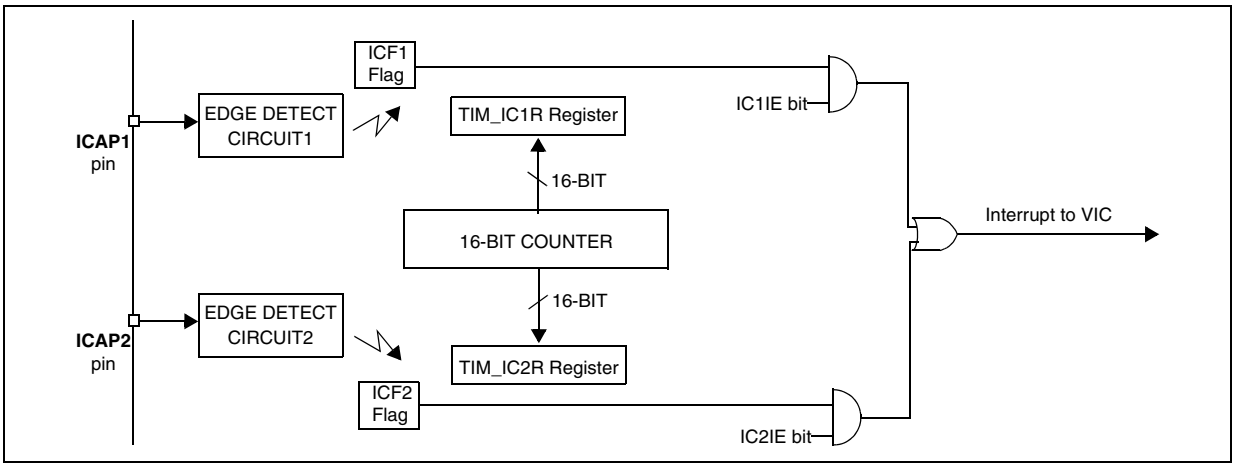
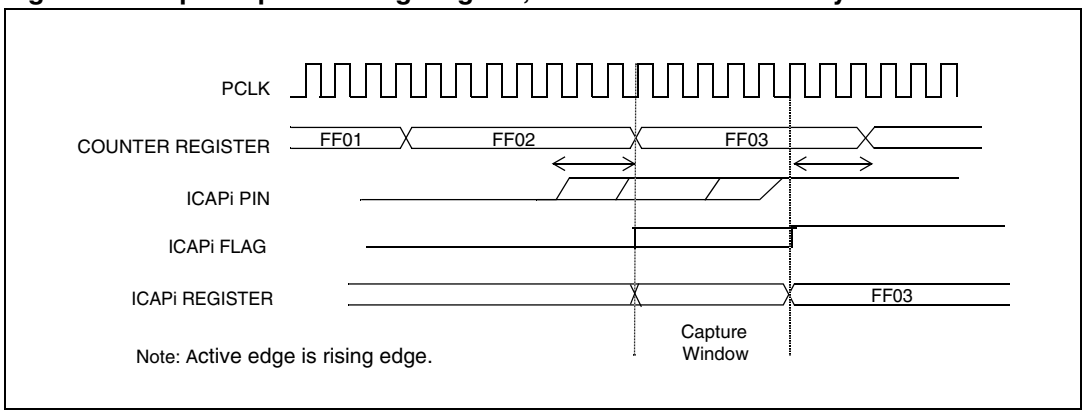


Figure 41. Input capture timing diagram, internal clock divided by 8



7.3.4 Output compare

In this section, the index, *i*, may be 1 or 2.

This function can be used to control an output waveform or indicate when a period of time has elapsed.

When a match is found between the Output Compare register and the counter, the output compare function:

- Assigns pins with a programmable value if the OC*i*E bit is set
- Sets a flag in the status register
- Generates an interrupt if enabled

Two 16-bit registers Output Compare Register 1 (TIM_OC1R) and Output Compare Register 2 (TIM_OC2R) contain the value to be compared to the counter register each timer clock cycle.

These registers are readable and writable and are not affected by the timer hardware. A reset event changes the TIM_OC*i*R value to 8000h.

Timing resolution is one count of the counter: ($f_{PCLK/CC[7:0]+1}$).

Procedure

To use the output compare function, select the following in the TIM_CR1 and TIM_CR2 registers:

- Set the OC*i*E bit if an output is needed then the OCMP*i* pin is dedicated to the output compare *i* function.
- Select the timer clock ECKEN and program the prescaler (CC[7:0])
- Select the OLVL*i* bit to applied to the OCMP*i* pins after the match occurs
- Set the OC*i*IE bit to generate an interrupt if required

When a match is found between TIM_OCRi register and TIM_CR register:

- The OCF*i* bit is set
- The OCMP*i* pin takes OLVL*i* bit value (OCMP*i* pin latch is forced low during reset and stays low until a valid compare changes it to OLVL*i* level).
- A timer interrupt is generated if the OC*i*IE bit is set in the TIM_CR1 register

To clear the interrupt, write 0 to the OCF*i* bit in the TIM_SR register.

The TIM_OC*i*R register value required for a specific timing application can be calculated using the following formula:

$$\Delta \text{ TIM\_OC}i\text{R} = \frac{\Delta t * f_{PCLK}}{(CC7:0+1)}$$

Where:

Δt = Output compare period (in seconds)

f_{PCLK} = Internal clock frequency

CC7:0 = Timer clock prescaler

Figure 42. Output compare block diagram

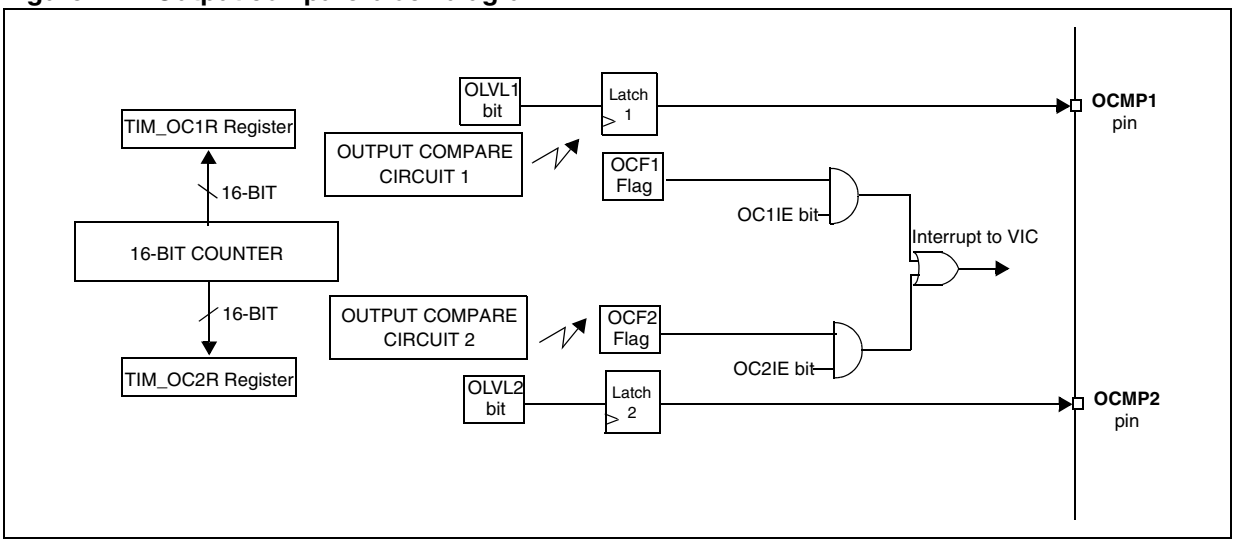
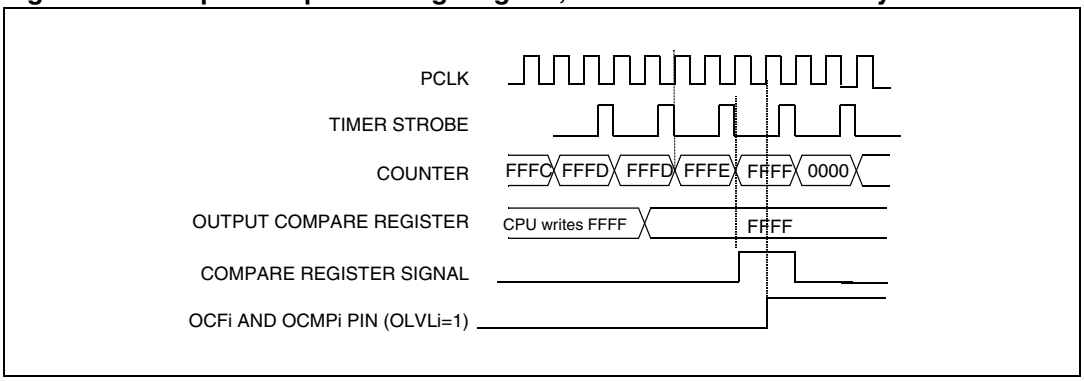


Figure 43. Output compare timing diagram, Internal Clock Divided by 2



7.3.5 Forced compare mode

In this section i may represent 1 or 2.

When the FOLV1 bit is set, the OLVL1 bit is copied to the OCMP1 pin if PWM and OPM are both cleared.

When FOLV2 bit is set, the OLVL2 bit is copied to the OCMP2 pin.

The OLVLi bit has to be toggled in order to toggle the OCMPi pin when it is enabled (OCiE bit=1).

Note: When FOLVi is set, no interrupt request is generated.

Nevertheless the OCFi bit can be set if OCiR = Counter and an interrupt can be generated if enabled.

Input capture function works in Forced compare mode.

7.3.6 One pulse mode

One Pulse mode (OPM) enables the generation of a pulse when an external event occurs. This mode is selected via the OPM bit in the TIM_CR1 register.

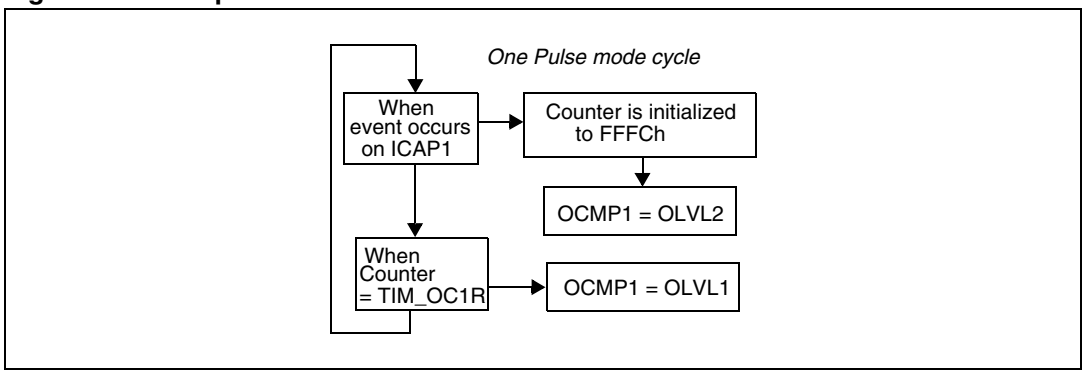
One Pulse mode uses the Input Capture1 function and the Output Compare1 function.

Procedure

To use One Pulse mode, select the following in the TIM_CR1 register:

- Using the OLVL1 bit, select the level to be applied to the OCMP1 pin after the pulse
- Using the OLVL2 bit, select the level to be applied to the OCMP1 pin during the pulse
- Select the edge of the active transition on the ICAP1 pin with the IEDG1 bit
- Set the OC1E bit, the OCMP1 pin is then dedicated to the Output Compare 1 function
- Set the OPM bit
- Select the timer clock ECKEN and the prescaler division factor CC[7:0]
- Load the OC1R register with the value corresponding to the length of the pulse (see the formula in PWM mode Section 1.1.3.7).

Figure 44. One pulse mode flowchart



Then, on a valid event on the ICAP1 pin, the counter is initialized to FFFCh and OLVL2 bit is loaded on the OCMP1 pin after 4 clock period. When the value of the counter is equal to the value of the contents of the OC1R register, the OLVL1 bit is output on the OCMP1 pin (see *Figure 45*).

Note: The OCF1 bit cannot be set by hardware in one pulse mode but the OCF2 bit can generate an Output Compare interrupt.

The ICF1 bit is set when an active edge occurs and can generate an interrupt if the ICIE bit is set. The IC1R register will have the value FFFCh.

When Pulse Width Modulation (PWM) and One Pulse Mode (OPM) bits are both set with FOLV1= 1, the OPM mode is the only active one, otherwise PWM mode is the only active one.

Forced Compare 2 mode works in OPM.

Input Capture 2 function works in OPM.

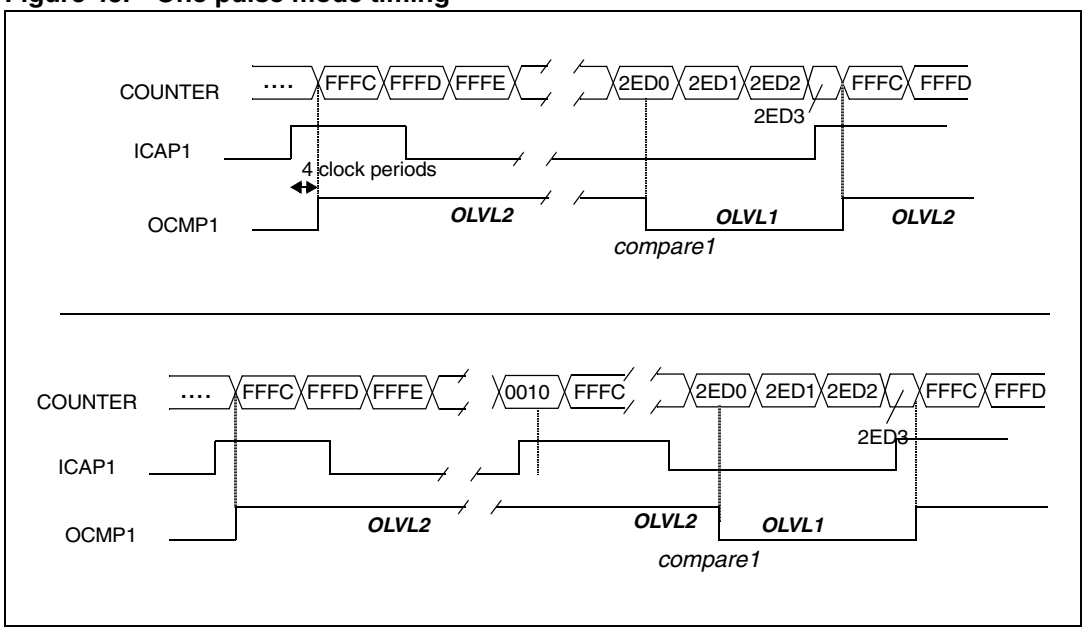
When OC1R = FFFBh in OPM, then a pulse of width FFFFh is generated.

If an event occurs on ICAP1 again before the Counter reaches the value of OC1R, then the Counter will be reset again and the pulse generated might be longer than expected (see *Figure 45*).

If a write operation is performed on the counter register before the Counter reaches the value of OC1R, then the Counter will be reset again and the pulse generated might be longer than expected.

If a write operation is performed on the counter register after the Counter reaches the value of OC1R, then there will be no effect on the waveform.

Figure 45. One pulse mode timing



1. In the top part of *Figure 45*, IEDG1=1, OC1R = 2ED0h, OLVL1 = 0, OLVL2 = 1.
2. In the bottom part of *Figure 45*, IEDG1 = 1, OC1R = 2ED0h, OLVL1 = 0, OLVL2 = 1.

7.3.7 Pulse width modulation mode

Pulse Width Modulation (PWM) mode enables the generation of a signal with a frequency and pulse length determined by the value of the TIM_OC1R and TIM_OC2R registers.

The Pulse Width Modulation mode uses the complete Output Compare 1 function plus the TIM_OC2R register.

Procedure

To use pulse width modulation mode select the following in the CR1 register:

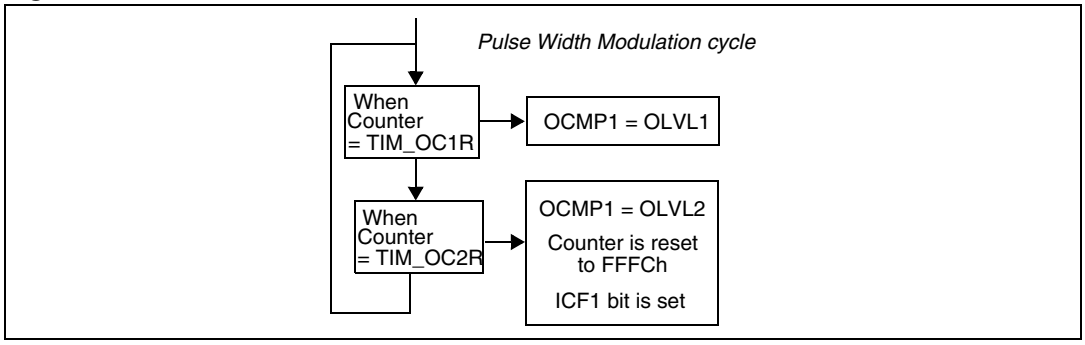
- Using the OLVL1 bit, select the level to be applied to the OCMP1 pin after a successful comparison with OC1R register.
- Using the OLVL2 bit, select the level to be applied to the OCMP1 pin after a successful comparison with OC2R register.
- Set OC1E bit: the OCMP1 pin is then dedicated to the output compare 1 function.
- Set the PWM bit.
- Select the timer clock (ECKEN) and the prescaler division factor (CC[7:0]).
- Load the OC2R register with the value corresponding to the period of the signal.
- Load the OC1R register with the value corresponding to the length of the pulse if (OLVL1= 0 and OLVL2 = 1).

If OLVL1= 1 and OLVL2 = 0 the length of the pulse is the difference between the OC2R and OC1R registers.

The OC*i*R register value required for a specific timing application can be calculated using the following formula:

$$OCiR\ Value = \frac{t * f_{PCLK}}{t_{PRESC}} - 5$$

Figure 46. PWM mode flowchart



Where:

t = Desired output compare period (seconds)

f_{PCLK} = Internal clock frequency

t_{PRESC} = Timer clock prescaler

The Output Compare 2 event causes the counter to be initialized to FFFCh (see *Figure 47*).

Note: 1 The OCF1 bit cannot be set by hardware in PWM mode, but OCF2 is set every time counter matches OC2R.

2 The Input Capture function is available in PWM mode.

3 When Counter = OC2R, then OCF2 bit will be set. This can generate an interrupt if OC2IE is set. This interrupt will help any application where pulse-width or period needs to be changed interactively.

4 When Pulse Width Modulation (PWM) and One Pulse Mode (OPM) bits are both set with FOLV1= 0, the PWM mode is the only active one, otherwise the OPM mode is the only active one.

5 The value loaded in OC2R must always be greater than *that in OC1R to produce meaningful waveforms. Note that 0000h is considered to be greater than FFFCh or FFFDh or FFFEh or FFFFh.*

6 When OC1R >OC2R, no waveform will be generated.

7 When OC2R = OC1R, a square *waveform with 50 % duty cycle will be generated as in Figure 47*

8 When OC2R and OC1R are loaded with FFFC (the counter reset value) then a square waveform will be generated & the counter will remain stuck at FFFC. The period will be calculated using the following formula:

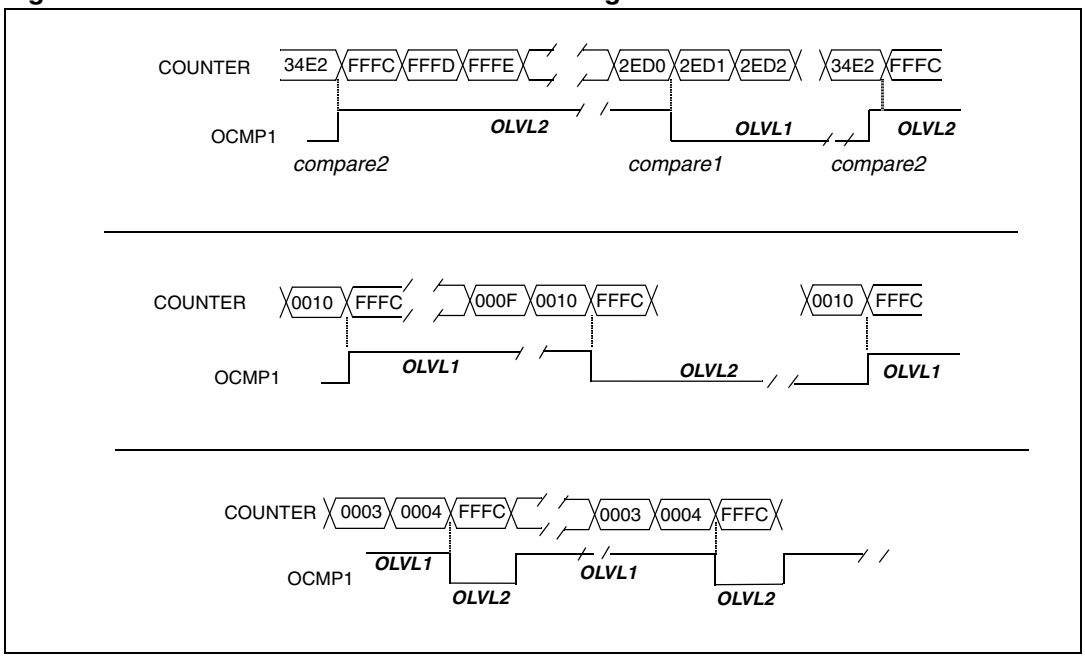
$$\text{Period} = \frac{2 * f_{PCLK}}{t_{PRESC}}$$

When OC1R is loaded with FFFC (the counter reset value) then the waveform will be generated as in Figure 47

When FOLV1 bit is set and PWM bit is set, then PWM mode is the active one. But if FOLV2 bit is set then the OLVL2 bit will appear on OCMP2(when OC2E bit = 1).

When a write is performed on CNTR register in PWM mode, then the Counter will be reset and the pulse-width/period of the waveform generated may not be as desired

Figure 47. Pulse width modulation mode timing



1. In the top part of *Figure 47*, OC1R = 2ED0h, OC2R = 34E2, OLVL1 = 0, OLVL2 = 1.
2. In the middle part of *Figure 47*, OC1R = OC2R = 0010h, OLVL1 = 1, OLVL2 = 0.
3. In the bottom part of *Figure 47*, OC1R = FFFCh, OC2R = 0004h, OLVL1 = 1, OLVL2 = 0.

7.3.8 Pulse width modulation input mode

The PWM Input functionality enables the measurement of the period and the pulse width of an external waveform. The initial edge is programmable.

It uses the two Input Capture registers and the Input signal of the Input Capture 1 module.

Procedure

The CR2 register must be programmed as needed for Interrupts and DMA.

To use PWM input mode select the following in the TIM_CR1 register:

- Set the PWMI bit
- Select the first edge in IEDG1
- Select the second edge IEDG2 as the negated of IEDG1
- Program the clock source and prescaler as needed.
- Enable the counter by setting the EN bit.

To have a coherent measurement the interrupt/DMA should be linked to the Input Capture 1 Interrupt, reading the period value in the TIM_IC1R register and in the pulse width in the IC2R register.

To obtain the time values:

$$\text{Period} = \frac{IC1R * f_{PCLK}}{t_{PRESC}}$$

$$\text{Pulse} = \frac{IC2R * f_{PCLK}}{t_{PRESC}}$$

Where:

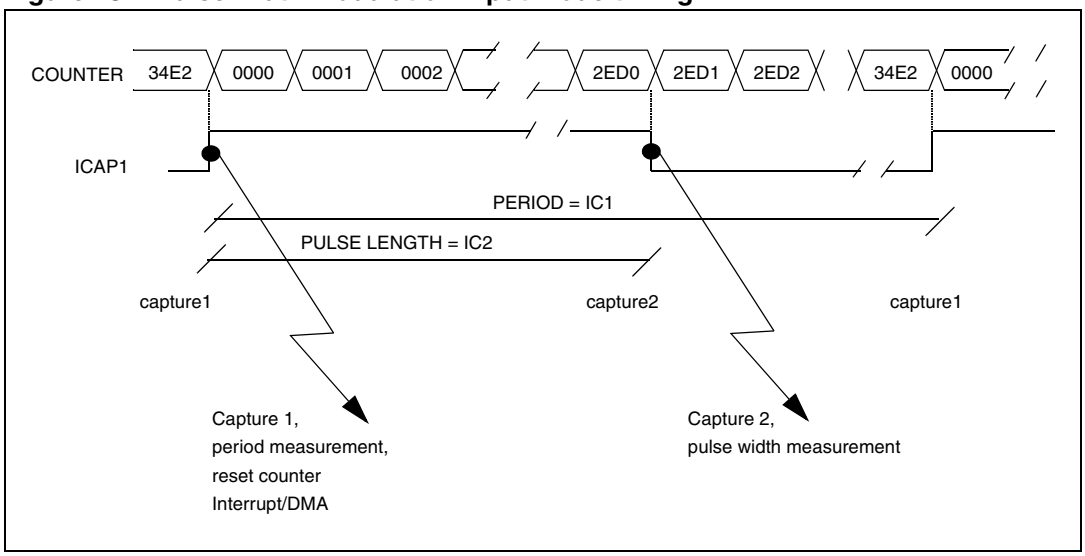
f_{PCLK} = Internal clock frequency

t_{PRESC} = Timer clock prescaler

The Input Capture 1 event causes the counter to be initialized to 0000h, allowing a new measurement to start.

The first Input Capture on IC1 does not generate the corresponding interrupt/DMA request.

Figure 48. Pulse width modulation input mode timing



7.4 Interrupt management

The five interrupt sources (IC1, OC1, IC2, OC2 and Timer Overflow) are mapped on the same input to the VIC (Vectored Interrupt Controller).

To enable the interrupt request, set the OC*i*IE and/or IC*i*IE and/or TOIE bits in the TIM_CR2 register and configure the corresponding VIC registers.

7.5 DMA

A DMA interface is available on TIM0 and TIM1; the source can be selected to be IC1, OC1, IC2, OC2.

To use the DMA feature:

- Select the DMA source by programming the DMAS[1:0] bits in the TIM_CR1 register
- Set the DMAIE bit in the TIM_CR2 register

This configuration allows the timer module to perform DMA requests.

7.6 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
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

7.6.1 Input capture register 1 (TIM_IC1R)

Address offset: 00h

Reset value: undefined (xxh)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IC1R[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16	Reserved, always read as 0
Bits 15:0	**IC1R[15:0]** *IC 1 Captured value* These bits contain the counter value transferred by the Input Capture 1 event.

7.6.2 Input capture register 2 (TIM_IC2R)

Address offset: 04h

Reset value: undefined

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IC2R[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16	Reserved, always read as 0
Bits 15:0	**IC2R[15:0]** *IC 2 Captured value* These bits contain the counter value transferred by the Input Capture 2 event.

7.6.3 Output compare register 1 (TIM_OC1R)

Address offset: 08h

Reset value: 0000 8000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC1R[15:0]															
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 15:0	**O1R[15:0]** *OC 1 Compare value* These bits are written by software, they contain the value to be compared to the counter.

7.6.4 Output compare register 2 (TIM_OC2R)

Address offset: 0Ch

Reset value: 0000 8000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2R[15:0]															
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 15:0	**OC2R[15:0]** *OC 2 Compare value* These bits are written by software, they contain the value to be compared to the counter.

7.6.5 Counter register (TIM_CNTR)

Address offset: 10h

Reset value: 0000 FFFCh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OCR2[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16	Reserved, always read as 0
Bits 15:0	**OCR2[15:0]** *Counter value* These bits contain the value of the free-running counter.

7.6.6 Control register 1 (TIM_CR1)

Address offset: 14h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	PWMI	DMAS[1:0]		FOLV2	FOLV1	OLVL2	OLVL1	OC2E	OC1E	OPM	PWM	IEDG2	IEDG1	EXEDG	ECKEN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, always read as 0
Bit 15	**EN** *Counter Enable* This bit is set and cleared by software 0: Counter disabled (stopped) 1: Counter enabled
Bit 14	**PWMI** *PWM Input Mode Enable* This bit is set and cleared by software. 0: PWM input mode disabled 1: PWM input mode enabled
Bit 13:12	**DMAS[1:0]** *DMA source select* These bits are set and cleared by software. 00: IC1 used as DMA source 01: OC1 used as DMA source 10: IC2 used as DMA source 11: OC2 used as DMA source
Bit 11	**FOLV2** *Forced Output Compare 2* This bit is set by software. 0: No effect 1: Forces the level of the OLVL2 bit to be copied to the OCMP2 pin
Bit 10	**FOLV1** *Forced Output Compare 1* This bit is set by software 0: No effect 1: Forces the level of the OLVL1 bit to be copied to the OCMP1 pin
Bit 9	**OLVL2** *Output Level 2* This bit is set and cleared by software. It is copied to the OCMP2 pin whenever a successful comparison occurs with the OC2R register and the OC2E bit is set.
Bit 8	**OLVL1** *Output Level 1* This bit is set and cleared by software. It is copied to the OCMP1 pin whenever a successful comparison occurs with the OC1R register and the OC1E bit is set.
Bit 7	**OC2E** *Output Compare 2 Enable* This bit is set and cleared by software. 0: Output Compare 2 function is enabled, but the output to the OCMP2 pin is disabled. 1: Output Compare 2 function is enabled, and output to the OCMP2 pin enabled. ***Note*:** The corresponding GPIO Alternate Function must be configured in the SCU_GPIOOUT register.

Bit 6	**OC1E** *Output Compare 1 Enable* This bit is set and cleared by software. 0: Output Compare 1 function is enabled, but the output to the OCMP1 pin is disabled. 1: Output Compare 1 function is enabled, and output to the OCMP1 pin enabled. ***Note*:** The corresponding GPIO Alternate Function must be configured in the *GPIO output register (SCU_GPIOOUTn) on page 110*.
Bit 5	**OPM** *One Pulse Mode* This bit is set and cleared by software. 0: One Pulse Mode is not active 1: One Pulse Mode is active, the ICAP1 pin can be used to trigger one pulse on the OCMP1 pin. The active transition is given by the IEDG1 bit. The length of the generated pulse depends on the contents of the TIM_OC1R register.
Bit 4	**PWM** *Pulse Width Modulation Mode* This bit is set and cleared by software. 0: PWM mode is not active 1: PWM mode is active, the OCMP1 pin outputs a programmable cyclic signal; the length of the pulse depends on the value of the TIM_OC1R register. The period depends on the value of the TIM_OC2R register.
Bit 3	**IEDG2** *Input Edge 2* This bit determines which type of level transition on the ICAP2 pin will trigger the capture. 0: A falling edge triggers the capture 1: A rising edge triggers the capture
Bit 2	**IEDG1** *Input Edge 1* This bit determines which type of level transition on the ICAP1 pin will trigger the capture. 0: A falling edge triggers the capture 1: A rising edge triggers the capture
Bit 1	**EXEDG** *External Clock Edge* This bit determines which type of level transition on the external clock pin EXTCLK will trigger the counter. 0: A falling edge triggers the counter 1: A rising edge triggers the counter
Bit 0	**ECKEN** *External Clock Enable* 0: Internal clock (PCLK), divided by CC[7:0] prescaler, is used to feed timer clock 1: External source (EXTCLK clock on GPIO pins) is used for timer clock ***Note*:** The External clock source is enabled using the TIM01SEL and TIM23SEL bits in the *Clock control register (SCU_CLKCNTR) on page 86*.

7.6.7 Control register 2 (TIM_CR2)

Address offset: 18h

Reset value: 0000 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IC1IE	OC1IE	TOIE	IC2IE	OC2IE	DMAE	Reserved		CC[7:0]							
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, always read as 0
Bit 15	**IC1IE** *Input Capture 1 interrupt enable* This bit is set and cleared by software. 0: IC1 interrupt disabled 1: Generate interrupt request to VIC if ICF1 flag is set
Bit 14	**OC1IE** *Output Compare 1 interrupt enable* This bit is set and cleared by software. 0: OC1 interrupt disabled 1: Generate interrupt request to VIC if OCF1 flag is set
Bit 13	**TOIE** *Timer Overflow interrupt enable* This bit is set and cleared by software. 0: TO interrupt disabled 1: Generate interrupt request to VIC if TOF flag is set
Bit 12	**IC2IE** *Input Capture 2 interrupt enable* This bit is set and cleared by software. 0: IC2 interrupt disabled 1: Generate interrupt request to VIC if ICF2 flag is set
Bit 11	**OC2IE** *Output Compare 2 interrupt enable* This bit is set and cleared by software. 0: OC2 interrupt disabled 1: Generate interrupt request to VIC if OCF2 flag is set
Bit 10	**DMAE** *DMA enable* This bit is set and cleared by software. 0: DMA disabled 1: DMA is enabled ***Note*:** DMA is available on TIM0 and TIM1 only.
Bits 9:8	Reserved, always read as 0
Bits 7:0	**CC[7:0]**: *Clock Control* These bit are written by software to select the frequency of the timer clock applied when internal clock is selected (ECKEN = 0): 00h: f_{PCLK} / 1 01h: f_{PCLK} / 2 ... FFh: f_{PCLK} / 256

7.6.8 Status register (TIM_SR)

Address offset: 18h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICF1	OCF1	TOF	ICF2	OCF2	Reserved										
rc_w0	rc_w0	rc_w0	rc_w0	rc_w0											

Bits 31:16	Reserved, always read as 0
Bit 15	**ICF1** *Input Capture Flag 1* This bit is read and clear only. 0: No input capture (reset value) 1: An input capture has occurred. An interrupt request is generated if IC1IE=1 in the TIM_CR2 register.
Bit 14	**OCF1** *Output Compare Flag 1* 0: No match (reset value) 1: The content of the counter matches the content of the TIM_OC1R register. An interrupt request is generated if OC1IE=1 in the TIM_CR2 register.
Bit 13	**TOF** *Timer Overflow Flag* 0: No timer overflow (reset value) 1: The counter has rolled over from FFFFh to 0000h. An interrupt request is generated if TCIE = 1 in the TIM_CR2 register.
Bit 12	**ICF2** *Input Capture Flag 2* 0: No input capture (reset value) 1: An input capture has occurred on the ICAP2 pin. An interrupt request is generated if IC2IE = 1 in the TIM_CR2 register.
Bit 11	**OCF2** *Output Compare Flag 2* 0: No match (reset value). 1: The content of the free running counter matches the content of the TIM_OC2R register. An interrupt request is generated if OC2IE=1 in the TIM_CR2 register.
Bit 10:0	Reserved, forced by hardware to 0

7.7 TIM register map

Table 18. TIM register map

Address offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	TIM_IC1R	Input Capture register 1															
04h	TIM_IC2R	Input Capture register															
08h	TIM_OC1R	Output Compare register 1															
0Ch	TIM_OC2R	Output Compare register 2															
10h	TIM_CNTR	Counter register															
14h	TIM_CR1	Control register 1															
18h	TIM_CR2	Control register 12															
1Ch	TIM_SR	Status Register					Reserved										

Refer to *Table 5 on page 35* for the register base addresses.

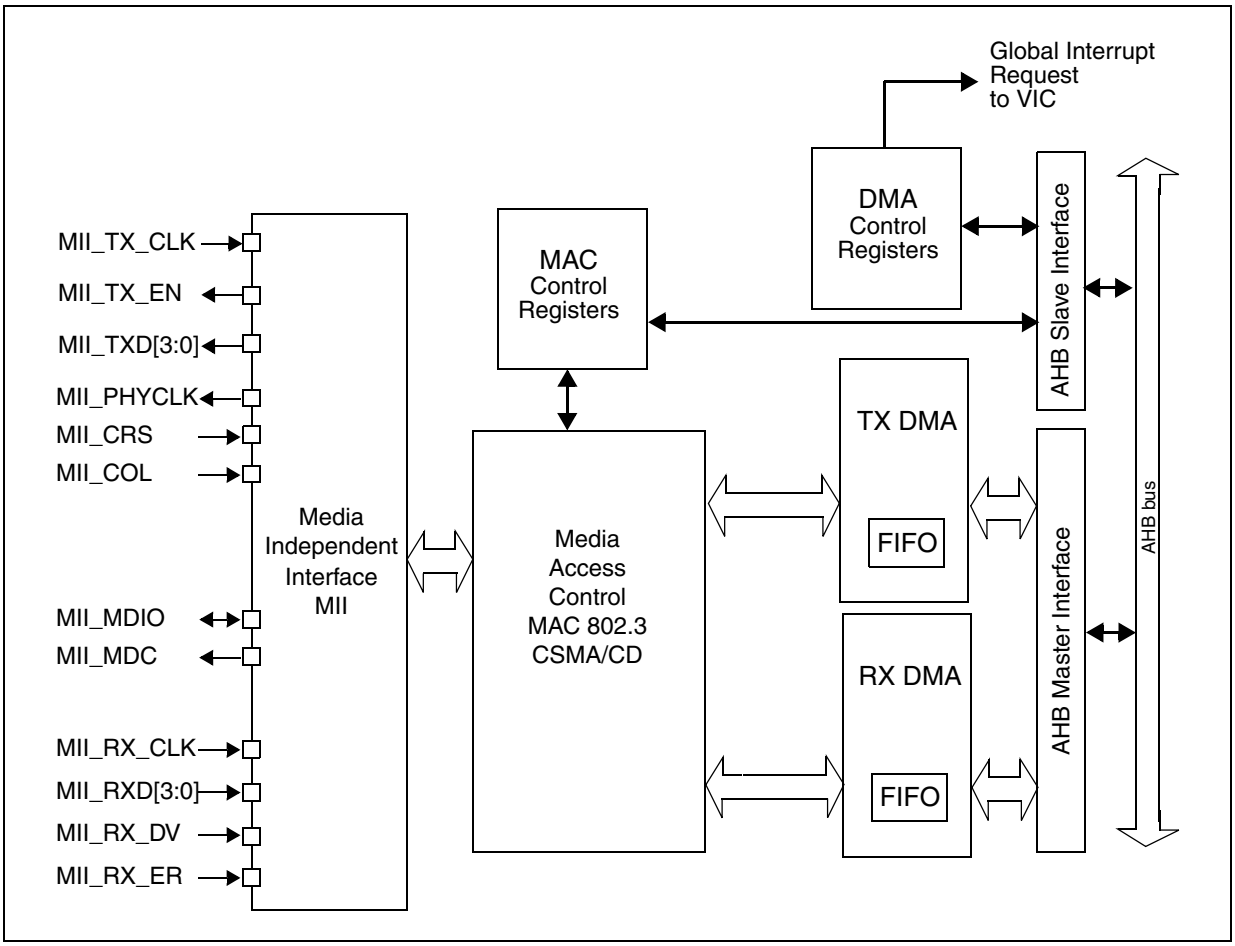
8 MAC/DMA controller with DMA (ENET)

The IEEE 802.3 International Standard for Local Area Network (LANs) employs the CSMA/CD (Carrier Sense Multiple Access with collision detection) as the access method.

The ENET peripheral consists of a MAC 802.3 (media access control) controller with media independent interface (MII) and a dedicated DMA controller.

The MAC block implements the LAN CSMA/CD sublayer for the following families of systems: 10 Mb/s and 100 Mb/s of data rates for baseband and broadband systems. Half and full-duplex operation modes are supported. The collision detection access method is applied only to the half-duplex operation mode. The MAC control frame sublayer is supported.

Figure 49. MAC/DMA block diagram



8.1 Functional description

8.1.1 MAC 802.3

The MAC sublayer performs the following functions associated with a data link control procedure:

- Data encapsulation (transmit and receive)
  - Framing (frame boundary delimitation, frame synchronization)
  - Addressing (handle of source and destination addresses)
  - Error detection
- Media access management
  - Medium allocation (collision avoidance)
  - Contention resolution (collision handling)

Basically there are two operating modes of the MAC sublayer:

- Half duplex mode: the stations contend for the use of the physical medium, using the CSMA/CD algorithms.
- Full duplex mode: simultaneous transmission and reception without contention resolution (CSMA/CD algorithm are unnecessary) when all the following conditions are matched:
  - physical medium capability to support simultaneous transmission and reception
  - exactly 2 stations connected to the LAN
  - both stations configured for full duplex operation.

8.1.2 MII

MII TX/RX interface

The MII TX/RX interface defines the interconnection between MAC sublayer and PHY for data transfer at 10 Mb/s and 100 Mb/s. These signals are implemented as alternate function I/Os on external pins of the microcontroller:

- MII_TX_CLK: Continuous clock that provides the timing reference for the TX data transfer. The nominal frequency is: 2.5 MHz at 10 Mb/s speed; 25 MHz at 100 Mb/s speed.
- MII_RX_CLK: Continuous clock that provides the timing reference for the RX data transfer. The nominal frequency is: 2.5 MHz at 10 Mb/s speed; 25 MHz at 100 Mb/s speed.
- MII_TX_EN: Transmission enable indicates that the MAC is presenting nibbles on the MII for transmission. It must be asserted synchronously (MII_TX_CLK) with the first nibble of the preamble and must remain asserted while all nibbles to be transmitted are presented to the MII.
- MII_TXD[3:0]: Transmit data is a bundle of 4 data signals driven synchronously by the MAC sublayer and qualified (valid data) on the assertion of the MII_TX_EN signal. MII_TXD[0] is the least significant bit, MII_TXD[3] is the most significant bit. While MII_TX_EN is deasserted the transmit data must have no effect upon the PHY.
- MII_CRS: Carrier sense is asserted by the PHY when either the transmit or receive medium is non idle. It shall be deasserted by the PHY when both the transmit and receive media are idle. The PHY must ensure that the MII_CS signal remains asserted

throughout the duration of a collision condition. This signal is not required to transition synchronously with respect to the TX and RX clocks. In full duplex mode the state of this signal is don't care for the MAC sublayer.

- MII_COL: Collision detection must be asserted by the PHY upon detection of a collision on the medium and must remain asserted while the collision condition persists. This signal is not required to transition synchronously with respect to the TX and RX clocks. In full duplex mode the state of this signal is don't care for the MAC sublayer.
- MII_RXD[3:0]: Reception data is a bundle of 4 data signals driven synchronously by the PHY and qualified (valid data) on the assertion of the MII_RX_DV signal. MII_RXD[0] is the least significant bit, MII_RXD[3] is the most significant bit. While MII_RX_EN is deasserted and MII_RX_ER is asserted, a specific MII_RXD[3:0] value is used to transfer specific information from the PHY (see *Table 20*).
- MII_RX_DV: Receive data valid indicates that the PHY is presenting recovered and decoded nibbles on the MII for reception. It must be asserted synchronously (MII_RX_CLK) with the first recovered nibble of the frame and must remain asserted through the final recovered nibble. It must be deasserted prior to the first clock cycle that follows the final nibble. In order to receive the frame correctly, the MII_RX_DV signal must encompass the frame, starting no later than the SFD field.
- MII_RX_ER: Receive error must be asserted for one or more clock periods (MII_RX_CLK) to indicate to the MAC sublayer that an error was detected somewhere in the frame. This error condition must be qualified by MII_RX_DV assertion as described in *Table 20*.

Table 19. TX interface signals encoding

MII_TX_EN	MII_TXD[3:0]	Description
0	0000 through 1111	Normal inter-frame
1	0000 through 1111	Normal data transmission

Table 20. RX interface signals encoding

MII_RX_DV	MII_RX_ERR	MII_RXD[3:0]	Description
0	0	0000 through 1111	Normal inter-frame
0	1	0000	Normal inter-frame
0	1	0001 through 1101	Reserved
0	1	1110	False carrier indication
0	1	1111	Reserved
1	0	0000 through 1111	Normal data reception
1	1	0000 through 1111	Data reception with errors

Figure 50. Transmission with no collision

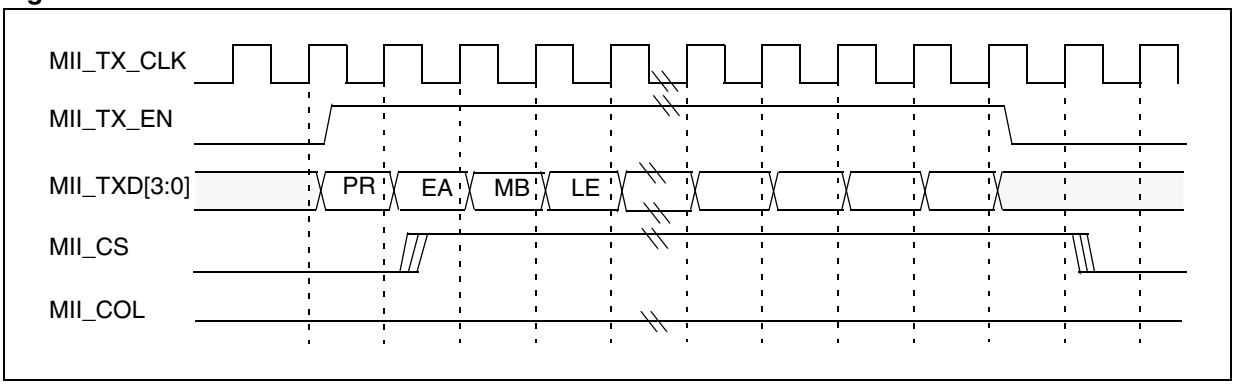


Figure 51. Transmission with collision

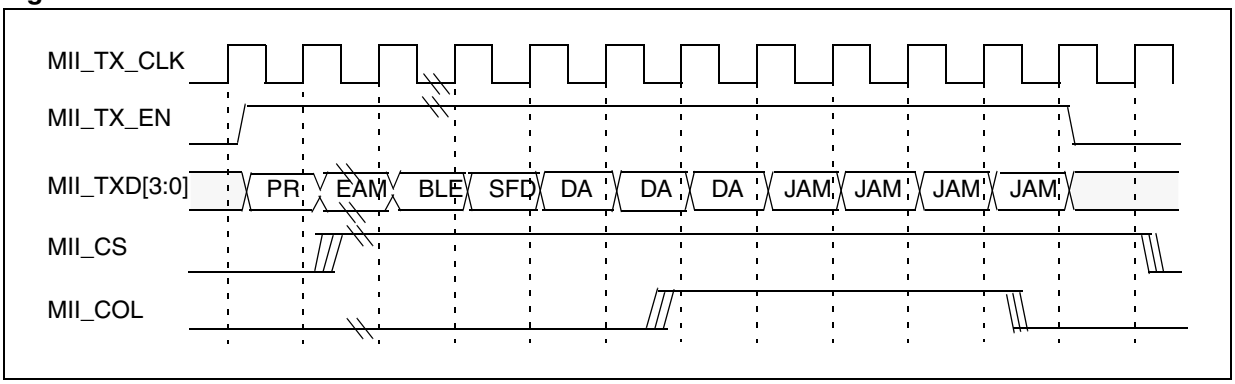


Figure 52. Reception with no errors

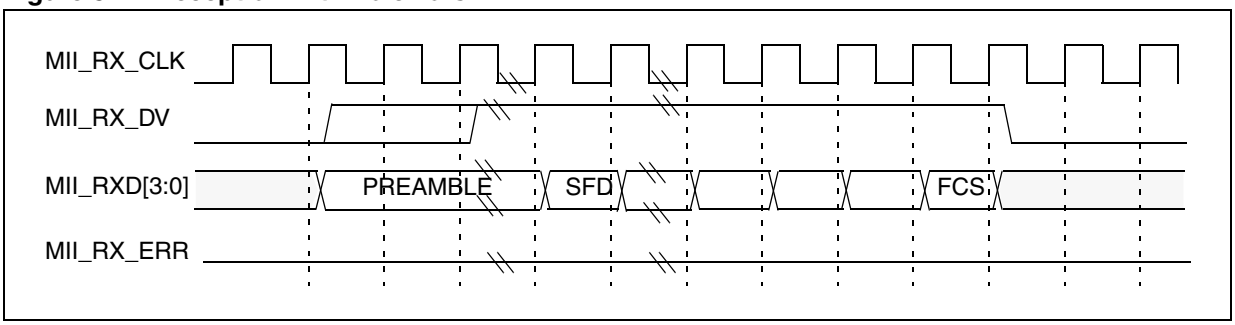


Figure 53. Reception with errors

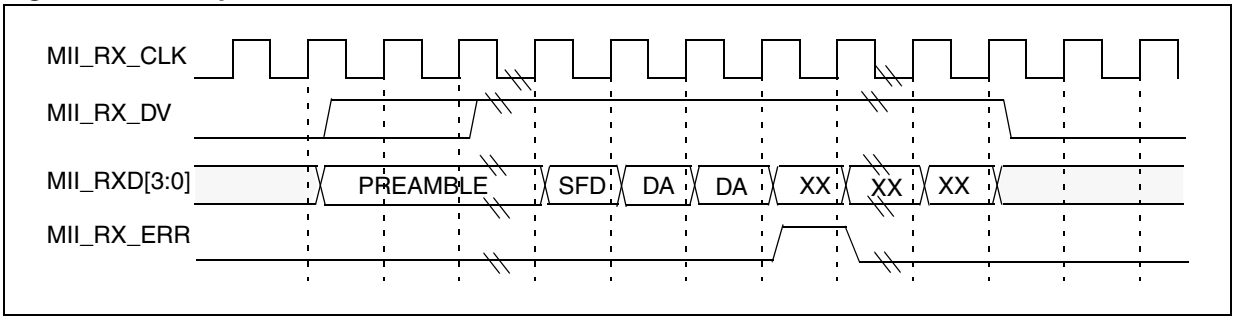


Figure 54. Reception with false carrier indication

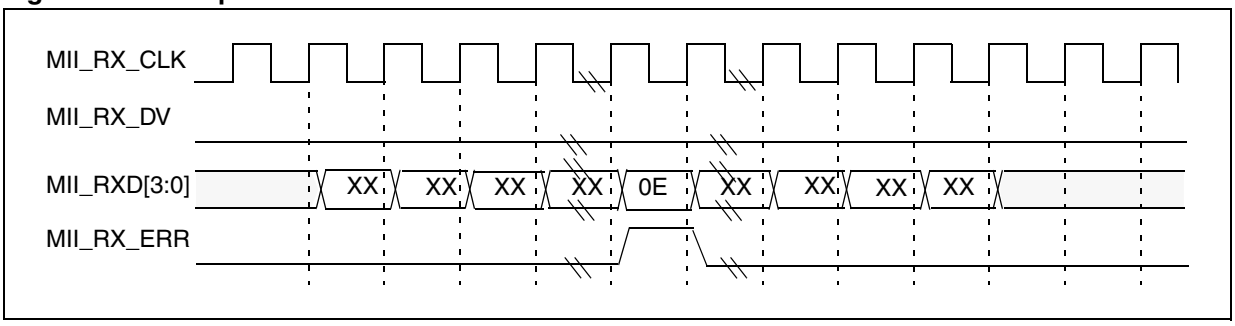


Figure 55. MII TX interface: output timing requirements

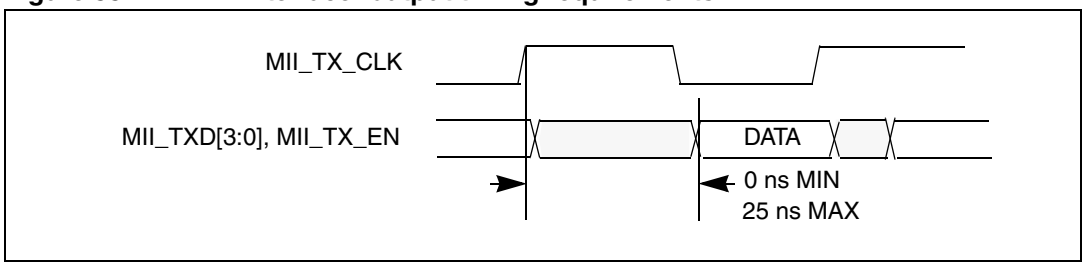
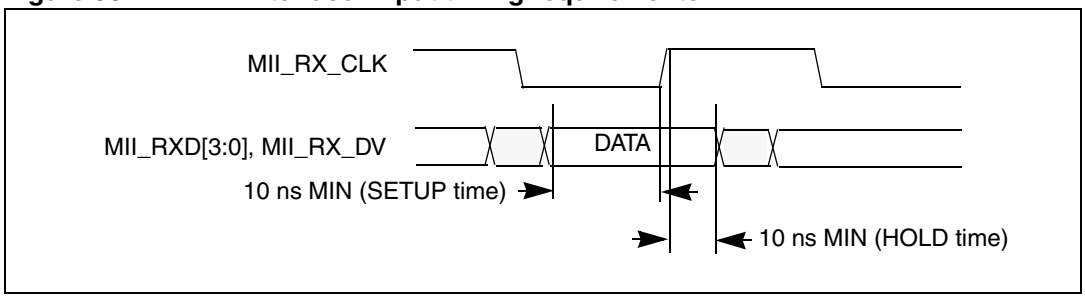


Figure 56. MII RX interface: input timing requirements



MII management interface

The MII management interface defines the interconnection and the protocol used to configure the internal registers of the PHY device. The MII_MDC signal is implemented as an alternate function I/O of the microcontroller. The MII_MDIO signal is a dedicated pin.

- MII_MDC: aperiodic clock that provides the timing reference for the data transfer at the maximum frequency of 2.5 Mhz. The minimum high and low times for MII_MDC must be 160 ns each, and the minimum period for MII_MDC must be 400 ns. In idle state the MIM management interface must drive the MII_MDC clock signal low.
- MII_MDIO: data input/output bit stream to transfer status information to/from the PHY device synchronously to the MII_MDC clock signal

The frame structure related to a read or write operation is shown in *Table 21*, the order of bit transmission must be from left to right.

Table 21. Management frame format

	Management frame fields							
	Preamble (32 bits)	Start	Operation	PADDR	RADDR	TA	Data (16 bits)	Idle
Read	1... 1	01	10	ppppp	rrrrr	Z0	dddddddddddddddd	Z
Write	1... 1	01	01	ppppp	rrrrr	10	dddddddddddddddd	Z

The management frame consists of eight fields:

- IDLE: the MDIO line is driven in high-impedance state. All three-state drivers must be disabled and the PHY's pull-up resistor keeps the line to logic one.
- PREAMBLE: each transaction (read or write) can be initiated with the preamble field that corresponds to 32 contiguous logic one bits on the MDIO line with 32 corresponding cycles on MDC. This field is used to establish synchronization with the PHY device and its generation is optional (depending on the PHY features) depending on the PR bit in the ENET_MIIA register.
- START: the start of frame is defined by a <0> pattern to verify transitions on the line from the default logic one state to zero and back to one.
- OPERATION: defines the type of transaction (read or write) in progress.
- PADDR: the PHY address is 5 bits, allowing 32 unique PHY addresses. The MSB bit of the address is the first transmitted and received.
- RADDR: the register address is 5 bits, allowing 32 individual registers to be addressed within the selected PHY device. The MSB bit of the address is the first transmitted and received.
- TA: the turn-around field defines a 2-bit pattern between the RADDR and DATA fields to avoid contention during a read transaction. For a read transaction the MAC controller drives high-impedance on the MDIO line for the 2 bits of TA. The PHY device must drive a high-impedance state on the first bit of TA, a zero bit on the second one.
  For a write transaction, the MAC controller drives a <10> pattern during the TA field. The PHY device must drive a high-impedance state for the 2 bits of TA.
- DATA: the data field is 16-bit. The first bit transmitted and received must be bit 15 of the ENET_MIID register.

Figure 57. MII management interface: input timing requirements (PHY device)

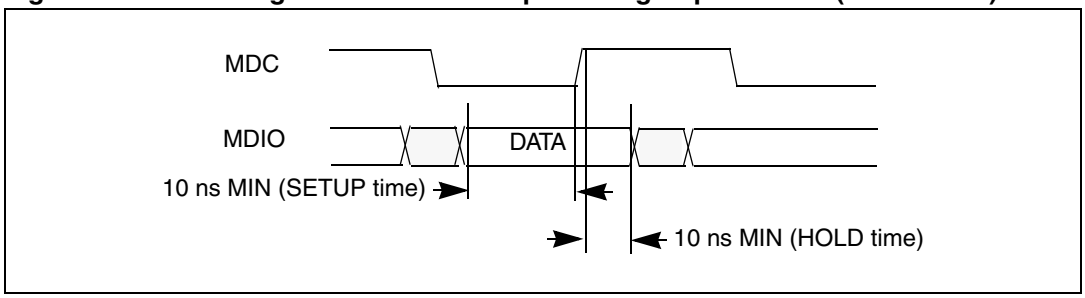
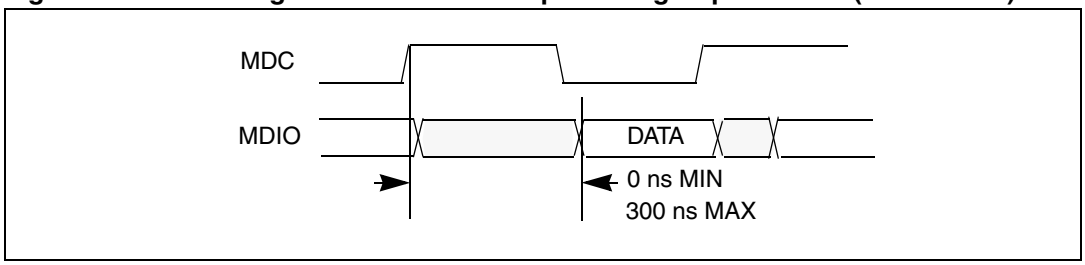


Figure 58. MII management interface: output timing requirements (PHY device)



8.1.3 DMA

The TX DMA and RX DMA blocks use the information written in the TX and RX configuration registers, to move data from the FIFOs to a specified memory area (master DMA).

When TX DMA or RX DMA is enabled via the configuration registers, they are able to manage the data transfer without further processor intervention.

The DMA transfer can be:

- DMA continuous/fixed size: the DMA can be required to run indefinitely or to stop after a configured number of data bytes has been transferred
- Fixed/incrementing address: the DMA address can be fixed (i.e. all the data are transferred to the same AHB word-aligned address) or it can be updated after each data transfer
- Linear incrementing or wrapping address: when the address is defined as incrementing, it can be required that, once reached a programmed value, the address counter wraps back to the initial address value (the address location, pointed by the wrapping address, is not modified)
- With FIFO entry threshold: the DMA starts transferring data to/from the AHB bus when a programmable number of 32-bit RX FIFO entries is valid

When the DMA is enabled, as soon as data appears in the RX FIFO (or one free entry appears in the TX FIFO), the DMA may either initiate an AHB transfer immediately, or be delayed until X data bytes are available in the FIFO (FIFO entry threshold).

The DMA can be configured to wrap-round the AHB address at some point to implement a circular buffer in CPU memory.

The DMA can be configured to run indefinitely or to stop after DMA_XFERCOUNT data have been transferred. The maximum DMA transfer count is 4 Kbytes.

When the DMA completes, it can either generate an interrupt request to the processor and wait for new instruction, or fetch a new DMA descriptor.

If an AHB error condition occurs, while the DMA is running, the DMA activity is suspended, until the error interrupt bit (MERR_INT) is reset. When the error condition is removed the DMA makes the same request previously interrupted by the error response.

RX/TX FIFOs

The FIFOs are readable (write has no effect) as a sequence of 32-bit registers, mapped at adjacent addresses.

For the FIFO address mapping refer to *Table 8.5*.

8.2 MAC 802.3 operation

8.2.1 MAC 802.3 frame format

The MAC block implements the MAC sublayer and the optional MAC control sublayer (10/100 Mb/s) as specified by the IEEE 802.3-2002 standard.

Two frame formats are specified for data communication systems using the CSMA/CD MAC:

- Basic MAC frame format
- Tagged MAC frame format (extension of the basic MAC frame format)

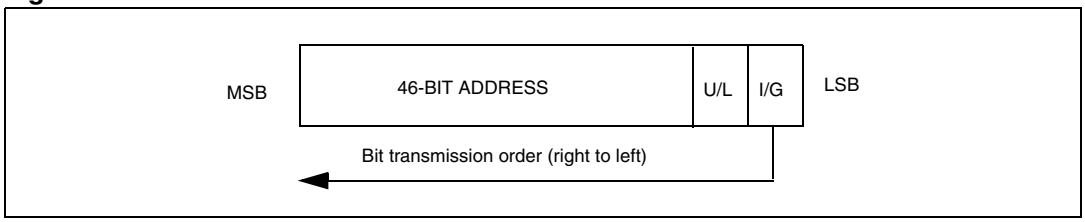
Figure 60 and *Figure 61* describe the frame structure (untagged and tagged) that include the following fields:

- Preamble: 7-byte field used for synchronization purpose (PLS circuitry).
  hexadecimal value: 55-55-55-55-55-55-55
  bit pattern: 01010101 01010101 01010101 01010101 01010101 01010101 01010101
  (right to left bit transmission)
- Start frame delimiter (SFD): 1-byte field used to indicate the start of a frame.
  hexadecimal value: D5
  bit pattern: 11010101 (right to left bit transmission)
- Destination and Source Address fields: 6-byte fields to indicate the destination and source station addresses as follows (see *Figure 60*):
  - Each address is 48 bits in length
  - The first LSB bit (I/G) in the destination address field is used to indicate an individual (I/G = 0) or a group address (I/G = 1). A group address could identify none, one or more, or all the stations connected to the LAN. In the source address the first bit is reserved and set to 0.
  - The second bit (U/L) distinguishes between locally (U/L = 1) or globally (U/L = 0) administered addresses. For broadcast addresses this bit is also 1.
  - Each byte of each address field must be transmitted least significant bit first.

The address designation is based on the following types:

- Individual address: this is the physical address associated with a particular station on the network.
- Group address. A multi destination address associated with one or more stations on a given network. There are 2 kinds of multicast address:
  - Multicast-Group address. An address associated with a group of logically related stations.
  - Broadcast address. A distinguished, predefined multicast address (all 1's in the destination address field) that always denotes all the stations on a given LAN.

Figure 59. Address field format



1. Legend: I/G = 0 individual address; I/G = 1 group address; U/L = 0 globally administered address; U/L = 1 locally administered address.

- QTag Prefix: 4-byte field inserted between the Source Address field and the MAC Client Length/Type field. This field is an extension of the basic frame (untagged) to obtain the tagged MAC frame. The untagged MAC frames don't include this field. The extension for tagging are as follows:
  - 2-byte constant Length/Type field value consistent with the Type interpretation (greater than 06-00 hexadecimal) equal to the value of the 802.1Q Tag Protocol Type (81-00 hexadecimal). This constant field allows to distinguish tagged and untagged MAC frames.
  - 2-byte field containing Tag control information field subdivided as follows: a 3-bit user priority, a canonical format indicator (CFI) bit and a 12-bit VLAN Identifier.

The length of the tagged MAC frame is extended by 4 bytes by the QTag Prefix.

- MAC Client Length/Type: 2-byte field with different meaning (mutually exclusive), depending on its value:
  - If the value is less than or equal to maxValidFrame (1500 decimal) than this field indicates the number of MAC client data bytes contained in the subsequent data field of the 802.3 frame (Length interpretation).
  - If the value is greater than or equal to MinTypeValue (1536 decimal, 06-00 hexadecimal) than this field indicates the nature of the MAC client protocol (Type interpretation) related to the ethernet frame.

Regardless of the interpretation of the Length/Type field, if the length of the data field is less than the minimum required for proper operation of the protocol, a PAD field is added after the data field but prior to the FCS field. The Length/Type field is transmitted and received with the high order byte first.
For Length/Type field values in the range between maxValidLength and minTypeValue (boundaries excluded), the behavior of the MAC sublayer is not specified: they may or may not be passed by the MAC sublayer.

- Data and PAD fields: n-byte data field. Full data transparency is provided, it means that any arbitrary sequence of byte values may appear in the data field. The size of the PAD, if any, is determined by the size of the data field. Max and min length of the data and PAD field are:
  - Maximum length = 1500 bytes
  - Minimum length for untagged MAC frames = 46 bytes
  - Minimum length for tagged MAC frames = 42 bytes

When the data field length is less than the minimum required, the PAD field is added to match the minimum length (42 bytes for tagged frames, 46 bytes for untagged frames).

- Frame Check Sequence: 4-byte field that contains the cyclic redundancy check (CRC) value. The CRC computation is based on the following fields: source address, destination address, QTag prefix, length/type, LLC data and pad (that is, all fields except the preamble, SFD). The generating polynomial is the following:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The CRC value of a frame is computed as follows:

- The first two bits of the frame are complemented
- The n-bits of the frame are the coefficients of a polynomial M(x) of degree (n-1). The first bit of the destination address corresponds to the x**(n-1) term and the last bit of the data field corresponds to the x**0 term.
- M(x) is multiplied by x**32 and divided by G(x), producing a remainder R(x) of degree <=31.
- The coefficients of R(x) are considered to be 32-bit sequence
- The bit-sequence is complemented and the result is the CRC

The 32-bits of the CRC value are placed in the frame check sequence. The x**31 term is the first transmitted, the term x**0 term is the last one.

Figure 60. MAC frame format

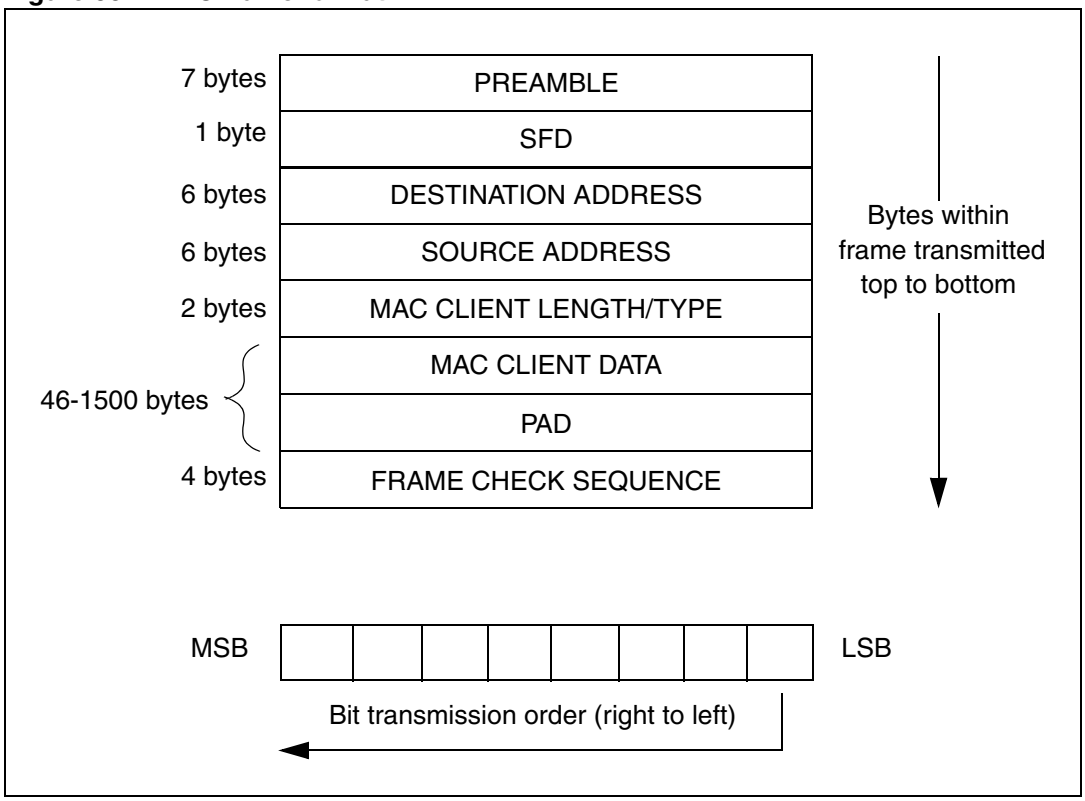
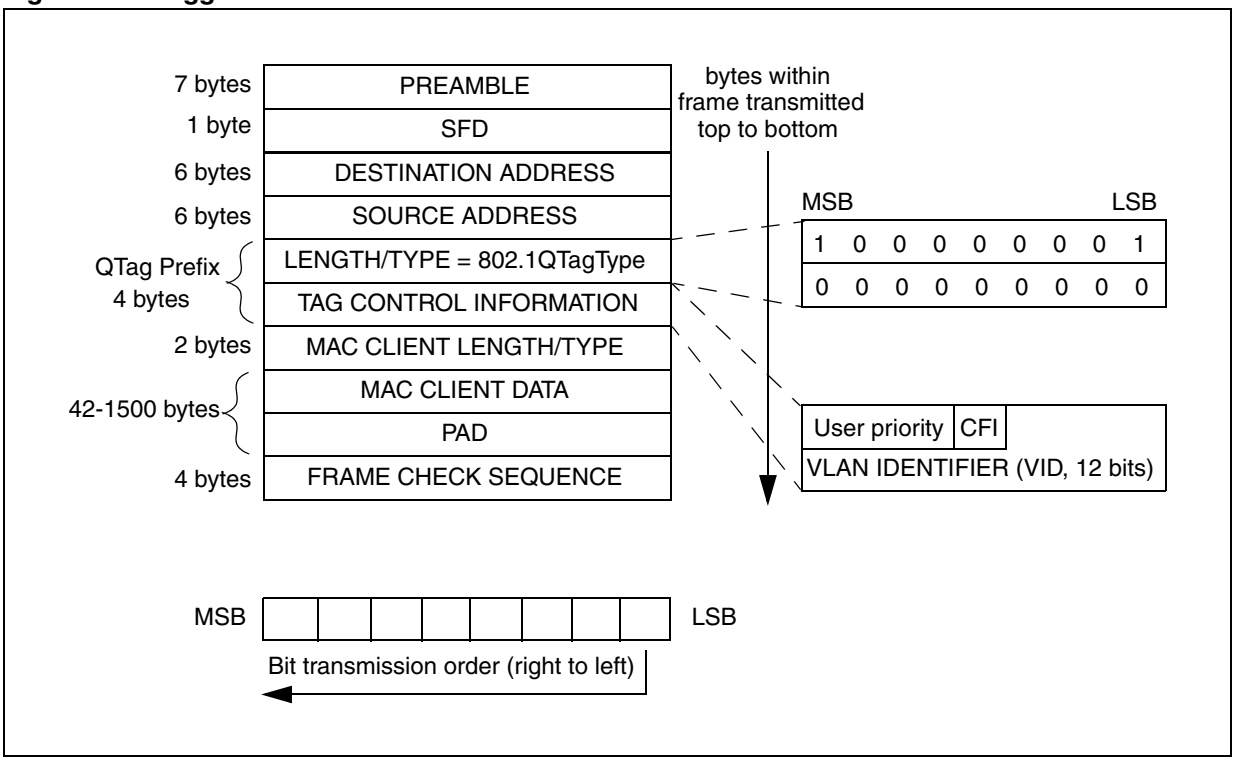


Figure 61. Tagged MAC frame format



Each byte of the MAC frame, except the FCS field, is transmitted low-order bit first.

An invalid MAC frame is defined by one of the following conditions:

- The frame length is inconsistent with the expected value as specified by the Length/Type field. If the Length/Type field contains a type value, then the frame length is assumed to be consistent with this field (no invalid frame)
- The frame length is not an integer number of bytes (extra bits).
- The CRC value computed on the incoming frame doesn't match with the included FCS.

8.2.2 MAC frame reception

When the MAC-802.3 receives a frame, it starts transferring data to the RX DMA block.

If the DMA has been properly enabled and a valid descriptor fetched, the data is transferred to the DMA RX FIFO. The DMA will then move this data to the main memory, as detailed in the DMA descriptor.

After receiving the last data frame, the MAC-802.3 checks the CRC, reports the end of frame and sends the DMA a 32 bit word named 'RX Packet Status'.

As soon as the RX DMA has successfully transferred to main memory all the frame data, has updated the Packet status and has reset the VALID bit, the DMA is considered to be completed and the descriptor fetch logic is invoked.

Prior to checking if a new descriptor has to be loaded, the 32-bit RX Frame Status, received by the MAC-802.3 is copied by the DMA to the following address in main memory:

CURRENT_DESCRIPTOR_START_ADDRESS + 'C' (hex)

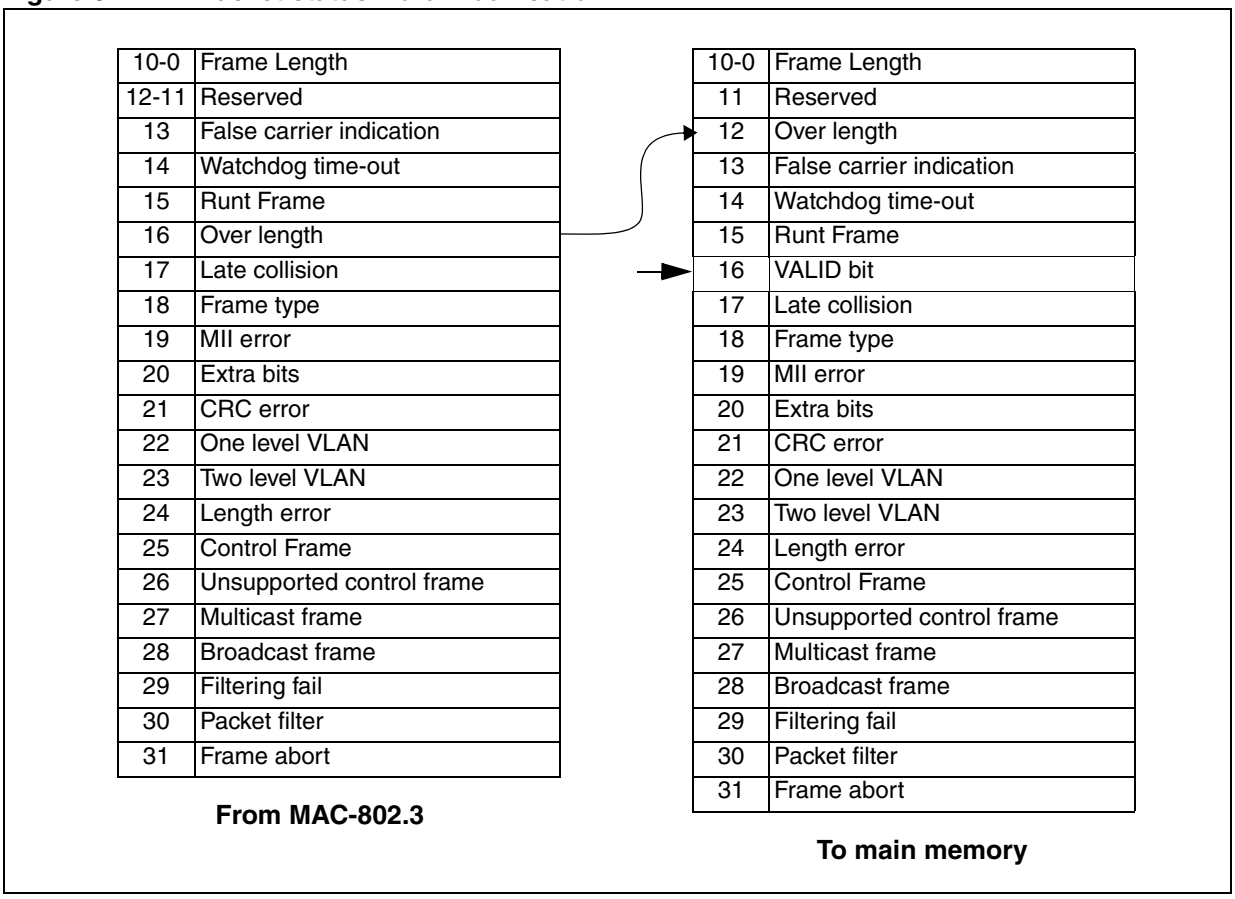
To increase the performance, it replaces the 16th bit of the Packet Status with the VALID bit. See *Figure 62.* The original value of this bit (Late Collision flag) is moved to replace the 1-st bit (it's an unused bit). In this way the DMA can reset the descriptor VALID bit and, by the same write operation, save the status of the received packet.

The VALID bit set to '0' indicating that the descriptor is no longer owned by the DMA and that the DMA operations are completed.

If enabled, an interrupt will then be sent to the CPU to notify the transfer completion.

The DMA descriptor fetch logic is then able to start processing the next DMA sequence, and a new descriptor fetch, starting at the NEXT descriptor memory address, is performed.

Figure 62. RX Packet status word modification



8.2.3 Frame reception errors

RX address filtering failed

When a new frame arrives to the MAC-802.3 logic, it transfers all the incoming data to the DMA, regardless of the frame attributes.

If the destination address doesn't match the address filtering policy programmed in the MAC-802.3 registers, the data are sent anyway to the DMA, but an 'Address filtering failed' flag is set.

The ADDRESS_FILTER bit, in the ENET_RXSTR register, forces the DMA logic to discharge the incoming frame if the address filtering failed: in this case, no data is transferred to memory and the fetched descriptors will be used for the next incoming frame.

RX packet too long

While the DMA RX logic is receiving data frame from the MAC-802.3 core, it checks that the received byte number does not exceed the DMA_XFERCOUNT value in the ENET_RXCR register.

If this condition occurs, no more data are accepted by the DMA, the already loaded data are written to the system memory and, when the frame completes, the RX packet status information will report the information that the frame was aborted by the receiving logic.

RX packet with MAC-802.3 reported error

Even when the incoming frame is received and downloaded to main memory by the DMA, without any problem, the RX packet status, sent by the MAC-802.3 when the transfer completes, can report some error conditions.

The DMA will check the COLLISION_SEEN and RUNT_FRAME bit in the of the RX packet status word and, depending on the value of bits 7 and 6 of the ENET_RXSTR register will proceed as follows.

If: Both the COLLISION_SEEN bit of the RX packet status and the bit 7 of the ENET_RXSTR are set to 1.

Or: Both the RUNT_FRAME bit of the RX packet status and the bit 6 of the ENET_RXSTR register are set to 1.

Then: The RX DMA will discard the received frame (even if it was already downloaded to memory) and will use the current DMA descriptor and memory buffer for the next incoming RX frame.

8.2.4 MAC frame transmission

After the DMA has been properly enabled for a TX data transfer, a valid descriptor fetched, and some data have been loaded to the TX FIFO, the DMA starts to move data from the DMA TX FIFO to the MAC-802.3 core.

The MAC-802.3 core will then try to transfer the TX frame to the PHY device, via the MII interface, and then from the PHY to the line.

Depending on the line traffic condition, the packet frame transfer can succeed or not (normal collision, late collision, deferral, excessive deferral and so on).

If the transaction succeeds, after the last data byte transfer, the TX DMA reads the TX packet status word.

The MAC-802.3 logic will hold this request until the frame transmission completes on the line, and then it will provide the TX packet status information to the DMA.

If no error occurred, the DMA is considered to be completed and the descriptor fetch logic is invoked.

Prior to checking if a new descriptor has to be loaded, the 32-bit TX Frame Status, received by the MAC-802.3 is copied by the DMA to the following address in main memory:

CURRENT_DESCRIPTOR_START_ADDRESS + 'C' (hex)

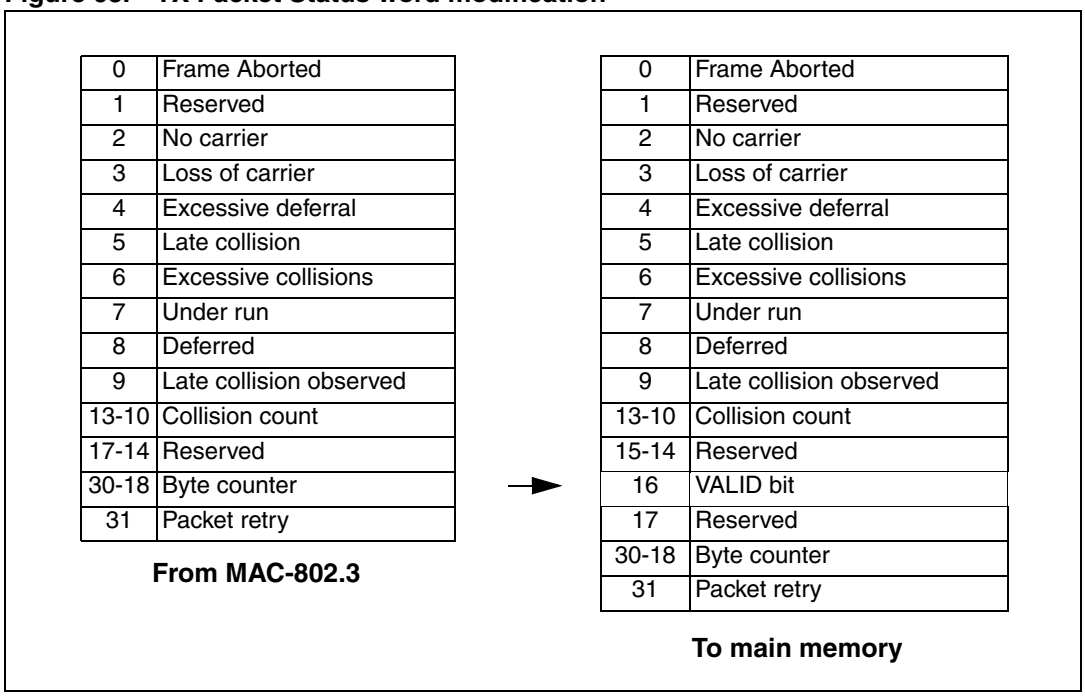
To increase the performance, it replaces the 16th bit of the Packet Status (it's an unused bit) with the VALID bit. See *Figure 63.* In this way the DMA can reset the descriptor VALID bit and, by the same write operation, save the status of the transmitted packet.

The VALID bit set to '0' indicating that the descriptor is no longer owned by the DMA and that the DMA operations are completed.

If enabled, an interrupt will then be sent to the CPU to notify the transfer completion.

The DMA descriptor fetch logic is then able to start processing the next DMA sequence, and a new descriptor fetch, starting at the NEXT descriptor memory address, is performed.

Figure 63. TX Packet Status word modification



8.2.5 Frame transmission errors

TX packet with MAC reported error

When the transmitted packet reports an error condition, the MAC-802.3 qualifies if the error was due to a ‘protocol related’ condition (i.e. collision) or if it was an error to be reported to the CPU attention. This kind of information is summarized in bit 31 (RETRY, see Ref. [1]) of the TX packet status word.

The DMA_MAC wrapper logic will check also the bit UNDER_RUN (bit 7) of the TX packet status word and, depending on the value of the bit 5 of the DMA_START register will proceed as in the following.

If: RETRY=1

Or: Both the UNDER_RUN bit of the TX packet status and bit 5 of the ENET_TXSTR register are set.

Then, the DMA retransmits the same packet to the MAC-802.3 core, without reporting the error condition to the CPU.

In all the other cases, the DMA updates the current descriptor status information in main memory (detailing the error condition), generates an interrupt (if enabled) and starts fetching the DMA descriptors for the next data transfer.

8.2.6 Loopback mode

Loop back mode is available for test purposes. You select it using the LM[1:0] bits in the ENET_MCR register.

When the loop back is active, the TX logic outputs are shorted to the RX logic inputs, and all the data transmitted by the TX are received by the RX.

At the end of each frame, to provide both the RX and TX channels with the expected status words, the loop back logic drives the proper signals on the interfaces and sends a received/transmitted packet status to the two blocks.

To allow you to test different packet status vector values, the loop back logic will use the last 3 (three) words transmitted by the TX DMA to generate the RX and TX status.

In detail:

- the last word of the TX frame is not used because it could be less than 32 valid bits
- the TX packet status bits will have the same value as the word before the last of the TX frames.
- the RX packet status bits will have the same value as the second word before the last of the TX frames.

Note: It is important to pay attention to the status word value, because the behavior of the TX/RX logic depends on these bits (e.g. if the TX status has the RETRY bit set, the TX logic can re-send the same packet forever).

8.3 DMA controller operation

8.3.1 RX DMA configuration

Before starting RX DMA operations, you have to program the following RX DMA configuration registers.

- RX DMA Start Register (ENET_RXSTR)
- RX DMA Control Register (ENET_RXCR)
- RX DMA Start Address Register (ENET_RXSAR)
- RX DMA Next Descriptor Address Register (ENET_RXNDAR)
- RX DMA Time Out Register (ENET_RXTOR)

8.3.2 RX DMA descriptors

The DMA operates by performing a DMA descriptor fetch, which allows it to load all the required information from main memory without any CPU intervention.

You have to allocate a 16-byte region (word-aligned) in main memory, for each DMA transfer. 3 words for the DMA descriptors, plus 1 word for Packet Status and descriptor VALID bit.

- In the first three descriptor words, you write the information to be loaded in the:
  - Control register (ENET_RXCR)
  - Start address register (ENET_RXSAR)
  - Next descriptor address register (ENET_RXNDAR)
- Then set to ‘1’, bit 16 (VALID bit) in the fourth word (Packet Status)
- The packet status word will be then updated by the DMA automatically when the DMA transfer has been completed.

Note: All the addresses in the DMA descriptors MUST be word aligned (32 bit).

When at least one descriptor is ready, you can start the RX DMA logic using this sequence:

- Load the memory address of the first descriptor word into the ENET_RXNDAR register
- Write a ‘1’ in the START_FETCH bit in the ENET_RXSTR register to start DMA operation

As soon as the RX DMA has been enabled, it loads the DMA descriptors and is ready to transfer the data sent by the MAC-802.3 core.

When a DMA operation is completed, the DMA hardware updates the packet status word and the descriptor VALID bit in main memory, and generates an interrupt (if enabled).

Then the DMA hardware verifies, in the ENET_RXCR register, if it is required to stop or to fetch a new descriptor for the next DMA transfer (NXT_EN = 1).

The location of the next DMA descriptor (DMA_DESCR_ADDR), and whether the DMA engine is enabled to load it (NXT_EN), is part of the information previously fetched with the current descriptor.

This has the advantage that the subsequent DMA descriptor starting address can be located anywhere in the memory area. Descriptors related to a single DMA operation are required to be contiguous, but data from different DMA operations can be scattered in memory.

8.3.3 RX error handling

Invalid descriptor

For added flexibility, when a descriptor is found to be not valid (VALID bit, in PACKET STATUS field, equal to '0'), the DMA engine can be programmed either to generate an interrupt (RX_NEXT) or to keep polling (NPOL_EN=1) the memory location (with a programmable period) until it has been set valid by the software application.

The interrupt register bit named RX_NEXT is always set when a non valid descriptor is loaded.

- If NPOL_EN = 0, the DMA stops and resets the DMA_EN bit in the ENET_RXSTR register. You have to re-enable the DMA operation by setting the START_FETCH bit in the ENET_RXSTR register to attempt a new descriptor fetch.
- In polling mode (NPOL_EN = 1, the DMA keeps reloading the descriptor, with an access frequency determined by the DFETCH_DLY field in the ENET_RXSTR register.

Master error

An AHB error response suspends the DMA activity, set the RX_MERR_INT bit or TX_MERR_INT in the ENET_ISR register and resets the DMA_EN bit in the ENET_RXSTR or ENET_TXSTR register. To help determine the error source, you can read the ENET_RXCAR or ENET_TXCAR register (current address register) which contains the address at which the error occurred.

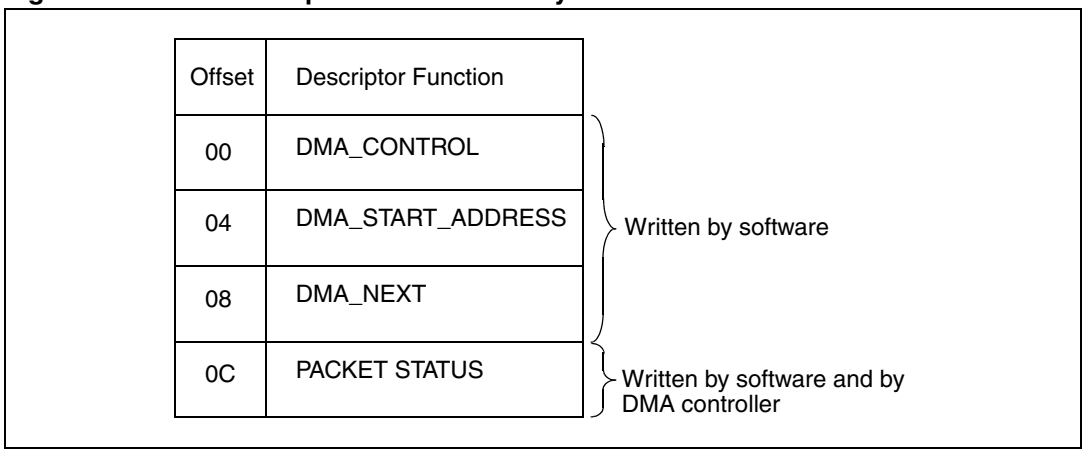
After clearing the error bit, you have to reprogram the DMA registers, to start a new descriptor fetch.

Caution: Special care must be taken when the FIFO entry to be read has 3 valid bytes: in this case, because the AHB protocol doesn't allow 3 byte transfers, the DMA splits the transfer in two single transfers (byte + half or half + byte) and sends an acknowledge signal to the FIFO only when the second one has been read. If the second read receives an error response then, when the error condition is removed, the DMA repeats the first one again (because the FIFO has not seen the acknowledge yet).

Note: While the DMA is running, the descriptor registers cannot be modified: the DMA must be stopped before attempting to change them.

For a description of the DMA configuration registers, refer to *Section 8.4*

Figure 64. DMA Descriptor in main memory



8.3.4 RX packet status word

The “0C” word of the DMA descriptor contains the following information:

- the packet status bits (updated by the MAC-802.3 core at the end of the received frame)
- the VALID bit (flag used to determine the descriptor ownership)

The MAC-802.3 packet status format is slightly modified by the DMA block, as shown in the *Figure 62*, to make room for the VALID bit.

The VALID bit contains the descriptor ownership information.

When the VALID bit is set, it indicates that the descriptor is up to date in memory and can be processed by the DMA.

When the VALID bit is reset, the descriptor either is not valid yet or has already been serviced by the DMA and can be checked by the application software.

A typical sequence of operations is:

- The application software loads the first three descriptor words in memory and then sets the VALID bit in the fourth descriptor word (new owner: DMA)
- The DMA loads the descriptor, transfers data and, at the end, in the same write operation, saves the packet status coming from the MAC-802.3 core and resets the VALID bit (new owner: host processor)
- The software checks the status and then, if needed, updates the descriptor fields and sets the VALID bit again for a new DMA transfer.

8.3.5 TX DMA configuration

Before starting TX DMA operations, you have to program the following TX DMA configuration registers.

- TX DMA Start Register (ENET_TXSTR)
- TX DMA Control Register (ENET_TXCR)
- TX DMA Start Address Register (ENET_TXSAR)
- TX DMA Next Descriptor Address Register (ENET_TXNDAR)
- TX DMA Time Out Register (ENET_TXTOR)

The RX DMA block uses the information written in the TX configuration registers, to move data to the TX FIFO from a specified memory area (master DMA).

8.3.6 TX DMA descriptors

The DMA operates by performing a DMA descriptor fetch, which allows it to load all the required information from main memory without any CPU intervention.

You have to allocate a 16-byte region (word-aligned) in main memory, for each DMA transfer. 3 words for the DMA descriptors, plus 1 word for Packet Status and descriptor VALID bit.

- In the first three descriptor words, you write the information to be loaded in the:
  - Control register (ENET_TXCR)
  - Start address register (ENET_TXSAR)
  - Next descriptor address register (ENET_TXNDAR)
- Then set to '1', bit 16 (VALID bit) in the fourth word (Packet Status)
- The packet status word will be then updated by the DMA automatically when the DMA transfer has been completed.

Note: All the addresses in the DMA descriptors MUST be word aligned (32 bit).

When at least one descriptor is ready, you can start the TX DMA logic using this sequence:

- Load the memory address of the first descriptor word into the ENET_TXNDAR register
- Write a '1' in the START_FETCH bit in the ENET_TXSTR register to start DMA operation.

As soon as the TX DMA has been enabled, it loads the DMA descriptors and is ready to transfer the data loaded from memory to the MAC-802.3 core.

When a DMA operation is completed, the DMA hardware updates the packet status word and the descriptor VALID bit in main memory, and generates an interrupt (if enabled).

Then the DMA hardware verifies, in the ENET_TXCR register, if it is required to stop or to fetch a new descriptor for the next DMA transfer (NXT_EN = 1).

The location of the next DMA descriptor (DMA_DESCR_ADDR), and whether the DMA engine is enabled to load it (NXT_EN), is part of the information previously fetched with the current descriptor.

This has the advantage that the subsequent DMA descriptor starting address can be located anywhere in the memory area. Descriptors related to a single DMA operation are required to be contiguous, but data from different DMA operations can be scattered in memory.

For added flexibility, when a descriptor is found to be not valid (VALID bit, in PACKET STATUS field, equal to '0'), the DMA engine can be programmed either to generate an interrupt (TX_NEXT) or to keep polling (NPOL_EN = 1) the memory location (with a programmable period) until it has been set valid by the software application.

Note: While the DMA is running, the descriptor registers cannot be modified: the DMA must be stopped before attempting to change them.

For a more detailed description of the TX DMA configuration registers, refer to *Section 8.4*

8.3.7 TX packet status word

The “0C” word of the DMA descriptor contains the following information:

- The packet status bits (updated by the MAC-802.3 core at the end of the transmitted frame).
- The VALID bit (flag used to determine the descriptor ownership)

The MAC-802.3 packet status format is slightly modified by the DMA block, as shown in the *Figure 62*, to make room for the VALID bit.

The VALID bit contains the descriptor ownership information.

When the VALID bit is set, it indicates that the descriptor is up to date in memory and can be processed by the DMA.

When the VALID bit is reset, the descriptor either is not valid yet or has already been serviced by the DMA and can be checked by the application software.

A typical sequence of operations is:

- The application software loads the first three descriptor words in memory and then sets the VALID bit in the fourth descriptor word (new owner: DMA).
- The DMA loads the descriptor, transfers data and, at the end, in the same write operation, saves the packet status coming from the MAC-802.3 core and resets the VALID bit (new owner: host processor).
- The software checks the status and then, if needed, updates the descriptor fields and sets the VALID bit again.

8.4 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
Read/clear (rc_w1)	Software can read as well as clear this bit by writing 1. Writing ‘0’ has no effect on the bit value
Read/set (rs)	Software can read as well as set this bit. Writing ‘0’ has no effect on the bit value
Write only (wo)	Software can only write to this bit. Reading the bit returns the reset value

8.4.1 DMA status/control register (ENET_SCR)

Address offset: 00h

Reset value: 5A5A0101h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TX_FIFO_SIZE				TX_IO_DATA_WIDTH		TX_CHAN_STATUS		RX_FIFO_SIZE				RX_IO_DATA_WIDTH		RX_CHAN _STATUS	
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								TX_MAX_BURST_SIZE		RX_MAX_BURST_SIZE		Reserved		LOOPB	SRESET
r	r	r			r	r	r	r	r	r	r	r	r		r

Bits 31:28	**TX_FIFO_SIZE[3:0]:** *Transmit FIFO size* These bits indicate the size of the transmitter data path FIFO. 0000: Reserved 0001: 2 * 32-bit words 0010: 4 * 32-bit words 0011: 8 * 32-bit words 0100: 16 * 32-bit words 0100: 32 * 32-bit words Others: Reserved
Bits 27:26	**TX_IO_DATA_WIDTH[1:0]**: *Transmit data bus width* 00: 8-bit 01: 16-bit 10: 32-bit 11: Reserved
Bits 25:24	**TX_CHAN_STATUS[1:0]**: *Transmit channel information* 00: No TX channel present 01: Low end TX channel (no DMA descriptor fetch) 10: High end TX channel 11: Reserved
Bits 23:20	**RX_FIFO_SIZE[3:0]**: *Receive FIFO size* These bits indicate the size of the receiver data path FIFO. 0000: Reserved 0001: 2 * 32-bit words 0010: 4 * 32-bit words 0011: 8 * 32-bit words 0100: 16 * 32-bit words 0100: 32 * 32-bit words Others: Reserved
Bits 19:18	**RX_IO_DATA_WIDTH[1:0]**: *Receive data bus width* 00: 8-bit 01: 16-bit 10: 32-bit 11: Reserved

Bits 17:16	**RX_CHAN _STATUS[1:0]**: *Receive channel information* 01: Low end RX channel (no DMA descriptor fetch) 10: High end RX channel 11: Reserved
Bits 15:8	**REVISION[7:0]:** *Revision number* These bits indicate the revision number of the DMA hardware
Bits 7:6	**TX_MAX_BURST_SIZE[1:0]:** *Transmit Max. burst size* These bits define the maximum size of bursts performed by the TX DMA logic on the AHB bus to read data from the main memory. 00: 16-beat incrementing burst (INCR16) 01: 8-beat incrementing burst (INCR8) 10: 4-beat incrementing burst (INCR4) 11: Single transfers only (SINGLE) Note: Descriptor fetch operation is not affected by this field.
Bits 5:4	**RX_MAX_BURST_SIZE[1:0]:** *Receive Max. burst size* These bits define the maximum size of bursts performed by the RX DMA logic on the AHB bus to write data to the main memory. 00: 16-beat incrementing burst (INCR16) 01: 8-beat incrementing burst (INCR8) 10: 4-beat incrementing burst (INCR4) 11: Single transfers only (SINGLE) Note: Descriptor fetch operation is not affected by this field.
Bits 3:2	Reserved, forced by hardware to 0
Bit 1	**LOOPB:** *Loopback mode* This bit selects loopback mode. 0: Normal mode 1: Loopback mode
Bit 0	**SRESET:** MAC DMA *Software reset* This bit is written by software. 0: Write ‘0’ to exit the reset phase. 1: Write ‘1’ to hold the whole DMA and MAC-802.3 logic in reset condition ***Notes***: - After a hardware reset, the DMA logic wakes up with the SRESET bit asserted (‘1’), to keep all the DMA and MAC-802.3 logic in the reset condition, until the software is sure that clocks and the other MII signal inputs to the MAC-802.3 core, are stable. - When this condition is met, the software is allowed to clear the SRESET bit (write ‘0’) to start the normal operation. - Until the SRESET bit is set to ‘1’, no operation is allowed on the DMA or MAC-802.3 registers, except the SRESET bit clear. - This signal has no effect on the AHB interface so, when asserted during runtime, the whole DMA will be reset only when the last AHB transfer, in the AHB master queue, has been completed.

8.4.2 DMA interrupt enable register (ENET_IER)

Address offset: 04h

Reset value: 0000 0000h

31	30 29	28	27 26	25	24	23	22	21 20	19	18	17	16
TX_CURR_DONE_EN	Reserved	MAC-802.3_INT_EN	Res.	TX_MERR_INT_EN	Reserved	TX_DONE_EN	TX_NEXT_EN	Reserved	TX_TO_EN	TX_ENTRY_EN	TX_FULL_EN	TX_EMPTY_EN
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
RX_CURR_DONE_EN	Reserved			RX_MERR_INT_EN	Reserved	RX_DONE_EN	RX_NEXT_EN	PACKET_LOST_EN	Reserved	RX_TO_EN	RX_ENTRY_EN	RX_FULL_EN	RX_EMPTY_EN
rw				rw		rw	rw	rw		rw	rw	rw	rw

Bit 31	**TX_CURR_DONE_EN:** *TX_CURR_DONE interrupt enable* 0: Disabled 1: Enabled
Bits 30:29	Reserved, forced by hardware to 0
Bit 28	**MAC-802.3_INT_EN:** *MAC-802.3 interrupt enable* 0: Disabled 1: Enabled
Bits 27:26	Reserved, forced by hardware to 0
Bit 25	**TX_MERR_INT_EN:** *TX_MERR interrupt enable* 0: Disabled 1: Enabled
Bit 24	Reserved, forced by hardware to 0
Bit 23	**TX_DONE_EN:** *TX_DONE interrupt enable* 0: Disabled 1: Enabled
Bit 22	**TX_NEXT_EN:** *TX_NEXT interrupt enable* 0: Disabled 1: Enabled
Bits 21:20	Reserved, forced by hardware to 0
Bit 19	**TX_TO_EN:** *TX_TO interrupt enable* 0: Disabled 1: Enabled

Bit 18	**TX_ENTRY_EN:** *TX_ENTRY interrupt enable* 0: Disabled 1: Enabled
Bit 17	**TX_FULL_EN:** *TX_FULL interrupt enable* 0: Disabled 1: Enabled
Bit 16	**TX_EMPTY_EN:** *TX_EMPTY interrupt enable* 0: Disabled 1: Enabled
Bit 15	**RX_CURR_DONE_EN:** *RX_CURR_DONE interrupt enable* 0: Disabled 1: Enabled
Bits 14:10	Reserved, forced by hardware to 0
Bit 9	**RX_MERR_INT_EN:** *RX_MERR interrupt enable* 0: Disabled 1: Enabled
Bit 8	Reserved, forced by hardware to 0
Bit 7	**RX_DONE_EN:** *RX_DONE interrupt enable* 0: Disabled 1: Enabled
Bit 6	**RX_NEXT_EN:** *RX_NEXT interrupt enable* 0: Disabled 1: Enabled
Bit 5	**PACKET_LOST_EN:** *PACKET_LOST interrupt enable* 0: Disabled 1: Enabled
Bit 4	Reserved, forced by hardware to 0
Bit 3	**RX_TO_EN:** *RX_TO interrupt enable* 0: Disabled 1: Enabled
Bit 2	**RX_ENTRY_EN:** *RX_ENTRY interrupt enable* 0: Disabled 1: Enabled
Bit 1	**RX_FULL_EN:** *RX_FULL interrupt enable* 0: Disabled 1: Enabled
Bit 0	**RX_EMPTY_EN:** *RX_EMPTY interrupt enable* 0: Disabled 1: Enabled

8.4.3 DMA interrupt status register (ENET_ISR)

Address offset: 08h

Reset value: 0005 0001h

31	30 29	28	27 26	25	24	23	22	21 20	19	18	17	16
TX_CURR_DONE	Reserved	MAC-802.3_INT	Reserved	TX_MERR_INT	Reserved	TX_DONE	TX_NEXT	Reserved	TX_TO	TX_ENTRY	TX_FULL	TX_EMPTY
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
RX_CURR_DONE	Reserved	RX_MERR_INT	Reserved	RX_DONE	RX_NEXT	PACKET_LOST	Reserved	RX_TO	RX_ENTRY	RX_FULL	RX_EMPTY
rc_w1		rc_w1		rc_w1	rc_w1	rc_w1		rc_w1	rc_w1	rc_w1	rc_w1

Bit 31	**TX_CURR_DONE:** *TX_CURR_DONE interrupt flag* 0: Cleared state 1: The TX master DMA has completed the current DMA transfers. ***Note*:** This bit differs from TX_DONE: TX_CURRENT_DONE is set after a single DMA descriptor execution has been completed, the status register updated and the descriptor valid bit cleared. TX_DONE is set only after all the descriptors in the descriptor chain have been fully executed.
Bits 30:29	Reserved, forced by hardware to 0
Bit 28	**MAC-802.3_INT:** *MAC-802.3 interrupt flag* 0: Cleared state 1: Interrupt request from external MAC-802.3 device (not used).
Bits 27:26	Reserved, forced by hardware to 0
Bit 25	**TX_MERR_INT:** *TX_MERR interrupt flag* 0: Cleared state 1: Master error during transmission. The AHB master has received an error response from the selected slave while the internal arbiter has granted the TX FIFO.
Bit 24	Reserved, forced by hardware to 0
Bit 23	**TX_DONE:** *TX_DONE interrupt flag* 0: Cleared state 1: TX master DMA completed

Bit 22	**TX_NEXT:** *TX_NEXT interrupt flag* 0: Cleared state 1: Invalid descriptor fetched (VALID bit in PACKET STATUS field, equal to '0')
Bits 21:20	Reserved, forced by hardware to 0
Bit 19	**TX_TO:** *TX_TO timeout interrupt flag* 0: Cleared state 1: Data has been stalled in the TX FIFO for too long
Bit 18	**TXENTRY:** *TX_ENTRY interrupt flag* 0: Cleared state 1: TX DMA has been triggered by a number of empty TX FIFO entries greater than the TX_FIFO_SIZE value set in the ENET_SCR register.
Bit 17	**TX_FULL:** *TX_FULL interrupt flag* 0: Cleared state 1: TX FIFO is full (< 4 byte entries available)
Bit 16	**TX_EMPTY:** *TX_EMPTY interrupt flag* 0: Cleared state 1: TX FIFO is empty
Bit 15	**RX_CURR_DONE:** *RX_CURR_DONE interrupt flag* 0: Cleared state 1: Set when the RX master DMA has completed the current DMA transfers. ***Note*:** This bit differs from RX_DONE: RX_CURRENT_DONE is set after a single DMA descriptor execution has been completed, the status register updated and the descriptor valid bit cleared. RX_DONE is set only after all the descriptors in the descriptor chain have been fully executed.
Bits 14:10	Reserved, forced by hardware to 0
Bit 9	**RX_MERR_INT:** *RX_MERR interrupt flag* 0: Cleared state 1: Master error during reception. The AHB master has received an error response from the selected slave while the internal arbiter has granted the RX FIFO.
Bit 8	Reserved, forced by hardware to 0
Bit 7	**RX_DONE:** *RX_DONE interrupt flag* 0: Cleared state 1: RX master DMA completed
Bit 6	**RX_NEXT:** *RX_NEXT interrupt flag* 0: Cleared state 1: Invalid descriptor fetched (VALID bit in PACKET STATUS field, equal to '0')
Bit 5	**PACKET_LOST:** *PACKET_LOST interrupt flag* 0: Cleared state 1: There is an incoming frame but the RX DMA logic cannot service it because the RX FIFO is not empty yet or the next descriptor fetch is still running.
Bit 4	Reserved, forced by hardware to 0
Bit 3	**RX_TO:** *RX_TO interrupt flag* 0: Cleared state 1: Data has been stalled in the RX FIFO for too long

Bit 2	**RXENTRY:** *RX_ENTRY interrupt flag* 0: Cleared state 1: RX DMA has been triggered by a number of valid RX FIFO entries greater than the RX_FIFO_SIZE value set in the ENET_SCR register.
Bit 1	**RX_FULL:** *RX_FULL interrupt flag* 0: Cleared state 1: RX FIFO is full and no more data can be accepted
Bit 0	**RX_EMPTY:** *RX_EMPTY interrupt flag* 0: Cleared state 1: RX FIFO is empty

8.4.4 Clock configuration register (ENET_CCR)

Address offset: 0Ch

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												SEL_CLK[1:0]		Reserved	
												rw	rw		

Bits 31:4	Reserved, forced by hardware to 0
Bits 3:2	**SEL_CLK[1:0]:** *Clock configuration* This register is the first register to be programmed after the ENET_SCR register. 00: HCLK = PCLK 01: HCLK = 2*PCLK 10: Reserved 11: Reserved ***Note*:** The MAC 802.3 specifications indicate that the MAC module should be run in the frequency range 50 - 200 MHz for proper functional operation.
Bits 1:0	Reserved, forced by hardware to 0

8.4.5 RX start register (ENET_RXSTR)

Address offset: 10h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								DFETCH_DLY[15:8]							
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DFETCH_DLY[7:0]								COLL_SEEN	RUNT_FRAME	FILTER_FAIL	Reserved		START_FETCH	Reserved	DMA_EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			rs		rc_w1

Bits 31:24	Reserved, forced by hardware to 0
Bits 23:8	**DFETCH_DLY[15:0]:** *Descriptor fetch delay* These bits specify, in bus clock periods, the delay between two descriptor fetches, in the event that the descriptor in main memory is not valid. When set to '0' it forces the DMA logic, in case of invalid descriptor, to wait for 2**16 system bus clocks before attempting a new fetch.
Bit 7	**COLL_SEEN**: *Late Collision Seen control bit* 0: No action by the DMA when a Late Collision Seen condition occurs 1: Received frames are discarded by the DMA if a Late Collision Seen condition is flagged by the MAC-802.3 in the RX packet status word.
Bit 6	**RUNT_FRAME**: *Damaged Frame control bit* 0: No action by the DMA when a Damaged Frame condition occurs 1: Received frames are discarded by the DMA if a Damaged frame condition is flagged by the MAC-802.3 in the RX packet status word (e. g. normal collision, frame too short, etc.).
Bit 5	**FILTER_FAIL**: *Address Filtering Fail control bit* 0: No action by the DMA when an Address Filtering Failed condition occurs 1: Received frames are discarded by the DMA when an Address Filtering Failed condition is flagged by the MAC-802.3 during the RX packet transmission. ***Note*:** Setting this bit reduces AHB bus utilization, because data packets filtered by the MAC core are not moved to memory.
Bits 4:3	Reserved, forced by hardware to 0

Bit 2	**START_FETCH**: *Start Fetching control bit* This bit is a Read/Set bit, that means it can be both read and written, but writing a '0' has no effect. 0: No effect 1: Start RX DMA fetching descriptors ***Notes***: - Before starting the DMA, the ENET_RXNDAR register has to be loaded with the starting address of the descriptor to be fetched. - The DMA logic will reset this bit and set the DMA_EN bit as soon as the first fetch has been completed
Bit 1	Reserved, forced by hardware to 0
Bit 0	**DMA_EN:** *DMA enable bit* Read/Clear bit: a write with '1' reset to '0' the bit value, while a write with '0' has no effect. This bit, set to '1' by the DMA after the first descriptor fetch, can be reset to '0' by the software to force a DMA abort and stop the data transfer as soon as possible, before the DMA completion. When all the DMA sequences complete normally, this bit is reset by the DMA logic and a new action by the software is required to restart the DMA engine. ***Notes***: - A DMA_EN 0->1 transition resets the FIFO content and the RX interrupts (ENET_ISR[15:0]). - A DMA_EN 1->0 transition forces the DMA to immediately close the transfers toward the AHB bus and MAC core. When the AHB transfer completes the RX_DONE bit in the ENET_ISR register is set and software can reprogram and reactivate the RX logic. - When the interrupt is received it is important to wait at least 1 RxClk before restarting the DMA i.e. before writing '1' to DMA_EN.

8.4.6 RX control register (ENET_RXCR)

Address offset: 14h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADDR_WRAP[9:0]										ENTRY_TRIG[4:0]					Res.
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
DLY_EN	NXT_EN	Res.	CONT_EN	DMA_XFERCOUNT[11:0]											
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:22	***ADDR_WRAP[9:0]:*** *DMA address counter wrap location* These bits define where the DMA address counter wraps by forcing it to retain the data originally written by the host in the ENET_RXSAR register. As soon as the DMA has written to the memory location prior to the value specified in ADD_WRAP the wrapping condition occurs. This can be used to restrict the address counter within an address window (e.g. circular buffer). The wrapping point MUST be 32-bit aligned, so the 10 bits of ADDR_WRAP are used to compare DMA address bits 11 to 2. If ADD_WRAP[9:0]= ENET_RXSAR[11:2] then a 4Kbyte buffer is defined. ***Note*:** ADDR_WRAP is ignored unless WRAP_EN is set.
Bits 21:17	**ENTRY_TRIG[4:0]:** *Entry trigger count* These bits define the amount of valid entries (in 32-bit words) required in the RX FIFO before the DMA is re-triggered. If the value is set to 0, as soon as one valid entry is present, the DMA logic starts the data transfer.
Bit 16	Reserved, forced by hardware to 0
Bit 15	**DLY_EN:** *DMA trigger delay enable* 0: Delay disabled 1: DMA trigger delay enabled. If valid data resides in the FIFO more than the time-out period programmed in the ENET_RXTOR register, a time-out condition occurs (RX_TO) and the RX FIFO is emptied even if the number of valid words does not exceed the threshold value.
Bit 14	**NXT_EN:** *Next Descriptor Fetch mode enable* 0: Next Descriptor Fetch Mode disabled. Whenever a DMA transfer is completed, no descriptor is fetched and an interrupt request is generated 1: Next Descriptor Fetch Mode enabled. Whenever a DMA transfer is completed, a new DMA descriptor is fetched. ***Note*:** When a descriptor is fetched, ENET_RXCR is one of the registers updated
Bits 13	Reserved, forced by hardware to 0

Bit 12	**CONT_EN:** *Continuous Mode Enable* 0: Normal mode 1: Continuous mode. The DMA runs indefinitely ignoring DMA_ XFERCOUNT ***Note*:** Continuous mode supersedes “Next Descriptor Mode”.
Bit 11:0	**DMA_XFERCOUNT[11:0]:** *DMA transfer count* These bits define the block size (in bytes) of DMA data transfers, up to 4 Kbytes. If DMA_XFERCOUNT is set to ‘0’, the DMA will transfer 4 Kbyte data.

8.4.7 RX start address register (ENET_RXSAR)

Address offset: 18h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RXADDR[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
RXADDR[15:0]														FIX_ADDR	WRAP_EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:2	**RXADDR[31:2]:** *Start address for master DMA transfer (32-bit word aligned)* These bits define the start address for master DMA transfer. ***Note*:** This register is taken into account by the DMA hardware before starting the DMA operation and when the wrap condition is met. Updating this register while DMA is running will have unpredictable effects.
Bit 1	**FIX_ADDR:** *Fixed address* 0: RXADDR incrementation enabled 1: RXADDR incrementation disabled. All the DMA data transfer operations are performed at the same AHB address, i.e. the ENET_RXSAR start address.
Bit 0	**WRAP_EN:** *Wrap enable* 0: Wrap disabled 1: Enables wrapping of the DMA transfer address to ENET_RXSAR when the memory location specified by the ADDR_WRAP[9:0] bits in the ENET_RXCR register, is reached.

8.4.8 RX next descriptor address register (ENET_RXNDAR)

Address offset: 1Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DESCADDR[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
DESCADDR[15:2]														Reserved	NPOL_EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bits 31:2	**DESCADDR[31:2]:** *RX DMA Next Descriptor pointer (32-bit word aligned)* When Next Descriptor Fetch mode is enabled (NXT_EN bit = 0 in the ENET_RXCR register), this register points to the next descriptor starting address. ***Notes:*** - DMA descriptors are 32-bit, so the Next Descriptor Address must be 32-bit aligned. - This register allows different DMA descriptors to be located in different memory areas, because part of the current DMA descriptor points to the next one (descriptor chaining). - If Next Descriptor Fetch mode is not enabled (NXT_EN bit = 0 in the ENET_RXCR register), this register does not need to be updated.
Bit 1	Reserved, forced by hardware to 0
Bit 0	**NPOL_EN:** *Next Descriptor Polling Enable* 0: Next Descriptor Polling disabled. If an invalid descriptor is fetched, the DMA logic sets the RX_DONE bit in the ENET_ISR register and clears the DMA_EN bit in the ENET_RXSTR register. 1: Next Descriptor Polling enabled. If an invalid descriptor is fetched, the RX_NEXT bit in the ENET_ISR register is set and a new descriptor fetch will be attempted after DFETCH_DLY clocks (refer to ENET_RXSTR register. This mode handles the case when the DMA logic fetches a descriptor that is not valid yet. The DMA logic keeps polling the DMA descriptor in main memory, until it's found to be valid.

8.4.9 RX current address register (ENET_RXCAR)

Address offset: 20h

Reset value: 0000 0000h

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

Bits 31:0	**CADDR[31:0]:** *RX DMA Current address (byte-aligned)* The value of this register changes while the DMA is running, reflecting the value driven by the core on the AHB bus.

8.4.10 RX current transfer count register (ENET_RXCTCR)

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				CXFER[11:0]											
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:12	Reserved, forced by hardware to 0
Bits 11:0	**CXFER[11:0]:** *RX DMA Current transfer count* This value is updated while the DMA is running, when a data word is moved from the MAC core to the DMA FIFO, indicating the number of bytes that can still be accepted.

8.4.11 RX time-out register (ENET_RXTOR)

Address offset: 28h

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
RXTO[15:0]															
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16	Reserved, forced by hardware to 0
Bits 15:0	**RXTO[15:0]:** *RX FIFO Time-out value* This value is used as initial value for the FIFO entry time-out counter. It is recommended not to use too low a value, to avoid generating interrupts too frequently. The time-out counter starts as soon as one valid entry is present in the FIFO and is reset each time a data entry is popped out of the FIFO. The counter expires if no FIFO data are popped for a period longer than the value programmed in the RXTO[15:0] bits. The time-out is flagged by the RX_TO bit in the ENET_ISR register and the DELAY_T bit in the ENET_RXSR register. If the RX_TO_RN bit in the ENET_IER register is set, an interrupt request is generated.

8.4.12 RX status register (ENET_RXSR)

Address offset: 2Ch

Reset value: 0000 0001h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		ENTRIES[5:0]						Reserved			DMA_POINTER[4:0]				
		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			IO_POINTER[4:0]					Reserved			DELAY_T	ENTRY_T		FULL	EMPTY
			r	r	r	r	r				r	r		r	r

Bits 31:30	Reserved, forced by hardware to 0
Bits 29:24	**ENTRIES[5:0]:** *RX FIFO entry count* These bits indicate the number of free entries (in 32 bit words) in the DMA RX FIFO
Bits 23:21	Reserved, forced by hardware to 0
Bits 20:16	**DMA_POINTER[4:0]:** *DMA RX FIFO Pointer* These bits indicate the value of the RX FIFO pointer on the DMA controller side.
Bits 15:13	Reserved, forced by hardware to 0
Bits 12:8	**IO_POINTER[4:0]:** *value* These bits indicate the value of the RX FIFO pointer on the I/O side.
Bits 7:4	Reserved, forced by hardware to 0
Bit 3	**DELAY_T:** *RX FIFO Time-out flag* 0: Normal state 1: The DMA RX FIFO delay time-out has expired
Bit 2	**ENTRY_T:** *RX FIFO Entry Threshold flag* 0: Normal state 1: The DMA RX FIFO entry trigger threshold has been reached
Bit 1	**FULL:** *RX FIFO Full flag* 0: Normal state 1: The DMA RX FIFO is full
Bit 0	**EMPTY:** *RX FIFO Empty flag* 0: Normal state 1: The DMA RX FIFO is empty

8.4.13 TX start register (ENET_TXSTR)

Address offset: 30h

Reset value: 0000 0000h

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

Bits 31:24	Reserved, forced by hardware to 0
Bits 23:8	**DFETCH_DLY[15:0]:** *Descriptor fetch delay* These bits specify, in bus clock periods, the delay between two descriptor fetches, in the event that the descriptor in main memory is not valid. When set to '0' it forces the DMA logic, in case of invalid descriptor, to wait for 2**16 system bus clocks before attempting a new fetch.
Bits 7:6	Reserved, forced by hardware to 0
Bit 5	**UNDER_RUN**: *Underrun enabled* 0: Normal state 1: If an underrun condition occurs, reported by the MAC in the TX packet status word, the DMA logic retransmits the same packet to the MAC-802.3 core, without reporting any error condition to the CPU.
Bits 4:3	Reserved, forced by hardware to 0
Bit 2	**START_FETCH**: *Start Fetching control bit* This bit is a Read/Set bit, that means it can be both read and written, but writing a '0' has no effect. 0: No effect 1: Start TX DMA fetching descriptors ***Notes***: - Before starting the DMA, the ENET_TXNDAR register has to be loaded with the starting address of the descriptor to be fetched. - The DMA logic will reset this bit and set the DMA_EN bit as soon as the first fetch has been completed

Bit 1	Reserved, forced by hardware to 0
Bit 0	**DMA_EN:** *DMA enable bit* Read/Clear bit: a write with '1' resets the bit value to '0', while a write with '0' has no effect. This bit is set to '1' by the DMA after the first descriptor fetch. It can be reset to '0' by the software to force a DMA abort and stop the data transfer as soon as possible, before the DMA completion. When all the DMA sequences complete normally, this bit is reset by the DMA logic and a new action by the software is required to restart the DMA engine. ***Notes:*** - A DMA_EN 0->1 transition resets the FIFO content and the TX interrupts (ENET_ISR [31:16]). - A DMA_EN 1->0 transition forces the DMA to immediately close the transfers toward the AHB bus and MAC core. When the AHB transfer completes the TX_DONE bit in the ENET_ISR register is set and software can reprogram and reactivate the TX logic. - When the interrupt is received it is important to wait at least 1 TxClk before restarting the DMA i.e. before writing '1' to DMA_EN.

8.4.14 TX control register (ENET_TXCR)

Address offset: 34h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADDR_WRAP[9:0]										ENTRY_TRIG[4:0]					Res.
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
DLY_ EN	NXT_EN	Res.	CONT_EN	DMA_XFERCOUNT[11:0]											
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:22	***ADDR_WRAP[9:0]:*** *DMA address counter wrap location* These bits define where the DMA address counter wraps by forcing it to retain the data originally written by the host in the ENET_TXSAR register. As soon as the DMA has read the memory location prior to the value specified in ADD_WRAP the wrapping condition occurs. This can be used to restrict the address counter within an address window (e.g. circular buffer). The wrapping point MUST be 32-bit aligned, so the 10 bits of ADDR_WRAP are used to compare DMA address bits 11 to 2. If ADD_WRAP[9:0]= ENET_TXSAR[11:2] then a 4 Kbyte buffer is defined. ***Note*:** ADDR_WRAP is ignored unless WRAP_EN is set.
Bits 21:17	**ENTRY_TRIG[4:0]:** *Entry trigger count* These bits define the amount of valid entries (in 32-bit words) required in the TX FIFO before the DMA is re-triggered. If the value is set to 0, as soon as one valid entry is present, the DMA logic starts the data request.
Bit 16	Reserved, forced by hardware to 0
Bit 15	***DLY_EN:*** *DMA trigger delay enable* 0: Delay disabled 1: DMA trigger delay enabled. If valid data resides in the FIFO more than the time-out period programmed in the ENET_TXTOR register, a time-out condition occurs (TX_TO).
Bit 14	**NXT_EN:** *Next Descriptor Fetch mode enable* 0: Next Descriptor Fetch Mode disabled. Whenever a DMA transfer is completed, no descriptor is fetched and an interrupt request is generated 1: Next Descriptor Fetch Mode enabled. Whenever a DMA transfer is completed, a new DMA descriptor is fetched. ***Note*:** When a descriptor is fetched, ENET_TXCR is one of the registers updated
Bits 13	Reserved, forced by hardware to 0

Bit 12	**CONT_EN:** *Continuous Mode Enable* 0: Normal mode 1: Continuous mode. The DMA runs indefinitely ignoring DMA_ XFERCOUNT. ***Note*:** Continuous mode supersedes “Next Descriptor Mode”.
Bit 11:0	**DMA_XFERCOUNT[11:0]:** *DMA transfer count* These bits define the block size (in bytes) of DMA data transfers, up to 4 Kbytes. If DMA_XFERCOUNT is set to ‘0’, the DMA will transfer 4 Kbyte data.

8.4.15 TX start address register (ENET_TXSAR)

Address offset: 38h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TXADDR[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
TXADDR[15:0]														FIX_ADDR	WRAP_EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:2	***TXADDR[31:2]:*** *Start address for master DMA transfer (32-bit word aligned)* These bits define the start address for master DMA transfer. ***Note*:** This register is taken into account by the DMA hardware before starting the DMA operation and when the wrap condition is met. Updating this register while DMA is running will have unpredictable effects.
Bit 1	**FIX_ADDR:** *Fixed address* 0: RXADDR incrementation enabled 1: RXADDR incrementation disabled. All the DMA data transfer operations are performed at the same AHB address, i.e. the ENET_TXSAR start address.
Bit 0	**WRAP_EN:** *Wrap enable* 0: Wrap disabled 1: Enables wrapping of the DMA transfer address to ENET_TXSAR when the memory location specified by the ADDR_WRAP[9:0] bits in the ENET_TXCR register, is reached.

8.4.16 TX next descriptor address register (ENET_TXNDAR)

Address offset: 3Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DESCADDR[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
DESCADDR[15:2]														Reserved	NPOL_EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bits 31:2	**DESCADDR[31:2]:** *TX DMA Next Descriptor pointer (32-bit word aligned)* When Next Descriptor Fetch mode is enabled (NXT_EN bit = 0 in the ENET_TXCR register), this register points to the next descriptor starting address. ***Notes:*** - DMA descriptors are 32-bit, so the Next Descriptor Address must be 32-bit aligned. - This register allows different DMA descriptors to be located in different memory areas, because part of the current DMA descriptor points to the next one (descriptor chaining). - If Next Descriptor Fetch mode is not enabled (NXT_EN bit = 0 in the ENET_TXCR register), this register does not need to be updated.
Bit 1	Reserved, forced by hardware to 0
Bit 0	**NPOL_EN:** *Next Descriptor Polling Enable* 0: Next Descriptor Polling disabled. If an invalid descriptor is fetched, the DMA logic sets the TX_DONE bit in the ENET_ISR register and clears the DMA_EN bit in the ENET_TXSTR register. 1: Next Descriptor Polling enabled. If an invalid descriptor is fetched, the TX_NEXT bit in the ENET_ISR register is set and a new descriptor fetch will be attempted after DFETCH_DLY clocks (refer to ENET_TXSTR register. This mode handles the case when the DMA logic fetches a descriptor that is not valid yet. The DMA logic keeps polling the DMA descriptor in main memory, until it's found to be valid.

8.4.17 TX current address register (ENET_TXCAR)

Address offset: 40h

Reset value: 0000 0000h

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

Bits 31:0	**CADDR[31:0]:** *TX DMA Current address (byte-aligned)* The value of this register changes while the DMA is running, reflecting the value driven by the core on the AHB bus.

8.4.18 TX current transfer count register (ENET_TXCTCR)

Address offset: 44h

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				CXFER[11:0]											
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:12	Reserved, forced by hardware to 0
Bits 11:0	**CXFER[11:0]:** *TX DMA Current transfer count (byte-aligned)* This value is updated while the DMA is running, when one word is moved from the main memory, to the DMA FIFO, indicating the number of bytes that must still be read.

8.4.19 TX time-out register (ENET_TXTOR)

Address offset: 48h

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
TXTO[15:0]															
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16	Reserved, forced by hardware to 0
Bits 15:0	**TXTO[15:0]:** *TX FIFO Time-out value* This value is used as initial value for the FIFO entry time-out counter. It is recommended not to use too low a value, to avoid generating interrupts too frequently. The time-out counter starts as soon as one valid entry is present in the FIFO and is reset each time a data entry is popped out of the FIFO. The counter expires if no FIFO data are popped for a period longer than the value programmed in the TXTO[15:0] bits. The time-out is flagged by the RX_TO bit in the ENET_ISR register and the DELAY_T bit in the ENET_TXSR register. If the TX_TO_RN bit in the ENET_IER register is set, an interrupt request is generated.

8.4.20 TX status register (ENET_TXSR)

Address offset: 4Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		ENTRIES[5:0]						Reserved			DMA_POINTER[4:0]				
		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			IO_POINTER[4:0]					Reserved				DELAY_T	ENTRY_T	FULL	EMPTY
			r	r	r	r	r					r	r	r	r

Bits 31:30	Reserved, forced by hardware to 0
Bits 29:24	**ENTRIES[5:0]:** *TX FIFO entry count* These bits indicate the number of free entries (in 32 bit words) in the DMA TX FIFO.
Bits 23:21	Reserved, forced by hardware to 0
Bits 20:16	**DMA_POINTER[4:0]:** *DMA TX FIFO Pointer* These bits indicate the value of the TX FIFO pointer on the DMA controller side.
Bits 15:13	Reserved, forced by hardware to 0
Bits 12:8	**IO_POINTER[4:0]:** *IO TX FIFO Pointer* These bits indicate the value of the TX FIFO pointer on the I/O side.
Bits 7:4	Reserved, forced by hardware to 0
Bit 3	**DELAY_T:** *TX FIFO Time-out flag* 0: Normal state 1: The DMA TX FIFO delay time-out has expired
Bit 2	**ENTRY_T:** *TX FIFO Entry Threshold flag* 0: Normal state 1: The DMA TX FIFO entry trigger threshold has been reached
Bit 1	**FULL:** *TX FIFO Full flag* 0: Normal state 1: The DMA TX FIFO is full
Bit 0	**EMPTY:** *TX FIFO Empty flag* 0: Normal state 1: The DMA TX FIFO is empty

8.4.21 MAC control register (ENET_MCR)

Address offset: 400h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RA	EN	Reserved				PS		DRO	LM[1:0]		FDM	AFM[2:0]			PWF
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
VFM	Res.		ELC	DBF	DPR	RVFF	APR	BL[1:0]		DCE	RVBE	TE	RE	Res.	RCFA
rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31	**RA:** *Receive All* This bit defines the reception mode of the incoming frames. 0: The incoming frames are received only if the address matches with the filtering rules programmed by the AFM bits. 1: All the valid incoming frames are received regardless of the destination address. The PF (packet filter) bit is set also if the address filter does not match the destination address (FF bit set).
Bit 30	**EN:** *Endianity* This bit selects the endianity mode (little or big) of the VCI interfaces when transmitting or receiving data frames. The Rx and Tx status are always transferred in little endian mode. 0: Little endian mode 1: Big endian mode
Bits 29:26	Reserved, forced by hardware to 0
Bits 25:24	**PS[1:0]:** *Prescaler bits* These bits select the HCLK divider (prescaler) used to generate the correct frequency of the aperiodic clock output on the MII_MDC pin. 00: prescaler factor = 1, system clock frequency range: f_{HCLK} <= 50 Mhz. 01: prescaler factor = 2, system clock frequency range: (50 MHz < f_{HCLK} <= 100 MHz). ***Note*:** To match the minimum period (400 ns) of the MII_MDC clock pin exactly, use the upper limit of the system clock frequency range for each setting of the prescaler field.
Bit 23	**DRO**: *Disable Receive Own* Set this bit to disable the reception of frames during transmission (MII_TX_EN pin asserted). You should reset this bit In full duplex or loopback mode and set it in half duplex mode (with no loopback). 0: Enable the MAC controller to receive all the incoming packets including those transmitted. 1: Disable the reception path during the frame transmission (MII_TX_EN pin asserted).

Bits 22:21	**LM [1:0]**: *Loopback Mode* These bits select normal or loopback operating mode. 00: Normal mode 01: Loopback mode 1x: Reserved
Bit 20	**FDM:** *Full Duplex Mode* This bit selects half duplex of full duplex operating mode. 0: Half duplex mode. 1: Full duplex mode.
Bits 19:17	**AFM[2:0]:** *Address filtering mode* These bits select the address filtering rules applied to the received frames. 000: MAC address perfect filtering for physical and multicast addresses. 001: MAC address perfect filtering for physical address and Hash filtering for multicast addresses. 010: Hash filtering for physical and multicast addresses. 011: Inverse filtering: identical to the configuration mode '000' but the filter operates in inverted mode (the frame is accepted when the MAC address perfect filtering for physical and multicast addresses failed). 100: Promiscuous filtering: any incoming valid frame is received regardless of its destination address and the FF bit (filtering fail) is never set (this filtering rule is similar to set the RA bit) 101: MAC address perfect filtering mode for physical addresses. The multicast frames are all received. 110: Hash filtering mode for physical addresses. The multicast frames are all received. 111: Reserved.
Bit 16	**PWF:** *Pass Wrong Frame* This bit select if wrong frames received are filtered or not. 0: Wrong frames are filtered. 1: Wrong frames (runt frames, overlength, late collision, MII error, extra bits, CRC error, length error, unsupported control frames) are passed.
Bit 15	**VFM**: *VLAN Filtering Mode* This bit selects the VLAN filtering mode. 0: Tagged MAC frames with the 13th and 14th bytes of the incoming frame corresponding to the content of the register field VLTAG1 or VLTAG2 (see the VL1 and VL2 registers) are received. 1: Tagged MAC frames with the 13th and 14th bytes of the incoming frame corresponding to the content of the register field VLTAG1 or VLTAG2 (see the VL1 and VL2 registers) and the 15th and 16th bytes of the incoming frame corresponding to the content of the register field VLID1 or VLID2 (see the VL1 and VL2 registers) are received.
Bits 14:13	Reserved, forced by hardware to 0
Bit 12	**ELC:** *Enable Late Collision* This bit enables/disables frame retransmission when a collision occurs outside the collision window. 0: When a late collision is detected, the LC and FA status bits (late collision and frame aborted) in the ENET_MTS register are set. 1: When a late collision is detected, the LCO and PR status bits (late collision observed and packet retry) in the ENET_MTS are set. The application should retransmit the collided frame.

Bit 11	**DBF**: *Disable Broadcast Frame* This bit enables/disables reception of broadcast frames. 0: Broadcast frame reception enabled 1: Broadcast frame reception disabled
Bit 10	**DPR**: *Disable Packet Retry* This bit enables/disables frame retransmission when a normal collision occurs. 0: in case of normal collision the PR status bit (packet retry) in the ENET_MTS register is set, indicating to the application that it should retransmit the packet. After 16 successive retransmission attempts, the frame is aborted and the EC and FA bits (excessive collisions and frame aborted) are set in the ENET_MTS register). 1: Disables the frame retransmission in case of normal collision. When a collision occurs the EC and FA bits (excessive collisions and frame aborted) are set in the ENET_MTS register).
Bit 9	**RVFF**: *VCI Rx Frame filtering* This bit enables filtering of received frames by the VCI Rx interface in the following cases: - FF bit (filtering fail) in the ENET_ MRS register is set - Early runt frame received. The purpose is to remove the filtered frames (FF bit set) or early runt frames (detected in the first 18 bytes of the frame). 0: VCI Rx Filtering disabled, all frames are transferred to the VCI Rx interface. 1: VCI Rx Filtering enabled, failed frames or early runt frames are not transferred to the VCI Rx interface.
Bit 8	**APR**: *Automatic Pad Removal* This bit enables/disables removal of the pad and CRC fields from all the incoming frames when the length field is less than 46 bytes for untagged frames or less than 42 bytes for tagged frames. 0: automatic pad removal disabled 1: automatic pad removal enabled
Bits 7:6	**BL[1:0]**: *Back-off Limit* These bits select the mode used to compute the back-off time after a collision occurrence. Once a collision is detected the MAC controller has to wait for R slot-times before re-transmitting the frame, where: 0 <= R < 2**K K = min(N, 10) N is the current number of retries (0...16) The random number R is computed using a 24-bit random generator based on a LFSR (linear feedback register) structure. The current number of retries (N) selects the bits of the LFSR used to initialize the slot-time counter. The max. number of bits of the LFSR is defined by programming the BL[1:0] bits. 00: #bits used from LFSR to initialize the slot-time counter = 10 01: #bits used from LFSR to initialize the slot-time counter = 8 10: #bits used from LFSR to initialize the slot-time counter = 6 11: #bits used from LFSR to initialize the slot-time counter = 3

Bit 5	**DCE**: *Deferral Check Enable* This bit enables the deferral check, starting the Defer counter when a Tx frame is pending because the carrier sense (MII_CRS signal) is active. The Defer counter is reset each time the transmission is started. 0: Deferral check disabled, the MAC controller can defer indefinitely. 1: Deferral check enabled. In case of excessive deferral (more than 24288 bit times) the transmission is aborted and the ED bit (excessive deferral) in the ENET_MTS register is set.
Bit 4	**RVBE**: *Reception VCI Burst Enable* This bit enables the VCI Rx interface to generate bursts. Burst write transactions are performed with zero wait states except for the first byte which requires one wait state. The burst size is fixed and equal to 4 words. The burst is initiated only when the internal Rx buffer (FIFO) has 4 entries. 0: The VCI Rx interface performs only single write transactions 1: The VCI Rx interface performs single or burst write transactions.
Bit 3	**TE**: *Transmission Enable* This bit enables frame transmission from the internal buffer to the MII interface: 0: Transmission disabled 1: Transmission enabled
Bit 2	**RE**: *Reception Enable* This bit enables frame reception from the MII interface: 0: reception disabled 1: reception enabled
Bit 1	Reserved, forced by hardware to 0
Bit 0	**RCFA**: *Reverse Control Frame Address* This bit selects the byte ordering of the control frame multicast address. 0: Multicast address = 0x010000C28001 (bits transmission order right to left) 1: Multicast address = 0x0180C2000001 (bits transmission order right to left)

8.4.22 MAC address high register (ENET_MAH)

Address offset: 404h

Reset value: 0000 FFFFh

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

Bits 31:16	Reserved, forced by hardware to 0
Bits 15:0	**MAH[15:0]:** *MAC address high* These bits contains the upper 16 bits of the Physical Address of the MAC controller. The address is loaded from the OTP sector or external EEPROM at power on but also the host can update it after the initialization.

8.4.23 MAC address low register (ENET_MAL)

Address offset: 408h

Reset value: FFFF FFFFh

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

Bits 31:0	**MAL[31:0]:** *MAC address low* These bits contains the lower 32 bits of the Physical Address of the MAC controller. The address is loaded from the OTP sector or external EEPROM at power on but also the host can update it after the initialization.

8.4.24 Multicast address high register (ENET_MCHA)

Address offset: 40Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HHT[63:48]															
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
MCA[47:32]/HHT[47:32]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**MCHA[47:32]/HHT[63:32]:** *Multicast address high / Hash table high* The content of this register depends on the filtering mode selected by the AFM[2:0] bits in the ENET_MCR register. - In perfect filtering mode it contains the upper 16 bits of the 48-bit multicast address. - In hash filtering mode it contains the upper 32 bits of the 64-bit hash table for multicast addresses.

8.4.25 Multicast address low register (ENET_MCLA)

Address offset: 410h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCA[31:16]/HLT[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
MCA[15:0]/HLT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**MCLA/HLT[31:0]:** *Multicast address low / Hash table low* The content of this register depends on the filtering mode selected by the AFM[2:0] bits in the ENET_MCR register. - In perfect filtering mode it contains the lower 32 bits of the 48-bit multicast address. - In hash filtering mode it contains the lower 32 bits of the 64-bit hash table for multicast addresses. ***Note:*** The Hash table is used for group address filtering. The upper 6 bits of the CRC register resulting from the CRC logic computation on the destination address of the incoming frame are used in the following way: - The MSB bit selects the Hash table registers (0 => HLT, 1 => HHT). - The 5 LSB bits select the n-bit (0 ..31) of the Hash table register (HLT or HHT depending on the MSB bit). If the selected n-bit is '1' then the multicast frame is accepted, else it's refused.

8.4.26 MII address register (ENET_MIIA)

Address offset: 414h

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
PADDR[4:0]					RADDR[4:0]					Reserved			PR	WR	BUSY
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw				rw	rw	rw

Bits 31:16	Reserved, forced by hardware to 0
Bits 15:11	**PADDR[4:0]:** *Physical address* These bits contain the address of the PHY device (32 available) attached to the MII interface. The MSB of the PHY address is bit 15.
Bits 10:6	**RADDR[4:0]:** *Register address* These bits contain the address of the MII register (32 available) in the selected PHY device. The MSB bit of the register address is bit 10.
Bits 5:3	Reserved, forced by hardware to 0.
Bit 2	**PR:** *Preamble removal bit* This is used to enable/disable generation of a preamble field (32 contiguous logic one bits) in all transactions (read or write). 0: Preamble generated 1: Preamble not generated
Bit 1	**WR:** *Write/Not read bit* This bit defines the type of operation (write or read) to be performed on the PHY device register. 0: Read operation 1: Write operation
Bit 0	**BUSY**: *Busy bit* This bit is used to start a write/read operation accessing the PHY register and to check (by polling) the completion of the operation in progress. 0: Idle state, the last operation (in progress) has been completed. In case of read operation, the data is available in the MIID register. 1: Busy state, is set by the application to start the programmed operation. On the completion of the operation the MAC clears this bit (idle state) and the application can read the data from the MIID register (in case of read operation) and program a new operation. ***Note*:** The PADDR, RADDR, WR fields and the MIID register must be programmed only in idle state (BUSY bit reset).

8.4.27 MII data register (ENET_MIID)

Address offset: 418h

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
RDATA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, forced by hardware to 0
Bits 15:0	**RDATA[15:0]:** *Register Data* These bits contain the read data from the PHY register after a read operation or the data to be written before a write operation.

8.4.28 MII control frame register (ENET_MCF)

Address offset: 41Ch

Reset value: 0000 0000h

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved													PCF	FCE	FCB
													rw	rw	rw

Bits 31:16	**PTIME[15:0]:** *Pause time* These bits contain the pause time (two bytes) used for the transmission of the control frame.
Bits 15:3	Reserved, forced by hardware to 0
Bit 2	**PCF**: *Pass Control Frame* This bit defines if the received control frames are passed to the host or not. 0: The MAC controller decodes the control frames but will not pass the control frames to the application. The CF bit is set, and the PF bit is reset in the ENET_MRS register to indicate to the application to flush the frame. 1: The MAC controller decodes the control frames and passes the control frames to the application. The CF and the PF bits in the ENET_MRS register are set.
Bit 1	**FCE:** *Flow Control Enable* This bit enables reception of control frames (PAUSE command) and performs the related command, disabling the transmitter for the specified time. Flow control is applicable only in full duplex mode. 0: No actions are performed (no transmission disabling) on reception of a control frame. The CF bit (control frame) in the ENET_MCR register is set anyway. 1: On reception of a control frame the transmission is disabled for the specified time. The current frame transmission is completed in anyway.
Bit 0	**FCB:** *Flow Control Busy* This bit is used to start a control frame (PAUSE command) transmission and to check (by polling) the related completion. 0: Idle state, the MAC clears this bit when the pending control frame transmission has been completed. 1: Busy state, is set by the application to start the control frame transmission. The PAUSE time parameter of the control frame is defined by the PTIME[15:0] bits. On completion of the frame transmission the MAC clears this bit (idle state). ***Note*:** The MCF register must be programmed only in idle state (FCB bit reset).

8.4.29 VLAN1 register (ENET_VL1)

Address offset: 420h

Reset value: 0000 8100h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				VLID1[11:0]											
				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
VLTAG1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:28	Reserved, forced by hardware to 0
Bits 27:16	**VLID1[11:0]:** *VLAN 1 Identifier* These bits contain contains the VLAN Identifier (12 bits) for the comparison with the 15th and 16th bytes of the tagged MAC frame (VID field) to identify the VLAN1 frames.
Bits 15:0	**VLTAG1[11:0]:** *VLAN 1 Tag* These bits contain the VLAN Tag control information for the comparison with the 13th and 14th bytes of the tagged MAC frame (13th and 14th bytes equal to 81-00h corresponding to the 802.1QTagType) to identify the VLAN1 frames. Programming a value different from 81-00h a proprietary VLAN network can be identified by this register. The 0x88-08 value (control frame type) and values less than 0x600 are forbidden for this field.

8.4.30 VLAN2 register (ENET_VL2)

Address offset: 424h

Reset value: 0000 8100h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				VLID2[11:0]											
				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
VLTAG2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:28	Reserved, forced by hardware to 0
Bits 27:16	**VLID2[11:0]:** *VLAN 2 Identifier* These bits contain contains the VLAN Identifier (12 bits) for the comparison with the 15th and 16th bytes of the tagged MAC frame (VID field) to identify the VLAN2 frames.
Bits 15:0	**VLTAG2[11:0]:** *VLAN 2 Tag* These bits contain the VLAN Tag control information for the comparison with the 13th and 14th bytes of the tagged MAC frame (13th and 14th bytes equal to 81-00h corresponding to the 802.1QTagType) to identify the VLAN2 frames. Programming a value different from 81-00h a proprietary VLAN network can be identified by this register. The 0x88-08 value (control frame type) and values less than 0x600 are forbidden for this field.

8.4.31 MAC transmission status register (ENET_MTS)

Address offset: 428h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PR	BC													Res.	
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
Res.		CC[3:0]				LCO	DEF	UR	EC	LC	ED	LOC	NC	Res.	FA
		r	r	r	r	r	r	r	r	r	r	r	r		r

Bit 31	**PR:** *Packet retry* This bit indicates that a packet retry is required. FA and PR bit settings are mutually exclusive. 0: The current frame transmission has been completed successfully (if the PR and FA bits are both reset). 1: The current frame has to be retransmitted due to a collision event
Bits 30:18	**BC[12:0]:** *Byte count* These bits contain the number of frame bytes transmitted (excluding the preamble and SFD delimiter).
Bits 17:14	Reserved, forced by hardware to 0
Bits 13:10	**CC[3:0]:** *Collision count* These bits contain the number of successive collisions that occurred before the frame re-transmission when the packet retry bit (PR) is set. In case of excessive collision (EC bit set) this field is not significant. This field is valid only in half duplex mode.
Bit 9	**LCO:** *Late Collision Observed* This bit is valid only in half duplex mode. 0: Normal state 1: Late collision observed. A collision after the collision window (64 bytes) occurred during the frame transmission while the ELC bit in the ENET_MCR register was set. The PR bit is set for the frame re-transmission request.
Bit 8	**DEF:** *Deferred* This bit is valid only in half duplex mode. 0: Normal state 1: Deferred. The transmitter has deferred starting the transmission because MII_CRS carrier sense signal was asserted.
Bit 7	**UR:** *Under run* 0: Normal state 1: Under run. A data underrun condition has occurred during the frame transmission. The frame transmission is aborted and the FA bit is set.

Bit 6	**EC:** *Excessive collision* This bit is valid only in half duplex mode. 0: Normal state 1: Excessive collision. 16 successive collisions occurred during the frame transmission when the DPR bit in the ENET_MCR register is reset. If the DPR bit is set, the EC bit is set on the first collision occurrence. The frame transmission is aborted and the FA bit is set.
Bit 5	**LC:** *Late collision* This bit is valid only in half duplex mode. 0: Normal state 1: Late collision. During the frame transmission, a collision occurred after the collision window (64 bytes). The frame transmission is aborted and the FA bit is set.
Bit 4	**ED:** *Excessive deferral* This bit is valid only in half duplex mode. 0: Normal state 1: Excessive deferral. The transmission is terminated due to excessive deferral when the DCE bit in the ENET_MCR register is set. The deferral time-out occurs when a frame transmission is deferred for more than 24288 bit times. The the frame transmission is then aborted and the FA bit is set.
Bit 3	**LOC:** *Loss of Carrier* This bit is valid only in half duplex mode. 0: Normal state 1: Loss of carrier. The MII_CRS carrier sense signal has been de-asserted for at least one TX clock cycle during frame transmission. The frame transmission is finished but when the status is returned, the LOC and FA bits are set, and the frame transmission must be considered failed.
Bit 2	**NC:** *No Carrier* This bit is valid only in half duplex mode. 0: Normal state 1: No carrier. The MII_CRS carrier sense signal was not present during the whole frame transmission. The frame transmission is aborted and the related bit FA (frame abort) is set.
Bit 1	Reserved, forced by hardware to 0.
Bit 0	**FA:** *Frame Aborted* This bit indicates that the current frame has been aborted by the MAC controller for one of the following reasons: - No Carrier (NC) - Loss of Carrier (LOC) - Excessive Deferral (ED) - Late Collision (LC) - Excessive Collisions (EC) - Under Run (UR) 0: Normal state 1: Frame aborted. For each abort condition, the related status bit is set in this register.

8.4.32 MAC reception status register (ENET_MRS)

Address offset: 42Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FA	PF	FF	BF	MCF	UCF	CF	LE	VL2	VL1	CE	EB	ME	FT	LC	OL
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
RF	WT	FCI	Reserved		FL										
r	r	r			r	r	r	r	r	r	r	r	r		r

Bit 31	**FA:** *Frame Aborted* This bit indicates that the current frame has been aborted for one of the following reasons: - Excessive latency (wait cycles) by the application in acknowledging the data when related buffer of the MAC controller is full. - Abort response from the application during data transfer 0: Normal state 1: Frame aborted
Bit 30	**PF:** *Packet filter* 0: Invalid frame. Detected in the following cases: runt frame, over length, late collision, MII error, extra bits, CRC error, length error, unsupported control frame, extension error. 1: Valid frame. A frame is considered valid when one of the following conditions is matched (in all cases, for broadcast frames, the DBF bit in the ENET_MCR register must be cleared and for control frames, the PCF bit in the ENET_MCF register must be set): - The FF (filtering fail) bit is reset and no error condition has been detected. - The FF bit is reset and an error condition has been detected (error frame) but the PWF (pass wrong frame) bit in the ENET_MCR register is set. - The FF bit is set, the RA (receive all) bit in the ENET_MCR register is reset, no error condition has been detected. - The FF bit is set, the RA bit is reset, error condition has been detected (error frame), and the PWF bit is set. - The FF bit is set, the RA bit is set, no error condition has been detected. - The FF bit is set, the RA bit is set, a error condition has been detected (error frame), the PWF bit is set.
Bit 29	**FF:** *Filtering fail* 0: Normal state 1: Filtering fail. The destination address field of the current frame failed the address filtering rules programmed in the AFM[2:0] bits in the ENET_MCR register.
Bit 28	**BF:** *Broadcast Frame* 0: Normal state 1: Broadcast frame. The destination address field of the current frame is a broadcast address (all the bits of the destination address set to 1).

Bit 27	**MCF:** *Multicast Frame* 0: Normal state 1: Multicast frame. The destination address field of the current frame is a multicast-group address (LSB bit of the destination address set to 1).
Bit 26	**UCF:** *Unsupported Control Frame* 0: Normal state 1: The current frame is an unsupported MAC control frame. A control frame is received (destination address hexadecimal value = 01-80-C2-00-00-01 or physical address matched, type field hexadecimal value: 88-08) and the opcode field doesn't correspond to the PAUSE command (opcode hexadecimal value: 00-01) or the frame length is not equal to 64 bytes. ***Note*:** When the CF bit is set and the UCF bit is reset, a PAUSE command has been received.
Bit 25	**CF:** *Control Frame* This bit is valid only in full duplex mode. 0: Normal state 1: The current frame is identified as a MAC control frame. The MAC control frame structure is described in *Section 8.1.1*. This bit is set when a Control frame is received (destination address hexadecimal value = 01-80-C2-00-00-01 or physical address matched, type field hexadecimal value: 88-08).
Bit 24	**LE**: *Length Error* 0: Normal state 1: The length field of the current frame is inconsistent with the payload size (number of data bytes) of the received packet. ***Notes***: - The length field is the 13th and 14th bytes of an untagged MAC frame, and the 17th and 18th bytes of a tagged MAC frame. - This bit is valid only if the FT (frame type) bit is reset. - This bit is not set when a runt frame is received.
Bit 23	**VL2**: *Vlan2 Tag* This bit indicates that the frame received is a VLAN2 type tagged frame. Its meaning depends on the setting of the VFM bit in the ENET_MCR register. 0: Normal state 1: ***If VFM = 0*:** a tagged MAC frame has been received and the 13th and 14th bytes the frame match the VLTAG2[15:0] bits in the ENET_VL2 register. ***If VFM = 1*:** a tagged MAC frame has been received and the 13th and 14th bytes the frame match the VLTAG2[15:0] bits and the 15th and 16th bytes of the frame match the VLID2[11:0] bits in the ENET_VL2 register.
Bit 22	**VL1**: *Vlan1 Tag* This bit indicates that the frame received is a VLAN1 type tagged frame. Its meaning depends on the setting of the VFM bit in the ENET_MCR register. 0: Normal state 1: ***If VFM = 0*:** a tagged MAC frame has been received and the 13th and 14th bytes the frame match the VLTAG1[15:0] bits in the ENET_VL1 register. ***If VFM = 1*:** a tagged MAC frame has been received and the 13th and 14th bytes the frame match the VLTAG1[15:0] bits and the 15th and 16th bytes of the frame match the VLID1[11:0] bits in the ENET_VL1 register.

Bit 21	**CE**: *CRC Error* 0: Normal state 1: The received frame contains a wrong CRC field. ***Notes:*** - When the ME bit (MII error) is set, the CE bit is set although the CRC field is correct. - This bit is not set when a runt frame or overflow condition occurs.
Bit 20	**EB**: *Extra Bit* 0: Normal state 1: The received frame contains a non integer number of bytes. ***Note:*** If EB bit is set and CE (CRC error) bit is reset, the frame is valid.
Bit 19	**ME**: *MII Error* 0: Normal state 1:The MII_RX_ER signal has been asserted during the frame data reception.
Bit 18	**FT**: *Frame Type* 0: Normal state 1: An Ethernet frame has been received (frame length/type greater or equal to the hexadecimal value 06-00). The Length/Type field to be checked is the 13th and 14th bytes for untagged MAC frame and the 17th and 18th bytes for tagged MAC frame. ***Note:*** If the received frame is a MAC control frame the CF bit is also set.
Bit 17	**LC**: *Late Collision* 0: Normal state 1: A collision occurred after the collision window (64 bytes following the SFD start frame delimiter)
Bit 16	**OL**: *Over Length* 0: Normal state 1: The frame is received correctly, but the frame length is greater than the maximum frame size (1518 bytes for a untagged MAC frame, 1522 bytes for a tagged MAC frame).
Bit 15	**RF**: *Runt Frame* 0: Normal state 1:The current frame was damaged within the collision window (64 bytes) due to a collision event or premature termination.
Bit 14	**WT**: *Watchdog Time-out* 0: Normal state 1:The watchdog time-out, the data field length of the received frame is over 2047 bytes (FLT overflow).
Bit 13	**FCI**: *False Carrier Indication* 0: Normal state 1:The PHY device indicates a false carrier condition.
Bits 12:11	Reserved, forced by hardware to 0
Bits 10:0	**FL**: *Frame Length* These bits indicates the byte length of the received frame (excluding preamble and SFD delimiter). This field is valid when the WT bit is not set.

8.5 Ethernet controller register map

Table 22. Ethernet controller register map

Address 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	**ENET_SCR**	DMA Status/Control Register																															
04h	**ENET_IER**	DMA Interrupt Enable Register																															
08h	**ENET_ISR**	DMA Interrupt Status Register																															
0Ch	**ENET_CCR**	Clock Control Register																															
10h	**ENET_RXSTR**	RX DMA Start Register																															
14h	**ENET_RXCR**	RX DMA Control Register																															
18	**ENET_RXSAR**	RX DMA Start Address Register																															
1C	**ENET_RXNDAR**	RX DMA Next Descriptor Address Register																															
20	**ENET_RXCAR**	RX DMA Current Address Register																															
24	**ENET_RXCTCR**	RX DMA Current Transfer Count Register																															
28	**ENET_RXTOR**	RX DMA Time Out Register																															
2C	**ENET_RXSR**	RX DMA Status Register																															
30	**ENET_TXSTR**	TX DMA Start Register																															
34	**ENET_TXCR**	TX DMA Control Register																															
38	**ENET_TXSAR**	TX DMA Start Address Register																															
3C	**ENET_TXNDAR**	TX DMA Next Descriptor Address Register																															
40	**ENET_TXCAR**	TX DMA Current Address Register																															
44	**ENET_TXTCR**	TX DMA Current Transfer Count Register																															
48	**ENET_TXTOR**	TX DMA Time Out Register																															
4C	**ENET_TXSR**	TX DMA Status Register																															
100	**RX_FIFO_0**	RX DMA FIFO																															
..	**..**																																
17C	**RX_FIFO_31**																																
180-1FF		Reserved																															
200	**TX_FIFO_0**	TX DMA FIFO																															
..	**..**																																
27C	**TX_FIFO_31**																																
400	**ENET_MCR**	MAC Control Register																															
404	**ENET_MAH**	MAC Address High Register																															
408	**ENET_MAL**	MAC Address Low Register																															
40C	**ENET_MCHA**	Multicast Address High Register																															
410	**ENET_MCLA**	Multicast Address Low Register																															
414	**ENET_MIIA**	MII Address Register																															
418	**ENET_MIID**	MII Data Register																															
41C	**ENET_MCF**	MAC Control Frame Register																															
420	**ENET_VL1**	VLAN1 register																															
424	**ENET_VL2**	VLAN2 register																															
428	**ENET_MTS**	MAC Transmission Status Register																															
42C	**ENET_MRS**	MAC Reception Status Register																															

Refer to *Table 5 on page 35* for the register base addresses.

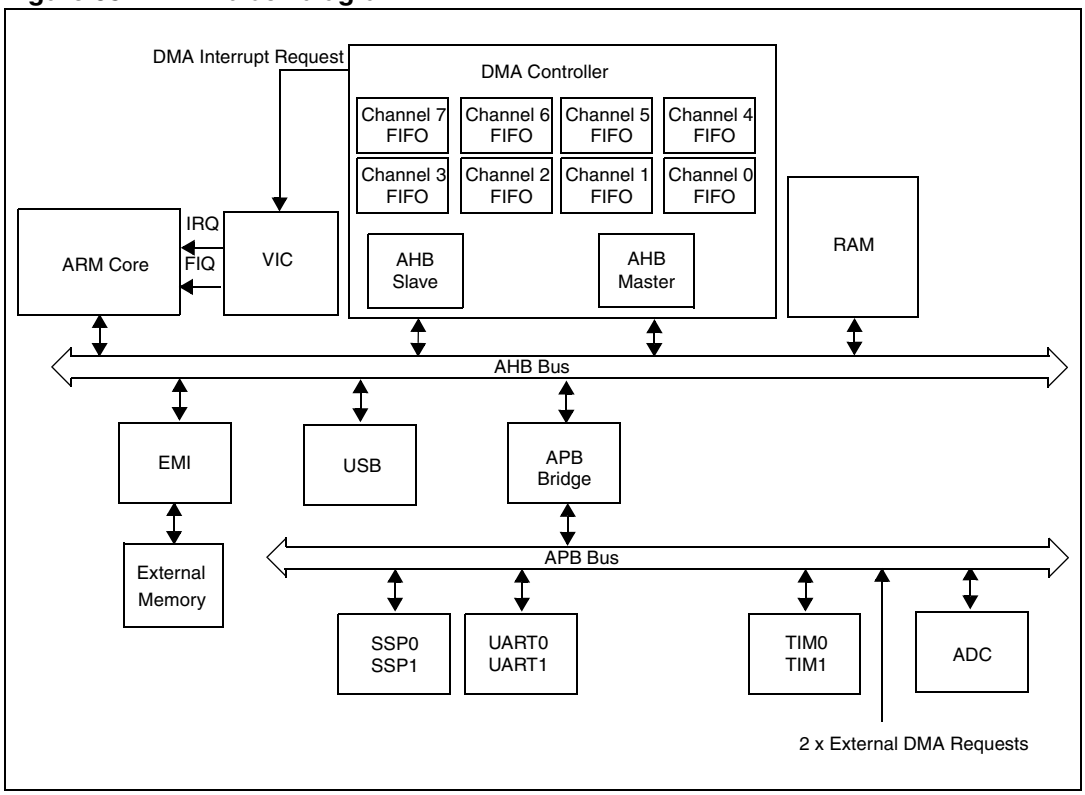
9 DMA controller (DMAC)

9.1 Introduction

The DMAC enables memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral transactions. Each DMA stream provides unidirectional serial DMA transfers for a single source and destination. For example, a bidirectional port requires one stream for transmit and one for receive. The source and destination areas can each be either a memory region or a peripheral, and you can access them through the same AHB master, or one area by each master. Refer to *Figure 65*.

Note: The Ethernet controller has its own dedicated DMA controller described in Section 8.

Figure 65. DMA block diagram



9.2 Main features

- Eight DMA channels. Each channel can support a unidirectional transfer.
- 14 DMA requests. The DMAC provides 16 peripheral DMA request lines.
- Single DMA and burst DMA request signals. Each peripheral connected to the DMAC can assert either a burst DMA request or a single DMA request. You set the DMA burst size by programming the DMAC.
- Memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral transfers.
- Scatter or gather DMA support through the use of linked lists.
- Hardware DMA channel priority. Each DMA channel has a specific hardware priority. DMA channel 0 has the highest priority and channel 7 has the lowest priority. If requests from two channels become active at the same time, the channel with the highest priority is serviced first.
- 32-bit AHB master bus width.
- Incrementing or non-incrementing addressing for source and destination.
- Programmable DMA burst size. You can program the DMA burst size to transfer data more efficiently. The burst size is usually set to half the size of the FIFO in the peripheral.
- Internal four word FIFO per channel.
- Supports 8, 16, and 32-bit wide transactions.
- Separate and combined DMA error and DMA Terminal Count interrupt requests. You can generate an interrupt to the processor on a DMA error or when a DMA count has reached 0 (Terminal Count event, usually used to indicate that a transfer has finished).
  - Interrupt masking. You can mask the DMA Error and DMA Terminal Count interrupt requests.
  - Raw interrupt status. You can read the DMA error and DMA count raw interrupt status prior to masking.

Table 23. DMA request signal mapping

DMA request signal	Associated peripheral function	Comments
0	USB RX	USB has only 1 DMA Req signal
1	USB TX	
2	TIM0	2 out of 4 TIMs have DMA support (TIM2 and 3 are not supported by DMA)
3	TIM1	
4	UART0 RX	2 out of 3 UARTs have DMA support (UART2 is not supported by DMA)
5	UART0 TX	
6	UART1 RX	
7	UART1 TX	
8	External DMA Req 0	Shared with GPIO
9	External DMA Req 1 or ADC	
10	Reserved	
11	Reserved	
12	SSP0 RX	
13	SSP0 TX	
14	SSP1 RX	
15	SSP1 TX	

9.3 Functional description

9.3.1 DMA request priority

DMA channel priority is fixed. DMA channel 0 has the highest priority and DMA channel 7 has the lowest priority.

If the DMAC is transferring data for a lower priority channel, and then a higher priority channel goes active, it completes the number of transfers delegated to the master interface by the lower priority channel before switching over to transfer data for the higher priority channel. In the worst case, this is as large as one quadword.

The two lowest priority channels in the DMAC, 6 and 7, are designed so that they cannot saturate the AHB bus. If one of these lower priority channels goes active, the DMAC relinquishes the bus for one cycle each four transfers of the programmed WIDTH irrespective of the size of the transfer. For example, if the programmed size WIDTH is 8, then after four transfers of 8 bits the DMAC relinquishes the bus. This enables other AHB masters to access the bus.

It is recommended that memory-to-memory transactions use one of these low-priority channels or other lower priority AHB bus masters cannot access the bus during DMAC memory-to-memory transfer.

9.3.2 Protection control

Software programs PROT[2:0] bits for each DMA channel. The bits are set as follows:

PROT[0] User or Privileged. User = 0, privileged = 1. Programmed by software. See *Channel control register x (DMA_CCx) on page 268*. During LLI loads, PROT[0] is made 1, privileged.

PROT[1] Bufferable or Nonbufferable. Nonbufferable = 0, bufferable = 1. Programmed by software. See *Channel control register x (DMA_CCx) on page 268*. During LLI loads, PROT[1] is made 0.

PROT[2] Cacheable or Noncacheable. Noncacheable = 0, cacheable = 1. Programmed by software. See *Channel control register x (DMA_CCx) on page 268*. During LLI loads, PROT[2] is made 1.

Peripherals can interpret the PROT information as required to help perform efficient transactions. For example:

- You can use the PROT[1] User or privileged bit to protect certain peripherals or memory spaces from User mode transactions.
- You can use the PROT[1] bufferable or nonbufferable bit to indicate to the APB bridge that the write can complete in zero wait states on the source bus. This is without waiting for it to arbitrate for the destination bus and for the slave to accept the data.
- The APB bridge can use the PROT[2] cacheable or noncacheable bit so that on the first read of a burst of eight, it can transfer the whole burst of eight reads on the destination bus, rather than pass the transactions through one at a time.

9.3.3 Lock control

Set the lock bit by programming bit 16 in the DMA_CCNFx Register. See *Channel control register x (DMA_CCx) on page 268*. When a burst occurs, the AHB arbiter must not degrant the master during the burst until the lock is deasserted. You can lock the DMAC for a single burst such as a long source fetch burst or a long destination drain burst. The DMAC does not usually assert the lock continuously for a source fetch burst followed by a destination drain burst.

There are situations when the DMAC asserts the lock for source transfers followed by destination transfers. This is possible when internal conditions in the DMAC enable it to perform a source fetch followed by a destination drain back-to-back.

This is possible when internal conditions in the DMAC enable it to perform a source fetch followed by a destination drain back-to-back, and when the following conditions are both met:

- Source width = destination width, and,
- Source burst size is a minimum of 4.

9.3.4 Bus width

The source width, SWidth, or destination width, DWidth, values in the DMA_CCRx Register program the bus transfer size.

9.3.5 Interrupt generation logic

The DMAC generates a maskable interrupt to the Interrupt Controller. The interrupt is an ORed function of the DMAC Error interrupt and DMAC Terminal Count interrupt.

9.4 Software considerations

You must take into account the following software considerations when programming the DMAC:

There must not be any write-operation to Channel registers in an active channel after the channel enable is made HIGH. If you must reprogram any DMAC channel parameters, you must reprogram after disabling the DMAC channel.

- If the source width is less than the destination width, the TransferSize value multiplied by the source width must be an integral multiple of the destination width.
- When the source peripheral is the flow controller and the source width is less than the destination width, the number of transfers that the source peripheral performs, before asserting an DMA request, must be so that the number of transfers multiplied by the source width is an integral multiple of the destination width. If this case is violated, the data can get stuck and lost in the FIFO causing UNPREDICTABLE results. You can abort the transfer by disabling the relevant DMAC channel.
- You must not program the SrcPeripheral and DestPeripheral bit fields in the DMA_CCNFx Register (see *Channel configuration register x (DMA_CCNFx) on page 271*) with any value greater than 15.
- The SWidth and DWidth bit fields in the DMA_CCRx Register (see *Channel control register x (DMA_CCx) on page 268*) must not indicate more than a 32-bit wide peripheral.
- After the software disables a channel by clearing the E bit in the DMA_CCNFx Register (see *Channel configuration register x (DMA_CCNFx) on page 271*), it must re-enable the bit only after it has polled a 0 in the corresponding DMA_ENCSR Register bit (see *Enabled channel status register (DMA_ENCSR) on page 261*). This is because the actual disabling does not immediately happen with the clearing of the E bit. You must accommodate the latency of the ongoing AHB burst.
- The LLI field in the DMA_LLIx Register (see *Channel linked list item register x (DMA_LLIx) on page 267*) must not indicate an address greater than 0xFFFFFFF0, otherwise the four-word LLI burst wraps over at 0x00000000 and the LLI data structure is not in contiguous memory locations.
- When the transfer size programmed in the DMAC is greater than the depth of the FIFO in a source or destination peripheral, you must only program the DMAC for non-incrementing address generation.

If you program the TransferSize field in the DMA_CCRx Register (see *Channel control register x (DMA_CCx) on page 268*) as zero, and the DMAC is the flow controller (the TransferSize field has no meaning in other flow-control modes) then the channel does not initiate any transfers. It is your responsibility to disable the channel by writing into the channel enable bit of the DMA_CCNFx Register and reprogramming the channel again.

- You must not run the normal read-write tests on the DMA_CCRx Register (see *Channel control register x (DMA_CCx) on page 268*) because the TransferSize field is not a typical write and read-back register field. While writing, the TransferSize bit-field is like a control register because it determines how many transfers the DMAC performs. However, during read-back, TransferSize behaves like a status register because it returns the number of remaining transfers in terms of source width. So when TransferSize is read back, it returns the number of destination-transfer-completed stored in a separate counter called TrfSizeDst multiplied by a factor. The same physical register is not being written into and read from, and normal write and read-back tests are not applicable.
- In the destination flow control mode, with peripheral-to-peripheral transfer, if sufficient data is present in the channel FIFO to service a DMA request raised by a destination peripheral without requiring data to be fetched from the source peripheral, then the source peripheral is issued a DMA Terminal count signal.
- For destination flow controlled case (peripheral-to-peripheral transfer) with DWidth < SWidth, the number of data bytes requested by the destination peripheral must be an integral multiple of Swidth expressed in bytes. If you do not ensure this, then the DMAC might fetch more data from the source peripheral than is required. This can result in data loss.
- At the end of accesses corresponding to low-priority channels, an IDLE cycle is inserted on the AHB bus to enable other masters to access the bus. This ensures that a low-priority channel does not monopolize the bus. It does, however, mean that the bus might be occupied by transactions corresponding to a low priority for up to 16 cycles in the worst case. This applies to all transfer configurations, including memory-to-memory transfers.

9.4.1 Error conditions

An error during a DMA transfer is flagged directly by the peripheral by asserting an Error response on the AHB bus during the transfer. The DMAC automatically disables the DMA stream after the current transfer has completed, and optionally generates an error interrupt to the CPU. You can mask this error interrupt.

9.4.2 Programming the DMAC

All transactions on the AHB Slave programming bus must be 32 bits wide. This eliminates endian issues when programming the DMAC. This section provides more information on programming the DMAC:

- Enabling the DMAC
- Disabling the DMAC
- Enabling a DMA channel
- Disabling a DMA channel
- Setting up a new DMA transfer
- Halting a DMA channel on page
- Programming a DMA channel

Enabling the DMAC

Enable the DMAC by setting the EN bit in the DMA_CNFR Register. See *Configuration register (DMA_CNFR) on page 263*.

Disabling the DMAC

To disable the DMAC:

1. Read the DMA_ENCSR Register and ensure that you have disabled all the DMA channels. If any channels are active, see *Disabling a DMA channel on page 249*.
2. Disable the DMAC by writing 0 to the EN bit in the DMA_CNFR Register. See *Configuration register (DMA_CNFR) on page 263*.

Enabling a DMA channel

Enable the DMA channel by setting the Channel Enable bit in the relevant DMA channel Configuration Register. See *Channel configuration register x (DMA_CCNFx) on page 271*.

Note: You must fully initialize the channel before you enable it. Additionally, you must set the EN bit of the DMAC before you enable any channels.

Disabling a DMA channel

You can disable a DMA channel in the following ways:

- Write directly to the Channel Enable bit.

Note: You lose any outstanding data in the FIFOs if you use this method.

- Use the Active and Halt bits in conjunction with the Channel Enable bit.
- Wait until the transfer completes. The channel is then automatically disabled.

Disabling a DMA channel and losing data in the FIFO

Clear the relevant Channel Enable bit in the relevant channel Configuration Register. See *Channel configuration register x (DMA_CCNFx) on page 271*. The current AHB transfer, if one is in progress, completes and the channel is disabled.

Note: You lose any data in the FIFO.

Disabling a DMA channel without losing data in the FIFO

To disable a DMA channel without losing data in the FIFO:

1. Set the Halt bit in the relevant channel Configuration Register. See *Channel configuration register x (DMA_CCNFx) on page 271*. This causes any subsequent DMA requests to be ignored.
2. Poll the Active bit in the relevant channel Configuration Register until it reaches 0. This bit indicates whether there is any data in the channel that has to be transferred.
3. Clear the Channel Enable bit in the relevant channel Configuration Register.

Setting up a new DMA transfer

To set up a new DMA transfer:

1. If the channel is not set aside for the DMA transaction:
   a) Read the DMA_ENCSR Register and determine the channels that are inactive. See *Enabled channel status register (DMA_ENCSR) on page 261*.
   b) Choose an inactive channel that has the necessary priority.
2. Program the DMAC.

Halting a DMA channel

Set the Halt bit in the relevant DMA channel Configuration Register. The current source request is serviced. Any subsequent source DMA requests are ignored until the Halt bit is cleared.

Programming a DMA channel

To program a DMA channel:

1. Choose a free DMA channel with the necessary priority. DMA channel 0 has the highest priority and DMA channel 7 has the lowest priority.
2. Clear any pending interrupts on the channel you want to use by writing to the DMA_TCICR and DMA_EICR Registers. See *Terminal count interrupt clear register (DMA_TCICR) on page 258* and *Error interrupt clear register (DMA_EICR) on page 259*. The previous channel operation might have left interrupts active.
3. Write the source address into the DMA_SRCx Register. See *Channel source address register x (DMA_SRCx) on page 265*.
4. Write the destination address into the DMA_DESTx Register. See *Channel destination address register x (DMA_DESTx) on page 266*.
5. Write the address of the next LLI into the DMA_LLIx Register. See *Channel linked list item register x (DMA_LLIx) on page 267*. If the transfer consists of a single packet of data, you must write 0 into this register.
6. Write the control information into the DMA_CCRx Register. See *Channel control register x (DMA_CCx) on page 268*.
7. Write the channel configuration information into the DMA_CCNFx Register. See *Channel configuration register x (DMA_CCNFx) on page 271*. If the Enable bit is set, then the DMA channel is automatically enabled.

9.4.3 Address generation

Address generation can be either incrementing or non-incrementing (address wrapping is not supported). Bursts do not cross the 1 Kb address boundary.

9.4.4 Scatter/gather

Scatter/gather is supported through the use of linked lists. This means that the source and destination areas do not have to occupy contiguous areas in memory. You must set the DMA_LLIx Register to 0 if you do not require scatter/gather.

9.4.5 Linked list items

An LLI consists of four words. These words are organized in the following order:

1. DMA_SRCx
2. DMA_DESTx
3. DMA_LLIx
4. DMA_CCRx

Note: The DMA_CCNFx Channel Configuration Register is not part of the LLI.

A series of linked lists define the source and destination data areas. Each LLI controls the transfer of one block of data, and then optionally loads another LLI to continue the DMA operation, or stops the DMA stream. The first LLI is programmed into the DMAC.

The data to be transferred described by an LLI, referred to as the packet of data, usually requires one or more DMA bursts (to each of the source and destination). *Figure 66* shows an example of an LLI. A rectangle of memory has to be transferred to a peripheral. The addresses of each line of data are given (in hexadecimal) at the left-hand side of the figure. The LLIs describing the transfer are to be stored contiguously from address 0x20000. The first LLI, stored at 0x20000, defines the first block of data to be transferred. This is the data stored between addresses 0x0A200 and 0x0AE00:

- source start address 0x0A200
- destination address set to the destination peripheral address
- transfer width, word (32-bit)
- transfer size, 3 072 bytes (0xC00)
- source and destination burst sizes, 16 transfers
- next LLI address, 0x20010

The second LLI, stored at 0x20010, describes the next block of data to be transferred:

- source start address 0x0B200
- destination address set to the destination peripheral address
- transfer width, word (32-bit)
- transfer size, 3 072 bytes (0xC00)
- source and destination burst sizes, 16 transfers
- next LLI address, 0x20020

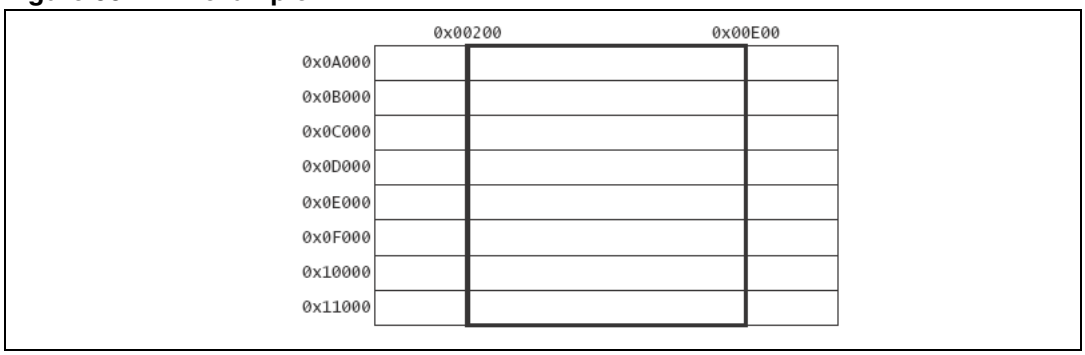
A chain of descriptors is built up, each one pointing to the next in the series. To initialize the DMA stream, the first LLI, 0x20000, is programmed into the DMAC. When the first packet of data has been transferred, the next LLI is automatically loaded.

The final LLI is stored at 0x20070 and contains:

- source start address 0x11200
- destination address set to the destination peripheral address
- transfer width, word (32-bit)
- transfer size, 3 072 bytes (0xC00)
- source and destination burst sizes, 16 transfers
- next LLI address, 0x0

Because the next LLI address is set to zero, this is the last descriptor, and the DMA channel is disabled after transferring the last item of data. The channel is probably set to generate an interrupt at this point to indicate to the ARM processor that the channel can be reprogrammed.

Figure 66. LLI example



9.4.6 Programming the DMAC for scatter/gather DMA

To program the DMAC for scatter/gather DMA:

1. Write the LLIs for the complete DMA transfer to memory. Each LLI contains four words:
   a) source address
   b) destination address
   c) pointer to next LLI
   d) control word.

   The last LLI has its linked list word pointer set to 0.
2. Choose a free DMA channel with the required priority. DMA channel 0 has the highest priority and DMA channel 7 the lowest priority.
3. Write the first LLI, previously written to memory, to the relevant channel in the DMAC.
4. Write the channel configuration information to the channel configuration register and set the Channel Enable bit. The DMAC then transfers the first and then subsequent packets of data as each LLI is loaded.
5. An interrupt can be generated at the end of each LLI depending on the Terminal Count bit in the DMA_CCRx Register. If this bit is set, an interrupt is generated at the end of the relevant LLI. You must then service the interrupt request, and you must set the relevant bit in the DMA_TCICR Register to clear the interrupt.

9.4.7 Interrupt requests

Interrupt requests can be generated when an AHB error is encountered, or at the end of a transfer (terminal count) after all the data corresponding to the current LLI has been transferred to the destination. The interrupts can be masked by programming the relevant bits on the relevant DMA_CCRx and DMA_CCNFx Channel Registers.

Interrupt Status Registers are provided. They group the interrupt requests from all the DMA channels prior to interrupt masking (DMA_TCRISR, DMA_ERISR), and after interrupt masking (DMA_TCISR, DMA_EISR).

The DMA_ISR Register combines both the DMA_TCISR and DMA_EISR requests into a single register to enable the source of an interrupt to be found quickly. Writing to the DMA_TCICR or the DMA_EICR Registers with a bit set HIGH enables selective clearing of interrupts.

The DMAC has a combined error and end of transfer complete interrupt request. To find the source of an interrupt, you must read both the DMA_ISR and DMA_TCISR Registers.

9.4.8 Combined terminal count and error interrupt sequence flow

When you use the DMACINTR interrupt request:

1. You must wait until the combined interrupt request from the DMAC goes active.
2. Assuming the interrupt is enabled in the interrupt controller and in the processor, the processor branches to the interrupt vector address and enters the interrupt service routine.
3. You must read the interrupt controller Status Register and determine whether the source of the request was the DMAC.
4. You must read the DMA_ISR Register to determine the channel that generated the interrupt. If more than one request is active, it is recommended that you check the highest priority channels first.
5. You must read the DMA_TCISR Register to determine whether the interrupt was generated because of the end of the transfer (terminal count) or because an error occurred. A HIGH bit indicates that the transfer completed.
6. You must read the DMA_EISR Register to determine whether the interrupt was generated because of the end of the transfer (terminal count) or because an error occurred. A HIGH bit indicates that an error occurred.
7. You must write a 1 to the relevant bit in the DMA_TCICR (or DMA_EICR) Register to clear the interrupt request.

9.4.9 Interrupt polling sequence flow

The DMAC interrupt request signal is masked out, disabled in the interrupt controller, or disabled in the processor. When polling the DMAC, you must:

1. Read the DMA_ISR Register. If none of the bits are HIGH repeat this step, otherwise, go to step 2. If more than one request is active, it is recommended that you check the highest priority channels first.
2. Read the DMA_TCISR Register to determine if the interrupt was generated because of the end of the transfer (terminal count) or because of error occurred. A HIGH bit indicates that the transfer completed.
3. Service the interrupt request.
4. For an error interrupt, write a 1 to the relevant bit of the DMA_EICR Register to clear the interrupt request. For a terminal count interrupt, write a 1 to the relevant bit of the DMA_TCICR Register.

9.4.10 DMAC data flow

This section describes the DMAC data flow sequences for:

- Memory-to-memory DMA flow
- Memory-to-peripheral, or peripheral-to-memory DMA flow
- Peripheral-to-peripheral DMA flow

Memory-to-memory DMA flow

For a memory-to-memory DMA flow:

1. Program and enable the DMA channel.
2. Transfer data whenever the DMA channel has the highest pending priority and the DMAC gains bus master ship of the AHB bus.
3. If an error occurs while transferring the data, generate an error interrupt and disable the DMA stream.
4. Decrement the transfer count.
5. If the count has reached zero:
   a) Generate a terminal count interrupt (you can mask the interrupt).
   b) If the DMA_LLIx Register is not 0, then reload the DMA_SRCx, DMA_DESTx, DMA_LLIx, and DMA_CCRx Registers and go to back to step 2. However, if DMA_LLIx is 0, the DMA stream is disabled and the flow sequence ends.

Memory-to-peripheral, or peripheral-to-memory DMA flow

For a peripheral-to-memory or memory-to-peripheral DMA flow:

1. Program and enable the DMA channel.
2. Wait for a DMA request.
3. The DMAC then starts transferring data when:
   a) The DMA request goes active.
   b) The DMA stream has the highest pending priority.
   c) The DMAC is the bus master of the AHB bus.
4. If an error occurs while transferring the data, an error interrupt is generated and the DMA stream is disabled, and the flow sequence ends.
5. Decrement the transfer count if the DMAC is controlling the flow control.
6. If the transfer has completed (indicated by the transfer count reaching 0 if the DMAC is performing flow control, or by the peripheral setting the DMA request signals if the peripheral is performing flow control):
   a) The DMAC asserts the DMA Terminal Count signal to the peripheral indicating that the transfer is complete and the packet of data is transferred.
   b) The terminal count interrupt is generated (you can mask this interrupt).
   c) If the DMA_LLIx Register is not 0, then reload the DMA_SRCx, DMA_DESTx, DMA_LLIx, and DMA_CCRx Registers and go to back to step 2. However, if DMA_LLIx is 0, the DMA stream is disabled and the flow sequence ends.

Peripheral-to-peripheral DMA flow

For a peripheral-to-peripheral DMA flow:

1. Program and enable the DMA channel.
2. Wait for a source DMA request.
3. The DMAC then starts transferring data when:
   a) The DMA request goes active.
   b) The DMA stream has the highest pending priority.
   c) The DMAC is the bus master of the AHB bus.
4. If an error occurs while transferring the data, an error interrupt is generated, then finishes.
5. Decrement the transfer count if the DMAC is controlling the flow control.
6. If the transfer has completed (indicated by the transfer count reaching 0 if the DMAC is performing flow control, or by the peripheral setting the DMA request signals if the peripheral is performing flow control):
   a) The DMAC asserts the the DMA Terminal Count signal to the source peripheral indicating that the transfer is complete and the packet of data is transferred
   b) Subsequent source DMA requests are ignored.
7. When the destination DMA request goes active and there is data in the DMAC FIFO, transfer data into the destination peripheral.
8. If an error occurs while transferring the data, an error interrupt is generated and the DMA stream is disabled, and the flow sequence ends.
9. If the transfer has completed, it is indicated by the transfer count reaching 0 if the DMAC is performing flow control, or by the peripheral setting the DMA request signals if the peripheral is performing flow control. The following happens:
   a) The DMAC asserts the DMA Terminal Count signal to the destination peripheral indicating that the transfer is complete and the packet of data is transferred.
   b) The Terminal Count interrupt is generated (you can mask this interrupt).
   c) If the DMA_LLIx Register is not 0, then reload the DMA_SRCx, DMA_DESTx, DMA_LLIx, and DMA_CCRx Registers and go to back to step 2. However, if DMA_LLIx is 0, the DMA stream is disabled and the flow sequence ends.

9.5 Register description

The DMA registers are accessed via the APB bus and the register data path is 32 bits wide.

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 (wo)	Software can only write to this bit. Reading the bit returns the reset value

The following applies to the registers used in the DMAC:

- You must not access reserved or unused address locations because this can result in unpredictable behavior of the device.
- You must write reserved or unused bits of registers as zero, and ignore them on read unless otherwise stated in the relevant text.
- A system or power-on reset resets all registers bits to a logic 0 unless otherwise stated in the relevant text.
- All registers support read/write accesses unless otherwise stated in the relevant text. A write updates the contents of a register and a read returns the contents of the register.
- You can only access registers defined in this document using word reads and word writes, unless otherwise stated in the relevant text.

9.5.1 Common registers

Interrupt status register (DMA_ISR)

Address offset: 000h

Reset value: 0000 0000h

7	6	5	4	3	2	1	0
IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
r	r	r	r	r	r	r	r

Bits 31:8	Reserved, always read as 0
Bits 7:0	**IS[7:0]** *Interrupt Status after masking* 0: No interrupt or interrupt masked on the corresponding DMA channel 1: Interrupt requested by the corresponding DMA channel

Terminal count interrupt status register (DMA_TCISR)

Address offset: 004h

Reset value: 0000 0000h

7	6	5	4	3	2	1	0
TCS7	TCS6	TCS5	TCS4	TCS3	TCS2	TCS1	TCS0
r	r	r	r	r	r	r	r

Bits 31:8	Reserved, always read as 0
Bits 7:0	**TCS[7:0]** *Terminal Count interrupt Status after masking* This register can be read to determine the source of a DMA interrupt request. A Terminal Count event occurs when a DMA transfer is complete and the transfer counter reaches 0. 0: No Terminal Count interrupt on the corresponding DMA channel 1: Terminal Count Interrupt requested by the corresponding DMA channel.

Terminal count interrupt clear register (DMA_TCICR)

Address offset: 008h

Reset value: xxxx xxxxh

7	6	5	4	3	2	1	0
TCC7	TCC6	TCC5	TCC4	TCC3	TCC2	TCC1	TCC0
wo	wo	wo	wo	wo	wo	wo	wo

Bits 31:8	Reserved, write as 0
Bits 7:0	**TCS[7:0]** *Terminal Count interrupt clear* This register can be written to clear any Terminal Count interrupt requests. 0: No effect 1: Clear a Terminal Count Interrupt on the corresponding DMA channel

Error interrupt status register (DMA_EISR)

Address offset: 00Ch

Reset value: 0000 0000h

7	6	5	4	3	2	1	0
ES7	ES6	ES5	ES4	ES3	ES2	ES1	ES0
r	r	r	r	r	r	r	r

Bits 31:8	Reserved, always read as 0
Bits 7:0	**ES[7:0]** *Error interrupt Status after masking* This register can be read to determine the source of a DMA interrupt request. 0: No Error interrupt on the corresponding DMA channel 1: Error Interrupt requested by the corresponding DMA channel

Error interrupt clear register (DMA_EICR)

Address offset: 010h

Reset value: xxxx xxxxh

7	6	5	4	3	2	1	0
EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0
wo	wo	wo	wo	wo	wo	wo	wo

Bits 31:8	Reserved, write as 0
Bits 7:0	**ES[7:0]** *Error interrupt Clear* This register can be written to clear any Error interrupt requests. 0: No effect 1: Clear an Error Interrupt on the corresponding DMA channel

Terminal count raw interrupt status register (DMA_TCRISR)

Address offset: 014h

Reset value: xxxx xxxxh

7	6	5	4	3	2	1	0
TCRS7	TCRS6	TCRS5	TCRS4	TCRS3	TCRS2	TCRS1	TCRS0
r	r	r	r	r	r	r	r

Bits 31:8	Reserved, always read as 0
Bits 7:0	**TCRS[7:0]** *Terminal Count raw interrupt status (before masking)* This register can be read to determine the Terminal Count status of a channel. A Terminal Count event occurs when a DMA transfer is complete and the transfer counter reaches 0. 0: No Terminal Count event on the corresponding DMA channel 1: Terminal Count event occured on the corresponding DMA channel

Error raw interrupt status register (DMA_ERISR)

Address offset: 018h

Reset value: xxxx xxxxh

7	6	5	4	3	2	1	0
ERS7	ERS6	ERS5	ERS4	ERS3	ERS2	ERS1	ERS0
r	r	r	r	r	r	r	r

Bits 31:8	Reserved, always read as 0
Bits 7:0	**ERS[7:0]** *Error raw interrupt status (before masking)* This register can be read to determine the Error status of a channel. 0: No Error event on the corresponding DMA channel 1: Error event occured on the corresponding DMA channel

Enabled channel status register (DMA_ENCSR)

Address offset: 01Ch

Reset value: 0000 0000h

7	6	5	4	3	2	1	0
ENCS7	ENCS6	ENCS5	ENCS4	ENCS3	ENCS2	ENCS1	ENCS0
r	r	r	r	r	r	r	r

Bits 31:8	Reserved, always read as 0
Bits 7:0	**ENCS[7:0]** *Enabled Channel status* You can read this register to determine the Enabled status of any channel. You can enabled or disable the channels by writing to the DMA_CNFR register. 0: The corresponding DMA channel is disabled 1: The corresponding DMA channel is enabled

Software burst request register (DMA_SBRR)

Address offset: 020h

Reset value: 0000 0000h

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

Bits 31:16	Reserved, always write as 0
Bits 15:0	**SBR[15:0]** *Software Burst Request for Source x* You can generate a DMA request for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Writing 0 to this register has no effect. Reading the register indicates the sources that are requesting DMA burst transfers. You can generate a request from either a peripheral or the software request register. 0: No effect 1: Generate burst transfer request for the corresponding DMA Request Source. ***Note:*** It is recommended not to use software and hardware peripheral requests the same time.

Software single request register (DMA_SSRR)

Address offset: 024h

Reset value: 0000 0000h

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

Bits 31:16	Reserved, always write as 0
Bits 15:0	**SSR[15:0]** *Software Single Request for Source x* You can generate a DMA request for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Writing 0 to this register has no effect. Reading the register indicates the sources that are requesting DMA single transfers. You can generate a request from either a peripheral or the software request register. 0: No effect 1: Generate single transfer request for the corresponding DMA Request Source. ***Note:*** It is recommended not to use software and hardware peripheral requests the same time.

Software last burst request register (DMA_SLBR)

Address offset: 028h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLB15	SLB14	SLB13	SLB12	SLB11	SLB10	SLB 9	SLB 8	SLB 7	SLB 6	SLB 5	SLB 4	SLB 3	SLB 2	SLB 1	SLB 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, always write as 0
Bits 15:0	**SLB[15:0]** *Software Last Burst Request for Source x* You can generate a DMA request for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Writing 0 to this register has no effect. Reading the register indicates the sources that are requesting DMA last burst transfers. You can generate a request from either a peripheral or the software request register. 0: No effect 1: Generate last burst transfer request for the corresponding DMA Request Source. ***Note:*** It is recommended not to use software and hardware peripheral requests the same time.

Software last single request register (DMA_SLSR)

Address offset: 02Ch

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLS15	SLS14	SLS13	SLS12	SLS11	SLS10	SLS9	SLS8	SLS7	SLS6	SLS5	SLS4	SLS3	SLS2	SLS1	SLS0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, always write as 0
Bits 15:0	**SLS[15:0]** *Software Last Single Request for Source x* You can generate a DMA request for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Writing 0 to this register has no effect. Reading the register indicates the sources that are requesting DMA last single transfers. You can generate a request from either a peripheral or the software request register. 0: No effect 1: Generate last single transfer request for the corresponding DMA Request Source. ***Note:*** It is recommended not to use software and hardware peripheral requests the same time.

Configuration register (DMA_CNFR)

Address offset: 030h

Reset value: 0000 0000h

7	6	5	4	3	2	1	0
Reserved							EN
							rw

Bits 31:1	Reserved, always read as 0
Bit 0	**EN:** *DMA Controller Enable* 0: DMA controller disabled 1: DMA controller enabled

Synchronization register (DMA_SYNC)

Address offset: 034h

Reset value: 0000 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SYNC15	SYNC14	SYNC13	SYNC12	SYNC11	SYNC10	SYNC 9	SYNC 8	SYNC7	SYNC6	SYNC5	SYNC4	SYNC3	SYNC2	SYNC 1	SYNC 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, always write as 0
Bits 15:0	**SYNC[15:0]** *Synchronization Enable/Disable* These bits are set and cleared by software. You must use synchronization logic when the peripheral generating the DMA request runs on a different clock to the DMAC. For peripherals running on the same clock as the DMAC, disabling the synchronization logic improves the DMA request response time. 0: Enable synchronization logic for the corresponding DMA Request Signal 1: Disable synchronization logic for the corresponding DMA Request Signal. ***Note:*** All DMA requests must be synchronized with the exception of the USB DMA Request.

9.5.2 Channel registers

The channel registers are for programming a DMA channel. These registers consist of:

- Eight DMA_SRCx Channel Source Address Registers
- Eight DMA_DESTx Channel Destination Address Registers
- Eight DMA_LLIx Channel Linked List Registers
- Eight DMA_CCx Channel Control Registers
- Eight DMA_CCNFx Channel Configuration Registers.

When performing scatter/gather DMA, the first four registers are automatically updated.

Note: Unpredictable behavior can result if you update the channel registers when a transfer is taking place. If you want to change the channel configurations, you must disable the channel first and then reconfigure the relevant register.

Channel source address register x (DMA_SRCx)

Address offset: See *Table 24*

Reset value: 0000 0000h

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

Bits 31:0	**SrcAddr[31:0]** *DMA Source Address for channel x* These bits contain the current source address (byte aligned) of the data to be transferred Software programs each register directly before the appropriate channel is enabled. When the DMA channel is enabled, this register is updated: - As the source address is incremented - By following the linked list when a complete packet of data has been transferred. Reading the register when the channel is active does not provide useful information. This is because by the time the software has processed the value read, the channel might have progressed. It is intended to be read-only when the channel has stopped, and in such case, it shows the source address of the last item read. ***Note:*** You must align source and destination addresses to the source and destination widths.

Channel destination address register x (DMA_DESTx)

Address offset: See *Table 24*

Reset value: 0000 0000h

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

Bits 31:0	**SrcAddr[31:0]** *DMA Destination Address for channel x* These bits contain the current destination address (byte aligned) of the data to be transferred Software programs each register directly before the appropriate channel is enabled. When the DMA channel is enabled, this register is updated: - As the destination address is incremented - By following the linked list when a complete packet of data has been transferred. Reading the register when the channel is active does not provide useful information. This is because by the time the software has processed the value read, the channel might have progressed. It is intended to be read-only when the channel has stopped, and in such case, it shows the destination address of the last item read. ***Note:*** You must align source and destination addresses to the source and destination widths.

Channel linked list item register x (DMA_LLIx)

Address offset: See *Table 24*

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LLI[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
LLI[15:2]														0	0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0	**LLI[31:2]** *Linked List Item for channel x* These bits contain the word aligned address of the next LLI to be transferred. Address bits 1:0 are 0. If the LLI is 0, then the current LLI is the last in the chain, and the DMA channel is disabled after all DMA transfers associated with it are completed. ***Note:*** Programming this register when the DMA channel is enabled has unpredictable results.
Bits 1:0	Reserved, always write as 0

Channel control register x (DMA_CCx)

Address offset: See *Table 24*

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TCIE	PROT2	PROT1	PROT0	DI	SI	0	0	DWIDTH			SWIDTH			DBSize[2:1]	
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
DBSize0	SBSize[2:0]			TransferSize[11:0]											
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		

Note: 1 This register contains DMA channel control information such as the transfer size, burst size, and transfer width. Software programs each register directly before the DMA channel is enabled.

2 When the channel is enabled, the register is updated by following the linked list when a complete packet of data has been transferred. Reading the register while the channel is active does not give useful information. This is because by the time that software has processed the value read, the channel might have progressed. It is intended to be read-only when a channel has stopped.

Bit 31	**TCIE** *Terminal Count Interrupt enable* This bit controls whether the current LLI is expected to trigger the terminal count interrupt. 0: Terminal Count interrupt disabled 1: Terminal Count interrupt enabled
Bit 30	**PROT2** *Cacheable/Noncacheable* This bit indicates whether or not the access is cacheable. For example, you can use this bit to indicate to the hardware that it can transfer the whole burst of eight reads on the destination bus, rather than pass the transactions through one at a time. 0: Noncacheable 1: Cacheable
Bit 29	**PROT1** *Bufferable/Nonbufferable* This bit indicates whether or not the access is bufferable. For example, you can use this bit to indicate that the write can complete in zero wait states on the source bus without waiting for arbitration of the destination bus and for the slave to accept the data. 0: Nonbufferable 1: Bufferable
Bit 28	**PROT0** *Privileged/user mode protection* This bit controls if the access is in privileged or user mode. 0: User mode 1: Privileged mode

Bit 27	**DI** *Destination increment* This bit is set and cleared by software. 0: Destination address not incremented 1: The destination address is incremented after each transfer
Bit 26	**SI** *Source increment* This bit is set and cleared by software. 0: Source address not incremented 1: The source address is incremented after each transfer
Bits 25:24	Reserved, always write as 0
Bits 23:21	**DWIDTH** *Destination width* These bits indicate the data width of the destination peripheral or memory. The source and destination widths can be different from each other. The hardware automatically packs and unpacks the data when required 000: Byte (8-bit) 001: Halfword (16-bit) 010: Word (32-bit) Other values: Reserved
Bits 20:18	**SWIDTH** *Source width* These bits indicate the data width of the source peripheral or memory. The source and destination widths can be different from each other. The hardware automatically packs and unpacks the data when required 000: Byte (8-bit) 001: Halfword (16-bit) 010: Word (32-bit) Other values: Reserved
Bits 17:15	**DBSize[2:0]** *Destination Burst size* These bits indicate the number of transfers that make up a destination burst transfer request. You must set this value to the burst size of the destination peripheral, or if the destination is memory, to the memory boundary size. 000: single transfer 001: 4 transfers 010: 8 transfers 011: 16 transfers 100: 32 transfers 101: 64 transfers 110: 128 transfers 111: 256 transfers
Bits 14:12	**SBSize[2:0]** *Source Burst size* These bits indicate the number of transfers that make up a source burst. You must set this value to the burst size of the source peripheral, or if the source is memory, to the memory boundary size. 000: single transfer 001: 4 transfers 010: 8 transfers 011: 16 transfers 100: 32 transfers 101: 64 transfers 110: 128 transfers 111: 256 transfers

Bits 11:0	**TransferSize[11:0]** *Transfer size* A write to this field sets the size of the transfer when the DMAC is the flow controller. A read from this field indicates the number of transfers completed on the destination bus. Reading the register when the channel is active does not give useful information because by the time the software has processed the value read, the channel might have progressed. You should only use it when a channel is enabled, and then disabled. Program the transfer size value to zero if the DMAC is not the flow controller. If you program the TransferSize to a non-zero value, the DMAC might attempt to use this value instead of ignoring the TransferSize.

Channel configuration register x (DMA_CCNFx)

Address offset: See *Table 24*

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													H	A	L
													rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ITC	IE	FlowCntrl[2:0]			Res.	DestPeripheral				Res.	SrcPeripheral				E
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Note: The channel configuration registers are not updated when a new LLI is requested.

Bits 31:19	Reserved, always write as 0
Bit 18	**H** *Halt* You can use this value with the Active and Channel Enable bits to cleanly disable a DMA channel. 0: DMA requests enabled on channel x 1: Any extra source DMA requests are ignored. The contents of the channel x FIFO are drained.
Bit 17	**A** *Active* You can use this value with the Halt and Channel Enable bits to cleanly disable a DMA channel. 0: No data in the channel x FIFO 1: The channel x FIFO has data.
Bit 16	**L** *Lock* 0: Locked transfers disabled on channel x 1: Locked transfers enabled channel x
Bit 15	**ITC** *Terminal Count Interrupt Mask* 0: Terminal Count interrupt disabled on channel x 1: Terminal Count interrupt enabled on channel x
Bit 14	**IE** *Error Interrupt Mask* 0: Error interrupt disabled on channel x 1: Error interrupt enabled on channel x

Bits 13:11	**FlowCntrl[2:0]** *Flow controller and transfer type* This value indicates the flow controller and transfer type. **Transfer Type Flow Controller** 000: Memory-to-memory DMA 001:Memory-to-peripheral DMA 010: Peripheral-to-memory DMA 011: Source peripheral-to-destination peripheral DMA 100: Source peripheral-to-destination peripheral Destination peripheral 101: Memory-to-peripheral Peripheral 110: Peripheral-to-memory Peripheral 111: Source peripheral-to-destination peripheral Source peripheral
Bit 10	Reserved, always write as 0
Bits 9:6	**DestPeripheral** *Destination peripheral selection* This value selects the DMA destination request peripheral. Refer to *Table 23.* This field is ignored if the destination of the transfer is to memory.
Bit 5	Reserved, always write as 0
Bits 4:1	**SrcPeripheral** *Source peripheral selection* This value selects the DMA source request peripheral. Refer to *Table 23.* This field is ignored if the source of the transfer is from memory.
Bit 0	**E** *Channel enable* Reading this bit indicates whether a channel is currently enabled or disabled. You can also determine the Channel Enable bit status by reading the DMA_ECSR register. You enable a channel by setting the E bit. You can disable a channel by clearing the E bit. This causes the current AHB transfer (if one is in progress) to complete, and the channel is then disabled. Any data in the channel's FIFO is lost. Restarting the channel by setting the Channel Enable bit has unpredictable effects and you must fully re-initialize the channel. The channel is also disabled, and the E bit cleared, when the last LLI is reached, or if a channel error is encountered. If a channel has to be disabled without losing data in a channel's FIFO, you must set the H (HALT) bit so that subsequent DMA requests are ignored. The A (Active) bit must then be polled until it reaches 0, indicating that there is no data left in the channel's FIFO. Finally, you can clear the Channel Enable bit. 0: Channel disabled 1: Channel enabled

9.6 DMA register map

The following table summarizes the DMA registers:

Table 24. DMA 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	DMA_ISR																									Interrupt Status							
04h	DMA_TCISR																									Terminal Count Interrupt Status							
08h	DMA_TCICR																									Terminal Count Interrupt Clear							
0Ch	DMA_EISR																									Error Interrupt Status							
10h	DMA_EICR																									Error Interrupt Clear							
14h	DMA_TCRISR																									Terminal Count Raw Interrupt Status							
18h	DMA_ERISR																									Error Raw Interrupt Status							
1Ch	DMA_ENCSR																									Enabled Channel Status							
20h	DMA_SBRR																	Software Burst Request															
24h	DMA_SSRR																	Software Single Request															
28h	DMA_SLBR																	Software Last Burst Request															
2Ch	DMA_SLSR																	Software Last Single Request															
30h	DMA_CNFR	Configuration																															EN
34h	DMA_SYNC																	Synchronization															
100h	DMA_SRC0	Channel 0 Source Address																															
104h	DMA_DEST0	Channel 0 Destination Address																															
108h	DMA_LLI0	Channel 0 Linked List Item																														0	0
10Ch	DMA_CC0	Channel 0 Control																															
110h	DMA_CCNF0	Channel 0 Configuration																															
120h	DMA_SRC1	Channel 1 Source Address																															
124h	DMA_DEST1	Channel 1 Destination Address																															
128h	DMA_LLI1	Channel 1 Linked List Item																														0	0
12Ch	DMA_CC1	Channel 1 Control																															
130h	DMA_CCNF1	Channel 1 Configuration																															
140h	DMA_SRC2	Channel 2 Source Address																															
144h	DMA_DEST2	Channel 2 Destination Address																															
148h	DMA_LLI2	Channel 2 Linked List Item																														0	0
14Ch	DMA_CC2	Channel 2 Control																															
150h	DMA_CCNF2	Channel 2 Configuration																															
160h	DMA_SRC3	Channel 3 Source Address																															
164h	DMA_DEST3	Channel 3 Destination Address																															
168h	DMA_LLI3	Channel 3 Linked List Item																														0	0
16Ch	DMA_CC3	Channel 3 Control																															
170h	DMA_CCNF3	Channel 3 Configuration																															
180h	DMA_SRC4	Channel 4 Source Address																															
184h	DMA_DEST4	Channel 4 Destination Address																															
188h	DMA_LLI4	Channel 4 Linked List Item																														0	0

Table 24. DMA 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
18Ch	DMA_CC4	Channel 4 Control																															
190h	DMA_CCNF4	Channel 4 Configuration																															
1A0h	DMA_SRC5	Channel 5 Source Address																															
1A4h	DMA_DEST5	Channel 5 Destination Address																															
1A8h	DMA_LLI5	Channel 5 Linked List Item																														0	0
1ACh	DMA_CC5	Channel 5 Control																															
1B0h	DMA_CCNF5	Channel 5 Configuration																															
1C0h	DMA_SRC6	Channel 6 Source Address																															
1C4h	DMA_DEST6	Channel 6 Destination Address																															
1C8h	DMA_LLI6	Channel 6 Linked List Item																														0	0
1CCh	DMA_CC6	Channel 6 Control																															
1D0h	DMA_CCNF6	Channel 6 Configuration																															
1E0h	DMA_SRC7	Channel 7 Source Address																															
1E4h	DMA_DEST7	Channel 7 Destination Address																															
1E8h	DMA_LLI7	Channel 7 Linked List Item																														0	0
1ECh	DMA_CC7	Channel 7 Control																															
1F0h	DMA_CCNF7	Channel 7 Configuration																															

Refer to *Table 5 on page 35* for the register base addresses.

10 Synchronous serial peripheral (SSP)

10.1 Introduction

The SSP is a master or slave interface for synchronous serial communication with peripheral devices that have either Motorola SPI, National Microwire 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 SCLK from the input clock BRCLK.

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 period expired while data was present in the receive FIFO.

The above interrupts are ORed to generate a single interrupt to the Vectored Interrupt Controller (VIC).

In addition to the above interrupts, a set of DMA signals are provided for interfacing with a DMA controller.

10.2 Main features

- Master and slave modes supported
- Programmable choice of interface operation: Motorola SPI, National Microwire 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
- Support for DMA
- Independent masking of transmit FIFO, receive FIFO, and overrun interrupts
- Internal loopback test mode available.
- Dynamic change from master to slave or slave to master operations

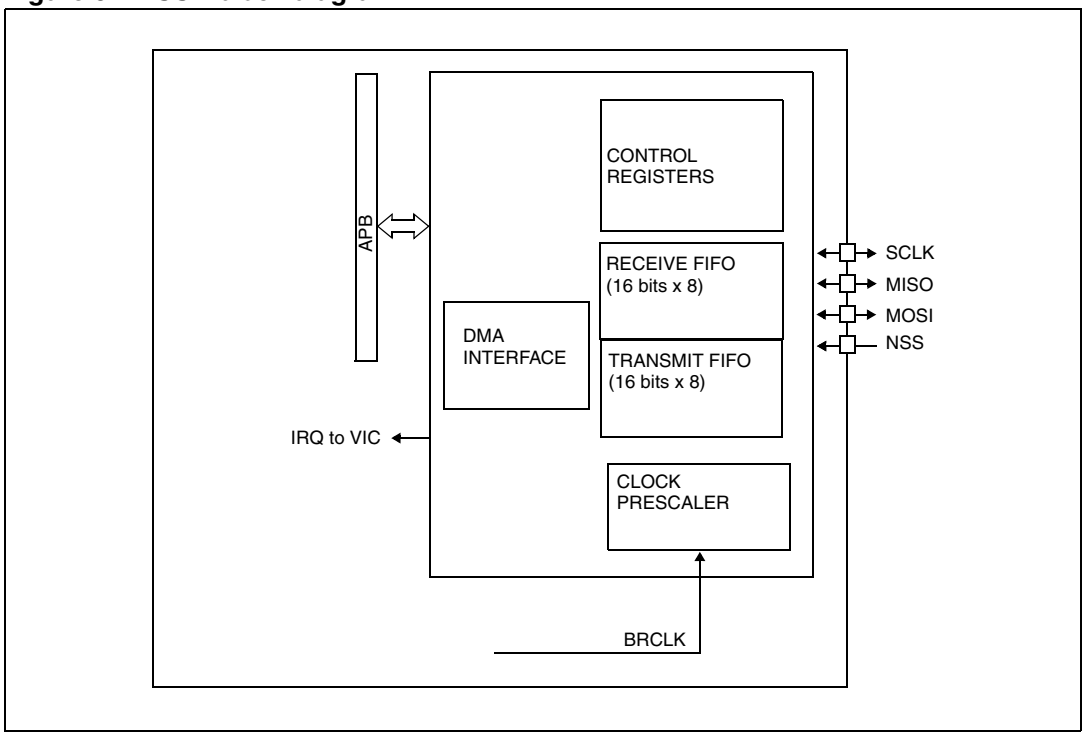
10.3 Functional description

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 transferred to/from memory using the DMA

A block diagram of the SSP is shown in *Figure 67*.

Figure 67. SSP block diagram



10.3.1 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 4 I/O pins, see table below:

- MISO: Master In Slave Out pin
- MOSI: Master Out Slave In pin
- SCLK: Serial Clock pin
- NSS: Slave Select pin

Table 25. SSP pins

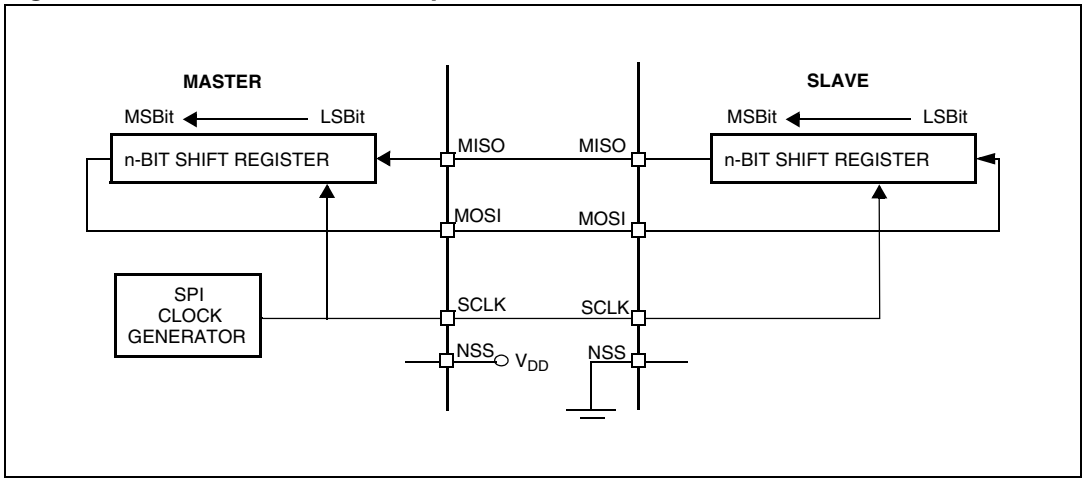
Pin Name	Description
SCLK	Serial Clock pin: The bit clock for all data transfers. When the SSP is a master the SCLK is output from the chip. When configured as a slave the SCLK is input from the external source.
MISO	Master Input/ Slave Output serial data line
MOSI	Master Output/ Slave Input serial data line
NSS	Slave Select pin: The NSS input pin is used to select a slave device. Must be pulled low after the SCLK is stable and held low for the duration of the data transfer. The NSS on the master must be deasserted high.

A basic example of interconnections between a single master and a single slave is illustrated in *Figure 68*.

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 SCLK pin).

Figure 68. Interconnection example



10.3.2 Master mode

When configured as a master, the clock to the attached slaves is derived from a divided down version of BRCLK 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 SCLK, 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 in its receive FIFO, for subsequent reading through the APB interface.

10.3.3 Slave mode

When configured as a slave, the SCLK 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 outputs 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 in its receive FIFO, for subsequent reading through the APB interface.

10.3.4 Slave Select management

The NSS Slave Select signal is output from the Master and received by the Slave.

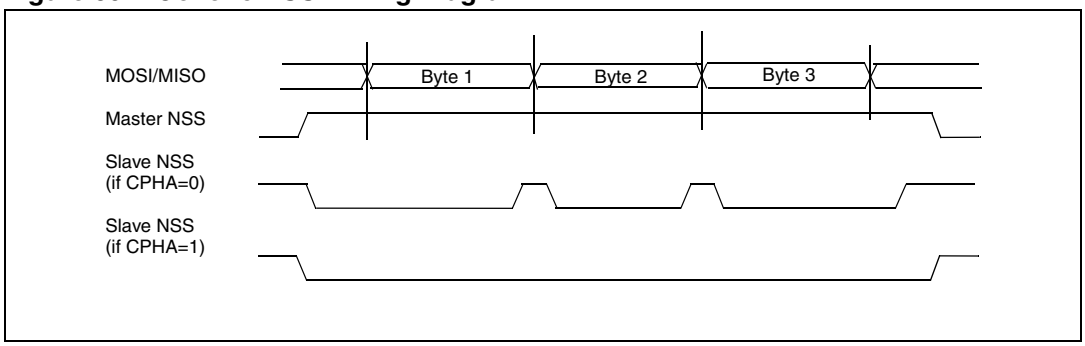
There are two cases depending on the data/clock timing relationship (see *Figure 69*):

- If CPHA=1 (data latched on 2nd clock edge):

  NSS must be held low during the entire transmission. This implies that in single slave applications the NSS pin can be tied to V_{SS}.
- If CPHA=0 (data latched on 1st clock edge):

  NSS must be held low during byte transmission and pulled high between each byte to allow the slave to write to the FIFO register.

Figure 69. Generic NSS Timing Diagram



10.4 SSP operation

The operation of the SSP is described in the following sections

10.4.1 Configuring the SSP

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
- National Semiconductor Microwire

The bit rate, derived from BRCLK, requires the programming of the clock prescaler register SSP_PR.

10.4.2 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.

10.4.3 Programming the SSP_CR0 control register

The SSP_CR0 register is used to:

- Program the serial clock rate
- Select one of the three protocols
- Select the data word size (where applicable)

The Serial Clock Rate (SCR) value, in conjunction with the SSP_PR clock prescale divisor value (CPSDVSR), is used to derive the SSP transmit and receive bit rate from the external SCLK.

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.

10.4.4 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.

10.4.5 Clock ratios

There is a constraint on the ratio of the frequencies of PCLK and SSPCLK (BRCLK). The frequency of SSPCLK must be less than or equal to that of PCLK. This ensures that control signals from the BRCLK domain to the PCLK domain are certain to get synchronized before on frame duration.

$f_{BRCLK} <= f_{PCLK}$

There is another constraint on the ratio of the frequencies of BRCLK to SCLK. To ensure correct device operation, BRCLK must be at least 12 times faster than the maximum expected frequency of SCLK in slave mode, and at least 2 times faster than the maximum expected frequency of SCLK in master mode.

To generate a maximum bit rate of 1.8432 Mbps in Master mode, the frequency of BRCLK must be at least 3.6864 MHz. With a BRCLK frequency of 3.6864 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 1.8432 Mbps in the slave mode, the frequency of BRCLK must be at least 22.12 MHz. With an BRCLK frequency of 22.12 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 BRCLK maximum frequency to SCLK minimum frequency is 254 x 256.

The minimum frequency of BRCLK is governed by the following equations, both of which have to be satisfied:

- $f_{BRCLK(min)} => 2 \times f_{SCLK(max)}$ [for master mode]
- $f_{BRCLK(min)} => 12 \times f_{SCLK(max)}$ [for slave mode]

The maximum frequency of BRCLK is governed by the following equations, both of which have to be satisfied:

- $f_{BRCLK(max)} <= 254 \times 256 \times f_{SCLK(min)}$ [for master mode]
- $f_{BRCLK(max)} <= 254 \times 256 \times f_{SCLK(min)}$ [for slave mode]

10.4.6 Bit rate generation

The serial bit rate is derived by dividing down the input clock BRCLK. The clock is first divided by an even prescale value CPSDVSR from 2 to 254, which is programmed in SSP_PR. The clock is further divided by a value from 1 to 256, which is 1 + SCR, where SCR is the value programmed in SSP_CR0.

The frequency of the output signal bit clock SCLK is defined below:

$f_{SCLK} = f_{BRCLK} / (CPSDVR * (1+SCR))$

For example, if BRCLK is 3.6864 MHz, and CPSDVSR = 2, then SCLK has a frequency range from 7.2 kHz to 1.8432 MHz.

10.4.7 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
- National Semiconductor Microwire

For all three formats, the serial clock (SCLK) 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 SCLK is utilized to provide a receive timeout indication that occurs when the receive FIFO still contains data after a timeout period.

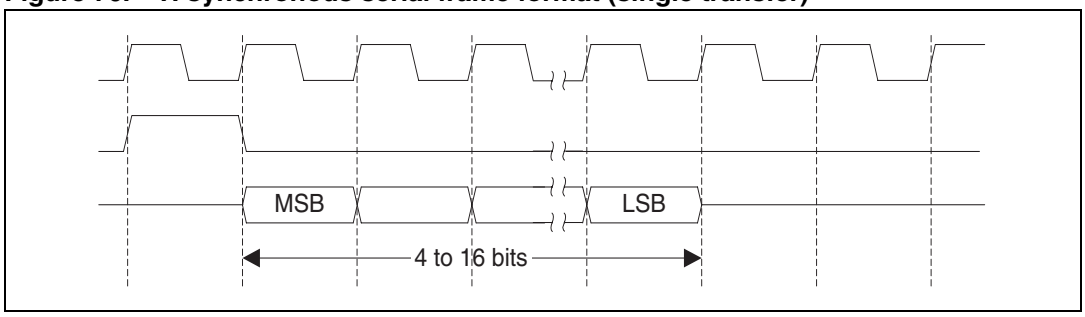
For Motorola SPI and National Semiconductor Microwire formats, the slave select (NSS) pin is active LOW, and is asserted (pulled down) during the entire transmission of the frame.

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 SCLK, and latch data from the other device on the falling edge.

Texas Instruments synchronous serial frame format

Figure 70 shows the Texas Instruments synchronous serial frame format for a single transmitted frame.

Figure 70. TI synchronous serial frame format (single transfer)

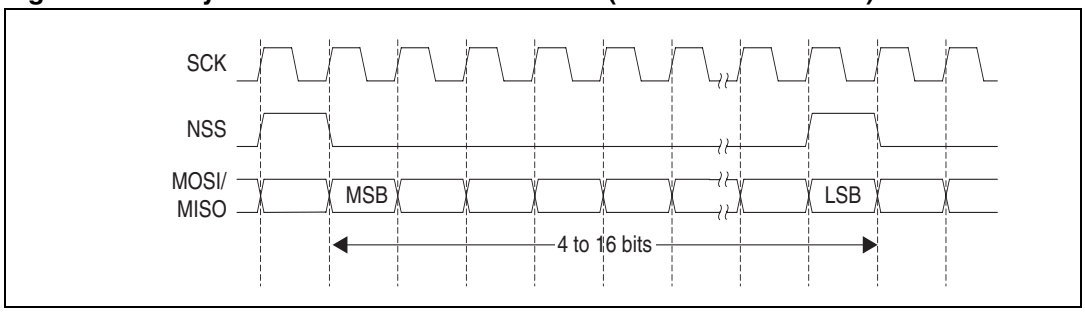


In this mode, SCLK and NSS are forced LOW, and the transmit data line is tristated whenever the SSP is idle. Once the bottom entry of the transmit FIFO contains data, NSS is pulsed HIGH for one SCLK period. The value to be transmitted is also transferred from the transmit FIFO to the serial shift register of the transmit logic. On the next rising edge of SCLK, 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 SCLK. The received data is transferred from the serial shifter to the receive FIFO on the first rising edge of SCLK after the LSB has been latched.

Figure 71 shows the Texas Instruments synchronous serial frame format when back-to-back frames are transmitted.

Figure 71. TI synchronous serial frame format (continuous transfer)



Note: When configured in TI slave mode, the internal SSPx SCLK clock rate must be configured (using SSPx_CRO and SSPx_PR registers) to match the SCLK clock rate of the external master. 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, a slave can connect to a master without knowing the SPI clock rate.

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 SCLK 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 SCLK pin. If the CPOL clock polarity control bit is HIGH, a steady state high value is placed on the SCLK pin when data is not being transferred.

CPHA, clock phase

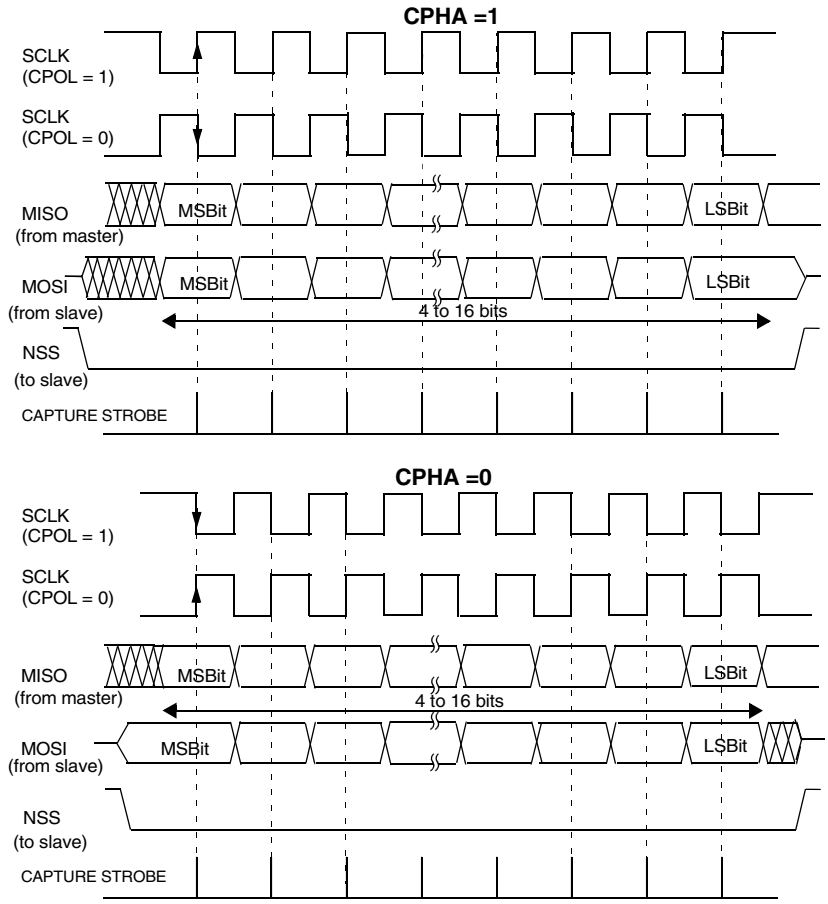
The CPHA control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge.

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 SCLK must correspond to the polarity selected in the SPICSR register (by pulling up SCLK if CPOL = 1 or pulling down SCLK if CPOL = 0).

Figure 72, 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 SCLK pin, the MISO pin, the MOSI pin are directly connected between the master and the slave device.

Figure 72. Motorola SPI frame format

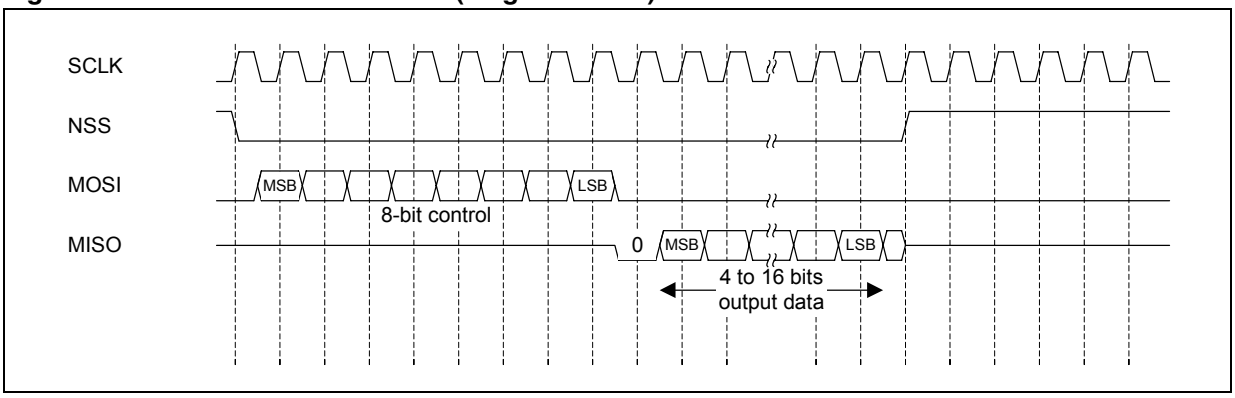


1. This figure should not be used as a replacement for parametric information.
2. Refer to the Electrical Characteristics chapter.

National Semiconductor Microwire frame format

Figure 73 shows the National Semiconductor Microwire frame format, again for a single frame. *Figure 74 on page 285* shows the same format when back to back frames are transmitted.

Figure 73. Microwire frame format (single transfer)



Microwire format is very similar to SPI format, except that transmission is half-duplex instead of full-duplex, using a master-slave message passing technique. Each serial transmission begins with an 8-bit control word that is transmitted from the SSP to the off-chip slave device. During this transmission, no incoming data is received by the SSP. After the message has been sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message has been sent, responds with the required data. The returned data is 4 to 16 bits in length, making the total frame length anywhere from 13 to 25 bits.

In this configuration, during idle periods:

- the SCLK signal is forced LOW
- NSS is forced HIGH
- the transmit data line MOSI/MISO is arbitrarily forced LOW

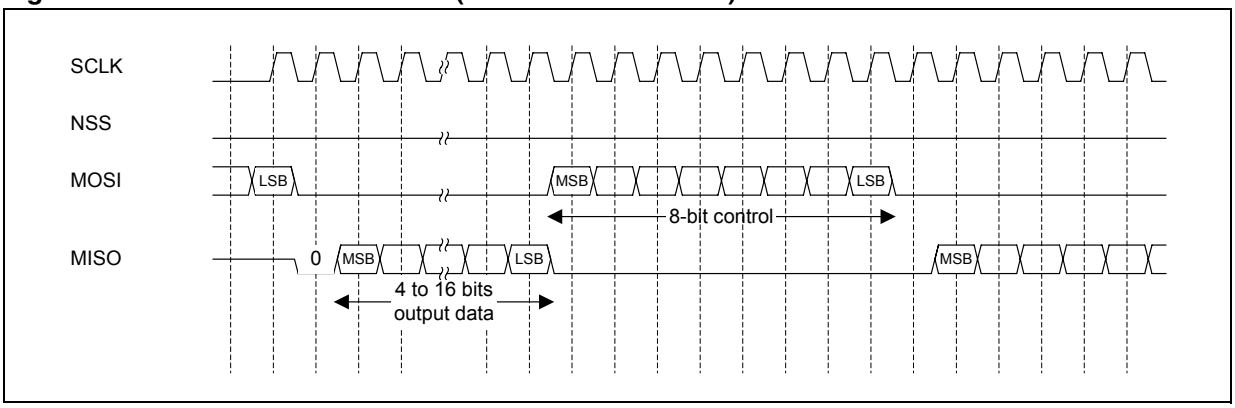
A transmission is triggered by writing a control byte to the transmit FIFO. The falling edge of NSS causes the value contained in the bottom entry of the transmit FIFO to be transferred to the serial shift register of the transmit logic, and the MSB of the 8-bit control frame to be shifted out onto the MOSI pin. NSS remains LOW for the duration of the frame transmission. The data input MISO pin remains tristated during this transmission.

The off-chip serial slave device latches each control bit into its serial shifter on the rising edge of each SCLK. After the last bit is latched by the slave device, the control byte is decoded during a one clock wait-state, and the slave responds by transmitting data back to the SSP. Each bit is driven onto MISO input line on the falling edge of SCLK. The SSP in turn latches each bit on the rising edge of SCLK. At the end of the frame, for single transfers, the NSS signal is pulled HIGH one clock period after the last bit has been latched in the receive serial shifter, that causes the data to be transferred to the receive FIFO.

The off-chip slave device can tristate the receive line either on the falling edge of SCLK after the LSB has been latched by the receive shifter, or when the NSS pin goes HIGH.

For continuous transfers, data transmission begins and ends in the same manner as a single transfer. However, the NSS line is continuously asserted (held LOW) and transmission of data occurs back to back. The control byte of the next frame follows directly after the LSB of the received data from the current frame. Each of the received values is transferred from the receive shifter on the falling edge SCLK, after the LSB of the frame has been latched into the SSP.

Figure 74. Microwire frame format (continuous transfers)



10.4.8 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).

10.4.9 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.

10.4.10 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 period 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-bit 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.

10.5 Register description

The registers can only accessed as 32-bit data. 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
Write-only (wo)	The software can only write to these bits

10.5.1 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 SCLK. The bit rate is f_{BRCLK}/(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 SCLK clock polarity (applicable only to Motorola SPI format). 0: SCLK is held low when no data is being transferred 1: SCLK 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 10: National Microwire frame format

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.

10.5.2 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												SOD	MS	SSE	LBM
												rw	rw	rw	rw

Bits 31:4	Reserved, must be written with 0
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.

10.5.3 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. When the SSP is programmed for National Microwire frame format, the default size for transmit data is eight bits (the most significant byte is ignored). The receive data size is controlled by the application program. 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.

10.5.4 Status register (SSP_SR)

Address offset: 0Ch

Reset value: 0000 0003h

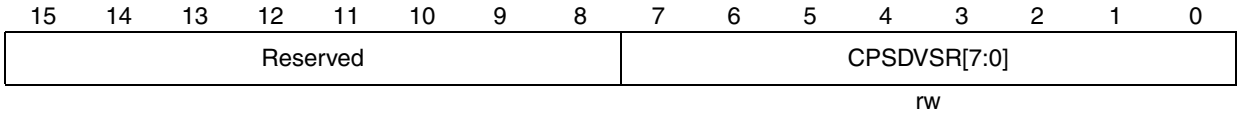
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved											BSY	RFF	RNE	TNF	TFE
											r	r	r	r	r

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

10.5.5 Clock prescaler register (SSP_PR)

Address offset: 10h

Reset value: 0000 0000h

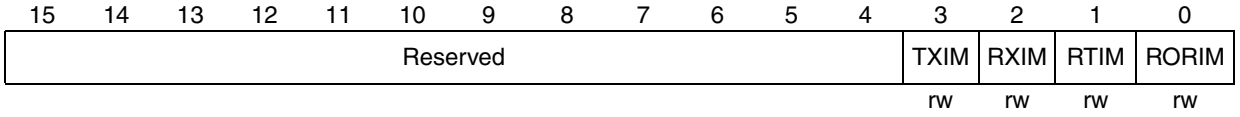


Bits 31:8	Reserved, must be written with 0
Bits 7:0	**CPSDVSR[7:0]:** *Clock prescaler divisor* These bits specify the division factor by which the input BRCLK must be divided for use by the SSP. The value written to this register must be an even number between 2 and 254. The least significant bit of 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.

10.5.6 Interrupt mask set and clear register (SSP_IMSCR)

Address offset: 14h

Reset value: 0000 0000h



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 period interrupt is masked 1: RxFIFO not empty and no read prior to timeout period 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

10.5.7 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 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)
Bit 0	**RORRIS**: *Receive overrun raw status flag* 0: No RX Overrun event occurred 1: A RX Overrun event occurred (prior to masking)

10.5.8 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)
Bit 0	**RORMIS**: *Receive overrun masked status flag* 0: No RX Overrun interrupt request (after masking) 1: RX Overrun interrupt request occurred (after masking)

10.5.9 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
														wo	wo

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

10.5.10 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														TXDMAE	RXDMAE
														rw	rw

Bits 31: 2	Reserved, must be written with 0
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

10.6 SSP register map

Table 26. SSP register map

Address offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	SSP_CR0	SCR[7:0]								CPHA	CPOL	FRF[1:0]		DSS[3:0]			
04h	SSP_CR1	Reserved												SOD	MS	SSE	LBM
08h	SSP_DR	Data Register															
0Ch	SSP_SR	Reserved											BSY	RFF	RNE	TNF	TFE
10h	SSP_PR	Reserved								Clock Prescaler Register							
14h	SSP_IMSCR	Reserved												TX IM	RX IM	RT IM	RORIM
18h	SSP_RISR	Reserved												TX RIS	RX RIS	RT RIS	RORRIS
1Ch	SSP_MISR	Reserved												TX MIS	RX MIS	RT MIS	RORMIS
20h	SSP_ICR	Reserved														RTIC	RORIC
24h	SSP_DMACR	Reserved														TXD MAE	RXDMAE
28h to 7Ch		Reserved															

Refer to *Table 5 on page 35* for the register base addresses.

11 Universal asynchronous receiver transmitter (UART)

11.1 Introduction

The UART interface provides serial communication between the STR91xF 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.

11.2 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 any clock with a frequency >3.6864 MHz as the reference clock.
- Standard asynchronous communication bits (start, stop and parity). These are added prior to transmission and removed on 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
- Full modem interface (on UART0 only)
- IrDA Mode
- 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 generation, dc up to BRCLK_max_freq/16

11.3 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 75 and Figure 76) 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 75. 8-bit data frames

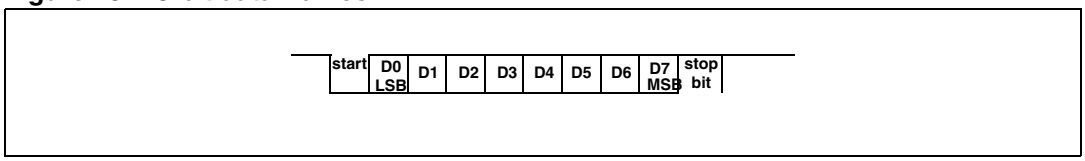
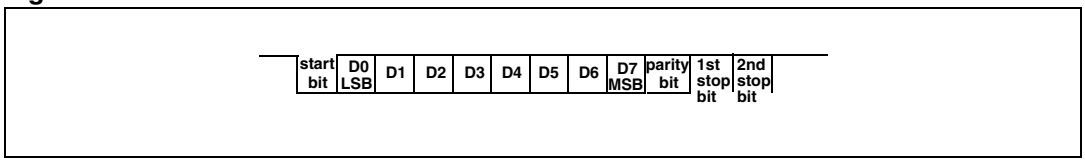


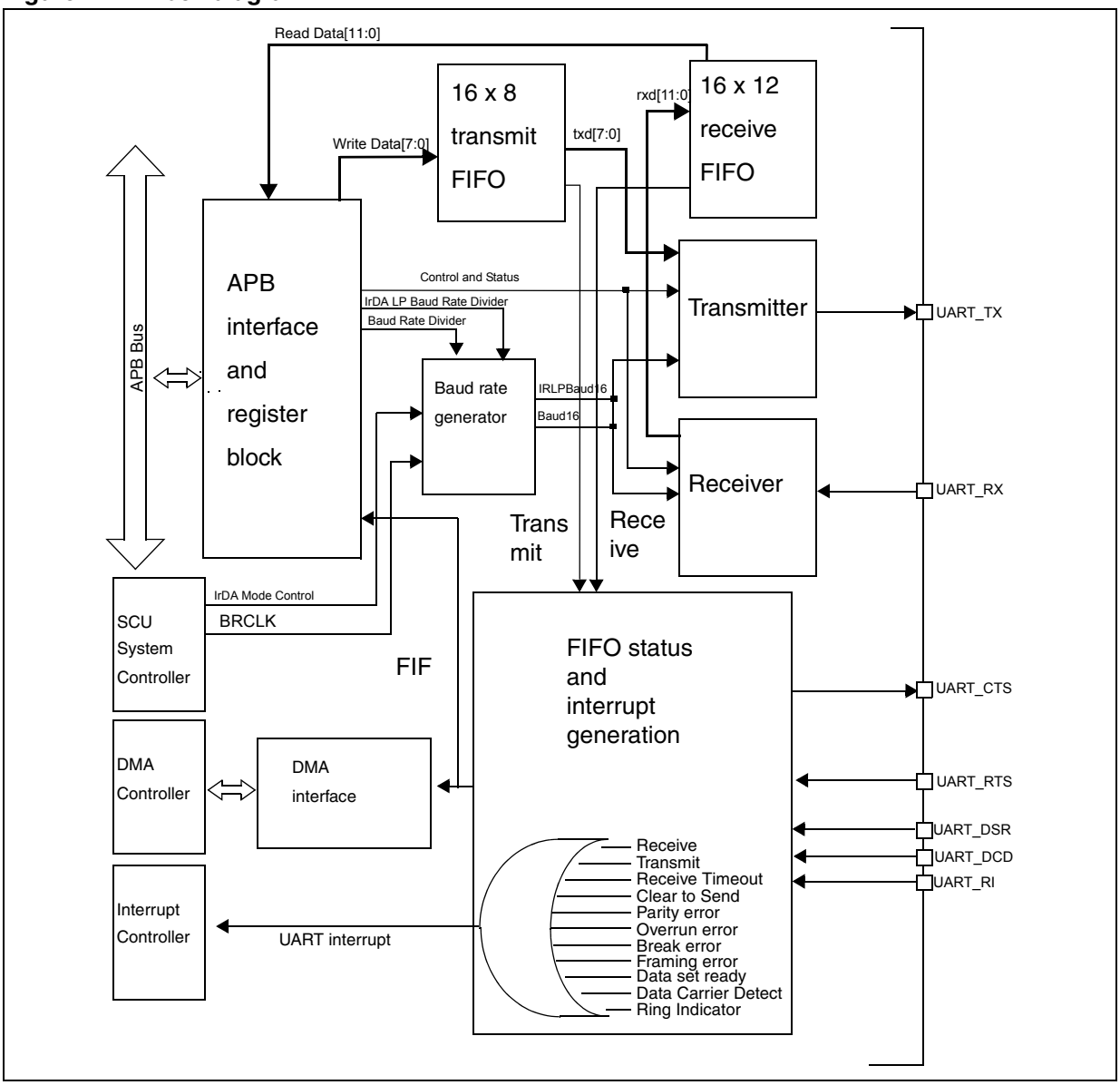
Figure 76. 8-bit data frames with PEN = 1 and STP2 = 1



11.3.1 Functional block diagram

Figure 77 shows the UART functional block diagram.

Figure 77. Block diagram



11.3.2 Fractional baud rate divider

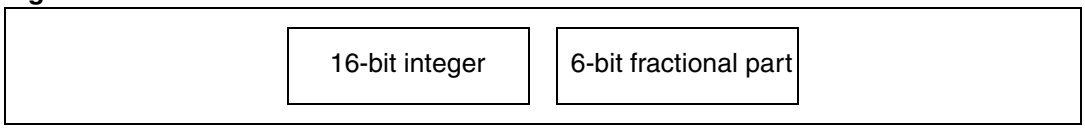
The baud rate divider is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part. It is used by the baud rate generator to determine the bit period. The fractional baud rate divider enables the use of any clock with a frequency >3.6864 MHz to act as BRCLK, while still being able to generate all the standard baud rates. The BRCLK clock is derived from the Master clock (f_{MSTR}). The BRCLK frequency can either be the same or half of the Master clock. The clock divider is specified in the SCU_CLKCNTR register.

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 BRCLK frequency:

Baud Rate Divider BAUDDIV = Freq(BRCLK**)** / (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 78.

Figure 78. 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 BRCLK 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 79 is an example of how to calculate the divider value.

Figure 79. Calculating the divider value

Baud Rate Divider = (24 * 10^6)/(16 * 230400) = 6.51
Therefore, BRD_I = 6 and BRD_F = 0.51
Therefore, fractional part, m = integer((0.51 * 64) + 0.5) = 33 (21h)
Generated baud rate divider = 6 + 33/64 = 6.515625
Generated baud rate = (24 * 10^6)/(16 * 6.515625) = 230215
Error = (230215 - 230400)/230400 * 100 = -0.08 %
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.

Table 27 shows some typical bit rates and their corresponding dividers, given the UART clock (BRCLK) frequency of 96 MHz.

Table 27. Typical baud rates and their corresponding integer and fractional (dividers (BRCLK = 96 MHz)

Programmed divider (integer)	Programmed divider (fraction)	Required bit rate (bps)	Generated bit rate (bps)	Error (%)
1Ah	03h	230400	230353.93	-0.02 %
34h	05h	115200	115211.52	0.01 %
4Eh	08h	76800	76800.00	0.00 %
9Ch	10h	38400	38400.00	0.00 %
138h	20h	19200	19200.00	0.00 %
1A0h	2Bh	14400	14399.82	0.00 %
271h	00h	9600	9600.00	0.00 %
9C4h	00h	2400	2400.00	0.00 %
D511h	1Dh	110	110.00	0.00 %

Table 28 shows some required bit rates and their corresponding integer and fractional divider values and generated bit rates with a clock frequency of 48 MHz.

Table 28. Typical baud rates and their corresponding integer and fractional dividers (BRCLK = 48 MHz)

Programmed divider (integer)	Programmed divider (fraction)	Required bit rate (bps)	Generated bit rate (bps)	Error (%)
0Dh	01h	230400	230492.20	0.04 %
1Ah	03h	115200	115176.96	-0.02 %
27h	04h	76800	76800.00	0.00 %
4Eh	08h	38400	38400.00	0.00 %
9Ch	10h	19200	19200.00	0.00 %
D0h	15h	14400	14400.36	0.00 %
138h	20h	9600	9600.00	0.00 %
4E2h	00h	2400	2400.00	0.00 %
6A88h	2Fh	110	110.00	0.00 %

Table 29 shows some required bit rates and their corresponding integer and fractional divider values and generated bit rates given a clock frequency of 24 MHz.

Table 29. Typical baud rates and their corresponding integer and fractional dividers (BRCLK = 24 MHz)

Programmed divider (integer)	Programmed divider (fraction)	Required bit rate (bps)	Generated bit rate (bps)	Error (%)
06h	21h	230400	230215.83	-0.08 %
0Dh	01h	115200	115246.10	0.04 %
13h	22h	76800	76800.00	0.00 %
27h	04h	38400	38400.00	0.00 %
4Eh	08h	19200	19200.00	0.00 %
68h	0Bh	14400	14399.28	0.00 %
9Ch	10h	9600	9600.00	0.00 %
271h	00h	2400	2400.00	0.00 %
3544h	17h	110	110.00	0.00 %

11.3.3 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.

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 30* shows the bit functions of the receive FIFO.

Table 30. 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

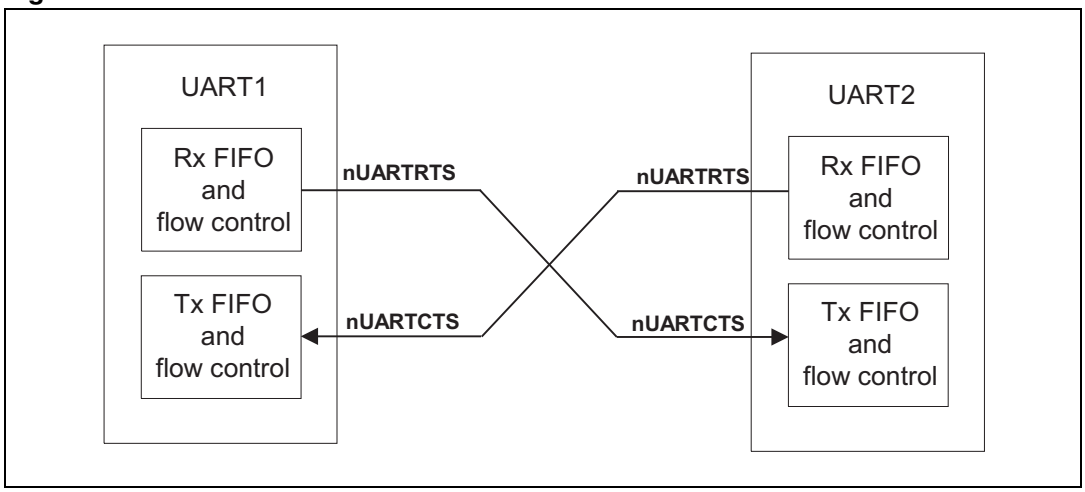
You can perform loop-back testing for UART data by setting the *Loop Back Enable* (LBE) bit to 1 in the control register UART_CR (bit 7).

Data transmitted on the UART_TX pin is received on the UART_RX input.

11.3.4 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 80* shows how two devices can communicate with each other using hardware flow control.

Figure 80. 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 31* shows how you must set the bits to enable RTS and CTS flow control both simultaneously and independently.

Table 31. 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.

11.3.5 IrDA mode

IrDA mode is supported by UART0, 1 and 2.

To select IrDA mode, set the corresponding UART_IRDA bit in the *System configuration register 0 (SCU_SCR0) on page 108*.

Program the baud rate by writing to the *IrDA low power counter divisor register (UART_ILPR) on page 308*.

Communication is performed via the UART_RX and UART_TX pins.

11.3.6 Interrupts

There are 11 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 32. 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
Data Set Ready Modem Interrupt	DSRRIS	DSRIM	DSRSMIS	DSRIC
Data Carrier Detect Modem Interrupt	DCDRIS	DCDIM	DCDSMIS	DCDIC
CTS Interrupt	CTSRIS	CTSIM	CTSMIS	CTSIC
Ring Indicator Modem Interrupt	RIRIS	RIIM	RIMIS	RIIC

11.4 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

11.4.1 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. ***Note*:** In FIFO mode, this error is associated with the character at the top of the FIFO.
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. ***Note*:** In FIFO mode, this error is associated with the character at the top of the FIFO.
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

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).

11.4.2 Receive status register/error clear register(UART_RSECR)

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_RSECR. ***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_RSECR. ***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_RSECR. ***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_RSECR. ***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_RSECR register. This read sequence cannot be reversed, because the UART_RSECR 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.

11.4.3 Flag register (UART_FR)

Address offset: 18h

Reset value: 0000 0000 1001 0xxxb

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							RI	TXFE	RXFF	TXFF	RXFE	BUSY	DCD	DSR	CTS
							r	r	r	r	r	r	r	r	r

Bits 15:8	Reserved, forced by hardware to 0
Bit 8	**RI:** *Ring indicator status* This bit is the complement of the UART0_RI input pin. That is the bit is 1 when the pin status is 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.
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).
Bit 2	**DCD:** *Data Carrier Detect* This bit is the complement of the UART0_DCD input pin. That is the bit is 1 when the pin status is 0.
Bit 1	**DSR:** *Data Set Ready* This bit is the complement of the UART0_DSR input pin. That is the bit is 1 when the pin status is 0.
Bit 0	**CTS:** *Clear To Send* This bit is the complement of the UART0_CTS input pin, that is, the bit is 1 when the pin status is 0.

11.4.4 IrDA low power counter divisor register (UART_ILPR)

Address offset: 20h

Reset value: 0000h

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

Bits 15:1	Reserved, forced by hardware to 0
Bits 7:0	**ILPDVSR[7:0]:** *IrDA Low Power Counter Divisor* *These bit are written by software to define the low power counter divisor value used to generate the IrLPBaud16 frequency from BRCLK.* The low-power divisor value is calculated as follows: low-power divisor (ILPDVSR) = (f_{BRCLK} / $f_{IrLPBaud16}$) where FIrLPBaud16 is nominally 1.8432 MHz. You must chose the divisor so that 1.42 MHz < $f_{IrLPBaud16}$ < 2.12 MHz, that results in a low-power pulse duration of 1.41-2.11µs (three times the period of IrLPBaud16). The minimum frequency of IrLPBaud16 ensures that pulses less than one period of IrLPBaud16 are rejected, but that pulses greater than 1.4µs are accepted as valid pulses. ***Note*:** Zero is an illegal value. Programming a zero value results in no IrLPBaud16 pulses being generated.

11.4.5 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* The baud rate divisor value BAUDDIV is comprised of the integer value BAUD DIVINT and the fractional value BAUD DIVFRAC BAUDDIV is calculated as follows: BAUDDIV= (f_{BRCLK} / 16 * baud rate) Refer to *Figure 79: Calculating the divider value* for an example. ***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.

11.4.6 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.

11.4.7 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								SPS	WLEN		FEN	STP2	EPS	PEN	BRK
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8	Reserved, forced by hardware to 0
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 33 on page 312* 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 33 on page 312* 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 33)
Bit 0	**BRK**: *Send Break* 0: Normal mode 1: Send Break, this continually outputs a low-level on the UART_TXD pin, after completing transmission of the current character. For proper execution of the break command, the software must set this bit for at least two complete frames. This bit cannot be written when the STP2 bit is set. (Break feature is not available in “2 stop bits” mode). The break is sent before or after the data depending on these two cases: - If a transmission is on-going, the break character will be sent at the end of the current transmission. - If the transmission has not started yet, the break will be inserted first.

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 33 is a truth table for the SPS, EPS and PEN bits of the UART_LCR register.

Table 33. 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.

11.4.8 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
CTS En	RST En	Reserved	RTS	DTR	RXE	TXE	LBE	Reserved	SIRLP	SIR EN	UART EN
rw	rw		rw	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	**DTR:** *Data Transmit Ready* This bit is the complement of the UART Data Transmit Ready (UART_DTR) modem status output. That is, when the bit is programmed to a 1, the output is 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 6:3	Reserved, forced by hardware to 0
Bit 2	**SIRLP:** *IrDA SIR Low power mode enable* 0: Low-level bits are transmitted as an active high pulse with a width of 3/16th of the bit period . 1: Low-level bits are transmitted with a pulse width which is 3 times the period of the IrLPBaud16 clock, regardless of the selected bit rate. ***Note*:** Setting this bit uses less power, but might reduce transmission distances

Bit 1	**SIREN:** *IrDA SIR enable* This bit set set and cleared by software. 0: IrDA SIR ENDEC disabled 1: IrDA SIR ENDEC enabled ***Note*:** Setting this bit has no effect unless UARTEN = 1
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.

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

11.4.9 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

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.

11.4.10 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	DSRIM	DCDIM	CTSIM	RIIM
-	-	-	-	-	rw	rw	rw	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
Bit 3	**DSRIM:** *Modem DSR Interrupt Mask* 0: DSR Interrupt Disabled 1: DSR Interrupt Enabled
Bit 2	**DCDIM:** *Modem DCD Interrupt Mask* 0: DCD Interrupt Disabled 1: DCD Interrupt Enabled
Bit 1	**CTSIM:** *Modem CTS Interrupt Mask* 0: CTS Interrupt Disable 1: CTS Interrupt Enable
Bit 0	**RIIM:** *Modem RI Interrupt Mask* 0: RI Interrupt Disabled 1: RI Interrupt Enabled

11.4.11 Raw interrupt status register (UART_RIS)

Address offset: 3Ch

Reset value: 0000 0000 0000 xxxxb

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					OERIS	BERIS	PERIS	FERIS	RTRIS	TXRIS	RXRIS	DSR RIS	DCD RIS	CTSRIS	RIRIS
					r	r	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-bit 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. ***Notes*:** - 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. **-** 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.
Bit 3	**DSRRIS:** *DSR Modem Interrupt Status* Gives the raw interrupt state (prior to masking) of the DSR interrupt. The DSR interrupt is asserted if there is a change on UART_DSR. It is cleared by writing a 1 to the corresponding bit in the UART_ICR register.
Bit 2	**DCDRIS:** *DCD Modem Interrupt Status* Gives the raw interrupt state (prior to masking) of the DCD interrupt. The DCD interrupt is asserted if there is a change on UART_DCD. It is cleared by writing a 1 to the corresponding bit in the UART_ICR register.
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	**RIRIS:** *RI Modem Interrupt Status* Gives the raw interrupt state (prior to masking) of the RI interrupt. The RI interrupt is asserted if there is a change on UART_RI. It is cleared by writing a 1 to the corresponding bit in the UART_ICR register.

11.4.12 Masked interrupt status register (UART_MIS)

Address offset: 40h

Reset value: 0000 0000 0000 xxxxb

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					OEMIS	BEMIS	PEMIS	FEMIS	RTMIS	TXMIS	RXMIS	DSRMIS	DCDRMIS	CTSMIS	RIMIS
					r	r	r	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.
Bit 3	**DSRMIS:** *DSR Modem Masked Interrupt Status* Gives the masked interrupt state (after masking) of the DSR interrupt.
Bit 2	**DCDMIS:** *DCD Modem Masked Interrupt Status* Gives the masked interrupt state (after masking) of the DCD interrupt.
Bit 1	**CTSMIS:** *CTS Masked Interrupt Status* Gives the masked interrupt state (after masking) of the CTS interrupt.
Bit 0	**RIMIS:** *RI Modem Masked Interrupt Status* Gives the masked interrupt state (after masking) of the RI interrupt.

11.4.13 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	DSRIC	DCDIC	CTSIC	RI
					w	w	w	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
Bit 3	**DSRIC:** *DSR Modem Interrupt Clear* Write ‘1’ clears the DSR interrupt Write ‘0’ has no effect
Bit 3	**DCDIC:** *DCD Modem Interrupt Clear* Write ‘1’ clears the DCD interrupt Write ‘0’ has no effect
Bit 1	**CTSIC:** *CTS Interrupt Clear* Write ‘1’ clears the CTS interrupt Write ‘0’ has no effect
Bit 0	**RIIC:** *RI Modem Interrupt Clear* Write ‘1’ clears the RI interrupt Write ‘0’ has no effect

11.4.14 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													DMAONERR	TXDMAE	RXDMAE
													rw	rw	rw

Bits 15:3	Reserved, forced by hardware to 0
Bit 2	**DMAONERR:** *DMA on Error* 0: DMA receive request enabled when the UART error interrupt is asserted 1: DMA receive request disabled when the UART error interrupt is asserted
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

Note: Only UART0 and UART1 support DMA functions.

Burst data transfers can be made depending on the programmed watermark level and the amount of data in the FIFO. *Table 34* shows the trigger points for DMA burst requests depending on the watermark level, for both the transmit and receive FIFOs.

Table 34. Trigger points for DMA burst requests

Watermark level	Burst length	
	Transmit (number of empty locations)	Receive (number of filled locations
1/8	14	2
1/4	12	4
1/2	8	8
3/4	4	12
7/8	2	14

11.5 UART register map

Table 35. UART 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_RSECR	Receive Status register/Error Clear Register															
18	UART_FR	Flag register															
20	UART_ILPR	Reserved								IrDA Low Power Counter 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															

Refer to *Table 5 on page 35* for the register base addresses.

12 I²C interface module (I2C)

An I²C Bus Interface serves as an interface between the microcontroller and the serial I²C bus. It provides both multimaster and slave functions, and controls all I²C bus-specific sequencing, protocol, arbitration and timing. It supports fast I²C mode (400 kHz).

12.1 Main features

- Parallel-bus/I²C protocol converter
- Multi-master capability
- 7-bit/10-bit addressing
- Transmitter/Receiver flag
- End-of-byte transmission flag
- Transfer problem detection
- Standard/fast I²C mode

I2C master features

- Clock generation
- I²C 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
- I²C bus busy flag
- Detection of misplaced start or stop condition
- Programmable I²C address detection
- Transfer problem detection
- End-of-byte transmission flag
- Transmitter/receiver flag

12.2 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.

12.2.1 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.

12.2.2 Communication flow

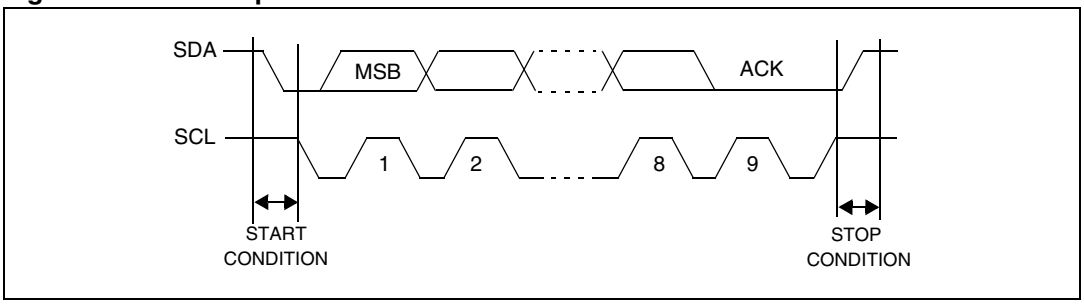
In Master mode, it initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by hardware as soon as the Master mode is selected.

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 81*.

Figure 81. 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 (0-100 kHz) and Fast I^2C (100-400 kHz).

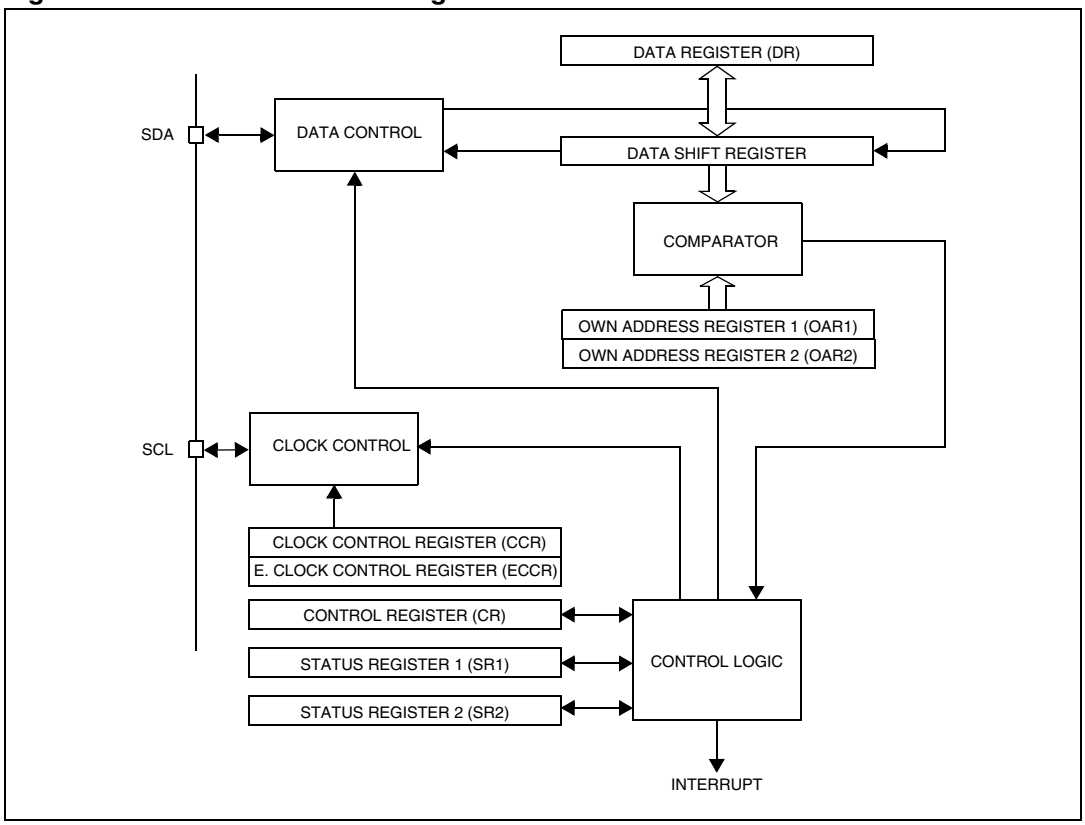
12.2.3 SDA/SCL line control

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.

Figure 82. I^2C interface block diagram



12.3 Functional description

Refer to the I2C_CR, I2C_SR1 and I2C_SR2 registers in *Section 12.5* 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.

12.3.1 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 83* 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 83* 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 83* 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 83* 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.

12.3.2 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 83* 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 83* 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 83* 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 83* 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 83* 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.

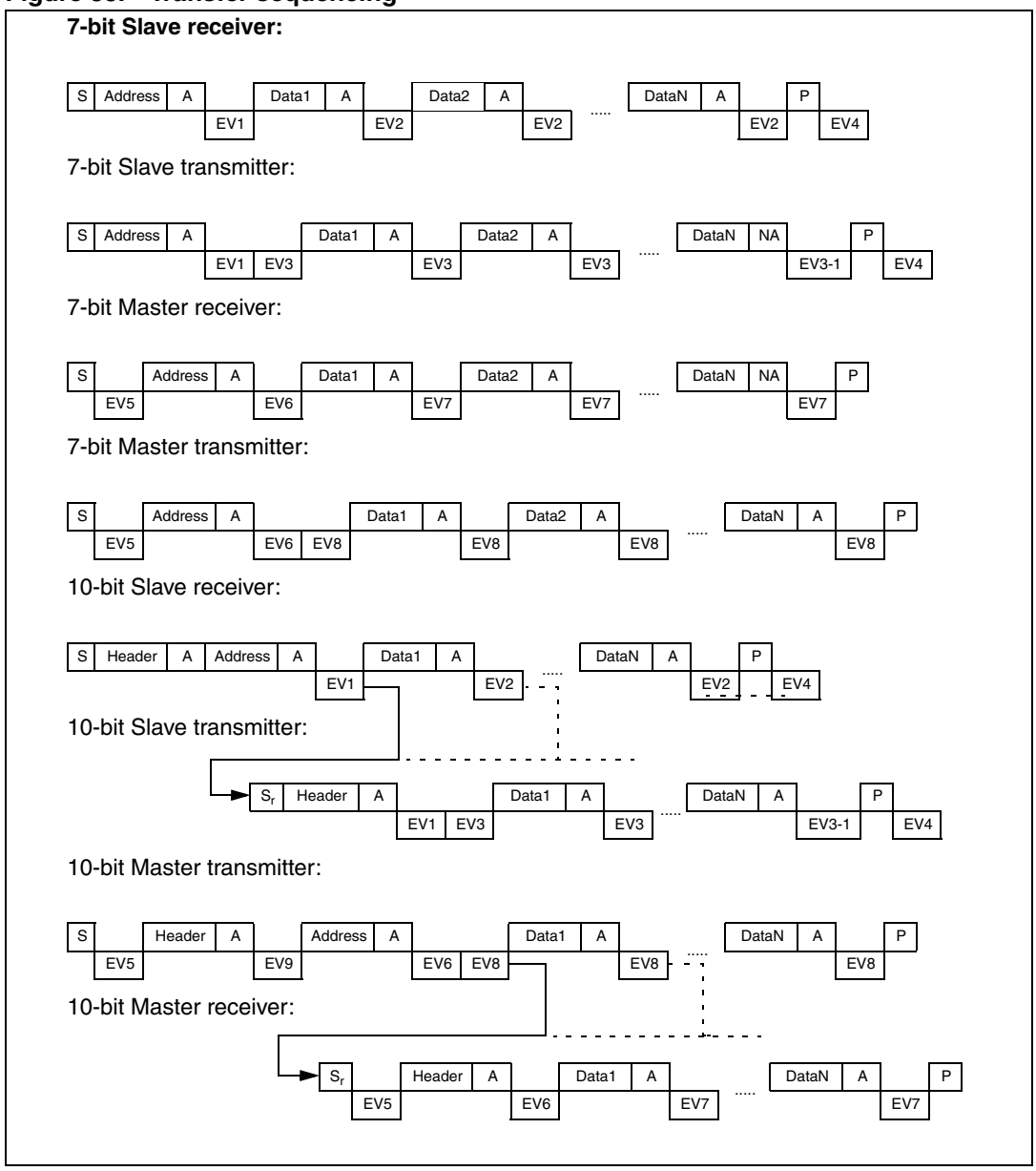
To close the communication: after writing the last byte to the I2C_DR register, set the STOP bit to generate the Stop condition. The interface goes automatically back to slave mode (M/SL bit cleared).

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 83. Transfer sequencing



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 by STOP = 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.

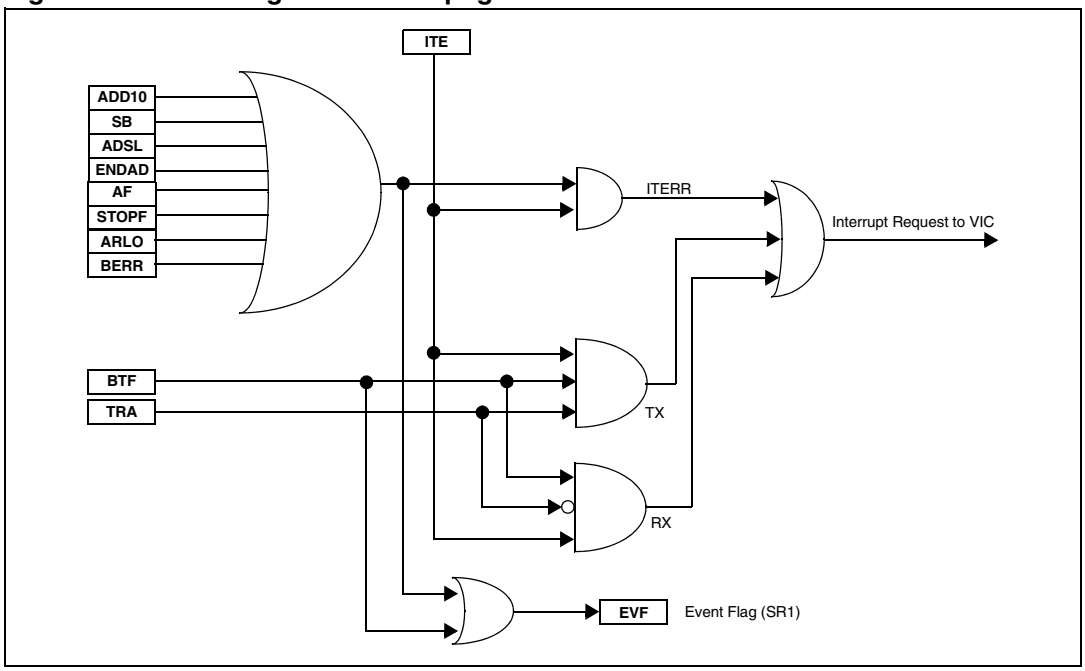
12.4 Interrupts

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 ORed together and issued to the interrupt controller on a single channel as shown in *Figure 84*. The different flags identify the events and can be polled by the software (interrupt service routine).

Figure 84. Event flags and interrupt generation



12.5 Register description

12.5.1 I2C 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

Bits 7:6	Reserved, forced by hardware to 0
Bit 5	**PE:** *Peripheral Enable* This bit is set and cleared by software 0: Peripheral disabled. All the bits in 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: Master/Slave capability enabled. The I2C I/O pins must be enabled by setting up the configuration registers in the corresponding GPIO port. ***Note*:** 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 84* 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 83*) is detected.

12.5.2 I²C 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 83*. 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 83*). 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

12.5.3 I²C 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

Bits 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

12.5.4 I²C 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 I²C mode.* This bit is set and cleared by software. It is not cleared when the interface is disabled (PE = 0). 0: Standard I²C mode 1: Fast I²C mode
Bits 6:0	**CC[6:0]**: *12-bit clock divider.* These bits along with the CC[11:7] bit in the Extended Clock Control Register select the speed of the bus (f_{SCL}) depending on the I²C 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): 100 kHz < f_{SCL} < 400 kHz $f_{SCL} = f_{PCLK} / (3 \times ([CC11...CC0] + 9))$ 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: 1 The programmed f_{SCL} assumes no load on SCL and SDA lines.

2 For correct usage of the divider, [CC11...CC0] must be equal or greater than 0x002 (000000000010b). [CC11...CC0] equal to 0x001 (000000000001b) is not admitted.

3 When the I2C interface is operating in slave mode, configuring the I2C clock by writing into register I2C_CCR has no effect. I2C_CCR register is configured only when the I2C interface is operating in Master mode.

12.5.5 I²C 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 I²C mode. They are not cleared when the interface is disabled (PE = 0)

12.5.6 I²C 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

Table 36. 7-bit addressing mode

Bits 7:1	**ADD[7:1]**: *Interface address* These bits define the I²C 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.

Table 37. 10-bit Addressing Mode

Bits 7:0	**ADD[7:0]**: *Interface address* These are the least significant bits of the I²C bus address of the interface. They are not cleared when the interface is disabled (PE = 0).

12.5.7 I²C own address register 2 (I2C_OAR2)

Address offset: 14h
Reset value: 00h

7	6	5	4	3	2	1	0
FR2	FR1	FR0	Reserved		ADD9	ADD8	Reserved
rw	rw	rw	-		rw	rw	-

Bits 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 I²C 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 100: f_{PCLK} = 40 to 53.33 MHz
Bits 4:3	Reserved, forced by hardware to 0
Bits 2:1	**ADD[9:8]**: *Interface address* These are the most significant bits of the I²C 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

12.5.8 I²C 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

Bits 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 starts 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.

12.6 I2C register map

Table 38. I2C register map

Address offset	Register name	7	6	5	4	3	2	1	0
00h	I2C_CR	Reserved		PE	ENGC	START	ACK	STOP	ITE
04h	I2C_SR1	EVF	ADD10	TRA	BUSY	BTF	ADSL	M/SL	SB
08h	I2C_SR2	Reserved		ENDAD	AF	STOPF	ARLO	BERR	GCAL
0Ch	I2C_CCR	FM/SM	CC6	CC5	CC4	CC3	CC2	CC1	CC0
10h	I2C_OAR1	ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0
14h	I2C_OAR2	FR2	FR1	FR0	Reserved		ADD9	ADD8	Reserved
18h	I2C_DR	D7	D6	D5	D4	D3	D2	D1	D0
1Ch	I2C_ECCR	Reserved			CC11	CC10	CC9	CC8	CC7

Refer to *Table 5 on page 35* for the register base addresses.

13 3-phase induction motor controller (MC)

13.1 Introduction

The MC controller is designed for variable speed motor control applications. Three PWM outputs are available for controlling a three-phase motor drive. Rotor speed feedback is provided by capturing a tachogenerator input signal.

13.2 Main features

- 16 or 10-bit PWM up/down counter
- Classical and zerocentered PWM operating modes
- Double update mode for enhanced PWM control
- Full-scale PWM generation
- 10 or 6-bit dead time generator
- Rotor speed measurement
- 8 interrupt sources + 1 emergency stop interrupt
- Register write protection
- ADC trigger capability

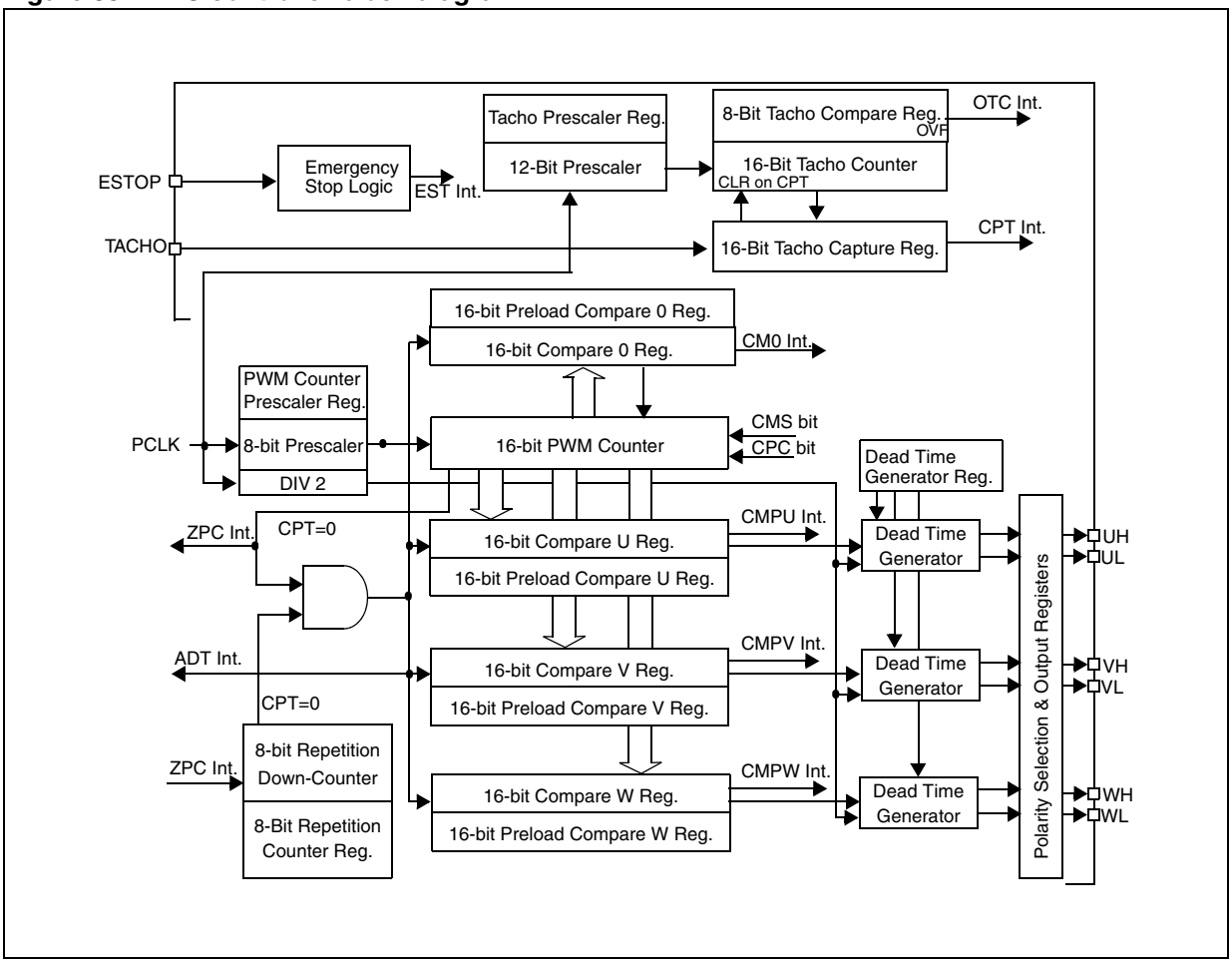
13.3 Functional description

The MC controller consists of the following function blocks:

- Input and output pins
- Rotor speed measurement
- 3-Phase PWM signal generation
- Dead time generation
- Polarity selection
- Interrupt generation

The block diagram is shown in *Figure 85*.

Figure 85. MC controller block diagram



Input and output pins

- Input Pins
  - TACHO: Signal input from a tachogenerator for measuring the rotor speed
  - ESTOP: Input signal for disabling the MC output and sending an interrupt request to the wakeup/interrupt unit (WIU). The ESTOP is an active low signal.
- Output Pins
  - UH, UL, VH, VL, WH, WL: 3-Phase PWM signals and complementary signals

Rotor speed measurement

The TACHO signal is input from a Schmitt trigger port. When a rising and/or falling edge occurs (programmable edge sensitivity), the MC controller does the following:

- Captures the 16-bit Tacho Counter
- Clears the Tacho Counter (if the CCPT bit is set)
- Generates a CPT interrupt

The 16-bit Tacho Counter clock is derived from PCLK through a 12-bit prescaler. The 12-bit prescaler divides by 1, 2, 3, ..., 4096.

If no edge occurs on the TACHO signal or the event sensitivity is disabled (see *Section 13.4.12*) and the 16-bit counter is running, an OTC overflow interrupt will be issued when the MSB (Most Significant Byte) of the Tacho Counter reaches the Tacho Compare register value.

Three-phase PWM generator

The PWM counter can be configured in 10-bit or 16-bit mode by programming the EPWM bit in the MC_ECR register.

In 10-bit mode:

- The 3-Phase PWM signal is generated using a 10-bit PWM Counter and three 11-bit Compare registers one for each phase (U, V, W).

In 16-bit mode:

- The 3-Phase PWM signal is generated using a 16-bit PWM Counter and three 16-bit Compare registers one for each phase (U, V, W).

The PWM Counter clock is supplied through an 8-bit prescaler (dividing by 1, 2, 3, .., 256).

The PWM generator can work in Zerocentered mode or in Classical mode. The mode is selected by the CMS bit in the MC_PCR0 register:

- ***Zerocentered Mode***: In this operating mode, the PWM Counter counts up to the value loaded in the 10-bit.
  Compare 0 register then counts down until it reaches zero and restarts counting up.
- ***Classical Mode***: In this operating mode, the PWM Counter counts up to the value loaded in the 10-bit Compare Register. Then the PWM Counter is cleared and it restarts counting up.

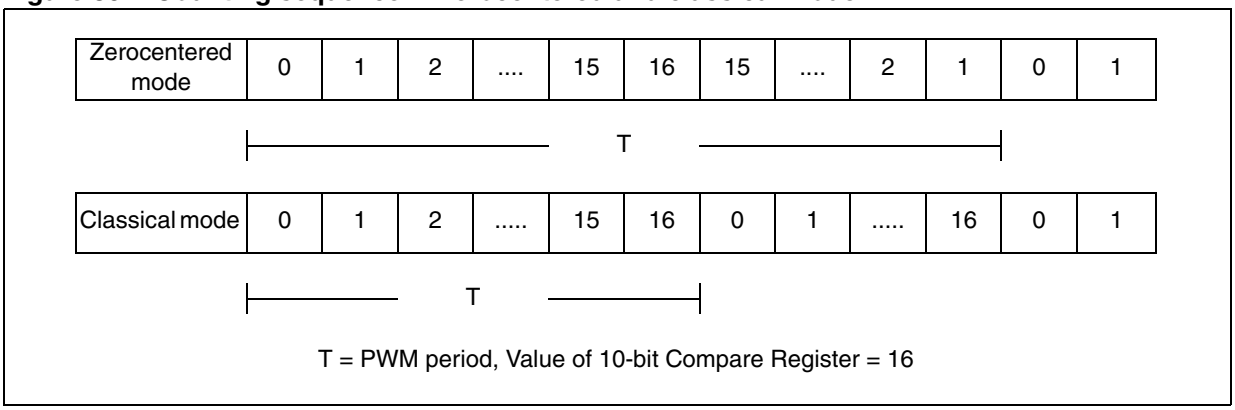
Figure 86 shows the counting sequence in Classical and Zerocentered mode.

PWM signal generation in zerocentered mode

In this mode, all three PWM signals are set to ‘0’ when the PWM Counter reaches, in up-counting, the corresponding W, V or U Compare register value and they are set to ‘1’ when the PWM Counter reaches the Compare value again in down-counting.

The comparison is performed between the PWM Counter value and W, V or U Compare register (either in Zerocentered or in Classical mode).

Figure 86. Counting sequence in zerocentered and classical mode

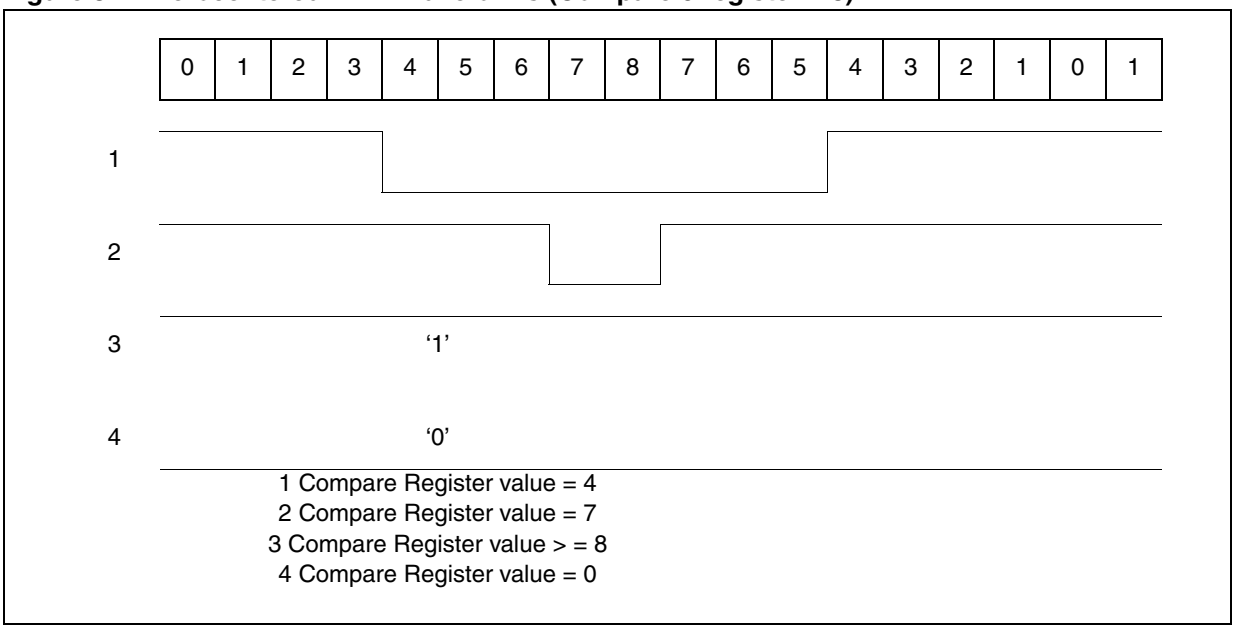


If the W, V or U Compare register value is greater than the Compare 0 Register, the corresponding PWM output signal is held at ‘1’.

If the W, V or U Compare register value is 0, the corresponding PWM output signal is held at ‘0’.

Figure 87 shows some Zerocentered PWM waveforms in an example where the Compare 0 register value = 8.

Figure 87. Zerocentered PWM waveforms (Compare 0 register = 8)



Double update mode

Double update mode can be used with Zerocentered mode. It doubles the frequency of control value updates to the PWM counter compare registers allowing a finer granularity of PWM control.

Double update mode is enabled by setting the DUM bit in the *Enhanced control register (MC_ECR)*.

The three phase compare registers CMU, CMV, and CMW are updated from their respective preload registers when the PWM counter register hits its maximum value as well as its minimum value (i.e., zero). The maximum value for the PWM counter is the value set in the CM0 register. When the PWM counter increments up to this value, it resets to zero (in classical mode) or reverses direction and begins counting down to zero (in zero-centered mode). Refer to *Figure 88*.

The Repetition Down Counter is also decremented at Max, CM0.

DUM is only applicable in zero-centered mode, because in classical mode, the maximum value for the PWM counter register is followed immediately by a clear to zero. The zero value triggers a normal update cycle, which would make the DUM update on the maximum counter value redundant.

Caution: In double update mode, the CMP0 register can only be modified in the following conditions:

1. If MC_REP = 0.
2. If MC_REP = 1 and update is done at crest. This is done by first programming MC_REP = 0, then starting the counter to cause the MC_REP preload value to be taken into account and re-programming MC_REP = 1 immediately afterwards.

When in double update mode, CMP0 must not be modified in conditions other than 1 and 2 above.

Figure 88. Normal zerocentered mode

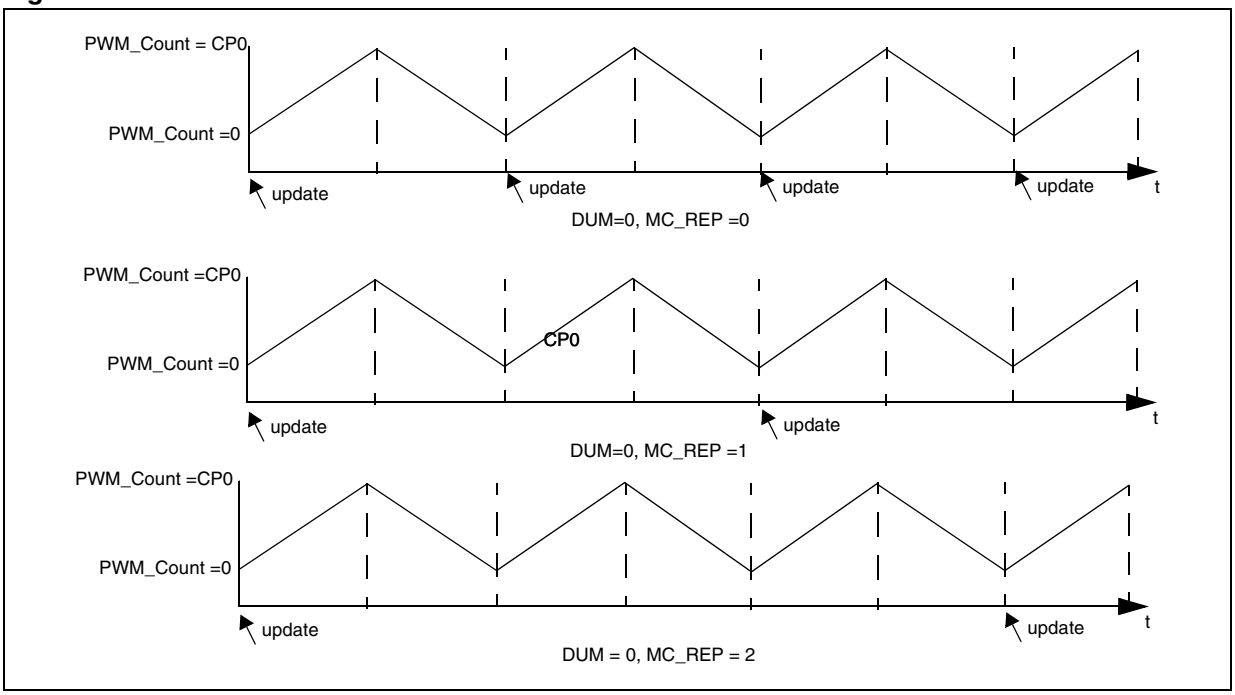
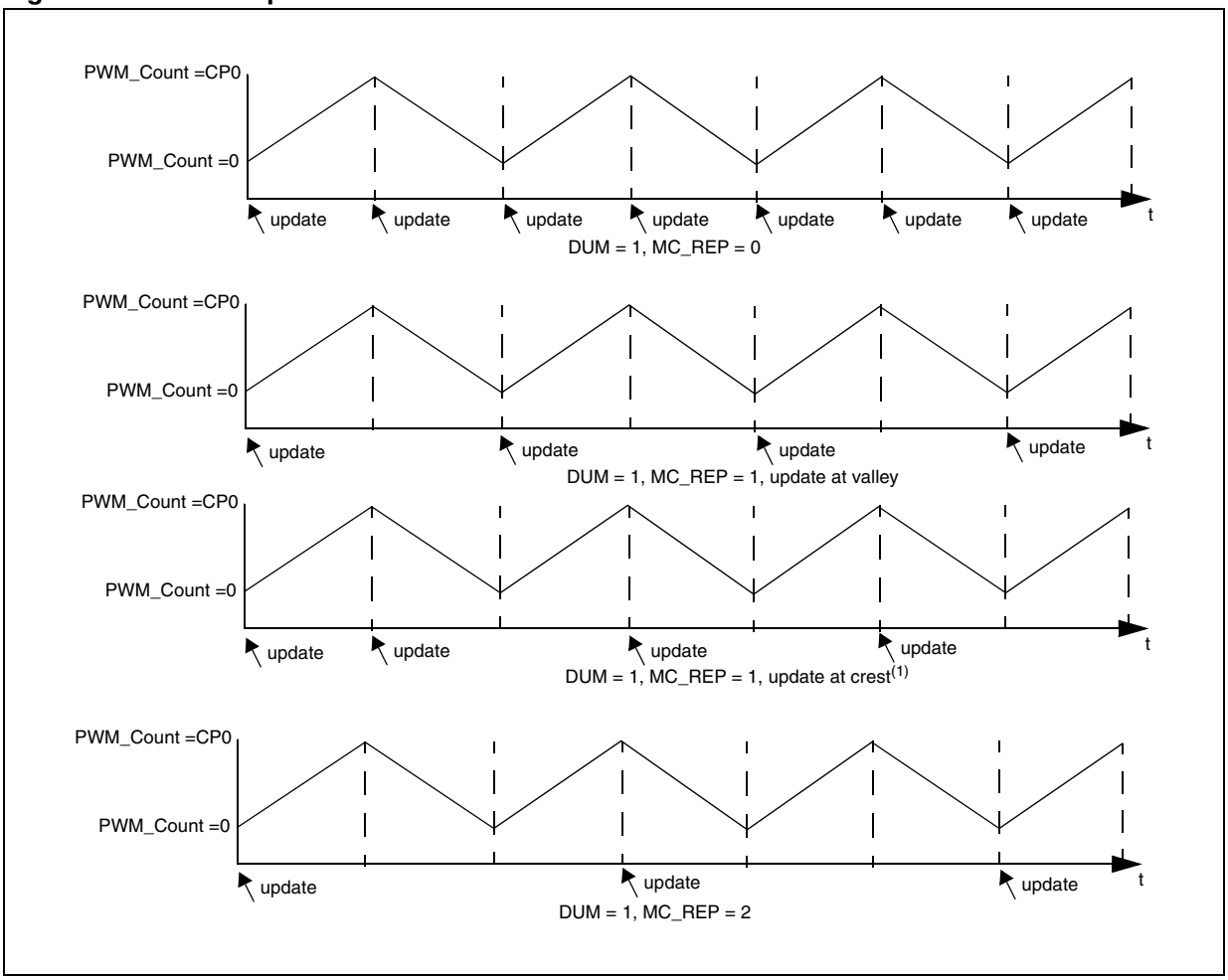


Figure 89. Double update zerocentered mode



1. Update at crest is done by first programming MC_REP = 0, then starting the counter to cause the MC_REP preload value to be taken into account and re-programming MC_REP = 1 immediately afterwards.

PWM signal generation in classical mode

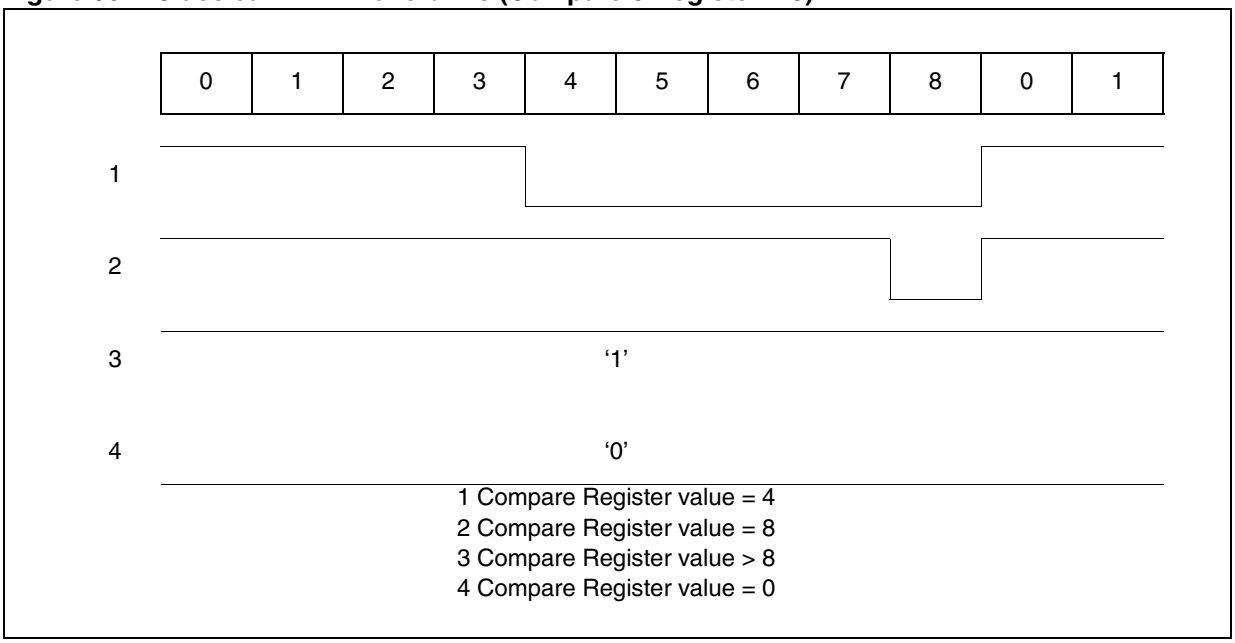
In this mode, each of the three PWM signals set to ‘0’ when the PWM Counter reaches, in up-counting, the corresponding W, V or U Compare register value and they are set to ‘1’ when the PWM Counter is cleared.

If the W, V or U Compare register value is greater than the Compare 0 register, the corresponding PWM output signal is held at ‘1’.

If the W, V or U Compare register value = 0, the corresponding PWM output signal is held at ‘0’.

Figure 90 shows some Classical PWM waves in an example where the Compare 0 register value = 8.

Figure 90. Classical PWM Waveforms (Compare 0 Register = 8)



Repetition down-counter

Both in Zerocentered and Classical working mode, the four Compare registers (one Compare 0 and three for the U, V and W phases) are updated when the PWM counter value is zero and the 8-bit Repetition Down-Counter has reached zero value by counting or by software programming (see *Section 13.4.13*).

This means that data transits from the Preload Compare registers to the Compare registers every N cycles of the PWM Counter, where N is the value of the 8-bit Repetition register (N = 1, 2, .., 256).

Dead time generator

For each phase there is one 10-bit or 6-bit Dead Time generator. The size of the Dead Time generator is selected by programming the EDTC bit in the MC_ECR register.

It generates two output signals: h and l.

- The h output signal is the same as the input phase signal except for the rising edge, which is delayed relatively to the input signal rising edge.
- The l output signal is the opposite of the input phase signal except the rising edge which is delayed relatively to the input signal falling edge.

The delay is the same for each phase (U,V,W) and its value depends on the content of the Compare Phase registers.

Delay = N x T when the MC_CMPx register is odd.

Delay = (N-1/2) x T when the MC_CMPx register is even.

where T is the period of the Dead Time Generator input clock (PCLK divided by 2) and N is the 10 or 6-bit number in the Dead Time register.

If the DTE bit in the MC_PCR0 register is reset, the Dead Time Generator is disabled. This means that no delays are added to the l complemented outputs.

Figure 91 shows an example waveform of the U phase.

If the delay is greater than the width of the active phase (l or h) then the corresponding pulse is not generated.

See *Figure 92* and *Figure 93*.

Figure 91. Dead Time waveforms

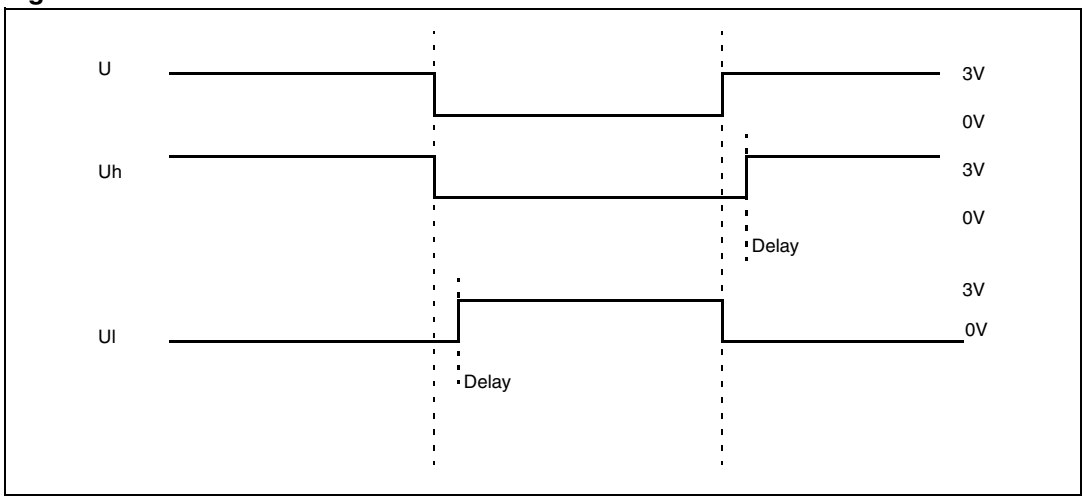


Figure 92. Dead time waveforms with delay greater than the negative PWM pulse

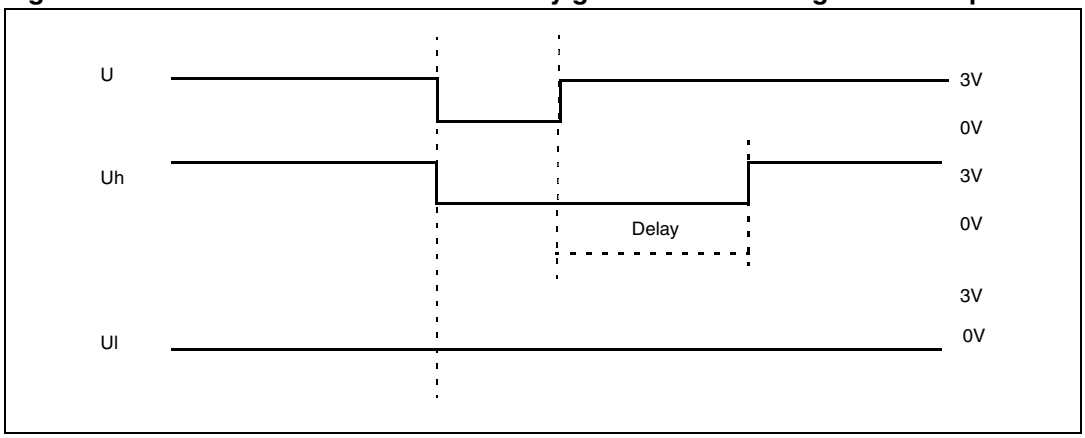
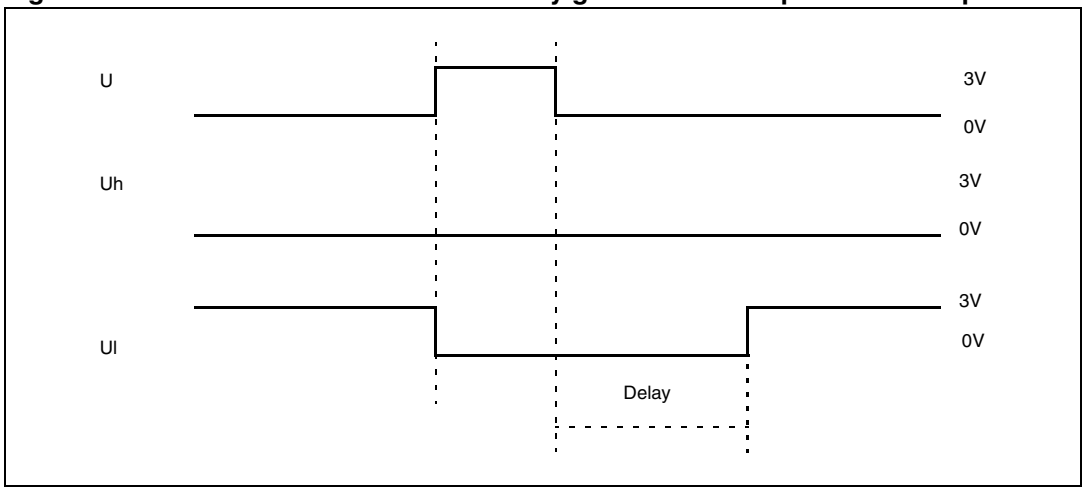


Figure 93. Dead time waveforms with delay greater than the positive PWM pulse



Polarity selection

The polarity selection performs a logical complement of the input signals (Uh, Ul, Vh, Vl, Wh, Wl) as programmed in the Polarity Selection register.

Interrupts/emergency stop

The MC controller generates the 8 interrupt events described in the Interrupt pending register (*Section 13.4.3*). These can be masked individually described in the Interrupt Mask register (*Section 13.4.16*) or globally by the GPIE bit in the MC_PCR2 register (*Section 13.4.13*). The interrupts are ORed together and a single interrupt request is sent to the interrupt controller (VIC).

The emergency stop interrupt request is enabled from the MC block after reset. Setting the DISEST bit in the MC_PCR1 register blocks the emergency stop input. The emergency stop request is connected as a special external interrupt request to the WIU. Refer to the VIC and WIU chapters for more information. When an emergency stop event occurs, the EST flag is set in the interrupt pending register. This flag can be cleared by writing a special 16-bit value in the MC_ESC register.

Output values in emergency stop and debug mode

Two control bits, EMS and ESP, are present in the MC_ECR register which, under particular circumstances, modify the logic values that are output on the six PWM output pins MC_UH, MC_UL, MC_VH, MC_VL, MC_WH, and MC_WL.

Bit MC_ECR[6] is designated as the Enhanced Motor Stop or EMS control bit. Writing a '1' to that bit enables an enhanced stop feature that is described below. Bit MC_ECR[1] is the Enhanced Stop Polarity or ESP bit. The ESP bit selects between normal and inverted output values under applicable conditions.

The conditions under which these bits affect the PWM output values are complex. However, they can be summarized as any condition under which normal PWM cycling is halted, or needs to be halted. The intent is to place all motor drive switches in the OFF state, allowing the motor to freewheel. The purpose is to protect the motor drive circuits from burnout, not to apply active braking to stop the motor.

There are three situations in which normal PWM cycling is halted. One is in response to an active signal on the motor controller's Emergency Stop pin, ESTOP. This signal, mapped through one of the GPIO interrupt request pins, normally originates with thermal protection circuitry in the motor drive power electronics. It should not be confused with the signal from a motor equipment operator's emergency OFF button. The latter calls for active braking of the motor, which requires continued PWM cycling.

In normal operation, the automatic response to the ESTOP signal is to force the PWM outputs to zeroes. Zero is assumed to be the value corresponding to the OFF state for the drive switches. In Enhanced Stop configuration, the value forced to the PWM outputs is determined by the ESP bit. This allows for the possibility of that a PWM output of '1' corresponds to the OFF state of a drive switch.

The second situation in which PWM cycling is halted is when software has explicitly stopped it, by clearing the PWM Counter Enable bit (PCE, bit 5) in register MC_PCR0. In normal operation, it is essential for the controller software to "safe" the PWM outputs before disabling the PWM counter. This is done by setting bit 6, the Output Data Selection bit (ODS) in the Output Peripheral Register MC_OPR. Setting the ODS bit in MC_OPR forces the six PWM outputs to the XOR of MC_OPR bits [5:0] and MC_PSR bits [5:0]. Failure to set the ODS bit before disabling the PWM counter could result in burnout of one or more power switches in the motor drive. As a protection measure, the Enhanced Motor Stop (EMS) feature has been implemented. If the PWM counter is disabled by software while EMS bit is enabled, control logic will check the ODS bit. If it is not set, then the PWM outputs are forced to a safe state in the same manner they would be in response to the ESTOP signal, where the PWM output values are the XOR of ESP bit and MC_PSR bits[5:0]

The third situation in which PWM cycling is halted is in debug mode, when a hardware break condition effectively halts the master clock. When EMS bit is set, the Debug Output Protection bit (DOP) in register MC_ECR enables the PWM outputs to be determined by the ESP bit.

ADC trigger

The microcontroller's Analog to Digital Converter trigger input is internally connected to the IMC. An ADC conversion can be triggered one of the following IMC events:

- ZPC, when the PWM counter reaches zero.
- CM0, when the PWM counter reaches its maximum count.
- ADT, when the PWM counter equals zero and the Repetition Down counter equals zero.

This is selected by the ATS[1:0] bits in the *Enhanced control register (MC_ECR)*.

13.3.1 Tacho counter operating modes

The Tacho Counter can work in One Shot mode or in Continuous mode.

In both Continuous or One Shot mode the Capture event can be generated by hardware (TACHO Pin) or by software (STC bit in the MC_PCR1 register) according to the value of the TES bit in the MC_PCR1 register.

When the CTC bit in the MC_PCR0 register is set, the TACHO Counter is cleared (this bit is reset by hardware).

Tacho counter in one shot mode

In this operating mode (TCB bit = 1 in the MC_PCR1 register) the Counter does the following:

- Counting is started by setting the TCE bit in the MC_PCR0 register.
- When a Capture event occurs, counting is stopped (TCE bit is cleared), the value is captured and a CPT interrupt is generated (if the CCPT bit in the MC_PCR1 register is set, the Counter is cleared).
- When the MSB of Tacho Counter reaches the Tacho Compare register value, the Counter is stopped (TCE bit is cleared) and the OTC interrupt is generated.

Tacho counter in continuous mode

In this operating mode (TCB bit = 0 in the MC_PCR1 register) the Counter does the following:

- Counting is started by setting the TCE bit in the MC_PCR0 register.
- Every Capture event, the value is captured and a CPT interrupt is generated (if the CCPT bit in the MC_PCR1 register is set, the Counter is cleared).
- When the MSB of Tacho Counter reaches the Tacho Compare register value, an OTC interrupt is generated.

13.3.2 MC operating modes

The MC controller can work in two different modes:

- Hardware Operating mode (DTS bit = 0 in the MC_PCR2 register)
- Software Operating mode (DTS bit = 1 in MC_PCR2 register)

In both modes, when the corresponding event occurs, the ADT and the other interrupts are generated.

When the CPC bit in the MC_PCR0 register is set, the PWM Counter is cleared (this bit is reset by software).

MC hardware operating mode

After system reset, the Compare U, V, W and Compare 0 register values are all “0”.

When the PWM Counter is enabled (by setting the PCE bit in the MC_PCR0 register) and every time the Repetition Counter and the PWM Counter reach “0” value, the Repetition Counter is loaded, the preload registers are loaded into the Compare registers and an ADT interrupt is generated.

Note: If an ADT (or any other interrupt) is generated and the previous one is not completed, the last one will be lost without any error condition being issued.

MC software operating mode

In this operating mode, the Repetition register and any Compare register can be independently updated by software by setting the SDT bit in the MC_PCR2 register (this bit will be reset by hardware) and the corresponding enable bit in the same register.

No hardware loading is performed when an ADT interrupt is generated.

Note: The Repetition Counter is decremented immediately when the Repetition Counter is updated.

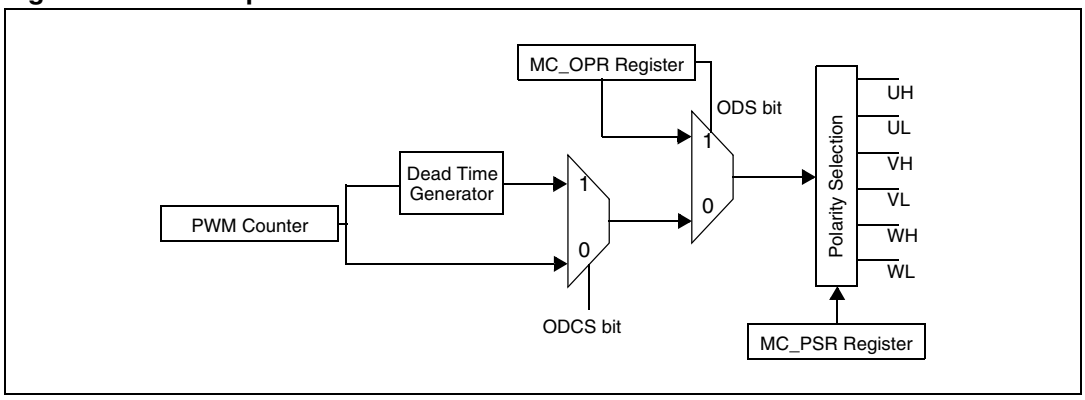
13.3.3 MC output selection

The MC Output can be selected from the following sources:

- MC_OPR register (bits 5:0), by setting the ODS bit in the MC_OPR register.
- Dead Time Generator outputs, by setting the ODCS bit in the MC_PCR0 register.
- PWM Counter outputs (h and l) are not complemented when the ODCS bit is reset.

Figure 94 shows the MC output selection.

Figure 94. MC output selection



13.4 Register description

13.4.1 Tacho capture register (MC_TCPT)

Address offset: 00h

Reset value: 0000 0000 0000 0000 (0000h)

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

Bits 15:0	**TCPT[15:0]**: *Captured value of tacho counter* These bits are read only. They contain the captured value of the tacho counter.

13.4.2 Tacho compare register (MC_TCMP)

Address offset: 04h

Reset value: 0000 0000 1111 1111 (00FFh)

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

Bits 15:8	Reserved, must be kept at reset value
Bits 7:0	**TCMP[7:0]**: *Tacho Compare register* These bits are written by software. They contain the value to be compared to the MSB of the Tacho counter. When the Most Significant Byte of the tacho counter reaches the TCMP value, the Tacho Counter is cleared and an OTC interrupt is generated both in Continuous and One Shot modes.

13.4.3 Interrupt pending register (MC_IPR)

Address offset: 08h

Reset value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							EST	CM0	CPT	OTC	ADT	ZPC	CMPU	CMPV	CMPW
							rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bits 15:9	Reserved, must be kept at reset value
Bit 8	**EST:** *Emergency stop bit* This bit is set by hardware in response to an emergency stop condition. The EST bit must be cleared by software. 0: No EST interrupt occurred 1: EST interrupt pending
Bit 7	**CM0:** *Compare0 of PWM pending bit* This bit is set by hardware when the PWM counter reaches the value in the Compare 0 register while CM0E = 1. The CM0 bit must be cleared by software. 0: No CM0 interrupt occurred 1: CM0 interrupt pending
Bit 6	**CPT:** *Capture of Tacho counter pending bit* This bit is set by hardware when a Tacho signal event occurs while CPTE = 1. The CPT bit must be cleared by software. 0: No CPT interrupt occurred 1: CPT interrupt pending
Bit 5	**OTC:** *Overflow of Tacho counter pending bit* This bit is set by hardware on a Tacho counter overflow while OTCE = 1. The OTC bit must be cleared by software. 0: No OTC interrupt occurred 1: OTC interrupt pending
Bit 4	**ADT:** *Automatic Data Transfer pending bit* This bit is set by hardware when data is transferred from the preload registers to the compare registers while ADTE = 1. The ADT bit must be cleared by software. 0: No ADT interrupt occurred 1: ADT interrupt pending
Bit 3	**ZPC:** *Zero of PWM counter pending bit* This bit is set by hardware when the PWM counter reaches zero while ZPCE = 1. The ZPC bit must be cleared by software. 0: No ZPC interrupt occurred 1: ZPC interrupt pending

Bit 2	**CMPU:** *Compare U pending bit* In Classical Mode (CMS bit = 0), this bit is set by hardware when the PWM Counter reaches the Compare U register value while CMPUE = 1. In Zerocentered Mode (CMS bit =1), this bit is set by hardware when the PWM Counter reaches the Compare U register value while CMPUE = 1 in up or downcounting (depending on the UDIS bit in the MC_PSR register). The CMPU bit must be cleared by software. 0: No CMPU interrupt occurred 1: CMPU interrupt pending
Bit 1	**CMPV:** *Compare V pending bit* In Classical Mode (CMS bit = 0), this bit is set by hardware when the PWM Counter reaches the Compare V register value while CMPVE = 1. In Zerocentered Mode (CMS bit =1), this bit is set by hardware when the PWM Counter reaches the Compare V register value while CMPVE = 1 in up or downcounting (depending on the UDIS bit in the MC_PSR register). The CMPV bit must be cleared by software. 0: No CMPV interrupt occurred 1: CMPV interrupt pending
Bit 0	**CMPW:** *Compare W pending bit* In Classical Mode (CMS bit = 0), this bit is set by hardware when the PWM Counter reaches the Compare W register value while CMPWE = 1. In Zerocentered Mode (CMS bit =1), this bit is set by hardware when the PWM Counter reaches the Compare W register value while CMPWE = 1 in up or downcounting (depending on the UDIS bit in the MC_PSR register). The CMPW bit must be cleared by software. 0: No CMPW interrupt occurred 1: CMPW interrupt pending

13.4.4 Tacho prescaler register (MC_TPRS)

Address offset: 0Ch

Reset value: 0000 0000 0000 0001 (0001h)

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

Bits 15:12	Reserved, must be kept at reset value
Bits 11:0	**TPRS[11:0]:** *Tacho Prescaler value N* This value N is written by software to define the tacho prescaler. The value divides the tacho frequency by N.

13.4.5 PWM counter prescaler register (MC_CPRS)

Address offset: 10h

Reset value: 0000 0000 0000 0001 (0001h)

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

Bits 15:8	Reserved, must be kept at reset value
Bits 7:0	**CPRS[7:0]**: *PWM counter prescaler value N* This value N is written by software to define the PWM counter prescaler. The value divides the PCLK frequency by N.

13.4.6 Repetition counter register (MC_REP)

Address offset: 14h

Reset value: 0000 0000 0000 0000 (0000h)

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

Bits 15:8	Reserved, must be kept at reset value
Bits 7:0	**REP[7:0]**: *Repetition counter value N* If N = 0, each time the PWM Counter reaches zero, the Compare registers are updated and an ADT interrupt is generated.

13.4.7 Compare phase W preload register (MC_CMPW)

Address offset: 18h

Reset value: 0000 0000 0000 0000 (0000h)

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

Bits 15:0	**CMPW[15:0]**: *Phase W preload value* This value is written by software to define the phase W compare value to be loaded at the next register update. ***In 10-bit PWM mode*** (EPWM bit = 0 in MC_ECR register): Bits 15:11 = Reserved, forced to 0 by hardware. Bits 10:0 = Phase W preload value (11 bits) ***In 16-bit PWM mode*** (EPWM bit = 1 in MC_ECR register): Bits 15:0 = Phase W preload value (16 bits)

13.4.8 Compare phase V preload register (MC_CMPV)

Address offset: 1Ch

Reset value: 0000 0000 0000 0000 (0000h)

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

Bits 15:0	**CMPV[15:0]**: *Phase V preload value* This value is written by software to define the phase V compare value to be loaded at the next register update. **In 10-bit PWM mode** (EPWM bit = 0 in MC_ECR register): Bits 15:11 = Reserved, forced to 0 by hardware Bits 10:0 = Phase V preload value (11 bits) **In 16-bit PWM mode** (EPWM bit = 1 in MC_ECR register): Bits 15:0 = Phase V preload value (16 bits)

13.4.9 Compare phase U preload register (MC_CMPU)

Address offset: 20h

Reset value: 0000 0000 0000 0000 (0000h)

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

Bits 15:0	**CMPU[15:0]**: *Phase U preload value* This value is written by software to define the phase U compare value to be loaded at the next register update. ***In 10-bit PWM mode*** (EPWM bit = 0 in MC_ECR register): Bits 15:11 = Reserved, forced to 0 Bits 10:0 = Phase U preload value (11 bits) ***In 16-bit PWM mode*** (EPWM bit = 1 in MC_ECR register): Bits 15:0 = Phase U preload value (16 bits)

13.4.10 Compare 0 preload register (MC_CMP0)

Address offset: 24h

Reset value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMP0[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**CMP0[15:0]**: *Compare 0 preload value* This value is written by software to define the compare 0 value to be loaded at the next register update. It must be greater than 1. ***In 10-bit PWM mode*** (EPWM bit = 0 in MC_ECR register): Bits 15:10 = Reserved, forced to 0 Bits 9:0 = CMP0 preload value ***In 16-bit PWM mode*** (EPWM bit = 1 in MC_ECR register): Bits 15:0 = CMP0 preload value

13.4.11 Peripheral control register 0 (MC_PCR0)

Address offset: 28h

Reset value: 0000 0000 0000 0011 (0003h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DTE	TCE	PCE	CTC	CPC	CMS	UDCS	ODCS
								rw	rw	rw	rw	rw	rw	r	rw

Bits 15:8	Reserved, must be kept at reset value
Bit 7	**DTE:** *Dead Time Counter Enable* 0: Stop and bypass the Dead Time counter 1: Enable the Dead Time counter
Bit 6	**TCE:** *Tacho Counter Enable* 0: Stop Tacho counter and prescaler 1: Start Tacho counter and prescaler ***Note*:** This bit is reset by the counter overflow or by the Tacho capture when the MC controller is in one shot mode.
Bit 5	**PCE:** *PWM Counter Enable* 0: Stop PWM Counter and prescaler 1: Start PWM Counter and prescaler
Bit 4	**CTC:** *Clear Tacho Counter* 0: No effect 1: Clear the Tacho Counter (this bit is reset by hardware)
Bit 3	**CPC:** *Clear PWM Counter* 0: No effect. 1: Clear the PWM Counter (this bit is reset by hardware)
Bit 2	**CMS:** *PWM Counter Mode Selection* 0: Classical mode 1: Zerocentered mode
Bit 1	**UDCS:** *Up/Down status (read only)* This bit is set and cleared by hardware. 0: The PWM Counter is counting down 1: The PWM Counter is counting up
Bit 0	**ODCS:** *Output Dead Time counter Selection* 0: Select the same signal for both (h, l) outputs 1: Select complementary signal for output (Dead Time Generator outputs)

13.4.12 Peripheral control register 1 (MC_PCR1)

Address offset: 2Ch

Reset value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									DISEST	CCPT	TES	STC	TCB	TIN[1:0]	
									rw	rw	rw	rw	rw	rw	

Bits 15:7	Reserved, must be kept at reset value
Bit 6	**DISEST**: *Emergency Stop Disable* 0: Emergency Stop input is enabled 1: Emergency Stop input is blocked from the MC. The EST bit in the IPR register is set when the ESTOP pin input goes low.
Bit 5	**CCPT:** *Clear on Capture of tacho counter* 0: No clear on capture 1: Clear on capture
Bit 4	**TES:** *Tacho Event Selection* 0: Select capture by tacho event signal 1: Select capture by software (STC bit)
Bit 3	**STC:** *Software tacho capture* 0: No effect 1: Capture the Tacho counter (while TES = 1). This bit is reset by hardware.
Bit 2	**TCB:** *Tacho Counter Mode* 0: Select continuous mode 1: Select one shot mode (counting starts when TCE bit is set and stops when a capture or an overflow event occurs)
Bit 1:0	**TIN[1:0]** *Tacho Signal Event Sensitivity* These bits select which Tacho signal event triggers the Tacho Capture register. 00: No operation 01: Falling edge 10: Rising edge 11: Rising and falling edge

13.4.13 Peripheral control register 2 (MC_PCR2)

Address offset: 30h

Reset value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								GPIE	RSE	CWSE	CVSE	CUSE	C0SE	SDT	DTS
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8	Reserved, must be kept at reset value
Bit 7	**GPIE:** *Global Peripheral Interrupt Enable* 0: Disable all MC controller interrupts 1: Enable all MC controller interrupts
Bit 6	**RSE:** *Enable Software Data Transfer to Repetition register* 0: Disable loading of Repetition register by SDT bit 1: Enable loading of Repetition register by SDT bit
Bit 5	**CWSE:** *Enable Software Data Transfer to Compare W* 0: Disable load of Compare W register by SDT bit 1: Enable load of Compare W register by SDT bit
Bit 4	**CVSE:** *Enable Software Data Transfer to Compare V register* 0: Disable loading of Compare V register by SDT bit 1: Enable loading of Compare V register by SDT bit
Bit 3	**CUSE:** *Enable Software Data Transfer to Compare U register* 0: Disable loading of Compare U register by SDT bit 1: Enable loading of Compare U register by SDT bit
Bit 2	**C0SE:** *Enable Software Data Transfer to Compare 0 register* 0: Disable loading of Compare 0 register by SDT bit 1: Enable loading of Compare 0 register by SDT bit
Bit 1	**SDT:** *Software Data Transfer* 0: No effect 1: Transfer Data from preload to compare register (while DTS = 1) (This bit is reset by hardware).
Bit 0	**DTS:** *Data Transfer Mode Selection* 0: Hardware transfer using Repetition counter 1: Software transfer using SDT bit.

13.4.14 Polarity selection register (MC_PSR)

Address offset: 34h

Reset value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									UDIS	PUH	PUL	PVH	PVL	PWH	PWL
									rw	rw	rw	rw	rw	rw	rw

Bits 15:7	Reserved, must be kept at reset value
Bit 6	**UDIS:** *Up-Down Interrupt Select* When the PWM Counter is working in Zerocentered Mode the meaning is: 0: Compare interrupts (CMPU, CMPV, CMPW) are issued when the counter is counting up. 1: Compare interrupts (CMPU, CMPV, CMPW) are issued when the counter is counting down. ***Note*:** This bit has no effect when the Counter is working in Classical Mode.
Bit 5	**PUH:** *Polarity of Uh phase* 0: Positive logical level 1: Complemented logical level
Bit 4	**PUL:** *Polarity of Ul phase* 0: Positive logical level 1: Complemented logical level
Bit 3	**PVH:** *Polarity of Vh phase* 0: Positive logical level 1: Complemented logical level
Bit 2	**PVL:** *Polarity of Vl phase* 0: Positive logical level 1: Complemented logical level
Bit 1	**PWH:** *Polarity of Wh phase* 0: Positive logical level 1: Complemented logical level
Bit 0	**PWL:** *Polarity of Wl phase* 0: Positive logical level 1: Complemented logical level

13.4.15 Output peripheral register (MC_OPR)

Address offset: 38h

Reset value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									ODS	UH	UL	VH	VL	WH	WL
									rw	rw	rw	rw	rw	rw	rw

Bits 15:7	Reserved, must be kept at reset value
Bit 6	**ODS:** *Output Data Selection* 0: Dead time generator data 1: Select the data in bits 5:0 (UH, UL, VH, VL, WH, WL**)**
Bits 5:0	**UH, UL, VH, VL, WH, WL***: UH, UL, VH, VL, WH, WL phases* These bits can be sent out through the output port.

13.4.16 Interrupt mask register (MC_IMR)

Address offset: 3Ch

Reset value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								CM0E	CPTE	OTCE	ADTE	ZPCE	CMPUE	CMPVE	CMPWE
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8	Reserved, must be kept at reset value
Bit 7	**CM0E:** *Compare 0 of PWM counter interrupt enable* 0: Disabled 1: Enabled
Bit 6	**CPTE:** *Capture of Tacho counter Interrupt enable* 0: Disabled 1: Enabled
Bit 5	**OTCE:** *Overflow of Tacho counter Interrupt enable* 0: Disabled. 1: Enabled
Bit 4	**ADTE:** *Automatic data transfer Interrupt enable* 0: Disabled 1: Enabled
Bit 3	**ZPCE:** *Zero of PWM counter interrupt enable* 0: Disabled 1: Enabled
Bits 2:0	**CMPUE, CMPVE, CMPWE:** *Compare U, V, W interrupt enable* 0: Disabled 1: Enabled

13.4.17 Dead time generator register (MC_DTG)

Address offset: 40h

Reset value (EDTC bit = 0) : 0000 0000 0011 1111 (003Fh)

Reset value (EDTC bit = 1) : 0000 0011 1111 1111 (03FFh)

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

Bits 15:0	**DTG[9:0]** *Dead time generator value (N)* This value N is written by software to define the dead time. The delay is N x PCLK period X2 for MC_CMPx register with an odd number. For even number, the delay is (N-1/2) x PCLK period X 2.. If N = 0 the delay is 0. ***In 6-bit mode*** (EDTC bit = 0 in MC_ECR register): Bits 15:6 = Reserved, forced to 0 by hardware Bits 5:0 = Dead time generator value (6 bits) ***In 10-bit mode*** (EDTC bit = 0 in MC_ECR register): Bits 15:10 = Reserved, forced to 0 by hardware Bits 9:0 = Dead time generator value (10 bits)

13.4.18 Emergency stop clear register (MC_ESC)

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
ESC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0	**ESC[15:0]:** *Emergency Stop Clear* This register is used to clear an Emergency Stop condition. Its behavior depends on the setting of the HRE bit set in the MC_ECR register. ***If the HRE bit = 1 in the MC_ECR register*** : After the ESTOP pin input becomes de-asserted, software must: Write 4321h to the MC_ESC register to re-start the PWM. This automatically rearms the Emergency Stop in 1 clock cycle. ***If the HRE bit = 0 in the MC_ECR register*** : After the ESTOP pin input becomes de-asserted, software must: 1. Write 4321h to the MC_ESC register to re-start the PWM 2. Write 00h to the MC_ESC register to re-arm the Emergency Stop

13.4.19 Enhanced control register (MC_ECR)

Address offset: 48h

Reset value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							HRE	DOP	EMS	EPWM	EDTC	ATS[1:0]		ESP	DUM
							rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:9	Reserved, must be kept at reset value.
Bit 8	**HRE:** *ESC register hardware reset* When this bit is set, the hardware clears the ESC register one PCLK after a 4321h was written in ESC, rearming the emergency stop trigger. 0: No automatic hardware clearing of ESC. Software clearing is needed 1: Automatic hardware clearing of ESC register in response to a 4321h value
Bit 7	**DOP:** *Debug output protection bit* When set, this bit make the outputs follow the value of the individual phase polarity setting, PSR, while in debug mode and the SCU_PECGR1 clock enable to IMC, is disabled (= 0). 0: Output phases remain in their last known state 1: Output phases follow the polarity set by PSR. If PUH = 1 and PUL = 0 then during the debug condition above, phase PUH will be stopped at 1 and PUL stopped at 0.
Bit 6	**EMS:** *Enhanced motor stop bit* If set, this bit forces the output phases to follow the value of the individual phase polarity setting when PCR0(5) = '0' (pwm_counter disabled), unless the ODS bit is set in which case the output assumes the value stored in the OPR register. 0: Output phases will remain in their last known state 1: Output phases will follow the polarity set by PSR
Bit 5	**EPWM:** *Enhanced PWM counter* 0: PWM Counter is set to 10 bits 1: PWM Counter is set to 16 bits
Bit 4	**EDTC:** *Enhanced Dead Time counter* 0: Dead Time Counter is set to 6 bits 1: Dead Time Counter is set to 10 bits
Bits 3:2	**ATS[1:0]:** *ADC trigger select bits* 00: No trigger is sent to ADC 01: ZPC event is selected as ADC trigger input 10: CM0 event is selected as ADC trigger input 11: ADT event is selected as ADC trigger input

Bit 1	**ESP:** *Emergency stop polarity bit* This bit inverts the value of phase polarity bits during a stop. 0: Output phases follow phase polarity bits during any stop or emergency stop 1: Output phases follow the inverted phase polarity bit value during any stop or emergency stop.
Bit 0	**DUM:** *Double Update Mode* This bit is set and cleared by software. 0: Double update mode disabled 1: Double update mode enabled. In this mode, the compare registers (CP0, CMU, CMV, CMW) and the Repetition Counter are updated from the preload registers when the Repetition counter = 0 (PWM counter at min) and on a CM0 event (PWM counter at max).

13.4.20 Lock register (MC_LOK)

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
											LOK4	LOK3	LOK2	LOK4	LOK0
											wo	wo	wo	wo	wo

Bits 15:5	Reserved, must be kept at reset value
Bit 4	**LOK4:** *Lock DTG register* This bit can be written once only. Once it is set it can be cleared only by a device reset. When the DTG register is locked, its value cannot be modified. 0: DTG register not locked 1: DTG register locked
Bit 3	**LOK3:** *Lock OPR register* This bit can be written once only. Once it is set it can be cleared only by a device reset. When the OPR register is locked, its value cannot be modified. 0: OPR[5:0] bits not locked 1: OPR[5:0] bits locked
Bit 2	**LOK2:** *Lock phase polarity bits* This bit can be written once only. Once it is set it can be cleared only by a device reset. When the PSR[5:0] bits are locked, the value of PUH, PUL, PVH, PVL, PWH, PWL cannot be modified. 0: PSR[5:0] bits not locked 1: PSR[5:0] bits locked
Bit 1	**LOK1:** *Lock emergency stop disable bit* This bit can be written once only. Once it is set it can be cleared only by a device reset. When the DISEST bit (Emergency Stop disable) is locked, its value cannot be modified. 0: DISEST bit not locked 1: DISEST bit locked
Bit 0	**LOK0:** *Lock Dead Time enable and Output DT counter selection bits* This bit can be written once only. Once it is set it can be cleared only by a device reset. When the DTE and ODCS bits are locked, their value cannot be modified. 0: DTE & ODCS bits not locked 1: DTE & ODCS bits locked

13.5 MC register map

Table 39. MC register map

Address offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	MC_TCPT	Tacho capture register TCPT[15:0]															
04h	MC_TCMP									Tacho compare register TCM[7:0]							
08h	MC_IPR								EST	CM0	CPT	OTC	ADT	ZPC	CMPU	CMPV	CMPW
0Ch	MC_TPRS					Tacho prescaler TPH[11:0]											
10h	MC_CPRS									PWM counter prescaler CPRS[7:0]							
14h	MC_REP									Repetition counter REP[7:0]							
18h	MC_CMPW	Compare Phase W CMPW[15:0] or [9:0]															
1Ch	MC_CMPV	Compare Phase V CMPV[15:0] or [9:0]															
20h	MC_CMPU	Compare Phase U CMPU[15:0] or [9:0]															
24h	MC_CMP0	Compare 0 Preload CP0[15:0] or [9:0]															
28h	MC_PCR0									DTE	TCE	PCE	CTC	CPC	CMS	UDCS	ODCS
2Ch	MC_PCR1										DISEST	CCPT	TES	STC	TCB	TIN1	TIN0
30h	MC_PCR2									GPIE	RSE	CWSE	CVSE	CUSE	COSE	SDT	DTS
34h	MC_PSR										UDIS	PUH	PUL	PVH	PVL	PWH	PWL
38h	MC_OPR										ODS	UH	UL	VH	VL	WH	WL
3Ch	MC_IMR									CM0E	CPTE	OTCE	ADTE	ZPCE	CMPUE	CMPVE	CMPWE
40h	MC_DTG							Dead time register DTG[9:0] or DTG[5:0]									
44h	MC_ESC	Emergency stop clear register															
48h	MC_ECR								HRE	DOP	EMS	EPWM	EDTC	ATS1	ATS0	ESP	DUM
4Ch	MC_LOK												LOK4	LOK3	LOK2	LOK4	LOK0

Refer to *Table 5 on page 35* for the register base addresses.

14 Controller area network (CAN)

14.1 Introduction

The CAN peripheral consists of the CAN Core, Message RAM, Message Handler, Control Registers and Module Interface (Refer to *Figure 95*).

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 1 Mbit/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.

14.2 Main features

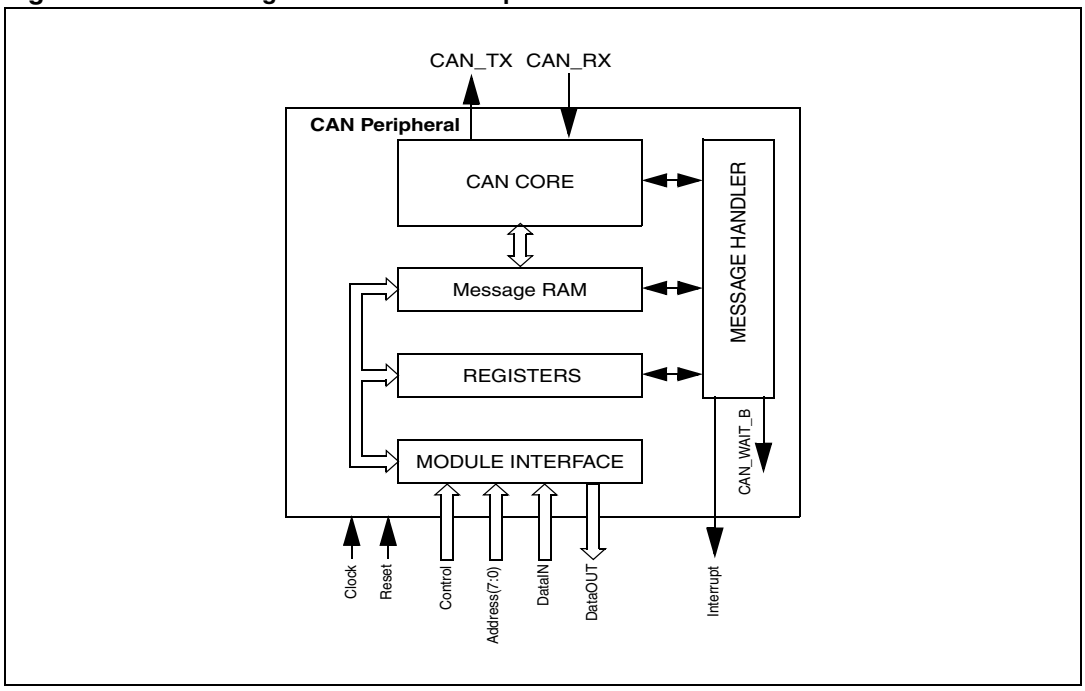
- Supports CAN protocol version 2.0 part A and B
- Bit rates up to 1 Mbit/s
- 32 Message Objects
- Each Message Object has its own identifier mask
- Programmable FIFO mode (concatenation of Message Objects)
- Maskable interrupt
- Disabled Automatic Re-transmission mode for Time Triggered CAN applications
- Programmable loop-back mode for self-test operation
- Two 16-bit module interfaces to the APB bus

14.3 Block diagram

The CAN peripheral interfaces with the AMBA APB bus. *Figure 95* 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 95. Block diagram of the CAN Peripheral



14.4 Functional description

14.4.1 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 14.7.10: Configuring the bit timing on page 411*) 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.

14.4.2 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.

14.4.3 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.

14.4.4 Test mode

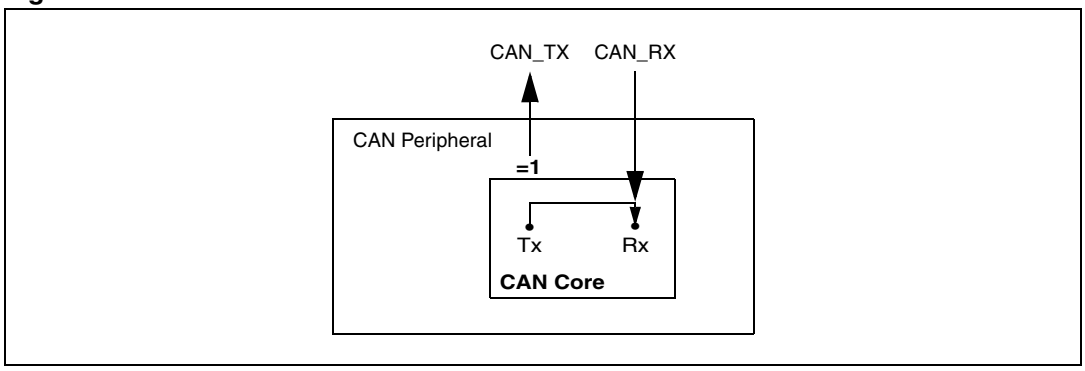
Test Mode is entered by setting the Test bit in the CAN Control Register. In Test Mode, bits Tx1, Tx0, LBack, Silent and Basic in the Test Register are writeable. Bit Rx monitors the state of the CAN_RX pin and therefore is only readable. All Test Register functions are disabled when the Test bit is 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 96* shows the connection of signals CAN_TX and CAN_RX to the CAN Core in Silent Mode.

Figure 96. CAN core in silent mode

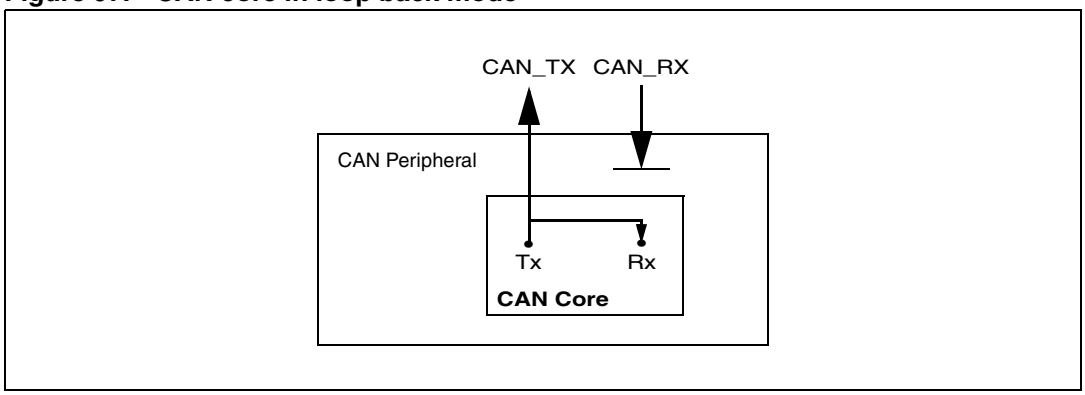


In ISO 11898-1, Silent Mode is called Bus Monitoring Mode.

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 97* shows the connection of signals, CAN_TX and CAN_RX, to the CAN Core in Loop Back Mode.

Figure 97. CAN core in loop back mode

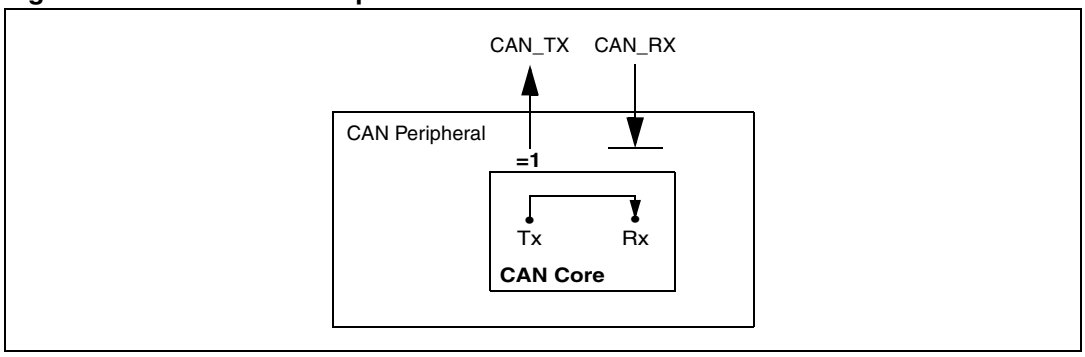


This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop Back Mode. In this mode, the CAN Core performs an internal feedback from its Tx output to 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 98* 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 98. 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.

14.5 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 40*.

Table 40. 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

14.5.1 CAN interface reset state

After the hardware reset, the CAN peripheral registers hold the reset values given in 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.

14.5.2 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* describes the error codes.

Table 41. 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	Reserved	
-	-	-	-	-	-	-	-	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.

14.5.3 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 *Message object in the message memory on page 394*) or parts of the Message Object may be transferred between the Message RAM and the IF*n* Message Buffer registers (see *IFn message buffer registers on page 391*) 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 42* 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 42. 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	ClrInt Pnd	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 the 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 is 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 cont'd	**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 *Message object in the message memory on page 394*.

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 *Message object in the message memory on page 394*.

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 *Message object in the message memory on page 394*.

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 *Message object in the message memory on page 394*.

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 *Message object in the message memory on page 394*.

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 *Message object in the message memory on page 394*.

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 43 provides an overview of the structures of a Message Object

Table 43. 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 *Acceptance filtering of received messages on page 406*.

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 14.7.7: 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.

14.5.4 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 44* 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 44. 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.

14.6 Can register map

Table 45. 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															

Table 45. CAN register map (continued)

Addr. offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	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												
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 5 on page 35* for the register base addresses.

14.7 CAN communications

14.7.1 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.

14.7.2 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.

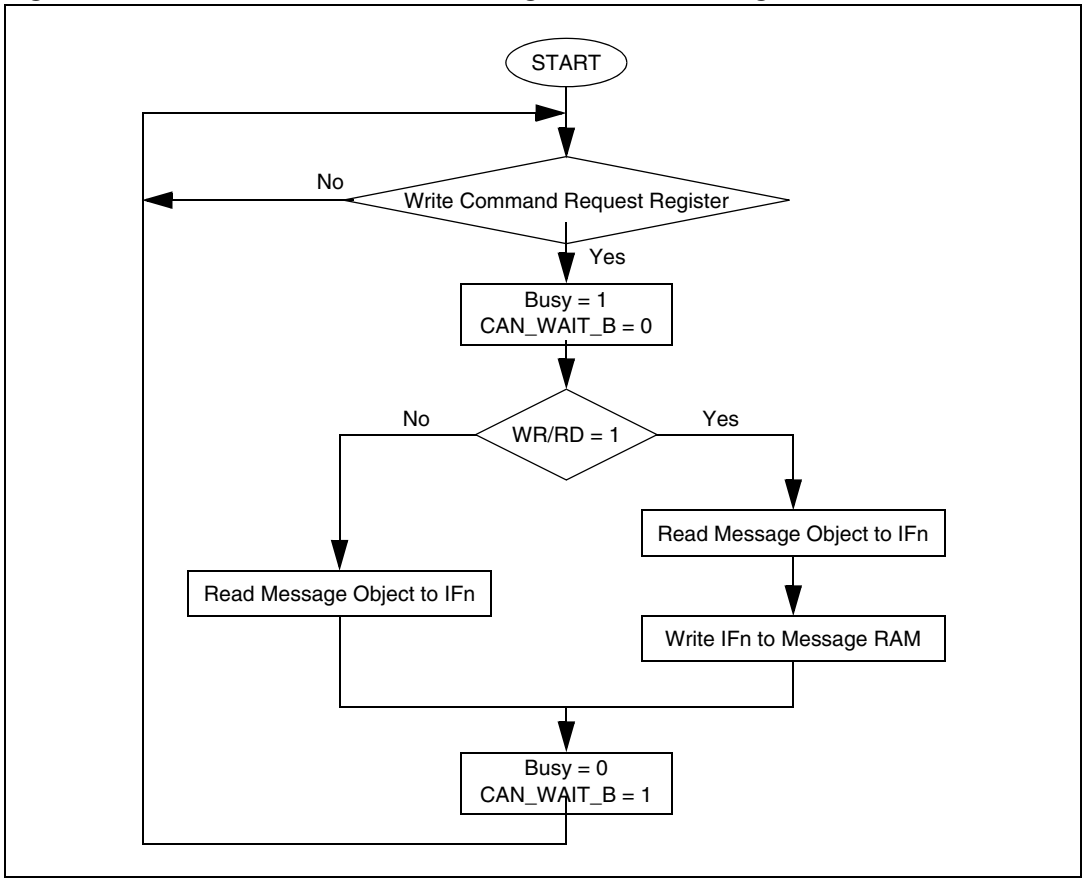
Data transfer from/to message RAM

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 99*).

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 99. 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.

14.7.3 Configuring a transmit object

*Table 46*shows how a Transmit Object should be initialized.

Table 46. 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.

14.7.4 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 *Message transmission on page 405*.

When NewDat is set together with TxRqst, NewDat will be reset as soon as the new transmission has started.

14.7.5 Configuring a receive object

Table 47 shows how a Receive Object should be initialized.

Table 47. 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.

14.7.6 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.

14.7.7 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 *Configuring a receive object on page 408*.

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.

14.7.8 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.

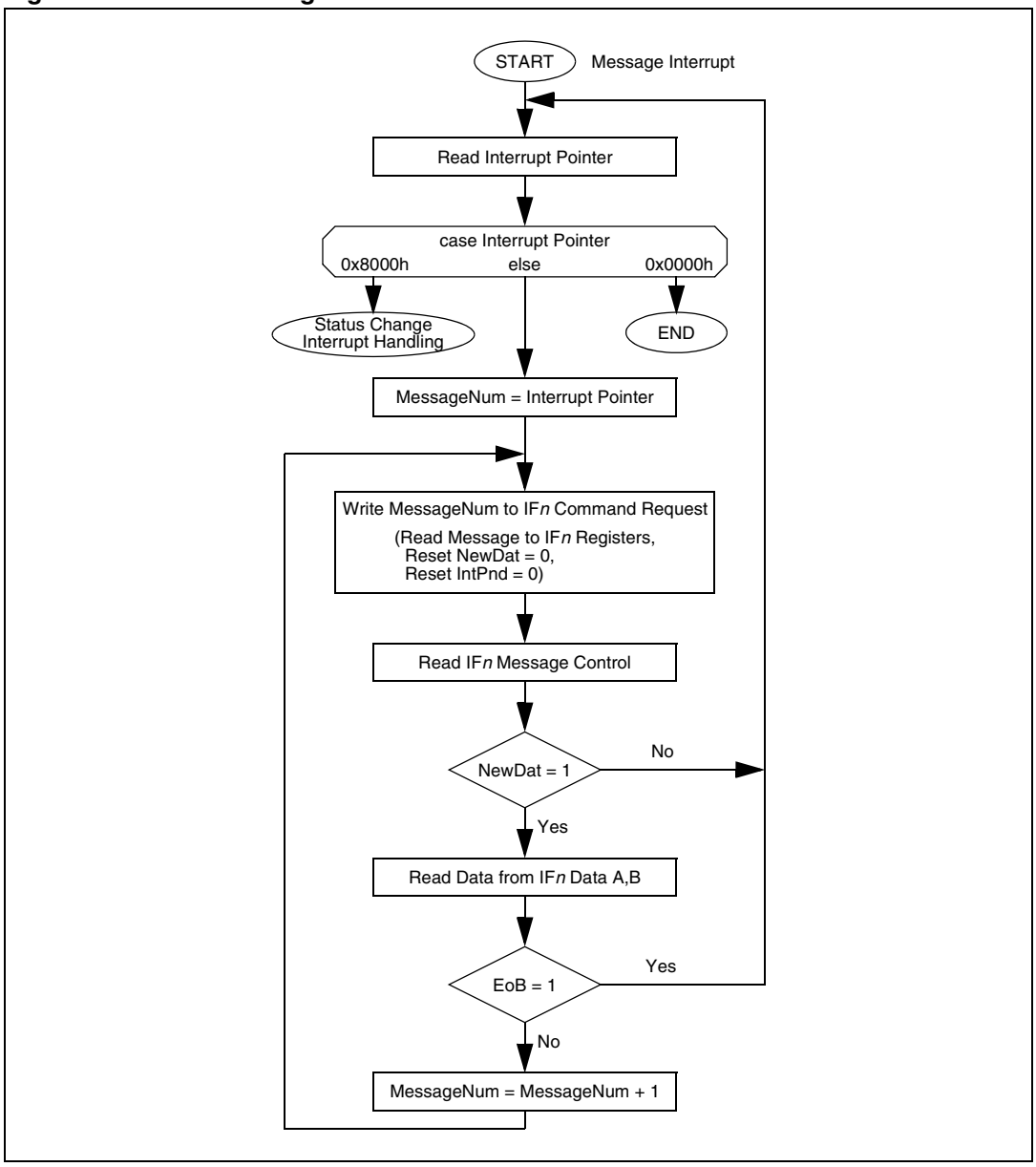
Reading from a FIFO buffer

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 100 shows how a set of Message Objects which are concatenated to a FIFO Buffer can be handled by the CPU.

Figure 100. CPU handling of a FIFO buffer



14.7.9 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 *Interrupt pending registers 1 & 2 (CAN_IPnR) on page 400*).

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.

14.7.10 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 101*). 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 48*). 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 101. Bit timing

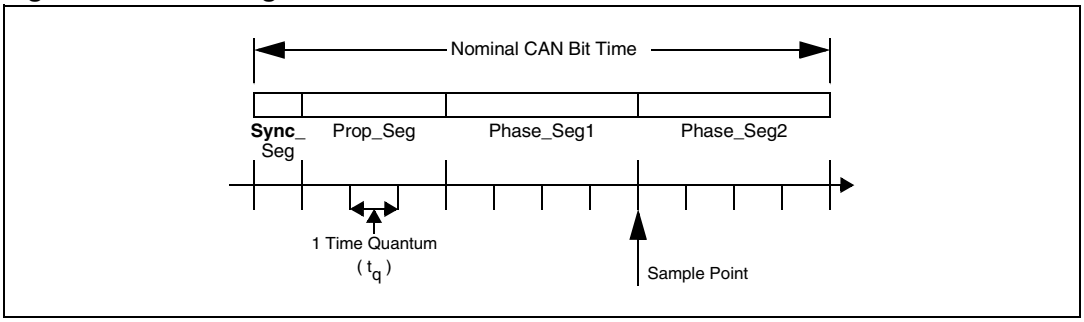


Table 48. CAN bit time parameters

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		

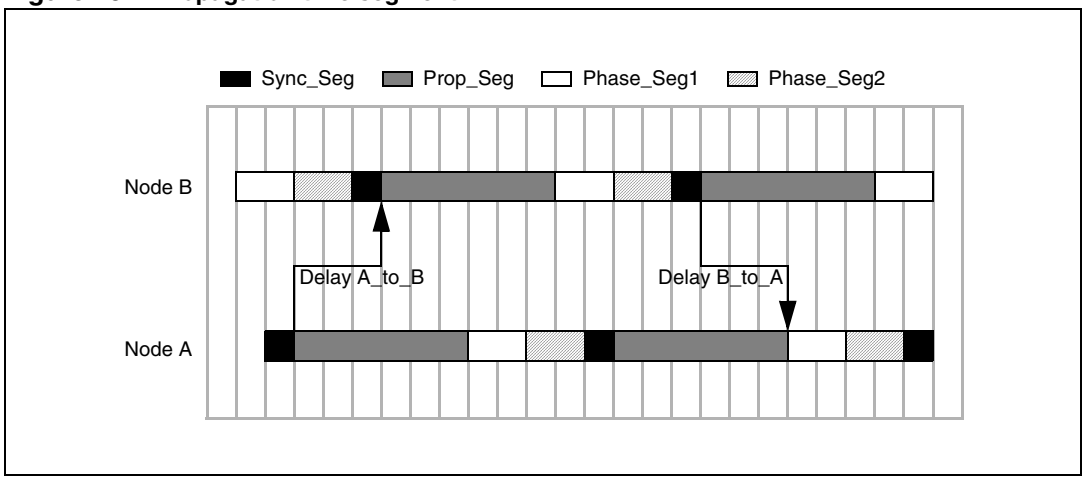
A given bit rate may be met by different bit time configurations, but for the proper function of the CAN network the physical delay times and the oscillator's tolerance range have to be considered.

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 102* shows the phase shift and propagation times between two CAN nodes.

Figure 102. Propagation time segment



1. Delay A_to_B >= node output delay(A) + bus line delay(A->B) + node input delay(B)
2. Prop_Seg >= Delay A_to_B + Delay B_to_A
3. 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.

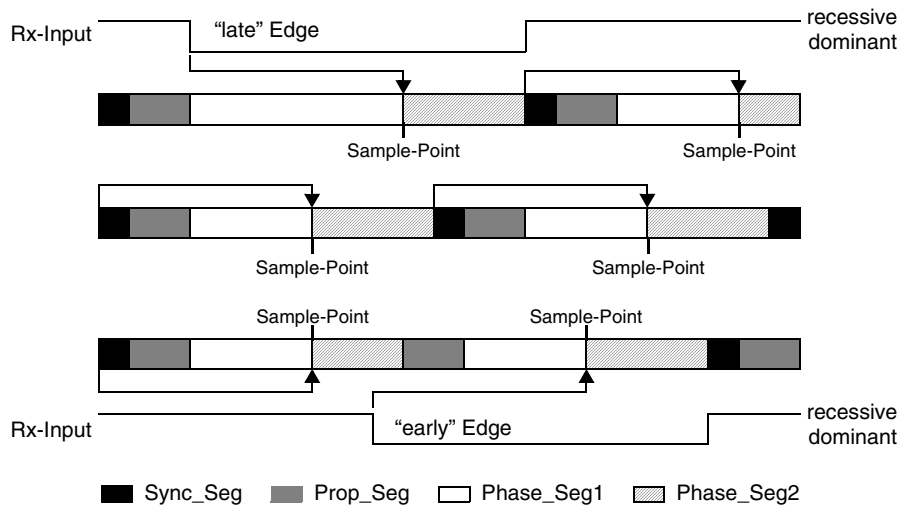
Only one synchronization may be done between two Sample Points. The Synchronizations maintain a minimum distance between edges and Sample Points, giving the bus level time to stabilize and filtering out spikes that are shorter than (Prop_Seg + Phase_Seg1).

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 103* 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 103. 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,

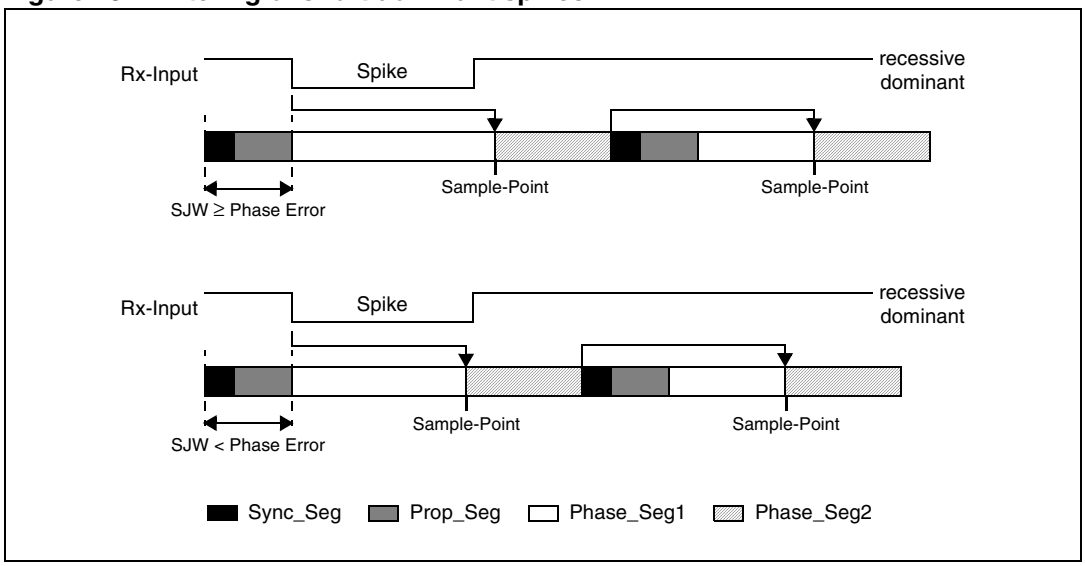
Phase_Seg2 is shortened and Sync_Seg is omitted, so that the distance from the edge to the Sample Point is the same as it would have been from an Sync_Seg to the Sample Point if no edge had occurred. As in the previous example, the magnitude of phase error of this “early” edge’s is less than SJW, so 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 104* 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 104. 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:

$$(1 - df) \bullet f_{nom} \leq f_{osc} \leq (1 + df) \bullet f_{nom}$$

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):

$$\text{I:} \quad df \leq \frac{\min(\text{Phase\_Seg1}, \text{Phase\_Seg2})}{2 \cdot (13 \cdot \text{bit\_time} - \text{Phase\_Seg2})}$$

$$\text{II:} \quad df \leq \frac{SJW}{20 \cdot \text{bit\_time}}$$

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.

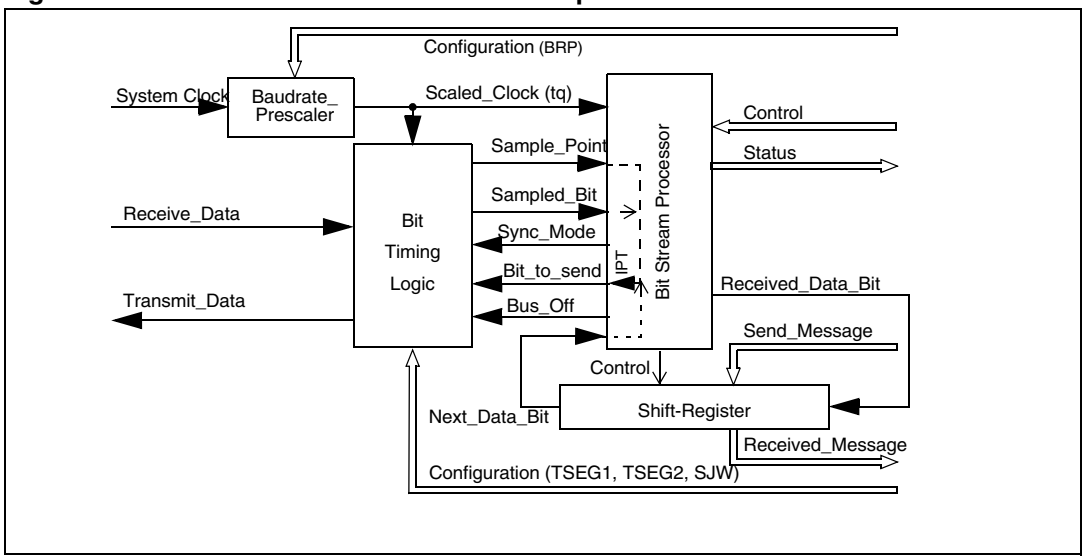
Configuring the CAN protocol controller

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 105 on page 418*).

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 105. 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 *Oscillator tolerance range on page 417*

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. 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 equal 0x1600.

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	$= 6 \bullet t_q$
t_{SJW}	100	ns	$= 1 \bullet t_q$
t_{TSeg1}	700	ns	$= t_{Prop} + t_{SJW}$
t_{TSeg2}	200	ns	= Information Processing Time + $1 \bullet t_q$
$t_{Sync\text{-}Seg}$	100	ns	$= 1 \bullet t_q$
Bit time	1000	ns	$= t_{Sync\text{-}Seg} + t_{TSeg1} + t_{TSeg2}$
Tolerance for APB_CLK	0.39	%	$= \frac{\min(PB1,PB2)}{2x(13xbit\_time\text{-}PB2)}$ $= \frac{0.1\mu s}{2x(13x1\mu s\text{-}0.2\mu s)}$

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. 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 equal 0x34C1.

t_q	1	µs	$= 2 \bullet t_{APB\_CLK}$
Delay of bus driver	200	ns	
Delay of receiver circuit	80	ns	
Delay of bus line (40m)	220	ns	
t_{Prop}	1	µs	$= 1 \bullet t_q$
t_{SJW}	4	µs	$= 4 \bullet t_q$
t_{TSeg1}	5	µs	$= t_{Prop} + t_{SJW}$
t_{TSeg2}	4	µs	= Information Processing Time + $3 \bullet t_q$
$t_{Sync\text{-}Seg}$	1	µs	$= 1 \bullet t_q$
Bit time	10	µs	$= t_{Sync\text{-}Seg} + t_{TSeg1} + t_{TSeg2}$
Tolerance for APB_CLK	1.58	%	$= \frac{\min(PB1,PB2)}{2x(13xbit\_time-PB2)}$ $= \frac{4\mu s}{2x(13x10\mu s-4\mu s)}$

15 USB slave interface (USB)

15.1 Introduction

The USB slave interface consists of both the USB Serial Interface Engine (SIE) and the USB Transceiver (Physical interface). It implements an interface between a full-speed USB 2.0 and the AHB bus. USB power management capabilities (suspend/resume) can be interfaced with STR91xFA *Low power modes* for efficient power management. The STR91xFA *DMA controller (DMAC)* can be used off-load the CPU and increase application performance. The 48 MHz USBCLK is supplied via the System Control Unit (SCU) from an internal or external clock source see *Section 2.4.6 on page 72*.

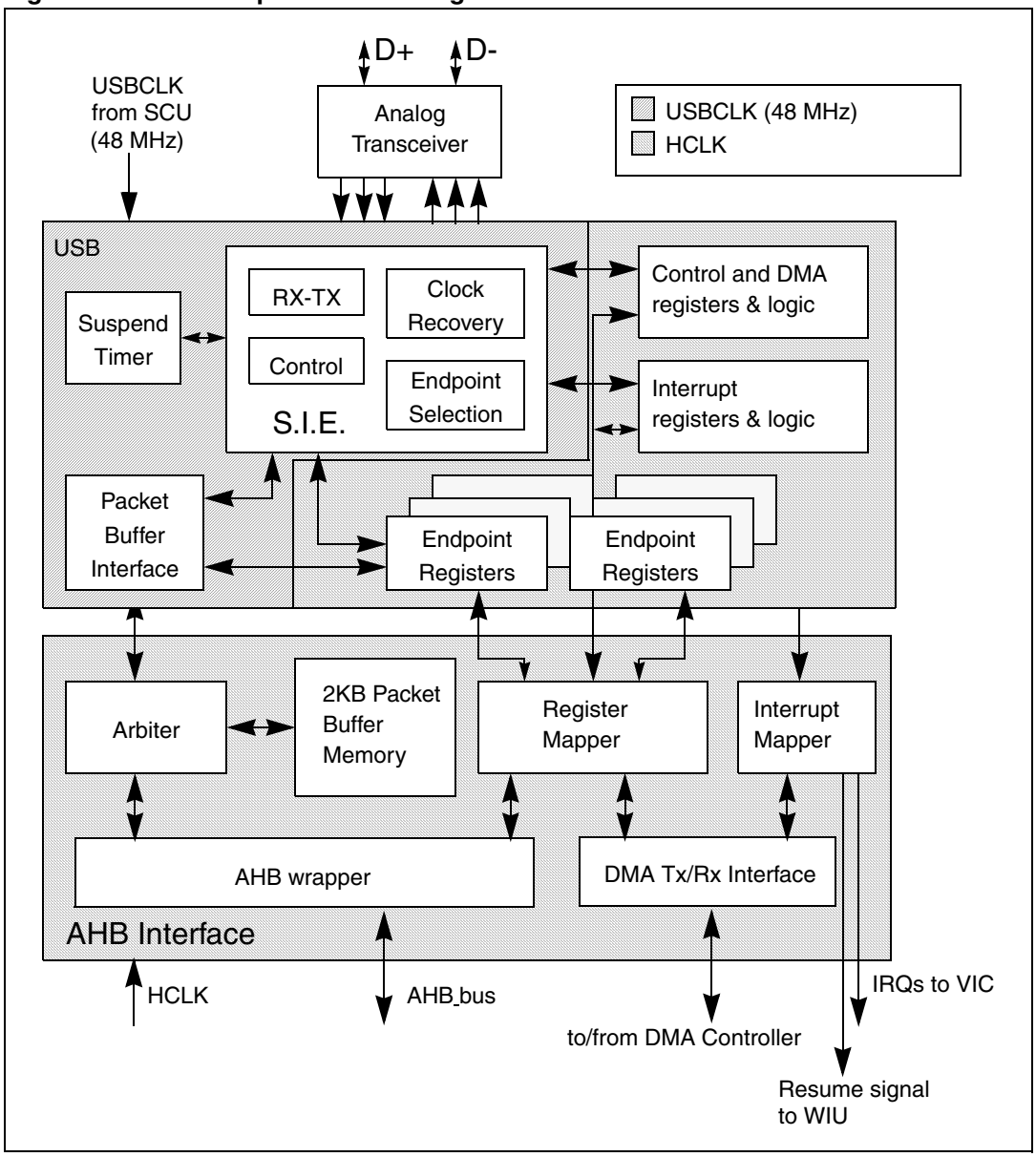
15.2 Main features

- Meets USB 2.0 Full Speed specification (12 Mbs) Slave mode
- Support up to 10 bidirectional or 20 mono-directional Endpoints
- Support Isochronous, Control, Interrupt and Bulk endpoints
- Each Endpoint is associated with two packet buffers (Tx and Rx) whose size may be up to 1024 bytes each.
- Packet Buffer Memory (2 Kb SRAM) to store the Endpoint buffers. Buffer size is user programmable.
- Support for Control EP0 with both IN and Out endpoints
- DMA controller that can be used to transfer data from the Endpoint Buffer to memory when data is transmitted or received.
- Interrupt sources to the Interrupt Controller
- USB suspend resume operations
- Located on the AHB bus
- 48 Mhz clock comes from PLL main CPU clock or external input pin

15.3 Block diagram

Figure 106 shows the block diagram of the USB Peripheral.

Figure 106. USB Peripheral block diagram



15.4 Functional description

The USB Peripheral provides a 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. This dedicated memory is 2 Kbytes in size supporting up to 20 mono-directional/single-buffered endpoints. 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
- 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 (refer to the *Table 13: VIC interrupt channels on page 124* for details).

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. The RESUME interrupt source can be connected directly to a wakeup line (see *Wakeup/Interrupt Unit (WIU) on page 136*) to allow the system to immediately restart the normal clock generation and/or support direct clock start/stop.

15.4.1 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.
- 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 determine the cause of an interrupt, read the interrupt status or clear a pending interrupt.

The USB Peripheral is connected to the AHB bus through an AHB 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 2 Kbytes, structured as 512 words by 32 bits.
- Arbiter: This block accepts memory requests coming from the AHB bus and from the USB interface. It resolves the conflicts by giving priority to AHB 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 AHB 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 AHB.
- Interrupt Mapper: This block maps the USB interrupts to IRQ lines of the VIC.
- AHB Wrapper: This provides an interface to the AHB for the memory and register. It also maps the whole USB Peripheral in the AHB address space.

15.5 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.

15.5.1 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.

15.5.2 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 the 48 MHz USBCLK and HCLK to the USB Peripheral and de-assert the specific reset signal using related control bits *Peripheral clock gating register 0 (SCU_PCGR0) on page 92*. and *Peripheral reset register 0 (SCU_PRR0) on page 96*

After that the analog part of the device related to the USB transceiver must be switched on using the PDWN bit in *USB control register (USB_CNTR)* register which requires a special handling. This bit is intended to switch on the internal voltage references supplying the port transceiver . Since this circuits have a defined start-up time, during which the behaviour of USB transceiver is not defined, it is necessary to wait this time, after having set the PDWN bit in the USB_CNTR register, then the reset condition on the USB part can be removed (clearing of FRES bit in USB_CNTR register) and the USB_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 in the *Peripheral clock gating register 0 (SCU_PCGR0) on page 92*.

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 event (see next 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.

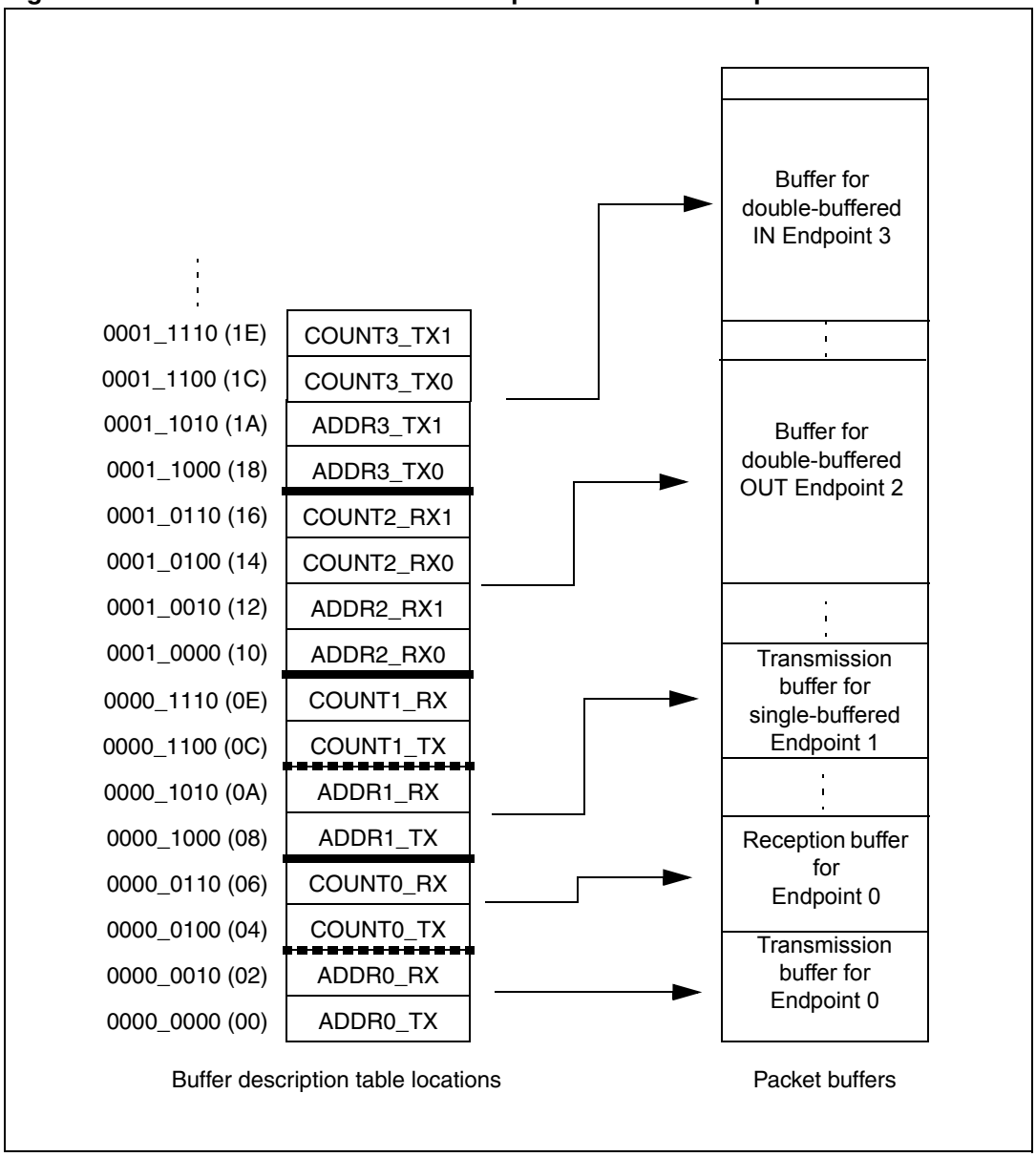
Structure and usage of packet buffers

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 AHB 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 USBCLK. The frequency of this dedicated clock is fixed by the requirements of the USB standard at 48 MHz.

Note: Due to USB data rate and packet memory interface requirements, the AHB 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 up to 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 15.6.4: 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 15.5.4: Isochronous transfers* and *Section 15.5.3: Double-buffered endpoints* respectively). The relationship between buffer description table entries and packet buffer areas is depicted in *Figure 107*.

Figure 107. 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 in the 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 *Structure and usage of packet buffers on page 427*) 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.

15.5.3 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 49. 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 50. 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.
- 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.

15.5.4 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 51*.

Table 51. 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.

15.5.5 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 52*, 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 52. Resume event detection

[RXDP,RXDM] status	Wakeup 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.

15.6 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
- DMA Registers: DMA control and configuration
- 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 register base address, except the buffer descriptor table locations, which starts at the address in packet memory specified by the USB_BTABLE register 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 the USB Peripheral register base address. See *Table 58: USB peripheral register page mapping*.

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_w0)	The software can only read or clear this bit by writing 0. Writing ‘1’ has no effect
Toggle (t)	The software can only toggle this bit by writing ‘1’. Writing ‘0’ has no effect

15.6.1 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: 840h

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	SZDPRM	Reserved		RESUME	FSUSP	LPMODE	PDWN	FRES
rw	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:** *Wakeup 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.

Bit 7	**SZDPRM:** *Short or Zero-Length Data Packet Received Mask.* 0: Short or Zero-Length Data Packet Received (SZDPR) Interrupt disabled. 1: SZDPR Interrupt enabled an interrupt request is generated when the corresponding bit in the USB_ISTR register is set. ***Note*:** When the SZDRP interrupt occurs and is not masked (SZDRPM = 1) then the DMA RX channel is automatically disabled in order to stop the DMA transfer because the programmed linked list (LLI) on the DMAC could be different from the packet size received (refer to *Section 9.4.6 on page 252*)
Bits 6:5	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: 844h

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	SZDPR	Reserved		DIR	EP_ID[3:0]			
r	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	-	-	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.

Bits 15:7 each represent a specific event. They 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. 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:** *Wakeup* 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. wakeup 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.
Bit 7	**SZDPR:** *Short or Zero-Length Received Data Packet* This bit is written by the DMA interface when a short or zero length data packet has been received and the DMA RX channel has been enabled in linked mode (see also). A short packet is received when the related COUNTn_RX register field is less than the predefined max packet size (NUM_BLOCK register field).
Bits 6:5	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: 848h

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 wakeup 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 wakeup 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: 84Ch

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 *Structure and usage of packet buffers on page 427*).
Bits 2:0	Reserved, forced by hardware to 0

15.6.2 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.

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.

USB Endpoint n register (USB_EPnR), n = [0..9]

Address offset: 800h 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	STATRX[1:0]		SETUP	EPTYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STATTX[1:0]		EA[3:0]			
rc_w0	t	t	t	r	rw	rw	rw	rc_w0	t	t	t	rw	rw	rw	rw

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 15.5.3: 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 15.5.4: 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 53: Reception status encoding on page 449*.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 15.5.3: 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 54: Endpoint type encoding on page 449*. 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 15.5.4: 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 55* 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 15.5.3: 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 15.5.3: 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 15.5.4: 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 56*. 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 15.5.3: 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 53. 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 54. Endpoint type encoding

EP_TYPE[1:0]	Meaning
00	BULK
01	CONTROL
10	ISO
11	INTERRUPT

Table 55. 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 56. 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

15.6.3 DMA registers

The DMAC has two separate request channels (Tx and Rx) dedicated to the USB peripheral see *Table 23: DMA request signal mapping on page 245*. The 10 USB endpoints can be mapped to the DMA Tx or Rx channels in Linked or Unlinked mode:

- **Linked mode:** A single Endpoint can be mapped in linked mode on the DMA channel (Tx/Rx). The DMA can prepare linked lists (LLI) in order to manage multiple data packet transfer without CPU intervention at the end of the single data packet transfer. The DMA interface provides transfer requests to the DMA controller until the LLI is completed. The CPU is only responsible for configuring the linked lists (descriptor chains) before enabling the DMA and, on termination of the DMA transfer (terminal count interrupt from the DMAC). The CTR_TX/CTR_RX interrupt (EPn registers) of the selected Endpoint (linked) is automatically cleared by the DMA interface, that masks the related source of the global CTR interrupt (USB_ISTR register). The CPU doesn't receive any CTR interrupt request for this Endpoint (linked).
  - A special case of linked lists (LLI) is represented by a chain including a single descriptor. The main difference with unlinked mode is that the descriptors related to the Tx/Rx Endpoint can be programmed independently and before the related Endpoint token (IN, OUT) is received by the USB device. The basic assumption is that the expected Endpoint and related packet size is known or assumed. In addition, the CTR_RX/CTR_TX interrupt is cleared automatically by the DMA interface.
- **Unlinked mode**: Multiple endpoints can be mapped on the channel (3 in Tx mode and 8 in Rx mode) without the use of linked lists (LLI). In this case the DMA cannot use linked lists (LLI) and only a single data packet can be transferred by the DMA controller. The CPU is responsible for configuring the new descriptor only on termination of the data transfer (DMAC terminal count interrupt) and when the new CTR_TX/CTR_RX interrupt is received. Software has to decode the Endpoint to be served before programming the next descriptor because multiple endpoints are mapped on the same channel (Tx/Rx). The DMA interface doesn't mask/clear any CTR_TX/CTR_RX interrupts (this has to be done by software). This operating mode requires CPU intervention and reduces the advantage in terms of CPU load.

Note: Control and interrupt data flow endpoints are usually managed by the CPU while DMA is used for isochronous and/or bulk pipes.

DMA control register 1 (USB_DMACR1)

Address offset: 854h

Reset value: 0000 0000 0000 0000 (0000h)

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

Bit 15:10	Reserved, forced by hardware to 0
Bits 9:0	**DMA_UNLK_RX_EN[9:0]:** *Unlinked mode Rx DMA enable* These bits are set and cleared by software. They are cleared by hardware on completion of a DMA data transfer to the corresponding Endpoint. Multiple endpoints (up to 8) can be enabled to be served by the DMA channel. In linked mode (LK_RX_EN = 1 in the USB_DMACR3 register) these bits are not used. 0: Rx DMA in unlinked mode disabled. Any data transfer from system memory to Packet Buffer Memory for the corresponding Endpoint is performed by the CPU. 1: Rx DMA in unlinked mode enabled. Any data transfer from system memory to Packet Buffer Memory for the corresponding Endpoint is performed by the DMAC.

DMA control register 2 (USB_DMACR2)

Address offset: 858h

Reset value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		DMA_UNLK_TX_EN [1:0]		UNLNK_TX_EP_ID3[3:0]				UNLNK_TX_EP_ID3[3:0]				UNLNK_TX_EP_ID3[3:0]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:14	Reserved, forced by hardware to 0
Bits 13:12	**DMA_UNLK_TX_EN[1:0]:** *Unlinked mode Tx DMA enable* These bits are set and cleared by software. In linked mode (LK_TX_EN = 1 in the USB_DMACR3 register) these bits are not used. 00: Tx DMA in unlinked mode disabled 01: Tx DMA in unlinked mode enabled to serve the Endpoint selected by the UNLK_TX_EPID1[3:0] bits. 10: Tx DMA in unlinked mode enabled to serve the Endpoint selected by the UNLK_TX_EPID2[3:0] bits. 11: Tx DMA in unlinked mode enabled to serve the Endpoint selected by the UNLK_TX_EPID3[3:0] bits.
Bits 11:8	**UNLK_TX_EP_ID3[3:0]:** *Unlinked mode Tx Endpoint ID 3* These bits can be set and cleared by software only when the DMA Tx interface is disabled. They select the Tx endpoints configured in unlinked mode. In linked mode (LK_TX_EN = 1 in the USB_DMACR3 register) these bits are not used. 0000: Endpoint 0 0001: Endpoint 1 .... 1001: Endpoint 9 Other values reserved
Bits 7:4	**UNLK_TX_EP_ID2[3:0]:** *Unlinked mode Tx Endpoint ID 2* These bits can be set and cleared by software only when the DMA Tx interface is disabled. They select the Tx endpoints configured in unlinked mode. In linked mode (LK_TX_EN = 1 in the USB_DMACR3 register) these bits are not used. 0000: Endpoint 0 0001: Endpoint 1 .... 1001: Endpoint 9 Other values reserved
Bits 3:0	**UNLK_TX_EP_ID1[3:0]:** *Unlinked mode Tx Endpoint ID 1* These bits can be set and cleared by software only when the DMA Tx interface is disabled. They select the Tx endpoints configured in unlinked mode. In linked mode (LK_TX_EN = 1 in the USB_DMACR3 register) these bits are not used. 0000: Endpoint 0 0001: Endpoint 1 .... 1001: Endpoint 9 Other values reserved

DMA control register 3 (USB_DMACR3)

Address offset: 85Ch

Reset value: 0000 0000 0000 0000 (0000h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SR_RX	SR_TX	DMA_LK_RX_EN	LK_RX_EP_ID[3:0]				LK_RX_EN	Reserved	SLE	DMA_LK_RX_EN	LK_TX_EP_ID[3:0]				LK_TX_EN
w	w	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bit 15	**SR_RX:** *DMA Rx Software reset* This bit is write only 0: No effect 1: Reset the DMA Rx interface state machine. The DMA Rx interface is disabled and configured in idle state. The DMA configuration registers are unchanged.
Bit 14	**SR_TX:** *DMA Tx Software reset* This bit is write only 0: No effect 1: Reset the DMA Tx interface state machine. The DMA Rx interface is disabled and configured in idle state. The DMA configuration registers are unchanged.
Bit 13	**DMA_LK_RX_EN**: *Linked mode Rx DMA enable* This bit is set and cleared by software. It is cleared by hardware on completion of a DMA data transfer (DMA TC interrupt) to the selected Endpoint. It is also cleared by hardware if the Rx Endpoint has been programmed in linked mode for a data transfer with unknown total length (bit SZDPRM = 1 in the USB CNTR register) and the SZDPR flag in the USB_ISTR register is set. In unlinked mode (LK_RX_EN = 0) this bit is not used. In linked mode only a single Endpoint (LK_RX_EP_ID) can be managed by the DMA channel 0: Rx DMA in linked mode disabled. Any data transfer from system memory to Packet Buffer Memory of the selected Endpoint is performed by the CPU. 1: Rx DMA in linked mode enabled to serve the Endpoint selected by the LK_RX_EP_ID[3:0] bits. Any data transfer from system memory to Packet Buffer Memory of the selected Endpoint is performed by the DMAC.
Bits 12:9	**LK_RX_EP_ID[3:0]:** *Linked mode Rx Endpoint ID.* These bits can be set and cleared by software only when the DMA Rx interface is disabled. They select the Rx Endpoint configured in linked mode. In unlinked mode (LK_RX_EN = 0) these bits are not used. 0000: Endpoint 0 0001: Endpoint 1 .... 1001: Endpoint 9 Other values reserved

Bit 8	**LK_RX_EN**: *Rx Linked mode enable* This bit is set and cleared by software.This bit is used to configure the Rx channel selected by the LK_RX_EP_ID[3:0] bits. 0: Rx Linked mode off 1: Rx linked mode configured for the Rx channel selected by the LK_RX_EP_ID[3:0] bits.
Bit 7	Reserved, forced by hardware to 0.
Bit 6	**SLE**: *Synchronization Logic enable* This bit is set and cleared by software. It is used to insert/bypass the synchronization logic (double stage register) on the input signals generated by the DMA controller. inserted. It is equivalent to the corresponding bit in the *Synchronization register (DMA_SYNC) on page 264*. You must use synchronization logic when the USB peripheral runs on a different clock to the DMAC. If the USB peripheral runs on the same clock as the DMAC, disabling the synchronization logic improves the DMA request response time. 0: Enable synchronization logic for USB peripheral DMA Request Signal 1: Disable synchronization logic for USB peripheral DMA Request Signal
Bit 5	**DMA_LK_TX_EN**: *Linked mode Tx DMA enable* This bit is set and cleared by software. It is cleared by hardware on completion of a DMA data transfer (DMA TC interrupt) to the selected Endpoint. In unlinked mode (LK_RX_EN = 0) this bit is not used. In linked mode only a single Endpoint (LK_RX_EP_ID) can be managed by the DMA channel. ***Note*:** In Tx mode the buffers to be transmitted by the USB slave device should be ready before the USB slave device receives the USB host requests. After the reset, when the DMAC is able to load the Tx data buffers, the CPU enables the DMA Tx interface for the selected Tx Endpoint (LK_TX_EP_ID). During the initialization phase the DMA receives the proper requests to transfer single or double packets depending on the Tx Endpoint configuration (single or double buffer). The Tx buffer initialization phase is transparent for the CPU. On the completion of the DMA data transfer (DMAC TC terminal count interrupt) sofware can configure a new LLI for the next pipe transfer with a different Tx configuration (LK_TX_EP_ID Endpoint, COUNT_TX packet size). If the double buffer scheme is adopted, the DMAC should be able to complete the data transfer before the USB slave device transmits the data packet, otherwise to simplify the software management it's recommended to adopt a single buffer configuration scheme. 0: Tx DMA in linked mode disabled. Any data transfer from the Packet Buffer Memory of the selected Endpoint to system memory is performed by the CPU. 1: Tx DMA in linked mode enabled to serve the Endpoint selected by the LK_TX_EP_ID[3:0] bits. Any data transfer from the Packet Buffer Memory of the selected Endpoint to system memory is performed by the DMAC.

Bits 4:1	**LK_TX_EP_ID[3:0]:** *Linked mode Tx Endpoint ID* These bits can be set and cleared by software only when the DMA Tx interface is disabled. They select the Tx Endpoint configured in linked mode. In unlinked mode (LK_TX_EN = 0) these bits are not used. 0000: Endpoint 0 0001: Endpoint 1 .... 1001: Endpoint 9 Other values reserved
Bit 0	**LK_TX_EN**: *Tx Linked mode enable* This bit is set and cleared by software.This bit is used to configure the Tx channel selected by the LK_TX_EP_ID[3:0] bits. 0: Tx Linked mode off 1: Tx linked mode configured for the Tx channel selected by the LK_TX_EP_ID[3:0] bits.

DMA burst size register (USB_DMABSIZE)

Address offset: 860h

Reset value: 0000 0001 0000 0000 (0100h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LLI_RX_NPACKETS[7:0]								Res.	DBSIZE			Res.	SBSIZE		
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw		rw	rw	rw

Bits 15:8	**LLI_RX_NPACKETS[7:0]** *LLI Rx Number of Packets* These bits are written by software to indicate the number of packets to be received for each single descriptor of the LLI in Rx linked mode. In unlinked mode this register is unused. If the total transfer length (pipe length) is unknown, on the completion of the DMA transfer (short packet received) this field includes the number (less or equal than the initial value) of packets transferred in the last served descriptor of the LLI. If the last served descriptor contains only a short packet the LLI_RX_NPACKETS[7:0] value is equal to 1. This information with the LLI_RX_LNG and the COUNT_RX register fields could be used by CPU to determine the real data transfer size when using linked reception without known length (short packet). See also *Section 9: DMA controller (DMAC) on page 243* for more details. If the total transfer length is known, this field is not needed, it gives the number of packets (with max packet size) transferred (equal to the initial value).
Bits 6:4	**DBSIZE** *Destination Burst Size* These bits are written by software to indicate the number of transfers which make up a destination burst. The same value must be written in the corresponding field in the *Channel control register x (DMA_CCx) on page 268* (the software application must ensure these fields are in line). If the amount of data left to transfer is less than the burst size, a burst including only the pending transfers (less than DBSIZE) is performed. The DBSIZE field must be programmed less or equal to the max packet size of the related Tx Endpoint enabled for the DMA transfer. 000: single transfer 001: 4 transfers 010: 8 transfers 011: 16 transfers 100: 32 transfers 101: 64 transfers 110: 128 transfers 111: 256 transfers

Bit 3	Reserved, forced by hardware to 0
Bits 2:0	**SBSIZE** *Source Burst Size* These bits are written by software to indicate the number of transfers which make up a source burst. The same value must be written in the corresponding field in the *Channel control register x (DMA_CCx) on page 268* (the software application must ensure these fields are in line). If the amount of data left to transfer is less than the burst size, a burst including only the pending transfers (less than SBSIZE) is performed. The SBSIZE field must be programmed less or equal to the max packet size of the related Rx Endpoint enabled for the DMA transfer. 000: single transfer 001: 4 transfers 010: 8 transfers 011: 16 transfers 100: 32 transfers 101: 64 transfers 110: 128 transfers 111: 256 transfers

DMA LLI register (USB_DMALLI)

Address offset: 864h

Reset value: 0000 0000 0000 0000 (0000h)

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

Bits 15:8	**LLI_RX_LNG[7:0]** *LLI Rx Length* These bits are written by software to indicate the number of descriptors (descriptor chain length) used to program the LLI (linked list item) on the DMA Rx channel. When the DMA Rx channel is programmed in unlinked mode (LK_RX_EN = 0) this register field is unused. The value 0 is forbidden when the DMA Rx channel is programmed in linked mode (LK_RX_EN = 1), otherwise the behavior is unpredictable. The valid range (write) for this field is 1 up to 255.
Bits 7:0	**LLI_TX_LNG[7:0]** *LLI Tx Length* These bits are written by software to indicate the number of descriptors (descriptor chain length) used to program the LLI (linked list item) on the DMA Tx channel. When the DMA Tx channel is programmed in unlinked mode (LK_TX_EN = 0) this register field is unused. The value 0 is forbidden when the DMA Rx channel is programmed in linked mode (LK_TX_EN = 1), otherwise the behavior is unpredictable. The valid range (write) for this field is 1 up to 255.

15.6.4 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 the USB macrocell and the STR91xF. All packet memory locations are accessed by the AHB using 16-bit aligned addresses. In the following pages, the actual memory address is always used. The first packet memory location is located at offset 0x800 (see *Section 15.6.5: USB peripheral register page maping*).

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 *Structure and usage of packet buffers on page 427*.

Packet buffer address n (USB_ADDRn)

Address offset: [USB_BTABLE] + n*8

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADDRn_RX[15:2]														0	0
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
ADDRn_TX[15:2]														0	0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		

Bits 31:18	**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.
Bits 17:16	These bits must always be written as ‘0’ since packet memory is word-wide and all packet buffers must be word-aligned.
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.
Bits 1:0	These bits must always be written as ‘0’ since packet memory is word-wide and all packet buffers must be word-aligned.

Packet byte count n (USB_COUNTn)

Address offset: [USB_BTABLE] + n*8 + 4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BL SIZE	NUMBLOCK					COUNTn_RX[9:0]									
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						COUNTn_TX[9:0]									
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The most significant half word of this 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 31	**BL_SIZE:** *Block SIZE.* This bit selects the size of the memory block used to define the allocated buffer area. – If BL_SIZE = 0, the memory block is 4-bytes wide. The allocated area is equal to the NUM_BLOCK multiplied by 2, if the NUM_BLOCK is an even number. The allocated area is equal to the (NUM_BLOCK-1) multiplied by 2, if the NUM_BLOCK is an odd number. This is due to the fact that the minimum block allowed is 4 bytes which implies that the NUM_BLOCK should be superior or equal to 2. With this block size the allocated buffer size ranges from 4 to 60 bytes. – If BL_SIZE = 1, the memory block is 32-bytes wide. With this block size the allocated buffer size ranges from 32 to 992 bytes. The maximum allowed block is 30.
Bits 30:26	**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* .
Bits 25:16	**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.
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.

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] (BUFFER 1)									
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] (BUFFER 0)									
rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	r	r

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] (BUFFER 1)									
-	-	-	-	-	-	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] (BUFFER 0)									
-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Table 57. 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’)	Not allowed	64 bytes
2 (‘00010’)	4 bytes	96 bytes
3 (‘00011’)	4 bytes	128 bytes
...	...	...
15 (‘01111’)	28 bytes	512 bytes
16 (‘10000’)	32 bytes	544 bytes
17 (‘10001’)	32 bytes	576 bytes
18 (‘10010’)	36 bytes	608 bytes
...	...	...
30 (‘11110’)	60 bytes	992 bytes
31 (‘11111’)	60 bytes	Not allowed

15.6.5 USB peripheral register page maping

Table 58 shows the mapping of all USB Peripheral registers and the Packet Buffer Memory.

Table 58. USB peripheral register page mapping

Address offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000 - 0x7FC	Packet Buffer Memory 2 Kbytes																
0x800	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]			
0x804	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]			
0x808	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]			
0x80C	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]			
0x810	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]			
0x814	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]			
0x818	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]			
0x81C	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]			
0x820	USB_EP8R	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]			
0x824	USB_EP9R	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]			
0x840	USB_CNTR	CTRM	DOVRM	ERRM	WKUPM	SUSPM	RESETM	SOFM	ESOFM	SZDPRM	Reserved		RESUME	FSUSP	LP MODE	PDWN	FRES
0x844	USB_ISTR	CTR	DOVR	ERR	WKUP	SUSP	RESET	SOF	ESOF	SZDPR	Reserved		DIR	EP_ID[3:0]			
0x848	USB_FNR	RXDP	RXDM	LCK	LSOF[1:0]		FN[10:0]										
0x84C	USB_DADDR	Reserved								EF	ADD[6:0]						

Table 58. USB peripheral register page mapping (continued)

Address offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x850	USB_BTABLE	BTABLE[15:3]													Reserved		
0x854	USB_DMACR1	DMA Control Register 1															
0x858	USB_DMACR2	DMA Control Register 2															
0x85C	USB_DMACR3	DMA Control Register 3															
0x860	USB_DMABSIZE	DMA Burst Size Register															
0x864	USB_DMALLI	DMA Linked List Item Register															

Refer to *Table 5 on page 35* for the register base addresses.

16 Analog-to-digital converter (ADC)

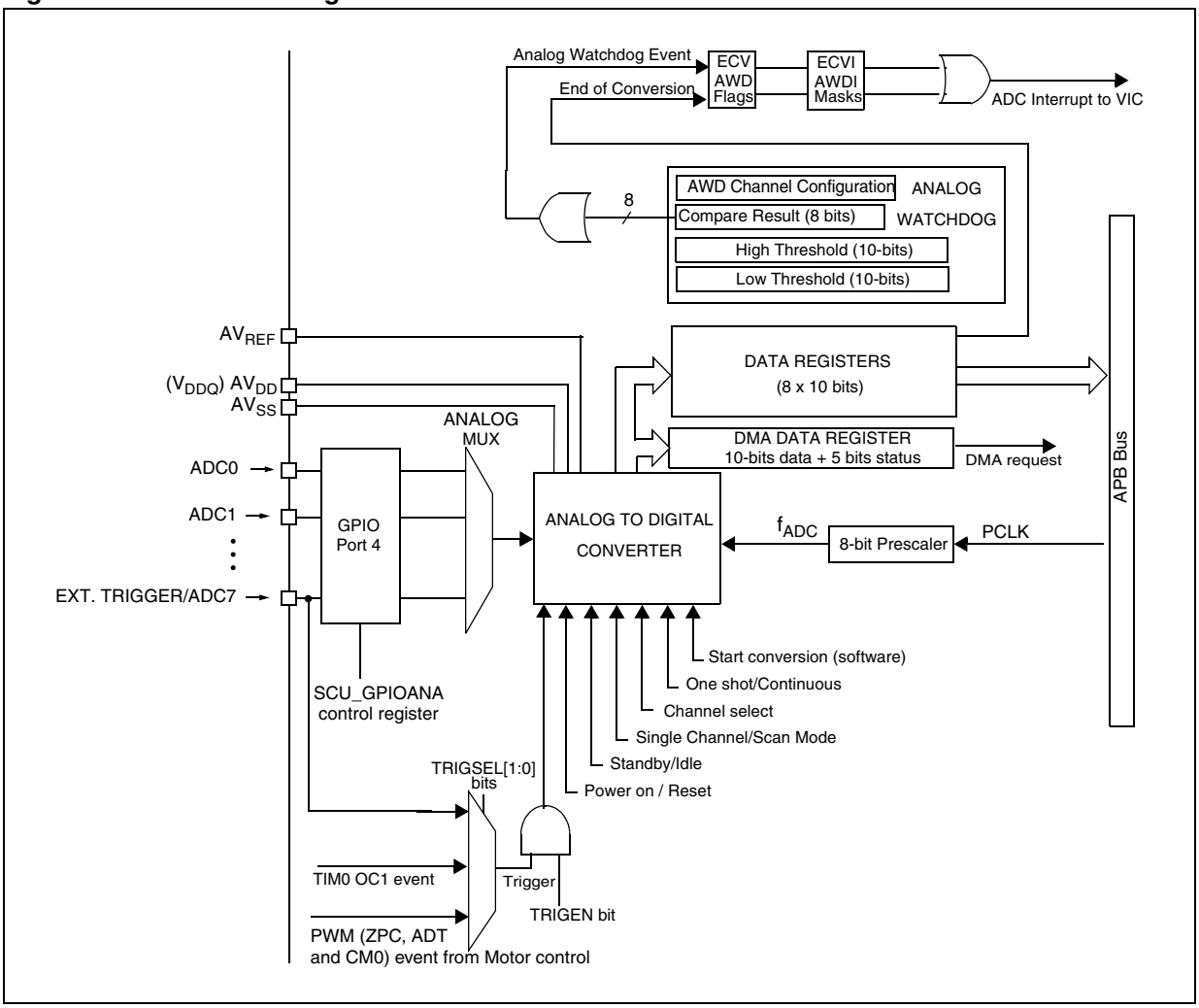
16.1 Main characteristics

- ADC clock derives from PCLK through a 8-bit frequency prescaler
- Resolution: 10 bits
- 8 input channels
- 0 to 3.6 V input range
- Single channel/ scan modes (converts one or all of 8 channels successively without any software interaction).
- One-Shot or continuous conversion
- Standby mode for low power consumption
- Analog watchdog with interrupt generation (when the converted value is above or below a threshold previously programmed by software).
- DMA support
- Start conversion can be triggered by software or by external pin, timer event or Motor control PWM event.
- Fast trigger mode (in Rev H devices)
- Sample Input every 16 ADC clocks (4 clocks for sampling and 12 clocks for successive approximation).

16.2 Introduction

The Analog-to-Digital Converter (ADC) comprises an input multiplexed channel selector feeding a successive approximation converter. The conversion resolution is 10 bits.

Figure 108. ADC block diagram



16.2.1 Clock prescaler

The conversion time depends on the ADC clock frequency The ADC clock is the PCLK divided by the prescaler factor stored in the ADC_PRS register.

You can change the conversion time by modifying the prescaling factor. Conversion time specified in the STR91xF 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.

16.2.2 Interrupts

The ADC can generate three maskable interrupt requests:

- ECV (End of Conversion) interrupt request
- AWD (Analog watchdog) interrupt request
- OVERRUN (DMA overrun) interrupt request

The logical OR of all previous requests is provided to the VIC.

Before returning from serving the interrupt, the ISR typically clears the interrupt by setting the corresponding EVC, AWD or OVERRUN flag bit in the ADC_CR register to ‘0’.

The ADC clock is used to clear the interrupt flags. The time it takes to clear the flags is longer when the ADC clock frequency is lower. There are situations where the CPU returns from interrupt routine and the interrupt flag has not been cleared yet. Since the Interrupt Controller input is level sensitive, the CPU will see it as another interrupt. For this reason it is recommended to clear the ADC flags at the start of the interrupt subroutine, instead of at the end.

16.2.3 DMA

The ADC can store conversion results in SRAM using the DMAC.

To configure the ADC in DMA mode, set the DMAEN bit in *ADC control register 2 (ADC_CR2)*. In this mode, all conversion results are stored in the *ADC DMA data register (ADC_DDR)* and a DMA request is generated to the DMAC at each conversion.

If the DMA does not read the results before the next conversion, the data in the DMA data register will be overwritten. In this case, the new result will be flagged with the overrun bit (OR).

The ADC DMA request shares channel 9 of the DMA with external DMA request number 1 (GPIO3.1).

When the ADC DMA feature is enabled, external DMA request number 1 is blocked.

Overrun flag

If the DMA is not able to read the data from the DMA data register (ADC_DDR) before a new data is written to it, an OVERRUN bit will be set and an interrupt is sent to the MCU unless it is disabled by setting the ORD bit in the *ADC control register 2 (ADC_CR2) on page 477*.

The interrupt and the OR bit are cleared when the ADC_DDR DMA data register is read. The DMA data register is read by either the CPU or the DMA Controller.

Note: If the DMAC reads the register before the OR interrupt is served, the DMAC read will also clear the OR bit.

16.3 External pins

The converter uses a fully differential analog input configuration for the best noise immunity and precision performance. Depending on the package size of the microcontroller, the AVREF voltage pin can be used for improved accuracy. Refer to *Figure 20: Power supply overview on page 66* and *Section 2.1.2: Independent A/D converter supply and reference voltage on page 67* for more information.

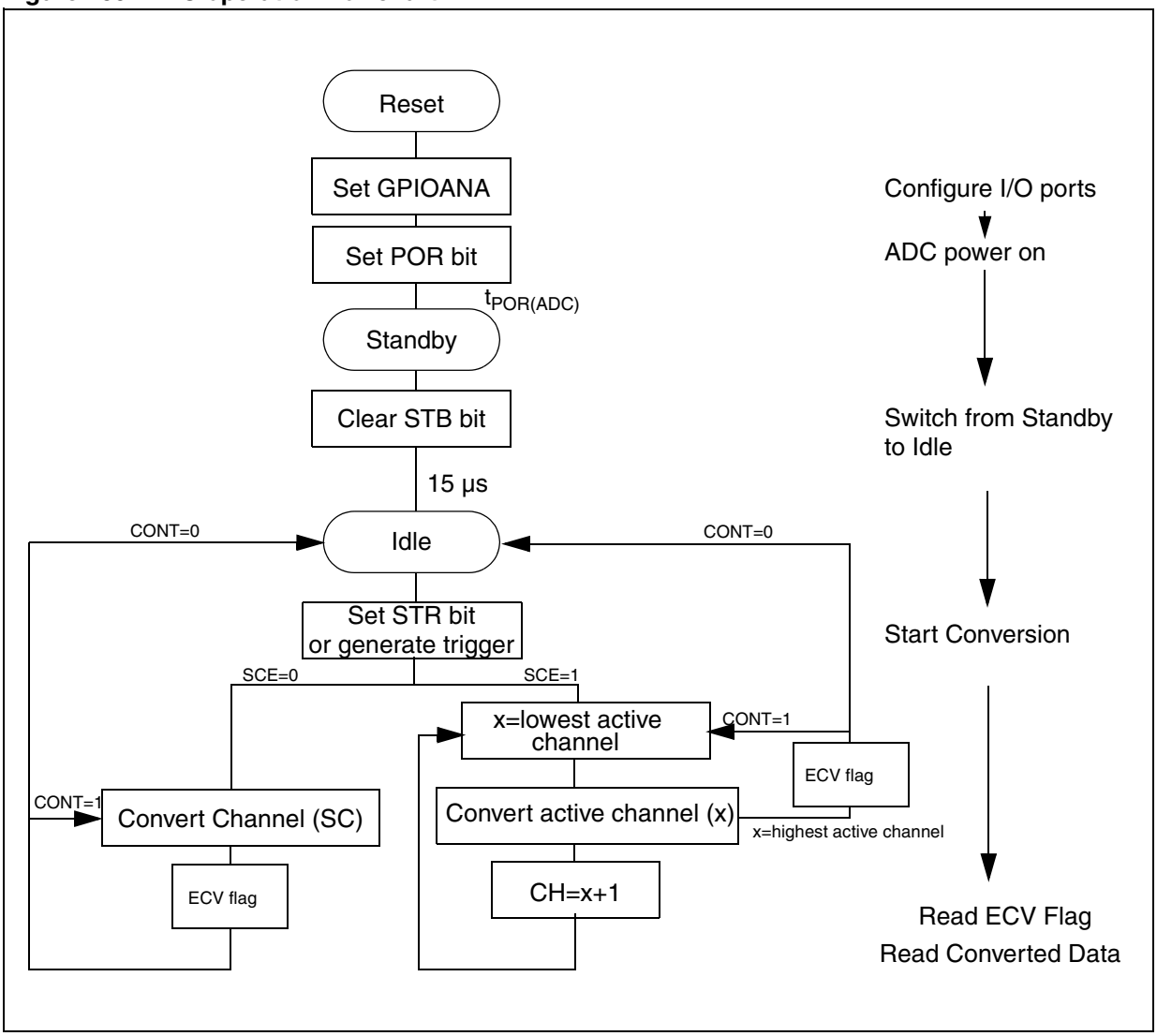
The converted digital value is referred to the analog reference voltage which determines the full-scale converted value. Of course, analog and digital grounds MUST be common (to be tied together externally).

Up to 8 multiplexed analog inputs are available. The eight analog input pins are connected to GPIO port 4. After reset, you have to configure the GPIOs as analog inputs by programming the *GPIO analog mode register (SCU_GPIOANA) on page 113*. and programming the *Channel configuration register (ADC_CCR)* to configure the channels. The SC[2:0] bits in the *ADC control register (ADC_CR)* must be programmed to select a channel for single conversion.

ADC7 can be used as external trigger if enabled in the *ADC control register 2 (ADC_CR2) on page 477*.

16.4 Functional description

Figure 109. ADC operation flowchart



16.4.1 Conversion modes

Two principal operating modes are available: Single Channel Mode and Scan Mode. You select these modes using the SCE bit in the Control Logic Register (ADC_CR).

Single channel mode

In Single Channel Mode (SCE = 0) a single channel selected by the SC[2:0] bits in the ADC_CR register is performed. At the end of the conversion:

- The digital result of the conversion (overflow status and result) is stored in the corresponding data register.
- The ECV flag is set and the ECV interrupt is generated if the ECVI bit = 1.
- If the analog watchdog is enabled, the AWD flag is updated (see *Section 16.4.5* for details). A interrupt is generated if the AWDI bit = 1.

Scan mode

In Scan mode (SCE = 1) all the channels configured as active for conversion (CCx[1:0] bits > 00b in the *Channel configuration register (ADC_CCR)*) are converted from the lowest active channel to the highest active channel. At the end of conversion of each channel:

- The digital result of the conversion (overflow status and result) is stored in the corresponding data register.
- If the analog watchdog is enabled, the AWD flag is updated (see *Section 16.4.5* for details). A interrupt is generated if the AWDI bit = 1.

At the end of the conversion of the last active channel:

- The ECV flag is set and the ECV interrupt is generated if the ECVI bit = 1.

One-shot/continuous modes

You can run single channel or scan mode in one-shot or continuous mode.

- In One-shot mode, the sequences described above for *Single channel mode* and *Scan mode* are run once and the STR bit is cleared by hardware.
- In Continuous mode, the sequences, described above *Single channel mode* and *Scan mode* are run until the STR bit is cleared by software.

16.4.2 Power management

Reset mode

In Reset mode, the ADC is stalled, the analog part of the ADC is switched off and digital part is held in reset state. The ADC cell is in zero power consumption mode. This mode can be used:

- To perform a software reset of the ADC
- As a power saving mode if the ADC is not used

At reset, the ADC is in Reset mode.

To switch from Reset mode to Standby mode, set the POR bit and the STB bit in the ADC_CR register. The ADC is switched on and enters Standby mode after $t_{POR(ADC)}$.

You can also switch directly to Idle mode by setting the POR bit and keeping the STB bit at 0.

Standby / idle mode

You can put the ADC in Standby mode to reduce power consumption when A/D conversion is not required. Otherwise, when the ADC is not converting, it is Idle mode.

To switch from Idle to Standby mode, set the STB bit. The ADC enters Standby mode at the next clock pulse.

To switch the ADC from Standby to Idle mode, clear the STB bit in the ADC_CR register. The ADC is fully powered on after 15 µs.

If STB is cleared and STR is set at the same time, the first conversion is delayed by 15 µs. If STB is cleared and STR is set before $t_{POR(ADC)}$ after POR is set, the first conversion is delayed by $t_{POR(ADC)}$).

16.4.3 Starting conversion

To start a conversion by software, set the STR bit in the ADC_CR register. Refer to the flowchart in *Figure 109*.

It is also possible to start conversion using an external trigger. The ADC configuration is the same as for starting a conversion by software except you have to set up the ADC Control Register 2 (ADC_CR2), using this procedure:

1. Select one of the 3 trigger sources (external pin, TIM or PWM) using the TRIGSEL bits.
   - ***Trigger using external pin***: In this case, application hardware controls the signal input on the P4.7 pin. The default input function of this pin is "ADC External Trigger". ADC conversions will be triggered on falling or rising edges of this input signal.
   - ***Trigger using TIM0 OC1 event***: The TIM0 OCMP1 feature can operate either in PWM mode or in output compare mode. If you choose to use TIM0 in output compare mode, be aware that ADC conversions are triggered on falling and rising edges of the output compare signal. Consequently, your software should ensure that the OC signal contains rising and falling edges. For example, this can be done by configuring Timer0 interrupt and by clearing OLVL1 bit if it is set and setting it if it is cleared in the Timer0 interrupt routine. In this case, when a match is found, Timer0 interrupt is generated and the pin OCMP1 will toggle and the ADC conversion will be triggered.
   - ***Trigger using PWM motor control***: If you select PWM as the trigger source, you also need to specify one of the three events that generate the trigger in the IMC_ECR register.
2. Choose the polarity of the external pin if needed with ETE bit.
3. Enable the trigger with the TRIGEN bit.

There is no need to set the start bit (STR) in the ADC_CR register to initiate a conversion. The ADC will start the conversion whenever the selected trigger event becomes activated.

To disable external trigger mode, or change the trigger configuration (polarity or trigger source), the following procedure must be followed:

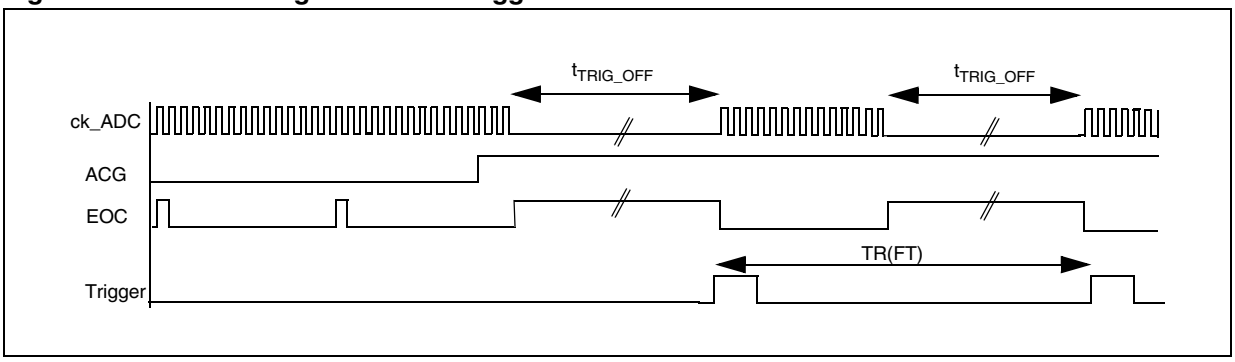
1. Select the default value of trigger selection. writing TRIGSEL bits to "00".
2. Disable trigger with TRIGEN bit.

The trigger event is synchronized with the rising edge of ADC clock (configured by ADC Prescaler Register). A minimum of two rising edges of the ADC clock must occur between two consecutive active triggers. Consequently the ADC clock frequency must be double the trigger frequency (external pin, TIM or PWM).

16.4.4 Fast trigger conversion in single mode

When trigger mode is enabled, a specific configuration can be used to provide a faster conversion time in single mode and cycle accurate synchronous data available after each trigger. Fast trigger mode is selected when the ACG bit is set in the *GPIO analog mode register (SCU_GPIOANA)*. The ADC clock will start on each trigger event for 16 clock cycles and will automatically stopped when the digital result of the conversion is stored in the corresponding data register.

Figure 110. ADC clock gated in Fast trigger conversion mode



In this mode, the ADC clock stops when no conversion is in progress, if this idle period (t_{TRIG}) is too long, the accuracy of the first subsequent conversion may be out of specification.

To avoid this limitation, the following two rules should be respected:

1. Do not keep the ADC clock stopped for a period longer than the $t_{CK\_OFF(ADC)}$ maximum value specified in the datasheet. t_{TRIG_OFF} must be less than $t_{CK\_OFF(ADC)}$.
2. Minimum Trigger Throughput Rate is limited to TR(FT) (see datasheet for the minimum value).

Note: This feature is not available in device Rev G or earlier devices (see datasheet for silicon revision information).

16.4.5 Analog watchdog

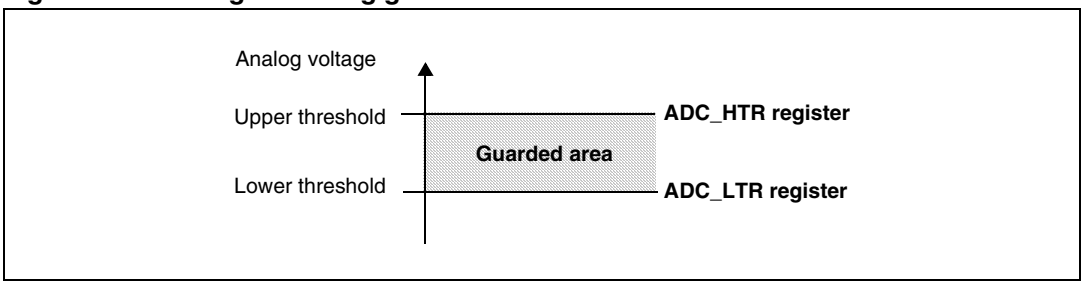
A programmable watchdog is available for analog threshold detection.

The low and high thresholds of the guarded area are selected by the ADC_HTR and ADC_LTR registers See *Figure 111*.

You configure the analog watchdog event individually for each channel using the CCx[1:0] bits in the ADC_CCR register.

After conversion of the selected channel is finished, a comparison is performed between the current channel and, depending on the CCx[1:0] bits, the threshold value in ADC_HTR or ADC_LTR. The compare result is stored in the ADC_CRR register, and, depending on the AWDI mask bit in the ADC_CR register, an AWD interrupt request is generated if the converted value has crossed the threshold.

Figure 111. Analog watchdog guarded area



16.5 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
Read/clear (rc_w1)	Software can read as well as clear this bit by writing ‘1’. Writing ‘0’ has no effect on the bit value

16.5.1 ADC control register (ADC_CR)

Address offset: 00h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ECV	AWD	Reserved			ECVI	AWDI	SC[2:0]			SCE	CONT	STB	res.	POR	STR
rc_w1	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, forced by hardware to 0
Bit 15	**ECV:** *End of conversion flag* This bit is set by hardware and cleared by software writing 1. 0: No end of conversion event 1: End of conversion. You can read the ADC_DRx registers to retrieve the result An interrupt request is generated if ECVI = 1.
Bit 14	**AWD:** *Analog Watchdog flag* 0: No analog watchdog event 1: An analog watchdog event occurred. You can read the ADC_CRR register to determine the result for each channel. An interrupt request is generated if AWDI = 1.
Bits 13:12	Reserved, forced by hardware to 0
Bit 11	Reserved, must be kept at reset value 0
Bit 10	**ECVI:** *End of conversion interrupt enable* 0: ECV interrupt disabled 1: ECV interrupt enabled
Bit 9	**AWDI:** *Analog watchdog interrupt enable* 0: AWD interrupt disabled 1: AWD interrupt enabled
Bits 8:6	**SC[2:0]:** *Selected channel to be converted* These bits are written by software to select the channel to be converted. The selection applies only when Scan mode is disabled (SCE=0). The channel to be converted must also be configured as active in the ADC_CCR register. 000: Channel 0 001: Channel 1 ... 111: Channel 7

Bit 5	**SCE:** *Scan mode enable* This bit is set and cleared by software. 0: Single channel mode. The channel selected by the SC[2:0] bits is enabled 1: Scan mode. All channels configured as active in the ADC_CCR register are converted.
Bit 4	**CONT:** *Continuous mode enable* This bit is set and cleared by software. 0: One shot mode: if SCE = 0 the channel selected by the SC[2:0] bits is converted once. The STR bit is cleared automatically and the ECV bit is set at the end of conversion. If SCE = 1 all channels are converted once. The STR bit is cleared automatically and the ECV bit is set at the end of conversion. 1: Continuous mode, if SCE = 0 the channel selected by the SC[2:0] is converted continuously. If SCE = 1 all channels are converted continuously. The STR bit must be cleared by software to stop conversion.
Bit 3	**STB:** *Standby mode enable* This bit is set and cleared by software. 0: Idle mode. The analog block is kept powered on 1: Standby mode enabled. The analog block is put in low power mode ***Note*:** When STB is cleared, the first conversion can start after 15 µs.
Bit 2	Reserved, must be kept at reset value 0.
Bit 1	**POR:** *Power on/ Reset mode* This bit is set and cleared by software. 0: Reset mode. The analog block is switched off, all registers are reset. Write access to all registers disabled except POR bit. 1: Power on mode. ADC digital block is running. Analog block in Idle mode or Standby mode depending on the STB bit. ***Note*:** When POR is set, the first conversion can start only after $t_{POR(ADC)}$
Bit 0	**STR:** *Start Conversion* This bit is set and cleared by software. 0: Stop Conversion The ADC returns to Idle state at the next clock pulse. 1: Start conversion The first conversion starts after up 3 x 16 ADC clock cycles (synchronously with analog block).

16.5.2 Channel configuration register (ADC_CCR)

Address offset: 04h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC7[1:0]		CC6[1:0]		CC5[1:0]		CC4[1:0]		CC3[1:0]		CC2[1:0]		CC1[1:0]		CC0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16	Reserved, forced by hardware to 0
Bits 15:0	**CCx[1:0]:** *Channel x Configuration (x = 7:0)* These bits are written by software to configure the corresponding ADC input channel. 00: No A/D conversion or watchdog feature on channel x 01: Active for A/D conversion, analog watchdog event configured to trigger when the converted result on channel x is greater than the low threshold (CDATA > LT). 10: Active for A/D conversion, analog watchdog configured to trigger when the converted result on channel x is less than the high threshold (CDATA < HT). 11: Active for A/D Conversion without watchdog feature on channel x

16.5.3 High threshold register (ADC_HTR)

Address offset: 08h
Reset value: 0000h

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

Bits 31:10	Reserved, must be kept at reset value 0
Bits 9:0	**HT[9:0]:** *Analog Watchdog High Threshold* These bits are written by software to define the high threshold value for the analog watchdog (see *Figure 111*).

16.5.4 Low threshold register (ADC_LTR)

Address offset: 0Ch
Reset value: 0000h

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

Bits 31:10	Reserved, must be kept at reset value 0
Bits 9:0	**LT[9:0]:** *Analog Watchdog Low Threshold* These bits are written by software to define the low threshold value for the analog watchdog (see *Figure 111*).

16.5.5 Compare result register (ADC_CRR)

Address offset: 10h
Reset value: 0000h

7	6	5	4	3	2	1	0
CR[7:0]							
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:8	Reserved, forced by hardware to 0
Bits 7:0	**CR[7:0]**: *Compare Result for Channel x* These bits are set by hardware when a watchdog event occurs on the corresponding channel. They are cleared by software, by writing ‘1’ in the corresponding bit. Writing this register also clears the AWD interrupt flag in the ADC_CR register, if all CR bits are cleared. When the CCx[1:0] bits in the ADC_CCR register are at "00" or "11" (watchdog disabled) then CRx is forced to '0'. 0: No analog watchdog event occurred on channel x 1: Analog watchdog event occurred on channel x (as configured in the ADC_CCR register).

16.5.6 ADC data register (ADC_DRx)

There are eight ADC data registers (x can be a value from 0 to 7)
Address offset: 14h...30h
Reset value: 0000h

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

Bits 31:16	Reserved, forced by hardware to 0
Bit 15	**OV:** *Channel x Overflow status* This bit is updated by hardware after each conversion. 0: No overflow on this channel 1: A conversion overflow occurred on this channel
Bits 14:10	Reserved, forced by hardware to 0
Bits 9:0	**CDATA[9:0]:** *Channel x Converted Data* The conversion results for the eight available channels are loaded into the eight different data registers following conversion of the corresponding analog input.

16.5.7 ADC prescaler register (ADC_PRS)

Address offset: 34h
Reset value: 00FFh

7	6	5	4	3	2	1	0
PRS[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:8	Reserved, forced by hardware to 0
Bits 7:0	**PRS[7:0]:** *ADC Prescaler* These bits are written by software to define the ADC clock prescaling factor. 00h: f_{ADC}=PCLK 01h: f_{ADC}=PCLK 02h: f_{ADC}=PCLK/2 .. FFh: f_{ADC}=PCLK/255

16.5.8 ADC DMA data register (ADC_DDR)

Address offset: 38h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OV	Res	OR	CHANNELID			CDATA[9:0]									
r		r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16	Reserved, forced by hardware to 0
Bit 15	**OV:** *Conversion Overflow status* This bit is updated by hardware after each conversion. 0: No overflow 1: A conversion overflow occurred overflow bit (the ADC analog input exceeds the analog reference value).
Bits 14	Reserved, forced by hardware to 0
Bit 13	**OR:** *DMA Overrun status* This bit is updated by hardware after each conversion. It is reset by hardware when this register is read. 0: No overrun 1: A DMA overrun occurred (the DMAEN bit is set and the DMA did not read the previous converted result).
Bits 12:10	**CHANNEL_ID**: *Channel ID status* These bits contain the number of the converted channel. 000: ADC0 .... 111: ADC7
Bits 9:0	**CDATA[9:0]:** *Channel Converted Data* The conversion results are loaded into the DMA data register after each conversion.

16.5.9 ADC control register 2 (ADC_CR2)

Address offset: 3Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										ETE	ORD	DMA EN	TRIG EN	TRIGSEL[1:0]	
										rw	rw	rw	rw	rw	rw

Bits 31:6	Reserved, forced by hardware to 0
Bit 5	**ETE** *External trigger edge* This bit is set and cleared by software. It selects the trigger edge polarity. 0: Rising edge (reset value) 1: Falling edge
Bit 4	**ORD**: *Overrun interrupt disable* This bit is set and cleared by software. 0: Overrun interrupt enabled (reset value) 1: Overrun interrupt disabled
Bit 3	**DMAEN:** *DMA trigger enable* This bit is set and cleared by software. When this bit is set the ADC DMA interface is enabled. Data written to the DDR register will generate a DMA request which triggers the DMA to read the conversion result from the ADC. 0: DMA disabled 1: DMA enabled
Bit 2	**TRIGEN:** *Trigger enable* This bit is set and cleared by software 0: Trigger disabled 1: Trigger enabled
Bits 1:0	**TRIGSEL[1:0]:** *Trigger selection* These bits are written by software to select the trigger event. 00: No trigger (default) 01: PWM trigger 10: Timer trigger 11: External trigger pin

16.6 ADC register map

Table 59. ADC register map

Address offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	ADC_CR	ECV	AWD	Reserved			ECVI	AWDI	SC[2:0]			SCE	CONT	STB	res.	POR	STR
04h	ADC_CCR	Channel Configuration Register															
08h	ADC_HTR	Reserved						High Threshold									
0Ch	ADC_LTR	Reserved						Low Threshold									
10h	ADC_CRR	Channel Compare Result Register															
14h	ADC_DR0	OV	Reserved					Channel 0 Converted Data									
18h	ADC_DR1	OV	Reserved					Channel 1 Converted Data									
1Ch	ADC_DR2	OV	Reserved					Channel 2 Converted Data									
20h	ADC_DR3	OV	Reserved					Channel 3 Converted Data									
24h	ADC_DR4	OV	Reserved					Channel 4 Converted Data									
28h	ADC_DR5	OV	Reserved					Channel 5 Converted Data									
2Ch	ADC_DR6	OV	Reserved					Channel 6 Converted Data									
30h	ADC_DR7	OV	Reserved					Channel 7 Converted Data									
34h	ADC_PRS	Reserved								ADC Clock Prescaler							
38h	ADC_DDR	OV	Res.	OR	CHANNEL_ID			Channel Converted Data									
3Ch	ADC_CR2	Reserved										ETE	ORD	DMA EN	TRIG EN	TRIGSEL[1:0]	

Refer to *Table 5 on page 35* for the register base addresses.

17 AHB/APB bridges (APB)

The two AHB/APB bridges provide completely asynchronous connections between the AHB and APB buses. Refer to *Table 5 on page 35* for the address mapping of the peripherals connected to each bridge.

17.1 Main features

- AHB slave interface
- APB master interface
- Asynchronous AHB/APB clock domains
- Two identical APB bridges, each supporting a fixed set of peripherals
- AHB split accesses
- Time-Out condition for peripheral transactions

17.2 Split transactions

The AHB/APB clock ratio can typically be around 1/2, depending on the application. The HCLK and PCLK frequencies are programmable via the SCU registers (refer to *Figure 22: Clock control on page 71*. As a consequence, an APB read access could need more than 2 AHB cycles to be completed. To prevent the AHB being stalled, waiting for the APB access to be performed, a split mechanism is implemented. This enables the master to initiate the request, the AHB/APB bridge then releases the bus until the data is available. When the data is ready, the slave will signal the master to complete the transaction.

17.3 Error handling

The AHB bridge registers can be used to troubleshoot errors that occur when accessing the APB peripherals.

If an error occurs the bridge ends the APB transaction and reports an ERROR conditions on the AHB bus (if enabled).

17.4 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
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

17.4.1 Bridge status register (APB_BSR)

Address offset: 00h

Reset value: 0x0000 0000

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										APBT	OUTM	Reserved			ERROR
										rc_w0	rc_w0				rc_w0

Bits 31:6	Reserved, forced by hardware to 0
Bit 5	**APBT:** *APB Time-out* This bit is set by hardware and cleared by software. 0: Normal state 1: A peripheral did not answer before the time-out
Bit 4	**OUTM:** *Out of Memory* This bit is set by hardware and cleared by software. 0: Normal state 1: An access outside memory has been attempted
Bits 3:1	Reserved, forced by hardware to 0
Bit 0	**ERROR:** *Error* This bit is set by hardware and cleared by software. 0: Normal state 1: An access has been aborted because it generated an error. The type of error is flagged in bits 5:4 of this register.

17.4.2 Bridge configuration register (APB_BCR)

Address offset: 04h

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved							SPLITEN	Reserved			SPLIT_CNT[4:0]				
							rw				rw	rw	rw	rw	rw

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

Bits 31:25	Reserved, forced by hardware to 0
Bit 24	**SPLITEN:** *Split enable* This bit is set and cleared by software. 0: The bridge will provide the bus with HREADY low until the peripheral replies or a time-out occurs. 1: The bridge allows accesses to be split after the number of AHB cycles defined in SPLIT_CNT.
Bits 23:21	Reserved, forced by hardware to 0
Bits 20:16	**SPLIT_CNT[4:0]:** *Split counter* These bits are written by software. They specify the number of AHB cycles to be performed before returning a split to the arbiter. The number of cycles is comprised between 0 (immediate split) and 31 AHB cycles.
Bits 15:9	Reserved, forced by hardware to 0
Bit 8	**ERREN:** *APB Time-out* This bit is set and cleared by software. 0: If an error occurs, the bridge sets the APBT bit in the APB_BSR register, but the operation on the ARM bus terminates normally. 1: An error is generated on the ARM bus when an APB Time-out condition occurs.
Bits 7:5	Reserved, forced by hardware to 0
Bits 4:0	**TOUT_CNT[4:0]:** *Time-out counter* These bits are written by software. When they are 00000 the time-out counter is disabled, otherwise, they define the delay, in terms of APB clock periods that the bridge waits for a target completion, before asserting the time-out error.

17.4.3 Peripheral address register (APB_PAER)

Address offset: 08h

Reset value: 0x0000 0000

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

Bits 31:25	Reserved, forced by hardware to 0
Bit 24	**RW:** *Access type* This bit is set and cleared by hardware. It indicates the type of access that generated the error condition flagged in the APB_BSR register. 0: Read access 1: Write access
Bits 23:0	PERIPHERAL_ADDRESS[23:0]: *Peripheral address* These bits are read only. They give the address of the slave that generated the error condition flagged in the APB_BSR register.

17.5 AHB/APB bridge register map

Table 60. Bridge 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	APB_BSR	Reserved																										APBT	OUTM	Reserved			ERROR
04h	APB_BCR	Reserved							SPLITEN	Reserved			SPLIT_CNT					Reserved							ERREN	Reserved			TOUT_CNT				
08h	APB_PAER	Reserved							RW	PERIPHERAL_ADDRESS[23:0]																							

Refer to *Table 5 on page 35* for the base addresses.

18 Revision history

Table 61. Document revision history

Date	Revision	Changes
13-Dec-2007	1	Created new document RM0006 to replace UM0388 and restart revision numbering. Changes compared to the last revision of UM00388 (Rev 1 dated 15-May-2007) Modified description of *2.5.2: Special interrupt run mode on page 77* Added Note on use of VBATT with LVD in *Section 2.1.3 on page 67* Added ACG bit in *GPIO analog mode register (SCU_GPIOANA) on page 113* Updated *Figure 36 on page 163* Updated description of ECKEN bit in *Section 7.6.6: Control register 1 (TIM_CR1) on page 177* Modified *Section 10.3.4: Slave Select management on page 278* Updated *Section 16.4.4: Fast trigger conversion in single mode on page 469*
02-Mar-2008	2	Added *Idle Mode entry timing on page 78* Modified SSPCLK in *Figure 22: Clock control on page 71* and *Figure 67: SSP block diagram on page 276* Updated SSP *Section 10.4.5: Clock ratios on page 280*. Added note in *Section 12.5.4: I2C clock control register (I2C_CCR) on page 337*. Modified ADC *Section 16.4.3: Starting conversion on page 469*
21-Apr-2008	3	Removed DMA feature from *Section 12: I2C interface module (I2C)*
03-Jul-2009	4	Converted document to new template. *Section 1.12.3: External memory interface (EMI) configuration/control*: Updated configuration of the EMI address port and chip select pins. *Section 1.12.6: Timing rules*: Replaced WSTOEN with WSTWEN. *Section 1.12.5: EMI bus timing configuration* and *Section 1.12.7: Bus mode configuration*: Replaced WSTEN with WSTWEN. *Section 2.2.1* and *Section 2.2.2*: Updated definition of system reset and global reset respectively to account for the RTC and SCU registers. *Figure 22: Clock control*: Replaced “SSPCLK” and “BRCLK” with “BRCLK”; Updated the PHYSEL/f_{OSC} and RTSEL/f_{RTC} And gates. *Section 2.4.4*: Replaced “Baud rate clock (BRCLK)” with “UART and SSP clock (BRCLK)”. *Section 2.5.3*: Replaced “RTC alarm interrupt” with “RTC alarm event”; added *Idle Mode exit timing*. *Section 2.5.4*: Added *Sleep Mode exit timing*. *Section 5.6*: Amended to account for the fact that “Normally close/Tamper open” is not supported. *Section 5.9.3*, *Section 5.9.4*, *Section 5.9.5*, and *Section 5.9.6*: Reset values updated from 0000h. *Section 5.9.4: RTC control register (RTC_CR)*: Bit “C” replaced by “RTCSEL”; TM bit updated to account for the fact that “Normally close/Tamper open” is not supported.

Table 61. Document revision history

Date	Revision	Changes
03-Jul-2009	4 cont'd	*Section 6.3.3: Programming considerations*: Added. *Section 8.4.21: MAC control register (ENET_MCR)*: Updated DRO bit description. *Section 15.4* and *Section 15.4.1*: Updated size (2 Kbytes), number of words (512) and number of bits (32) of dedicated packet buffer memory. *Texas Instruments synchronous serial frame format* in *Section 10.4.7*: Added a note. *Figure 107: Packet buffer areas with examples of buffer description table locations*: Modified buffer description table locations. *Section 15.6.4*: Updated offset of first packet memory location (0x800). *Packet byte count n (USB_COUNTn)* in *Section 15.6.4*: Updated bit 31 concerning the width of the memory block. *Figure 57: Definition of allocated buffer memory*: Updated the memory allocated when BL_SIZE = 0 and when BL_SIZE = 1.

Index

G

I

M

R

S

T

U

V

W

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com