Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	ARM® Cortex®-M4
Core Size	32-Bit Single-Core
Speed	120MHz
Connectivity	CANbus, Ethernet, IrDA, MMC/SD, SPI, UART/USART, USB
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	79
Program Memory Size	1MB (1M x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	128K x 8
Voltage - Supply (Vcc/Vdd)	1.62V ~ 3.6V
Data Converters	A/D 16x12b; D/A 2x12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	100-TFBGA
Supplier Device Package	100-TFBGA (9x9)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atsam4e16cb-cn

11.6.4.8 LDREX and STREX

Load and Store Register Exclusive.

Syntax

```
LDREX{cond} Rt, [Rn {, #offset}]
STREX{cond} Rd, Rt, [Rn {, #offset}]
LDREXB{cond} Rt, [Rn]
STREXB{cond} Rd, Rt, [Rn]
LDREXH{cond} Rt, [Rn]
STREXH{cond} Rd, Rt, [Rn]
```

where:

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register for the returned status.

Rt is the register to load or store.

Rn is the register on which the memory address is based.

offset is an optional offset applied to the value in *Rn*.

If *offset* is omitted, the address is the value in *Rn*.

Operation

LDREX, LDREXB, and LDREXH load a word, byte, and halfword respectively from a memory address.

STREX, STREXB, and STREXH attempt to store a word, byte, and halfword respectively to a memory address. The address used in any Store-Exclusive instruction must be the same as the address in the most recently executed Load-exclusive instruction. The value stored by the Store-Exclusive instruction must also have the same data size as the value loaded by the preceding Load-exclusive instruction. This means software must always use a Load-exclusive instruction and a matching Store-Exclusive instruction to perform a synchronization operation, see “Synchronization Primitives” .

If an Store-Exclusive instruction performs the store, it writes 0 to its destination register. If it does not perform the store, it writes 1 to its destination register. If the Store-Exclusive instruction writes 0 to the destination register, it is guaranteed that no other process in the system has accessed the memory location between the Load-exclusive and Store-Exclusive instructions.

For reasons of performance, keep the number of instructions between corresponding Load-Exclusive and Store-Exclusive instruction to a minimum.

The result of executing a Store-Exclusive instruction to an address that is different from that used in the preceding Load-Exclusive instruction is unpredictable.

Restrictions

In these instructions:

- Do not use PC
- Do not use SP for *Rd* and *Rt*
- For STREX, *Rd* must be different from both *Rt* and *Rn*
- The value of *offset* must be a multiple of four in the range 0–1020.

Condition Flags

These instructions do not change the flags.

Examples

```
MOV     R1, #0x1            ; Initialize the ‘lock taken’ value try
LDREX   R0, [LockAddr]      ; Load the lock value
CMP     R0, #0              ; Is the lock free?
```

11.6.11.17 VMOV Two ARM Core Registers to Two Single Precision

Transfers two consecutively numbered single-precision registers to and from two ARM core registers.

Syntax

```
VMOV{cond} Sm, Sm1, Rt, Rt2
VMOV{cond} Rt, Rt2, Sm, Sm
```

where:

cond — is an optional condition code, see “Conditional Execution” .

Sm — is the first single-precision register.

Sm1 — is the second single-precision register.
This is the next single-precision register after *Sm*.

Rt — is the ARM core register that *Sm* is transferred to or from.

Rt2 — is the The ARM core register that *Sm1* is transferred to or from.

Operation

This instruction transfers:

- The contents of two consecutively numbered single-precision registers to two ARM core registers.
- The contents of two ARM core registers to a pair of single-precision registers.

Restrictions

- The restrictions are:
- The floating-point registers must be contiguous, one after the other.
- The ARM core registers do not have to be contiguous.
- *Rt* cannot be PC or SP.

Condition Flags

These instructions do not change the flags.

11.6.12.4 DSB

Data Synchronization Barrier.

Syntax

```
DSB{cond}
```

where:

cond is an optional condition code, see “Conditional Execution” .

Operation

DSB acts as a special data synchronization memory barrier. Instructions that come after the DSB, in program order, do not execute until the DSB instruction completes. The DSB instruction completes when all explicit memory accesses before it complete.

Condition Flags

This instruction does not change the flags.

Examples

```
DSB ; Data Synchronisation Barrier
```

XN and Strongly-ordered rules always apply to the System Control Space regardless of the value of the ENABLE bit.

When the ENABLE bit is set to 1, at least one region of the memory map must be enabled for the system to function unless the PRIVDEFENA bit is set to 1. If the PRIVDEFENA bit is set to 1 and no regions are enabled, then only privileged software can operate.

When the ENABLE bit is set to 0, the system uses the default memory map. This has the same memory attributes as if the MPU is not implemented. The default memory map applies to accesses from both privileged and unprivileged software.

When the MPU is enabled, accesses to the System Control Space and vector table are always permitted. Other areas are accessible based on regions and whether PRIVDEFENA is set to 1.

Unless HFNMIENA is set to 1, the MPU is not enabled when the processor is executing the handler for an exception with priority –1 or –2. These priorities are only possible when handling a hard fault or NMI exception, or when FAULTMASK is enabled. Setting the HFNMIENA bit to 1 enables the MPU when operating with these two priorities.

The SAMPLE, EXTEST and BYPASS functions are implemented. In SWD/JTAG debug mode, the ARM processor responds with a non-JTAG chip ID that identifies the processor. This is not IEEE 1149.1 JTAG-compliant.

It is not possible to switch directly between JTAG Boundary Scan and SWJ Debug Port operations. A chip reset must be performed after JTAGSEL is changed.

A Boundary-scan Descriptor Language (BSDL) file to set up the test is provided on www.atmel.com.

12.6.9.1 JTAG Boundary-scan Register

The Boundary-scan Register (BSR) contains a number of bits which correspond to active pins and associated control signals.

Each SAM4 input/output pin corresponds to a 3-bit register in the BSR. The OUTPUT bit contains data that can be forced on the pad. The INPUT bit facilitates the observability of data applied to the pad. The CONTROL bit selects the direction of the pad.

For more information, please refer to BDSL files available for the SAM4 Series.

Configuring the RTPRES field value to 0x8000 (default value) corresponds to feeding the real-time counter with a 1Hz signal (if the slow clock is 32.768 kHz). The 32-bit counter can count up to 2^{32} seconds, corresponding to more than 136 years, then roll over to 0. Bit RTTINC in the “Real-time Timer Status Register” (RTT_SR) is set each time there is a prescaler roll-over (see Figure 14-2)

The real-time 32-bit counter can also be supplied by the 1Hz RTC clock. This mode is interesting when the RTC 1Hz is calibrated (CORRECTION field ≠ 0 in RTC_MR) in order to guaranty the synchronism between RTC and RTT counters.

Setting the RTC1HZ bit in the RTT_MR drives the 32-bit RTT counter from the 1Hz RTC clock. In this mode, the RTPRES field has no effect on the 32-bit counter.

The prescaler roll-over generates an increment of the real-time timer counter if RTC1HZ = 0. Otherwise, if RTC1HZ = 1, the real-time timer counter is incremented every second. The RTTINC bit is set independently from the 32-bit counter increment.

The real-time timer can also be used as a free-running timer with a lower time-base. The best accuracy is achieved by writing RTPRES to 3 in RTT_MR.

Programming RTPRES to 1 or 2 is forbidden.

If the RTT is configured to trigger an interrupt, the interrupt occurs two slow clock cycles after reading the RTT_SR. To prevent several executions of the interrupt handler, the interrupt must be disabled in the interrupt handler and re-enabled when the RTT_SR is cleared.

The CRTV field can be read at any time in the “Real-time Timer Value Register” (RTT_VR). As this value can be updated asynchronously with the Master Clock, the CRTV field must be read twice at the same value to read a correct value.

The current value of the counter is compared with the value written in the “Real-time Timer Alarm Register” (RTT_AR). If the counter value matches the alarm, the ALMS bit in the RTT_SR is set. The RTT_AR is set to its maximum value (0xFFFF_FFFF) after a reset.

The ALMS flag is always a source of the RTT alarm signal that may be used to exit the system from low power modes (see Figure 14-1).

The alarm interrupt must be disabled (ALMIEN must be cleared in RTT_MR) when writing a new ALMV value in the RTT_AR.

The RTTINC bit can be used to start a periodic interrupt, the period being one second when the RTPRES field value = 0x8000 and the slow clock = 32.768 kHz.

The RTTINCIEN bit must be cleared prior to writing a new RTPRES value in the RTT_MR.

Reading the RTT_SR automatically clears the RTTINC and ALMS bits.

Writing the RTTRST bit in the RTT_MR immediately reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

When not used, the Real-time Timer can be disabled in order to suppress dynamic power consumption in this module. This can be achieved by setting the RTTDIS bit in the RTT_MR.

14.5.1 Real-time Timer Mode Register

Name: RTT_MR

Address: 0x400E1830

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	RTC1HZ

23	22	21	20	19	18	17	16
–	–	–	RTTDIS	–	RTTRST	RTTINCIEN	ALMIEN

15	14	13	12	11	10	9	8
RTPRES							

7	6	5	4	3	2	1	0
RTPRES							

- **RTPRES: Real-time Timer Prescaler Value**

Defines the number of SLCK periods required to increment the Real-time timer. RTPRES is defined as follows:

RTPRES = 0: The prescaler period is equal to 2^{16} * SLCK periods.

RTPRES = 1 or 2: forbidden.

RTPRES ≠ 0,1 or 2: The prescaler period is equal to RTPRES * SLCK periods.

Note: The RTTINCIEN bit must be cleared prior to writing a new RTPRES value.

- **ALMIEN: Alarm Interrupt Enable**

0: The bit ALMS in RTT_SR has no effect on interrupt.

1: The bit ALMS in RTT_SR asserts interrupt.

- **RTTINCIEN: Real-time Timer Increment Interrupt Enable**

0: The bit RTTINC in RTT_SR has no effect on interrupt.

1: The bit RTTINC in RTT_SR asserts interrupt.

- **RTTRST: Real-time Timer Restart**

0: No effect.

1: Reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

- **RTTDIS: Real-time Timer Disable**

0: The real-time timer is enabled.

1: The real-time timer is disabled (no dynamic power consumption).

Note: RTTDIS is write only.

- **RTC1HZ: Real-Time Clock 1Hz Clock Selection**

0: The RTT 32-bit counter is driven by the 16-bit prescaler roll-over events.

1: The RTT 32-bit counter is driven by the 1Hz RTC clock.

Note: RTC1HZ is write only.

Single-buffer DMAC transfer: Consists of a single buffer.

Multi-buffer DMAC transfer: A DMAC transfer may consist of multiple DMAC buffers. Multi-buffer DMAC transfers are supported through buffer chaining (linked list pointers), auto-reloading of channel registers, and contiguous buffers. The source and destination can independently select which method to use.

- **Linked lists (buffer chaining) –** A descriptor pointer (DSCR) points to the location in system memory where the next linked list item (LLI) exists. The LLI is a set of registers that describe the next buffer (buffer descriptor) and a descriptor pointer register. The DMAC fetches the LLI at the beginning of every buffer when buffer chaining is enabled.
- **Contiguous buffers –** Where the address of the next buffer is selected to be a continuation from the end of the previous buffer.

Channel locking: Software can program a channel to keep the AHB master interface by locking the arbitration for the master bus interface for the duration of a DMAC transfer, buffer, or chunk.

Bus locking: Software can program a channel to maintain control of the AMBA bus by asserting hmastlock for the duration of a DMAC transfer, buffer, or transaction (single or chunk). Channel locking is asserted for the duration of bus locking at a minimum.

25.6.2 Memory Peripherals

Figure 25-3 on page 470 shows the DMAC transfer hierarchy of the DMAC for a memory peripheral. There is no handshaking interface with the DMAC, and therefore the memory peripheral can never be a flow controller. Once the channel is enabled, the transfer proceeds immediately without waiting for a transaction request. The alternative to not having a transaction-level handshaking interface is to allow the DMAC to attempt AMBA transfers to the peripheral once the channel is enabled. If the peripheral slave cannot accept these AMBA transfers, it inserts wait states onto the bus until it is ready; it is not recommended that more than 16 wait states be inserted onto the bus. By using the handshaking interface, the peripheral can signal to the DMAC that it is ready to transmit/receive data, and then the DMAC can access the peripheral without the peripheral inserting wait states onto the bus.

25.6.3 Handshaking Interface

Handshaking interfaces are used at the transaction level to control the flow of single or chunk transfers. The operation of the handshaking interface is different and depends on whether the peripheral or the DMAC is the flow controller.

The peripheral uses the handshaking interface to indicate to the DMAC that it is ready to transfer/accept data over the AMBA bus. A non-memory peripheral can request a DMAC transfer through the DMAC using one of two handshaking interfaces:

- Hardware handshaking
- Software handshaking

Software selects between the hardware or software handshaking interface on a per-channel basis. Software handshaking is accomplished through memory-mapped registers, while hardware handshaking is accomplished using a dedicated handshaking interface.

25.6.3.1 Software Handshaking

When the slave peripheral requires the DMAC to perform a DMAC transaction, it communicates this request by sending an interrupt to the CPU or interrupt controller.

The interrupt service routine then uses the software registers to initiate and control a DMAC transaction. These software registers are used to implement the software handshaking interface.

The SRC_H2SEL/DST_H2SEL bit in the Channel Configuration Register (DMAC_CFGx) must be cleared to enable software handshaking.

When the peripheral is not the flow controller, then the Software Last Transfer Flag Register (DMAC_LAST) is not used, and the values in these registers are ignored.

Chunk Transactions

Writing a ‘1’ to the Software Chunk Transfer Request Register (DMAC_CREQ[2x]) starts a source chunk transaction request, where x is the channel number. Writing a ‘1’ to the DMAC_CREQ[2x+1] register starts a destination chunk transfer request, where x is the channel number.

Upon completion of the chunk transaction, the hardware clears the DMAC_CREQ[2*x*] or DMAC_CREQ[2x+1].

Single Transactions

Writing a ‘1’ to the Software Single Request Register (DMAC_SREQ[2x]) starts a source single transaction request, where x is the channel number. Writing a ‘1’ to the DMAC_SREQ[2x+1] register starts a destination single transfer request, where x is the channel number.

Upon completion of the chunk transaction, the hardware clears the DMAC_SREQ[x] or DMAC_SREQ[2x+1].

The software can poll the relevant channel bit in the DMAC_CREQ[2x]/DMAC_CREQ[2x+1] and DMAC_SREQ[x]/DMAC_SREQ[2x+1] registers. When both are 0, then either the requested chunk or single transaction has completed.

25.6.4 DMAC Transfer Types

A DMAC transfer may consist of single or multi-buffer transfers. On successive buffers of a multi-buffer transfer, DMAC_SADDRx/DMAC_DADDRx in the DMAC are reprogrammed using either of the following methods:

- Buffer chaining using linked lists
- Contiguous address between buffers

On successive buffers of a multi-buffer transfer, the DMAC_CTRLAx and DMAC_CTRLBx registers in the DMAC are reprogrammed using either of the following methods:

- Buffer chaining using linked lists

When buffer chaining using linked lists is the multi-buffer method of choice, and on successive buffers, DMAC_DSCRx in the DMAC is reprogrammed using the following method:

- Buffer chaining using linked lists

A buffer descriptor (LLI) consists of the following registers: DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLAx, and DMAC_CTRLBx. These registers, along with DMAC_CFGx, are used by the DMAC to set up and describe the buffer transfer.

25.6.4.1 Multi-buffer Transfers

Buffer Chaining Using Linked Lists

In this case, the DMAC reprograms the channel registers prior to the start of each buffer by fetching the buffer descriptor for that buffer from system memory. This is known as an LLI update.

DMAC buffer chaining is supported by using a descriptor pointer register (DMAC_DSCRx) that stores the address in memory of the next buffer descriptor. Each buffer descriptor contains the corresponding buffer descriptor (DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLAx DMAC_CTRLBx).

To set up buffer chaining, a sequence of linked lists must be programmed in memory.

DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLAx and DMAC_CTRLBx are fetched from system memory on an LLI update. The updated content of DMAC_CTRLAx is written back to memory on buffer completion. Figure 25-4 on page 473 shows how to use chained linked lists in memory to define multi-buffer transfers using buffer chaining.

25.8.1 DMAC Global Configuration Register

Name: DMAC_GCFG

Address: 0x400C0000

Access: Read/Write

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
–	–	–	ARB_CFG	–	–	–	–

Note: Bit fields 0, 1, 2, and 3 have a default value of 0. This should not be changed.

This register can only be written if the WPEN bit is cleared in “DMAC Write Protection Mode Register” .

- **ARB_CFG: Arbiter Configuration**

Value	Name	Description
0	FIXED	Fixed priority arbiter (see “Basic Definitions”)
1	ROUND_ROBIN	Modified round robin arbiter.

26.5.2 Receive Counter Register

Name: PERIPH_RCR

Access: Read/Write

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

- **RXCTR: Receive Counter Register**

RXCTR must be set to receive buffer size.

When a half-duplex peripheral is connected to the PDC, RXCTR = TXCTR.

0: Stops peripheral data transfer to the receiver.

1–65535: Starts peripheral data transfer if the corresponding channel is active.

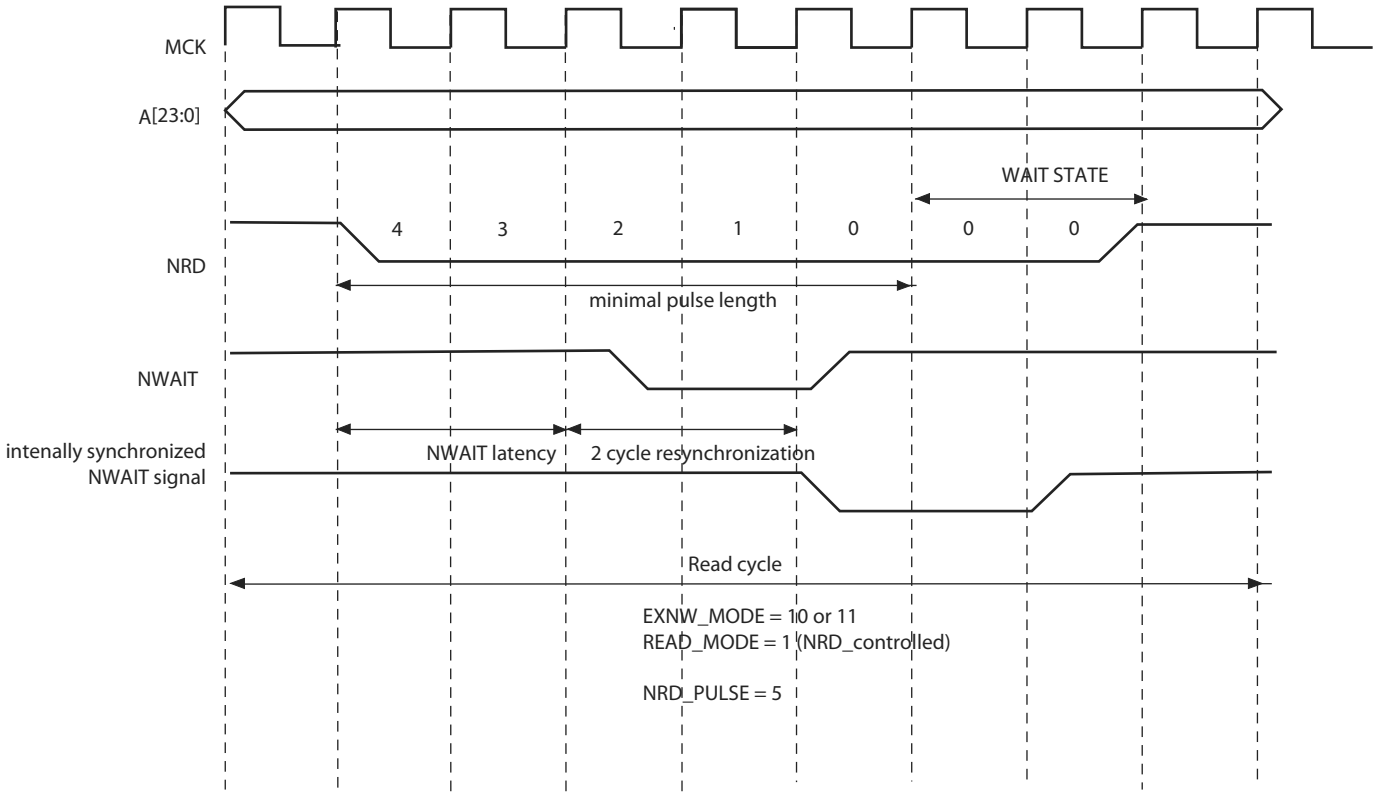
27.13.4 NWAIT Latency and Read/Write Timings

There may be a latency between the assertion of the read/write controlling signal and the assertion of the NWAIT signal by the device. The programmed pulse length of the read/write controlling signal must be at least equal to this latency plus the 2 cycles of resynchronization + one cycle. Otherwise, the SMC may enter the hold state of the access without detecting the NWAIT signal assertion. This is true in Frozen mode as well as in Ready mode. This is illustrated on Figure 27-27.

When SMC_MODE.EXNW_MODE is enabled (ready or frozen), the user must program a pulse length of the read and write controlling signal of at least:

Minimal pulse length = NWAIT latency + 2 resynchronization cycles + 1 cycle

Figure 27-27. NWAIT Latency



- **KEYSIZE: Key Size**

Value	Name	Description
0	AES128	AES Key Size is 128 bits
1	AES192	AES Key Size is 192 bits
2	AES256	AES Key Size is 256 bits

Values which are not listed in the table must be considered as “reserved”.

- **OPMOD: Operation Mode**

Value	Name	Description
0	ECB	ECB: Electronic Code Book mode
1	CBC	CBC: Cipher Block Chaining mode
2	OFB	OFB: Output Feedback mode
3	CFB	CFB: Cipher Feedback mode
4	CTR	CTR: Counter mode (16-bit internal counter)

Values which are not listed in the table must be considered as “reserved”.

For CBC-MAC operating mode, set OPMOD to CBC and LOD to 1.

- **LOD: Last Output Data Mode**

0: No effect.

After each end of encryption/decryption, the output data are available either on the output data registers (Manual and Auto modes) or at the address specified in the Channel Buffer Transfer Descriptor for DMA mode.

In Manual and Auto modes, the DATRDY flag is cleared when at least one of the Output Data registers is read.

1: The DATRDY flag is cleared when at least one of the Input Data Registers is written.

No more Output Data Register reads is necessary between consecutive encryptions/decryptions (see Section 30.4.5 “Last Output Data Mode”).

Warning: In DMA mode, reading to the Output Data registers before the last data encryption/decryption process may lead to unpredictable results.

- **CFBS: Cipher Feedback Data Size**

Value	Name	Description
0	SIZE_128BIT	128-bit
1	SIZE_64BIT	64-bit
2	SIZE_32BIT	32-bit
3	SIZE_16BIT	16-bit
4	SIZE_8BIT	8-bit

Values which are not listed in table must be considered as “reserved”.

31.8.3.2 Transmission Handling

A mailbox is in Transmit Mode once the MOT field in the CAN_MMRx has been configured. Message ID and Message Acceptance mask must be set before Receive Mode is enabled.

After Transmit Mode is enabled, the MRDY flag in the CAN_MSR is automatically set until the first command is sent. When the MRDY flag is set, the software application can prepare a message to be sent by writing to the CAN_MDx registers. The message is sent once the software asks for a transfer command setting the MTCR bit and the message data length in the CAN_MCRx.

The MRDY flag remains at zero as long as the message has not been sent or aborted. It is important to note that no access to the mailbox data register is allowed while the MRDY flag is cleared. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked depending on the mailbox flag in the CAN_IMR global register.

It is also possible to send a remote frame setting the MRTR bit instead of setting the MDLC field. The answer to the remote frame is handled by another reception mailbox. In this case, the device acts as a consumer but with the help of two mailboxes. It is possible to handle the remote frame emission and the answer reception using only one mailbox configured in Consumer Mode. Refer to the section “Remote Frame Handling” on page 664.

Several messages can try to win the bus arbitration in the same time. The message with the highest priority is sent first. Several transfer request commands can be generated at the same time by setting MBx bits in the CAN_TCR. The priority is set in the PRIOR field of the CAN_MMRx. Priority 0 is the highest priority, priority 15 is the lowest priority. Thus it is possible to use a part of the message ID to set the PRIOR field. If two mailboxes have the same priority, the message of the mailbox with the lowest number is sent first. Thus if mailbox 0 and mailbox 5 have the same priority and have a message to send at the same time, then the message of the mailbox 0 is sent first.

Setting the MACR bit in the CAN_MCRx aborts the transmission. Transmission for several mailboxes can be aborted by writing MBx fields in the CAN_ACR. If the message is being sent when the abort command is set, then the application is notified by the MRDY bit set and not the MABT in the CAN_MSRx. Otherwise, if the message has not been sent, then the MRDY and the MABT are set in the CAN_MSR.

When the bus arbitration is lost by a mailbox message, the CAN controller tries to win the next bus arbitration with the same message if this one still has the highest priority. Messages to be sent are re-tried automatically until they win the bus arbitration. This feature can be disabled by setting the bit DRPT in the CAN_MR. In this case if the message was not sent the first time it was transmitted to the CAN transceiver, it is automatically aborted. The MABT flag is set in the CAN_MSRx until the next transfer command.

Figure 31-15 shows three MBx message attempts being made (MRDY of MBx set to 0).

The first MBx message is sent, the second is aborted and the last one is trying to be aborted but too late because it has already been transmitted to the CAN transceiver.

34.8 Serial Peripheral Interface (SPI) User Interface

In the “Offset” column of Table 34-5, ‘CS_number’ denotes the chip select number.

Table 34-5. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	SPI_CR	Write-only	–
0x04	Mode Register	SPI_MR	Read/Write	0x0
0x08	Receive Data Register	SPI_RDR	Read-only	0x0
0x0C	Transmit Data Register	SPI_TDR	Write-only	–
0x10	Status Register	SPI_SR	Read-only	0x000000F0
0x14	Interrupt Enable Register	SPI_IER	Write-only	–
0x18	Interrupt Disable Register	SPI_IDR	Write-only	–
0x1C	Interrupt Mask Register	SPI_IMR	Read-only	0x0
0x20–0x2C	Reserved	–	–	–
0x30 + (CS_number * 0x04)	Chip Select Register	SPI_CSR	Read/Write	0x0
0x40–0x48	Reserved	–	–	–
0x4C–0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	SPI_WPMR	Read/Write	0x0
0xE8	Write Protection Status Register	SPI_WPSR	Read-only	0x0
0xEC–0xF8	Reserved	–	–	–
0xFC	Reserved	–	–	–
0x100–0x124	Reserved for PDC Registers	–	–	–

37.7.5 USART Interrupt Enable Register

Name: US_IER

Address: 0x400A0008 (0), 0x400A4008 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	MANE
23	22	21	20	19	18	17	16
–	–	–	–	CTSIC	DCDIC	DSRIC	RIIC
15	14	13	12	11	10	9	8
–	–	NACK	RXBUFF	TXBUFE	ITER	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

For SPI specific configuration, see Section 37.7.6 ”USART Interrupt Enable Register (SPI_MODE)”.

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Enables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **RXBRK: Receiver Break Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable (available in all USART modes of operation)**
- **ENDTX: End of Transmit Buffer Interrupt Enable (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Enable**
- **PARE: Parity Error Interrupt Enable**
- **TIMEOUT: Time-out Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **ITER: Max number of Repetitions Reached Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable (available in all USART modes of operation)**
- **RXBUFF: Receive Buffer Full Interrupt Enable (available in all USART modes of operation)**
- **NACK: Non Acknowledge Interrupt Enable**
- **RIIC: Ring Indicator Input Change Enable**

41.7.4 UDP Interrupt Enable Register

Name: UDP_IER

Address: 0x40084010

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Enable Endpoint 0 Interrupt**
- **EP1INT: Enable Endpoint 1 Interrupt**
- **EP2INT: Enable Endpoint 2Interrupt**
- **EP3INT: Enable Endpoint 3 Interrupt**
- **EP4INT: Enable Endpoint 4 Interrupt**
- **EP5INT: Enable Endpoint 5 Interrupt**
- **EP6INT: Enable Endpoint 6 Interrupt**
- **EP7INT: Enable Endpoint 7 Interrupt**

0: No effect

1: Enables corresponding Endpoint Interrupt

- **RXSUSP: Enable UDP Suspend Interrupt**

0: No effect

1: Enables UDP Suspend Interrupt

- **RXRSM: Enable UDP Resume Interrupt**

0: No effect

1: Enables UDP Resume Interrupt

- **SOFINT: Enable Start Of Frame Interrupt**

0: No effect

1: Enables Start Of Frame Interrupt

42.8.5 GMAC DMA Configuration Register

Name: GMAC_DCFGR

Address: 0x40034010

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
DRBS							

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
ESPA	ESMA	–	FBLDO				

- **FBLDO: Fixed Burst Length for DMA Data Operations:**

Selects the burst length to attempt to use on the AHB when transferring frame data. Not used for DMA management operations and only used where space and data size allow. Otherwise SINGLE type AHB transfers are used.

Upper bits become non-writable if the configured DMA TX and RX FIFO sizes are smaller than required to support the selected burst size.

One-hot priority encoding enforced automatically on register writes as follows, where ‘x’ represents don’t care:

Value	Name	Description
0	–	Reserved
1	SINGLE	00001: Always use SINGLE AHB bursts
2	–	Reserved
4	INCR4	001xx: Attempt to use INCR4 AHB bursts (Default)
8	INCR8	01xxx: Attempt to use INCR8 AHB bursts
16	INCR16	1xxxx: Attempt to use INCR16 AHB bursts

- **ESMA: Endian Swap Mode Enable for Management Descriptor Accesses**

When set, selects swapped endianism for AHB transfers. When clear, selects little endian mode.

- **ESPA: Endian Swap Mode Enable for Packet Data Accesses**

When set, selects swapped endianism for AHB transfers. When clear, selects little endian mode.

- **DRBS: DMA Receive Buffer Size**

DMA receive buffer size in AHB system memory. The value defined by these bits determines the size of buffer to use in main AHB system memory when writing received data.

The value is defined in multiples of 64 bytes, thus a value of 0x01 corresponds to buffers of 64 bytes, 0x02 corresponds to 128 bytes etc. For example:

– 0x02: 128 bytes

– 0x18: 1536 bytes (1 × max length frame/buffer)

– 0xA0: 10240 bytes (1 × 10K jumbo frame/buffer)

Note that this value should never be written as zero.

42.8.44 GMAC PTP Peer Event Frame Transmitted Seconds Low Register

Name: GMAC_PEFTSL

Address: 0x400341F0

Access: Read-only

31	30	29	28	27	26	25	24
RUD							

23	22	21	20	19	18	17	16
RUD							

15	14	13	12	11	10	9	8
RUD							

7	6	5	4	3	2	1	0
RUD							

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Seconds Register holds when the SFD of a PTP transmit peer event crosses the MII interface. An interrupt is issued when the register is updated.

43.7.2 AFEC Mode Register

Name: AFEC_MR

Address: 0x400B0004 (0), 0x400B4004 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
USEQ	–	TRANSFER		TRACKTIM			
23	22	21	20	19	18	17	16
ANACH	–	SETTLING		STARTUP			
15	14	13	12	11	10	9	8
PRESCAL							
7	6	5	4	3	2	1	0
FREERUN	FWUP	SLEEP	–	TRGSEL			TRGEN

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **TRGEN: Trigger Enable**

Value	Name	Description
0	DIS	Hardware triggers are disabled. Starting a conversion is only possible by software.
1	EN	Hardware trigger selected by TRGSEL field is enabled.

- **TRGSEL: Trigger Selection**

Value	Name	Description
0	AFEC_TRIG0	ADTRG pin
1	AFEC_TRIG1	TIO Output of the Timer Counter Channel 0
2	AFEC_TRIG2	TIO Output of the Timer Counter Channel 1
3	AFEC_TRIG3	TIO Output of the Timer Counter Channel 2
4	AFEC_TRIG4	PWM Event Line 0
5	AFEC_TRIG5	PWM Event Line 1
6	AFEC_TRIG6	Reserved
7	–	Reserved

- **SLEEP: Sleep Mode**

Value	Name	Description
0	NORMAL	Normal mode: The AFE and reference voltage circuitry are kept ON between conversions.
1	SLEEP	Sleep mode: The AFE and reference voltage circuitry are OFF between conversions.

- **FWUP: Fast Wake-up**

Value	Name	Description
0	OFF	Normal Sleep mode: The sleep mode is defined by the SLEEP bit.
1	ON	Fast wake-up Sleep mode: The voltage reference is ON between conversions and AFE is OFF.