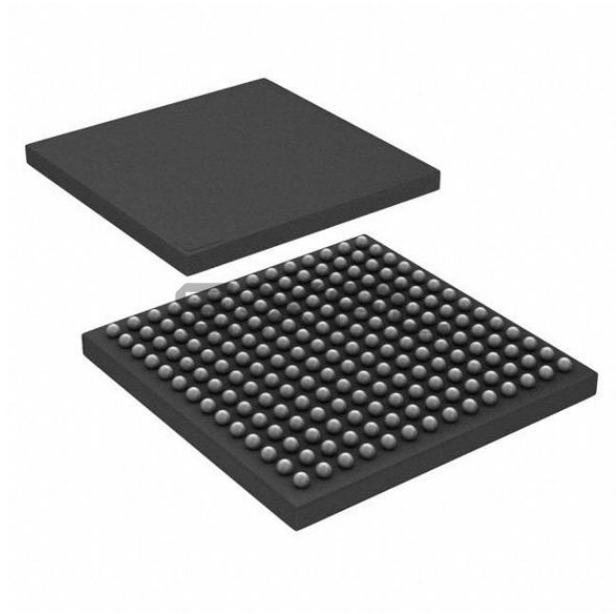


**Welcome to E-XFL.COM**

## Understanding Embedded - Microprocessors

Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, efficiency, and reliability. These microprocessors are integral to the operation of countless electronic devices, providing the computational power necessary for controlling processes, handling data, and managing communications.

## Applications of Embedded - Microprocessors

Embedded microprocessors are utilized across a broad spectrum of applications, making them indispensable in

| Details | |
|---|---|
| Product Status | Active |
| Core Processor | ARM® Cortex®-A5 |
| Number of Cores/Bus Width | 1 Core, 32-Bit |
| Speed | 500MHz |
| Co-Processors/DSP | Multimedia; NEON™ MPE |
| RAM Controllers | LPDDR1, LPDDR2, LPDDR3, DDR2, DDR3, DDR3L, QSPI |
| Graphics Acceleration | Yes |
| Display & Interface Controllers | Keyboard, LCD, Touchscreen |
| Ethernet | 10/100Mbps (1) |
| SATA | - |
| USB | USB 2.0 + HSIC |
| Voltage - I/O | 3.3V |
| Operating Temperature | -40°C ~ 105°C (TA) |
| Security Features | ARM TZ, Boot Security, Cryptography, RTIC, Secure Fusebox, Secure JTAG, Secure Memory, Secure RTC |
| Package / Case | 196-TFBGA, CSBGA |
| Supplier Device Package | 196-TFBGA (11x11) |
| Purchase URL | https://www.e-xfl.com/product-detail/microchip-technology/atsama5d23a-cn |

| Value | Name | Description |
|---|---|---|
| 11 | – | Reserved |
| 12 | 1024K | 1024 Kbytes |
| 13 | – | Reserved |
| 14 | 2048K | 2048 Kbytes |
| 15 | – | Reserved |

**NVPSIZ2: Second Nonvolatile Program Memory Size**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | 8K | 8 Kbytes |
| 2 | 16K | 16 Kbytes |
| 3 | 32K | 32 Kbytes |
| 4 | – | Reserved |
| 5 | 64K | 64 Kbytes |
| 6 | – | Reserved |
| 7 | 128K | 128 Kbytes |
| 8 | – | Reserved |
| 9 | 256K | 256 Kbytes |
| 10 | 512K | 512 Kbytes |
| 11 | – | Reserved |
| 12 | 1024K | 1024 Kbytes |
| 13 | – | Reserved |
| 14 | 2048K | 2048 Kbytes |
| 15 | – | Reserved |

**SRAMSIZ: Internal SRAM Size**

| Value | Name | Description |
|---|---|---|
| 0 | 48K | 48 Kbytes |
| 1 | 192K | 192 Kbytes |
| 2 | 384K | 384 Kbytes |
| 3 | 6K | 6 Kbytes |
| 4 | 24K | 24 Kbytes |
| 5 | 4K | 4 Kbytes |
| 6 | 80K | 80 Kbytes |
| 7 | 160K | 160 Kbytes |
| 8 | 8K | 8 Kbytes |
| 9 | 16K | 16 Kbytes |
| 10 | 32K | 32 Kbytes |

## 25.5 Periodic Interval Timer (PIT) User Interface

**Table 25-1: Register Mapping**

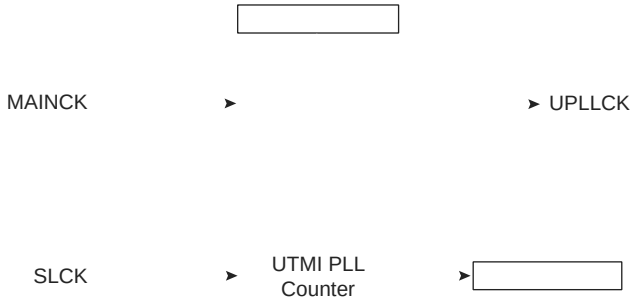
| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Mode Register | PIT_MR | Read/Write | 0x000F_FFFF |
| 0x04 | Status Register | PIT_SR | Read-only | 0x0000_0000 |
| 0x08 | Periodic Interval Value Register | PIT_PIVR | Read-only | 0x0000_0000 |
| 0x0C | Periodic Interval Image Register | PIT_PIIR | Read-only | 0x0000_0000 |

## 32.7 UTMI PLL Clock

The source of the UTMI PLL (UPLL) is the Main clock (MAINCK). MAINCK must select the Main crystal oscillator to meet the frequency accuracy required by USB.

The crystal frequency selection among 12, 16 or 24 MHz must be configured to the correct value in the field SFR_UTMICKTRIM.FREQ, in order to apply the correct multiplier, x40, x30 or x20, respectively.

**Figure 32-6: UTMI PLL Block Diagram**




Whenever the UTMI PLL is enabled by writing UPLLEN in the UTMI Clock register (CKGR_UCKR), PMC_SR.LOCKU is automatically cleared. The values written in CKGR_UCKR.UPLLCOUNT are loaded in the UTMI PLL counter. The UTMI PLL counter then decrements at the speed of the Slow clock divided by 8 until it reaches 0. At this time, the PMC_SR.LOCKU is set in and can trigger an interrupt to the processor. The user has to load the number of Slow clock cycles required to cover the UTMI PLL transient time into CKGR_UCKR.UPLLCOUNT.

## 32.8 Audio PLL

The Audio PLL is a high-resolution fractional-N digital PLL specifically designed for low jitter operation.

In audio applications, the CLK_AUDIO output pin typically serves as the Master clock frequency generator for external components such as Audio DAC, Audio ADCs, or Audio Codecs, thus saving one crystal on the board.

The reference clock of the Audio PLL is the fast crystal oscillator. The PLL core operating frequency is defined as:

$$f_{AUDIOCORECLK} = f_{ref}\left(ND + 1 + \frac{FRACR}{2^{22}}\right)$$

where $f_{ref}$ is the frequency of the main crystal oscillator. Refer to Section 66.8 “PLL Characteristics” for the limits of $f_{AUDIOCORECLK}$.

The PLL core features two post-dividers enabling the generation of two output clock signals, AUDIOPLLCLK and AUDIOPINCLK. AUDIOPLLCLK is dedicated to the PMC and can be sent to the GCLK input of peripherals or to the Programmable clock outputs PCKx. AUDIOPINCLK is dedicated to driving the external audio pin CLK_AUDIO.

The AUDIOPLLCLK frequency is defined by the following formula:

$$f_{AUDIOPLLCLK} = \frac{f_{AUDIOCORECLK}}{(QDPMC + 1)}$$

The AUDIOPINCLK frequency is defined by the following formula:

$$f_{AUDIOPINCLK} = \frac{f_{AUDIOCORECLK}}{(DIV \times QDAUDIO)}$$

The typical programming sequence of the audio PLL is the following:

1. Disable the PLL by writing ‘0’ in bits PLLEN and RESETN in the Audio PLL Control register 0 (PMC_AUDIO_PLL0).
2. Release the reset of the PLL by writing ‘1’ in PMC_AUDIO_PLL0.RESETN.
3. Configure the PLL frequency by writing QDPMC and ND in PMC_AUDIO_PLL0, QDAUDIO, DIV and FRACR in PMC_AUDIO_PLL1. ND and FRACR must be configured so as to set AUDIOCORECLK frequency in its authorized range. Refer to Section 66. “Electrical Characteristics”.
4. Enable the PLL by writing ‘1’ in PMC_AUDIO_PLL0.PLLEN, PMC_AUDIO_PLL0.PADEN and PMC_AUDIO_PLL0.PMCEN.
5. Wait for the startup time of this PLL. Refer to Section 66. “Electrical Characteristics”.

the MRS field to 16. Mode register Write command cycle is issued to program parameters of the low-power DDR2-SDRAM device, in particular Partial Array Self Refresh (PASR). Read the MPDDRC_MR and add a memory barrier assembler instruction just after the read. Perform a write access to any low-power DDR2-SDRAM address to acknowledge this command. The Mode register Write command is now issued.

14. In the DDR Configuration register (SFR_DDRCFG), the application must write a '1' to bits 17 and 16 to open the input buffers (Refer to Section 19. "Special Function Registers (SFR)").
15. A NOP command is issued to the low-power DDR2-SDRAM. Program the NOP command in the Mode register (MPDDRC_MR). The application must configure the MODE field to 1 in the MPDDRC_MR. Read the MPDDRC_MR and add a memory barrier assembler instruction just after the read. Perform a write access to any low-power DDR2-SDRAM address to acknowledge this command.
16. A Mode register Read command is issued to the low-power DDR2-SDRAM. In MPDDRC_MR, configure the MODE field to 7 and the MRS field to 5. The Mode register Read command cycle is used to read the LPDDR2 Manufacturer ID from the low-power DDR2-SDRAM mode registers. Read the MPDDRC_MR and add a memory barrier assembler instruction just after the read. Perform a write access to any low-power DDR2-SDRAM address to acknowledge this command. The Mode register Read command is now issued. The LPDDR2 Manufacturer ID is set in MPDDRC_MD. See Section 36.7.8 "MPDDRC Memory Device Register".
17. A Mode register Read command is issued to the low-power DDR2-SDRAM. In MPDDRC_MR, configure the MODE field to 7 and the MRS field to 6. The Mode register Read command cycle is used to read Revision ID1 from the low-power DDR2-SDRAM mode registers. Read the MPDDRC_MR and add a memory barrier assembler instruction just after the read. Perform a write access to any low-power DDR2-SDRAM address to acknowledge this command. The Mode register Read command is now issued. Revision ID1 is set in register MPDDRC_MD. See Section 36.7.8 "MPDDRC Memory Device Register".
18. A Mode register Read command is issued to the low-power DDR2-SDRAM. In MPDDRC_MR, configure the MODE field to 7 and the MRS field to 8. The Mode register Read command cycle is used to read the memory organization (I/O width, Density, Type) from the low-power DDR2-SDRAM mode registers. Read the MPDDRC_MR and add a memory barrier assembler instruction just after the read. Perform a write access to any low-power DDR2-SDRAM address to acknowledge this command. The Mode register Read command is now issued. Memory organization is set in register MPDDRC_MD. See Section 36.7.8 "MPDDRC Memory Device Register".
19. A Mode register Read command is issued to the low-power DDR2-SDRAM. In MPDDRC_MR, configure the MODE field to 7 and the MRS field to 0. The Mode register Read command cycle is used to read device information (RZQI, DAI) from the low-power DDR2-SDRAM mode registers. Read the MPDDRC_MR and add a memory barrier assembler instruction just after the read. Perform a write access to any low-power DDR2-SDRAM address to acknowledge this command. The Mode register Read command is now issued. Device information RZQI is set in register Timing Calibration (see Section 36.7.11 "MPDDRC Low-power DDR2 Low-power DDR3 and DDR3 Timing Calibration Register") and DAI is set in Mode register (see Section 36.7.1 "MPDDRC Mode Register").
20. A Normal Mode command is provided. Program the Normal mode in the MPDDRC_MR. Read the MPDDRC_MR and add a memory barrier assembler instruction just after the read. Perform a write access to any low-power DDR2-SDRAM address to acknowledge this command.
21. In the DDR configuration register (SFR_DDRCCFG), the application must write a '0' to bits 17 and 16 to close the input buffers. The buffers are then driven by the HMPDDRC controller.
22. Write the refresh rate into the COUNT field in the Refresh Timer register (MPDDRC_RTR). To compute the value, refer to Section 36.7.2 "MPDDRC Refresh Timer Register".

After initialization, the low-power DDR2-SDRAM devices are fully functional.

### 36.4.4 DDR3-SDRAM/DDR3L-SDRAM Initialization

The initialization sequence is generated by software. The DDR3-SDRAM devices are initialized by the following sequence:

1. Program the memory device type in the Memory Device register (MPDDRC_MD).
2. Program features of the DDR3-SDRAM device in the Configuration register (MPDDRC_CR) (number of columns, rows, banks, CAS latency and output driver impedance control) and in the Timing Parameter 0 register/Timing Parameter 1 register (MPDDRC_TPR0/1) (asynchronous timing - TRC, TRAS, etc.).
3. A NOP command is issued to the DDR3-SDRAM. Program the NOP command in the Mode register (MPDDRC_MR). The application must configure the MODE field to 1 in the MPDDRC_MR. Read the MPDDRC_MR and add a memory barrier assembler instruction just after the read. Perform a write access to any DDR3-SDRAM address to acknowledge this command. The clocks which drive the DDR3-SDRAM device are now enabled.
4. A pause of at least 500 µs must be observed before a signal toggle.
5. A NOP command is issued to the DDR3-SDRAM. Program the NOP command in the MPDDRC_MR. The application must configure the MODE field to 1 in the MPDDRC_MR. Read the MPDDRC_MR and add a memory barrier assembler instruction just after the read. Perform a write access to any DDR3-SDRAM address to acknowledge this command. CKE is now driven high.

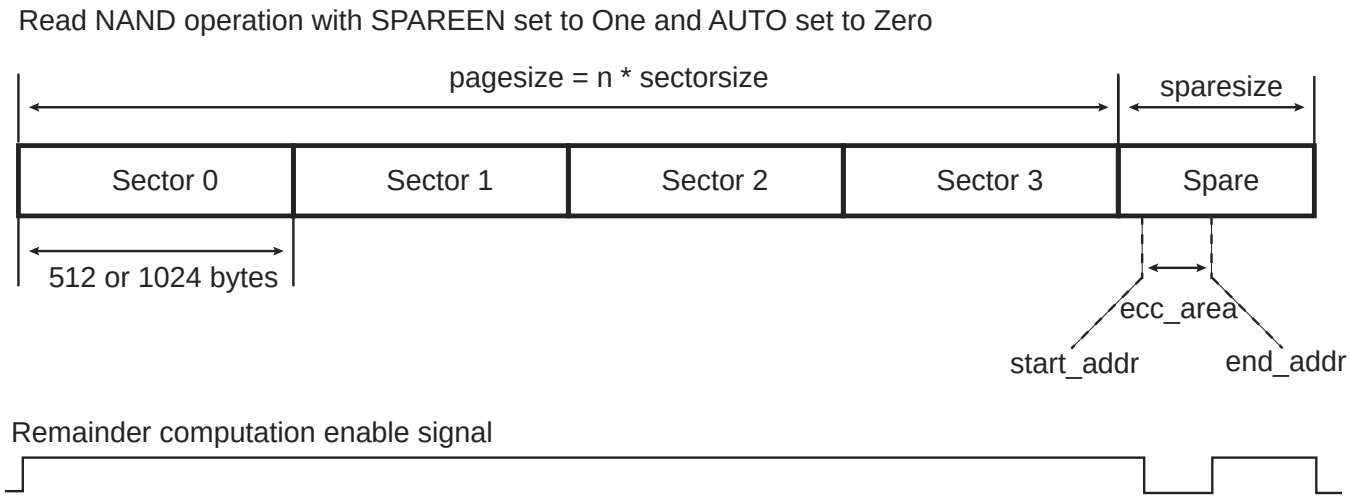
**Table 37-16: Relevant Remainder Registers (Continued)**

| BCH_ERR Field | Sector Size Set to 512 Bytes | Sector Size Set to 1024 Bytes |
|---|---|---|
| 4 | PMECCREM0, PMECCREM1, PMECCREM2, PMECCREM3, PMECCREM4, PMECCREM5, PMECCREM6, PMECCREM7, PMECCREM8, PMECCREM9, PMECCREM10, PMECCREM11 | PMECCREM0, PMECCREM1, PMECCREM2, PMECCREM3, PMECCREM4, PMECCREM5, PMECCREM6, PMECCREM7, PMECCREM8, PMECCREM9, PMECCREM10, PMECCREM11 |
| 5 | PMECCREM0, PMECCREM1, PMECCREM2, PMECCREM3, PMECCREM4, PMECCREM5, PMECCREM6, PMECCREM7, PMECCREM8, PMECCREM9, PMECCREM10, PMECCREM11, PMECCREM12, PMECCREM13, PMECCREM14, PMECCREM15 | PMECCREM0, PMECCREM1, PMECCREM2, PMECCREM3, PMECCREM4, PMECCREM5, PMECCREM6, PMECCREM7, PMECCREM8, PMECCREM9, PMECCREM10, PMECCREM11, PMECCREM12, PMECCREM13, PMECCREM14, PMECCREM15 |

#### 37.18.2.1 MLC/SLC Read Operation with Spare Decoding

When the spare area is protected, it contains valid data. As the redundancy may be included in the middle of the information stream, the user shall program the start address and the end address of the ECC area. The controller will automatically skip the ECC area. This mode is entered writing a 1 in the DATA bit of the PMECCTRL register. When the page has been fully retrieved from the NAND, the ECC area shall be read using the User mode, writing a 1 to the USER bit of the PMECCTRL register.

**Figure 37-42: Read Operation with Spare Decoding**




#### 37.18.2.2 MLC/SLC Read Operation

If the spare area is not protected with the error correcting code, the redundancy area is retrieved directly. This mode is entered writing a 1 in the DATA bit of the PMECCTRL register. When AUTO field is set to one, the ECC is retrieved automatically; otherwise, the ECC must be read using the User mode.

#### 40.6.3.5 DMA Bursting on the AHB

The DMA will always use SINGLE, or INCR type AHB accesses for buffer management operations. When performing data transfers, the AHB burst length used can be programmed using bits 4:0 of the DMA Configuration register so that either SINGLE, INCR or fixed length incrementing bursts (INCR4, INCR8 or INCR16) are used where possible.

When there is enough space and enough data to be transferred, the programmed fixed length bursts will be used. If there is not enough data or space available, for example when at the beginning or the end of a buffer, SINGLE type accesses are used. Also SINGLE type accesses are used at 1024 byte boundaries, so that the 1 Kbyte boundaries are not burst over as per AHB requirements.

The DMA will not terminate a fixed length burst early, unless an error condition occurs on the AHB or if receive or transmit are disabled in the Network Control register.

#### 40.6.3.6 DMA Packet Buffer

The DMA uses packet buffers for both transmit and receive paths. This mode allows multiple packets to be buffered in both transmit and receive directions. This allows the DMA to withstand far greater access latencies on the AHB and make more efficient use of the AHB bandwidth. There are two modes of operation—Full Store and Forward and Partial Store and Forward.

As described above (Section 40.6.3.2 ”Partial Store and Forward Using Packet Buffer DMA”), the DMA can be programmed into a low latency mode, known as Partial Store and Forward. For further details of this mode, see **Section 40.6.3.2 “Partial Store and Forward Using Packet Buffer DMA”**.

When the DMA is in full store and forward mode, full packets are buffered which provides the possibility to:

- Discard packets with error on the receive path before they are partially written out of the DMA, thus saving AHB bus bandwidth and driver processing overhead,
- Retry collided transmit frames from the buffer, thus saving AHB bus bandwidth,
- Implement transmit IP/TCP/UDP checksum generation offload.

With the packet buffers included, the structure of the GMAC data paths is shown in Figure 40-2.

Isochronous-IN is used to transmit a stream of data whose timing is implied by the delivery rate. Isochronous transfer provides periodic, continuous communication between host and device.

It guarantees bandwidth and low latencies appropriate for telephony, audio, video, etc.

If the endpoint is not available (TXRDY_TRER = 0), then the device does not answer to the host. An ERR_FL_ISO interrupt is generated in the UDPHS_EPTSTAx register and once enabled, then sent to the CPU.

The STALL_SNT command bit is not used for an ISO-IN endpoint.

- High Bandwidth Isochronous Endpoint Handling: IN Example

For high bandwidth isochronous endpoints, the DMA can be programmed with the number of transactions (BUFF_LENGTH field in UDPHS_DMACONTROLx) and the system should provide the required number of packets per microframe, otherwise, the host will notice a sequencing problem.

A response should be made to the first token IN recognized inside a microframe under the following conditions:

- If at least one bank has been validated, the correct DATAx corresponding to the programmed Number Of Transactions per Microframe (NB_TRANS) should be answered. In case of a subsequent missed or corrupted token IN inside the microframe, the USB 2.0 Core available data bank(s) that should normally have been transmitted during that microframe shall be flushed at its end. If this flush occurs, an error condition is flagged (ERR_FLUSH is set in UDPHS_EPTSTAx).
- If no bank is validated yet, the default DATA0 ZLP is answered and underflow is flagged (ERR_FL_ISO is set in UDPHS_EPTSTAx). Then, no data bank is flushed at microframe end.
- If no data bank has been validated at the time when a response should be made for the second transaction of NB_TRANS = 3 transactions microframe, a DATA1 ZLP is answered and underflow is flagged (ERR_FL_ISO is set in UDPHS_EPTSTAx). If and only if remaining untransmitted banks for that microframe are available at its end, they are flushed and an error condition is flagged (ERR_FLUSH is set in UDPHS_EPTSTAx).
- If no data bank has been validated at the time when a response should be made for the last programmed transaction of a microframe, a DATA0 ZLP is answered and underflow is flagged (ERR_FL_ISO is set in UDPHS_EPTSTAx). If and only if the remaining untransmitted data bank for that microframe is available at its end, it is flushed and an error condition is flagged (ERR_FLUSH is set in UDPHS_EPTSTAx).
- If at the end of a microframe no valid token IN has been recognized, no data bank is flushed and no error condition is reported.

At the end of a microframe in which at least one data bank has been transmitted, if less than NB_TRANS banks have been validated for that microframe, an error condition is flagged (ERR_TRANS is set in UDPHS_EPTSTAx).

Cases of Error (in UDPHS_EPTSTAx)

- ERR_FL_ISO: There was no data to transmit inside a microframe, so a ZLP is answered by default.
- ERR_FLUSH: At least one packet has been sent inside the microframe, but the number of token INs received is less than the number of transactions actually validated (TXRDY_TRER) and likewise with the NB_TRANS programmed.
- ERR_TRANS: At least one packet has been sent inside the microframe, but the number of token INs received is less than the number of programmed NB_TRANS transactions and the packets not requested were not validated.
- ERR_FL_ISO + ERR_FLUSH: At least one packet has been sent inside the microframe, but the data has not been validated in time to answer one of the following token INs.
- ERR_FL_ISO + ERR_TRANS: At least one packet has been sent inside the microframe, but the data has not been validated in time to answer one of the following token INs and the data can be discarded at the microframe end.
- ERR_FLUSH + ERR_TRANS: The first token IN has been answered and it was the only one received, a second bank has been validated but not the third, whereas NB_TRANS was waiting for three transactions.
- ERR_FL_ISO + ERR_FLUSH + ERR_TRANS: The first token IN has been treated, the data for the second Token IN was not available in time, but the second bank has been validated before the end of the microframe. The third bank has not been validated, but three transactions have been set in NB_TRANS.

#### 41.6.10.4 Data OUT

- Bulk OUT or Interrupt OUT

Like data IN, data OUT packets are sent by the host during the data or the status stage of control transfer or during an interrupt/bulk/isochronous OUT transfer. Data buffers are sent packet by packet under the control of the application or under the control of the DMA channel.

- Bulk OUT or Interrupt OUT: Receiving a Packet Under Application Control (Host to Device)

Algorithm Description for Each Packet:

- The application enables an interrupt on RXRDY_TXKL.
- When an interrupt on RXRDY_TXKL is received, the application knows that UDPHS_EPTSTAx register BYTE_COUNT bytes have been received.
- The application reads the BYTE_COUNT bytes from the endpoint.

This bit is set by hardware after a new packet has been stored in the endpoint FIFO.

This bit is cleared by the device firmware after reading the OUT data from the endpoint.

For multi-bank endpoints, this bit may remain active even when cleared by the device firmware, this if an other packet has been received meanwhile.

Hardware assertion of this bit may generate an interrupt if enabled by the UDPHS_EPTCTLx register RXRDY_TXKL bit.

This bit is reset by UDPHS_EPTRST register EPT_x (reset endpoint) and by UDPHS_EPTCTLDISx (disable endpoint).

– **KILL Bank** (for IN endpoint):

– The bank is really cleared or the bank is sent, BUSY_BANK_STA is decremented.

– The bank is not cleared but sent on the IN transfer, TX_COMPLT

– The bank is not cleared because it was empty. The user should wait that this bit is cleared before trying to clear another packet.

**Note:** "Kill a packet" may be refused if at the same time, an IN token is coming and the current packet is sent on the UDPHS line. In this case, the TX_COMPLT bit is set. Take notice however, that if at least two banks are ready to be sent, there is no problem to kill a packet even if an IN token is coming. In fact, in that case, the current bank is sent (IN transfer) and the last bank is killed.

**TX_COMPLT: Transmitted IN Data Complete (cleared upon USB reset)**

This bit is set by hardware after an IN packet has been sent.

This bit is reset by UDPHS_EPTRST register EPT_x (reset endpoint), and by UDPHS_EPTCTLDISx (disable endpoint).

**TXRDY_TRER: TX Packet Ready/Transaction Error (cleared upon USB reset)**

– **TX Packet Ready**:

This bit is cleared by hardware, as soon as the packet has been sent.

For Multi-bank endpoints, this bit may remain clear even after software is set if another bank is available to transmit.

Hardware clear of this bit may generate an interrupt if enabled by the UDPHS_EPTCTLx register TXRDY_TRER bit.

This bit is reset by UDPHS_EPTRST register EPT_x (reset endpoint), and by UDPHS_EPTCTLDISx (disable endpoint).

– **Transaction Error** (for high bandwidth isochronous OUT endpoints) (Read-Only):

This bit is set by hardware when a transaction error occurs inside one microframe.

If one toggle sequencing problem occurs among the n-transactions (n = 1, 2 or 3) inside a microframe, then this bit is still set as long as the current bank contains one "bad" n-transaction (refer to "CURBK: Current Bank (cleared upon USB reset)" ). As soon as the current bank is relative to a new "good" n-transactions, then this bit is reset.

**Note 1:** A transaction error occurs when the toggle sequencing does not comply with the *Universal Serial Bus Specification, Rev 2.0* (5.9.2 High Bandwidth Isochronous endpoints) (Bad PID, missing data, etc.)

**2:** When a transaction error occurs, the user may empty all the "bad" transactions by clearing the Received OUT Data flag (RXRDY_TXKL).

If this bit is reset, then the user should consider that a new n-transaction is coming.

This bit is reset by UDPHS_EPTRST register EPT_x (reset endpoint), and by UDPHS_EPTCTLDISx (disable endpoint).

**ERR_FL_ISO: Error Flow (cleared upon USB reset)**

This bit is set by hardware when a transaction error occurs.

– Isochronous IN transaction is missed, the micro has no time to fill the endpoint (underflow).

– Isochronous OUT data is dropped because the bank is busy (overflow).

This bit is reset by UDPHS_EPTRST register EPT_x (reset endpoint) and by UDPHS_EPTCTLDISx (disable endpoint).

**ERR_CRC_NTR: CRC ISO Error/Number of Transaction Error (cleared upon USB reset)**

– **CRC ISO Error** (for Isochronous OUT endpoints) (Read-only):

This bit is set by hardware if the last received data is corrupted (CRC error on data).

This bit is updated by hardware when new data is received (Received OUT Data bit).

– **Number of Transaction Error** (for High Bandwidth Isochronous IN endpoints):

### 42.7.1 UHPHS Host Controller Capability Register

**Name:** UHPHS_HCCAPBASE

**Access:** Read-only

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

**CAPLENGTH: Capability Registers Length**

10h: Default value.

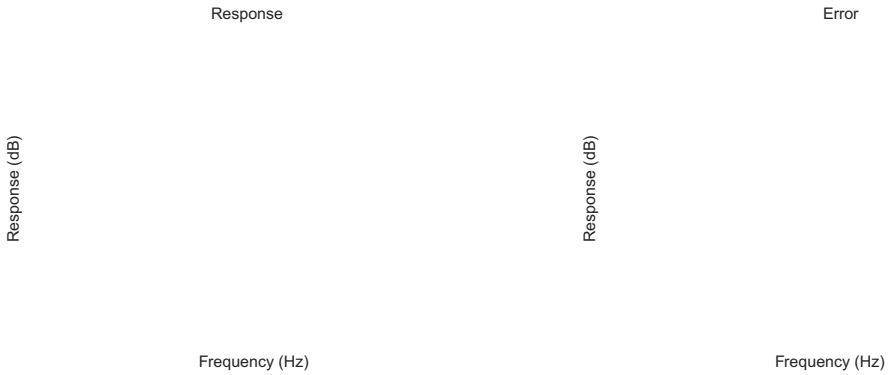
This field is used as an offset to add to register base to find the beginning of the Operational Register Space.

**HCIVERSION: Host Controller Interface Version Number**

0100h: Default value.

This is a two-byte field containing a BCD encoding of the EHCI revision number supported by this host controller. The most significant byte of this field represents a major revision and the least significant byte is the minor revision.

**Figure 43-10: De-emphasis Filter: Frequency Response & Error ($f_S$ = 48 kHz)**




### 43.6.4 Attenuator and Recommended Input Levels

The CLASSD features a digital attenuator with an attenuation range of 0–77 dB and a step size of 1 dB. When attenuation greater than 77 dB is programmed, the attenuator mutes the channel.

To avoid saturations in the PWM stage, it is recommended to avoid input levels greater than 1 dB below the digital full scale (-1 dBFS). This can done by programming a minimum attenuation of 1 dB.

### 44.8.10 I2SC Transmitter Holding Register

**Name:** I2SC_THR

**Address:** 0xF8050024 (0), 0xFC04C024 (1)

**Access:** Write-only

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

**THR: Transmitter Holding Register**

Next data word to be transmitted after the current word if TXRDY is not set. If I2SC_MR.DATALENGTH specifies fewer than 32 bits, data is right-justified in the THR field.

## 46.6 Functional Description

### 46.6.1 Transfer Format

The data put on the TWD line must be 8 bits long. Data is transferred MSB first; each byte must be followed by an acknowledgement. The number of bytes per transfer is unlimited. See Figure 46-3.

Each transfer begins with a START condition and terminates with a STOP condition. See Figure 46-2.

- A high-to-low transition on the TWD line while TWCK is high defines the START condition.
- A low-to-high transition on the TWD line while TWCK is high defines the STOP condition.

**Figure 46-2: START and STOP Conditions**

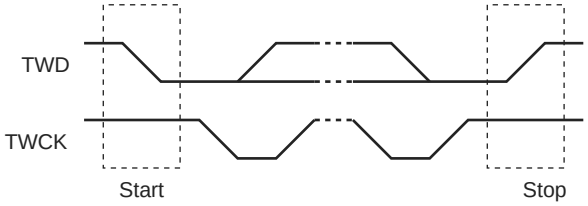
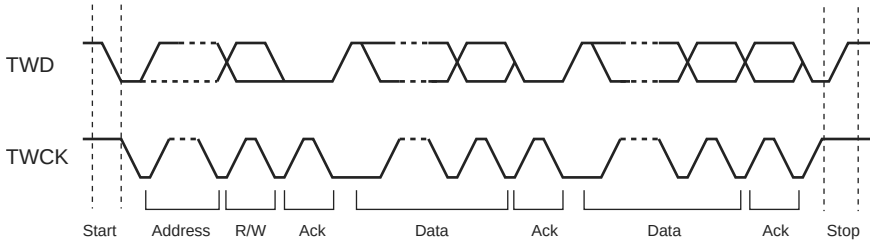



**Figure 46-3: Transfer Format**




### 46.6.2 Modes of Operation

The TWIHS has different modes of operation:

- Master Transmitter mode (Standard and Fast modes only)
- Master Receiver mode (Standard and Fast modes only)
- Multimaster Transmitter mode (Standard and Fast modes only)
- Multimaster Receiver mode (Standard and Fast modes only)
- Slave Transmitter mode (Standard, Fast and High-speed modes)
- Slave Receiver mode (Standard, Fast and High-speed modes)

These modes are described in the following sections.

**Figure 47-17: ASK Modulator Output**

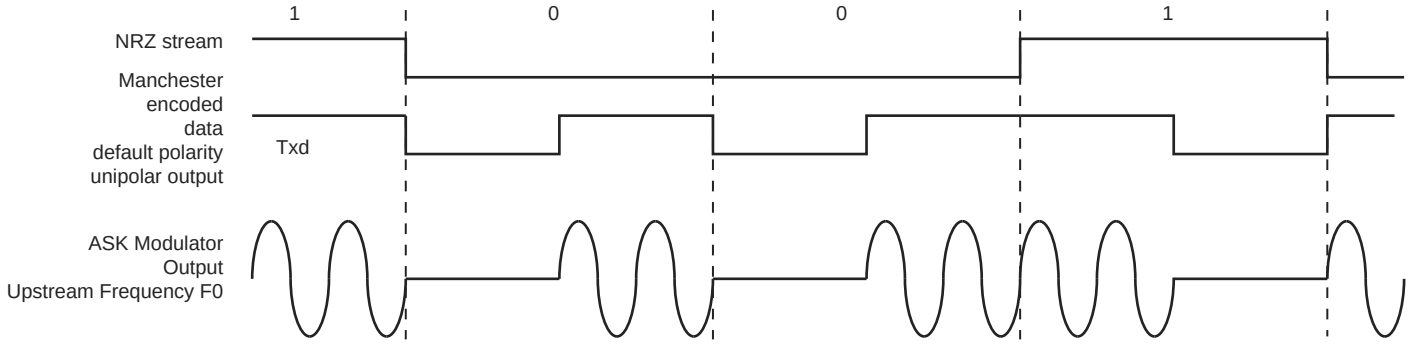
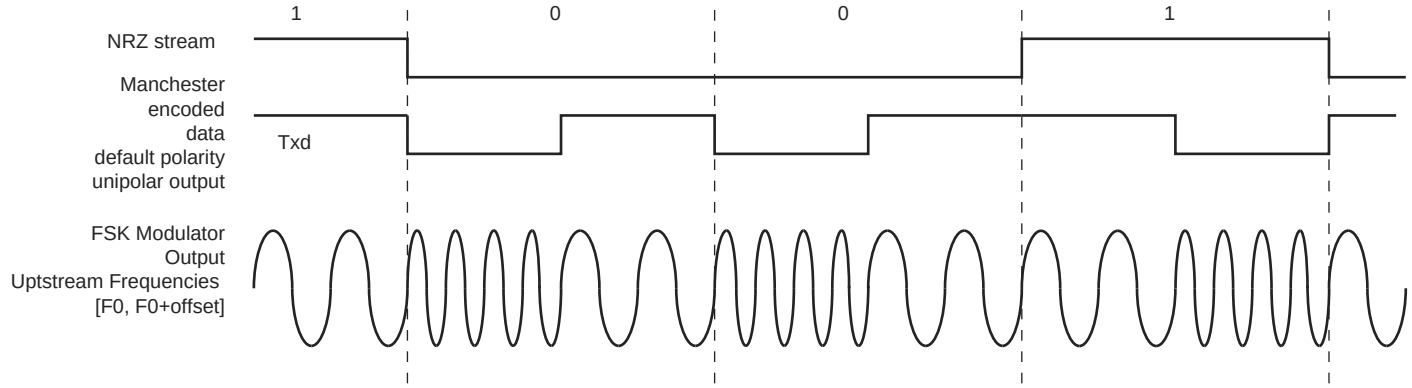



**Figure 47-18: FSK Modulator Output**



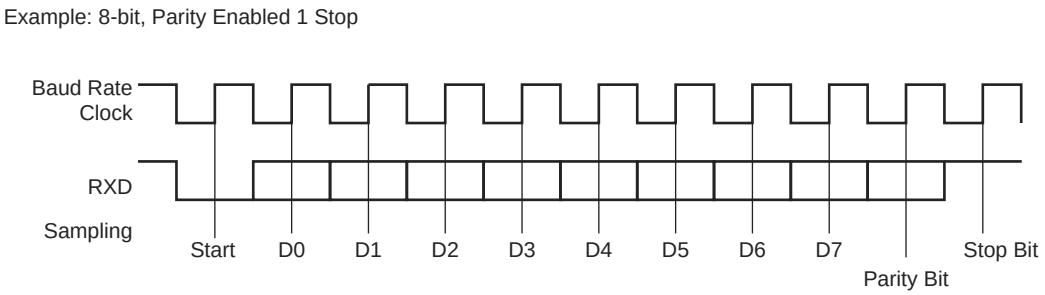

#### 47.7.3.6 Synchronous Receiver

In Synchronous mode (SYNC = 1), the receiver samples the RXD signal on each rising edge of the baud rate clock. If a low level is detected, it is considered as a start. All data bits, the parity bit and the stop bits are sampled and the receiver waits for the next start bit. Synchronous mode operations provide a high-speed transfer capability.

Configuration fields and bits are the same as in Asynchronous mode.

Figure 47-19 illustrates a character reception in Synchronous mode.

**Figure 47-19: Synchronous Mode Character Reception**




#### 47.7.3.7 Receiver Operations

When a character reception is completed, it is transferred to the Receive Holding Register (FLEX_US_RHR) and the FLEX_US_CSR.RXRDY bit is raised. If a character is completed while the RXRDY is set, the Overrun Error (OVRE) bit is set. The last character is transferred into FLEX_US_RHR and overwrites the previous one. The OVRE bit is cleared by writing a one to Reset Status bit FLEX_US_CR.RSTSTA.

### 47.10.15 USART Interrupt Mask Register (SPI_MODE)

**Name:** FLEX_US_IMR (SPI_MODE)

**Address:** 0xF8034210 (0), 0xF8038210 (1), 0xFC010210 (2), 0xFC014210 (3), 0xFC018210 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | CMP | – | – | NSSE | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | UNRE | TXEMPTY | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | OVRE | – | – | – | TXRDY | RXRDY |

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

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

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

**RXRDY: RXRDY Interrupt Mask**

**TXRDY: TXRDY Interrupt Mask**

**OVRE: Overrun Error Interrupt Mask**

**TXEMPTY: TXEMPTY Interrupt Mask**

**UNRE: SPI Underrun Error Interrupt Mask**

**NSSE: NSS Line (Driving CTS Pin) Rising or Falling Edge Event**

**CMP: Comparison Interrupt Mask**

### 54.6.15 Output Controller

The output controller defines the output level changes on TIOAx and TIOBx following an event. TIOBx control is used only if TIOBx is defined as output (not as an external event).

The following events control TIOAx and TIOBx:

- Software trigger
- External event
- RC compare

RA Compare controls TIOAx, and RB Compare controls TIOBx. Each of these events can be programmed to set, clear or toggle the output as defined in the corresponding parameter in TC_CMR.

### 54.6.16 Quadrature Decoder

#### 54.6.16.1 Description

The quadrature decoder (QDEC) is driven by TIOA0, TIOB0 and TIOB1 input pins and drives the timer counter of channel 0 and 1. Channel 2 can be used as a time base in case of speed measurement requirements (refer to Figure 54-17).

When writing a ‘0’ to TC_BMR.QDEN, the QDEC is bypassed and the IO pins are directly routed to the timer counter function.

TIOA0 and TIOB0 are to be driven by the two dedicated quadrature signals from a rotary sensor mounted on the shaft of the off-chip motor.

A third signal from the rotary sensor can be processed through pin TIOB1 and is typically dedicated to be driven by an index signal if it is provided by the sensor. This signal is not required to decode the quadrature signals PHA, PHB.

TC_CMRx.TCCLKS must be configured to select XC0 input (i.e., 0x101). TC_BMR.TC0XC0S has no effect as soon as the QDEC is enabled.

Either speed or position/revolution can be measured. Position channel 0 accumulates the edges of PHA, PHB input signals giving a high accuracy on motor position whereas channel 1 accumulates the index pulses of the sensor, therefore the number of rotations. Concatenation of both values provides a high level of precision on motion system position.

In Speed mode, position cannot be measured but revolution can be measured.

Inputs from the rotary sensor can be filtered prior to downstream processing. Accommodation of input polarity, phase definition and other factors are configurable.

Interruptions can be generated on different events.

A compare function (using TC_RC) is available on channel 0 (speed/position) or channel 1 (rotation) and can generate an interrupt by means of TC_SRx.CPCS.

# 60. Advanced Encryption Standard (AES)

## 60.1 Description

The Advanced Encryption Standard (AES) is compliant with the American *FIPS (Federal Information Processing Standard) Publication 197* specification.

The AES supports all five confidentiality modes of operation for symmetrical key block cipher algorithms (ECB, CBC, OFB, CFB and CTR), as specified in the *NIST Special Publication 800-38A Recommendation,* as well as Galois/Counter Mode (GCM) as specified in the *NIST Special Publication 800-38D Recommendation*. It is compatible with all these modes via DMA Controller channels, minimizing processor intervention for large buffer transfers.

The AES key is loaded by the software. The 128-bit/192-bit/256-bit AES key is stored in the AES Key Register made of four/six/eight 32-bit write-only AES Key Word registers (AES_KEYWR0–7).

The 128-bit input data and initialization vector (for some modes) are each stored in four 32-bit write-only AES Input Data registers (AES_IDATAR0–3) and AES Initialization Vector registers (AES_IVR0–3).

As soon as the initialization vector, the input data and the key are configured, the encryption/decryption process may be started. Then the encrypted/decrypted data are ready to be read out on the four 32-bit AES Output Data registers (AES_ODATAR0–3) or through the DMA channels.

## 60.2 Embedded Characteristics

- Compliant with *FIPS Publication 197, Advanced Encryption Standard (AES)*
- 128-bit/192-bit/256-bit Cryptographic Key
- 10/12/14 Clock Cycles Encryption/Decryption Inherent Processing Time with a 128-bit/192-bit/256-bit Cryptographic Key
- Double Input Buffer Optimizes Runtime
- Automatic Padding supported for IPSEC and SSL standards
- IPSEC and SSL Protocol Layers Improved Performances (Tightly coupled with SHA)
- Support of the Modes of Operation Specified in the *NIST Special Publication 800-38A* and *NIST Special Publication 800-38D*:
  - Electronic Codebook (ECB)
  - Cipher Block Chaining (CBC) including CBC-MAC
  - Cipher Feedback (CFB)
  - Output Feedback (OFB)
  - Counter (CTR)
  - Galois/Counter Mode (GCM)
  - XEX-Based Tweaked-Codebook Mode (XTS)
- 8, 16, 32, 64 and 128-bit Data Sizes Possible in CFB Mode
- Last Output Data Mode Allows Optimized Message Authentication Code (MAC) Generation
- Connection to DMA Optimizes Data Transfers for all Operating Modes

### 64.6.2 SECUMOD System Status Register

**Name:** SECUMOD_SYSR

**Address:** 0xFC040004

**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 |
|---|---|---|---|---|---|---|---|
| SCRAMB | AUTOBKP | – | – | SWKUP | BACKUP | ERASE_ON | ERASE_DONE |

**ERASE_DONE: Erasable Memories State (RW)**

0: Secure memories content has not been erased since the last clear.

1: Secure memories content has been erased since the last clear. The user must write 1 into this bit to clear this flag. Note that not clearing this flag does not prevent the next erase processes. This flag also activates the SECURAM interrupt line as long as it is not cleared.

**ERASE_ON: Erase Process Ongoing (RO)**

0: Erase automaton is not running.

1: Erase automaton is currently running, memories are not accessible.

When ERASE_ON returns to 0, ERASE_DONE is set after half a period of ICLK.

| ERASE_ON | ERASE_DONE | Status | Action |
|---|---|---|---|
| 0 | 0 | No Erase ongoing or since the last Erase. | Nothing. |
| 1 | 0 | An Erase process is running. | Wait until the ERASE_ON flag is reset.<br>ERASE_DONE will rise, see line below. |
| 0 | 1 | An Erase occurred and is finished. | Clear the ERASE_DONE flag. |
| 1 | 1 | An Erase process is running.<br>The ERASE_DONE flag refers to a previous Erase process, but was not cleared. | Wait until the ERASE_ON flag is reset, then clear the ERASE_DONE flag. |

**BACKUP: Backup Mode (RO)**

0: Normal mode active.

1: Backup mode active.

**SWKUP: SWKUP State (RO)**

0: No SWKUP signal sent since the last clear.

1: SWKUP signal has been sent since the last clear.

**NOPEN: No Pen Contact (cleared on read)**

0: No loss of pen contact since the last read of ADC_ISR.

1: At least one loss of pen contact since the last read of ADC_ISR.

**PENS: Pen Detect Status**

0: The pen does not press the screen.

1: The pen presses the screen.

**Note:** PENS is not a source of interruption.

**Table 66-76: QSPI1 IOSET3 Timings (Continued)**

| Symbol | Power Supply / Parameter | 1.8V Min | 1.8V Max | 3.3V Min | 3.3V Max | Unit |
|---|---|---|---|---|---|---|
| Mode 1 | | | | | | |
| $QSPI_3$ | QIOx Input setup time before SCK rises | 13.1 | – | 10.9 | – | ns |
| $QSPI_4$ | QIOx Input hold time after SCK rises | 0.4 | – | 0.2 | – | ns |
| $QSPI_5$ | SCK rising to QIOx valid | 0 | 1.7 | 0 | 1.7 | ns |
| Mode 2 | | | | | | |
| $QSPI_6$ | QIOx Input setup time before SCK rises | 2.2 | – | 2.1 | – | ns |
| $QSPI_7$ | QIOx Input hold time after SCK rises | 0.4 | – | 0.2 | – | ns |
| $QSPI_8$ | SCK rising to QIOx valid | 0 | 2 | 0 | 1.8 | ns |
| Mode 3 | | | | | | |
| $QSPI_9$ | QIOx Input setup time before SCK falls | 12.5 | – | 11 | – | ns |
| $QSPI_{10}$ | QIOx Input hold time after SCK falls | 0.9 | – | 0.7 | – | ns |
| $QSPI_{11}$ | SCK falling to QIOx valid | 0 | 1.1 | 0 | 1.7 | ns |

## 66.20 MPDDRC Timings

### 66.20.1 Board Design Constraints

As the SAMA5D2 series embeds impedance calibrated pads, there are no capacitive constraints on DDR signals. However, a board must be designed and equipped in order to respect propagation time and intrinsic delay in the SDRAM device. In all cases, line length to memory device must not exceed 5 cm.

### 66.20.2 DDR2-SDRAM

Note: For DDR2 memory, the SHIFT_SAMPLING field value in the MPRDDRC_RD_DATA_PATH register must be configured to 1.

**Table 66-77: System Clock Waveform Parameters**

| Symbol | Parameter | Conditions | Min | Max | Unit |
|---|---|---|---|---|---|
| $t_{DDRCK}$ | DDRCK Cycle Time | VDDCORE[1.1V, 1.32V], $T_A$ = 85°C | 7.5 | 8.0 | ns |
| | | VDDCORE[1.2V, 1.32V], VDDIODDR[1.75V, 1.9V], $T_A$ = 85°C | 6.0 | 8.0 | ns |

### 66.20.3 LPDDR1-SDRAM

Note: For LPDDR1 memory, the SHIFT_SAMPLING field value in the MPRDDRC_RD_DATA_PATH register must be configured as follows:

SHIFT_SAMPLING = 0 for 0 < DDR_CLK < 94 MHz

SHIFT_SAMPLING = 1 for 94 MHz < DDR_CLK < 166 MHz

When the MCAN loses arbitration again at the immediately following retransmission, then actually and previously transmitted Tx Buffers are the same and this Tx Buffer's MCAN_TXBRP.TRPxx bit is cleared and its MCAN_TXBCF.CFxx bit is set.

**Workaround:** None.

### 71.2.18 MCAN Tx FIFO Message

**Issue: Tx FIFO message sequence inversion**

Assume the case that there are two Tx FIFO messages in the output pipeline of the Tx Message Handler. Transmission of Tx FIFO message 1 is started:

Position 1: Tx FIFO message 1 (transmission ongoing)

Position 2: Tx FIFO message 2

Position 3: --

Now a non Tx FIFO message with a higher CAN priority is requested. Due to its priority it will be inserted into the output pipeline. The TxMH performs so called "message scans" to keep the output pipeline up to date with the highest priority messages from the Message RAM. After the following two message scans, the output pipeline has the following content:

Position 1: Tx FIFO message 1 (transmission ongoing)

Position 2: non Tx FIFO message with higher CAN priority

Position 3: Tx FIFO message 2

If the transmission of Tx FIFO message 1 is not successful (lost arbitration or CAN bus error) it is pushed from the output pipeline by the non Tx FIFO message with higher CAN priority. The following scan re-inserts Tx FIFO message 1 into the output pipeline at position 3:

Position 1: non Tx FIFO message with higher CAN priority (transmission ongoing)

Position 2: Tx FIFO message 2

Position 3: Tx FIFO message 1

Now Tx FIFO message 2 is in the output pipeline in front of Tx FIFO message 1 and they are transmitted in that order, resulting in a message sequence inversion.

**Workaround:**

1. First Workaround

Use two dedicated Tx Buffers, e.g. use Tx Buffers 4 and 5 instead of the Tx FIFO. The pseudo-code below replaces the function that fills the Tx FIFO.

Write message to Tx Buffer 4.

Transmit Loop:

- Request Tx Buffer 4 - write MCAN_TXBAR.A4
- Write message to Tx Buffer 5
- Wait until transmission of Tx Buffer 4 completed - MCAN_IR.TC, read MCAN_TXBTO.TO4
- Request Tx Buffer 5 - write MCAN_TXBAR.A5
- Write message to Tx Buffer 4
- Wait until transmission of Tx Buffer 5 completed - MCAN_IR.TC, read MCAN_TXBTO.TO5

2. Second Workaround

Assure that only one Tx FIFO element is pending for transmission at any time. The Tx FIFO elements may be filled at any time with messages to be transmitted, but their transmission requests are handled separately. Each time a Tx FIFO transmission has completed and the Tx FIFO gets empty (MCAN_IR.TFE = '1') the next Tx FIFO element is requested.

3. Third Workaround

Use only a Tx FIFO. Send the message with the higher priority also from Tx FIFO.

Drawback: The higher priority message has to wait until the preceding messages in the Tx FIFO have been sent.