



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, EBI/EMI, Ethernet, IrDA, SD, SPI, UART/USART, USB
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	117
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	144-LQFP
Supplier Device Package	144-LQFP (20x20)
Purchase URL	<a href="https://www.e-xfl.com/product-detail/microchip-technology/atsam4e16ea-anr">https://www.e-xfl.com/product-detail/microchip-technology/atsam4e16ea-anr</a>

## 11. Cortex-M4 processor

### 11.1 Description

The Cortex-M4 processor is a high performance 32-bit processor designed for the microcontroller market. It offers significant benefits to developers, including outstanding processing performance combined with fast interrupt handling, enhanced system debug with extensive breakpoint and trace capabilities, efficient processor core, system and memories, ultra-low power consumption with integrated sleep modes, and platform security robustness, with integrated memory protection unit (MPU).

The Cortex-M4 processor is built on a high-performance processor core, with a 3-stage pipeline Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design, providing high-end processing hardware including IEEE754-compliant single-precision floating-point computation, a range of single-cycle and SIMD multiplication and multiply-with-accumulate capabilities, saturating arithmetic and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M4 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M4 processor implements a version of the Thumb® instruction set based on Thumb-2 technology, ensuring high code density and reduced program memory requirements. The Cortex-M4 instruction set provides the exceptional performance expected of a modern 32-bit architecture, with the high code density of 8-bit and 16-bit microcontrollers.

The Cortex-M4 processor closely integrates a configurable NVIC, to deliver industry-leading interrupt performance. The NVIC includes a non-maskable interrupt (NMI), and provides up to 256 interrupt priority levels. The tight integration of the processor core and NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing the interrupt latency. This is achieved through the hardware stacking of registers, and the ability to suspend load-multiple and store-multiple operations. Interrupt handlers do not require wrapping in assembler code, removing any code overhead from the ISRs. A tail-chain optimization also significantly reduces the overhead when switching from one ISR to another.

To optimize low-power designs, the NVIC integrates with the sleep modes, that include a deep sleep function that enables the entire device to be rapidly powered down while still retaining program state.

#### 11.1.1 System Level Interface

The Cortex-M4 processor provides multiple interfaces using AMBA® technology to provide high speed, low latency memory accesses. It supports unaligned data accesses and implements atomic bit manipulation that enables faster peripheral controls, system spinlocks and thread-safe Boolean data handling.

The Cortex-M4 processor has a Memory Protection Unit (MPU) that provides fine grain memory control, enabling applications to utilize multiple privilege levels, separating and protecting code, data and stack on a task-by-task basis. Such requirements are becoming critical in many embedded applications such as automotive.

#### 11.1.2 Integrated Configurable Debug

The Cortex-M4 processor implements a complete hardware debug solution. This provides high system visibility of the processor and memory through either a traditional JTAG port or a 2-pin Serial Wire Debug (SWD) port that is ideal for microcontrollers and other small package devices.

For system trace the processor integrates an Instrumentation Trace Macrocell (ITM) alongside data watchpoints and a profiling unit. To enable simple and cost-effective profiling of the system events these generate, a Serial Wire Viewer (SWV) can export a stream of software-generated messages, data trace, and profiling information through a single pin.

### 11.6.5.15 TST and TEQ

Test bits and Test Equivalence.

#### Syntax

```
TST{cond} Rn, Operand2
TEQ{cond} Rn, Operand2
```

where

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

**Rn** is the register holding the first operand.

**Operand2** is a flexible second operand. See “Flexible Second Operand” for details of the options.

#### Operation

These instructions test the value in a register against *Operand2*. They update the condition flags based on the result, but do not write the result to a register.

The TST instruction performs a bitwise AND operation on the value in *Rn* and the value of *Operand2*. This is the same as the ANDS instruction, except that it discards the result.

To test whether a bit of *Rn* is 0 or 1, use the TST instruction with an *Operand2* constant that has that bit set to 1 and all other bits cleared to 0.

The TEQ instruction performs a bitwise Exclusive OR operation on the value in *Rn* and the value of *Operand2*. This is the same as the EORS instruction, except that it discards the result.

Use the TEQ instruction to test if two values are equal without affecting the V or C flags.

TEQ is also useful for testing the sign of a value. After the comparison, the N flag is the logical Exclusive OR of the sign bits of the two operands.

#### Restrictions

Do not use SP and do not use PC.

#### Condition Flags

These instructions:

- Update the N and Z flags according to the result
- Can update the C flag during the calculation of *Operand2*, see “Flexible Second Operand”
- Do not affect the V flag.

#### Examples

```
TST    R0, #0x3F8 ; Perform bitwise AND of R0 value to 0x3F8,
                ; APSR is updated but result is discarded
TEQEQ  R10, R9    ; Conditionally test if value in R10 is equal to
                ; value in R9, APSR is updated but result is discarded.
```

## 11.6.8 Packing and Unpacking Instructions

The table below shows the instructions that operate on packing and unpacking data.

**Table 11-23. Packing and Unpacking Instructions**

Mnemonic	Description
PKH	Pack Halfword
SXTAB	Extend 8 bits to 32 and add
SXTAB16	Dual extend 8 bits to 16 and add
SXTAH	Extend 16 bits to 32 and add
SXTB	Sign extend a byte
SXTB16	Dual extend 8 bits to 16 and add
SXTH	Sign extend a halfword
UXTAB	Extend 8 bits to 32 and add
UXTAB16	Dual extend 8 bits to 16 and add
UXTAH	Extend 16 bits to 32 and add
UXTB	Zero extend a byte
UXTB16	Dual zero extend 8 bits to 16 and add
UXTH	Zero extend a halfword

- **MEMFAULTPENDEd: Memory Management Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **USGFAULTPENDEd: Usage Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **SYSTICKACT: SysTick Exception Active**

Read:

0: The exception is not active.

1: The exception is active.

Note: The user can write to these bits to change the active status of the exceptions.

- Caution: A software that changes the value of an active bit in this register without a correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure that the software writing to this register retains and subsequently restores the current active status.

- Caution: After enabling the system handlers, to change the value of a bit in this register, the user must use a read-modify-write procedure to ensure that only the required bit is changed.

- **PENDSVACT: PendSV Exception Active**

0: The exception is not active.

1: The exception is active.

- **MONITORACT: Debug Monitor Active**

0: Debug monitor is not active.

1: Debug monitor is active.

- **SVCALLACT: SVC Call Active**

0: SVC call is not active.

1: SVC call is active.

- **USGFAULTACT: Usage Fault Exception Active**

0: Usage fault exception is not active.

1: Usage fault exception is active.

- **BUSFAULTACT: Bus Fault Exception Active**

0: Bus fault exception is not active.

1: Bus fault exception is active.

- **MEMFAULTACT: Memory Management Fault Exception Active**

0: Memory management fault exception is not active.

1: Memory management fault exception is active.

### 20.4.3.9 User Signature Area

Each product contains a user signature area of 512-bytes. It can be used for storage. Read, write and erase of this area is allowed.

See Figure 20-1 "Flash Memory Areas".

The sequence to read the user signature area is the following:

1. Execute the 'Start Read User Signature' command by writing EEFC\_FCR.FCMD with the STUS command. Field EEFC\_FCR.FARG is meaningless.
2. Wait until the bit EEFC\_FSR.FRDY falls to read the user signature area. The user signature area is located in the first 512 bytes of the Flash memory mapping. The 'Start Read User Signature' command reuses some addresses of the memory plane but the user signature area is physically different from the memory plane
3. To stop reading the user signature area, execute the 'Stop Read User Signature' command by writing EEFC\_FCR.FCMD with the SPUS command. Field EEFC\_FCR.FARG is meaningless.
4. When the SPUI command has been executed, the bit EEFC\_FSR.FRDY rises. If an interrupt was enabled by setting the bit EEFC\_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Note that during the sequence, the software cannot be fetched from the Flash or from the second plane in case of dual plane.

One error can be detected in EEFC\_FSR after this sequence:

- Command Error: A bad keyword has been written in EEFC\_FCR.

The sequence to write the user signature area is the following:

1. Write the full page, at any page address, within the internal memory area address space.
2. Execute the 'Write User Signature' command by writing EEFC\_FCR.FCMD with the WUS command. Field EEFC\_FCR.FARG is meaningless.
3. When programming is completed, the bit EEFC\_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC\_FMR.FRDY, the corresponding interrupt line of the interrupt controller is activated.

Two errors can be detected in EEFC\_FSR after this sequence:

- Command Error: A bad keyword has been written in EEFC\_FCR.
- Flash Error: At the end of the programming, the WriteVerify test of the Flash memory has failed.

The sequence to erase the user signature area is the following:

1. Execute the 'Erase User Signature' command by writing EEFC\_FCR.FCMD with the EUS command. Field EEFC\_FCR.FARG is meaningless.
2. When programming is completed, the bit EEFC\_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC\_FMR.FRDY, the corresponding interrupt line of the interrupt controller is activated.

Two errors can be detected in EEFC\_FSR after this sequence:

- Command Error: A bad keyword has been written in EEFC\_FCR.
- Flash Error: At the end of the programming, the EraseVerify test of the Flash memory has failed.

In the same way, the **Clear GPNVM** command (**CGPB**) is used to clear general-purpose NVM bits. The general-purpose NVM bit is deactivated when the corresponding bit in the pattern value is set to 1.

**Table 21-11. Set/Clear GP NVM Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	SGPB or CGPB
2	Write handshaking	DATA	GP NVM bit pattern value

General-purpose NVM bits can be read using the **Get GPNVM Bit** command (**GGPB**). The  $n^{\text{th}}$  GP NVM bit is active when bit  $n$  of the bit mask is set.

**Table 21-12. Get GP NVM Bit Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	GGPB
2	Read handshaking	DATA	GP NVM Bit Mask Status 0 = GP NVM bit is cleared 1 = GP NVM bit is set

#### 21.3.5.6 Flash Security Bit Command

A security bit can be set using the **Set Security Bit** command (SSE). Once the security bit is active, the Fast Flash programming is disabled. No other command can be run. An event on the Erase pin can erase the security bit once the contents of the Flash have been erased.

**Table 21-13. Set Security Bit Command**

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	SSE
2	Write handshaking	DATA	0

Once the security bit is set, it is not possible to access FFPI. The only way to erase the security bit is to erase the Flash.

To erase the Flash, perform the following steps:

1. Power-off the chip.
2. Power-on the chip with TST = 0.
3. Assert the ERASE pin for at least the ERASE pin assertion time as defined in the section “Electrical Characteristics”.
4. Power-off the chip.

Return to FFPI mode to check that the Flash is erased.

#### 21.3.5.7 Memory Write Command

This command is used to perform a write access to any memory location.

## 22.5.2 Cache Controller Configuration Register

**Name:** CMCC\_CFG

**Address:** 0x400C4004

**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
–	–	–	–	–	–	–	GCLKDIS

- **GCLKDIS: Disable Clock Gating**

0: Clock gating is activated.

1: Clock gating is disabled.

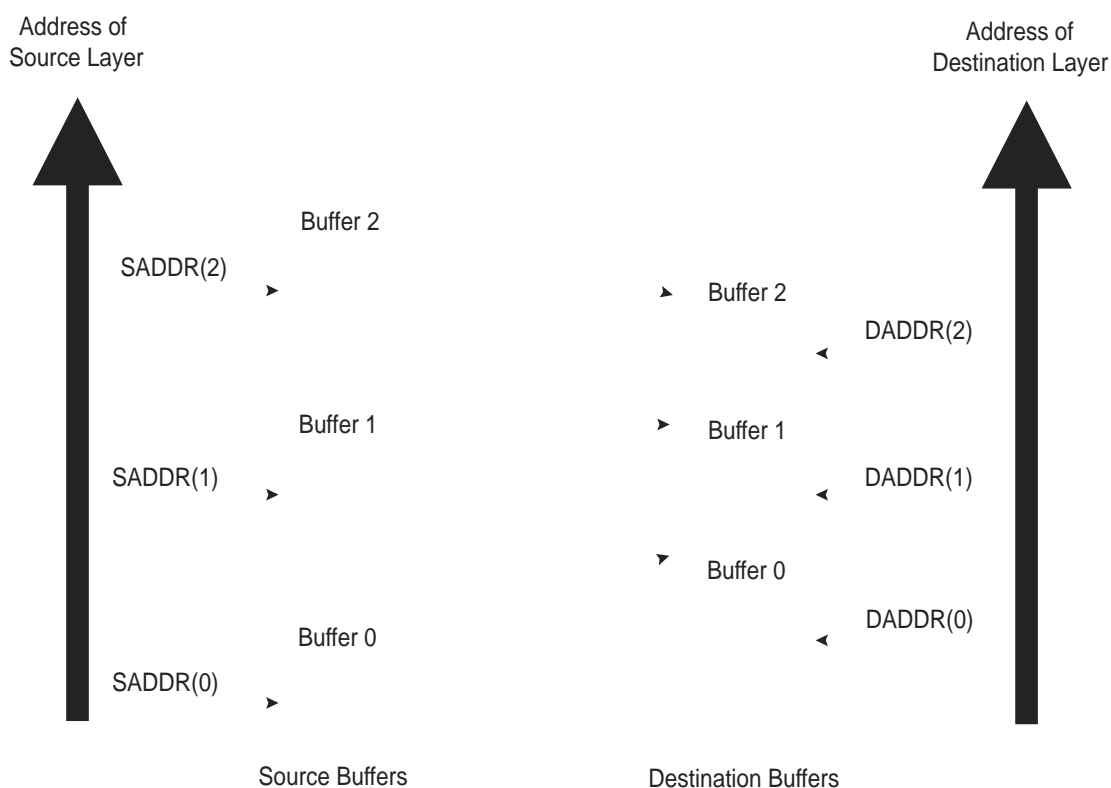


Note: Do not poll the DMAC\_CTRLAx.DONE bit in the DMAC memory map. Instead, poll the LLI.DMAC\_CTRLAx.DONE bit in the LLI for that buffer. If the poll LLI.DMAC\_CTRLAx.DONE bit is asserted, then this buffer transfer has completed. This LLI.DMAC\_CTRLAx.DONE bit was cleared at the start of the transfer.

16. The DMAC does not wait for the buffer interrupt to be cleared, but continues and fetches the next LLI from the memory location pointed to by the current DMAC\_DSCRx register, then automatically reprograms the DMAC\_SADDRx, DMAC\_CTRLAx, DMAC\_CTRLBx and DMAC\_DSCRx channel registers. DMAC\_DADDRx is left unchanged. The DMAC transfer continues until the DMAC samples the DMAC\_CTRLAx, DMAC\_CTRLBx and DMAC\_DSCRx registers at the end of a buffer transfer match that described in Row 1 of Table 25-3 on page 473. The DMAC then knows that the previous buffer transferred was the last buffer in the DMAC transfer.

The DMAC transfer might look like that shown in Figure 25-8. Note that the destination address is decrementing.

**Figure 25-8. DMAC Transfer with Linked List Source Address and Contiguous Destination Address**



The DMAC transfer flow is shown in Figure 25-9 on page 481.

## 25.8.20 DMAC Write Protection Status Register

**Name:** DMAC\_WPSR

**Address:** 0x400C01E8

**Access:** Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the DMAC\_WPSR.

1: A write protection violation has occurred since the last read of the DMAC\_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSR.

- **WPVSR: Write Protection Violation Source**

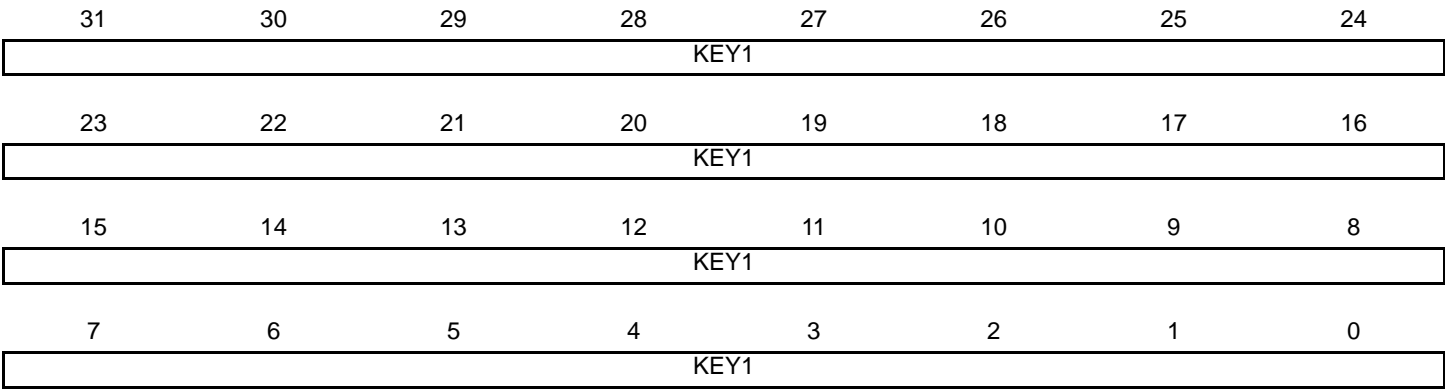
When WPVS = 1, WPVSR indicates the register address offset at which a write access has been attempted.

27.16.6 SMC Off-Chip Memory Scrambling Key1 Register

Name: SMC\_KEY1

Address: 0x40060084

Access: Write-once<sup>(1)</sup>



Note: 1. 'Write-once' access indicates that the first write access after a system reset prevents any further modification of the value of this register.  
2.

• KEY1: Off-Chip Memory Scrambling (OCMS) Key Part 1

When off-chip memory scrambling is enabled, KEY1 and KEY2 values determine data scrambling.

### 32.3.2 Chip ID Extension Register

**Name:** CHIPID\_EXID

**Address:** 0x400E0744

**Access:** Read-only

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

- **EXID: Chip ID Extension**

This field is cleared if CHIPID\_CIDR.EXT = 0.CHIPID\_EXID[1:0]: Package Type

Value	Name	Description
0	PACKAGE_TYPE	Package 144
1	PACKAGE_TYPE	Package 100

CHIPID\_EXID[4:2]: Flash Size

Value	Name	Description
0	FLASH_SIZE	1024 Kbytes
2	FLASH_SIZE	512 Kbytes

CHIPID\_EXID[31:5]: Product Number

Value	Name	Description
0x0012_020	PRODUCT_NUMBER	SAM4E Product Series

### 33.6.31 PIO Pad Pull-Down Enable Register

**Name:** PIO\_PPDER

**Address:** 0x400E0E94 (PIOA), 0x400E1094 (PIOB), 0x400E1294 (PIOC), 0x400E1494 (PIOD), 0x400E1694 (PIOE)

**Access:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

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

- **P0–P31: Pull-Down Enable**

0: No effect.

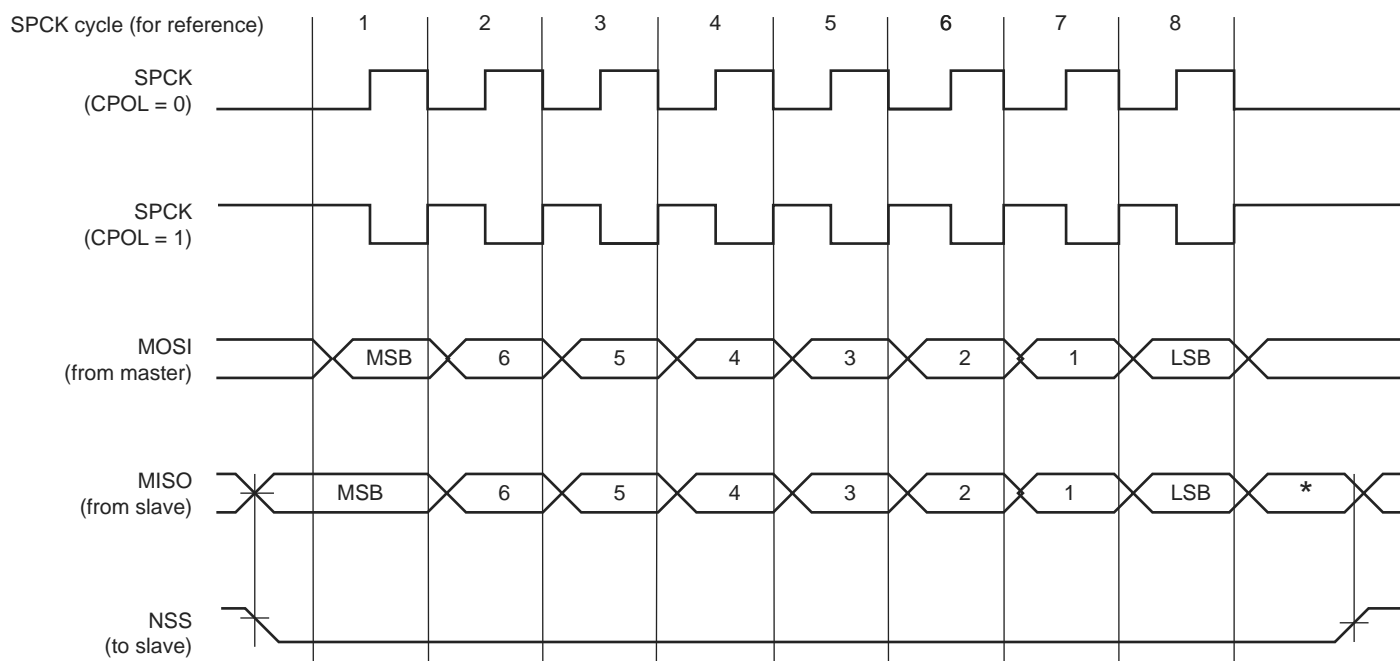
1: Enables the pull-down resistor on the I/O line.

**Table 34-4. SPI Bus Protocol Modes**

SPI Mode	CPOL	NCPHA	Shift SPCK Edge	Capture SPCK Edge	SPCK Inactive Level
1	0	0	Rising	Falling	Low
2	1	1	Rising	Falling	High
3	1	0	Falling	Rising	High

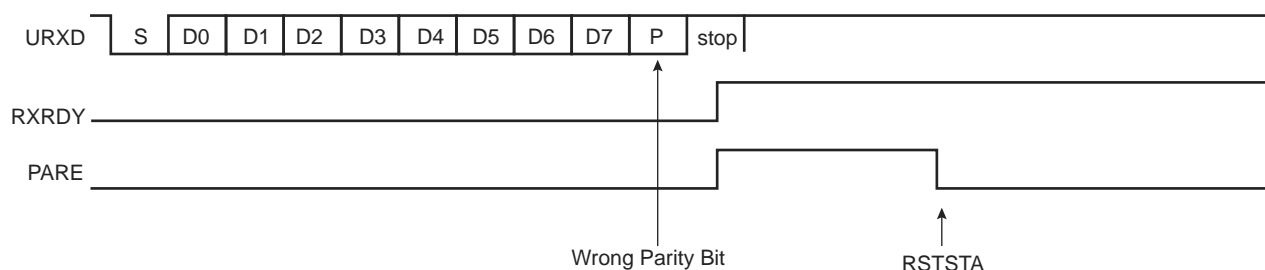
Figure 34-3 and Figure 34-4 show examples of data transfers.

**Figure 34-3. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)**



\* Not defined.

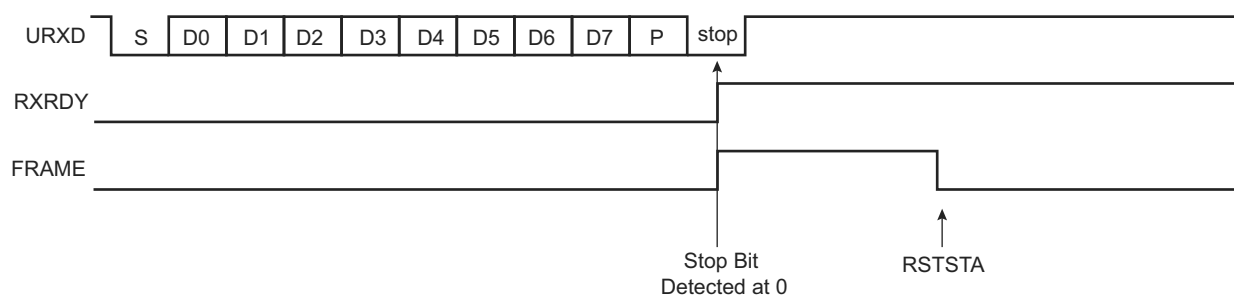
**Figure 36-7. Parity Error**



### 36.5.2.6 Receiver Framing Error

When a start bit is detected, it generates a character reception when all the data bits have been sampled. The stop bit is also sampled and when it is detected at 0, the FRAME (Framing Error) bit in UART\_SR is set at the same time the RXRDY bit is set. The FRAME bit remains high until the Control Register (UART\_CR) is written with the bit RSTSTA at 1.

**Figure 36-8. Receiver Framing Error**



## 36.5.3 Transmitter

### 36.5.3.1 Transmitter Reset, Enable and Disable

After device reset, the UART transmitter is disabled and must be enabled before being used. The transmitter is enabled by writing UART\_CR with the bit TXEN at 1. From this command, the transmitter waits for a character to be written in the Transmit Holding Register (UART\_THR) before actually starting the transmission.

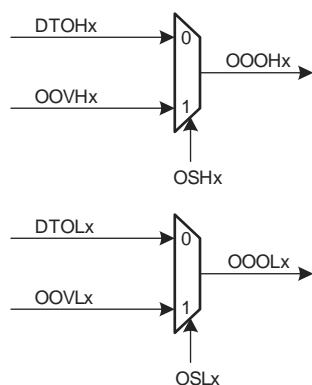
The programmer can disable the transmitter by writing UART\_CR with the bit TXDIS at 1. If the transmitter is not operating, it is immediately stopped. However, if a character is being processed into the internal shift register and/or a character has been written in the UART\_THR, the characters are completed before the transmitter is actually stopped.

The programmer can also put the transmitter in its reset state by writing the UART\_CR with the bit RSTTX at 1. This immediately stops the transmitter, whether or not it is processing characters.

### 36.5.3.2 Transmit Format

The UART transmitter drives the pin UTXD at the baud rate clock speed. The line is driven depending on the format defined in UART\_MR and the data stored in the internal shift register. One start bit at level 0, then the 8 data bits, from the lowest to the highest bit, one optional parity bit and one stop bit at 1 are consecutively shifted out as shown in the following figure. The field PARE in UART\_MR defines whether or not a parity bit is shifted out. When a parity bit is enabled, it can be selected between an odd parity, an even parity, or a fixed space or mark bit.

**Figure 39-10. Override Output Selection**



The fields OSHx and OSLx in the PWM Output Selection Register (PWM\_OS) allow the outputs of the dead-time generator DTOHx and DTOLx to be overridden by the value defined in the fields OOVHx and OOV Lx in the PWM Output Override Value Register (PWM\_OOV).

The set registers PWM Output Selection Set Register (PWM\_OSS) and PWM Output Selection Set Update Register (PWM\_OSSUPD) enable the override of the outputs of a channel regardless of other channels. In the same way, the clear registers PWM Output Selection Clear Register (PWM\_OSC) and PWM Output Selection Clear Update Register (PWM\_OSCUPD) disable the override of the outputs of a channel regardless of other channels.

By using buffer registers PWM\_OSSUPD and PWM\_OSCUPD, the output selection of PWM outputs is done synchronously to the channel counter, at the beginning of the next PWM period.

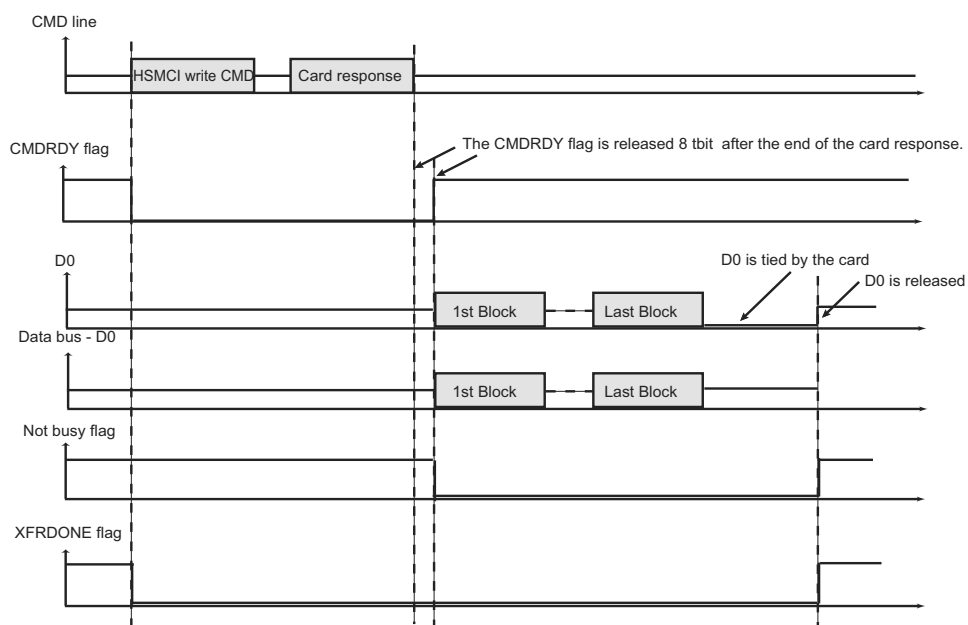
By using registers PWM\_OSS and PWM\_OSC, the output selection of PWM outputs is done asynchronously to the channel counter, as soon as the register is written.

The value of the current output selection can be read in PWM\_OS.

While overriding PWM outputs, the channel counters continue to run, only the PWM outputs are forced to user defined values.



**Figure 40-12. XFRDONE During a Write Access**



- **RTOE: Response Time-out Error Interrupt Disable**
- **DCRCE: Data CRC Error Interrupt Disable**
- **DTOE: Data Time-out Error Interrupt Disable**
- **CSTOE: Completion Signal Time out Error Interrupt Disable**
- **FIFOEMPTY: FIFO empty Interrupt Disable**
- **XFRDONE: Transfer Done Interrupt Disable**
- **ACKRCV: Boot Acknowledge Interrupt Disable**
- **ACKRCVE: Boot Acknowledge Error Interrupt Disable**
- **OVRE: Overrun Interrupt Disable**
- **UNRE: Underrun Interrupt Disable**

### 41.6.1.3 USB Transfer Event Definitions

As indicated below, transfers are sequential events carried out on the USB bus.

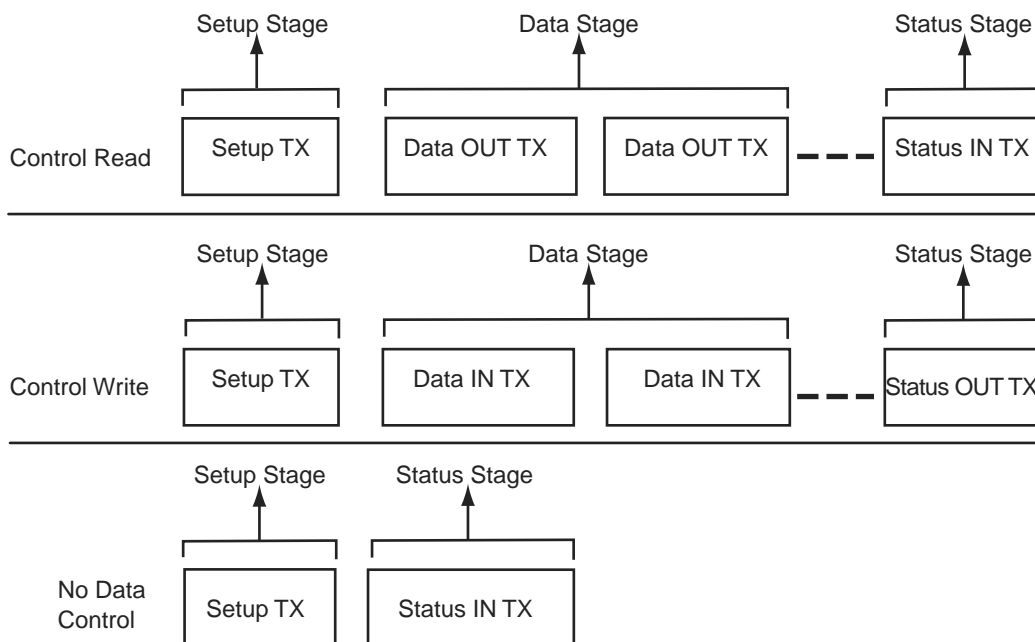
**Table 41-5. USB Transfer Events**

Transfer		Transaction
Direction	Type	
CONTROL (bidirectional)	Control <sup>(1)(3)</sup>	Setup transaction → Data IN transactions → Status OUT transaction
		Setup transaction → Data OUT transactions → Status IN transaction
		Setup transaction → Status IN transaction
IN (device toward host)	Interrupt IN	Data IN transaction → Data IN transaction
	Isochronous IN <sup>(2)</sup>	
	Bulk IN	
OUT (host toward device)	Interrupt OUT	Data OUT transaction → Data OUT transaction
	Isochronous OUT <sup>(2)</sup>	
	Bulk OUT	

- Notes:
1. Control transfer must use endpoints with no ping-pong attributes.
  2. Isochronous transfers must use endpoints with ping-pong attributes.
  3. Control transfers can be aborted using a stall handshake.

A status transaction is a special type of host-to-device transaction used only in a control transfer. The control transfer must be performed using endpoints with no ping-pong attributes. According to the control sequence (read or write), the USB device sends or receives a status transaction.

**Figure 41-4. Control Read and Write Sequences**



- Notes:
1. During the Status IN stage, the host waits for a zero length packet (Data IN transaction with no data) from the device using DATA1 PID. Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0*, for more information on the protocol layer.

- **FORCESTALL: Force Stall (used by Control, Bulk and Isochronous Endpoints)**

Read:

0: Normal state.

1: Stall state.

Write:

0: Return to normal state.

1: Send STALL to the host.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

Control endpoints: During the data stage and status stage, this bit indicates that the microcontroller cannot complete the request.

Bulk and interrupt endpoints: This bit notifies the host that the endpoint is halted.

The host acknowledges the STALL, device firmware is notified by the STALLSENT flag.

- **RX\_DATA\_BK1: Receive Data Bank 1 (only used by endpoints with ping-pong attributes)**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Notifies USB device that data have been read in the FIFO's Bank 1.

1: To leave the read value unchanged.

Read (set by the USB peripheral):

0: No data packet has been received in the FIFO's Bank 1.

1: A data packet has been received, it has been stored in FIFO's Bank 1.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to microcontroller memory. The number of bytes received is available in RXBYTECNT field. Bank 1 FIFO values are read through UDP\_FDRx. Once a transfer is done, the device firmware must release Bank 1 to the USB device by clearing RX\_DATA\_BK1.

After setting or clearing this bit, a wait time of 3 UDPOCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **DIR: Transfer Direction (only available for control endpoints) (Read/Write)**

0: Allows Data OUT transactions in the control data stage.

1: Enables Data IN transactions in the control data stage.

Refer to Chapter 8.5.3 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the control data stage.

This bit must be set before UDP\_CSRx/RXSETUP is cleared at the end of the setup stage. According to the request sent in the setup data packet, the data stage is either a device to host (DIR = 1) or host to device (DIR = 0) data transfer. It is not necessary to check this bit to reverse direction for the status stage.

- **EPTYPE[2:0]: Endpoint Type (Read/Write)**

Value	Name	Description
0	CTRL	Control
1	ISO_OUT	Isochronous OUT
5	ISO_IN	Isochronous IN
2	BULK_OUT	Bulk OUT

- **TRANSFER: Transfer Period**

The TRANSFER field should be configured to 2 to guarantee the optimal transfer time.

- **USEQ: User Sequence Enable**

Value	Name	Description
0	NUM_ORDER	Normal mode: The controller converts channels in a simple numeric order.
1	REG_ORDER	User Sequence mode: The sequence respects what is defined in AFEC_SEQ1R and AFEC_SEQ1R.