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	I²C, IrDA, Memory Card, SPI, SSC, UART/USART, USB
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	34
Program Memory Size	256KB (256K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	64K x 8
Voltage - Supply (Vcc/Vdd)	1.62V ~ 3.6V
Data Converters	A/D 8x12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	48-VFQFN Exposed Pad
Supplier Device Package	48-QFN (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atsam4s4ab-mnr

Figure 2-5.   SAM4S4/S2 100-pin Version Block Diagram
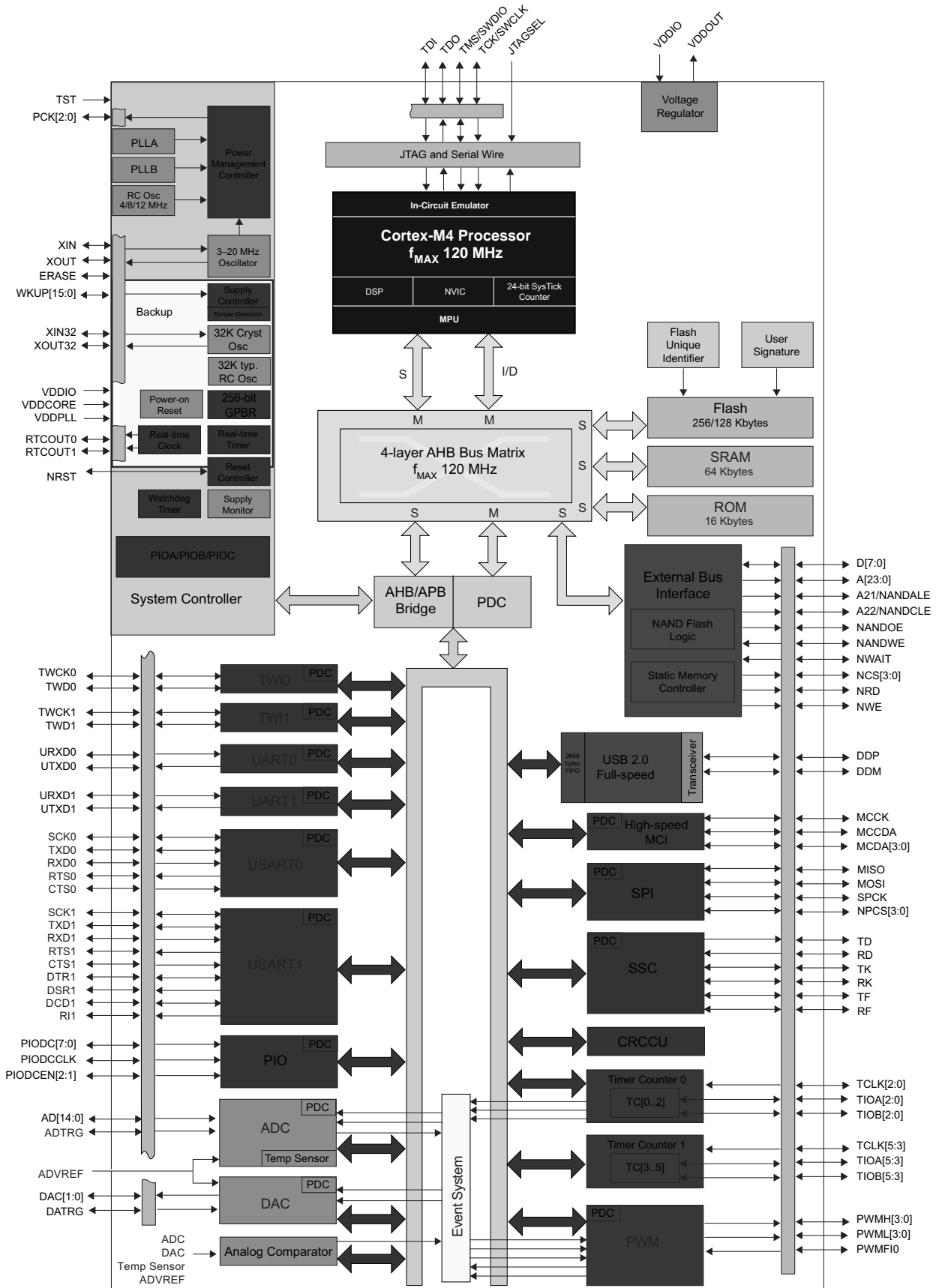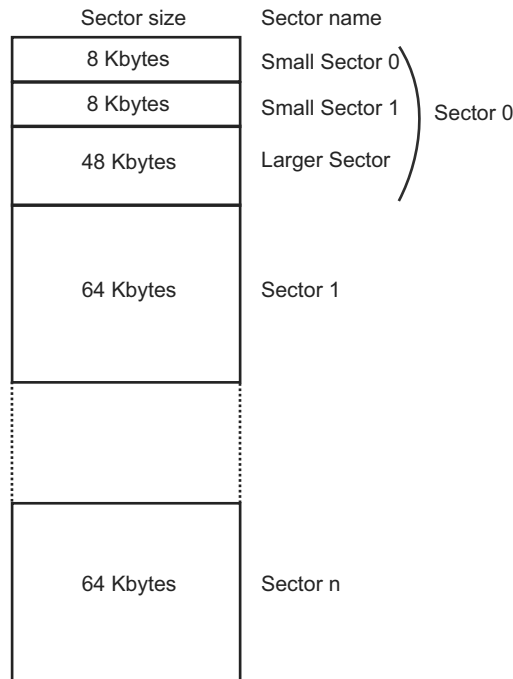
Atmel

Figure 8-1.   Global Flash Organization

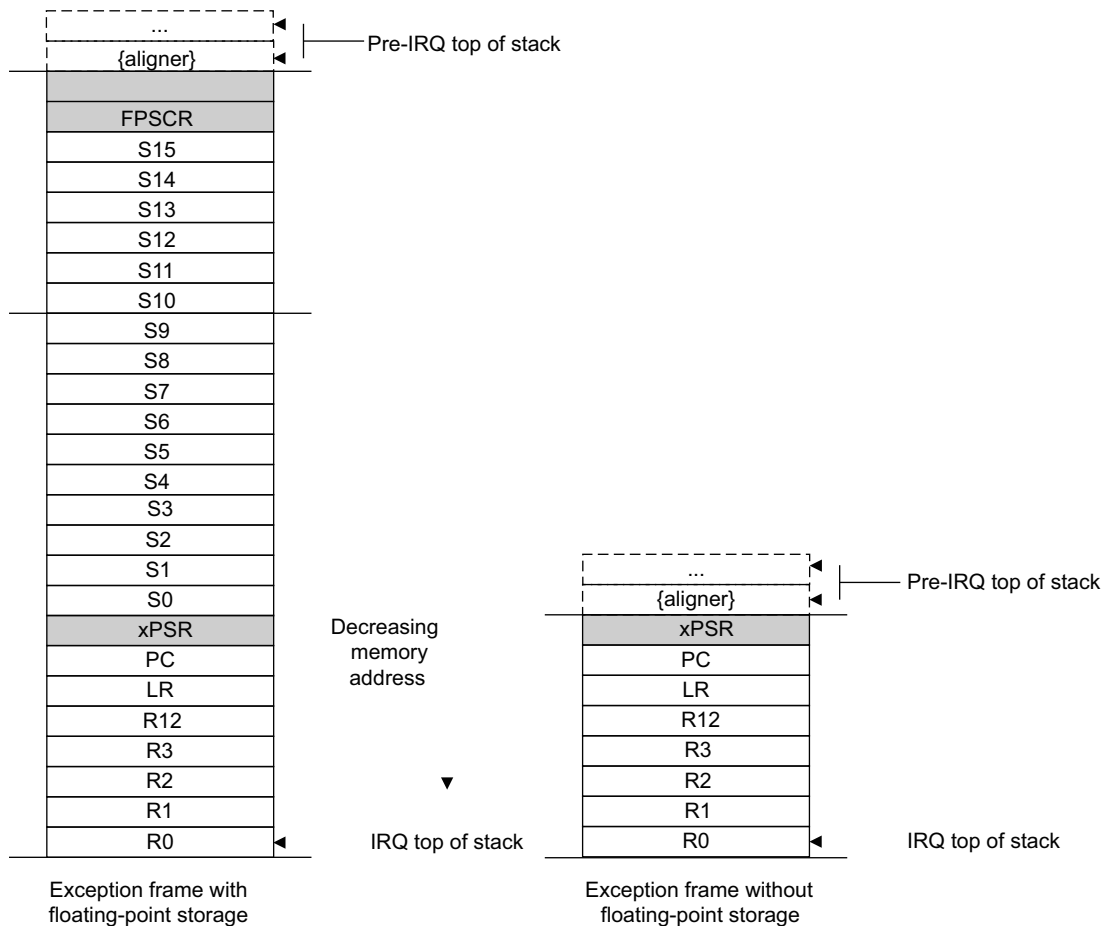Each sector is organized in pages of 512 bytes.

For sector 0:

- The smaller sector 0 has 16 pages of 512 bytes
- The smaller sector 1 has 16 pages of 512 bytes
- The larger sector has 96 pages of 512 bytes

From Sector 1 to n:

The rest of the array is composed of 64-Kbyte sectors of 128 pages, each page of 512 bytes. Refer to Figure 8-2, "Flash Sector Organization".

Atmel

Figure 12-7. Exception Stack Frame

Exception frame with floating-point storage

Exception frame without floating-point storage

Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. The alignment of the stack frame is controlled via the STKALIGN bit of the Configuration Control Register (CCR).

The stack frame includes the return address. This is the address of the next instruction in the interrupted program. This value is restored to the PC at exception return so that the interrupted program resumes.

In parallel to the stacking operation, the processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC_RETURN value to the LR. This indicates which stack pointer corresponds to the stack frame and what operation mode the processor was in before the entry occurred.

If no higher priority exception occurs during the exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher priority exception occurs during the exception entry, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception. This is the late arrival case.

Exception Return

An Exception return occurs when the processor is in Handler mode and executes one of the following instructions to load the EXC_RETURN value into the PC:

- An LDM or POP instruction that loads the PC
- An LDR instruction with the PC as the destination.
- A BX instruction using any register.

Atmel

12.6.5.10 SHADD16 and SHADD8

Signed Halving Add 16 and Signed Halving Add 8

Syntax

    op{cond}{Rd,} Rn, Rm

where:

op   is any of:

            SHADD16 Signed Halving Add 16.

            SHADD8 Signed Halving Add 8.

cond   is an optional condition code, see "Conditional Execution" .

Rd   is the destination register.

Rn   is the first operand register.

Rm   is the second operand register.

Operation

Use these instructions to add 16-bit and 8-bit data and then to halve the result before writing the result to the destination register:

The SHADD16 instruction:

1. Adds each halfword from the first operand to the corresponding halfword of the second operand.
2. Shuffles the result by one bit to the right, halving the data.
3. Writes the halfword results in the destination register.

The SHADDB8 instruction:

1. Adds each byte of the first operand to the corresponding byte of the second operand.
2. Shuffles the result by one bit to the right, halving the data.
3. Writes the byte results in the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

    SHADD16 R1, R0   ; Adds halfwords in R0 to corresponding halfword of R1
                        ; and writes halved result to corresponding halfword in
                        ; R1
    SHADD8 R4, R0, R5 ; Adds bytes of R0 to corresponding byte in R5 and
                        ; writes halved result to corresponding byte in R4.

Atmel

12.6.6.12 SDIV and UDIV

Signed Divide and Unsigned Divide.

Syntax

```
SDIV{cond} {Rd,} Rn, Rm
UDIV{cond} {Rd,} Rn, Rm
```

where:

cond   is an optional condition code, see "Conditional Execution" .

Rd   is the destination register. If *Rd* is omitted, the destination register is *Rn*.

Rn   is the register holding the value to be divided.

Rm   is a register holding the divisor.

Operation

SDIV performs a signed integer division of the value in *Rn* by the value in *Rm*.

UDIV performs an unsigned integer division of the value in *Rn* by the value in *Rm*.

For both instructions, if the value in *Rn* is not divisible by the value in *Rm*, the result is rounded towards zero.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SDIV  R0, R2, R4  ; Signed divide, R0 = R2/R4
UDIV  R8, R8, R1  ; Unsigned divide, R8 = R8/R1
```

12.6.7.2 SSAT16 and USAT16

Signed Saturate and Unsigned Saturate to any bit position for two halfwords.

Syntax

```
op{cond} Rd, #n, Rm
```

where:

op	is one of:
	SSAT16 Saturates a signed halfword value to a signed range.
	USAT16 Saturates a signed halfword value to an unsigned range.
cond	is an optional condition code, see "Conditional Execution" .
Rd	is the destination register.
n	specifies the bit position to saturate to:
n ranges from 1 to 16 for SSAT	n ranges from 0 to 15 for USAT.
Rm	is the register containing the value to saturate.

Operation

The SSAT16 instruction:

Saturates two signed 16-bit halfword values of the register with the value to saturate from selected by the bit position in *n*.

Writes the results as two signed 16-bit halfwords to the destination register.

The USAT16 instruction:

Saturates two unsigned 16-bit halfword values of the register with the value to saturate from selected by the bit position in *n*.

Writes the results as two unsigned halfwords in the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

If saturation occurs, these instructions set the Q flag to 1.

Examples

```
SSAT16    R7, #9, R2    ; Saturates the top and bottom highwords of R2
                        ; as 9-bit values, writes to corresponding halfword
                        ; of R7
USAT16NE  R0, #13, R5   ; Conditionally saturates the top and bottom
                        ; halfwords of R5 as 13-bit values, writes to
                        ; corresponding halfword of R0.
```

12.6.7.3 QADD and QSUB

Saturating Add and Saturating Subtract, signed.

Syntax

```
op{cond} {Rd}, Rn, Rm
op{cond} {Rd}, Rn, Rm
```

where:

Atmel

12.6.10 Branch and Control Instructions

The table below shows the branch and control instructions.

Table 12-25.   Branch and Control Instructions

Mnemonic	Description
B	Branch
BL	Branch with Link
BLX	Branch indirect with Link
BX	Branch indirect
CBNZ	Compare and Branch if Non Zero
CBZ	Compare and Branch if Zero
IT	If-Then
TBB	Table Branch Byte
TBH	Table Branch Halfword

12.6.10.2   CBZ and CBNZ

Compare and Branch on Zero, Compare and Branch on Non-Zero.

Syntax

```
CBZ Rn, label
CBNZ Rn, label
```

where:

Rn   is the register holding the operand.

label   is the branch destination.

Operation

Use the CBZ or CBNZ instructions to avoid changing the condition code flags and to reduce the number of instructions.

CBZ Rn, label does not change condition flags but is otherwise equivalent to:

```
CMP     Rn, #0
BEQ     label
```

CBNZ Rn, label does not change condition flags but is otherwise equivalent to:

```
CMP     Rn, #0
BNE     label
```

Restrictions

The restrictions are:

- ●   *Rn* must be in the range of R0 to R7
- ●   The branch destination must be within 4 to 130 bytes after the instruction
- ●   These instructions must not be used inside an IT block.

Condition Flags

These instructions do not change the flags.

Examples

```
CBZ    R5, target  ; Forward branch if R5 is zero
CBNZ   R0, target  ; Forward branch if R0 is not zero
```

Atmel

Exception

An event that interrupts program execution. When an exception occurs, the processor suspends the normal program flow and starts execution at the address indicated by the corresponding exception vector. The indicated address contains the first instruction of the handler for the exception.

An exception can be an interrupt request, a fault, or a software-generated system exception. Faults include attempting an invalid memory access, attempting to execute an instruction in an invalid processor state, and attempting to execute an undefined instruction.

Exception service routine

See "Interrupt handler" .

Exception vector

See "Interrupt vector" .

Flat address mapping

A system of organizing memory in which each physical address in the memory space is the same as the corresponding virtual address.

Halfword

A 16-bit data item.

Illegal instruction

An instruction that is architecturally Undefined.

Implementation-defined

The behavior is not architecturally defined, but is defined and documented by individual implementations.

Implementation-specific

The behavior is not architecturally defined, and does not have to be documented by individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.

Index register

In some load and store instruction descriptions, the value of this register is used as an offset to be added to or subtracted from the base register value to form the address that is sent to memory. Some addressing modes optionally enable the index register value to be shifted prior to the addition or subtraction.

See also "Base register" .

Instruction cycle count

The number of cycles that an instruction occupies the Execute stage of the pipeline.

Interrupt handler

A program that control of the processor is passed to when an interrupt occurs.

Interrupt vector

One of a number of fixed addresses in low memory, or in high memory if high vectors are configured, that contains the first instruction of the corresponding interrupt handler.

Atmel

The glitch filters are controlled by the Input Filter Enable Register (PIO_IFER), the Input Filter Disable Register (PIO_IFDR) and the Input Filter Status Register (PIO_IFSR). Writing PIO_IFER and PIO_IFDR respectively sets and clears bits in PIO_IFSR. This last register enables the glitch filter on the I/O lines.

When the glitch and/or debouncing filter is enabled, it does not modify the behavior of the inputs on the peripherals. It acts only on the value read in PIO_PDSR and on the input change interrupt detection. The glitch and debouncing filters require that the peripheral clock is enabled.
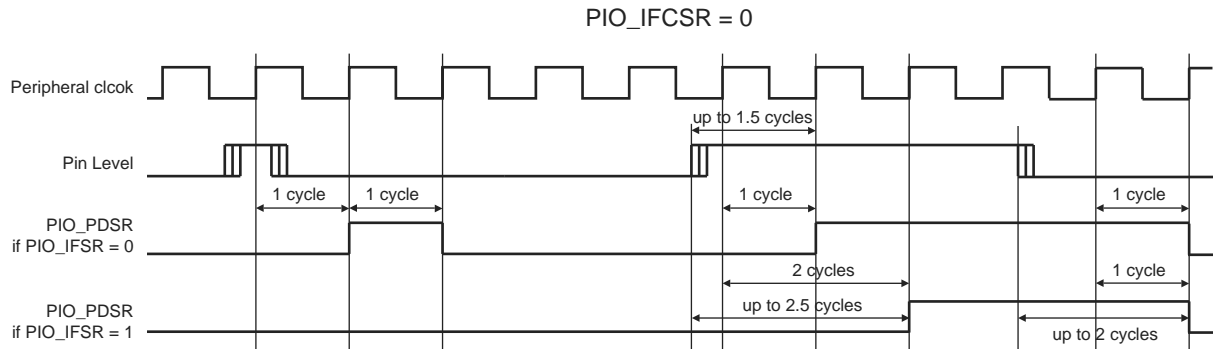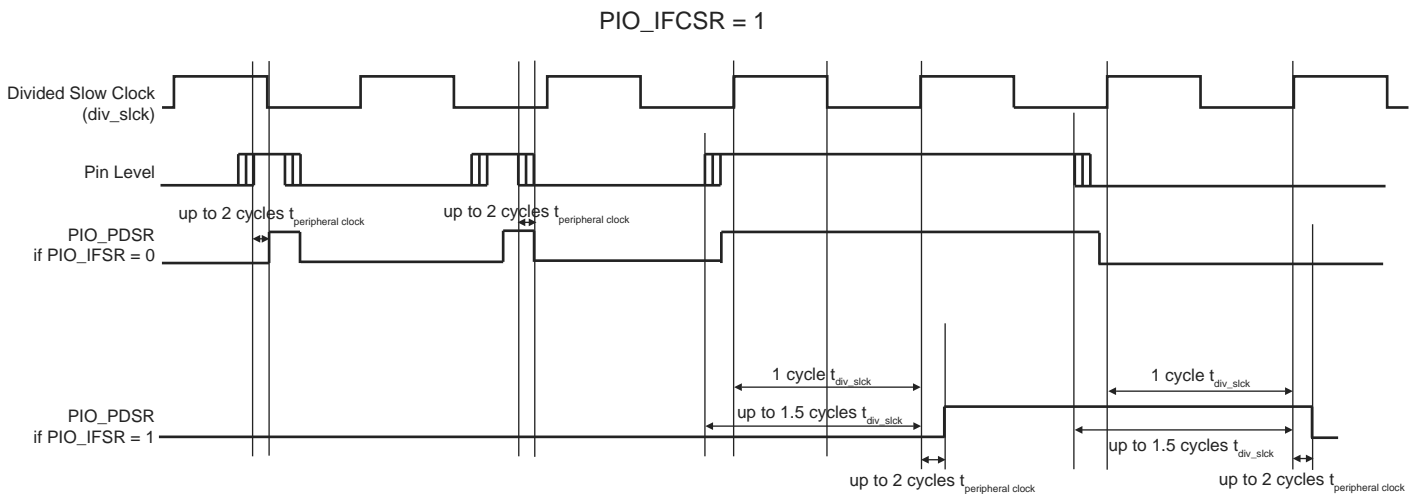
Figure 31-4. Input Glitch Filter Timing

Figure 31-5. Input Debouncing Filter Timing

31.5.10 Input Edge/Level Interrupt

The PIO Controller can be programmed to generate an interrupt when it detects an edge or a level on an I/O line. The Input Edge/Level interrupt is controlled by writing the Interrupt Enable Register (PIO_IER) and the Interrupt Disable Register (PIO_IDR), which enable and disable the input change interrupt respectively by setting and clearing the corresponding bit in the Interrupt Mask Register (PIO_IMR). As input change detection is possible only by comparing two successive samplings of the input of the I/O line, the peripheral clock must be enabled. The Input Change interrupt is available regardless of the configuration of the I/O line, i.e., configured as an input only, controlled by the PIO Controller or assigned to a peripheral function.

By default, the interrupt can be generated at any time an edge is detected on the input.

Some additional interrupt modes can be enabled/disabled by writing in the Additional Interrupt Modes Enable Register (PIO_AIMER) and Additional Interrupt Modes Disable Register (PIO_AIMDR). The current state of this selection can be read through the Additional Interrupt Modes Mask Register (PIO_AIMMR).

These additional modes are:

Figure 34-17. TWI Read Operation with Single Data Byte without Internal Address

BEGIN

Set TWI clock
(CLDIV, CHDIV, CKDIV) in TWI_CWGR
(Needed only once)

Set the Control register:
- Master enable
TWI_CR = MSEN + SVDIS

Set the Master Mode register:
- Device slave address
- Transfer direction bit
Read ==> bit MREAD = 1

Start the transfer
TWI_CR = START | STOP

Read status register

RXRDY = 1?   No

Yes

Read Receive Holding Register

Read Status register

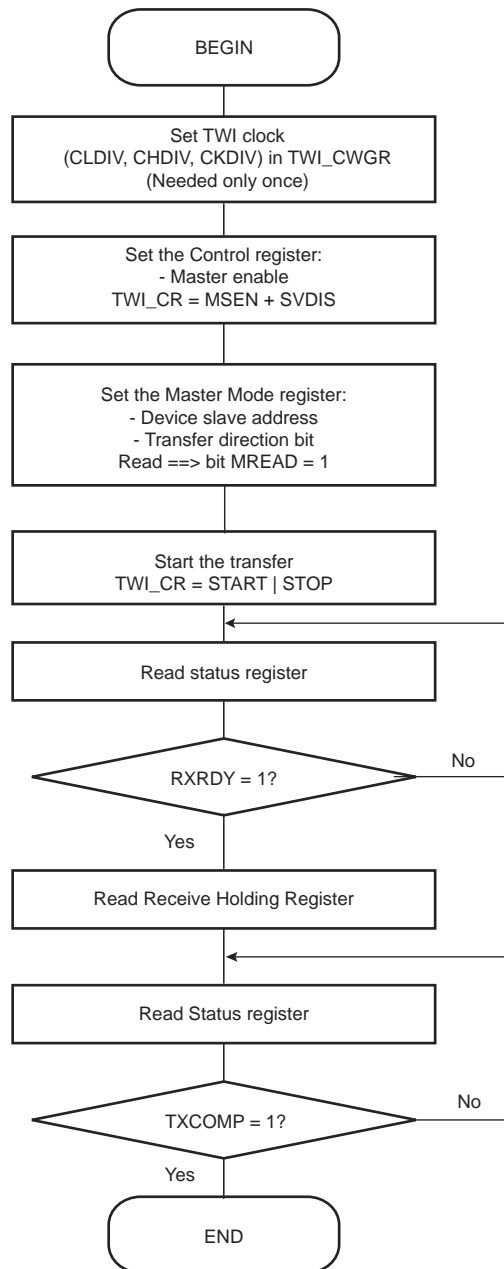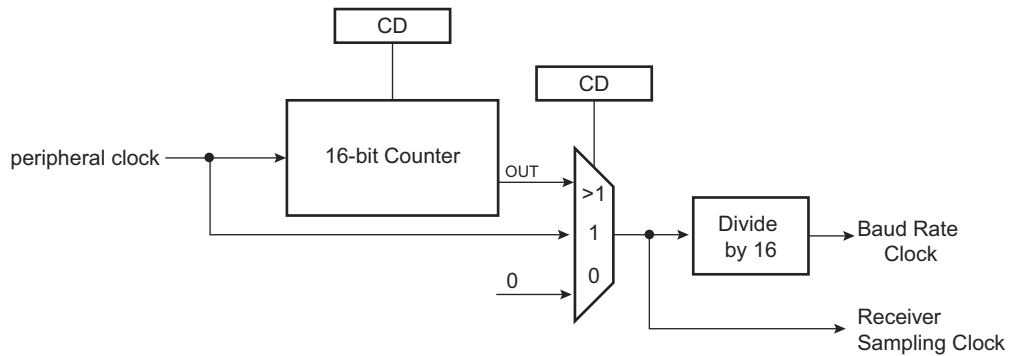TXCOMP = 1?   No

Yes

END

Atmel

Figure 35-2.   Baud Rate Generator

35.5.2 Receiver

35.5.2.1   Receiver Reset, Enable and Disable

After device reset, the UART receiver is disabled and must be enabled before being used. The receiver can be enabled by writing the Control Register (UART_CR) with the bit RXEN at 1. At this command, the receiver starts looking for a start bit.

The programmer can disable the receiver by writing UART_CR with the bit RXDIS at 1. If the receiver is waiting for a start bit, it is immediately stopped. However, if the receiver has already detected a start bit and is receiving the data, it waits for the stop bit before actually stopping its operation.

The receiver can be put in reset state by writing UART_CR with the bit RSTRX at 1. In this case, the receiver immediately stops its current operations and is disabled, whatever its current state. If RSTRX is applied when data is being processed, this data is lost.

35.5.2.2   Start Detection and Data Sampling

The UART only supports asynchronous operations, and this affects only its receiver. The UART receiver detects the start of a received character by sampling the URXD signal until it detects a valid start bit. A low level (space) on URXD is interpreted as a valid start bit if it is detected for more than seven cycles of the sampling clock, which is 16 times the baud rate. Hence, a space that is longer than 7/16 of the bit period is detected as a valid start bit. A space which is 7/16 of a bit period or shorter is ignored and the receiver continues to wait for a valid start bit.

When a valid start bit has been detected, the receiver samples the URXD at the theoretical midpoint of each bit. It is assumed that each bit lasts 16 cycles of the sampling clock (1-bit period) so the bit sampling point is eight cycles (0.5-bit period) after the start of the bit. The first sampling point is therefore 24 cycles (1.5-bit periods) after detecting the falling edge of the start bit.

Each subsequent bit is sampled 16 cycles (1-bit period) after the previous one.

Figure 35-3.   Start Bit Detection
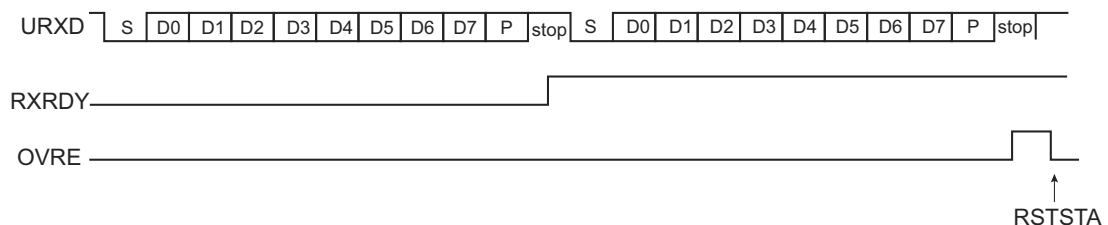
The Local loopback mode allows the transmitted characters to be received. UTXD and URXD pins are not used and the output of the transmitter is internally connected to the input of the receiver. The URXD pin level has no effect and the UTXD line is held high, as in idle state.

The Remote loopback mode directly connects the URXD pin to the UTXD line. The transmitter and the receiver are disabled and have no effect. This mode allows a bit-by-bit retransmission.

Figure 35-11. Test Modes

Atmel
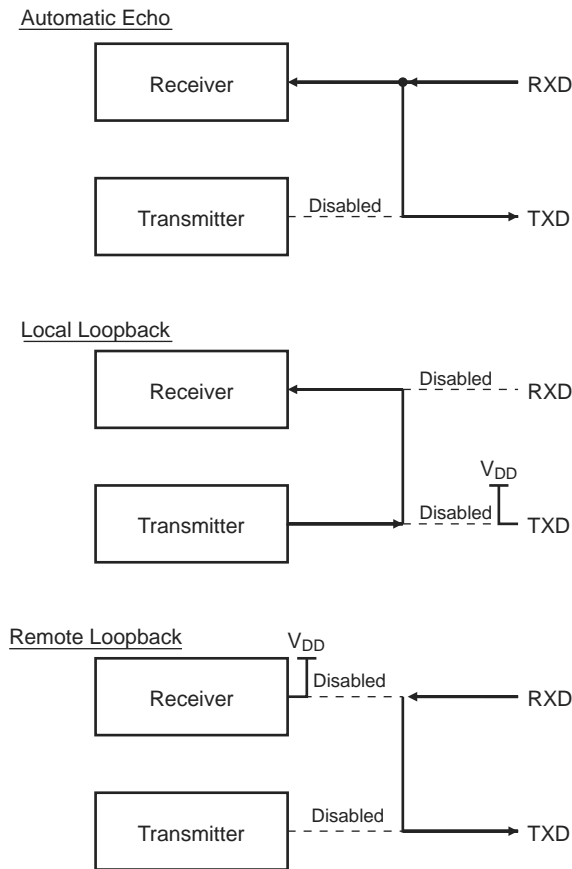
36.7.9 USART Interrupt Mask Register

Name: US_IMR

Address: 0x40024010 (0), 0x40028010 (1)

Access: Read-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 36.7.10 "USART Interrupt Mask Register (SPI_MODE)".

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**

- **RXBRK: Receiver Break Interrupt Mask**

- **ENDRX: End of Receive Buffer Interrupt Mask (available in all USART modes of operation)**

- **ENDTX: End of Transmit Buffer Interrupt Mask (available in all USART modes of operation)**

- **OVRE: Overrun Error Interrupt Mask**

- **FRAME: Framing Error Interrupt Mask**

- **PARE: Parity Error Interrupt Mask**

- **TIMEOUT: Time-out Interrupt Mask**

- **TXEMPTY: TXEMPTY Interrupt Mask**

- **ITER: Max Number of Repetitions Reached Interrupt Mask**

- **TXBUFE: Transmit Buffer Empty Interrupt Mask (available in all USART modes of operation)**

- **RXBUFF: Receive Buffer Full Interrupt Mask (available in all USART modes of operation)**

- **NACK: Non Acknowledge Interrupt Mask**

- **RIIC: Ring Indicator Input Change Mask**

Table 37-2.   Signal Description

Block/Channel	Signal Name	Description
Channel Signal	XC0, XC1, XC2	External Clock Inputs
	TIOA	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Output
	TIOB	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Input/Output
	INT	Interrupt Signal Output (internal signal)
	SYNC	Synchronization Input Signal (from configuration register)

37.4   Pin List

Table 37-3.   Pin List

Pin Name	Description	Type
TCLK0–TCLK2	External Clock Input	Input
TIOA0–TIOA2	I/O Line A	I/O
TIOB0–TIOB2	I/O Line B	I/O

37.5   Product Dependencies

37.5.1   I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the TC pins to their peripheral functions.

Table 37-4.   I/O Lines

Instance	Signal	I/O Line	Peripheral
TC0	TCLK0	PA4	B
TC0	TCLK1	PA28	B
TC0	TCLK2	PA29	B
TC0	TIOA0	PA0	B
TC0	TIOA1	PA15	B
TC0	TIOA2	PA26	B
TC0	TIOB0	PA1	B
TC0	TIOB1	PA16	B
TC0	TIOB2	PA27	B
TC1	TCLK3	PC25	B
TC1	TCLK4	PC28	B
TC1	TCLK5	PC31	B
TC1	TIOA3	PC23	B
TC1	TIOA4	PC26	B
TC1	TIOA5	PC29	B

The command ALL_SEND_CID and the fields and values for the HSMCI_CMDR are described in Table 38-6 and Table 38-7.

Table 38-6.   ALL_SEND_CID Command Description

CMD Index	Type	Argument	Response	Abbreviation	Command Description
CMD2	bcr[(1)]	[31:0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the CMD line

Note: 1.   bcr means broadcast command with response.

Table 38-7.   Fields and Values for HSMCI_CMDR

Field	Value
CMDNB (command number)	2 (CMD2)
RSPTYP (response type)	2 (R2: 136 bits response)
SPCMD (special command)	0 (not a special command)
OPCMD (open drain command)	1
MAXLAT (max latency for command to response)	0 (NID cycles ==> 5 cycles)
TRCMD (transfer command)	0 (No transfer)
TRDIR (transfer direction)	X (available only in transfer command)
TRTYP (transfer type)	X (available only in transfer command)
IOSPCMD (SDIO special command)	0 (not a special command)

The HSMCI_ARGR contains the argument field of the command.

To send a command, the user must perform the following steps:

●   Fill the argument register (HSMCI_ARGR) with the command argument.
●   Set the command register (HSMCI_CMDR) (see Table 38-7).

The command is sent immediately after writing the command register.

While the card maintains a busy indication (at the end of a STOP_TRANSMISSION command CMD12, for example), a new command shall not be sent. The NOTBUSY flag in the Status Register (HSMCI_SR) is asserted when the card releases the busy indication.

If the command requires a response, it can be read in the HSMCI Response Register (HSMCI_RSPR). The response size can be from 48 bits up to 136 bits depending on the command. The HSMCI embeds an error detection to prevent any corrupted data during the transfer.

The following flowchart shows how to send a command to the card and read the response if needed. In this example, the status register bits are polled but setting the appropriate bits in the HSMCI Interrupt Enable Register (HSMCI_IER) allows using an interrupt method.

Atmel

10. The update of these registers will occur at the next PWM period of the synchronous channels when the Update Period is elapsed. Go to Step 8. for new values.

Figure 39-11. Method 2 (UPDM = 1)
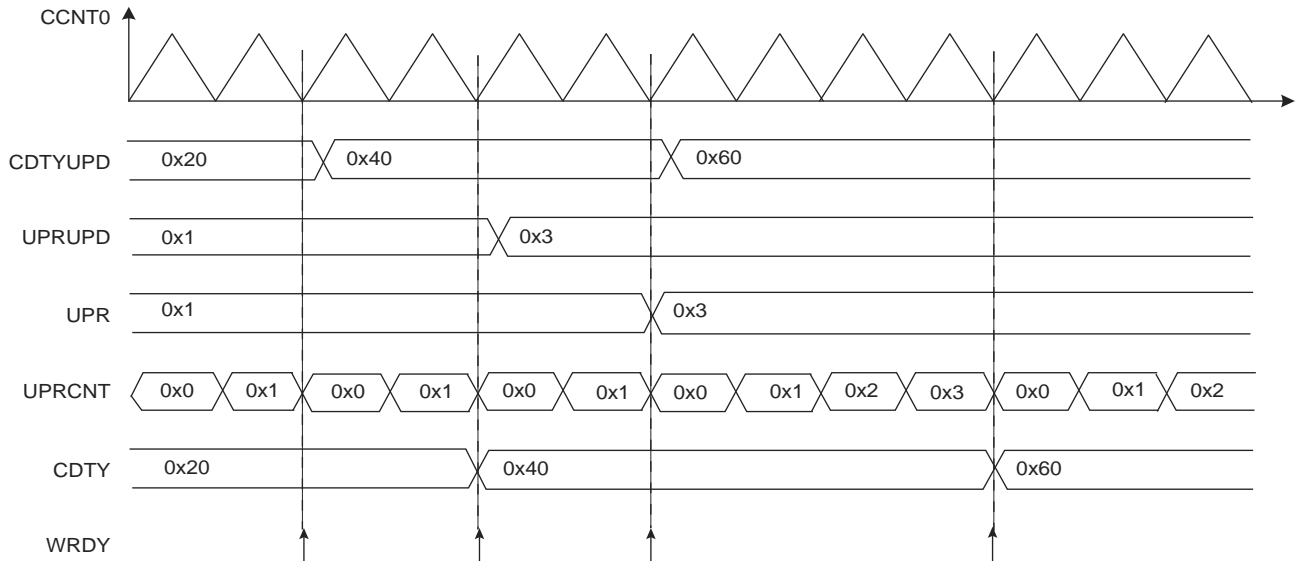
Method 3: Automatic write of duty-cycle values and automatic trigger of the update

In this mode, the update of the duty cycle values is made automatically by the Peripheral DMA Controller. The update of the period value, the dead-time values and the update period value must be done by writing in their respective update registers with the processor (respectively PWM_CPRDUPDx, PWM_DTUPDx and PWM_SCUPUPD).

To trigger the update of the period value and the dead-time values, the user must use the bit UPDULOCK which allows to update synchronously (at the same PWM period) the synchronous channels:

- If the bit UPDULOCK is set to '1', the update is done at the next PWM period of the synchronous channels.
- If the UPDULOCK bit is not set to '1', the update is locked and cannot be performed.

After writing the UPDULOCK bit to '1', it is held at this value until the update occurs, then it is read 0.

The update of the duty-cycle values and the update period value is triggered automatically after an update period.

To configure the automatic update, the user must define a value for the Update Period by the field UPR in the PWM_SCUP register. The PWM controller waits UPR+1 periods of synchronous channels before updating automatically the duty values and the update period value.

Using the Peripheral DMA Controller removes processor overhead by reducing its intervention during the transfer. This significantly reduces the number of clock cycles required for a data transfer, which improves microcontroller performance.

The Peripheral DMA Controller must write the duty-cycle values in the synchronous channels index order. For example if the channels 0, 1 and 3 are synchronous channels, the Peripheral DMA Controller must write the duty-cycle of the channel 0 first, then the duty-cycle of the channel 1, and finally the duty-cycle of the channel 3.

The status of the Peripheral DMA Controller transfer is reported in the PWM_ISR2 by the following flags:

- WRDY: this flag is set to '1' when the PWM Controller is ready to receive new duty-cycle values and a new update period value. It is reset to '0' when the PWM_ISR2 is read. The user can choose to synchronize the WRDY flag and the Peripheral DMA Controller transfer request with a comparison match (see Section 39.6.3 "PWM Comparison Units"), by the fields PTRM and PTRCS in the PWM_SCM register.
- ENDTX : this flag is set to '1' when a PDC transfer is completed
- TXBUFE : this flag is set to '1' when the PDC buffer is empty (no pending PDC transfers)

40.6.1.3   USB Transfer Event Definitions

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

Table 40-5.   USB Transfer Events

Transfer		Transaction
Direction	**Type**	**Transaction**
CONTROL (bidirectional)	Control[(1)(3)]	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[(2)]	
	Bulk IN	
OUT (host toward device)	Interrupt OUT	Data OUT transaction → Data OUT transaction
	Isochronous OUT[(2)]	
	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 40-4.   Control Read and Write Sequences
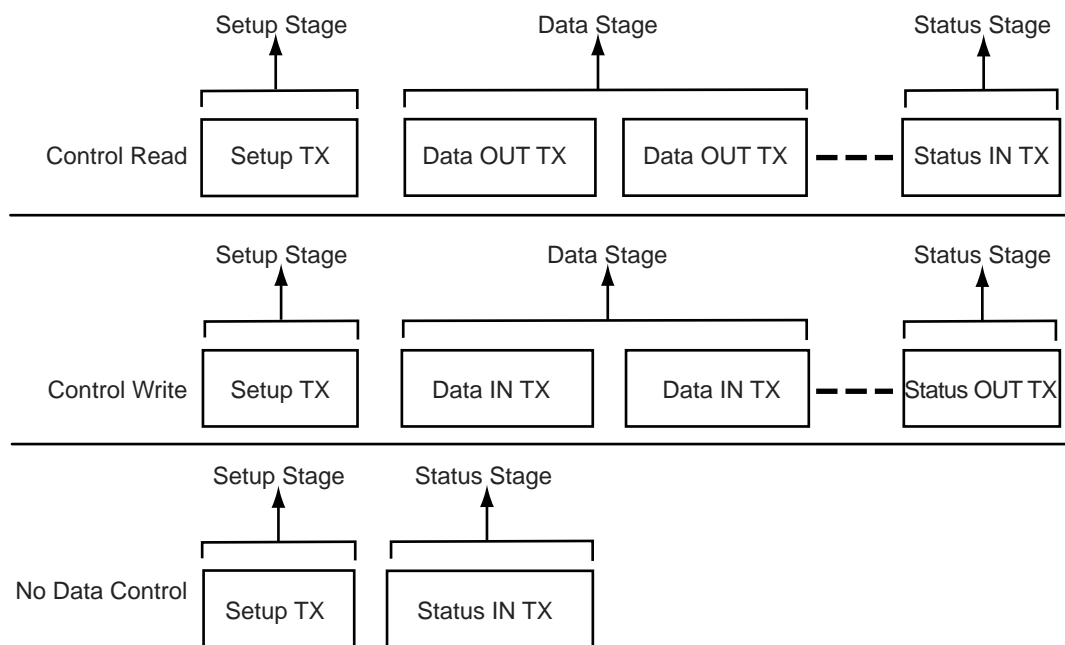
Notes: 1.   During the Status IN stage, the host waits for a zero length packet (Data IN transaction with no data) from the device using DATA1 PID. Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0,* for more information on the protocol layer.

41.7.6 ACC Interrupt Status Register

Name: ACC_ISR

Address: 0x40040030

Access: Read-only

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

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	SCO	CE

- **CE: Comparison Edge (cleared on read)**

0: No edge occurred (defined by EDGETYP) on analog comparator output since the last read of ACC_ISR.

1: A selected edge (defined by EDGETYP) on analog comparator output occurred since the last read of ACC_ISR.

- **SCO: Synchronized Comparator Output**

Returns an image of the analog comparator output after being pre-processed (refer to Figure 41-1 on page 1069).

    If INV = 0

        SCO = 0 if inn > inp

        SCO = 1 if inp > inn

    If INV = 1

        SCO = 1 if inn > inp

        SCO = 0 if inp > inn

- **MASK: Flag Mask**

0: The CE flag and SCO value are valid.

1: The CE flag and SCO value are invalid.

Atmel

45.6 64-ball WLCSP Mechanical Characteristics

This package respects the recommendations of the NEMI User Group.

Figure 45-6. 64-ball WLCSP Package Mechanical Drawing (SAM4SD32/SAM4SD16)