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®-M3
Core Size	32-Bit Single-Core
Speed	96MHz
Connectivity	EBI/EMI, I²C, Memory Card, SPI, SSC, UART/USART, USB
Peripherals	Brown-out Detect/Reset, DMA, I²S, POR, PWM, WDT
Number of I/O	57
Program Memory Size	64KB (64K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	20K x 8
Voltage - Supply (Vcc/Vdd)	1.62V ~ 3.6V
Data Converters	A/D 4x10b, 4x12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	100-LQFP
Supplier Device Package	100-LQFP (14x14)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atsam3u1cb-au

12.5.1.3 Active

An exception that is being serviced by the processor but has not completed.

An exception handler can interrupt the execution of another exception handler. In this case both exceptions are in the active state.

12.5.1.4 Active and pending

The exception is being serviced by the processor and there is a pending exception from the same source.

12.5.2 Exception types

The exception types are:

12.5.2.1 Reset

Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in Thread mode.

12.5.2.2 Non Maskable Interrupt (NMI)

A non maskable interrupt (NMI) can be signalled by a peripheral or triggered by software. This is the highest priority exception other than reset. It is permanently enabled and has a fixed priority of -2.

NMIs cannot be:

- Masked or prevented from activation by any other exception.
- Preempted by any exception other than Reset.

12.5.2.3 Hard fault

A hard fault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism. Hard faults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.

12.5.2.4 Memory management fault

A memory management fault is an exception that occurs because of a memory protection related fault. The MPU or the fixed memory protection constraints determines this fault, for both instruction and data memory transactions. This fault is used to abort instruction accesses to *Execute Never* (XN) memory regions, even if the MPU is disabled.

12.5.2.5 Bus fault

A bus fault is an exception that occurs because of a memory related fault for an instruction or data memory transaction. This might be from an error detected on a bus in the memory system.

12.5.2.6 Usage fault

A usage fault is an exception that occurs because of a fault related to instruction execution. This includes:

- an undefined instruction
- an illegal unaligned access
- invalid state on instruction execution
- an error on exception return.

The following can cause a usage fault when the core is configured to report them:

- an unaligned address on word and halfword memory access
- division by zero.

12.11.2 LDR and STR, immediate offset

Load and Store with immediate offset, pre-indexed immediate offset, or post-indexed immediate offset.

12.11.2.1 Syntax

```
op{type}{cond} Rt, [Rn {, #offset}]        ; immediate offset
op{type}{cond} Rt, [Rn, #offset]!          ; pre-indexed
op{type}{cond} Rt, [Rn], #offset           ; post-indexed
opD{cond} Rt, Rt2, [Rn {, #offset}]        ; immediate offset, two words
opD{cond} Rt, Rt2, [Rn, #offset]!          ; pre-indexed, two words
opD{cond} Rt, Rt2, [Rn], #offset           ; post-indexed, two words
```

where:

op is one of:

- LDR Load Register.
- STR Store Register.

type is one of:

- B unsigned byte, zero extend to 32 bits on loads.
- SB signed byte, sign extend to 32 bits (LDR only).
- H unsigned halfword, zero extend to 32 bits on loads.
- SH signed halfword, sign extend to 32 bits (LDR only).
- - omit, for word.

cond is an optional condition code, see “Conditional execution” on page 87.

Rt is the register to load or store.

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

offset is an offset from *Rn*. If *offset* is omitted, the address is the contents of *Rn*.

Rt2 is the additional register to load or store for two-word operations.

12.11.2.2 Operation

LDR instructions load one or two registers with a value from memory.

STR instructions store one or two register values to memory.

Load and store instructions with immediate offset can use the following addressing modes:

12.11.2.3 Offset addressing

The offset value is added to or subtracted from the address obtained from the register *Rn*. The result is used as the address for the memory access. The register *Rn* is unaltered. The assembly language syntax for this mode is:

```
[Rn, #offset]
```

12.11.2.4 Pre-indexed addressing

The offset value is added to or subtracted from the address obtained from the register *Rn*. The result is used as the address for the memory access and written back into the register *Rn*. The assembly language syntax for this mode is:

```
[Rn, #offset]!
```

12.11.2.5 Post-indexed addressing

The address obtained from the register *Rn* is used as the address for the memory access. The offset value is added to or subtracted from the address, and written back into the register *Rn*. The assembly language syntax for this mode is:

```
[Rn], #offset
```

12.11.3.5 Examples

```
STR    R0, [R5, R1]         ; Store value of R0 into an address equal to
                            ; sum of R5 and R1
LDRSB  R0, [R5, R1, LSL #1] ; Read byte value from an address equal to
                            ; sum of R5 and two times R1, sign extended it
                            ; to a word value and put it in R0
STR    R0, [R1, R2, LSL #2] ; Stores R0 to an address equal to sum of R1
                            ; and four times R2
```

12.12.8 REV, REV16, REVSH, and RBIT

Reverse bytes and Reverse bits.

12.12.8.1 Syntax

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

where:

op is any of:

- REV Reverse byte order in a word.
- REV16 Reverse byte order in each halfword independently.
- REVSH Reverse byte order in the bottom halfword, and sign extend to 32 bits.
- RBIT Reverse the bit order in a 32-bit word.

cond is an optional condition code, see “Conditional execution” on page 87.

Rd is the destination register.

Rn is the register holding the operand.

12.12.8.2 Operation

Use these instructions to change endianness of data:

REV converts 32-bit big-endian data into little-endian data or 32-bit little-endian data into big-endian data.

REV16 converts 16-bit big-endian data into little-endian data or 16-bit little-endian data into big-endian data.

REVSH converts either:

- 16-bit signed big-endian data into 32-bit signed little-endian data
- 16-bit signed little-endian data into 32-bit signed big-endian data.

12.12.8.3 Restrictions

Do not use SP and do not use PC.

12.12.8.4 Condition flags

These instructions do not change the flags.

12.12.8.5 Examples

```
REV    R3, R7  ; Reverse byte order of value in R7 and write it to R3
REV16  R0, R0  ; Reverse byte order of each 16-bit halfword in R0
REVSH  R0, R5  ; Reverse Signed Halfword
REVHS  R3, R7  ; Reverse with Higher or Same condition
RBIT   R7, R8  ; Reverse bit order of value in R8 and write the result to R7
```

12.20.9.3 System Handler Priority Register 3

The bit assignments are:

31	30	29	28	27	26	25	24
PRI_15							
23	22	21	20	19	18	17	16
PRI_14							
15	14	13	12	11	10	9	8
Reserved							
7	6	5	4	3	2	1	0
Reserved							

- **PRI_15**

Priority of system handler 15, SysTick exception

- **PRI_14**

Priority of system handler 14, PendSV

12.20.12Hard Fault Status Register

The HFSR gives information about events that activate the hard fault handler. See the register summary in Table 12-30 on page 161 for its attributes.

This register is read, write to clear. This means that bits in the register read normally, but writing 1 to any bit clears that bit to 0. The bit assignments are:

31	30	29	28	27	26	25	24
DEBUGEVT	FORCED	Reserved					

23	22	21	20	19	18	17	16
Reserved							

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
Reserved						VECTTBL	Reserved

- **DEBUGEVT**

Reserved for Debug use. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.

- **FORCED**

Indicates a forced hard fault, generated by escalation of a fault with configurable priority that cannot be handles, either because of priority or because it is disabled:

0 = no forced hard fault

1 = forced hard fault.

When this bit is set to 1, the hard fault handler must read the other fault status registers to find the cause of the fault.

- **VECTTBL**

Indicates a bus fault on a vector table read during exception processing:

0 = no bus fault on vector table read

1 = bus fault on vector table read.

This error is always handled by the hard fault handler.

When this bit is set to 1, the PC value stacked for the exception return points to the instruction that was preempted by the exception.

The HFSR bits are sticky. This means as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

24.16 NAND Flash Controller Operations

24.16.1 NFC Overview

The NFC can handle automatic transfers, sending the commands and address to the NAND Flash and transferring the contents of the page (for read and write) to the NFC SRAM. It minimizes the CPU overhead.

24.16.2 NFC Control Registers

NAND Flash Read and NAND Flash Program operations can be performed through the NFC Command Registers. In order to minimize CPU intervention and latency, commands are posted in a command buffer. This buffer provides zero wait state latency. The detailed description of the command encoding scheme is explained below.

The NFC handles automatic transfer between the external NAND Flash and the chip via the NFC SRAM. It is done via NFC Command Registers.

The NFC Command Registers are very efficient to use. When writing to these registers:

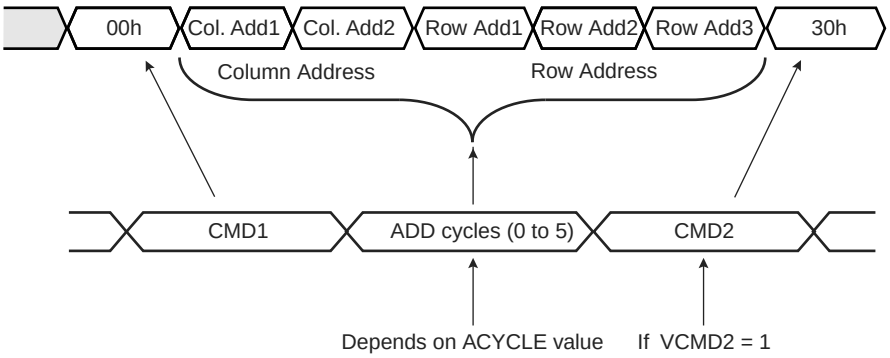
- the address of the register (NFCADDR_CMD) contains the command used,
- the data of the register (NFCDATA_ADDT) contains the address to be sent to the NAND Flash.

So, in one single access the command is sent and immediately executed by the NFC. Even two commands can be programmed within a single access (CMD1, CMD2) depending on the VCMD2 value.

The NFC can send up to 5 Address cycles.

Figure 24-31 below shows a typical NAND Flash Page Read Command of a NAND Flash Memory and correspondence with NFC Address Command Register.

Figure 24-31. NFC/NAND Flash Access Example



For more details refer to “NFC Address Command” on page 374.

The NFC Command Registers can be found at address 0x68000000 - 0x6FFFFFFF. (See Table 24-4, “External Memory Mapping”.)

Reading the NFC command register (to any address) will give the status of the NFC. Especially useful to know if the NFC is busy, for example.

24.18.8 SMC NFC Bank Register

Name: SMC_BANK

Address: 0x400E001C

Access: Read-write

Reset: 0x00000000

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
–	–	–	–	–	BANK		

- **BANK: Bank Identifier**

Number of the bank used

24.18.13 SMC ECC Parity Register 0 for a Page of 512/1024/2048/4096 Bytes

Name: SMC_ECC_PR0

Address: 0x400E002C

Access: Read-only

Reset: 0x00000000

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

Once the entire main area of a page is written with data, the register content must be stored at any free location of the spare area.

- **BITADDR: Bit Address**

During a page read, this value contains the corrupted bit offset where an error occurred, if a single error was detected. If multiple errors were detected, this value is meaningless.

- **WORDADDR: Word Address**

During a page read, this value contains the word address (8-bit or 16-bit word depending on the memory plane organization).where an error occurred, if a single error was detected. If multiple errors were detected, this value is meaningless.

27. Power Management Controller (PMC)

27.1 Description

The Power Management Controller (PMC) optimizes power consumption by controlling all system and user peripheral clocks. The PMC enables/disables the clock inputs to many of the peripherals and the Cortex-M3 Processor.

The Power Management Controller provides the following clocks:

- MCK, the Master Clock, programmable from a few hundred Hz to the maximum operating frequency of the device. It is available to the modules running permanently, such as the Enhanced Embedded Flash Controller.
- Processor Clock (HCLK) is automatically switched off when the processor enters Sleep Mode.
- Free running processor Clock (FCLK)
- the Cortex-M3 SysTick external clock
- the USB Device HS Clock (UDPCK)
- Peripheral Clocks, typically MCK, provided to the embedded peripherals (USART, PMC, SPI, TWI, TC, HSMCI, etc.) and independently controllable. In order to reduce the number of clock names in a product, the Peripheral Clocks are named MCK in the product datasheet.
- Programmable Clock Outputs can be selected from the clocks provided by the clock generator and driven on the PCKx pins.

27.14.19 PMC Fault Output Clear Register

Name: PMC_FOCR

Address: 0x400E0478

Access: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	FOCLR

- **FOCLR: Fault Output Clear**

Clears the clock failure detector fault output.

29.7.16 PIO Controller Interrupt Mask Register

Name: PIO_IMR

Address: 0x400E0C48 (PIOA), 0x400E0E48 (PIOB), 0x400E1048 (PIOC)

Access: Read-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

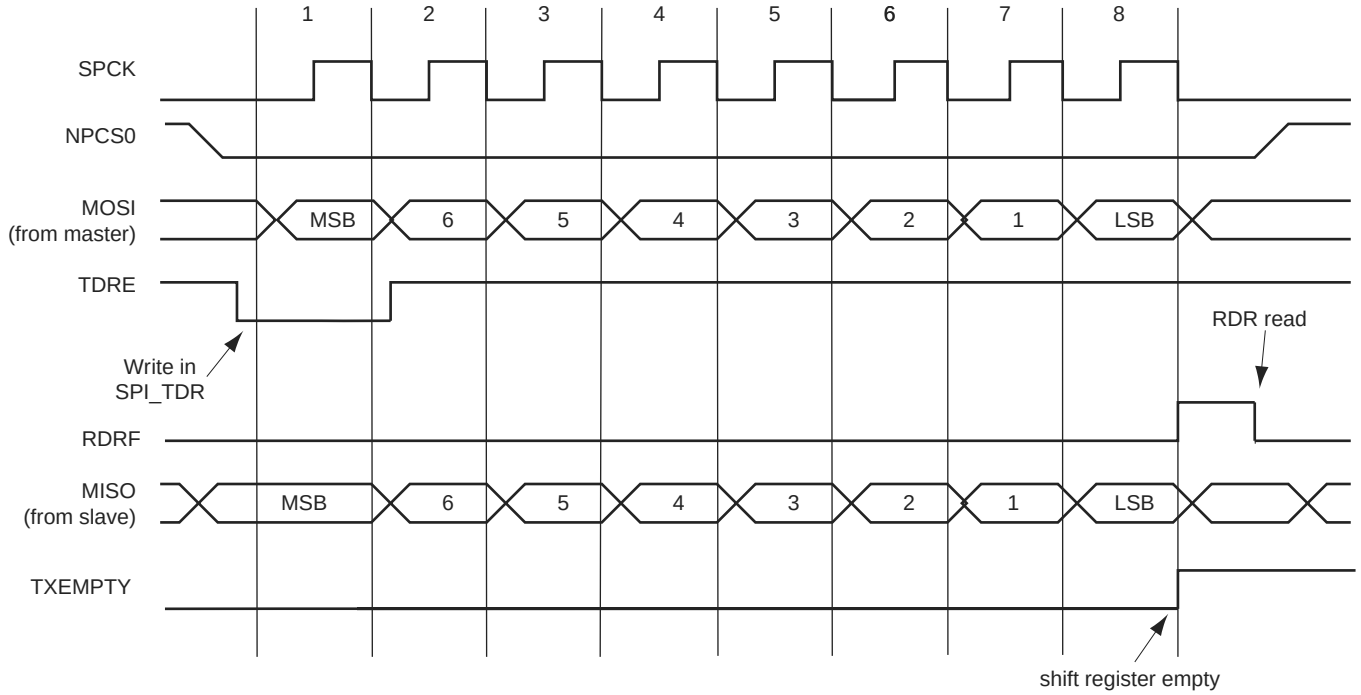
- **P0-P31: Input Change Interrupt Mask**

0 = Input Change Interrupt is disabled on the I/O line.

1 = Input Change Interrupt is enabled on the I/O line.

Figure 31-7 shows Transmit Data Register Empty (TDRE), Receive Data Register (RDRF) and Transmission Register Empty (TXEMPTY) status flags behavior within the SPI_SR (Status Register) during an 8-bit data transfer in fixed mode and no Peripheral Data Controller involved.

Figure 31-7. Status Register Flags Behavior



31.7.3.3 Clock Generation

The SPI Baud rate clock is generated by dividing the Master Clock (MCK), by a value between 1 and 255.

This allows a maximum operating baud rate at up to Master Clock and a minimum operating baud rate of MCK divided by 255.

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

The divisor can be defined independently for each chip select, as it has to be programmed in the SCBR field of the Chip Select Registers. This allows the SPI to automatically adapt the baud rate for each interfaced peripheral without reprogramming.

31.7.3.4 Transfer Delays

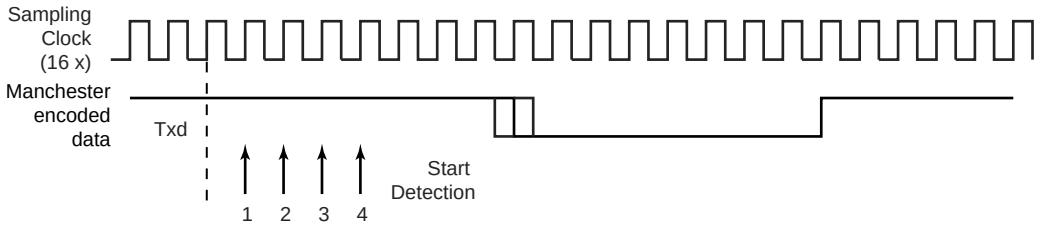
Figure 31-8 shows a chip select transfer change and consecutive transfers on the same chip select. Three delays can be programmed to modify the transfer waveforms:

- The delay between chip selects, programmable only once for all the chip selects by writing the DLYBCS field in the Mode Register. Allows insertion of a delay between release of one chip select and before assertion of a new one.
- The delay before SPCK, independently programmable for each chip select by writing the field DLYBS. Allows the start of SPCK to be delayed after the chip select has been asserted.
- The delay between consecutive transfers, independently programmable for each chip select by writing the DLYBCT field. Allows insertion of a delay between two transfers occurring on the same chip select

These delays allow the SPI to be adapted to the interfaced peripherals and their speed and bus release time.

Unlike preamble, the start frame delimiter is shared between Manchester Encoder and Decoder. So, if ONEBIT field is set to 1, only a zero encoded Manchester can be detected as a valid start frame delimiter. If ONEBIT is set to 0, only a sync pattern is detected as a valid start frame delimiter. Decoder operates by detecting transition on incoming stream. If RXD is sampled during one quarter of a bit time to zero, a start bit is detected. See Figure 34-14. The sample pulse rejection mechanism applies.

Figure 34-14. Asynchronous Start Bit Detection



The receiver is activated and starts Preamble and Frame Delimiter detection, sampling the data at one quarter and then three quarters. If a valid preamble pattern or start frame delimiter is detected, the receiver continues decoding with the same synchronization. If the stream does not match a valid pattern or a valid start frame delimiter, the receiver re-synchronizes on the next valid edge.The minimum time threshold to estimate the bit value is three quarters of a bit time.

If a valid preamble (if used) followed with a valid start frame delimiter is detected, the incoming stream is decoded into NRZ data and passed to USART for processing. Figure 34-15 illustrates Manchester pattern mismatch. When incoming data stream is passed to the USART, the receiver is also able to detect Manchester code violation. A code violation is a lack of transition in the middle of a bit cell. In this case, MANE flag in US_CSR register is raised. It is cleared by writing the Control Register (US_CR) with the RSTSTA bit to 1. See Figure 34-16 for an example of Manchester error detection during data phase.

Figure 34-15. Preamble Pattern Mismatch

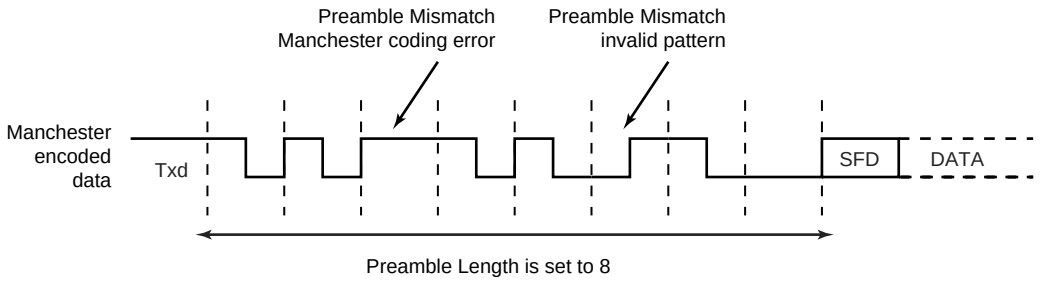


Figure 34-16. Manchester Error Flag

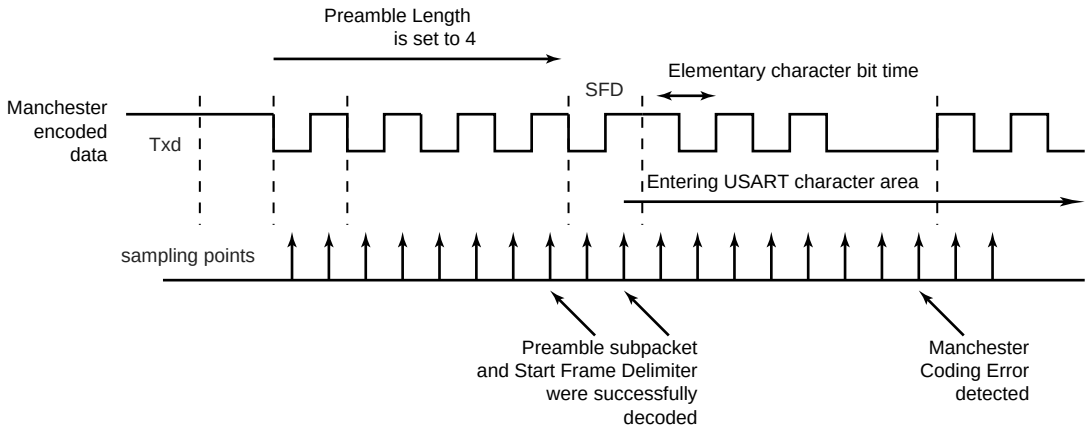
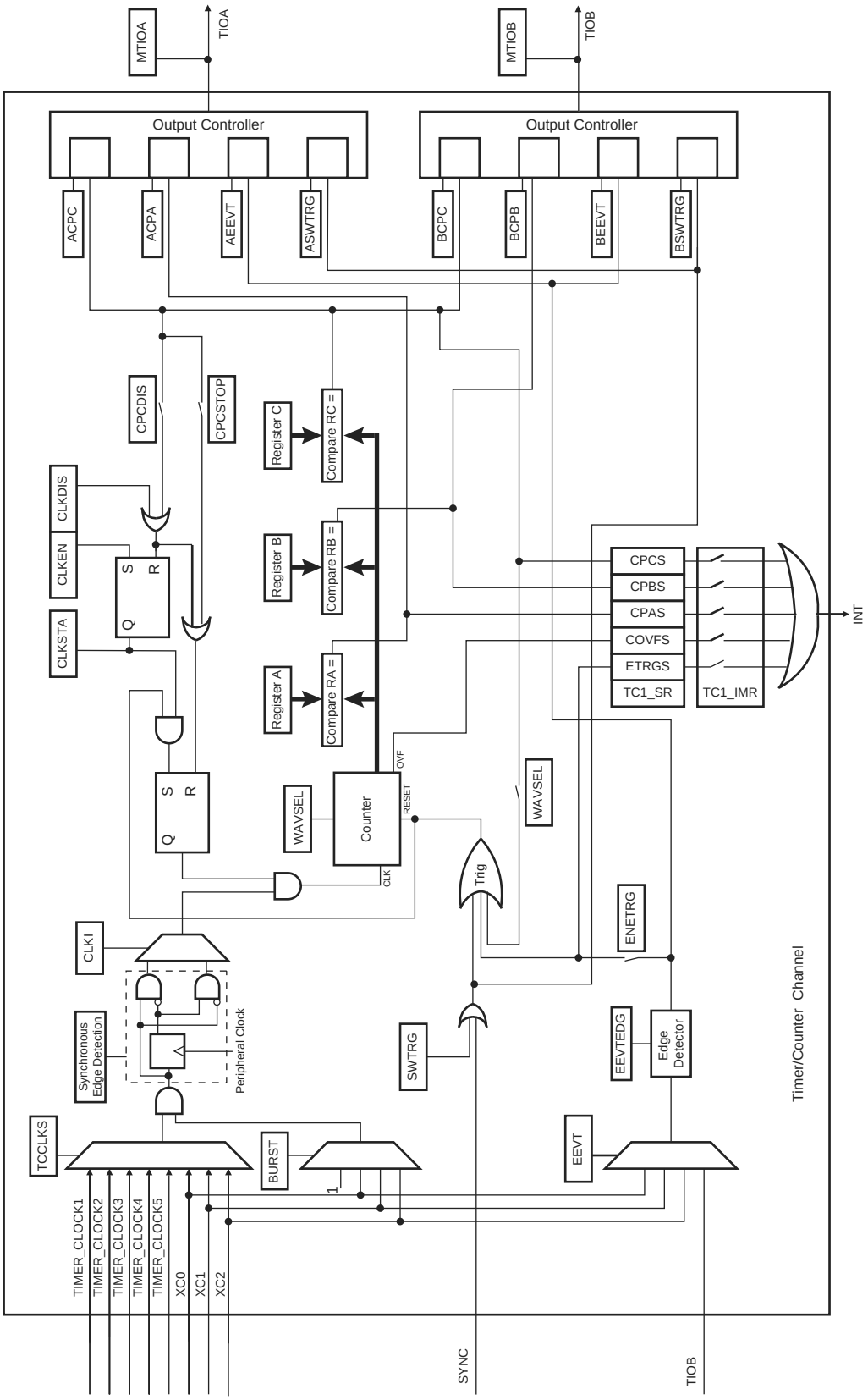


Figure 35-6. Waveform Mode



35.6.12 External Event/Trigger Conditions

An external event can be programmed to be detected on one of the clock sources (XC0, XC1, XC2) or TIOB. The external event selected can then be used as a trigger.

The EEVT parameter in TC_CMR selects the external trigger. The EEVTEDG parameter defines the trigger edge for each of the possible external triggers (rising, falling or both). If EEVTEDG is cleared (none), no external event is defined.

If TIOB is defined as an external event signal (EEVT = 0), TIOB is no longer used as an output and the compare register B is not used to generate waveforms and subsequently no IRQs. In this case the TC channel can only generate a waveform on TIOA.

When an external event is defined, it can be used as a trigger by setting bit ENETRG in the TC_CMR.

As in Capture mode, the SYNC signal and the software trigger are also available as triggers. RC Compare can also be used as a trigger depending on the parameter WAVSEL.

35.6.13 Output Controller

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

The following events control TIOA and TIOB: software trigger, external event and RC compare. RA compare controls TIOA and RB compare controls TIOB. Each of these events can be programmed to set, clear or toggle the output as defined in the corresponding parameter in TC_CMR.

35.6.14 Quadrature Decoder

35.6.14.1 Description

The quadrature decoder (QDEC) is driven by TIOA0, TIOB0, 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 35-15).

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

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.

Field TCCLKS of TC_CMRx must be configured to select XC0 input (i.e., 0x101). Field 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 down-stream 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 the CPCS flag in the TC_SRx.

Method 1: Manual write of duty-cycle values and manual trigger of the update

In this mode, the update of the period value, the duty-cycle values and the dead-time values must be done by writing in their respective update registers with the CPU (respectively PWM_CPRDUPDx, PWM_CDTYUPDx and PWM_DTUPDx).

To trigger the update, the user must use the bit UPDULOCK of the “PWM Sync Channels Update Control Register” (PWM_SCUC) which allows to update synchronously (at the same PWM period) the synchronous channels:

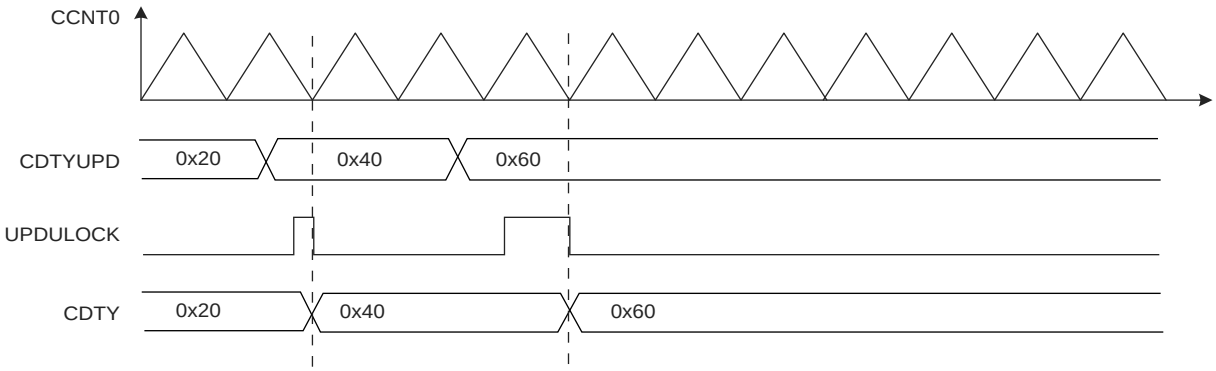
- If the bit UPDULOCK is set to 1, the update is done at the next PWM period of the synchronous channels.
- If the UPDULOCK bit is not set to 1, the update is locked and cannot be performed.

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

Sequence for Method 1:

1. Select the manual write of duty-cycle values and the manual update by setting the UPDM field to 0 in the PWM_SCM register
2. Define the synchronous channels by the SYNCx bits in the PWM_SCM register.
3. Enable the synchronous channels by writing CHID0 in the PWM_ENA register.
4. If an update of the period value and/or the duty-cycle values and/or the dead-time values is required, write registers that need to be updated (PWM_CPRDUPDx, PWM_CDTYUPDx and PWM_DTUPDx).
5. Set UPDULOCK to 1 in PWM_SCUC.
6. The update of the registers will occur at the beginning of the next PWM period. At this moment the UPDULOCK bit is reset, go to Step 4.) for new values.

Figure 37-9. Method 1 (UPDM = 0)



- **EPT_x: Endpoint x Interrupt Enable**

0 = disable the interrupts for this endpoint.

1 = enable the interrupts for this endpoint.

- **DMA_x: DMA Channel x Interrupt Enable**

0 = disable the interrupts for this channel.

1 = enable the interrupts for this channel.

39.3.3.2 Chunk Transactions

Writing a 1 to the DMAC_CREQ[2x] register starts a source chunk transaction request, where x is the channel number. Writing a 1 to the DMAC_CREQ[2x+1] register starts a destination chunk transfer request, where x is the channel number.

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

39.3.3.3 Single Transactions

Writing a 1 to the DMAC_SREQ[2x] register starts a source single transaction request, where x is the channel number. Writing a 1 to the DMAC_SREQ[2x+1] register starts a destination single transfer request, where x is the channel number.

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

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

39.3.4 DMAC Transfer Types

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

- Buffer chaining using linked lists
- Contiguous address between buffers

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

- Buffer chaining using linked lists

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

- Buffer chaining using linked lists

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

39.3.4.1 Multi-buffer Transfers

39.3.4.2 Buffer Chaining Using Linked Lists

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

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

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

The DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLAx and DMAC_CTRLBx registers are fetched from system memory on an LLI update. The updated content of the DMAC_CTRLAx register is written back to memory on buffer completion. Figure 39-4 on page 1010 shows how to use chained linked lists in memory to define multi-buffer transfers using buffer chaining.

The Linked List multi-buffer transfer is initiated by programming DMAC_DSCRx with DSCRx(0) (LLI(0) base address) and DMAC_CTRLBx register with both SRC_DSCR and DST_DSCR set to 0. Other fields and registers are ignored and overwritten when the descriptor is retrieved from memory.

43.5 Soldering Profile

Table 43-15 gives the recommended soldering profile from J-STD-020C.

Table 43-15. Soldering Profile

Profile Feature	Green Package
Average Ramp-up Rate (217°C to Peak)	3°C/sec. max.
Preheat Temperature 175°C ±25°C	180 sec. max.
Temperature Maintained Above 217°C	60 sec. to 150 sec.
Time within 5°C of Actual Peak Temperature	20 sec. to 40 sec.
Peak Temperature Range	260°C
Ramp-down Rate	6°C/sec. max.
Time 25°C to Peak Temperature	8 min. max.

Note: The package is certified to be backward compatible with Pb/Sn soldering profile.

A maximum of three reflow passes is allowed per component.

43.6 Packaging Resources

Land Pattern Definition.

Refer to the following IPC Standards:

- IPC-7351A and IPC-782 (*Generic Requirements for Surface Mount Design and Land Pattern Standards*) http://landpatterns.ipc.org/default.asp
- Atmel Green and RoHS Policy and Package Material Declaration Data Sheet available on www.atmel.com