Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	ARM® Cortex®-M4
Core Size	32-Bit Single-Core
Speed	120MHz
Connectivity	CANbus, Ethernet, IrDA, MMC/SD, SPI, UART/USART, USB
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	79
Program Memory Size	512KB (512K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	128K x 8
Voltage - Supply (Vcc/Vdd)	1.62V ~ 3.6V
Data Converters	A/D 16x12b; D/A 2x12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	100-TFBGA
Supplier Device Package	100-TFBGA (9x9)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atsam4e8cb-cnr

11.6.2 CMSIS Functions

ISO/IEC cannot directly access some Cortex-M4 instructions. This section describes intrinsic functions that can generate these instructions, provided by the CMIS and that might be provided by a C compiler. If a C compiler does not support an appropriate intrinsic function, the user might have to use inline assembler to access some instructions.

The CMSIS provides the following intrinsic functions to generate instructions that ISO/IEC C code cannot directly access:

Table 11-14. CMSIS Functions to Generate some Cortex-M4 Instructions

Instruction	CMSIS Function
CPSIE I	void __enable_irq(void)
CPSID I	void __disable_irq(void)
CPSIE F	void __enable_fault_irq(void)
CPSID F	void __disable_fault_irq(void)
ISB	void __ISB(void)
DSB	void __DSB(void)
DMB	void __DMB(void)
REV	uint32_t __REV(uint32_t int value)
REV16	uint32_t __REV16(uint32_t int value)
REVSH	uint32_t __REVSH(uint32_t int value)
RBIT	uint32_t __RBIT(uint32_t int value)
SEV	void __SEV(void)
WFE	void __WFE(void)
WFI	void __WFI(void)

The CMSIS also provides a number of functions for accessing the special registers using MRS and MSR instructions:

Table 11-15. CMSIS Intrinsic Functions to Access the Special Registers

Special Register	Access	CMSIS Function
PRIMASK	Read	uint32_t __get_PRIMASK (void)
	Write	void __set_PRIMASK (uint32_t value)
FAULTMASK	Read	uint32_t __get_FAULTMASK (void
	Write	void __set_FAULTMASK (uint32_t value)
BASEPRI	Read	uint32_t __get_BASEPRI (void)
	Write	void __set_BASEPRI (uint32_t value)
CONTROL	Read	uint32_t __get_CONTROL (void)
	Write	void __set_CONTROL (uint32_t value)
MSP	Read	uint32_t __get_MSP (void)
	Write	void __set_MSP (uint32_t TopOfMainStack)
PSP	Read	uint32_t __get_PSP (void)
	Write	void __set_PSP (uint32_t TopOfProcStack)

11.6.6.5 SMLAL and SMLALD

Signed Multiply Accumulate Long, Signed Multiply Accumulate Long (halfwords) and Signed Multiply Accumulate Long Dual.

Syntax

```
op{cond} RdLo, RdHi, Rn, Rm
op{XY}{cond} RdLo, RdHi, Rn, Rm
op{X}{cond} RdLo, RdHi, Rn, Rm
```

where:

op is one of:

MLAL Signed Multiply Accumulate Long.

SMLAL Signed Multiply Accumulate Long (halfwords, X and Y).

X and Y specify which halfword of the source registers *Rn* and *Rm* are used as the first and second multiply operand:

If *X* is B, then the bottom halfword, bits [15:0], of *Rn* is used.
If *X* is T, then the top halfword, bits [31:16], of *Rn* is used.

If *Y* is B, then the bottom halfword, bits [15:0], of *Rm* is used.
If *Y* is T, then the top halfword, bits [31:16], of *Rm* is used.

SMLALD Signed Multiply Accumulate Long Dual.

SMLALDX Signed Multiply Accumulate Long Dual Reversed.

If the *X* is omitted, the multiplications are bottom × bottom and top × top.

If *X* is present, the multiplications are bottom × top and top × bottom.

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

RdHi, RdLo are the destination registers.
RdLo is the lower 32 bits and *RdHi* is the upper 32 bits of the 64-bit integer.
For SMLAL, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD and SMLALDX, they also hold the accumulating value.

Rn, Rm are registers holding the first and second operands.

Operation

The SMLAL instruction:

- Multiplies the two’s complement signed word values from *Rn* and *Rm*.
- Adds the 64-bit value in *RdLo* and *RdHi* to the resulting 64-bit product.
- Writes the 64-bit result of the multiplication and addition in *RdLo* and *RdHi*.

The SMLALBB, SMLALBT, SMLALTB and SMLALTT instructions:

- Multiplies the specified signed halfword, Top or Bottom, values from *Rn* and *Rm*.
- Adds the resulting sign-extended 32-bit product to the 64-bit value in *RdLo* and *RdHi*.
- Writes the 64-bit result of the multiplication and addition in *RdLo* and *RdHi*.

The non-specified halfwords of the source registers are ignored.

The SMLALD and SMLALDX instructions interpret the values from *Rn* and *Rm* as four halfword two’s complement signed 16-bit integers. These instructions:

- If *X* is not present, multiply the top signed halfword value of *Rn* with the top signed halfword of *Rm* and the bottom signed halfword values of *Rn* with the bottom signed halfword of *Rm*.
- Or if *X* is present, multiply the top signed halfword value of *Rn* with the bottom signed halfword of *Rm* and the bottom signed halfword values of *Rn* with the top signed halfword of *Rm*.

11.6.11.16 VMOV ARM Core Register to Single Precision

Transfers a single-precision register to and from an ARM core register.

Syntax

```
VMOV{cond} Sn, Rt
VMOV{cond} Rt, Sn
```

where:

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

Sn is the single-precision floating-point register.

Rt is the ARM core register.

Operation

This instruction transfers:

- The contents of a single-precision register to an ARM core register.
- The contents of an ARM core register to a single-precision register.

Restrictions

Rt cannot be PC or SP.

Condition Flags

These instructions do not change the flags.

11.6.12.2 CPS

Change Processor State.

Syntax

```
CPSeffect iflags
```

where:

effect is one of:

- IE Clears the special purpose register.
- ID Sets the special purpose register.

iflags is a sequence of one or more flags:

- i Set or clear PRIMASK.
- f Set or clear FAULTMASK.

Operation

CPS changes the PRIMASK and FAULTMASK special register values. See “Exception Mask Registers” for more information about these registers.

Restrictions

The restrictions are:

- Use CPS only from privileged software, it has no effect if used in unprivileged software
- CPS cannot be conditional and so must not be used inside an IT block.

Condition Flags

This instruction does not change the condition flags.

Examples

```
CPSID i  ; Disable interrupts and configurable fault handlers (set PRIMASK)
CPSID f  ; Disable interrupts and all fault handlers (set FAULTMASK)
CPSIE i  ; Enable interrupts and configurable fault handlers (clear PRIMASK)
CPSIE f  ; Enable interrupts and fault handlers (clear FAULTMASK)
```

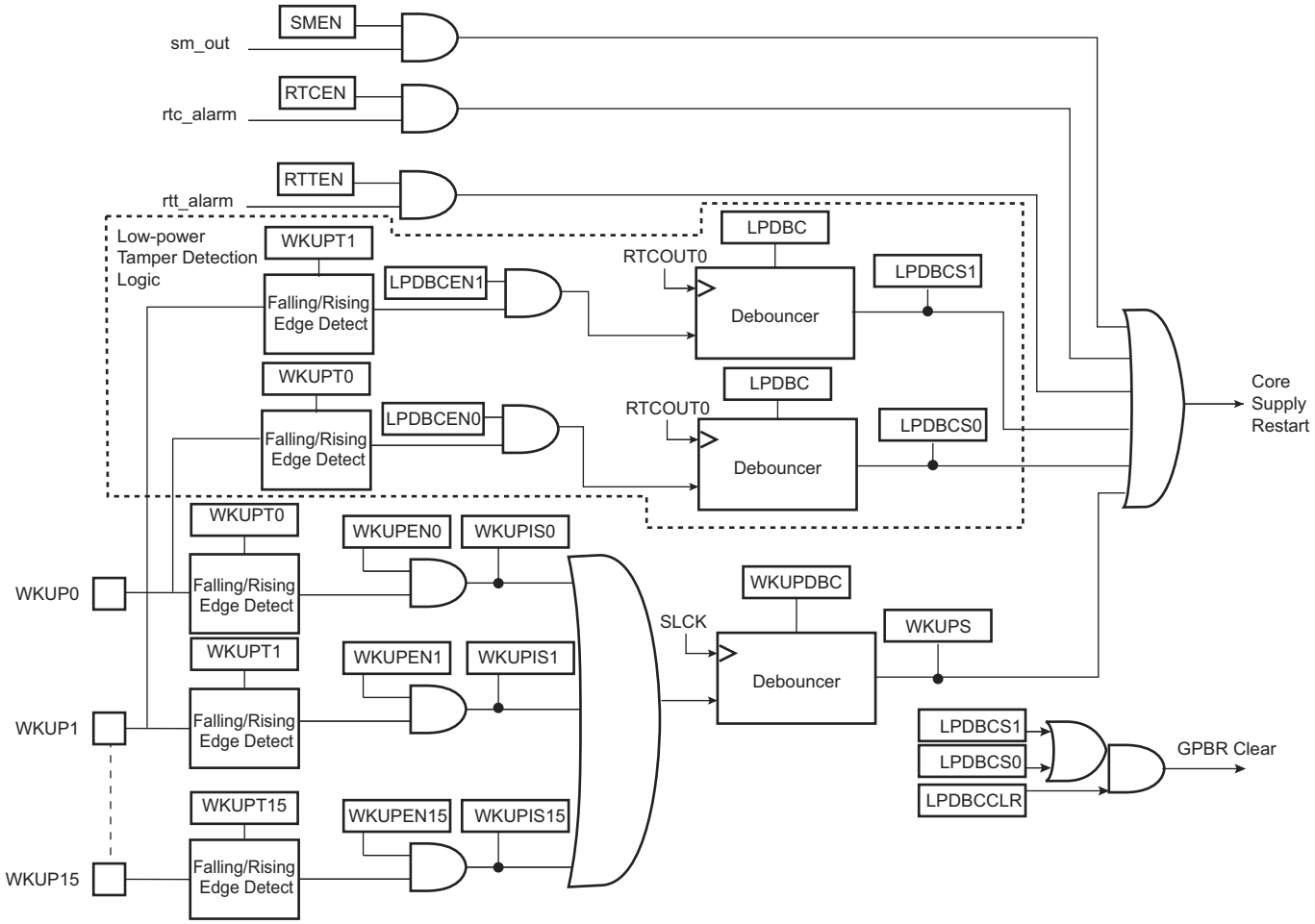
If BODRSTEN is set and the voltage regulation is lost (output voltage of the regulator too low), the vddcore_nreset signal is asserted for a minimum of one slow clock cycle and then released if bodcore_in has been reactivated. The BODRSTS bit in SUPC_SR indicates the source of the last reset.

Until bodcore_in is deactivated, the vddcore_nreset signal remains active.

18.4.7 Wake-up Sources

The wake-up events allow the device to exit Backup mode. When a wake-up event is detected, the SUPC performs a sequence that automatically reenables the core power supply.

Figure 18-4. Wake-up Sources



18.4.7.1 Force Wake-up

The FWUP pin is enabled as a wake-up source by writing a 1 to the FWUPEN bit in SUPC_WUMR. The FWUPDBC field then selects a debouncing period of 3, 32, 512, 4,096 or 32,768 slow clock cycles. The duration of these periods corresponds, respectively, to about 100 µs, about 1 ms, about 16 ms, about 128 ms and about 1 second (for a typical slow clock frequency of 32 kHz). Programming FWUPDBC to 0x0 selects an immediate wake-up, i.e., the FWUP must be low during at least one slow clock period to wake up the core power supply.

If the FWUP pin is asserted for a time longer than the debouncing period, a wake-up of the core power supply is started and the FWUPS bit in SUPC_SR is set and remains high until the register is read.

24.12 Bus Matrix (MATRIX) User Interface

Table 24-3. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Master Configuration Register 0	MATRIX_MCFG0	Read-write	0x00000001
0x0004	Master Configuration Register 1	MATRIX_MCFG1	Read-write	0x00000000
0x0008	Master Configuration Register 2	MATRIX_MCFG2	Read-write	0x00000000
0x000C	Master Configuration Register 3	MATRIX_MCFG3	Read-write	0x00000000
0x0010	Master Configuration Register 4	MATRIX_MCFG4	Read-write	0x00000000
0x0014	Master Configuration Register 5	MATRIX_MCFG5	Read-write	0x00000000
0x0018	Master Configuration Register 6	MATRIX_MCFG6	Read-write	0x00000000
0x001C-0x003C	Reserved	–	–	–
0x0040	Slave Configuration Register 0	MATRIX_SCFG0	Read-write	0x000001FF
0x0044	Slave Configuration Register 1	MATRIX_SCFG1	Read-write	0x000001FF
0x0048	Slave Configuration Register 2	MATRIX_SCFG2	Read-write	0x000001FF
0x004C	Slave Configuration Register 3	MATRIX_SCFG3	Read-write	0x000001FF
0x0050	Slave Configuration Register 4	MATRIX_SCFG4	Read-write	0x000001FF
0x0054	Slave Configuration Register 5	MATRIX_SCFG5	Read-write	0x000001FF
0x0058-0x007C	Reserved	–	–	–
0x0080	Priority Register A for Slave 0	MATRIX_PRAS0	Read-write	0x33333333(1)
0x0084	Reserved	–	–	–
0x0088	Priority Register A for Slave 1	MATRIX_PRAS1	Read-write	0x33333333(1)
0x008C	Reserved	–	–	–
0x0090	Priority Register A for Slave 2	MATRIX_PRAS2	Read-write	0x33333333(1)
0x0094	Reserved	–	–	–
0x0098	Priority Register A for Slave 3	MATRIX_PRAS3	Read-write	0x33333333(1)
0x009C	Reserved	–	–	–
0x00A0	Priority Register A for Slave 4	MATRIX_PRAS4	Read-write	0x33333333(1)
0x00A4	Reserved	–	–	–
0x00A8	Priority Register A for Slave 5	MATRIX_PRAS5	Read-write	0x33333333(1)
0x00AC	Reserved	–	–	–
0x00B4-0x00FC	Reserved	–	–	–
0x0100	Master Remap Control Register	MATRIX_MRCR	Read-write	0x00000000
0x0104 - 0x010C	Reserved	–	–	–
0x0110	Reserved	–	–	–
0x0114	System I/O Configuration Register	CCFG_SYSIO	Read-write	0x00000000
0x0118 - 0x0120	Reserved	–	–	–
0x0124	SMC NAND Flash Chip Select Configuration Register	CCFG_SMCNFCS	Read-write	0x00000000

25.8.12 DMAC Channel Handler Status Register

Name: DMAC_CHSR

Address: 0x400C0030

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	STAL3	STAL2	STAL1	STAL0
23	22	21	20	19	18	17	16
–	–	–	–	EMPT3	EMPT2	EMPT1	EMPT0
15	14	13	12	11	10	9	8
–	–	–	–	SUSP3	SUSP2	SUSP1	SUSP0
7	6	5	4	3	2	1	0
–	–	–	–	ENA3	ENA2	ENA1	ENA0

- **ENAx: Enable [3:0]**

A one in any position of this field indicates that the relevant channel is enabled.

- **SUSPx: Suspend [3:0]**

A one in any position of this field indicates that the channel transfer is suspended.

- **EMPTx: Empty [3:0]**

A one in any position of this field indicates that the relevant channel is empty.

- **STALx: Stalled [3:0]**

A one in any position of this field indicates that the relevant channel is stalling.

33.6.8 PIO Input Filter Disable Register

Name: PIO_IFDR

Address: 0x400E0E24 (PIOA), 0x400E1024 (PIOB), 0x400E1224 (PIOC), 0x400E1424 (PIOD), 0x400E1624 (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: Input Filter Disable**

0: No effect.

1: Disables the input glitch filter on the I/O line.

34.5 Signal Description

Table 34-1. Signal Description

Pin Name	Pin Description	Type	
		Master	Slave
MISO	Master In Slave Out	Input	Output
MOSI	Master Out Slave In	Output	Input
SPCK	Serial Clock	Output	Input
NPCS1–NPCS3	Peripheral Chip Selects	Output	Unused
NPCS0/NSS	Peripheral Chip Select/Slave Select	Output	Input

34.6 Product Dependencies

34.6.1 I/O Lines

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

Table 34-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
SPI	MISO	PA12	A
SPI	MOSI	PA13	A
SPI	NPCS0	PA11	A
SPI	NPCS1	PA9	B
SPI	NPCS1	PA31	A
SPI	NPCS1	PB14	A
SPI	NPCS1	PC4	B
SPI	NPCS2	PA10	B
SPI	NPCS2	PA30	B
SPI	NPCS2	PB2	B
SPI	NPCS3	PA3	B
SPI	NPCS3	PA5	B
SPI	NPCS3	PA22	B
SPI	SPCK	PA14	A

34.6.2 Power Management

The SPI can be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the SPI clock.

sending the IADR) is sometimes called “repeated start” (Sr) in I^2C fully-compatible devices. See Figure 35-11. See Figure 35-10 and Figure 35-12 for master write operation with internal address.

The three internal address bytes are configurable through the Master Mode register (TWI_MMR).

If the slave device supports only a 7-bit address, i.e., no internal address, IADRSZ must be set to 0.

Table 35-6 shows the abbreviations used in Figure 35-10 and Figure 35-11.

Table 35-6. Abbreviations

Abbreviation	Definition
S	Start
Sr	Repeated Start
P	Stop
W	Write
R	Read
A	Acknowledge
NA	Not Acknowledge
DADR	Device Address
IADR	Internal Address

Figure 35-10. Master Write with One, Two or Three Bytes Internal Address and One Data Byte

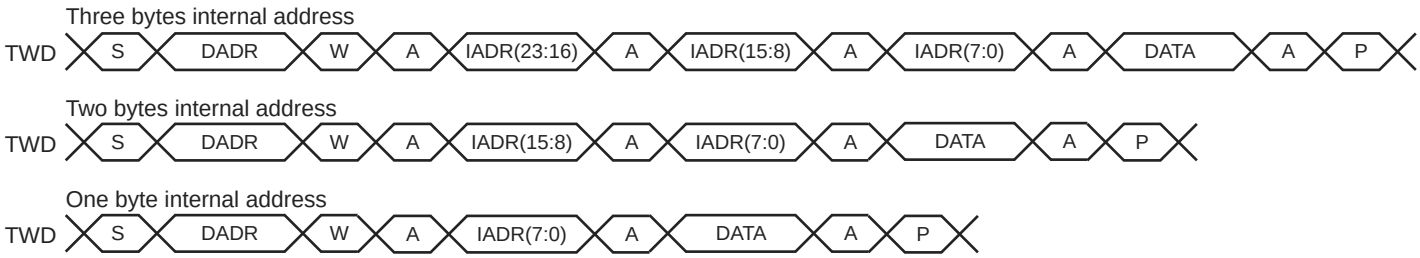
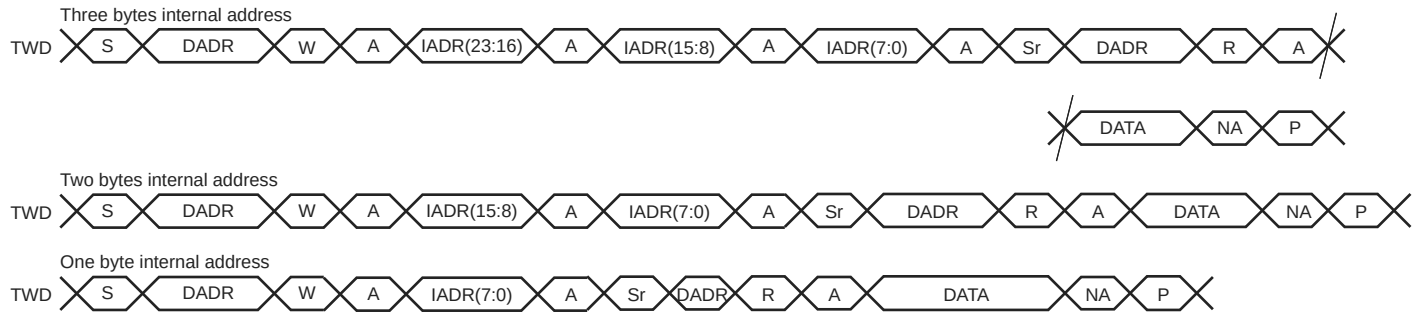


Figure 35-11. Master Read with One, Two or Three Bytes Internal Address and One Data Byte



10-bit Slave Addressing

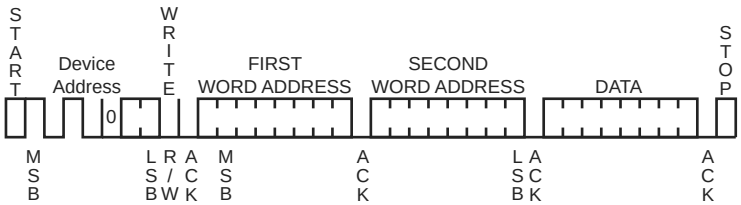
For a slave address higher than seven bits, the user must configure the address size **(**IADRSZ) and set the other slave address bits in the Internal Address register (TWI_IADR). The two remaining internal address bytes, IADR[15:8] and IADR[23:16] can be used the same way as in 7-bit slave addressing.

Example: Address a 10-bit device (10-bit device address is b1 b2 b3 b4 b5 b6 b7 b8 b9 b10)

1. Program IADRSZ = 1,
2. Program DADR with 1 1 1 1 0 b1 b2 (b1 is the MSB of the 10-bit address, b2, etc.)
3. Program TWI_IADR with b3 b4 b5 b6 b7 b8 b9 b10 (b10 is the LSB of the 10-bit address)

Figure 35-12 below shows a byte write to a memory device. This demonstrates the use of internal addresses to access the device.

Figure 35-12. Internal Address Usage



35.7.3.6 Using the Peripheral DMA Controller (PDC)

The use of the PDC significantly reduces the CPU load.

To ensure correct implementation, proceed as follows.

Data Transmit with the PDC

1. Initialize the transmit PDC (memory pointers, transfer size - 1).
2. Configure the master (DADR, CKDIV, MREAD = 0, etc.)
3. Start the transfer by setting the PDC TXTEN bit.
4. Wait for the PDC ENDTX Flag either by using the polling method or ENDTX interrupt.
5. Disable the PDC by setting the PDC TXTDIS bit.
6. Wait for the TXRDY flag in TWI_SR.
7. Set the STOP bit in TWI_CR.
8. Write the last character in TWI_THR.
9. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

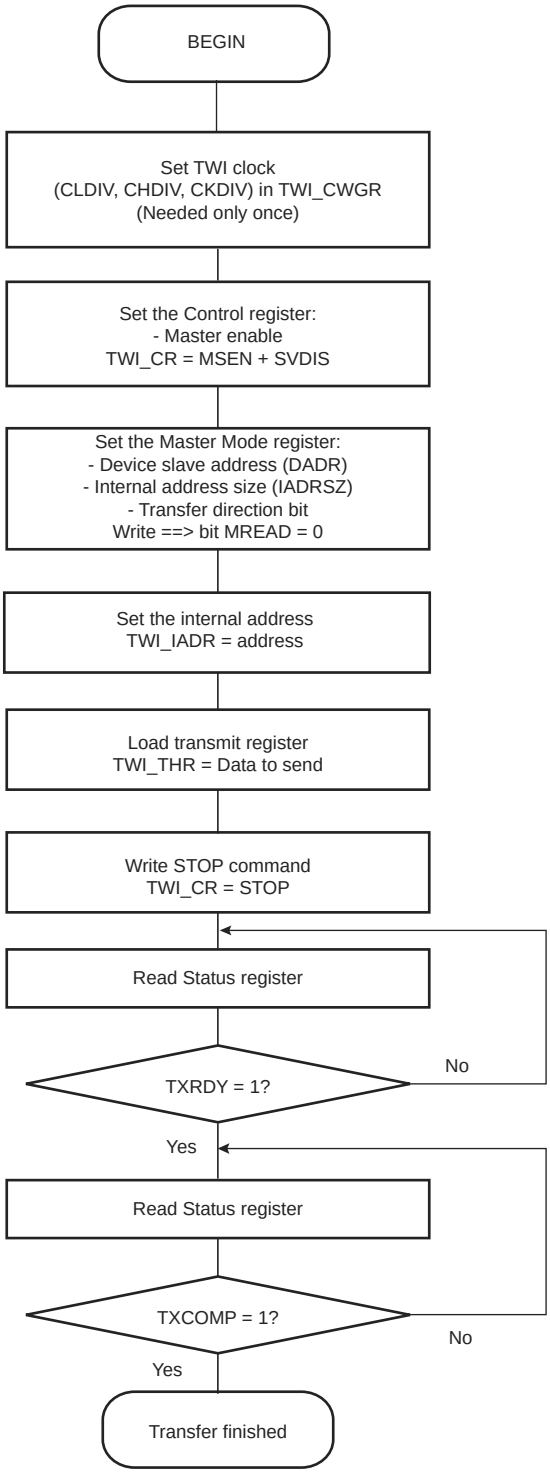
Data Receive with the PDC

The PDC transfer size must be defined with the buffer size minus 2. The two remaining characters must be managed without PDC to ensure that the exact number of bytes are received regardless of system bus latency conditions encountered during the end of buffer transfer period.

In Slave mode, the number of characters to receive must be known in order to configure the PDC.

1. Initialize the receive PDC (memory pointers, transfer size - 2).
2. Configure the master (DADR, CKDIV, MREAD = 1, etc.)
3. Set the PDC RXTEN bit.
4. (Master Only) Write the START bit in the TWI_CR to start the transfer.
5. Wait for the PDC ENDRX Flag either by using polling method or ENDRX interrupt.
6. Disable the PDC by setting the PDC RXTDIS bit.
7. Wait for the RXRDY flag in TWI_SR.
8. Set the STOP bit in TWI_CR.
9. Read the penultimate character in TWI_RHR.
10. Wait for the RXRDY flag in TWI_SR.
11. Read the last character in TWI_RHR.
12. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

Figure 35-15. TWI Write Operation with Single Data Byte and Internal Address



35.7.4 Multi-master Mode

35.7.4.1 Definition

In Multi-master mode, more than one master may handle the bus at the same time without data corruption by using arbitration.

Arbitration starts as soon as two or more masters place information on the bus at the same time, and stops (arbitration is lost) for the master that intends to send a logical one while the other master sends a logical zero.

As soon as a master lose arbitration, it stops sending data and listens to the bus in order to detect a stop. When the stop is detected, the master may put its data on the bus by performing arbitration.

Arbitration is illustrated in Figure 35-21.

35.7.4.2 Two Multi-master Modes

Two Multi-master modes may be distinguished:

1. TWI is considered as a master only and will never be addressed.
2. TWI may be either a master or a slave and may be addressed.

Note: Arbitration is supported in both Multi-master modes.

TWI as Master Only

In this mode, TWI is considered as a Master only (MSEN is always one) and must be driven like a Master with the ARBLST (Arbitration Lost) flag in addition.

If arbitration is lost (ARBLST = 1), the user must reinitiate the data transfer.

If the user starts a transfer (ex.: DADR + START + W + Write in THR) and if the bus is busy, the TWI automatically waits for a STOP condition on the bus to initiate the transfer (see Figure 35-20).

Note: The state of the bus (busy or free) is not shown in the user interface.

TWI as Master or Slave

The automatic reversal from Master to Slave is not supported in case of a lost arbitration.

Then, in the case where TWI may be either a Master or a Slave, the user must manage the pseudo Multi-master mode described in the steps below.

1. Program TWI in Slave mode (SADR + MSDIS + SVEN) and perform a slave access (if TWI is addressed).
2. If the TWI has to be set in Master mode, wait until the TXCOMP flag is at 1.
3. Program the Master mode (DADR + SVDIS + MSEN) and start the transfer (ex: START + Write in THR).
4. As soon as the Master mode is enabled, the TWI scans the bus in order to detect if it is busy or free. When the bus is considered free, TWI initiates the transfer.
5. As soon as the transfer is initiated and until a STOP condition is sent, the arbitration becomes relevant and the user must monitor the ARBLST flag.
6. If the arbitration is lost (ARBLST is set to 1), the user must program the TWI in Slave mode in case the Master that won the arbitration is required to access the TWI.
7. If the TWI has to be set in Slave mode, wait until TXCOMP flag is at 1 and then program the Slave mode.

Note: If the arbitration is lost and the TWI is addressed, the TWI will not acknowledge even if it is programmed in Slave mode as soon as ARBLST is set to 1. Then the Master must repeat SADR.

36.4 Product Dependencies

36.4.1 I/O Lines

The UART pins are multiplexed with PIO lines. The user must first configure the corresponding PIO Controller to enable I/O line operations of the UART.

Table 36-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
UART0	URXD0	PA9	A
UART0	UTXD0	PA10	A
UART1	URXD1	PA5	C
UART1	UTXD1	PA6	C

36.4.2 Power Management

The UART clock can be controlled through the Power Management Controller (PMC). In this case, the user must first configure the PMC to enable the UART clock. Usually, the peripheral identifier used for this purpose is 1.

36.4.3 Interrupt Sources

The UART interrupt line is connected to one of the interrupt sources of the Interrupt Controller. Interrupt handling requires programming of the Interrupt Controller before configuring the UART.

Table 36-3. Peripheral IDs

Instance	ID
UART0	7
UART1	45

36.5 Functional Description

The UART operates in Asynchronous mode only and supports only 8-bit character handling (with parity). It has no clock pin.

The UART is made up of a receiver and a transmitter that operate independently, and a common baud rate generator. Receiver timeout and transmitter time guard are not implemented. However, all the implemented features are compatible with those of a standard USART.

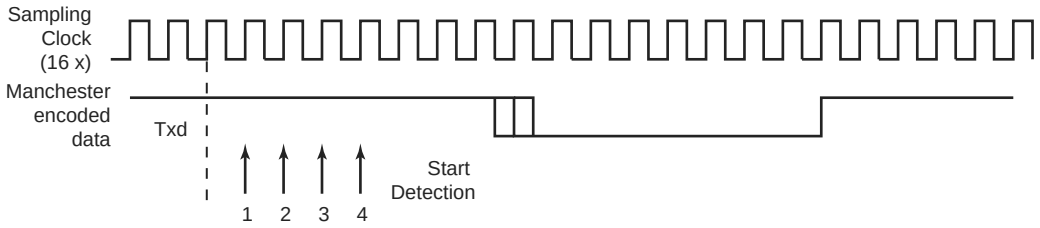
36.5.1 Baud Rate Generator

The baud rate generator provides the bit period clock named baud rate clock to both the receiver and the transmitter.

The baud rate clock is the peripheral clock divided by 16 times the clock divisor (CD) value written in the Baud Rate Generator register (UART_BRGR). If UART_BRGR is set to 0, the baud rate clock is disabled and the UART remains inactive. The maximum allowable baud rate is peripheral clock divided by 16. The minimum allowable baud rate is peripheral clock divided by (16 x 65536).

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 37-13. The sample pulse rejection mechanism applies.

Figure 37-13. 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 resynchronizes 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 37-14 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, the MANERR flag in the US_CSR is raised. It is cleared by writing a 1 to the RSTSTA in the US_CR. See Figure 37-15 for an example of Manchester error detection during data phase.

Figure 37-14. Preamble Pattern Mismatch

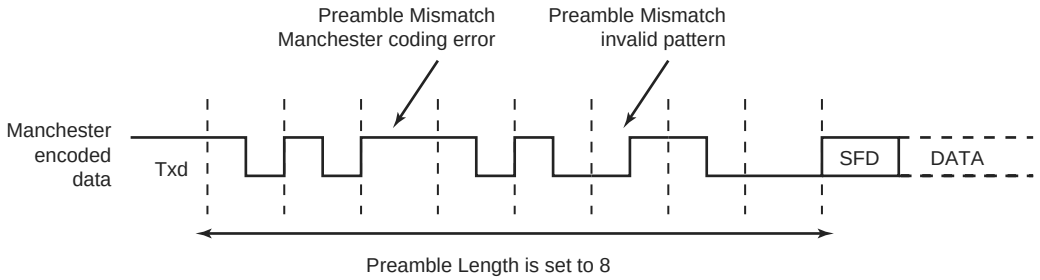
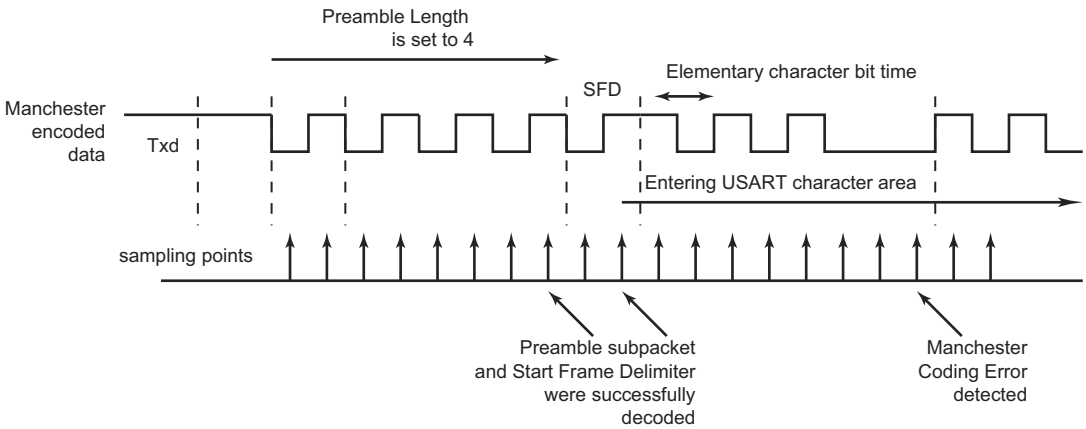


Figure 37-15. Manchester Error Flag



37.7.10 USART Interrupt Mask Register (SPI_MODE)

Name: US_IMR (SPI_MODE)

Address: 0x400A0010 (0), 0x400A4010 (1)

Access: Read-only

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

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

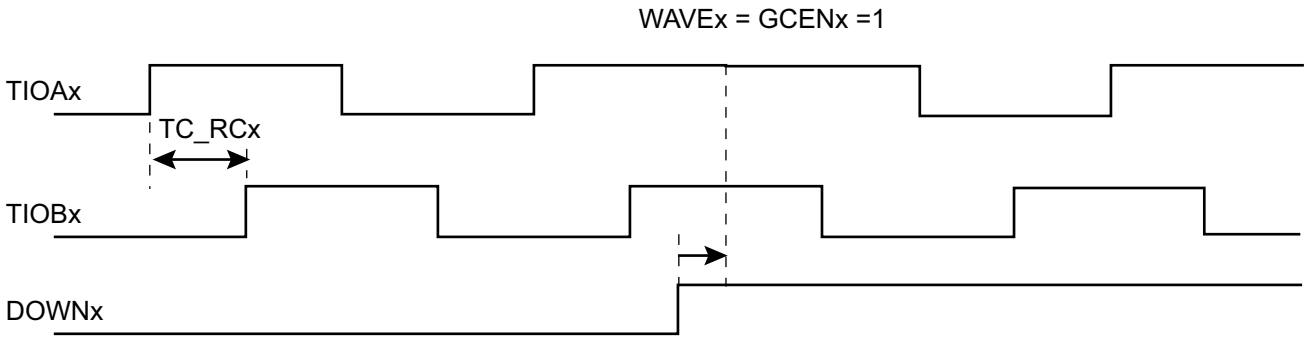
The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **UNRE: SPI Underrun Error Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **NSSE: NSS Line (Driving CTS Pin) Rising or Falling Edge Event Interrupt Mask**

Figure 38-22. 2-bit Gray Up/Down Counter



38.6.18 Fault Mode

At any time, the TC_RCx registers can be used to perform a comparison on the respective current channel counter value (TC_CVx) with the value of TC_RCx register.

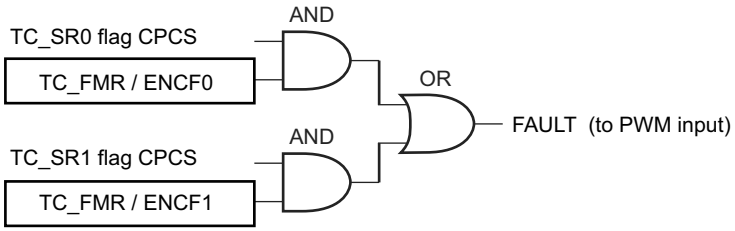
The CPCSx flags can be set accordingly and an interrupt can be generated.

This interrupt is processed but requires an unpredictable amount of time to be achieve the required action.

It is possible to trigger the FAULT output of the TIMER1 with CPCS from TC_SR0 and/or CPCS from TC_SR1. Each source can be independently enabled/disabled in the TC_FMR.

This can be useful to detect an overflow on speed and/or position when QDEC is processed and to act immediately by using the FAULT output.

Figure 38-23. Fault Output Generation



41.6.3.2 Entering Attached State

To enable integrated pull-up, the PUON bit in the UDP_TXVC register must be set.

Warning: To write to the UDP_TXVC register, MCK clock must be enabled on the UDP. This is done in the Power Management Controller.

After pull-up connection, the device enters the powered state. In this state, the UDPCK and MCK must be enabled in the Power Management Controller. The transceiver can remain disabled.

41.6.3.3 From Powered State to Default State

After its connection to a USB host, the USB device waits for an end-of-bus reset. The unmaskable flag ENDBUSRES is set in the UDP_ISR and an interrupt is triggered.

Once the ENDBUSRES interrupt has been triggered, the device enters Default State. In this state, the UDP software must:

- Enable the default endpoint, setting the EPEDS flag in the UDP_CSR0 and, optionally, enabling the interrupt for endpoint 0 by writing 1 to the UDP_IER. The enumeration then begins by a control transfer.
- Configure the interrupt mask register which has been reset by the USB reset detection
- Enable the transceiver clearing the TXVDIS flag in the UDP_TXVC register.

In this state UDPCK and MCK must be enabled.

Warning: Each time an ENDBUSRES interrupt is triggered, the Interrupt Mask Register and UDP_CSRs have been reset.

41.6.3.4 From Default State to Address State

After a set address standard device request, the USB host peripheral enters the address state.

Warning: Before the device enters in address state, it must achieve the Status IN transaction of the control transfer, i.e., the UDP device sets its new address once the TXCOMP flag in the UDP_CSR0 has been received and cleared.

To move to address state, the driver software sets the FADDEN flag in the UDP_GLB_STAT register, sets its new address, and sets the FEN bit in the UDP_FADDR register.

41.6.3.5 From Address State to Configured State

Once a valid Set Configuration standard request has been received and acknowledged, the device enables endpoints corresponding to the current configuration. This is done by setting the EPEDS and EPTYPE fields in the UDP_CSRx and, optionally, enabling corresponding interrupts in the UDP_IER.

41.6.3.6 Entering in Suspend State

When a Suspend (no bus activity on the USB bus) is detected, the RXSUSP signal in the UDP_ISR is set. This triggers an interrupt if the corresponding bit is set in the UDP_IMR. This flag is cleared by writing to the UDP_ICR. Then the device enters Suspend Mode.

In this state bus powered devices must drain no more than 2.5 mA from the 5V VBUS. As an example, the microcontroller switches to slow clock, disables the PLL and main oscillator, and goes into Idle Mode. It may also switch off other devices on the board.

The USB device peripheral clocks can be switched off. Resume event is asynchronously detected. MCK and UDPCK can be switched off in the Power Management controller and the USB transceiver can be disabled by setting the TXVDIS bit in the UDP_TXVC register.

Warning: Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. Switching off MCK for the UDP peripheral must be one of the last operations after writing to the UDP_TXVC register and acknowledging the RXSUSP.

Table 42-17. Register Mapping (Continued)

Offset[1] [2]	Register	Name	Access	Reset
0x0C8	Specific Address 1 Mask Bottom Register	GMAC_SAMB1	Read/Write	0x0000_0000
0x0CC	Specific Address 1 Mask Top Register	GMAC_SAMT1	Read/Write	0x0000_0000
0x0D0–0x0E4	Reserved	–	–	–
0x0E8–0x0FC	Reserved	–	–	–
0x100–0x1B0	Reserved	–	–	–
0x1B4–0x1CC	Reserved	–	–	–
0x1D0	1588 Timer Seconds Low Register	GMAC_TSL	Read/Write	0x0000_0000
0x1D4	1588 Timer Nanoseconds Register	GMAC_TN	Read/Write	0x0000_0000
0x1D8	1588 Timer Adjust Register	GMAC_TA	Write-only	–
0x1DC	1588 Timer Increment Register	GMAC_TI	Read/Write	0x0000_0000
0x1E0	PTP Event Frame Transmitted Seconds Low Register	GMAC_EFTSL	Read-only	0x0000_0000
0x1E4	PTP Event Frame Transmitted Nanoseconds Register	GMAC_EFTN	Read-only	0x0000_0000
0x1E8	PTP Event Frame Received Seconds Low Register	GMAC_EFRSL	Read-only	0x0000_0000
0x1EC	PTP Event Frame Received Nanoseconds Register	GMAC_EFRN	Read-only	0x0000_0000
0x1F0	PTP Peer Event Frame Transmitted Seconds Low Register	GMAC_PEFTSL	Read-only	0x0000_0000
0x1F4	PTP Peer Event Frame Transmitted Nanoseconds Register	GMAC_PEFTN	Read-only	0x0000_0000
0x1F8	PTP Peer Event Frame Received Seconds Low Register	GMAC_PEFRSL	Read-only	0x0000_0000
0x1FC	PTP Peer Event Frame Received Nanoseconds Register	GMAC_PEFRN	Read-only	0x0000_0000
0x200–0x7FC	Reserved	–	–	–

Notes: 1. If an offset is not listed in the Register Mapping, it must be considered as ‘reserved’.
2. Some register groups are not continuous in memory.

42.8.36 GMAC 1588 Timer Seconds Low Register

Name: GMAC_TSL

Address: 0x400341D0

Access: Read/Write

31	30	29	28	27	26	25	24
TCS							

23	22	21	20	19	18	17	16
TCS							

15	14	13	12	11	10	9	8
TCS							

7	6	5	4	3	2	1	0
TCS							

- **TCS: Timer Count in Seconds**

This register is writable. It increments by one when the 1588 nanoseconds counter counts to one second. It may also be incremented when the Timer Adjust Register is written.