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/M4F
Core Size	32-Bit Dual-Core
Speed	120MHz
Connectivity	EBI/EMI, I²C, IrDA, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, DMA, LCD, POR, PWM, WDT
Number of I/O	57
Program Memory Size	256KB (256K 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 6x10b
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/atsam4cms4cc-au

Table 12-13. Cortex-M4 Instructions (Continued)

Mnemonic	Operands	Description	Flags
UHADD16	{Rd,} Rn, Rm	Unsigned Halving Add 16	–
UHADD8	{Rd,} Rn, Rm	Unsigned Halving Add 8	–
UHASX	{Rd,} Rn, Rm	Unsigned Halving Add and Subtract with Exchange	–
UHSAX	{Rd,} Rn, Rm	Unsigned Halving Subtract and Add with Exchange	–
UHSUB16	{Rd,} Rn, Rm	Unsigned Halving Subtract 16	–
UHSUB8	{Rd,} Rn, Rm	Unsigned Halving Subtract 8	–
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract	–
UDIV	{Rd,} Rn, Rm	Unsigned Divide	–
UMAAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply Accumulate Accumulate Long ($32 \times 32 + 32 + 32$), 64-bit result	–
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate ($32 \times 32 + 64$), 64-bit result	–
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply (32×32), 64-bit result	–
UQADD16	{Rd,} Rn, Rm	Unsigned Saturating Add 16	–
UQADD8	{Rd,} Rn, Rm	Unsigned Saturating Add 8	–
UQASX	{Rd,} Rn, Rm	Unsigned Saturating Add and Subtract with Exchange	–
UQSAX	{Rd,} Rn, Rm	Unsigned Saturating Subtract and Add with Exchange	–
UQSUB16	{Rd,} Rn, Rm	Unsigned Saturating Subtract 16	–
UQSUB8	{Rd,} Rn, Rm	Unsigned Saturating Subtract 8	–
USAD8	{Rd,} Rn, Rm	Unsigned Sum of Absolute Differences	–
USADA8	{Rd,} Rn, Rm, Ra	Unsigned Sum of Absolute Differences and Accumulate	–
USAT	Rd, #n, Rm {,shift #s}	Unsigned Saturate	Q
USAT16	Rd, #n, Rm	Unsigned Saturate 16	Q
UASX	{Rd,} Rn, Rm	Unsigned Add and Subtract with Exchange	GE
USUB16	{Rd,} Rn, Rm	Unsigned Subtract 16	GE
USUB8	{Rd,} Rn, Rm	Unsigned Subtract 8	GE
UXTAB	{Rd,} Rn, Rm,{,ROR #}	Rotate, extend 8 bits to 32 and Add	–
UXTAB16	{Rd,} Rn, Rm,{,ROR #}	Rotate, dual extend 8 bits to 16 and Add	–
UXTAH	{Rd,} Rn, Rm,{,ROR #}	Rotate, unsigned extend and Add Halfword	–
UXTB	{Rd,} Rm {,ROR #n}	Zero extend a byte	–
UXTB16	{Rd,} Rm {,ROR #n}	Unsigned Extend Byte 16	–
UXTH	{Rd,} Rm {,ROR #n}	Zero extend a halfword	–
VABS.F32	Sd, Sm	Floating-point Absolute	–
VADD.F32	{Sd,} Sn, Sm	Floating-point Add	–
VCMP.F32	Sd, <Sm | #0.0>	Compare two floating-point registers, or one floating-point register and zero	FPSCR
VCMPE.F32	Sd, <Sm | #0.0>	Compare two floating-point registers, or one floating-point register and zero with Invalid Operation check	FPSCR
VCVT.S32.F32	Sd, Sm	Convert between floating-point and integer	–

12.6.5 General Data Processing Instructions

The table below shows the data processing instructions.

Table 12-20. Data Processing Instructions

Mnemonic	Description
ADC	Add with Carry
ADD	Add
ADDW	Add
AND	Logical AND
ASR	Arithmetic Shift Right
BIC	Bit Clear
CLZ	Count leading zeros
CMN	Compare Negative
CMP	Compare
EOR	Exclusive OR
LSL	Logical Shift Left
LSR	Logical Shift Right
MOV	Move
MOVT	Move Top
MOVW	Move 16-bit constant
MVN	Move NOT
ORN	Logical OR NOT
ORR	Logical OR
RBIT	Reverse Bits
REV	Reverse byte order in a word
REV16	Reverse byte order in each halfword
REVSH	Reverse byte order in bottom halfword and sign extend
ROR	Rotate Right
RRX	Rotate Right with Extend
RSB	Reverse Subtract
SADD16	Signed Add 16
SADD8	Signed Add 8
SASX	Signed Add and Subtract with Exchange
SSAX	Signed Subtract and Add with Exchange
SBC	Subtract with Carry
SHADD16	Signed Halving Add 16
SHADD8	Signed Halving Add 8
SHASX	Signed Halving Add and Subtract with Exchange
SHSAX	Signed Halving Subtract and Add with Exchange

12.6.5.16 UADD16 and UADD8

Unsigned Add 16 and Unsigned Add 8

Syntax

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

where:

op is any of:

UADD16 Performs two 16-bit unsigned integer additions.

UADD8 Performs four 8-bit unsigned integer additions.

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

Rd is the destination register.

Rn is the first register holding the operand.

Rm is the second register holding the operand.

Operation

Use these instructions to add 16- and 8-bit unsigned data:

The UADD16 instruction:

1. Adds each halfword from the first operand to the corresponding halfword of the second operand.
2. Writes the unsigned result in the corresponding halfwords of the destination register.

The UADD16 instruction:

1. Adds each byte of the first operand to the corresponding byte of the second operand.
2. Writes the unsigned result in the corresponding byte of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
UADD16 R1, R0      ; Adds halfwords in R0 to corresponding halfword of R1,
                   ; writes to corresponding halfword of R1
UADD8  R4, R0, R5  ; Adds bytes of R0 to corresponding byte in R5 and
                   ; writes to corresponding byte in R4.
```

Examples

```
QADD16   R7, R4, R2  ; Adds halfwords of R4 with corresponding halfword of
                     ; R2, saturates to 16 bits and writes to
                     ; corresponding halfword of R7
QADD8    R3, R1, R6  ; Adds bytes of R1 to the corresponding bytes of R6,
                     ; saturates to 8 bits and writes to corresponding
                     ; byte of R3
QSUB16   R4, R2, R3  ; Subtracts halfwords of R3 from corresponding
                     ; halfword of R2, saturates to 16 bits, writes to
                     ; corresponding halfword of R4
QSUB8    R4, R2, R5  ; Subtracts bytes of R5 from the corresponding byte
                     ; in R2, saturates to 8 bits, writes to corresponding
                     ; byte of R4.
```

12.6.7.4 QASX and QSAX

Saturating Add and Subtract with Exchange and Saturating Subtract and Add with Exchange, signed.

Syntax

op{cond} {Rd}, Rm, Rn

where:

op — is one of:

QASX Add and Subtract with Exchange and Saturate.

QSAX Subtract and Add with Exchange and Saturate.

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

Rd — is the destination register.

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

Operation

The QASX instruction:

1. Adds the top halfword of the source operand with the bottom halfword of the second operand.
2. Subtracts the top halfword of the second operand from the bottom highword of the first operand.
3. Saturates the result of the subtraction and writes a 16-bit signed integer in the range $-2^{15} \le x \le 2^{15} - 1$, where *x* equals 16, to the bottom halfword of the destination register.
4. Saturates the results of the sum and writes a 16-bit signed integer in the range $-2^{15} \le x \le 2^{15} - 1$, where *x* equals 16, to the top halfword of the destination register.

The QSAX instruction:

1. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
2. Adds the bottom halfword of the source operand with the top halfword of the second operand.
3. Saturates the results of the sum and writes a 16-bit signed integer in the range $-2^{15} \le x \le 2^{15} - 1$, where *x* equals 16, to the bottom halfword of the destination register.
4. Saturates the result of the subtraction and writes a 16-bit signed integer in the range $-2^{15} \le x \le 2^{15} - 1$, where *x* equals 16, to the top halfword of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

12.6.10.3 IT

If-Then condition instruction.

Syntax

```
IT{x{y{z}}} cond
```

where:

x specifies the condition switch for the second instruction in the IT block.

y specifies the condition switch for the third instruction in the IT block.

z specifies the condition switch for the fourth instruction in the IT block.

cond specifies the condition for the first instruction in the IT block.

The condition switch for the second, third and fourth instruction in the IT block can be either:

T Then. Applies the condition *cond* to the instruction.

E Else. Applies the inverse condition of *cond* to the instruction.

It is possible to use AL (the *always* condition) for *cond* in an IT instruction. If this is done, all of the instructions in the IT block must be unconditional, and each of *x*, *y*, and *z* must be T or omitted but not E.

Operation

The IT instruction makes up to four following instructions conditional. The conditions can be all the same, or some of them can be the logical inverse of the others. The conditional instructions following the IT instruction form the *IT block*.

The instructions in the IT block, including any branches, must specify the condition in the {*cond*} part of their syntax.

The assembler might be able to generate the required IT instructions for conditional instructions automatically, so that the user does not have to write them. See the assembler documentation for details.

A BKPT instruction in an IT block is always executed, even if its condition fails.

Exceptions can be taken between an IT instruction and the corresponding IT block, or within an IT block. Such an exception results in entry to the appropriate exception handler, with suitable return information in LR and stacked PSR.

Instructions designed for use for exception returns can be used as normal to return from the exception, and execution of the IT block resumes correctly. This is the only way that a PC-modifying instruction is permitted to branch to an instruction in an IT block.

Restrictions

The following instructions are not permitted in an IT block:

- IT
- CBZ and CBNZ
- CPSID and CPSIE.

Other restrictions when using an IT block are:

- A branch or any instruction that modifies the PC must either be outside an IT block or must be the last instruction inside the IT block. These are:
  - ADD PC, PC, Rm
  - MOV PC, Rm
  - B, BL, BX, BLX
  - Any LDM, LDR, or POP instruction that writes to the PC
  - TBB and TBH
- Do not branch to any instruction inside an IT block, except when returning from an exception handler

- **MLSPERR: MemManage During Lazy State Preservation**

This is part of “MMFSR: Memory Management Fault Status Subregister”.

0: No MemManage fault occurred during the floating-point lazy state preservation.

1: A MemManage fault occurred during the floating-point lazy state preservation.

- **MMARVALID: Memory Management Fault Address Register (SCB_MMFAR) Valid Flag**

This is part of “MMFSR: Memory Management Fault Status Subregister”.

0: The value in SCB_MMFAR is not a valid fault address.

1: SCB_MMFAR holds a valid fault address.

If a memory management fault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems on return to a stacked active memory management fault handler whose SCB_MMFAR value has been overwritten.

- **IBUSERR: Instruction Bus Error**

This is part of “BFSR: Bus Fault Status Subregister”.

0: No instruction bus error.

1: Instruction bus error.

The processor detects the instruction bus error on prefetching an instruction, but it sets the IBUSERR flag to 1 only if it attempts to issue the faulting instruction.

When the processor sets this bit to 1, it does not write a fault address to the BFAR.

- **PRECISERR: Precise Data Bus Error**

This is part of “BFSR: Bus Fault Status Subregister”.

0: No precise data bus error.

1: A data bus error has occurred, and the PC value stacked for the exception return points to the instruction that caused the fault.

When the processor sets this bit to 1, it writes the faulting address to the SCB_BFAR.

- **IMPRECISERR: Imprecise Data Bus Error**

This is part of “BFSR: Bus Fault Status Subregister”.

0: No imprecise data bus error.

1: A data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error.

When the processor sets this bit to 1, it does not write a fault address to the SCB_BFAR.

This is an asynchronous fault. Therefore, if it is detected when the priority of the current process is higher than the bus fault priority, the bus fault becomes pending and becomes active only when the processor returns from all higher priority processes. If a precise fault occurs before the processor enters the handler for the imprecise bus fault, the handler detects that both this bit and one of the precise fault status bits are set to 1.

- **UNSTKERR: Bus Fault on Unstacking for a Return From Exception**

This is part of “BFSR: Bus Fault Status Subregister”.

0: No unstacking fault.

1: Unstack for an exception return has caused one or more bus faults.

12.11 Memory Protection Unit (MPU)

The MPU divides the memory map into a number of regions, and defines the location, size, access permissions, and memory attributes of each region. It supports:

- Independent attribute settings for each region
- Overlapping regions
- Export of memory attributes to the system.

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M4 MPU defines:

- Eight separate memory regions, 0–7
- A background region.

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M4 MPU memory map is unified. This means that instruction accesses and data accesses have the same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a memory management fault. This causes a fault exception, and might cause the termination of the process in an OS environment.

In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

The configuration of MPU regions is based on memory types (see "Memory Regions, Types and Attributes").

Table 12-36 shows the possible MPU region attributes. These include Share ability and cache behavior attributes that are not relevant to most microcontroller implementations. See "MPU Configuration for a Microcontroller" for guidelines for programming such an implementation.

Table 12-36. Memory Attributes Summary

Memory Type	Shareability	Other Attributes	Description
Strongly-ordered	–	–	All accesses to Strongly-ordered memory occur in program order. All Strongly-ordered regions are assumed to be shared.
Device	Shared	–	Memory-mapped peripherals that several processors share.
	Non-shared	–	Memory-mapped peripherals that only a single processor uses.
Normal	Shared	–	Normal memory that is shared between several processors.
	Non-shared	–	Normal memory that only a single processor uses.

18.5.2 Watchdog Timer Mode Register

Name: WDT_MR

Address: 0x400E1454

Access: Read/Write Once

31	30	29	28	27	26	25	24
–	–	WDIDLEHLT	WDDBGHLT	WDD			
23	22	21	20	19	18	17	16
WDD							
15	14	13	12	11	10	9	8
WDDIS	WDRPROC	WDRSTEN	WDFIEN	WDV			
7	6	5	4	3	2	1	0
WDV							

Note: The first write access prevents any further modification of the value of this register. Read accesses remain possible.

Note: The WDD and WDV values must not be modified within three slow clock periods following a restart of the watchdog performed by a write access in WDT_CR. Any modification will cause the watchdog to trigger an end of period earlier than expected.

- **WDV: Watchdog Counter Value**

Defines the value loaded in the 12-bit watchdog counter.

- **WDFIEN: Watchdog Fault Interrupt Enable**

0: A watchdog fault (underflow or error) has no effect on interrupt.

1: A watchdog fault (underflow or error) asserts interrupt.

- **WDRSTEN: Watchdog Reset Enable**

0: A watchdog fault (underflow or error) has no effect on the resets.

1: A watchdog fault (underflow or error) triggers a watchdog reset.

- **WDRPROC: Watchdog Reset Processor**

0: If WDRSTEN is 1, a watchdog fault (underflow or error) activates all resets.

1: If WDRSTEN is 1, a watchdog fault (underflow or error) activates the processor reset.

- **WDDIS: Watchdog Disable**

0: Enables the Watchdog Timer.

1: Disables the Watchdog Timer.

- **WDD: Watchdog Delta Value**

Defines the permitted range for reloading the Watchdog Timer.

If the Watchdog Timer value is less than or equal to WDD, setting bit WDT_CR.WDRSTT restarts the timer.

If the Watchdog Timer value is greater than WDD, setting bit WDT_CR.WDRSTT causes a watchdog error.

- **CLSIZE: Cache Line Size**

Value	Name	Description
0	CLSIZE_1KB	Cache line size is 4 bytes
1	CLSIZE_2KB	Cache line size is 8 bytes
2	CLSIZE_4KB	Cache line size is 16 bytes
3	CLSIZE_8KB	Cache line size is 32 bytes

28.5.1 Receive Pointer Register

Name: PERIPH_RPR

Access: Read/Write

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

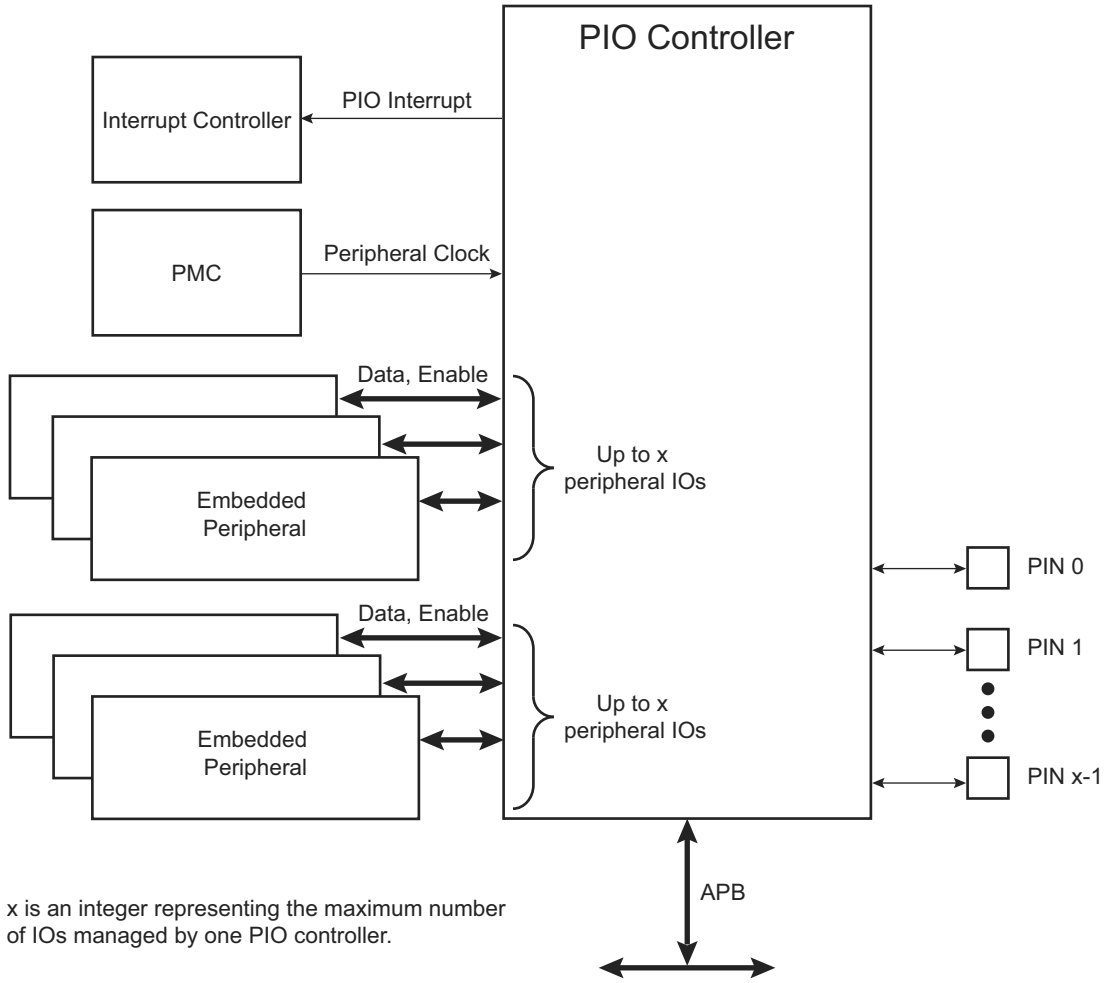
- **RXPTR: Receive Pointer Register**

RXPTR must be set to receive buffer address.

When a half-duplex peripheral is connected to the PDC, RXPTR = TXPTR.

32.3 Block Diagram

Figure 32-1. Block Diagram



32.4 Product Dependencies

32.4.1 Pin Multiplexing

Each pin is configurable, depending on the product, as either a general-purpose I/O line only, or as an I/O line multiplexed with one or two peripheral I/Os. As the multiplexing is hardware defined and thus product-dependent, the hardware designer and programmer must carefully determine the configuration of the PIO Controllers required by their application. When an I/O line is general-purpose only, i.e., not multiplexed with any peripheral I/O, programming of the PIO Controller regarding the assignment to a peripheral has no effect and only the PIO Controller can control how the pin is driven by the product.

32.4.2 Power Management

The Power Management Controller controls the peripheral clock in order to save power. Writing any of the registers of the user interface does not require the peripheral clock to be enabled. This means that the configuration of the I/O lines does not require the peripheral clock to be enabled.

33.8.2 SPI Mode Register

Name: SPI_MR

Address: 0x40008004 (0), 0x48000004 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
DLYBCS							
23	22	21	20	19	18	17	16
–	–	–	–	PCS			
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
LLB	–	WDRBT	MODFDIS	–	PCSDEC	PS	MSTR

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

- **MSTR: Master/Slave Mode**

0: SPI is in Slave mode

1: SPI is in Master mode

- **PS: Peripheral Select**

0: Fixed Peripheral Select

1: Variable Peripheral Select

- **PCSDEC: Chip Select Decode**

0: The chip selects are directly connected to a peripheral device.

1: The four NPCS chip select lines are connected to a 4-bit to 16-bit decoder.

When PCSDEC = 1, up to 15 Chip Select signals can be generated with the four NPCS lines using an external 4-bit to 16-bit decoder. The Chip Select registers define the characteristics of the 15 chip selects, with the following rules:

SPI_CSR0 defines peripheral chip select signals 0 to 3.

SPI_CSR1 defines peripheral chip select signals 4 to 7.

SPI_CSR2 defines peripheral chip select signals 8 to 11.

SPI_CSR3 defines peripheral chip select signals 12 to 14.

- **MODFDIS: Mode Fault Detection**

0: Mode fault detection enabled

1: Mode fault detection disabled

- **WDRBT: Wait Data Read Before Transfer**

0: No Effect. In Master mode, a transfer can be initiated regardless of the SPI_RDR state.

1: In Master mode, a transfer can start only if the SPI_RDR is empty, i.e., does not contain any unread data. This mode prevents overrun error in reception.

10-bit Slave Addressing

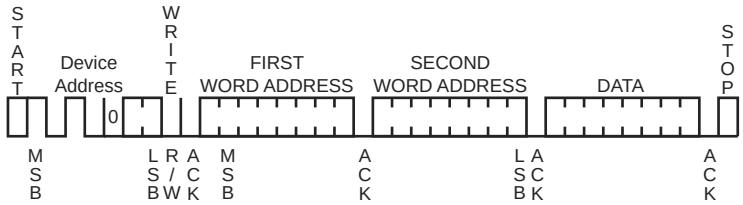
For a slave address higher than seven bits, the user must configure the address size **(**IADRSZ) and set the other slave address bits in the Internal Address register (TWI_IADR). The two remaining internal address bytes, IADR[15:8] and IADR[23:16] can be used the same way as in 7-bit slave addressing.

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 34-13 below shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

Figure 34-13. Internal Address Usage



34.7.3.7 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.

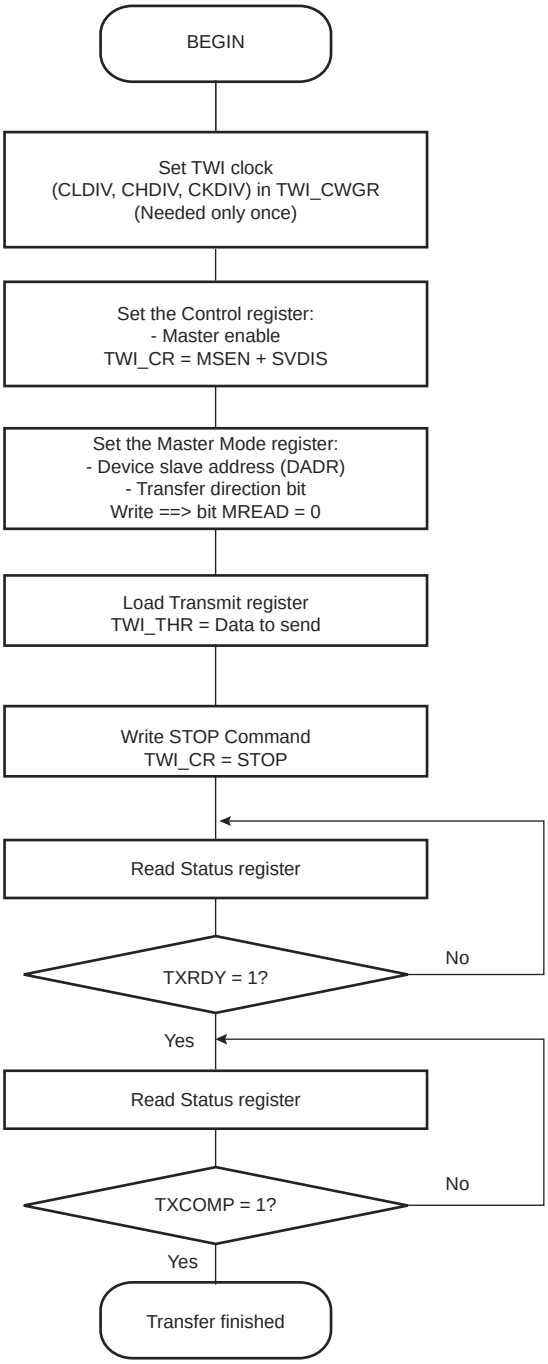
Data Receive with the PDC

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.

Figure 34-15. TWI Write Operation with Single Data Byte without Internal Address



When OPT_EN = 1, the URXD pad is automatically configured in Analog mode and the analog comparator is enabled (see Figure 35-11).

To match the characteristics of the off-chip optical receiver circuitry, the voltage reference threshold of the embedded comparator can be adjusted from VDDIO/10 up to VDD/2 by programming the OPT_CMPTH field in UART_MR.

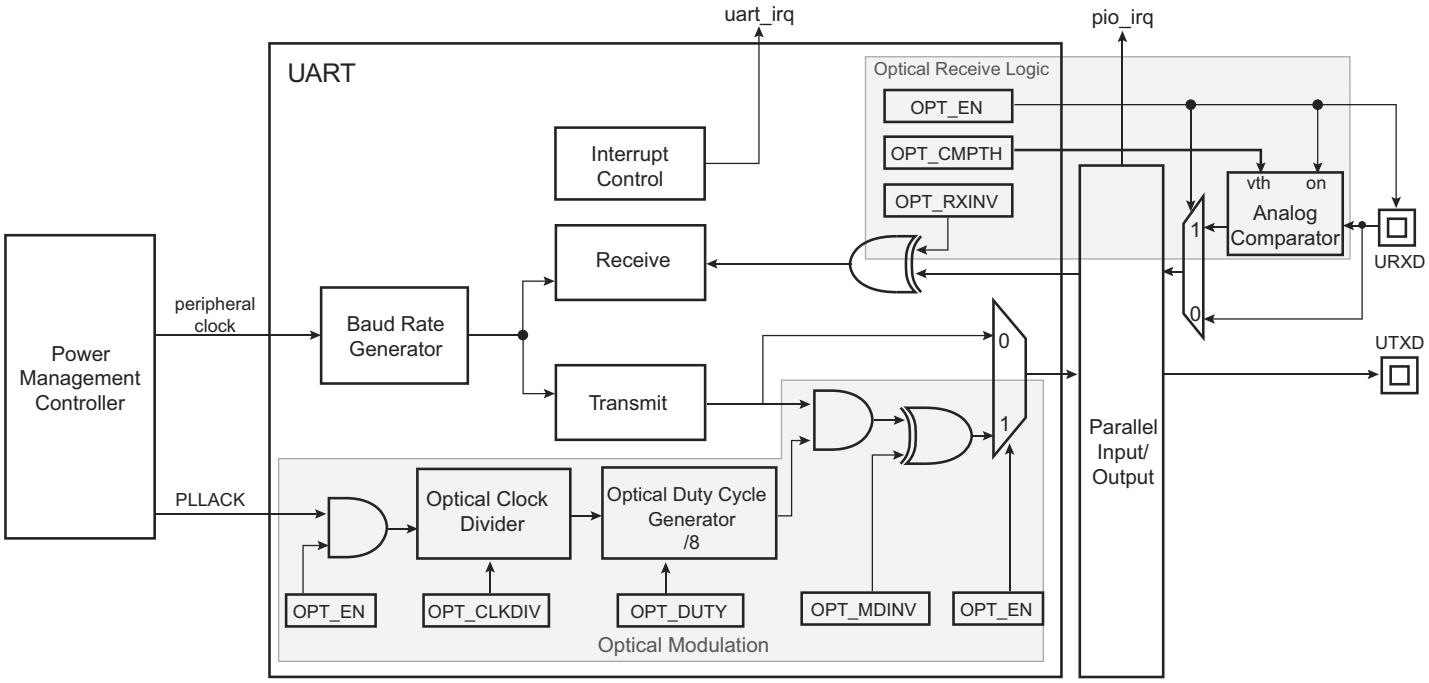
The NRZ output of the UART transmitter sub-module is modulated with the 30 up to 60 kHz modulation clock prior to driving the PIO controller.

A logical 0 on the UART transmitter sub-module output generates the said modulated signal (see Figure 35-12) having a frequency programmable from 30 kHz up to 60 kHz (38 kHz is the default value assuming the PLLA clock frequency is 8192 kHz). A logical 1 on the UART transmitter sub-module output generates a stuck-at 1 output signal (no modulation). The idle polarity of the modulated signal is 1 (OPT_MDINV = 0 in UART_MR).

The idle polarity of the modulated signal can be inverted by writing a 1 to the OPT_MDINV bit in UART_MR.

The duty cycle of the modulated signal can be adjusted from 6.25% up to 50% (default value) by steps of 6.25% by programming the OPT_DUTY field in UART_MR.

Figure 35-11. Optical Interface Block Diagram



36.7.4 USART Mode Register (SPI_MODE)

Name: US_MR (SPI_MODE)

Address: 0x40024004 (0), 0x40028004 (1), 0x4002C004 (2), 0x40030004 (3), 0x40034004 (4)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	WRDBT	–	CLKO	–	CPOL
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	CPHA
7	6	5	4	3	2	1	0
CHRL		USCLKS		USART_MODE			

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

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

- **USART_MODE: USART Mode of Operation**

Value	Name	Description
0xE	SPI_MASTER	SPI master
0xF	SPI_SLAVE	SPI Slave

- **USCLKS: Clock Selection**

Value	Name	Description
0	MCK	Peripheral clock is selected
1	DIV	Peripheral clock divided (DIV=8) is selected
3	SCK	Serial Clock SLK is selected

- **CHRL: Character Length**

Value	Name	Description
3	8_BIT	Character length is 8 bits

- **CPHA: SPI Clock Phase**
  - Applicable if USART operates in SPI mode (USART_MODE = 0xE or 0xF):

0: Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

1: Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

CPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. CPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

37.7.8 TC Register C

Name: TC_RCx [x=0..2]

Address: 0x4001001C (0)[0], 0x4001005C (0)[1], 0x4001009C (0)[2], 0x4001401C (1)[0], 0x4001405C (1)[1], 0x4001409C (1)[2]

Access: Read/Write

31	30	29	28	27	26	25	24
RC							

23	22	21	20	19	18	17	16
RC							

15	14	13	12	11	10	9	8
RC							

7	6	5	4	3	2	1	0
RC							

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

- **RC: Register C**

RC contains the Register C value in real time.

IMPORTANT: For 16-bit channels, RC field size is limited to register bits 15:0.

37.7.16 TC QDEC Interrupt Disable Register

Name: TC_QIDR

Address: 0x400100CC (0), 0x400140CC (1)

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
–	–	–	–	–	QERR	DIRCHG	IDX

- **IDX: Index**

0: No effect.

1: Disables the interrupt when a rising edge occurs on IDX input.

- **DIRCHG: Direction Change**

0: No effect.

1: Disables the interrupt when a change on rotation direction is detected.

- **QERR: Quadrature Error**

0: No effect.

1: Disables the interrupt when a quadrature error occurs on PHA, PHB.

Table 46-20. LCD Buffers Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{DDIN}	Current consumption (VDDIN)	LDO enabled	–	25	35	µA
Z_{OUT}	Buffer output impedance	GPIO in LCD mode (SEG or COM)	200	500	1500	Ω
C_{LOAD}	Capacitive output load	–	10p	–	50n	F
t_r / t_f	Rising or falling time 95% convergence	C_{LOAD} = 10 pF C_{LOAD} = 50 nF	–	–	3 225	µs

46.5.4 VDDCORE Brownout Detector

Table 46-21. Core Power Supply Brownout Detector Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IT-}	Negative-going input threshold voltage (VDDCORE) (1)	–	0.98	1.0	1.04	V
V_{IT+}	Positive-going input threshold voltage (VDDCORE)	–	0.80	1.0	1.08	V
V_{HYST}	Hysteresis voltage	V_{IT+} - V_{IT-}	–	25	50	mV
t_{d-}	V_{IT-} detection propagation time	VDDCORE = V_{IT+} to (V_{IT-} - 100mV)	–	200	300	ns
t_{START}	Start-up time	From disabled state to enabled state	–	–	300	µs
I_{DDCORE}	Current consumption (VDDCORE)	Brownout detector enabled	–	–	15	µA
I_{DDIO}	Current consumption (VDDIO)	Brownout detector enabled	–	–	18	µA

Note: 1. The product is guaranteed to be functional at V_{IT-}.

Figure 46-13. Core Brownout Output Waveform

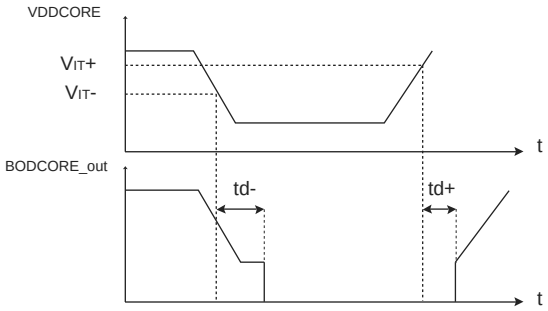


Figure 46-14. Core Brownout Transfer Characteristics

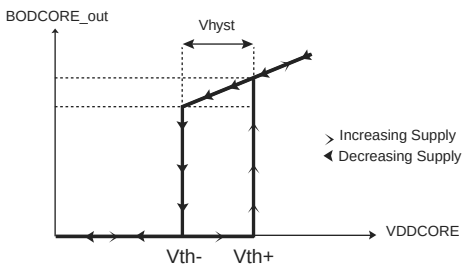
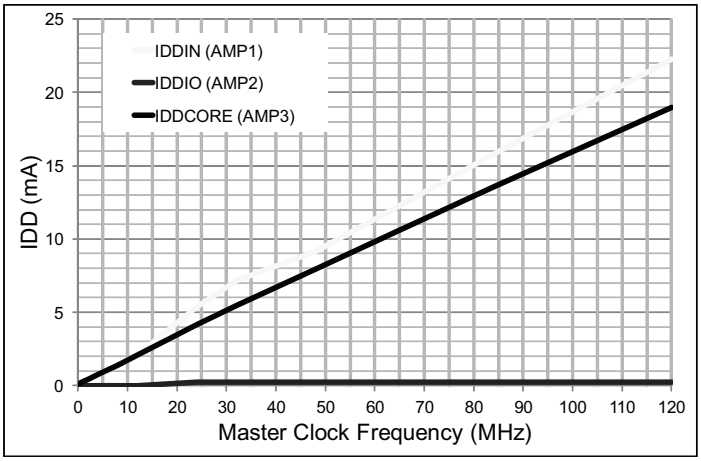
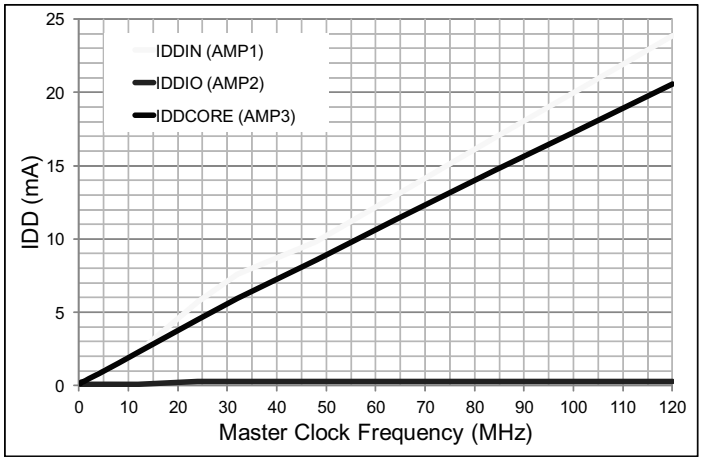


Figure 46-28. Typical Current Consumption in Active Mode (Test Setup 2)

SAM4Cx4/8/16



SAM4Cx32



46.7.4.3 Test Setup 3: CoreMark

- CoreMark on Core 0 (CM4P0) running out of Flash in 128-bit or 64-bit Access mode with and without Cache Enabled. Cache is enabled above 0 WS.
- CoreMark on Core 1 (CM4P1) running out of SRAM1 (Code) / SRAM2 (Data)

Table 46-66. SAM4CM4/8/16 Test Setup 3 Current Consumption

Clock (MHz)	128-bit Flash Access						64-bit Flash Access						Unit
	Cache Enabled			Cache Disabled			Cache Enabled			Cache Disabled			
	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	
120	31.3	0.28	28.0	34.2	1.9	30.9	31.3	0.28	28.0	30.7	1.8	27.4	mA
100	26.4	0.28	23.6	29.8	1.8	27.1	26.4	0.28	23.6	27.0	1.8	24.3	
84	22.4	0.28	20.1	26.3	1.7	24.0	22.4	0.28	20.1	24.1	1.7	21.8	
64	17.2	0.28	15.6	21.0	1.5	19.3	17.2	0.28	15.6	19.6	1.6	18.0	
48	13.1	0.28	11.8	16.6	1.4	15.3	13.1	0.28	11.8	16.0	1.6	14.7	
32	9.8	0.28	8.1	12.6	1.2	10.9	9.8	0.28	8.1	12.3	1.4	10.6	
24	7.4	0.28	6.2	9.5	1.1	8.3	7.4	0.28	6.2	9.4	1.3	8.1	
12	3.1	0.11	3.1	4.2	0.88	4.2	3.1	0.11	3.1	4.2	1.2	4.2	
8	2.1	0.11	2.1	2.8	0.78	2.8	2.1	0.11	2.1	2.8	1.0	2.8	
4	1.1	0.11	1.1	1.5	0.58	1.5	1.1	0.11	1.1	1.5	0.9	1.5	
2	0.63	0.11	0.61	0.82	0.40	0.81	0.63	0.11	0.61	0.82	0.66	0.81	
1	0.38	0.11	0.37	0.47	0.26	0.46	0.38	0.11	0.37	0.47	0.38	0.46	
0.5	0.25	0.11	0.24	0.30	0.18	0.29	0.25	0.11	0.24	0.30	0.23	0.29	
0.25	0.14	0.11	0.13	0.16	0.12	0.15	0.14	0.11	0.13	0.16	0.14	0.15	