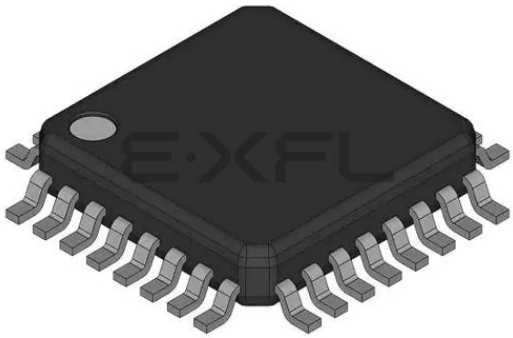


Welcome to E-XFL.COM

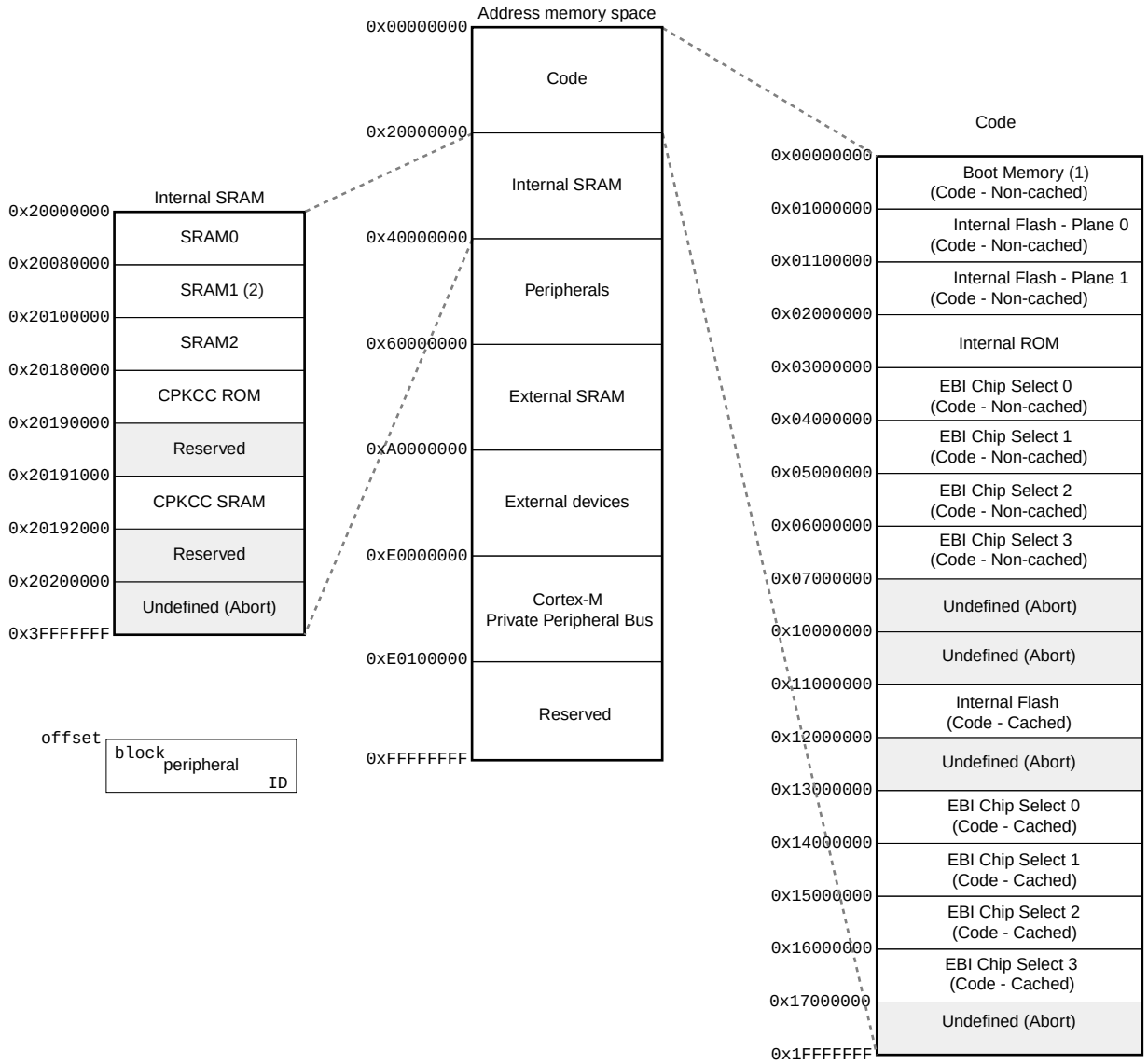
What is "Embedded - Microcontrollers"?

"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	74
Program Memory Size	1MB (1M x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	128K x 8
Voltage - Supply (Vcc/Vdd)	1.62V ~ 3.6V
Data Converters	A/D 8x10b
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/atmel/atsam4c16cb-au

Figure 7-3. SAM4C32 Memory Mapping of CODE and SRAM Area



Notes: 1. Boot Memory for Core 0.
2. Boot Memory for Core 1 at 0x00000000.

11.4.3 PIO Controller A Multiplexing

Table 11-5. Multiplexing on PIO Controller A (PIOA)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function	Feature	Reset State	Comments
PA0	RTS3	PCK2	A10	COM0	WKUP5	- PUP(P) / PDN(P) - ST(P) - MaxDRV(NP)	PIO, I, PU	
PA1	CTS3	NCS1	A9	COM1	–	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)		
PA2	SCK3	NCS2	A8	COM2	–			
PA3	RXD3	NCS3	A7	COM3	WKUP6			
PA4	TXD3	–	A6	COM4/AD1	–			
PA5	SPI0_NPCS0	–	A5	COM5/AD2	–			
PA6	SPI0_MISO	–	A4	SEG0	–			
PA7	SPI0_MOSI	–	A3	SEG1	–			
PA8	SPI0_SPCK	–	A2	SEG2	–	- PUP(P) / PDN(P) - ST(P) - MaxDRV(NP)		
PA9	RXD2	–	A1	SEG3	WKUP2	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)		
PA10	TXD2	–	A0/NBS0	SEG4	–			
PA11	RXD1	–	A23	SEG5	WKUP9			
PA12	TXD1	–	A22/ NANDCLE	SEG6/AD0	–			
PA13	SCK2	TIOA0	A21/ NANDALE	SEG7	–			
PA14	RTS2	TIOB0	A20	SEG8	WKUP3			
PA15	CTS2	TIOA4	A19	SEG9	–			
PA16	SCK1	TIOB4	A18	SEG10	–			
PA17	RTS1	TCLK4	A17	SEG11	WKUP7			
PA18	CTS1	TIOA5	A16	SEG12	–			
PA19	RTS0	TCLK5	A15	SEG13	WKUP4			
PA20	CTS0	TIOB5	A14	SEG14	–			
PA21	SPI0_NPCS1	–	A13	SEG15	–			
PA22	SPI0_NPCS2	–	A12	SEG16	–			
PA23	SPI0_NPCS3	–	A11	SEG17	–			
PA24	TWD0	–	A10	SEG18	WKUP1			
PA25	TWCK0	–	A9	SEG19	–			
PA26	CTS4	–	A8	SEG20	–			
PA27	–	–	NCS0	SEG21	–			
PA28	–	–	NRD	SEG22	–			

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.
```

12.6.11.11 VLDR

Loads a single extension register from memory

Syntax

```
VLDR{cond}{.64} Dd, [Rn{#imm}]
VLDR{cond}{.64} Dd, label
VLDR{cond}{.64} Dd, [PC, #imm}]
VLDR{cond}{.32} Sd, [Rn {, #imm}]
VLDR{cond}{.32} Sd, label
VLDR{cond}{.32} Sd, [PC, #imm]
```

where:

cond	is an optional condition code, see “Conditional Execution”.
64, 32	are the optional data size specifiers.
Dd	is the destination register for a doubleword load.
Sd	is the destination register for a singleword load.
Rn	is the base register. The SP can be used.
imm	is the + or - immediate offset used to form the address. Permitted address values are multiples of 4 in the range 0 to 1020.
label	is the label of the literal data item to be loaded.

Operation

This instruction:

- Loads a single extension register from memory, using a base address from an ARM core register, with an optional offset.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.25 VPUSH

Floating-point extension register Push.

Syntax

```
VPUSH{cond}{.size} list
```

where:

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

size — is an optional data size specifier.
If present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.

list — is a list of the extension registers to be stored, as a list of consecutively numbered doubleword or singleword registers, separated by commas and sur
rounded by brackets.

Operation

This instruction:

- Stores multiple consecutive extension registers to the stack.

Restrictions

The restrictions are:

- list must contain at least one register, and not more than sixteen.

Condition Flags

These instructions do not change the flags.

12.11.2.5 MPU Region Attribute and Size Register

Name: MPU_RASR

Access: Read/Write

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

23	22	21	20	19	18	17	16
–	–	TEX			S	C	B

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–	–	SIZE					ENABLE

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- The most significant halfword holds the region attributes.
- The least significant halfword holds the region size, and the region and subregion enable bits.

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 12-39.

- **TEX, C, B: Memory Access Attributes**

See Table 12-37.

- **S: Shareable**

See Table 12-37.

- **SRD: Subregion Disable**

For each bit in this field:

0: Corresponding subregion is enabled.

1: Corresponding subregion is disabled.

See “Subregions” for more information.

Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as 0x00.

Region	A partition of memory space.
Reserved	A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.
Thread-safe	In a multi-tasking environment, thread-safe functions use safeguard mechanisms when accessing shared resources, to ensure correct operation without the risk of shared access conflicts.
Thumb instruction	One or two halfwords that specify an operation for a processor to perform. Thumb instructions must be halfword-aligned.
Unaligned	A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.
Undefined	Indicates an instruction that generates an Undefined instruction exception.
Unpredictable	One cannot rely on the behavior. Unpredictable behavior must not represent security holes. Unpredictable behavior must not halt or hang the processor, or any parts of the system.
Warm reset	Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if debugging features of a processor.
Word	A 32-bit data item.
Write	Writes are defined as operations that have the semantics of a store. Writes include the Thumb instructions STM, STR, STRH, STRB, and PUSH.

15.4.3.4 Software Reset

The Reset Controller offers commands to assert the different reset signals. These commands are performed by writing the Control Register (RSTC_CR) or Coprocessor Mode Register (RSTC_CPMR) with the following bits at 1:

- RSTC_CR.PROCRST: Writing a 1 to PROCRST resets the processor and the watchdog timer.
- RSTC_CR.PERRST: Writing a 1 to PERRST resets all the embedded peripherals associated to processor whereas the coprocessor peripherals are not reset, including the memory system, and, in particular, the Remap Command. The Peripheral Reset is generally used for debug purposes.
  Except for debug purposes, PERRST must always be used in conjunction with PROCRST (PERRST and PROCRST set both at 1 simultaneously).
- RSTC_CPMR.CPROCEN: Writing a 0 to CPROCEN resets the coprocessor only.
- RSTC_CPMR.CPEREN: Writing a 0 to CPEREN resets all the embedded peripherals associated to coprocessor whereas the processor peripherals are not reset.
- RSTC_CR.EXTRST: Writing a 1 to EXTRST asserts low the NRST pin during a time defined by the field RSTC_MR.ERSTL.

The software reset is entered if at least one of these bits is written to 1 by the software. All these commands can be performed independently or simultaneously. The software reset lasts three slow clock cycles.

The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset has ended, i.e., synchronously to SLCK.

If EXTRST is written to 1, the nrst_out signal is asserted depending on the configuration of field RSTC_MR.ERSTL. However, the resulting falling edge on NRST does not lead to a user reset.

If and only if the PROCRST bit is written to 1, the Reset Controller reports the software status in field RSTC_SR.RSTTYP. Other software resets are not reported in RSTTYP.

As soon as a software operation is detected, the bit SRCMP (Software Reset Command in Progress) is written to 1 in the RSTC_SR. SRCMP is cleared at the end of the software reset. No other software reset can be performed while the SRCMP bit is written to 1, and writing any value in the RSTC_CR has no effect.

19.5.2 Reinforced Safety Watchdog Timer Mode Register

Name: RSWDT_MR

Address: 0x400E1504

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	–	WDV			
7	6	5	4	3	2	1	0
WDV							

Notes:
1. The first write access prevents any further modification of the value of this register; read accesses remain possible.
2. The WDD and WDV values must not be modified within three slow clock periods following a restart of the watchdog performed by means of a write access in the RSWDT_CR, else the watchdog may trigger an end of period earlier than expected.

- **WDV: Watchdog Counter Value**

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

- **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.

- **WDD: Watchdog Delta Value**

Defines the permitted range for reloading the RSWDT.

If the RSWDT value is less than or equal to WDD, writing RSWDT_CR with WDRSTT = 1 restarts the timer.

If the RSWDT value is greater than WDD, writing RSWDT_CR with WDRSTT = 1 causes a Watchdog error.

- **WDDBGHLT: Watchdog Debug Halt**

0: The RSWDT runs when the processor is in debug state.

1: The RSWDT stops when the processor is in debug state.

- **WDIDLEHLT: Watchdog Idle Halt**

0: The RSWDT runs when the system is in Idle mode.

1: The RSWDT stops when the system is in idle state.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

The status of lock bits can be returned by the EEFC. The ‘Get Lock Bit’ sequence is the following:

1. Execute the ‘Get Lock Bit’ command by writing EEFC_FCR.FCMD with the GLB command. Field EEFC_FCR.FARG is meaningless.
2. Lock bits can be read by the software application in EEFC_FRR. The first word read corresponds to the 32 first lock bits, next reads providing the next 32 lock bits as long as it is meaningful. Extra reads to EEFC_FRR return 0.

For example, if the third bit of the first word read in EEFC_FRR is set, the third lock region is locked.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

Note: Access to the Flash in read is permitted when a ‘Set Lock Bit’, ‘Clear Lock Bit’ or ‘Get Lock Bit’ command is executed.

22.4.3.5 GPNVM Bit

GPNVM bits do not interfere with the embedded Flash memory plane. Refer to Section 8. ”Memories” for more details.

The ‘Set GPNVM Bit’ sequence is the following:

1. Execute the ‘Set GPNVM Bit’ command by writing EEFC_FCR.FCMD with the SGPB command and EEFC_FCR.FARG with the number of GPNVM bits to be set.
2. When the GPNVM bit is set, the bit EEFC_FSR.FRDY rises. If an interrupt was enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.
3. The result of the SGPB command can be checked by running a ‘Get GPNVM Bit’ (GGPB) command.

Note: The value of the FARG argument passed together with SGPB command must not exceed the higher GPNVM index available in the product. Flash data content is not altered if FARG exceeds the limit. Command Error is detected only if FARG is greater than 8.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

It is possible to clear GPNVM bits previously set. The ‘Clear GPNVM Bit’ sequence is the following:

1. Execute the ‘Clear GPNVM Bit’ command by writing EEFC_FCR.FCMD with the CGPB command and EEFC_FCR.FARG with the number of GPNVM bits to be cleared.
2. When the clear completes, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Note: The value of the FARG argument passed together with CGPB command must not exceed the higher GPNVM index available in the product. Flash data content is not altered if FARG exceeds the limit. Command Error is detected only if FARG is greater than 8.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

All NWE and NCS (write) timings are defined separately for each chip select as an integer number of Master Clock cycles. To ensure that the NWE and NCS timings are consistent, the user must define the total write cycle instead of the hold timing. This implicitly defines the NWE hold time and NCS (write) hold times as:

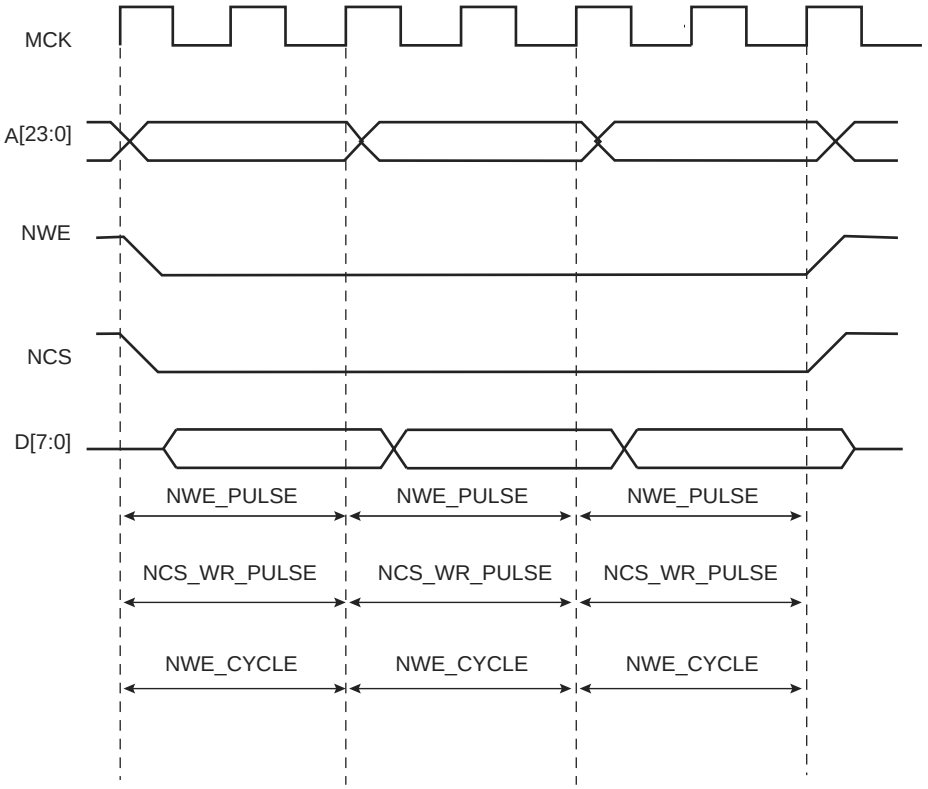
NWE_HOLD = NWE_CYCLE - NWE_SETUP - NWE_PULSE

NCS_WR_HOLD = NWE_CYCLE - NCS_WR_SETUP - NCS_WR_PULSE

27.9.3.4 Null Delay Setup and Hold

If null setup parameters are programmed for NWE and/or NCS, NWE and/or NCS remain active continuously in case of consecutive write cycles in the same memory (see Figure 27-12). However, for devices that perform write operations on the rising edge of NWE or NCS, such as SRAM, either a setup or a hold must be programmed.

Figure 27-12. Null Setup and Hold Values of NCS and NWE in Write Cycle



27.9.3.5 Null Pulse

Programming null pulse is not permitted. Pulse must be at least set to 1. A null value leads to unpredictable behavior.

27.9.4 Write Mode

The bit WRITE_MODE in the SMC_MODE register of the corresponding chip select indicates which signal controls the write operation.

27.9.4.1 Write is Controlled by NWE (WRITE_MODE = 1):

Figure 27-13 shows the waveforms of a write operation with WRITE_MODE set . The data is put on the bus during the pulse and hold steps of the NWE signal. The internal data buffers are switched to Output mode after the NWE_SETUP time, and until the end of the write cycle, regardless of the programmed waveform on NCS.

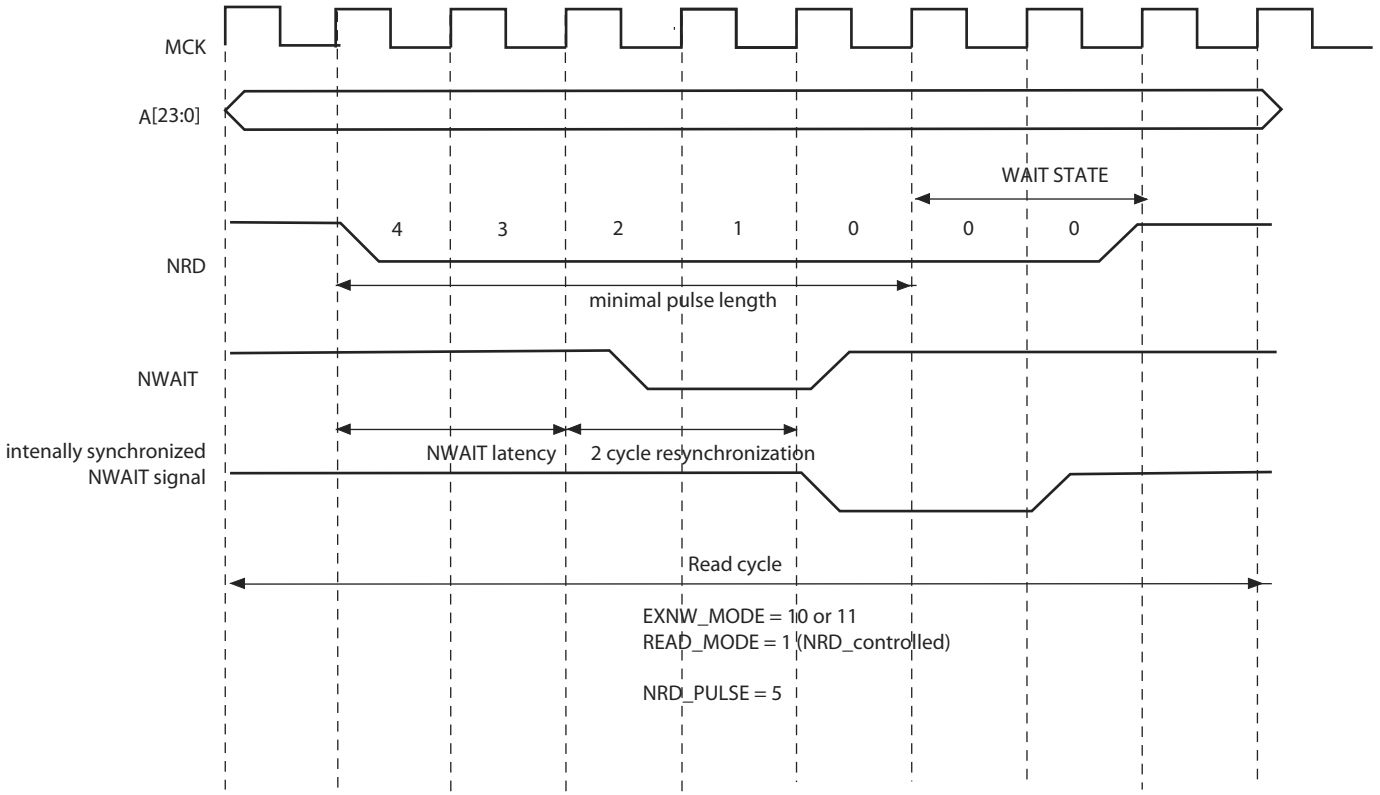
27.13.4 NWAIT Latency and Read/Write Timings

There may be a latency between the assertion of the read/write controlling signal and the assertion of the NWAIT signal by the device. The programmed pulse length of the read/write controlling signal must be at least equal to this latency plus the 2 cycles of resynchronization + one cycle. Otherwise, the SMC may enter the hold state of the access without detecting the NWAIT signal assertion. This is true in Frozen mode as well as in Ready mode. This is illustrated on Figure 27-29.

When EXNW_MODE is enabled (ready or frozen), the user must program a pulse length of the read and write controlling signal of at least:

minimal pulse length = NWAIT latency + 2 resynchronization cycles + one cycle

Figure 27-29. NWAIT Latency



32.6.8 PIO Input Filter Disable Register

Name: PIO_IFDR

Address: 0x400E0E24 (PIOA), 0x400E1024 (PIOB), 0x4800C024 (PIOC), 0x400E1224 (PIOD)

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

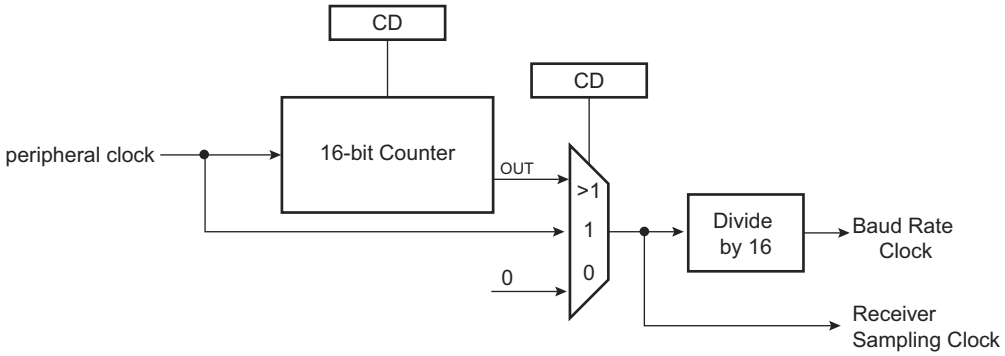
This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Input Filter Disable**

0: No effect.

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

Figure 35-2. Baud Rate Generator



35.5.2 Receiver

35.5.2.1 Receiver Reset, Enable and Disable

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

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

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

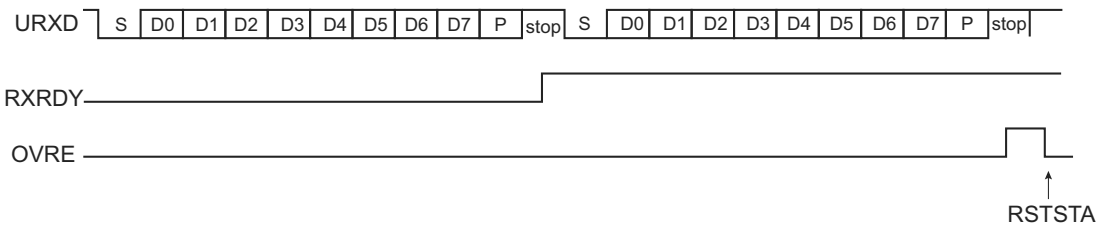
35.5.2.2 Start Detection and Data Sampling

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

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

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

Figure 35-3. Start Bit Detection



37.7.11 TC Interrupt Disable Register

Name: TC_IDRx [x=0..2]

Address: 0x40010028 (0)[0], 0x40010068 (0)[1], 0x400100A8 (0)[2], 0x40014028 (1)[0], 0x40014068 (1)[1], 0x400140A8 (1)[2]

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
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0: No effect.

1: Disables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0: No effect.

1: Disables the Load Overrun Interrupt (if TC_CMRx.WAVE = 0).

- **CPAS: RA Compare**

0: No effect.

1: Disables the RA Compare Interrupt (if TC_CMRx.WAVE = 1).

- **CPBS: RB Compare**

0: No effect.

1: Disables the RB Compare Interrupt (if TC_CMRx.WAVE = 1).

- **CPCS: RC Compare**

0: No effect.

1: Disables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0: No effect.

1: Disables the RA Load Interrupt (if TC_CMRx.WAVE = 0).

- **LDRBS: RB Loading**

0: No effect.

1: Disables the RB Load Interrupt (if TC_CMRx.WAVE = 0).

43. Classical Public Key Cryptography Controller (CPKCC)

43.1 Description

The Classical Public Key Cryptography Controller (CPKCC) is an Atmel macrocell that processes public key cryptography algorithm calculus in both GF(p) and GF(2^n) fields. The ROMed CPKCL, the Classical Public Key Cryptography Library, is the library built on the top of the CPKCC.

The Classical Public Key Cryptography Library includes complete implementation of the following public key cryptography algorithms:

- RSA, DSA:
  - Modular Exponentiation with CRT up to 6144 bits
  - Modular Exponentiation without CRT up to 5408 bits
  - Prime generation
  - Utilities: GCD/modular Inverse, Divide, Modular reduction, Multiply, …
- Elliptic Curves:
  - ECDSA up to 1504 bits
  - Point Multiply,
  - Point Add/Doubling
  - Elliptic Curves in GF(p) or GF(2^n)
  - Choice of the curves parameters so compatibility with NIST Curves or others.
- Deterministic Random Number Generation (DRNG ANSI X9.31) for DSA

45.4.3 Interrupts

The USBFS interrupt request line is connected to the interrupt controller. Using the USBFS interrupt requires the interrupt controller to be programmed first.

Table 45-3. Peripheral IDs

Instance	ID
USBFS	22

45.4.4 USB Pipe/Endpoint x FIFO Data Register (USBFIFOxDATA)

The application has access to each pipe/endpoint FIFO through its reserved 32 KB address space. The application can access a 64-KB buffer linearly or fixedly as the DPRAM address increment is fully handled by hardware. Byte, half-word and word accesses are supported. Data should be accessed in a big-endian way.

Disabling the USBFS (by writing a zero to the USBFS_CTRL.USBE bit) does not reset the DPRAM.

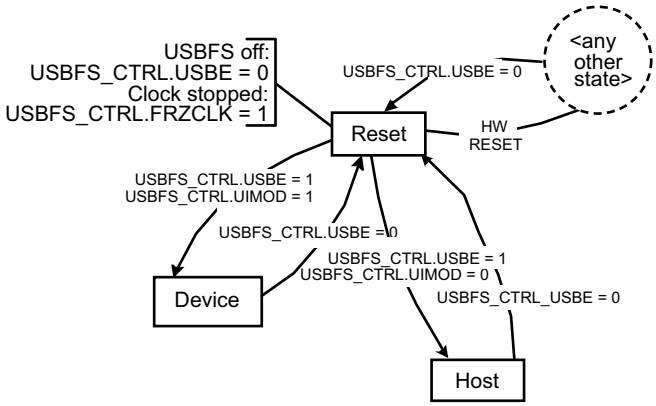
45.5 Functional Description

45.5.1 USB General Operation

45.5.1.1 Power-On and Reset

Figure 45-5 describes the USBFS general states.

Figure 45-5. General States



After a hardware reset, the USBFS is in the Reset state. In this state:

- The USBFS is disabled. The USBFS Enable bit in the General Control register (USBFS_CTRL.USBE) is zero.
- The USBFS clock is stopped in order to minimize the power consumption. The Freeze USB Clock bit (USBFS_CTRL.FRZCLK) is set.
- The transceiver is in Suspend mode.
- The internal states and registers of the Device and Host modes are reset.
- The DPRAM is not cleared and is accessible.

After writing a one to USBFS_CTRL.USBE, the USBFS enters the Device or the Host mode in idle state.

The USBFS can be disabled at any time by writing a zero to USBFS_CTRL.USBE. In fact, writing a zero to USBFS_CTRL.USBE acts as a hardware reset, except that the USBFS_CTRL.FRZCLK and USBFS_CTRL.UIMOD bits are not reset.

Figure 45-17. Example of an OUT Endpoint with one Data Bank

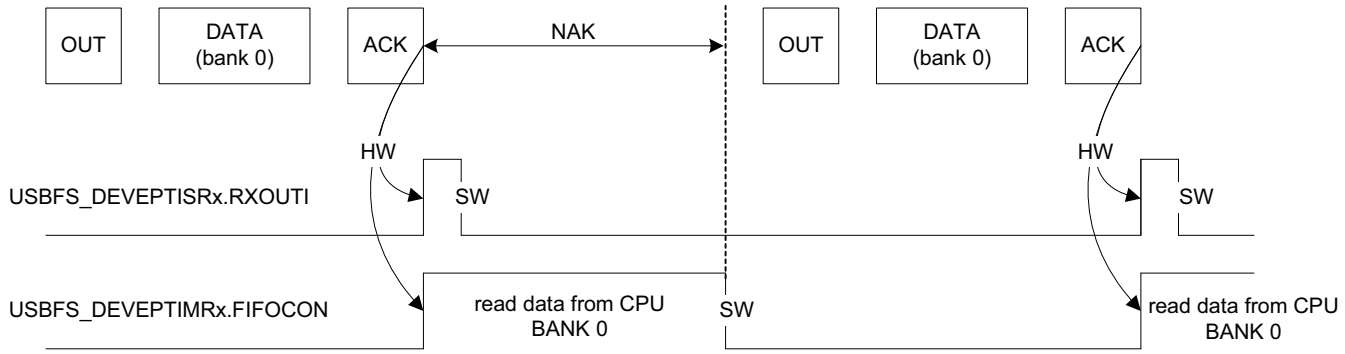
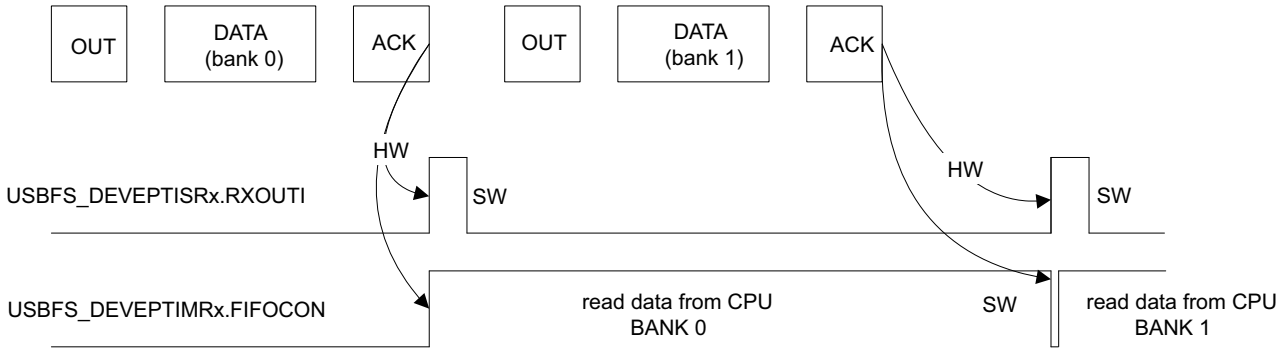


Figure 45-18. Example of an OUT Endpoint with two Data Banks



Detailed Description

The data is read as follows:

- When the bank is full, USBFS_DEVEPTISRx.RXOUTI and USBFS_DEVEPTIMRx.FIFOCON are set, which triggers a PEP_x interrupt if USBFS_DEVEPTIMRx.RXOUTE = 1.
- The user acknowledges the interrupt by writing a one to USBFS_DEVEPTICRx.RXOUTIC in order to clear USBFS_DEVEPTISRx.RXOUTI.
- The user can read the byte count of the current bank from USBFS_DEVEPTISRx.BYCT to know how many bytes to read, rather than polling USBFS_DEVEPTISRx.RWALL.
- The user reads the data from the current bank by using the USBFIFOnDATA register, until all the expected data frame is read or the bank is empty (in which case USBFS_DEVEPTISRx.RWALL is cleared and USBFS_DEVEPTISRx.BYCT reaches zero).
- The user frees the bank and switches to the next bank (if any) by clearing USBFS_DEVEPTIMRx.FIFOCON.

If the endpoint uses several banks, the current one can be read while the following one is being written by the host. Then, when the user clears USBFS_DEVEPTIMRx.FIFOCON, the following bank can already be read and USBFS_DEVEPTISRx.RXOUTI is set immediately.

45.5.2.14 Underflow

Underflow errors exist only for isochronous IN/OUT endpoints. An underflow error sets the Underflow Interrupt (USBFS_DEVEPTISRx.UNDERFI) bit, which triggers a PEP_x interrupt if the Underflow Interrupt Enable (USBFS_DEVEPTIMRx.UNDERFE) bit is one.

- An underflow can occur during the IN stage if the host attempts to read from an empty bank. A zero-length packet is then automatically sent by the USBFS.

45.6.9 Device Global Interrupt Mask Register

Name: USBFS_DEVIMR

Address: 0x40020010

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	DMA_4	DMA_3	DMA_2	DMA_1	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	PEP_4
15	14	13	12	11	10	9	8
PEP_3	PEP_2	PEP_1	PEP_0	–	–	–	–
7	6	5	4	3	2	1	0
–	UPRSME	EORSME	WAKEUPE	EORSTE	SOFE	–	SUSPE

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

0: The corresponding interrupt is not enabled.

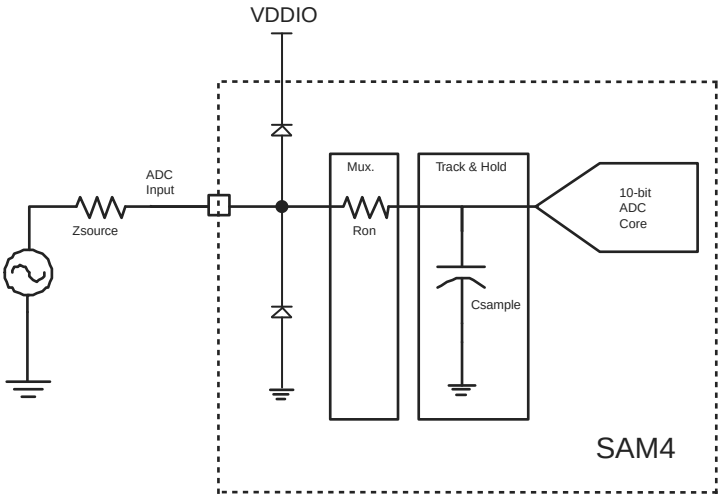
1: The corresponding interrupt is enabled.

- **SUSPE: Suspend Interrupt Mask**
- **SOFE: Start of Frame Interrupt Mask**
- **EORSTE: End of Reset Interrupt Mask**
- **WAKEUPE: Wake-Up Interrupt Mask**
- **EORSME: End of Resume Interrupt Mask**
- **UPRSME: Upstream Resume Interrupt Mask**
- **PEP_x: Endpoint x Interrupt Mask**
- **DMA_x: DMA Channel x Interrupt Mask**

46.5.17.1 Track and Hold Time versus Source Output Impedance, Effective Sampling Rate

The following figure gives a simplified view of the acquisition path.

Figure 46-18. Simplified Acquisition Path



During its tracking phase, the 10-bit ADC charges its sampling capacitor through various serial resistors: source output resistor, multiplexer series resistor and the sampling switch series resistor. In case of high output source resistance (low power resistive divider, for example), the track time must be increased to ensure full settling of the sampling capacitor voltage. The following formulas give the minimum track time that guarantees a 10-bit accurate settling:

- V_{DDIN} > 3.0V: t_{TRACK} (ns) = 0.12 x $R_{SOURCE}(\Omega)$ + 500
- $V_{DDIN} \leq$ 3.0V: t_{TRACK} (ns) = 0.12 x $R_{SOURCE}(\Omega)$ + 1000

According to the calculated track time (t_{TRACK}), the actual track time of the ADC must be adjusted through the TRACKTIM field in the ADC_MR register. TRACKTIM is obtained by the following formula:

$$TRACKTIM = floor\left(\frac{T_{TRACK}}{T_{CK\_ADC}}\right)$$

with $t_{CK\_ADC} = 1 / f_{CK\_ADC}$ and floor(x) the mathematical function that rounds x to the greatest previous integer.

The actual conversion time of the converter is obtained by the following formula:

$$T_{CONV} = (TRACKTIM + 24) \times T_{CK\_ADC}$$

When converting in Free Run mode, the actual sampling rate of the converter is ($1 / T_{CONV}$) or as defined by the following formula:

$$F_S = \frac{F_{CK\_ADC}}{(TRACKTIM + 24)}$$

The maximum source resistance with the actual TRACKTIM setting is:

- $R_{SOURCE\_MAX}(\Omega)$ = ((TRACKTIM + 1) x t_{CK_ADC}(ns) - 500) / 0.12 for V_{DDIN} > 3.0V; or
- $R_{SOURCE\_MAX}(\Omega)$ = ((TRACKTIM + 1) x t_{CK_ADC}(ns) - 1000) / 0.12 for $V_{DDIN} \leq$ 3.0V