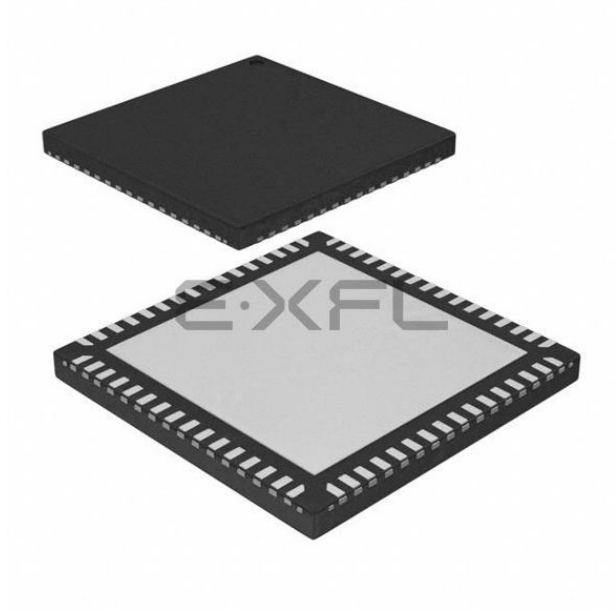


Welcome to E-XFL.COM

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	I²C, IrDA, Memory Card, SPI, SSC, UART/USART, USB
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	47
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 11x12b; D/A 2x12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	64-VFQFN Exposed Pad
Supplier Device Package	64-QFN (9x9)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atsam4s8bb-mnr

- For all other instructions that use labels, the value of the PC is the address of the current instruction plus 4 bytes, with bit[1] of the result cleared to 0 to make it word-aligned.
- Your assembler might permit other syntaxes for PC-relative expressions, such as a label plus or minus a number, or an expression of the form [PC, #number].

12.6.3.7 Conditional Execution

Most data processing instructions can optionally update the condition flags in the *Application Program Status Register* (APSR) according to the result of the operation, see “Application Program Status Register” . Some instructions update all flags, and some only update a subset. If a flag is not updated, the original value is preserved. See the instruction descriptions for the flags they affect.

An instruction can be executed conditionally, based on the condition flags set in another instruction, either:

- Immediately after the instruction that updated the flags
- After any number of intervening instructions that have not updated the flags.

Conditional execution is available by using conditional branches or by adding condition code suffixes to instructions. See Table 12-16 for a list of the suffixes to add to instructions to make them conditional instructions. The condition code suffix enables the processor to test a condition based on the flags. If the condition test of a conditional instruction fails, the instruction:

- Does not execute
- Does not write any value to its destination register
- Does not affect any of the flags
- Does not generate any exception.

Conditional instructions, except for conditional branches, must be inside an If-Then instruction block. See “IT” for more information and restrictions when using the IT instruction. Depending on the vendor, the assembler might automatically insert an IT instruction if there are conditional instructions outside the IT block.

The CBZ and CBNZ instructions are used to compare the value of a register against zero and branch on the result.

This section describes:

- “Condition Flags”
- “Condition Code Suffixes” .

Condition Flags

The APSR contains the following condition flags:

N	Set to 1 when the result of the operation was negative, cleared to 0 otherwise.
Z	Set to 1 when the result of the operation was zero, cleared to 0 otherwise.
C	Set to 1 when the operation resulted in a carry, cleared to 0 otherwise.
V	Set to 1 when the operation caused overflow, cleared to 0 otherwise.

For more information about the APSR, see “Program Status Register” .

A carry occurs:

- If the result of an addition is greater than or equal to 2^{32}
- If the result of a subtraction is positive or zero
- As the result of an inline barrel shifter operation in a move or logical instruction.

An overflow occurs when the sign of the result, in bit[31], does not match the sign of the result, had the operation been performed at infinite precision, for example:

- If adding two negative values results in a positive value
- If adding two positive values results in a negative value
- If subtracting a positive value from a negative value generates a positive value

12.6.5.9 SADD16 and SADD8

Signed Add 16 and Signed Add 8

Syntax

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

where:

op is any of:

SADD16 Performs two 16-bit signed integer additions.

SADD8 Performs four 8-bit signed 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 perform a halfword or byte add in parallel:

The SADD16 instruction:

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

The SADD8 instruction:

1. Adds each byte of the first operand to the corresponding byte of the second operand.

Writes the result in the corresponding bytes of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SADD16 R1, R0     ; Adds the halfwords in R0 to the corresponding
                  ; halfwords of R1 and writes to corresponding halfword
                  ; of R1.
SADD8  R4, R0, R5 ; Adds bytes of R0 to the corresponding byte in R5 and
                  ; writes to the corresponding byte in R4.
```

The table below shows the encodings for the TEX, C, B, and S access permission bits.

Table 12-36. TEX, C, B, and S Encoding

TEX	C	B	S	Memory Type	Shareability	Other Attributes
b000	0	0	x(1)	Strongly-ordered	Shareable	–
		1	x(1)	Device	Shareable	–
	1	0	0	Normal	Not shareable	Outer and inner write-through. No write allocate.
			1		Shareable	
		1	0	Normal	Not shareable	Outer and inner write-back. No write allocate.
			1		Shareable	
b001	0	0	0	Normal	Not shareable	Outer and inner noncacheable.
			1		Shareable	
		1	x(1)	Reserved encoding		–
	1	0	x(1)	Implementation defined attributes.		–
		1	0	Normal	Not shareable	Outer and inner write-back. Write and read allocate.
			1		Shareable	
b010	0	0	x(1)	Device	Not shareable	Nonshared Device.
		1	x(1)	Reserved encoding		–
	1	x(1)	x(1)	Reserved encoding		–
b1BB	A	A	0	Normal	Not shareable	Cached memory BB = outer policy, AA = inner policy.
			1		Shareable	

Note: 1. The MPU ignores the value of this bit.

Table 12-37 shows the cache policy for memory attribute encodings with a TEX value is in the range 4–7.

Table 12-37. Cache Policy for Memory Attribute Encoding

Encoding, AA or BB	Corresponding Cache Policy
00	Non-cacheable
01	Write back, write and read allocate
10	Write through, no write allocate
11	Write back, no write allocate

12.11.2.9 MPU Region Attribute and Size Register Alias 2

Name: MPU_RASR_A2

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 12-38.

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

See Table 12-36.

- **S: Shareable**

See Table 12-36.

- **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.

18. Supply Controller (SUPC)

18.1 Description

The Supply Controller (SUPC) controls the supply voltages of the system and manages the Backup mode. In this mode, current consumption is reduced to a few microamps for backup power retention. Exit from this mode is possible on multiple wake-up sources. The SUPC also generates the slow clock by selecting either the low-power RC oscillator or the low-power crystal oscillator.

18.2 Embedded Characteristics

- Manages the Core Power Supply VDDCORE and Backup Mode by Controlling the Embedded Voltage Regulator
- A Supply Monitor Detection on VDDIO or a Brownout Detection on VDDCORE Triggers a Core Reset
- Generates the Slow Clock SLCK by Selecting Either the 22-42 kHz Low-Power RC Oscillator or the 32 kHz Low-Power Crystal Oscillator
- Low-power Tamper Detection on Two Inputs
- Anti-tampering by Immediate Clear of the General-purpose Backup Registers
- Supports Multiple Wake-up Sources for Exit from Backup Mode
  - 16 Wake-up Inputs with Programmable Debouncing
  - Real-Time Clock Alarm
  - Real-Time Timer Alarm
  - Supply Monitor Detection on VDDIO, with Programmable Scan Period and Voltage Threshold

The supply monitor can also be enabled during one slow clock period on every one of either 32, 256 or 2048 slow clock periods, depending on the user selection. This is configured in the SMSMPL field in SUPC_SMMR.

Enabling the supply monitor for such reduced times divides the typical supply monitor power consumption by factors of 2, 16 and 128, respectively, if continuous monitoring of the VDDIO power supply is not required.

A supply monitor detection generates either a reset of the core power supply or a wake-up of the core power supply. Generating a core reset when a supply monitor detection occurs is enabled by setting the SMRSTEN bit in SUPC_SMMR.

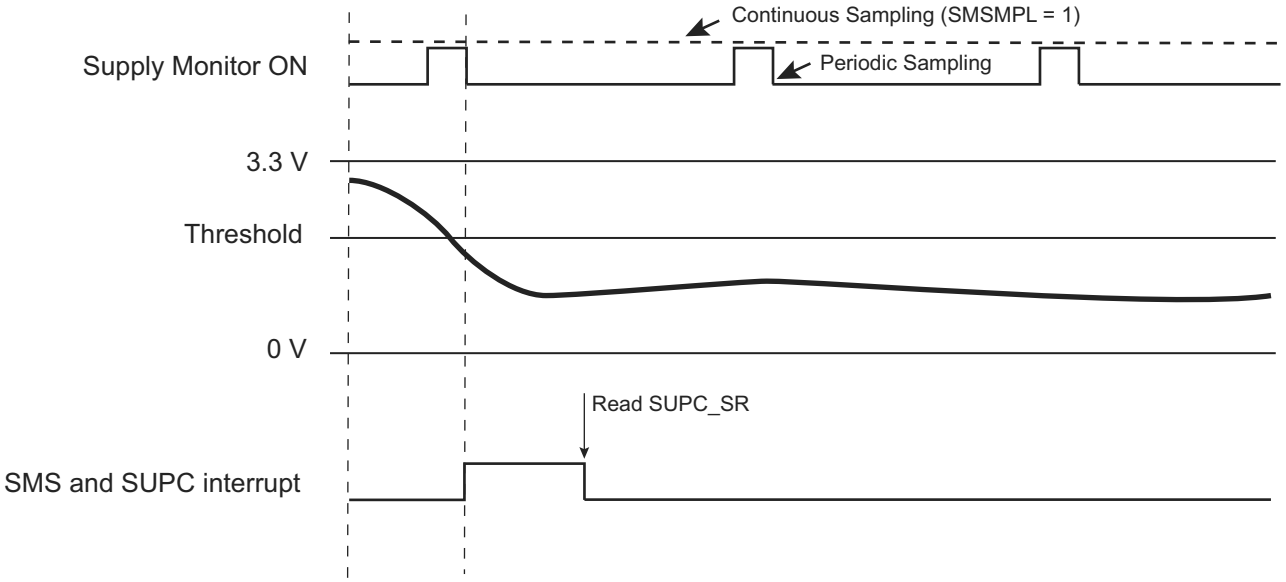
Waking up the core power supply when a supply monitor detection occurs can be enabled by setting the SMEN bit in the Wake-up Mode register (SUPC_WUMR).

The SUPC provides two status bits in the SUPC_SR for the supply monitor that determine whether the last wake-up was due to the supply monitor:

- The SMOS bit provides real-time information, updated at each measurement cycle or updated at each slow clock cycle, if the measurement is continuous.
- The SMS bit provides saved information and shows a supply monitor detection has occurred since the last read of SUPC_SR.

The SMS flag generates an interrupt if the SMIEN bit is set in SUPC_SMMR.

Figure 18-2. Supply Monitor Status Bit and Associated Interrupt



18.4.5 Backup Power Supply Reset

18.4.5.1 Raising the Backup Power Supply

When the backup voltage VDDIO rises, the RC oscillator is powered up and the zero-power power-on reset cell maintains its output low as long as VDDIO has not reached its target voltage. During this period, the SUPC is reset. When the VDDIO voltage becomes valid and the zero-power power-on reset signal is released, a counter is started for five slow clock cycles. This is the time required for the 32 kHz RC oscillator to stabilize.

After this time, the voltage regulator is enabled. The core power supply rises and the brownout detector provides the bodcore_in signal as soon as the core voltage VDDCORE is valid. This results in releasing the vddcore_nreset signal to the Reset Controller after the bodcore_in signal has been confirmed as being valid for at least one slow clock cycle.

Table 20-3. Flash Descriptor Definition

Symbol	Word Index	Description
FL_ID	0	Flash interface description
FL_SIZE	1	Flash size in bytes
FL_PAGE_SIZE	2	Page size in bytes
FL_NB_PLANE	3	Number of planes
FL_PLANE[0]	4	Number of bytes in the plane
FL_NB_LOCK	4 + FL_NB_PLANE	Number of lock bits. A bit is associated with a lock region. A lock bit is used to prevent write or erase operations in the lock region.
FL_LOCK[0]	4 + FL_NB_PLANE + 1	Number of bytes in the first lock region

20.4.3.2 Write Commands

Several commands are used to program the Flash.

Only 0 values can be programmed using Flash technology; 1 is the erased value. In order to program words in a page, the page must first be erased. Commands are available to erase the full memory plane or a given number of pages. With the EWP and EWPL commands, a page erase is done automatically before a page programming.

After programming, the page (the entire lock region) can be locked to prevent miscellaneous write or erase sequences. The lock bit can be automatically set after page programming using WPL or EWPL commands.

Data to be programmed in the Flash must be written in an internal latch buffer before writing the programming command in EEFC_FCR. Data can be written at their final destination address, as the latch buffer is mapped into the Flash memory address space and wraps around within this Flash address space.

Byte and half-word AHB accesses to the latch buffer are not allowed. Only 32-bit word accesses are supported.

32-bit words must be written continuously, in either ascending or descending order. Writing the latch buffer in a random order is not permitted. This prevents mapping a C-code structure to the latch buffer and accessing the data of the structure in any order. It is instead recommended to fill in a C-code structure in SRAM and copy it in the latch buffer in a continuous order.

Write operations in the latch buffer are performed with the number of wait states programmed for reading the Flash.

The latch buffer is automatically re-initialized, i.e., written with logical 1, after execution of each programming command. However, after power-up, the latch buffer is not initialized. If only part of the page is to be written with user data, the remaining part must be erased (written with 1).

The programming sequence is the following:

1. Write the data to be programmed in the latch buffer.
2. Write the programming command in EEFC_FCR. This automatically clears the bit EEFC_FSR.FRDY.
3. When Flash programming is completed, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the EEFC is activated.

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

- Command Error: A bad keyword has been written in EEFC_FCR.
- Lock Error: The page to be programmed belongs to a locked region. A command must be run previously to unlock the corresponding region.
- Flash Error: When programming is completed, the WriteVerify test of the Flash memory has failed.

Only one page can be programmed at a time. It is possible to program all the bits of a page (full page programming) or only some of the bits of the page (partial page programming).

21.3.5 Device Operations

Several commands on the Flash memory are available. These commands are summarized in Table 21-3. Each command is driven by the programmer through the parallel interface running several read/write handshaking sequences.

When a new command is executed, the previous one is automatically achieved. Thus, chaining a read command after a write automatically flushes the load buffer in the Flash.

21.3.5.1 Flash Read Command

This command is used to read the contents of the Flash memory. The read command can start at any valid address in the memory plane and is optimized for consecutive reads. Read handshaking can be chained; an internal address buffer is automatically increased.

Table 21-6. Read Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	READ
2	Write handshaking	ADDR0	Memory Address LSB
3	Write handshaking	ADDR1	Memory Address
4	Read handshaking	DATA	*Memory Address++
5	Read handshaking	DATA	*Memory Address++
...	...	...	...
n	Write handshaking	ADDR0	Memory Address LSB
n+1	Write handshaking	ADDR1	Memory Address
n+2	Read handshaking	DATA	*Memory Address++
n+3	Read handshaking	DATA	*Memory Address++
...	...	...	...

21.3.5.2 Flash Write Command

This command is used to write the Flash contents.

The Flash memory plane is organized into several pages. Data to be written are stored in a load buffer that corresponds to a Flash memory page. The load buffer is automatically flushed to the Flash:

- before access to any page other than the current one
- when a new command is validated (MODE = CMDE)

The **Write Page** command **(WP)** is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

Table 21-7. Write Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	WP or WPL or EWP or EWPL
2	Write handshaking	ADDR0	Memory Address LSB
3	Write handshaking	ADDR1	Memory Address
4	Write handshaking	DATA	*Memory Address++
5	Write handshaking	DATA	*Memory Address++
...	...	...	...
n	Write handshaking	ADDR0	Memory Address LSB

23.5 CRCCU Functional Description

23.5.1 CRC Calculation Unit

The CRCCU integrates a dedicated Cyclic Redundancy Check (CRC) engine. When configured and activated, this CRC engine performs a checksum computation on a memory area. CRC computation is performed from the LSB to MSB. Three different polynomials are available: CCITT802.3, CASTAGNOLI and CCITT16 (see field description “PTYPE: Primitive Polynomial” in Section 23.7.10 “CRCCU Mode Register” for details).

23.5.2 CRC Calculation Unit Operation

The CRCCU has a DMA controller that supports programmable CRC memory checks. When enabled, the DMA channel reads a programmable amount of data and computes CRC on the fly.

The CRCCU is controlled by two registers, TR_ADDR and TR_CTRL, which need to be mapped in the internal SRAM. The addresses of these two registers are pointed to by the CRCCU_DSCR.

Table 23-1. CRCCU Descriptor Memory Mapping

		SRAM Memory
CRCCU_DSCR+0x0	---->	TR_ADDR
CRCCU_DSCR+0x4	---->	TR_CTRL
CRCCU_DSCR+0x8	---->	Reserved
CRCCU_DSCR+0xC	---->	Reserved
CRCCU_DSCR+0x10	---->	TR_CRC

TR_ADDR defines the start address of memory area targeted for CRC calculation.

TR_CTRL defines the buffer transfer size, the transfer width (byte, halfword, word) and the transfer-completed interrupt enable.

To start the CRCCU, set the CRC enable bit (ENABLE) and configure the mode of operation in the CRCCU Mode Register (CRCCU_MR), then configure the Transfer Control Registers and finally, set the DMA enable bit (DMAEN) in the CRCCU DMA Enable Register (CRCCU_DMA_EN).

When the CRCCU is enabled, the CRCCU reads the predefined amount of data (defined in TR_CTRL) located from TR_ADDR start address and computes the checksum.

The CRCCU_SR contains the temporary CRC value.

The BTSIZE field located in the TR_CTRL register (located in memory), is automatically decremented if its value is different from zero. Once the value of the BTSIZE field is equal to zero, the CRCCU is disabled by hardware. In this case, the relevant CRCCU DMA Status Register bit DMASR is automatically cleared.

If the COMPARE field of the CRCCU_MR is set to true, the TR_CRC (Transfer Reference Register) is compared with the last CRC computed. If a mismatch occurs, an error flag is set and an interrupt is raised (if unmasked).

The CRCCU accesses the memory by single access (TRWIDTH size) in order not to limit the bandwidth usage of the system, but the DIVIDER field of the CRCCU Mode Register can be used to lower it by dividing the frequency of the single accesses.

The CRCCU scrolls the defined memory area using ascending addresses.

In order to compute the CRC for a memory size larger than 256 Kbytes or for non-contiguous memory area, it is possible to re-enable the CRCCU on the new memory area and the CRC will be updated accordingly. Use the RESET field of the CRCCU_CR to reset the CRCCU Status Register to its default value (0xFFFFFFFF).

31.6.31 PIO Pad Pull-Down Enable Register

Name: PIO_PPDER

Address: 0x400E0E94 (PIOA), 0x400E1094 (PIOB), 0x400E1294 (PIOC)

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: Pull-Down Enable**

0: No effect.

1: Enables the pull-down resistor on the I/O line.

31.6.53 PIO Parallel Capture Interrupt Status Register

Name: PIO_PCISR

Address: 0x400E0F60 (PIOA), 0x400E1160 (PIOB), 0x400E1360 (PIOC)

Access: Read-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
–	–	–	–	RXBUFF	ENDRX	OVRE	DRDY

- **DRDY: Parallel Capture Mode Data Ready**

0: No new data is ready to be read since the last read of PIO_PCRHR.

1: A new data is ready to be read since the last read of PIO_PCRHR.

The DRDY flag is automatically reset when PIO_PCRHR is read or when the parallel capture mode is disabled.

- **OVRE: Parallel Capture Mode Overrun Error**

0: No overrun error occurred since the last read of this register.

1: At least one overrun error occurred since the last read of this register.

The OVRE flag is automatically reset when this register is read or when the parallel capture mode is disabled.

- **ENDRX: End of Reception Transfer**

0: The End of Transfer signal from the reception PDC channel is inactive.

1: The End of Transfer signal from the reception PDC channel is active.

- **RXBUFF: Reception Buffer Full**

0: The signal Buffer Full from the reception PDC channel is inactive.

1: The signal Buffer Full from the reception PDC channel is active.

- **START: Transmit Start Selection**

Value	Name	Description
0	CONTINUOUS	Continuous, as soon as a word is written in the SSC_THR (if Transmit is enabled), and immediately after the end of transfer of the previous data
1	RECEIVE	Receive start
2	TF_LOW	Detection of a low level on TF signal
3	TF_HIGH	Detection of a high level on TF signal
4	TF_FALLING	Detection of a falling edge on TF signal
5	TF_RISING	Detection of a rising edge on TF signal
6	TF_LEVEL	Detection of any level change on TF signal
7	TF_EDGE	Detection of any edge on TF signal

- **STTDLY: Transmit Start Delay**

If STTDLY is not 0, a delay of STTDLY clock cycles is inserted between the start event and the actual start of transmission of data. When the Transmitter is programmed to start synchronously with the Receiver, the delay is also applied.

Note: Note: STTDLY must be set carefully. If STTDLY is too short in respect to TAG (Transmit Sync Data) emission, data is emitted instead of the end of TAG.

- **PERIOD: Transmit Period Divider Selection**

This field selects the divider to apply to the selected Transmit Clock to generate a new Frame Sync Signal. If 0, no period signal is generated. If not 0, a period signal is generated at each 2 × (PERIOD + 1) Transmit Clock.

32.9.16 SSC Interrupt Mask Register

Name: SSC_IMR

Address: 0x4000404C

Access: Read-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
–	–	–	–	RXSYN	TXSYN	CP1	CP0

7	6	5	4	3	2	1	0
RXBUFF	ENDRX	OVRUN	RXRDY	TXBUFE	ENDTX	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready Interrupt Mask**

0: The Transmit Ready Interrupt is disabled.

1: The Transmit Ready Interrupt is enabled.

- **TXEMPTY: Transmit Empty Interrupt Mask**

0: The Transmit Empty Interrupt is disabled.

1: The Transmit Empty Interrupt is enabled.

- **ENDTX: End of Transmission Interrupt Mask**

0: The End of Transmission Interrupt is disabled.

1: The End of Transmission Interrupt is enabled.

- **TXBUFE: Transmit Buffer Empty Interrupt Mask**

0: The Transmit Buffer Empty Interrupt is disabled.

1: The Transmit Buffer Empty Interrupt is enabled.

- **RXRDY: Receive Ready Interrupt Mask**

0: The Receive Ready Interrupt is disabled.

1: The Receive Ready Interrupt is enabled.

- **OVRUN: Receive Overrun Interrupt Mask**

0: The Receive Overrun Interrupt is disabled.

1: The Receive Overrun Interrupt is enabled.

- **ENDRX: End of Reception Interrupt Mask**

0: The End of Reception Interrupt is disabled.

1: The End of Reception Interrupt is enabled.

33.7.5 Register Write Protection

To prevent any single software error from corrupting SPI behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the SPI Write Protection Mode Register (SPI_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the SPI Write Protection Status Register (SPI_WPSR) is set and the WPVSRC field indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading SPI_WPSR.

The following registers can be write-protected:

- SPI Mode Register
- SPI Chip Select Register

The BITS field determines the number of data bits transferred. Reserved values should not be used.

Value	Name	Description
0	8_BIT	8 bits for transfer
1	9_BIT	9 bits for transfer
2	10_BIT	10 bits for transfer
3	11_BIT	11 bits for transfer
4	12_BIT	12 bits for transfer
5	13_BIT	13 bits for transfer
6	14_BIT	14 bits for transfer
7	15_BIT	15 bits for transfer
8	16_BIT	16 bits for transfer
9	–	Reserved
10	–	Reserved
11	–	Reserved
12	–	Reserved
13	–	Reserved
14	–	Reserved
15	–	Reserved

- **SCBR: Serial Clock Bit Rate**

In Master mode, the SPI Interface uses a modulus counter to derive the SPCK bit rate from the peripheral clock. The bit rate is selected by writing a value from1 to 255 in the SCBR field. The following equation determines the SPCK bit rate:

SCBR = $f_{peripheral\ clock}$ / SPCK Bit Rate

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

If BRSRCCLK = 1 in SPI_MR, SCBR must be programmed with a value greater than 1.

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

Note: If one of the SCBR fields in SPI_CSRx is set to 1, the other SCBR fields in SPI_CSRx must be set to 1 as well, if they are used to process transfers. If they are not used to transfer data, they can be set at any value.

- **DLYBS: Delay Before SPCK**

This field defines the delay from NPCS falling edge (activation) to the first valid SPCK transition.

When DLYBS = 0, the delay is half the SPCK clock period.

Otherwise, the following equation determines the delay:

DLYBS = Delay Before SPCK × $f_{peripheral\ clock}$

- **DLYBCT: Delay Between Consecutive Transfers**

This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select. The delay is always inserted after each transfer and before removing the chip select if needed.

When DLYBCT = 0, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers.

Otherwise, the following equation determines the delay:

DLYBCT = Delay Between Consecutive Transfers × $f_{peripheral\ clock}$ / 32

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.

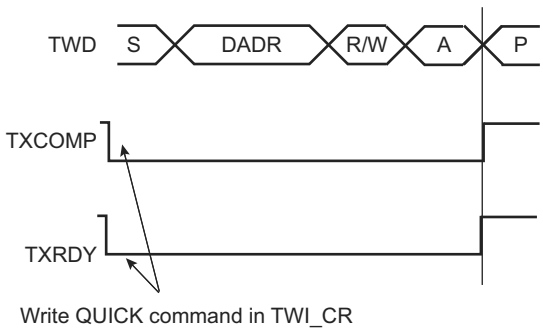
1. Initialize the receive PDC (memory pointers, transfer size - 2).
2. Configure the master (DADR, CKDIV, MREAD = 1, etc.)
3. Set the PDC RXTEN bit.
4. (Master Only) Write the START bit in the TWI_CR to start the transfer.
5. Wait for the PDC ENDRX Flag either by using polling method or ENDRX interrupt.
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.

34.7.3.8 SMBus Quick Command (Master Mode Only)

The TWI can perform a quick command:

1. Configure the Master mode (DADR, CKDIV, etc.).
2. Write the MREAD bit in the TWI_MMR at the value of the one-bit command to be sent.
3. Start the transfer by setting the QUICK bit in the TWI_CR.

Figure 34-13. SMBus Quick Command



34.7.3.9 Read/Write Flowcharts

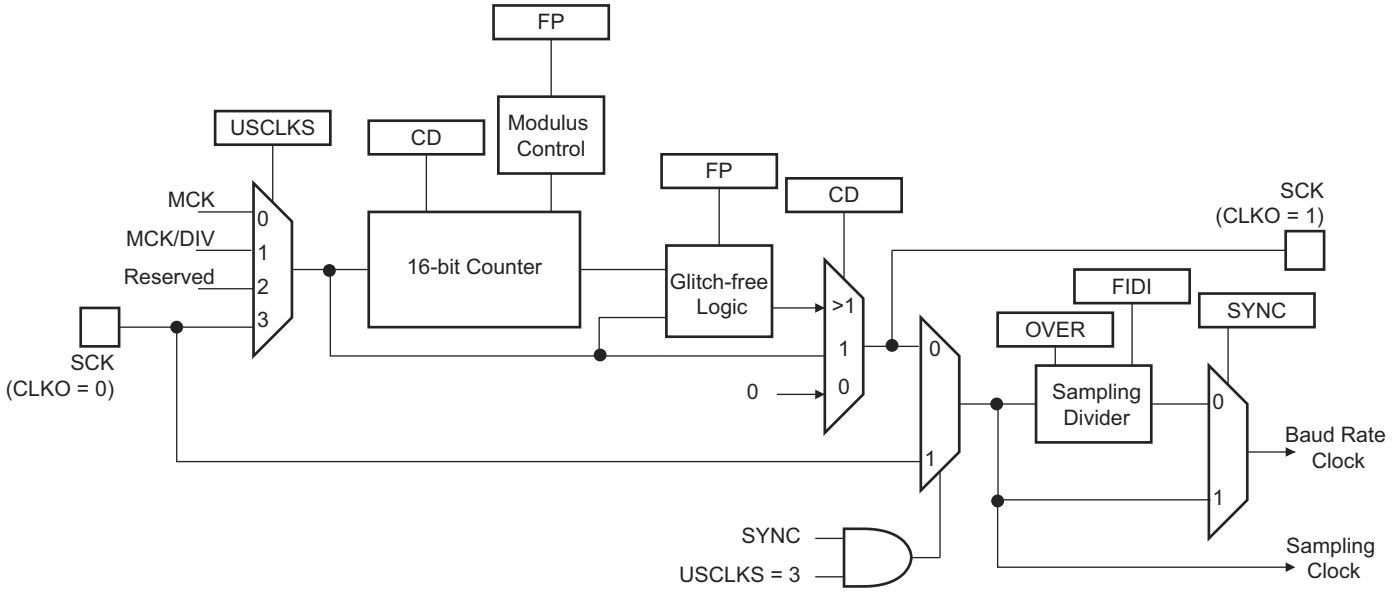
The flowcharts in the following figures provide examples of read and write operations. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the Interrupt Enable Register (TWI_IER) be configured first.

fractional part is activated. The resolution is one eighth of the clock divider. This feature is only available when using USART normal mode. The fractional baud rate is calculated using the following formula:

$$Baudrate = \frac{SelectedClock}{\left(8(2-Over)\left(CD + \frac{FP}{8}\right)\right)}$$

The modified architecture is presented in the following Figure 36-3.

Figure 36-3. Fractional Baud Rate Generator



36.6.1.3 Baud Rate in Synchronous Mode or SPI Mode

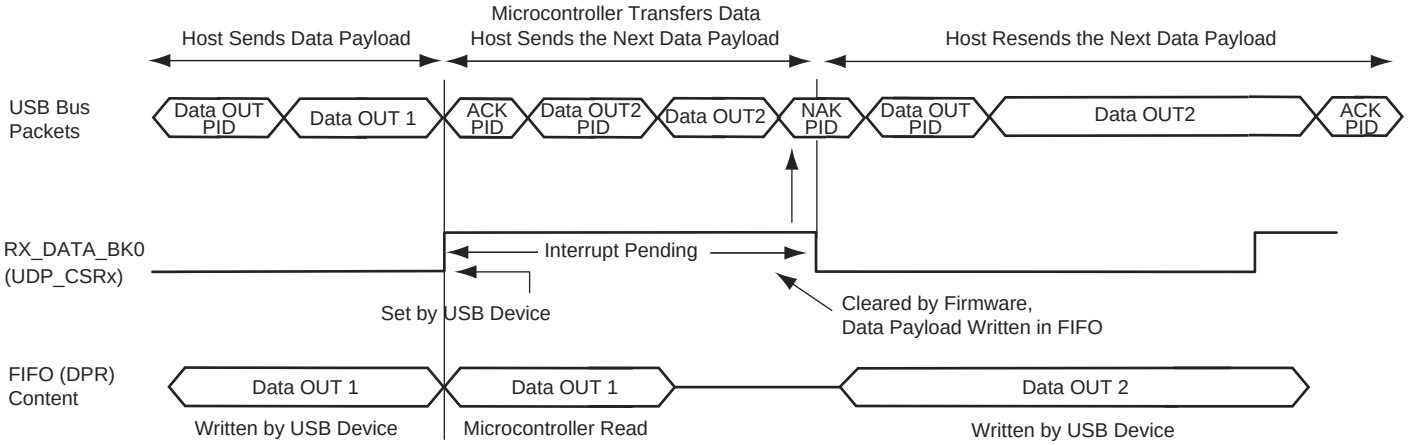
If the USART is programmed to operate in Synchronous mode, the selected clock is simply divided by the field CD in the US_BRGR.

$$BaudRate = \frac{SelectedClock}{CD}$$

In Synchronous mode, if the external clock is selected (USCLKS = 3), the clock is provided directly by the signal on the USART SCK pin. No division is active. The value written in US_BRGR has no effect. The external clock frequency must be at least 3 times lower than the system clock. In Synchronous mode master (USCLKS = 0 or 1, CLKO set to 1), the receive part limits the SCK maximum frequency to $f_{peripheral\ clock}/3$ in USART mode, or $f_{peripheral\ clock}/6$ in SPI mode.

When either the external clock SCK or the internal clock divided (peripheral clock/DIV) is selected, the value programmed in CD must be even if the user has to ensure a 50:50 mark/space ratio on the SCK pin. When the peripheral clock is selected, the baud rate generator ensures a 50:50 duty cycle on the SCK pin, even if the value programmed in CD is odd.

Figure 40-9. Data OUT Transfer for Non Ping-pong Endpoints

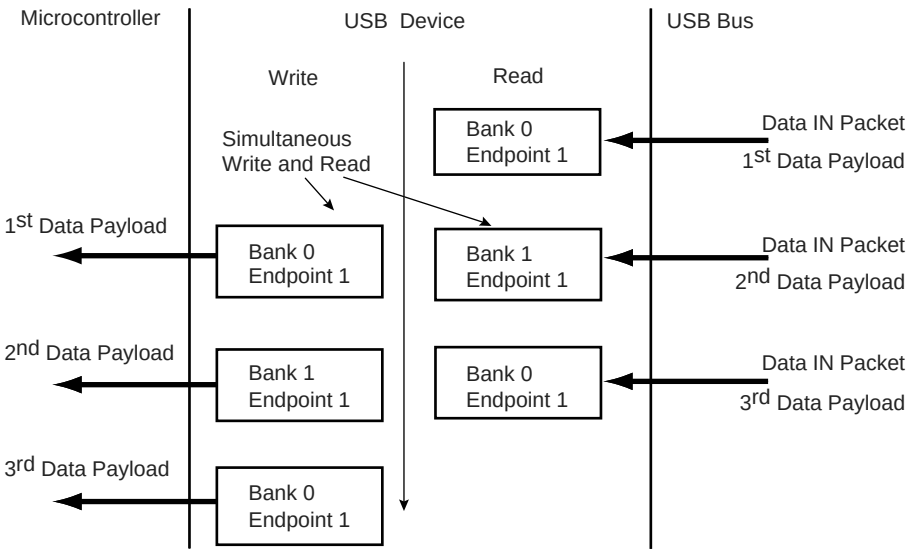


An interrupt is pending while the flag RX_DATA_BK0 is set. Memory transfer between the USB device, the FIFO and microcontroller memory is not possible after RX_DATA_BK0 has been cleared. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the current Data OUT packet in the FIFO.

Using Endpoints With Ping-pong Attributes

During isochronous transfer, using an endpoint with ping-pong attributes is obligatory. To be able to guarantee a constant bandwidth, the microcontroller must read the previous data payload sent by the host, while the current data payload is received by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

Figure 40-10. Bank Swapping in Data OUT Transfers for Ping-pong Endpoints



When using a ping-pong endpoint, the following procedures are required to perform Data OUT transactions:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. It is written in the endpoint's FIFO Bank 0.
3. The USB device sends an ACK PID packet to the host. The host can immediately send a second Data OUT packet. It is accepted by the device and copied to FIFO Bank 1.
4. The microcontroller is notified that the USB device has received a data payload, polling RX_DATA_BK0 in the endpoint's UDP_CSRx. An interrupt is pending for this endpoint while RX_DATA_BK0 is set.

Table 49-3. SAM4S Datasheet Rev. 11100I Revision History (Continued)

Doc. Date	Changes
03-Apr-15	**Section 41., "Analog Comparator Controller (ACC)"** Added Table 41-1 "List of External Analog Data Inputs" Changed all occurrences of 'MCK' to 'peripheral clock'. Updated Table 41-2 "ACC Pin List" Updated Figure 41-1, "Analog Comparator Controller Block Diagram" Section 41.7.2, "ACC Mode Register": replaced two instances of "CF" with "CE" in SELFS bit description Section 41.7.6, "ACC Interrupt Status Register": added (cleared on read) to CE bit description
	Section 42., "Analog-to-Digital Converter (ADC)" Corrected ADC_SR to ADC_ISR in Section 42.6.5, "Conversion Results". Figure 42-1 "Analog-to-Digital Converter Block Diagram": Added bus clock and added ADCClock output from Control Logic block Revised Section 42.6.1, "Analog-to-Digital Conversion" Added Section 42.6.2, "ADC Clock" Section 42.6.3, "ADC Reference Voltage": changed title (was "Conversion Reference") Modified Figure 42-4 "EOCx and DRDY Flag Behavior" and Figure 42-5 "EOCx, OVREx and GOVREx Flag Behavior" Modified warning below Figure 42-5 "EOCx, OVREx and GOVREx Flag Behavior". Section 42.6.8, "Comparison Window": Removed paragraph on LOWRES bit use. Updated Section 42.6.9, "Differential Inputs" Modified Table 42-5 "Gain of the Sample and Hold Unit" and Table 42-6 "Offset of the Sample and Hold Unit". Section 42.6.10, "Input Gain and Offset": all references to "OFFSET" bit changed to OFFx. Added Figure 42-8 "Buffer Structure". Modified Section 42.7.2, "ADC Mode Register", Section 42.7.6, "ADC Channel Disable Register", Section 42.7.12, "ADC Interrupt Status Register", Section 42.7.14, "ADC Extended Mode Register", Section 42.7.16, "ADC Channel Gain Register" Replaced references to vrefin by V_{ADVREF} in text and figures. Section 42.6.12 "Automatic Calibration": Removed sentence on ADC running mode. Replaced instances of "ADCClock" with "ADC clock" or acronym "ADCCLK" throughout
	Section 43., "Digital-to-Analog Converter Controller (DACC)" MCK replaced with Peripheral clock Updated Section 43.2, "Embedded Characteristics" Updated Figure 43-1 "DACC Block Diagram" Section 43.6.6, "DACC Timings": replaced two instances of "DACC clock periods" with "peripheral clock periods" Table 43-3 "Register Mapping": - Removed reset value from write-only register DACC_CDR - Displayed offset ranges 0x34–0x90 and 0x98–0xE0 as "Reserved" Modified Section 43.7.2, "DACC Mode Register" Removed references to Sleep mode and refresh period