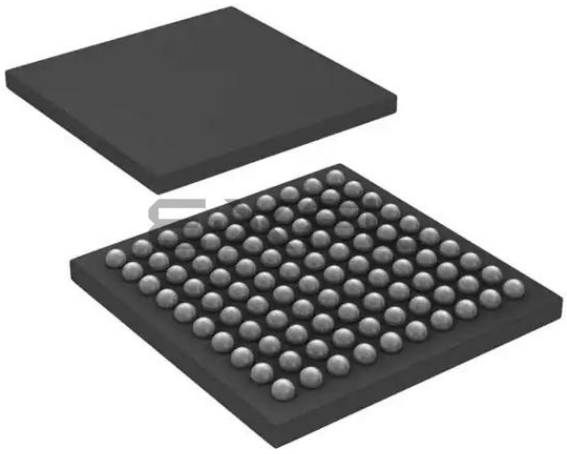


Welcome to E-XFL.COM

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	EBI/EMI, I²C, IrDA, Memory Card, SPI, SSC, UART/USART, USB
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	79
Program Memory Size	256KB (256K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	64K x 8
Voltage - Supply (Vcc/Vdd)	1.62V ~ 3.6V
Data Converters	A/D 16x12b; D/A 2x12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	100-VFBGA
Supplier Device Package	100-VFBGA (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atsam4s4cb-cfn

In addition, if the SEVONPEND bit in the SCR is set to 1, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause an exception entry. For more information about the SCR, see “System Control Register” .

12.5.2.3 External Event Input

The processor provides an external event input signal. Peripherals can drive this signal, either to wake the processor from WFE, or to set the internal WFE event register to 1 to indicate that the processor must not enter sleep mode on a later WFE instruction. See “Wait for Event” for more information.

12.5.3 Power Management Programming Hints

ISO/IEC C cannot directly generate the WFI and WFE instructions. The CMSIS provides the following functions for these instructions:

```
void __WFE(void) // Wait for Event
void __WFI(void) // Wait for Interrupt
```

12.6.4.1 ADR

Load PC-relative address.

Syntax

```
ADR{cond} Rd, label
```

where:

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

Rd is the destination register.

label is a PC-relative expression. See “PC-relative Expressions” .

Operation

ADR determines the address by adding an immediate value to the PC, and writes the result to the destination register.

ADR produces position-independent code, because the address is PC-relative.

If ADR is used to generate a target address for a BX or BLX instruction, ensure that bit[0] of the address generated is set to 1 for correct execution.

Values of *label* must be within the range of –4095 to +4095 from the address in the PC.

Note: The user might have to use the .W suffix to get the maximum offset range or to generate addresses that are not word-aligned. See “Instruction Width Selection” .

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

```
ADR     R1, TextMessage   ; Write address value of a location labelled as
                          ; TextMessage to R1
```

where

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

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See “Flexible Second Operand” for details of the options.

Operation

These instructions test the value in a register against *Operand2*. They update the condition flags based on the result, but do not write the result to a register.

The TST instruction performs a bitwise AND operation on the value in *Rn* and the value of *Operand2*. This is the same as the ANDS instruction, except that it discards the result.

To test whether a bit of *Rn* is 0 or 1, use the TST instruction with an *Operand2* constant that has that bit set to 1 and all other bits cleared to 0.

The TEQ instruction performs a bitwise Exclusive OR operation on the value in *Rn* and the value of *Operand2*. This is the same as the EORS instruction, except that it discards the result.

Use the TEQ instruction to test if two values are equal without affecting the V or C flags.

TEQ is also useful for testing the sign of a value. After the comparison, the N flag is the logical Exclusive OR of the sign bits of the two operands.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions:

- Update the N and Z flags according to the result
- Can update the C flag during the calculation of *Operand2*, see “Flexible Second Operand”
- Do not affect the V flag.

Examples

```
TST    R0, #0x3F8 ; Perform bitwise AND of R0 value to 0x3F8,
                  ; APSR is updated but result is discarded
TEQEQ  R10, R9    ; Conditionally test if value in R10 is equal to
                  ; value in R9, APSR is updated but result is discarded.
```

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

Examples

```
UQASX   R7, R4, R2  ; Adds top halfword of R4 with bottom halfword of R2,
                    ; saturates to 16 bits, writes to top halfword of R7
                    ; Subtracts top halfword of R2 from bottom halfword of
                    ; R4, saturates to 16 bits, writes to bottom halfword of R7
UQSAX   R0, R3, R5  ; Subtracts bottom halfword of R5 from top halfword of R3,
                    ; saturates to 16 bits, writes to top halfword of R0
                    ; Adds bottom halfword of R4 to top halfword of R5
                    ; saturates to 16 bits, writes to bottom halfword of R0.
```

12.6.7.7 UQADD and UQSUB

Saturating Add and Saturating Subtract Unsigned.

Syntax

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

where:

op is one of:

UQADD8 Saturating four unsigned 8-bit integer additions.

UQADD16 Saturating two unsigned 16-bit integer additions.

UDSUB8 Saturating four unsigned 8-bit integer subtractions.

UQSUB16 Saturating two unsigned 16-bit integer subtractions.

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

These instructions add or subtract two or four values and then writes an unsigned saturated value in the destination register.

The UQADD16 instruction:

- Adds the respective top and bottom halfwords of the first and second operands.
- Saturates the result of the additions for each halfword in the destination register to the unsigned range $0 \le x \le 2^{16}-1$, where x is 16.

The UQADD8 instruction:

- Adds each respective byte of the first and second operands.
- Saturates the result of the addition for each byte in the destination register to the unsigned range $0 \le x \le 2^{8}-1$, where x is 8.

The UQSUB16 instruction:

- Subtracts both halfwords of the second operand from the respective halfwords of the first operand.
- Saturates the result of the differences in the destination register to the unsigned range $0 \le x \le 2^{16}-1$, where x is 16.

The UQSUB8 instructions:

- Subtracts the respective bytes of the second operand from the respective bytes of the first operand.
- Saturates the results of the differences for each byte in the destination register to the unsigned range $0 \le x \le 2^{8}-1$, where x is 8.

15.4 Functional Description

The programmable 16-bit prescaler value can be configured through the RTPRES field in the “Real-time Timer Mode Register” (RTT_MR).

Configuring the RTPRES field value to 0x8000 (default value) corresponds to feeding the real-time counter with a 1Hz signal (if the slow clock is 32.768 kHz). The 32-bit counter can count up to 2^{32} seconds, corresponding to more than 136 years, then roll over to 0. Bit RTTINC in the “Real-time Timer Status Register” (RTT_SR) is set each time there is a prescaler roll-over (see Figure 15-2)

The real-time 32-bit counter can also be supplied by the 1Hz RTC clock. This mode is interesting when the RTC 1Hz is calibrated (CORRECTION field $\neq 0$ in RTC_MR) in order to guaranty the synchronism between RTC and RTT counters.

Setting the RTC1HZ bit in the RTT_MR drives the 32-bit RTT counter from the 1Hz RTC clock. In this mode, the RTPRES field has no effect on the 32-bit counter.

The prescaler roll-over generates an increment of the real-time timer counter if RTC1HZ = 0. Otherwise, if RTC1HZ = 1, the real-time timer counter is incremented every second. The RTTINC bit is set independently from the 32-bit counter increment.

The real-time timer can also be used as a free-running timer with a lower time-base. The best accuracy is achieved by writing RTPRES to 3 in RTT_MR.

Programming RTPRES to 1 or 2 is forbidden.

If the RTT is configured to trigger an interrupt, the interrupt occurs two slow clock cycles after reading the RTT_SR. To prevent several executions of the interrupt handler, the interrupt must be disabled in the interrupt handler and re-enabled when the RTT_SR is cleared.

The CRTV field can be read at any time in the “Real-time Timer Value Register” (RTT_VR). As this value can be updated asynchronously with the Master Clock, the CRTV field must be read twice at the same value to read a correct value.

The current value of the counter is compared with the value written in the “Real-time Timer Alarm Register” (RTT_AR). If the counter value matches the alarm, the ALMS bit in the RTT_SR is set. The RTT_AR is set to its maximum value (0xFFFF_FFFF) after a reset.

The ALMS flag is always a source of the RTT alarm signal that may be used to exit the system from low power modes (see Figure 15-1).

The alarm interrupt must be disabled (ALMIEN must be cleared in RTT_MR) when writing a new ALMV value in the RTT_AR.

The RTTINC bit can be used to start a periodic interrupt, the period being one second when the RTPRES field value = 0x8000 and the slow clock = 32.768 kHz.

The RTTINCIEN bit must be cleared prior to writing a new RTPRES value in the RTT_MR.

Reading the RTT_SR automatically clears the RTTINC and ALMS bits.

Writing the RTTRST bit in the RTT_MR immediately reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

When not used, the Real-time Timer can be disabled in order to suppress dynamic power consumption in this module. This can be achieved by setting the RTTDIS bit in the RTT_MR.

16.6.4 RTC Calendar Register

Name: RTC_CALR

Address: 0x400E146C

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	DATE					

23	22	21	20	19	18	17	16
DAY			MONTH				

15	14	13	12	11	10	9	8
YEAR							

7	6	5	4	3	2	1	0
–	CENT						

- **CENT: Current Century**

The range that can be set is 19–20 (gregorian) or 13–14 (persian) (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **YEAR: Current Year**

The range that can be set is 00–99 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **MONTH: Current Month**

The range that can be set is 01–12 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **DAY: Current Day in Current Week**

The range that can be set is 1–7 (BCD).

The coding of the number (which number represents which day) is user-defined as it has no effect on the date counter.

- **DATE: Current Day in Current Month**

The range that can be set is 01–31 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

All non-significant bits read zero.

20.5.4 EEFC Flash Result Register

Name: EEFC_FRR

Address: 0x400E0A0C (0), 0x400E0C0C (1)

Access: Read-only

31	30	29	28	27	26	25	24
FVALUE							

23	22	21	20	19	18	17	16
FVALUE							

15	14	13	12	11	10	9	8
FVALUE							

7	6	5	4	3	2	1	0
FVALUE							

- **FVALUE: Flash Result Value**

The result of a Flash command is returned in this register. If the size of the result is greater than 32 bits, the next resulting value is accessible at the next register read.

23.6.2 Transfer Control Register

Name: TR_CTRL

Access: Read/Write

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

- **BTSIZE: Buffer Transfer Size**

- **TRWIDTH: Transfer Width Register**

Value	Name	Description
0	BYTE	The data size is 8-bit
1	HALFWORD	The data size is 16-bit
2	WORD	The data size is 32-bit

- **IEN: Context Done Interrupt Enable (Active Low)**

0: Bit DMAISR of CRCCU_DMA_ISR is set at the end of the current descriptor transfer.

1: Bit DMAISR of CRCCU_DMA_ISR remains cleared.

25.2.3 Master to Slave Access

Table 25-3 gives valid paths for master to slave access on Matrix 0. The paths shown as “-” are forbidden or not wired, e.g. access from the Cortex-M4 S Bus to the internal ROM.

Table 25-3. Master to Slave Access

Slaves	Masters	0	1	2	3
		Cortex-M4 I/D Bus	**Cortex-M4 S Bus**	**PDC**	**CRCCU**
0	Internal SRAM	–	X	X	X
1	Internal ROM	X	–	X	X
2	Internal Flash	X	–	-	X
3	External Bus Interface	–	X	X	X
4	Peripheral Bridge	–	X	X	–

25.3 Memory Mapping

The Bus Matrix provides one decoder for every AHB master interface. The decoder offers each AHB master several memory mappings. In fact, depending on the product, each memory area may be assigned to several slaves. Thus it is possible to boot at the same address while using different AHB slaves.

25.4 Special Bus Granting Techniques

The Bus Matrix provides some speculative bus granting techniques in order to anticipate access requests from some masters. This technique reduces latency at the first access of a burst or single transfer. Bus granting sets a default master for every slave.

At the end of the current access, if no other request is pending, the slave remains connected to its associated default master. A slave can be associated with one of the three implementations of default masters:

- No default master
- Last access master
- Fixed default master

25.4.1 No Default Master

At the end of the current access, if no other request is pending, the slave is disconnected from all masters. No default master suits low-power mode.

25.4.2 Last Access Master

At the end of the current access, if no other request is pending, the slave remains connected to the last master that performed an access request.

25.4.3 Fixed Default Master

At the end of the current access, if no other request is pending, the slave connects to its fixed default master. Unlike the last access master, the fixed master does not change unless the user modifies it by software (field FIXED_DEFMSTR of the related MATRIX_SCFG).

To change from one kind of default master to another, the Bus Matrix user interface provides the Slave Configuration registers (MATRIX_SCFGx), one for each slave, used to set a default master for each slave. MATRIX_SCFGx contains the fields DEFMSTR_TYPE and FIXED_DEFMSTR. The 2-bit DEFMSTR_TYPE field selects the default master type (no default, last access master, fixed default master) whereas the 4-bit FIXED_DEFMSTR field selects a fixed default master, provided that DEFMSTR_TYPE is set to fixed default master. Refer to Table 25-4, “Register Mapping”.

automatically forces the fast RC oscillator to be the source clock for MAINCK. If the fast RC oscillator is disabled when a clock failure detection occurs, it is automatically re-enabled by the clock failure detection mechanism.

It takes two slow RC oscillator clock cycles to detect and switch from the main oscillator, to the fast RC oscillator if the source master clock (MCK) is main clock (MAINCK), or three slow clock RC oscillator cycles if the source of MCK is PLLACKor PLLBCK.

A clock failure detection activates a fault output that is connected to the Pulse Width Modulator (PWM) Controller. With this connection, the PWM controller is able to force its outputs and to protect the driven device, if a clock failure is detected.

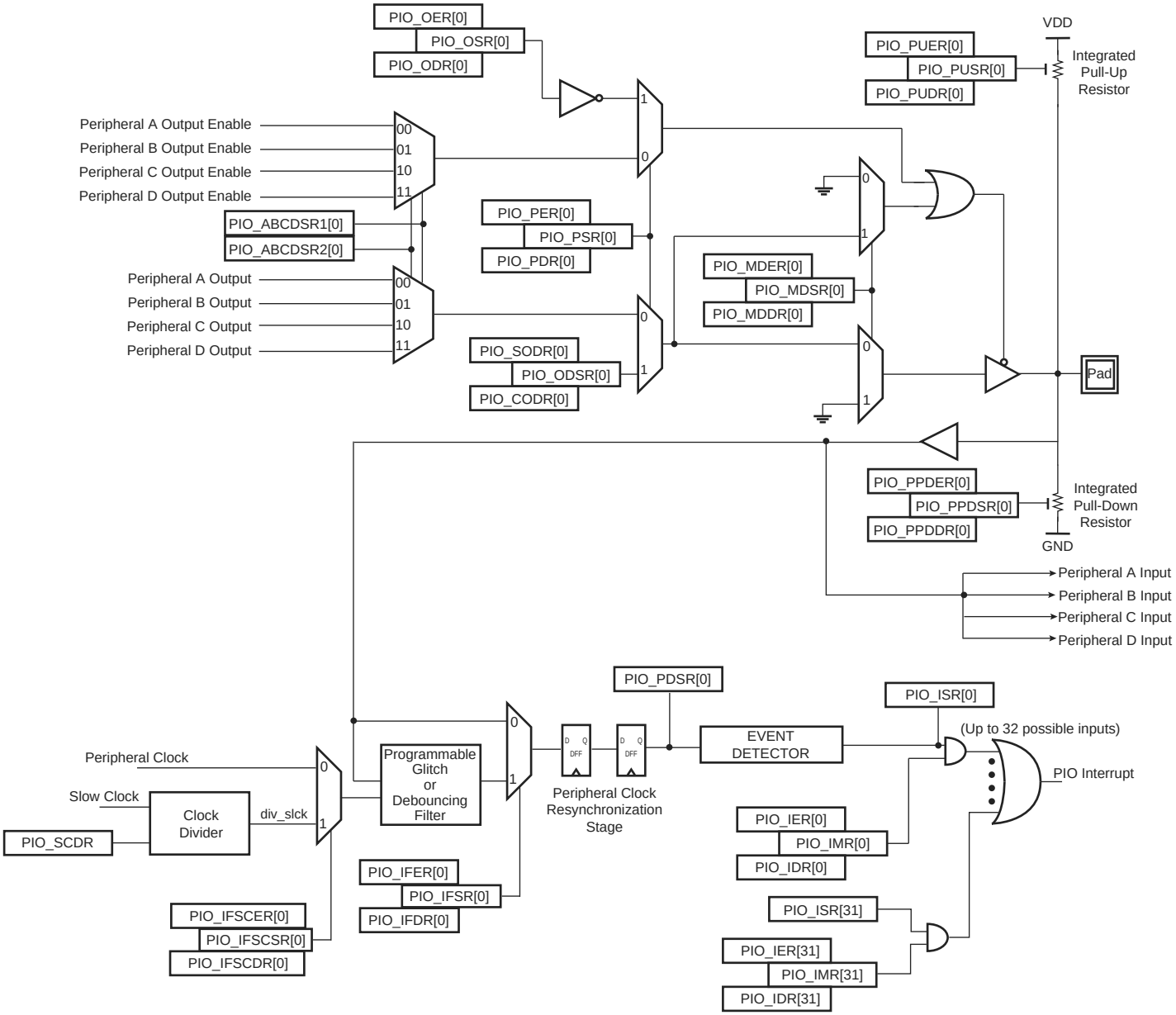
The user can know the status of the clock failure detector at any time by reading the FOS bit in PMC_SR.

This fault output remains active until the defect is detected and until it is cleared by the bit FOCLR in the PMC Fault Output Clear Register (PMC_FOCR).

31.5 Functional Description

The PIO Controller features up to 32 fully-programmable I/O lines. Most of the control logic associated to each I/O is represented in Figure 31-2. In this description each signal shown represents one of up to 32 possible indexes.

Figure 31-2. I/O Line Control Logic



31.5.1 Pull-up and Pull-down Resistor Control

Each I/O line is designed with an embedded pull-up resistor and an embedded pull-down resistor. The pull-up resistor can be enabled or disabled by writing to the Pull-up Enable Register (PIO_PUER) or Pull-up Disable Register (PIO_PUDR), respectively. Writing to these registers results in setting or clearing the corresponding bit in the Pull-up Status Register (PIO_PUSR). Reading a one in PIO_PUSR means the pull-up is disabled and reading a zero means the pull-up is enabled. The pull-down resistor can be enabled or disabled by writing the Pull-down Enable Register (PIO_PPDER) or the Pull-down Disable Register (PIO_PPDDR), respectively. Writing in these

31.5.4 Output Control

When the I/O line is assigned to a peripheral function, i.e., the corresponding bit in PIO_PSR is at zero, the drive of the I/O line is controlled by the peripheral. Peripheral A or B or C or D depending on the value in PIO_ABCDSR1 and PIO_ABCDSR2 determines whether the pin is driven or not.

When the I/O line is controlled by the PIO Controller, the pin can be configured to be driven. This is done by writing the Output Enable Register (PIO_OER) and Output Disable Register (PIO_ODR). The results of these write operations are detected in the Output Status Register (PIO_OSR). When a bit in this register is at zero, the corresponding I/O line is used as an input only. When the bit is at one, the corresponding I/O line is driven by the PIO Controller.

The level driven on an I/O line can be determined by writing in the Set Output Data Register (PIO_SODR) and the Clear Output Data Register (PIO_CODR). These write operations, respectively, set and clear the Output Data Status Register (PIO_ODSR), which represents the data driven on the I/O lines. Writing in PIO_OER and PIO_ODR manages PIO_OSR whether the pin is configured to be controlled by the PIO Controller or assigned to a peripheral function. This enables configuration of the I/O line prior to setting it to be managed by the PIO Controller.

Similarly, writing in PIO_SODR and PIO_CODR affects PIO_ODSR. This is important as it defines the first level driven on the I/O line.

31.5.5 Synchronous Data Output

Clearing one or more PIO line(s) and setting another one or more PIO line(s) synchronously cannot be done by using PIO_SODR and PIO_CODR. It requires two successive write operations into two different registers. To overcome this, the PIO Controller offers a direct control of PIO outputs by single write access to PIO_ODSR. Only bits unmasked by the Output Write Status Register (PIO_OWSR) are written. The mask bits in PIO_OWSR are set by writing to the Output Write Enable Register (PIO_OWER) and cleared by writing to the Output Write Disable Register (PIO_OWDR).

After reset, the synchronous data output is disabled on all the I/O lines as PIO_OWSR resets at 0x0.

31.5.6 Multi-Drive Control (Open Drain)

Each I/O can be independently programmed in open drain by using the multi-drive feature. This feature permits several drivers to be connected on the I/O line which is driven low only by each device. An external pull-up resistor (or enabling of the internal one) is generally required to guarantee a high level on the line.

The multi-drive feature is controlled by the Multi-driver Enable Register (PIO_MDER) and the Multi-driver Disable Register (PIO_MDDR). The multi-drive can be selected whether the I/O line is controlled by the PIO Controller or assigned to a peripheral function. The Multi-driver Status Register (PIO_MDSR) indicates the pins that are configured to support external drivers.

After reset, the multi-drive feature is disabled on all pins, i.e., PIO_MDSR resets at value 0x0.

31.5.7 Output Line Timings

Figure 31-3 shows how the outputs are driven either by writing PIO_SODR or PIO_CODR, or by directly writing PIO_ODSR. This last case is valid only if the corresponding bit in PIO_OWSR is set. Figure 31-3 also shows when the feedback in the Pin Data Status Register (PIO_PDSR) is available.

31.6.40 PIO Level Select Register

Name: PIO_LSR

Address: 0x400E0EC4 (PIOA), 0x400E10C4 (PIOB), 0x400E12C4 (PIOC)

Access: Write-only

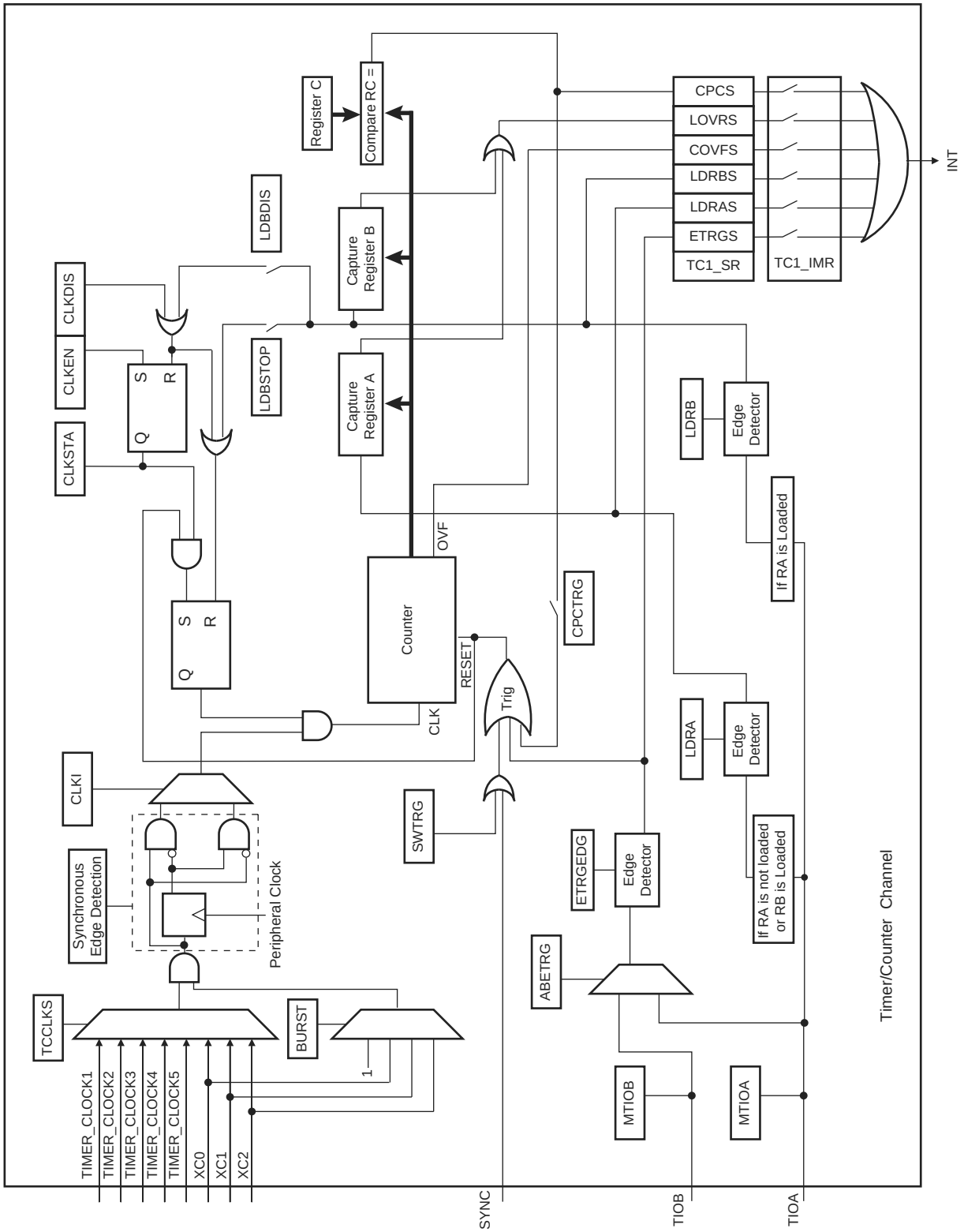
31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Level Interrupt Selection**

0: No effect.

1: The interrupt source is a level-detection event.

Figure 37-5. Capture Mode



38.14.5 HSMCI Argument Register

Name: HSMCI_ARGR

Address: 0x40000010

Access: Read/Write

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

- **ARG: Command Argument**

41.7.8 ACC Write Protection Mode Register

Name: ACC_WPMR

Address: 0x400400E4

Access: Read/Write

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

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x414343 (“ACC” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x414343 (“ACC” in ASCII).

See “Register Write Protection” on page 1071 for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x414343	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

44.12.3 SPI Characteristics

Figure 44-22. SPI Master Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)

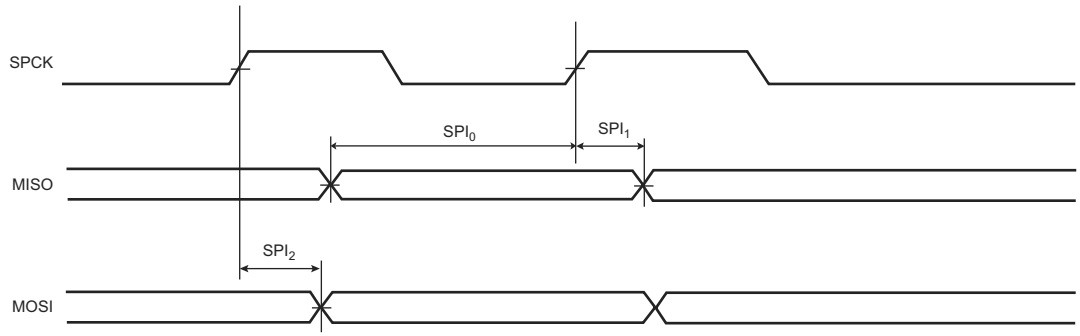


Figure 44-23. SPI Master Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)

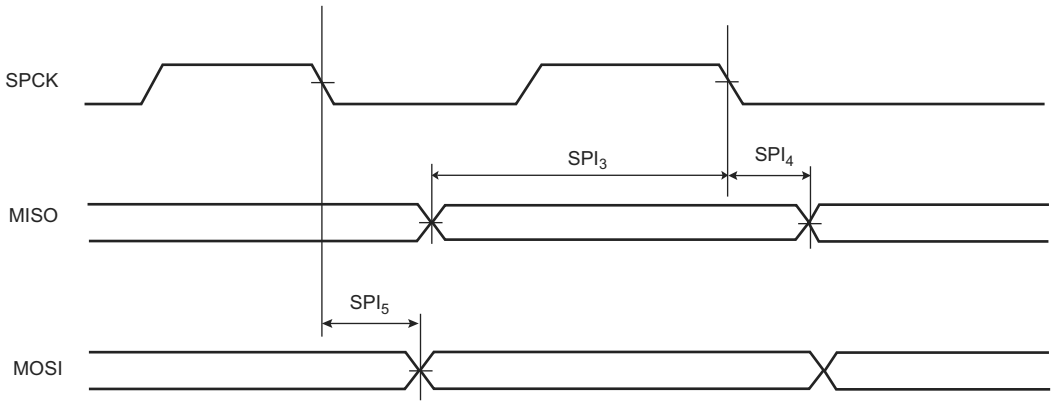
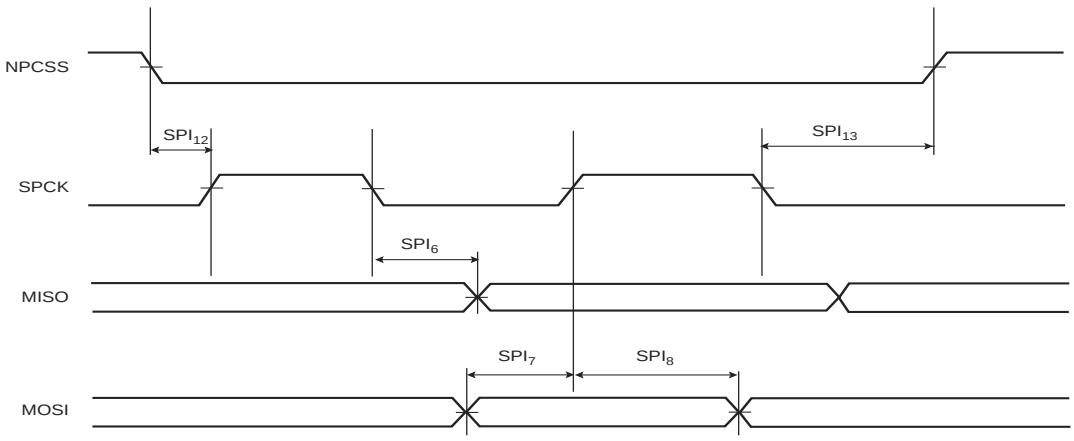


Figure 44-24. SPI Slave Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)



44.12.7 USART in SPI Mode Timings

Timings are given in the following domain:

- 1.8V domain: V_{DDIO} from 1.65 to 1.95 V, maximum external capacitor = 20 pF
- 3.3V domain: V_{DDIO} from 2.85 to 3.6 V, maximum external capacitor = 40 pF

Figure 44-37. USART SPI Master Mode

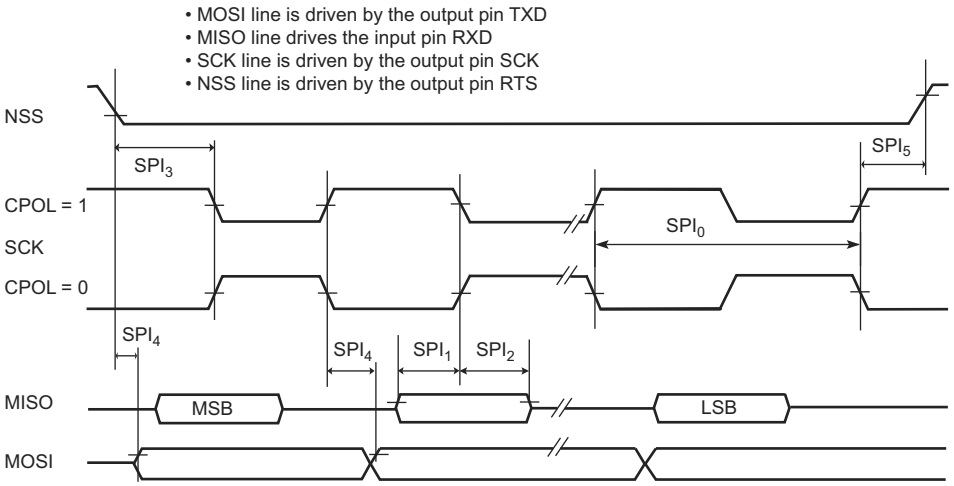
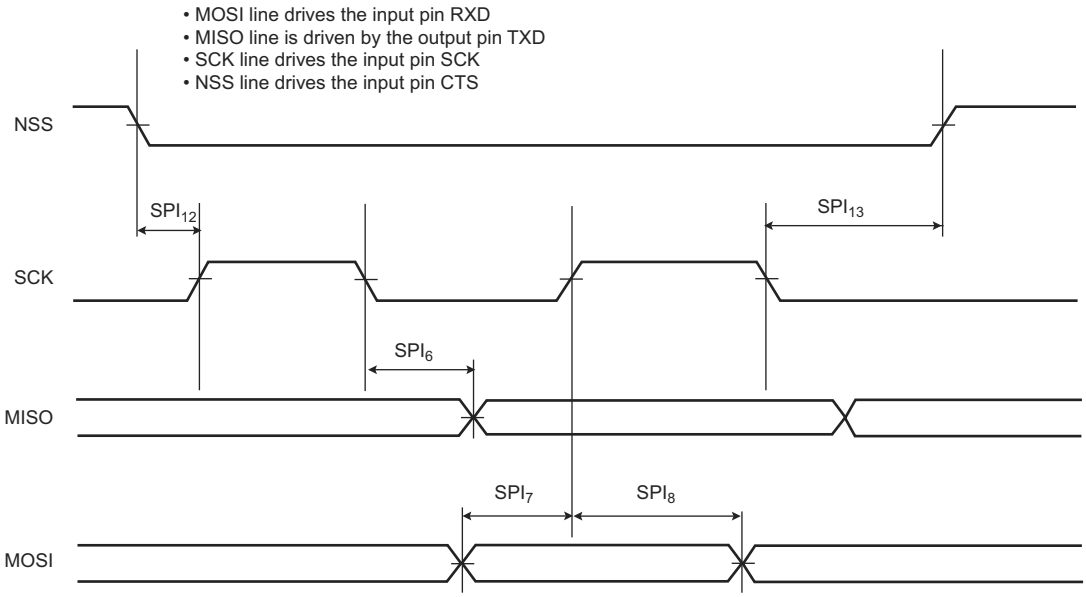


Figure 44-38. USART SPI Slave Mode: (Mode 1 or 2)



48.3.3 Brownout Detector

Issue: Unpredictable Behavior if BOD is Disabled, VDDCORE is Lost and VDDIO is Connected

In active mode or in wait mode, if the Brownout Detector is disabled (SUPC_MR.BODDIS = 1) and power is lost on VDDCORE while VDDIO is powered, the device might not be properly reset and may behave unpredictably.

Workaround: When the Brownout Detector is disabled in active or in wait mode, VDDCORE always needs to be powered.

48.3.4 Low-power Mode

Issue: Unpredictable Behavior When Entering Sleep Mode

When entering Sleep mode, if an interrupt occurs during WFI or WFE (PMC_FSMR.LPM = 0) instruction processing, the ARM core may read an incorrect data, thus leading to unpredictable behavior of the software. This issue is not present in Wait mode.

Workaround: The following conditions must be met:

1. The interrupt vector table must be located in Flash.
2. The Matrix slave interface for the Flash must be set to 'No default master'. This is done by setting the field DEFMSTR_TYPE to 0 in the register MATRIX_SCFG. The code example below can be used to program the NO_DEFAULT_MASTER state:

```
MATRIX_SCFG[2] = MATRIX_SCFG_SLOT_CYCLE(0xFF) | MATRIX_SCFG_DEFMSTR_TYPE(0x0);
```

   This must be done once in the software before entering Sleep mode.

48.3.5 PIO

Issue: PB4 Input Low-level Voltage Range

The undershoot is limited to -0.1V.

In normal operating conditions, the V_{IL} minimum value on PB4 is limited to 0V.

Workaround: The voltage on PB4 with respect to ground must be in the range -0.1V to + VDDIO + 0.4V instead of -0.3V to + VDDIO + 0.4V for all other input pins, as shown in Table 44.1 "Absolute Maximum Ratings".

The minimum V_{IL} on PB4 must be 0V instead of -0.3V for all other input pins, as shown in Table 44.3 "DC Characteristics".

49. Revision History

In the tables that follow, the most recent version of the document appears first.

Table 49-1. SAM4S Datasheet Rev. 11100K Revision History

Doc. Date	Changes
09-Jun-15	**"Features":** updated "Memories" section.
	Added section "Safety Features Highlight".
	Section 8., "Memories" Added Section 8.1.3.4 "Error Code Correction (ECC)".
	Section 44., "Electrical Characteristics" Added Table 44-24, "SAM4SD32/SA16/SD16 Typical Active Power Consumption with VDDCORE@ 1.2V running from Flash Memory (AMP2) or SRAM".

Table 49-2. SAM4S Datasheet Rev. 11100J Revision History

Doc. Date	Changes
28-May-15	**"Features":** updated "System" and "Peripherals" sections
	Section 2., "Block Diagram" Updated Figures
	Section 4., "Package and Pinout" Modified AD13 and AD14 position in Table 4-2 "SAM4SD32/SD16/SA16/S16/S8/S4/S2 100-ball TFBGA Pinout" and Table 4-3 "SAM4SD32/SD16/SA16/S16/S8/S4/S2 100-ball VFBGA Pinout"
	Section 44., "Electrical Characteristics" Updated Table 44-41, "ADC Timing Characteristics" Table 44-74, "AC Flash Characteristics": added one "Endurance" value
	Section 48., "Errata" Section 48.1.5, "Low-power Mode" and Section 48.3.4, "Low-power Mode": modified Workaround 2 (code example)