23.7 EIC Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0				CKSEL			ENABLE	SWRST
0x01	NMICTRL	7:0				NMIASYNCH	NMIFILTEN	NMISENSE[2:0]		
0x02	NMIFLAG	7:0								NMI
0x03	Reserved									
0x04	SYNCBUSY	7:0							ENABLE	SWRST
		15:8								
		23:16								
		31:24								
0x08	EVCTRL	7:0								
		15:8								
		23:16								
		31:24								
0x0C	INTENCLR	7:0								
		15:8								
		23:16								
		31:24								
0x10	INTENSET	7:0								
		15:8								
		23:16								
		31:24								
0x14	INTFLAG	7:0								
		15:8								
		23:16								
		31:24								
0x18	ASYNCH	7:0								
		15:8								
		23:16								
		31:24								
0x1C	CONFIG	7:0	FILTEN1	SENSE1[2:0]			FILTEN0	SENSE0[2:0]		
		15:8	FILTEN3	SENSE3[2:0]			FILTEN2	SENSE2[2:0]		
		23:16								
		31:24								
0x20 ... 0x2F	Reserved									
0x30	DEBOUNCEN	7:0								
		15:8								
		23:16								
		31:24								
0x34	DPRESCALER	7:0								
		15:8								
		23:16								TICKON
		31:24								
0x38	PINSTATE	7:0								
		15:8								
		23:16								
		31:24								

23.8 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register, and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers require synchronization when read and/or written. Synchronization is denoted by the “Read-Synchronized” and/or “Write-Synchronized” property in each individual register description.

Some registers are enable-protected, meaning they can only be written when the module is disabled. Enable-protection is denoted by the “Enable-Protected” property in each individual register description.

23.8.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00
Property: PAC Write-Protection, Write-Synchronized

Bit	7	6	5	4	3	2	1	0
				CKSEL			ENABLE	SWRST
Access				RW			RW	W
Reset				0			0	0

Bit 4 – CKSEL Clock Selection
The EIC can be clocked either by GCLK_EIC (when a frequency higher than 32.768 KHz is required for filtering) or by CLK_ULP32K (when power consumption is the priority).
This bit is not Write-Synchronized.

Value	Description
0	The EIC is clocked by GCLK_EIC.
1	The EIC is clocked by CLK_ULP32K.

Bit 1 – ENABLE Enable
Due to synchronization there is a delay between writing to CTRLA.ENABLE until the peripheral is enabled/disabled. The value written to CTRLA.ENABLE will read back immediately and the Enable bit in the Synchronization Busy register will be set (SYNCBUSY.ENABLE=1). SYNCBUSY.ENABLE will be cleared when the operation is complete.
This bit is not Enable-Protected.
This bit is Write-Synchronized.

Value	Description
0	The EIC is disabled.
1	The EIC is enabled.

Bit 0 – SWRST Software Reset
Writing a '0' to this bit has no effect.
Writing a '1' to this bit resets all registers in the EIC to their initial state, and the EIC will be disabled.
Writing a '1' to CTRLA.SWRST will always take precedence, meaning that all other writes in the same write operation will be discarded.
Due to synchronization there is a delay from writing CTRLA.SWRST until the Reset is complete. CTRLA.SWRST and SYNCBUSY.SWRST will both be cleared when the Reset is complete.
This bit is not Enable-Protected.
This bit is Write-Synchronized.

Value	Description
0	There is no ongoing reset operation.
1	The reset operation is ongoing.

23.8.2 Non-Maskable Interrupt Control

Name: NMICTRL
Offset: 0x01
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
				NMIASYNCH	NMIFILTEN	NMISENSE[2:0]		
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit 4 – NMIASYNCH NMI Asynchronous Edge Detection Mode
The NMI edge detection can be operated synchronously or asynchronously to the EIC clock.

Value	Description
0	The NMI edge detection is synchronously operated.
1	The NMI edge detection is asynchronously operated.

Bit 3 – NMIFILTEN Non-Maskable Interrupt Filter Enable

Value	Description
0	NMI filter is disabled.
1	NMI filter is enabled.

Bits 2:0 – NMISENSE[2:0] Non-Maskable Interrupt Sense Configuration
These bits define on which edge or level the NMI triggers.
Note: NMI cannot be triggered based on level but it is always based on edge.

Value	Name	Description
0x0	NONE	No detection
0x1	RISE	Rising-edge detection
0x2	FALL	Falling-edge detection
0x3	BOTH	Both-edge detection
0x4	HIGH	High-level detection
0x5	LOW	Low-level detection
0x6 – 0x7	–	Reserved

23.8.3 Non-Maskable Interrupt Flag Status and Clear

Name: NMIFLAG
Offset: 0x2
Reset: 0x00

Bit	7	6	5	4	3	2	1	0
								NMI
Access								RW
Reset								0

Bit 0 – NMI Non-Maskable Interrupt
This flag is cleared by writing a '1' to it.
This flag is set when the NMI pin matches the NMI sense configuration, and will generate an interrupt request.
Writing a '0' to this bit has no effect.

23.8.4 Synchronization Busy

Name: SYNCBUSY
Offset: 0x04
Reset: 0x00000000

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
							ENABLE	SWRST
Access							R	R
Reset							0	0

Bit 1 – ENABLE Enable Synchronization Busy Status

Value	Description
0	Write synchronization for CTRLA.ENABLE bit is complete.
1	Write synchronization for CTRLA.ENABLE bit is ongoing.

Bit 0 – SWRST Software Reset Synchronization Busy Status

Note: During a SWRST, access to registers/bits without SWRST are disallowed until SYNCBUSY.SWRST cleared by hardware.

Value	Description
0	Write synchronization for CTRLA.SWRST bit is complete.
1	Write synchronization for CTRLA.SWRST bit is ongoing.

23.8.5 Event Control

Name: EVCTRL
Offset: 0x08
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

23.8.6 Interrupt Enable Clear

Name: INTENCLR
Offset: 0x0C
Reset: 0x00000000
Property: PAC Write-Protection

This register allows the user to disable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Set register (INTENSET).

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

23.8.7 Interrupt Enable Set

Name: INTENSET
Offset: 0x10
Reset: 0x00000000
Property: PAC Write-Protection

This register allows the user to enable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Clear (INTENCLR) register.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

23.8.8 Interrupt Flag Status and Clear

Name: INTFLAG
Offset: 0x14
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

23.8.9 External Interrupt Asynchronous Mode

Name: ASYNCH
Offset: 0x18
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

23.8.10 External Interrupt Sense Configuration

Name: CONFIG
Offset: 0x1C
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	FILTEN3	SENSE3[2:0]			FILTEN2	SENSE2[2:0]		
Access	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	FILTEN1	SENSE1[2:0]			FILTEN0	SENSE0[2:0]		
Access	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0

Bits 3, 7, 11, 15 – FILTENx Filter Enable x [x=3..0]
Note: The filter must be disabled if the asynchronous detection is enabled.

Value	Description
0	Filter is disabled for EXTINT[x] input.
1	Filter is enabled for EXTINT[x] input.

Bits 0:2, 4:6, 8:10, 12:14 – SENSEx Input Sense Configuration x [x=3..0]
These bits define on which edge or level the interrupt or event for EXTINT[x] will be generated.

Value	Name	Description
0x0	NONE	No detection
0x1	RISE	Rising-edge detection
0x2	FALL	Falling-edge detection
0x3	BOTH	Both-edge detection
0x4	HIGH	High-level detection
0x5	LOW	Low-level detection
0x6 - 0x7	-	Reserved

23.8.11 Debouncer Enable

Name: DEBOUNCEN
Offset: 0x30
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

23.8.12 Debouncer Prescaler

Name: DPRESCALER
Offset: 0x34
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								RW
Reset								0

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

Bit 16 – TICKON Pin Sampler frequency selection
This bit selects the clock used for the sampling of bounce during transition detection.

Value	Description
0	The bounce sampler is using GCLK_EIC.
1	The bounce sampler is using the low frequency clock.

23.8.13 Pin State

Name: PINSTATE
Offset: 0x38
Reset: 0x00000000

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

24. Flash Memory

24.1 Overview

The PIC32CX-BZ2 devices contain a single bank of Flash memory with their Program Flash Memory (PFM) partition and Boot Flash Memory (BFM) partition for storing user code or non-volatile data. The Flash controller is used to access the Flash memory. The peripheral bus interface is used for commands and configuration of the Flash controller.

24.2 Features

Flash Controller

- PB-Bridge-D interface that provides access to the Flash controller registers
- AHB Initiator for bus hosted reads the row programming data from SRAM
- Write Protect for Program Flash (PFM)
  - Single page protection resolution
  - Protect “Less Than” Address
  - Protect “Greater Than or Equal to” Address
- Individual page write protection for boot Flash (BFM)
- Error-correction code (ECC) support
- Supports chip and page erase
- Supports Single Word, Quad Word and row program options
- Supports flash Erase/Retry to increase Retention and Endurance

Flash Memory

- 128-bit wide Flash Memory Access
- 4 Kbytes page size
- Row size is 1 KB (256 IW)
- Flash-based OTP (one-time-programmable) page

The Flash controller allows the Flash memory to be accessed through the following methods:

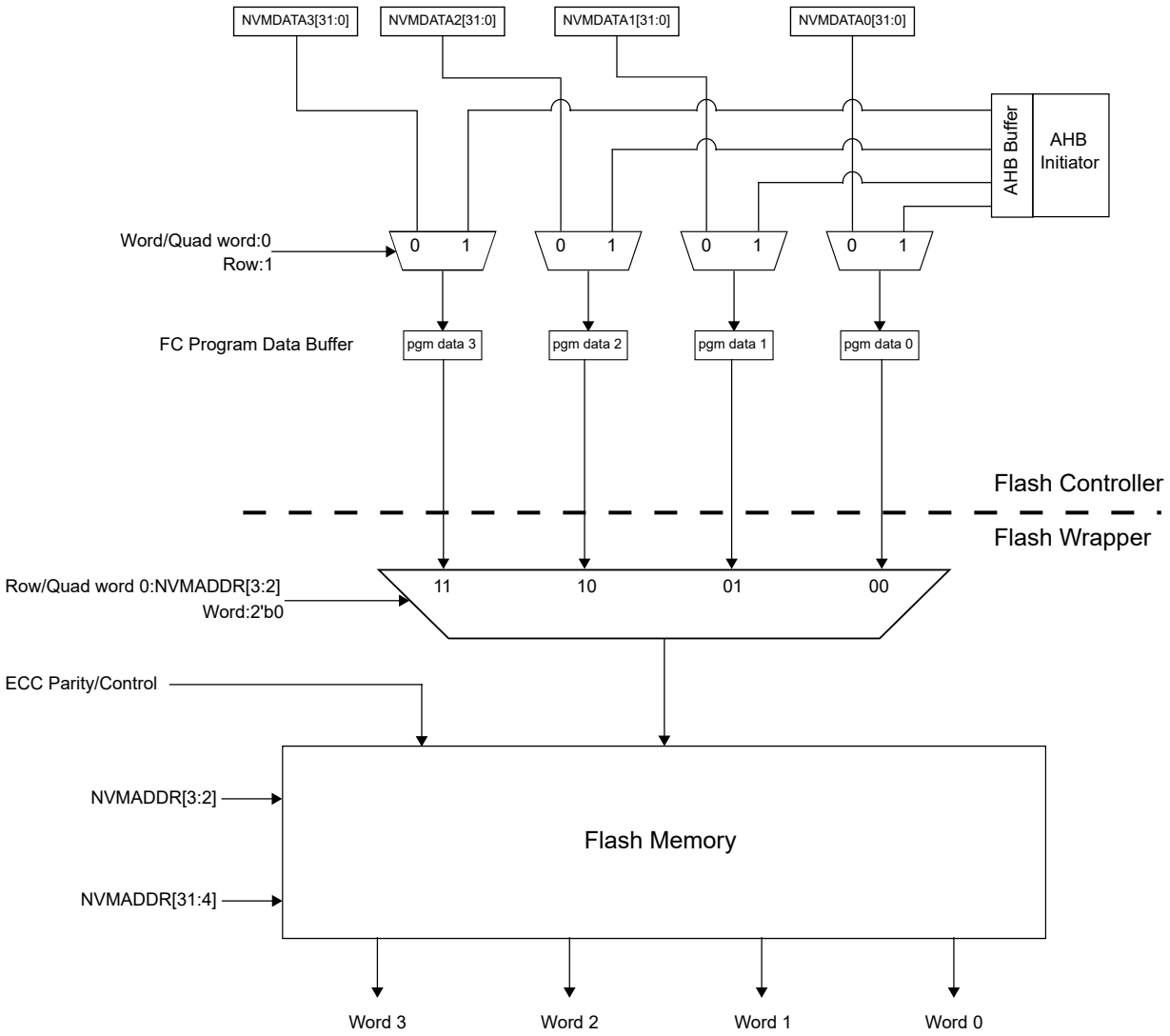
1. Run-Time Self-Programming (RTSP)
2. Serial Wire Debug (SWD) programming using DSU (See *Device Service Unit (DSU)* from Related Links and *PIC32CX-BZ2 Programming Specification*.)

Related Links

12. Device Service Unit (DSU)

24.3 Functional Block Diagram

Figure 24-1. Flash Memory Block Diagram



24.4 Flash Memory Addressing

Flash memory addressing uses physical addresses only. For more information on addressing, see *Product Memory Mapping Overview* from Related Links.

Related Links

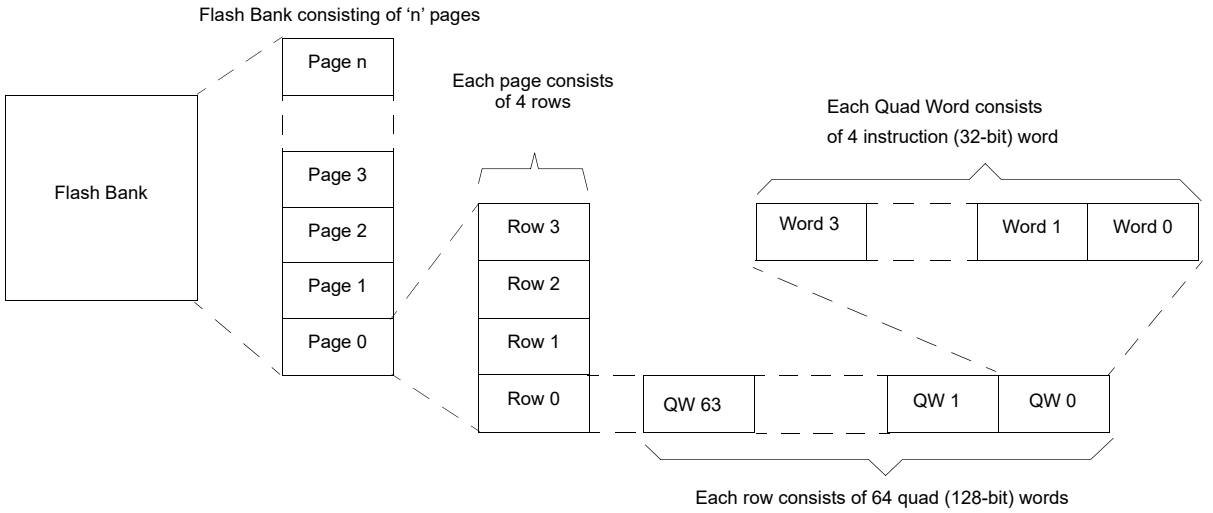
8. Product Memory Mapping Overview

24.5 Memory Configuration

24.5.1 Flash Memory Construction

Flash memory is divided into pages. A page is the smallest unit of memory that can be erased at one time. Each page of memory is segmented into four rows. A row is the largest unit of memory that can be programmed at one time. A row consists of 64 Quad (128-bit) Word. Each Quad Word consists of a four instruction (32-bit) Word. Flash memory can be programmed in rows, Quad Word (128-bit) or Single Word (32-bit) units.

Figure 24-2. Flash Construction



24.5.2 Flash Memory Organization

The Device Flash memory is divided into two logical Flash partitions:

1. Main Program Flash Memory (PFM)
2. Boot/Configuration Flash Memory (BFM)
   a. Boot Flash
   b. Device/Boot Configuration – Device and boot configuration data
   c. OTP (One Time Programmable) – User system calibration data

Each Flash section has a different protection status; refer to the following table.

Table 24-1. Protection Status

Flash Partition	Memory Region	Write Protection	Erase Protection	Chip Erase through DSU
BFM	Boot Flash	Yes. Page-wise Configurable	Yes. Page-wise Configurable	Erased
	Device/Boot Configuration	Yes. Configurable	Yes. Configurable	Erased
	OTP (One-Time-Programmable)	Yes. Configurable	Always Erase protected. Can not be erased	Not Erased
PFM	Program Flash	Yes. Configurable	Yes. Configurable	Erased

24.6 Boot Flash Memory (BFM) Partitions

24.6.1 BFM Write Protection

Pages in the BFM regions can be protected individually using bits in the NVMLBWP register. At Reset, all pages are in a write-protected state and must be disabled prior to performing any programming operations on the BFM regions. There is also an unlock bit, ULOCK(NVMLBWP[31]), that is set at Reset and can be cleared by the user software. When cleared, changes to write protection for that region can no longer be made. Once cleared, the ULOCK bit can only be set by a Reset.

The NVMLBWP write-protect register can only be changed when the unlock sequence is followed. See *NVMKEY Register Unlocking Sequence* from Related Links.

Related Links

24.11. NVMKEY Register Unlocking Sequence

24.7 Program Flash Memory (PFM) Partitions

24.7.1 PFM Write Protection

Write protection for the PFM region is implemented by pages, defined by the NVMPWPLT and NVMPWPGTE registers. The NVMPWP* registers define an area within the program space (PFM) that is write-protected. This write-protected address resolves to Flash page boundaries; therefore, the 12 LSBs for a 4 KB page Flash of any address written to the NVMPWP* registers are ignored. The width of each NVMPWP* address register is determined by the size of the Flash. The NVMPWPLT register is used to set the Program Flash pages lower than the provided address as write-protected. The NVMPWPGTE register is used to set the Program Flash pages greater than or equal to the provided address as write-protected. Therefore, a value of all `0`s in the NVMPWPLT register and all `1`s in the NVMPWPGTE register results in no region of Flash being write-protected (default state at Reset).

There is also an unlock bit, ULOCK (NVMPWPLT [31] and NVMPWPGTE[31]), that is set at Reset and can be cleared by the user software. When cleared, changes to the write-protection of the PFM can no longer be made, including the ULOCK bit. The NVMPWPLT and NVMPWPGTE write-protected register can only be changed when the unlock sequence is followed. See *NVMKEY Register Unlocking Sequence* from Related Links.

Related Links

24.11. NVMKEY Register Unlocking Sequence

24.8 Error Correcting Code (ECC) and Flash Programming

The PIC32CX-BZ2 devices incorporate Error Correcting Code (ECC) features that detect and correct errors resulting in extended Flash memory life. For more details on this feature, see *Prefetch Cache* from Related Links.

ECC is implemented in 128-bit Quad Flash Words or 32-bit Single Word. As a result, when programming Flash memory on a device where ECC is employed, the programming operation must be, at minimum, four instruction Words or in groups of four instruction Words. This is the reason that the Quad Word programming command exists and why row programming always programs multiples of four Words.

For a given software application, ECC can be enabled at all times, disabled at all times or dynamically enabled using the ECCCTL Configuration bits in the CFGCON0 register. When ECC is enabled at all times, the Single Word NVMOP programming command does not function and the Quad Word is the smallest unit of memory that can be programmed. When ECC is disabled or enabled dynamically, both the Single Word and Quad Word programming NVMOP commands are functional and the programming method used determines how ECC is handled.

In the case of dynamic ECC, if the memory was programmed with the Singe Word command, ECC is turned off for that Word, and, when it is read, no error correction is performed. If the memory was programmed with the Quad Word or Row Programming commands, ECC data is written and tested for errors (and corrected if needed) when read. The following table describes the different ECC scenarios.

Table 24-2. ECC Programming Summary

ECCCTL Setting	Programming Operation			Data Read
	Single Word Write	Quad Word Write	Row Write	
Disabled	Allowed	Allowed	Allowed	ECC is never applied on a Flash read
Enabled	Not allowed	Allowed	Allowed	ECC is applied on every Flash Word read

.....continued				
ECCCTL Setting	Programming Operation			Data Read
	Single Word Write	Quad Word Write	Row Write	
Dynamic	Allowed but when used, the programmed word is flagged to NOT USE ECC	Writes ECC data and flags programmed words to USE ECC	Writes ECC data and flags programmed words to USE ECC	ECC is only applied on words that are flagged to USE ECC

Note: When using dynamic ECC, all non-ECC locations must be programmed with the 32-bit Word programming command, while all ECC-enabled locations must be programmed with a 128-bit Quad Word or Row programming command. Divisions between ECC and non-ECC memory must be on even Quad Word boundaries (address bits 0 through 3 are equal to '0').

Related Links

9. Prefetch Cache (PCHE)

24.9 Interrupts

An interrupt is generated when the WR bit is cleared by the Flash Controller upon completion of a Flash program or erase operation. The interrupt event will cause a CPU interrupt if it was configured and enabled in the Nested Interrupt Vector Controller. See *Nested Vector Interrupt Controller (NVIC)* from Related Links for the vector mapping table. The interrupt occurs regardless of the outcome of the program or erase operation, successful or unsuccessful. The only exception is the No Operation (NOP) programming operation (NVMOP = 0), which is used to manually clear the error flags and does not create an interrupt event on completion but does clear the WR bit.

The Flash Controller interrupts are not persistent, and, therefore, no additional steps are required to clear the cause or source of the interrupt.

Once the Interrupt Controller is configured, the Flash event causes the CPU to jump to the vector assigned to the Flash event. The CPU starts executing the code at the vector address. The user software at this vector address must perform the required operations and, then, exit.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)

24.9.1 Interrupts and CPU Stalling

Code cannot be fetched by the CPU from the same Flash bank, either BFM or PFM, that is the target of the programming operation. When this operation is attempted, the CPU will cease to execute code (stall) while the programming operation is in progress. CPU code execution does not resume until the programming operation is complete, and, when this occurs, any pending interrupts, including those from the Flash Controller, will be processed in order of priority.

Note: Code that is already loaded into the processor cache will continue to execute up to the point where an attempt is made to fetch code or data from the same Flash bank as the active programming operation. At this point the CPU will stall.

The stalling of the CPU can also be avoided by placing any needed executable code in SRAM during Flash programming.

24.10 Error Detection

The NVMCON register includes two bits for detecting error conditions during a program or erase operation. They are Low-Voltage detect error, LVDERR bit (NVMCON[12]), and Write Error, WRERR bit (NVMCON[13]).

The WRERR is set each time the WR bit (NVMCON[15]) is set, initiating a programming operation. When the Flash operation is complete, indicated by hardware clearing the value of the WR bit (i.e., WR bit is set to '0'), hardware will update the value in the WRERR bit to indicate if an error occurred. Firmware must check the value of the WR bit to see if the Flash operation completed

before checking the value of the WRERR bit. When the WRERR bit is set, any future attempt to initiate programming or erase operation is ignored. WRERR must be cleared before commencing Flash program or erase operations.

The LVDERR bit is set when a Brown-out Reset (BOR) occurs during a programming operation. The only Reset that clears the LVDERR bit is a Power-on Reset (POR). Other Reset types do not affect the LVDERR bit. When the LVDERR bit is set, any attempt to initiate programming or erase operation is ignored. The LVDERR bit must be cleared before commencing Flash program or erase operations.

Both the WRERR and LVDERR bits must be cleared manually in software by initiating a Flash operation (setting WR) referred to as NOP (0x00) (see the NVMOP bit fields).
Note: Executing the NVMOP NOP command clears WRERR, LVDERR and WR bits, but does not generate an interrupt event on completion.

Table 24-3. Programming Error Cause and Effects

Cause of Error	Effect on Programming Erase Operation	Indication
A low-voltage event occurred during a programming sequence.	The last programming or erase operation may not have completed.	LVDERR = 1, WRERR = 1
A non-POR Reset occurred during programming.	Programming or erase operation is aborted.	WRERR = 1
Attempt to program or erase a page out of the Flash memory range.	Erase or programming operation is not initiated.	WRERR = 1
Attempt to erase or program a write-protected PFM page.	Erase or programming operation is not initiated.	WRERR = 1
Attempt to erase or program a write-protected BFM page.	Operation occurs, but the page is not programmed or erased.	WRERR = 0
Bus host error or row programming data underrun error during programming.	Programming or erase operation is aborted.	WRERR = 1

24.11 NVMKEY Register Unlocking Sequence

Important register settings that could compromise the Flash memory if inadvertently changed are protected by a register unlocking sequence. This feature is implemented using the NVMKEY register. The NVMKEY register is a write-only register that is used to implement an unlock sequence to help prevent accidental writes or erasures of Flash memory.

In some instances, the operation is also dependent on the setting of the WREN bit (NVMCON[14]), as shown in the following table.

Table 24-4. NVMKEY Register Unlocking and WREN

Operation	WREN Setting	Unlock Sequence Required
Changing value of NVMOP[3:0] (NVMCON[3:0])	0	No
Setting WR (NVMCON[15]) to start a write or erase operation	1	Yes
Changing any fields in the NVMPWP* register	—	Yes
Changing any fields in the NVMLBWP register	—	Yes

The following steps must be followed in the exact order as shown to enable writes to registers that require this unlock sequence:

1. Write 0x00000000 to NVMKEY.
2. Write 0xAA996655 to NVMKEY.

3. Write 0x556699AA to NVMKEY.
4. Write the value to the register NVMCON, NVMCON2, NVMPWP* or NVMLBWP requiring the unlock sequence.

When using the unlock sequence to set or clear bits in the NVMCON register, as shown in Step 4, Steps 2 through 4 must be executed without any other activity on the peripheral bus that is in use by the Flash Controller. Interrupts and DMA transfers that access the same peripheral bus as the Flash Controller must be disabled. In addition, the operation in Step 4 must be atomic. The Set, Clear and Invert registers may be used, where applicable, for the target register in Step 4.

The following code shows code written in the C language to initiate a NVM Operation (NVMOP) command. In this particular example, the WR bit is being set in the NVMCON register and, therefore, must include the unlock sequence.

Initiate NVM Operation (System Unlock Sequence Example):

```
void NVMInitiateOperation(void)
{
  // Disable Interrupts
  asm volatile("di%0" : "=r"(int_status));
  uint32_t globalInterruptState= __get_PRIMASK();
// Disable Interrupts
__disable_irq();
  NVMKEY = 0x0;
  NVMKEY = 0xAA996655;
  NVMKEY = 0x556699AA;
  NVMCONSET = 1 << 15;// must be an atomic instruction

  // Restore Interrupts
  __set_PRIMASK(globalInterruptState);
}
```

Note: Once the unlock codes are written to the NVMKEY register, the next activity on the same peripheral bus as the Flash Controller will Reset the lock. As a result, only atomic operations can be used. Use of the NVMCONSET register sets the WR bit in a single instruction without changing other bits in the register. Using NVMCONbits.WR = 1 will fail, as this line of code compiles to a read-modify-write sequence.

24.12 Word Programming

The smallest block of data that can be programmed in a single operation is one Flash write Word (32-bit). The data to be programmed must be written to the NVMDATA0 register, and the address of the Word must be loaded into the NVMADDR register before the programming sequence is initiated. The instruction Word at the physical location pointed to by the NVMADDR register is, then, programmed. Programming occurs on 32-bit Word boundaries; therefore, bits '0' and '1' of the NVMADDR register are ignored.

When a Word is programmed, it must be erased before it can be programmed again, even if changing a bit from an erased '1' state to a '0' state.

Word programming will only succeed if the target address is in a page that is not write-protected. Programming to a write-protected PFM page will fail and result in the WRERR bit being set in the NVMCON register. Programming a write-protected BFM page will fail but does not set the WRERR bit.

A programming sequence consists of the following steps:

1. Write 32-bit data to be programmed to the NVMDATA0 register.
2. Load the NVMADDR register with the address to be programmed.
3. Set the WREN bit = 1 and NVMOP bits = 1 in the NVMCON register. This defines and enables the programming operation.
4. Initiate the programming operation. (See *NVMKEY Register Unlocking Sequence* from Related Links.)

5. Monitor the WR bit of the NVMCON register to flag completion of the operation.
6. Clear the WREN bit in the NVMCON register.
7. Check for errors and process accordingly.

The following code shows code for Word programming, where a value of 0x12345678 is programmed into location 0x1008000.

Word Programming Code Example:

```
…
// Set up Address and Data Registers
  NVMADDR= 0x1008000;          // physical address
  NVMDATA0 = 0x12345678;        // value

// set the operation, assumes WREN = 0
  NVMCONbits.NVMOP = 0x1;         // NVMOP for Word programming

// Enable Flash for write operation and set the NVMOP
  NVMCONbits.WREN = 1;

// Start programming
  NVMInitiateOperation();               // see Initiate NVM Operation (Unlock Sequence
Example)

// Wait for WR bit to clear
  while (NVMCONbits.WR);

// Disable future Flash Write/Erase operations
  NVMCONbits.WREN = 0;

// Check Error Status
  if(NVMCON & 0x3000)            // mask for WRERR and LVDERR
  {
        // process errors
    }
…
```

Related Links

24.11. NVMKEY Register Unlocking Sequence

24.13 Quad Word Programming

The process for Quad Word programming is identical to Word programming except that all four of the NVMDATAx registers are used. The value of the NVMDATA0 register is programmed at address NVMADDR, NVMDATA1 at NVMADDR + 0x4, NVMDATA2 at NVMADDR + 0x8, and NVMDATA3 at address NVMDATA + 0xC.

Quad Word programming is always performed on a Quad Word boundary; therefore, NVMADDR address bits 3 through 0 are ignored.

Quad Word programming will only succeed if the target address is in a page that is not write-protected. When a Quad Word is programmed, it must be erased before any Word in it can be programmed again, even if changing a bit from an erased '1' state to a '0' state.

Where a value of 0x11111111 is programmed into location 0x1008000, 0x22222222 into 0x1008004, 0x33333333 into 0x1008008, and 0x44444444 into location 0x100800C. Refer to the following code example for details.

Quad Word Programming Code Example:

```
…

// Set up Address and Data Registers
    NVMADDR  = 0x1008000;      // physical address
    NVMDATA0 = 0x11111111;     // value written to 0x1008000
    NVMDATA1 = 0x22222222;     // value written to 0x1008004
    NVMDATA2 = 0x33333333;     // value written to 0x1008008
    NVMDATA3 = 0x44444444;     // value written to 0x100800C
```

```
// set the operation, assumes WREN = 0
    NVMCONbits.NVMOP = 0x2;    // NVMOP for Quad Word programming

// Enable Flash for write operation and set the NVMOP
    NVMCONbits.WREN = 1;

// Start programming
    NVMInitiateOperation();    // see Initiate NVM Operation (Unlock Sequence Example)

// Wait for WR bit to clear
    while(NVMCON & NVMCON_WR);

// Disable future Flash Write/Erase operations
    NVMCONbits.WREN = 0;

// Check Error Status
    if(NVMCON & 0x3000)    // mask for WRERR and LVDERR bits
```

24.14 Row Programming

The largest block of data that can be programmed is a row.

Unlike Word and Quad Word Programming where the data source is stored in SFR memory, Row programming source data is stored in SRAM. The NVMSRCADDR register is a pointer to the physical location of the source data for Row programming.

Like other Non-Volatile Memory (NVM) programming commands, the NVMADDR register points to the target address of the operation. Row programming always occurs on row boundaries with the row size of 1024, bits 0 through 9 of the NVMADDR register are ignored.

Row Word programming will only succeed if the target address is in a page that is not write-protected. When a row is programmed, it must be erased before any Word in it can be programmed again, even if changing a bit from an erased '1' state to a '0' state.

Array `rowbuff` is populated with data and programmed into a row located at physical address `0x1008000`.

Note: When assigning the value to the NVMSRCADDR register, it must be converted to a physical address.

Row Programming Code Example:

```
…

unsigned long rowbuff[256]; // example is for a 256 Word row size.
int x;                      // loop counter

// put some data in the source buffer
      for (x = 0; x < (sizeof(rowbuff) * sizeof (int)); x++)
            ((char *)rowbuff)[x] = x;

// set destination row address
     NVMADDR = 0x1008000;             // row physical address

// set source address. Must be converted to a physical address.
     NVMSRCADDR = (unsigned int)((int)rowbuff & 0x1FFFFFF);

// define Flash operation
      NVMCONbits.NVMOP = 0x3;            // NVMOP for Row programming

// Enable Flash Write
      NVMCONbits.WREN = 1;

// commence programming
      NVMInitiateOperation();            // see Initiate NVM Operation (Unlock Sequence
Example)

// Wait for WR bit to clear
      while(NVMCONbits.WR);

// Disable future Flash Write/Erase operations
       NVMCONbits.WREN = 0;
```

```
// Check Error Status
       if(NVMCON & 0x3000)            // mask for WRERR and LVDERR bits
      {
         // process errors
       }
…
```

24.15 Page Erase

A Page Erase performs an erase of a single page of either PFM or BFM.

The page to be erased is selected using the NVMADDR register. Pages are always erased on page boundaries; therefore, for a device with an instruction Word page size of 4096, bits 0 through 11 of the NVMADDR register are ignored.

A Page Erase will only succeed if the target address is a page that is not write-protected. Erasing a write-protected page will fail and result in the WRERR bit being set in the NVMCON register.

The following code shows the code for a single Page Erase operation at address 0x1008000.

Page Erase Code Example:

```
…

// set destination page address
    NVMADDR = 0x1008000;     // page physical address

// define Flash operation
    NVMCONbits.NVMOP = 0x4;    // NVMOP for Page Erase

// Enable Flash Write
    NVMCONbits.WREN = 1;

// commence programming
    NVMInitiateOperation();    // see Initiate NVM Operation (Unlock Sequence Example)

// Wait for WR bit to clear
    while(NVMCONbits.WR);

// Disable future Flash Write/Erase operations
    NVMCONbits.WREN = 0;

// Check Error Status
    if(NVMCON & 0x3000)    // mask for WRERR and LVDERR bits
    {
        // process errors
    }

…
```

24.15.1 Page Erase Retry

Page Erase Retry is a method to improve the life of a Flash by attempting to erase again if the Page Erase was not successful. Page Erase Retry can only be used for a Page Erase.

Page Erase Retry works by increasing the voltage used on the Flash when erasing. Initially, the minimum voltage necessary is applied by setting the RETRY[1:0] bits (NVMCON2[9:8]) = 00. If the page erase is not successful, the voltage may be increased by incrementing the setting of the RETRY[1:0] bits.
Note: Each Flash page, as it ages and wears, may have different voltage requirements; therefore, a higher setting on one Flash page does not indicate that the same setting must be used on all pages.

The maximum voltage for Page Erase is used when the RETRY[1:0] bits = `11`. If Page Erase is not successful after 7 trials, this means that the Flash for that page, or the Words that did not erase, must be considered “non-functional”.

Together with the normal Page Erase controls, Page Erase Retry also uses the WS[4:0], CREAD1, VREAD1 and RETRY[1:0] bits in the NVMCON2 register. The ERS[3:0] bits (NVMCON2[31:28]) are for the benefit of software performing the programming sequence in the event that a drop in power causes a BOR event but not a POR event.

Perform the following steps to set up a Page Erase Retry:

1. Set the NVMADDR register with the address of the page to be erased.
2. Execute the write unlock sequence.
3. Save the value of the NVMCON2 register.
4. Do the following in the NVMCON2 register:
   a. Set the ERS[3:0] bits as desired.
   b. Set the WS[4:0] bits per the description.
   c. Set the VREAD1 bit to '1'.
   d. Set the CREAD1 bit to '1'.
   e. Set the RETRY[1:0] bits to '00'.
5. Run the unlock sequence using the Page Erase command to start the sequence.
6. Wait for the WR bit (NVMCON[15]) to be cleared by hardware.
7. Clear the WREN bit (NVMCON[14]).
8. Verify the erase using the CPU. To shorten the verify time, use CREAD1 = 1 to perform a hardware compare to logic '1' of each bit in the Flash Word including ECC. A successful compare yields a read of 0x00000001 in the lowest addressed word in a Flash Word (128 bits). This is the Compare Word. All other Words are 0x00010000. If any bit is logic '0', all Words in the Flash Word read 0x00000000. Remember to increment the address by the number of bytes in a Flash Word between reads.
9. If all Compare Words verify correctly, the Page Erase Retry process is complete. Go to step 11.
10. If a Compare Word yields a read of 0x00000000, perform steps 4 through 9 up to six more times with the following change to step 4:
    a. Increment the RETRY[1:0] bits by one if the bit has not already reached the '11' setting.
    b. Maintain all other fields.
11. Restore the value of the NVMCON2 register, which was saved in step 3.

Notes:

1. When the VREAD1 = 1, the Flash uses the WS[3:0] bits for Flash access wait state generation to the panel selected by NVMADDR. Software is responsible for writing the VREAD1 bit back to '0' when both erase and verify is complete.
2. The device configuration boot page (the page containing the DEVCFGx values) does not support Page Erase Retry.

The following code provides code for a single page erase operation at address 0x1008000, where Page Erase Retry is used.

Page Erase Retry Code Example:

```
uint32_tsaveNVMCON2;
uint32_t*cmpPtr;
uint8_terased;
uint8_ttryCount;

// set destination page address
    NVMADDR = 0x1008000;    // Page physical address

// define flash operation
    NVMCONbits.NVMOP = 0x4;    // NVMOP for Page Erase

// Unlock sequence
    NVMKEY = 0x0;
    NVMKEY = 0xAA996655;
    NVMKEY = 0x556699AA;

// save NVMCON2
```

```
    saveNVMCON2 = NVMCON2;

// set up Page Erase Retry
    NVMCON2bits.ERS = 0;    // Stage 0 - SW use only
    NVMCON2bits.VREAD1 = 1;
    NVMCON2bits.CREAD1 = 1;
    NVMCON2bits.RETRY = 0b00;

    tryCount = 0;        // Up to 4 attempts

    do {
        tryCount++;

        // commence programming
        NVMInitiateOperation();

        // Wait for WR bit to clear
        while(NVMCONbits.WR);

        // Turn off WREN
        NVMCONbits.WREN = 0;

        // Check that the page was erased
        erased = 1;
        cmpPtr = (uint32_t *)NVMADDR;
        erased &= (*cmpPtr == 0x00000001);
        cmpPtr++;
        erased &= (*cmpPtr == 0x00010000);
        cmpPtr++;
        erased &= (*cmpPtr == 0x00010000);
        cmpPtr++;
        erased &= (*cmpPtr == 0x00010000);

        if (!erased) {
            // Erase failed. Try with different settings.
            NVMCON2bits.RETRY++;

            NVMCONbits.NVMOP = 0x4;
            NVMCONbits.WREN = 1;
        }
    } while (!erased && (tryCount < 4));

// Restore settings
    NVMCON2 = saveNVMCON2;
```

24.16 Program Flash Memory (PFM) Erase

Program Flash memory can be erased entirely. All three discrete NVMOP values, 0111, 0110, 0101, do the same operation of erase of entire Flash. When erasing the entire PFM area, in case of RTSP (Run Time Self Programming), the code must be executing from BFM. When erasing the entire PFM area, PFM write-protection must be completely disabled.

The following code shows code for erasing the entire Flash bank.

Program Flash Erase Code Example:

```
…

// define Flash operation
    NVMCONbits.NVMOP = 0x7;        // NVMOP for entire PFM erase

// Enable Flash Write
    NVMCONbits.WREN = 1;

// commence programming
    NVMInitiateOperation();        // see Initiate NVM Operation (Unlock Sequence Example)

// Wait for WR bit to clear
    while(NVMCONbits.WR);

// Disable future Flash Write/Erase operations
    NVMCONbits.WREN = 0;

// Check Error Status
    if(NVMCON & 0x3000)        // mask for WRERR and LVDERR bits
    {
```

```
            // process errors
        }

…
```

24.17 Pre-Program

The PIC32CX-BZ2 Flash supports an option to programming that increases endurance and retention. This feature is called Pre-Program, and it requires the user to perform the programming operation twice, first, with NVMCON2.NVMPREPG = 1 and, secondly, with NVMCON2.NVMPREPG = 0. Any of the programming operations (Single, Quad, Row) can be performed with this method. In all other respects, the SFR setup is identical. To use this feature, set or clear the NVMCON2.NVMPREPG SFR bit prior to setting the NVMWR bit. Pre-Program, typically double, the native Endurance and Retention of the Flash.

24.18 Device Code Protection bit (CP)

The PIC32CX-BZ2 family of devices features code protection, which, when enabled, prevents reading of the Flash memory by an external programming device (SWD through DSU).

When code protection is enabled, it can only be disabled by erasing the device with the Chip Erase command through an external programmer. See *Device Service Unit (DSU)* from Related Links.

When programming a device that has opted to utilize code protection, the external programming device must perform verification prior to enabling code protection. Enabling code protection must be the last step of the programming process. For the location of the code protection enable bits, refer to *PIC32CX-BZ2 Programming Specification* and *System Configuration Registers (CFG)* from Related Links.

Related Links

12. Device Service Unit (DSU)
18. System Configuration and Register Locking (CFG)

24.19 Operation in Power-Saving Modes

The Flash Controller does not operate in power-saving modes. If a WAIT instruction is encountered when programming, the CPU will stop execution (stall), wait for the programming operation to complete, then enter the Power-Saving mode.

24.20 Operation in Debug Mode

Programming operations will continue to completion if the processor execution is halted in Debug mode.

24.21 Effects of Various Resets

Device Resets, other than a Power-on Reset (POR), reset the entire contents of the NVMPWP and NVMLBWP registers. All other register content persists through a non-POR reset.

All Flash Controller registers are forced to their reset states upon a POR.

24.22 Control Registers

Note: The following conventions are used in the following registers:

- R = Readable bit
- W = Writable bit
- U = Unimplemented bit, read as '0'
- 1= Bit is set
- 0= Bit is cleared
- x = Bit is unknown

- -n = Value at POR
- HS = Hardware Set
- HC = Hardware Cleared

Note: All registers in this table have corresponding CLR, SET and INV registers at its virtual address, plus an offset of 0x4, 0x8 and 0xC, respectively. See *CLR, SET and INV Registers* from Related Links.

Related Links

6.1.9. CLR, SET and INV Registers

24.22.1 Register Summary

The following registers provides a brief summary of the Flash programming-related registers.

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	NVMCON	7:0					NVMOP[3:0]			
		15:8	WR	WREN	WRERR	LVDERR				HTDPGM
		23:16								
		31:24								
0x04 ... 0x0F	Reserved									
0x10	NVMCON2	7:0								NVMPREPG
		15:8		TEMP	CREAD1	VREAD1			RETRY[1:0]	
		23:16				WS[4:0]				
		31:24	ERS[3:0]							SLEEP
0x14 ... 0x1F	Reserved									
0x20	NVMKEY	7:0	NVMKEY[7:0]							
		15:8	NVMKEY[15:8]							
		23:16	NVMKEY[23:16]							
		31:24	NVMKEY[31:24]							
0x24 ... 0x2F	Reserved									
0x30	NVMADDR	7:0	NVMADDR[7:0]							
		15:8	NVMADDR[15:8]							
		23:16	NVMADDR[23:16]							
		31:24	NVMADDR[31:24]							
0x34 ... 0x3F	Reserved									
0x40	NVMDATA0	7:0	NVMDATA[7:0]							
		15:8	NVMDATA[15:8]							
		23:16	NVMDATA[23:16]							
		31:24	NVMDATA[31:24]							
0x44 ... 0x4F	Reserved									
0x50	NVMDATA1	7:0	NVMDATA[7:0]							
		15:8	NVMDATA[15:8]							
		23:16	NVMDATA[23:16]							
		31:24	NVMDATA[31:24]							
0x54 ... 0x5F	Reserved									
0x60	NVMDATA2	7:0	NVMDATA[7:0]							
		15:8	NVMDATA[15:8]							
		23:16	NVMDATA[23:16]							
		31:24	NVMDATA[31:24]							
0x64 ... 0x6F	Reserved									
0x70	NVMDATA3	7:0	NVMDATA[7:0]							
		15:8	NVMDATA[15:8]							
		23:16	NVMDATA[23:16]							
		31:24	NVMDATA[31:24]							
0x74 ... 0xBF	Reserved									

.....continued										
Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0xC0	NVMSRCADDR	7:0	NVMSRCADDR[7:0]							
		15:8	NVMSRCADDR[15:8]							
		23:16	NVMSRCADDR[23:16]							
		31:24	NVMSRCADDR[31:24]							
0xC4 ... 0xCF	Reserved									
0xD0	NVMPWPLT	7:0	PWPLT[7:0]							
		15:8	PWPLT[15:8]							
		23:16	PWPLT[23:16]							
		31:24	ULOCK							
0xD4 ... 0xDF	Reserved									
0xE0	NVMPWPGTE	7:0	PWPGTE[7:0]							
		15:8	PWPGTE[15:8]							
		23:16	PWPGTE[23:16]							
		31:24	ULOCK							
0xE4 ... 0xEF	Reserved									
0xF0	NVMLBWP	7:0	LBWP[7:0]							
		15:8	LBWP[15:8]							
		23:16	LBWP[23:16]							
		31:24	ULOCK							

24.22.2 Register Description

The following NVM control registers control the Flash program, erase and write protection operations:

- NVMCON: Programming Control Register
  - This register is the control register for Flash program/erase operations. The following are the uses of this register:
    - Selects the operation to be performed
    - Initiates the operation
    - Provides status of the result after completing the operation
- NVMCON2: Programming Control2 Register
  - This register is the control and status register for Flash program/erase operations.
- NVMKEY: Programming Unlock Register
  - This is a write-only register that helps to or that helps the user to implement an unlock sequence to help prevent accidental writes/erasures of Flash memory and write permission settings.
- NVMADDR: Flash Address Register
  - This register stores the physical target address for row, Quad Double Word and Single Double Word programming as well as page erasing.
- NVMDATAx: Flash Program Data Register (x = 0-3)
  - These registers hold the data to be programmed during Flash Word program operations.
- NVMSRCADDR: Source Data Address Register
  - This register points to the physical address of the data to be programmed when executing a row program operation.
- NVMPWPLT: Flash Program Write Protect Lower Register
  - This register sets the program flash pages lower than provided address as a write protected.

- NVMPWPGTE: Flash Program Write Protect Greater Register
  - This register sets the program flash pages greater than provided address as a write protected.
- NVMLBWP: Flash Boot Write Protect Register
  - This register sets the boot flash partition pages as a write protected.

The following is the list of conventions available in the register description:

- – R = Readable bit
- – W = Writable bit
- – U = Unimplemented bit, read as '0'
- – -n = Value at POR
- – 1 = Bit is set
- – 0 = Bit is cleared
- – x = Bit is unknown
- HS = Hardware Set
- HC = Hardware Cleared

24.22.2.1 NVMCON – Programming Control Register

Name: NVMCON
Offset: 0x00
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	WR	WREN	WRERR	LVDERR				HTDPGM
Access	R/HS/HC	R/W	R/HS/HC	R/HS/HC				R/HS/HC
Reset	0	0	0	0				0

Bit	7	6	5	4	3	2	1	0
					NVMOP[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit 15 – WR Write Control Bit(1)

Note: This field can only be modified when WREN = 1, TEMP = 1 and the NVMKEY unlock sequence is satisfied.

Value	Description
1	Initiate a Flash operation. Hardware clears this bit when the operation completes
0	Flash operation complete or inactive

Bit 14 – WREN Write Enable Bit(1)

Value	Description
1	Enables writes to WR
0	Disables writes to WR

Bit 13 – WRERR Write Error Bit(1)

Note: Cleared by setting NVMOP == 0000b and initiating a Flash operation (WR).

Value	Description
1	Program or erase sequence did not complete successfully
0	Program or erase sequence completed normally

Bit 12 – LVDERR Low Voltage Detect Error Bit(1)

The error is only captured for programming/erase operations (when WR = 1).

Note: Cleared by setting NVMOP == 0000b and initiating a Flash operation (WR).

Value	Description
1	Low voltage is detected (possible data corruption if WRERR is set)
0	Normal voltage is detected

Bit 8 – HTDPGM High Temperature Detected during Program/Erase Operation bit
This status is only captured for programming/erase operations (when WR = 1).
Note: Cleared by setting NVMOP == 0000b and initiating a Flash operation (WR).

Value	Description
1	High temperature is detected (possible data corruption, verify operation)
0	High temperature is not detected

Bits 3:0 – NVMOP[3:0] NVM Operation bits
These bits are only writable when WREN = 0.

Value	Description
1111	Reserved
1110	Chip Erase Operation: Erases PFM, BFM (except configuration page) when accessed through SWD interface only.
...	
...	
...	
1000	Reserved
0111	Program erase operation: erase all of program Flash memory (PFM) (all pages must be unprotected)
0110	Upper program Flash memory erase operation: erases only the upper mapped region of program Flash (all pages in that region must be unprotected). It is a single bank Flash in PIC32CX-BZ2; therefore, this NVMOP performs the same as NVMOP = 0111.
0101	Lower program Flash memory erase operation: erases only the lower mapped region of program Flash (all pages in that region must be unprotected). It is a single bank Flash in PIC32CX-BZ2; therefore, this NVMOP performs the same as NVMOP = 0111.
0100	Page erase operation: erases the page selected by NVMADDR if it is not write-protected.
0011	Row program operation: programs the row selected by NVMADDR if it is not write-protected.
0010	Quad Word (128-bit) program operation: programs the 128-bit Flash Word selected by NVMADDR if it is not write-protected.
0001	Word program operation: programs the Word selected by NVMADDR if it is not write-protected(2).
0000	No operation

Notes:

1. These bits are reset by a POR only and are not affected by other Reset sources.
2. This operation results in a No Operation (NOP) when the Dynamic Flash ECC Configuration bits = 00 (ECCCTL[1:0](CFGCON0[29:28])), which enables ECC at all times. For all other ECCCTL[1:0] bit settings, this command will execute but will not write the ECC bits for the Word. It can cause DED (Double-bit Error Detected) errors if dynamic Flash ECC is enabled (ECCCTL[1:0] = 01).

24.22.2.2 NVMCON2 – Programming Control 2 Register

Name: NVMCON2
Offset: 0x10
Reset: 0x011F4000
Property: -

Bit	31	30	29	28	27	26	25	24
	ERS[3:0]							SLEEP
Access	R/W	R/W	R/W	R/W				R/W
Reset	0	0	0	0				1

Bit	23	22	21	20	19	18	17	16
				WS[4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				1	1	1	1	1

Bit	15	14	13	12	11	10	9	8
		TEMP	CREAD1	VREAD1			RETRY[1:0]	
Access		R	R/W	R/W			R/W	R/W
Reset		1	0	0			0	0

Bit	7	6	5	4	3	2	1	0
								NVMPREPG
Access								R/W
Reset								0

Bits 31:28 – ERS[3:0] Erase Retry State
These bits are used by software to track the software state of the erase retry procedure in the event of a system Reset (NMCLR) or Brown-out Reset (BOR) event.

Bit 24 – SLEEP Power Down in Sleep mode
Note: This field can only be modified when the NVMKEY unlock sequence is satisfied.

Value	Description
1	Configures Flash for power-down when the system is in Sleep mode
0	Configures Flash for standby when the system is in Sleep mode

Bits 20:16 – WS[4:0] Flash Access Wait State Control for VREAD1 = 1
Notes:
1. When VREAD1 = 1, WS[4:0] only affects the memory containing NVMADDR[31:0].
2. This field can only be modified when the NVMKEY unlock sequence is satisfied.

Value	Description
11111	31 wait states (32 total system clocks)
11110	30 wait states (31 total system clocks)
...	
00010	2 wait states (3 total system clocks)
00001	1 wait state (2 total system clocks)
00000	0 wait state (1 total system clock)

Bit 14 – TEMP Operating Temperature Control bit

Bit 13 – CREAD1 Compare Read of Logic 1 bit

Compare read 1 causes all bits in a Flash Word (including ECC if it exists) to be evaluated during the read. If all bits are '1', the lowest Word in the Flash Word evaluates to 0x0000_0001, all other Words are 0x0001_0000. If any bit is '0', the read evaluates to 0x0000_0000 for all Words in the Flash Word.

Notes:

1. When using erase retry in an ECC Flash system, CREAD1 = 1 must be used.
2. This field can only be modified when the NVMKEY unlock sequence is satisfied.

Value	Description
1	Compare read enabled only if VREAD1 = 1
0	Compare read disabled

Bit 12 – VREAD1 Verify Read of logic 1 Control bit

Notes:

1. When VREAD1 = 1, the Flash wait state control is from WS[4:0] for the memory containing NVMADDR[31:0].
2. Using Page Erase Retry and Verify Read procedure increase the life of the Flash memory.
3. This field can only be modified when NVMCON.WR == 0 and the NVMKEY unlock sequence is satisfied.

Value	Description
1	Selects erase retry procedure with verify read
0	Selects single erase without verify read

Bits 9:8 – RETRY[1:0] Erase Retry Control bit, only used when VREAD1 = 1

Note: This field can only be modified when NVMCON.WR == 0.

Value	Description
11	Erase strength for last retry cycle
10	Erase strength for third retry cycle
01	Erase strength for second retry cycle
00	Erase strength for first retry cycle

Bit 0 – NVMPREPG NVM Pre-Program Control Bit

Note: This field can only be modified when NVMCON.NVMWR= = 0.

Value	Description
1	Program Operations include the Pre-Program step
0	Program Operations exclude the Pre-Program step

24.22.2.3 NVMKEY – Programming Unlock Register

Name: NVMKEY
Offset: 0x20
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
	NVMKEY[31:24]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	NVMKEY[23:16]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	NVMKEY[15:8]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	NVMKEY[7:0]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – NVMKEY[31:0] Unlock Register bits
These bits are write-only and read '0' on any read.
Note: This register is used as part of the unlock sequence to prevent inadvertent writes to the program Flash.

24.22.2.4 NVMADDR – Flash Address Register

Name: NVMADDR
Offset: 0x30
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
	NVMADDR[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	NVMADDR[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	NVMADDR[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	NVMADDR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – NVMADDR[31:0] Flash (Word) Address bits

Table 24-5. Flash (Word) Address Bits

NVMOP	Flash Address Bits
Page Erase	• Address identifies the page to erase • Any address within a 4 Kbytes page boundary will cause the page to be erased
Row program	• Address identifies the row to program • The value of the address must be aligned to a row boundary
Word program	• Address identifies the 32-bit Word to program • NVMADDR[1:0] bits are ignored • Must be aligned to a Word boundary
Quad Word program	• Address identifies the 128-bit Quad Word to program • NVMADDR[3:0] bits are ignored • Must be aligned to a Quad Word boundary

Notes:

1. Hardware prevents writes to this register when NVMCON.WR = `1`.
2. For all other NVMOP[3:0] bit settings, the Flash address is ignored. For additional information on these bits, see the *NVMCON* register from Related Links.
3. The bits in this register are reset by a POR only and are not affected by other Reset sources.

Related Links

24.22.2.1. NVMCON

24.22.2.5 NVMDATA0 – Flash Program Data Register 0

Name: NVMDATA0
Offset: 0x40
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
	NVMDATA[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	NVMDATA[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	NVMDATA[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	NVMDATA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – NVMDATA[31:0] Flash Programming Data bits
The value in this register is written to Flash when a program operation is commanded.

- Single Word program (32-bit)
  - Writes NVMDATA0 to the target Flash address defined in NVMADDR[31:2].
- Quad Word program (128-bit)
  - Writes NVMDATA3:NVMDATA2:NVMDATA1:NVMDATA0 to the target Flash address defined in NVMADDR[31:4]. NVMDATA0 contains the Least Significant Instruction Word.

Notes:

1. Hardware prevents writes to this register when NVMCON.WR = `1`.
2. The bits in this register are reset on a POR only and are unaffected by other Reset sources.

24.22.2.6 NVMDATA1 – Flash Program Data Register 1

Name: NVMDATA1
Offset: 0x50
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
	NVMDATA[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	NVMDATA[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	NVMDATA[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	NVMDATA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – NVMDATA[31:0] Flash Programming Data bits
The value in this register is written to Flash when a program operation is commanded.

- Single Word program (32-bit)
  - Writes NVMDATA0 to the target Flash address defined in NVMADDR[31:2].
- Quad Word program (128-bit)
  - Writes NVMDATA3:NVMDATA2:NVMDATA1:NVMDATA0 to the target Flash address defined in NVMADDR[31:4]. NVMDATA0 contains the Least Significant Instruction Word.

Notes:

1. Hardware prevents writes to this register when NVMCON.WR = `1`.
2. The bits in this register are reset on a POR only and are unaffected by other Reset sources.

24.22.2.7 NVMDATA2 – Flash Program Data Register 2

Name: NVMDATA2
Offset: 0x60
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
	NVMDATA[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	NVMDATA[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	NVMDATA[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	NVMDATA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – NVMDATA[31:0] Flash Programming Data bits
The value in this register is written to Flash when a program operation is commanded.

- Single Word program (32-bit)
  - Writes NVMDATA0 to the target Flash address defined in NVMADDR[31:2].
- Quad Word program (128-bit)
  - Writes NVMDATA3:NVMDATA2:NVMDATA1:NVMDATA0 to the target Flash address defined in NVMADDR[31:4]. NVMDATA0 contains the Least Significant Instruction Word.

Notes:

1. Hardware prevents writes to this register when NVMCON.WR = `1`.
2. The bits in this register are reset on a POR only and are unaffected by other Reset sources.

24.22.2.8 NVMDATA3 – Flash Program Data Register 3

Name: NVMDATA3
Offset: 0x70
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
	NVMDATA[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	NVMDATA[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	NVMDATA[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	NVMDATA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – NVMDATA[31:0] Flash Programming Data bits
The value in this register is written to Flash when a program operation is commanded.

- Single Word program (32-bit)
  - Writes NVMDATA0 to the target Flash address defined in NVMADDR[31:2].
- Quad Word program (128-bit)
  - Writes NVMDATA3:NVMDATA2:NVMDATA1:NVMDATA0 to the target Flash address defined in NVMADDR[31:4]. NVMDATA0 contains the Least Significant Instruction Word.

Notes:

1. Hardware prevents writes to this register when NVMCON.WR = `1`.
2. The bits in this register are reset on a POR only and are unaffected by other Reset sources.

24.22.2.9 NVMSRCADDR – Source Data Address Register

Name: NVMSRCADDR
Offset: 0xC0
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
	NVMSRCADDR[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	NVMSRCADDR[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	NVMSRCADDR[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	NVMSRCADDR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – NVMSRCADDR[31:0] Source Data (Word) Address bits
This is the system physical Word address of the data (in DRM) to be programmed into the Flash when NVMCON.NVMOP is set to row programming.

Notes:

1. Hardware prevents writes to this register when NVMCON.WR = 1.
2. The bits in this register are reset on a POR only and are unaffected by other reset sources.

24.22.2.10 NVMPWPLT – Flash Program Write Protect Lower Register

Name: NVMPWPLT
Offset: 0xD0
Reset: 0x80000000
Property: -

Bit	31	30	29	28	27	26	25	24
	ULOCK							
Access	R/C							
Reset	1							

Bit	23	22	21	20	19	18	17	16
	PWPLT[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	PWPLT[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	PWPLT[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 31 – ULOCK NVMPWPLT Register Unlock bit

Notes:

1. This field can only be modified when the NVMKEY unlock sequence is satisfied.
2. This field can be cleared at the same time as writing to PWPLT[23:0].

Value	Description
1	NVMPWPLT register is not locked and can be modified
0	NVMPWPLT register is locked and cannot be modified

Bits 23:0 – PWPLT[23:0] Flash Program Write Protect Less Than Address

Pages at Flash addresses less than this value are write-protected.

Notes:

1. This field can only be modified when the NVMKEY unlock sequence is satisfied, and ULOCK = 1.
2. This is a byte address force to align to page boundaries.

24.22.2.11 NVMPWPGTE – Flash Program Write Protect Greater Register

Name: NVMPWPGTE
Offset: 0xE0
Reset: 0x80FFFFFF
Property: -

Bit	31	30	29	28	27	26	25	24
	ULOCK							
Access	R/C							
Reset	1							

Bit	23	22	21	20	19	18	17	16
	PWPGTE[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bit	15	14	13	12	11	10	9	8
	PWPGTE[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bit	7	6	5	4	3	2	1	0
	PWPGTE[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bit 31 – ULOCK NVMPWPGTE Register Unlock bit

Notes:

1. This field can only be modified when the NVMKEY unlock sequence is satisfied.
2. This field can be cleared at the same time as writing to PWPGTE[23:0].

Value	Description
1	NVMPWPGTE register is not locked and can be modified
0	NVMPWPGTE register is locked and cannot be modified

Bits 23:0 – PWPGTE[23:0] Flash Program Write Protect Address

Pages at Flash addresses greater than or equal to this value are write-protected.

Notes:

1. This field can only be modified when the NVMKEY unlock sequence is satisfied and ULOCK = 1.
2. This is a byte address forced to align to page boundaries.

24.22.2.12 NVMLBWP – Flash Boot Write Protect Register

Name: NVMLBWP
Offset: 0xF0
Reset: 0x80FFFFFF
Property: -

Bit	31	30	29	28	27	26	25	24
	ULOCK							
Access	R/C							
Reset	1							

Bit	23	22	21	20	19	18	17	16
	LBWP[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bit	15	14	13	12	11	10	9	8
	LBWP[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bit	7	6	5	4	3	2	1	0
	LBWP[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bit 31 – ULOCK Lower Boot Write Protect (LBWPn) Unlock bit

Notes:

1. This field can only be modified when the NVMKEY unlock sequence is satisfied.
2. This field can be cleared at the same time as writing to LBWP[msb:lsb].

Value	Description
1	LBWPn bits are not locked and can be modified
0	LBWPn bits are locked and cannot be modified

Bits 23:0 – LBWP[23:0] Boot Pages Write Protect bits

Notes:

1. This field can only be modified when the NVMKEY unlock sequence is satisfied and ULOCK = 1.
2. The OTP page is always erase-protected and its associated LBWP bit is only for write-protection.

Value	Description
1	Erase and write-protection for upper boot page n is enabled
0	Erase and write-protection for upper boot page n is disabled

25. Integrity Check Monitor (ICM)

25.1 Overview

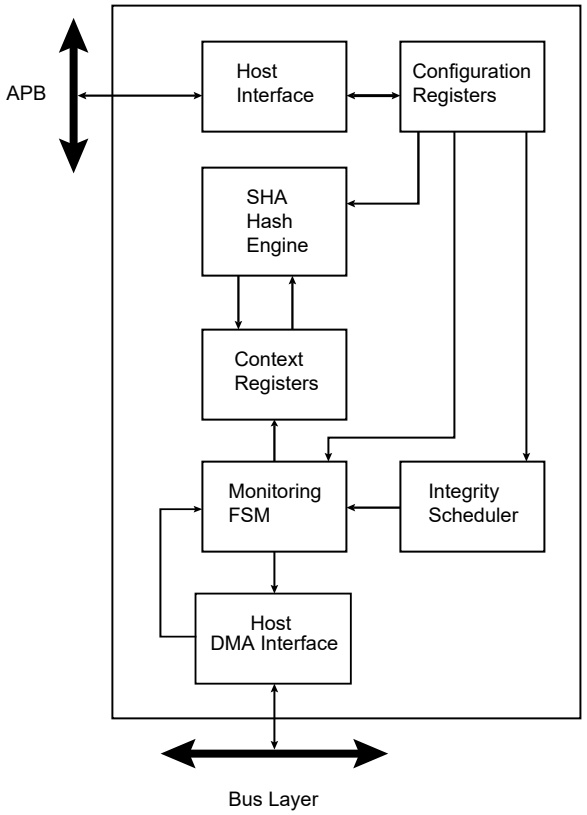
The Integrity Check Monitor (ICM) is a DMA controller that performs hash calculation over multiple memory regions using transfer descriptors located in memory (ICM Descriptor Area). The Hash function is based on the Secure Hash Algorithm (SHA). The ICM controller integrates two modes of operation. The first mode is used to hash a list of memory regions and save the digests to memory (ICM Hash Area). The second mode is an active monitoring of the memory. In this mode, the hash function is evaluated and compared to the digest located at a predefined memory address (ICM Hash Area). If a mismatch occurs, an interrupt is raised.

25.2 Features

- DMA AHB manager interface
- Supports monitoring of up to four non-contiguous memory regions
- Supports block gathering using a linked list
- Supports Secure Hash Algorithm (SHA1, SHA256)
- Compliant with FIPS Publication 180-2
- Configurable processing period:
  - When SHA1 algorithm is processed, the run-time period is either 85 or 209 clock cycles
  - When SHA256 algorithm is processed, the run-time period is either 72 or 194 clock cycles
- Programmable bus burden

25.3 Block Diagram

Figure 25-1. Integrity Check Monitor Block Diagram



25.4 Signal Description

Not applicable.

25.5 Product Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

25.5.1 Power Management

25.5.2 Clocks

The ICM bus clocks (PB2_CLK) can be enabled and disabled in the CRU module or the PMD2.ICMMD bit. For more details, see *Peripheral Module Disable Register (PMD)* from Related Links.

Related Links

20. Peripheral Module Disable Register (PMD)

25.5.3 DMA

Not applicable.

25.5.4 Events

Not applicable.

25.5.5 Debug Operation

Not applicable.

25.6 Functional Description

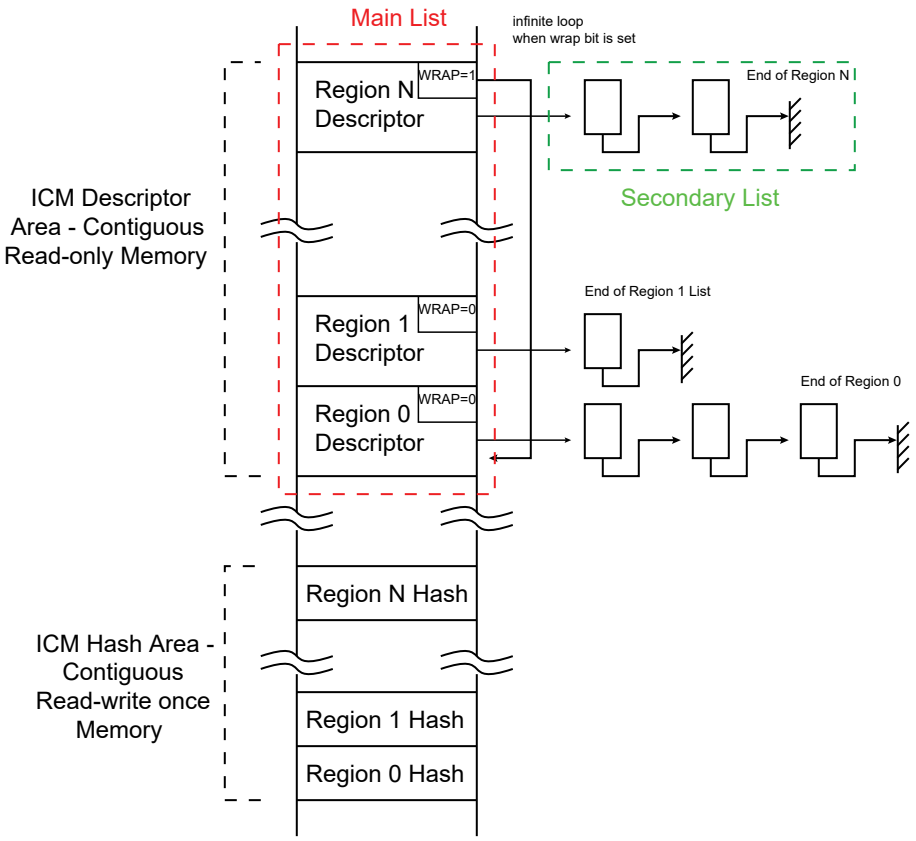
25.6.1 Overview

The Integrity Check Monitor (ICM) is a DMA controller that performs SHA-based memory hashing over memory regions. As shown in the Block Diagram (see *Block Diagram* from Related Links), it integrates a DMA interface, a Monitoring Finite State Machine (FSM), an integrity scheduler, a set of context registers, a SHA engine, an interface for configuration and status registers.

The SHA engine requires a message padded according to FIPS180-4 specification when used as a SHA calculation unit only. Otherwise, if the ICM is used as an integrated check for memory content, the padding is not mandatory. The SHA module produces an N-bit message digest each time a block is read and a processing period ends. N is 160 for SHA1, 256 for SHA256.

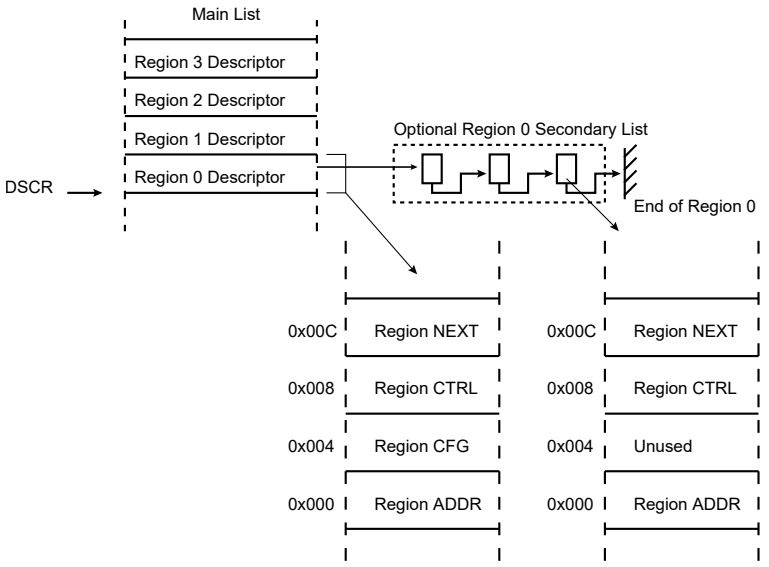
When the ICM module is enabled, it sequentially retrieves a circular list of region descriptors from the memory (Main List described in the following figure). Up to four regions may be monitored. Each region descriptor is composed of four words indicating the layout of the memory region (see *Region Descriptor Structure* from Related Links). It also contains the hashing engine configuration on a per region basis. As soon as the descriptor is loaded from the memory and context registers are updated with the data structure, the hashing operation starts. A programmable number of blocks (see TRSIZE field of the RCTRL structure member) is transferred from the memory to the SHA engine. When the desired number of blocks have transferred, the digest is either moved to memory (Write Back function) or compared with a digest reference located in the system memory (Compare function). If a digest mismatch occurs, an interrupt is triggered if enabled. The ICM module parses through the region descriptor list until the end of the list, marked by an end of list bit set to one. To continuously monitor the list of regions, the WRAP bit must be set to one in the last data structure, and EOM must be cleared.

Figure 25-2. ICM Region Descriptor and Hash Areas



Each region descriptor supports gathering of data through the use of the Secondary List. Unlike the Main List, the Secondary List cannot modify the configuration attributes of the region. When the end of the Secondary List is encountered, the ICM returns to the Main List. Memory integrity monitoring can be considered a background service, and the mandatory bandwidth is very limited. To limit the ICM memory bandwidth, use the BBC field of the CFG register to control the ICM memory load.

Figure 25-3. Region Descriptor



Related Links

25.3. Block Diagram
25.6.3. Region Descriptor Structure

25.6.2 ICM Hash Area

The ICM Hash Area is a contiguous area of system memory that the controller and the processor can access. The physical location is configured in the ICM hash area start address register. This address is a multiple of bytes. If the CDWBN bit of the context register is cleared (i.e., Write Back activated), the ICM controller performs a digest write operation at the following starting location: *(HASH) + (RID<<), where RID is the current region context identifier. If the CDWBN bit of the context register is set (i.e., Digest Comparison activated), the ICM controller performs a digest read operation at the same address.

25.6.2.1 Message Digest Example

Considering the following 512 bits message (example given in FIPS 180-4):

“61626380000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000018”

The message is written to memory in a Little Endian (LE) system architecture.

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000	80	63	62	61
0x004–0x038	00	00	00	00
0x03C	18	00	00	00

The digest is stored at the memory location pointed at by the ICM_HASH pointer with a Region Offset.

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000	36	3e	99	a9
0x004	6a	81	06	47
0x008	71	25	3e	ba
0x00C	6c	c2	50	78
0x010	9d	d8	d0	9c

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000	22	7d	09	23
0x004	22	d8	05	34
0x008	77	a4	42	86
0x00C	b3	55	a2	bd
0x010	e4	bc	ad	2a
0x014	f7	b3	a0	bd
0x018	a7	9d	6c	e3

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000	bf	16	78	ba
0x004	ea	cf	01	8f
0x008	de	40	41	41

...........continued

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x00C	23	22	ae	5d
0x010	a3	61	03	b0
0x014	9c	7a	17	96
0x018	61	ff	10	b4
0x01C	ad	15	00	f2

Considering the following 1024 bits message (example given in FIPS 180-4):

“6162638000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000018”

The message is written to memory in a Little Endian (LE) system architecture.

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000	80	63	62	61
0x004–0x078	00	00	00	00
0x07C	18	00	00	00

25.6.3 Region Descriptor Structure

The ICM Region Descriptor Area is a contiguous area of system memory that the controller and the processor can access. When the ICM controller is activated, the controller performs a descriptor fetch operation at the DSCR address. If the Main List contains more than one descriptor (i.e., more than one region is to be moderated), the fetch address is DSCR + RID<<4, where RID is the region identifier.

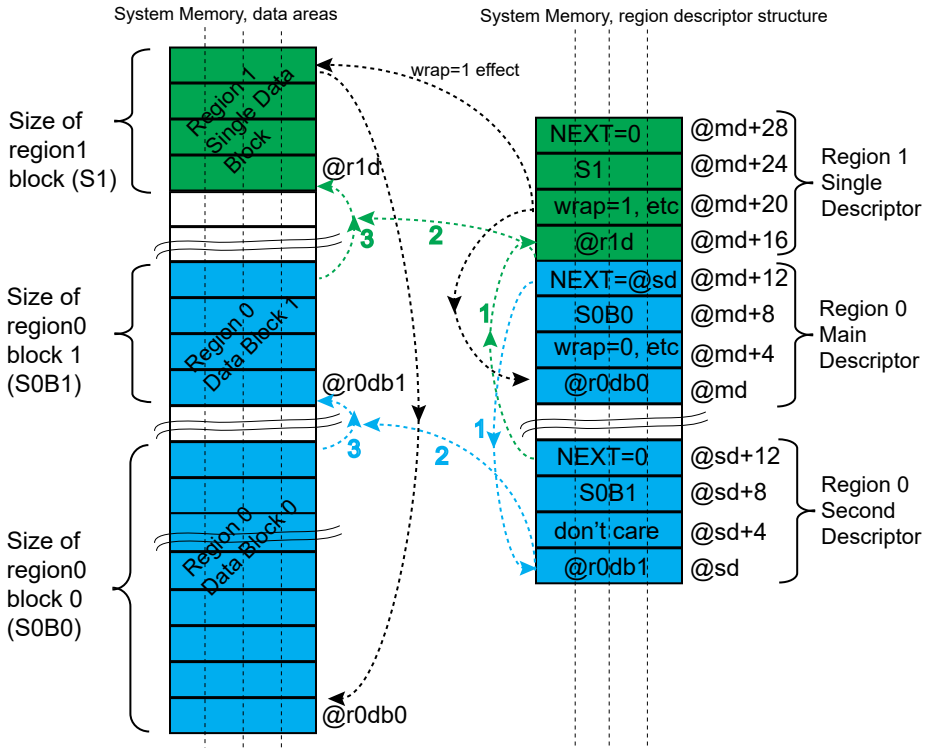
Table 25-1. Region Descriptor Structure (Main List)

Offset	Structure Member	Name
DSCR+0x00+RID*(0x10)	ICM Region Start Address	RADDR
DSCR+0x04+RID*(0x10)	ICM Region Configuration	RCFG
DSCR+0x08+RID*(0x10)	ICM Region Control	RCTRL
DSCR+0x0C+RID*(0x10)	ICM Region Next Address	RNEXT

Example 25-1. ICM Monitoring of 3 Memory Data Blocks (Defined as 2 Regions)

The following figure shows the mandatory ICM settings to monitor three memory data blocks of the system memory (defined as two regions), with one region being not contiguous (two separate areas) and one contiguous memory area. For each said region, the SHA algorithm may be independently selected (different for each region). The wrap allows continuous monitoring.

Figure 25-4. Example – Monitoring of 3 Memory Data Blocks (Defined as 2 Regions)



25.6.3.1 Region Descriptor Structure Overview

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	RADDR	7:0	RADDR[7:0]							
		15:8	RADDR[15:8]							
		23:16	RADDR[23:16]							
		31:24	RADDR[31:24]							
0x04	RCFG	7:0	WCIEN	BEIEN	DMIEN	RHIEN		EOM	WRAP	CDWBN
		15:8		ALGO[2:0]				PROCDLY	SUIEN	ECIEN
		23:16								
		31:24								
0x08	RCTRL	7:0	TRSIZE[7:0]							
		15:8	TRSIZE[15:8]							
		23:16								
		31:24								
0x0C	RNEXT	7:0								
		15:8								
		23:16								
		31:24								

25.6.3.1.1 Region Start Address Structure Member

Name: RADDR
Offset: 0x00
Reset: 0x00000000
Property: Read/Write

Bit	31	30	29	28	27	26	25	24
	RADDR[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	RADDR[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	RADDR[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	RADDR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – RADDR[31:0] Region Start Address
This field indicates the first byte address of the region

25.6.3.1.2 Region Configuration Structure Member

Name: RCFG
Offset: 0x04
Reset: 0x00000000
Property: Read/Write

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
		ALGO[2:0]				PROCDLY	SUIEN	ECIEN
Access		R/W	R/W	R/W		R/W	R/W	R/W
Reset		0	0	0		0	1	1

Bit	7	6	5	4	3	2	1	0
	WCIEN	BEIEN	DMIEN	RHIEN		EOM	WRAP	CDWBN
Access	R/W	R/W	R/W	R/W		R/W	R/W	R/W
Reset	1	1	1	1		0	0	0

Bits 14:12 – ALGO[2:0] User SHA Algorithm

Value	Name	Description
0	SHA1	SHA1 algorithm processed
1	SHA256	SHA256 algorithm processed
Other	-	Reserved

Bit 10 – PROCDLY Processing Delay

For a given SHA algorithm, the runtime period has two possible lengths:

Table 25-2. SHA Processing Runtime Periods

Algorithm	SHORTEST [number of cycles]	LONGEST [number of cycles]
SHA1	85	209
SHA256	72	194

Value	Name	Description
0	SHORTEST	SHA processing runtime is the shortest one
1	LONGEST	SHA processing runtime is the longest one

Bit 9 – SUIEN Monitoring Status Updated Condition Interrupt Enable

0: The RSU flag is set when the corresponding descriptor is loaded from memory to ICM.
1: The RSU flag remains cleared even if the condition is met.

Bit 8 – ECIEN End Bit Condition Interrupt Enable

0: The REC flag is set when the descriptor having the EOM bit set is processed.
1: The REC flag remains cleared even if the setting condition is met.

Bit 7 – WCIEN Wrap Condition Interrupt Disable
0: The RWC flag is set when the WRAP
1: The RWC flag remains cleared even if the setting condition is met.

Bit 6 – BEIEN Bus Error Interrupt Disable
0: The flag is set when an error is reported on the system bus by the bus MATRIX.
1: The flag remains cleared even if the setting condition is met.

Bit 5 – DMIEN Digest Mismatch Interrupt Disable
0: The RBE flag is set when the hash value just calculated from the processed region dffers from expected hash value.
1: The RBE flag remains cleared even if the setting condition is met.

Bit 4 – RHIEN Region Hash Completed Interrupt Disable
0: The RHC flag is set when the field NEXT = 0 in a descriptor of the main or second list.
1: The RHC flag remains cleared even if the setting condition is met.

Bit 2 – EOM End of Monitoring
0: The current descriptor does not terminate the monitoring.
1: The current descriptor terminates the Main List. WRAP bit value has no effect.

Bit 1 – WRAP Wrap Command
0: The next region descriptor address loaded is the current region identifier descriptor address incremented by 0x10.
1: The next region descriptor address loaded is DSCR.

Bit 0 – CDWBN Compare Digest or Write Back Digest
0: The digest is written to the Hash area.
1: The digest value is compared to the digest stored in the Hash area.

25.6.3.1.3 Region Control Structure Member

Name: RCTRL
Offset: 0x08
Reset: 0x00000000
Property: R/W

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	TRSIZE[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	TRSIZE[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 15:0 – TRSIZE[15:0] Transfer Size for the Current Chunk of Data

25.6.3.1.4 Region Next Address Structure Member

Name: RNEXT
Offset: 0x0C
Reset: 0x00000000
Property: Read/Write

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

25.6.4 Using ICM as an SHA Engine

The ICM can be configured to only calculate a SHA1, SHA256 digest value.

25.6.4.1 Settings for Simple SHA Calculation

The start address of the system memory containing the data to hash must be configured in the transfer descriptor of the DMA embedded in the ICM.

The transfer descriptor is a system memory area integer multiple of 4 x 32-bit word and the start address of the descriptor must be configured in DSCR (the start address must be aligned on 64-bytes; six LSB must be cleared). If the data to hash is already padded according to SHA standards, only a single descriptor is required, and the EOM bit of RCFGn must be written to '1'. If the data to hash does not contain a padding area, it is possible to define the padding area in another system memory location, the ICM can be configured to automatically jump from a memory area to another one by writing the descriptor register RNEXT with a value that differs from 0. Writing the RNEXT register with the start address of the padding area forces the ICM to concatenate both areas, thus providing the SHA result from the start address of the hash area configured in HASH.

Whether the system memory is configured as a single or multiple data block area, the bits CDWBN and WRAP must be cleared in the region descriptor structure member RCFGn. The bits WBDIS, EOMDIS, SLBDIS must be cleared in CFG.

Write the bits RHIEN and ECIEN in the Region Configuration Structure Member (RCFGn) to '0':

- The flag RHC[i], 'i' being the region index, is set (if RHIEN is '0') when the hash result is available at address defined in HASH.
- The flag REC[i], 'i' being the region index, is set (if ECIEN is '0') when the hash result is available at the address defined in HASH.

An interrupt is generated if the bit RHC[i] is written to '1' in the IER (if RHC[i] is set in RCTRL of region i) or if the bit REC[i] is written to '1' in the IER (if REC[i] is set in RCTRL of region i).

25.6.4.2 Processing Period

The SHA engine processing period can be configured by writing to the Region Configuration Structure Member register (RCFGn).

The short processing period allows to allocate bandwidth to the SHA module whereas the long processing period allocates more bandwidth on the system bus to other applications.

In SHA mode, the shortest processing period is 85 clock cycles + 2 clock cycles for start command synchronization. The longest period is 209 clock cycles + 2 clock cycles.

In SHA256 mode, the shortest processing period is 72 clock cycles + 2 clock cycles for start command synchronization. The longest period is 194 clock cycles + 2 clock cycles.

25.6.5 ICM Automatic Monitoring Mode

The ASCD bit of the CFG register is used to activate the ICM Automatic Mode. When CFG.ASCD is set, the ICM performs the following actions:

- The ICM controller passes through the Main List once with CDWBN bit in RCFGn at '0' (in other words, Write Back activated) and EOM bit in the RCFGn context register at '0'.
- When RCFGn.WRAP=1, the ICM controller enters active monitoring, with CDWBN bit in context register now set, and EOM bit in context register cleared. Writing to the CDWBN and EOM bits in RCFGn has no effect.

25.6.6 ICM Configuration Parameters

Transfer Type		Main List	RCFG			RNEXT	Comments
			CDWBN	WRAP	EOM	NEXT	
Single Region	Contiguous list of blocks Digest written to memory Monitoring disabled	1 item	0	0	1	0	The Main List contains only one descriptor. The Secondary List is empty for that descriptor. The digest is computed and saved to memory.
	Non-contiguous list of blocks Digest written to memory Monitoring disabled	1 item	0	0	1	Secondary List address of the current region identifier	The Main List contains only one descriptor. The Secondary List describes the layout of the non-contiguous region.
	Contiguous list of blocks Digest comparison enabled Monitoring enabled	1 item	1	1	0	0	When the hash computation is terminated, the digest is compared with the one saved in memory.

.....continued

Transfer Type		Main List	RCFG			RNEXT	Comments
			CDWBN	WRAP	EOM	NEXT	
Multiple Regions	Contiguous list of blocks Digest written to memory Monitoring disabled	More than one item	0	0	1 for the last, 0 otherwise	0	ICM passes through the list once.
	Contiguous list of blocks Digest comparison is enabled Monitoring is enabled	More than one item	1	1 for the last, 0 otherwise	0	0	ICM performs active monitoring of the regions. If a mismatch occurs, an interrupt is raised.
	Non-contiguous list of blocks Digest is written to memory Monitoring is disabled	More than one item	0	0	1	Secondary List address	ICM performs hashing and saves digests to the Hash area.
	Non-contiguous list of blocks Digest comparison is enabled Monitoring is enabled	More than one item	1	1	0	Secondary List address	ICM performs data gathering on a per region basis.

25.6.7 Security Features

When an undefined register access occurs, the URAD bit in the Interrupt Status Register (ISR) is set if unmasked. Its source is then reported in the Undefined Access Status Register (UASR). Only the first undefined register access is available through the UASR.URAT field.

Several kinds of unspecified register accesses can occur:

- Unspecified structure member set to one detected when the descriptor is loaded
- Configuration register (CFG) modified during active monitoring
- Descriptor register (DSCR) modified during active monitoring
- Hash register (HASH) modified during active monitoring
- Write-only register read access

The URAD bit and the URAT field can only be reset by writing a '1' to the CTRL.SWRST bit.

25.7 Register Summary - ICM

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CFG	7:0	BBC[3:0]					SLBDIS	EOMDIS	WBDIS
		15:8	UALGO[2:0]			UIHASH			DUALBUFF	ASCD
		23:16								
		31:24								
0x04	CTRL	7:0	REHASH[3:0]					SWRST	DISABLE	ENABLE
		15:8	RMEN[3:0]				RMDIS[3:0]			
		23:16								
		31:24								
0x08	SR	7:0								ENABLE
		15:8	RMDIS[3:0]				RAWRMDIS[3:0]			
		23:16								
		31:24								
0x0C ... 0x0F	Reserved									
0x10	IER	7:0	RDM[3:0]				RHC[3:0]			
		15:8	RWC[3:0]				RBE[3:0]			
		23:16	RSU[3:0]				REC[3:0]			
		31:24								URAD
0x14	IDR	7:0	RDM[3:0]				RHC[3:0]			
		15:8	RWC[3:0]				RBE[3:0]			
		23:16	RSU[3:0]				REC[3:0]			
		31:24								URAD
0x18	IMR	7:0	RDM[3:0]				RHC[3:0]			
		15:8	RWC[3:0]				RBE[3:0]			
		23:16	RSU[3:0]				REC[3:0]			
		31:24								URAD
0x1C	ISR	7:0	RDM[3:0]				RHC[3:0]			
		15:8	RWC[3:0]				RBE[3:0]			
		23:16	RSU[3:0]				REC[3:0]			
		31:24								URAD
0x20	UASR	7:0						URAT[2:0]		
		15:8								
		23:16								
		31:24								
0x24 ... 0x2F	Reserved									
0x30	DSCR	7:0	DASA[1:0]							
		15:8	DASA[9:2]							
		23:16	DASA[17:10]							
		31:24	DASA[25:18]							
0x34	HASH	7:0								
		15:8								
		23:16								
		31:24								
0x38	UIHVALx0	7:0	VAL[7:0]							
		15:8	VAL[15:8]							
		23:16	VAL[23:16]							
		31:24	VAL[31:24]							
0x3C	UIHVALx1	7:0	VAL[7:0]							
		15:8	VAL[15:8]							
		23:16	VAL[23:16]							
		31:24	VAL[31:24]							
0x40	UIHVALx2	7:0	VAL[7:0]							
		15:8	VAL[15:8]							
		23:16	VAL[23:16]							
		31:24	VAL[31:24]							

.....continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x44	UIHVALx3	7:0	VAL[7:0]							
		15:8	VAL[15:8]							
		23:16	VAL[23:16]							
		31:24	VAL[31:24]							
0x48	UIHVALx4	7:0	VAL[7:0]							
		15:8	VAL[15:8]							
		23:16	VAL[23:16]							
		31:24	VAL[31:24]							
0x4C	UIHVALx5	7:0	VAL[7:0]							
		15:8	VAL[15:8]							
		23:16	VAL[23:16]							
		31:24	VAL[31:24]							
0x50	UIHVALx6	7:0	VAL[7:0]							
		15:8	VAL[15:8]							
		23:16	VAL[23:16]							
		31:24	VAL[31:24]							
0x54	UIHVALx7	7:0	VAL[7:0]							
		15:8	VAL[15:8]							
		23:16	VAL[23:16]							
		31:24	VAL[31:24]							

25.8 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers are optionally write protected by the Peripheral Access Controller (PAC). Optional PAC write protection is denoted by the “PAC Write-Protection” property in each individual register description. For details, refer to 22.5.7. Register Access Protection.

Some registers are enable protected, meaning they can only be written when the peripheral is disabled. Enable protection is denoted by the “Enable-Protected” property in each individual register description.

25.8.1 Configuration Register

Name: CFG
Offset: 0x00
Reset: 0x0
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	UALGO[2:0]			UIHASH			DUALBUFF	ASCD
Access	-	-	-	-			-	R/W
Reset	0	0	0	0			0	0

Bit	7	6	5	4	3	2	1	0
	BBC[3:0]					SLBDIS	EOMDIS	WBDIS
Access	R/W	R/W	R/W	R/W		R/W	R/W	R/W
Reset	0	0	0	0		0	0	0

Bits 15:13 – UALGO[2:0] User SHA Algorithm

Value	Name	Description
0	SHA1	SHA1 algorithm processed
1	SHA256	SHA256 algorithm processed
Other	-	Reserved

Bit 12 – UIHASH User Initial Hash Value

Value	Description
0	The secure hash standard provides the initial hash value.
1	The initial hash value is programmable. Field UALGO provides the SHA algorithm. The ALGO field of the RCFGn structure member has no effect.

Bit 9 – DUALBUFF Dual Input Buffer

Value	Description
0	Dual Input buffer mode is disabled.
1	Dual Input buffer mode is enabled (Better performances, higher bandwidth required on system bus).

Bit 8 – ASCD Automatic Switch To Compare Digest

Value	Description
0	Automatic mode is disabled.
1	When this mode is enabled, the ICM controller automatically switches to active monitoring after the first Main List pass. Both CDWBN and WBDIS bits have no effect. A '1' must be written to the End of Monitoring bit in the Region Configuration register (RCFG.EOM) to terminate the monitoring.

Bits 7:4 – BBC[3:0] Bus Burden Control

This field is used to control the burden of the ICM system bus. The number of system clock cycles between the end of the current processing and the next block transfer is set to 2^{BBC}. Up to 32768 cycles can be inserted.

Bit 2 – SLBDIS Secondary List Branching Disable

Value	Description
0	Branching to the Secondary List is permitted.
1	Branching to the Secondary List is forbidden. The NEXT field of the RNEXT structure member has no effect and is always considered as zero.

Bit 1 – EOMDIS End of Monitoring Disable

Value	Description
0	End of Monitoring is permitted.
1	End of Monitoring is forbidden. The EOM bit of the RCFG structure member has no effect.

Bit 0 – WBDIS Write Back Disable

When the Automatic Switch to Compare Digest bit of this register (CFG.ASCD) is written to '1', this bit value has no effect.

Value	Description
0	Write Back Operations are permitted.
1	Write Back Operations are forbidden: Context register CDWBN bit is internally set to '1' and cannot be modified by a linked list element. The CDWBN bit of the RCFG structure member has no effect.

25.8.2 Control Register

Name: CTRL
Offset: 0x04
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	RMEN[3:0]				RMDIS[3:0]			
Access	W	W	W	W	W	W	W	W
Reset								

Bit	7	6	5	4	3	2	1	0
	REHASH[3:0]					SWRST	DISABLE	ENABLE
Access	W	W	W	W		W	W	W
Reset							0	0

Bits 15:12 – RMEN[3:0] Region Monitoring Enable

Value	Description
0	No effect.
1	When bit RMEN[i] is written to '1', the monitoring of region with identifier i is activated.

Bits 11:8 – RMDIS[3:0] Region Monitoring Disable

Value	Description
0	No effect.
1	When REHASH[i] is written to '1', Region i digest is re-computed. This bit is only available when region monitoring is disabled.

Bits 7:4 – REHASH[3:0] Recompute Internal Hash

Value	Description
0	No effect.
1	When REHASH[i] is written to '1', Region i digest is re-computed. This bit is only available when region monitoring is disabled.

Bit 2 – SWRST Software Reset

Value	Description
0	No effect.
1	Resets the ICM controller.

Bit 1 – DISABLE ECM Disable

Value	Description
0	No effect.
1	The ICM controller is disabled. If a region is activated, the region is terminated.

Bit 0 – ENABLE ICM Enable

Value	Description
0	No effect.
1	The ICM controller is activated.

25.8.3 Status Register

Name: SR
Offset: 0x08
Property: Read-Only

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	RMDIS[3:0]				RAWRMDIS[3:0]			
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
								ENABLE
Access								R
Reset								0

Bits 15:12 – RMDIS[3:0] Region Monitoring Disabled Status

Value	Description
0	Region i is being monitored (occurs after integrity check value has been calculated and written to Hash area).
1	Region i is not being monitored.

Bits 11:8 – RAWRMDIS[3:0] Region Monitoring Disabled Raw Status

Value	Description
0	Region i monitoring has been activated by writing a 1 in RMEN[i] of CTRL
1	Region i monitoring has been deactivated by writing a 1 in RMDIS[i] of CTRL

Bit 0 – ENABLE ICM Controller Enable Register

Value	Description
0	ICM controller is disabled.
1	ICM controller is activated.

25.8.4 Interrupt Enable Register

Name: IER
Offset: 0x10
Reset: 0x00000000
Property: Write-Only

Bit	31	30	29	28	27	26	25	24
								URAD
Access								W
Reset								0

Bit	23	22	21	20	19	18	17	16
	RSU[3:0]				REC[3:0]			
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	RWC[3:0]				RBE[3:0]			
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	RDM[3:0]				RHC[3:0]			
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit 24 – URAD Undefined Register Access Detection Interrupt Enable
0: No effect
1: The Undefined Register Access interrupt is enabled.

Bits 23:20 – RSU[3:0] Region Status Updated Interrupt Enable
0: No effect
1: When RSU[i] is written to ‘1’, the region i Status Updated interrupt is enabled.

Bits 19:16 – REC[3:0] Region End bit Condition Detected Interrupt Enable
0: No effect
1: When REC[i] is written to ‘1’, the region i End bit Condition interrupt is enabled.

Bits 15:12 – RWC[3:0] Region Wrap Condition detected Interrupt Enable
0: No effect
1: When RWC[i] is written to ‘1’, the Region i Wrap Condition interrupt is enabled.

Bits 11:8 – RBE[3:0] Region Bus Error Interrupt Enable

Value	Description
0	No effect.
1	When RBE[i] is written to '1', the Region i Bus Error interrupt is enabled.

Bits 7:4 – RDM[3:0] Region Digest Mismatch Interrupt Enable

Value	Description
0	No effect.
1	When RDM[i] is written to '1', the Region i Digest Mismatch interrupt is enabled.

Bits 3:0 – RHC[3:0] Region Hash Completed Interrupt Enable

Value	Description
0	No effect.
1	When RHC[i] is written to '1', the Region i Hash Completed interrupt is enabled.

25.8.5 Interrupt Disable Register

Name: IDR
Offset: 0x14
Property: Write-Only

Bit	31	30	29	28	27	26	25	24
								URAD
Access								W
Reset								

Bit	23	22	21	20	19	18	17	16
	RSU[3:0]				REC[3:0]			
Access	W	W	W	W	W	W	W	W
Reset								

Bit	15	14	13	12	11	10	9	8
	RWC[3:0]				RBE[3:0]			
Access	W	W	W	W	W	W	W	W
Reset								

Bit	7	6	5	4	3	2	1	0
	RDM[3:0]				RHC[3:0]			
Access	W	W	W	W	W	W	W	W
Reset								

Bit 24 – URAD Undefined Register Access Detection Interrupt Disable

Value	Description
0	No effect.
1	Undefined Register Access Detection interrupt is disabled.

Bits 23:20 – RSU[3:0] Region Status Updated Interrupt Disable

Value	Description
0	No effect.
1	When RSU[i] is written to '1', the region i Status Updated interrupt is disabled.

Bits 19:16 – REC[3:0] Region End bit Condition detected Interrupt Disable

Value	Description
0	No effect.
1	When REC[i] is written to '1', the region i End bit Condition interrupt is disabled.

Bits 15:12 – RWC[3:0] Region Wrap Condition Detected Interrupt Disable

Value	Description
0	No effect.
1	When RWC[i] is written to '1', the Region i Wrap Condition interrupt is disabled.

Bits 11:8 – RBE[3:0] Region Bus Error Interrupt Disable

Value	Description
0	No effect.
1	When RBE[i] is written to '1', the Region i Bus Error interrupt is disabled.

Bits 7:4 – RDM[3:0] Region Digest Mismatch Interrupt Disable

Value	Description
0	No effect.

Value	Description
1	When RDM[i] is written to '1', the Region i Digest Mismatch interrupt is disabled.

Bits 3:0 – RHC[3:0] Region Hash Completed Interrupt Disable

Value	Description
0	No effect.
1	When RHC[i] is written to '1', the Region i Hash Completed interrupt is disabled.

25.8.6 Interrupt Mask Register

Name: IMR
Offset: 0x18
Reset: 0x00000000
Property: Read-Only

Bit	31	30	29	28	27	26	25	24
								URAD
Access								R
Reset								0

Bit	23	22	21	20	19	18	17	16
	RSU[3:0]				REC[3:0]			
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	RWC[3:0]				RBE[3:0]			
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	RDM[3:0]				RHC[3:0]			
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit 24 – URAD Undefined Register Access Detection Interrupt Mask

Value	Description
0	The interrupt is disabled.
1	The interrupt is enabled.

Bits 23:20 – RSU[3:0] Region Status Updated Interrupt Mask

Value	Description
0	When RSU[i] is reading '0', the interrupt is disabled for region i.
1	When RSU[i] is reading '1', the interrupt is enabled for region i.

Bits 19:16 – REC[3:0] Region End bit Condition Detected Interrupt Mask

Value	Description
0	When REC[i] is reading '0', the interrupt is disabled for region i.
1	When REC[i] is reading '1', the interrupt is enabled for region i.

Bits 15:12 – RWC[3:0] Region Wrap Condition Detected Interrupt Mask

Value	Description
0	When RWC[i] is reading '0', the interrupt is disabled for region i.
1	When RWC[i] is reading '1', the interrupt is enabled for region i.

Bits 11:8 – RBE[3:0] Region Bus Error Interrupt Mask

Value	Description
0	When RBE[i] is reading '0', the interrupt is disabled for region i.
1	When RBE[i] is reading '1', the interrupt is enabled for region i.

Bits 7:4 – RDM[3:0] Region Digest Mismatch Interrupt Mask

Value	Description
0	When RDM[i] is reading '0', the interrupt is disabled for region i.
1	When RDM[i] is reading '1', the interrupt is enabled for region i.

Bits 3:0 – RHC[3:0] Region Hash Completed Interrupt Mask

Value	Description
0	When RHC[i] is reading '0', the interrupt is disabled for region i.
1	When RHC[i] is reading '1', the interrupt is enabled for region i.

25.8.7 Interrupt Status Register

Name: ISR
Offset: 0x1C
Reset: 0x0
Property: Read-Only

Bit	31	30	29	28	27	26	25	24
								URAD
Access								R
Reset								0

Bit	23	22	21	20	19	18	17	16
	RSU[3:0]				REC[3:0]			
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	RWC[3:0]				RBE[3:0]			
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	RDM[3:0]				RHC[3:0]			
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit 24 – URAD Undefined Register Access Detection Status
The URAD bit is only reset by the SWRST bit in the CTRL register.
The Undefined Register Access Trace bit field in the Undefined Access Status Register (UASR.URAT) indicates the unspecified access type.

Value	Description
0	No undefined register access has been detected since the last SWRST.
1	At least one undefined register access has been detected since the last SWRST.

Bits 23:20 – RSU[3:0] Region Status Updated Detected
RSU[i] is set when a region status updated condition is detected.

Bits 19:16 – REC[3:0] Region End bit Condition Detected
REC[i] is set when an end bit condition is detected.

Bits 15:12 – RWC[3:0] Region Wrap Condition Detected
RWC[i] is set when a wrap condition is detected.

Bits 11:8 – RBE[3:0] Region Bus Error
RBE[i] is set when a bus error is detected while hashing memory region i.

Bits 7:4 – RDM[3:0] Region Digest Mismatch
RDM[i] is set when there is a digest comparison mismatch between the hash value of region i and the reference value located in the Hash Area.

Bits 3:0 – RHC[3:0] Region Hash Completed
RHC[i] is set when the ICM has completed the region with identifier i.

25.8.8 Undefined Access Status Register

Name: UASR
Offset: 0x20
Reset: 0x0
Property: Read-Only

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
						URAT[2:0]		
Access						R	R	R
Reset						0	0	0

Bits 2:0 – URAT[2:0] Undefined Register Access Trace
Only the first Undefined Register Access Trace is available through the URAT field.
The URAT field is only reset by the Software Reset bit in the Control register (CTRL.SWRST).

Value	Name	Description
0	UNSPEC_STRUCT_MEMBER	Unspecified structure member set to '1' detected when the descriptor is loaded.
1	ICM_CFG_MODIFIED	CFG modified during active monitoring.
2	ICM_DSCR_MODIFIED	DSCR modified during active monitoring.
3	ICM_HASH_MODIFIED	HASH modified during active monitoring
4	READ_ACCESS	Write-only register read access Only the first Undefined Register Access Trace is available through the URAT field. The URAT field is only reset by the SWRST bit in the CTRL register.

25.8.9 Descriptor Area Start Address Register

Name: DSCR
Offset: 0x30
Reset: 0x0
Property: -

Bit	31	30	29	28	27	26	25	24
	DASA[25:18]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	DASA[17:10]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	DASA[9:2]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DASA[1:0]							
Access	R/W	R/W						
Reset	0	0						

Bits 31:6 – DASA[25:0] Descriptor Area Start Address
The start address is a multiple of the total size of the data structure (64 bytes).

25.8.10 Hash Area Start Address Register

Name: HASH
Offset: 0x34
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

25.8.11 User Initial Hash Value Register

Name: UIHVALx
Offset: 0x38 + x*0x04 [x=0..7]
Reset: 0
Property: -

Bit	31	30	29	28	27	26	25	24
	VAL[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	VAL[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	VAL[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	VAL[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – VAL[31:0] Initial Hash Value

When UIHASH bit of CFG register is set, the Initial Hash Value is user-programmable.
To meet the desired standard, use the following example values.
For UIHVAL0 field:

Example	Comment
0x67452301	SHA1 algorithm
0x6A09E667	SHA256 algorithm

For UIHVAL1 field:

Example	Comment
0xEFCDAB89	SHA1 algorithm
0xBB67AE85	SHA256 algorithm

For UIHVAL2 field:

Example	Comment
0x98BADCFE	SHA1 algorithm
0x3C6EF372	SHA256 algorithm

For UIHVAL3 field:

Example	Comment
0x10325476	SHA1 algorithm
0xA54FF53A	SHA256 algorithm

For UIHVAL4 field:

Example	Comment
0xC3D2E1F0	SHA1 algorithm
0x510E527F	SHA256 algorithm

For UIHVAL5 field:

Example	Comment
0x9B05688C	SHA256 algorithm

For UIHVAL6 field:

Example	Comment
0x1F83D9AB	SHA256 algorithm

For UIHVAL7 field:

Example	Comment
0x5BE0CD19	SHA256 algorithm

Example of Initial Value for SHA-1 Algorithm

Register Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000 UIHVAL0	01	23	45	67
0x004 UIHVAL1	89	ab	cd	ef
0x008 UIHVAL2	fe	dc	ba	98
0x00C UIHVAL3	76	54	32	10
0x010 UIHVAL4	f0	e1	d2	c3

26. Peripheral Access Controller (PAC)

26.1 Overview

The Peripheral Access Controller provides an interface for the locking and unlocking of peripheral registers within the device. It reports all violations that could happen when accessing a peripheral: write protected access, illegal access, enable protected access, access when clock synchronization or software reset is on-going. These errors are reported in a unique interrupt flag for a peripheral. The PAC module also reports errors occurring at the client bus level, when an access to a non-existing address is detected.

Notes:

1. The modules attached to the PB-PIC bridge and wireless subsystem as well as RTCC, DSCON, PUKCC and ICM are excluded from the PAC. The protection mechanism described in the System Configuration Registers (CFG) protects critical system registers (see *System Configuration Registers (CFG)* from Related Links).
2. Traditional Peripheral Access Controller (PAC) documentation uses the terminology “Master” and “Slave”. The equivalent Microchip terminology used in this document is “Host” and “Client”, respectively.

Related Links

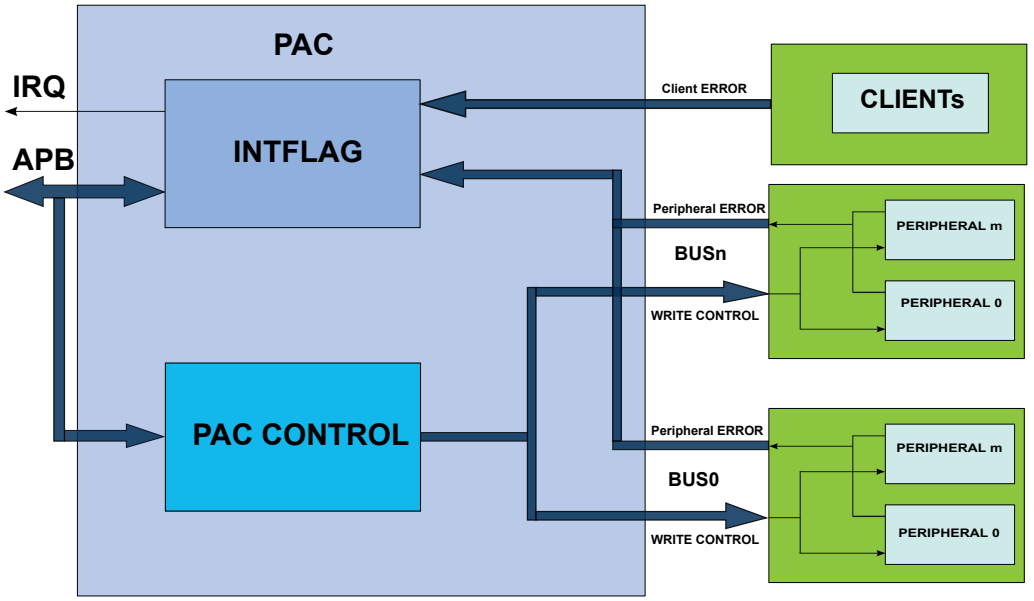
18. System Configuration and Register Locking (CFG)

26.2 Features

- Manages write protection access and reports access errors for the peripheral modules or bridges.

26.3 Block Diagram

Figure 26-1. PAC Block Diagram



26.4 Product Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

26.4.1 IO Lines

Not applicable.

26.4.2 Power Management

The PAC can continue to operate in any Sleep mode where the selected source clock is running. The PAC interrupts can be used to wake up the device from Sleep modes. The events can trigger other operations in the system without exiting sleep modes.

26.4.3 DMA

Not applicable.

26.4.4 Interrupts

The interrupt request line is connected to the Interrupt Controller (NVIC). Using the PAC interrupt requires the Interrupt Controller to be configured first.

Table 26-1. Interrupt Lines

Instances	NVIC Line
PAC	PACERR

26.4.5 Events

The events are connected to the Event System, which may need configuration. See *Event System (EVSYS)* from Related Links.

Related Links

28. Event System (EVSYS)

26.4.6 Debug Operation

When the CPU is halted in Debug mode, write protection of all peripherals is disabled and the PAC continues normal operation.

26.4.7 Register Access Protection

All registers with write access can be write-protected optionally by the Peripheral Access Controller (PAC), except for the following PAC registers:

- Write Control (WRCTRL) register
- AHB Subordinate Bus Interrupt Flag Status and Clear (INTFLAGAHB) register
- Peripheral Interrupt Flag Status and Clear n (INTFLAG A/B/C...) registers

Optional write protection by the Peripheral Access Controller (PAC) is denoted by the “PAC Write Protection” property in each individual register description.

Note: PAC write protection does not apply to accesses through an external debugger.

26.5 Functional Description

26.5.1 Principle of Operation

The Peripheral Access Control module allows the user to set a write protection on peripheral modules and generate an interrupt in case of a peripheral access violation. The peripheral’s protection can be set, cleared or locked at the user discretion. A set of Interrupt Flag and Status registers informs the user on the status of the violation in the peripherals. In addition, client bus errors can be also reported in the cases where reserved area is accessed by the application.

26.5.2 Basic Operation

26.5.2.1 Initialization, Enabling and Resetting

The PAC is always enabled after reset.

Only a hardware reset will reset the PAC module.

26.5.2.2 Operations

The PAC module allows the user to set, clear or lock the write protection status of all peripherals on all Peripheral Bridges, except the peripherals on PB-PIC bus.

If a peripheral register violation occurs, the Peripheral Interrupt Flag n registers (INTFLAGn) are updated to inform the user on the status of the violation in the peripherals connected to the Peripheral Bridge n (n = A,B,C ...). The corresponding Peripheral Write Control Status n register (STATUSn) gives the state of the write protection for all peripherals connected to the corresponding Peripheral Bridge n. See *Peripheral Access Errors* from Related Links.

The PAC module also report the errors occurring at client bus level when an access to reserved area is detected. AHB Subordinate Bus Interrupt Flag register (INTFLAGAHB) informs the user on the status of the violation in the corresponding client. See *AHB Subordinate Bus Errors* from Related Links.

Related Links

26.5.2.3. Peripheral Access Errors
26.5.2.6. AHB Subordinate Bus Errors

26.5.2.3 Peripheral Access Errors

The following events will generate a Peripheral Access Error:

- Protected write: To avoid unexpected writes to a peripheral's registers, each peripheral can be write protected. Only the registers denoted as “PAC Write-Protection” in the module’s datasheet can be protected. If a peripheral is not write protected, write data accesses are performed normally. If a peripheral is write protected and if a write access is attempted, data will not be written and peripheral returns an access error. The corresponding interrupt flag bit in the INTFLAGn register will be set.
- Illegal access: Access to an unimplemented register within the module.
- Synchronized write error: For write-synchronized registers an error will be reported if the register is written while a synchronization is ongoing.

When any of the INTFLAGn registers bit are set, an interrupt will be requested if the PAC interrupt enable bit is set.

26.5.2.4 Write Access Protection Management

Peripheral access control can be enabled or disabled by writing to the WRCTRL register.

The data written to the WRCTRL register is composed of two fields; WRCTRL.PERID and WRCTRL.KEY. The WRCTRL.PERID is an unique identifier corresponding to a peripheral. The WRCTRL.KEY is a key value that defines the operation to be done on the control access bit. These operations can be “clear protection”, “set protection” and “set and lock protection bit”.

The “clear protection” operation will remove the write access protection for the peripheral selected by WRCTRL.PERID. Write accesses are allowed for the registers in this peripheral.

The “set protection” operation will set the write access protection for the peripheral selected by WRCTRL.PERID. Write accesses are not allowed for the registers with write protection property in this peripheral.

The “set and lock protection” operation will set the write access protection for the peripheral selected by WRCTRL.PERID and locks the access rights of the selected peripheral registers. The write access protection will only be cleared by a hardware reset.

The peripheral access control status can be read from the corresponding STATUSn register.

26.5.2.5 Write Access Protection Management Errors

Only word-wise writes to the WRCTRL register will effectively change the access protection. Other type of accesses will have no effect and will cause a PAC write access error. This error is reported in the INTFLAGA.PAC bit.

PAC also offers an additional safety feature for correct program execution with an interrupt generated on double write clear protection or double write set protection. If a peripheral is write protected and a subsequent set protection operation is detected then the PAC returns an error, and similarly for a double clear protection operation.

In addition, an error is generated when writing a “set and lock” protection to a write-protected peripheral or when a write access is done to a locked set protection. This can be used to ensure that the application follows the intended program flow by always following a write protect with an unprotect and conversely. However in applications where a write protected peripheral is used in several contexts, for example, interrupt, care must be taken so that either the interrupt can not happen while the main application or other interrupt levels manipulates the write protection status or when the interrupt handler needs to unprotect the peripheral based on the current protection status by reading the STATUS register.

The errors generated while accessing the PAC module registers (for example, key error, double protect error and so on) will set the INTFLAGA.PAC flag.

26.5.2.6 AHB Subordinate Bus Errors

The PAC module reports errors occurring at the AHB Subordinate bus level. These errors are generated when an access is performed at an address where no subordinate (bridge or peripheral) is mapped. These errors are reported in the corresponding bits of the INTFLAGAHB register.

26.5.2.7 Generating Events

The PAC module can also generate an event when any of the Interrupt Flag registers bit are set. To enable the PAC event generation, the control bit EVCTRL.ERREO must be set a '1'.

26.5.3 DMA Operation

Not applicable.

26.5.4 Interrupts

The PAC has the following interrupt source:

- Error (ERR): Indicates that a peripheral access violation occurred in one of the peripherals controlled by the PAC module, or a bridge error occurred in one of the bridges reported by the PAC
  - This interrupt is a synchronous wake-up source

Each interrupt source has an interrupt flag associated with it. The interrupt flag in the Interrupt Flag Status and Clear (INTFLAGAHB and INTFLAGn) registers is set when the interrupt condition occurs. Each interrupt can be individually enabled by writing a '1' to the corresponding bit in the Interrupt Enable Set (INTENSET) register, and disabled by writing a '1' to the corresponding bit in the Interrupt Enable Clear (INTENCLR) register.

An interrupt request is generated when the interrupt flag is set and the corresponding interrupt is enabled. The interrupt request remains active until the interrupt flag is cleared, the interrupt is disabled, or the PAC is reset. All interrupt requests from the peripheral are ORed together on system level to generate one combined interrupt request to the NVIC. The user must read the INTFLAGAHB and INTFLAGn registers to determine which interrupt condition is present.

Note that interrupts must be globally enabled for interrupt requests to be generated.

26.5.5 Events

The PAC can generate the following output event:

- Error (ERR): Generated when one of the interrupt flag registers bits is set

Writing a '1' to an Event Output bit in the Event Control Register (EVCTRL.ERREO) enables the corresponding output event. Writing a '0' to this bit disables the corresponding output event.

26.5.6 Sleep Mode Operation

In Sleep mode, the PAC is kept enabled if an available bus host (CPU, DMA) is running. The PAC will continue to catch access errors from the module and generate interrupts or events.

26.5.7 Synchronization

Not applicable.

26.6 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	WRCTRL	7:0	PERID[7:0]							
		15:8	PERID[15:8]							
		23:16	KEY[7:0]							
		31:24								
0x04	EVCTRL	7:0								ERREO
0x05 ... 0x07	Reserved									
0x08	INTENCLR	7:0								ERR
0x09	INTENSET	7:0								ERR
0x0A ... 0x0F	Reserved									
0x10	INTFLAGAHB	7:0	PBBB	PBAB	PFLASH	CFLASH	SRAM3	SRAM2	SRAM1	SRAM0
		15:8						QSPI	PBPICB	PBCB
		23:16								
		31:24								
0x14	INTFLAGA	7:0	TC2	TC1	TC0	SERCOM1	SERCOM0	EIC	FREQM	PAC
		15:8					TCC2	TCC1	TCC0	TC3
		23:16								
		31:24								
0x18	INTFLAGB	7:0				RAMECC	EVSYS	DMAC		DSU
		15:8								
		23:16								
		31:24								
0x1C	INTFLAGC	7:0	AC	CCL						
		15:8								
		23:16								
		31:24								
0x20 ... 0x33	Reserved									
0x34	STATUSA	7:0	TC2	TC1	TC0	SERCOM1	SERCOM0	EIC	FREQM	PAC
		15:8						TCC1		TC3
		23:16								
		31:24								
0x38	STATUSB	7:0				RAMECC	EVSYS	DMAC		
		15:8								
		23:16								
		31:24								
0x3C	STATUSC	7:0	AC	CCL		SERCOM3	SERCOM2			
		15:8								
		23:16								
		31:24								

26.7 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers are optionally write protected by the Peripheral Access Controller (PAC). Optional PAC write protection is denoted by the “PAC Write-Protection” property in each individual register description. For details, refer to the related links.

26.7.1 Write Control

Name: WRCTRL
Offset: 0x00
Reset: 0x00000000
Property: –

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	KEY[7:0]							
Access	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	PERID[15:8]							
Access	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	PERID[7:0]							
Access	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0

Bits 23:16 – KEY[7:0] Peripheral Access Control Key
These bits define the peripheral access control key:

Value	Name	Description
0x0	OFF	No action
0x1	CLEAR	Clear the peripheral write control
0x2	SET	Set the peripheral write control
0x3	LOCK	Set and lock the peripheral write control until the next hardware reset

Bits 15:0 – PERID[15:0] Peripheral Identifier
The PERID represents the peripheral whose control is changed using the WRCTRL.KEY. The Peripheral Identifier is calculated following formula:
$PERID = 32^{*}\,\mathrm{BridgeNumber} + \mathrm{N}$

Where BridgeNumber represents the Peripheral Bridge Number (0 for Peripheral Bridge A, 1 for Peripheral Bridge B, etc). N represents the peripheral index from the respective Bridge Number:

Table 26-2. PERID Values

Periph. Bridge Name	BridgeNumber	PERID Values
A	0	0+N
B	1	32+N
C	2	64+N
D	3	96+N

Note: GMAC, ICM, SDHC, CAN and PCC peripherals do not support that feature.

26.7.2 Event Control

Name: EVCTRL
Offset: 0x04
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
								ERREO
Access								RW
Reset								0

Bit 0 – ERREO Peripheral Access Error Event Output
This bit indicates if the Peripheral Access Error Event Output is enabled or disabled. When enabled, an event will be generated when one of the interrupt flag registers bits (INTFLAGAHB, INTFLAGn) is set:

Value	Description
0	Peripheral Access Error Event Output is disabled.
1	Peripheral Access Error Event Output is enabled.

26.7.3 Interrupt Enable Clear

Name: INTENCLR
Offset: 0x08
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to disable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Set register (INTENSET).

Bit	7	6	5	4	3	2	1	0
								ERR
Access								RW
Reset								0

Bit 0 – ERR Peripheral Access Error Interrupt Disable
This bit indicates that the Peripheral Access Error Interrupt is enabled and an interrupt request will be generated when one of the interrupt flag registers bits (INTFLAGAHB, INTFLAGn) is set:
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the Peripheral Access Error interrupt Enable bit and disables the corresponding interrupt request.

Value	Description
0	Peripheral Access Error interrupt is disabled.
1	Peripheral Access Error interrupt is enabled.

26.7.4 Interrupt Enable Set

Name: INTENSET
Offset: 0x09
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to enable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Set register (INTENCLR).

Bit	7	6	5	4	3	2	1	0
								ERR
Access								RW
Reset								0

Bit 0 – ERR Peripheral Access Error Interrupt Enable
This bit indicates that the Peripheral Access Error Interrupt is enabled and an interrupt request will be generated when one of the interrupt flag registers bits (INTFLAGAHB, INTFLAGn) is set:
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will set the Peripheral Access Error interrupt Enable bit and enables the corresponding interrupt request.

Value	Description
0	Peripheral Access Error interrupt is disabled.
1	Peripheral Access Error interrupt is enabled.

26.7.5 Bridge Interrupt Flag Status

Name: INTFLAGAHB
Offset: 0x10
Reset: 0x00000000
Property: -

These flags are cleared by writing a '1' to the corresponding bit.

These flags are set when an access error is detected by the corresponding AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
						QSPI	PBPICB	PBCB
Access						RW	RW	RW
Reset						0	0	0

Bit	7	6	5	4	3	2	1	0
	PBBB	PBAB	PFLASH	CFLASH	SRAM3	SRAM2	SRAM1	SRAM0
Access	RW	RW	RW	RW	RW	U	U	RW
Reset	0	0	0	0	0	0	0	0

Bit 10 – QSPI Interrupt Flag for QSPI
This flag is set when an access error is detected by the QSPI AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.
Writing a '1' to this bit will clear the QSPI interrupt flag.

Bit 9 – PBPICB Interrupt Flag for PBPICB (PB-PIC-Bridge)
This flag is set when an access error is detected by the PBPICB AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.
Writing a '1' to this bit will clear the PBPICB interrupt flag.

Bit 8 – PBCB Interrupt Flag for PBCB (PB-Bridge-C)
This flag is set when an access error is detected by the PBCB AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.
Writing a '1' to this bit will clear the PBCB interrupt flag.

Bit 7 – PBBB Interrupt Flag for PBBB (PB-Bridge-B)
This flag is set when an access error is detected by the PBBB AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.

Writing a '1' to this bit will clear the PBBB interrupt flag.

Bit 6 – PBAB Interrupt Flag for HPB1 (PB-Bridge-A)
This flag is set when an access error is detected by the PBAB AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.
Writing a '1' to this bit will clear the PBAB interrupt flag.

Bit 5 – PFLASH Interrupt Flag for PFLASH (Peripheral Flash)
This flag is set when an access error is detected by the PFLASH AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.
Writing a '1' to this bit will clear the PFLASH interrupt flag.

Bit 4 – CFLASH Interrupt Flag for CFLASH (CPU Flash)
This flag is set when an access error is detected by the CFLASH AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.
Writing a '1' to this bit will clear the CFLASH interrupt flag.

Bit 3 – SRAM3 Interrupt Flag for SRAM3
This flag is set when an access error is detected by the SRAM3 AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.
Writing a '1' to this bit will clear the SRAM3 interrupt flag.

Bit 2 – SRAM2 Interrupt Flag for SRAM2
This flag is set when an access error is detected by the SRAM2 AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.
Writing a '1' to this bit will clear the SRAM2 interrupt flag.

Bit 1 – SRAM1 Interrupt Flag for SRAM1
This flag is set when an access error is detected by the SRAM1 AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.
Writing a '1' to this bit will clear the SRAM1 interrupt flag.

Bit 0 – SRAM0 Interrupt Flag for SRAM0
This flag is set when an access error is detected by the SRAM0 AHB Subordinate, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' has no effect.
Writing a '1' to this bit will clear the SRAM0 interrupt flag.

26.7.6 Peripheral Interrupt Flag Status – Bridge A

Name: INTFLAGA
Offset: 0x14
Reset: 0x00000000
Property: –

These flags are set when a Peripheral Access Error occurs while accessing the peripheral associated with the respective INTFLAGx bit, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.

Writing a '0' to these bits has no effect.

Writing a '1' to these bits will clear the corresponding INTFLAGx interrupt flag.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
					TCC2	TCC1	TCC0	TC3
Access					RW	RW	RW	RW
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
	TC2	TC1	TC0	SERCOM1	SERCOM0	EIC	FREQM	PAC
Access	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0

Bit 11 – TCC2 Interrupt Flag for TCC2
This bit is set when a Peripheral Access Error occurs while accessing the TCC2, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 10 – TCC1 Interrupt Flag for TCC1
This bit is set when a Peripheral Access Error occurs while accessing the TCC1, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 9 – TCC0 Interrupt Flag for TCC0
This bit is set when a Peripheral Access Error occurs while accessing the TCC0, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 8 – TC3 Interrupt Flag for TC3
This bit is set when a Peripheral Access Error occurs while accessing the TC3, and will generate an interrupt request if SET.ERR is '1'.

Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 7 – TC2 Interrupt Flag for TC2
This bit is set when a Peripheral Access Error occurs while accessing the TC2, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 6 – TC1 Interrupt Flag for TC1
This bit is set when a Peripheral Access Error occurs while accessing the TC1, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 5 – TC0 Interrupt Flag for TC0
This bit is set when a Peripheral Write Access Error occurs while accessing the TC0, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 4 – SERCOM1 Interrupt Flag for SERCOM1
This bit is set when a Peripheral Access Error occurs while accessing the SERCOM1, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 3 – SERCOM0 Interrupt Flag for SERCOM0
This bit is set when a Peripheral Access Error occurs while accessing the SERCOM0, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 2 – EIC Interrupt Flag for EIC
This bit is set when a Peripheral Access Error occurs while accessing the EIC, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 1 – FREQM Interrupt Flag for FREQM
This bit is set when a Peripheral Access Error occurs while accessing the FREQM, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 0 – PAC Interrupt Flag for PAC
This bit is set when a Peripheral Write Access Error occurs while accessing the PAC, and will generate an interrupt request if SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

26.7.7 Peripheral Interrupt Flag Status – Bridge B

Name: INTFLAGB
Offset: 0x18
Reset: 0x00000000
Property: –

These flags are set when a Peripheral Access Error occurs while accessing the peripheral associated with the respective INTFLAGx bit, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.

Writing a '0' to these bits has no effect.

Writing a '1' to these bits will clear the corresponding INTFLAGx interrupt flag.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
				RAMECC	EVSYS	DMAC		DSU
Access				RW	RW	RW		RW
Reset				0	0	0		0

Bit 4 – RAMECC Interrupt Flag for RAMECC
This flag is set when a Peripheral Access Error occurs while accessing the RAMECC, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the RAMECC interrupt flag.

Bit 3 – EVSYS Interrupt Flag for EVSYS
This flag is set when a Peripheral Access Error occurs while accessing the EVSYS, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the EVSYS interrupt flag.

Bit 2 – DMAC Interrupt Flag for DMAC
This flag is set when a Peripheral Access Error occurs while accessing the DMAC, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the DMAC interrupt flag.

Bit 0 – DSU Interrupt Flag for DSU
This flag is set when a Peripheral Access Error occurs while accessing the DSU, and will generate an interrupt request if INTENCLR/SET.ERR is '1'.

Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the DSU interrupt flag.

26.7.8 Peripheral Interrupt Flag Status – Bridge C

Name: INTFLAGC
Offset: 0x1C
Reset: 0x00000000
Property: –

These flags are set when a Peripheral Access Error occurs while accessing the peripheral associated with the respective INTFLAGx bit and will generate an interrupt request if INTENCLR/SET.ERR is '1'.

Writing a '0' to these bits has no effect.

Writing a '1' to these bits will clear the corresponding INTFLAGx interrupt flag.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	AC	CCL						
Access	RW	RW						
Reset	0	0						

Bit 7 – AC Interrupt Flag for AC
This flag is set when a Peripheral Access Error occurs while accessing the peripheral associated with the AC and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the AC interrupt flag.

Bit 6 – CCL Interrupt Flag for CCL
This flag is set when a Peripheral Access Error occurs while accessing the peripheral associated with the CCL and will generate an interrupt request if INTENCLR/SET.ERR is '1'.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the CCL interrupt flag.

26.7.9 Peripheral Write Protection Status A

Name: STATUSA
Offset: 0x34
Reset: 0x00010000
Property: PAC Write-Protection

Writing to this register has no effect.

Reading STATUS register returns peripheral write protection status:

Value	Description
0	Peripheral is not write protected.
1	Peripheral is write protected.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
						TCC1		TC3
Access						R		R
Reset						0		0

Bit	7	6	5	4	3	2	1	0
	TC2	TC1	TC0	SERCOM1	SERCOM0	EIC	FREQM	PAC
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit 10 – TCC1 TCC1 APB Protect Enable

Value	Description
0	TCC1 is not write protected
1	TCC1 is write protected

Bit 8 – TC3 TC3 APB Protect Enable

Value	Description
0	TC3 is not write protected
1	TC3 is write protected

Bit 7 – TC2 TC2 APB Protect Enable

Value	Description
0	TC2 is not write protected
1	TC2 is write protected

Bit 6 – TC1 TC1 APB Protect Enable

Value	Description
0	TC1 is not write protected
1	TC1 is write protected

Bit 5 – TC0 TC0 APB Protect Enable

Value	Description
0	TC0 is not write protected
1	TC0 is write protected

Bit 4 – SERCOM1 SERCOM1 APB Protect Enable

Value	Description
0	SERCOM1 is not write protected
1	SERCOM1 is write protected

Bit 3 – SERCOM0 SERCOM0 APB Protect Enable

Value	Description
0	SERCOM0 is not write protected
1	SERCOM0 is write protected

Bit 2 – EIC EIC APB Protect Enable

Value	Description
0	EIC is not write protected
1	EIC is write protected

Bit 1 – FREQM FREQM APB Protect Enable

Value	Description
0	FREQM is not write protected
1	FREQM is write protected

Bit 0 – PAC PAC APB Protect Enable

Value	Description
0	PAC is not write protected
1	PAC is write protected

26.7.10 Peripheral Write Protection Status – Bridge B

Name: STATUSB
Offset: 0x38
Reset: 0x00000002
Property: PAC Write-Protection

Writing to this register has no effect.

Reading STATUS register returns peripheral write protection status:

Value	Description
0	Peripheral is not write protected.
1	Peripheral is write protected.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
				RAMECC	EVSYS	DMAC		
Access				R	R	R		
Reset				0	0	0		

Bit 4 – RAMECC RAMECC APB Protect Enable

Value	Description
0	RAMECC peripheral is not write protected
1	RAMECC peripheral is write protected

Bit 3 – EVSYS EVSYS APB Protect Enable

Value	Description
0	EVSYS peripheral is not write protected
1	EVSYS peripheral is write protected

Bit 2 – DMAC DMAC APB Protect Enable

Value	Description
0	DMAC peripheral is not write protected
1	DMAC peripheral is write protected

26.7.11 Peripheral Write Protection Status – Bridge C

Name: STATUSC
Offset: 0x3C
Reset: 0x00000000
Property: PAC Write-Protection

Writing to this register has no effect.

Reading STATUS register returns peripheral write protection status:

Value	Description
0	Peripheral is not write protected.
1	Peripheral is write protected.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	AC	CCL		SERCOM3	SERCOM2			
Access	R	R		R	R			
Reset	0	0		0	0			

Bit 7 – AC AC APB Protection Enable

Value	Description
0	Peripheral is not write protected
1	Peripheral is write protected

Bit 6 – CCL CCL APB Protection Enable

Value	Description
0	Peripheral is not write protected
1	Peripheral is write protected

Bit 4 – SERCOM3 SERCOM3 APB Protection Enable

Value	Description
0	Peripheral is not write protected
1	Peripheral is write protected

Bit 3 – SERCOM2 SERCOM2 APB Protection Enable

Value	Description
0	Peripheral is not write protected
1	Peripheral is write protected

27. Frequency Meter (FREQM)

27.1 Overview

The Frequency Meter (FREQM) can be used to accurately measure the frequency of a clock by comparing it to a known reference clock.

27.2 Features

- Ratio can be measured with 24-bit accuracy
- Accurately measures the frequency of an input clock with respect to a reference clock
- Reference clock can be selected from the available GCLK_FREQM_REF sources
- Measured clock can be selected from the available GCLK_FREQM_MSR sources

27.3 Block Diagram

Figure 27-1. FREQM Block Diagram

27.4 Signal Description

Not applicable.

27.5 Product Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

27.5.1 I/O Lines

The REFO lines (REFO[4:1]) can be used as measurement or reference clock sources. This requires the I/O pins to be configured.

27.5.2 Power Management

The FREQM will continue to operate in idle sleep mode where the selected source clock is running. The FREQM's interrupts can be used to wake up the device from idle sleep mode. See *Power Management Unit (PMU)* from Related Links for details on the different sleep modes.

Related Links

15. Power Management Unit (PMU)

27.5.3 Clocks

Two generic clocks are used by the FREQM: Reference Clock (GCLK_FREQM_REF) and Measurement Clock (GCLK_FREQM_MSR).

GCLK_FREQM_REF is required to clock the internal reference timer, which acts as the frequency reference.

GCLK_FREQM_MSR is required to clock a ripple counter for frequency measurement. These clocks must be configured and enabled in the generic clock controller before using the FREQM.

27.5.4 DMA

Not applicable.

27.5.5 Interrupts

The interrupt request line is connected to the interrupt controller. Using FREQM interrupt requires the interrupt controller to be configured first.

27.5.6 Events

Not applicable.

27.5.7 Debug Operation

When the CPU is halted in debug mode the FREQM continues its normal operation. The FREQM cannot be halted when the CPU is halted in debug mode. If the FREQM is configured in a way that requires it to be periodically serviced by the CPU, improper operation or data loss may result during debugging.

27.5.8 Register Access Protection

All registers with write access can be write-protected optionally by the Peripheral Access Controller (PAC), except the following registers:

- Control B register (CTRLB)
- Interrupt Flag Status and Clear register (INTFLAG)
- Status register (STATUS)

Optional write protection by the Peripheral Access Controller (PAC) is denoted by the "PAC Write Protection" property in each individual register description.

Write-protection does not apply to accesses through an external debugger.

27.6 Functional Description

27.6.1 Principle of Operation

FREQM counts the number of periods of the measured clock (GCLK_FREQM_MSR) with respect to the reference clock (GCLK_FREQM_REF). The measurement is done for a period of REFNUM/f_{CLK_REF} and stored in the Value register (VALUE.VALUE). REFNUM is the number of Reference clock cycles selected in the Configuration A register (CFGA.REFNUM).

The frequency of the measured clock, f_{CLK_MSR}, is calculated by

$f_{CLK\_MSR} = \left(\frac{VALUE}{REFNUM}\right) f_{CLK\_REF}$. The error can be maximum two measured clock cycles.

27.6.2 Basic Operation

27.6.2.1 Initialization

Before enabling FREQM, the device and peripheral must be configured:

- Write the number of Reference clock cycles for which the measurement is to be done in the Configuration A register (CFGA.REFNUM). This must be a non-zero number.
- Configuration A register (CFGA)

Enable-protection is denoted by the "Enable-Protected" property in the register description.

27.6.2.2 Enabling, Disabling and Resetting

The FREQM is enabled by writing a '1' to the Enable bit in the Control A register (CTRLA.ENABLE). The peripheral is disabled by writing CTRLA.ENABLE=0.

The FREQM is reset by writing a '1' to the Software Reset bit in the Control A register (CTRLA.SWRST). On software reset, all registers in the FREQM will be reset to their initial state, and the FREQM will be disabled.

Then ENABLE and SWRST bits are write-synchronized.

Related Links

27.6.7. Synchronization

27.6.2.3 Measurement

In the Configuration A register, the Number of Reference Clock Cycles field (CFGA.REFNUM) selects the duration of the measurement. The measurement is given in number of GCLK_FREQM_REF periods.
Note: The REFNUM field must be written before the FREQM is enabled.

After the FREQM is enabled, writing a '1' to the START bit in the Control B register (CTRLB.START) starts the measurement. The BUSY bit in Status register (STATUS.BUSY) is set when the measurement starts, and cleared when the measurement is complete.

There is also an interrupt request for Measurement Done: When the Measurement Done bit in Interrupt Enable Set register (INTENSET.DONE) is '1' and a measurement is finished, the Measurement Done bit in the Interrupt Flag Status and Clear register (INTFLAG.DONE) will be set and an interrupt request is generated.

The result of the measurement can be read from the Value register (VALUE.VALUE). The frequency of the measured clock GCLK_FREQM_MSR is then:

$$f_{\mathrm{CLK\_MSR}} = \left(\frac{\mathrm{VALUE}}{\mathrm{REFNUM}}\right) f_{\mathrm{CLK\_REF}}$$

Notes:

1. In order to make sure the measurement result (VALUE.VALUE[23:0]) is valid, the overflow status (STATUS.OVF) must be checked.
2. Due to asynchronous operations, the VALUE Error measurement can be up to two samples.

If an overflow condition occurred, indicated by the overflow bit in the STATUS register (STATUS.OVF), either the number of reference clock cycles must be reduced (CFGA.REFNUM) or a faster reference clock must be configured. Once the configuration is adjusted, clear the overflow status by writing a '1' to STATUS.OVF. Then, another measurement can be started by writing a '1' to CTRLB.START.

Note: See *CFGA*, *CTRLB*, *STATUS*, *INTENSET*, *INTFLAG*, *VALUE* registers in the *Register Summary - FREQM* from Related Links.

Related Links
27.7. Register Summary - FREQM

27.6.3 DMA Operation

Not applicable.

27.6.4 Interrupts

- DONE: A frequency measurement is done.

The interrupt flag in the Interrupt Flag Status and Clear INTLFLAG register is set when the interrupt condition occurs. The interrupt can be enabled by writing a '1' to the corresponding bit in the Interrupt Enable Set register, and disabled by writing a '1' to the corresponding bit in the Interrupt Enable Clear (INTENCLR) register. The status of enabled interrupts can be read from either INTENSET or INTENCLR.

An interrupt request is generated when the interrupt flag is set and the corresponding interrupt is enabled. The interrupt request remains active until the interrupt flag is cleared, the interrupt is disabled, or the FREQM is reset. See INTLFLAG for details on how to clear interrupt flags. All interrupt requests from the peripheral are ORed together on system level to generate one combined interrupt request to the NVIC. The user must read the INTLFLAG register to determine which interrupt condition is present.

This interrupt is a synchronous wake-up source.

Note: Interrupts must be globally enabled for interrupt requests to be generated.

27.6.5 Events

Not applicable.

27.6.6 Sleep Mode Operation

For lowest chip power consumption in sleep modes, FREQM must be disabled before entering a Sleep mode.

27.6.7 Synchronization

Due to asynchronicity between the main clock domain and the peripheral clock domains, some registers need to be synchronized when written or read.

The following bits and registers are write-synchronized:

- Software Reset bit in Control A register (CTRLA.SWRST)
- Enable bit in Control A register (CTRLA.ENABLE)

Required write synchronization is denoted by the "Write-Synchronized" property in the register description.

27.7 Register Summary - FREQM

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0							ENABLE	SWRST
0x01	CTRLB	7:0								START
0x02	CFGA	7:0	REFNUM[7:0]							
		15:8								
0x04 ... 0x07	Reserved									
0x08	INTENCLR	7:0								DONE
0x09	INTENSET	7:0								DONE
0x0A	INTFLAG	7:0								DONE
0x0B	STATUS	7:0							OVF	BUSY
0x0C	SYNCBUSY	7:0							ENABLE	SWRST
		15:8								
		23:16								
		31:24								
0x10	VALUE	7:0	VALUE[7:0]							
		15:8	VALUE[15:8]							
		23:16	VALUE[23:16]							
		31:24								

27.8 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register, and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers require synchronization when read and/or written. Synchronization is denoted by the “Read-Synchronized” and/or “Write-Synchronized” property in each individual register description.

Some registers are enable-protected, meaning they can only be written when the module is disabled. Enable-protection is denoted by the “Enable-Protected” property in each individual register description.

Some registers are optionally write protected by the Peripheral Access Controller (PAC). Optional PAC write protection is denoted by the “PAC Write-Protection” property in each individual register description.

27.8.1 Control A

Name: CTRLA
Offset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
							ENABLE	SWRST
Access							R/W	R/W
Reset							0	0

Bit 1 – ENABLE Enable

Due to synchronization there is delay from writing CTRLA.ENABLE until the peripheral is enabled or disabled. The value written to CTRLA.ENABLE will read back immediately and the ENABLE bit in the Synchronization Busy register (SYNCBUSY.ENABLE) will be set. SYNCBUSY.ENABLE will be cleared when the operation is complete.

Value	Description
0	The peripheral is disabled.
1	The peripheral is enabled.

Bit 0 – SWRST Software Reset

Writing a '0' to this bit has no effect.
Writing a '1' to this bit resets all registers in the FREQM to their initial state, and the FREQM will be disabled. Writing a '1' to this bit will always take precedence, meaning that all other writes in the same write-operation will be discarded.

Notes:

1. When the CTRLA.SWRST is written, the user must poll the SYNCBUSY.SWRST bit to know when the reset operation is complete.
2. During a SWRST, access to registers/bits without SWRST are disallowed until the SYNCBUSY.SWRST is cleared by hardware.

Value	Description
0	There is no ongoing Reset operation.
1	The Reset operation is ongoing.

27.8.2 Control B

Name: CTRLB
Offset: 0x01
Reset: 0x00
Property: –

Bit	7	6	5	4	3	2	1	0
								START
Access								W
Reset								0

Bit 0 – START Start Measurement

Value	Description
0	Writing a '0' has no effect.
1	Writing a '1' starts a measurement.

27.8.3 Configuration A

Name: CFGA
Offset: 0x02
Reset: 0x0000
Property: PAC Write-Protection, Enable-protected

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	REFNUM[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – REFNUM[7:0] Number of Reference Clock Cycles
Selects the duration of a measurement in number of CLK_FREQM_REF cycles. This must be a non-zero value, i.e. 0x01 (one cycle) to 0xFF (255 cycles).

27.8.4 Interrupt Enable Clear

Name: INTENCLR
Offset: 0x08
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
								DONE
Access								R/W
Reset								0

Bit 0 – DONE Measurement Done Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the Measurement Done Interrupt Enable bit, which disables the Measurement Done interrupt.

Value	Description
0	The Measurement Done interrupt is disabled.
1	The Measurement Done interrupt is enabled.

27.8.5 Interrupt Enable Set

Name: INTENSET
Offset: 0x09
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
								DONE
Access								R/W
Reset								0

Bit 0 – DONE Measurement Done Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will set the Measurement Done Interrupt Enable bit, which enables the Measurement Done interrupt.

Value	Description
0	The Measurement Done interrupt is disabled.
1	The Measurement Done interrupt is enabled.

27.8.6 Interrupt Flag Status and Clear

Name: INTFLAG
Offset: 0x0A
Reset: 0x00
Property: –

Bit	7	6	5	4	3	2	1	0
								DONE
Access								R/W
Reset								0

Bit 0 – DONE Mesurement Done
This flag is cleared by writing a '1' to it.
This flag is set when a new measurement is completed.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit clears the DONE interrupt flag.

27.8.7 Status

Name: STATUS
Offset: 0x0B
Reset: 0x00
Property: –

Bit	7	6	5	4	3	2	1	0
							OVF	BUSY
Access							R/W	R
Reset							0	0

Bit 1 – OVF Sticky Count Value Overflow
This bit is cleared by writing a '1' to it.
This bit is set when an overflow condition occurs to the value counter.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the OVF status.

Bit 0 – BUSY FREQM Status

Value	Description
0	No ongoing frequency measurement.
1	Frequency measurement is ongoing.

27.8.8 Synchronization Busy

Name: SYNCBUSY
Offset: 0x0C
Reset: 0x00000000
Property: –

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
							ENABLE	SWRST
Access							R	R
Reset							0	0

Bit 1 – ENABLE Enable
This bit is cleared when the synchronization of CTRLA.ENABLE is complete.
This bit is set when the synchronization of CTRLA.ENABLE is started.

Bit 0 – SWRST Synchronization Busy
This bit is cleared when the synchronization of CTRLA.SWRST is complete.
This bit is set when the synchronization of CTRLA.SWRST is started.
Note: During a SWRST, access to registers/bits without SWRST are disallowed until SYNCBUSY.SWRST cleared by hardware.

27.8.9 Value

Name: VALUE
Offset: 0x10
Reset: 0x00000000
Property: –

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	VALUE[23:16]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	VALUE[15:8]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	VALUE[7:0]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bits 23:0 – VALUE[23:0] Measurement Value
Result from measurement.

28. Event System (EVSYS)

28.1 Overview

The Event System allows autonomous, low-latency, and configurable communication between peripherals.

Several peripherals can be configured to generate and/or respond to signals known as events. The exact condition to generate an event, or the action taken upon receiving an event, is specific to each peripheral. Peripherals that respond to events are called event users. Peripherals that generate events are called event generators. A peripheral can have one or more event generators and can have one or more event users.

Communication is made without CPU intervention and without consuming system resources, such as bus or RAM bandwidth. This reduces the load on the CPU and other system resources, compared to a traditional interrupt-based system.

28.2 Features

- 32 configurable event channels:
  - All channels can be connected to any event generator
  - All channels provide a pure asynchronous path
  - Twelve channels provide a resynchronized or synchronous path
- 69 event generators.
- 52 event users.
- Configurable edge detector.
- Peripherals can be event generators, event users, or both.
- SleepWalking and interrupt for operation in sleep modes.
- Software event generation.
- Each event user can choose which channel to respond to.
- Optional Static or Round-Robin interrupt priority arbitration.

28.3 Block Diagram

Figure 28-1. Event System Block Diagram

28.4 Product Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

28.4.1 I/O Lines

Not applicable.

28.4.2 Power Management

The EVSYS can be used to wake up the CPU from all sleep modes (Deep Sleep/BACKUP and Extreme Deep Sleep/OFF Mode), even if the clock used by the EVSYS channel and the EVSYS bus clock are disabled. See *Power Management Unit (PMU)* from Related Links for details on the different sleep modes.

Although the clock for the EVSYS is stopped, the device still can wake up the EVSYS clock. Some event generators can generate an event when their clocks are stopped. The generic clock for the channel (GCLK_EVSYS_CHANNEL_n) will be restarted if that channel uses a synchronized path or a resynchronized path. It does not need to wake the system from sleep.

Related Links
15. Power Management Unit (PMU)

28.4.3 Clocks

Each EVSYS channel which can be configured as synchronous or resynchronized has a dedicated generic clock (GCLK_EVSYS_CHANNEL_n). These are used for event detection and propagation for each channel. These clocks must be configured and enabled in the generic clock controller before using the EVSYS (see *Clock and Reset (CRU)* from Related Links).

Important: Only EVSYS channel 0 to 11 can be configured as synchronous or resynchronized.

Related Links
13. Clock and Reset Unit (CRU)

28.4.4 DMA

Not applicable.

28.4.5 Interrupts

The interrupt request line is connected to the interrupt controller. Using the EVSYS interrupts requires the interrupt controller to be configured first (see *Nested Vector Interrupt Controller (NVIC)* from Related Links).

Related Links
10.2. Nested Vector Interrupt Controller (NVIC)

28.4.6 Events

Not applicable.

28.4.7 Debug Operation

When the CPU is halted in Debug mode, this peripheral will continue normal operation. If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during debugging. This peripheral can be forced to halt operation during debugging.

28.4.8 Register Access Protection

Registers with write access can be optionally write-protected by the Peripheral Access Controller (PAC), except for the following:

- Channel Pending Interrupt (INTPEND)
- Channel n Interrupt Flag Status and Clear (CHINTFLAGn)

Note: Optional write protection is indicated by the "PAC Write Protection" property in the register description.

Write protection does not apply for accesses through an external debugger.

28.4.9 Analog Connections

Not applicable.

28.5 Functional Description

28.5.1 Principle of Operation

The Event System consists of channels which route the internal events from peripherals (generators) to other internal peripherals. Each event generator can be selected as source for multiple channels, but a channel cannot be set to use multiple event generators at the same time.

A channel path can be configured in asynchronous, synchronous or resynchronized mode of operation. The mode of operation must be selected based on the requirements of the application.

When using synchronous or resynchronized path, the Event System includes options to transfer events to users when rising, falling or both edges are detected on event generators.

See *Channel Path* from Related Links.

Related Links

28.5.2.6. Channel Path

28.5.2 Basic Operation

28.5.2.1 Initialization

Before enabling event routing within the system, the Event Users Multiplexer and Event Channels must be selected in the Event System (EVSYS), and the two peripherals that generate and use the event must be configured. Follow these steps to configure the event:

1. In the event generator peripheral, enable output of event by writing a '1' to the respective Event Output Enable bit ("EO") in the peripheral's Event Control register, for example, AC.EVCTRL.WINEO0, RTC.EVCTRL.OVFEO.
2. Configure the EVSYS:
   a. Configure the Event User multiplexer by writing the respective EVSYS.USERm register, refer to 28.5.2.3. User Multiplexer Setup.
   b. Configure the Event Channel by writing the respective EVSYS.CHANNELn register, refer to 28.5.2.4. Event System Channel.
3. Configure the action to be executed by the event user peripheral by writing to the Event Action bits (EVACT) in the respective Event control register, for example, TC.EVCTRL.EVACT.
   Note: This step is not applicable for all the peripherals.
4. In the event user peripheral, enable event input by writing a '1' to the respective Event Input Enable bit ("EI") in the peripheral's Event Control register, for example, AC.EVCTRL.IVEI0, ADC.EVCTRL.STARTEI.

28.5.2.2 Enabling, Disabling, and Resetting

The EVSYS is always enabled.

The EVSYS is reset by writing a ‘1’ to the Software Reset bit in the Control A register (CTRLA.SWRST). All registers in the EVSYS will be reset to their initial state and all ongoing events will be canceled.

Refer to CTRLA.SWRST register for details.

28.5.2.3 User Multiplexer Setup

The user multiplexer defines the channel to be connected to which event user. Each user multiplexer is dedicated to one event user. A user multiplexer receives all event channels output and must be configured to select one of these channels, as shown in Block Diagram section. The channel is selected with the Channel bit group in the User register (USERm.CHANNEL).

The user multiplexer must always be configured before the channel. A list of all available event users is found in the User (USERm) register description.

Related Links

28.3. Block Diagram

28.5.2.4 Event System Channel

An event channel can select one event from a list of event generators. Depending on configuration, the selected event could be synchronized, resynchronized or asynchronously sent to the users. When synchronization or resynchronization is required, the channel includes an internal edge detector, allowing the Event System to generate internal events when rising, falling or both edges are detected on the selected event generator.

An event channel is able to generate internal events for the specific software commands. A channel block diagram is shown in *Block Diagram* section.

Related Links

28.3. Block Diagram

28.5.2.5 Event Generators

Each event channel can receive the events form all event generators. All event generators are listed in the Event Generator bit field in the Channel n register (CHANNELn.EVGEN). For details on event generation, refer to the corresponding module chapter. The channel event generator is selected by the Event Generator bit group in the Channel register (CHANNELn.EVGEN). By default, the channels are not connected to any event generators (ie, CHANNELn.EVGEN = 0)

28.5.2.6 Channel Path

There are different ways to propagate the event from an event generator:

- Asynchronous path
- Resynchronized path

The path is decided by writing to the Path Selection bit group of the Channel register (CHANNELn.PATH).

Asynchronous Path

When using the asynchronous path, the events are propagated from the event generator to the event user without intervention from the Event System. The GCLK for this channel (GCLK_EVSYS_CHANNEL_n) is not mandatory, meaning that an event will be propagated to the user without any clock latency.

When the asynchronous path is selected, the channel cannot generate any interrupts, and the Channel x Status register (CHSTATUSx) is always zero. The edge detection is not required and must be disabled by software. Each peripheral event user has to select which event edge must trigger internal actions. For further details, refer to each peripheral chapter description.

Resynchronized Path

The resynchronized path are used when the event generator and the event channel do not share the same generator for the generic clock. When the resynchronized path is used, resynchronization of the event from the event generator is done in the channel.

When the resynchronized path is used, the channel is able to generate interrupts. The channel status bits in the Channel Status register (CHSTATUS) are also updated and available for use.

28.5.2.7 Edge Detection

When synchronous or resynchronized paths are used, edge detection must be enabled. The event system can execute edge detection in three different ways:

- Generate an event only on the rising edge
- Generate an event only on the falling edge
- Generate an event on rising and falling edges.

Edge detection is selected by writing to the Edge Selection bit group of the Channel register (CHANNELn.EDGSEL).

28.5.2.8 Event Latency

The latency from event generator to event user depends on the channel's configuration:

- Asynchronous Path: The maximum routing latency of an external event is related to the internal signal routing and it is device dependent.
- Resynchronized Path: The maximum routing latency of an external event is three GCLK_EVSYS_CHANNEL_n clock cycles.

The maximum propagation latency of a user event to the peripheral clock core domain is three peripheral clock cycles.

The event generators, event channel and event user clocks ratio must be selected in relation with the internal event latency constraints. Events propagation or event actions in peripherals may be lost if the clock setup violates the internal latencies.

28.5.2.9 The Overrun Channel n Interrupt

The Overrun Channel n Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAGn.OVR) will be set, and the optional interrupt will be generated in the following cases:

- One or more event users on channel n is not ready when there is a new event
- An event occurs when the previous event on channel m has not been handled by all event users connected to that channel

The flag will only be set when using resynchronized paths. In the case of asynchronous path, the INTFLAGn.OVR is always read as zero.

28.5.2.10 The Event Detected Channel n Interrupt

The Event Detected Channel n Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAGn.EVD) is set when an event coming from the event generator configured on channel n is detected.

The flag will only be set when using a resynchronized path. In the case of an asynchronous path, the INTFLAGn.EVD is always zero.

28.5.2.11 Channel Status

The Channel Status register (CHSTATUS) shows the status of the channels when using a synchronous or resynchronized path. There are two different status bits in CHSTATUS for each of the available channels:

- The CHSTATUSn.BUSYCH bit will be set when an event on the corresponding channel n has not been handled by all event users connected to that channel.
- The CHSTATUSn.RDYUSR bit will be set when all event users connected to the corresponding channel are ready to handle incoming events on that channel.

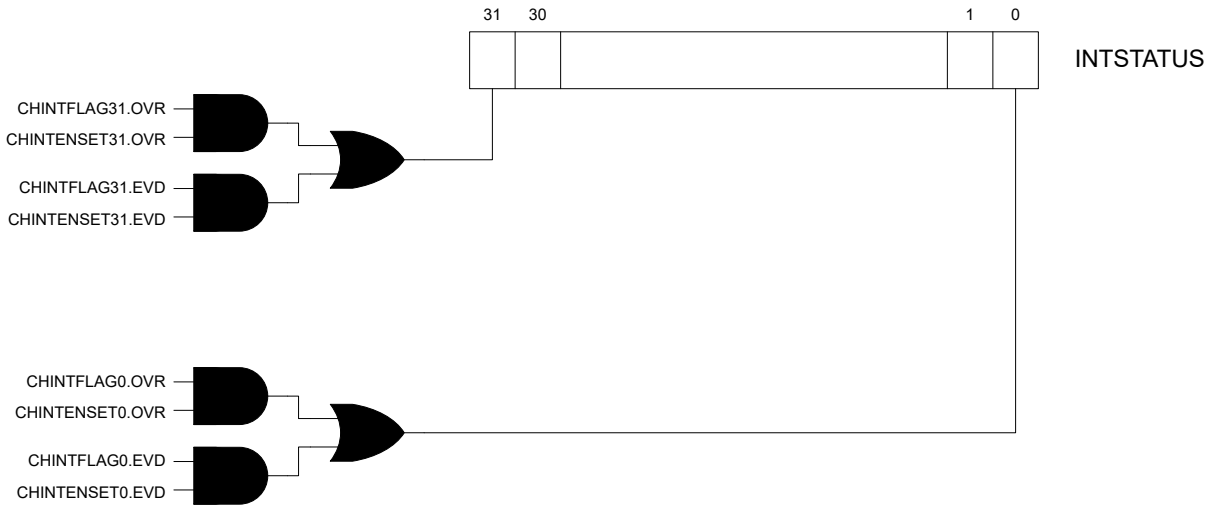
28.5.2.12 Software Event

A software event can be initiated on a channel by writing a '1' to the Software Event bit in the Channel register (SWEVT.CHANNELn). Then the software event can be serviced as any event generator; i.e., when a bit is set to ‘1’, the corresponding event will be generated on the respective channel.

28.5.2.13 Interrupt Status and Interrupts Arbitration

The Interrupt Status register stores all channels with pending interrupts, as shown below.

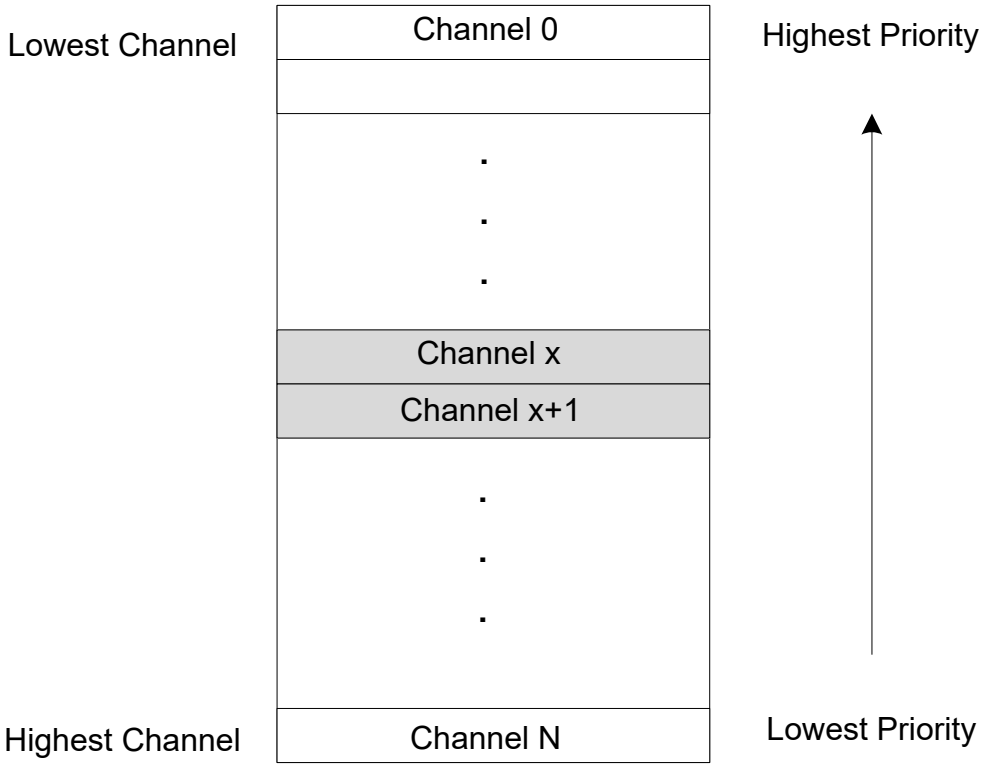
Figure 28-2. Interrupt Status Register



The Event System can arbitrate between all channels with pending interrupts. The arbiter can be configured to prioritize statically or dynamically the incoming events. The priority is evaluated each time a new channel has an interrupt pending, or an interrupt has been cleared. The Channel Pending Interrupt register (INTPEND) will provide the channel number with the highest interrupt priority, and the corresponding channel interrupt flags and status bits.

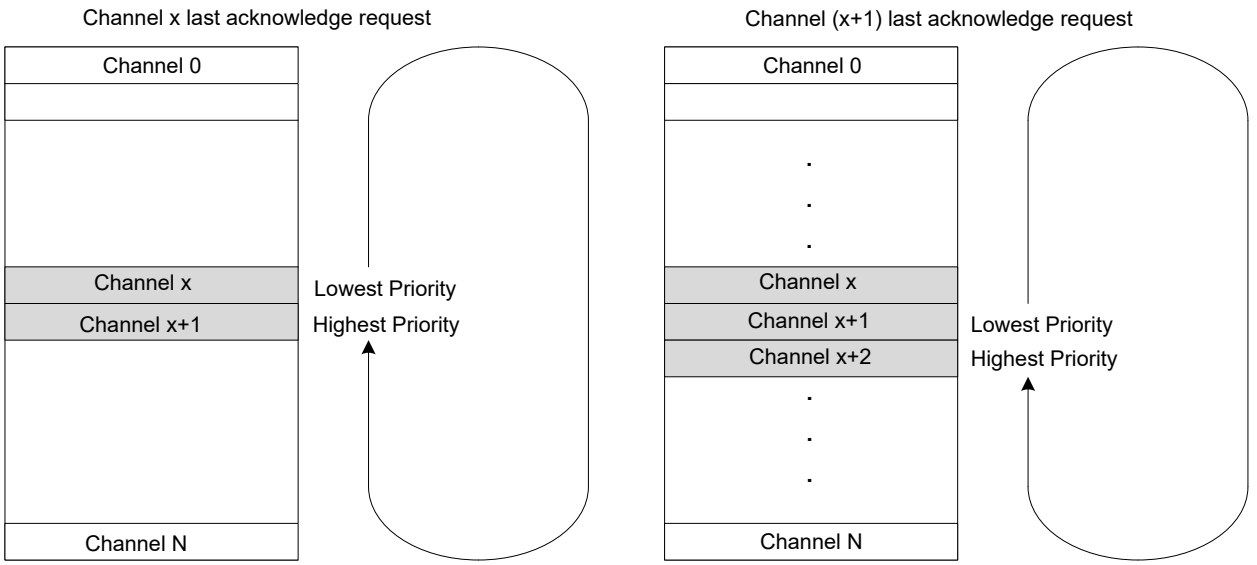
By default, static arbitration is enabled (PRICTRL.RRENx is '0'), the arbiter will prioritize a low channel number over a high channel number as shown below. When using the status scheme, there is a risk of high channel numbers never being granted access by the arbiter. This can be avoided using a dynamic arbitration scheme.

Figure 28-3. Static Priority



The dynamic arbitration scheme available in the Event System is round-robin. Round-robin arbitration is enabled by writing PRICTRL.RREN to one. With the round-robin scheme, the channel number of the last channel being granted access will have the lowest priority the next time the arbiter has to grant access to a channel, as shown below. The channel number of the last channel being granted access, will be stored in the Channel Priority Number bit group in the Priority Control register (PRICTRL.PRI).

Figure 28-4. Round-Robin Scheduling



The Channel Pending Interrupt register (INTPEND) also offers the possibility to indirectly clear the interrupt flags of a specific channel. Writing a flag to one in this register, will clear the corresponding interrupt flag of the channel specified by the INTPEND.ID bits.

28.5.3 Interrupts

The EVSYS has the following interrupt sources for each channel:

- Overrun Channel n interrupt (OVR)
- Event Detected Channel n interrupt (EVD)

These interrupts events are asynchronous wake-up sources.

Each interrupt source has an interrupt flag associated with it. The interrupt flag in the corresponding Channel n Interrupt Flag Status and Clear (CHINTFLAG) register is set when the interrupt condition occurs.

Note: Interrupts must be globally enabled to allow the generation of interrupt requests.

Each interrupt can be individually enabled by writing a '1' to the corresponding bit in the Channel n Interrupt Enable Set (CHINTENSET) register, and disabled by writing a '1' to the corresponding bit in the Channel n Interrupt Enable Clear (CHINTENCLR) register. An interrupt request is generated when the interrupt flag is set and the corresponding interrupt is enabled. The interrupt request remains active until the interrupt flag is cleared, the interrupt is disabled or the Event System is reset. All interrupt requests are ORed together on system level to generate one combined interrupt request to the NVIC.

The user must read the Channel Interrupt Status (INTSTATUS) register to identify the channels with pending interrupts, and must read the Channel n Interrupt Flag Status and Clear (CHINTFLAG) register to determine which interrupt condition is present for the corresponding channel. It is also possible to read the Interrupt Pending register (INTPEND), which provides the highest priority channel with pending interrupt and the respective interrupt flags.

28.5.4 Sleep Mode Operation

The Event System can generate interrupts to wake up the device from Idle or Standby mode.

To be able to run in standby, the run in Standby bit in the Channel register (CHANNELn.RUNSTDBY) must be set to '1'. When the Generic Clock On Demand bit in the Channel register

(CHANNELn.ONDEMAND) is set to '1' and the event generator is detected, the event channel will request its clock (GCLK_EVSYS_CHANNEL_n). The event latency for a resynchronized channel path will increase by two GCLK_EVSYS_CHANNEL_n clock (that is., up to five GCLK_EVSYS_CHANNEL_n clock cycles).

A channel will behave differently in different sleep modes regarding to CHANNELn.RUNSTDBY and CHANNELn.ONDEMAND:

Table 28-1. Event Channel Sleep Behavior

CHANNELn.PATH	CHANNELn. ONDEMAND	CHANNELn. RUNSTDBY	Sleep Behavior
ASYNC	0	0	Only run in Idle mode if an event must be propagated. Disabled in Standby mode.
SYNC/RESYNC	0	0	N/A. Works only in Active mode.
SYNC/RESYNC	0	1	Run in both Idle and Standby modes.
SYNC/RESYNC	1	0	Only run in Idle mode if an event must be propagated. Disabled in Standby mode. Two GCLK_EVSYS_n latency added in RESYNC path before the event is propagated internally.
SYNC/RESYNC	1	1	Run in both Idle and Standby modes. Two GCLK_EVSYS_n latency added in RESYNC path before the event is propagated internally.

Note: The ONDEMAND and RUNSTDBY bits have no effect for channels when asynchronous path is selected.

28.6 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0								SWRST
0x01 ... 0x03	Reserved									
0x04	SWEVT	7:0	CHANNEL7	CHANNEL6	CHANNEL5	CHANNEL4	CHANNEL3	CHANNEL2	CHANNEL1	CHANNEL0
		15:8	CHANNEL15	CHANNEL14	CHANNEL13	CHANNEL12	CHANNEL11	CHANNEL10	CHANNEL9	CHANNEL8
		23:16	CHANNEL23	CHANNEL22	CHANNEL21	CHANNEL20	CHANNEL19	CHANNEL18	CHANNEL17	CHANNEL16
		31:24	CHANNEL31	CHANNEL30	CHANNEL29	CHANNEL28	CHANNEL27	CHANNEL26	CHANNEL25	CHANNEL24
0x08	PRICTRL	7:0	RREN			PRI[4:0]				
0x09 ... 0x0F	Reserved									
0x10	INTPEND	7:0				ID[4:0]				
		15:8	BUSY	READY					EVD	OVR
0x12 ... 0x13	Reserved									
0x14	INTSTATUS	7:0	CHINT7	CHINT6	CHINT5	CHINT4	CHINT3	CHINT2	CHINT1	CHINT0
		15:8					CHINT11	CHINT10	CHINT9	CHINT8
		23:16								
		31:24								
0x18	BUSYCH	7:0	BUSYCH7	BUSYCH6	BUSYCH5	BUSYCH4	BUSYCH3	BUSYCH2	BUSYCH1	BUSYCH0
		15:8					BUSYCH11	BUSYCH10	BUSYCH9	BUSYCH8
		23:16								
		31:24								
0x1C	READYUSR	7:0	READYUSR7	READYUSR6	READYUSR5	READYUSR4	READYUSR3	READYUSR2	READYUSR1	READYUSR0
		15:8					READYUSR11	READYUSR10	READYUSR9	READYUSR8
		23:16								
		31:24								
0x20	CHANNEL0	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x24	CHINTENCLR0	7:0							EVD	OVR
0x25	CHINTENSET0	7:0							EVD	OVR
0x26	CHINTFLAG0	7:0							EVD	OVR
0x27	CHSTATUSn0	7:0							BUSYCH	RDYUSR
0x28	CHANNEL1	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x2C	CHINTENCLR1	7:0							EVD	OVR
0x2D	CHINTENSET1	7:0							EVD	OVR
0x2E	CHINTFLAG1	7:0							EVD	OVR
0x2F	CHSTATUSn1	7:0							BUSYCH	RDYUSR
0x30	CHANNEL2	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x34	CHINTENCLR2	7:0							EVD	OVR
0x35	CHINTENSET2	7:0							EVD	OVR
0x36	CHINTFLAG2	7:0							EVD	OVR
0x37	CHSTATUSn2	7:0							BUSYCH	RDYUSR
0x38	CHANNEL3	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x3C	CHINTENCLR3	7:0							EVD	OVR
0x3D	CHINTENSET3	7:0							EVD	OVR

.....continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x3E	CHINTFLAG3	7:0							EVD	OVR
0x3F	CHSTATUSn3	7:0							BUSYCH	RDYUSR
0x40	CHANNEL4	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x44	CHINTENCLR4	7:0							EVD	OVR
0x45	CHINTENSET4	7:0							EVD	OVR
0x46	CHINTFLAG4	7:0							EVD	OVR
0x47	CHSTATUSn4	7:0							BUSYCH	RDYUSR
0x48	CHANNEL5	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x4C	CHINTENCLR5	7:0							EVD	OVR
0x4D	CHINTENSET5	7:0							EVD	OVR
0x4E	CHINTFLAG5	7:0							EVD	OVR
0x4F	CHSTATUSn5	7:0							BUSYCH	RDYUSR
0x50	CHANNEL6	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x54	CHINTENCLR6	7:0							EVD	OVR
0x55	CHINTENSET6	7:0							EVD	OVR
0x56	CHINTFLAG6	7:0							EVD	OVR
0x57	CHSTATUSn6	7:0							BUSYCH	RDYUSR
0x58	CHANNEL7	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x5C	CHINTENCLR7	7:0							EVD	OVR
0x5D	CHINTENSET7	7:0							EVD	OVR
0x5E	CHINTFLAG7	7:0							EVD	OVR
0x5F	CHSTATUSn7	7:0							BUSYCH	RDYUSR
0x60	CHANNEL8	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x64	CHINTENCLR8	7:0							EVD	OVR
0x65	CHINTENSET8	7:0							EVD	OVR
0x66	CHINTFLAG8	7:0							EVD	OVR
0x67	CHSTATUSn8	7:0							BUSYCH	RDYUSR
0x68	CHANNEL9	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x6C	CHINTENCLR9	7:0							EVD	OVR
0x6D	CHINTENSET9	7:0							EVD	OVR
0x6E	CHINTFLAG9	7:0							EVD	OVR
0x6F	CHSTATUSn9	7:0							BUSYCH	RDYUSR
0x70	CHANNEL10	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x74	CHINTENCLR10	7:0							EVD	OVR
0x75	CHINTENSET10	7:0							EVD	OVR
0x76	CHINTFLAG10	7:0							EVD	OVR
0x77	CHSTATUSn10	7:0							BUSYCH	RDYUSR

.....continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x78	CHANNEL11	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x7C	CHINTENCLR11	7:0							EVD	OVR
0x7D	CHINTENSET11	7:0							EVD	OVR
0x7E	CHINTFLAG11	7:0							EVD	OVR
0x7F	CHSTATUSn11	7:0							BUSYCH	RDYUSR
0x80	CHANNEL12	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x84 ... 0x87	Reserved									
0x88	CHANNEL13	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x8C ... 0x8F	Reserved									
0x90	CHANNEL14	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x94 ... 0x97	Reserved									
0x98	CHANNEL15	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x9C ... 0x9F	Reserved									
0xA0	CHANNEL16	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xA4 ... 0xA7	Reserved									
0xA8	CHANNEL17	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xAC ... 0xAF	Reserved									
0xB0	CHANNEL18	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xB4 ... 0xB7	Reserved									
0xB8	CHANNEL19	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								

.....continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0xBC ... 0xBF	Reserved									
0xC0	CHANNEL20	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xC4 ... 0xC7	Reserved									
0xC8	CHANNEL21	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xCC ... 0xCF	Reserved									
0xD0	CHANNEL22	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xD4 ... 0xD7	Reserved									
0xD8	CHANNEL23	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xDC ... 0xDF	Reserved									
0xE0	CHANNEL24	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xE4 ... 0xE7	Reserved									
0xE8	CHANNEL25	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xEC ... 0xEF	Reserved									
0xF0	CHANNEL26	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xF4 ... 0xF7	Reserved									
0xF8	CHANNEL27	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0xFC ... 0xFF	Reserved									

.....continued										
Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x0100	CHANNEL28	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x0104 ... 0x0107	Reserved									
0x0108	CHANNEL29	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x010C ... 0x010F	Reserved									
0x0110	CHANNEL30	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x0114 ... 0x0117	Reserved									
0x0118	CHANNEL31	7:0	EVGEN[7:0]							
		15:8	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
		23:16								
		31:24								
0x011C ... 0x011F	Reserved									
0x0120	USER0	7:0	CHANNEL[7:0]							
...										
0x0153	USER51	7:0	CHANNEL[7:0]							

28.7 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register, and the 8-bit halves of a 16-bit register can be accessed directly.

Optional write protection by the Peripheral Access Controller (PAC) is denoted by the "PAC Write Protection" property in each individual register description.

For more details, see *Register Access Protection* and *Peripheral Access Controller (PAC)* from Related Links.

Related Links

28.4.8. Register Access Protection
26. Peripheral Access Controller (PAC)

28.7.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
								SWRST
Access								W
Reset								0

Bit 0 – SWRST Software Reset
Writing '0' to this bit has no effect.
Writing '1' to this bit resets all registers in the EVSYS to their initial state.
Note: Before applying a Software Reset it is recommended to disable the event generators.

28.7.2 Software Event

Name: SWEVT
Offset: 0x04
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
	CHANNEL31	CHANNEL30	CHANNEL29	CHANNEL28	CHANNEL27	CHANNEL26	CHANNEL25	CHANNEL24
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	CHANNEL23	CHANNEL22	CHANNEL21	CHANNEL20	CHANNEL19	CHANNEL18	CHANNEL17	CHANNEL16
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	CHANNEL15	CHANNEL14	CHANNEL13	CHANNEL12	CHANNEL11	CHANNEL10	CHANNEL9	CHANNEL8
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	CHANNEL7	CHANNEL6	CHANNEL5	CHANNEL4	CHANNEL3	CHANNEL2	CHANNEL1	CHANNEL0
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 – CHANNELx Channel x Software Selection [x=0..7]
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will trigger a software event for channel x.
These bits always return '0' when read.

28.7.3 Priority Control

Name: PRICTRL
Offset: 0x08
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
	RREN			PRI[4:0]				
Access	RW			RW	RW	RW	RW	RW
Reset	0			0	0	0	0	0

Bit 7 – RREN Round-Robin Scheduling Enable
For details on scheduling schemes, refer to Interrupt Status and Interrupts Arbitration

Value	Description
0	Static scheduling scheme for channels with level priority
1	Round-robin scheduling scheme for channels with level priority

Bits 4:0 – PRI[4:0] Channel Priority Number
When round-robin arbitration is enabled (PRICTRL.RREN=1) for priority level, this register holds the channel number of the last EVSYS channel being granted access as the active channel with priority level. The value of this bit group is updated each time the INTPEND or any of CHINTFLAG registers are written.
When static arbitration is enabled (PRICTRL.RREN=0) for priority level, and the value of this bit group is nonzero, it will not affect the static priority scheme.
This bit group is not reset when round-robin scheduling gets disabled (PRICTRL.RREN written to zero).

28.7.4 Channel Pending Interrupt

Name: INTPEND
Offset: 0x10
Reset: 0x4000

An interrupt that handles several channels must consult the INTPEND register to find out which channel number has priority (ignoring/filtering each channel that has its own interrupt line). An interrupt dedicated to only one channel must not use the INTPEND register.

Bit	15	14	13	12	11	10	9	8
	BUSY	READY					EVD	OVR
Access	R	R					RW	RW
Reset	0	1					0	0

Bit	7	6	5	4	3	2	1	0
				ID[4:0]				
Access				RW	RW	RW	RW	RW
Reset				0	0	0	0	0

Bit 15 – BUSY Busy
This bit is read '1' when the event on a channel selected by Channel ID field (ID) has not been handled by all the event users connected to this channel.

Bit 14 – READY Ready
This bit is read '1' when all event users connected to the channel selected by Channel ID field (ID) are ready to handle incoming events on this channel.

Bit 9 – EVD Channel Event Detected
This flag is set on the next CLK_EVSYS_APB cycle when an event is being propagated through the channel, and an interrupt request will be generated if CHINTENCLR/SET.EVD is '1'.
When the event channel path is asynchronous, the EVD bit will not be set.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear it. It will also clear the corresponding flag in the Channel n Interrupt Flag Status and Clear register (CHINTFLAGn) of this peripheral, where n is determined by the Channel ID bit field (ID) in this register.

Bit 8 – OVR Channel Overrun
This flag is set on the next CLK_EVSYS cycle after an overrun channel condition occurs, and an interrupt request will be generated if CHINTENCLR/SET.OVRx is '1'.
There are two possible overrun channel conditions:

- One or more of the event users on channel selected by Channel ID field (ID) are not ready when a new event occurs
- An event happens when the previous event on channel selected by Channel ID field (ID) has not yet been handled by all event users

When the event channel path is asynchronous, the OVR interrupt flag will not be set.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear it. It will also clear the corresponding flag in the Channel n Interrupt Flag Status and Clear register (CHINTFLAGn) of this peripheral, where n is determined by the Channel ID bit field (ID) in this register.

Bits 4:0 – ID[4:0] Channel ID
These bits store the channel number of the highest priority.

When the bits are written, indirect access to the corresponding Channel Interrupt Flag register is enabled.

28.7.5 Interrupt Status

Name: INTSTATUS
Offset: 0x14
Reset: 0x00000000

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
					CHINT11	CHINT10	CHINT9	CHINT8
Access					R	R	R	R
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
	CHINT7	CHINT6	CHINT5	CHINT4	CHINT3	CHINT2	CHINT1	CHINT0
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 – CHINTx Channel x Pending Interrupt
This bit is set when Channel x has a pending interrupt.
This bit is cleared when the corresponding Channel x interrupts are disabled, or the source interrupt sources are cleared.

28.7.6 Busy Channels

Name: BUSYCH
Offset: 0x18
Reset: 0x00000000

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
					BUSYCH11	BUSYCH10	BUSYCH9	BUSYCH8
Access					R	R	R	R
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
	BUSYCH7	BUSYCH6	BUSYCH5	BUSYCH4	BUSYCH3	BUSYCH2	BUSYCH1	BUSYCH0
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 – BUSYCHx Busy Channel x
This bit is set if an event occurs on channel x has not been handled by all event users connected to channel x.
This bit is cleared when channel x is idle.
When the event channel x path is asynchronous, this bit is always read '0'.

28.7.7 Ready Users

Name: READYUSR
Offset: 0x1C
Reset: 111111111111

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
					READYUSR11	READYUSR10	READYUSR9	READYUSR8
Access					R	R	R	R
Reset					1	1	1	1

Bit	7	6	5	4	3	2	1	0
	READYUSR7	READYUSR6	READYUSR5	READYUSR4	READYUSR3	READYUSR2	READYUSR1	READYUSR0
Access	R	R	R	R	R	R	R	R
Reset	1	1	1	1	1	1	1	1

Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 – READYUSRn Ready User for Channel n
This bit is set when all event users connected to channel n are ready to handle incoming events on channel n.
This bit is cleared when at least one of the event users connected to the channel is not ready.
When the event channel n path is asynchronous, this bit is always read zero.

28.7.8 Channel n Control

Name: CHANNEL
Offset: 0x20 + n*0x08 [n=0..31]
Reset: 0x00008000
Property: PAC Write-Protection, Mix-Secure

This register allows the user to configure channel n. To write to this register, do a single, 32-bit write of all the configuration data.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	ONDEMAND	RUNSTDBY			EDGSEL[1:0]		PATH[1:0]	
Access	RW	RW			RW	RW	RW	RW
Reset	1	0			0	0	0	0

Bit	7	6	5	4	3	2	1	0
	EVGEN[7:0]							
Access	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0

Bit 15 – ONDEMAND Generic Clock On Demand

Value	Description
0	Generic clock for a channel is always on, if the channel is configured and generic clock source is enabled.
1	Generic clock is requested on demand while an event is handled

Bit 14 – RUNSTDBY Run in Standby

This bit is used to define the behavior during standby sleep mode.

Value	Description
0	The channel is disabled in standby sleep mode.
1	The channel is not stopped in standby sleep mode and depends on the CHANNEL.ONDEMAND bit.

Bits 11:10 – EDGSEL[1:0] Edge Detection Selection

These bits set the type of edge detection to be used on the channel.
These bits must be written to zero when using the asynchronous path.

Value	Name	Description
0x0	NO_EVT_OUTPUT	No event output when using the resynchronized path
0x1	RISING_EDGE	Event detection only on the rising edge of the signal from the event generator
0x2	FALLING_EDGE	Event detection only on the falling edge of the signal from the event generator
0x3	BOTH_EDGES	Event detection on rising and falling edges of the signal from the event generator

Bits 9:8 – PATH[1:0] Path Selection

These bits are used to choose which path will be used by the selected channel.
Note: The path choice can be limited by the channel source (see *USERm* from Related Links).

Important: Only EVSYS channel 0 to 3 can be configured as synchronous or resynchronized.

Value	Name	Description
0x1	RESYNCHRONIZED	Resynchronized path
0x2	ASYNCHRONOUS	Asynchronous path
Other	-	Reserved

Bits 7:0 – EVGEN[7:0] Event Generator Selection
These bits are used to choose the event generator to connect to the selected channel.

Table 28-2. Event Generator Selection

Value	Name	Description
0x00 - 0x07	RTC_PERx	RTC period x=0..7
0x08 - 0x0B	RTC_CMPx	RTC comparison x=0..3
0x0C	RTC_TAMPER	RTC tamper detection
0x0D	RTC_OVF	RTC Overflow
0x0E - 0x11	EIC_EXTINTx	EIC external interrupt x=0..3
0x12 - 0x15	DMAC_CHx	DMA channel x=0..3
0x16	PAC_ACCERR	PAC Acc. error
0x17	TCC0_OVF	TCC0 Overflow
0x18	TCC0_TRG	TCC0 Trigger Event
0x19	TCC0_CNT	TCC0 Counter
0x1A-0x1F	TCC0_MCx	TCC0 Match/Compare x=0..5
0x20	TCC1_OVF	TCC1 Overflow
0x21	TCC1_TRG	TCC1 Trigger Event
0x22	TCC1_CNT	TCC1 Counter
0x23 - 0x28	TCC1_MCx	TCC1 Match/Compare x=0..5
0x29	TCC2_OVF	TCC2 Overflow
0x2A	TCC2_TRG	TCC2 Trigger Event
0x2B	TCC2_CNT	TCC2 Counter
0x2C - 0x2D	TCC2_MCx	TCC2 Match/Compare x=0..1
0x2E	TC0_OVF	TC0 Overflow
0x2F-0x30	TC0_MCx	TC0 Match/Compare x=0..1
0x31	TC1_OVF	TC1 Overflow
0x32 - 0x33	TC1_MCx	TC1 Match/Compare x=0..1
0x34	TC2_OVF	TC2 Overflow
0x35 - 0x36	TC2_MCx	TC2 Match/Compare x=0..1
0x37	TC3_OVF	TC3 Overflow
0x38 - 0x39	TC3_MCx	TC3 Match/Compare x=0..1
0x3A	ADC_RESRDY	ADC End-Of-Scan Ready Interrupt
0x3B - 0x3C	Not used	—
0x3D - 0x3E	AC_COMPx	AC Comparator, x=0..1
0x3F	AC_WIN_0	AC0 Window
0x40	TRNG_READY	TRNG ready
0x41 - 0x42	CCL_LUTOUTx	CCL LUTOUT x-0..1
0x43	ZB_TX_TS_ACTIVE	Zigbee Transmit Packet Active time
0x44	ZB_RX_TS_ACTIVE	Zigbee Receive Packet Active time

Related Links
28.7.13. USERm

28.7.9 Channel n Interrupt Enable Clear

Name: CHINTENCLR
Offset: 0x24 + n*0x08 [n=0..11]
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
							EVD	OVR
Access							RW	RW
Reset							0	0

Bit 1 – EVD Channel Event Detected Interrupt Disable
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the Event Detected Channel Interrupt Enable bit, which disables the Event Detected Channel interrupt.

Value	Description
0	The Event Detected Channel interrupt is disabled.
1	The Event Detected Channel interrupt is enabled.

Bit 0 – OVR Channel Overrun Interrupt Disable
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the Overrun Channel Interrupt Enable bit, which disables the Overrun Channel interrupt.

Value	Description
0	The Overrun Channel interrupt is disabled.
1	The Overrun Channel interrupt is enabled.

28.7.10 Channel n Interrupt Enable Set

Name: CHINTENSET
Offset: 0x25 + n*0x08 [n=0..11]
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
							EVD	OVR
Access							RW	RW
Reset							0	0

Bit 1 – EVD Channel Event Detected Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will set the Event Detected Channel Interrupt Enable bit, which enables the Event Detected Channel interrupt.

Value	Description
0	The Event Detected Channel interrupt is disabled.
1	The Event Detected Channel interrupt is enabled.

Bit 0 – OVR Channel Overrun Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will set the Overrun Channel Interrupt Enable bit, which enables the Overrun Channel interrupt.

Value	Description
0	The Overrun Channel interrupt is disabled.
1	The Overrun Channel interrupt is enabled.

28.7.11 Channel n Interrupt Flag Status and Clear

Name: CHINTFLAG
Offset: 0x26 + n*0x08 [n=0..11]
Reset: 0x00

Bit	7	6	5	4	3	2	1	0
							EVD	OVR
Access							RW	RW
Reset							0	0

Bit 1 – EVD Channel Event Detected
This flag is set on the next CLK_EVSYS_APB cycle when an event is being propagated through the channel, and an interrupt request will be generated if CHINTENCLR/SET.EVD is '1'.
When the event channel path is asynchronous, the EVD interrupt flag will not be set.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the Event Detected Channel interrupt flag.

Bit 0 – OVR Channel Overrun
This flag is set on the next CLK_EVSYS cycle after an overrun channel condition occurs, and an interrupt request will be generated if CHINTENCLR/SET.OVR is '1'.
There are two possible overrun channel conditions:

- One or more of the event users on the channel are not ready when a new event occurs.
- An event happens when the previous event on channel has not yet been handled by all event users.

When the event channel path is asynchronous, the OVR interrupt flag will not be set.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the Overrun Channel interrupt flag.

28.7.12 Channel n Status

Name: CHSTATUSn
Offset: 0x27 + n*0x08 [n=0..11]
Reset: 0x01

Bit	7	6	5	4	3	2	1	0
							BUSYCH	RDYUSR
Access							R	R
Reset							0	0

Bit 1 – BUSYCH Busy Channel
This bit is cleared when channel is idle.
This bit is set if an event on channel has not been handled by all event users connected to channel.
When the event channel path is asynchronous, this bit is always read '0'.

Bit 0 – RDYUSR Ready User
This bit is cleared when at least one of the event users connected to the channel is not ready.
This bit is set when all event users connected to channel are ready to handle incoming events on the channel.
When the event channel path is asynchronous, this bit is always read zero.

28.7.13 Event User m

Name: USERm
Offset: 0x0120 + m*0x01 [m=0..51]
Reset: 0x0
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
	CHANNEL[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – CHANNEL[7:0] Channel Event Selection
These bits select channel n to connect to the event user m.
Note: A value x of this bit field selects channel n = x-1.

USER*m*	UserMultiplexer	Description	PathType(1)
m = 0	RTC_TAMPER	RTCTamper	A, S, R
m = 1..8	DMAC_CH0..7	Channel0..7	S, R
m = 9	CM4_TRACE_START	CM4trace start	A, S, R
m = 10	CM4_TRACE_STOP	CM4trace stop	A, S, R
m = 11	CM4_TRACE_TRIG	CM4trace trigger	A, S, R
m = 12..13	TCC0EV0..1	TCC0 EVx	A, S, R
m = 14..19	TCC0MC0..5	TCC0 MCx	A, S, R
m = 20..21	TCC1EV0..1	TCC1 EVx	A, S, R
m = 22..27	TCC1MC0..5	TCC1 MCx	A, S, R
m = 28..29	TCC2EV0..1	TCC2 EVx	A, S, R
m = 30..31	TCC2MC0..1	TCC2 MCx	A, S, R
m = 32	TC0 EVU	TC0 EVU	A, S, R
m = 33	TC1 EVU	TC1 EVU	A, S, R
m = 34	TC2 EVU	TC2 EVU	A, S, R
m = 35	TC3 EVU	TC3 EVU	A, S, R
m = 36..47	ADC_TRIGGER5..16	ADC_TRIGGERx	A
m = 48..49	AC_SOC0..1	AC_SOCx	A, S, R
m = 50..51	CCL_LUTIN0..1	CCL_LUTINx	A, S, R

1) A = Asynchronous path, S = Synchronous path, R = Resynchronized path

Value	Description
11	12 bits (default)
10	10 bits
01	8 bits
00	6 bits

29. Serial Communication Interface (SERCOM)

29.1 Overview

There are instances of the Serial Communication interface (SERCOM) peripheral.

A SERCOM can be configured to support a number of modes: I^2C, SPI and USART. When an instance of SERCOM is configured and enabled, all of the resources of that SERCOM instance will be dedicated to the selected mode.

The SERCOM serial engine consists of a transmitter and receiver, baud-rate generator and address matching functionality. It can use the internal generic clock or an external clock. Using an external clock allows the SERCOM to be operated in all Sleep modes.

Note: Traditional Serial Communication Interface documentation uses the terminology “Master” and “Slave”. The equivalent Microchip terminology used in this document is “Host” and “Client”, respectively.

Note: SERCOM3 (4th instance of SERCOM) is only supported using Peripheral Pin Select (PPS).

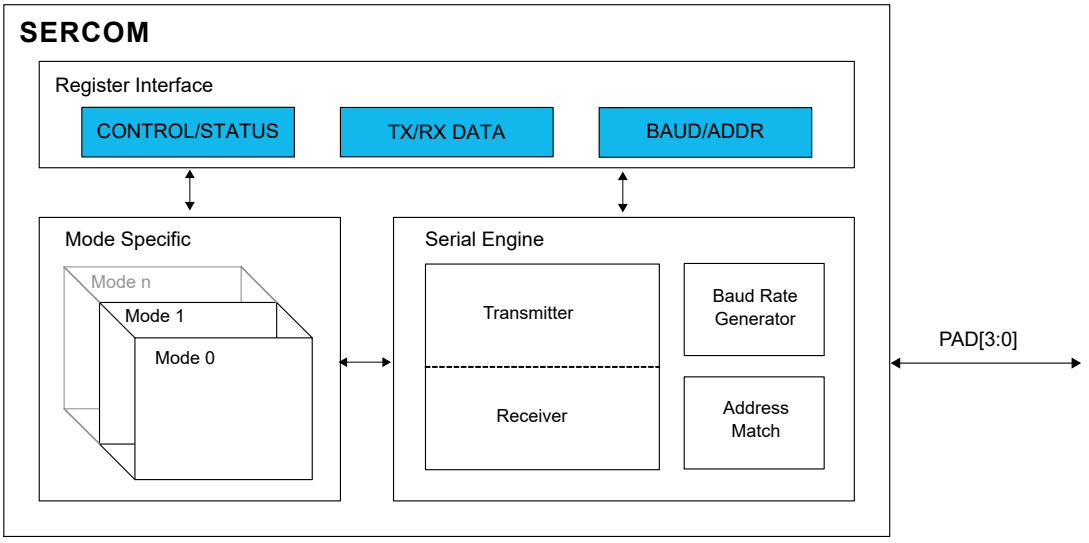
29.2 Features

- Interface for Configuring into one of the following (selected by CTRLA.MODE[2:0]):
  - Inter-Integrated Circuit (I^2C) two-wire serial interface
  - System Management Bus (SMBus™) compatible
  - Serial Peripheral Interface (SPI)
  - Universal Synchronous/Asynchronous Receiver/Transmitter (USART)
- Single Transmit Buffer and Double Receive Buffer
- Baud-rate Generator
- Address Match/mask Logic
- Operational in all Sleep modes with an External Clock Source
- Can be used with DMA

See the Related Links for full feature lists of the interface configurations.

29.3 Block Diagram

Figure 29-1. SERCOM Block Diagram



29.4 Signal Description

See the respective SERCOM mode chapters for details.

29.5 Product Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

29.5.1 I/O Lines

Using the SERCOM I/O lines requires the I/O pins to be configured using the System Configuration registers or PPS registers.

The SERCOM has four internal pads, PAD[3:0], and the signals from I^2C, SPI and USART are routed through these SERCOM pads through a multiplexer. The configuration of the multiplexer is available from the different SERCOM modes. Refer to the mode specific chapters for additional information.

29.5.2 Power Management

The SERCOM can operate in any Sleep mode provided the selected clock source is running. SERCOM interrupts can be configured to wake the device from sleep modes.

29.5.3 Clocks

The SERCOM uses two generic clocks: GCLK_SERCOMx_CORE and GCLK_SERCOMx_SLOWGCLK_SERCOMx_SLOW. The core clock (GCLK_SERCOMx_CORE) is required to clock the SERCOM while working as a host. The slow clock (GCLK_SERCOMx_SLOW) is only required for certain functions. See specific mode chapters for details.

These clocks must be configured and enabled in the Clock and Reset Unit (CRU) registers before using the SERCOM.

The generic clocks are asynchronous to the bus clock (PBx_CLK). Therefore, writing to certain registers will require synchronization between the clock domains.

29.5.4 DMA

The DMA request lines are connected to the DMA Controller (DMAC). The DMAC must be configured before the SERCOM DMA requests are used.

29.5.5 Interrupts

The interrupt request line is connected to the Interrupt Controller (NVIC). The NVIC must be configured before the SERCOM interrupts are used.

29.5.6 Events

Not applicable.

29.5.7 Debug Operation

When the CPU is halted in Debug mode, this peripheral will continue normal operation. If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during debugging. This peripheral can be forced to halt operation during debugging - refer to the Debug Control (DBGCTRL) register for details.

29.5.8 Register Access Protection

Registers with write-access can be write-protected optionally by the Peripheral Access Controller (PAC).

Optional write protection by the Peripheral Access Controller (PAC) is denoted by the "PAC Write Protection" property in each individual register description.

PAC write protection does not apply to accesses through an external debugger.

29.5.9 Analog Connections

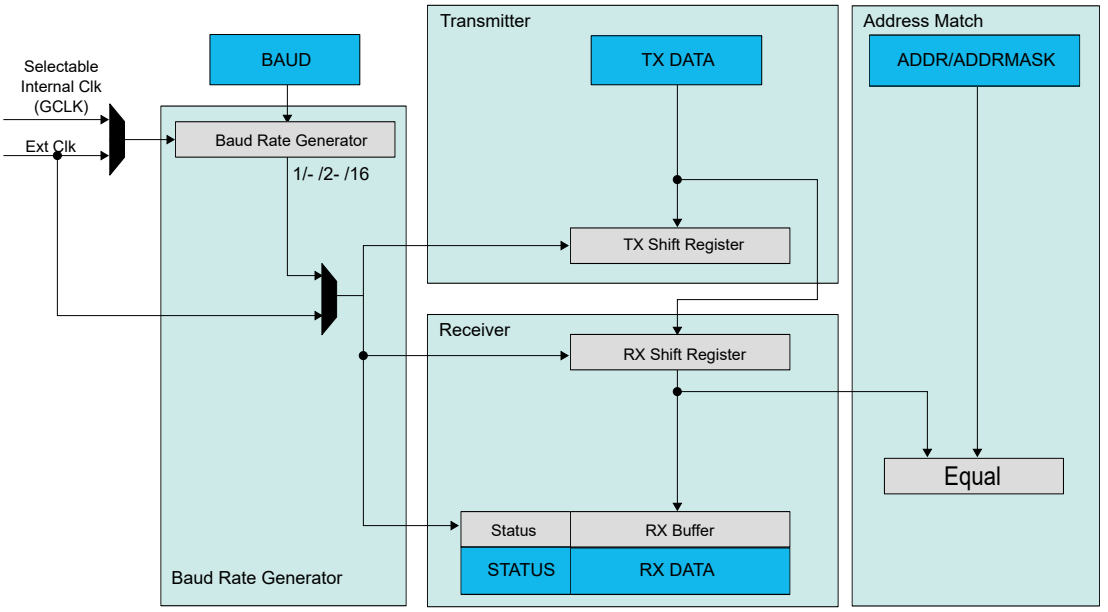
Not applicable.

29.6 Functional Description

29.6.1 Principle of Operation

The basic structure of the SERCOM serial engine is shown in *SERCOM Serial Engine*. Labels in capital letters are synchronous to the system clock and accessible by the CPU; labels in lowercase letters can be configured to run on the GCLK_SERCOMx_CORE clock or an external clock.

Figure 29-2. SERCOM Serial Engine



The transmitter consists of a single write buffer and a Shift register.

The receiver consists of a one-level (I^2C), or two-level (USART, SPI) receive buffer and a Shift register.

The baud-rate generator is capable of running on the GCLK_SERCOMx_CORE clock or an external clock.

Address matching logic is included for SPI and I^2C operation.

29.6.2 Basic Operation

29.6.2.1 Initialization

The SERCOM must be configured to the desired mode by writing the Operating Mode bits in the Control A register (CTRLA.MODE) as shown in the table below.

Table 29-1. SERCOM Modes

CTRLA.MODE	Description
0x0	USART with external clock
0x1	USART with internal clock
0x2	SPI in client operation
0x3	SPI in host operation
0x4	I^2C client operation
0x5	I^2C host operation

.....continued	
CTRLA.MODE	Description
0x6-0x7	Reserved

For further initialization information, see the respective SERCOM mode chapters:

29.6.2.2 Enabling, Disabling, and Resetting

This peripheral is enabled by writing '1' to the Enable bit in the Control A register (CTRLA.ENABLE), and disabled by writing '0' to it.

Writing '1' to the Software Reset bit in the Control A register (CTRLA.SWRST) will reset all registers of this peripheral to their initial states, except the DBGCTRL register, and the peripheral is disabled.

Refer to the CTRLA register description for details.

Related Links

30.8.1. CTRLA

29.6.2.3 Clock Generation – Baud-Rate Generator

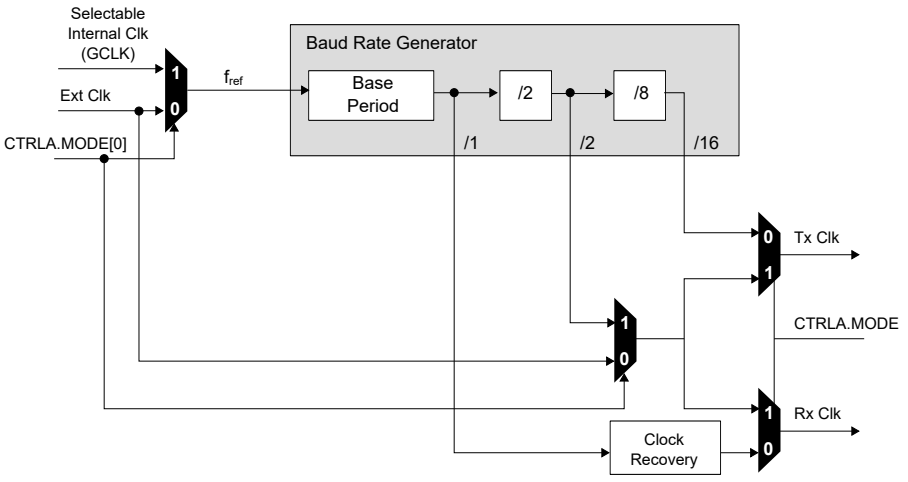
The baud-rate generator, as shown in the following figure, generates internal clocks for asynchronous and synchronous communication. The output frequency (f_{BAUD}) is determined by the Baud register (BAUD) setting and the baud reference frequency (f_{ref}). The baud reference clock is the serial engine clock, and it can be internal or external.

For asynchronous communication, the /16 (divide-by-16) output is used when transmitting, whereas the /1 (divide-by-1) output is used while receiving.

For synchronous communication, the /2 (divide-by-2) output is used.

This functionality is automatically configured, depending on the selected operating mode.

Figure 29-3. Baud Rate Generator



The following table contains equations for the baud rate (in bits per second) and the BAUD register value for each operating mode.

For asynchronous operation, there are two modes:

- *Arithmetic mode*: the BAUD register value is 16 bits (0 to 65,535)
- *Fractional mode*: the BAUD register value is 13 bits, while the fractional adjustment is 3 bits. In this mode the BAUD setting must be greater than or equal to 1.

For synchronous operation, the BAUD register value is 8 bits (0 to 255).

Table 29-2. Baud Rate Equations

Operating Mode	Condition	Baud Rate (Bits Per Second)	BAUD Register Value Calculation
Asynchronous Arithmetic	$f_{BAUD} \leq \frac{f_{ref}}{16}$	$f_{BAUD} = \frac{f_{ref}}{16}\left(1 - \frac{BAUD}{65536}\right)$	$BAUD = 65536 \cdot \left(1 - S \cdot \frac{f_{BAUD}}{f_{ref}}\right)$
Asynchronous Fractional	$f_{BAUD} \leq \frac{f_{ref}}{S}$	$f_{BAUD} = \frac{f_{ref}}{S \cdot \left(BAUD + \frac{FP}{8}\right)}$	$BAUD = \frac{f_{ref}}{S \cdot f_{BAUD}} - \frac{FP}{8}$
Synchronous	$f_{BAUD} \leq \frac{f_{ref}}{2}$	$f_{BAUD} = \frac{f_{ref}}{2 \cdot (BAUD + 1)}$	$BAUD = \frac{f_{ref}}{2 \cdot f_{BAUD}} - 1$

S - Number of samples per bit, which can be 16, 8, or 3.

The Asynchronous Fractional option is used for auto-baud detection.

The baud rate error is represented by the following formula:

$$\text{Error} = 1 - \left(\frac{\text{ExpectedBaudRate}}{\text{ActualBaudRate}}\right)$$

29.6.3 Additional Features

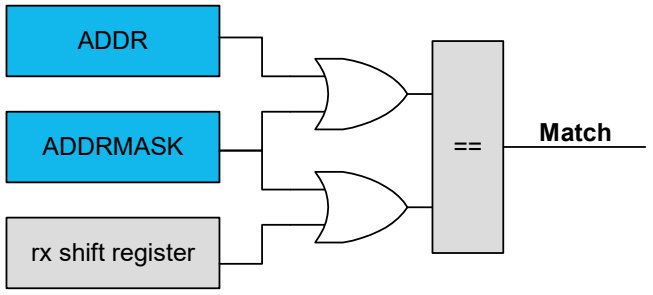
29.6.3.1 Address Match and Mask

The SERCOM address match and mask feature is capable of matching either one address, two unique addresses, or a range of addresses with a mask, based on the mode selected. The match uses seven or eight bits, depending on the mode.

29.6.3.1.1 Address With Mask

An address written to the Address bits in the Address register (ADDR.ADDR), and a mask written to the Address Mask bits in the Address register (ADDR.ADDRMASK) will yield an address match. All bits that are masked are not included in the match. Note that writing the ADDR.ADDRMASK to 'all zeros' will match a single unique address, while writing ADDR.ADDRMASK to 'all ones' will result in all addresses being accepted.

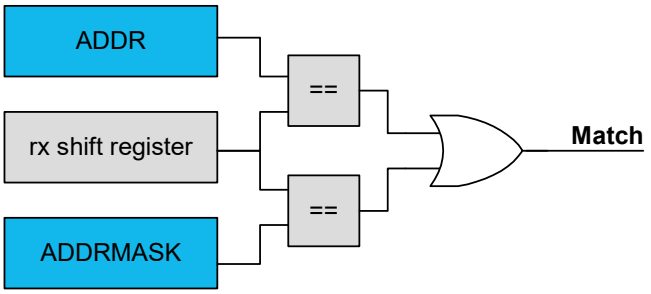
Figure 29-4. Address With Mask



29.6.3.1.2 Two Unique Addresses

The two addresses written to ADDR and ADDRMASK will cause a match.

Figure 29-5. Two Unique Addresses



29.6.3.1.3 Address Range

The range of addresses between and including ADDR.ADDR and ADDR.ADDRMASK will cause a match. ADDR.ADDR and ADDR.ADDRMASK can be set to any two addresses, with ADDR.ADDR acting as the upper limit and ADDR.ADDRMASK acting as the lower limit.

Figure 29-6. Address Range

29.6.4 DMA Operation

The available DMA interrupts and their depend on the operation mode of the SERCOM peripheral. Refer to the Functional Description sections of the respective SERCOM mode.

29.6.5 Interrupts

Interrupt sources are mode specific. See the respective SERCOM mode chapters for details.

Each interrupt source has its own Interrupt flag.

The Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG) will be set when the Interrupt condition is met.

Each interrupt can be individually enabled by writing '1' to the corresponding bit in the Interrupt Enable Set register (INTENSET), and disabled by writing '1' to the corresponding bit in the Interrupt Enable Clear register (INTENCLR).

An interrupt request is generated when the Interrupt flag is set and the corresponding interrupt is enabled. The interrupt request remains active until either the Interrupt flag is cleared, the interrupt is disabled, or the SERCOM is reset. For details on clearing Interrupt flags, refer to the INTFLAG register description.

The value of INTFLAG indicates which Interrupt condition occurred. The user must read the INTFLAG register to determine which Interrupt condition is present.

Note: Interrupts must be globally enabled for interrupt requests to be generated.

Related Links

30.8.8. INTFLAG

29.6.6 Events

Not applicable.

29.6.7 Sleep Mode Operation

The peripheral can operate in any Sleep mode where the selected serial clock is running. This clock can be external or generated by the internal baud-rate generator.

The SERCOM interrupts can be used to wake-up the device from Sleep modes. Refer to the different SERCOM mode chapters for details.

29.6.8 Synchronization

Due to asynchronicity between the main clock domain and the peripheral clock domains, some registers need to be synchronized when written or read.

Required write synchronization is denoted by the "Write-Synchronized" property in the register description.

Required read synchronization is denoted by the "Read-Synchronized" property in the register description.

30. SERCOM Synchronous and Asynchronous Receiver and Transmitter (SERCOM USART)

30.1 Overview

The Universal Synchronous and Asynchronous Receiver and Transmitter (USART) is one of the available modes in the Serial Communication Interface (SERCOM).

The USART uses the SERCOM transmitter and receiver (see *USART Block Diagram* in the *Block Diagram* section from Related Links). Labels in uppercase letters are synchronous to PBx_CLK and accessible for CPU. Labels in lowercase letters can be programmed to run on the internal generic clock or an external clock.

The transmitter consists of a single write buffer, a Shift register, and control logic for different frame formats. The write buffer supports data transmission without any delay between frames. The receiver consists of a two-level receive buffer and a Shift register. Status information of the received data is available for error checking. Data and clock recovery units ensure robust synchronization and noise filtering during asynchronous data reception.

Note: Traditional Universal Synchronous and Asynchronous Receiver and Transmitter (USART) documentation uses the terminology “Master” and “Slave”. The equivalent Microchip terminology used in this document is “Commander” and “Responder”, respectively.

Related Links

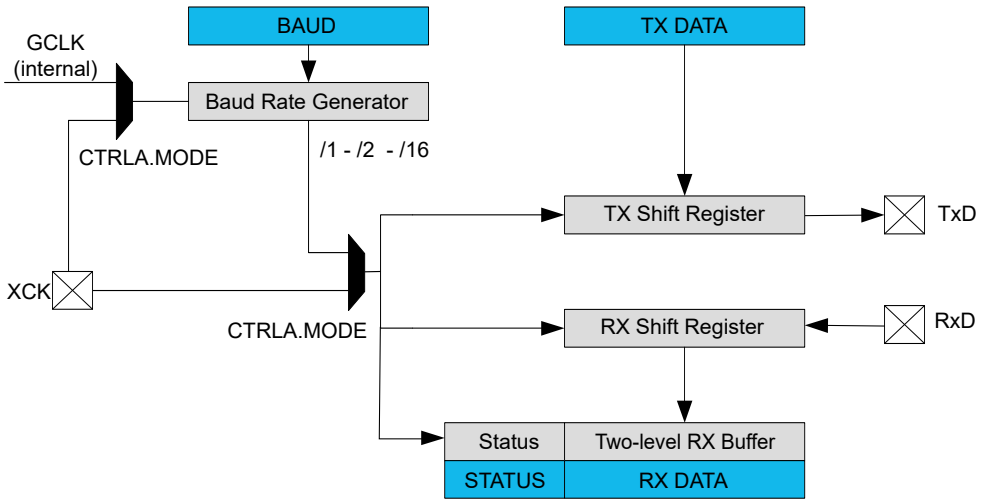
30.3. Block Diagram

30.2 USART Features

- Full-duplex Operation
- Asynchronous (with Clock Reconstruction) or Synchronous Operation
- Internal or External Clock source for Asynchronous and Synchronous Operation
- Baud-rate Generator
- Supports Serial Frames with 5, 6, 7, 8 or 9 Data bits and 1 or 2 Stop bits
- Odd or Even Parity Generation and Parity Check
- Selectable LSB- or MSB-first Data Transfer
- Buffer Overflow and Frame Error Detection
- Noise Filtering, Including False Start bit Detection and Digital Low-pass Filter
- Collision Detection
- Can Operate in all Sleep modes
- Operation at Speeds up to Half the System Clock for Internally Generated Clocks
- Operation at Speeds up to the System Clock for Externally Generated Clocks
- RTS and CTS Flow Control
- IrDA Modulation and Demodulation up to 115.2 kbps
- LIN Commander Support
- LIN Responder Support
  - Auto-baud and break character detection
- Start-of-frame detection
- Can work with DMA

30.3 Block Diagram

Figure 30-1. USART Block Diagram



30.4 Signal Description

Table 30-1. SERCOM USART Signals

Signal Name	Type	Description
PAD[3:0]	Digital I/O	General SERCOM pins

One signal can be mapped to one of several pins.

30.5 Product Dependencies

To use this peripheral, other parts of the system must be configured correctly, as described below.

30.5.1 I/O Lines

Using the USART's I/O lines requires the I/O pins to be configured using the System Configuration registers or PPS registers.

When the SERCOM is used in USART mode, the SERCOM controls the direction and value of the I/O pins according to the table below. If the receiver or transmitter is disabled, these pins can be used for other purposes.

Table 30-2. USART Pin Configuration

Pin	Pin Configuration
TxD	Output
RxD	Input
XCK	Output or input

The combined configuration of PORT and the Transmit Data Pinout and Receive Data Pinout bit fields in the Control A register (CTRLA.TXPO and CTRLA.RXPO, respectively) will define the physical position of the USART signals in the above table.

30.5.2 Power Management

This peripheral can continue to operate in any Sleep mode where its source clock is running. The interrupts can wake-up the device from Sleep modes.

30.5.3 Clocks

A generic clock (GCLK_SERCOMx_CORE) is required to clock the SERCOMx_CORE. This clock must be configured and enabled in the CRU registers before using the SERCOMx_CORE. See *Clock and Reset (CRU)* and *Peripheral Module Disable Register (PMD)* from Related Links.

This generic clock is asynchronous to the bus clock (PBx_CLK). Therefore, writing to certain registers will require synchronization to the clock domains.

Related Links

20. Peripheral Module Disable Register (PMD)
13. Clock and Reset Unit (CRU)

30.5.4 DMA

The DMA request lines are connected to the DMA Controller (DMAC). To use DMA requests with this peripheral, the DMAC must be configured first (see *Direct Memory Access Controller (DMAC)* from Related Links).

Related Links

22. Direct Memory Access Controller (DMAC)

30.5.5 Interrupts

The interrupt request line is connected to the Interrupt Controller. In order to use interrupt requests of this peripheral, the NVIC must be configured first. See *Nested Vector Interrupt Controller (NVIC)* from Related Links.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)
30.8.8. INTFLAG

30.5.6 Events

Not applicable.

30.5.7 Debug Operation

When the CPU is halted in Debug mode, this peripheral will continue normal operation. If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during debugging. This peripheral can be forced to halt operation during debugging - refer to the Debug Control (DBGCTRL) register for details.

Related Links

30.8.12. DBGCTRL

30.5.8 Register Access Protection

Registers with write access can be write-protected optionally by the Peripheral Access Controller (PAC).

PAC write protection is not available for the following registers:

- Interrupt Flag Clear and Status register (INTFLAG)
- Status register (STATUS)
- Data register (DATA)

Optional PAC write protection is denoted by the "PAC Write-Protection" property in each individual register description.

Write-protection does not apply to accesses through an external debugger.

30.5.9 Analog Connections

Not applicable.

30.6 Functional Description

30.6.1 Principle of Operation

The USART uses the following lines for data transfer:

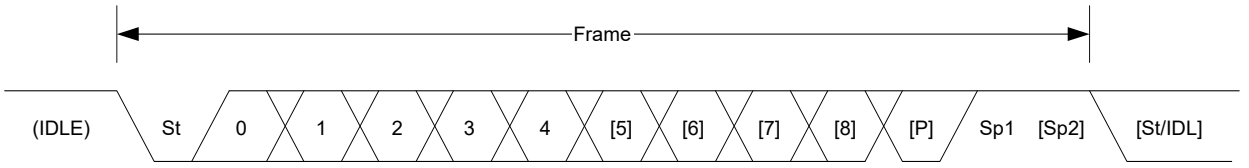
- RxD for receiving
- TxD for transmitting
- XCK for the transmission clock in synchronous operation

USART data transfer is frame based. A serial frame consists of:

- 1 start bit
- From 5 to 9 data bits (MSB or LSB first)
- No, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the Start bit followed by one character of Data bits. If enabled, the parity bit is inserted after the Data bits and before the first Stop bit. After the stop bit(s) of a frame, either the next frame can follow immediately, or the communication line can return to the Idle (high) state. The figure below illustrates the possible frame formats. Values inside brackets ([x]) denote optional bits.

Figure 30-2. Frame Formats



St	Start bit. Signal is always low.
n, [n]	Data bits. 0 to [5..9]
[P]	Parity bit. Either odd or even.
Sp, [Sp]	Stop bit. Signal is always high.
IDLE	No frame is transferred on the communication line. Signal is always high in this state.

30.6.2 Basic Operation

30.6.2.1 Initialization

The following registers are enable-protected, meaning they can only be written when the USART is disabled (CTRL.ENABLE=0):

- Control A register (CTRLA), except the Enable (ENABLE) and Software Reset (SWRST) bits.
- Control B register (CTRLB), except the Receiver Enable (RXEN) and Transmitter Enable (TXEN) bits.
- Baud register (BAUD)

When the USART is enabled or is being enabled (CTRLA.ENABLE=1), any writing attempt to these registers will be discarded. If the peripheral is being disabled, writing to these registers will be executed after disabling is completed. Enable-protection is denoted by the "Enable-Protection" property in the register description.

Before the USART is enabled, it must be configured by these steps:

1. Select either external (0x0) or internal clock (0x1) by writing the Operating Mode value in the CTRLA register (CTRLA.MODE).
2. Select either Asynchronous (0) or Synchronous (1) Communication mode by writing the Communication Mode bit in the CTRLA register (CTRLA.CMODE).
3. Select pin for receive data by writing the Receive Data Pinout value in the CTRLA register (CTRLA.RXPO).
4. Select pads for the transmitter and external clock by writing the Transmit Data Pinout bit in the CTRLA register (CTRLA.TXPO).
5. Configure the Character Size field in the CTRLB register (CTRLB.CHSIZE) for character size.
6. Set the Data Order bit in the CTRLA register (CTRLA.DORD) to determine MSB- or LSB-first data transmission.
7. To use parity mode:
   a. Enable Parity mode by writing 0x1 to the Frame Format field in the CTRLA register (CTRLA.FORM).
   b. Configure the Parity Mode bit in the CTRLB register (CTRLB.PMODE) for even or odd parity.
8. Configure the number of stop bits in the Stop Bit Mode bit in the CTRLB register (CTRLB.SBMODE).
9. When using an internal clock, write the Baud register (BAUD) to generate the desired baud rate.
10. Enable the transmitter and receiver by writing '1' to the Receiver Enable and Transmitter Enable bits in the CTRLB register (CTRLB.RXEN and CTRLB.TXEN).

30.6.2.2 Enabling, Disabling, and Resetting

This peripheral is enabled by writing '1' to the Enable bit in the Control A register (CTRLA.ENABLE), and disabled by writing '0' to it.

Writing '1' to the Software Reset bit in the Control A register (CTRLA.SWRST) will reset all registers of this peripheral to their initial states, except the DBGCTRL register, and the peripheral is disabled.

Refer to the CTRLA register description for details.

Related Links

30.8.1. CTRLA

30.6.2.3 Clock Generation and Selection

For both Synchronous and Asynchronous modes, the clock used for shifting and sampling data can be generated internally by the SERCOM baud-rate generator or supplied externally through the XCK line.

The Synchronous mode is selected by writing a '1' to the Communication Mode bit in the Control A register (CTRLA.CMODE), the Asynchronous mode is selected by writing '0' to CTRLA.CMODE.

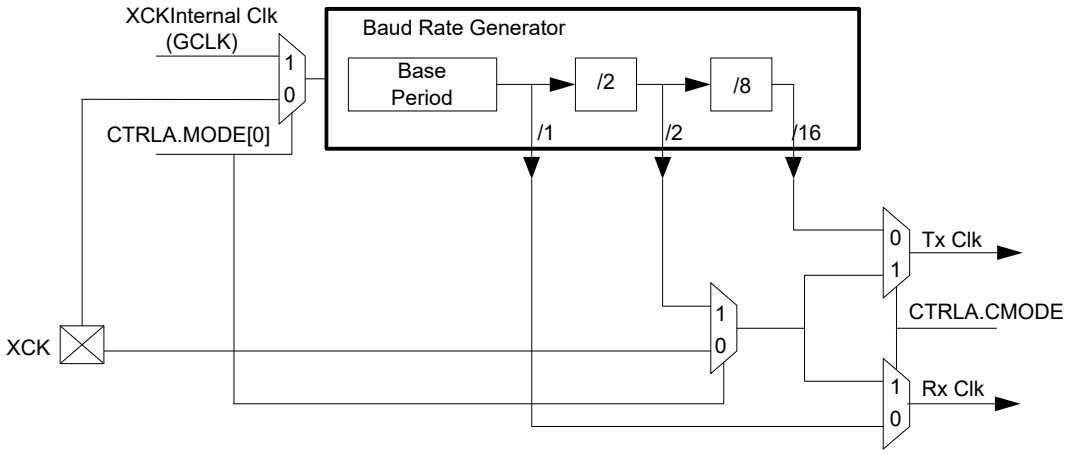
The internal clock source is selected by writing '1' to the Operation Mode bit field in the Control A register (CTRLA.MODE), the external clock source is selected by writing '0' to CTRLA.MODE.

The SERCOM baud-rate generator is configured as in the following figure.

In Asynchronous mode (CTRLA.CMODE=0), the 16-bit Baud register value is used.

In Synchronous mode (CTRLA.CMODE=1), the eight LSBs of the Baud register are used. For more details on configuring the baud rate (see *Clock Generation – Baud-Rate Generator* from Related Links).

Figure 30-3. Clock Generation



Related Links

29.6.2.3. Clock Generation – Baud-Rate Generator

30.6.2.3.1 Synchronous Clock Operation

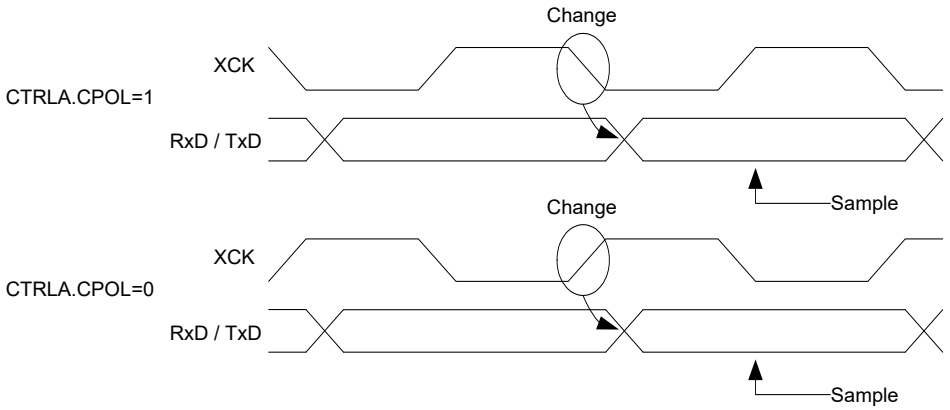
In Synchronous mode, the CTRLA.MODE bit field determines whether the transmission clock line (XCK) serves either as input or output. The dependency between clock edges, data sampling, and data change is the same for internal and external clocks. Data input on the RxD pin is sampled at the opposite XCK clock edge when data is driven on the TxD pin.

The Clock Polarity bit in the Control A register (CTRLA.CPOL) selects which XCK clock edge is used for RxD sampling, and which is used for TxD change:

When CTRLA.CPOL is '0', the data will be changed on the rising edge of XCK, and sampled on the falling edge of XCK.

When CTRLA.CPOL is '1', the data will be changed on the falling edge of XCK, and sampled on the rising edge of XCK.

Figure 30-4. Synchronous Mode XCK Timing



When the clock is provided through XCK (CTRLA.MODE=0x0), the Shift registers operate directly on the XCK clock. This means that XCK is not synchronized with the system clock and, therefore, can operate at frequencies up to the system frequency.

30.6.2.4 Data Register

The USART Transmit Data register (TxDATA) and USART Receive Data register (RxDATA) share the same I/O address, referred to as the Data register (DATA). Writing the DATA register will update the TxDATA register. Reading the DATA register will return the contents of the RxDATA register.

30.6.2.5 Data Transmission

Data transmission is initiated by writing the data to be sent into the DATA register. Then, the data in TxDATA will be moved to the Shift register when the Shift register is empty and ready to send a new frame. After the Shift register is loaded with data, the data frame will be transmitted.

When the entire data frame including Stop bit(s) has been transmitted and no new data was written to DATA, the Transmit Complete Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.TXC) will be set, and the optional interrupt will be generated.

The Data Register Empty flag in the Interrupt Flag Status and Clear register (INTFLAG.DRE) indicates that the register is empty and ready for new data. The DATA register must be written to when INTFLAG.DRE is set.

Disabling the Transmitter

The transmitter is disabled by writing '0' to the Transmitter Enable bit in the CTRLB register (CTRLB.TXEN).

Disabling the transmitter will complete only after any ongoing and pending transmissions are completed, in other words, there is no data in the transmit shift register and TxDATA to transmit.

30.6.2.5.1 Disabling the Transmitter

The transmitter is disabled by writing '0' to the Transmitter Enable bit in the CTRLB register (CTRLB.TXEN).

Disabling the transmitter will complete only after any ongoing and pending transmissions are completed, that is, there is no data in the Transmit Shift register and TxDATA to transmit.

30.6.2.6 Data Reception

The receiver accepts data when a valid Start bit is detected. Each bit following the Start bit will be sampled according to the baud rate or XCK clock, and shifted into the receive Shift register until the first Stop bit of a frame is received. The second Stop bit will be ignored by the receiver.

When the first Stop bit is received and a complete serial frame is present in the Receive Shift register, the contents of the Shift register will be moved into the two-level receive buffer. Then, the Receive Complete Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.RXC) will be set, and the optional interrupt will be generated.

The received data can be read from the DATA register when the Receive Complete Interrupt flag is set.

Disabling the Receiver

Writing '0' to the Receiver Enable bit in the CTRLB register (CTRLB.RXEN) will disable the receiver, flush the two-level receive buffer, and data from ongoing receptions will be lost.

Error Bits

The USART receiver has three error bits in the Status (STATUS) register: Frame Error (FERR), Buffer Overflow (BUFOVF), and Parity Error (PERR). Once an error happens, the corresponding error bit will be set until it is cleared by writing '1' to it. These bits are also cleared automatically when the receiver is disabled.

There are two methods for buffer overflow notification, selected by the Immediate Buffer Overflow Notification bit in the Control A register (CTRLA.IBON):

When CTRLA.IBON=1, STATUS.BUFOVF is raised immediately upon buffer overflow. Software can then empty the receive FIFO by reading RxDATA, until the receiver complete interrupt flag (INTFLAG.RXC) is cleared.

When CTRLA.IBON=0, the buffer overflow condition is attending data through the receive FIFO. After the received data is read, STATUS.BUFOVF will be set along with INTFLAG.RXC.

Asynchronous Data Reception

The USART includes a clock recovery and data recovery unit for handling asynchronous data reception.

The clock recovery logic can synchronize the incoming asynchronous serial frames at the RxD pin to the internally generated baud-rate clock.

The data recovery logic samples and applies a low-pass filter to each incoming bit, thereby improving the noise immunity of the receiver.

Asynchronous Operational Range

The operational range of the asynchronous reception depends on the accuracy of the internal baud-rate clock, the rate of the incoming frames, and the frame size (in number of bits). In addition, the operational range of the receiver is depending on the difference between the received bit rate and the internally generated baud rate. If the baud rate of an external transmitter is too high or too low compared to the internally generated baud rate, the receiver will not be able to synchronize the frames to the start bit.

There are two possible sources for a mismatch in baud rate: First, the reference clock will always have some minor instability. Second, the baud-rate generator cannot always do an exact division of the reference clock frequency to get the baud rate desired. In this case, the BAUD register value must be set to give the lowest possible error, see *Clock Generation – Baud-Rate Generator* from Related Links.

Recommended maximum receiver baud-rate errors for various character sizes are shown in the table below.

Table 30-3. Asynchronous Receiver Error for 16-fold Oversampling

D (Data bits+Parity)	R_{SLOW} [%]	R_{FAST} [%]	Max. total error [%]	Recommended max. Rx error [%]
5	94.12	107.69	+5.88/-7.69	±2.5
6	94.92	106.67	+5.08/-6.67	±2.0
7	95.52	105.88	+4.48/-5.88	±2.0
8	96.00	105.26	+4.00/-5.26	±2.0
9	96.39	104.76	+3.61/-4.76	±1.5
10	96.70	104.35	+3.30/-4.35	±1.5

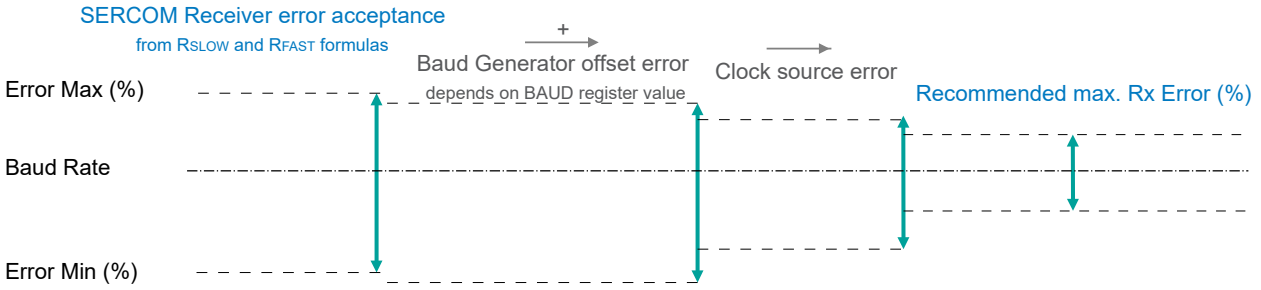
The following equations calculate the ratio of the incoming data rate and internal receiver baud rate:

$$R_{\mathrm{SLOW}} = \frac{16(D+1)}{16(D+1)+6} \quad , \qquad R_{\mathrm{FAST}} = \frac{16(D+2)}{16(D+1)+8}$$

- R_{SLOW} is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate
- R_{FAST} is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate
- D is the sum of character size and parity size (D = 5 to 10 bits)

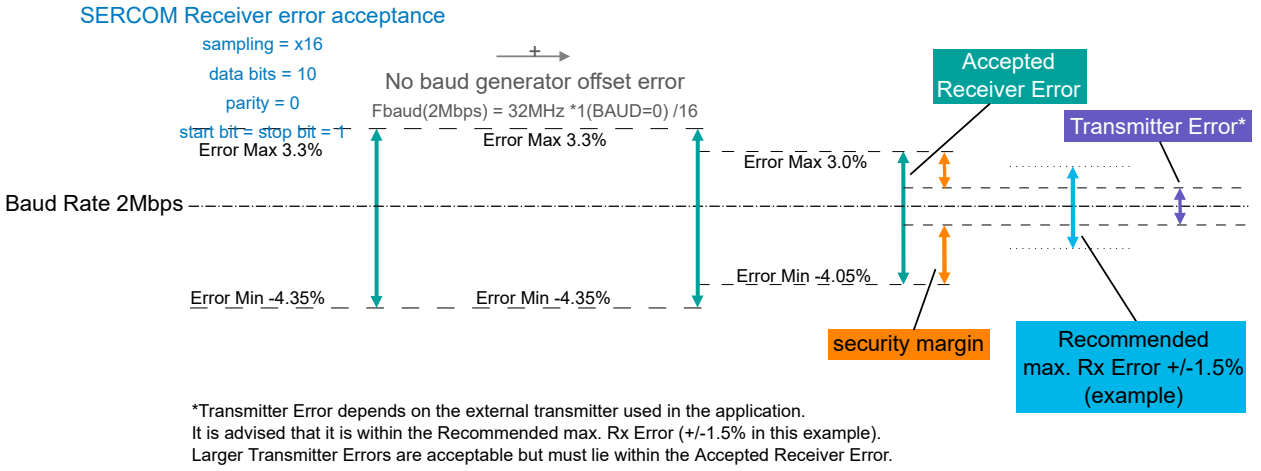
The recommended maximum Rx Error assumes that the receiver and transmitter equally divide the maximum total error. Its connection to the SERCOM Receiver error acceptance is depicted in this figure:

Figure 30-5. USART Rx Error Calculation



The recommendation values in the table above accommodate errors of the clock source and the baud generator. The following figure gives an example for a baud rate of 3Mbps:

Figure 30-6. USART Rx Error Calculation Example



*Transmitter Error depends on the external transmitter used in the application.
It is advised that it is within the Recommended max. Rx Error (+/-1.5% in this example).
Larger Transmitter Errors are acceptable but must lie within the Accepted Receiver Error.

Related Links

29.6.2.3. Clock Generation – Baud-Rate Generator

30.6.2.6.1 Disabling the Receiver

Writing '0' to the Receiver Enable bit in the CTRLB register (CTRLB.RXEN) will disable the receiver, flush the two-level receive buffer, and data from ongoing receptions will be lost.

30.6.2.6.2 Error Bits

The USART receiver has three error bits in the Status (STATUS) register: Frame Error (FERR), Buffer Overflow (BUFOVF), and Parity Error (PERR). Once an error happens, the corresponding error bit will be set until it is cleared by writing '1' to it. These bits are also cleared automatically when the receiver is disabled.

There are two methods for buffer overflow notification, selected by the Immediate Buffer Overflow Notification bit in the Control A register (CTRLA.IBON):

When CTRLA.IBON=1, STATUS.BUFOVF is raised immediately upon buffer overflow. Software can then empty the receive FIFO by reading RxDATA, until the Receiver Complete Interrupt flag (INTFLAG.RXC) is cleared.

When CTRLA.IBON=0, the Buffer Overflow condition is attending data through the receive FIFO, which will then set the INTFLAG.ERROR bit. After the received data is read, STATUS.BUFOVF (and INTFLAG.ERROR) will be set along with INTFLAG.RXC.

30.6.2.6.3 Asynchronous Data Reception

The USART includes a clock recovery and data recovery unit for handling asynchronous data reception.

The clock recovery logic can synchronize the incoming asynchronous serial frames at the RxD pin to the internally generated baud-rate clock.

The data recovery logic samples and applies a low-pass filter to each incoming bit, thereby improving the noise immunity of the receiver.

30.6.2.6.4 Asynchronous Operational Range

The operational range of the asynchronous reception depends on the accuracy of the internal baud-rate clock, the rate of the incoming frames, and the frame size (in number of bits). In addition, the operational range of the receiver is depending on the difference between the received bit rate and the internally generated baud rate. If the baud rate of an external transmitter is too high or too low compared to the internally generated baud rate, the receiver will not be able to synchronize the frames to the start bit.

There are two possible sources for a mismatch in baud rate: First, the reference clock will always have some minor instability. Second, the baud-rate generator cannot always do an exact division of the reference clock frequency to get the baud rate desired. In this case, the BAUD register value must be set to give the lowest possible error (see *Clock Generation – Baud-Rate Generator* from Related Links).

Recommended maximum receiver baud-rate errors for various character sizes are shown in the following table.

Table 30-4. Asynchronous Receiver Error for 16-fold Oversampling

D (Data bits+Parity)	R_{SLOW} [%]	R_{FAST} [%]	Max. total error [%]	Recommended max. Rx error [%]
5	94.12	107.69	+5.88/-7.69	±2.5
6	94.92	106.67	+5.08/-6.67	±2.0
7	95.52	105.88	+4.48/-5.88	±2.0
8	96.00	105.26	+4.00/-5.26	±2.0
9	96.39	104.76	+3.61/-4.76	±1.5
10	96.70	104.35	+3.30/-4.35	±1.5

The following equations calculate the ratio of the incoming data rate and internal receiver baud rate:

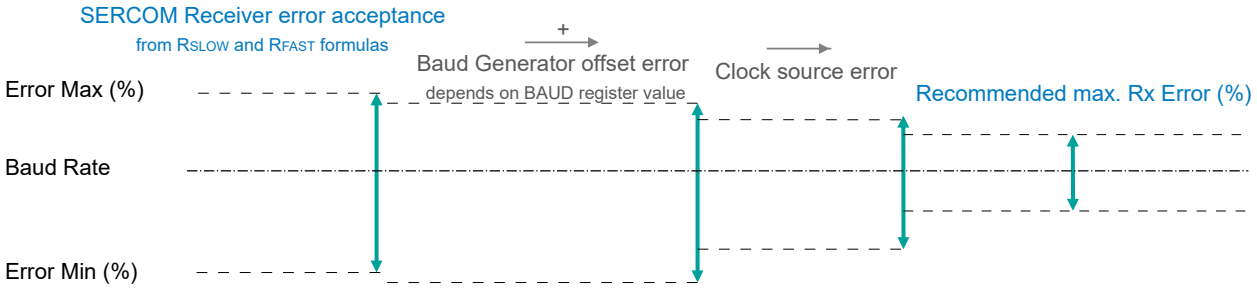
$$R_{\mathrm{SLOW}} = \frac{(D+1)S}{S-1+D\cdot S+S_F} \quad , \qquad R_{\mathrm{FAST}} = \frac{(D+2)S}{(D+1)S+S_M}$$

- R_{SLOW} is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate
- R_{FAST} is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate
- D is the sum of character size and parity size (D = 5 to 10 bits)
- S is the number of samples per bit (S = 16, 8 or 3)

- S_F is the first sample number used for majority voting (S_F = 7, 3 or 2) when CTRLA.SAMPA=0.
- S_M is the middle sample number used for majority voting (S_M = 8, 4 or 2) when CTRLA.SAMPA=0.

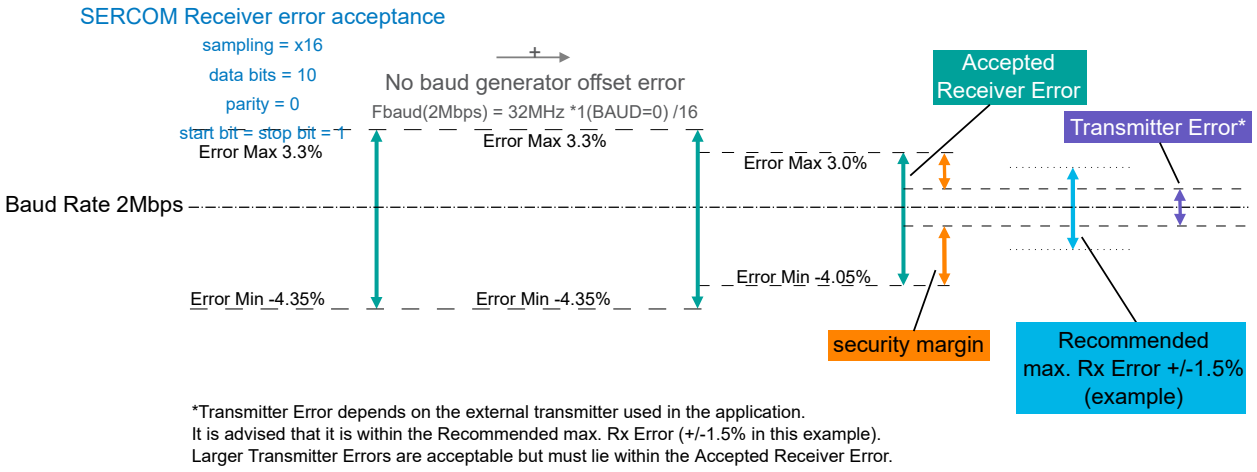
The recommended maximum Rx Error assumes that the receiver and transmitter equally divide the maximum total error. Its connection to the SERCOM Receiver error acceptance is depicted in this figure:

Figure 30-7. USART Rx Error Calculation



The recommendation values in the table above accommodate errors of the clock source and the baud generator. The following figure gives an example for a baud rate of 3 Mbps:

Figure 30-8. USART Rx Error Calculation Example



Related Links

29.6.2.3. Clock Generation – Baud-Rate Generator

30.6.3 Additional Features

30.6.3.1 Parity

Even or odd parity can be selected for error checking by writing 0x1 to the Frame Format bit field in the Control A register (CTRLA.FORM).

If *even parity* is selected (CTRLB.PMODE=0), the Parity bit of an outgoing frame is '1' if the data contains an odd number of bits that are '1', making the total number of '1' even.

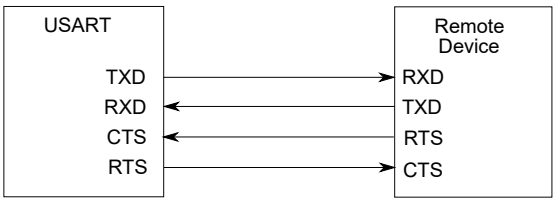
If *odd parity* is selected (CTRLB.PMODE=1), the Parity bit of an outgoing frame is '1' if the data contains an even number of bits that are '0', making the total number of '1' odd.

When parity checking is enabled, the parity checker calculates the parity of the data bits in incoming frames and compares the result with the Parity bit of the corresponding frame. If a parity error is detected, the Parity Error bit in the Status register (STATUS.PERR) is set.

30.6.3.2 Hardware Handshaking

The USART features an out-of-band hardware handshaking flow control mechanism, implemented by connecting the RTS and CTS pins with the remote device, as shown in the figure below.

Figure 30-9. Connection with a Remote Device for Hardware Handshaking

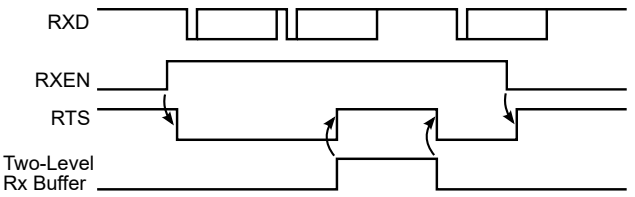


Hardware handshaking is only available in the following configuration:

- USART with internal clock (CTRLA.MODE=1),
- Asynchronous mode (CTRLA.CMODE=0), and
- Flow control pinout (CTRLA.TXPO=2).

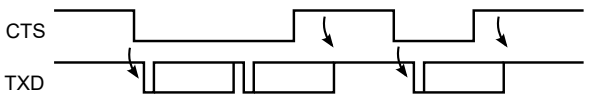
When the receiver is disabled or the receive FIFO is full, the receiver will drive the RTS pin high. This notifies the remote device to stop transfer after the ongoing transmission. Enabling and disabling the receiver by writing to CTRLB.RXEN will set/clear the RTS pin after a synchronization delay. When the receive FIFO goes full, RTS will be set immediately and the frame being received will be stored in the Shift register until the receive FIFO is no longer full.

Figure 30-10. Receiver Behavior when Operating with Hardware Handshaking



The current CTS Status is in the STATUS register (STATUS.CTS). Character transmission will start only if STATUS.CTS=0. When CTS is set, the transmitter will complete the ongoing transmission and stop transmitting.

Figure 30-11. Transmitter Behavior when Operating with Hardware Handshaking



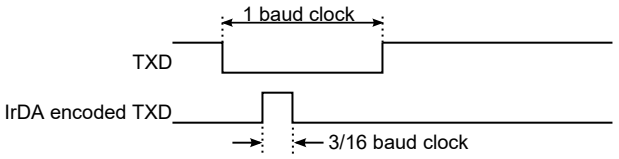
30.6.3.3 IrDA Modulation and Demodulation

Transmission and reception can be encoded IrDA compliant up to 115.2 kb/s. IrDA modulation and demodulation work in the following configuration:

- IrDA encoding enabled (CTRLB.ENC=1)
- Asynchronous mode (CTRLA.CMODE=0)
- 16x sample rate (CTRLA.SAMPR[0]=0)

During transmission, each low bit is transmitted as a high pulse. The pulse width is 3/16 of the baud rate period, as illustrated in the following figure.

Figure 30-12. IrDA Transmit Encoding



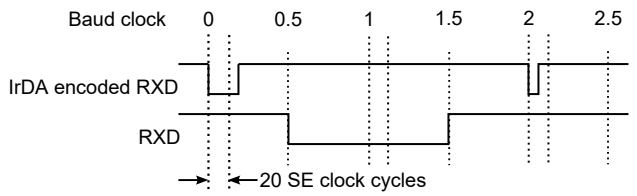
The reception decoder has two main functions:

- To synchronize the incoming data to the IrDA baud rate counter. Synchronization is performed at the start of each zero pulse.
- To decode incoming Rx data. If a pulse width meets the minimum length set by configuration (RXPL.RXPL), it is accepted. When the baud rate counter reaches its middle value (1/2 bit length), it is transferred to the receiver.

Note: The polarity of the transmitter and receiver are opposite: During transmission, a '0' bit is transmitted as a '1' pulse. During reception, an accepted '0' pulse is received as a '0' bit.

Example: The following figure illustrates reception where RXPL.RXPL is set to 19. This indicates that the pulse width must be at least 20 SE clock cycles. When using BAUD=0xE666 or 160 SE cycles per bit, this corresponds to 2/16 baud clock as minimum pulse width required. In this case the first bit is accepted as a '0', the second bit is a '1', and the third bit is also a '1'. A low pulse is rejected since it does not meet the minimum requirement of 2/16 baud clock.

Figure 30-13. IrDA Receive Decoding



30.6.3.4 Break Character Detection and Auto-Baud

Break character detection and auto-baud are available in this configuration:

- Auto-baud frame format (CTRLA.FORM = 0x04 or 0x05),
- Asynchronous mode (CTRLA.CMODE = 0),
- and 16x sample rate using fractional baud rate generation (CTRLA.SAMPR = 1).

The USART uses a break detection threshold of greater than 11 nominal bit times at the configured baud rate. At any time, if more than 11 consecutive dominant bits are detected on the bus, the USART detects a Break Field. When a break field has been detected, the Receive Break Interrupt Flag (INTFLAG.RXBRK) is set and the USART expects the sync field character to be 0x55. This field is used to update the actual baud rate in order to stay synchronized. If the received sync character is not 0x55, then the Inconsistent Sync Field error flag (STATUS.ISF) is set along with the Error Interrupt Flag (INTFLAG.ERROR), and the baud rate is unchanged.

After a break field is detected and the Start bit of the sync field is detected, a counter is started. The counter is then incremented for the next 8 bit times of the sync field. At the end of these 8 bit times, the counter is stopped. At this moment, the 13 Most Significant bits of the counter (value divided by 8) give the new clock divider (BAUD.BAUD), and the 3 Least Significant bits of this value (the remainder) give the new Fractional Part (BAUD.FP).

When the sync field has been received, the clock divider (BAUD.BAUD) and the Fractional Part (BAUD.FP) are updated after a synchronization delay. After the break and sync fields are received, multiple characters of data can be received.

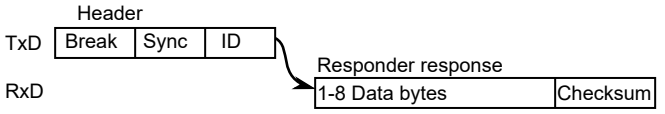
30.6.3.5 LIN Commander

LIN commander is available with the following configuration:

- LIN commander format (CTRLA.FORM = 0x02)
- Asynchronous mode (CTRLA.CMODE = 0)
- 16x sample rate using fractional baud rate generation (CTRLA.SAMPR = 1)

LIN frames start with a header transmitted by the commander. The header consists of the break, sync, and identifier fields. After the commander transmits the header, the addressed responder will respond with 1-8 bytes of data plus checksum.

Figure 30-14. LIN Frame Format



Using the LIN command field (CTRLB.LINCMD), the complete header can be automatically transmitted, or software can control transmission of the various header components.

When CTRLB.LINCMD=0x1, software controls transmission of the LIN header. In this case, software uses the following sequence.

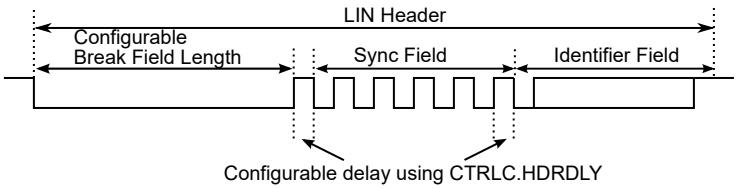
- CTRLB.LINCMD is written to 0x1.
- DATA register written to 0x00. This triggers transmission of the break field by hardware. Note that writing the DATA register with any other value will also result in the transmission of the break field by hardware.
- DATA register written to 0x55. The 0x55 value (sync) is transmitted.
- DATA register written to the identifier. The identifier is transmitted.

When CTRLB.LINCMD=0x2, hardware controls transmission of the LIN header. In this case, software uses the following sequence.

- CTRLB.LINCMD is written to 0x2.
- DATA register written to the identifier. This triggers transmission of the complete header by hardware. First the break field is transmitted. Next, the sync field is transmitted, and finally the identifier is transmitted.

In LIN commander mode, the length of the break field is programmable using the break length field (CTRLC.BRKLEN). When the LIN header command is used (CTRLB.LINCMD=0x2), the delay between the break and sync fields, in addition to the delay between the sync and ID fields are configurable using the header delay field (CTRLC.HDRDLY). When manual transmission is used (CTRLB.LINCMD=0x1), software controls the delay between break and sync.

Figure 30-15. LIN Header Generation



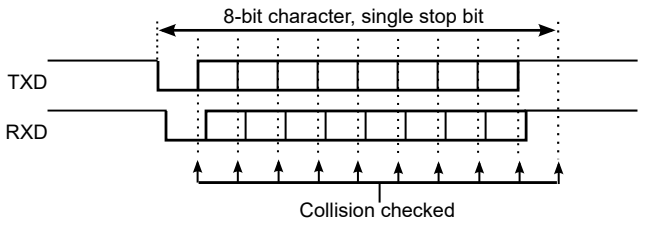
After header transmission is complete, the responder responds with 1-8 data bytes plus checksum.

30.6.3.6 Collision Detection

When the receiver and transmitter are connected either through pin configuration or externally, transmit collision can be detected after selecting the Collision Detection Enable bit in the CTRLB register (CTRLB.COLDEN=1). To detect collision, the receiver and transmitter must be enabled (CTRLB.RXEN=1 and CTRLB.TXEN=1).

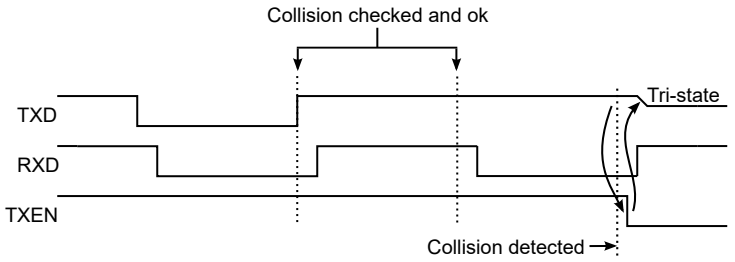
Collision detection is performed for each bit transmitted by comparing the received value with the transmit value, as shown in the figure below. While the transmitter is idle (no transmission in progress), characters can be received on RxD without triggering a collision.

Figure 30-16. Collision Checking



The next figure shows the conditions for a collision detection. In this case, the Start bit and the first Data bit are received with the same value as transmitted. The second received Data bit is found to be different than the transmitted bit at the detection point, which indicates a collision.

Figure 30-17. Collision Detected



When a collision is detected, the USART follows this sequence:

1. Abort the current transfer.
2. Flush the transmit buffer.
3. Disable transmitter (CTRLB.TXEN=0)
   - This is done after a synchronization delay. The CTRLB Synchronization Busy bit (SYNCBUSY.CTRLB) will be set until this is complete.
   - After disabling, the TxD pin will be tri-stated.
4. Set the Collision Detected bit (STATUS.COLL) along with the Error Interrupt Flag (INTFLAG.ERROR).

5. Set the Transmit Complete Interrupt Flag (INTFLAG.TXC), since the transmit buffer no longer contains data.

After a collision, software must manually enable the transmitter again before continuing, after assuring that the CTRLB Synchronization Busy bit (SYNCBUSY.CTRLB) is not set.

30.6.3.7 Loop-Back Mode

For Loop-Back mode, configure the Receive Data Pinout (CTRLA.RXPO) and Transmit Data Pinout (CTRLA.TXPO) to use the same data pins for transmit and receive. The loop-back is through the pad, so the signal is also available externally.

30.6.3.8 Start-of-Frame Detection

The USART start-of-frame detector can wake up the CPU when it detects a Start bit. In Standby Sleep mode, the internal fast start-up oscillator must be selected as the GCLK_SERCOMx_CORE source.

When a 1-to-0 transition is detected on RxD, the 8 MHz Internal Oscillator is powered up and the USART clock is enabled. After start-up, the rest of the data frame can be received, provided that the baud rate is slow enough in relation to the fast start-up internal oscillator start-up time. See *Electrical Characteristics* from Related Links for details. The start-up time of this oscillator varies with supply voltage and temperature.

The USART start-of-frame detection works both in Asynchronous and Synchronous modes. It is enabled by writing '1' to the Start of Frame Detection Enable bit in the Control B register (CTRLB.SFDE).

If the Receive Start Interrupt Enable bit in the Interrupt Enable Set register (INTENSET.RXS) is set, the Receive Start interrupt is generated immediately when a start is detected.

When using start-of-frame detection without the Receive Start interrupt, start detection will force the 8 MHz internal oscillator and USART clock active while the frame is being received. In this case, the CPU will not wake up until the receive complete interrupt is generated.

Related Links

43. Electrical Characteristics

30.6.3.9 Sample Adjustment

In asynchronous mode (CTRLA.CMODE = 0), three samples in the middle are used to determine the value based on majority voting. The three samples used for voting can be selected using the Sample Adjustment bit field in the Control A register (CTRLA.SAMPA). When CTRLA.SAMPA = 0, samples 7-8-9 are used for 16x oversampling, and samples 3-4-5 are used for 8x oversampling.

Note: In full asynchronous mode, the start of frame may not occur at the UART clock reference rising edge meaning the counter can start incrementing from 0 to 1 in less than one UART clock reference period. The counter will then continue to increment at each positive edge of the UART clock reference regardless of the incoming bits.

30.6.4 DMA, Interrupts and Events

30.6.4.1 DMA Operation

The USART generates the following DMA requests:

30.6.4.2 Interrupts

The USART has the following interrupt sources. These are asynchronous interrupts, and can wake-up the device from any Sleep mode:

- Data Register Empty (DRE)
- Receive Complete (RXC)
- Transmit Complete (TXC)

- Receive Start (RXS)
- Clear to Send Input Change (CTSIC)
- Received Break (RXBRK)
- Error (ERROR)

Each interrupt source has its own Interrupt flag. The Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG) will be set when the Interrupt condition is met. Each interrupt can be individually enabled by writing '1' to the corresponding bit in the Interrupt Enable Set register (INTENSET), and disabled by writing '1' to the corresponding bit in the Interrupt Enable Clear register (INTENCLR). The status of enabled interrupts can be read from either INTENSET or INTENCLR.

An interrupt request is generated when the Interrupt flag is set and if the corresponding interrupt is enabled. The interrupt request remains active until either the Interrupt flag is cleared, the interrupt is disabled, or the USART is reset. For details on clearing Interrupt flags, see *INTFLAG* from Related Links.

The value of INTFLAG indicates which interrupt is executed. Note that interrupts must be globally enabled for interrupt requests. See *Nested Vector Interrupt Controller (NVIC)* from Related Links.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)

30.8.8. INTFLAG

30.6.4.3 Events

Not applicable.

30.6.5 Sleep Mode Operation

The behavior in Sleep mode is depending on the clock source and the Run In Standby bit in the Control A register (CTRLA.RUNSTDBY):

- Internal clocking, CTRLA.RUNSTDBY=1: GCLK_SERCOMx_CORE can be enabled in all Sleep modes. Any interrupt can wake-up the device.
- External clocking, CTRLA.RUNSTDBY=1: The Receive Complete interrupt(s) can wake-up the device.
- Internal clocking, CTRLA.RUNSTDBY=0: Internal clock will be disabled, after any ongoing transfer was completed. The Receive Complete interrupt(s) can wake-up the device.
- External clocking, CTRLA.RUNSTDBY=0: External clock will be disconnected, after any ongoing transfer was completed. All reception will be dropped.

30.6.6 Synchronization

Due to asynchronicity between the main clock domain and the peripheral clock domains, some registers need to be synchronized when written or read.

The following bits are synchronized when written:

- Software Reset bit in the CTRLA register (CTRLA.SWRST)
- Enable bit in the CTRLA register (CTRLA.ENABLE)
- Receiver Enable bit in the CTRLB register (CTRLB.RXEN)
- Transmitter Enable bit in the Control B register (CTRLB.TXEN)

Note: CTRLB.RXEN is write-synchronized somewhat differently. See also 30.8.2. CTRLB for details.

Required write synchronization is denoted by the "Write-Synchronized" property in the register description.

30.7 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0	RUNSTDBY			MODE[2:0]			ENABLE	SWRST
		15:8	SAMPR[2:0]							IBON
		23:16	SAMPA[1:0]		RXPO[1:0]				TXPO[1:0]	
		31:24		DORD	CPOL	CMODE	FORM[3:0]			
0x04	CTRLB	7:0		SBMODE				CHSIZE[2:0]		
		15:8			PMODE			ENC		COLDEN
		23:16							RXEN	TXEN
		31:24								
0x08	CTRLC	7:0								
		15:8								
		23:16								
		31:24								
0x0C	BAUD	7:0	BAUD[7:0]							
		15:8	BAUD[15:8]							
0x0E	RXPL	7:0	RXPL[7:0]							
0x0F ... 0x13	Reserved									
0x14	INTENCLR	7:0	ERROR		RXBRK	CTSIC		RXC	TXC	DRE
0x15	Reserved									
0x16	INTENSET	7:0	ERROR		RXBRK	CTSIC		RXC	TXC	DRE
0x17	Reserved									
0x18	INTFLAG	7:0	ERROR		RXBRK	CTSIC		RXC	TXC	DRE
0x19	Reserved									
0x1A	STATUS	7:0		TXE	COLL	ISF	CTS	BUFOVF	FERR	PERR
		15:8								
0x1C	SYNCBUSY	7:0						CTRLB	ENABLE	SWRST
		15:8								
		23:16								
		31:24								
0x20 ... 0x27	Reserved									
0x28	DATA	7:0	DATA[7:0]							
		15:8	DATA[15:8]							
		23:16	DATA[23:16]							
		31:24	DATA[31:24]							
0x2C ... 0x2F	Reserved									
0x30	DBGCTRL	7:0								DBGSTOP

30.8 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register, and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers require synchronization when read and/or written. Synchronization is denoted by the “Read-Synchronized” and/or “Write-Synchronized” property in each individual register description.

Optional write protection by the Peripheral Access Controller (PAC) is denoted by the "PAC Write Protection" property in each individual register description.

Some registers are enable-protected, meaning they can only be written when the module is disabled. Enable-protection is denoted by the “Enable-Protected” property in each individual register description.

30.8.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected, Write-Synchronized

Bit	31	30	29	28	27	26	25	24
		DORD	CPOL	CMODE	FORM[3:0]			
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	SAMPA[1:0]		RXPO[1:0]				TXPO[1:0]	
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

Bit	15	14	13	12	11	10	9	8
	SAMPR[2:0]							IBON
Access	R/W	R/W	R/W					R/W
Reset	0	0	0					0

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY			MODE[2:0]			ENABLE	SWRST
Access	R/W			R/W	R/W	R/W	R/W	R/W
Reset	0			0	0	0	0	0

Bit 30 – DORD Data Order
This bit selects the data order when a character is shifted out from the Data register.
This bit is not synchronized.

Value	Description
0	MSB is transmitted first.
1	LSB is transmitted first.

Bit 29 – CPOL Clock Polarity
This bit selects the relationship between data output change and data input sampling in synchronous mode.
This bit is not synchronized.

CPOL	TxD Change	RxD Sample
0x0	Rising XCK edge	Falling XCK edge
0x1	Falling XCK edge	Rising XCK edge

Bit 28 – CMODE Communication Mode
This bit selects asynchronous or synchronous communication.
This bit is not synchronized.

Value	Description
0	Asynchronous communication.
1	Synchronous communication.

Bits 27:24 – FORM[3:0] Frame Format
These bits define the frame format.
These bits are not synchronized.

FORM[3:0]	Description
0x0	USART frame
0x1	USART frame with parity
0x4	Auto-baud (LIN Responder) - break detection and auto-baud.
0x5	Auto-baud - break detection and auto-baud with parity

Bits 23:22 – SAMPA[1:0] Sample Adjustment
These bits define the sample adjustment.
These bits are not synchronized.

SAMPA[1:0]	16x Over-sampling (CTRLA.SAMPR=0 or 1)	8x Over-sampling (CTRLA.SAMPR=2 or 3)
0x0	7-8-9	3-4-5
0x1	9-10-11	4-5-6
0x2	11-12-13	5-6-7
0x3	13-14-15	6-7-8

Bits 21:20 – RXPO[1:0] Receive Data Pinout
These bits define the receive data (RxD) pin configuration.
These bits are not synchronized.

RXPO[1:0]	Name	Description
0x0	PAD[0]	SERCOM PAD[0] is used for data reception
0x1	PAD[1]	SERCOM PAD[1] is used for data reception
0x2	PAD[2]	SERCOM PAD[2] is used for data reception
0x3	PAD[3]	SERCOM PAD[3] is used for data reception

Bits 17:16 – TXPO[1:0] Transmit Data Pinout
These bits define the transmit data (TxD) and XCK pin configurations.
This bit is not synchronized.

Bits 15:13 – SAMPR[2:0] Sample Rate
These bits select the sample rate.
These bits are not synchronized.

SAMPR[2:0]	Description
0x0	16x over-sampling using arithmetic baud rate generation.
0x1	16x over-sampling using fractional baud rate generation.
0x2	8x over-sampling using arithmetic baud rate generation.
0x3	8x over-sampling using fractional baud rate generation.
0x4	3x over-sampling using arithmetic baud rate generation.
0x5-0x7	Reserved

Bit 8 – IBON Immediate Buffer Overflow Notification
This bit controls when the buffer overflow status bit (STATUS.BUFOVF) is asserted when a buffer overflow occurs.
This bit is not synchronized.

Value	Description
0	STATUS.BUFOVF is asserted when it occurs in the data stream.
1	STATUS.BUFOVF is asserted immediately upon buffer overflow.

Bit 7 – RUNSTDBY Run In Standby
This bit defines the functionality in standby sleep mode.
This bit is not synchronized.

RUNSTDBY	External Clock	Internal Clock
0x0	External clock is disconnected when ongoing transfer is finished. All reception is dropped.	Generic clock is disabled when ongoing transfer is finished. The device will not wake up on Transfer Complete interrupt unless the appropriate ONDEMAND bits are set in the clocking chain.
0x1	Wake on Receive Complete interrupt.	Generic clock is enabled in all sleep modes. Any interrupt can wake up the device.

Bits 4:2 – MODE[2:0] Operating Mode

These bits select the USART serial communication interface of the SERCOM.
These bits are not synchronized.

Value	Description
0x0	USART with external clock
0x1	USART with internal clock

Bit 1 – ENABLE Enable

Due to synchronization, there is delay from writing CTRLA.ENABLE until the peripheral is enabled/disabled. The value written to CTRLA.ENABLE will read back immediately and the Enable Synchronization Busy bit in the Synchronization Busy register (SYNCBUSY.ENABLE) will be set. SYNCBUSY.ENABLE is cleared when the operation is complete.
This bit is not enable-protected.

Value	Description
0	The peripheral is disabled or being disabled.
1	The peripheral is enabled or being enabled.

Bit 0 – SWRST Software Reset

Writing '0' to this bit has no effect.
Writing '1' to this bit resets all registers in the SERCOM, except DBGCTRL, to their initial state, and the SERCOM will be disabled.
Writing '1' to CTRLA.SWRST will always take precedence, meaning that all other writes in the same write-operation will be discarded. Any register write access during the ongoing reset will result in an APB error. Reading any register will return the reset value of the register.
Due to synchronization, there is a delay from writing CTRLA.SWRST until the reset is complete. CTRLA.SWRST and SYNCBUSY.SWRST will both be cleared when the reset is complete.
This bit is not enable-protected.
Note: During a SWRST, access to registers/bits without SWRST are disallowed until SYNCBUSY.SWRST cleared by hardware.

Value	Description
0	There is no reset operation ongoing.
1	The reset operation is ongoing.

30.8.2 Control B

Name: CTRLB
Offset: 0x04
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected, Write-Synchronized

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
							RXEN	TXEN
Access							R/W	R/W
Reset							0	0

Bit	15	14	13	12	11	10	9	8
			PMODE			ENC		COLDEN
Access			R/W			R/W		R/W
Reset			0			0		0

Bit	7	6	5	4	3	2	1	0
		SBMODE				CHSIZE[2:0]		
Access		R/W				R/W	R/W	R/W
Reset		0				0	0	0

Bit 17 – RXEN Receiver Enable

Writing '0' to this bit will disable the USART receiver. Disabling the receiver will flush the receive buffer and clear the FERR, PERR and BUFOVF bits in the STATUS register.
Writing '1' to CTRLB.RXEN when the USART is disabled will set CTRLB.RXEN immediately. When the USART is enabled, CTRLB.RXEN will be cleared, and SYNCBUSY.CTRLB will be set and remain set until the receiver is enabled. When the receiver is enabled, CTRLB.RXEN will read back as '1'.
Writing '1' to CTRLB.RXEN when the USART is enabled will set SYNCBUSY.CTRLB, which will remain set until the receiver is enabled, and CTRLB.RXEN will read back as '1'.
This bit is not enable-protected.

Value	Description
0	The receiver is disabled or being enabled.
1	The receiver is enabled or will be enabled when the USART is enabled.

Bit 16 – TXEN Transmitter Enable

Writing '0' to this bit will disable the USART transmitter. Disabling the transmitter will not become effective until ongoing and pending transmissions are completed.
Writing '1' to CTRLB.TXEN when the USART is disabled will set CTRLB.TXEN immediately. When the USART is enabled, CTRLB.TXEN will be cleared, and SYNCBUSY.CTRLB will be set and remain set until the transmitter is enabled. When the transmitter is enabled, CTRLB.TXEN will read back as '1'.
Writing '1' to CTRLB.TXEN when the USART is enabled will set SYNCBUSY.CTRLB, which will remain set until the transmitter is enabled, and CTRLB.TXEN will read back as '1'.
This bit is not enable-protected.

Value	Description
0	The transmitter is disabled or being enabled.
1	The transmitter is enabled or will be enabled when the USART is enabled.

Bit 13 – PMODE Parity Mode

This bit selects the type of parity used when parity is enabled (CTRLA.FORM is '1'). The transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The receiver will generate a parity value for the incoming data and parity bit, compare it to the parity mode and, if a mismatch is detected, STATUS.PERR will be set.
This bit is not synchronized.

Value	Description
0	Even parity.
1	Odd parity.

Bit 10 – ENC Encoding Format

This bit selects the data encoding format.
This bit is not synchronized.

Value	Description
0	Data is not encoded.
1	Data is IrDA encoded.

Bit 8 – COLDEN Collision Detection Enable

This bit enables collision detection.
This bit is not synchronized.

Value	Description
0	Collision detection is not enabled.
1	Collision detection is enabled.

Bit 6 – SBMODE Stop Bit Mode

This bit selects the number of stop bits transmitted.
This bit is not synchronized.

Value	Description
0	One stop bit.
1	Two stop bits.

Bits 2:0 – CHSIZE[2:0] Character Size

These bits select the number of bits in a character.
These bits are not synchronized.

CHSIZE[2:0]	Description
0x0	8 bits
0x1	9 bits
0x2-0x4	Reserved
0x5	5 bits
0x6	6 bits
0x7	7 bits

30.8.3 Control C

Name: CTRLC
Offset: 0x08
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

30.8.4 Baud

Name: BAUD
Offset: 0x0C
Reset: 0x0000
Property: Enable-Protected, PAC Write-Protection

Bit	15	14	13	12	11	10	9	8
	BAUD[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	BAUD[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 15:0 – BAUD[15:0] Baud Value

Arithmetic Baud Rate Generation (`CTRLA.SAMPR[0]=0`):
These bits control the clock generation, as described in the SERCOM Baud Rate section.
If Fractional Baud Rate Generation (`CTRLA.SAMPR[0]=1 or =3`) bit positions 15 to 13 are replaced by FP[2:0] Fractional Part:

- **Bits 15:13 - FP[2:0]: Fractional Part**
  These bits control the clock generation, as described in the *SERCOM Clock Generation – Baud-Rate Generator* section.
- **Bits 12:0 - BAUD[12:0]: Baud Value**
  These bits control the clock generation, as described in the *SERCOM Clock Generation – Baud-Rate Generator* section.

30.8.5 Receive Pulse Length Register

Name: RXPL
Offset: 0x0E
Reset: 0x00
Property: Enable-Protected, PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
	RXPL[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – RXPL[7:0] Receive Pulse Length
When the encoding format is set to IrDA (CTRLB.ENC=1), these bits control the minimum pulse length that is required for a pulse to be accepted by the IrDA receiver with regards to the serial engine clock period SE_{per}.
$PULSE \geq (\mathrm{RXPL} + 1) \cdot SE_{per}$

30.8.6 Interrupt Enable Clear

Name: INTENCLR
Offset: 0x14
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to disable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Set register (INTENSET).

Bit	7	6	5	4	3	2	1	0
	ERROR		RXBRK	CTSIC		RXC	TXC	DRE
Access	R/W		R/W	R/W		R/W	R/W	R/W
Reset	0		0	0		0	0	0

Bit 7 – ERROR Error Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Error Interrupt Enable bit, which disables the Error interrupt.

Value	Description
0	Error interrupt is disabled.
1	Error interrupt is enabled.

Bit 5 – RXBRK Receive Break Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Receive Break Interrupt Enable bit, which disables the Receive Break interrupt.

Value	Description
0	Receive Break interrupt is disabled.
1	Receive Break interrupt is enabled.

Bit 4 – CTSIC Clear to Send Input Change Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Clear To Send Input Change Interrupt Enable bit, which disables the Clear To Send Input Change interrupt.

Value	Description
0	Clear To Send Input Change interrupt is disabled.
1	Clear To Send Input Change interrupt is enabled.

Bit 2 – RXC Receive Complete Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Receive Complete Interrupt Enable bit, which disables the Receive Complete interrupt.

Value	Description
0	Receive Complete interrupt is disabled.
1	Receive Complete interrupt is enabled.

Bit 1 – TXC Transmit Complete Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Transmit Complete Interrupt Enable bit, which disables the Receive Complete interrupt.

Value	Description
0	Transmit Complete interrupt is disabled.
1	Transmit Complete interrupt is enabled.

Bit 0 – DRE Data Register Empty Interrupt Enable

Writing '0' to this bit has no effect.

Writing '1' to this bit will clear the Data Register Empty Interrupt Enable bit, which disables the Data Register Empty interrupt.

Value	Description
0	Data Register Empty interrupt is disabled.
1	Data Register Empty interrupt is enabled.

30.8.7 Interrupt Enable Set

Name: INTENSET
Offset: 0x16
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to disable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Clear register (INTENCLR).

Bit	7	6	5	4	3	2	1	0
	ERROR		RXBRK	CTSIC		RXC	TXC	DRE
Access	R/W		R/W	R/W		R/W	R/W	R/W
Reset	0		0	0		0	0	0

Bit 7 – ERROR Error Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Error Interrupt Enable bit, which enables the Error interrupt.

Value	Description
0	Error interrupt is disabled.
1	Error interrupt is enabled.

Bit 5 – RXBRK Receive Break Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Receive Break Interrupt Enable bit, which enables the Receive Break interrupt.

Value	Description
0	Receive Break interrupt is disabled.
1	Receive Break interrupt is enabled.

Bit 4 – CTSIC Clear to Send Input Change Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Clear To Send Input Change Interrupt Enable bit, which enables the Clear To Send Input Change interrupt.

Value	Description
0	Clear To Send Input Change interrupt is disabled.
1	Clear To Send Input Change interrupt is enabled.

Bit 2 – RXC Receive Complete Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Receive Complete Interrupt Enable bit, which enables the Receive Complete interrupt.

Value	Description
0	Receive Complete interrupt is disabled.
1	Receive Complete interrupt is enabled.

Bit 1 – TXC Transmit Complete Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Transmit Complete Interrupt Enable bit, which enables the Transmit Complete interrupt.

Value	Description
0	Transmit Complete interrupt is disabled.
1	Transmit Complete interrupt is enabled.

Bit 0 – DRE Data Register Empty Interrupt Enable

Writing '0' to this bit has no effect.

Writing '1' to this bit will set the Data Register Empty Interrupt Enable bit, which enables the Data Register Empty interrupt.

Value	Description
0	Data Register Empty interrupt is disabled.
1	Data Register Empty interrupt is enabled.

30.8.8 Interrupt Flag Status and Clear

Name: INTFLAG
Offset: 0x18
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	ERROR		RXBRK	CTSIC		RXC	TXC	DRE
Access	R/W		R/W	R/W		R	R/W	R
Reset	0		0	0		0	0	0

Bit 7 – ERROR Error
This flag is cleared by writing '1' to it.
This bit is set when any error is detected. Errors that will set this flag have corresponding status flags in the STATUS register. Errors that will set this flag are COLL, ISF, BUFOVF, FERR, and PERR.Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the flag.

Bit 5 – RXBRK Receive Break
This flag is cleared by writing '1' to it.
This flag is set when auto-baud is enabled (CTRLA.FORM) and a break character is received.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the flag.

Bit 4 – CTSIC Clear to Send Input Change
This flag is cleared by writing a '1' to it.
This flag is set when a change is detected on the CTS pin.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the flag.

Bit 2 – RXC Receive Complete
This flag is cleared by reading the Data register (DATA) or by disabling the receiver.
This flag is set when there are unread data in DATA.
Writing '0' to this bit has no effect.
Writing '1' to this bit has no effect.

Bit 1 – TXC Transmit Complete
This flag is cleared by writing '1' to it or by writing new data to DATA.
This flag is set when the entire frame in the Transmit Shift register has been shifted out and there are no new data in DATA.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the flag.

Bit 0 – DRE Data Register Empty
This flag is cleared by writing new data to DATA.
This flag is set when DATA is empty and ready to be written.
Writing '0' to this bit has no effect.
Writing '1' to this bit has no effect.

30.8.9 Status

Name: STATUS
Offset: 0x1A
Reset: 0x0000
Property: -

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
		TXE	COLL	ISF	CTS	BUFOVF	FERR	PERR
Access		R/W	R/W	R/W	R	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit 6 – TXE Transmitter Empty
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear it.

Bit 5 – COLL Collision Detected
This bit is cleared by writing '1' to the bit or by disabling the receiver.
This bit is set when collision detection is enabled (CTRLB.COLDEN) and a collision is detected.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear it.

Bit 4 – ISF Inconsistent Sync Field
This bit is cleared by writing '1' to the bit or by disabling the receiver.
This bit is set when the frame format is set to auto-baud (CTRLA.FORM) and a sync field not equal to 0x55 is received.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear it.

Bit 3 – CTS Clear to Send
This bit indicates the current level of the CTS pin when flow control is enabled (CTRLA.TXPO).
Writing '0' to this bit has no effect.
Writing '1' to this bit has no effect.

Bit 2 – BUFOVF Buffer Overflow
Reading this bit before reading the Data register will indicate the error status of the next character to be read.
This bit is cleared by writing '1' to the bit or by disabling the receiver.
This bit is set when a buffer overflow condition is detected. A buffer overflow occurs when the receive buffer is full, there is a new character waiting in the receive shift register and a new start bit is detected.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear it.

Bit 1 – FERR Frame Error
Reading this bit before reading the Data register will indicate the error status of the next character to be read.
This bit is cleared by writing '1' to the bit or by disabling the receiver.

This bit is set if the received character had a frame error, i.e., when the first stop bit is zero.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear it.

Bit 0 – PERR Parity Error

Reading this bit before reading the Data register will indicate the error status of the next character to be read.
This bit is cleared by writing '1' to the bit or by disabling the receiver.
This bit is set if parity checking is enabled (CTRLA.FORM is 0x1, 0x5) and a parity error is detected.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear it.

30.8.10 Synchronization Busy

Name: SYNCBUSY
Offset: 0x1C
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
						CTRLB	ENABLE	SWRST
Access						R	R	R
Reset						0	0	0

Bit 2 – CTRLB CTRLB Synchronization Busy
Writing to the CTRLB register when the SERCOM is enabled requires synchronization. When writing to CTRLB the SYNCBUSY.CTRLB bit will be set until synchronization is complete. If CTRLB is written while SYNCBUSY.CTRLB is asserted, an APB error will be generated.

Value	Description
0	CTRLB synchronization is not busy.
1	CTRLB synchronization is busy.

Bit 1 – ENABLE SERCOM Enable Synchronization Busy
Enabling and disabling the SERCOM (CTRLA.ENABLE) requires synchronization. When written, the SYNCBUSY.ENABLE bit will be set until synchronization is complete.

Value	Description
0	Enable synchronization is not busy.
1	Enable synchronization is busy.

Bit 0 – SWRST Software Reset Synchronization Busy
Resetting the SERCOM (CTRLA.SWRST) requires synchronization. When written, the SYNCBUSY.SWRST bit will be set until synchronization is complete.
Note: During a SWRST, access to registers/bits without SWRST are disallowed until SYNCBUSY.SWRST cleared by hardware.

Value	Description
0	SWRST synchronization is not busy.
1	SWRST synchronization is busy.

30.8.11 Data

Name: DATA
Offset: 0x28
Reset: 0x0000
Property: -

Bit	31	30	29	28	27	26	25	24
	DATA[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	DATA[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	DATA[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DATA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – DATA[31:0] Data

Reading these bits will return the contents of the Receive Data register. The register must be read only when the Receive Complete Interrupt Flag bit in the Interrupt Flag Status and Clear register (INTFLAG.RXC) is set. The status bits in STATUS must be read before reading the DATA value in order to get any corresponding error.

Writing these bits will write the Transmit Data register. This register must be written only when the Data Register Empty Interrupt Flag bit in the Interrupt Flag Status and Clear register (INTFLAG.DRE) is set.

Reads and writes are 32-bit or CTLB.CHSIZE based on the CTRLC.DATA32B setting.

30.8.12 Debug Control

Name: DBGCTRL
Offset: 0x30
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
								DBGSTOP
Access								R/W
Reset								0

Bit 0 – DBGSTOP Debug Stop Mode
This bit controls the baud-rate generator functionality when the CPU is halted by an external debugger.

Value	Description
0	The baud-rate generator continues normal operation when the CPU is halted by an external debugger.
1	The baud-rate generator is halted when the CPU is halted by an external debugger.

31. SERCOM Serial Peripheral Interface (SERCOM SPI)

31.1 Overview

The Serial Peripheral Interface (SPI) is one of the available modes in the Serial Communication Interface (SERCOM).

The SPI uses the SERCOM transmitter and receiver configured as shown in the Block Diagram (see *Full-Duplex SPI Host Client Interconnection* in the *Block Diagram* from Related Links). Each side, host and client, depicts a separate SPI containing a Shift register, a transmit buffer and a two-level receive buffer. In addition, the SPI host uses the SERCOM baud-rate generator, while the SPI Client can use the SERCOM address match logic. Labels in capital letters are synchronous to PBx_CLK and accessible by the CPU, while labels in lowercase letters are synchronous to the SCK clock.

Note: Traditional Serial Peripheral Interface (SPI) documentation uses the terminology “Master” and “Slave”. The equivalent Microchip terminology used in this document is “Host” and “Client”, respectively.

Related Links

31.3. Block Diagram

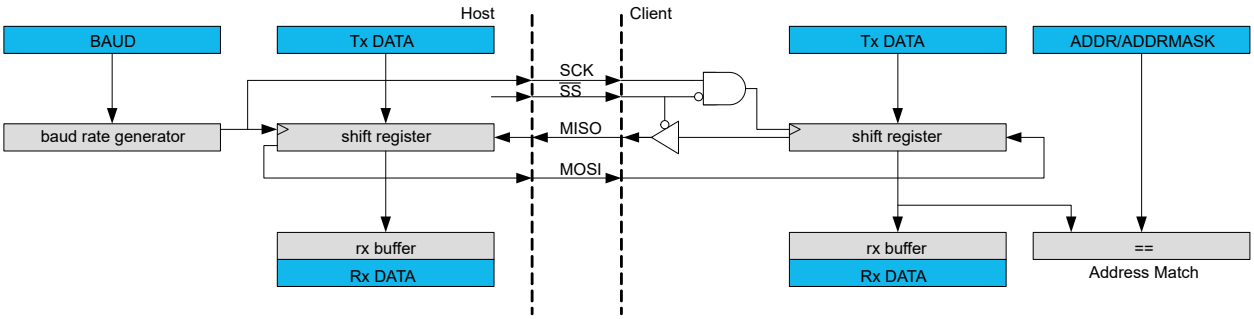
31.2 Features

SERCOM SPI includes the following features:

- Full-duplex, four-wire interface (MISO, MOSI, SCK, $\overline{SS}$)
- One-level transmit buffer, two-level receive buffer
- Supports all four SPI modes of operation
- Single data direction operation allows alternate function on MISO or MOSI pin
- Selectable LSB- or MSB-first data transfer
- Can be used with DMA
- Host operation:
  - Serial clock speed up to half the system clock
  - 8-bit clock generator
  - Hardware controlled $\overline{SS}$
- Client operation:
  - Serial clock speed up to half the system clock
  - Optional 8-bit address match operation
  - Operation in all sleep modes
  - Wake on $\overline{SS}$ transition

31.3 Block Diagram

Figure 31-1. Full-Duplex SPI Host Client Interconnection



31.4 Signal Description

Table 31-1. SERCOM SPI Signals

Signal Name	Type	Description
PAD[3:0]	Digital I/O	General SERCOM pins

One signal can be mapped to one of several pins.

31.5 Product Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

31.5.1 I/O Lines

In order to use the SERCOM's I/O lines, the I/O pins must be configured using the System Configuration registers or PPS registers.

When the SERCOM is configured for SPI operation, the SERCOM controls the direction and value of the I/O pins according to the following table. Both PORT Control bits PINCFGn.PULLEN and PINCFGn.DRVSTR are still effective. If the receiver is disabled, the data input pin can be used for other purposes. In Host mode, the Client Select line ($\overline{SS}$) is hardware controlled when the Host Client Select Enable bit in the Control B register (CTRLB.MSSEN) is '1'.

Table 31-2. SPI Pin Configuration

Pin	Host SPI	Client SPI
MOSI	Output	Input
MISO	Input	Output
SCK	Output	Input

The combined configuration of PORT, the Data In Pinout and the Data Out Pinout bit groups in the Control A register (CTRLA.DIPO and CTRLA.DOPO) define the physical position of the SPI signals in the table above.

31.5.2 Power Management

This peripheral can continue to operate in any Sleep mode where its source clock is running. The interrupts can wake-up the device from Sleep modes.

31.5.3 Clocks

A generic clock (GCLK_SERCOMx_CORE) is required to clock the SPI. This clock must be configured and enabled in the Clock and Reset Unit (CRU) and Configuration (CFG.CFGPCLKGEN1) registers before using the SPI.

This generic clock is asynchronous to the bus clock (PBx_CLK). Therefore, writes to certain registers will require synchronization to the clock domains.

31.5.4 DMA

The DMA request lines are connected to the DMA Controller (DMAC). To use DMA requests with this peripheral, the DMAC must be configured first (see *Direct Memory Access Controller (DMAC)* from Related Links).

Related Links

22. Direct Memory Access Controller (DMAC)

31.5.5 Interrupts

The interrupt request line is connected to the Interrupt Controller. In order to use interrupt requests of this peripheral, the Interrupt Controller (NVIC) must be configured first. See *Nested Vector Interrupt Controller (NVIC)* from Related Links.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)

31.5.6 Events

Not applicable.

31.5.7 Debug Operation

When the CPU is halted in Debug mode, this peripheral will continue normal operation. If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during debugging. This peripheral can be forced to halt operation during debugging - refer to the Debug Control (DBGCTRL) register for details.

Related Links

31.8.11. DBGCTRL

31.5.8 Register Access Protection

Registers with write access can be write-protected optionally by the Peripheral Access Controller (PAC).

PAC write protection is not available for the following registers:

- Interrupt Flag Clear and Status register (INTFLAG)
- Status register (STATUS)
- Data register (DATA)

Optional PAC write protection is denoted by the "PAC Write-Protection" property in each individual register description.

Write-protection does not apply to accesses through an external debugger.

31.5.9 Analog Connections

Not applicable.

31.6 Functional Description

31.6.1 Principle of Operation

The SPI is a high-speed synchronous data transfer interface. It allows high-speed communication between the device and peripheral devices.

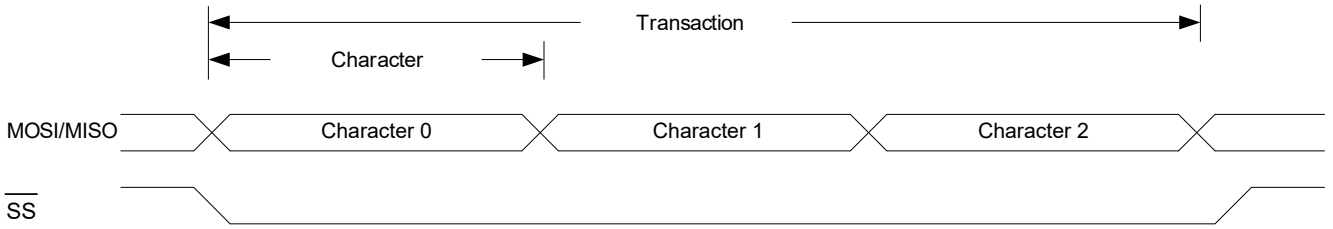
The SPI can operate as Host or Client. As Host, the SPI initiates and controls all data transactions. The SPI is single buffered for transmitting and double buffered for receiving.

When transmitting data, the Data register can be loaded with the next character to be transmitted during the current transmission.

When receiving, the data is transferred to the two-level receive buffer, and the receiver is ready for a new character.

The SPI transaction format is shown in SPI Transaction Format. Each transaction can contain one or more characters. The character size is configurable, and can be either 8 or 9 bits.

Figure 31-2. SPI Transaction Format



The SPI Host must pull the SPI select line ($\overline{SS}$) of the desired Client low to initiate a transaction. The Host and Client prepare data to send via their respective Shift registers, and the Host generates the serial clock on the SCK line.

Data are always shifted from Host to Client on the Host Output Client Input line (MOSI); data is shifted from Client to Host on the Host Input Client Output line (MISO).

Each time character is shifted out from the Host, a character will be shifted out from the Client simultaneously. To signal the end of a transaction, the Host will pull the $\overline{SS}$ line high

31.6.2 Basic Operation

31.6.2.1 Initialization

The following registers are enable-protected, meaning that they can only be written when the SPI is disabled (CTRL.ENABLE=0):

- Control A register (CTRLA), except Enable (CTRLA.ENABLE) and Software Reset (CTRLA.SWRST)
- Control B register (CTRLB), except Receiver Enable (CTRLB.RXEN)
- Baud register (BAUD)
- Address register (ADDR)

When the SPI is enabled or is being enabled (CTRLA.ENABLE=1), any writing to these registers will be discarded.

When the SPI is being disabled, writing to these registers will be completed after the disabling.

Enable-protection is denoted by the Enable-Protection property in the register description.

Initialize the SPI by following these steps:

1. Select SPI mode in host/client operation in the Operating Mode bit group in the CTRLA register (CTRLA.MODE= 0x2 or 0x3).
2. Select Transfer mode for the Clock Polarity bit and the Clock Phase bit in the CTRLA register (CTRLA.CPOL and CTRLA.CPHA) if desired.
3. Select the Frame Format value in the CTRLA register (CTRLA.FORM).
4. Configure the Data In Pinout field in the Control A register (CTRLA.DIPO) for SERCOM pads of the receiver.
5. Configure the Data Out Pinout bit group in the Control A register (CTRLA.DOPO) for SERCOM pads of the transmitter.

6. Select the Character Size value in the CTRLB register (CTRLB.CHSIZE).
7. Write the Data Order bit in the CTRLA register (CTRLA.DORD) for data direction.
8. If the SPI is used in Host mode:
   a. Select the desired baud rate by writing to the Baud register (BAUD).
   b. If Hardware $\overline{SS}$ control is required, write '1' to the Host SPI Select Enable bit in CTRLB register (CTRLB.MSSEN).
9. Enable the receiver by writing the Receiver Enable bit in the CTRLB register (CTRLB.RXEN=1).

31.6.2.2 Enabling, Disabling, and Resetting

This peripheral is enabled by writing '1' to the Enable bit in the Control A register (CTRLA.ENABLE), and disabled by writing '0' to it.

Writing '1' to the Software Reset bit in the Control A register (CTRLA.SWRST) will reset all registers of this peripheral to their initial states, except the DBGCTRL register, and the peripheral is disabled.

Refer to the CTRLA register description for details.

Related Links

31.8.1. CTRLA

31.6.2.3 Clock Generation

In the SPI host operation (CTRLA.MODE = `0x3`), the serial clock (SCK) is generated internally by the SERCOM Baud Rate Generator (BRG).

In the SPI mode, the BRG is set to Synchronous mode. The 8-bit Baud register (BAUD) value is used for generating SCK and clocking the Shift register (see *Clock Generation – Baud-Rate Generator* from Related Links).

In the SPI client operation (CTRLA.MODE = `0x2`), the clock is provided by an external host on the SCK pin. This clock is used to clock the SPI Shift register.

Related Links

29.6.2.3. Clock Generation – Baud-Rate Generator

31.6.2.4 Data Register

The SPI Transmit Data register (TxDATA) and SPI Receive Data register (RxDATA) share the same I/O address, referred to as the SPI Data register (DATA). Writing DATA register will update the Transmit Data register. Reading the DATA register will return the contents of the Receive Data register.

31.6.2.5 SPI Transfer Modes

There are four combinations of SCK phase and polarity to transfer serial data. The SPI Data Transfer modes are shown in SPI Transfer Modes (Table) and SPI Transfer Modes (Figure).

SCK phase is configured by the Clock Phase bit in the CTRLA register (CTRLA.CPHA). SCK polarity is programmed by the Clock Polarity bit in the CTRLA register (CTRLA.CPOL). Data bits are shifted out and latched in on opposite edges of the SCK signal. This ensures sufficient time for the data signals to stabilize.

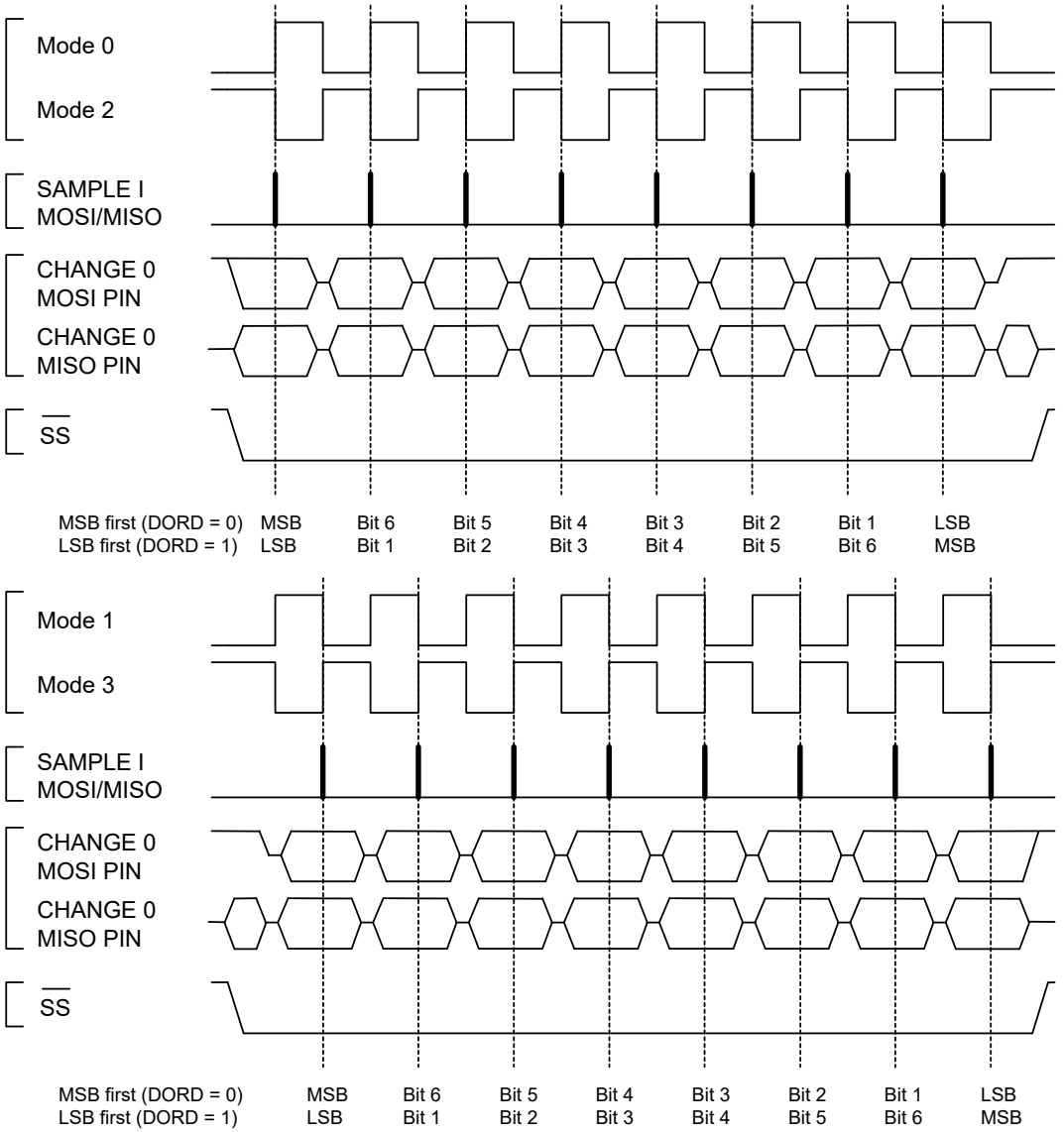
Table 31-3. SPI Transfer Modes

Mode	CPOL	CPHA	Leading Edge	Trailing Edge
0	0	0	Rising, sample	Falling, setup
1	0	1	Rising, setup	Falling, sample
2	1	0	Falling, sample	Rising, setup
3	1	1	Falling, setup	Rising, sample

Note:
Leading edge is the first clock edge in a clock cycle.

Trailing edge is the second clock edge in a clock cycle.

Figure 31-3. SPI Transfer Modes



31.6.2.6 Transferring Data

31.6.2.6.1 Host

In Host mode (CTRLA.MODE=0x3), when Host SPI Select Enable (CTRLB.MSSEN) is '1', hardware will control the $\overline{SS}$ line.

When Host SPI Select Enable (CTRLB.MSSEN) is '0', the $\overline{SS}$ line must be configured as an output. $\overline{SS}$ can be assigned to any general purpose I/O pin. When the SPI is ready for a data transaction, software must pull the $\overline{SS}$ line low.

When writing a character to the Data register (DATA), the character will be transferred to the Shift register. Once the content of TxDATA has been transferred to the Shift register, the Data Register Empty flag in the Interrupt Flag Status and Clear register (INTFLAG.DRE) will be set. And a new character can be written to DATA.

Each time one character is shifted out from the Host, another character will be shifted in from the Client simultaneously. If the receiver is enabled (CTRLA.RXEN=1), the contents of the Shift register will be transferred to the two-level receive buffer. The transfer takes place in the same clock cycle as the last data bit is shifted in. And the Receive Complete Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.RXC) will be set. The received data can be retrieved by reading DATA.

When the last character has been transmitted and there is no valid data in DATA, the Transmit Complete Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.TXC) will be set. When the transaction is finished, the Host must pull the $\overline{SS}$ line high to notify the Client. If Host SPI Select Enable (CTRLB.MSSEN) is set to '0', the software must pull the $\overline{SS}$ line high.

31.6.2.6.2 Client

In Client mode (CTRLA.MODE = `0x2`), the SPI interface will remain inactive with the MISO line tri-stated as long as the $\overline{SS}$ pin is pulled high. Software may update the contents of DATA at any time as long as the Data Register Empty flag in the Interrupt Status and Clear register (INTFLAG.DRE) is set.

When $\overline{SS}$ is pulled low and SCK is running, the client will sample and shift out data according to the Transaction mode set. Once the content of TxDATA is loaded into the Shift register, INTFLAG.DRE will be set and new data can be written to DATA.

Similar to the host, the client will receive one character for each character transmitted. A character will be transferred into the two-level receive buffer within the same clock cycle its last data bit is received. The received character can be retrieved from DATA when the Receive Complete interrupt flag (INTFLAG.RXC) is set.

When the host pulls the $\overline{SS}$ line high, the transaction is done and the Transmit Complete Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.TXC) will be set.

After DATA is written it takes up to three SCK clock cycles until the content of DATA is ready to be loaded into the Shift register on the next character boundary. As a consequence, the first character transferred in a SPI transaction will not be the content of DATA. This can be avoided by using the preloading feature (see *Preloading of the Client Shift Register* from Related Links).

When transmitting several characters in one SPI transaction, the data has to be written into DATA register with at least three SCK clock cycles left in the current character transmission. If this criteria is not met, the previously received character will be transmitted.

Once the DATA register is empty, it takes three CLK_SERCOM_APB cycles for INTFLAG.DRE to be set.

Related Links

31.6.3.2. Preloading of the Client Shift Register

31.6.2.7 Receiver Error Bit

The SPI receiver has one error bit: the Buffer Overflow bit (BUFOVF), which can be read from the Status register (STATUS). Once an error happens, the bit will stay set until it is cleared by writing '1' to it. The bit is also automatically cleared when the receiver is disabled.

There are two methods for buffer overflow notification, selected by the immediate Buffer Overflow Notification bit in the Control A register (CTRLA.IBON):

If CTRLA.IBON=1, STATUS.BUFOVF is raised immediately upon buffer overflow. Software can then empty the receive FIFO by reading RxDATA until the receiver complete Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.RXC) goes low.

If CTRLA.IBON=0, the Buffer Overflow condition travels with data through the receive FIFO. After the received data is read, STATUS.BUFOVF and INTFLAG.ERROR will be set along with INTFLAG.RXC, and RxDATA will be zero.

31.6.3 Additional Features

31.6.3.1 Address Recognition

When the SPI is configured for client operation (CTRLA.MODE=0x2) with address recognition (CTRLA.FORM is 0x2), the SERCOM address recognition logic is enabled: the first character in a transaction is checked for an address match.

If there is a match, the Receive Complete Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.RXC) is set, the MISO output is enabled, and the transaction is processed. If the device is in Sleep mode, an address match can wake-up the device in order to process the transaction.

If there is no match, the complete transaction is ignored.

If a 9-bit frame format is selected, only the lower 8 bits of the Shift register are checked against the Address register (ADDR).

Preload must be disabled (CTRLB.PLOADEN=0) in order to use this mode.

Related Links

29.6.3.1. Address Match and Mask

31.6.3.2 Preloading of the Client Shift Register

When starting a transaction, the client will first transmit the contents of the shift register before loading new data from DATA. The first character sent can be either the reset value of the shift register (if this is the first transmission since the last reset) or the last character in the previous transmission.

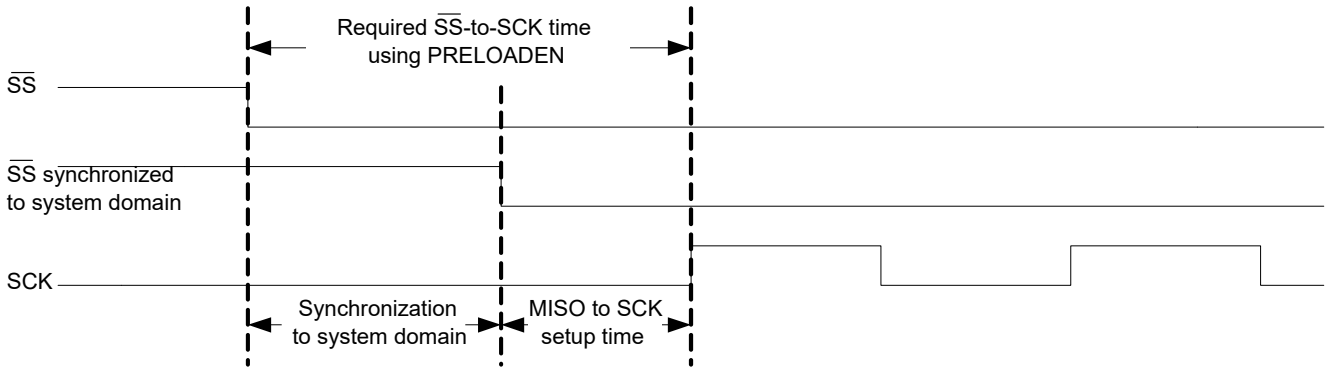
Preloading can be used to preload data into the shift register while $\overline{SS}$ is high: this eliminates sending a dummy character when starting a transaction. If the shift register is not preloaded, the current contents of the shift register will be shifted out.

Only one data character will be preloaded into the shift register while the synchronized $\overline{SS}$ signal is high. If the next character is written to DATA before $\overline{SS}$ is pulled low, the second character will be stored in DATA until transfer begins.

For proper preloading, sufficient time must elapse between $\overline{SS}$ going low and the first SCK sampling edge, as shown in the following figure. For timing details, see *Electrical Characteristics* from Related Links.

Preloading is enabled by writing '1' to the Client Data Preload Enable bit in the CTRLB register (CTRLB.PLOADEN).

Figure 31-4. Timing Using Preloading



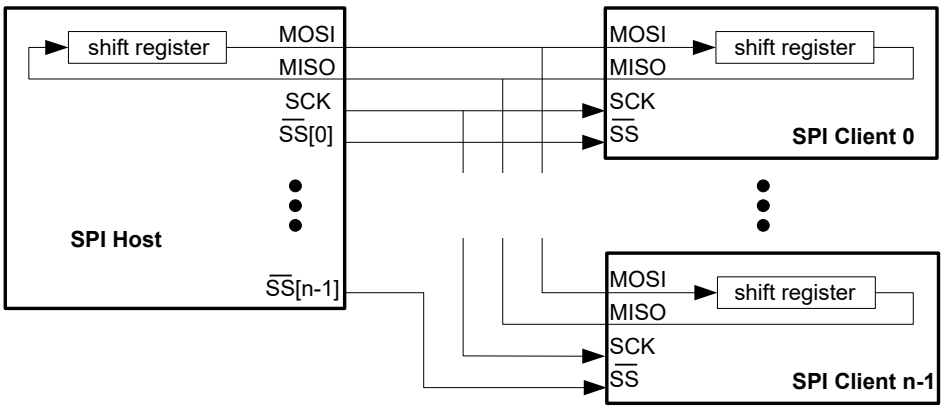
Related Links

43. Electrical Characteristics

31.6.3.3 Host with Several Clients

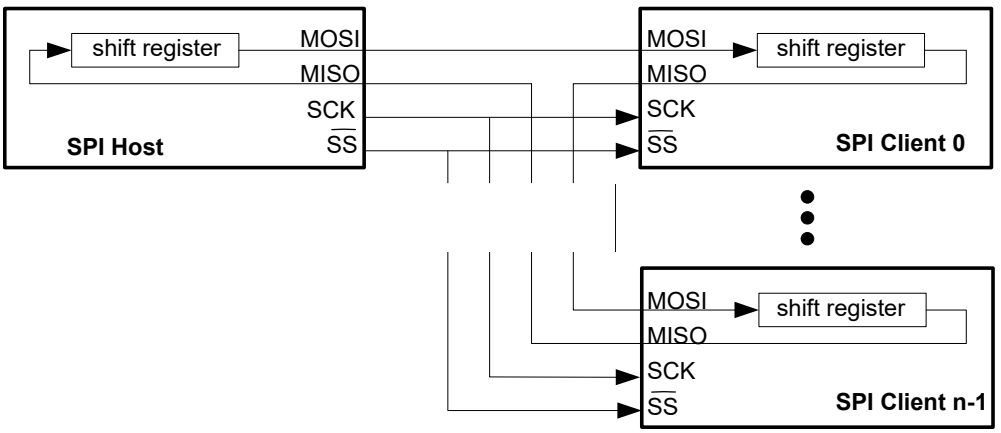
If the bus consists of several SPI clients, a SPI host can use general purpose I/O pins to control the $\overline{SS}$ line to each of the clients on the bus, as shown in the following figure. In this configuration, the single selected SPI client will drive the tri-state MISO line.

Figure 31-5. Multiple Clients in Parallel



Another configuration is multiple clients in series, as shown in the following figure. In this configuration, all n attached clients are connected in series. A common $\overline{SS}$ line is provided to all clients, enabling them simultaneously. The host must shift n characters for a complete transaction. The $\overline{SS}$ line is controlled by a normal GPIO.

Figure 31-6. Multiple Clients in Series



31.6.3.4 Loop-Back Mode

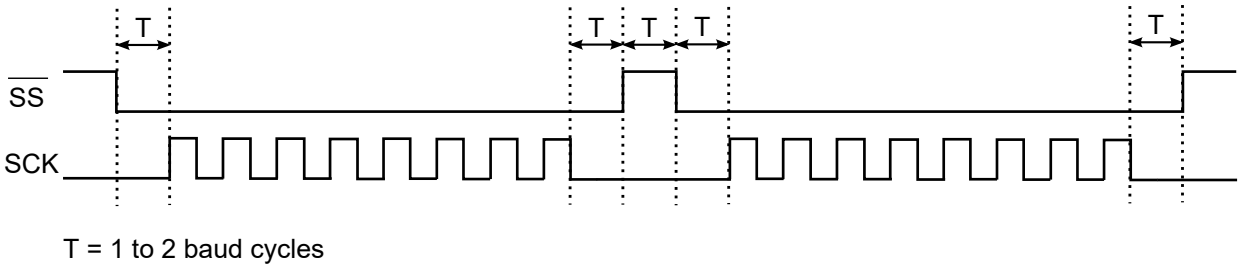
For Loop-back mode, configure the Data In Pinout (CTRLA.DIPO) and Data Out Pinout (CTRLA.DOPO) to use the same data pins for transmit and receive. The loop-back is through the pad, so the signal is also available externally.

31.6.3.5 Hardware Controlled $\overline{SS}$

In Host mode, a single $\overline{SS}$ chip select can be controlled by hardware by writing the Host SPI Select Enable (CTRLB.MSSEN) bit to '1'. In this mode, the $\overline{SS}$ pin is driven low for a minimum of one baud cycle before transmission begins, and stays low for a minimum of one baud cycle after transmission completes. The $\overline{SS}$ pin will always be driven high for a minimum of one baud cycle between each data sent.

In Hardware Controlled $\overline{SS}$, the time T is between one and two baud cycles depending on the SPI Transfer mode.

Figure 31-7. Hardware Controlled $\overline{SS}$



When CTRLB.MSSEN=0, the $\overline{SS}$ pin(s) is/are controlled by user software and normal GPIO.

31.6.3.6 SPI Select Low Detection

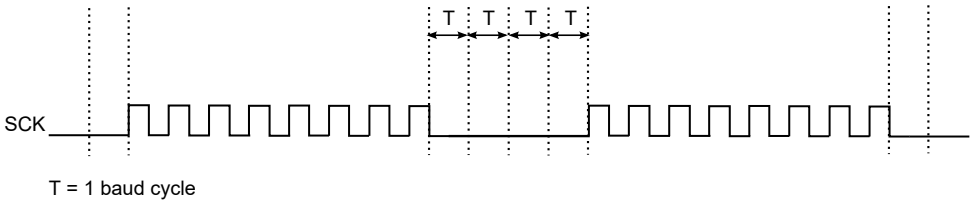
In Client mode, the SPI can wake the CPU when the SPI Select ($\overline{SS}$) goes low. When the SPI Select Low Detect is enabled (CTRLB.SSDE=1), a high-to-low transition will set the SPI Select Low Interrupt flag (INTFLAG.SSL) and the device will wake-up if applicable.

31.6.3.7 Host Inter-Character Spacing

When configured as host, inter-character spacing can be increased by writing a non-zero value to the Inter-Character Spacing bit field in the Control C register (CTRLC.ICSPACE). When non-zero, CTRLC.ICSPACE represents the minimum number of baud cycles that the SCK clock line does not toggle and the next character is stalled.

The figure gives an example for CTRLC.ICSPACE=4; In this case, the SCK is inactive for 4 baud cycles.

Figure 31-8. Four Cycle Inter-Character Spacing Example



31.6.3.8 32-bit Extension

For better system bus utilization, 32-bit data receive and transmit can be enabled by writing to the Data 32-bit bit field in the Control C register (CTRLC.DATA32B=1). When enabled, write and read transaction to/from the DATA register are 32 bit in size.

If frames are not multiples of 4 Bytes, the Length Counter (LENGTH.LEN) and Length Enable (LENGTH.LENEN) must be configured before data transfer begins. LENGTH.LEN must be enabled only when CTRLC.DATA32B is enabled.

The following figure shows the order of transmit and receive when using 32-bit mode. Bytes are transmitted or received and stored in order from 0 to 3.

Only 8-bit character size is supported.

Figure 31-9. 32-bit Extension Byte Ordering

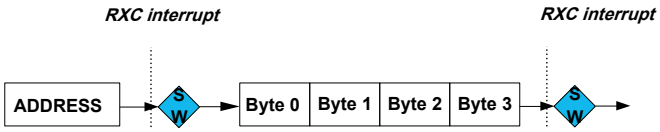
APB Write/Read	BYTE3	BYTE2	BYTE1	BYTE0
Bit Position	31			0

32-bit Extension Client Operation

The following figure shows a transaction with 32-bit Extension enabled (CTRLC.DATA32B=1). When address recognition is enabled (CTRLA.FORM=0x2) and there is an address match, the address is loaded into the FIFO as Byte zero and data begins with Byte 1. INTFLAGS.RXC will then be raised for

every 4 Bytes transferred. For transmit, there is a 32-bit holding buffer in the core domain. Once DATA has been registered in the core domain, INTFLAG.DRE will be raised, so that the next 32 bits can be written to the DATA register.

Figure 31-10. 32-bit Extension Client Operation



When utilizing the length counter, the LENGTH register must be written before the frame begins. If the frame length while $\overline{SS}$ is low is not a multiple of LENGTH.LEN Bytes, the Length Error Status bit (STATUS.LENERR) is raised. If LENGTH.LEN is not a multiple of 4 Bytes, the final INTFLAG.RXC interrupt will be raised when the last Byte is received.

The length count is based on the received Bytes, or the number of clocks if the receiver is not enabled. If pre-loading is disabled and DATA is written to for transmit before SCK starts, transmitted data will be delayed by one Byte, but the length counter will still increment for the first (empty) Byte transmission. When the counter reaches LENGTH.LEN, the internal length counter, Rx Byte counter, and Tx Byte counter are reset. If multiple lengths are to be transmitted, INTFLAG.TXC must go high before writing DATA for subsequent lengths.

If there is a Length Error (STATUS.LENERR), the remaining Bytes in the length will be transmitted at the beginning of the next frame. If this is not desired, the SERCOM must be disabled and re-enabled in order to flush the Tx and Rx pipelines.

Writing the LENGTH register while a frame is in progress will produce unpredictable results. If LENGTH.LENEN is not configured and a frame is not a multiple of 4 Bytes (while $\overline{SS}$ is low), the remainder will be transmitted in the next frame.

32-bit Extension Host Operation

When using the SPI configured as Host, the Length and the Length Enable bit fields (LENGTH.LEN and LENGTH.LENEN) must be written before the frame begins. When LENGTH.LENEN is written to '1', the value of LENGTH.LEN determines the number of data bytes in the transaction from 1 to 255.

For receive data, INTFLAG.RXC is raised every 4 Bytes received. If LENGTH.LEN is not a multiple of 4 Bytes, the final INTFLAG.RXC is set when the final byte is received.

For transmit, there is a holding buffer for the 32-bit data in the core domain. Once DATA has been registered in the core domain, INTFLAG.DRE will be raised so that the next 32 bits can be written to the DATA register.

If multiple lengths are to be transmitted, INTFLAG.TXC must go high before writing DATA for subsequent lengths.

31.6.4 DMA, Interrupts, and Events

Table 31-4. Module Request for SERCOM SPI

Condition	Request		
	DMA	Interrupt	Event
Data Register Empty (DRE)	Yes (request cleared when data is written)	Yes	NA
Receive Complete (RXC)	Yes (request cleared when data is read)	Yes	
Transmit Complete (TXC)	NA	Yes	
Client Select low (SSL)	NA	Yes	
Error (ERROR)	NA	Yes	

31.6.4.1 DMA Operation

The SPI generates the following DMA requests:

31.6.4.2 Interrupts

The SPI has the following interrupt sources. These are asynchronous interrupts, and can wake-up the device from any Sleep mode:

- Data Register Empty (DRE)
- Receive Complete (RXC)
- Transmit Complete (TXC)
- SPI Select Low (SSL)
- Error (ERROR)

Each interrupt source has its own Interrupt flag. The Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG) will be set when the Interrupt condition is met. Each interrupt can be individually enabled by writing '1' to the corresponding bit in the Interrupt Enable Set register (INTENSET), and disabled by writing '1' to the corresponding bit in the Interrupt Enable Clear register (INTENCLR). The status of enabled interrupts can be read from either INTENSET or INTENCLR.

An interrupt request is generated when the Interrupt flag is set and if the corresponding interrupt is enabled. The interrupt request remains active until either the Interrupt flag is cleared, the interrupt is disabled, or the SPI is reset. For details on clearing Interrupt flags, see INTFLAG register description.

The value of INTFLAG indicates which interrupt is executed. Note that interrupts must be globally enabled for interrupt requests. See *Nested Vector Interrupt Controller (NVIC)* from Related Links.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)

31.6.4.3 Events

Not applicable.

31.6.5 Sleep Mode Operation

The behavior in sleep mode is depending on the Host/Client configuration and the Run In Standby bit in the Control A register (CTRLA.RUNSTDBY):

- **Host operation, CTRLA.RUNSTDBY=1:** The peripheral clock GCLK_SERCOM_CORE will continue to run in idle sleep mode and in standby sleep mode. Any interrupt can wake up the device.

- **Host operation, CTRLA.RUNSTDBY=0:** GLK_SERCOMx_CORE will be disabled after the ongoing transaction is finished. Any interrupt can wake up the device.
- **Client operation, CTRLA.RUNSTDBY=1:** The Receive Complete interrupt can wake up the device
- **Client operation, CTRLA.RUNSTDBY=0:** All reception will be dropped, including the ongoing transaction

31.6.6 Synchronization

Due to asynchronicity between the main clock domain and the peripheral clock domains, some registers need to be synchronized when written or read.

The following bits are synchronized when written:

- Software Reset bit in the CTRLA register (CTRLA.SWRST)
- Enable bit in the CTRLA register (CTRLA.ENABLE)
- Receiver Enable bit in the CTRLB register (CTRLB.RXEN)

Note: CTRLB.RXEN is write-synchronized somewhat differently. See *CTRLB* register from Related Links.

Required write synchronization is denoted by the "Write-Synchronized" property in the register description.

Related Links

31.8.2. CTRLB

31.7 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0	RUNSTDBY			MODE[2:0]			ENABLE	SWRST
		15:8								IBON
		23:16			DIPO[1:0]				DOPO[1:0]	
		31:24		DORD	CPOL	CPHA	FORM[3:0]			
0x04	CTRLB	7:0		PLOADEN				CHSIZE[2:0]		
		15:8	AMODE[1:0]		MSSEN				SSDE	
		23:16							RXEN	
		31:24								
0x08 ... 0x0B	Reserved									
0x0C	BAUD	7:0	BAUD[7:0]							
0x0D ... 0x13	Reserved									
0x14	INTENCLR	7:0	ERROR				SSL	RXC	TXC	DRE
0x15	Reserved									
0x16	INTENSET	7:0	ERROR				SSL	RXC	TXC	DRE
0x17	Reserved									
0x18	INTFLAG	7:0	ERROR				SSL	RXC	TXC	DRE
0x19	Reserved									
0x1A	STATUS	7:0						BUFOVF		
		15:8								
0x1C	SYNCBUSY	7:0						CTRLB	ENABLE	SWRST
		15:8								
		23:16								
		31:24								
0x20 ... 0x23	Reserved									
0x24	ADDR	7:0	ADDR[7:0]							
		15:8								
		23:16	ADDRMASK[7:0]							
		31:24								
0x28	DATA	7:0	DATA[7:0]							
		15:8	DATA[15:8]							
		23:16	DATA[23:16]							
		31:24	DATA[31:24]							
0x2C ... 0x2F	Reserved									
0x30	DBGCTRL	7:0								DBGSTOP

31.8 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register, and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers require synchronization when read and/or written. Synchronization is denoted by the “Read-Synchronized” and/or “Write-Synchronized” property in each individual register description.

Some registers are enable-protected, meaning they can only be written when the module is disabled. Enable-protection is denoted by the “Enable-Protected” property in each individual register description.

Optional write protection by the Peripheral Access Controller (PAC) is denoted by the "PAC Write Protection" property in each individual register description.

See *Peripheral Access Controller (PAC)* from Related Links.

Related Links

26. Peripheral Access Controller (PAC)

31.8.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected, Write-Synchronized

Bit	31	30	29	28	27	26	25	24
		DORD	CPOL	CPHA	FORM[3:0]			
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
			DIPO[1:0]				DOPO[1:0]	
Access			R/W	R/W			R/W	R/W
Reset			0	0			0	0

Bit	15	14	13	12	11	10	9	8
								IBON
Access								R/W
Reset								0

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY			MODE[2:0]			ENABLE	SWRST
Access	R/W			R/W	R/W	R/W	R/W	R/W
Reset	0			0	0	0	0	0

Bit 30 – DORD Data Order

This bit selects the data order when a character is shifted out from the shift register.
This bit is not synchronized.

Value	Description
0	MSB is transferred first.
1	LSB is transferred first.

Bit 29 – CPOL Clock Polarity

In combination with the Clock Phase bit (CPHA), this bit determines the SPI transfer mode.
This bit is not synchronized.

Value	Description
0	SCK is low when idle. The leading edge of a clock cycle is a rising edge, while the trailing edge is a falling edge.
1	SCK is high when idle. The leading edge of a clock cycle is a falling edge, while the trailing edge is a rising edge.

Bit 28 – CPHA Clock Phase

In combination with the Clock Polarity bit (CPOL), this bit determines the SPI transfer mode.
This bit is not synchronized.

Mode	CPOL	CPHA	Leading Edge	Trailing Edge
0x0	0	0	Rising, sample	Falling, change
0x1	0	1	Rising, change	Falling, sample
0x2	1	0	Falling, sample	Rising, change
0x3	1	1	Falling, change	Rising, sample

Value	Description
0	The data is sampled on a leading SCK edge and changed on a trailing SCK edge.
1	The data is sampled on a trailing SCK edge and changed on a leading SCK edge.

Bits 27:24 – FORM[3:0] Frame Format

This bit field selects the various frame formats supported by the SPI in client mode. When the 'SPI frame with address' format is selected, the first byte received is checked against the ADDR register.

FORM[3:0]	Name	Description
0x0	SPI	SPI frame
0x1	—	Reserved
0x2	SPI_ADDR	SPI frame with address
0x3-0xF	—	Reserved

Bits 21:20 – DIPO[1:0] Data In Pinout

These bits define the data in (DI) pad configurations.
In host operation, DI is MISO.
In client operation, DI is MOSI.
These bits are not synchronized.

DIPO[1:0]	Name	Description
0x0	PAD[0]	SERCOM PAD[0] is used as data input
0x1	PAD[1]	SERCOM PAD[1] is used as data input
0x2	PAD[2]	SERCOM PAD[2] is used as data input
0x3	PAD[3]	SERCOM PAD[3] is used as data input

Bits 17:16 – DOPO[1:0] Data Out Pinout

This bit defines the available pad configurations for data out (DO), the serial clock (SCK) and the SPI Select ($\overline{SS}$). In Client operation, the SPI Select line ($\overline{SS}$) is controlled by DOPO. In host operation, the SPI Select line ($\overline{SS}$) is either controlled by DOPO when CTRLB.MSSEN=1, or by a GPIO driven by the application when CTRLB.MSSEN=0.
In host operation, DO is MOSI.
In client operation, DO is MISO.
These bits are not synchronized.

DOPO	DO	SCK	Client $\overline{SS}$	Host $\overline{SS}$ (MSSEN = 1)	Host $\overline{SS}$ (MSSEN = 0)
0x0	PAD[0]	PAD[1]	PAD[2]	PAD[2]	Any GPIO configured by the application
0x2	PAD[3]	PAD[1]	PAD[2]	PAD[2]	Any GPIO configured by the application

Bit 8 – IBON Immediate Buffer Overflow Notification

This bit controls when the Buffer Overflow Status bit (STATUS.BUFOVF) is set when a buffer overflow occurs.
This bit is not synchronized.

Value	Description
0	STATUS.BUFOVF is set when it occurs in the data stream.
1	STATUS.BUFOVF is set immediately upon buffer overflow.

Bit 7 – RUNSTDBY Run In Standby

This bit defines the functionality in Standby mode.
These bits are not synchronized.

RUNSTDBY	Client	Host
0x0	Disabled. All reception is dropped, including the ongoing transaction.	Generic clock is disabled when ongoing transaction is finished. All interrupts can wake up the device.
0x1	Ongoing transaction continues, wake on Receive Complete interrupt.	Generic clock is enabled while in sleep modes. All interrupts can wake up the device.

Bits 4:2 – MODE[2:0] Operating Mode

These bits must be written to 0x2 or 0x3 to select the SPI serial communication interface of the SERCOM.
0x2: SPI client operation

0x3: SPI host operation
These bits are not synchronized.

Bit 1 – ENABLE Enable
Due to synchronization, there is delay from writing CTRLA.ENABLE until the peripheral is enabled/disabled. The value written to CTRL.ENABLE will read back immediately and the Synchronization Enable Busy bit in the Synchronization Busy register (SYNCBUSY.ENABLE) will be set. SYNCBUSY.ENABLE is cleared when the operation is complete.
This bit is not enable-protected.

Value	Description
0	The peripheral is disabled or being disabled.
1	The peripheral is enabled or being enabled.

Bit 0 – SWRST Software Reset
Writing '0' to this bit has no effect.
Writing '1' to this bit resets all registers in the SERCOM, except DBGCTRL, to their initial state, and the SERCOM will be disabled.
Writing '1' to CTRL.SWRST will always take precedence, meaning that all other writes in the same write-operation will be discarded. Any register write access during the ongoing reset will result in an APB error. Reading any register will return the reset value of the register.
Due to synchronization, there is a delay from writing CTRLA.SWRST until the reset is complete. CTRLA.SWRST and SYNCBUSY. SWRST will both be cleared when the reset is complete.
This bit is not enable-protected.
Note: During a SWRST, access to registers/bits without SWRST are disallowed until SYNCBUSY.SWRST cleared by hardware.

Value	Description
0	There is no reset operation ongoing.
1	The reset operation is ongoing.

31.8.2 Control B

Name: CTRLB
Offset: 0x04
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected, Write-Synchronized

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
							RXEN	
Access							R/W	
Reset							0	

Bit	15	14	13	12	11	10	9	8
	AMODE[1:0]		MSSEN				SSDE	
Access	R/W	R/W	R/W				R/W	
Reset	0	0	0				0	

Bit	7	6	5	4	3	2	1	0
		PLOADEN				CHSIZE[2:0]		
Access		R/W				R/W	R/W	R/W
Reset		0				0	0	0

Bit 17 – RXEN Receiver Enable

Writing '0' to this bit will disable the SPI receiver immediately. The receive buffer will be flushed, data from ongoing receptions will be lost and STATUS.BUFOVF will be cleared.
Writing '1' to CTRLB.RXEN when the SPI is disabled will set CTRLB.RXEN immediately. When the SPI is enabled, CTRLB.RXEN will be cleared, SYNCBUSY.CTRLB will be set and remain set until the receiver is enabled. When the receiver is enabled CTRLB.RXEN will read back as '1'.
Writing '1' to CTRLB.RXEN when the SPI is enabled will set SYNCBUSY.CTRLB, which will remain set until the receiver is enabled, and CTRLB.RXEN will read back as '1'.
This bit is not enable-protected.

Value	Description
0	The receiver is disabled.
1	The receiver is enabled or it will be enabled when SPI is enabled.

Bits 15:14 – AMODE[1:0] Address Mode

These bits set the Client Addressing mode when the frame format (CTRLA.FORM) with address is used. They are unused in Host mode.
These bits are not synchronized.

AMODE[1:0]	Name	Description
0x0	MASK	ADDRMASK is used as a mask to the ADDR register
0x1	2_ADDRS	The client responds to the two unique addresses in ADDR and ADDRMASK
0x2	RANGE	The client responds to the range of addresses between and including ADDR and ADDRMASK. ADDR is the upper limit
0x3	—	Reserved

Bit 13 – MSSEN Host SPI Select Enable

This bit enables hardware SPI Select ($\overline{SS}$) control.

This bit is not synchronized.

Value	Description
0	Hardware $\overline{SS}$ control is disabled.
1	Hardware $\overline{SS}$ control is enabled.

Bit 9 – SSDE SPI Select Low Detect Enable

This bit enables wake-up when the SPI Select ($\overline{SS}$) pin transitions from high-to-low.
This bit is not synchronized.

Value	Description
0	$\overline{SS}$ low detector is disabled.
1	$\overline{SS}$ low detector is enabled.

Bit 6 – PLOADEN Client Data Preload Enable

Setting this bit will enable preloading of the Client Shift register when there is no transfer in progress. If the $\overline{SS}$ line is high when DATA is written, it will be transferred immediately to the Shift register.
This bit is not synchronized.

Bits 2:0 – CHSIZE[2:0] Character Size

These bits are not synchronized.

CHSIZE[2:0]	Name	Description
0x0	8BIT	8 bits
0x1	9BIT	9 bits
0x2-0x7	—	Reserved

31.8.3 Baud Rate

Name: BAUD
Offset: 0x0C
Reset: 0x00
Property: PAC Write-Protection, Enable-Protected

Bit	7	6	5	4	3	2	1	0
	BAUD[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – BAUD[7:0] Baud Register
These bits control the clock generation, as described in the *SERCOM Clock Generation – Baud-Rate Generator*.

31.8.4 Interrupt Enable Clear

Name: INTENCLR
Offset: 0x14
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to disable an interrupt without read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Set register (INTENSET).

Bit	7	6	5	4	3	2	1	0
	ERROR				SSL	RXC	TXC	DRE
Access	R/W				R/W	R/W	R/W	R/W
Reset	0				0	0	0	0

Bit 7 – ERROR Error Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Error Interrupt Enable bit, which disables the Error interrupt.

Value	Description
0	Error interrupt is disabled.
1	Error interrupt is enabled.

Bit 3 – SSL Client Select Low Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Client Select Low Interrupt Enable bit, which disables the Client Select Low interrupt.

Value	Description
0	Client Select Low interrupt is disabled.
1	Client Select Low interrupt is enabled.

Bit 2 – RXC Receive Complete Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Receive Complete Interrupt Enable bit, which disables the Receive Complete interrupt.

Value	Description
0	Receive Complete interrupt is disabled.
1	Receive Complete interrupt is enabled.

Bit 1 – TXC Transmit Complete Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Transmit Complete Interrupt Enable bit, which disable the Transmit Complete interrupt.

Value	Description
0	Transmit Complete interrupt is disabled.
1	Transmit Complete interrupt is enabled.

Bit 0 – DRE Data Register Empty Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Data Register Empty Interrupt Enable bit, which disables the Data Register Empty interrupt.

Value	Description
0	Data Register Empty interrupt is disabled.
1	Data Register Empty interrupt is enabled.

31.8.5 Interrupt Enable Set

Name: INTENSET
Offset: 0x16
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to disable an interrupt without read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Clear register (INTENCLR).

Bit	7	6	5	4	3	2	1	0
	ERROR				SSL	RXC	TXC	DRE
Access	R/W				R/W	R/W	R/W	R/W
Reset	0				0	0	0	0

Bit 7 – ERROR Error Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Error Interrupt Enable bit, which enables the Error interrupt.

Value	Description
0	Error interrupt is disabled.
1	Error interrupt is enabled.

Bit 3 – SSL Client Select Low Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Client Select Low Interrupt Enable bit, which enables the Client Select Low interrupt.

Value	Description
0	Client Select Low interrupt is disabled.
1	Client Select Low interrupt is enabled.

Bit 2 – RXC Receive Complete Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Receive Complete Interrupt Enable bit, which enables the Receive Complete interrupt.

Value	Description
0	Receive Complete interrupt is disabled.
1	Receive Complete interrupt is enabled.

Bit 1 – TXC Transmit Complete Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Transmit Complete Interrupt Enable bit, which enables the Transmit Complete interrupt.

Value	Description
0	Transmit Complete interrupt is disabled.
1	Transmit Complete interrupt is enabled.

Bit 0 – DRE Data Register Empty Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Data Register Empty Interrupt Enable bit, which enables the Data Register Empty interrupt.

Value	Description
0	Data Register Empty interrupt is disabled.
1	Data Register Empty interrupt is enabled.

31.8.6 Interrupt Flag Status and Clear

Name: INTFLAG
Offset: 0x18
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	ERROR				SSL	RXC	TXC	DRE
Access	R/W				R/W	R	R/W	R
Reset	0				0	0	0	0

Bit 7 – ERROR Error
This flag is cleared by writing '1' to it.
This bit is set when any error is detected. Errors that will set this flag have corresponding Status flags in the STATUS register. The BUFOVF error will set this Interrupt flag.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the flag.

Bit 3 – SSL SPI Select Low
This flag is cleared by writing '1' to it.
This bit is set when a high to low transition is detected on the $\overline{SS}$ pin in Client mode and SPI Select Low Detect (CTRLB.SSDE) is enabled.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the flag.

Bit 2 – RXC Receive Complete
This flag is cleared by reading the Data (DATA) register or by disabling the receiver.
This flag is set when there are unread data in the receive buffer. If address matching is enabled, the first data received in a transaction will be an address.
Writing '0' to this bit has no effect.
Writing '1' to this bit has no effect.

Bit 1 – TXC Transmit Complete
This flag is cleared by writing '1' to it or by writing new data to DATA.
In Host mode, this flag is set when the data have been shifted out and there are no new data in DATA.
In Client mode, this flag is set when the $\overline{SS}$ pin is pulled high. If address matching is enabled, this flag is only set if the transaction was initiated with an address match.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the flag.

Bit 0 – DRE Data Register Empty
This flag is cleared by writing new data to DATA.
This flag is set when DATA is empty and ready for new data to transmit.
Writing '0' to this bit has no effect.
Writing '1' to this bit has no effect.

31.8.7 Status

Name: STATUS
Offset: 0x1A
Reset: 0x0000
Property: –

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
						BUFOVF		
Access						R/W		
Reset						0		

Bit 2 – BUFOVF Buffer Overflow

Reading this bit before reading DATA will indicate the error status of the next character to be read.
This bit is cleared by writing '1' to the bit or by disabling the receiver.
This bit is set when a Buffer Overflow condition is detected. See *CTRLA* from Related Links for overflow handling.
When set, the corresponding RxDATA will be zero.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear it.

Value	Description
0	No Buffer Overflow has occurred.
1	A Buffer Overflow has occurred.

Related Links

31.8.1. CTRLA

31.8.8 Synchronization Busy

Name: SYNCBUSY
Offset: 0x1C
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
						CTRLB	ENABLE	SWRST
Access						R	R	R
Reset						0	0	0

Bit 2 – CTRLB CTRLB Synchronization Busy
Writing to the CTRLB when the SERCOM is enabled requires synchronization. Ongoing synchronization is indicated by SYNCBUSY.CTRLB=1 until synchronization is complete. If CTRLB is written while SYNCBUSY.CTRLB=1, an APB error will be generated.

Value	Description
0	CTRLB synchronization is not busy.
1	CTRLB synchronization is busy.

Bit 1 – ENABLE SERCOM Enable Synchronization Busy
Enabling and disabling the SERCOM (CTRLA.ENABLE) requires synchronization. Ongoing synchronization is indicated by SYNCBUSY.ENABLE=1 until synchronization is complete.

Value	Description
0	Enable synchronization is not busy.
1	Enable synchronization is busy.

Bit 0 – SWRST Software Reset Synchronization Busy
Resetting the SERCOM (CTRLA.SWRST) requires synchronization. Ongoing synchronization is indicated by SYNCBUSY.SWRST=1 until synchronization is complete.
Note: During a SWRST, access to registers/bits without SWRST are disallowed until SYNCBUSY.SWRST cleared by hardware.

Value	Description
0	SWRST synchronization is not busy.
1	SWRST synchronization is busy.

31.8.9 Address

Name: ADDR
Offset: 0x24
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	ADDRMASK[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	ADDR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 23:16 – ADDRMASK[7:0] Address Mask
These bits hold the address mask when the transaction format with address is used (CTRLA.FORM, CTRLB.AMODE).

Bits 7:0 – ADDR[7:0] Address
These bits hold the address when the transaction format with address is used (CTRLA.FORM, CTRLB.AMODE).

31.8.10 Data

Name: DATA
Offset: 0x28
Reset: 0x0000
Property: –

Bit	31	30	29	28	27	26	25	24
	DATA[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	DATA[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	DATA[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DATA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – DATA[31:0] Data

Reading these bits will return the contents of the receive data buffer. The register must be read only when the Receive Complete Interrupt Flag bit in the Interrupt Flag Status and Clear register (INTFLAG.RXC) is set.

Writing these bits will write the transmit data buffer. This register must be written only when the Data Register Empty Interrupt Flag bit in the Interrupt Flag Status and Clear register (INTFLAG.DRE) is set.

Reads and writes are 32-bit or CTLB.CHSIZE based on the CTRLC.DATA32B setting.

31.8.11 Debug Control

Name: DBGCTRL
Offset: 0x30
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
								DBGSTOP
Access								R/W
Reset								0

Bit 0 – DBGSTOP Debug Stop Mode
This bit controls the functionality when the CPU is halted by an external debugger.

Value	Description
0	The baud-rate generator continues normal operation when the CPU is halted by an external debugger.
1	The baud-rate generator is halted when the CPU is halted by an external debugger.

32. SERCOM Inter-Integrated Circuit (SERCOM I²C)

32.1 Overview

The Inter-Integrated Circuit (I²C) interface is one of the available modes in the serial communication interface (SERCOM).

The I²C interface uses the SERCOM transmitter and receiver configured as shown in Figure 32-1. Labels in capital letters are registers accessible by the CPU, while lowercase labels are internal to the SERCOM.

A SERCOM instance can be configured to be either an I²C host or an I²C client. Both host and client have an interface containing a shift register, a transmit buffer and a receive buffer. In addition, the I²C host uses the SERCOM baud-rate generator, while the I²C client uses the SERCOM address match logic.

Note: Traditional Inter-Integrated Circuit (I²C) documentation uses the terminology “Master” and “Slave”. The equivalent Microchip terminology used in this document is “Host” and “Client”, respectively.

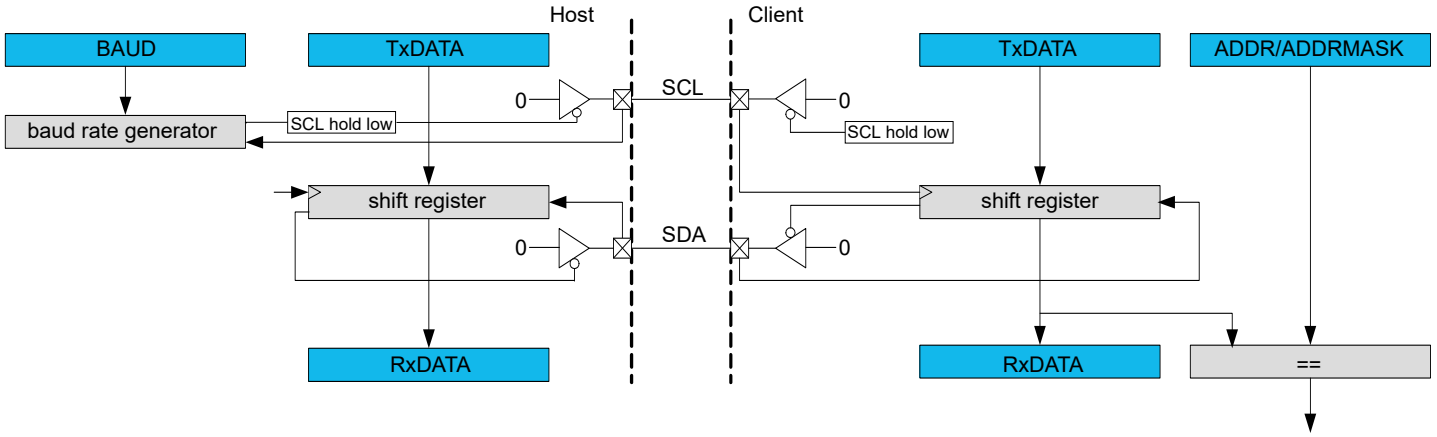
32.2 Features

SERCOM I²C includes the following features:

- Host or Client Operation
- Can be used with DMA
- Philips I²C Compatible
- SMBus Compatible
- PMBus™ Compatible
- Support of 100 kHz and 400 kHz, 1 MHz I²C mode
- 4-Wire Operation Supported
- Physical interface includes:
  - Slew-rate limited outputs
  - Filtered inputs
- Client Operation:
  - Operation in all Sleep modes
  - Wake-up on address match
  - 7-bit Address match in hardware for:
  - - Unique address and/or 7-bit general call address
    - Address range
    - Two unique addresses can be used with DMA

32.3 Block Diagram

Figure 32-1. I²C Single-Host Single-Client Interconnection



32.4 Signal Description

Signal Name	Type	Description
PAD[0]	Digital I/O	SDA
PAD[1]	Digital I/O	SCL
PAD[2]	Digital I/O	SDA_OUT (4-wire operation)
PAD[3]	Digital I/O	SCL_OUT (4-wire operation)

One signal can be mapped on several pins.

Not all the pins are I²C pins.

Related Links

32.6.3.3. 4-Wire Mode

32.5 Product Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

32.5.1 I/O Lines

In order to use the SERCOM's I/O lines, the I/O pins must be configured using the System Configuration registers only. I2C does not operate through PPS. See DEVCFG1 configuration bits SCOMn_HSEN in Configuration Bits Fuses and also CFGCON1 SCOMn_HSEN in CFGCON1(L) register. See *CFGCON1(L)* register from Related Links.

When the SERCOM is used in I²C mode, the SERCOM controls the direction and value of the I/O pins. In I²C mode pull-up resistors are disabled. External pull-up resistors are required for proper function.

Related Links

18.4.2. CFGCON1(L)

32.5.2 Power Management

This peripheral can continue to operate in any Sleep mode where its source clock is running. The interrupts can wake-up the device from Sleep modes.

32.5.3 Clocks

Two generic clocks are used by SERCOM, GCLK_SERCOMx_CORE and GCLK_SERCOMx_SLOW. The core clock (GCLK_SERCOMx_CORE) can clock the I^2C when working as a host. The slow clock (GCLK_SERCOMx_SLOW) is required only for certain functions, e.g., SMBus timing. These two clocks must be configured and enabled in the CRU registers before using the I^2C.

These generic clocks are asynchronous to the bus clock (PBx_CLK). Due to this asynchronicity, writes to certain registers will require synchronization between the clock domains.

32.5.4 DMA

The DMA request lines are connected to the DMA Controller (DMAC). To use DMA requests with this peripheral, the DMAC must be configured first (see *Direct Memory Access Controller (DMAC)* from Related Links).

Related Links

22. Direct Memory Access Controller (DMAC)

32.5.5 Interrupts

The interrupt request line is connected to the Interrupt Controller. In order to use interrupt requests of this peripheral, the Interrupt Controller (NVIC) must be configured first. See *Nested Vector Interrupt Controller (NVIC)* from Related Links.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)

32.5.6 Events

Not applicable.

32.5.7 Debug Operation

When the CPU is halted in Debug mode, this peripheral will continue normal operation. If the peripheral is configured to require periodical service by the CPU through interrupts or similar, improper operation or data loss may result during debugging. This peripheral can be forced to halt operation during debugging - refer to the Debug Control (DBGCTRL) register for details.

Related Links

32.10.11. DBGCTRL

32.5.8 Register Access Protection

Registers with write access can be write-protected optionally by the Peripheral Access Controller (PAC).

PAC write protection is not available for the following registers:

- Interrupt Flag Clear and Status register (INTFLAG)
- Status register (STATUS)
- Data register (DATA)
- Address register (ADDR) in Host mode

Optional PAC write protection is denoted by the "PAC Write-Protection" property in each individual register description.

Write-protection does not apply to accesses through an external debugger.

32.5.9 Analog Connections

Not applicable.

32.6 Functional Description

32.6.1 Principle of Operation

The I²C interface uses two physical lines for communication:

- Serial Data Line (SDA) for data transfer
- Serial Clock Line (SCL) for the bus clock

A transaction starts with the I²C host sending the Start condition, followed by a 7-bit address and a direction bit (read or write to/from the client).

The addressed I²C client will then Acknowledge (ACK) the address, and data packet transactions can begin. Every 9-bit data packet consists of 8 data bits followed by a one-bit reply indicating whether the data was acknowledged or not.

If a data packet is Not Acknowledged (NACK), whether by the I²C client or host, the I²C host takes action by either terminating the transaction by sending the Stop condition, or by sending a repeated start to transfer more data.

The figure below illustrates the possible transaction formats and Transaction Diagram Symbols explains the transaction symbols. These symbols will be used in the following descriptions.

Figure 32-2. Transaction Diagram Symbols

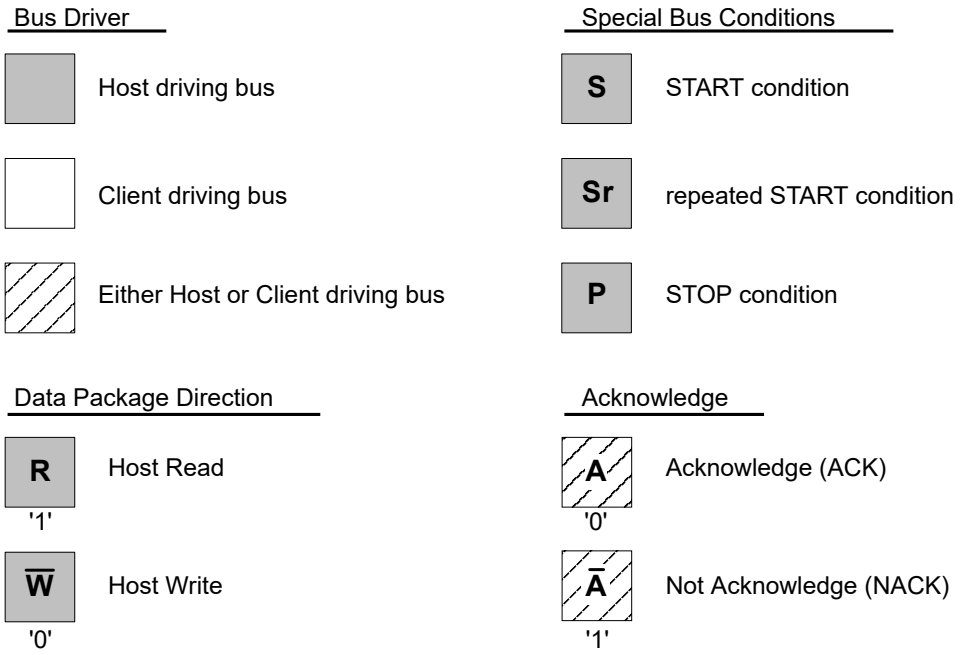
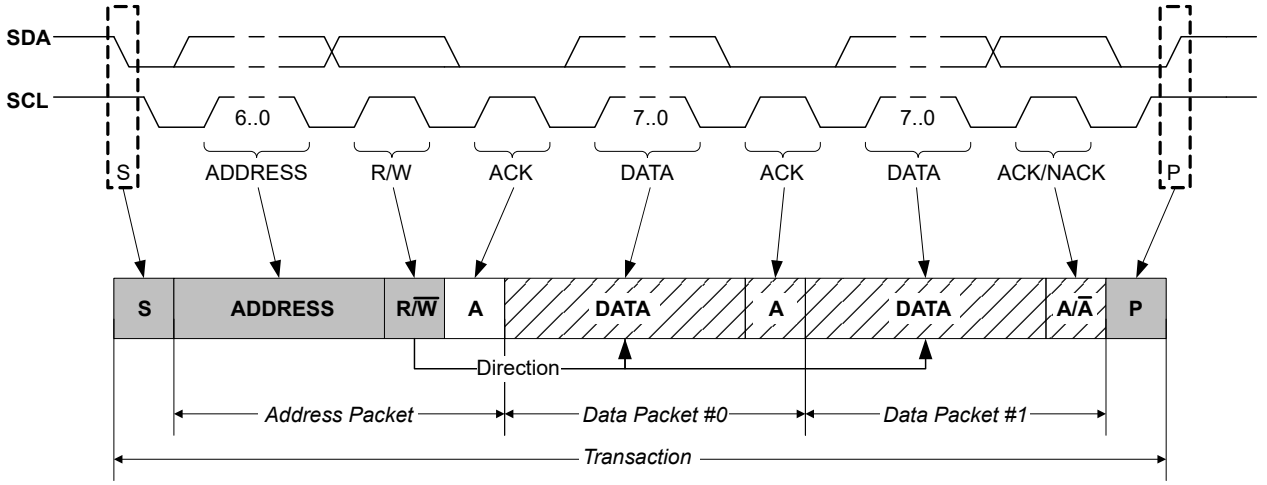


Figure 32-3. Basic I²C Transaction Diagram



32.6.2 Basic Operation

32.6.2.1 Initialization

The following registers are enable-protected, meaning they can be written only when the I²C interface is disabled (CTRLA.ENABLE is '0'):

- Control A register (CTRLA), except Enable (CTRLA.ENABLE) and Software Reset (CTRLA.SWRST) bits
- Control B register (CTRLB), except Acknowledge Action (CTRLB.ACKACT) and Command (CTRLB.CMD) bits
- Baud register (BAUD)
- Address register (ADDR) in client operation.

When the I²C is enabled or is being enabled (CTRLA.ENABLE=1), writing to these registers will be discarded. If the I²C is being disabled, writing to these registers will be completed after the disabling.

Enable-protection is denoted by the "Enable-Protection" property in the register description.

Before the I²C is enabled it must be configured as outlined by the following steps:

1. Select I²C Host or Client mode by writing 0x4 (Client mode) or 0x5 (Host mode) to the Operating Mode bits in the CTRLA register (CTRLA.MODE).
2. If desired, select the SDA Hold Time value in the CTRLA register (CTRLA.SDAHOLD).
3. If desired, enable smart operation by setting the Smart Mode Enable bit in the CTRLB register (CTRLB.SMEN).
4. If desired, enable SCL low time-out by setting the SCL Low Time-Out bit in the Control A register (CTRLA.LOWTOUT).
5. In Host mode:
   a. Select the inactive bus time-out in the Inactive Time-Out bit group in the CTRLA register (CTRLA.INACTOUT).
   b. Write the Baud Rate register (BAUD) to generate the desired baud rate.

   In Client mode:
   a. Configure the address match configuration by writing the Address Mode value in the CTRLB register (CTRLB.AMODE).
   b. Set the Address and Address Mask value in the Address register (ADDR.ADDR and ADDR.ADDRMASK) according to the address configuration.

32.6.2.2 Enabling, Disabling, and Resetting

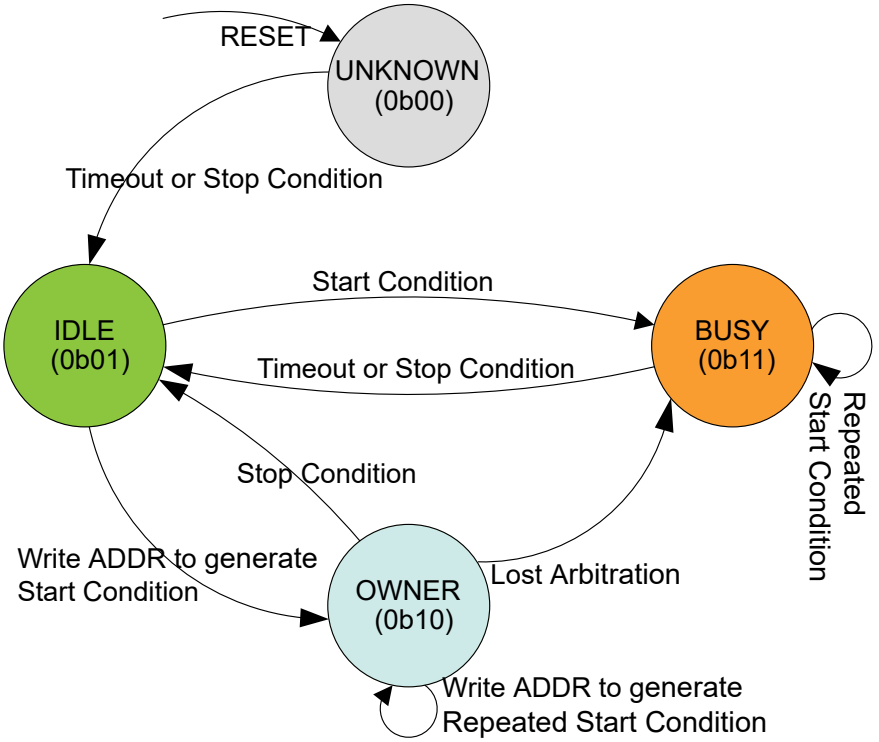
This peripheral is enabled by writing '1' to the Enable bit in the Control A register (CTRLA.ENABLE), and disabled by writing '0' to it.

Writing '1' to the Software Reset bit in the Control A register (CTRLA.SWRST) will reset all registers of this peripheral to their initial states, except the DBGCTRL register, and the peripheral is disabled.

32.6.2.3 I²C Bus State Logic

The Bus state logic includes several logic blocks that continuously monitor the activity on the I²C bus lines in all Sleep modes with running GCLK_SERCOM_x clocks. The start and stop detectors and the bit counter are all essential in the process of determining the current Bus state. The Bus state is determined according to Bus State Diagram. Software can get the current Bus state by reading the Host Bus State bits in the Status register (STATUS.BUSSTATE). The value of STATUS.BUSSTATE in the figure is shown in binary.

Figure 32-4. Bus State Diagram



The Bus state machine is active when the I²C host is enabled.

After the I²C host has been enabled, the Bus state is UNKNOWN (0b00). From the UNKNOWN state, the bus will transition to IDLE (0b01) by either:

- Forcing by writing 0b01 to STATUS.BUSSTATE
- A Stop condition is detected on the bus
- If the inactive bus time-out is configured for SMBus compatibility (CTRLA.INACTOUT) and a time-out occurs.

Note: Once a known Bus state is established, the Bus state logic will not re-enter the UNKNOWN state.

When the bus is IDLE it is ready for a new transaction. If a Start condition is issued on the bus by another I²C host in a multi-host setup, the bus becomes BUSY (0b11). The bus will re-enter IDLE either when a Stop condition is detected, or when a time-out occurs (inactive bus time-out needs to be configured).

If a Start condition is generated internally by writing the Address bit group in the Address register (ADDR.ADDR) while IDLE, the OWNER state (0b10) is entered. If the complete transaction was performed without interference, i.e., arbitration was not lost, the I²C host can issue a Stop condition, which will change the Bus state back to IDLE.

However, if a packet collision is detected while in OWNER state, the arbitration is assumed lost and the Bus state becomes BUSY until a Stop condition is detected. A repeated Start condition will change the Bus state only if arbitration is lost while issuing a repeated start.

Note: Violating the protocol may cause the I²C to hang. If this happens it is possible to recover from this state by a software Reset (CTRLA.SWRST='1').

32.6.2.4 I²C Host Operation

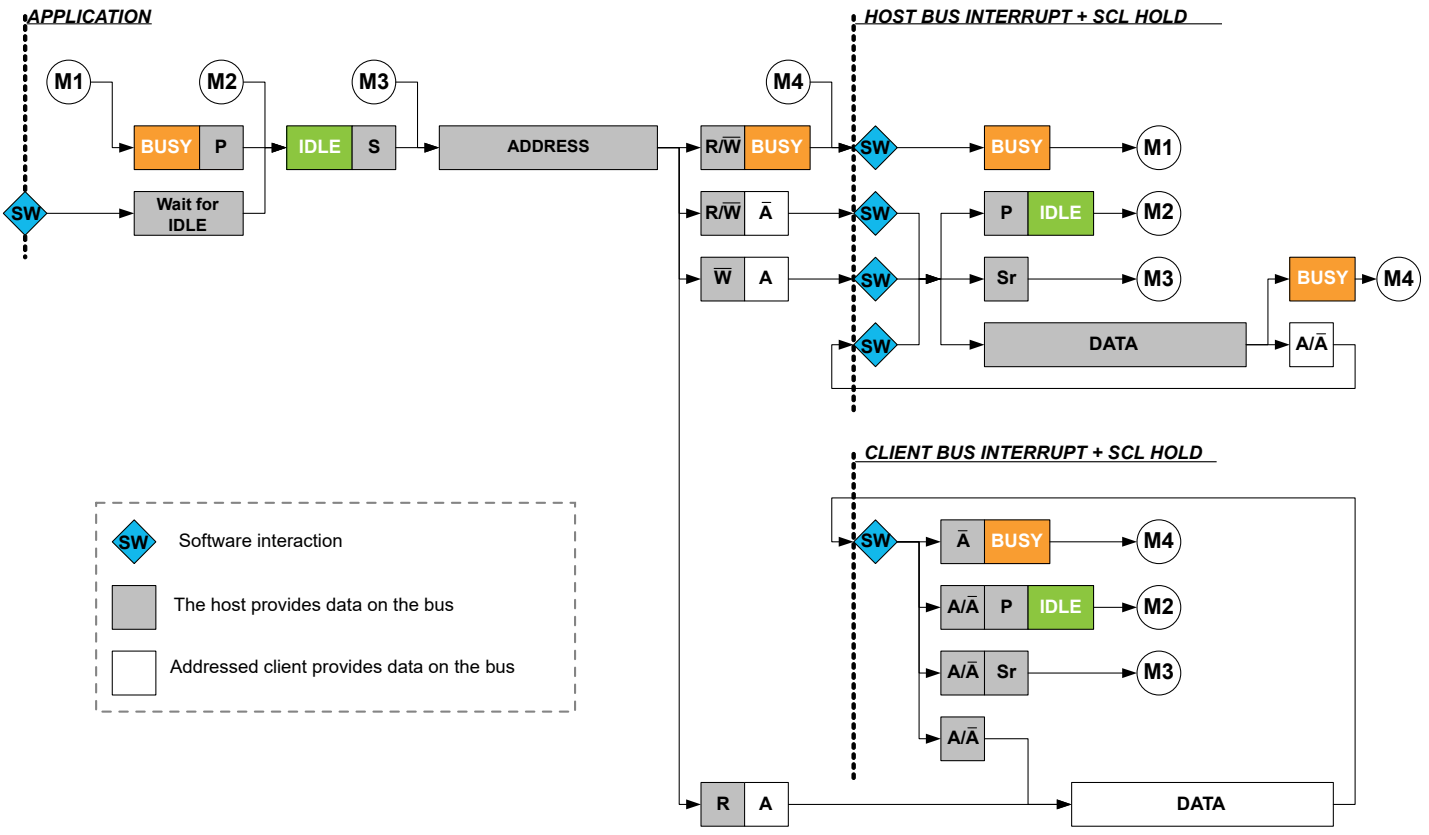
The I²C host is byte-oriented and interrupt based. The number of interrupts generated is kept at a minimum by automatic handling of most incidents. The software driver complexity and code size are reduced by auto-triggering of operations, and a Special Smart mode, which can be enabled by the Smart Mode Enable bit in the Control A register (CTRLA.SMEN).

The I²C host has two interrupt strategies.

When SCL Stretch Mode (CTRLA.SCLSM) is '0', SCL is stretched before or after the Acknowledge bit . In this mode the I²C host operates according to *I²C Host Behavioral Diagram (SCLSM=0)* as shown in the following figure. The circles labeled "M*n*" (M1, M2..) indicate the nodes the bus logic can jump to, based on software or hardware interaction.

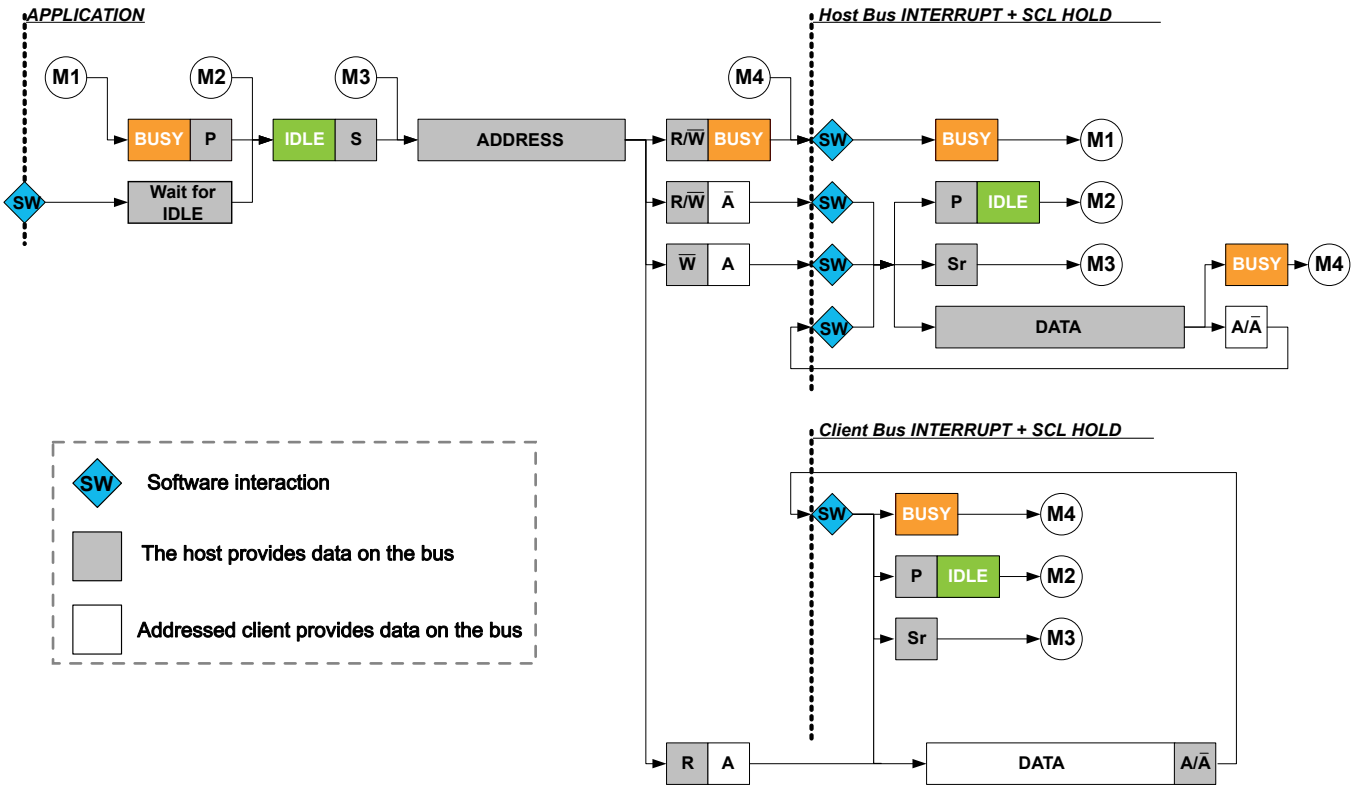
This diagram is used as reference for the description of the I²C host operation throughout the document.

Figure 32-5. I²C Host Behavioral Diagram (SCLSM=0)



In the second strategy (CTRLA.SCLSM=1), interrupts only occur after the ACK bit, as in *Host Behavioral Diagram (SCLSM=1)* shown in the following figure. This strategy can be used when it is not necessary to check DATA before acknowledging.

Figure 32-6. I²C Host Behavioral Diagram (SCLSM=1)



32.6.2.4.1 Host Clock Generation

The SERCOM peripheral supports several I²C bidirectional modes:

- Standard mode (*Sm*) up to 100 kHz
- Fast mode (*Fm*) up to 400 kHz
- Fast mode Plus (*Fm+*) up to 1 MHz

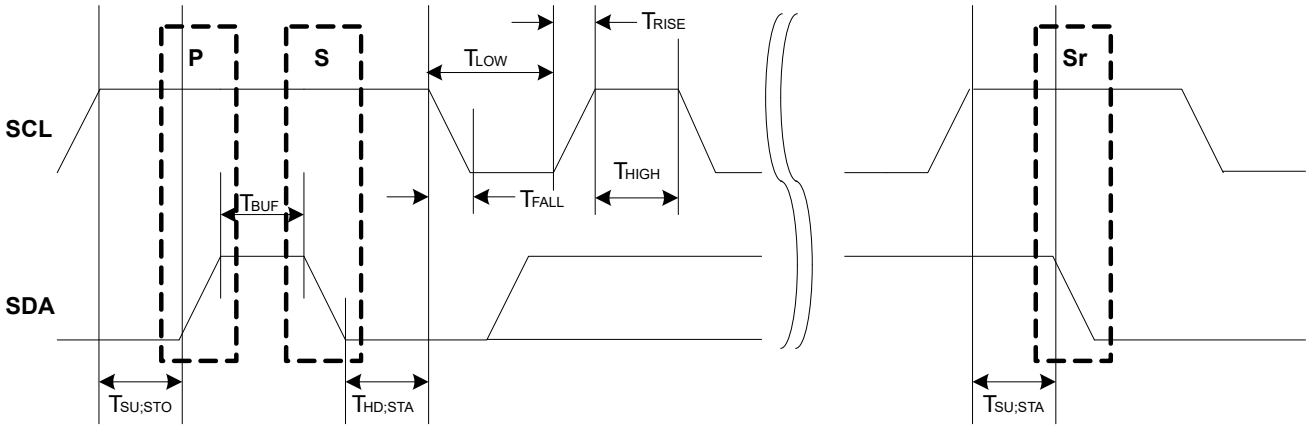
The Host clock configuration for *Sm, Fm* and *Fm+* are described in *Clock Generation (Standard-Mode, Fast-Mode and Fast-Mode Plus)* as follows.

Clock Generation (Standard-Mode, Fast-Mode, and Fast-Mode Plus)

In I²C *Sm, Fm*, and *Fm+* mode, the Host clock (SCL) frequency is determined as described in this section:

The low (T_{LOW}) and high (T_{HIGH}) times are determined by the Baud Rate register (BAUD), while the rise (T_{RISE}) and fall (T_{FALL}) times are determined by the bus topology. Because of the wired-AND logic of the bus, T_{FALL} will be considered as part of T_{LOW}. Likewise, T_{RISE} will be in a state between T_{LOW} and T_{HIGH} until a high state has been detected.

Figure 32-7. SCL Timing



The following parameters are timed using the SCL low time period T_{LOW}. This comes from the Host Baud Rate Low bit group in the Baud Rate register (BAUD.BAUDLOW). When BAUD.BAUDLOW=0, or the Host Baud Rate bit group in the Baud Rate register (BAUD.BAUD) determines it.

- T_{LOW} – Low period of SCL clock
- $T_{SU;STO}$ – Set-up time for stop condition
- T_{BUF} – Bus free time between stop and start conditions
- $T_{HD;STA}$ – Hold time (repeated) start condition
- $T_{SU;STA}$ – Set-up time for repeated start condition
- T_{HIGH} is timed using the SCL high time count from BAUD.BAUD
- T_{RISE} is determined by the bus impedance; for internal pull-ups.
- T_{FALL} is determined by the open-drain current limit and bus impedance; can typically be regarded as zero.

The SCL frequency is given by:

$$f_{\mathrm{SCL}} = \frac{1}{T_{\mathrm{LOW}} + T_{\mathrm{HIGH}} + T_{\mathrm{RISE}}}$$

When BAUD.BAUDLOW is zero, the BAUD.BAUD value is used to time both SCL high and SCL low. In this case the following formula will give the SCL frequency:

$$f_{\mathrm{SCL}} = \frac{f_{\mathrm{GCLK}}}{10 + 2BAUD + f_{\mathrm{GCLK}} \cdot T_{\mathrm{RISE}}}$$

When BAUD.BAUDLOW is non-zero, the following formula determines the SCL frequency:

$$f_{\mathrm{SCL}} = \frac{f_{\mathrm{GCLK}}}{10 + BAUD + BAUDLOW + f_{\mathrm{GCLK}} \cdot T_{\mathrm{RISE}}}$$

The following formulas can determine the SCL T_{LOW} and T_{HIGH} times:

$$T_{\mathrm{LOW}} = \frac{BAUDLOW + 5}{f_{\mathrm{GCLK}}}$$

$$T_{\mathrm{HIGH}} = \frac{BAUD + 5}{f_{\mathrm{GCLK}}}$$

Note: The I²C standard *Fm+* (Fast-mode plus) requires a nominal high to low SCL ratio of 1:2, and BAUD must be set accordingly. At a minimum, BAUD.BAUD and/or BAUD.BAUDLOW must be non-zero.

Start-up Timing: The minimum time between SDA transition and SCL rising edge is 6 APB cycles when the DATA register is written in smart mode. If a greater start-up time is required due to long rise times, the time between DATA write and IF clear must be controlled by software.
Note: When timing is controlled by user, the Smart Mode cannot be enabled.

32.6.2.4.2 Transmitting Address Packets

The I^2C host starts a bus transaction by writing the I^2C client address to ADDR.ADDR and the direction bit, as described in Principle of Operation, see *Principle of Operation* from Related Links. If the bus is busy, the I^2C host will wait until the bus becomes idle before continuing the operation. When the bus is idle, the I^2C host will issue a start condition on the bus. The I^2C host will then transmit an address packet using the address written to ADDR.ADDR. After the address packet has been transmitted by the I^2C host, one of four cases will arise according to arbitration and transfer direction.

Case 1: Arbitration lost or bus error during address packet transmission

If arbitration was lost during transmission of the address packet, the Host on Bus bit in the Interrupt Flag Status and Clear register (INTFLAG.MB) and the Arbitration Lost bit in the Status register (STATUS.ARBLOST) are both set. Serial data output to SDA is disabled, and the SCL is released, which disables clock stretching. In effect the I^2C host is no longer allowed to execute any operation on the bus until the bus is idle again. A bus error will behave similarly to the Arbitration Lost condition. In this case, the MB Interrupt flag and Host Bus Error bit in the Status register (STATUS.BUSERR) are both set in addition to STATUS.ARBLOST.

The Host Received Not Acknowledge bit in the Status register (STATUS.RXNACK) will always contain the last successfully received acknowledge or not acknowledge indication.

In this case, software will typically inform the application code of the condition and then clear the Interrupt flag before exiting the interrupt routine. No other flags have to be cleared at this moment, because all flags will be cleared automatically the next time the ADDR.ADDR register is written.

Case 2: Address packet transmit complete – No ACK received

If there is no I^2C client device responding to the address packet, then the INTFLAG.MB Interrupt flag and STATUS.RXNACK will be set. The clock hold is active at this point, preventing further activity on the bus.

The missing ACK response can indicate that the I^2C client is busy with other tasks or sleeping. Therefore, it is not able to respond. In this event, the next step can be either issuing a Stop condition (recommended) or resending the address packet by a repeated Start condition. When using SMBus logic, the client must ACK the address. If there is no response, it means that the client is not available on the bus.

Case 3: Address packet transmit complete – Write packet, Host on Bus set

If the I^2C host receives an acknowledge response from the I^2C client, INTFLAG.MB will be set and STATUS.RXNACK will be cleared. The clock hold is active at this point, preventing further activity on the bus.

In this case, the software implementation becomes highly protocol dependent. Three possible actions can enable the I^2C operation to continue:

- Initiate a data transmit operation by writing the data byte to be transmitted into DATA.DATA.
- Transmit a new address packet by writing ADDR.ADDR. A repeated Start condition will automatically be inserted before the address packet.
- Issue a Stop condition, consequently terminating the transaction.

Case 4: Address packet transmit complete – Read packet, Client on Bus set

If the I^2C host receives an ACK from the I^2C client, the I^2C host proceeds to receive the next byte of data from the I^2C client. When the first data byte is received, the Client on Bus bit in the Interrupt

Flag register (INTFLAG.SB) will be set and STATUS.RXNACK will be cleared. The clock hold is active at this point, preventing further activity on the bus.

In this case, the software implementation becomes highly protocol dependent. Three possible actions can enable the I²C operation to continue:

- Let the I²C host continue to read data by acknowledging the data received. ACK can be sent by software, or automatically in Smart mode.
- Transmit a new address packet.
- Terminate the transaction by issuing a Stop condition.

Note: An ACK or NACK will be automatically transmitted if Smart mode is enabled. The Acknowledge Action bit in the Control B register (CTRLB.ACKACT) determines whether ACK or NACK must be sent.

Related Links

32.6.1. Principle of Operation

32.6.2.4.3 Transmitting Data Packets

When an address packet with direction Host Write (see Figure 32-3) was transmitted successfully , INTFLAG.MB will be set. The I²C host will start transmitting data via the I²C bus by writing to DATA.DATA, and monitor continuously for packet collisions.

If a collision is detected, the I²C host will lose arbitration and STATUS.ARBLOST will be set. If the transmit was successful, the I²C host will receive an ACK bit from the I²C client, and STATUS.RXNACK will be cleared. INTFLAG.MB will be set in both cases, regardless of arbitration outcome.

It is recommended to read STATUS.ARBLOST and handle the arbitration lost condition in the beginning of the I²C Host on Bus interrupt. This can be done as there is no difference between handling address and data packet arbitration.

STATUS.RXNACK must be checked for each data packet transmitted before the next data packet transmission can commence. The I²C host is not allowed to continue transmitting data packets if a NACK is received from the I²C client.

32.6.2.4.4 Receiving Data Packets (SCLSM=0)

When INTFLAG.SB is set, the I²C host will already have received one data packet. The I²C host must respond by sending either an ACK or NACK. Sending a NACK may be unsuccessful when arbitration is lost during the transmission. In this case, a lost arbitration will prevent setting INTFLAG.SB. Instead, INTFLAG.MB will indicate a change in arbitration. Handling of lost arbitration is the same as for data bit transmission.

32.6.2.4.5 Receiving Data Packets (SCLSM=1)

When INTFLAG.SB is set, the I²C host will already have received one data packet and transmitted an ACK or NACK, depending on CTRLB.ACKACT. At this point, CTRLB.ACKACT must be set to the correct value for the next ACK bit, and the transaction can continue by reading DATA and issuing a command if not in the Smart mode.

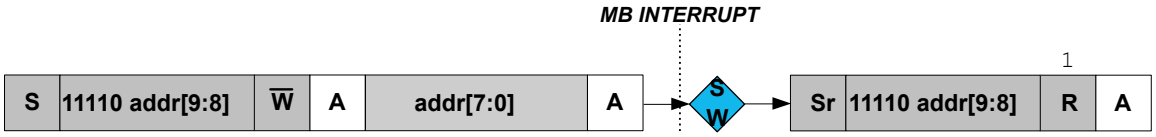
32.6.2.4.6 10-Bit Addressing

When 10-bit addressing is enabled by the Ten Bit Addressing Enable bit in the Address register (ADDR.TENBITEN=1) and the Address bit field ADDR.ADDR is written, the two address bytes will be transmitted, see 10-bit Address Transmission for a Read Transaction. The addressed client acknowledges the two address bytes, and the transaction continues. Regardless of whether the transaction is a read or write, the host must start by sending the 10-bit address with the direction bit (ADDR.ADDR[0]) being zero.

If the host receives a NACK after the first byte, the Write Interrupt flag will be raised and the STATUS.RXNACK bit will be set. If the first byte is acknowledged by one or more clients, then the host will proceed to transmit the second address byte and the host will first see the Write Interrupt flag after the second byte is transmitted. If the transaction direction is read-from-client, the 10-bit

address transmission must be followed by a repeated start and the first 7 bits of the address with the read/write bit equal to '1'.

Figure 32-8. 10-bit Address Transmission for a Read Transaction



This implies the following procedure for a 10-bit read operation:

1. Write the 10-bit address to ADDR.ADDR[10:1]. ADDR.TENBITEN must be '1', the direction bit (ADDR.ADDR[0]) must be '0' (can be written simultaneously with ADDR).
2. Once the Host on Bus interrupt is asserted, Write ADDR[7:0] register to '11110 `address[9:8]` 1'. ADDR.TENBITEN must be cleared (can be written simultaneously with ADDR).
3. Proceed to transmit data.

32.6.2.5 I²C Client Operation

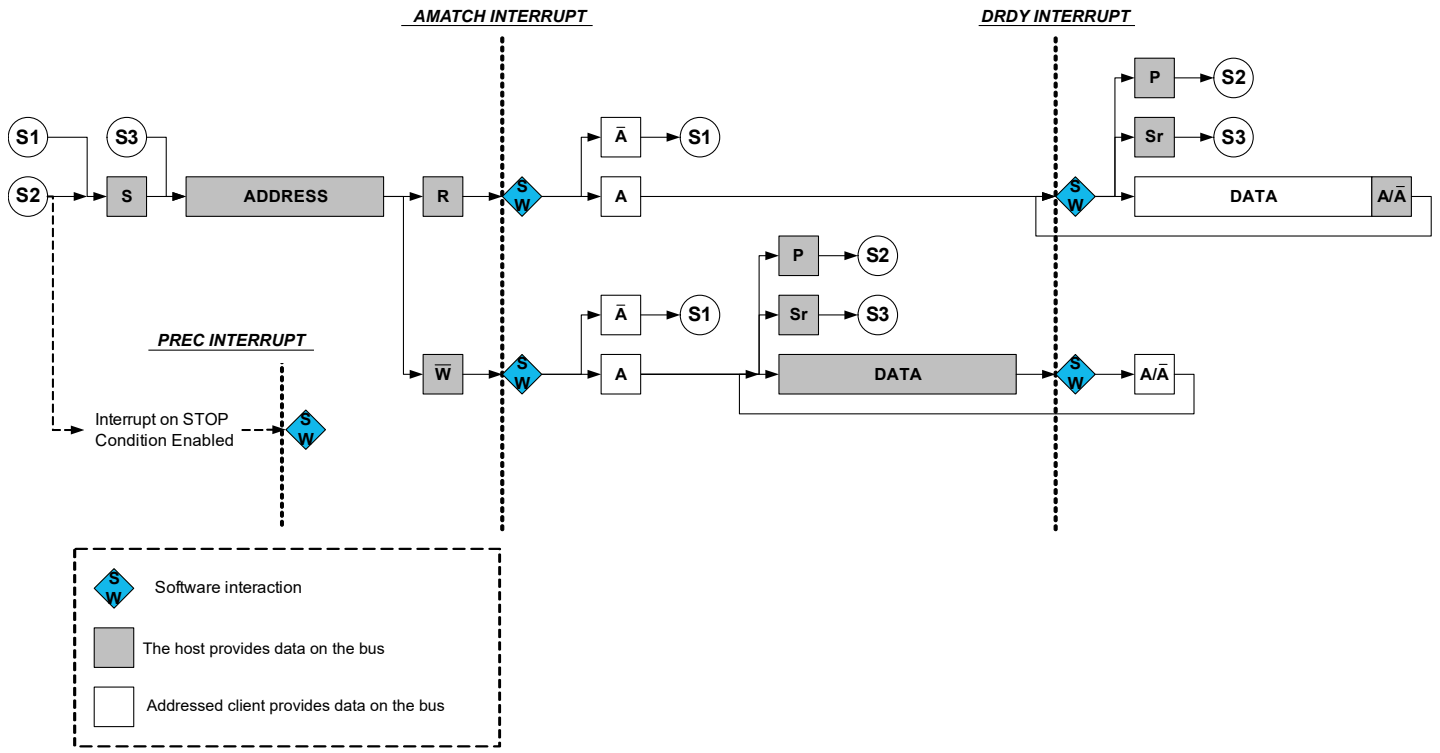
The I²C client is byte-oriented and interrupt-based. The number of interrupts generated is kept at a minimum by automatic handling of most events. The software driver complexity and code size are reduced by auto-triggering of operations, and a special smart mode, which can be enabled by the Smart Mode Enable bit in the Control A register (CTRLA.SMEN).

The I²C client has two interrupt strategies.

When SCL Stretch Mode bit (CTRLA.SCLSM) is '0', SCL is stretched before or after the acknowledge bit. In this mode, the I²C client operates according to *I²C Client Behavioral Diagram (SCLSM=0)* as shown in the following figure. The circles labelled "S*n*" (S1, S2..) indicate the nodes the bus logic can jump to, based on software or hardware interaction.

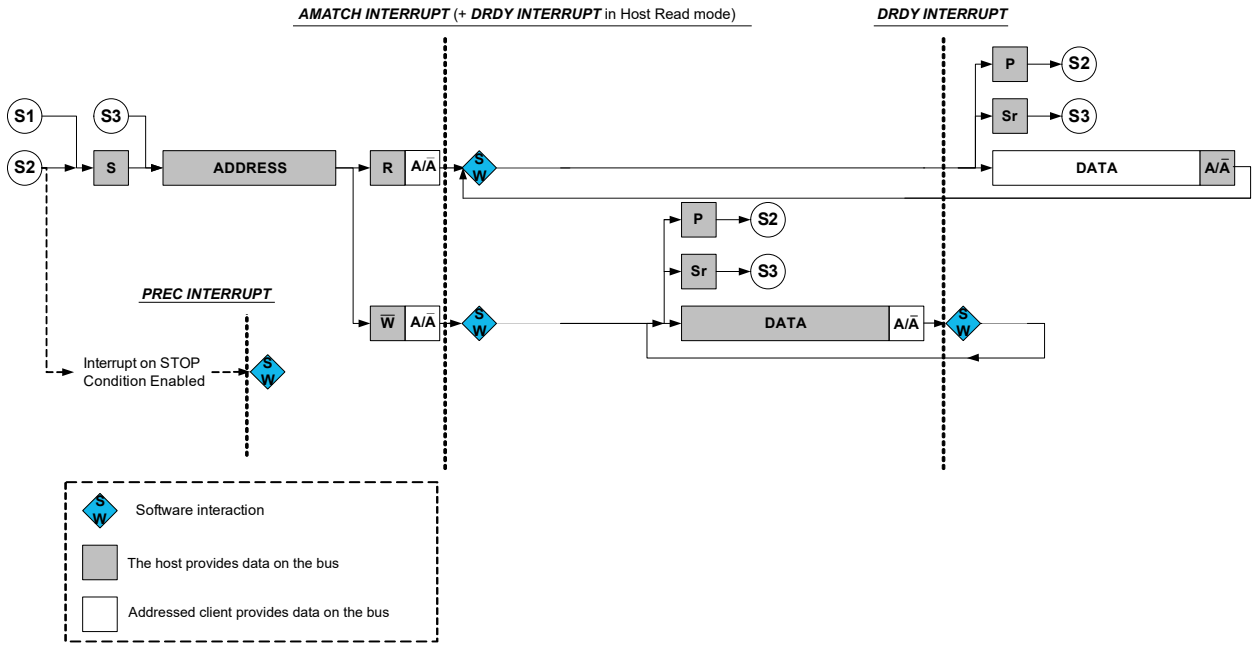
This diagram is used as reference for the description of the I²C client operation throughout the document.

Figure 32-9. I²C Client Behavioral Diagram (SCLSM=0)



In the second strategy (CTRLA.SCLSM=1), interrupts only occur after the ACK bit is sent as shown in the following figure *I²C Client Behavioral Diagram (SCLSM=1)*. This strategy can be used when it is not necessary to check DATA before acknowledging. For host reads, an address and data interrupt will be issued simultaneously after the address acknowledge. However, for host writes, the first data interrupt will be seen after the first data byte has been received by the client and the acknowledge bit has been sent to the host.

Figure 32-10. I²C Client Behavioral Diagram (SCLSM=1)



32.6.2.5.1 Receiving Address Packets (SCLSM=0)

When CTRLA.SCLSM=0, the I²C client stretches the SCL line according to Figure 32-9. When the I²C client is properly configured, it will wait for a Start condition.

When a Start condition is detected, the successive address packet will be received and checked by the address match logic. If the received address is not a match, the packet will be rejected, and the I²C client will wait for a new Start condition. If the received address is a match, the Address Match bit in the Interrupt Flag register (INTFLAG.AMATCH) will be set.

SCL will be stretched until the I²C client clears INTFLAG.AMATCH. As the I²C client holds the clock by forcing SCL low, the software has unlimited time to respond.

The direction of a transaction is determined by reading the Read/Write Direction bit in the Status register (STATUS.DIR). This bit will be updated only when a valid address packet is received.

If the Transmit Collision bit in the Status register (STATUS.COLL) is set, this indicates that the last packet addressed to the I²C client had a packet collision. A collision causes the SDA and SCL lines to be released without any notification to software. Therefore, the next AMATCH interrupt is the first indication of the previous packet's collision. Collisions are intended to follow the SMBus Address Resolution Protocol (ARP).

After the address packet has been received from the I²C host, one of two cases will arise based on transfer direction.

Case 1: Address packet accepted – Read flag set

The STATUS.DIR bit is '1', indicating an I²C host read operation. The SCL line is forced low, stretching the bus clock. If an ACK is sent, I²C client hardware will set the Data Ready bit in the Interrupt Flag register (INTFLAG.DRDY), indicating data are needed for transmit. If a NACK is sent, the I²C client will wait for a new Start condition and address match.

Typically, software will immediately acknowledge the address packet by sending an ACK/NACK bit. The I²C client Command bit field in the Control B register (CTRLB.CMD) can be written to '0x3' for both read and write operations as the command execution is dependent on the STATUS.DIR bit. Writing '1' to INTFLAG.AMATCH will also cause an ACK/NACK to be sent corresponding to the CTRLB.ACKACT bit.

Case 2: Address packet accepted – Write flag set

The STATUS.DIR bit is cleared, indicating an I²C host write operation. The SCL line is forced low, stretching the bus clock. If an ACK is sent, the I²C client will wait for data to be received. Data, repeated start or stop can be received.

If a NACK is sent, the I²C client will wait for a new Start condition and address match. Typically, software will immediately acknowledge the address packet by sending an ACK/NACK. The I²C client command CTRLB.CMD = 3 can be used for both read and write operation as the command execution is dependent on STATUS.DIR.

Writing '1' to INTFLAG.AMATCH will also cause an ACK/NACK to be sent corresponding to the CTRLB.ACKACT bit.

32.6.2.5.2 Receiving Address Packets (SCLSM=1)

When SCLSM=1, the I²C client will stretch the SCL line only after an ACK (see Figure 32-10). When the I²C client is properly configured, it will wait for a Start condition to be detected.

When a Start condition is detected, the successive address packet will be received and checked by the address match logic.

If the received address is not a match, the packet will be rejected and the I²C client will wait for a new Start condition.

If the address matches, the acknowledge action as configured by the Acknowledge Action bit Control B register (CTRLB.ACKACT) will be sent and the Address Match bit in the Interrupt Flag register

(INTFLAG.AMATCH) is set. SCL will be stretched until the I²C client clears INTFLAG.AMATCH. As the I²C client holds the clock by forcing SCL low, the software is given unlimited time to respond to the address.

The direction of a transaction is determined by reading the Read/Write Direction bit in the Status register (STATUS.DIR). This bit will be updated only when a valid address packet is received.

If the Transmit Collision bit in the Status register (STATUS.COLL) is set, the last packet addressed to the I²C client had a packet collision. A collision causes the SDA and SCL lines to be released without any notification to software. The next AMATCH interrupt is, therefore, the first indication of the previous packet's collision. Collisions are intended to follow the SMBus Address Resolution Protocol (*ARP*).

After the address packet has been received from the I²C host, INTFLAG.AMATCH can be set to '1' to clear it.

32.6.2.5.3 Receiving and Transmitting Data Packets

After the I²C client has received an address packet, it will respond according to the direction either by waiting for the data packet to be received or by starting to send a data packet by writing to DATA.DATA. When a data packet is received or sent, INTFLAG.DRDY will be set. After receiving data, the I²C client will send an acknowledge according to CTRLB.ACKACT.

Case 1: Data received

INTFLAG.DRDY is set, and SCL is held low, pending for SW interaction.

Case 2: Data sent

When a byte transmission is successfully completed, the INTFLAG.DRDY Interrupt flag is set. If NACK is received, indicated by STATUS.RXNACK=1, the I²C client must expect a stop or a repeated start to be received. The I²C client must release the data line to allow the I²C host to generate a stop or repeated start. Upon detecting a Stop condition, the Stop Received bit in the Interrupt Flag register (INTFLAG.PREC) will be set and the I²C client will return to IDLE state.

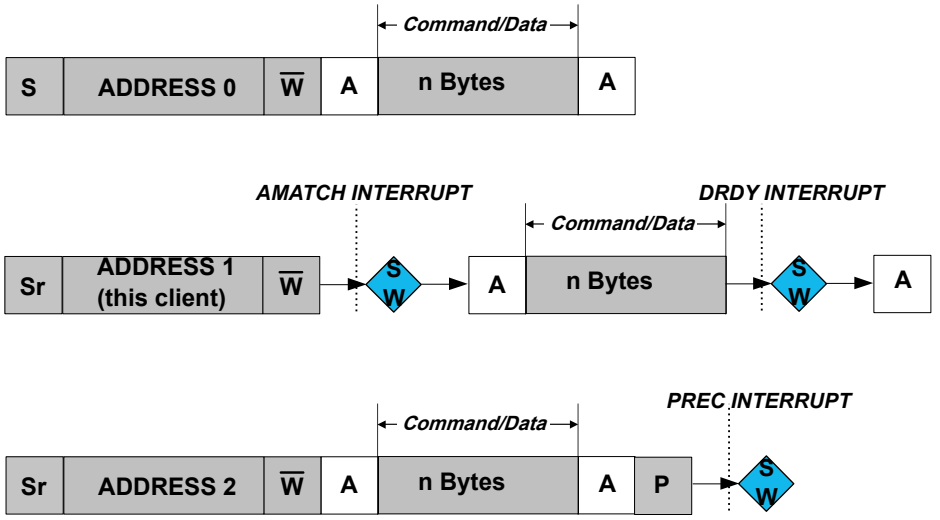
32.6.2.5.4 PMBus Group Command

When the PMBus Group Command bit in the CTRLB register is set (CTRLB.GCMD=1) and 7-bit addressing is used, INTFLAG.PREC will be set if the client has been addressed since the last STOP condition. When CTRLB.GCMD=0, a STOP condition without address match will not set INTFLAG.PREC.

The group command protocol is used to send commands to more than one device. The commands are sent in one continuous transmission with a single STOP condition at the end. When the STOP condition is detected by the clients addressed during the group command, they all begin executing the command they received.

The following figure shows an example where this client, bearing ADDRESS 1, is addressed after a repeated START condition. There can be multiple clients addressed before and after this client. Eventually, at the end of the group command, a single STOP is generated by the host. At this point a STOP interrupt is asserted.

Figure 32-11. PMBus Group Command Example



32.6.3 Additional Features

32.6.3.1 SMBus

The I^{2}C includes three hardware SCL low time-outs which allow a time-out to occur for SMBus SCL low time-out, host extend time-out, and client extend time-out. This allows for SMBus functionality These time-outs are driven by the GCLK_SERCOM_SLOW clock. The GCLK_SERCOM_SLOW clock is used to accurately time the time-out and must be configured to use a 32.768 kHz oscillator. The I^{2}C interface also allows for a SMBus compatible SDA hold time.

- **$T_{TIMEOUT}$:** SCL low time of 25..35ms – Measured for a single SCL low period. It is enabled by CTRLA.LOWTOUTEN
- **$T_{LOW:SEXT}$:** Cumulative clock low extend time of 25 ms – Measured as the cumulative SCL low extend time by a client device in a single message from the initial START to the STOP. It is enabled by CTRLA.SEXTTOEN.
- **$T_{LOW:MEXT}$:** Cumulative clock low extend time of 10 ms – Measured as the cumulative SCL low extend time by the host device within a single byte from START-to-ACK, ACK-to-ACK, or ACK-to-STOP. It is enabled by CTRLA.MEXTTOEN.

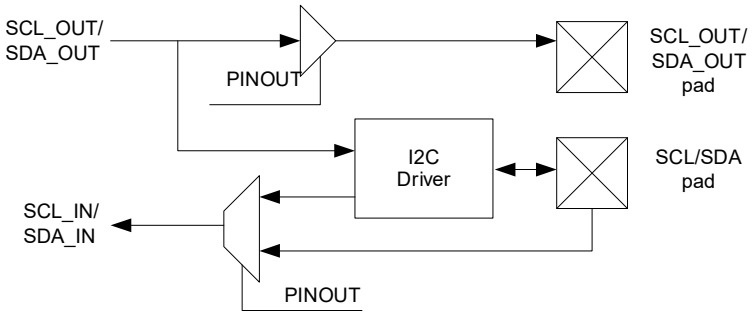
32.6.3.2 Smart Mode

The I^{2}C interface has a Smart mode that simplifies application code and minimizes the user interaction needed to adhere to the I^{2}C protocol. The Smart mode accomplishes this by automatically issuing an ACK or NACK (based on the content of CTRLB.ACKACT) as soon as DATA.DATA is read.

32.6.3.3 4-Wire Mode

Writing a '1' to the Pin Usage bit in the Control A register (CTRLA.PINOUT) will enable 4-Wire mode operation. In this mode, the internal I^{2}C tri-state drivers are bypassed, and an external I^{2}C compliant tri-state driver is needed when connecting to an I^{2}C bus.

Figure 32-12. I²C Pad Interface



32.6.3.4 Quick Command

Setting the Quick Command Enable bit in the Control B register (CTRLB.QCEN) enables quick command. When quick command is enabled, the corresponding Interrupt flag (INTFLAG.SB or INTFLAG.MB) is set immediately after the client acknowledges the address. At this point, the software can either issue a Stop command or a repeated start by writing CTRLB.CMD or ADDR.ADDR.

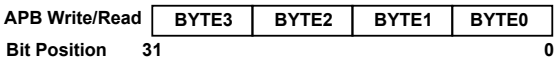
32.6.3.5 32-bit Extension

For better system bus utilization, 32-bit data receive and transmit can be enabled by writing to the Data 32-bit bit field in the Control C register (CTRLC.DATA32B=1). When enabled, write and read transaction to/from the DATA register are 32 bit in size.

If frames are not multiples of 4 Bytes, the Length Counter (LENGTH.LEN) and Length Enable (LENGTH.LENEN) must be configured before data transfer begins. LENGTH.LEN must be enabled only when CTRLC.DATA32B is enabled.

The following figure shows the order of transmit and receive when using 32-bit mode. Bytes are transmitted or received and stored in order from 0 to 3.

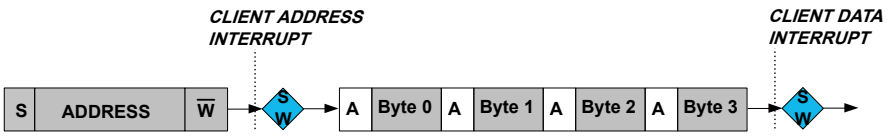
Figure 32-13. 32-bit Extension Byte Ordering



32-bit Extension Client Operation

The following figure shows a transaction with 32-bit Extension enabled (CTRLC.DATA32B=1). In client operation, the Address Match interrupt in the Interrupt Flag Status and Clear register (INTFLAG.AMATCH) is set after the address is received and available in the DATA register. The Data Ready interrupt (INTFLAG.DRDY) will then be raised for every 4 Bytes transferred.

Figure 32-14. 32-bit Extension Client Operation



The LENGTH register can be written before the frame begins, or when the AMATCH interrupt is set. If the frame size is not LENGTH.LEN Bytes, the Length Error status bit (STATUS.LENERR) is raised. If LENGTH.LEN is not a multiple of 4 Bytes, the final INTFLAG.DRDY interrupt is raised when the last Byte is received for host reads. For host writes, the last data byte will be automatically NACKed. On address recognition, the internal length counter is reset in preparation for the incoming frame.

High Speed transactions start with a Full Speed Host Code. When a Host Code is detected, no data is received and the next expected operation is a repeated start. For this reason, the length is not

counted after a Host Code is received. In this case, no Length Error (STATUS.LENERR) is registered, regardless of the LENGTH.LENEN setting.

When SCL clock stretch mode is selected (CTRLA.SCLSM=1) and the transaction is a host write, the selected Acknowledge Action (CTRLB.ACKACT) will only be used to ACK/NACK each 4th byte. All other bytes are ACKed. This allows the user to write CTRLB.ACKACT=1 in the final interrupt, so that the last byte in a 32-bit word will be NACKed.

Writing to the LENGTH register while a frame is in progress will produce unpredictable results. If LENGTH.LENEN is not set and a frame is not a multiple of 4 Bytes, the remainder will be lost.

32-bit Extension Host Operation

When using the I^2C configured as Host, the Address register must be written with the desired address (ADDR.ADDR), and optionally, the transaction Length and transaction Length Enable bits (ADDR.LEN and ADDR.LENEN) can be written. When ADDR.LENEN is written to '1' along with ADDR.ADDR, ADDR.LEN determines the number of data bytes in the transaction from 0 to 255. Then, the ADDR.LEN bytes are transferred, followed by an automatically generated NACK (for host reads) and a STOP.

The INTFLAG.SB or INTFLAG.MB are raised for every 4 Bytes transferred. If the transaction is a host read and ADDR.LEN is not a multiple of 4 Bytes, the final INTFLAG.SB is set when the last byte is received.

When SCL clock stretch mode is enabled (CTRLA.SCLSM=1) and the transaction is a host read, the selected Acknowledge Action (CTRLB.ACKACT) will only be used to ACK/NACK each 4th Byte. All other bytes are ACKed. This allows the user to set CTRLB.ACKACT=1 in the final interrupt, so that the last byte in a 32-bit word will be NACKed.

If a NACK is received by the client for a host write transaction before ADDR.LEN bytes, a STOP will be automatically generated, and the length error (STATUS.LENERR) is raised along with the INTFLAG.ERROR interrupt.

32.6.4 DMA, Interrupts and Events

Each interrupt source has its own Interrupt flag. The Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG) will be set when the Interrupt condition is meet. Each interrupt can be individually enabled by writing '1' to the corresponding bit in the Interrupt Enable Set register (INTENSET), and disabled by writing '1' to the corresponding bit in the Interrupt Enable Clear register (INTENCLR). An interrupt request is generated when the Interrupt flag is set and the corresponding interrupt is enabled. The interrupt request is active until the Interrupt flag is cleared, the interrupt is disabled or the I^2C is reset. See the INTFLAG (Client) or INTFLAG (Host) register for details on how to clear Interrupt flags.

Table 32-1. Module Request for SERCOM I^2C Client

Condition	Request		
	DMA	Interrupt	Event
Data needed for transmit (TX) (Client Transmit mode)	Yes (request cleared when data is written)	—	NA
Data received (RX) (Client Receive mode)	Yes (request cleared when data is read)	—	
Data Ready (DRDY)	—	Yes	
Address Match (AMATCH)	—	Yes	
Stop received (PREC)	—	Yes	
Error (ERROR)	—	Yes	

Table 32-2. Module Request for SERCOM I²C Host

Condition	Request		
	DMA	Interrupt	Event
Data needed for transmit (TX) (Host Transmit mode)	Yes (request cleared when data is written)	—	NA
Data needed for transmit (RX) (Host Transmit mode)	Yes (request cleared when data is read)	—	
Host on Bus (MB)	—	Yes	
Stop received (SB)	—	Yes	
Error (ERROR)	—	Yes	

32.6.4.1 DMA Operation

Smart mode must be enabled for DMA operation in the Control B register by writing CTRLB.SMEN=1.

32.6.4.1.1 Client DMA

When using the I²C client with DMA, an address match will cause the address Interrupt flag (INTFLAG.ADDRMATCH) to be raised. After the interrupt has been serviced, data transfer will be performed through DMA.

The I²C client generates the following requests:

32.6.4.1.2 Host DMA

When using the I²C host with DMA, the ADDR register must be written with the desired address (ADDR.ADDR), transaction length (ADDR.LEN), and transaction length enable (ADDR.LENEN). When ADDR.LENEN is written to 1 along with ADDR.ADDR, ADDR.LEN determines the number of data bytes in the transaction from 0 to 255. DMA is then used to transfer ADDR.LEN bytes followed by an automatically generated NACK (for host reads) and a STOP.

If a NACK is received by the client for a host write transaction before ADDR.LEN bytes, a STOP will be automatically generated and the length error (STATUS.LENERR) will be raised along with the INTFLAG.ERROR interrupt.

The I²C host generates the following requests:

32.6.4.2 Interrupts

The I²C client has the following interrupt sources. These are asynchronous interrupts. They can wake-up the device from any Sleep mode:

- Error (ERROR)
- Data Ready (DRDY)
- Address Match (AMATCH)
- Stop Received (PREC)

The I²C host has the following interrupt sources. These are asynchronous interrupts. They can wake-up the device from any Sleep mode:

- Error (ERROR)
- Client on Bus (SB)
- Host on Bus (MB)

Each interrupt source has its own Interrupt flag. The Interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG) will be set when the Interrupt condition is met. Each interrupt can be individually enabled by writing '1' to the corresponding bit in the Interrupt Enable Set register (INTENSET), and disabled by writing '1' to the corresponding bit in the Interrupt Enable Clear register (INTENCLR).

The status of enabled interrupts can be read from either INTENSET or INTENCLR. An interrupt request is generated when the Interrupt flag is set and the corresponding interrupt is enabled. The interrupt request remains active until the Interrupt flag is cleared, the interrupt is disabled or the I^2C is reset. For details on how to clear Interrupt flags, see *INTFLAG* register from Related Links.

The value of INTFLAG indicates which interrupt is executed. Note that interrupts must be globally enabled for interrupt requests. See *Nested Vector Interrupt Controller (NVIC)* from Related Links.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)
32.10.6. INTFLAG

32.6.4.3 Events

Not applicable.

32.6.5 Sleep Mode Operation

I^2C Host Operation

The generic clock (GCLK_SERCOMx_CORE) will continue to run in idle sleep mode. If the Run In Standby bit in the Control A register (CTRLA.RUNSTDBY) is '1', the GLK_SERCOMx_CORE will also run in Standby Sleep mode. Any interrupt can wake-up the device.

If CTRLA.RUNSTDBY=0, the GLK_SERCOMx_CORE will be disabled after any ongoing transaction is finished. Any interrupt can wake-up the device.

I^2C Client Operation

Writing CTRLA.RUNSTDBY=1 will allow the Address Match interrupt to wake-up the device.

When CTRLA.RUNSTDBY=0, all receptions will be dropped.

32.7 Register Summary - I2C Client

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0	RUNSTDBY			MODE[2:0]			ENABLE	SWRST
		15:8								
		23:16	SEXTTOEN		SDAHOLD[1:0]					PINOUT
		31:24		LOWTOUT			SCLSM		SPEED[1:0]	
0x04	CTRLB	7:0								
		15:8	AMODE[1:0]					AACKEN	GCMD	SMEN
		23:16						ACKACT	CMD[1:0]	
		31:24								
0x08	CTRLC	7:0								
		15:8								
		23:16								
		31:24								DATA32B
0x0C ... 0x13	Reserved									
0x14	INTENCLR	7:0	ERROR					DRDY	AMATCH	PREC
0x15	Reserved									
0x16	INTENSET	7:0	ERROR					DRDY	AMATCH	PREC
0x17	Reserved									
0x18	INTFLAG	7:0	ERROR					DRDY	AMATCH	PREC
0x19	Reserved									
0x1A	STATUS	7:0	CLKHOLD	LOWTOUT		SR	DIR	RXNACK	COLL	BUSERR
		15:8					LENERR		SEXTTOUT	
0x1C	SYNCBUSY	7:0							ENABLE	SWRST
		15:8								
		23:16								
		31:24								
0x20 ... 0x23	Reserved									
0x24	ADDR	7:0	ADDR[6:0]							GENCEN
		15:8						ADDR[9:7]		
		23:16	ADDRMASK[6:0]							
		31:24						ADDRMASK[9:7]		
0x28	DATA	7:0	DATA[7:0]							
		15:8	DATA[15:8]							
		23:16	DATA[23:16]							
		31:24	DATA[31:24]							

32.8 Register Description - I2C Client

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register, and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers are optionally write-protected by the Peripheral Access Controller (PAC). Optional PAC write protection is denoted by the “PAC Write-Protection” property in each individual register description.

Some registers are synchronized when read and/or written. Synchronization is denoted by the “Write-Synchronized” or the “Read-Synchronized” property in each individual register description.

Some registers are enable-protected, meaning they can only be written when the peripheral is disabled. Enable-protection is denoted by the “Enable-Protected” property in each individual register description.

32.8.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected, Write-Synchronized

Bit	31	30	29	28	27	26	25	24
		LOWTOUT			SCLSM		SPEED[1:0]	
Access		R/W			R/W		R/W	R/W
Reset		0			0		0	0

Bit	23	22	21	20	19	18	17	16
	SEXTTOEN		SDAHOLD[1:0]					PINOUT
Access	R/W		R/W	R/W				R/W
Reset	0		0	0				0

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY			MODE[2:0]			ENABLE	SWRST
Access	R/W			R/W	R/W	R/W	R/W	R/W
Reset	0			0	0	0	0	0

Bit 30 – LOWTOUT SCL Low Time-Out

This bit enables the SCL low time-out. If SCL is held low for 25 ms-35 ms, the client will release its clock hold, if enabled, and reset the internal state machine. Any interrupt flags set at the time of time-out will remain set.
This bit is not synchronized.

Value	Description
0	Time-out disabled.
1	Time-out enabled.

Bit 27 – SCLSM SCL Clock Stretch Mode

This bit controls when SCL will be stretched for software interaction.
This bit is not synchronized.

Value	Description
0	SCL stretch according to Figure 32-9
1	SCL stretch only after ACK bit according to Figure 32-10

Bits 25:24 – SPEED[1:0] Transfer Speed

These bits define bus speed.
These bits are not synchronized.

Value	Description
0x0	Standard-mode (Sm) up to 100 kHz and Fast-mode (Fm) up to 400 kHz
0x1	Fast-mode Plus (Fm+) up to 1 MHz
0x2	Reserved
0x3	Reserved

Bit 23 – SEXTTOEN Client SCL Low Extend Time-Out

This bit enables the client SCL low extend time-out. If SCL is cumulatively held low for greater than 25 ms from the initial START to a STOP, the client will release its clock hold if enabled and reset the internal state machine. Any interrupt flags set at the time of time-out will remain set. If the address was recognized, PREC will be set when a STOP is received.
This bit is not synchronized.

Value	Description
0	Time-out disabled
1	Time-out enabled

Bits 21:20 – SDAHOLD[1:0] SDA Hold Time

These bits define the SDA hold time with respect to the negative edge of SCL.
These bits are not synchronized.

Value	Name	Description
0x0	DIS	Disabled
0x1	75	50-100ns hold time
0x2	450	300-600ns hold time
0x3	600	400-800ns hold time

Bit 16 – PINOUT Pin Usage

This bit sets the pin usage to either two- or four-wire operation:
This bit is not synchronized.

Value	Description
0	4-wire operation disabled
1	4-wire operation enabled

Bit 7 – RUNSTDBY Run in Standby

This bit defines the functionality in standby sleep mode.
This bit is not synchronized.

Value	Description
0	Disabled – All reception is dropped.
1	Wake on address match, if enabled.

Bits 4:2 – MODE[2:0] Operating Mode

These bits must be written to 0x04 to select the I^2C client serial communication interface of the SERCOM.
These bits are not synchronized.

Bit 1 – ENABLE Enable

Due to synchronization, there is delay from writing CTRLA.ENABLE until the peripheral is enabled/disabled. The value written to CTRL.ENABLE will read back immediately and the Enable Synchronization Busy bit in the Synchronization Busy register (SYNCBUSY.ENABLE) will be set. SYNCBUSY.ENABLE will be cleared when the operation is complete.
This bit is not enable-protected.

Value	Description
0	The peripheral is disabled or being disabled.
1	The peripheral is enabled.

Bit 0 – SWRST Software Reset

Writing '0' to this bit has no effect.
Writing '1' to this bit resets all registers in the SERCOM, except DBGCTRL, to their initial state, and the SERCOM will be disabled.
Writing '1' to CTRLA.SWRST will always take precedence, meaning that all other writes in the same write-operation will be discarded. Any register write access during the ongoing reset will result in an APB error. Reading any register will return the reset value of the register.

Due to synchronization, there is a delay from writing CTRLA.SWRST until the reset is complete. CTRLA.SWRST and SYNCBUSY.SWRST will both be cleared when the reset is complete.
This bit is not enable-protected.
Note: During a SWRST, access to registers/bits without SWRST are disallowed until SYNCBUSY.SWRST cleared by hardware.

Value	Description
0	There is no reset operation ongoing.
1	The reset operation is ongoing.

32.8.2 Control B

Name: CTRLB
Offset: 0x04
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
						ACKACT	CMD[1:0]	
Access						R/W	W	W
Reset						0	0	0

Bit	15	14	13	12	11	10	9	8
	AMODE[1:0]					AACKEN	GCMD	SMEN
Access	R/W	R/W				R/W	R/W	R/W
Reset	0	0				0	0	0

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

Bit 18 – ACKACT Acknowledge Action

This bit defines the client's acknowledge behavior after an address or data byte is received from the host. The acknowledge action is executed when a command is written to the CMD bits. If smart mode is enabled (CTRLB.SMEN=1), the acknowledge action is performed when the DATA register is read.
ACKACT shall not be updated more than once between each peripheral interrupts request.
This bit is not enable-protected.

Value	Description
0	Send ACK
1	Send NACK

Bits 17:16 – CMD[1:0] Command

This bit field triggers the client operation as the below. The CMD bits are strobe bits, and always read as zero. The operation is dependent on the client interrupt flags, INTFLAG.DRDY and INTFLAG.AMATCH, in addition to STATUS.DIR.
All interrupt flags (INTFLAG.DRDY, INTFLAG.AMATCH and INTFLAG.PREC) are automatically cleared when a command is given.
This bit is not enable-protected.

Table 32-3. Command Description

CMD[1:0]	DIR	Action
0x0	X	(No action)
0x1	X	(Reserved)
0x2	Used to complete a transaction in response to a data interrupt (DRDY)	
	0 (Host write)	Execute acknowledge action succeeded by waiting for any start (S/Sr) condition
	1 (Host read)	Wait for any start (S/Sr) condition

.....continued		
CMD[1:0]	DIR	Action
0x3	Used in response to an address interrupt (AMATCH)	
	0 (Host write)	Execute acknowledge action succeeded by reception of next byte
	1 (Host read)	Execute acknowledge action succeeded by client data interrupt
	Used in response to a data interrupt (DRDY)	
	0 (Host write)	Execute acknowledge action succeeded by reception of next byte
	1 (Host read)	Execute a byte read operation followed by ACK/NACK reception

Bits 15:14 – AMODE[1:0] Address Mode

These bits set the addressing mode.

Value	Name	Description
0x0	MASK	The client responds to the address written in ADDR.ADDR masked by the value in ADDR.ADDRMASK.
0x1	2_ADDRS	The client responds to the two unique addresses in ADDR.ADDR and ADDR.ADDRMASK.
0x2	RANGE	The client responds to the range of addresses between and including ADDR.ADDR and ADDR.ADDRMASK. ADDR.ADDR is the upper limit.
0x3	—	Reserved.

Bit 10 – AACKEN Automatic Acknowledge Enable

This bit enables the address to be automatically acknowledged if there is an address match.

Value	Description
0	Automatic acknowledge is disabled.
1	Automatic acknowledge is enabled.

Bit 9 – GCMD PMBus Group Command

This bit enables PMBus group command support. When enabled, the Stop Received interrupt flag (INTFLAG.PREC) will be set when a STOP condition is detected if the client has been addressed since the last STOP condition on the bus.

Value	Description
0	Group command is disabled.
1	Group command is enabled.

Bit 8 – SMEN Smart Mode Enable

When smart mode is enabled, data is acknowledged automatically when DATA.DATA is read.

Value	Description
0	Smart mode is disabled.
1	Smart mode is enabled.

32.8.3 Control C

Name: CTRLC
Offset: 0x08
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
								DATA32B
Access								R/W
Reset								0

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

Bit 24 – DATA32B Data 32 Bit

This bit enables 32-bit data writes and reads to/from the DATA register.

Value	Description
0	Data transaction to/from DATA are 8-bit in size
1	Data transaction to/from DATA are 32-bit in size

32.8.4 Interrupt Enable Clear

Name: INTENCLR
Offset: 0x14
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to disable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Set register (INTENSET).

Bit	7	6	5	4	3	2	1	0
	ERROR					DRDY	AMATCH	PREC
Access	R/W					R/W	R/W	R/W
Reset	0					0	0	0

Bit 7 – ERROR Error Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Error Interrupt Enable bit, which disables the Error interrupt.

Value	Description
0	Error interrupt is disabled.
1	Error interrupt is enabled.

Bit 2 – DRDY Data Ready Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Data Ready bit, which disables the Data Ready interrupt.

Value	Description
0	The Data Ready interrupt is disabled.
1	The Data Ready interrupt is enabled.

Bit 1 – AMATCH Address Match Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Address Match Interrupt Enable bit, which disables the Address Match interrupt.

Value	Description
0	The Address Match interrupt is disabled.
1	The Address Match interrupt is enabled.

Bit 0 – PREC Stop Received Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Stop Received Interrupt Enable bit, which disables the Stop Received interrupt.

Value	Description
0	The Stop Received interrupt is disabled.
1	The Stop Received interrupt is enabled.

32.8.5 Interrupt Enable Set

Name: INTENSET
Offset: 0x16
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to enable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Clear register (INTENCLR).

Bit	7	6	5	4	3	2	1	0
	ERROR					DRDY	AMATCH	PREC
Access	R/W					R/W	R/W	R/W
Reset	0					0	0	0

Bit 7 – ERROR Error Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Error Interrupt Enable bit, which enables the Error interrupt.

Value	Description
0	Error interrupt is disabled.
1	Error interrupt is enabled.

Bit 2 – DRDY Data Ready Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Data Ready bit, which enables the Data Ready interrupt.

Value	Description
0	The Data Ready interrupt is disabled.
1	The Data Ready interrupt is enabled.

Bit 1 – AMATCH Address Match Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Address Match Interrupt Enable bit, which enables the Address Match interrupt.

Value	Description
0	The Address Match interrupt is disabled.
1	The Address Match interrupt is enabled.

Bit 0 – PREC Stop Received Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Stop Received Interrupt Enable bit, which enables the Stop Received interrupt.

Value	Description
0	The Stop Received interrupt is disabled.
1	The Stop Received interrupt is enabled.

32.8.6 Interrupt Flag Status and Clear

Name: INTFLAG
Offset: 0x18
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	ERROR					DRDY	AMATCH	PREC
Access	R/W					R/W	R/W	R/W
Reset	0					0	0	0

Bit 7 – ERROR Error
This bit is set when any error is detected. Errors that will set this flag have corresponding status flags in the STATUS register. The corresponding bits in STATUS are SEXTTOUT, LOWTOUT, COLL and BUSERR.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the flag.

Bit 2 – DRDY Data Ready
This flag is set when a I^2C client byte transmission is successfully completed.
The flag is cleared by hardware when either:
- Writing to the DATA register.
- Reading the DATA register with Smart mode enabled.
- Writing a valid command to the CMD register.

Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Data Ready Interrupt flag.

Bit 1 – AMATCH Address Match
This flag is set when the I^2C client address match logic detects that a valid address has been received.
The flag is cleared by hardware when CTRL.CMD is written.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Address Match Interrupt flag. When cleared, an ACK/NACK will be sent according to CTRLB.ACKACT.

Bit 0 – PREC Stop Received
This flag is set when a Stop condition is detected for a transaction being processed. A Stop condition detected between a bus host and another client will not set this flag, unless the PMBus Group Command is enabled in the Control B register (CTRLB.GCMD=1).
This flag is cleared by hardware after a command is issued on the next address match.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Stop Received Interrupt flag.

32.8.7 Status

Name: STATUS
Offset: 0x1A
Reset: 0x0000
Property: -

Bit	15	14	13	12	11	10	9	8
					LENERR		SEXTTOUT	
Access					R/W		R/W	
Reset					0		0	

Bit	7	6	5	4	3	2	1	0
	CLKHOLD	LOWTOUT		SR	DIR	RXNACK	COLL	BUSERR
Access	R	R/W		R	R	R	R/W	R/W
Reset	0	0		0	0	0	0	0

Bit 11 – LENERR Transaction Length Error
This bit is set when the length counter is enabled (LENGTH.LENEN) and a STOP or repeated START is received before or after the length in LENGTH.LEN is reached.
This bit is cleared automatically if responding to a new start condition with ACK or NACK (write 3 to CTRLB.CMD) or when INTFLAG.AMATCH is cleared.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the status.

Value	Description
0	No length error has occurred.
1	Length error has occurred.

Bit 9 – SEXTTOUT Client SCL Low Extend Time-Out
This bit is set if a client SCL low extend time-out occurs.
This bit is cleared automatically if responding to a new start condition with ACK or NACK (write 3 to CTRLB.CMD) or when INTFLAG.AMATCH is cleared.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the status.

Value	Description
0	No SCL low extend time-out has occurred.
1	SCL low extend time-out has occurred.

Bit 7 – CLKHOLD Clock Hold
The client Clock Hold bit (STATUS.CLKHOLD) is set when the client is holding the SCL line low, stretching the I2C clock. Software must consider this bit a read-only status flag that is set when INTFLAG.DRDY or INTFLAG.AMATCH is set.
This bit is automatically cleared when the corresponding interrupt is also cleared.

Bit 6 – LOWTOUT SCL Low Time-out
This bit is set if an SCL low time-out occurs.
This bit is cleared automatically if responding to a new start condition with ACK or NACK (write 3 to CTRLB.CMD) or when INTFLAG.AMATCH is cleared.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the status.

Value	Description
0	No SCL low time-out has occurred.
1	SCL low time-out has occurred.

Bit 4 – SR Repeated Start

When INTFLAG.AMATCH is raised due to an address match, SR indicates a repeated start or start condition.
This flag is only valid while the INTFLAG.AMATCH flag is one.

Value	Description
0	Start condition on last address match
1	Repeated start condition on last address match

Bit 3 – DIR Read / Write Direction

The Read/Write Direction (STATUS.DIR) bit stores the direction of the last address packet received from a host .

Value	Description
0	Host write operation is in progress.
1	Host read operation is in progress.

Bit 2 – RXNACK Received Not Acknowledge

This bit indicates whether the last data packet sent was acknowledged or not.

Value	Description
0	Host responded with ACK.
1	Host responded with NACK.

Bit 1 – COLL Transmit Collision

If set, the I2C client was not able to transmit a high data or NACK bit, the I2C client will immediately release the SDA and SCL lines and wait for the next packet addressed to it.
This flag is intended for the SMBus address resolution protocol (ARP). A detected collision in non-ARP situations indicates that there has been a protocol violation, and must be treated as a bus error.
Note: This status will not trigger any interrupt, and must be checked by software to verify that the data were sent correctly. This bit is cleared automatically if responding to an address match with an ACK or a NACK (writing 0x3 to CTRLB.CMD), or INTFLAG.AMATCH is cleared.

Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the status.

Value	Description
0	No collision detected on last data byte sent.
1	Collision detected on last data byte sent.

Bit 0 – BUSERR Bus Error

The Bus Error bit (STATUS.BUSERR) indicates that an illegal bus condition has occurred on the bus, regardless of bus ownership. An illegal bus condition is detected if a protocol violating start, repeated start or stop is detected on the I2C bus lines. A start condition directly followed by a stop condition is one example of a protocol violation. If a time-out occurs during a frame, this is also considered a protocol violation, and will set STATUS.BUSERR.
This bit is cleared automatically if responding to an address match with an ACK or a NACK (writing 0x3 to CTRLB.CMD) or INTFLAG.AMATCH is cleared.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the status.

Value	Description
0	No bus error detected.
1	Bus error detected.

32.8.8 Synchronization Busy

Name: SYNCBUSY
Offset: 0x1C
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
							ENABLE	SWRST
Access							R	R
Reset							0	0

Bit 1 – ENABLE SERCOM Enable Synchronization Busy
Enabling and disabling the SERCOM (CTRLA.ENABLE) requires synchronization. Ongoing synchronization is indicated by SYNCBUSY.ENABLE = 1 until synchronization is complete.

Value	Description
0	Enable synchronization is not busy.
1	Enable synchronization is busy.

Bit 0 – SWRST Software Reset Synchronization Busy
Resetting the SERCOM (CTRLA.SWRST) requires synchronization. Ongoing synchronization is indicated by SYNCBUSY.SWRST = 1 until synchronization is complete.
Note: During a SWRST, access to registers/bits without SWRST are disallowed until SYNCBUSY.SWRST cleared by hardware.

Value	Description
0	SWRST synchronization is not busy.
1	SWRST synchronization is busy.

32.8.9 Address

Name: ADDR
Offset: 0x24
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
						ADDRMASK[9:7]		
Access						R/W	R/W	R/W
Reset						0	0	0

Bit	23	22	21	20	19	18	17	16
	ADDRMASK[6:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Reset	0	0	0	0	0	0	0	

Bit	15	14	13	12	11	10	9	8
						ADDR[9:7]		
Access						R/W	R/W	R/W
Reset						0	0	0

Bit	7	6	5	4	3	2	1	0
	ADDR[6:0]							GENCEN
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 26:17 – ADDRMASK[9:0] Address Mask
These bits act as a second address match register, an address mask register or the lower limit of an address range, depending on the CTRLB.AMODE setting.

Bits 10:1 – ADDR[9:0] Address
These bits contain the I^2C client address used by the client address match logic to determine if a host has addressed the client.
When using 7-bit addressing, the client address is represented by ADDR[6:0].
When the address match logic detects a match, INTFLAG.AMATCH is set and STATUS.DIR is updated to indicate whether it is a read or a write transaction.

Bit 0 – GENCEN General Call Address Enable
A general call address is an address consisting of all-zeroes, including the direction bit (host write).

Value	Description
0	General call address recognition disabled.
1	General call address recognition enabled.

32.8.10 Data

Name: DATA
Offset: 0x28
Reset: 0x00000000
Property: Read/Write

Bit	31	30	29	28	27	26	25	24
	DATA[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	DATA[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	DATA[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DATA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – DATA[31:0] Data

The client data register I/O location (DATA.DATA) provides access to the host transmit and receive data buffers. Reading valid data or writing data to be transmitted can be successfully done only when SCL is held low by the client (STATUS.CLKHOLD is set). An exception occurs when reading the last data byte after the stop condition has been received.

Accessing DATA.DATA auto-triggers I^2C bus operations. The operation performed depends on the state of CTRLB.ACKACT, CTRLB.SMEN and the type of access (read/write).

When CTRLC.DATA32B=1, read and write transactions from/to the DATA register are 32 bit in size. Otherwise, reads and writes are 8 bit.

32.9 Register Summary - I2C Host

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0	RUNSTDBY			MODE[2:0]			ENABLE	SWRST
		15:8								
		23:16	SEXTTOEN	MEXTTOEN	SDAHOLD[1:0]					PINOUT
		31:24		LOWTOUT	INACTOUT[1:0]		SCLSM		SPEED[1:0]	
0x04	CTRLB	7:0								
		15:8							QCEN	SMEN
		23:16						ACKACT	CMD[1:0]	
		31:24								
0x08 ... 0x0B	Reserved									
0x0C	BAUD	7:0	BAUD[7:0]							
		15:8	BAUDLOW[7:0]							
		23:16								
		31:24								
0x10 ... 0x13	Reserved									
0x14	INTENCLR	7:0	ERROR						SB	MB
0x15	Reserved									
0x16	INTENSET	7:0	ERROR						SB	MB
0x17	Reserved									
0x18	INTFLAG	7:0	ERROR						SB	MB
0x19	Reserved									
0x1A	STATUS	7:0	CLKHOLD	LOWTOUT	BUSSTATE[1:0]			RXNACK	ARBLOST	BUSERR
		15:8						LENERR	SEXTTOUT	MEXTTOUT
0x1C	SYNCBUSY	7:0						SYSOP	ENABLE	SWRST
		15:8								
		23:16								
		31:24								
0x20 ... 0x23	Reserved									
0x24	ADDR	7:0	ADDR[7:0]							
		15:8	TENBITEN		LENEN			ADDR[10:8]		
		23:16	LEN[7:0]							
		31:24								
0x28	DATA	7:0	DATA[7:0]							
		15:8	DATA[15:8]							
		23:16	DATA[23:16]							
		31:24	DATA[31:24]							
0x2C ... 0x2F	Reserved									
0x30	DBGCTRL	7:0								DBGSTOP

32.10 Register Description – I²C Host

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register, and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers are optionally write-protected by the Peripheral Access Controller (PAC). Optional PAC write protection is denoted by the “PAC Write-Protection” property in each individual register description.

Some registers are synchronized when read and/or written. Synchronization is denoted by the “Write-Synchronized” or the “Read-Synchronized” property in each individual register description.

Some registers are enable-protected, meaning they can only be written when the peripheral is disabled. Enable-protection is denoted by the “Enable-Protected” property in each individual register description.

32.10.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected, Write-Synchronized

Bit	31	30	29	28	27	26	25	24
		LOWTOUT	INACTOUT[1:0]		SCLSM		SPEED[1:0]	
Access		R/W	R/W	R/W	R/W		R/W	R/W
Reset		0	0	0	0		0	0

Bit	23	22	21	20	19	18	17	16
	SEXTTOEN	MEXTTOEN	SDAHOLD[1:0]					PINOUT
Access	R/W	R/W	R/W	R/W				R/W
Reset	0	0	0	0				0

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY			MODE[2:0]			ENABLE	SWRST
Access	R/W			R/W	R/W	R/W	R/W	R/W
Reset	0			0	0	0	0	0

Bit 30 – LOWTOUT SCL Low Time-Out

This bit enables the SCL low time-out. If SCL is held low for 25ms-35ms, the host will release its clock hold, if enabled, and complete the current transaction. A stop condition will automatically be transmitted.
INTFLAG.SB or INTFLAG.MB will be set as normal, but the clock hold will be released. The STATUS.LOWTOUT and STATUS.BUSERR status bits will be set.
This bit is not synchronized.

Value	Description
0	Time-out disabled.
1	Time-out enabled.

Bits 29:28 – INACTOUT[1:0] Inactive Time-Out

If the inactive bus time-out is enabled and the bus is inactive for longer than the time-out setting, the bus state logic will be set to idle. An inactive bus arise when either an I^2C host or client is holding the SCL low.
Enabling this option is necessary for SMBus compatibility, but can also be used in a non-SMBus set-up.
Calculated time-out periods are based on a 100kHz baud rate.
These bits are not synchronized.

Value	Name	Description
0x0	DIS	Disabled
0x1	55US	5-6 SCL cycle time-out (50-60µs)
0x2	105US	10-11 SCL cycle time-out (100-110µs)
0x3	205US	20-21 SCL cycle time-out (200-210µs)

Bit 27 – SCLSM SCL Clock Stretch Mode

This bit controls when SCL will be stretched for software interaction.

This bit is not synchronized.

Value	Description
0	SCL stretch according to Figure 32-5
1	SCL stretch only after ACK bit, Figure 32-6

Bits 25:24 – SPEED[1:0] Transfer Speed

These bits define bus speed.
These bits are not synchronized.

Value	Description
0x0	Standard-mode (Sm) up to 100 kHz and Fast-mode (Fm) up to 400 kHz
0x1	Fast-mode Plus (Fm+) up to 1 MHz
0x2	Reserved
0x3	Reserved

Bit 23 – SEXTTOEN Client SCL Low Extend Time-Out

This bit enables the client SCL low extend time-out. If SCL is cumulatively held low for greater than 25ms from the initial START to a STOP, the host will release its clock hold if enabled, and complete the current transaction. A STOP will automatically be transmitted.
SB or MB will be set as normal, but CLKHOLD will be release. The MEXTTOUT and BUSERR status bits will be set.
This bit is not synchronized.

Value	Description
0	Time-out disabled
1	Time-out enabled

Bit 22 – MEXTTOEN Host SCL Low Extend Time-Out

This bit enables the host SCL low extend time-out. If SCL is cumulatively held low for greater than 10ms from START-to-ACK, ACK-to-ACK, or ACK-to-STOP the host will release its clock hold if enabled, and complete the current transaction. A STOP will automatically be transmitted.
SB or MB will be set as normal, but CLKHOLD will be released. The MEXTTOUT and BUSERR status bits will be set.
This bit is not synchronized.

Value	Description
0	Time-out disabled
1	Time-out enabled

Bits 21:20 – SDAHOLD[1:0] SDA Hold Time

These bits define the SDA hold time with respect to the negative edge of SCL.
These bits are not synchronized.

Value	Name	Description
0x0	DIS	Disabled
0x1	75NS	50-100ns hold time
0x2	450NS	300-600ns hold time
0x3	600NS	400-800ns hold time

Bit 16 – PINOUT Pin Usage

This bit set the pin usage to either two- or four-wire operation:
This bit is not synchronized.

Value	Description
0	4-wire operation disabled.
1	4-wire operation enabled.

Bit 7 – RUNSTDBY Run in Standby

This bit defines the functionality in standby sleep mode.
This bit is not synchronized.

Value	Description
0	GCLK_SERCOMx_CORE is disabled and the I²C host will not operate in standby sleep mode.
1	GCLK_SERCOMx_CORE is enabled in all sleep modes.

Bits 4:2 – MODE[2:0] Operating Mode

These bits must be written to 0x5 to select the I²C host serial communication interface of the SERCOM.
These bits are not synchronized.

Bit 1 – ENABLE Enable

Due to synchronization, there is delay from writing CTRLA.ENABLE until the peripheral is enabled/disabled. The value written to CTRL.ENABLE will read back immediately and the Synchronization Enable Busy bit in the Synchronization Busy register (SYNCBUSY.ENABLE) will be set. SYNCBUSY.ENABLE will be cleared when the operation is complete.
This bit is not enable-protected.

Value	Description
0	The peripheral is disabled or being disabled.
1	The peripheral is enabled.

Bit 0 – SWRST Software Reset

Writing '0' to this bit has no effect.
Writing '1' to this bit resets all registers in the SERCOM, except DBGCTRL, to their initial state, and the SERCOM will be disabled.
Writing '1' to CTRLA.SWRST will always take precedence, meaning that all other writes in the same write-operation will be discarded. Any register write access during the ongoing reset will result in an APB error. Reading any register will return the reset value of the register.
Due to synchronization there is a delay from writing CTRLA.SWRST until the reset is complete. CTRLA.SWRST and SYNCBUSY.SWRST will both be cleared when the reset is complete.
This bit is not enable-protected.
Note: During a SWRST, access to registers/bits without SWRST are disallowed until SYNCBUSY.SWRST cleared by hardware.

Value	Description
0	There is no reset operation ongoing.
1	The reset operation is ongoing.

32.10.2 Control B

Name: CTRLB
Offset: 0x04
Reset: 0x00000000
Property: PAC Write-Protection, Enable-Protected, Write-Synchronized

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
						ACKACT	CMD[1:0]	
Access						R/W	W	W
Reset						0	0	0

Bit	15	14	13	12	11	10	9	8
							QCEN	SMEN
Access							R/W	R/W
Reset							0	0

Bit	7	6	5	4	3	2	1	0
Access								
Reset								

Bit 18 – ACKACT Acknowledge Action

This bit defines the I^2C Host's acknowledge behavior after a data byte is received from the I^2C Client. The acknowledge action is executed when a command is written to CTRLB.CMD, or if Smart mode is enabled (CTRLB.SMEN is written to one), when DATA.DATA is read.
This bit is not enable-protected.
This bit is not write-synchronized.

Value	Description
0	Send ACK.
1	Send NACK.

Bits 17:16 – CMD[1:0] Command

Writing these bits triggers a Host operation as described below. The CMD bits are strobe bits, and always read as zero. The acknowledge action is only valid in Host Read mode. In Host Write mode, a command will only result in a repeated Start or Stop condition. The CTRLB.ACKACT bit and the CMD bits can be written at the same time, and then the acknowledge action will be updated before the command is triggered.
Commands can only be issued when either the Client on Bus Interrupt flag (INTFLAG.SB) or Host on Bus Interrupt flag (INTFLAG.MB) is '1'.
If CMD 0x1 is issued, a repeated start will be issued followed by the transmission of the current address in ADDR.ADDR. If another address is desired, ADDR.ADDR must be written instead of the CMD bits. This will trigger a repeated start followed by transmission of the new address.
Issuing a command will set the System Operation bit in the Synchronization Busy register (SYNCBUSY.SYSOP).

Table 32-4. Command Description

CMD[1:0]	Direction	Action
0x0	X	(No action)

.....continued

CMD[1:0]	Direction	Action
0x1	X	Execute acknowledge action succeeded by repeated Start
0x2	0 (Write)	No operation
	1 (Read)	Execute acknowledge action succeeded by a byte read operation
0x3	X	Execute acknowledge action succeeded by issuing a Stop condition

These bits are not enable-protected.

Bit 9 – QCEN Quick Command Enable

This bit is not write-synchronized.

Value	Description
0	Quick Command is disabled.
1	Quick Command is enabled.

Bit 8 – SMEN Smart Mode Enable

When Smart mode is enabled, acknowledge action is sent when DATA.DATA is read.
This bit is not write-synchronized.

Value	Description
0	Smart mode is disabled.
1	Smart mode is enabled.

32.10.3 Baud Rate

Name: BAUD
Offset: 0x0C
Reset: 0x0000
Property: PAC Write-Protection, Enable-Protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	BAUDLOW[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	BAUD[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 15:8 – BAUDLOW[7:0] Host Baud Rate Low

If this bit field is non-zero, the SCL low time will be described by the value written.
For more details on how to calculate the frequency, see *Clock Generation – Baud-Rate Generator* from Related Links.

Bits 7:0 – BAUD[7:0] Host Baud Rate

This bit field is used to derive the SCL high time if BAUD.BAUDLOW is non-zero. If BAUD.BAUDLOW is zero, BAUD will be used to generate both high and low periods of the SCL.
For more details on how to calculate the frequency, see *Clock Generation – Baud-Rate Generator* from Related Links.

Related Links

29.6.2.3. Clock Generation – Baud-Rate Generator

32.10.4 Interrupt Enable Clear

Name: INTENCLR
Offset: 0x14
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to disable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Set register (INTENSET).

Bit	7	6	5	4	3	2	1	0
	ERROR						SB	MB
Access	R/W						R/W	R/W
Reset	0						0	0

Bit 7 – ERROR Error Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Error Interrupt Enable bit, which disables the Error interrupt.

Value	Description
0	Error interrupt is disabled.
1	Error interrupt is enabled.

Bit 1 – SB Client on Bus Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Client on Bus Interrupt Enable bit, which disables the Client on Bus interrupt.

Value	Description
0	The Client on Bus interrupt is disabled.
1	The Client on Bus interrupt is enabled.

Bit 0 – MB Host on Bus Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the Host on Bus Interrupt Enable bit, which disables the Host on Bus interrupt.

Value	Description
0	The Host on Bus interrupt is disabled.
1	The Host on Bus interrupt is enabled.

32.10.5 Interrupt Enable Set

Name: INTENSET
Offset: 0x16
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to enable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Clear register (INTENCLR).

Bit	7	6	5	4	3	2	1	0
	ERROR						SB	MB
Access	R/W						R/W	R/W
Reset	0						0	0

Bit 7 – ERROR Error Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Error Interrupt Enable bit, which enables the Error interrupt.

Value	Description
0	Error interrupt is disabled.
1	Error interrupt is enabled.

Bit 1 – SB Client on Bus Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Client on Bus Interrupt Enable bit, which enables the Client on Bus interrupt.

Value	Description
0	The Client on Bus interrupt is disabled.
1	The Client on Bus interrupt is enabled.

Bit 0 – MB Host on Bus Interrupt Enable
Writing '0' to this bit has no effect.
Writing '1' to this bit will set the Host on Bus Interrupt Enable bit, which enables the Host on Bus interrupt.

Value	Description
0	The Host on Bus interrupt is disabled.
1	The Host on Bus interrupt is enabled.

32.10.6 Interrupt Flag Status and Clear

Name: INTFLAG
Offset: 0x18
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	ERROR						SB	MB
Access	R/W						R/W	R/W
Reset	0						0	0

Bit 7 – ERROR Error

This flag is cleared by writing '1' to it.
This bit is set when any error is detected. Errors that will set this flag have corresponding status bits in the STATUS register. These status bits are LENERR, SEXTTOUT, MEXTTOUT, LOWTOUT, ARBLOST, and BUSERR.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear the flag.

Bit 1 – SB Client on Bus

The Client on Bus flag (SB) is set when a byte is successfully received in Host Read mode, for example, no arbitration lost or bus error occurred during the operation. When this flag is set, the host forces the SCL line low, stretching the I^2C clock period. The SCL line will be released and SB will be cleared on one of the following actions:

- Writing to ADDR.ADDR
- Writing to DATA.DATA
- Reading DATA.DATA when Smart mode is enabled (CTRLB.SMEN)
- Writing a valid command to CTRLB.CMD

Writing '1' to this bit location will clear the SB flag. The transaction will not continue or be terminated until one of the above actions is performed.
Writing '0' to this bit has no effect.

Bit 0 – MB Host on Bus

This flag is set when a byte is transmitted in Host Write mode. The flag is set regardless of the occurrence of a bus error or an Arbitration Lost condition. MB is also set when arbitration is lost during sending of NACK in Host Read mode, or when issuing a Start condition if the bus state is unknown. When this flag is set and arbitration is not lost, the host forces the SCL line low, stretching the I^2C clock period. The SCL line will be released and MB will be cleared on one of the following actions:

- Writing to ADDR.ADDR
- Writing to DATA.DATA
- Reading DATA.DATA when Smart mode is enabled (CTRLB.SMEN)
- Writing a valid command to CTRLB.CMD

Writing '1' to this bit location will clear the MB flag. The transaction will not continue or be terminated until one of the above actions is performed.
Writing '0' to this bit has no effect.

32.10.7 Status

Name: STATUS
Offset: 0x1A
Reset: 0x0000
Property: Write-Synchronized

Bit	15	14	13	12	11	10	9	8
						LENERR	SEXTTOUT	MEXTTOUT
Access						R/W	R/W	R/W
Reset						0	0	0

Bit	7	6	5	4	3	2	1	0
	CLKHOLD	LOWTOUT	BUSSTATE[1:0]			RXNACK	ARBLOST	BUSERR
Access	R	R/W	R/W	R/W		R	R/W	R/W
Reset	0	0	0	0		0	0	0

Bit 10 – LENERR Transaction Length Error
This bit is set when automatic length is used for a DMA transaction and the client sends a NACK before ADDR.LEN bytes have been written by the host.
Writing '1' to this bit location will clear STATUS.LENERR. This flag is automatically cleared when writing to the ADDR register.
Writing '0' to this bit has no effect.
This bit is not write-synchronized.

Bit 9 – SEXTTOUT Client SCL Low Extend Time-Out
This bit is set if a client SCL low extend time-out occurs.
This bit is automatically cleared when writing to the ADDR register.
Writing '1' to this bit location will clear SEXTTOUT. Normal use of the I^2C interface does not require the SEXTTOUT flag to be cleared by this method.
Writing '0' to this bit has no effect.
This bit is not write-synchronized.

Bit 8 – MEXTTOUT Host SCL Low Extend Time-Out
This bit is set if a Host SCL low time-out occurs.
Writing '1' to this bit location will clear STATUS.MEXTTOUT. This flag is automatically cleared when writing to the ADDR register.
Writing '0' to this bit has no effect.
This bit is not write-synchronized.

Bit 7 – CLKHOLD Clock Hold
This bit is set when the host is holding the SCL line low, stretching the I^2C clock. Software must consider this bit when INTFLAG.SB or INTFLAG.MB is set.
This bit is cleared when the corresponding Interrupt flag is cleared and the next operation is given.
Writing '0' to this bit has no effect.
Writing '1' to this bit has no effect.
This bit is not write-synchronized.

Bit 6 – LOWTOUT SCL Low Time-Out
This bit is set if an SCL low time-out occurs.
Writing '1' to this bit location will clear this bit. This flag is automatically cleared when writing to the ADDR register.
Writing '0' to this bit has no effect.
This bit is not write-synchronized.

Bits 5:4 – BUSSTATE[1:0] Bus State

These bits indicate the current I²C Bus state.
When in UNKNOWN state, writing 0x1 to BUSSTATE forces the bus state into the IDLE state. The bus state cannot be forced into any other state.
Writing BUSSTATE to idle will set SYNCBUSY.SYSOP.

Value	Name	Description
0x0	UNKNOWN	The Bus state is unknown to the I²C host and will wait for a Stop condition to be detected or wait to be forced into an Idle state by software
0x1	IDLE	The Bus state is waiting for a transaction to be initialized
0x2	OWNER	The I²C host is the current owner of the bus
0x3	BUSY	Some other I²C host owns the bus

Bit 2 – RXNACK Received Not Acknowledge

This bit indicates whether the last address or data packet sent was acknowledged or not.
Writing '0' to this bit has no effect.
Writing '1' to this bit has no effect.
This bit is not write-synchronized.

Value	Description
0	Client responded with ACK.
1	Client responded with NACK.

Bit 1 – ARBLOST Arbitration Lost

This bit is set if arbitration is lost while transmitting a high data bit or a NACK bit, or while issuing a Start or Repeated Start condition on the bus. The Host on Bus Interrupt flag (INTFLAG.MB) will be set when STATUS.ARBLOST is set.
Writing the ADDR.ADDR register will automatically clear STATUS.ARBLOST.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear it.
This bit is not write-synchronized.

Bit 0 – BUSERR Bus Error

This bit indicates that an illegal Bus condition has occurred on the bus, regardless of bus ownership. An illegal Bus condition is detected if a protocol violating start, repeated start or stop is detected on the I²C bus lines. A Start condition directly followed by a Stop condition is one example of a protocol violation. If a time-out occurs during a frame, this is also considered a protocol violation, and will set BUSERR.
If the I²C host is the bus owner at the time a bus error occurs, STATUS.ARBLOST and INTFLAG.MB will be set in addition to BUSERR.
Writing the ADDR.ADDR register will automatically clear the BUSERR flag.
Writing '0' to this bit has no effect.
Writing '1' to this bit will clear it.
This bit is not write-synchronized.

32.10.8 Synchronization Busy

Name: SYNCBUSY
Offset: 0x1C
Reset: 0x00000000

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
						SYSOP	ENABLE	SWRST
Access						R	R	R
Reset						0	0	0

Bit 2 – SYSOP System Operation Synchronization Busy

Value	Description
0	System operation synchronization is not busy.
1	System operation synchronization is busy.

Bit 1 – ENABLE SERCOM Enable Synchronization Busy

Enabling and disabling the SERCOM (CTRLA.ENABLE) requires synchronization. When written, the SYNCBUSY.ENABLE bit will be set until synchronization is complete.

Value	Description
0	Enable synchronization is not busy.
1	Enable synchronization is busy.

Bit 0 – SWRST Software Reset Synchronization Busy

Resetting the SERCOM (CTRLA.SWRST) requires synchronization. When written, the SYNCBUSY.SWRST bit will be set until synchronization is complete.

Note: During a SWRST, access to registers/bits without SWRST are disallowed until SYNCBUSY.SWRST cleared by hardware.

Value	Description
0	SWRST synchronization is not busy.
1	SWRST synchronization is busy.

32.10.9 Address

Name: ADDR
Offset: 0x24
Reset: 0x0000
Property: Write-Synchronized

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	LEN[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	TENBITEN		LENEN			ADDR[10:8]		
Access	R/W		R/W			R/W	R/W	R/W
Reset	0		0			0	0	0

Bit	7	6	5	4	3	2	1	0
	ADDR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 23:16 – LEN[7:0] Transaction Length
These bits define the transaction length of a DMA transaction from 0 to 255 bytes. The Transfer Length Enable (LENEN) bit must be written to '1' in order to use DMA.

Bit 15 – TENBITEN Ten Bit Addressing Enable
This bit enables 10-bit addressing. This bit can be written simultaneously with ADDR to indicate a 10-bit or 7-bit address transmission.

Value	Description
0	10-bit addressing disabled.
1	10-bit addressing enabled.

Bit 13 – LENEN Transfer Length Enable

Value	Description
0	Automatic transfer length disabled.
1	Automatic transfer length enabled.

Bits 10:0 – ADDR[10:0] Address
When ADDR is written, the consecutive operation will depend on the bus state:
UNKNOWN: INTFLAG.MB and STATUS.BUSERR are set, and the operation is terminated.
BUSY: The I^2C host will await further operation until the bus becomes IDLE.
IDLE: The I^2C host will issue a start condition followed by the address written in ADDR. If the address is acknowledged, SCL is forced and held low, and STATUS.CLKHOLD and INTFLAG.MB are set.
OWNER: A repeated start sequence will be performed. If the previous transaction was a read, the acknowledge action is sent before the repeated start bus condition is issued on the bus. Writing ADDR to issue a repeated start is performed while INTFLAG.MB or INTFLAG.SB is set.
STATUS.BUSERR, STATUS.ARBLOST, INTFLAG.MB and INTFLAG.SB will be cleared when ADDR is written.

The ADDR register can be read at any time without interfering with ongoing bus activity, as a read access does not trigger the host logic to perform any bus protocol related operations.
The I^2C host control logic uses bit 0 of ADDR as the bus protocol's read/write flag (R/W); 0 for write and 1 for read.

32.10.10 Data

Name: DATA
Offset: 0x28
Reset: 0x00000000
Property: Read/Write

Bit	31	30	29	28	27	26	25	24
	DATA[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	DATA[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	DATA[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DATA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – DATA[31:0] Data

The host data register I/O location (DATA) provides access to the host transmit and receive data buffers. Reading valid data or writing data to be transmitted can be successfully done only when SCL is held low by the host (STATUS.CLKHOLD is set). An exception is reading the last data byte after the stop condition has been sent.

Accessing DATA.DATA auto-triggers I^2C bus operations. The operation performed depends on the state of CTRLB.ACKACT, CTRLB.SMEN and the type of access (read/write).

When CTRLC.DATA32B=1, read and write transactions from/to the DATA register are 32 bit in size. Otherwise, reads and writes are 8 bit.

32.10.11 Debug Control

Name: DBGCTRL
Offset: 0x30
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
								DBGSTOP
Access								R/W
Reset								0

Bit 0 – DBGSTOP Debug Stop Mode
This bit controls functionality when the CPU is halted by an external debugger.

Value	Description
0	The baud-rate generator continues normal operation when the CPU is halted by an external debugger.
1	The baud-rate generator is halted when the CPU is halted by an external debugger.

33. Quad Serial Peripheral Interface (QSPI)

33.1 Overview

The Quad SPI Interface (QSPI) circuit is a synchronous serial data link that provides communication with external devices in Host mode.

The QSPI can be used in “SPI mode” to interface serial peripherals, such as ADCs, DACs, LCD controllers and sensors, or in “Serial Memory Mode” to interface serial Flash memories.

The QSPI allows the system to execute code directly from a serial Flash memory (XIP) without code shadowing to SRAM. The serial Flash memory mapping is seen in the system as other memories (ROM, SRAM, DRAM, embedded Flash memories, etc.,).

With the support of the quad-SPI protocol, the QSPI allows the system to use high performance serial Flash memories which are small and inexpensive, in place of larger and more expensive parallel Flash memories.

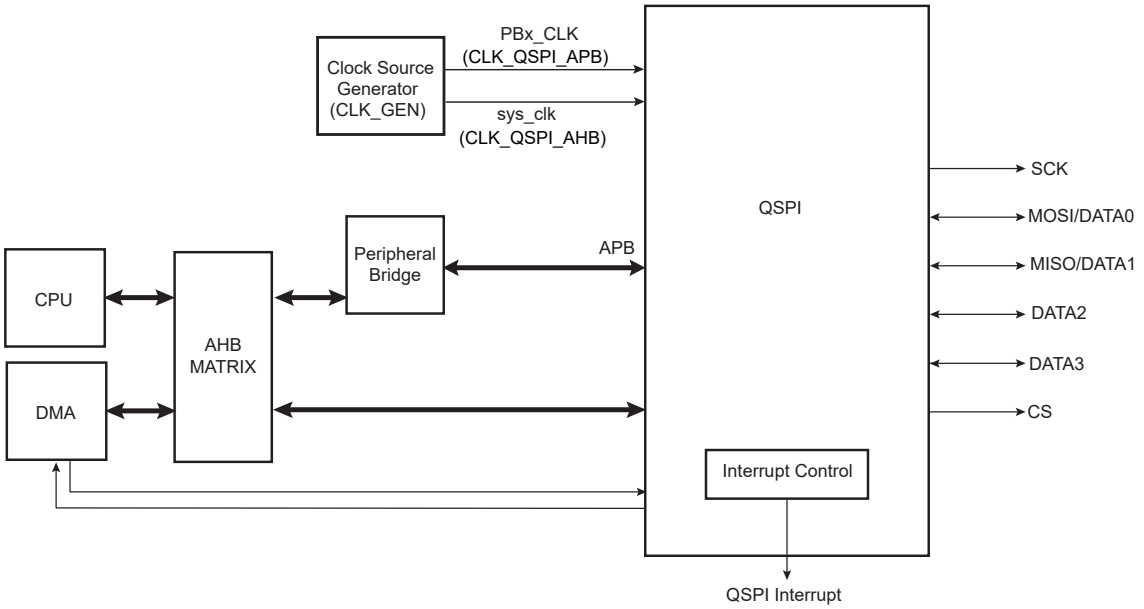
Note: Traditional Quad SPI Interface (QSPI) documentation uses the terminology “Master” and “Slave”. The equivalent Microchip terminology used in this document is “Host” and “Client” respectively.

33.2 Features

- Host SPI Interface:
  - Programmable clock phase and clock polarity
  - Programmable transfer delays between consecutive transfers, between clock and data, between deactivation and activation of chip select (CS)
- SPI Mode:
  - To use serial peripherals, such as ADCs, DACs, LCD controllers, and sensors
  - 8-bit, 16-bit, or 32-bit programmable data length
- Serial Memory Mode:
  - To use serial Flash memories operating in single-bit SPI, Dual SPI and Quad SPI
  - Supports “execute in place” (XIP). The system can execute code directly from a Serial Flash memory
  - Flexible instruction register, to be compatible with all serial Flash memories
  - 32-bit Address mode (default is 24-bit address) to support serial Flash memories larger than 128 Mbit
  - Continuous Read mode
  - Scrambling/Unscrambling “On-the-Fly”
  - Double data rate support
- Connection to DMA Channel Capabilities Optimizes Data Transfers
  - One channel for the receiver and one channel for the transmitter
- Register Write Protection

33.3 Block Diagram

Figure 33-1. QSPI Block Diagram



33.4 Signal Description

Table 33-1. Quad-SPI Signals

Signal	Description	Type
SCK	Serial Clock	Output
CS	Chip Select	Output
MOSI(DATA0)	Data Output (Data Input Output 0)	Output (Input/Output)
MISO(DATA1)	Data Input (Data Input Output 1)	Input (Input/Output)
DATA2	Data Input Output 2	Input/Output
DATA3	Data Input Output 3	Input/Output

Notes:

1. MOSI and MISO are used for single-bit SPI operation.
2. DATA0-DATA1 are used for Dual SPI operation.
3. DATA0-DATA3 are used for Quad SPI operation.

See *I/O Ports and Peripheral Pin Select (PPS)* from Related Links for details on the pin mapping for the QSPI peripheral.

Related Links

6. I/O Ports and Peripheral Pin Select (PPS)

33.5 Product Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

33.5.1 I/O Lines

Using the QSPI I/O lines requires the I/O pins to be configured.

33.5.2 Power Management

The QSPI will continue to operate in any Sleep mode where the selected source clock is running. The QSPI interrupts can be used to wake up the device from sleep modes. See *Power Management Unit (PMU)* from Related Links for details on the different sleep modes.

Related Links

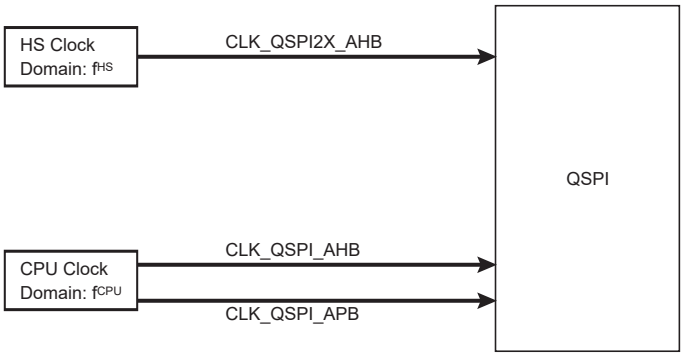
15. Power Management Unit (PMU)

33.5.3 Clocks

An AHB clock (CLK_QSPI_AHB) is required to clock the QSPI. This clock can be enabled and disabled in the CRU.

A FAST clock (CLK_QSPI2X_AHB) is required to clock the QSPI. This clock can be enabled and disabled in the CFGCON1 register, bit 29 (CFGCON1.QSPIDDRM). When using QSPI DDR mode, the System Clock (SYS_CLK) must be <= 48 MHz.

Figure 33-2. QSPI Clock Organization



Important: The CLK_QSPI2x_AHB must be 2 times faster to CLK_QSPI_AHB when the QSPI is operated in DDR mode. In SDR, the CLK_QSPI2x_AHB is not used.

CLK_QSPI_APB, CLK_QSPI_AHB and CLK_QSPI2X_AHB, respectively, are all synchronous but can be divided by a prescaler and may run even when the module clock is turned off.

33.5.4 DMA

The DMA request lines are connected to the DMA Controller (DMAC). Using the QSPI DMA requests requires the DMA Controller to be configured first.

Note: DMAC write access must be 32-bit aligned. If a single byte is to be written in a 32-bit word, the rest of the word must be filled with 'ones'.

33.5.5 Interrupts

The interrupt request lines are connected to the interrupt controller. Using the QSPI interrupts requires the interrupt controller to be configured first. See *Nested Vector Interrupt Controller (NVIC)* from Related Links.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)

33.5.6 Events

Not applicable.

33.5.7 Debug Operation

When the CPU is halted in debug mode the QSPI continues normal operation. If the QSPI is configured in a way that requires it to be periodically serviced by the CPU through interrupts or similar, improper operation or data loss may result during debugging.

33.5.8 Register Access Protection

All registers with write-access are optionally write-protected by the peripheral access controller (PAC), except the following registers:

- Control A (CTRLA) register
- Transmit Data (TXDATA) register
- Interrupt Flag Status and Clear (INTFLAG) register
- Scrambling Key (SCRAMBKEY) register

PAC write-protection is denoted by the 'PAC Write-Protection' property in the register description.

Write-protection does not apply to accesses through an external debugger.

33.6 Functional Description

33.6.1 Principle of Operation

The QSPI is a high-speed synchronous data transfer interface. It allows high-speed communication between the device and peripheral or serial memory devices.

The QSPI operates as a host. It initiates and controls all data transactions.

When transmitting, the TXDATA register can be loaded with the next character to be transmitted during the current transmission.

When receiving, the data is transferred to the RXDATA register, and the receiver is ready for a new character.

33.6.2 Basic Operation

33.6.2.1 Initialization

After Power-On Reset, this peripheral is enabled .

33.6.2.2 Enabling, Disabling and Resetting

The peripheral is enabled by writing a '1' to the Enable bit in the Control A register (CTRLA.ENABLE).

The peripheral is disabled by writing a '0' to CTRLA.ENABLE.

The peripheral is reset by writing a '1' to the Software Reset bit (CTRLA.SWRST).

33.6.3 Transfer Data Rate

By default, the QSPI module is enabled in single data rate mode. In this operating mode, the CLK_QSPI2X_AHB clock is not used and must be disabled.

The dual data rate operating mode is enabled by writing a '1' to the Double Data Rate Enable bit in the CFGCON1 register (CFGCON1.QSPIDDRM). This operating mode requires the CLK_QSPI2X_AHB clock and must be enabled before writing the DDREN bit.

33.6.4 Serial Clock Baud Rate

The QSPI Baud rate clock is generated by dividing the module clock (CLK_QSPI_AHB) by a value between 1 and 255.

This allows a maximum operating baud rate at up to Host Clock and a minimum operating baud rate of CLK_QSPI_AHB divided by 255.

33.6.5 Serial Clock Phase and Polarity

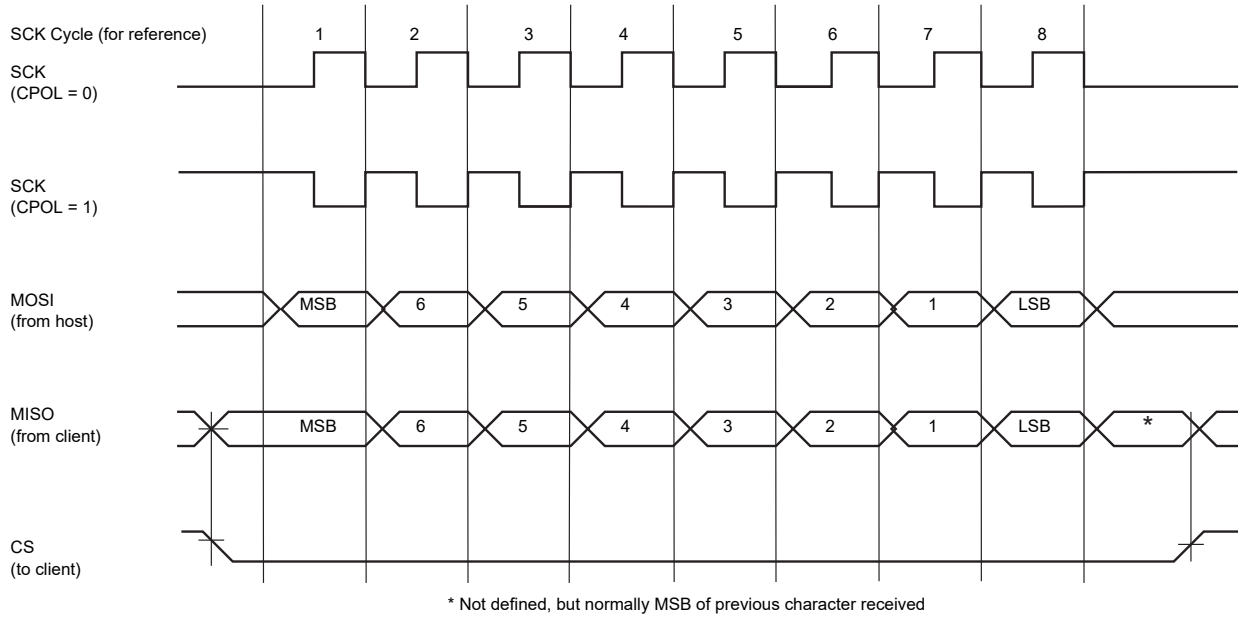
Four combinations of polarity and phase are available for data transfers. Writing the Clock Polarity bit in the QSPI Baud register (BAUD.CPOL) selects the polarity. The Clock Phase bit in the BAUD register programs the clock phase (BAUD.CPHA). These two parameters determine the edges of the clock signal on which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations

Note: The polarity/phase combinations are incompatible. Thus, the interfaced client must use the same parameter values to communicate.

Table 33-2. SPI Transfer Mode

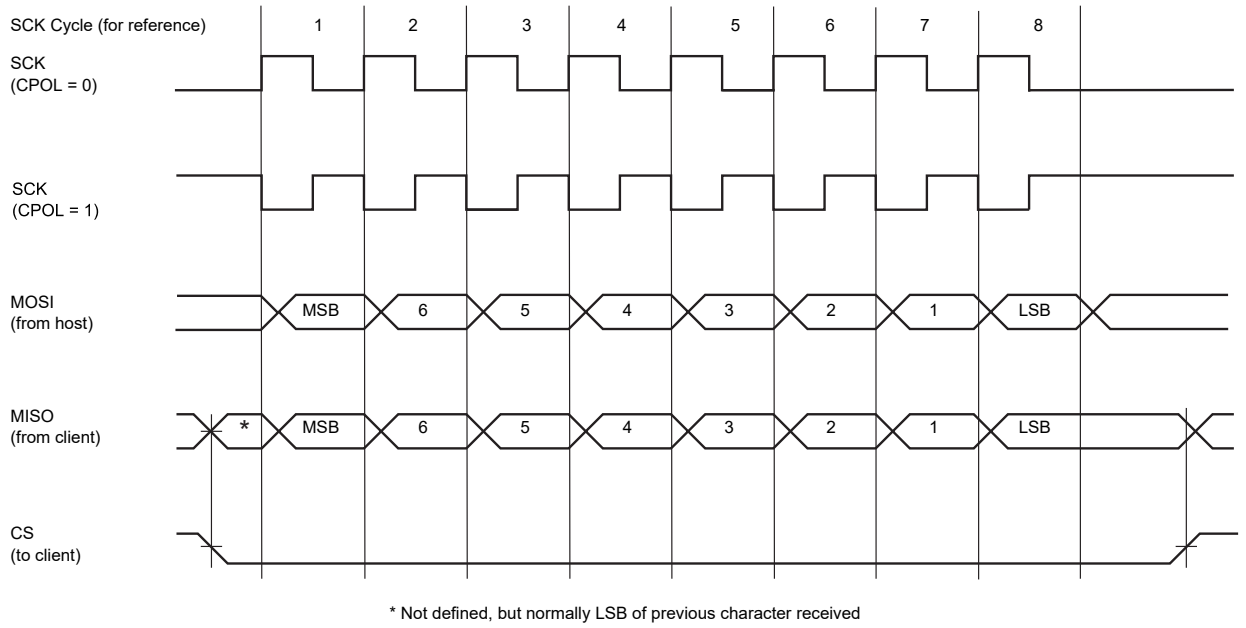
Clock Mode	BAUD.CPOL	BAUD.CPHA	Shift SCK Edge	Capture SCK Edge	SCK Inactive Level
0	0	0	Falling	Rising	Low
1	0	1	Rising	Falling	Low
2	1	0	Rising	Falling	High
3	1	1	Falling	Rising	High

Figure 33-3. QSPI Transfer Modes (BAUD.CPHA = 0, 8-bit transfer)



* Not defined, but normally MSB of previous character received

Figure 33-4. QSPI Transfer Modes (BAUD.CPHA = 1, 8-bit transfer)



* Not defined, but normally LSB of previous character received

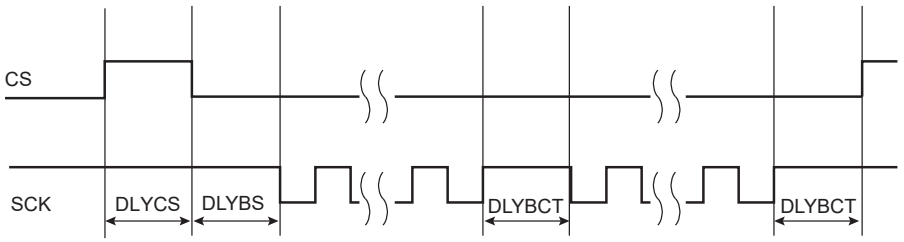
33.6.6 Transfer Delays

The QSPI supports several consecutive transfers while the chip select is active. Three delays can be programmed to modify the transfer waveforms:

- The delay between the inactivation and the activation of CS is programmed by writing the Minimum Inactive CS Delay bit field in the Control B register (CTRLB.DLYCS), allowing to tune the minimum time of CS at high level.
- The delay between consecutive transfers is programmed by writing the Delay Between Consecutive Transfers bit field in the Control B register (CTRLB.DLYBCT), allowing to insert a delay between two consecutive transfers. In Serial Memory mode, this delay is not programmable and DLYBCT settings are ignored.
- The delay before SCK is programmed by writing the Delay Before SCK bit field in the BAUD register (BAUD.DLYBS), allowing to delay the start of SPCK after the chip select has been asserted.

These delays allow the QSPI to be adapted to the interfaced peripherals and their speed and bus release time.

Figure 33-5. Programmable Delay



33.6.7 QSPI SPI Mode

In this mode, the QSPI acts as a regular SPI Host.

To activate this mode, the MODE bit in the Control B register must be cleared (CTRLB.MODE=0).

33.6.7.1 SPI Mode Operations

The QSPI in standard SPI mode operates on the clock generated by the internal programmable baud rate generator. It fully controls the data transfers to and from the client connected to the SPI bus. The QSPI drives the chip select line to the client (CS) and the serial clock signal (SCK).

The QSPI features a single internal shift register and two holding registers: the Transmit Data Register (TXDATA) and the Receive Data Register (RXDATA). The holding registers maintain the data flow at a constant rate.

After enabling the QSPI, a data transfer begins when the processor writes to the TXDATA. The written data is immediately transferred into the internal shift register and transfer on the SPI bus starts. While the data in the internal shift register is shifted on the MOSI line, the MISO line is sampled and shifted into the internal shift register. Receiving data cannot occur without transmitting data.

If new data is written in TXDATA during the transfer, it stays in TXDATA until the current transfer is completed. Then, the received data is transferred from the internal shift register to the RXDATA, the data in TXDATA is loaded into the internal shift register, and a new transfer starts.

The transfer of data written in TXDATA in the internal shift register is indicated by the Transmit Data Register Empty (DRE) bit in the Interrupt Flag Status and Clear register (INTFLAG.DRE). When new data is written in TXDATA, this bit is cleared. The DRE bit is used to trigger the Transmit DMA channel.

The end of transfer is indicated by the Transmission Complete flag (INTFLAG.TXC). If the transfer delay for the last transfer was configured to be greater than 0 (CTRLB.DLYBCT), TXC is set after the completion of the delay. The module clock (CLK_QSPI_AHB) can be switched off at this time.

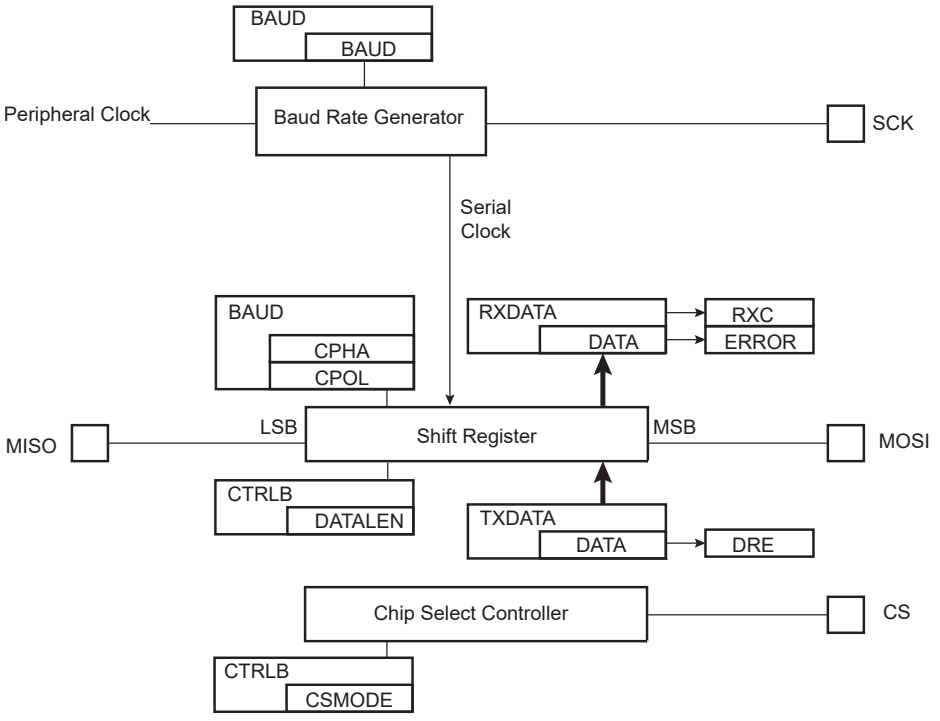
Ongoing transfer of received data from the internal shift register into RXDATA is indicated by the Receive Data Register Full flag (INTFLAG.RXC). When the received data is read, the RXC bit is cleared.

If the RXDATA has not been read before new data is received, the Overrun Error flag in INTFLAG register (INTFLAG.ERROR) is set. As long as this flag is set, data is loaded in RXDATA.

The SPI Mode Block Diagram shows a flow chart describing how transfers are handled.

33.6.7.2 SPI Mode Block Diagram

Figure 33-6. SPI Mode Block Diagram



33.6.7.3 SPI Mode Flow Diagram

Figure 33-7. SPI Mode Flow Diagram

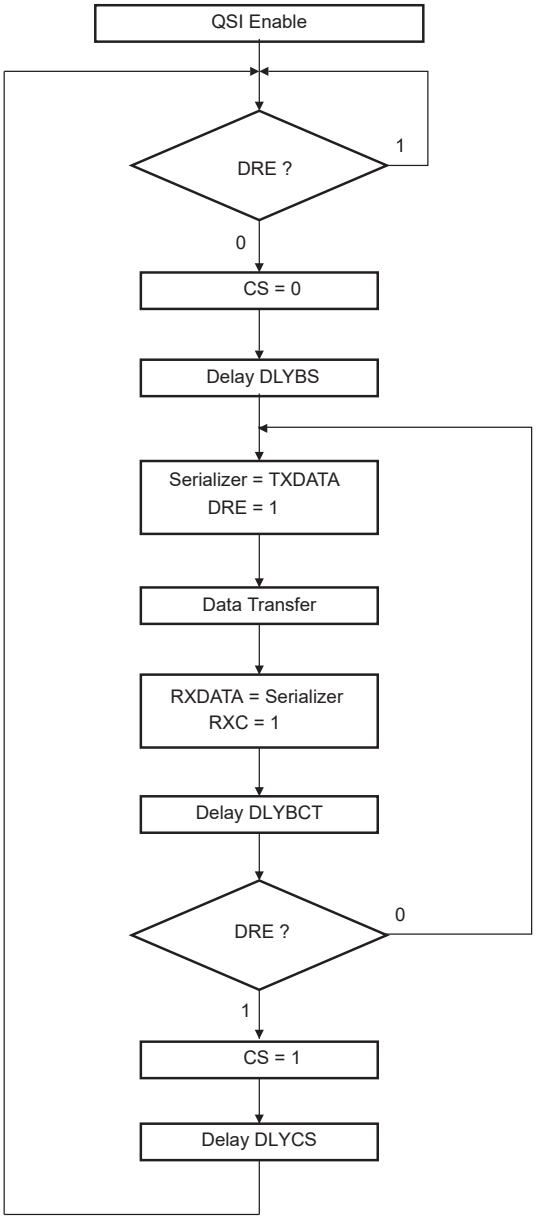
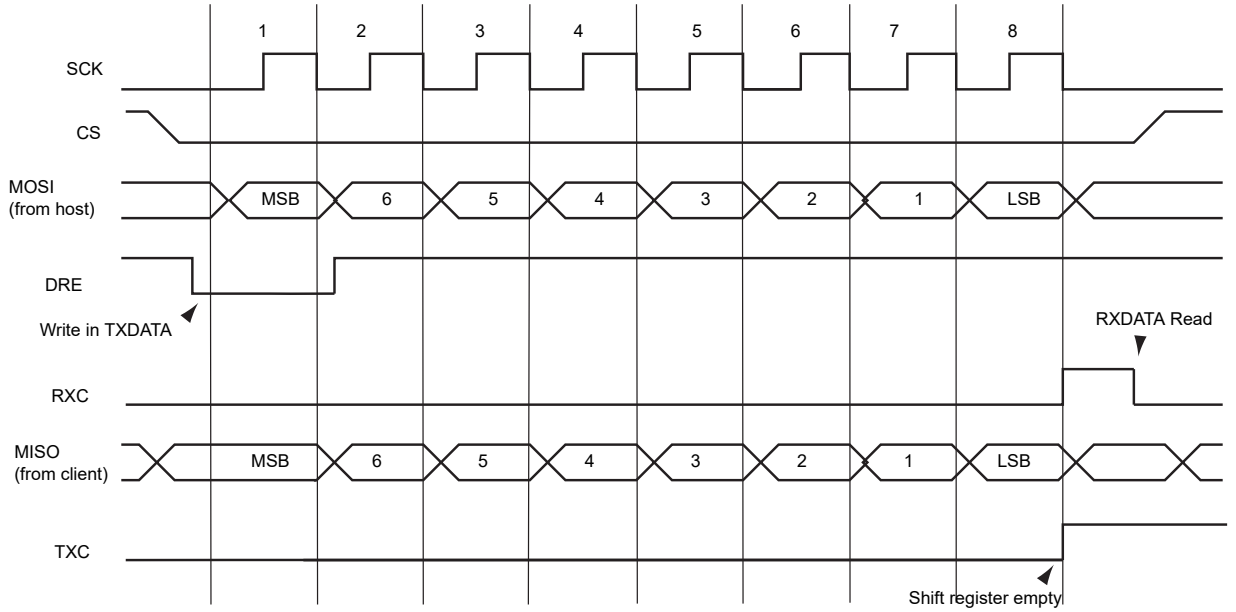


Figure 33-8. Interrupt Flags Behaviour



33.6.7.4 Peripheral Deselection with DMA

When the Direct Memory Access Controller is used, the Chip Select line will remain low during the whole transfer because the Transmit Data Register Empty flag in the Interrupt Flag Status and Clear register (INTFLAG.DRE) is managed by the DMA itself. The reloading of the TXDATA by the DMA is done as soon as the INTFLAG.DRE flag is set. In this case, setting the Chip Select mode bit field in the Control B register (CTRLB.CSMODE) to 0x1 is not mandatory.

However, it may happen that when other DMA channels connected to other peripherals are in use as well, the QSPI DMA could be delayed by another DMA transfer with a higher priority on the bus. Having DMA buffers in slower memories, like Flash memory or SDRAM (compared to fast internal SRAM), may lengthen the reload time of the TXDATA by the DMA as well. This means that TXDATA might not be reloaded in time to keep the Chip Select line low. In this case, the Chip Select line may toggle between data transfer and some SPI Client devices, and the communication might get lost. Writing CTRLB.CSMODE=0x1 can prevent this loss.

When CTRLB.CSMODE=0x0, the CS does not rise in all cases between two transfers on the same peripheral. During a transfer on a Chip Select, the INTFLAG.DRE flag is raised as soon as the content of the TXDATA is transferred into the internal shifter. When this flag is detected, the TXDATA can be reloaded. If this reload occurs before the end of the current transfer and if the next transfer is performed on the same Chip Select as the current transfer, the Chip Select is not de-asserted between the two transfers. This may lead to difficulties for interfacing with some serial peripherals requiring the Chip Select to be de-asserted after each transfer. To facilitate interfacing with such devices, it is recommended to write CTRLB.CSMODE to 0x2.

33.6.7.5 Peripheral Deselection without DMA

During multiple data transfers on a Chip Select without the DMA, the TXDATA is loaded by the processor, and the Transmit Data Register Empty flag in the Interrupt Flag Status and Clear register (INTFLAG.DRE) rises as soon as the content of the RXDATA is transferred into the internal shift register. When this flag is detected high, the TXDATA can be reloaded. If this reload-by-processor occurs before the end of the current transfer and if the next transfer is performed on the same Chip Select as the current transfer, the Chip Select is not de-asserted between the two transfers.

Depending on the application software handling the flags or servicing other interrupts or other tasks, the processor may not reload the TXDATA in time to keep the Chip Select active (low). A null

Delay Between Consecutive Transfer bit field value in the CTRLB register (CTRLB.DLYBCT) will give even less time for the processor to reload the TXDATA. With some SPI client peripherals, requiring the Chip Select line to remain active (low) during a full set of transfers might lead to communication errors.

To facilitate interfacing with such devices, the Chip Select Mode bit field in the CTRLB register (CTRLB.CSMODE) can be written to 0x1. This allows the Chip Select lines to remain in their current state (low = active) until the end of transfer is indicated by the Last Transfer bit in the CTRLA register (CTRLA.LASTXFER). Even if the TXDATA is not reloaded, the Chip Select will remain active. To have the Chip Select line rise at the end of the last data transfer, the LASTXFER bit in the CTRLA must be set before writing the last data to transmit into the TXDATA.

33.6.8 QSPI Serial Memory Mode

In this mode the QSPI acts as a serial Flash memory controller. The QSPI can be used to read data from the serial Flash memory allowing the CPU to execute code from it (XIP execute in place). The QSPI can also be used to control the serial Flash memory (Program, Erase, Lock, and so on) by sending specific commands. In this mode, the QSPI is compatible with single-bit SPI, Dual-SPI and Quad-SPI protocols.

To activate this mode, the MODE bit in Control B register must be set to one (CTRLB.MODE = 1).

In serial memory mode, data cannot be transferred by the TXDATA and the RXDATA, but by writing or reading the QSPI memory space (0x0400 0000 – 0x0500 0000).

Important: QSPI memory space region can be cached to improve data transfer speed.
However, external Flash devices which have command/status registers mapped in the QSPI memory space region must be managed carefully by applying any one of the following configurations:

- Data cache must be disabled.
- If data cache is required, then cache line must be invalidated before reading the status register.

33.6.8.1 Instruction Frame

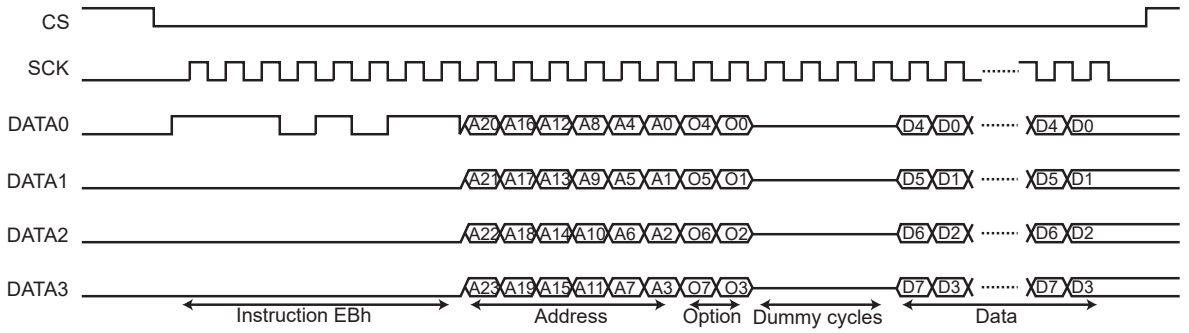
In order to control serial Flash memories, the QSPI is able to sent instructions by the SPI bus (ex: READ, PROGRAM, ERASE, LOCK, etc.). Because instruction set implemented in serial Flash memories is memory vendor dependent, the QSPI includes a complete instruction registers, which makes it very flexible and compatible with all serial Flash memories.

An instruction frame includes:

- **An instruction code (size: 8 bits):** The instruction can be optional in some cases
- **An address (size: 24 bits or 32 bits):** The address is optional but is required by instructions such as READ, PROGRAM, ERASE, LOCK. By default the address is 24 bits long, but it can be 32 bits long to support serial Flash memories larger than 128 Mbit (16 Mbyte).
- **An option code (size: 1/2/4/8 bits):** The option code is optional but is useful for activate the “XIP mode” or the “Continuous Read Mode” for READ instructions, in some serial Flash memory devices. These modes allow to improve the data read latency.
- **Dummy cycles:** Dummy cycles are optional but required by some READ instructions
- **Data bytes are optional:** Data bytes are present for data transfer instructions such as READ or PROGRAM

The instruction code, the address/option and the data can be sent with Single-bit SPI, Dual SPI or Quad SPI protocols.

Figure 33-9. Instruction Frame



33.6.8.2 Instruction Frame Sending

To send an instruction frame, the user must first configure the address to send by writing the field ADDR in the Instruction Address Register (INSTRADDR.ADDR). This step is required if the instruction frame includes an address and no data. When data is present, the address of the instruction is defined by the address of the data accesses in the QSPI memory space, and not by the INSTRADDR register.

If the instruction frame includes the instruction code and/or the option code, the user must configure the instruction code and/or the option code to send by writing the fields INST and OPTCODE bit fields in the Instruction Control Register (INSTRCTRL.OPTCODE, INSTRCTRL.INSTR).

Then, the user must write the Instruction Frame Register (INSTRFRAME) to configure the instruction frame depending on which instruction must be sent. If the instruction frame does not include data, writing in this register triggers the send of the instruction frame in the QSPI. If the instruction frame includes data, the send of the instruction frame is triggered by the first data access in the QSPI memory space.

The instruction frame is configured by the following bits and fields of INSTRFRAME:

- WIDTH field is used to configure which data lanes are used to send the instruction code, the address, the option code and to transfer the data. It is possible to use two unidirectional data lanes (MISO-MOSI Single-bit SPI), two bidirectional data lanes (DATA0 - DATA1 Dual SPI) or four bidirectional data lanes (DATA0 - DATA3).

Table 33-3. WIDTH Encoding

INSTRFRAME	Instruction	Address/Option	Data
0	Single-bit SPI	Single-bit SPI	Single-bit SPI
1	Single-bit SPI	Single-bit SPI	Dual SPI
2	Single-bit SPI	Single-bit SPI	Quad SPI
3	Single-bit SPI	Dual SPI	Dual SPI
4	Single-bit SPI	Quad SPI	Quad SPI
5	Dual SPI	Dual SPI	Dual SPI
6	Quad SPI	Quad SPI	Quad SPI
7	Reserved		

- INSTREN bit enables sending an instruction code
- ADDREN bit enables sending of an address after the instruction code
- OPTCODEEN bit enables sending of an option code after the address
- DATAEN bit enables the transfer of data (READ or PROGRAM instruction)
- OPTCODELEN field configures the option code length (0 -> 1-bit / 1 -> 2-bit / 2 -> 4-bit / 3 -> 8-bit). The value written in OPTCODELEN must be consistent with value written in the field WIDTH. For

example: OPTCODELEN = 0 (1-bit option code) is not coherent with WIDTH = 6 (option code sent with QuadSPI protocol, thus the minimum length of the option code is 4-bit).

- ADDRLEN bit configures the address length (0 -> 24 bits / 1-> 32 bits)
- TFRTYPE field defines which type of data transfer must be performed
- DUMMYLEN field configures the number of dummy cycles when reading data from the serial Flash memory. Between the address/option and the data, with some instructions, dummy cycles are inserted by the serial Flash memory.

If data transfer is enabled, the user can access the serial memory by reading or writing the QSPI memory space following these rules:

- Reading from the serial memory, but not memory data (for example reading the JEDEC-ID or the STATUS), requires TFRTYPE to be written to 0x0
- Reading from the serial memory, and particularly memory data, requires TFRTYPE to be written to '1'
- Writing to the serial memory, but not memory data (for example writing the configuration or STATUS), requires TFRTYPE to be written to 0x2
- Writing to the serial memory, and particularly memory data, requires TFRTYPE to be written to 0x3

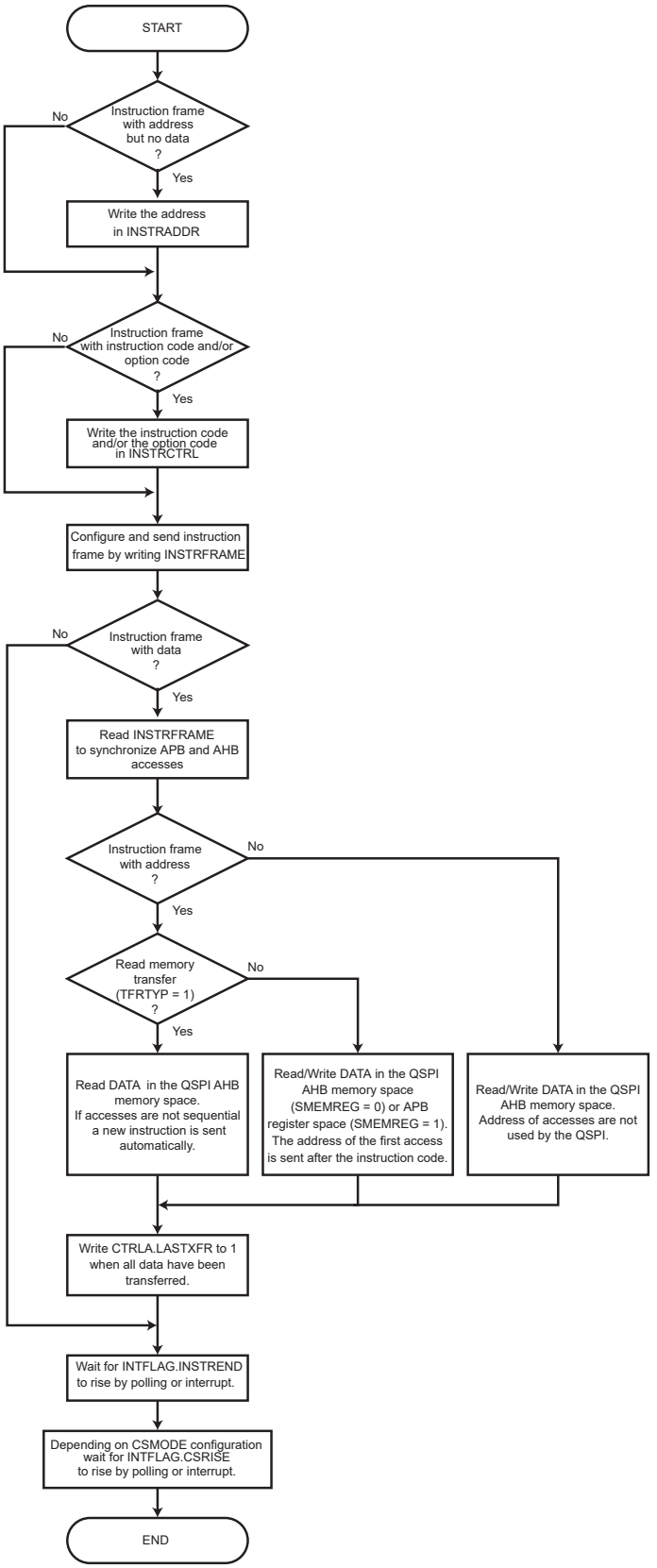
If TFRTYP has a value other than 0x1 and CTRLB.SMEMREG=0, the address sent in the instruction frame is the address of the first system bus accesses. The addresses of the subsequent access actions are not used by the QSPI. At each system bus access, an SPI transfer is performed with the same size. For example, a half-word system bus access leads to a 16-bit SPI transfer, and a byte system bus access leads to an 8-bit SPI transfer.

If CTRLB.SMEMREG=1, accesses are made via the QSPI registers and the address sent in the instruction frame is the address defined in the INSTRADDR register. Each time the INSTRFRAME or TXDATA registers are written, an SPI transfer is performed with a byte size. Another byte is read each time RXDATA register is read or written each time TXDATA register is written. The SPI transfer ends by writing the LASTXFER bit in Control A register (CTRLA.LASTXFER).

If TFRTYP=0x1, the address of the first instruction frame is the one of the first read access in the QSPI memory space. Each time the read accesses become non-sequential (addresses are not consecutive), a new instruction frame is sent with the last system bus access address. In this way, the system can read data at a random location in the serial memory. The size of the SPI transfers may differ from the size of the system bus read accesses.

When data transfer is not enabled, the end of the instruction frame is indicated when the INSTREND interrupt flag in the INTFLAG register is set. When data transfer is enabled, the user must indicate when data transfer is completed in the QSPI memory space by setting the bit LASTXFR in the CTRLA. The end of the instruction frame is indicated when the INSTREND interrupt flag in the INTFLAG register is set.

Figure 33-10. Instruction Transmission Flow Diagram



33.6.8.3 Read Memory Transfer

The user can access the data of the serial memory by sending an instruction with DATAEN=1 and TFRTYP=0x1 in the Instruction Frame register (INSTRFRAME).

In this mode the QSPI is able to read data at random address into the serial Flash memory, allowing the CPU to execute code directly from it (XIP execute-in-place).

In order to fetch data, the user must first configure the instruction frame by writing the INSTRFRAME. Then data can be read at any address in the QSPI address space mapping. The address of the system bus read accesses match the address of the data inside the serial Flash memory.

When Fetch Mode is enabled, several instruction frames can be sent before writing the bit LASTXFR in the CTRLA. Each time the system bus read accesses become non-sequential (addresses are not consecutive), a new instruction frame is sent with the corresponding address.

33.6.8.4 Continuous Read Mode

The QSPI is compatible with Continuous Read Mode (CRM) which is implemented in some Serial Flash memories.

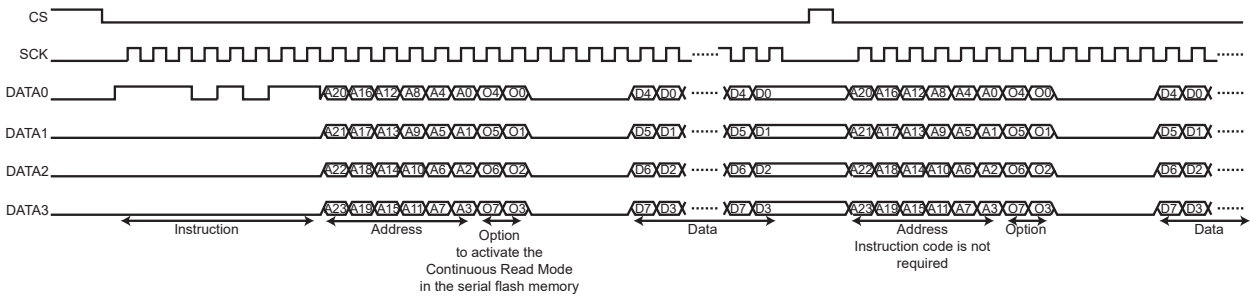
The CRM allows to reduce the instruction overhead by excluding the instruction code from the instruction frame. When CRM is activated in a Serial Flash memory (by a specific option code), the instruction code is stored in the memory. For the next instruction frames, the instruction code is not required, as the memory uses the stored one.

In the QSPI, CRM is used when reading data from the memory (INSTFRAME.TFRTYPE=0x1). The addresses of the system bus read accesses are often non-sequential, this leads to many instruction frames with always the same instruction code. By disabling the sending of the instruction code, the CRM reduces the access time of the data.

To be functional, this mode must be enabled in both the QSPI and the Serial Flash memory. The CRM is enabled in the QSPI by setting the CRM bit in the INSTRFRAME register (INSTFRAME.CRMODE=1, INSTFRAME.TFRTYPE must be 0x1). The CRM is enabled in the Serial Flash memory by sending a specific option code.

CAUTION If CRM is not supported by the Serial Flash memory or disabled, the CRMODE bit must not be set. Otherwise, data read out the Serial Flash memory is not valid.

Figure 33-11. Continuous Read Mode



33.6.8.5 Instruction Frame Transmission Examples

All waveforms in the following examples describe SPI transfers in SPI Clock mode 0 (BAUD.CPOL=0 and BAUD.CPHA=0). All system bus accesses described below refer to the system bus address phase. System bus wait cycles and system bus data phases are not shown.

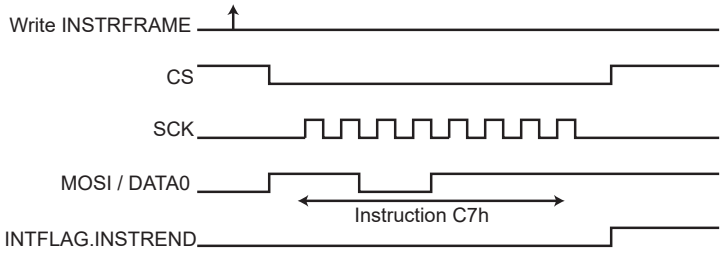
Example 33-1. Example 1

Instruction in Single-bit SPI, without address, without option, without data.

Command: CHIP ERASE (C7h).

- Write 0x0000_00C7 to INSTRCTRL register
- Write 0x0000_0010 to INSTRFRAME register
- Wait for INTFLAG.INSTREND to rise

Figure 33-12. Instruction Transmission Waveform 1



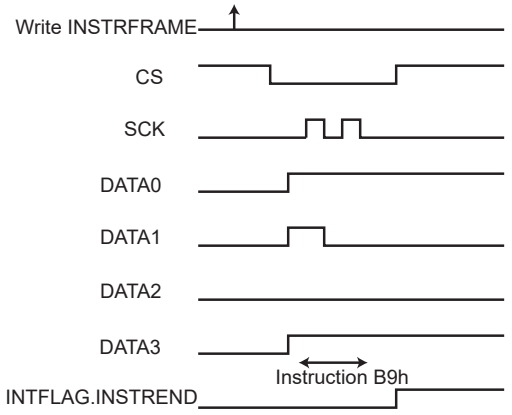
Example 33-2. Example 2

Instruction in Quad SPI, without address, without option, without data.

Command: POWER DOWN (B9h)

- Write 0x0000_00B9 to INSTRCTRL register
- Write 0x0000_0016 to INSTRFRAME register
- Wait for INTFLAG.INSTREND to rise

Figure 33-13. Instruction Transmission Waveform 2



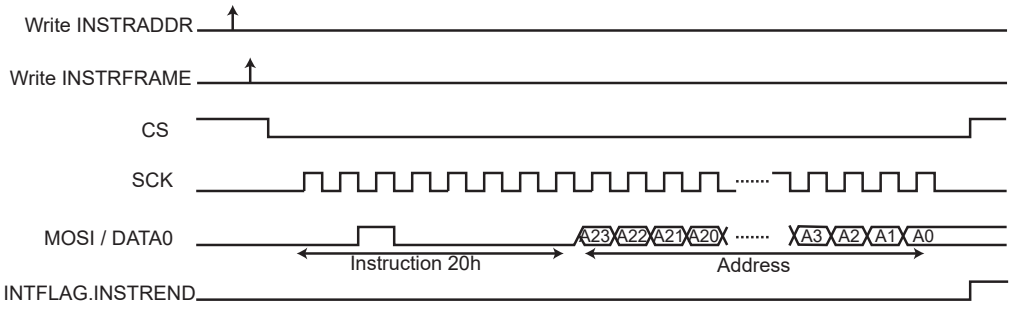
Example 33-3. Example 3

Instruction in Single-bit SPI, with address in Single-bit SPI, without option, without data.

Command: BLOCK ERASE (20h)

- Write the address (of the block to erase) to QSPI_AR
- Write 0x0000_0020 to INSTRCTRL register
- Write 0x0000_0030 toINSTRFRAME register
- Wait for INTFLAG.INSTREND to rise

Figure 33-14. Instruction Transmission Waveform 3



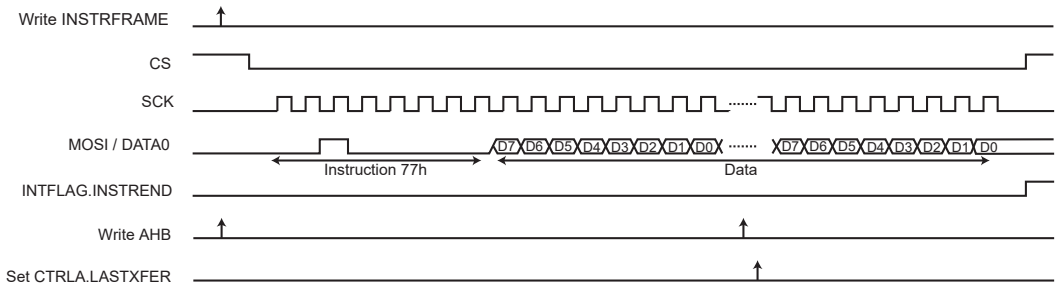
Example 33-4. Example 4

Instruction in Single-bit SPI, without address, without option, with data write in Single-bit SPI.

Command: SET BURST (77h)

- Write 0x0000_0077 to INSTRCTRL register.
- Write 0x0000_2090 to INSTRFRAME register.
- Read INSTRFRAME register (dummy read) to synchronize system bus accesses.
- Write data to the system bus memory space (0x0400_0000–0x0500_0000). The address of the system bus write accesses is not used.
- Write the LASTXFR bit in CTRLA register to '1'.
- Wait for INTFLAG.INSTREND to rise.

Figure 33-15. Instruction Transmission Waveform 4



Example 33-5. Example 5

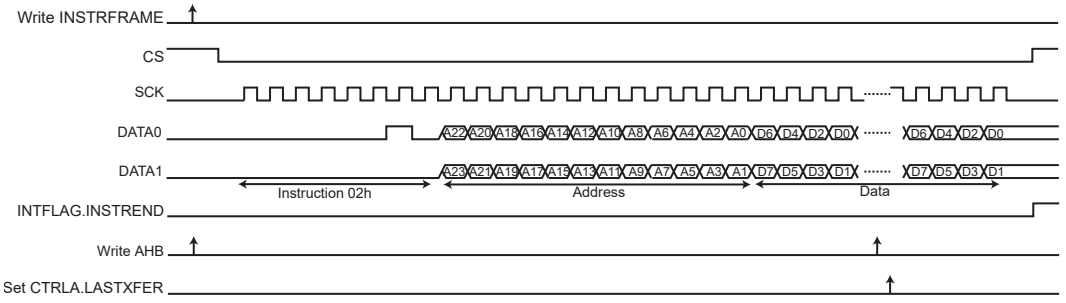
Instruction in Single-bit SPI, with address in Dual SPI, without option, with data write in Dual SPI.

Command: BYTE/PAGE PROGRAM (02h)

- Write 0x0000_0002 to INSTRCTRL register.
- Write 0x0000_30B3 to INSTRFRAME register.
- Read INSTRFRAME register (dummy read) to synchronize system bus accesses.
- Write data to the QSPI system bus memory space (0x040 00000–0x0500_0000). The address of the first system bus write access is sent in the instruction frame. The address of the next system bus write accesses is not used.

- Write LASTXFR bit in CTRLA register to '1'.
- Wait for INTFLAG.INSTREND to rise.

Figure 33-16. Instruction Transmission Waveform 5



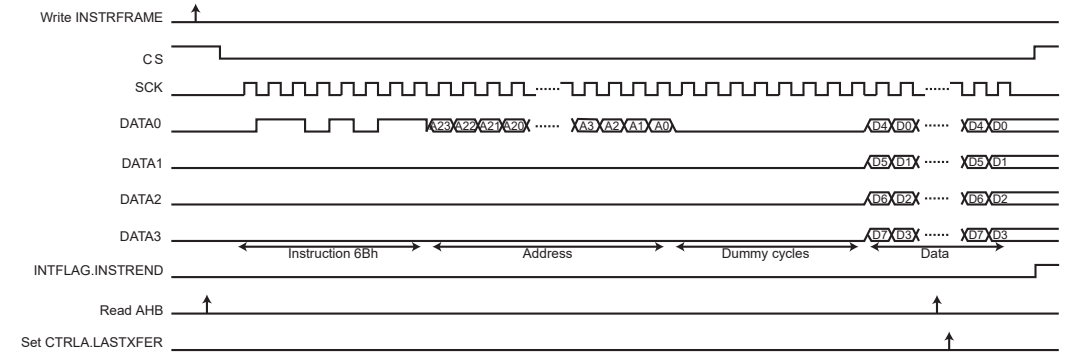
Example 33-6. Example 6

Instruction in Single-bit SPI, with address in Single-bit SPI, without option, with data read in Quad SPI, with eight dummy cycles.

Command: QUAD_OUTPUT READ ARRAY (6Bh)

- Write 0x0000_006B to INSTRCTRL register.
- Write 0x0008_10B2 ti INSTRFRAME register.
- Read QSPI_IR (dummy read) to synchronize system bus accesses.
- Read data from the QSPI system bus memory space (0x040 00000–0x0500_0000).
  The address of the first system bus read access is sent in the instruction frame.
  The address of the next system bus read accesses is not used.
- Write the LASTXFR bit in CTRLA register to '1'.
- Wait for INTFLAG.INSTREND to rise.

Figure 33-17. Instruction Transmission Waveform 6



Example 33-7. Example 7

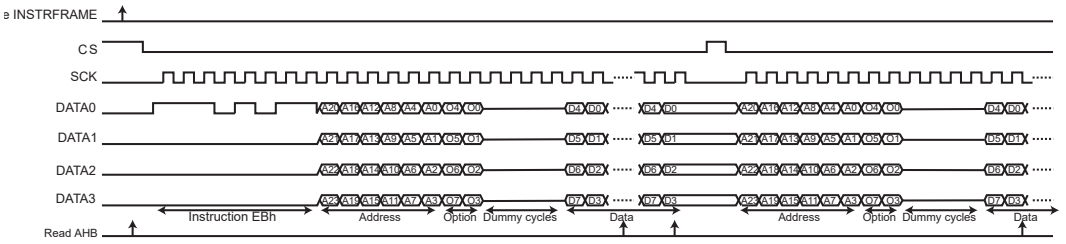
Instruction in Single-bit SPI, with address and option in Quad SPI, with data read from Quad SPI, with four dummy cycles, with fetch and continuous read.

Command: FAST READ QUAD I/O (EBh) - 8-BIT OPTION (0x30h)

- Write 0x0030_00EB to INSTRCTRL register.

- Write 0x0004_33F4 to INSTRFRAME register.
- Read INSTRFRAME register (dummy read) to synchronize system bus accesses.
- Read data from the QSPI system bus memory space (0x040 00000–0x0500_0000). Fetch is enabled, the address of the system bus read accesses is always used.
- Write LASTXFR bit in CTRLA register to '1'.
- Wait for INTFLAG.INSTREND to rise.

Figure 33-18. Instruction Transmission Waveform 7



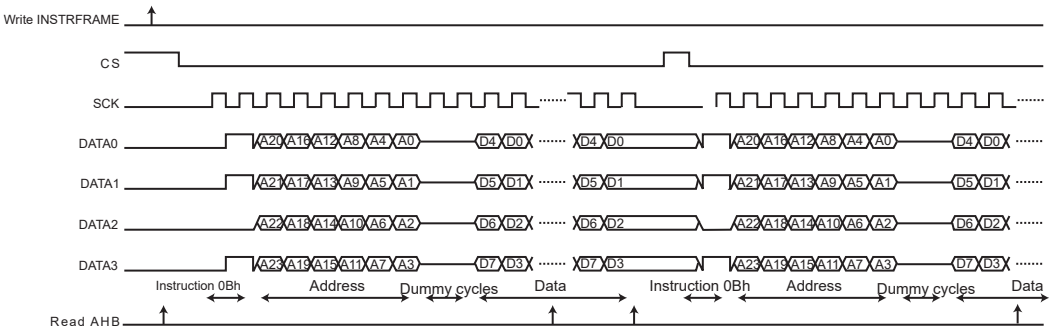
Example 33-8. Example 8

Instruction in Quad SPI, with address in Quad SPI, without option, with data read from Quad SPI, with two dummy cycles, with fetch.

Command: HIGH-SPEED READ (0Bh)

- Write 0x0000_000B to INSTRCTRL register.
- Write 0x0002_20B6 to INSTRFRAME register.
- Read INSTRFRAME register (dummy read) to synchronize system bus accesses.
- Read data in the QSPI system bus memory space (0x040 00000–0x0500_0000). Fetch is enabled, the address of the system bus read accesses is always used.
- Write LASTXFR bit in CTRLA register to '1'.
- Wait for INTFLAG.INSTREND to rise.

Figure 33-19. Instruction Transmission Waveform 8



33.6.9 Scrambling/Unscrambling Function

The scrambling/unscrambling function cannot be performed on devices other than memories. Data is scrambled when written to memory and unscrambled when data is read.

The external data lines can be scrambled to prevent intellectual property data located in off-chip memories from being easily recovered by analyzing data at the package pin level of either the micro-controller or the QSPI client device (e.g., memory).

The scrambling/unscrambling function can be enabled by writing a '1' to the ENABLE bit in the Scrambling Control register (SCRAMBCTRL.ENABLE).

The scrambling and unscrambling are performed on-the-fly without impacting the throughput.

The scrambling method depends on the user-configurable Scrambling User Key in the Scrambling Key register (SCRAMBKEY.KEY). This register is only accessible in Write mode.

By default, the scrambling and unscrambling algorithm includes the scrambling user key, plus a device-dependent random value. This random value is not included when the Scrambling/Unscrambling Random Value Disable bit in the Scrambling Mode register (SCRAMBCTRL.RANDOMDIS) is written to '1'.

The random value is neither user-configurable nor readable. If SCRAMBCTRL.RANDOMDIS=0, data scrambled by a given circuit cannot be unscrambled by a different circuit.

If SCRAMBCTRL.RANDOMDIS=1, the scrambling/unscrambling algorithm includes only the scrambling user key, making it possible to manage data by different circuits.

The scrambling user key must be securely stored in a reliable Non-Volatile Memory to recover data from the off-chip memory. Any data scrambled with a given key cannot be recovered if the key is lost.

33.6.10 DMA Operation

The QSPI generates the following DMA requests:

- Data received (RX): The request is set when data is available in the RXDATA register, and cleared when RXDATA is read.
- Data transmit (TX): The request is set when the transmit buffer (TXDATA) is empty, and cleared when TXDATA is written.

Note: If DMA and RX memory modes are selected, a QSPI memory space read operation is required to force the first triggering.

If the CPU accesses the registers which are source of DMA request set/clear condition, the DMA request can be lost or the DMA transfer can be corrupted.

33.6.11 Interrupts

The QSPI has the following interrupt source:

- Interrupt Request (INTREQ): Indicates that at least one bit in the Interrupt Flag Status and Clear register (INTFLAG) is set to '1'.

Each interrupt source has an interrupt flag associated with it. The interrupt flag in the Interrupt Flag Status and Clear (INTFLAG) register is set when the interrupt condition occurs. Each interrupt can be individually enabled by writing a '1' to the corresponding bit in the Interrupt Enable Set (INTENSET) register, and disabled by writing a '1' to the corresponding bit in the Interrupt Enable Clear (INTENCLR) register. An interrupt request is generated when the interrupt flag is set and the corresponding interrupt is enabled. The interrupt request remains active until the interrupt flag is cleared, the interrupt is disabled, or the QSPI is reset. All interrupt requests from the peripheral are ORed together on system level to generate one combined interrupt request to the NVIC. The user must read the INTFLAG register to determine which interrupt condition is present.

Note that interrupts must be globally enabled for interrupt requests to be generated.

33.7 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0							ENABLE	SWRST
		15:8								
		23:16								
		31:24								LASTXFER
0x04	CTRLB	7:0			CSMODE[1:0]		SMEMREG	WDRBT	LOOPEN	MODE
		15:8					DATALEN[3:0]			
		23:16	DLYBCT[7:0]							
		31:24	DLYCS[7:0]							
0x08	BAUD	7:0							CPHA	CPOL
		15:8	BAUD[7:0]							
		23:16	DLYBS[7:0]							
		31:24								
0x0C	RXDATA	7:0	DATA[7:0]							
		15:8	DATA[15:8]							
		23:16								
		31:24								
0x10	TXDATA	7:0	DATA[7:0]							
		15:8	DATA[15:8]							
		23:16								
		31:24								
0x14	INTENCLR	7:0					ERROR	TXC	DRE	RXC
		15:8						INSTREND		CSRISE
		23:16								
		31:24								
0x18	INTENSET	7:0					ERROR	TXC	DRE	RXC
		15:8						INSTREND		CSRISE
		23:16								
		31:24								
0x1C	INTFLAG	7:0					ERROR	TXC	DRE	RXC
		15:8						INSTREND		CSRISE
		23:16								
		31:24								
0x20	STATUS	7:0							ENABLE	
		15:8							CSSTATUS	
		23:16								
		31:24								
0x24 ... 0x2F	Reserved									
0x30	INSTRADDR	7:0	ADDR[7:0]							
		15:8	ADDR[15:8]							
		23:16	ADDR[23:16]							
		31:24	ADDR[31:24]							
0x34	INSTRCTRL	7:0	INSTR[7:0]							
		15:8								
		23:16	OPTCODE[7:0]							
		31:24								
0x38	INSTRFRAME	7:0	DATAEN	OPTCODEEN	ADDREN	INSTREN		WIDTH[2:0]		
		15:8	DDREN	CRMODE	TFRTYPE[1:0]			ADDRLEN	OPTCODELEN[1:0]	
		23:16				DUMMYLEN[4:0]				
		31:24								
0x3C ... 0x3F	Reserved									
0x40	SCRAMBCTRL	7:0							RANDOMDIS	ENABLE
		15:8								
		23:16								
		31:24								

.....continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x44	SCRAMBKEY	7:0	KEY[7:0]							
		15:8	KEY[15:8]							
		23:16	KEY[23:16]							
		31:24	KEY[31:24]							

33.8 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register, and the 8-bit halves of a 16-bit register can be accessed directly.

Optional write protection by the Peripheral Access Controller (PAC) is denoted by the "PAC Write Protection" property in each individual register description.

See *Peripheral Access Controller (PAC)* from Related Links.

Some registers are enable-protected, meaning they can only be written when the QSPI is disabled. Enable-protection is denoted by the Enable-protected property in each individual register description.

Related Links

26. Peripheral Access Controller (PAC)

33.8.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00000000
Property: -

Control A

Bit	31	30	29	28	27	26	25	24
								LASTXFER
Access								W
Reset								0

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
							ENABLE	SWRST
Access							W	W
Reset							0	0

Bit 24 – LASTXFER Last Transfer
0: No effect.
1: The chip select will be de-asserted after the character written in TD has been transferred.

Bit 1 – ENABLE Enable
Writing a '0' to this bit disables the QSPI.
Writing a '1' to this bit enables the QSPI to transfer and receive data.
As soon as ENABLE is reset, QSPI finishes its transfer.
All pins are set in input mode and no data is received or transmitted.
If a transfer is in progress, the transfer is finished before the QSPI is disable.

Bit 0 – SWRST Software Reset
Writing a '0' to this bit has no effect.
Writing a '1' to this bit resets the QSPI. A software-triggered hardware reset of the QSPI interface is performed.
DMAC channels are not affected by software reset.

33.8.2 Control B

Name: CTRLB
Offset: 0x04
Reset: 0x00000000
Property: PAC Write-Protection

Control B

Bit	31	30	29	28	27	26	25	24
	DLYCS[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	DLYBCT[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
					DATALEN[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
			CSMODE[1:0]		SMEMREG	WDRBT	LOOPEN	MODE
Access			R/W	R/W	R/W	R/W	R/W	R/W
Reset			0	0	0	0	0	0

Bits 31:24 – DLYCS[7:0] Minimum Inactive CS Delay
This bit field defines the minimum delay between the inactivation and the activation of CS. The DLYCS time guarantees the client minimum deselect time.
If DLYCS is 0x00, one CLK_QSPI_AHB period will be inserted by default.
Otherwise, the following equation determines the delay:
DLYCS = Minimum inactive × fperipheral clock

Bits 23:16 – DLYBCT[7:0] 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=0x00, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers. In Serial Memory mode (MODE=1), DLYBCT is ignored and no delay is inserted. Otherwise, the following equation determines the delay:
DLYBCT = (Delay Between Consecutive Transfers × fperipheral clock) / 32

Bits 11:8 – DATALEN[3:0] Data Length
The DATALEN field determines the number of data bits transferred. Reserved values must not be used.

Value	Name	Description
0x0	8BITS	8-bits transfer
0x1	9BITS	9-bits transfer
0x2	10BITS	10-bits transfer
0x3	11BITS	11-bits transfer
0x4	12BITS	12-bits transfer
0x5	13BITS	13-bits transfer

Value	Name	Description
0x6	14BITS	14-bits transfer
0x7	15BITS	15-bits transfer
0x8	16BITS	16-bits transfer
0x9-0xF		Reserved

Bits 5:4 – CSMODE[1:0] Chip Select Mode

The CSMODE field determines how the chip select is de-asserted.

Value	Name	Description
0x0	NORELOAD	The chip select is de-asserted if TD has not been reloaded before the end of the current transfer.
0x1	LASTXFER	The chip select is de-asserted when the bit LASTXFER is written at 1 and the character written in TD has been transferred.
0x2	SYSTEMATICALLY	The chip select is de-asserted systematically after each transfer.
0x3		Reserved

Bit 3 – SMEMREG Serial Memory Register Mode

Value	Description
0	Serial memory registers are written via AHB access.
1	Serial memory registers are written via APB access. Reset the QSPI.

Bit 2 – WDRBT Wait Data Read Before Transfer

This bit determines the Wait Data Read Before Transfer option.

Bit 1 – LOOPEN Local Loopback Enable

This bit defines if the Local Loopback is enabled or disabled.

LOOPEN controls the local loopback on the data serializer for testing in SPI Mode only. (MISO is internally connected on MOSI).

Value	Description
0	Local Loopback is disabled.
1	Local Loopback is enabled.

Bit 0 – MODE Serial Memory Mode

This bit defines if the QSPI is in SPI Mode or Serial Memory Mode.

Value	Name	Description
0	SPI	SPI operating mode
1	MEMORY	Serial Memory operating mode

33.8.3 Baud Rate

Name: BAUD
Offset: 0x08
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	DLYBS[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	BAUD[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
							CPHA	CPOL
Access							R/W	R/W
Reset							0	0

Bits 23:16 – DLYBS[7:0] Delay Before SCK
This field defines the delay from CS valid to the first valid SCK transition.
When DLYBS equals zero, the CS valid to SCK transition is 1/2 the SCK clock period.
Otherwise, the following equation determines the delay:

Equation 33-1. Delay Before SCK

$$Delay\ \ Before\ SCK = \frac{DLYBS}{MCK}$$

Bits 15:8 – BAUD[7:0] Serial Clock Baud Rate
The QSPI uses a modulus counter to derive the SCK baud rate from the module clock (MCK) CLK_QSPI_AHB. The Baud rate is selected by writing a value from 1 to 255 in the BAUD field. The following equation determines the SCK baud rate:

Equation 33-2. SCK Baud Rate

$$SCK\ \ Baud\ \ Rate = \frac{MCK}{(BAUD + 1)}$$

Bit 1 – CPHA Clock Phase
CPHA determines which edge of SCK causes data to change and which edge causes data to be captured. CPHA is used with CPOL to produce the required clock/data relationship between host and client devices.

Value	Description
0	Data is captured on the leading edge of SCK and changed on the following edge of SCK.
1	Data is changed on the leading edge of SCK and captured on the following edge of SCK.

Bit 0 – CPOL Clock Polarity

CPOL is used to determine the inactive state value of the serial clock (SCK). It is used with CPHA to produce the required clock/data relationship between host and client devices.

Value	Description
0	The inactive state value of SCK is logic level zero.
0	The inactive state value of SCK is logic level 'one'.

33.8.4 Receive Data

Name: RXDATA
Offset: 0x0C
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	DATA[15:8]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DATA[7:0]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bits 15:0 – DATA[15:0] Receive Data
Data received by the QSPI is stored in this register right-justified. Unused bits read zero.

33.8.5 Transmit Data

Name: TXDATA
Offset: 0x10
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	DATA[15:8]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DATA[7:0]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bits 15:0 – DATA[15:0] Transmit Data
Data to be transmitted by the QSPI is stored in this register. Information to be transmitted must be written to the transmit data register in a right-justified format.

33.8.6 Interrupt Enable Clear

Name: INTENCLR
Offset: 0x14
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
						INSTREND		CSRISE
Access						R/W		R/W
Reset						0		0

Bit	7	6	5	4	3	2	1	0
					ERROR	TXC	DRE	RXC
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit 10 – INSTREND Instruction End Interrupt Disable
Writing a '0' to this bit has no effect.
Writing a '1' will clear the corresponding interrupt request.

Value	Description
0	The INSTREND interrupt is disabled.
1	The INSTREND interrupt is enabled.

Bit 8 – CSRISE Chip Select Rise Interrupt Disable
Writing a '0' to this bit has no effect.
Writing a '1' will clear the corresponding interrupt request.

Value	Description
0	The CSRISE interrupt is disabled.
1	The CSRISE interrupt is enabled.

Bit 3 – ERROR Overrun Error Interrupt Disable
Writing a '0' to this bit has no effect.
Writing a '1' will clear the corresponding interrupt request.

Value	Description
0	The ERROR interrupt is disabled.
1	The ERROR interrupt is enabled.

Bit 2 – TXC Transmission Complete Interrupt Disable
Writing a '0' to this bit has no effect.
Writing a '1' will clear the corresponding interrupt request.

Value	Description
0	The TXC interrupt is disabled.
1	The TXC interrupt is enabled.

Bit 1 – DRE Transmit Data Register Empty Interrupt Disable
Writing a '0' to this bit has no effect.
Writing a '1' will clear the corresponding interrupt request.

Value	Description
0	The DRE interrupt is disabled.
1	The DRE interrupt is enabled.

Bit 0 – RXC Receive Data Register Full Interrupt Disable
Writing a '0' to this bit has no effect.
Writing a '1' will clear the corresponding interrupt request.

Value	Description
0	The RXC interrupt is disabled.
1	The RXC interrupt is enabled.

33.8.7 Interrupt Enable Set

Name: INTENSET
Offset: 0x18
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
						INSTREND		CSRISE
Access						R/W		R/W
Reset						0		0

Bit	7	6	5	4	3	2	1	0
					ERROR	TXC	DRE	RXC
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit 10 – INSTREND Instruction End Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' will set the corresponding interrupt request.

Value	Description
0	The INSTREND interrupt is disabled.
1	The INSTREND interrupt is enabled.

Bit 8 – CSRISE Chip Select Rise Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' will set the corresponding interrupt request.

Value	Description
0	The CSRISE interrupt is disabled.
1	The CSRISE interrupt is enabled.

Bit 3 – ERROR Overrun Error Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' will set the corresponding interrupt request.

Value	Description
0	The ERROR interrupt is disabled.
1	The ERROR interrupt is enabled.

Bit 2 – TXC Transmission Complete Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' will set the corresponding interrupt request.

Value	Description
0	The TXC interrupt is disabled.
1	The TXC interrupt is enabled.

Bit 1 – DRE Transmit Data Register Empty Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' will set the corresponding interrupt request.

Value	Description
0	The DRE interrupt is disabled.
1	The DRE interrupt is enabled.

Bit 0 – RXC Receive Data Register Full Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' will set the corresponding interrupt request.

Value	Description
0	The RXC interrupt is disabled.
1	The RXC interrupt is enabled.

33.8.8 Interrupt Flag Status and Clear

Name: INTFLAG
Offset: 0x1C
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
						INSTREND		CSRISE
Access						R/W		R/W
Reset						0		0

Bit	7	6	5	4	3	2	1	0
					ERROR	TXC	DRE	RXC
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit 10 – INSTREND Instruction End
This bit is set when an Instruction End has been detected.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 8 – CSRISE Chip Select Rise
The bit is set when a Chip Select Rise has been detected.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 3 – ERROR Overrun Error
This bit is set when an ERROR has occurred.
An ERROR occurs when RXDATA is loaded at least twice from the serializer.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the flag.

Bit 2 – TXC Transmission Complete
0: As soon as data is written in TXDATA.
1: TXDATA and internal shifter are empty. If a transfer delay has been defined, TXC is set after the completion of such delay.

Bit 1 – DRE Transmit Data Register Empty
0: Data has been written to TXDATA and not yet transferred to the serializer.
1: The last data written in the TXDATA has been transferred to the serializer.
This bit is '0' when the QSPI is disabled or at reset.
The bit is set as soon as ENABLE bit is set.

Bit 0 – RXC Receive Data Register Full
0: No data has been received since the last read of RXDATA.
1: Data has been received and the received data has been transferred from the serializer to RXDATA since the last read of RXDATA.

33.8.9 Status

Name: STATUS
Offset: 0x20
Reset: 0x00000200
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
							CSSTATUS	
Access							R	
Reset							1	

Bit	7	6	5	4	3	2	1	0
							ENABLE	
Access							R	
Reset							0	

Bit 9 – CSSTATUS Chip Select

Value	Description
0	Chip Select is asserted.
1	Chip Select is not asserted.

Bit 1 – ENABLE Enable

Value	Description
0	QSPI is disabled.
1	QSPI is enabled.

33.8.10 Instruction Address

Name: INSTRADDR
Offset: 0x30
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
	ADDR[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	ADDR[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	ADDR[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	ADDR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – ADDR[31:0] Instruction Address
Address to send to the serial Flash memory in the instruction frame.

33.8.11 Instruction Code

Name: INSTRCTRL
Offset: 0x34
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	OPTCODE[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	INSTR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 23:16 – OPTCODE[7:0] Option Code
These bits define the option code to send to the serial flash memory.

Bits 7:0 – INSTR[7:0] Instruction Code
Instruction code to send to the serial flash memory.

33.8.12 Instruction Frame

Name: INSTRFRAME
Offset: 0x38
Reset: 0x00000000
Property: -

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
				DUMMYLEN[4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	DDREN	CRMODE	TFRTYPE[1:0]			ADDRLEN	OPTCODELEN[1:0]	
Access	R/W	R/W	R/W	R/W		R/W	R/W	R/W
Reset	0	0	0	0		0	0	0

Bit	7	6	5	4	3	2	1	0
	DATAEN	OPTCODEEN	ADDREN	INSTREN		WIDTH[2:0]		
Access	R/W	R/W	R/W	R/W		R/W	R/W	R/W
Reset	0	0	0	0		0	0	0

Bits 20:16 – DUMMYLEN[4:0] Dummy Cycles Length
The DUMMYLEN field defines the number of dummy cycles required by the serial Flash memory before data transfer.

Bit 15 – DDREN Double Data Rate Enable
Note: Double Data Rate operating is only supported in Read.

Value	Description
0	Double Data Rate operating mode is disabled.
1	Double Data Rate operating mode is enabled.

Bit 14 – CRMODE Continuous Read Mode
This bit defines if the Continuous Read Mode is enabled or disabled.

Value	Description
0	Continuous Read Mode is disabled.
1	Continuous Read Mode is enabled.

Bits 13:12 – TFRTYPE[1:0] Data Transfer Type
These bits define the data type transfer.

Value	Name	Description
0x0	READ	Read transfer from the serial memory.Scrambling is not performed.Read at random location (fetch) in the serial flash memory is not possible.
0x1	READMEMORY	Read data transfer from the serial memory.If enabled, scrambling is performed.Read at random location (fetch) in the serial flash memory is possible.
0x2	WRITE	Write transfer into the serial memory.Scrambling is not performed.
0x3	WRITEMEMORY	Write data transfer into the serial memory. If enabled, scrambling is performed.

Bit 10 – ADDRLEN Address Length

The ADDRLEN bit determines the length of the address.

Value	Name	Description
0x0	24BITS	24-bits address length
0x1	32BITS	32-bits address length

Bits 9:8 – OPTCODELEN[1:0] Option Code Length

The OPTCODELEN field determines the length of the option code. The value written in OPTCODELEN must be coherent with value written in the field WIDTH. For example: OPTCODELEN=0 (1-bit option code) is not coherent with WIDTH=6 (option code sent with QuadSPI protocol, thus the minimum length of the option code is 4-bit).

Value	Name	Description
0x0	1BIT	1-bit length option code
0x1	2BITS	2-bits length option code
0x2	4BITS	4-bits length option code
0x3	8BITS	8-bits length option code

Bit 7 – DATAEN Data Enable

Value	Description
0	No data is sent/received to/from the serial flash memory.
1	Data is sent/received to/from the serial flash memory.

Bit 6 – OPTCODEEN Option Enable

Value	Description
0	The option is not sent to the serial flash memory
1	The option is sent to the serial flash memory.

Bit 5 – ADDREN Address Enable

Value	Description
0	The transfer address is not sent to the serial flash memory.
1	The transfer address is sent to the serial flash memory.

Bit 4 – INSTREN Instruction Enable

Value	Description
0	The instruction is not sent to the serial flash memory.
1	The instruction is sent to the serial flash memory.

Bits 2:0 – WIDTH[2:0] Instruction Code, Address, Option Code and Data Width

This field defines the width of the instruction code, the address, the option and the data.

Value	Name	Description
0x0	SINGLE_BIT_SPI	Instruction: Single-bit SPI / Address-Option: Single-bit SPI / Data: Single-bit SPI
0x1	DUAL_OUTPUT	Instruction: Single-bit SPI / Address-Option: Single-bit SPI / Data: Dual SPI
0x2	QUAD_OUTPUT	Instruction: Single-bit SPI / Address-Option: Single-bit SPI / Data: Quad SPI
0x3	DUAL_IO	Instruction: Single-bit SPI / Address-Option: Dual SPI / Data: Dual SPI
0x4	QUAD_IO	Instruction: Single-bit SPI / Address-Option: Quad SPI / Data: Quad SPI
0x5	DUAL_CMD	Instruction: Dual SPI / Address-Option: Dual SPI / Data: Dual SPI
0x6	QUAD_CMD	Instruction: Quad SPI / Address-Option: Quad SPI / Data: Quad SPI
0x7		Reserved

33.8.13 Scrambling Mode

Name: SCRAMBCTRL
Offset: 0x40
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
							RANDOMDIS	ENABLE
Access							R/W	R/W
Reset							0	0

Bit 1 – RANDOMDIS Scrambling/Unscrambling Random Value Disable

Value	Description
0	The scrambling/unscrambling algorithm includes the scrambling user key plus a random value that may differ from chip to chip.
1	The scrambling/unscrambling algorithm includes only the scrambling user key.

Bit 0 – ENABLE Scrambling/Unscrambling Enable

This bit defines if the scrambling/unscrambling is enabled or disabled.

Value	Description
0	Scrambling/unscrambling is disabled.
1	Scrambling/unscrambling is enabled.

33.8.14 Scrambling Key

Name: SCRAMBKEY
Offset: 0x44
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
	KEY[31:24]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	KEY[23:16]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	KEY[15:8]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	KEY[7:0]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – KEY[31:0] Scrambling User Key
This field defines the user key value.

34. Configurable Custom Logic (CCL)

34.1 Overview

The Configurable Custom Logic (CCL) is a programmable logic peripheral which can be connected to the device pins, to events, or to other internal peripherals. This allows the user to eliminate logic gates for simple glue logic functions on the PCB.

Each LookUp Table (LUT) consists of three inputs, a truth table, an optional synchronizer/filter, and an optional edge detector. Each LUT can generate an output as a user programmable logic expression with three inputs. Inputs can be individually masked.

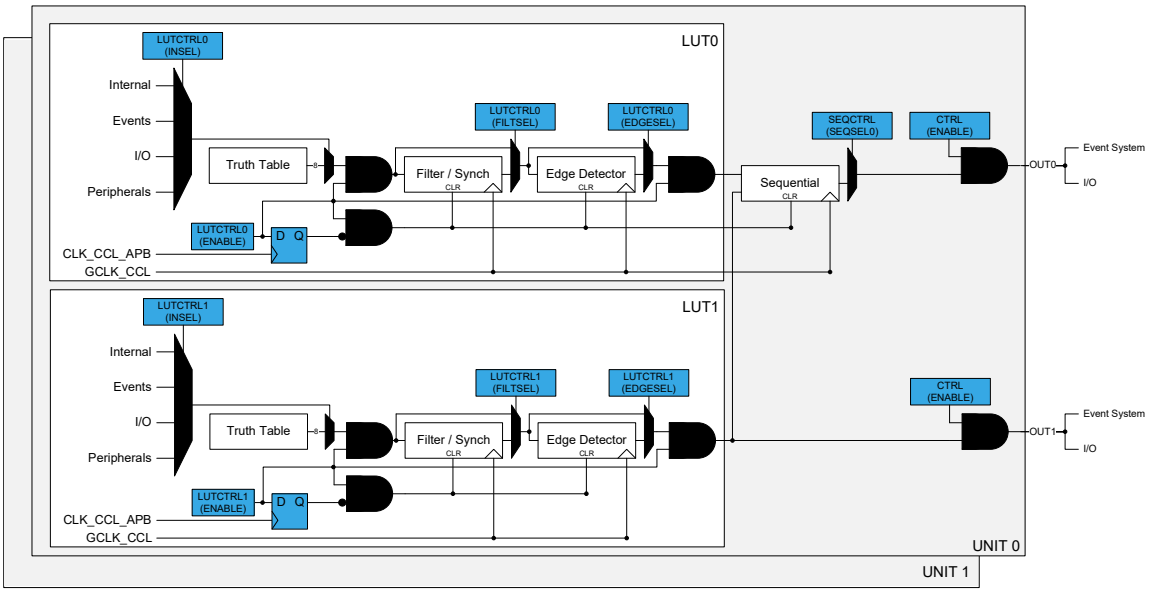
The output can be combinatorially generated from the inputs, and can be filtered to remove spikes. Optional sequential logic can be used. The inputs of the sequential module are individually controlled by two independent, adjacent LUT (LUT0/LUT1) outputs, enabling complex waveform generation.

34.2 Features

- Glue logic for general purpose PCB design
- Two programmable Look-up Tables (LUTs)
- Combinatorial logic functions: AND, NAND, OR, NOR, XOR, XNOR, NOT
- Sequential logic functions: Gated D Flip-Flop, JK Flip-Flop, gated D Latch, RS Latch
- Flexible LUT inputs selection:
  - I/Os
  - Events
  - Internal peripherals
  - Subsequent LUT output
- Output can be connected to the I/O pins or the Event System
- Optional synchronizer, filter or edge detector available on each LUT output

34.3 Block Diagram

Figure 34-1. Configurable Custom Logic



34.4 Signal Description

Pin Name	Type	Description
OUT[1:0]	Digital output	Output from lookup table
IN[5:0]	Digital input	Input to lookup table

For details on the pin mapping for this peripheral, see *I/O Ports and Peripheral Pin Select (PPS)* from Related Links. One signal can be mapped on several pins.

Related Links

6. I/O Ports and Peripheral Pin Select (PPS)

34.5 Product Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described below.

34.5.1 I/O Lines

The CCL can take inputs and generate output through I/O pins. For this to function properly, the I/O pins must be configured to be used by a Look Up Table (LUT).

34.5.2 Power Management

This peripheral can continue to operate in any Sleep mode where its source clock is running. Events connected to the event system can trigger other operations in the system without exiting Sleep modes.

34.5.3 Clocks

A generic clock (GCLK_CCL) is optionally required to clock the CCL. This clock must be configured and enabled in the Generic Clock Controller (GCLK) before using input events, filter, edge detection or sequential logic. GCLK_CCL is required when input events, a filter, an edge detector or a sequential sub-module is enabled.

This generic clock is asynchronous to the user interface clock (PB2_CLK).

34.5.4 DMA

Not applicable.

34.5.5 Interrupts

Not applicable.

34.5.6 Events

The CCL can use events from other peripherals and generate events that can be used by other peripherals. For this feature to function, the events have to be configured properly. Refer to the Related Links below for more information about the event users and event generators.

Related Links

28. Event System (EVSYS)

34.5.7 Debug Operation

When the CPU is halted in Debug mode the CCL continues normal operation. However, the CCL cannot be halted when the CPU is halted in Debug mode. If the CCL is configured in a way that requires it to be periodically serviced by the CPU, improper operation or data loss may result during debugging.

34.5.8 Register Access Protection

All registers with write access can be write-protected optionally by the Peripheral Access Controller (PAC). See *Peripheral Access Controller (PAC)* from Related Links.

Optional write protection by the Peripheral Access Controller (PAC) is denoted by the "PAC Write Protection" property in each individual register description.

PAC write protection does not apply to accesses through an external debugger.

Related Links
26. Peripheral Access Controller (PAC)

34.5.9 Analog Connections

Not applicable.

34.6 Functional Description

34.6.1 Principle of Operation

Configurable Custom Logic (CCL) is a programmable logic block that can use the device port pins, internal peripherals, and the internal Event System as both input and output channels. The CCL can serve as glue logic between the device and external devices. The CCL can eliminate the need for external logic component and can also help the designer overcome challenging real-time constrains by combining core independent peripherals in clever ways to handle the most time critical parts of the application independent of the CPU.

34.6.2 Operation

34.6.2.1 Initialization

The following bits are enable-protected, meaning that they can only be written when the corresponding even LUT is disabled (LUTCTRLx.ENABLE=0):

- Sequential Selection bits in the Sequential Control x (SEQCTRLx.SEQSEL) register

The following registers are enable-protected, meaning that they can only be written when the corresponding LUT is disabled (LUTCTRLx.ENABLE=0):

- LUT Control x (LUTCTRLx) register, except the ENABLE bit

Enable-protected bits in the LUTCTRLx registers can be written at the same time as LUTCTRLx.ENABLE is written to '1', but not at the same time as LUTCTRLx.ENABLE is written to '0'.

Enable-protection is denoted by the Enable-Protected property in the register description.

34.6.2.2 Enabling, Disabling, and Resetting

The CCL is enabled by writing a '1' to the Enable bit in the Control register (CTRL.ENABLE). The CCL is disabled by writing a '0' to CTRL.ENABLE.

Each LUT is enabled by writing a '1' to the Enable bit in the LUT Control x register (LUTCTRLx.ENABLE). Each LUT is disabled by writing a '0' to LUTCTRLx.ENABLE.

The CCL is reset by writing a '1' to the Software Reset bit in the Control register (CTRL.SWRST). All registers in the CCL will be reset to their initial state, and the CCL will be disabled. Refer to 34.8.1. CTRL for details.

34.6.2.3 Lookup Table Logic

The lookup table in each LUT unit can generate any logic expression OUT as a function of three inputs (IN[2:0]), as shown in Figure 34-2. One or more inputs can be masked. The truth table for the expression is defined by TRUTH bits in LUT Control x register (LUTCTRLx.TRUTH).

Figure 34-2. Truth Table Output Value Selection

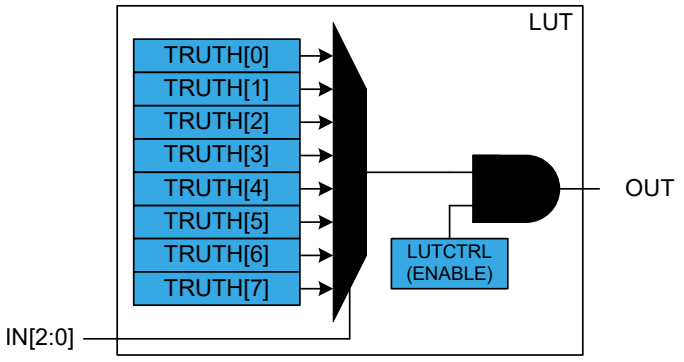


Table 34-1. Truth Table of LUT

IN[2]	IN[1]	IN[0]	OUT
0	0	0	TRUTH[0]
0	0	1	TRUTH[1]
0	1	0	TRUTH[2]
0	1	1	TRUTH[3]
1	0	0	TRUTH[4]
1	0	1	TRUTH[5]
1	1	0	TRUTH[6]
1	1	1	TRUTH[7]

34.6.2.4 Truth Table Inputs Selection

Input Overview

The inputs can be individually:

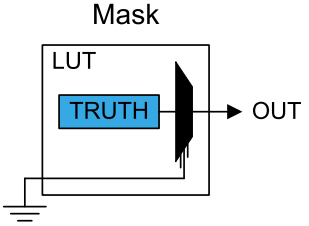
- Masked
- Driven by peripherals:
  - Analog comparator output (AC)
  - Timer/Counters waveform outputs (TC)
  - Serial Communication output transmit interface (SERCOM)
- Driven by internal events from Event System
- Driven by other CCL sub-modules

The Input Selection for each input 'y' of LUT x is configured by writing the Input 'y' Source Selection bit in the LUT x Control register (LUTCTRLx.INSELy).

Masked Inputs (MASK)

When a LUT input is masked (LUTCTRLx.INSELy = MASK), the corresponding TRUTH input (IN) is internally tied to zero, as shown in this figure:

Figure 34-3. Masked Input Selection



Internal Feedback Inputs (FEEDBACK)

When selected (LUTCTRLx.INSELy = FEEDBACK), the Sequential (SEQ) output is used as input for the corresponding LUT.

The output from an internal sequential sub-module can be used as input source for the LUT, see figure below for an example for LUT0 and LUT1. The sequential selection for each LUT follows the formula:

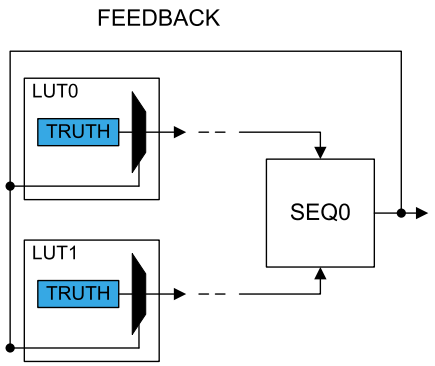
$IN[2N][i] = \mathrm{SEQ}[N]$

$IN[2N+1][i] = \mathrm{SEQ}[N]$

With *N* representing the sequencer number and *i*=0,1,2 representing the LUT input index.

See *Sequential Logic* from Related Links.

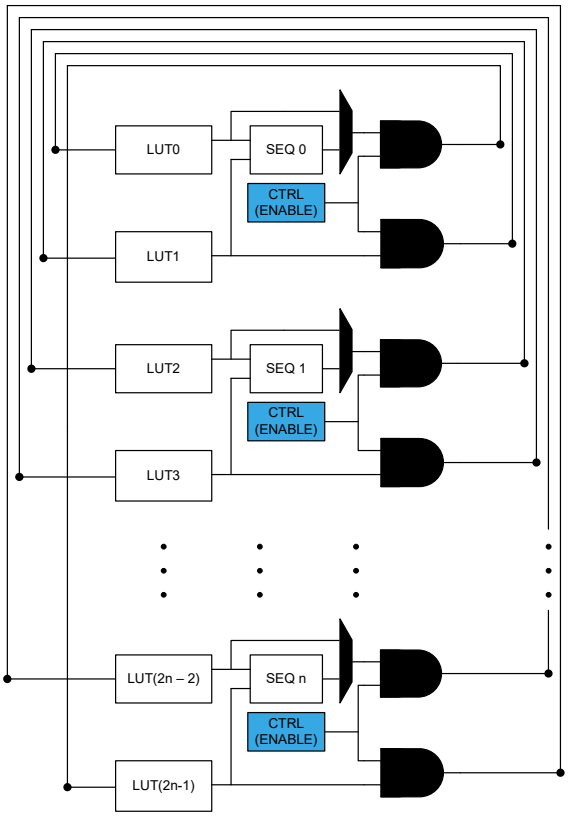
Figure 34-4. Feedback Input Selection



Linked LUT (LINK)

When selected (LUTCTRLx.INSELy=LINK), the subsequent LUT output is used as the LUT input (for example, LUT2 is the input for LUT1), as shown in the figure below:

Figure 34-5. Linked LUT Input Selection



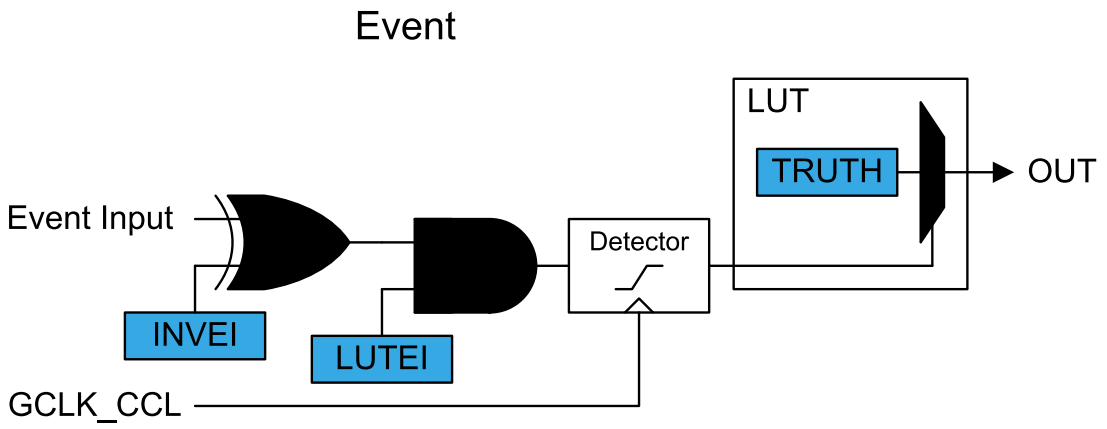
Internal Events Inputs Selection (EVENT)

Asynchronous events from the Event System can be used as input selection, as shown in the following figure. For each LUT, one event input line is available and can be selected on each LUT input. Before enabling the event selection by writing LUTCTRLx.INSELy=EVENT, the Event System must be configured first.

By default, CCL includes an edge detector. When the event is received, an internal strobe is generated when a rising edge is detected. The pulse duration is one GCLK_CCL clock cycle. Writing the LUTCTRLx.INSELy=ASYNCEVENT will disable the edge detector. In this case, it is possible to combine an asynchronous event input with any other input source. This is typically useful with event levels inputs for example, (external I/O pin events). The following steps ensure proper operation:

1. Enable the GCLK_CCL clock.
2. Configure the Event System to route the event asynchronously.
3. Select the event input type (LUTCTRLx.INSEL = ASYNCEVENT).
4. If a strobe must be generated on the event input falling edge, write a '1' to the Inverted Event Input Enable bit in the LUT Control register (LUTCTRLx.INVEI) .
5. Enable the event input by writing the Event Input Enable bit in the LUT Control register (LUTCTRLx.LUTEI) to '1'.

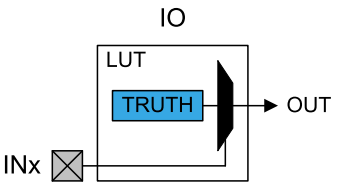
Figure 34-6. Event Input Selection



I/O Pin Inputs (IO)

When the I/O pin is selected as LUT input (LUTCTRLx.INSELy = IO), the corresponding LUT input will be connected to the pin, as shown in the figure below.

Figure 34-7. I/O Pin Input Selection



Analog Comparator Inputs (AC)

The AC outputs can be used as input source for the LUT (LUTCTRLx.INSELy=AC).

The analog comparator outputs are distributed following the formula:

$$IN[N]\,[i] \;=\; AC[N\,\%\;ComparatorOutput\_Number]$$

With *N* representing the LUT number and *i*=[0,1,2] representing the LUT input index.

Before selecting the comparator output, the AC must be configured first.

Figure 34-8. AC Input Selection

Timer/Counter Inputs (TC)

The TC waveform output WO[0] can be used as input source for the LUT (LUTCTRLx.INSELy = TC). Only consecutive instances of the TC, that is, TCx and the subsequent TC(x+1), are available as default and alternative TC selections (for example, TC0 and TC1 are sources for LUT0, TC1 and TC2 are sources for LUT1). See the figure below for an example for LUT0. More general, the Timer/Counter selection for each LUT follows the formula:

$$\mathrm{IN}[N][i] = DefaultTC[N\,\%\,\mathrm{TC\_Instance\_Number}]$$

$$\mathrm{IN}[N][i] = AlternativeTC[(N+1)\,\%\,\mathrm{TC\_Instance\_Number}]$$

Where *N* represents the LUT number and *i* represents the LUT input index (i=0,1,2).

Before selecting the waveform outputs, the TC must be configured first.

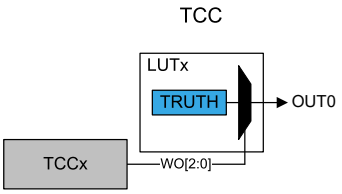
Figure 34-9. TC Input Selection

Timer/Counter for Control Application Inputs (TCC)

The TCC waveform outputs can be used as input source for the LUT. Only WO[2:0] outputs can be selected and routed to the respective LUT input (that is, IN0 is connected to WO0, IN1 to WO1, and IN2 to WO2), as shown in the figure below.

Before selecting the waveform outputs, the TCC must be configured first.

Figure 34-10. TCC Input Selection



Serial Communication Output Transmit Inputs (SERCOM)

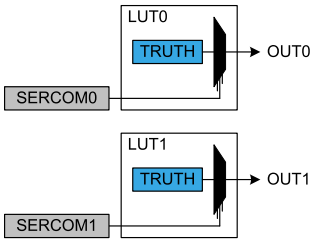
The serial engine transmitter output from Serial Communication Interface (SERCOM TX, TXd for USART, MOSI for SPI) can be used as input source for the LUT. The figure below shows an example for LUT0 and LUT1. The SERCOM selection for each LUT follows the formula:

$$\mathrm{IN}[N][i] = SERCOM[N \,\%\, \mathrm{SERCOM\_Instance\_Number}]$$

With *N* representing the LUT number and *i*=0,1,2 representing the LUT input index.

Before selecting the SERCOM as input source, the SERCOM must be configured first: the SERCOM TX signal must be output on SERCOMn/pad[0], which serves as input pad to the CCL.

Figure 34-11. SERCOM Input Selection



Related Links

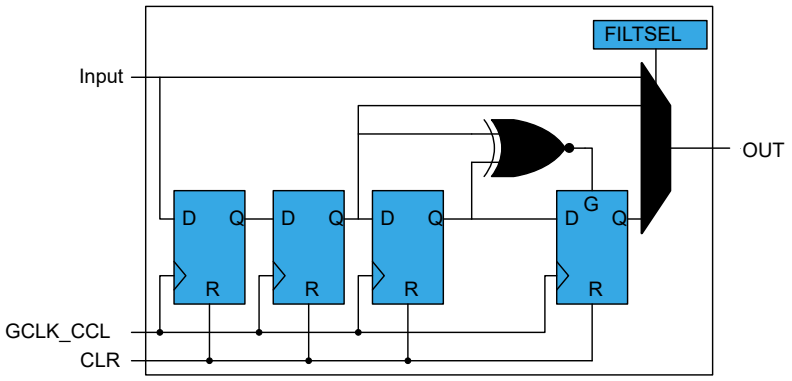
34.6.2.7. Sequential Logic

34.6.2.5 Filter

By default, the LUT output is a combinatorial function of the LUT inputs. This may cause some short glitches when the inputs change value. These glitches can be removed by clocking through filters, if demanded by application needs.

The Filter Selection bits in LUT Control register (LUTCTRLx.FILTSEL) define the synchronizer or digital filter options. When a filter is enabled, the OUT output will be delayed by two to five GCLK cycles. One APB clock after the corresponding LUT is disabled, all internal filter logic is cleared.
Note: Events used as LUT input will also be filtered, if the filter is enabled.

Figure 34-12. Filter



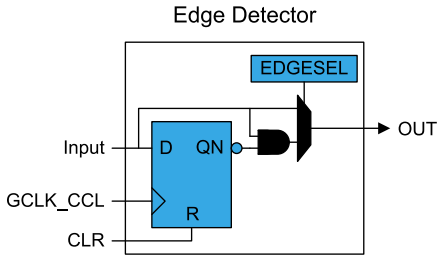
34.6.2.6 Edge Detector

The edge detector can be used to generate a pulse when detecting a rising edge on its input. To detect a falling edge, the TRUTH table must be inverted.

The edge detector is enabled by writing '1' to the Edge Selection bit in LUT Control register (LUTCTRLx.EDGESEL). In order to avoid unpredictable behavior, either the filter or synchronizer must be enabled.

Edge detection is disabled by writing a '0' to LUTCTRLx.EDGESEL. After disabling a LUT, the corresponding internal Edge Detector logic is cleared one APB clock cycle later.

Figure 34-13. Edge Detector



34.6.2.7 Sequential Logic

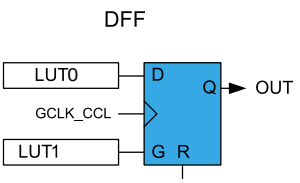
Each LUT pair can be connected to the internal sequential logic, which can be configured to work as D flip flop, JK flip flop, gated D-latch or RS-latch by writing the Sequential Selection bits on the corresponding Sequential Control x register (SEQCTRLx.SEQSEL). Before using sequential logic, the GCLK_CCL clock and optionally each LUT filter or edge detector must be enabled.

Note: While configuring the sequential logic, the even LUT must be disabled. When configured, the even LUT must be enabled.

Gated D Flip-Flop (DFF)

When the DFF is selected, the D-input is driven by the even LUT output LUT0, and the G-input is driven by the odd LUT output LUT1, as shown in the following figure.

Figure 34-14. D Flip Flop



When the even LUT is disabled LUTCTRL0.ENABLE=0, the flip-flop is asynchronously cleared. The reset command (R) is kept enabled for one APB clock cycle. In all other cases, the flip-flop output (OUT) is refreshed on rising edge of the GCLK_CCL, as shown in the following table.

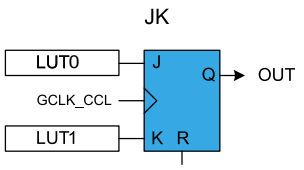
Table 34-2. DFF Characteristics

R	G	D	OUT
1	X	X	Clear
0	1	1	Set
		0	Clear
	0	X	Hold state (no change)

JK Flip-Flop (JK)

When this configuration is selected, the J-input is driven by the even LUT output LUT0, and the K-input is driven by the odd LUT output LUT1, as shown in the following figure.

Figure 34-15. JK Flip Flop



When the even LUT is disabled LUTCTRL0.ENABLE=0, the flip-flop is asynchronously cleared. The reset command (R) is kept enabled for one APB clock cycle. In all other cases, the flip-flop output (OUT) is refreshed on rising edge of the GCLK_CCL, as shown in the following table.

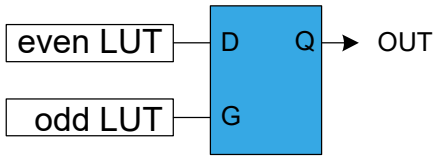
Table 34-3. JK Characteristics

R	J	K	OUT
1	X	X	Clear
0	0	0	Hold state (no change)
0	0	1	Clear
0	1	0	Set
0	1	1	Toggle

Gated D-Latch (DLATCH)

When the DLATCH is selected, the D-input is driven by the even LUT output LUT0, and the G-input is driven by the odd LUT output LUT1, as shown in the following figure.

Figure 34-16. D-Latch



When the even LUT is disabled LUTCTRL0.ENABLE=0, the latch output will be cleared. The G-input is forced enabled for one more APB clock cycle, and the D-input to zero. In all other cases, the latch output (OUT) is refreshed as shown in the following table.

Table 34-4. D-Latch Characteristics

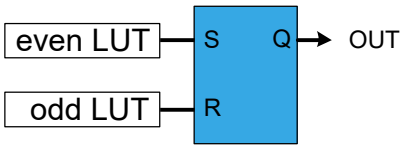
G	D	OUT
0	X	Hold state (no change)

.....continued		
G	D	OUT
1	0	Clear
1	1	Set

RS Latch (RS)

When this configuration is selected, the S-input is driven by the even LUT output LUT0, and the R-input is driven by the odd LUT output LUT1, as shown in the following figure.

Figure 34-17. RS-Latch



When the even LUT is disabled LUTCTRL0.ENABLE=0, the latch output will be cleared. The R-input is forced enabled for one more APB clock cycle and S-input to zero. In all other cases, the latch output (OUT) is refreshed as shown in the following table.

Table 34-5. RS-Latch Characteristics

S	R	OUT
0	0	Hold state (no change)
0	1	Clear
1	0	Set
1	1	Forbidden state

34.6.3 Events

The CCL can generate the following output events:

- LUTn where n=0-1: Lookup Table Output Value

Writing a '1' to the LUT Control Event Output Enable bit (LUTCTRL.LUTEO) enables the corresponding output event. Writing a '0' to this bit disables the corresponding output event.

The CCL can take the following actions on an input event:

- INSELx where x=0-2: The event is used as input for the TRUTH table. For additional information, refer to 34.5.6. Events.

Writing a '1' to the LUT Control Event Input Enable bit (LUTCTRL.LUTEI) enables the corresponding action on input event. Writing a '0' to this bit disables the corresponding action on input event.

Related Links

28. Event System (EVSYS)

34.6.4 Sleep Mode Operation

When using the GCLK_CCL internal clocking, writing the Run In Standby bit in the Control register (CTRL.RUNSTDBY) to '1' will allow GCLK_CCL to be enabled in Standby Sleep mode.

If CTRL.RUNSTDBY=0, the GCLK_CCL will be disabled in Standby Sleep mode. If the Filter, Edge Detector or Sequential logic are enabled, the LUT output will be forced to zero in STANDBY mode. In all other cases, the TRUTH table decoder will continue operation and the LUT output will be refreshed accordingly.

34.7 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRL	7:0		RUNSTDBY					ENABLE	SWRST
0x01 ... 0x03	Reserved									
0x04	SEQCTRLX	7:0					SEQSEL[3:0]			
0x05 ... 0x07	Reserved									
0x08	LUTCTRL0	7:0	EDGESEL		FILTSEL[1:0]				ENABLE	
		15:8	INSEL1[3:0]				INSEL0[3:0]			
		23:16		LUTEO	LUTEI	INVEI	INSEL2[3:0]			
		31:24	TRUTH[7:0]							
0x0C	LUTCTRL1	7:0	EDGESEL		FILTSEL[1:0]				ENABLE	
		15:8	INSEL1[3:0]				INSEL0[3:0]			
		23:16		LUTEO	LUTEI	INVEI	INSEL2[3:0]			
		31:24	TRUTH[7:0]							

34.8 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers are optionally write-protected by the Peripheral Access Controller (PAC). Optional PAC write protection is denoted by the "PAC Write-Protection" property in each individual register description. See *Register Access Protection* from Related Links.

Some registers are enable protected, meaning they can only be written when the peripheral is disabled. Enable protection is denoted by the “Enable-Protected” property in each individual register description.

Related Links

34.5.8. Register Access Protection

34.8.1 Control

Name: CTRL
Offset: 0x00
Reset: 0x00
Property: PAC Write-Protection

Note: CTRL register (except the bits ENABLE & SWRST) is Enable Protected when CCL.CTRL.ENABLE = 1.

Bit	7	6	5	4	3	2	1	0
		RUNSTDBY					ENABLE	SWRST
Access		R/W					R/W	W
Reset		0					0	0

Bit 6 – RUNSTDBY Run in Standby
This bit indicates if the GCLK_CCL clock must be kept running in standby mode. The setting is ignored for configurations where the generic clock is not required. For details refer to 34.6.4. Sleep Mode Operation.

Important: This bit must be written before enabling the CCL.

Value	Description
0	Generic clock is not required in standby sleep mode.
1	Generic clock is required in standby sleep mode.

Bit 1 – ENABLE Enable

Value	Description
0	The peripheral is disabled.
1	The peripheral is enabled.

Bit 0 – SWRST Software Reset
Writing a '0' to this bit has no effect.
Writing a '1' to this bit resets all registers in the CCL to their initial state.

Value	Description
0	There is no reset operation ongoing.
1	The reset operation is ongoing.

34.8.2 Sequential Control X

Name: SEQCTRLX
Offset: 0x04
Reset: 0x00
Property: PAC Write-Protection, Enable-protected

Note: SEQCTRLX register is Enable-protected when CCL.LUTCTRL0.ENABLE = 1.

Bit	7	6	5	4	3	2	1	0
					SEQSEL[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bits 3:0 – SEQSEL[3:0] Sequential Selection
These bits select the sequential configuration:
Sequential Selection

Value	Name	Description
0x0	DISABLE	Sequential logic is disabled
0x1	DFF	D flip flop
0x2	JK	JK flip flop
0x3	LATCH	D latch
0x4	RS	RS latch
0x5 - 0xF	—	Reserved

34.8.3 LUT Control n

Name: LUTCTRL
Offset: 0x08 + n*0x04 [n=0..1]
Reset: 0x00000000
Property: PAC Write-Protection, Enable-protected

Note: The LUTCTRLn register is Enable Protected when CCL.LUTCTRLn.ENABLE = 1.

Bit	31	30	29	28	27	26	25	24
	TRUTH[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
		LUTEO	LUTEI	INVEI	INSEL2[3:0]			
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	INSEL1[3:0]				INSEL0[3:0]			
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	EDGESEL		FILTSEL[1:0]				ENABLE	
Access	R/W		R/W	R/W			R/W	
Reset	0		0	0			0	

Bits 31:24 – TRUTH[7:0] Truth Table
These bits define the value of truth logic as a function of inputs IN[2:0].

Bit 22 – LUTEO LUT Event Output Enable

Value	Description
0	LUT event output is disabled.
1	LUT event output is enabled.

Bit 21 – LUTEI LUT Event Input Enable

Value	Description
0	LUT incoming event is disabled.
1	LUT incoming event is enabled.

Bit 20 – INVEI Inverted Event Input Enable

Value	Description
0	Incoming event is not inverted.
1	Incoming event is inverted.

Bits 8:11, 12:15, 16:19 – INSELx LUT Input x Source Selection
These bits select the LUT input x source.

Value	Name	Description
0x0	MASK	Masked input
0x1	FEEDBACK	Feedback input source
0x2	LINK	Linked LUT input source
0x3	EVENT	Event input source
0x4	IO	I/O pin input source

Value	Name	Description
0x5	AC	AC input source: CMP[0] (LUT0) / CMP[1] (LUT1)
0x6	TC	TC input source: TC0 WO[0] (LUT0) / TC1 WO[0] (LUT1)
0x7	ALTTC	Alternative TC input source: TC1 WO[0] (LUT0) / TC2 WO[0] (LUT1)
0x8	TCC	TCC input source: TCC0 (LUT0) / TCC1 (LUT1)
0x9	SERCOM	SERCOM input source: SERCOM0 PAD0 (LUT0) / SERCOM1 PAD0 (LUT1)
0xA	ALT2TC	1'b0
0xB	ASYNCEVENT	1'b0
0xC - 0xF	Reserved	Reserved

Bit 7 – EDGESEL Edge Selection

Value	Description
0	Edge detector is disabled.
1	Edge detector is enabled.

Bits 5:4 – FILTSEL[1:0] Filter Selection

These bits select the LUT output filter options:

Filter Selection

Value	Name	Description
0x0	DISABLE	Filter disabled
0x1	SYNCH	Synchronizer enabled
0x2	FILTER	Filter enabled
0x3	—	Reserved

Bit 1 – ENABLE LUT Enable

Note: Prevents/protects write access to the other bits in the LUTCTRL registers.

Value	Description
0	The LUT is disabled.
1	The LUT is enabled.

35. True Random Number Generator (TRNG)

35.1 Overview

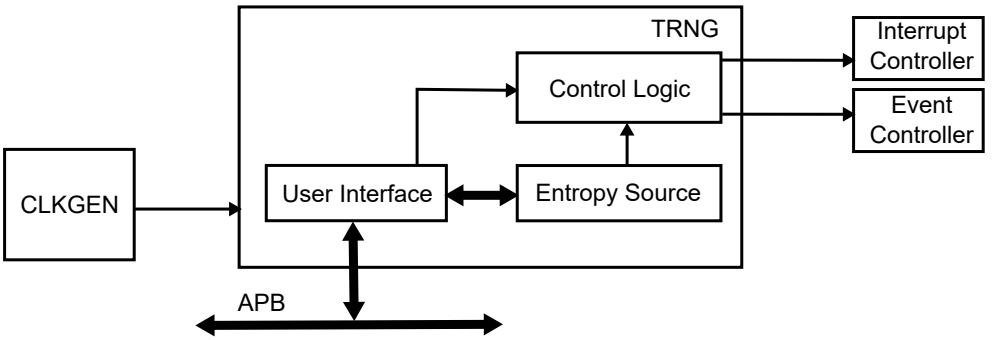
The True Random Number Generator (TRNG) generates unpredictable random numbers that are not generated by an algorithm.

35.2 Features

- Provides a 32-bit random number for every 84-clock cycles,

35.3 Block Diagram

Figure 35-1. TRNG Block Diagram.



35.4 Signal Description

Not applicable.

35.5 Product Dependencies

In order to use this peripheral, other parts of the system must be configured correctly, as described as follows.

35.5.1 I/O Lines

Not applicable.

35.5.2 Power Management

The functioning of TRNG depends on the Sleep mode of the device.

The TRNG interrupts can be used to wake up the device from sleep modes. Events connected to the event system can trigger other operations in the system without exiting sleep modes.

Related Links

35.6.5. Sleep Mode Operation

35.5.3 Clocks

The TRNG bus clock () can be enabled and disabled in the CRU module or PMD3.RNGMD bit (see *Peripheral Module Disable Register (PMD)* from Related Links).

Related Links

20. Peripheral Module Disable Register (PMD)

35.5.4 DMA

Not applicable.

35.5.5 Interrupts

The interrupt request line is connected to the interrupt controller. Using the TRNG interrupt(s) requires the interrupt controller to be configured first. See *Nested Vector Interrupt Controller (NVIC)* from Related Links.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)

35.5.6 Events

TRNG can generate Events that are used by the Event System (EVSYS) and EVSYS users.

TRNG cannot use any Events from other peripherals, as it is not an Event User.

Related Links

28. Event System (EVSYS)

35.5.7 Debug Operation

When the CPU is halted in debug mode the TRNG continues normal operation. If the TRNG is configured in a way that requires it to be periodically serviced by the CPU through interrupts or similar, improper operation or data loss may result during debugging.

35.5.8 Register Access Protection

All registers with write access are optionally write-protected by the Peripheral Access Controller (PAC), except the following register:

- Interrupt Flag Status and Clear (INTFLAG) register

Optional write protection by the Peripheral Access Controller (PAC) is denoted by the "PAC Write Protection" property in each individual register description.

35.5.9 Analog Connections

Not applicable.

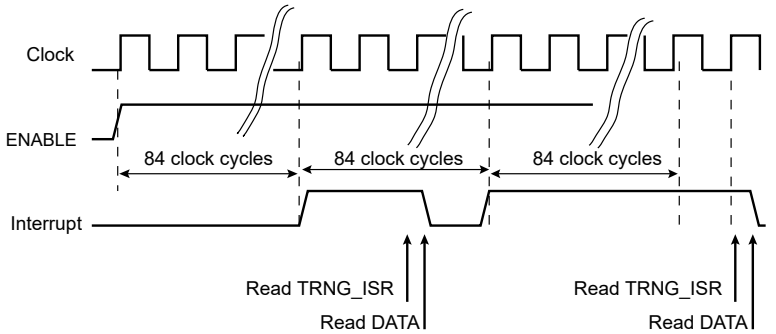
35.6 Functional Description

35.6.1 Principle of Operation

When the TRNG is enabled, the peripheral starts providing new 32-bit random numbers every 84 PB2_CLK clock cycles.

The TRNG can be configured to generate an interrupt or event when a new random number is available.

Figure 35-2. TRNG Data Generation Sequence



35.6.2 Basic Operation

35.6.2.1 Initialization

To operate the TRNG, do the following:

- Ensure PB2_CLK is enabled in the CRU and TRNG is enabled in the PMD3 register, PMD3.RNGMD bit.
- Optional: Enable the output event by writing a '1' to the EVCTRL.DATARDYEO bit.
- Optional: Enable the TRNG to Run in Standby sleep mode by writing a '1' to CTRLA.RUNSTDBY.
- Enable the TRNG operation by writing a '1' to CTRLA.ENABLE.

35.6.2.2 Enabling, Disabling and Resetting

The TRNG is enabled by writing '1' to the Enable bit in the Control A register (CTRLA.ENABLE). The TRNG is disabled by writing a zero to CTRLA.ENABLE.

35.6.3 Interrupts

The TRNG has the following interrupt source:

- Data Ready (DATARDY): Indicates that a new random number is available in the DATA register and ready to be read.
  This interrupt is a synchronous wake-up source. See *Sleep Mode Controller* for details.

The interrupt source has an interrupt flag associated with it. The interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.DATARDY) is set to '1' when the interrupt condition occurs. The interrupt can be enabled by writing a '1' to the corresponding bit in the Interrupt Enable Set register (INTENSET.DATARDY), and disabled by writing a '1' to the corresponding bit in the Interrupt Enable Clear (INTENCLR) register.

An interrupt request is generated when the interrupt flag is set and the corresponding interrupt is enabled. The interrupt request remains active until the interrupt flag is cleared, or the interrupt is disabled. See *INTFLAG* register from Related Links for details on how to clear interrupt flags.

Note that interrupts must be globally enabled for interrupt requests to be generated.

Related Links

35.8.5. INTFLAG

35.6.4 Events

The TRNG can generate the following output event:

- Data Ready (DATARDY): Generated when a new random number is available in the DATA register.

Writing '1' to the Data Ready Event Output bit in the Event Control Register (EVCTRL.DATARDYEO) enables the DTARDY event. Writing a '0' to this bit disables the corresponding output event. Refer to *EVSYS – Event System* for details on configuring the Event System.

Related Links

28. Event System (EVSYS)

35.6.5 Sleep Mode Operation

The Run in Standby bit in Control A register (CTRLA.RUNSTDBY) controls the behavior of the TRNG during standby sleep mode:

When this bit is '0', the TRNG is disabled during sleep, but maintains its current configuration.

When this bit is '1', the TRNG continues to operate during sleep and any enabled TRNG interrupt source can wake up the CPU.

35.7 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0		RUNSTDBY					ENABLE	
0x01 ... 0x03	Reserved									
0x04	EVCTRL	7:0								DATARDYEO
0x05 ... 0x07	Reserved									
0x08	INTENCLR	7:0								DATARDY
0x09	INTENSET	7:0								DATARDY
0x0A	INTFLAG	7:0								DATARDY
0x0B ... 0x1F	Reserved									
0x20	DATA	7:0	DATA[7:0]							
		15:8	DATA[15:8]							
		23:16	DATA[23:16]							
		31:24	DATA[31:24]							

35.8 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register, and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers require synchronization when read and/or written. Synchronization is denoted by the “Read-Synchronized” and/or “Write-Synchronized” property in each individual register description.

Optional write protection by the Peripheral Access Controller (PAC) is denoted by the "PAC Write Protection" property in each individual register description.

Some registers are enable-protected, meaning they can only be written when the module is disabled. Enable-protection is denoted by the “Enable-Protected” property in each individual register description.

35.8.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
		RUNSTDBY					ENABLE	
Access		R/W					R/W	
Reset		0					0	

Bit 6 – RUNSTDBY Run in Standby

This bit controls how the TRNG behaves during standby sleep mode:

Value	Description
0	The TRNG is halted during standby sleep mode.
1	The TRNG is not stopped in standby sleep mode.

Bit 1 – ENABLE Enable

Value	Description
0	The TRNG is disabled.
1	The TRNG is enabled.

35.8.2 Event Control

Name: EVCTRL
Offset: 0x04
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
								DATARDYEO
Access								R/W
Reset								0

Bit 0 – DATARDYEO Data Ready Event Output
This bit indicates whether the Data Ready event output is enabled and whether an output event will be generated when a new random value is ready.

Value	Description
0	Data Ready event output is disabled and an event will not be generated.
1	Data Ready event output is enabled and an event will be generated.

35.8.3 Interrupt Enable Clear

Name: INTENCLR
Offset: 0x08
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to disable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Set (INTENSET) register.

Bit	7	6	5	4	3	2	1	0
								DATARDY
Access								R/W
Reset								0

Bit 0 – DATARDY Data Ready Interrupt Enable
Writing a '1' to this bit will clear the Data Ready Interrupt Enable bit, which disables the corresponding interrupt request.

Value	Description
0	The DATARDY interrupt is disabled.
1	The DATARDY interrupt is enabled.

35.8.4 Interrupt Enable Set

Name: INTENSET
Offset: 0x09
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to enable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Clear (INTENCLR) register.

Bit	7	6	5	4	3	2	1	0
								DATARDY
Access								R/W
Reset								0

Bit 0 – DATARDY Data Ready Interrupt Enable
Writing a '1' to this bit will set the Data Ready Interrupt Enable bit, which enables the corresponding interrupt request.

Value	Description
0	The DATARDY interrupt is disabled.
1	The DATARDY interrupt is enabled.

35.8.5 Interrupt Flag Status and Clear

Name: INTFLAG
Offset: 0x0A
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
								DATARDY
Access								R/W
Reset								0

Bit 0 – DATARDY Data Ready
This flag is set when a new random value is generated, and an interrupt will be generated if INTENCLR/SET.DATARDY=1.
This flag is cleared by writing a '1' to the flag or by reading the DATA register. Writing a '0' to this bit has no effect.

35.8.6 Output Data

Name: DATA
Offset: 0x20
Reset: -
Property: -

Bit	31	30	29	28	27	26	25	24
	DATA[31:24]							
Access	R	R	R	R	R	R	R	R
Reset	–	–	–	–	–	–	–	–

Bit	23	22	21	20	19	18	17	16
	DATA[23:16]							
Access	R	R	R	R	R	R	R	R
Reset	–	–	–	–	–	–	–	–

Bit	15	14	13	12	11	10	9	8
	DATA[15:8]							
Access	R	R	R	R	R	R	R	R
Reset	–	–	–	–	–	–	–	–

Bit	7	6	5	4	3	2	1	0
	DATA[7:0]							
Access	R	R	R	R	R	R	R	R
Reset	–	–	–	–	–	–	–	–

Bits 31:0 – DATA[31:0] Output Data
These bits hold the 32-bit randomly generated output data.

36. Advanced Encryption Standard (AES)

36.1 Overview

The Advanced Encryption Standard peripheral (AES) provides a means for symmetric-key encryption of 128-bit blocks, in compliance to NIST specifications.

The symmetric-key algorithm requires the same key for both encryption and decryption.

Different key sizes are supported. The key size determines the number of repetitions of transformation rounds that convert the input (called the "plaintext") into the final output ("ciphertext"). The number of rounds of repetition is as follows:

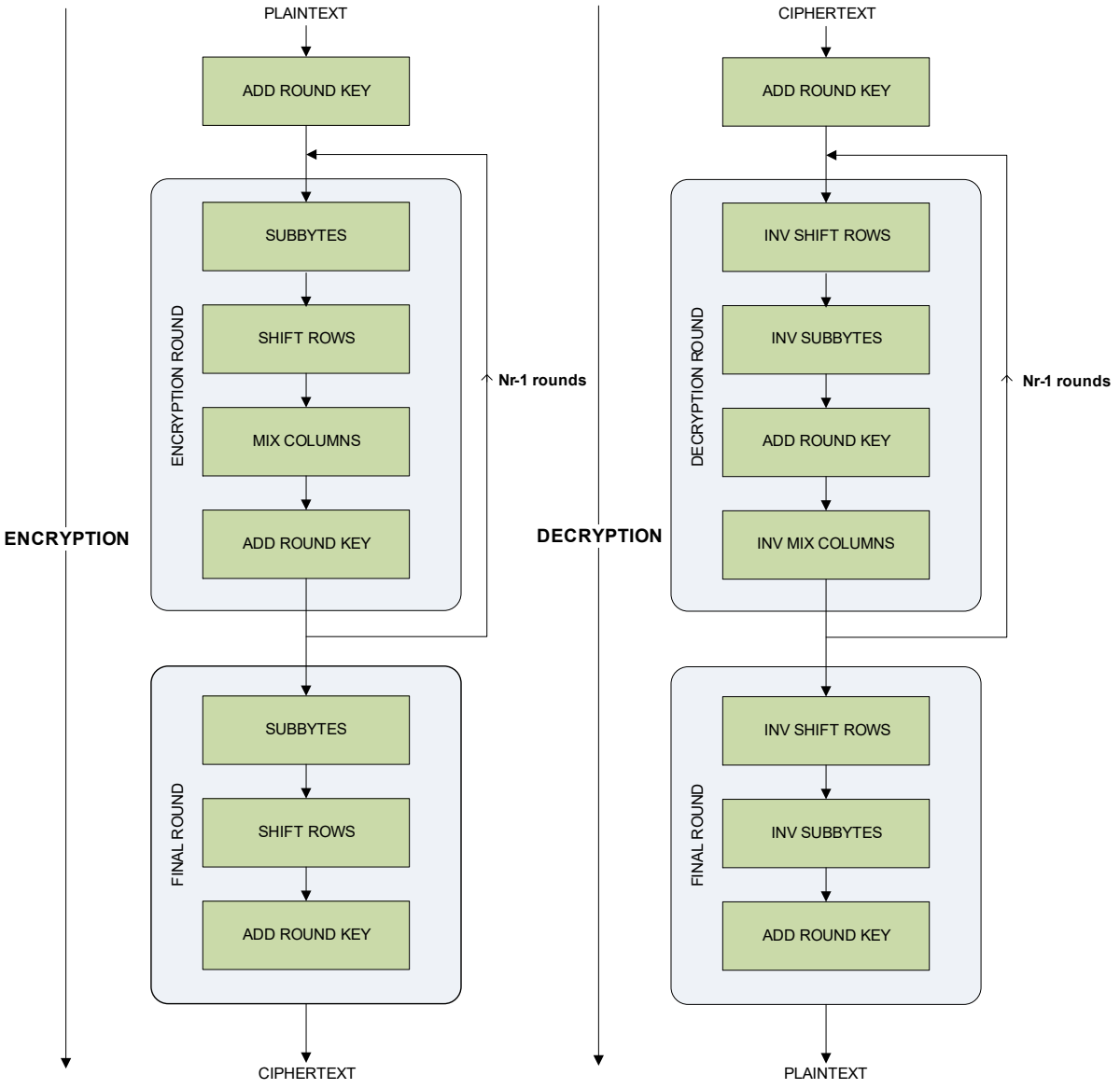
- 10 rounds of repetition for 128-bit keys
- 12 rounds of repetition for 192-bit keys
- 14 rounds of repetition for 256-bit keys

36.2 Features

- Compliant with FIPS Publication 197, Advanced Encryption Standard (AES)
- 128/192/256 bit cryptographic key supported
- Encryption time of 57/67/77 cycles with 128-bit/192-bit/256-bit cryptographic key
- Five confidentiality modes of operation as recommended in NIST Special Publication 800-38A
- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter (CTR)
- Supports Counter with CBC-MAC (CCM/CCM*) mode for authenticated encryption
- 8, 16, 32, 64, 128-bit data sizes possible in CFB mode
- Galois Counter mode (GCM) encryption and authentication

36.3 Block Diagram

Figure 36-1. AES Block Diagram



36.4 Signal Description

Not applicable.

36.5 Product Dependencies

In order to use this AES module, other parts of the system must be configured correctly, as described below.

36.5.1 I/O Lines

Not applicable.

36.5.2 Power Management

The AES will continue to operate in Standby sleep mode, if it's source clock is running.

The AES interrupts can be used to wake up the device from Standby sleep mode. Refer to the Power Manager chapter for details on the different sleep modes.

AES is clocked only on the following conditions:

- When the DMA is enabled.
- Whenever there is an APB access for any read and write operation to the AES registers. (Not in Standby sleep mode.)
- When the AES is enabled & encryption/decryption is ongoing.

Related Links

15. Power Management Unit (PMU)

36.5.3 Clocks

The AES bus clock (PB2_CLK) can be enabled and disabled in the CRU module.

36.5.4 DMA

The AES has two DMA request lines; one for input data and one for output data. They are both connected to the DMA Controller (DMAC). These DMA request triggers will be acknowledged by the DMAC ACK signals. Using the AES DMA requests requires the DMA Controller to be configured first. See *Direct Memory Access Controller (DMAC)* from Related Links.

Related Links

22. Direct Memory Access Controller (DMAC)

36.5.5 Interrupts

The interrupt request line is connected to the interrupt controller. Using the AES interrupt requires the interrupt controller to be configured first. Refer to the Processor and Architecture chapter for details.

All the AES interrupts are synchronous wake-up sources. See *Sleep Mode Controller* for details.

Related Links

10. Processor and Architecture

36.5.6 Events

Not applicable.

36.5.7 Debug Operation

When the CPU is halted in debug mode, the AES module continues normal operation. If the AES module is configured in a way that requires it to be periodically serviced by the CPU through interrupts or similar, improper operation or data loss may result during debugging. The AES module can be forced to halt operation during debugging.

36.5.8 Register Access Protection

All registers with write access are optionally write-protected by the peripheral access controller (PAC), except the following register:

- Interrupt Flag Register (INTFLAG)

Write protection is denoted by the Write-Protected property in the register description.

Write protection does not apply to accesses through an external debugger. See *Peripheral Access Controller (PAC)* from Related Links.

Related Links

26. Peripheral Access Controller (PAC)

36.5.9 Analog Connections

Not applicable.

36.6 Functional Description

36.6.1 Principle of Operation

The following is a high level description of the algorithm. These are the steps:

- KeyExpansion: Round keys are derived from the cipher key using Rijndael's key schedule.
- InitialRound:
  - AddRoundKey: Each byte of the state is combined with the round key using bitwise XOR.
- Rounds:
  - SubBytes: A non-linear substitution step where each byte is replaced with another according to a lookup table.
  - ShiftRows: A transposition step where each row of the state is shifted cyclically a certain number of steps.
  - MixColumns: A mixing operation which operates on the columns of the state, combining the four bytes in each column.
  - AddRoundKey
- Final Round (no MixColumns):
  - SubBytes
  - ShiftRows
  - AddRoundKey

The relationship between the module's clock frequency and throughput (in bytes per second) is given by:

Clock Frequency = (Throughput/2) x (Nr+1) for 2 byte parallel processing

Clock Frequency = (Throughput/4) x (Nr+1) for 4 byte parallel processing

where Nr is the number of rounds, depending on the key length.

36.6.2 Basic Operation

36.6.2.1 Initialization

The following register is enable-protected:

- Control A (CTRLA)

Enable-protection is denoted by the Enable-Protected property in the register description.

36.6.2.2 Enabling, Disabling, and Resetting

The AES module is enabled by writing a one to the Enable bit in the Control A register (CTRLA.ENABLE). The module is disabled by writing a zero to CTRLA.ENABLE. The module is reset by writing a one to the Software Reset bit in the Control A register (CTRLA.SWRST).

36.6.2.3 Basic Programming

The CIPHER bit in the Control A Register (CTRLA.CIPHER) allows selection between the encryption and the decryption processes. The AES is capable of using cryptographic keys of 128/192/256 bits to encrypt and decrypt data in blocks of 128 bits. The Key Size (128/192/256) can be programmed in the KEYSIZE field in the Control A Register (CTRLA.KEYSIZE). This 128-bit/192-bit/256-bit key is defined in the Key Word Registers (KEYWORD). By setting the XORKEY bit of CTRLA register, keyword can be updated with the resulting XOR value of user keyword and previous keyword content.

The input data for processing is written to a data buffer consisting of four 32-bit registers through the Data register address. The data buffer register (note that input and output data shares the same data buffer register) that is written to when the next write is performed is indicated by the Data Pointer in the Data Buffer Pointer (DATABUFPTR) register. This field is incremented by one or wrapped by hardware when a write to the INDATA register address is performed. This field can also be programmed, allowing the user direct control over which input buffer register to write. Note that when AES module is in the CFB operation mode with the data segment size less than 128 bits, the input data must be written to the first (DATABUFPTR = 0) and second (DATABUFPTR = 1) input buffer registers (see Table 36-1).

The input to the encryption processes of the CBC, CFB and OFB modes includes, in addition to the plaintext, a 128-bit data block called the Initialization Vector (IV), which must be set in the Initialization Vector Registers (INTVECT). Additionally, the GCM mode 128-bit authentication data needs to be programmed. The Initialization Vector is used in the initial step in the encryption of a message and in the corresponding decryption of the message. The Initialization Vector Registers are also used by the Counter mode to set the counter value.

It is necessary to notify AES module whenever the next data block it is going to process is the beginning of a new message. This is done by writing a one to the New Message bit in the Control B register (CTRLB.NEWMSG).

The AES modes of operation are selected by setting the AESMODE field in the Control A Register (CTRLA.AESMODE). In Cipher Feedback Mode (CFB), five data sizes are possible (8, 16, 32, 64 or 128 bits), configurable by means of the CFBS field in the Control A Register (CTRLA.CFBS). In Counter mode, the size of the block counter embedded in the module is 16 bits. Therefore, there is a rollover after processing 1 megabyte of data. The data pre-processing, post-processing and data chaining for the concerned modes are automatically performed by the module.

When data processing has completed, the Encryption Complete bit in the Interrupt Flag register (INTFLAG.ENCCMP) is set by hardware (which triggers an interrupt request if the corresponding interrupt is enabled). The processed output data is read out through the Output Data register (INDATA) address from the data buffer consisting of four 32-bit registers. The data buffer register that is read when the next read is performed is indicated by the Data Pointer field in the Data Buffer Pointer register (DATABUFPTR). This field is incremented by one or wrapped by hardware when a read from the INDATA register address is performed. This field can be programmed, giving the user direct control over which output buffer register to read from. Note that when AES module is in the CFB operation mode with the data segment size less than 128 bits, the output data must be read from the first (DATABUFPTR = 0) and second (DATABUFPTR = 1) output buffer registers (see Table 36-1). The Encryption Complete bit (INTFLAG.ENCCMP) is cleared by hardware after the processed data has been read from the relevant output buffer registers.

Table 36-1. Relevant Input/Output Data Registers for Different Confidentiality Modes

Confidentiality Mode	Relevant Input / Output Data Registers
ECB	All

.....continued	
Confidentiality Mode	**Relevant Input / Output Data Registers**
CBC	All
OFB	All
128-bit CFB	All
64-bit CFB	First and Second
32-bit CFB	First
16-bit CFB	First
8-bit CFB	First
CTR	All

36.6.2.4 Start Modes

The Start mode field in the Control A Register (CTRLA.STARTMODE) allows the selection of encryption start mode.

1. Manual Start Mode
   In the Manual Start Mode the sequence is as follows:
   a. Write the 128/192/256 bit key in the Key Register (KEYWORD)
   b. Write the initialization vector or counter in the Initialization Vector Register (INTVECT). The initialization vector concerns all modes except ECB
   c. Enable interrupts in Interrupt Enable Set Register (INTENSET), depending on whether an interrupt is required or not at the end of processing.
   d. Write the data to be encrypted or decrypted in the Data Registers (INDATA).
   e. Set the START bit in Control B Register (CTRLB.START) to begin the encryption or the decryption process.
   f. When the processing completes, the Encryption Complete bit in the Interrupt Flag Register (INTFLAG.ENCCMP) raises. If Encryption Complete interrupt has been enabled, the interrupt line of the AES is activated.
   g. When the software reads one of the Output Data Registers (INDATA), INTFLAG.ENCCMP bit is automatically cleared.
2. Auto start Mode
   The Auto Start Mode is similar to the manual one, but as soon as the correct number of input data registers is written, processing is automatically started without setting the START bit in the Control B Register. DMA operation uses this mode.
3. Last Output Data Mode (LOD)
   This mode is used to generate message authentication code (MAC) on data in CCM mode of operation. The CCM mode combines counter mode for encryption and CBC-MAC generation for authentication.

When LOD is disabled in CCM mode then counter mode of encryption is performed on the input data block.

When LOD is enabled in CCM mode then CBC-MAC generation is performed. Zero block is used as the initialization vector by the hardware. Reading from the Output Data Register (INDATA) is not required to clear the ENCCMP flag. The ENCCMP flag is automatically cleared by writing into the Input Data Register (INDATA). This allows retrieval of only the last data in several encryption/decryption processes. No output data register reads are necessary between each block of encryption/decryption process.

Note that assembling message depending on the security level identifier in CCM* has to be done in software.

36.6.2.5 Computation of last Nk words of expanded key

The AES algorithm takes the cryptographic key provided by the user and performs a Key Expansion routine to generate an expanded key. The expanded key contains a total of 4(Nr + 1) 32-bit words, where the first Nk (4/6/8 for a 128-/192-/256-bit key) words are the user-provided key. For data encryption, the expanded key is used in the forward direction, i.e., the first four words are used in the initial round of data processing, the second four words in the first round, the third four words in the second round, and so on. On the other hand, for data decryption, the expanded key is used in the reverse direction, i.e.,the last four words are used in the initial round of data processing, the last second four words in the first round, the last third four words in the second round, and so on.

To reduce gate count, the AES module does not generate and store the entire expanded key prior to data processing. Instead, it computes on-the-fly the round key (four 32-bit words) required for the current round of data processing. In general, the round key for the current round of data processing can be computed from the Nk words of the expanded key generated in the previous rounds. When AES module is operating in the encryption mode, the round key for the initial round of data processing is simply the user-provided key written to the KEY registers. On the other hand, when AES module is operating in the decryption mode, the round key for the initial round of data processing is the last four words of the expanded key, which is not available unless AES module has performed at least one encryption process prior to operating in the decryption mode.

In general, the last Nk words of the expanded key must be available before decryption can start. If desired, AES module can be instructed to compute the last Nk words of the expanded key in advance by writing a one to the Key Generate (KEYGEN) bit in the CTRLA register (CTRLA.KEYGEN). The computation takes Nr clock cycles. Alternatively, the last Nk words of the expanded key can be automatically computed by AES module when a decryption process is initiated if they have not been computed in advance or have become invalid. Note that this will introduce a latency of Nr clock cycles to the first decryption process.

36.6.2.6 Hardware Countermeasures against Differential Power Analysis Attacks

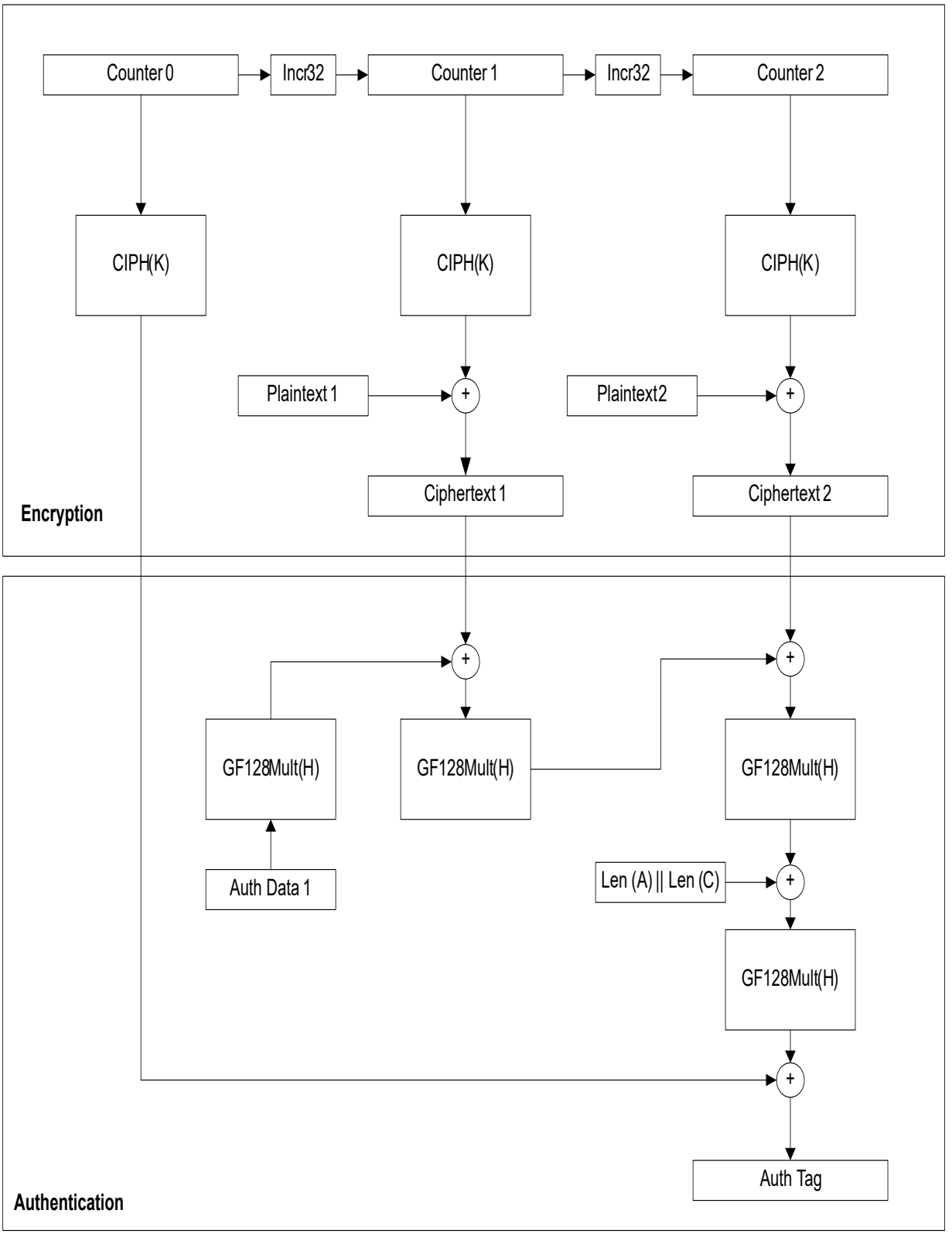
The AES module features four types of hardware countermeasures that are useful for protecting data against differential power analysis attacks:

- Type 1: Randomly add one cycle to data processing
- Type 2: Randomly add one cycle to data processing (other version)
- Type 3: Add a random number of clock cycles to data processing, subject to a maximum of 11/13/15 clock cycles for key sizes of 128/192/256 bits
- Type 4: Add random spurious power consumption during data processing

By default, all countermeasures are enabled, but require a write in DRNGSEED register to be effective. One or more of the countermeasures can be disabled by programming the Countermeasure Type field in the Control A (CTRLA.CTYPE) register. The countermeasures use random numbers generated by a deterministic random number generator embedded in AES module. The seed for the random number generator is written to the RANDSEED register. Note also that a new seed must be written after a change in the keysize. Note that enabling countermeasures reduces AES module's throughput. In short, the throughput is highest with all the countermeasures disabled. On the other hand, with all of the countermeasures enabled, the best protection is achieved but the throughput is worst.

36.6.3 Galois Counter Mode (GCM)

GCM is comprised of the AES engine in CTR mode along with a universal hash function (GHASH engine) that is defined over a binary Galois field to produce a message authentication tag. The GHASH engine processes data packets after the AES operation. GCM provides assurance of the confidentiality of data through the AES Counter mode of operation for encryption. Authenticity of the confidential data is assured through the GHASH engine. Refer to the NIST Special Publication 800-38D Recommendation for more information.



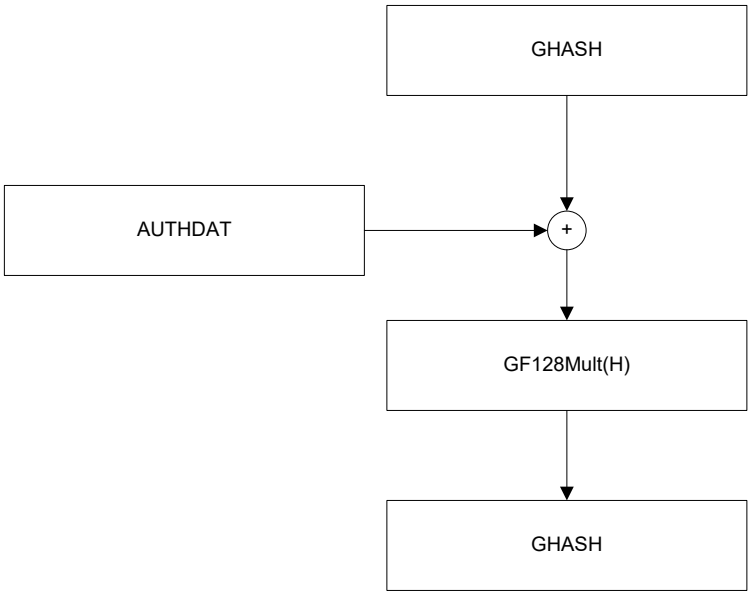
36.6.3.1 GCM Operation

36.6.3.1.1 Hashkey Generation

- Configure CTRLA register as follows:
  a. CTRLA.STARTMODE as Manual (Auto for DMAC)
  b. CTRLA.CIPHER as Encryption
  c. CTRLA.KEYSIZE as per the key used
  d. CTRLA.AESMODE as ECB
  e. CTRLA.CTYPE as per the countermeasures required.
- Set CTRLA.ENABLE
- Write zero to CIPLEN reg.
- Write the key in KEYWORD register
- Write the zeros to INDATA reg
- Set CTRLB.Start.
- Wait for INTFLAG.ENCCMP to be set
- AES Hardware generates Hash Subkey in HASHKEY register.

36.6.3.1.2 Authentication Header Processing

- Configure CTRLA register as follows:
  a. CTRLA.STARTMODE as Manual
  b. CTRLA.CIPHER as Encryption
  c. CTRLA.KEYSIZE as per the key used
  d. CTRLA.AESMODE as GCM
  e. CTRLA.CTYPE as per the countermeasures required.
- Set CTRLA.ENABLE
- Write the key in KEYWORD register
- Set CTRLB.GFMUL
- Write the Authdata to INDATA reg
- Set CTRLB.START as1
- Wait for INTFLAG.GFMCMP to be set.
- AES Hardware generates output in GHASH register
- Continue steps 4 to 7 for remaining Authentication Header.
  Note: If the Auth data is less than 128 bit, it has to be padded with zero to make it 128 bit aligned.



36.6.3.1.3 Plain text Processing

- Set CTRLB.NEWMSG for the new set of plain text processing.
- Load CIPLEN reg.
- Load (J0+1) in INTVECT register.
- As described in NIST documentation J 0 = IV || 0 31 || 1 when len(IV)=96 and J0 =$GHASH_H$ (IV || 0 s+64 || [len(IV)] 64) (s is the minimum number of zeroes that must be padded with the Initialization Vector to make it a multiple of 128) if len(IV) != 96.
- Load plain text in INDATA register.
- Set CTRLB.START as 1.
- Wait for INTFLAG.ENCCMP to be set.
- AES Hardware generates output in INDATA register.
- Intermediate GHASH is stored in GHASH register and Cipher Text available in INDATA register.
- Continue 3 to 6 till the input of plain text to get the cipher text and the Hash keys.
- At the last input, set CTRLB.EOM.
- Write last in-data to INDATA reg.
- Set CTRLB.START as 1.
- Wait for INTFLAG.ENCCMP to be set.
- AES Hardware generates output in INDATA register and final Hash key in GHASH register.
- Load [LEN(A)]64||[LEN(C)]64 in INDATA register and set CTRLB.GFMUL and CTRLB.START as 1.
- Wait for INTFLAG.GFMCMP to be set.
- AES Hardware generates final GHASH value in GHASH register.

36.6.3.1.4 Plain text processing with DMAC

- Set CTRLB.NEWMSG for the new set of plain text processing.
- Load CIPLEN reg.
- Load (J0+1) in INTVECT register.
- Load plain text in INDATA register.
- Wait for INTFLAG.ENCCMP to be set.

- AES Hardware generates output in INDATA register.
- Intermediate GHASH is stored in GHASH register and Cipher Text available in INDATA register.
- Continue 3 to 5 till the input of plain text to get the cipher text and the Hash keys.
- At the last input, set CTRLB.EOM.
- Write last in-data to INDATA reg.
- Wait for INTFLAG.ENCCMP to be set.
- AES Hardware generates output in INDATA register and final Hash key in GHASH register.
- Load [LEN(A)]64||[LEN(C)]64 in INDATA register and set CTRLB.GFMUL and CTRLB.START as 1.
- Wait for INTFLAG.GFMCMP to be set.
- AES Hardware generates final GHASH value in GHASH register.

36.6.3.1.5 Tag Generation

- Configure CTRLA
  a. Set CTRLA.ENABLE to 0
  b. Set CTRLA.AESMODE as CTR
  c. Set CTRLA.ENABLE to 1
- Load J0 value to INITVECTV reg.
- Load GHASH value to INDATA reg.
- Set CTRLB.NEWMSG and CTRLB.START to start the Counter mode operation.
- Wait for INTFLAG.ENCCMP to be set.
- AES Hardware generates the GCM Tag output in INDATA register.

36.6.4 Synchronization

Not applicable.

36.7 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0		CFBS[2:0]			AESMODE[2:0]		ENABLE	SWRST
		15:8		XORKEY	KEYGEN	LOD	STARTMODE	CIPHER	KEYSIZE[1:0]	
		23:16					CTYPE[3:0]			
		31:24								
0x04	CTRLB	7:0					GFMUL	EOM	NEWMSG	START
0x05	INTENCLR	7:0							GFMCMP	ENCCMP
0x06	INTENSET	7:0							GFMCMP	ENCCMP
0x07	INTFLAG	7:0							GFMCMP	ENCCMP
0x08	DATABUFPTR	7:0							INDATAPTR[1:0]	
0x09	DBGCTRL	7:0								DBGRUN
0x0A ... 0x0B	Reserved									
0C	KEYWORD0	7:0	KEYWORD[7:0]							
		15:8	KEYWORD[15:8]							
		23:16	KEYWORD[23:16]							
		31:24	KEYWORD[31:24]							
10	KEYWORD1	7:0	KEYWORD[7:0]							
		15:8	KEYWORD[15:8]							
		23:16	KEYWORD[23:16]							
		31:24	KEYWORD[31:24]							
14	KEYWORD2	7:0	KEYWORD[7:0]							
		15:8	KEYWORD[15:8]							
		23:16	KEYWORD[23:16]							
		31:24	KEYWORD[31:24]							
18	KEYWORD3	7:0	KEYWORD[7:0]							
		15:8	KEYWORD[15:8]							
		23:16	KEYWORD[23:16]							
		31:24	KEYWORD[31:24]							
1C	KEYWORD4	7:0	KEYWORD[7:0]							
		15:8	KEYWORD[15:8]							
		23:16	KEYWORD[23:16]							
		31:24	KEYWORD[31:24]							
20	KEYWORD5	7:0	KEYWORD[7:0]							
		15:8	KEYWORD[15:8]							
		23:16	KEYWORD[23:16]							
		31:24	KEYWORD[31:24]							
24	KEYWORD6	7:0	KEYWORD[7:0]							
		15:8	KEYWORD[15:8]							
		23:16	KEYWORD[23:16]							
		31:24	KEYWORD[31:24]							
28	KEYWORD7	7:0	KEYWORD[7:0]							
		15:8	KEYWORD[15:8]							
		23:16	KEYWORD[23:16]							
		31:24	KEYWORD[31:24]							
0x2C ... 0x37	Reserved									
0x38	INDATA	7:0	INDATA[7:0]							
		15:8	INDATA[15:8]							
		23:16	INDATA[23:16]							
		31:24	INDATA[31:24]							
3C	INTVECTV0	7:0	INTVECTV[7:0]							
		15:8	INTVECTV[15:8]							
		23:16	INTVECTV[23:16]							
		31:24	INTVECTV[31:24]							

.....continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
40	INTVECTV1	7:0	INTVECTV[7:0]							
		15:8	INTVECTV[15:8]							
		23:16	INTVECTV[23:16]							
		31:24	INTVECTV[31:24]							
44	INTVECTV2	7:0	INTVECTV[7:0]							
		15:8	INTVECTV[15:8]							
		23:16	INTVECTV[23:16]							
		31:24	INTVECTV[31:24]							
48	INTVECTV3	7:0	INTVECTV[7:0]							
		15:8	INTVECTV[15:8]							
		23:16	INTVECTV[23:16]							
		31:24	INTVECTV[31:24]							
0x4C ... 0x5B	Reserved									
0x5C	HASHKEY0	7:0	HASHKEY[7:0]							
		15:8	HASHKEY[15:8]							
		23:16	HASHKEY[23:16]							
		31:24	HASHKEY[31:24]							
0x60	HASHKEY1	7:0	HASHKEY[7:0]							
		15:8	HASHKEY[15:8]							
		23:16	HASHKEY[23:16]							
		31:24	HASHKEY[31:24]							
0x64	HASHKEY2	7:0	HASHKEY[7:0]							
		15:8	HASHKEY[15:8]							
		23:16	HASHKEY[23:16]							
		31:24	HASHKEY[31:24]							
0x68	HASHKEY3	7:0	HASHKEY[7:0]							
		15:8	HASHKEY[15:8]							
		23:16	HASHKEY[23:16]							
		31:24	HASHKEY[31:24]							
0x6C	GHASH0	7:0	GHASH[7:0]							
		15:8	GHASH[15:8]							
		23:16	GHASH[23:16]							
		31:24	GHASH[31:24]							
0x70	GHASH1	7:0	GHASH[7:0]							
		15:8	GHASH[15:8]							
		23:16	GHASH[23:16]							
		31:24	GHASH[31:24]							
0x74	GHASH2	7:0	GHASH[7:0]							
		15:8	GHASH[15:8]							
		23:16	GHASH[23:16]							
		31:24	GHASH[31:24]							
0x78	GHASH3	7:0	GHASH[7:0]							
		15:8	GHASH[15:8]							
		23:16	GHASH[23:16]							
		31:24	GHASH[31:24]							
0x7C ... 0x7F	Reserved									
0x80	CIPLEN	7:0	CIPLEN[7:0]							
		15:8	CIPLEN[15:8]							
		23:16	CIPLEN[23:16]							
		31:24	CIPLEN[31:24]							
0x84	RANDSEED	7:0	RANDSEED[7:0]							
		15:8	RANDSEED[15:8]							
		23:16	RANDSEED[23:16]							
		31:24	RANDSEED[31:24]							

36.8 Register Description

Registers can be 8, 16 or 32 bits wide. Atomic 8-, 16- and 32-bit accesses are supported. In addition, the 8-bit quarters and 16-bit halves of a 32-bit register and the 8-bit halves of a 16-bit register can be accessed directly.

Some registers are optionally write-protected by the Peripheral Access Controller (PAC). Optional PAC write protection is denoted by the “PAC Write-Protection” property in each individual register description. See *Register Access Protection* from Related Links.

Some registers are enable protected, meaning they can only be written when the peripheral is disabled. Enable protection is denoted by the “Enable-Protected” property in each individual register description.

Related Links

36.5.8. Register Access Protection

36.8.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00000000
Property: PAC Write-Protection, Enable-protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
					CTYPE[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit	15	14	13	12	11	10	9	8
		XORKEY	KEYGEN	LOD	STARTMODE	CIPHER	KEYSIZE[1:0]	
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	CFBS[2:0]			AESMODE[2:0]			ENABLE	SWRST
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 19:16 – CTYPE[3:0] Counter Measure Type

Value	Name	Description
XXX0	CTYPE1 disabled	Countermeasure1 disabled
XXX1	CTYPE1 enabled	Countermeasure1 enabled
XX0X	CTYPE2 disabled	Countermeasure2 disabled
XX1X	CTYPE2 enabled	Countermeasure2 enabled
X0XX	CTYPE3 disabled	Countermeasure3 disabled
X1XX	CTYPE3 enabled	Countermeasure3 enabled
0XXX	CTYPE4 disabled	Countermeasure4 disabled
1XXX	CTYPE4 enabled	Countermeasure4 enabled

Bit 14 – XORKEY XOR Key Operation

Value	Description
0	No effect
1	The user keyword gets XORed with the previous keyword register content.

Bit 13 – KEYGEN Last Key Generation

Value	Description
0	No effect
1	Start Computation of the last NK words of the expanded key

Bit 12 – LOD Last Output Data Mode

Value	Description
0	No effect
1	Start encryption in Last Output Data mode

Bit 11 – STARTMODE Start Mode Select

Value	Name	Description
0	Manual Mode	Start Encryption / Decryption in Manual mode
1	Auto Mode	Start Encryption / Decryption in Auto mode

Bit 10 – CIPHER Cipher Mode Select

Value	Description
0	Decryption
1	Encryption

Bits 9:8 – KEYSIZE[1:0] Encryption Key Size

Value	Name	Description
0	128-bit Key	128-bit Key for Encryption / Decryption
1	192-bit Key	192-bit Key for Encryption / Decryption
2	256-bit Key	256-bit Key for Encryption / Decryption
3	Reserved	Reserved

Bits 7:5 – CFBS[2:0] Cipher Feedback Block Size

Value	Name	Description
0	128-bit data block	128-bit Input data block for Encryption/Decryption in Cipher Feedback mode
1	64-bit data block	64-bit Input data block for Encryption/Decryption in Cipher Feedback mode
2	32-bit data block	32-bit Input data block for Encryption/Decryption in Cipher Feedback mode
3	16-bit data block	16-bit Input data block for Encryption/Decryption in Cipher Feedback mode
4	8-bit data block	8-bit Input data block for Encryption/Decryption in Cipher Feedback mode
5-7	Reserved	Reserved

Bits 4:2 – AESMODE[2:0] AES Modes of Operation

Value	Name	Description
0	ECB	Electronic code book mode
1	CBC	Cipher block chaining mode
2	OFB	Output feedback mode
3	CFB	Cipher feedback mode
4	Counter	Counter mode
5	CCM	CCM mode
6	GCM	Galois counter mode
7	Reserved	Reserved

Bit 1 – ENABLE Enable

Value	Description
0	The peripheral is disabled
1	The peripheral is enabled

Bit 0 – SWRST Software Reset

Writing a '0' to this bit has no effect.
Writing a '1' to this bit resets all registers in the AES module to their initial state, and the module will be disabled.
Writing a '1' to `SWRST` will always take precedence, meaning that all other writes in the same write operation will be discarded.

Value	Description
0	There is no reset operation ongoing
1	The reset operation is ongoing

36.8.2 Control B

Name: CTRLB
Offset: 0x04
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
					GFMUL	EOM	NEWMSG	START
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit 3 – GFMUL GF Multiplication
This bit is applicable only to GCM mode.

Value	Description
0	No action
1	Setting this bit calculates GF multiplication with data buffer content and hashkey register content.

Bit 2 – EOM End of Message
This bit is applicable only to GCM mode.

Value	Description
0	No action
1	Setting this bit generates final GHASH value for the message.

Bit 1 – NEWMSG New Message
This bit is used in cipher block chaining (CBC), cipher feedback (CFB) and output feedback (OFB), counter (CTR) modes to indicate the hardware to use Initialization vector for encrypting the first block of message.

Value	Description
0	No action
1	Setting this bit indicates start of new message to the module.

Bit 0 – START Start Encryption/Decryption

Value	Description
0	No action
1	Start encryption / decryption in manual mode.

36.8.3 Interrupt Enable Clear

Name: INTENCLR
Offset: 0x05
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to disable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Set (`INTENSET`) register.

Bit	7	6	5	4	3	2	1	0
							GFMCMP	ENCCMP
Access							R/W	R/W
Reset							0	0

Bit 1 – GFMCMP GF Multiplication Complete Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the GF Multiplication Complete Interrupt Enable bit, which disables the GF Multiplication Complete interrupt.

Value	Description
0	The GF Multiplication Complete interrupt is disabled.
1	The GF Multiplication Complete interrupt is enabled.

Bit 0 – ENCCMP Encryption Complete Interrupt Enable
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the Encryption Complete Interrupt Enable bit, which disables the Encryption Complete interrupt.

Value	Description
0	The Encryption Complete interrupt is disabled.
1	The Encryption Complete interrupt is enabled.

36.8.4 Interrupt Enable Set

Name: INTENSET
Offset: 0x06
Reset: 0x00
Property: PAC Write-Protection

This register allows the user to enable an interrupt without doing a read-modify-write operation. Changes in this register will also be reflected in the Interrupt Enable Clear (`INTENCLR`) register.

Bit	7	6	5	4	3	2	1	0
							GFMCMP	ENCCMP
Access							R/W	R/W
Reset							0	0

Bit 1 – GFMCMP GF Multiplication Complete Interrupt Enable
Writing a '0' to this bit has no effect. Writing a '1' to this bit will clear the GF Multiplication Complete Interrupt Enable bit, which enables the GF Multiplication Complete interrupt.

Value	Description
0	The GF Multiplication Complete interrupt is disabled.
1	The GF Multiplication Complete interrupt is enabled.

Bit 0 – ENCCMP Encryption Complete Interrupt Enable
Writing a '0' to this bit has no effect. Writing a '1' to this bit will clear the Encryption Complete Interrupt Enable bit, which enables the Encryption Complete interrupt.

Value	Description
0	The Encryption Complete interrupt is disabled.
1	The Encryption Complete interrupt is enabled.

36.8.5 Interrupt Flag Status and Clear

Name: INTFLAG
Offset: 0x07
Reset: 0x00

Bit	7	6	5	4	3	2	1	0
							GFMCMP	ENCCMP
Access							R/W	R/W
Reset							0	0

Bit 1 – GFMCMP GF Multiplication Complete
This flag is cleared by writing a '1' to it.
This flag is set when GHASH value is available on the Galois Hash Registers (GHASHx) in GCM mode.
Writing a '0' to this bit has no effect.
This flag is also automatically cleared in the following cases.

1. Manual encryption/decryption occurs (START in CTRLB register).
2. Reading from the GHASHx register.

Bit 0 – ENCCMP Encryption Complete
This flag is cleared by writing a '1' to it.
This flag is set when encryption/decryption is complete and valid data is available on the Data Register.
Writing a '0' to this bit has no effect.
This flag is also automatically cleared in the following cases:

1. Manual encryption/decryption occurs (START in CTRLA register). (This feature is needed only if we do not support double buffering of INDATA registers).
2. Reading from the data register (INDATAx) when LOD = 0.
3. Writing into the data register (INDATAx) when LOD = 1.
4. Reading from the Hash Key register (HASHKEYx).

36.8.6 Data Buffer Pointer

Name: DATABUFPTR
Offset: 0x08
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
							INDATAPTR[1:0]	
Access							R/W	R/W
Reset							0	0

Bits 1:0 – INDATAPTR[1:0] Input Data Pointer
Writing to this field changes the value of the input data pointer, which determines which of the four data registers is written to/read from when the next write/read to the `INDATA` register address is performed.

36.8.7 Debug

Name: DBGCTRL
Offset: 0x09
Reset: 0x00
Property: PAC Write-Protection

Bit	7	6	5	4	3	2	1	0
								DBGRUN
Access								W
Reset								0

Bit 0 – DBGRUN Debug Run
Writing a '0' to this bit causes the AES to halt during debug mode.
Writing a '1' to this bit allows the AES to continue normal operation during debug mode. This bit can only be changed while the AES is disabled.

36.8.8 Keyword

Name: KEYWORD
Offset: 0x0C + n*0x04 [n=0..7]
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
	KEYWORD[31:24]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	KEYWORD[23:16]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	KEYWORD[15:8]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	KEYWORD[7:0]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – KEYWORD[31:0] Key Word Value

The four/six/eight 32-bit Key Word registers set the 128-bit/192-bit/256-bit cryptographic key used for encryption/decryption. `KEYWORD0.KEYWORD` corresponds to the first word of the key and `KEYWORD3/KEYWORD5/KEYWORD7.KEYWORD` to the last one.

Note: By setting the XORKEY bit of CTRLA register, keyword will update with the resulting XOR value of user keyword and previous keyword content.

36.8.9 Data

Name: INDATA
Offset: 0x38
Reset: 0x00000000

Bit	31	30	29	28	27	26	25	24
	INDATA[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	INDATA[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	INDATA[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	INDATA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – INDATA[31:0] Data Value

A write to or read from this register corresponds to a write to or read from one of the four data registers. The four 32-bit Data registers set the 128-bit data block used for encryption/decryption. The data register that is written to or read from is given by the `DATABUFPTR.INDATPTR` field.

Note: Both input and output shares the same data buffer. Reading INDATA register will return 0's when AES is performing encryption or decryption operation.

36.8.10 Initialization Vector Register

Name: INTVECTV
Offset: 0x3C + n*0x04 [n=0..3]
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
	INTVECTV[31:24]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	INTVECTV[23:16]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	INTVECTV[15:8]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	INTVECTV[7:0]							
Access	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – INTVECTV[31:0] Initialization Vector Value

The four 32-bit Initialization Vector registers `INTVECTVn` set the 128-bit Initialization Vector data block that is used by some modes of operation as an additional initial input. `INTVECTV0.INTVECTV` corresponds to the first word of the Initialization Vector, `INTVECTV3.INTVECTV` to the last one. These registers are write-only to prevent the Initialization Vector from being read by another application. For CBC, OFB, and CFB modes, the Initialization Vector corresponds to the initialization vector. For CTR mode, it corresponds to the counter value.

36.8.11 Hash Key (GCM mode only)

Name: HASHKEY
Offset: 0x5C + n*0x04 [n=0..3]
Reset: 0x00000000
Property: PAC Write-protection

Bit	31	30	29	28	27	26	25	24
	HASHKEY[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	HASHKEY[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	HASHKEY[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	HASHKEY[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – HASHKEY[31:0] Hash Key Value
The four 32-bit `HASHKEY` registers contain the 128-bit Hash Key value computed from the AES KEY. The Hash Key value can also be programmed offering single GF128 multiplication possibilities.

36.8.12 Galois Hash (GCM mode only)

Name: GHASH
Offset: 0x6C + n*0x04 [n=0..3]
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
	GHASH[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	GHASH[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	GHASH[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	GHASH[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – GHASH[31:0] Galois Hash Value

The four 32-bit Hash Word registers GHASHcontain the GHASH value after GF128 multiplication in GCM mode. Writing a new key to KEYWORD registers causes GHASH to be initialized with zeroes. These registers can also be programmed.

36.8.13 Galois Hash x (GCM mode only)

Name: CIPLEN
Offset: 0x80
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
	CIPLEN[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	CIPLEN[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	CIPLEN[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	CIPLEN[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – CIPLEN[31:0] Cipher Length
This register contains the length in bytes of the Cipher text that is to be processed. This is programmed by the user in GCM mode for Tag generation.

36.8.14 Random Seed

Name: RANDSEED
Offset: 0x84
Reset: 0x00000000
Property: PAC Write-Protection

Bit	31	30	29	28	27	26	25	24
	RANDSEED[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	RANDSEED[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	RANDSEED[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	RANDSEED[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – RANDSEED[31:0] Random Seed
A write to this register corresponds to loading a new seed into the Random number generator.

37. Public Key Cryptography Controller (PUKCC)

37.1 Overview

The Public Key Cryptography Controller (PUKCC) processes public key cryptography algorithm calculus in both GF(p) and GF(2n) fields.

The Public Key Cryptography Library (PUKCL) is stored in ROM inside the device. The library can be used in applications to access features of PUKCC, and includes the complete implementation of the following public key cryptography algorithms:

- RSA (Rivest-Shamir-Adleman public key cryptosystem), DSA (Digital Signature Algorithm):
  - Modular Exponentiation with CRT up to 7168 bits
  - Modular Exponentiation without CRT up to 5376 bits
  - Prime generation
  - Utilities: GCD/modular Inverse, Divide, Modular reduction, Multiply, ...
- Elliptic Curves:
  - ECDSA GF(p) up to 521 bits for common curves (up to 1120 bits for future use)
  - ECDSA GF(2n) up to 571 bits for common curves (up to 1440 bits for future use)
  - Choice of the curve parameters for compatibility with NIST Curves or other curves in Weierstrass equation
  - Point Multiply
  - Point Add/Doubling
  - Other high level elliptic curve algorithms (ECDH, ...) can be implemented by user using library functions
- Deterministic Random Number Generation (DRNG ANSI X9.31) for DSA

37.2 Product Dependencies

37.2.1 I/O Lines

Not applicable.

37.2.2 Power Management

The PUKCC will continue to operate in any sleep mode, as long as its source clock is running.

37.2.3 Clocks

The bus clock (PB2_CLK) can be enabled and disabled by the CRU.

37.2.4 DMA

Not applicable.

37.2.5 Interrupts

Not applicable.

37.2.6 Events

Not applicable.

37.3 Functional Description

37.3.1 Public Key Cryptography Library (PUKCL) Application Programming Interface (API)

The Public Key Cryptography Controller (PUKCC) is a peripheral that can be used to accelerate public key cryptography, and processes public key cryptography algorithm calculus in both Prime field (GF(p)) and Binary field (GF(2^n)). Different functionalities of the PUKCC are accessed with the help of the Public Key Cryptography Library (PUKCL), which is embedded into a dedicated ROM inside the microcontroller.

The PUKCL provides access to many algorithms and functions. The features provided, start from basic addition or comparison, up to the RSA or ECDSA complete computation. The library can be utilized by including the PUKCL Driver in the application and passing parameters through a common Application Programming Interface (API). The PUKCC Driver is available in Harmony 3. This library can be used in conjunction with a SSL software stack to improve performance and helps to reduce the RAM usage and time taken to perform different cryptographic functions.

37.3.2 PUKCL Features

PUKCL features include:

- 37.3.4. Basic Arithmetic and Cryptographic Services - PUKCL self-test, GCD, integral division, etc.
- 37.3.5. Modular Arithmetic Services - Modular reduction, modular exponentiation, probable prime generation and modular exponentiation
- 37.3.6. Elliptic Curves Over GF(p) Services - Point addition and doubling on an elliptic curve in a prime field, ECDSA signature generation and verification on an elliptic curve over GF(p)
- 37.3.7. Elliptic Curves Over GF(2n) Services - Point addition and doubling on an elliptic curve in a prime field, ECDSA signature generation and verification on an elliptic curve over GF(2^n)

37.3.3 PUKCL Usage

The following sections provide details on accessing the PUKCL and its features.

37.3.3.1 Initializing the PUKCC and PUKCL

For a project created with Harmony 3, the clock initialization is handled by the initialization function CLK_Initialize(). After a power-on reset, and when the PUKCC Clock is enabled, a Crypto RAM clear process is launched. It is mandatory to wait until the end of this process before using the Crypto Library.

The following code shows how to wait for the Crypto RAM clear process.

```
while ((PUKCCSR & BIT_PUKCCSR_CLRRAM_BUSY) != 0);
```

The next task to be done is self-test. From the generated project in Harmony 3, copy the example for the PUKCC Driver SelfTest and add it to the main source file. This is a mandatory step before using the library. The return values from the SelfTest service must be compared against known values mentioned in the service description (see the **Description** section in 37.3.4.1. SelfTest).

Example 37-1. PUKCC Initialization

```
void PUKCC_self_test(void)
{
    // Clear contents of PUKCLParam
    memset(&PUKCLParam, 0, sizeof(PUKCL_PARAM));

    pvPUKCLParam = &PUKCLParam;
    vPUKCL_Process(SelfTest, pvPUKCLParam);

    // In case of error, loop here
    while (PUKCL(u2Status) != PUKCL_OK) {
    ;
```

```
    }
    while (pvPUKCLParam->P.PUKCL_SelfTest.u4Version != PUKCL_VERSION) {
    ;
    }
    while (pvPUKCLParam->P.PUKCL_SelfTest.u4CheckNum1 != 0x6E70DDD2) {
    ;
    }
    while (pvPUKCLParam->P.PUKCL_SelfTest.u4CheckNum2 != 0x25C8D64F) {
    ;
    }
}

int main(void)
{
    /* Initializes MCU, drivers and middleware */
    SYS_Initialize();

    // Wait for Crypto RAM clear process
    while ((PUKCCSR & BIT_PUKCCSR_CLRRAM_BUSY) != 0);

    // Initialize PUKCC and perform self test
    PUKCC_self_test();
    while(1)
    {
    }
}
```

Note: It may also be necessary to initialize the Random Number Generator (RNG) on the microcontroller, as some services in the library use the peripheral. Before calling such services, be sure to follow the directives given for random number generation on the selected microcontroller (particularly initialization and seeding) and compulsorily start the RNG. For details refer to each service.

37.3.3.2 Accessing Different Library Services

All cryptographic services in the library are accessed by the macro vPUKCL_Process. All of these services use the same process for receiving and returning parameters. PUKCL receives two arguments: the requested service and a pointer to a structure called the parameter block. The parameter block contains two structures, a common parameter structure for all commands and specific parameter structure for each service. A specific service is accessed with vPUKCL_Process by passing the service name as the first argument. For example, to perform SelfTest, use vPUKCL_Process(SelfTest, pvPUKCLParam).

Example 37-2. PUKCL Parameter Block

```
typedef struct _PUKCL_param {
    PUKCL_HEADER PUKCL_Header;
    union {
    _PUKCL_CLEARFLAGS PUKCL_ClearFlags;
    _PUKCL_COMP       PUKCL_Comp;
    _PUKCL_CONDCOPY   PUKCL_CondCopy;
    _PUKCL_CRT        PUKCL_CRT;
    _PUKCL_DIV        PUKCL_Div;
    _PUKCL_EXPMOD     PUKCL_ExpMod;
    _PUKCL_FASTCOPY   PUKCL_FastCopy;
    _PUKCL_FILL       PUKCL_Fill;
    _PUKCL_FMULT      PUKCL_Fmult;
    _PUKCL_GCD        PUKCL_GCD;
    _PUKCL_PRIMEGEN   PUKCL_PrimeGen;
    _PUKCL_REDMOD     PUKCL_RedMod;
    _PUKCL_RNG        PUKCL_Rng;
    _PUKCL_SELFTEST   PUKCL_SelfTest;
    _PUKCL_SMULT      PUKCL_Smult;
    _PUKCL_SQUARE     PUKCL_Square;
    _PUKCL_SWAP       PUKCL_Swap;

    // ECC
    _PUKCL_ZPECCADD                 PUKCL_ZpEccAdd;
    _PUKCL_ZPECCDBL                 PUKCL_ZpEccDbl;
    _PUKCL_ZPECCADDSUB              PUKCL_ZpEccAddSub;
    _PUKCL_ZPECCMUL                 PUKCL_ZpEccMul;
    _PUKCL_ZPECDSAGENERATE          PUKCL_ZpEcDsaGenerate;
```

```
    _PUKCL_ZPECDSAVERIFY                PUKCL_ZpEcDsaVerify;
    _PUKCL_ZPECDSAQUICKVERIFY           PUKCL_ZpEcDsaQuickVerify;
    _PUKCL_ZPECCQUICKDUALMUL            PUKCL_ZpEccQuickDualMul;
    _PUKCL_ZPECCONVPROJTOAFFINE         PUKCL_ZpEcConvProjToAffine;
    _PUKCL_ZPECCONVAFFINETOPROJECTIVE PUKCL_ZpEcConvAffineToProjective;
    _PUKCL_ZPECRANDOMIZECOORDINATE      PUKCL_ZpEcRandomiseCoordinate;
    _PUKCL_ZPECPOINTISONCURVE           PUKCL_ZpEcPointIsOnCurve;

    // ECC
    _PUKCL_GF2NECCADD                     PUKCL_GF2NEccAdd;
    _PUKCL_GF2NECCDBL                     PUKCL_GF2NEccDbl;
    _PUKCL_GF2NECCMUL                     PUKCL_GF2NEccMul;
    _PUKCL_GF2NECDSAGENERATE              PUKCL_GF2NEcDsaGenerate;
    _PUKCL_GF2NECDSAVERIFY                PUKCL_GF2NEcDsaVerify;
    _PUKCL_GF2NECCONVPROJTOAFFINE         PUKCL_GF2NEcConvProjToAffine;
    _PUKCL_GF2NECCONVAFFINETOPROJECTIVE PUKCL_GF2NEcConvAffineToProjective;
    _PUKCL_GF2NECRANDOMIZECOORDINATE      PUKCL_GF2NEcRandomiseCoordinate;
    _PUKCL_GF2NECPOINTISONCURVE           PUKCL_GF2NEcPointIsOnCurve;
    } P;
} PUKCL_PARAM,
```

37.3.3.2.1 PUKCL_HEADER Structure

The PUKCL_HEADER is common for all services of the library. This header includes standard fields to indicate the requested service, sub-service, options, return status, and so on, as shown in the following tables.

Different terms used in the below description to be understood, are as follows:

- Parameter – Represents a variable used by the PUKCL. Every parameter belongs to either PUKCL_HEADER or PUKCL Service Specific Header
- Type – Indicates the data type. For details on data type, please refer to `CryptoLib_typedef_pb.h` file in the library
- Dir – Direction. Indicates whether PUKCL considers the variable as input or output. Input means that the application passes data to the PUKCL using the variable. Output means that the PUKCL uses the variable to pass data to the application.
- Location – Suggests whether the parameter need to be stored in Crypto RAM or device SRAM. The PUKCL driver has macros for placing parameters into Crypto RAM, so that the user does not have to worry about the addresses
- Data Length – If a parameter is a pointer variable, the Data Length column shows the size of the data pointed by the pointer

Table 37-1. PUKCL_HEADER Structure

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u1Service	u1	I	–	–	Required service	Executed service
u1SubService	u1	I	–	–	Required sub-service	Executed sub-service
u2Option	u2	I	–	–	Required option	Executed option
Specific	PUKCL_STATUS	I/O	–	–	See the following table PUKCL_STATUS Structure	See the following table PUKCL_STATUS Structure
u2Status	u2	I/O	–	–	–	Output Status
Reserved	u2	–	–	–	–	–
Reserved	u4	–	–	–	–	–

The Specific field in the PUKCL_HEADER structure is another structure named PUKCL_STATUS. The following table describes this structure. The details of the use of these bits are provided in the individual service descriptions.

37.3.3.2.2 PUKCL_STATUS Structure

Members of the PUKCL_STATUS structure are shown in the following table.

Table 37-2. PUKCL_STATUS Structure

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
CarryIn (see **Note 1**)	bit	I	–	–	CarryIn	–
CarryOut	bit	O	–	–	–	CarryOut
Zero	bit	O	–	–	–	1: Result is zero 0: Result is not zero
Gf2n (see **Note 1**)	bit	I	–	–	Mathematical field 0: Integers (Z_p) 1: Field GF(2^n)	–
Violation	bit	O	–	–	–	Indicates a violation

Note:

1. Two of these fields must be filled in to avoid problems during computations. If the Gf2n and CarryIn fields are not reset or initialized properly, problems may be encountered during computations. For instance, not initializing the Gf2n field may result in getting a correct mathematical result, but computed over GF(2^n) instead of Z_p.

37.3.3.2.3 PUKCL Service Specific Header

Details about each service specific header are provided with service descriptions in a subsequent section. Such structures may contain input or output parameters. A parameter is considered as an input parameter when it used for passing information to the PUKCL, and it is considered as an output parameter when the PUKCL uses it to pass a result back to the application code.

The following code provides the service specific header example for the SelfTest service.

```
typedef struct _PUKCL_selftest {
    u4 u4Version;
    u4 u4PUKCCVersion;
    u4 u4CheckNum1;
    u4 u4CheckNum2;
    u1 u1Step;
} _PUKCL_SELFTEST;
```

After the SelfTest service is invoked (with vPUKCL_Process(SelfTest, pvPUKCLParam)), the service specific return values can be checked using pvPUKCLParam.

To check whether the version returned by the PUKCL is correct, the following code can be used.

```
while (pvPUKCLParam->P.PUKCL_SelfTest.u4Version != PUKCL_VERSION);
```

In a similar way, other returns can also be accessed.

37.3.3.3 Parameter Passing (Special Considerations)

Most of the PUKCL services work with memory area and accept pointers and lengths as parameters to define input and output areas. Most of the time, the pointers and lengths are untouched by the services, while the defined areas are read, filled, or overwritten. These memory areas are defined with an initial pointer and a byte length. For most of the commands, the memory area location must be in the PUKCC Cryptographic RAM. The Cryptographic RAM is the memory area for parameter exchange with the PUKCL and is 4 Kbytes large. Sometimes memory areas can be located in Embedded SRAM, which is detailed in the Location column of the parameters description tables.

When working with binary fields, polynomials in GF(2^n) need no transformation to be written in an area:

- Each bit represents a polynomial coefficient 0 or 1
- The polynomials must be written Low Significant Byte First
- A zero padding on the Most Significant Bytes may be added if the area is larger than the real size of the polynomial

Important: The Cryptographic RAM is 4 Kbytes in size and is dedicated to PUKCC. However, to ensure correct library operation, the two last 32-bit words must not be used. Unless otherwise specified, these memory areas contain integers in GF(p) or polynomials in GF(2^n) with the Less Significant Byte first.

Unless otherwise specified, the length must be a multiple of four and the pointers must be four bytes aligned. This is because most of the services work with 32-bit words.

37.3.3.4 Aligned Significant Length

Parameters in memory areas can have any Significant Length in bytes. As the lengths in PUKCL must be a multiple of four, a padding is processed on the Most Significant Side with zero to three bytes cleared to zero. Now the parameter can be considered to meet the Aligned Significant Length requirement for PUKCL.

37.3.3.5 Processing Field GF(p) and GF(2^n)

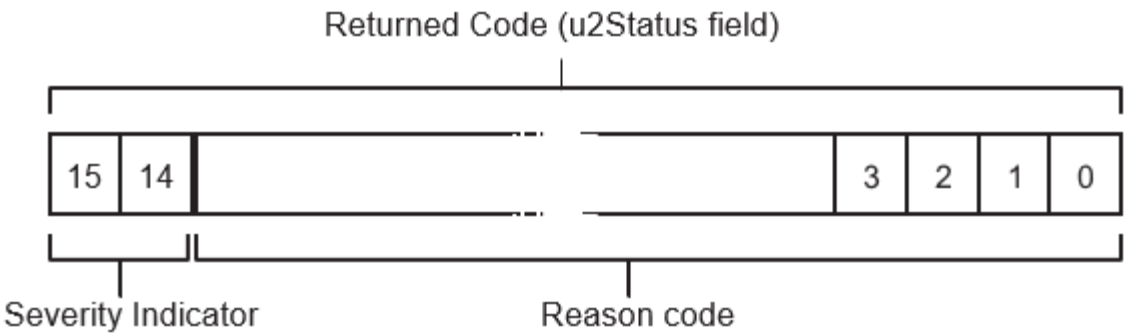
The library can process arithmetic functions over GF(p) (or Zp integers) and GF(2^n), when applicable. The choice of these processing fields is made using the following rules:

- If a processing field is not applicable to the function, it is not mentioned and the Specific.GF2n bit has no effect.
- If the function can support both processing fields, the choice is mentioned and the Specific.GF2n bit must be filled according to the choice.
- If the function supports only one of the processing fields, the processing field is mentioned and the Specific.GF2n bit has no effect.

37.3.3.6 Return Codes

Each call to one of the PUKCL services returns a status code indicating whether or not the execution is correct, which can be decoded, as shown in the following figure.

Figure 37-1. Return Code Status Decoding



The following table shows how the severity indicators must be decoded.

Table 37-3. Severity Indicators

Value for Bits 14–15	Severity	Comment
0xC000	Severe	Indicates a blocking error condition
0x8000	Warning	Indicates a cautionary use of the return values
0x4000	Information	Indicates the result is correct and gives information
0x0000	–	No error or no severity given

The following table contains the exhaustive list of all reason codes.

Table 37-4. Return Codes

Value for Bits 00–13	Severity Code	Reason Code
0x0000	—	PUKCL_OK
0x4001	Informative	PUKCL_NUMBER_IS_NOT_PRIME
0x4002	Informative	PUKCL_NUMBER_IS_PRIME
0xC001	Severe	PUKCL_COMPUTATION_NOT_STARTED
0xC002	Severe	PUKCL_UNKNOWN_SERVICE
0xC003	Severe	PUKCL_UNEXPLOITABLE_OPTIONS
0xC004	Severe	PUKCL_HARDWARE_ISSUE
0xC005	Severe	PUKCL_WRONG_HARDWARE
0xC006	Severe	PUKCL_LIBRARY_MALFORMED
0xC007	Severe	PUKCL_ERROR
0xC008	Severe	PUKCL_UNKNOWN_SUBSERVICE
0xC101	Severe	PUKCL_DIVISION_BY_ZERO
0xC102	Severe	PUKCL_MALFORMED_MODULUS
0xC103	Severe	PUKCL_FAULT_DETECTED
0xC104	Severe	PUKCL_MALFORMED_KEY

Please note the following rules about return codes:

- A status value indicating a severe error, means that an expected operation has not been executed or has been corrupted. Therefore, the result of such an operation must not be used.
- A status value indicating a warning must be looked at precisely, as the expected correctness of the result cannot be guaranteed.
- A status value indicating an information always means that the result is correct with no possible misinterpretation of the values.
- A status value zero indicates that there is no error or no severity.

In the following sections, for each service, the constraints on the parameters placement are detailed. For reduced code size and higher execution speed, tests are processed on these constraints. It is important that PUKCL users take these placement constraints into consideration at the development and test stages to ensure the correct functioning of the library.

37.3.4 Basic Arithmetic and Cryptographic Services

37.3.4.1 SelfTest

37.3.4.1.1 Purpose

This service is used to initialize the PUKCL. It resets the PUKCC, clears the Crypto RAM, and returns the library and PUKCC version numbers.

It must be called before using any other services in the library and the user must verify the return status at the end of the service execution.

37.3.4.1.2 How to Use the Service

37.3.4.1.3 Description

This service processes internal tests and returns information and status codes as described in 37.3.4.1.7. Status Returned Values. The service name for this operation is `SelfTest`.

37.3.4.1.4 Parameters Definition

It is possible to directly address this service through the `PUKCL_SelfTest()` macro.

Table 37-5. SelfTest Service Parameters

Parameter	Type	Dir.	Location	Data Length	Before Executing the Service	After Executing the Service
u4Version	u4	O	–	–	–	PUKCL version
u4PUKCCVersion	u4	O	–	–	–	PUKCC Version
u4CheckNum1	u4	O	–	–	–	Test result value 1
u4CheckNum2	u4	O	–	–	–	Test result value 2
u1Step	u1	O	–	–	–	Latest correctly executed step

37.3.4.1.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library
vPUKCL_Process(SelfTest,pvPUKCLParam);

if (PUKCL(u2Status) == PUKCL_OK)
            {
            // The Library version is available
            // in PUKCL_SelfTest(u4Version)
            // The PUKCL version is available
            // in PUKCL_SelfTest(u4PUKCCVersion)
            }
```

37.3.4.1.6 Returned Values

The expected u4Version value depends on the version of PUKCL being used, and the u4PUKCCVersion value depends on the version of PUKCC being used.

The expected u4CheckNum1 value is 0x6e70ddd2 and the expected one for u4CheckNum2 is 0x25c8d64f. The expected final u1Step value is 3.

37.3.4.1.7 Status Returned Values

Table 37-6. SelfTest Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly.
PUKCL_ERROR	Severe	An issue has been encountered.

37.3.4.2 Clear Flags

37.3.4.2.1 Purpose

This service can be used to clear parameter structure flags.

37.3.4.2.2 How to Use the Service

37.3.4.2.3 Description

This service clears CarryOut, CarryIn, Zero and Violation flags in the Specific bit field. The Gf2n flag is untouched.

The service name for this operation is `ClearFlags`.

37.3.4.2.4 Parameters Definition

It is possible to directly address this service through the `PUKCL_ClearFlags()` macro.

Table 37-7. Clear Flags Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
Specific/CarryOut	Bit	O	–	–	–	Cleared
Specific/CarryIn	Bit	O	–	–	–	Cleared
Specific/Zero	Bit	O	–	–	–	Cleared

.....continued						
Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
Specific/Violation	Bit	O	–	–	–	Cleared

37.3.4.2.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ClearFlags,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            // Success
            }
else // Manage the error
```

37.3.4.2.6 Status Returned Values

Table 37-8. ClearFlags Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly.

37.3.4.3 Swap

37.3.4.3.1 Purpose

This service performs swapping of two buffers.

37.3.4.3.2 How to Use the Service

37.3.4.3.3 Description

This service swaps two buffers, X and Y, of the same size in memory.

The service name for this operation is `Swap`.

37.3.4.3.4 Parameters Definition

This service can easily be accessed through the use of the `PUKCL_Swap()` macro.

Table 37-9. Swap Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1XBase	nu1	I	Crypto RAM	u2Length	Base of the number X	Base of X filled with Y
nu1YBase	nu1	I	Crypto RAM	u2Length	Base of the number Y	Base of Y filled with X
u2XLength	u2	I	–	–	Length of X and Y	Length of X and Y

37.3.4.3.5 Code Example

```
_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// Initialize parameters
PUKCL_Swap(nu1XBase) = <Base of the X number>;
PUKCL_Swap(nu1YBase) = <Base of the Y number>;
PUKCL_Swap(u2XLength) = <Length of the numbers>;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(Swap,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.4.3.6 Constraints

The following conditions must be avoided to ensure that the service works correctly:

- nu1XBase or nu1YBase are not aligned on 32-bit boundaries
- u2XLength is either <4, > 0xffc, or not a 32-bit length
- {nu1XBase, u2XLength} or {nu1YBase, u2XLength} do not entirely lie in PUKCCRAM
- {nu1XBase, u2XLength} overlaps {nu1YBase,u2YLength}

37.3.4.3.7 Status Returned Values

Table 37-10. Swap Service Return Codes

Returned status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly

37.3.4.4 Fill

37.3.4.4.1 Purpose

This service performs a memory fill operation, with a given 32-bit constant.

37.3.4.4.2 How to Use the Service

37.3.4.4.3 Description

This service fills a Crypto RAM space with a provided 32-bit constant: Fill (R, FillValue)

The service name for this operation is `Fill`.

37.3.4.4.4 Parameters Definition

This service can easily be accessed through the use of the `PUKCL_Fill()` macro.

Table 37-11. Fill Service Parameters

Parameter	Type	Direction.	Location	Data Length	Before Executing the Service	After Executing the Service
nu1RBase	nu1	I	Crypto RAM	u2RLength	Base of R	Base of R value filled repetitively with u4FillValue
u2RLength	u2	I	Crypto RAM	–	Length of R	Length of R
u4FillValue	u4	I	–	–	Filling value	Filling value

37.3.4.4.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// Initialize parameters
PUKCL_Fill(nu1RBase) = <Base of the R number>;
PUKCL_Fill(u2RLength) = <Length of the R number>;
PUKCL_Fill(u4FillValue) = <32-bits value to fill with>;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(Fill,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.4.4.6 Constraints

The following conditions must be avoided to ensure that the service works correctly:

- nu1RBase are not aligned on 32-bit boundaries
- u2RLength is either: <4, >0xffc or not a 32-bit length

- {nu1RBase, u2RLength} do not entirely lie in Crypto RAM

37.3.4.4.7 Status Returned Values

Table 37-12. Fill Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly.

37.3.4.5 Fast Copy/Clear

37.3.4.5.1 Purpose

This service performs a copy from a memory area to another or a memory area clear.

37.3.4.5.2 How to Use the Service

37.3.4.5.3 Description

This service copies a number X into another number R, padding with zero on the MSB side up to the length specified for R.

$R = X$

If the lengths of R and X are equal, a complete fast copy is processed.

If the length of R is strictly greater than the length of X, X is first copied in the Low Significant Bytes side of R, and R is padded with zeros on the Most Significant Bytes side.

If the pointer on the X area equals zero, R is filled with zeros. This operation can also be made by using the Fill service (see 37.3.4.4. Fill).

The service name for this operation is `FastCopy`.

Important: The length of R must be greater or equal to the length of X.

37.3.4.5.4 Parameters Definition

This service can easily be accessed through the use of the `PUKCL_FastCopy()` macro.

Table 37-13. FastCopy Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1XBase	nu1	I	Crypto RAM	u2XLength	Base of X	Base of X number untouched
nu1RBase	nu1	I	Crypto RAM	u2RLength	Base of R	Base of R filled with X
u2RLength	u2	I	–	–	Length of R	Length of R
u2XLength	u2	I	–	–	Length of X	Length of X

37.3.4.5.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// Initialize parameters
PUKCL_FastCopy(nu1XBase) = <Base of the X number>;
PUKCL_FastCopy(nu1RBase) = <Base of the R number>;
PUKCL_FastCopy(u2XLength) = <Length of the X number>;
PUKCL_FastCopy(u2RLength) = <Length of the R number>;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(FastCopy,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            ...
```

```
            }
else // Manage the error
```

37.3.4.5.6 Constraints

The parameter placements that are not allowed are are as follows.

If nu1XBase equals zero, no checks are made on nu1XBase (fixed) and u2XLength (unused).

The following conditions must be avoided to ensure that the service works correctly:

- nu1XBase or nu1RBase are not aligned on 32-bit boundaries
- u2XLength or u2RLength is either: <4, >0xffc or not a 32-bit length or u2XLength >u2RLength
- {nu1XBase, u2XLength} or {nu1RBase, u2RLength} do not entirely lie in Crypto RAM
- {nu1XBase, u2XLength} overlaps {nu1RBase,u2RLength}

37.3.4.5.7 Status Returned Values

Table 37-14. FastCopy Service Return Codes

Returned status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly

37.3.4.6 Conditional Copy/Clear

37.3.4.6.1 Purpose

This service conditionally performs a copy from a memory area to another or a memory area clear.

37.3.4.6.2 How to Use the Service

37.3.4.6.3 Description

This service copies a number X into another number R, padding with zero on the MSB side up to the length specified for R. This copy operation is performed under the conditions specified in the options.

If the condition is verified, R = X.

The copy or clear action is made under condition.

The four possible options for the condition are described in the following table. Two of the conditions check the Specific.CarryIn bit.

The processing is done as follows:

- If the condition is not verified, nothing is processed.
- If the condition is verified the copy or clear follows the rules:
  - If the lengths of R and X are equal, a complete fast copy is processed
  - If the length of R is strictly greater than the length of X, X is first copied in the Low Significant Bytes side of R, and R is padded with zeros on the Most Significant Bytes side.
  - If the pointer on the X area equals zero, R is filled with zeros.

The service name for this operation is `CondCopy`.

Important: If the condition is verified, the length of R must be greater or equal to the length of X.

37.3.4.6.4 Parameters Definition

This service can easily be accessed through the use of the `PUKCL_CondCopy()` and `PUKCL()` macros.

Table 37-15. CondCopy Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	–	–	Option for condition (see the following table)	Option for condition (see the following table)
Specific/CarryIn	Bit	I	–	–	Bit CarryIn	Bit CarryIn
nu1XBase	nu1	I	Crypto RAM	u2XLength	Base of X	Base of X number untouched
nu1RBase	nu1	I	Crypto RAM	u2RLength	Base of R	Base of R filled with X if condition holds
u2RLength	u2	I	–	–	Length of R	Length of R
u2XLength	u2	I	–	–	Length of X	Length of X

37.3.4.6.5 Available Options

The option for the condition is set by the u2Options input parameter that must take one of the values listed in the following table.

Table 37-16. CondCopy Service Options

Option	Purpose	Needed parameters
PUKCL_CONDCOPY_ALWAYS	Always perform the copy	nu1XBase,u2XLength,nu1RBase, u2RLength
PUKCL_CONDCOPY_NEVER	Never perform the copy	None
PUKCL_CONDCOPY_IF_CARRY	Perform the copy if CarryIn is 1	Specific/CarryIn nu1XBase,u2XLength,nu1RBase, u2RLength
PUKCL_CONDCOPY_IF_NOT_CARRY	Perform the copy if CarryIn is zero	Specific/CarryIn nu1XBase,u2XLength,nu1RBase, u2RLength

37.3.4.6.6 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// CarryIn shall be beforehand filled (with zero or one) PUKCL(Specific).CarryIn = ...;

// Condition Option PUKCL(u2Options) = ...;

// Initialize parameters
PUKCL_CondCopy(nu1XBase) = <Base of the X number>;
PUKCL_CondCopy(nu1RBase) = <Base of the R number>;
PUKCL_CondCopy(u2XLength) = <Length of the X number>;
PUKCL_CondCopy(u2RLength) = <Length of the R number>;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(CondCopy,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.4.6.7 Constraints

The parameters placement that are not allowed are listed below.

If the conditional option and the CarryIn do not lead to execute the copy, no checks are made on the constraints to be respected.

If nu1XBase equals zero, no checks are made on nu1XBase (fixed) and u2XLength (unused).

The following conditions must be avoided to ensure that the service works correctly:

- nu1XBase or nu1RBase are not aligned on 32-bit boundaries
- u2XLength or u2RLength is either: <4, >0xffc or not a 32-bit length or u2XLength >u2RLength
- {nu1XBase, u2XLength} or {nu1RBase, u2RLength} do not entirely lie in Crypto RAM

- {nu1XBase, u2XLength} overlaps {nu1RBase,u2RLength}

37.3.4.6.8 Status Returned Values

Table 37-17. CondCopy Service Return Codes

Returned status	Importance	Meaning
PUKCL_WRONG_SERVICE	Severe	An inconsistency has been detected between the called service and the provided service number.
PUKCL_OK	–	Service functioned correctly

37.3.4.7 Small Multiply, Add, Subtract, Exclusive OR

Related Links

37.3.4.5. Fast Copy/Clear
37.3.5.1. Modular Reduction

37.3.4.7.1 Purpose

This purpose of this service is to multiply a large number X by a single-word number, MulValue, and perform an optional accumulation/subtract with a large number Z, returning the result R.

The following options are available:

- Work in the GF(2^n) or in the standard GF(p) arithmetic integer field
- Add of a supplemental CarryOperand
- Overlap of the operands is possible, taking into account some constraints
- Modulo-reduction of the computation result (see *Modular Reduction* from Related Links)

In addition to a multiply, possible uses of this service can include:

- Copy a block of data from one place to another (if u4MulValue is 1). This operation can alternatively be made by using the Fast Copy service (see *Fast Copy/Clear* from Related Links)
- Adding/Subtracting two numbers (if u4MulValue is1)
- Xoring two blocks of data (if u4MulValue is 1 and the selected mathematical field is GF(2^n))

37.3.4.7.2 How to Use the Service

37.3.4.7.3 Description

This service processes the following operation (if not computing a modular reduction of the result):

$R = [Z] \pm (MulValue \times X + CarryOperand)$

Or (if computing a modular reduction of the result):

$R = ([Z] \pm (MulValue \times X + CarryOperand)) mod\ N$

The service name for this operation is `Smult`.

The result of the Small Multiply Operation is stored on u2RLength bytes, so the choice of this length compared to u2XLength may lead to:

- A truncation if the result is too big to be stored on u2RLengthbytes.
- A padding on the MSB side if the result does not take all the u2RLengthbytes.
  However, in all cases this rule must be followed:

> **Important:** The length of R must be greater than or equal to the length of X.

In these computations, the following parameters need to be provided:

- R the result (pointed by{nu1RBase,u2Rlength})
- X one input number or GF(2^n) polynomial (pointed by{nu1XBase,u2XLength})
- Z one optional input number or GF(2^n) polynomial (pointed by{nu1ZBase,u2Rlength}).
- MulValue one input number or GF(2^n)polynomial on one word (provided in u4MulValue)
- CarryOperand (provided through the CarryOptions and Carry values).

Important: Even if neither accumulation nor subtraction is specified, the nu1ZBase must always be filled and point to a Crypto RAM space. It this case, nu1ZBase can point to the same space as the nu1RBase.

If using the modular reduction option, the Multiply operation is followed by a reduction (see *Modular Reduction* from Related Links) and the following parameters must be additionally provided:

- N—the modulus (pointed by {nu1ModBase,u2Modlength +4})
- Cns—the reduction constant
  - In case of Big reduction, Cns is pointed by {nu1CnsBase,64bytes}.
  - In case of Fast or Normalized reduction, Cns is pointed by {nu1CnsBase,u2ModLength +8}

Important:
The result buffer R must first be padded with zero bytes until its length is sufficient to perform the reduction (2*u2ModLength + 8) to be used by the Modular Reduction service as an input parameter.

The result of the reduction is written in the area X pointed by {nu1XBase, u2ModLength + 4}.

- For example, if relevant u2ModLength is 0x80 bytes and u2XLength is 0x80 too, the length of the Rspace may be 2*(u2ModLength + 4) = 0x108 bytes.
  In case of fast or normalized reduction, the length of the result may be u2ModLength + 4 so 0x84 bytes. Therefore, the zone X may lengths 0x84 bytes (at least). The multiplication of X by 1 word provide a result in the zone R which MSB bytes will be padded with zero bytes.
  In that example, the length of the zone R will be 2*u2ModLength + 8 = 0x108 bytes.

37.3.4.7.4 Parameters Definition

Table 37-18. Smult Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	—	—	Options (see below)	Options (see below)
Specific/Gf2n CarryIn	Bits	I	—	—	GF(2^n) Bit and Carry In	—
Specific/CarryOut Zero Violation	Bits	I	—	—	—	Carry Out, Zero Bit and Violation Bit filled according to the result
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of N	Base of N untouched
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns untouched
u2ModLength	u2	I	—	—	Length of N	Length of N
nu1XBase	nu1	I	Crypto RAM	u2XLength or u2ModLength + 4(1)	Base of X	Base of X(2)
u2XLength	u2	I	—	—	Length of X	Length of X

.....continued

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ZBase	nu1	I	Crypto RAM	u2RLength	Base of Z	Base of Z untouched
nu1RBase	nu1	I	Crypto RAM	u2RLength	Base of R	Base of R (see **Note 3**)
u2RLength	u2	I	—	—	Length of R	Length of R
u4MulValue	u4	I	—	—	Value of MulValue	Value of MulValue untouched

Notes:

1. If a reduction option is specified, the area X will be, if necessary, extended to u2ModLength + 4 bytes.
2. If Smult is without reduction, X is untouched. If Smult is with reduction, X is filled with the final result.
3. If Smult is without reduction, R is filled with the final result. If Smult is with reduction, R is corrupted.

37.3.4.7.5 Available Options

The options are set by the u2Options input parameter, which is composed of:

- The mandatory Small Multiplication operation option described in the following table.
- The mandatory CarryOperand option described in Smult Service (with Accumulate/Subtract From) Carry Settings and Smult Service Carry Settings tables.
- The facultative Modular Reduction option (see Modular Reduction). If the Modular Reduction is not requested, this option is absent.

The u2Options number is calculated by an “Inclusive OR” of the options. Some examples in C language are:

- Operation: Small Multiply only without carry and without Modular Reduction
  ```
  PUKCL(u2Options) = SET_MULTIPLIEROPTION(PUKCL_SMULT_ONLY) |
  SET_CARRYOPTION(CARRY_NONE);
  ```
- Operation: Small Multiply with addition with Specific/CarryIn addition and with Fast Modular Reduction
  ```
  PUKCL(u2Options) =SET_MULTIPLIEROPTION(PUKCL_SMULT_ADD) |
  SET_CARRYOPTION(ADD_CARRY) | PUKCL_REDMOD_REDUCTION |
  PUKCL_REDMOD_USING_FASTRED;
  ```

The following table lists all of the necessary parameters for the Small Multiply option. When the Addition or Subtraction option is not chosen, it is not necessary to fill in the nu1ZBase parameter.

Table 37-19. Smult Service Operation Options

Option	Purpose	Required Parameters
SET_MULTIPLIEROPTION(PUKCL_SMULT_ ONLY)	Perform R = MulValue*X + CarryOperand	nu1RBase, u2RLength, nu1XBase, u2XLength, u4MulValue
SET_MULTIPLIEROPTION(PUKCL_SMULT_ ADD)	Perform R = Z + MulValue*X + CarryOperand	nu1RBase, u2RLength, nu1ZBase, nu1XBase, u2XLength,u4MulValue
SET_MULTIPLIEROPTION(PUKCL_SMULT_ SUB)	Perform R = Z - (MulValue*X + CarryOperand)	nu1RBase, u2RLength, nu1ZBase, nu1XBase, u2XLength,u4MulValue

37.3.4.7.6 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// Gf2n and CarryIn shall be beforehand filled (with zero or one)
PUKCL(Specific).Gf2n = ...;
```

```
PUKCL(Specific).CarryIn = ...; PUKCL(u2Options) =...;

// Depending on the option specified, all fields must not be filled
PUKCL_Smult(nu1XBase) = <Base of the X number>;
PUKCL_Smult(u2XLength) = <Length of the X number>;
PUKCL_Smult(nu1RBase) = <Base of the R number>;
PUKCL_Smult(u2RLength) = <Length of the R number>;
PUKCL_Smult(nu1ZBase) = <Base of the Z number>;
PUKCL_Smult(u4MulValue) = <Value to be multiplied with>;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(Smult,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            // The Small multiplication has been executed correctly
            ...
            }
else // Manage the error
```

Note:
The length of R must be greater or equal to the length of X. Additional options are available through the use of a modular reduction to be executed at the end of this operation. Some important considerations have to be taken into account concerning the length of resulting operands to get a mathematically correct result.

The output of this operation is not obviously compatible with the modular reduction, as it may be either smaller or bigger. In the case (most of the time) where the result (pointed by nu1RBase) is smaller in size than twice the modulus plus one word, it is mandatory to add padding bytes to zero. Otherwise, the reduced value will be taken considering the high order words (potentially uninitialized) as part of the number, thus resulting in a mathematically correct but unexpected result.

In the case that the result is bigger than twice the modulus plus one word, the modular reduction feature has to be executed as a separate operation, using an Euclidean division.

37.3.4.7.7 Constraints

For the case of a small multiplication with an option indicating either subtraction or accumulation, the following conditions must be avoided to ensure the service works correctly:

- nu1XBase, nu1RBase or nu1ZBase are not aligned on 32-bit boundaries
- {nu1XBase, u2XLength}, {nu1ZLength, u2RLength} or {nu1RBase, u2RLength} do not entirely lie in Crypto RAM
- u2XLength or u2RLength is either: < 4, > 0xffc or not a 32-bit length or u2XLength >u2RLength
- {nu1RBase, u2RLength} overlaps {nu1XBase, u2XLength} or nu1R < nu1Z and {nu1RBase,u2RLength} overlaps {nu1ZBase, u2RLength}

If the nu1R value is greater or equals to the nu1Z one, the overlapping between R and Z is allowed.

If a modular reduction is specified, the relevant parameters must be defined according to the chosen reduction and follow the description in Modular Reduction. Additional constraints to be respected and error codes are described in this section and in Smult Service Return Codes.

Multiplication with Accumulation or Subtraction

When the options bits specify that either an Accumulation or a Subtraction must be performed, this service performs the following operation:

$R = (Z \pm (MulValue \times X + CarryOperand)) mod\ B^{RLength}$

Table 37-20. Smult Service (with Accumulate/Subtract From) Carry Settings

Carry Options	CarryOperand	Resulting Operation
SET_CARRYOPTION(ADD_CARRY)	CarryIn	R = Z ± (MulValue*X + CarryIn)
SET_CARRYOPTION(SUB_CARRY)	- CarryIn	R = Z ± (MulValue*X - CarryIn)

.....continued		
Carry Options	CarryOperand	Resulting Operation
SET_CARRYOPTION(ADD_1_PLUS_CARRY)	1 + CarryIn	R = Z ± (MulValue*X + 1 + CarryIn)
SET_CARRYOPTION(ADD_1_MINUS_CARRY)	1 - CarryIn	R = Z ± (MulValue*X + 1 - CarryIn)
SET_CARRYOPTION(CARRY_NONE)	0	R = Z ± (MulValue*X)
SET_CARRYOPTION(ADD_1)	1	R = Z ± (MulValue*X + 1)
SET_CARRYOPTION(SUB_1)	- 1	R = Z ± (MulValue*X - 1)
SET_CARRYOPTION(ADD_2)	2	R = Z ± (MulValue*X + 2)

Multiplication without Accumulation or Subtraction

When the case the options bits specify that neither an Accumulation nor a Subtraction must be performed, this service performs the following operation:

$R = (MulValue \times X + CarryOperand) mod\ B^{RLength}$

Table 37-21. Smult Service Carry Settings

Carry Options	CarryOperand	Resulting Operation
SET_CARRYOPTION(ADD_CARRY)	CarryIn	R = MulValue*X + CarryIn
SET_CARRYOPTION(SUB_CARRY)	- CarryIn	R = MulValue*X - CarryIn
SET_CARRYOPTION(ADD_1_PLUS_CARRY)	1 + CarryIn	R = MulValue*X + 1 + CarryIn
SET_CARRYOPTION(ADD_1_MINUS_CARRY)	1 - CarryIn	R = MulValue*X + 1 - CarryIn
SET_CARRYOPTION(CARRY_NONE)	0	R = MulValue*X
SET_CARRYOPTION(ADD_1)	1	R = MulValue*X + 1
SET_CARRYOPTION(SUB_1)	-1	R = MulValue*X - 1
SET_CARRYOPTION(ADD_2)	2	R = MulValue*X + 2

37.3.4.7.8 Status Returned Values

Table 37-22. Smult Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	—	Service functioned correctly

37.3.4.8 Compare

37.3.4.8.1 Purpose

The purpose of this service is to compare two numbers in classical arithmetic GF(p).

Important: This service works only with integers.

37.3.4.8.2 How to Use the Service

37.3.4.8.3 Description

This service accepts two numbers in classical arithmetic in input and performs a comparison, virtually subtracting (X + CarryIn) from Y:

CompareGetFlags (Y - (X + CarryIn))

The numbers X and Y are untouched but the resulting flags CarryOut and the Zero Bit are filled. If the lengths of Y and X are equal, a comparison is processed.

If the length of Y is strictly greater than the length of X, X is first virtually padded with zeros on the Most Significant Bytes side, then a comparison is processed.

Note: The length of Y must be greater or equal to the length of X.

In this computation, the following data need to be provided:

- X (pointed by{nu1XBase,u2XLength})
- Y (pointed by{nu1YBase,u2YLength})

The service name for this operation is `Comp`.

37.3.4.8.4 Parameters Definition

Table 37-23. Comp Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
Specific/Gf2n CarryIn	Bits	I	–	–	GF(2n) Bit and Carry In	–
Specific/CarryOut Zero Violation	Bits	I	–	–	–	Carry Out, Zero Bit and Violation Bit filled according to the result
nu1XBase	nu1	I	Crypto RAM	u2XLength	Base of X	Base of X
u2XLength	u2	I	–	–	Length of X	Length of X
nu1YBase	nu1	I	Crypto RAM	u2YLength	Base of Y	Base of Y
u2YLength	u2	I	–	–	Length of Y	Length of Y

37.3.4.8.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// CarryIn shall be beforehand filled (with zero or one) PUKCL(Specific).CarryIn = ...;

// Initializing parameters
PUKCL_Comp(nu1XBase) = <Base of the ram location of X>;
PUKCL_Comp(u2XLength) = <Length of X>;
PUKCL_Comp(nu1YBase) = <Base of the ram location of Y>;
PUKCL_Comp(u2YLength) = <Length of Y>;

// vPUKCL_Process() is a macro command,
// and then calls the library...
vPUKCL_Process(Comp,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            // The COMPARE has been executed correctly
            // CarryOut, Zero ... are available
            ... = PUKCL(Specific).CarryOut;
            ... = PUKCL(Specific).Zero;
            }
else // Manage the error
```

37.3.4.8.6 Constraints

The following conditions must be avoided to ensure that the service works correctly:

- nu1XBase or nu1YBase are not aligned on 32-bit boundaries
- {nu1XBase, u2XLength} or {nu1YLength, u2YLength} are not in Crypto RAM
- u2XLength or u2YLength is either: < 4, > 0xffc or not a 32-bit length or u2XLength >u2YLength

37.3.4.8.7 Status Returned Values

Table 37-24. Comp Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly

37.3.4.9 Full Multiply

Related Links

37.3.5.1. Modular Reduction

37.3.4.9.1 Purpose

The purpose of this service is to multiply two large numbers, X and Y, and optionally accumulate/subtract from a third large number, Z, returning the result, R.

The available options are as follows:

- Work in the GF(2^n) field or in the standard arithmetic field
- Add of a supplemental CarryOperand
- Overlap of the operands is possible, taking into account some constraints
- Modular Reduction of the computation result (see *Modular Reduction* from Related Links)

37.3.4.9.2 How to Use the Service

37.3.4.9.3 Description

This service provides the following (if not computing a modular reduction of the result):

$R = [Z] \pm (X \times Y + CarryOperand)$

Or (if computing a modular reduction of the result):

$R = ([Z] \pm (X \times Y + CarryOperand)) mod\ N$

The service name for this operation is `Fmult`.

In these computations, the following data has to be provided:

- R the result (pointed by {nu1RBase,u2Xlength +u2YLength})
- X one input number or GF(2^n) polynomial (pointed by{nu1XBase,u2XLength})
- Y one input number or GF(2^n) polynomial (pointed by{nu1YBase,u2YLength})
- Z one optional input number or GF(2n) polynomial (pointed by {nu1ZBase,u2Xlength +u2YLength})
- CarryOperand (provided through the Carry Options and Carry values)

Important: Even if neither accumulation nor subtraction is specified, the nu1ZBase must always be filled and point to a Crypto RAM space. It this case, nu1ZBase can point to the same space as the nu1RBase.

If using the big modular reduction option, the Multiply operation is followed by a reduction (see *Modular Reduction* from Related Links). In this case, the length of Cns is 64 bytes.

If using the modular reduction option, the Multiply operation is followed by a reduction (see *Modular Reduction* from Related Links). In this case the following parameters must be additionally provided:

- N—the modulus (pointed by {nu1ModBase,u2Modlength +4})
- Cns—the reduction constant
  - In case of Big reduction, Cns is pointed by {nu1CnsBase,64bytes}.
  - In case of Fast or Normalized reduction, Cns is pointed by (pointed by {nu1CnsBase,u2ModLength+ 8})

Note:
The result buffer R must first be padded with zero bytes until its length is sufficient to perform the reduction (2*u2ModLength + 8) to be used by the Modular Reduction service as an input parameter.

The result of the reduction is written in the area X pointed by {nu1XBase, u2ModLength + 4}.

For example, if u2ModLength, u2XLength and u2YLength are 0x80 bytes, the length of the R space is 2*(u2ModLength + 4) = 0x108 bytes because of the constraints of modular reduction.

In case of Fast or Normalized Reduction, the length of the result is u2ModLength + 4 so 0x84 bytes. Thus, the zone X has a length of 0x84 bytes (at least). The multiplication of X by Y provides a result of length 0x100 bytes in the zone R so the 8 MSB bytes must be previously padded with zero bytes (in offsets 0x100 to 0x107).

37.3.4.9.4 Parameters Definition

Table 37-25. Fmult Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	–	–	Options (see below)	Options (see below)
Specific/Gf2n CarryIn	Bits	I	–	–	GF(2n) Bit and Carry In	–
Specific/CarryOut Zero Violation	Bits	I	–	–	–	Carry Out, Zero Bit and Violation Bit filled according to the result
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of N	Base of N untouched
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8 or 64 bytes	Base of Cns	Base of Cns untouched
u2ModLength	u2	I	–	–	Length of N	Length of N
nu1XBase	nu1	I	Crypto RAM	u2XLength or u2ModLength + 4(1)	Base of X	Base of X(2)
u2XLength	u2	I	–	–	Length of X	Length of X
nu1YBase	nu1	I	Crypto RAM	u2YLength	Base of Y	Base of Y
u2YLength	u2	I	–	–	Length of Y	Length of Y
nu1ZBase	nu1	I	Crypto RAM	u2XLength + u2YLength	Base of Z	Base of Z untouched
nu1RBase	nu1	I	Crypto RAM	u2XLength + u2YLength	Base of R	Base of R(3)

Notes:

1. In case of a reduction option is specified, if necessary, the area X will be extended to u2ModLength + 4 bytes.
2. If FMult is without reduction, X is untouched. If FMult is with reduction, X is filled with the final result.
3. If FMult is without reduction, R is filled with the final result. If FMult is with reduction, R is corrupted.

37.3.4.9.5 Available Options

The options are set by the u2Options input parameter, which is composed of:

- the mandatory Full Multiplication operation option described in Table 37-26
- the mandatory CarryOperand option described in Table 37-27 and Table 37-28
- the facultative Modular Reduction option(see *Modular Reduction* from Related Links). If the Modular Reduction is not requested, this option is absent.

The u2Options number is calculated by an Inclusive OR of the options.

Some Examples in C language are:

- Operation: Full Multiply only without carry and without Modular Reduction
  ```
  PUKCL(u2Options) = SET_MULTIPLIEROPTION(PUKCL_FMULT_ONLY) |
  SET_CARRYOPTION(CARRY_NONE);
  ```
- Operation: Full Multiply with addition with Specific/CarryIn addition and with Fast Modular Reduction
  ```
  PUKCL(u2Options) = SET_MULTIPLIEROPTION(PUKCL_FMULT_ADD) |
  SET_CARRYOPTION(ADD_CARRY) |
  PUKCL_REDMOD_REDUCTION |
  PUKCL_REDMOD_USING_FASTRED;
  ```

The following table shows all of the necessary parameters for the Full Multiply option. When the Addition or Subtraction option is not chosen, it is not necessary to fill in the nu1ZBase parameter.

Table 37-26. Fmult Service Options

Option	Purpose	Required Parameters
SET_MULTIPLIEROPTION(PUKCL_FMUL_ONLY)	Perform R = X*Y + CarryOperand	nu1RBase, nu1YBase, u2YLength, nu1XBase, u2XLength
SET_MULTIPLIEROPTION(PUKCL_FMUL_ADD)	Perform R = Z + X*Y + CarryOperand	nu1RBase, nu1ZBase, nu1YBase, u2YLength, nu1XBase, u2XLength
SET_MULTIPLIEROPTION(PUKCL_FMUL_SUB)	Perform R = Z - (X*Y + CarryOperand)	nu1RBase, nu1ZBase, nu1YBase, u2YLength, nu1Xlength, u2XLength

37.3.4.9.6 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;


// Gf2n and CarryIn shall be beforehand filled (with zero or one)
PUKCL(Specific).Gf2n = ...;
PUKCL(Specific).CarryIn = ...;

PUKCL(u2Option) =...;
// Depending on the option specified, not all fields must be filled
PUKCL_Fmult(nu1XBase) = <Base of the ram location of X>;
PUKCL_Fmult(u2XLength) = <Length of X>;
PUKCL_Fmult(nu1YBase) = <Base of the ram location of Y>;
PUKCL_Fmult(u2YLength) = <Length of Y>;
PUKCL_Fmult(nu1ZBase) = <Base of the ram location of Z>;
PUKCL_Fmult(nu1RBase) = <Base of the ram location of R>;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(Fmult,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            // The Full multiply has been executed correctly
            ...
            }
else // Manage the error
```

37.3.4.9.7 Important Considerations for Modular Reduction of a Fmult Computation Result

Note:
Additional options are available through the use of a modular reduction to be executed at the end of this operation. Some important considerations have to be taken into account concerning the length of resulting operands to get a mathematically correct result.

The output of this operation is not always compatible with the modular reduction as it may be either smaller or bigger. In the case (most of the time) the result (pointed by nu1RBase) is smaller in size than “twice the modulus plus one word” by one word, a padding word must be added to zero. Otherwise, the reduced value will be taken considering the high order words (potentially uninitialized) as part of the number, thus resulting in getting a mathematically correct but unexpected result.

In the case that the result is bigger than twice the modulus plus one word, the modular reduction feature has to be executed as a separate operation, using an Euclidean division.

37.3.4.9.8 Constraints

The following conditions must be avoided to ensure that the service works correctly:

- nu1XBase, nu1YBase, nu1RBase or nu1ZBase are not aligned on 32-bit boundaries
- {nu1XBase, u2XLength}, {nu1YLength, u2YLength}, {nu1ZBase, u2XLength+u2YLength} or{nu1RBase, u2XLength+u2YLength} are not in Crypto RAM
- u2XLength, u2YLength is either: < 4, > 0xffc or not a 32-bit length
- {nu1RBase, u2XLength+u2YLength} overlaps {nu1YBase, u2YLength} or{nu1RBase, u2XLength+u2YLength} overlaps {nu1XBase, u2XLength}
- {nu1RBase, u2XLength+u2YLength} overlaps {nu1ZBase, u2XLength+u2YLength} and nu1RBase> nu1ZBase

If a modular reduction is specified, the relevant parameters must be defined according to the chosen reduction and follow the description in Modular Reduction (see *Modular Reduction* from Related Links). Additional constraints to be respected and error codes are described in this section and in Table 37-49.

Multiplication with Accumulation or Subtraction

In the case where the options bits specify that either an Accumulation or a subtraction must be performed, this service performs the following operation:

$R = (Z \pm (X \times Y + CarryOperand)) mod\ B^{XLength + YLength}$

Table 37-27. Fmult Service (with Accumulate/Subtract From) Carry Settings

Option AND CARRYOPTIONS	CarryOperand	Resulting Operation
SET_CARRYOPTION(ADD_CARRY)	CarryIn	R = Z ± (X*Y + CarryIn)
SET_CARRYOPTION(SUB_CARRY)	- CarryIn	R = Z ± (X*Y - CarryIn)
SET_CARRYOPTION(ADD_1_PLUS_CARRY)	1 + CarryIn	R = Z ± (X*Y + 1 + CarryIn)
SET_CARRYOPTION(ADD_1_MINUS_CARRY)	1 - CarryIn	R = Z ± (X*Y + 1 - CarryIn)
SET_CARRYOPTION(CARRY_NONE)	0	R = Z ± (X*Y)
SET_CARRYOPTION(ADD_1)	1	R = Z ± (X*Y + 1)
SET_CARRYOPTION(SUB_1)	- 1	R = Z ± (X*Y - 1)
SET_CARRYOPTION(ADD_2)	2	R = Z ± (X*Y + 2)

Multiplication without Accumulation or Subtraction

In the case the options bits specify that either an Accumulation or a subtraction must be performed, this service performs the following operation:

$R = (X \times Y + CarryOperand) mod\ B^{XLength + YLength}$

Table 37-28. Fmult Service Carry Settings

Option AND CARRYOPTIONS	CarryOperand	Resulting Operation
SET_CARRYOPTION(ADD_CARRY)	CarryIn	R = X*Y + CarryIn
SET_CARRYOPTION(SUB_CARRY)	- CarryIn	R = X*Y - CarryIn
SET_CARRYOPTION(ADD_1_PLUS_CARRY)	1 + CarryIn	R = X*Y + 1 + CarryIn
SET_CARRYOPTION(ADD_1_MINUS_CARRY)	1 - CarryIn	R = X*Y + 1 - CarryIn
SET_CARRYOPTION(CARRY_NONE)	0	R = X*Y
SET_CARRYOPTION(ADD_1)	1	R = X*Y + 1
SET_CARRYOPTION(SUB_1)	- 1	R = X*Y - 1
SET_CARRYOPTION(ADD_2)	2	R = X*Y + 2

37.3.4.9.9 Status Returned Values

Table 37-29. Fmult Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly

37.3.4.10 Square

Related Links

37.3.5.1. Modular Reduction

37.3.4.10.1 Purpose

The purpose of this service is to compute the square of a big number and optionally accumulate/subtract from a second big number.

Please note that this service uses an optimized implementation of the squaring. It also means that when the GF(2^n) flag is set, the execution time will be smaller than when not set (in that case, the squaring execution time will still be smaller than for a standard multiplication).

The available options are as follows:

- Work in the GF(2^n) or in the standard integer arithmetic field
- Add of a supplemental CarryOperand
- Overlapping of the operands is possible, taking into account some constraints
- Modular Reduction of the computation result

37.3.4.10.2 How to Use the Service

37.3.4.10.3 Description

This service provides the following (if not computing a modular reduction of the result):

$R = [Z] \pm (X^2 + CarryOperand)$

Or (if computing a modular reduction of the result):

$R = ([Z] \pm (X^2 + CarryOperand))mod\ N$

The service name for this operation is `Square`.

In these computations, the following data has to be provided:

- R the result (pointed by {nu1RBase,2 *u2Xlength})
- X one input number or GF(2n) polynomial (pointed by{nu1XBase,u2XLength})
- Z one optional input number or GF(2n) polynomial (pointed by {nu1ZBase,2 *u2Xlength})
- CarryOperand (provided through the CarryOptions and Carry values)

Important: Even if neither accumulation nor subtraction is specified, the nu1ZBase must always be filled and point to a Crypto RAM space. It this case, nu1ZBase can point to the same space as the nu1RBase.

If using the big modular reduction option, the Multiply operation is followed by a reduction (see *Modular Reduction* from Related Links). In this case, the length of Cns is 64 bytes.

If using the modular reduction option the Square operation is followed by a reduction (see *Modular Reduction* from Related Links). In this case the following parameters must be additionally provided:

- N—the modulus (pointed by {nu1ModBase,u2Modlength +4}).
- Cns—the reduction constant (pointed by {nu1CnsBase,u2Modlength +8})
  - In case of big reduction option, the length of Cns is 64bytes.

Note:
The result buffer R must first be padded with zero bytes until its length is sufficient to perform the reduction (2*u2ModLength + 8) to be used by the Modular Reduction service as an input parameter.

The result of the reduction is written in the area X pointed by {nu1XBase, u2ModLength + 4}.

For example, if u2ModLength, u2XLength is 0x80 bytes, the length of the R space is 2*(u2ModLength + 4) = 0x108 bytes because of the constraints of modular reduction.

In case of Fast or Normalized Reduction, the length of the result is u2ModLength + 4 so 0x84 bytes. Thus, the zoneX has a length of 0x84 bytes (at least). The square of X provides a result of length 0x100 bytes in the zone R so the 8 MSB bytes previously must be previously padded with zero bytes (in offsets 0x100 to 0x107).

37.3.4.10.4 Parameters Definition

Table 37-30. Square Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	–	–	Options (see below)	Options (see below)
Specific/Gf2n CarryIn	Bits	I	–	–	GF(2n) Bit and Carry In	–
Specific/CarryOut Zero Violation	Bits	I	–	–	–	Carry Out, Zero Bit and Violation Bit filled according to the result
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of N	Base of N untouched
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8 or 64 bytes	Base of Cns	Base of Cns untouched
u2ModLength	u2	I	–	–	Length of N	Length of N
nu1XBase	nu1	I	Crypto RAM	u2XLength or u2ModLength + 4(1)	Base of X	Base of X(2)
u2XLength	u2	I	–	–	Length of X	Length of X
nu1ZBase	nu1	I	Crypto RAM	2 * u2XLength	Base of Z	Base of Z
nu1RBase	nu1	I	Crypto RAM	2 * u2XLength	Base of R	Base of R(3)

Notes:

1. In case of a reduction option is specified, the area X will be, if necessary, extended to u2ModLength + 4 bytes.
2. If Square is without reduction, X is untouched. If Square is with reduction, X is filled with the final result.
3. If Square is without reduction, R is filled with the final result. If Square is with reduction, R is corrupted.

37.3.4.10.5 Available Options

The options are set by the u2Options input parameter, which is composed of:

- the mandatory Square operation option described in Table 37-31
- the mandatory CarryOperand option described in Table 37-32 and Table 37-33
- the facultative Modular Reduction option (see *Modular Reduction* from Related Links). If the Modular Reduction is not requested, this option is absent.

The u2Options number is calculated by an Inclusive OR of the options. Some Examples in C language are:

- Operation: Square only without carry and without Modular Reduction
  ```
  PUKCL(u2Options) = SET_MULTIPLIEROPTION(PUKCL_SQUARE_ONLY) |
  SET_CARRYOPTION(CARRY_NONE);
  ```
- Operation: Square with addition with Specific/CarryIn addition and with Fast Modular Reduction
  ```
  PUKCL(u2Options) = SET_MULTIPLIEROPTION(PUKCL_SQUARE_ADD) |
  SET_CARRYOPTION(ADD_CARRY) | PUKCL_REDMOD_REDUCTION |
  PUKCL_REDMOD_USING_FASTRED;
  ```

The following table lists all of the necessary parameters for the Square option. When the Addition or Subtraction option is not chosen it is not necessary to fill in the nu1ZBase parameter.

Table 37-31. Square Service Options

Option	Purpose	Required Parameters
SET_MULTIPLIEROPTION(PUKCL_ SQUARE_ONLY)	Perform R = X^2 + CarryOperand	nu1RBase, nu1ZBase, nu1XBase, u2XLength
SET_MULTIPLIEROPTION(PUKCL_ SQUARE_ADD)	Perform R = Z + X^2 + CarryOperand	nu1RBase, nu1ZBase, nu1XBase, u2XLength
SET_MULTIPLIEROPTION(PUKCL_ SQUARE_SUB)	Perform R = Z - (X^2 + CarryOperand)	nu1RBase, nu1ZBase, nu1Xlength, u2XLength

37.3.4.10.6 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;


// Gf2n and CarryIn shall be beforehand filled (with zero or one)
PUKCL(Specific).Gf2n = ...;
PUKCL(Specific).CarryIn = ...;

PUKCL(u2Option) =...;
// Depending on the option specified, not all fields must be filled
PUKCL_Fmult(nu1XBase) = <Base of the ram location of X>;
PUKCL_Fmult(u2XLength) = <Length of X>;
PUKCL_Fmult(nu1ZBase) = <Base of the ram location of Z>;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(Square,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            // The Squaring has been executed correctly
```

```
            ...
            }
else // Manage the error
```

37.3.4.10.7 Important Considerations for Modular Reduction of a Square Computation

Note:
Additional options are available through the use of a modular reduction to be executed at the end of this operation. Some important considerations have to be taken into account concerning the length of resulting operands to get a mathematically correct result.

The output of this operation is not obviously compatible with the modular reduction as it may be either smaller or bigger. In the case (most of the time) the result (pointed by nu1RBase) is smaller in size than "twice the modulus plus one word" by one word, a padding word must be added to zero. Otherwise, the reduced value will be taken considering the high order words (potentially uninitialized) as part of the number, thus resulting in getting a mathematically correct but unexpected result.

In the case that the result is greater than twice the modulus plus one word, the modular reduction feature has to be executed as a separate operation, using an Euclidean division.

37.3.4.10.8 Constraints

When the options only indicate a square, the constraints involving nu1ZBase are not checked. The following conditions must be avoided to ensure that the service works correctly:

- nu1XBase, nu1RBase or nu1ZBase are not aligned on 32-bit boundaries
- {nu1XBase, u2XLength}, {nu1ZBase, 2*u2XLength} or {nu1RBase, 2*u2XLength} are not in Crypto RAM
- u2XLength is either: < 4, > 0xffc or not a 32-bit length
- {nu1RBase, 2*u2XLength} overlaps {nu1XBase,u2XLength}
- {nu1RBase, 2*u2XLength} overlaps {nu1ZBase, 2*u2XLength} and nu1RBase >nu1ZBase

If a modular reduction is specified, the relevant parameters must be defined according to the chosen reduction and follow the description in Modular Reduction (see *Modular Reduction* from Related Links). Additional constraints to be respected and error codes are described in this section and in Table 37-49.

Multiplication with Accumulation or Subtraction

Where the options bits specify that either an Accumulation or a subtraction must be performed, this command performs the following operation:

$R = (Z \pm (X^2 + CarryOperand)) mod\ B^{2 * XLength}$

Table 37-32. Multiplication with Accumulation or Subtraction

Option AND CARRYOPTIONS	CarryOperand	Resulting Operation
SET_CARRYOPTION(ADD_CARRY)	CarryIn	$R = Z \pm (X^2 + CarryIn)$
SET_CARRYOPTION(SUB_CARRY)	- CarryIn	$R = Z \pm (X^2 - CarryIn)$
SET_CARRYOPTION(ADD_1_PLUS_CARRY)	1 + CarryIn	$R = Z \pm (X^2 + 1 + CarryIn)$
SET_CARRYOPTION(ADD_1_MINUS_CARRY)	1 - CarryIn	$R = Z \pm (X^2 + 1 - CarryIn)$
SET_CARRYOPTION(CARRY_NONE)	0	$R = Z \pm (X^2)$
SET_CARRYOPTION(ADD_1)	1	$R = Z \pm (X^2 + 1)$
SET_CARRYOPTION(SUB_1)	- 1	$R = Z \pm (X^2 - 1)$
SET_CARRYOPTION(ADD_2)	2	$R = Z \pm (X^2 + 2)$

37.3.4.10.9 Multiplication without Accumulation or Subtraction

Where the options bits specify that either an accumulation or a subtraction must be performed, this command performs the following operation:

$R = (X^2 + CarryOperand) mod\ B^{2 * XLength}$

Table 37-33. Square Service Carry Settings

Option AND CARRYOPTIONS	CarryOperand	Resulting Operation
SET_CARRYOPTION(ADD_CARRY)	CarryIn	$R = X^2 + CarryIn$
SET_CARRYOPTION(SUB_CARRY)	- CarryIn	$R = X^2 - CarryIn$
SET_CARRYOPTION(ADD_1_PLUS_CARRY)	1 + CarryIn	$R = X^2 + 1 + CarryIn$
SET_CARRYOPTION(ADD_1_MINUS_CARRY)	1 - CarryIn	$R = X^2 + 1 - CarryIn$
SET_CARRYOPTION(CARRY_NONE)	0	$R = X^2$
SET_CARRYOPTION(ADD_1)	1	$R = X^2 + 1$
SET_CARRYOPTION(SUB_1)	- 1	$R = X^2 - 1$
SET_CARRYOPTION(ADD_2)	2	$R = X^2 + 2$

37.3.4.10.10 Status Returned Values

Table 37-34. Square Service Return Codes

Returned status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly

37.3.4.11 Integral (Euclidean) Division

37.3.4.11.1 Purpose

The purpose of this service is to compute the Euclidean Division of two multiple precision numbers in GF(p) or polynomial in GF(2^n). The Numerator is divided by the Denominator giving the Quotient “Quo” and the Remainder “R”.

The following options are available:

- Work in the GF(2^n) field or in the standard integer arithmetic field GF(p)

37.3.4.11.2 How to Use the Service

37.3.4.11.3 Description

This service processes the calculus corresponding to:

$$Num = Mod \times Quo + R \quad with \quad 0 \leq R < Mod \quad and \quad Quo = \left[\frac{Num}{Mod}\right]$$

The Numerator is Num.

The Divisior (Modulus) is Mod.

The Quotient is Quo.

The Remainder is R.

The Inputs are, the Numerator Num, and the Denominator Mod. The service calculates the Quotient and the Remainder. The Remainder overwrites the Numerator and is copied to the R area.

If the parameter nu1QuoBase equals zero, the Quotient is not stored in memory.

If nu1QuoBase is different from zero, the Quotient length is (<Numerator Length> - <Denominator Length>) + 4 bytes.

In this computation, the following areas need to be provided:

- Num (pointed by {nu1NumBase,u2NumLength}) filled with the Numerator (with MSB word to zero).
- Mod (pointed by {nu1ModBase,u2ModLength}) filled with the Denominator.
- Workspace (pointed by {nu1CnsBase,64 or68}).

- Quo (pointed by {nu1QuoBase,u2NumLength - u2ModLength + 4}) to contain calculated Quotient.
  - When the quotient is not needed, the nu1QuoBase pointer can be provided as NULL. In that case, only the remainder will be provided as a result.
- R (pointed by {nu1RBase,u2ModLength}) to contain the calculated Remainder.

The service name for this operation is `Div`.

37.3.4.11.4 Parameters Definition

Table 37-35. Div Service Parameters

Parameter	Type	Dir.	Location	Data Length	Before Executing the Service	After Executing the Service
Specific/Gf2n	Bit	I	–	–	GF(2n) Bit	–
nu1NumBase	nu1	I	Crypto RAM	u2NumLength	Base of Num Filled with the Numerator	Base of Num Filled with the Remainder
u2NumLength	u2	I	–	–	Length of the Numerator	Length of the Numerator
nu1ModBase	nu1	I	Crypto RAM	u2ModLengt	Base of the Divisor	Base of the Divisor untouched
u2ModLength	u2	I	–	–	Length of the Divisor	Length of the Divisor
nu1QuoBase (see **Note 1**)	nu1	I	Crypto RAM	u2NumLength - u2ModLength + 4	Base of the Quotient	Base of the Quotient
nu1WorkSpace	nu1	I	Crypto RAM	GF(p): 64 GF(2n): 68	Base of the WorkSpace	Base of the WorkSpace corrupted
nu1RBase (see **Note 2**)	nu1	I	Crypto RAM	u2ModLength	Base of the Remainder	Base of the Remainder

Notes:

1. If the quotient is not needed, set nu1QuoBase to zero and the quotient will not be written to memory. If the quotient is needed, set the nu1QuoBase to the beginning of an area of size (u2NumLength - u2ModLength + 4) to write the whole quotient.
2. The Remainder is present in the area {nu1NumBase, u2NumLength} at the end of the calculus. The nu1RBase parameter makes it possible to copy this result in the other area {nu1RBase, u2ModLength}, if this copy is not needed, set nu1RBase to the same value as nu1NumBase and the copy will not be done.

Note: The parameter Num must have its most significant 32-bit word cleared to zero. The length u2NumLength is the length of Num including this zero word.
One additional word is used on the LSB side of the Num parameter, this word is restored at the end of the calculus. As a consequence the parameter nu1NumBase must never been at the beginning of the Crypto RAM, i.e., ensure that nu1NumBase ≥ <Crypto RAM Base> + 4 bytes.

One additional word is used on the MSB side of the Num parameter, this word is not corrupted. As a consequence the Area {nu1NumBase, u2NumLength} must not be at the end of the Crypto RAM, i.e., en sure that nu1NumBase+u2NumLength ≤ <Crypto RAM End> - 4.

u2ModLength must be the true length of the Modulus, i.e., the MSB word of the area {nu1ModBase, u2ModLength} must be different from zero.

The minimum value for u2ModLength is 8 bytes, so the significant length of Num must be at least 8 bytes. To divide by a 32-bit value, the divider and numerator shall be multiplied by 232. The resulting remainder will have to be divided by 2^{32}, the quotient will be exact.

37.3.4.11.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;
```

```
// Fill all the fields
// In that case, the quotient will be computed
// If it was not needed, set nu1QuoBase to NULL
PUKCL_Div(nu1NumBase) = <Base of the ram location of Num>;
PUKCL_Div(nu1ModBase) = <Base of the ram location of Mod>;
PUKCL_Div(nu1QuoBase) = <Base of the ram location of Quo>;
PUKCL_Div(nu1WorkSpace) = <Base of the workspace>;
PUKCL_Div(nu1RBase) = <Base of the ram location of R>;
PUKCL_Div(u2NumLength) = <Length of Num>;
PUKCL_Div(u2ModLength) = <Length of Mod>;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(Div,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            // The Division has been executed correctly
            ...
            }
else // Manage the error
```

37.3.4.11.6 Constraints

The following conditions must be avoided to ensure the service works correctly:

- nu1ModBase, nu1RBase, nu1QuoBase, nu1WorkSpace or nu1NumBase are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength}, {nu1RBase, u2ModLength}, {nu1WorkSpace, 64} or{nu1NumBase, u2NumLength} are not in Crypto RAM
- u2ModLength, u2NumLength is either: < 4, > 0xffc or not a 32-bit length
- One or more overlaps exist between two of the areas: {nu1ModBase,u2ModLength},{nu1RBase, u2ModLength} {nu1NumBase, u2NumLength}(1) or {nu1WorkSpace,64}
- If nu1QuoBase is different from zero and: {nu1QuoBase, u2NumLength - u2ModLength + 4} are not in Crypto RAM
- If nu1QuoBase is different from zero and one or more overlaps exist between two of the areas: {nu1QuoBase, u2NumLength - u2ModLength + 4}, {nu1ModBase, u2ModLength}, {nu1RBase, u2ModLength}, {nu1NumBase, u2NumLength} or {nu1WorkSpace, 64}

Overlaps between {nu1RBase, u2ModLength} and {nu1NumBase, u2NumLength} are forbidden, but the equality between nu1RBase and nu1NumBase is authorized

37.3.4.11.7 Status Returned Values

Table 37-36. Div Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly.
PUKCL_DIVISION_BY_ZERO	Severe	The operation was not performed because the Denominator value is zero.

37.3.4.12 GCD, Modular Inverse

37.3.4.12.1 Purpose

The purpose of this command is to compute the Greatest Common Divisor (GCD) and the Modular Inverse. The algorithm used is the Extended Euclidean Algorithm for the GCD.

This command accepts as input two multiple precision numbers in GF(p) or two polynomials in GF(2^n) X and Y and computes their GCD (D), if D equals one, the command also supplies the inverse of X modulo Y.

The available options are as follows:

- Work in the GF(2^n) field or in the standard integer arithmetic field GF(p)

37.3.4.12.2 How to Use the Service

37.3.4.12.3 Description

This command calculates:

$D = GCD(X,Y)$.

and parameter A in the Bezout equation:

$A \times X + B \times Y = D$.

The first input, or input to inverse is X.

The second input, or modulus is Y.

The GCD is output in D.

The modular inverse if X and Y are co-primes is output A:

$A = X^{-1} mod(Y)$

The command calculates the GCD and the value A. The value A is the multiplicative inverse of X, only if X and Y are co-prime. As a supplemental result, Z is given back, being the quotient of Y divided by D only if D is different from zero:

$$Z = \left[\frac{Y}{D}\right]$$

At the end of the command: X is overwritten by D.

Y is cleared.

The value of A is calculated and stored.

The value of Z is calculated and stored if D is different from zero.

The service name for this operation is `GCD`.

In this computation, the following areas have to be provided:

- X (pointed by {nu1XBase,u2Length}) filled with X (with MSB word to zero)
- Y (pointed by {nu1YBase,u2Length}) filled with Y (with MSB word to zero)
- A (pointed by {nu1ABase,u2Length}) to contain calculated A
- Z (pointed by {nu1ZBase,u2Length}) to contain calculated Z
- The workspace (pointed by {nu1WorkSpace,32})

37.3.4.12.4 Parameters Definition

Table 37-37. GCD Service Parameters

Parameter	Type	Dir.	Location	Data Length	Before Executing the Service	After Executing the Service
Specific/Gf2n	Bit	I	–	–	GF(2n) Bit	–
nu1XBase	nu1	I	Crypto RAM	u2Length	Base of X Number X	Base of X Filled with the GCD D
u2Length	u2	I	–	–	Length of the Areas X, Y, A, Z	Length of the Areas X, Y, A, Z
nu1YBase	nu1	I	Crypto RAM	u2Length	Base of Y Number Y	Base of Y Cleared area
nu1ABase	nu1	I	Crypto RAM	u2Length	Base of A	Base of A Filled with the result
nu1ZBase	nu1	I	Crypto RAM	u2Length + 4 (see **Note 1**)	Base of Z	Base of Z Filled with the result
nu1WorkSpace	nu1	I	Crypto RAM	32 bytes	Base of the workspace	Base of the workspace corrupted

Note:

1. The additional word is 4 zero bytes.

The parameters X and Y must have their most significant 32-bit word cleared to zero. The length u2Length is the length of the longer of the parameters X and Y including this zero word.

To clarify here is an example:

- X is an 8 bytes number.
- Y is a 12 bytes number.

This example is processed this way before the use of the GCD service:

- The longer number is Y so its length is taken and increased by 4 bytes for the 32-bit word cleared to zero, this gives u2Length = 16 bytes. Therefore, X, Y, A and Z areas have a length of 16 bytes.
- Y is padded with 4 bytes cleared to zero on the MSB side and the u2Length = 16 bytes are written in memory (LSB first).
- X is padded with 8 bytes cleared to zero on the MSB side and the u2Length = 16 bytes are written in memory (LSB first).
- The areas A and Z are mapped in memory with a size of u2Length = 16 bytes.
- The workspace is mapped in memory with its constant size of 32 bytes

37.3.4.12.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;
// Fill all the fields
PUKCL(u2Option) = 0;
PUKCL_GCD(nu1XBase) = <Base of the ram location of X>;
PUKCL_GCD(nu1YBase) = <Base of the ram location of Y>;
PUKCL_GCD(nu1ABase) = <Base of the ram location of A>;
PUKCL_GCD(nu1ZBase) = <Base of the ram location of Z>;
PUKCL_GCD(nu1WorkSpace) = <Base of the workspace>;
PUKCL_GCD(u2Length) = <Length of X, Y, A and Z>;
// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(GCD, pvPUKCLParam);
if (PUKCL_Param.Status == PUKCL_OK)
                {
                // The GCD has been executed correctly
                ...
                }
else // Manage the error
```

37.3.4.12.6 Constraints

The following conditions must be avoided to ensure that the service works correctly:

- nu1XBase, nu1YBase, nu1ABase or nu1ZBase are not aligned on 32-bit boundaries
- {nu1XBase, u2Length}, {nu1YBase, u2Length}, {nu1ABase, u2Length} or {nu1ZBase, u2Length} are not in Crypto RAM
- u2Length is either: < 4, > 0xffc or not a 32-bit length
- {nu1XBase, u2Length} overlaps {nu1YBase, u2Length} or {nu1XBase, u2Length} overlaps {nu1ABase, u2Length} or {nu1XBase, u2Length} overlaps {nu1ZBase, u2Length} or {nu1YBase, u2Length}overlaps

{nu1ABase, u2Length} or {nu1YBase, u2Length} overlaps {nu1ZBase, u2Length} or {nu1ABase, u2Length} overlaps {nu1ZBase, u2Length}

37.3.4.12.7 Status Returned Values

Table 37-38. GCD Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly

37.3.4.13 Get Random Number

37.3.4.13.1 Purpose

The purpose of this command is to provide the user with a source of entropy. The options available for this service are:

- Generation of random numbers from a Hardware Random Number Generator (TRNG).
- Generation of random numbers from a Deterministic Random Number Generator (DRNG).

Important:
When using this service, be sure to strictly follow the directives given for the RNG on the chip you use (particularly initialization, seeding) and compulsorily start the RNG. If the directives require not to use this service, follow them and use the proposed method to get random numbers.

This service only has the option to get random numbers and does not seed, initialize or start the RNG. Other options are reserved for future use.

Neither continuous testing nor entropy testing is included in this service. If this is needed (FIPS 140, ZKA, ...), this service must not be used and the users develops their own command.

The DRNG is compatible with both ANSI X9.31 and FIPS 186-2 standards (see the important note below). The DRNG is designed according to:

- The algorithm described in the document ANSI Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA) X9.31 dated September 9, 1998.
- The Change recommendation for ANSI X9.0 - 1995 (Part 1) and ANSI X9.31 -1998:

The algorithm B.2.1 Algorithm for computing m Values of x is the one applied in the Toolbox 3 X9.31 DRNG. The DRNG is compatible with:

- The DRNG is described in the document NIST Digital Signature Standard (DSS) FIPS Pub 186-2 January 27, 2000 Appendix 3.1
- The FIPS 186-2 Change Notice 1 dated October 5, 2001 modifies this algorithm.

Important: To apply the FIPS 186-2 algorithm, the parameters XSeed[0] and XSeed[1] must be set to the same value.

37.3.4.13.2 How to Use the Service

37.3.4.13.3 Description

This service has four possible options described in Table 37-41. Two of these options are reserved for future use. This service performs the following operations:

- Generation of a random number from the Hardware RNG
- Generation of a random number from the Deterministic RNG

Generation of a Random Number from the Hardware RNG

This service, activated with the option PUKCL_RNG_GET, makes it possible to get a random number R from the Hardware RNG:

R = HardwareRandomGenerate()

In the Generation of random from the RNG service, the following parameters need to be provided:

- R the generated number area (pointed by{nu1RBase,u2RLength})

37.3.4.13.4 Generation of a Random Number from the Deterministic RNG

This service, activated with the option PUKCL_RNG_X931_GET, makes it possible to get a random number R from the Deterministic Random Number Generator with input parameters the Key XKey and the Seed XSeed:

(XKey, R) = DeterministicRandomGenerateFromSeed (XKey, XSeed, Q)

In the generation of a random number from the Deterministic RNG service, the following parameters need to be provided:

- XKey the input and output Key (pointed by {nu1XKeyBase,u2XKeyLength})
- XSeed the input Seed (pointed by {nu1XseedBase,u2XKeyLength})
- Q the prime number (pointed by {nu1QBase, 20bytes})
- R the generated number area (pointed by {nu1RBase, 20bytes})

37.3.4.13.5 Hardware RNG Parameters Definition

The parameters for the generation of random from the Hardware RNG are described in the following table. This service can easily be accessed through the use of the `PUKCL_Rng()` and `PUKCL()` macros.

Table 37-39. RNG Service Hardware Generated Parameters

Parameter	Type	Dir.	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	–	–	Option (see Table 37-41)	Option (see Table 37-41)
nu1RBase	nu1	I	Crypto RAM or Device RAM	u2RLength	Base of R	Base of R filled with random values depending on the option
u2RLength	u2	I	–	–	Length of R	Length of R

37.3.4.13.6 Deterministic RNG Parameters Definition

The parameters for the generation of random from the Deterministic RNG are described in the following table. This service can easily be accessed through the use of the `PUKCL_Rng()` and `PUKCL()` macros.

Table 37-40. RNG Service Deterministic Generated Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	–	–	Option (see Table 37-41)	Option (see Table 37-41)
nu1XKeyBase	nu1	I/O	Crypto RAM	u2XKeyLength	Base of XKey	Base of XKey filled with the resulting XKey
nu1Workspace	nu1	NA	Crypto RAM	64 bytes	Base of the workspace	Base of the workspace corrupted
nu1Workspace2(1)	nu1	NA	Crypto RAM	2*u1XKeyLength + 4	Base of the workspace 2	Base of the workspace corrupted
nu1XSeedBase	nu1	I/O	Crypto RAM	max (2*u2XKeyLength, 44 bytes)	Base of the values XSeed[0] and XSeed[1]	Base of XSeed filled with the result on 20 bytes

.....continued						
Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2XKeyLength	u2	I	–	–	Length of XKey, Xseed[0] and Xseed[1]	Length of XKey, Xseed[0] and Xseed[1]
nu1QBase	nu1	I	Crypto RAM	20 bytes	Base of Q	Base of Q
nu1RBase	nu1	I	Crypto RAM	u2RLength	Base of R	Base of R filled with the result on 20 bytes

Note:

1. The nu1 Workspace2 must be a multiple of 256.

37.3.4.13.7 Options

The option is set by the u2Options input parameter that must take one of the values listed in the following table.
Note: The values, OPTION_RNG_SEED and OPTION_RNG_GETSEED, are reserved for future use.

Table 37-41. RNG Service Options

Option	Purpose	Required Parameters
PUKCL_RNG_SEED	Reserved	Reserved
PUKCL_RNG_GET	Generation of a random number from the RNG	nu1RBase, u2RLength
PUKCL_RNG_X931_GET	Generation of a random number from the Deterministic RNG	nu1XKeyBase, nu1Workspace, nu1XSeedBase, u2XKeyLength, nu1QBase, nu1RBase
PUKCL_RNG_GETSEED	Reserved	Reserved

37.3.4.13.8 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
// ! following the directives given for the RNG on the chip

PUKCL(u2Option) =...;

// Initializing parameters
PUKCL_Rng(nu1RBase) = <Base of the ram location to store the rng>;
PUKCL_Rng(u2RLength) = <Length of the rng to get>;

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(Rng,pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            // The RNG generation has been executed correctly
            ...
            }
else // Manage the error
```

37.3.4.13.9 Constraints

Random Number Generation

The following conditions must be avoided to ensure that the service works correctly:

- {nu1RBase,u2RLength} not in RAM
- {nu1RBase,u2RLength} not accessible or authorized for writing

Deterministic Random Number Generation

The length of the parameter nu1XSeedbase is: XSeedLength = max(2*u2XKeyLength, 44 bytes) The `max()` macro takes a maximum of two values.

The following conditions must be avoided to ensure that the service works correctly:

- nu1XKeyBase,nu1Workspace, nu1Workspace2, nu1XSeedBase, nu1QBase, nu1RBase are not aligned on 32-bit boundaries
- {nu1XKeyBase, u2XKeyLength}, {nu1Workspace, 64 bytes}, {nu1Workspace2, 2*u1XKeyLength +4}, {nu1XSeedBase, XSeedLength}, {nu1QBase, 24 bytes} or {nu1RBase, 20 bytes} are not in PUKCC RAM
- u2XKeyLength is either: < 20, > 64 or not a 32-bit length
- nu1Workspace2 not multiple of 256.
- Overlaps exist between two or more of the areas: {nu1XKeyBase, u2XKeyLength}, {nu1Workspace,64 bytes}, {nu1XSeedBase, XSeedLength}, {nu1QBase, 24 bytes} or {nu1RBase, 20 bytes}
  The area {nu1RBase, 20} can overlap with {nu1Workspace, 64 bytes} or {nu1QBas, 24 bytes}. The pointer nu1RBase can equal the pointer nu1XSeedBase.

37.3.4.13.10 Status Returned Values

Table 37-42. RNG Service Return Codes

Returned status	Importance	Meaning
PUKCL_OK	Information	Service functioned correctly

37.3.5 Modular Arithmetic Services

This section provides a complete description of the modular arithmetic services, which consists of two sets:

- Modular reductions, which can be used as stand alone operations, or used as a final step of most arithmetic operations (full and small multiplications, squaring).
- Modular operations, which include modular exponentiations (with or without using the CRT) and a probabilistic prime number generation.

These operations work on general data so the modulus has no special form. The modular services are available through:

- a Fast form (may return a congruence of the result, with a high probability to have a Normalized result)
- a Normalized form (returns the exact result, strictly lower than the modulus)
- a Euclidean form (returns the exact result, strictly lower than the modulus)

The following table describes the modes of the modular reduction with the hypothesis:

- In GF(p): The modulus is N with length NLength in bytes
- In GF(2^n): The modulus is P[X] with length NLength in bytes

For the exact calculus of NLength see below.

Table 37-43. Modular Reduction Modes

Modular Reduction Form	Input Dynamic	Result Dynamic	Comments
Fast	GF(p): 0 ≤ Input < (N^2) * (2^{32}) GF(2n): Input < (($P[x])^2$) * (X^{32})	GF(p): 0 ≤ Res < N * 4 GF(2^n): Res < P[X] * (X^2)	The fastest reduction available, needs a precomputed constant.
Normalized	InputLength < NLength + 4 bytes	GF(p): 0 ≤ Res < N GF(2^n): Res < P[X]	The correction step does not runs in constant time. Needs a precomputed constant. The Normalize function cannot be applied to the product of two numbers of length u2NLength.

.....continued			
Modular Reduction Form	Input Dynamic	Result Dynamic	Comments
Using Euclidean division	InputLength < 2 * NLength + 4 bytes	GF(p): 0 ≤ Res < N GF(2^n): Res < P[X]	Does not need any precomputed constant.

To be able to use these modular reduction services (except the Euclidean division), first the implementer shall call the setup service, providing the modulus as well as one free memory space for the constant (this constant is used to speed up the modular reduction). In most commands (except the modular exponentiation), the quotient is stored in the high order bytes of the number to be reduced, using only eight bytes more than the maximum size of the number to be reduced.

The following rules must be respected to ensure the modular reduction services function correctly:

- The numbers to be reduced can have any significant length, given the fact it CANNOT BE GREATER than 2*u2ModLength + 4 bytes.
- The modulus SHALL ALWAYS HAVE a significant length of <u2ModLength> bytes. The modulus must be provided as a <u2ModLength + 4> bytes long number, padded on the most significant side with a 32-bit word cleared to zero. Not respecting this rule leads to unexpected and wrong results from the modular reduction.
- The normalization operation ALWAYS performs a modular reduction step, and will therefore have the same memory usage as this one.
- The very first operation before any modular operation SHALL BE a modular setup.

37.3.5.1 Modular Reduction

Related Links

37.3.3.4. Aligned Significant Length

37.3.5.1.1 Purpose

This service is used to perform the various steps necessary to perform a modular reduction and accepts as input numbers in GF(p) or polynomials in GF(2^n) .

The available options for this service are:

- Work in the GF(2^n) or in the standard integer arithmetic field GF(p)
- Operation is the generation of the reduction constant.
- Operation is a Modular Reduction.
- Operation is a Normalization.

37.3.5.1.2 How to Use the Service

37.3.5.1.3 Description

This service performs one of the following operations:

- Setup of the Fast or Normalize functions: generation of the reduction constant
- Fast Modular Reduction
- Big Modular Reduction (using Euclide's division)
- Normalization

The service name for this operation is `RedMod`.

37.3.5.1.4 Modular Reduction Setup

This service calculates the constant Cns, computed from the modulus and used to speed up the modular reduction:

Cns = SetupConstant(N)

This service must be processed before the use of the Fast or Normalize functions. In the Setup computations, the following data must be provided:

- N the modulus (pointed by {nu1ModBase,u2ModLength +4}).
- Cns the Setup Constant Result (pointed by {nu1CnsBase,u2ModLength +12}).
- X used as a workspace (pointed by {nu1XBase,2 * u2ModLength + 8}) (include the supplementary bytes; see **Note 2** in Table 37-44
- R used as a workspace (pointed by {nu1RBase,64 or 68bytes}).
- u2ModLength is the Aligned Significant Length of the modulus and is not the byte Significant Length (see *Aligned Significant Length* from Related Links).

37.3.5.1.5 Fast Reductions and Normalization

These commands calculate an approximated or exact Modular Reduction, that is, the result may be greater than the modulus, but is always congruent to the true result.

Important: Before using these functions, ensure that the constant Cns has been calculated with the setup for the Modular Reduction service.

Input and Result significant values verify:

- For the Fast Modular Reduction:

$0 \leq X < N^2 \times 2^{32}$

$R = Xmod(N) + k \times N \quad with \quad 0 \leq k \leq 4$

- For the Normalize:

$XLength < (NLength + 4)bytes \;\; R$

$= Xmod(N)$

In these Fast Modular Reduction and Normalize computations, the following data have to be provided:

- X (pointed by {nu1XBase,2 * u2ModLength +8})
  - The Normalize computation accept as entry a value whose length is lower or equal to u2ModLength + 4 (that is, for example, a value yet reduced but not normalized.). The u2ModLength + 4 MSB bytes are cleared at the beginning of the computation.
  - in case of Fast RedMod computations, the value X mayverify: X < (N^2) *(2^{32}).
  - include the supplementary bytes; see **Note 3** in 37.3.5.1.8. Fast Modular Reductions Service Parameters Definition.
- R (pointed by {nu1RBase,u2Modlength +4})
- N (pointed by {nu1ModBase,u2ModLength +4})
- Cns (pointed by {nu1CnsBase,u2ModLength +12})
- u2ModLength is the Aligned Significant Length of the modulus and is not the byte Significant Length (see *Aligned Significant Length* from Related Links).

The Fast Modular Reduction is able to reduce inputs up to <2*u2ModLength + 4> bytes. The input can come from a multiplication of 2 <u2ModLength + 4> bytes numbers. The input X is considered as a <2*u2ModLength + 8> bytes number.

Important: Additionally the Fast Reduction and Normalize functions need supplemental bytes located on the MSB side of the number to be reduced but these bytes are restored at the end of the operation and are therefore unchanged. However, these bytes are to be taken into account when the mapping is created, and could lead to unexpected results if overlapping with other area used by the function.

The Fast Modular Reduction returns a <u2ModLength + 4> bytes number, but this number is in fact a <u2ModLength + 2> significant bytes number. When using the Fast Modular Reduction, the two MSB bytes of the <u2ModLength + 2> can have a maximum of two lsb bits set (depending on the reduced number and the modulo).

The Normalize computation accepts as entry a resulting value of Fast Modular Reduction and computes an exact result. It can not be applied to the result of the product of two numbers of size NLength: a Fast Modular Reduction must be applied before.

For the Normalize computation, the X value is limited by the preceding formula but the area in memory is bigger as described in 37.3.5.1.8. Fast Modular Reductions Service Parameters Definition.

As input, the Normalize sub-service only accept values, which length is lower or equal to u2ModLength + 4. The Most Significant u2ModLength + 4 bytes are firstly cleared by this service.

37.3.5.1.6 Big Modular Reduction Using Euclide's Division

This command calculates:

$$XLength < (2 \times NLength + 4)bytes \quad R$$

$$= Xmod(N)$$

In this Big Modular Reduction computations, the following data must be provided:

- X (pointed by {nu1XBase,2 * u2ModLength + 8}) (include the supplementary bytes; see **Note 1** in Table 37-46)
- R (pointed by {nu1RBase,u2Modlength +4})
- N (pointed by {nu1ModBase,u2ModLength +4})
- u2ModLength is the Aligned Significant Length of the modulus and is not the byte Significant Length (see *Aligned Significant Length* from Related Links).
- Workspace (pointed by {nu1CnsBase,64 or 68}).

37.3.5.1.7 Modular Reductions Service Parameters Definition

Table 37-44. RedMod Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	–	–	Options (see below)	Options (see below)
Specific/CarryIn	Bits	I	–	–	Must be set to zero.	–
Specific/Gf2n	Bit	I	–	–	GF(2^n) Bit	–
Specific/CarryOut Zero Violation	Bits	I	–	–	–	Carry Out, Zero Bit and Violation Bit filled according to the result
nu1ModBase(1)	nu1	I	Crypto RAM	u2ModLength + 4	Base of N	Base of N untouched
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 12	Base of Cns	Base of Cns filled with the Setup Constant
u2ModLength	u2	I	–	–	Length of N	Length of N

.....continued						
Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1RBase	nu1	I	Crypto RAM	GF(p): 64 bytes GF(2n): 68 bytes	Base of R as a workspace	Base of R workspace corrupted
nu1XBase(2)	nu1	I	Crypto RAM	2*u2ModLength + 8	Base of X as a workspace	Base of X workspace corrupted

Notes:

1. The Modulus is to be given as a u2ModLength Aligned Significant Length Bytes however, it has to be provided as a u2ModLength + 4 bytes long number, having the four high-order bytes set to zero.
2. Before the X (pointed by {nu1XBase,2 * u2ModLength + 8}) LSB bytes, four supplementary bytes will be saved/restored. Other four supplementary bytes will also be saved/restored after the X MSB bytes. All these supplementary bytes may be entirely in the Crypto RAM (therefore, do not place the X area too near the end of the Crypto RAM) and shall not overlap with other area used by the service.

37.3.5.1.8 Fast Modular Reductions Service Parameters Definition

Table 37-45. Fast RedMode and Normalize Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	–	–	Options (see below)	Options (see below)
Specific/CarryIn	Bits	I	–	–	Must be set to zero.	–
Specific/Gf2n	Bit	I	–	–	GF(2n) Bit	–
Specific/CarryOut Zero Violation	Bits	I	–	–	–	Carry Out, Zero Bit and Violation Bit filled according to the result
nu1ModBase(1)	nu1	I	Crypto RAM	u2ModLength + 4	Base of N	Base of N untouched
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 12	Base of Cns	Base of Cns untouched
u2ModLength	u2	I	–	–	Length of N	Length of N
nu1RBase(2)	nu1	I	Crypto RAM	u2ModLength + 4	Base of R	Base of R filled with the result
nu1XBase(3)	nu1	I	Crypto RAM	2*u2ModLength + 8	Base of X the number to reduce	Base of X corrupted

Notes:

1. The Modulus is to be given as a u2ModLength Aligned Significant Length Bytes however, it has to be provided as a u2ModLength + 4 bytes long number, having the four high-order bytes set to zero.
2. To make profitable the space memory, it is possible to set nu1RBase exactly equal to nu1XBase.
3. After the X (pointed by {nu1XBase,2 * u2ModLength + 8}) MSB bytes, supplementary bytes will be saved/restored (8 bytes in case of Fast RedMod, otherwise; 12 bytes). These supplementary bytes may be entirely in the Crypto RAM (therefore, do not place the X area too near the end of the Crypto RAM) and shall not overlap with other area used by the service.

37.3.5.1.9 Big Modular Reduction Parameters Definition

Table 37-46. Big RedMod Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	–	–	Options (see below)	Options (see below)

.....continued

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
Specific/CarryIn	Bits	I	–	–	Must be set to zero	–
Specific/Gf2n	Bit	I	–	–	GF(2n) Bit	–
Specific/CarryOut Zero Violation	Bits	I	–	–	–	Carry Out, Zero Bit and Violation Bit filled according to the result
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of N	Base of N untouched
nu1CnsBase	nu1	I	Crypto RAM	GF(p): 64 bytes GF(2n): 68 bytes	Base of Cns as a workspace	Base of Cns corrupted
u2ModLength	u2	I	–	–	Length of N	Length of N
nu1RBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of R	Base of R filled with the result
nu1XBase(1)	nu1	I	Crypto RAM	2*u2ModLength + 8	Base of X the number to reduce	Base of X filled with the result

Note:

1. Before the X (pointed by {nu1XBase,2 * u2ModLength + 8}) LSB bytes, four supplementary bytes will be saved/restored. Other four supplementary bytes will also be saved/restored after the X MSB bytes. All of these supplementary bytes may be entirely in the Crypto RAM (therefore, do not place the X area too near the end of the Crypto RAM) and shall not overlap with other area used by the service.

37.3.5.1.10 Options

The options are set by the u2Options input parameter, which is composed of:

- the mandatory Operation Option described in Table 37-47
- if the Operation Option is PUKCL_REDMOD_REDUCTION, the Modular Reduction Sub-Option described in Table 37-48

The u2Options number is calculated by an Inclusive OR of the options. Some Examples in C language are:

- Operation: Setup for the ModularReductions.
  ```
  PUKCL(u2Options) = PUKCL_ REDMOD_SETUP;
  ```
- Operation: Fast ModularReduction.
  ```
  PUKCL(u2Options) = PUKCL_REDMOD_REDUCTION | PUKCL_REDMOD_USING_FASTRED;
  ```

For this command three exclusive options can be specified. The following table lists the operations that can be performed.

Table 37-47. RedMod Service Options

Option	Purpose	Required Parameters
PUKCL_REDMOD_SETUP	Perform the Cns value computation	nu1ModBase, u2ModLength, nu1CnsBase, nu1XBase
PUKCL_REDMOD_REDUCTION	Perform R ≡ X Mod N, see sub-option for details	nu1ModBase, u2ModLength, nu1CndBase, nu1XBase, nu1RBase
PUKCL_REDMOD_NORMALIZE	Perform R = X Mod N	nu1ModBase, u2ModLength, nu1CndBase, nu1XBase, nu1RBase

When selecting the PUKCL_REDMOD_REDUCTION option, one of the two sub-options listed in the following table must be selected.

Table 37-48. RedMode Service Options with PUKCL_RED_MOD_REDUCTION

Option	Purpose	Required Parameters
PUKCL_REDMOD _USING_DIVISION	Perform R = X Mod N	nu1ModBase, u2ModLength, nu1CndBase, nu1XBase
PUKCL_REDMOD _USING_FASTRED	Perform R ≡ X Mod N The entropy is minimized (~2 bits)	nu1ModBase, u2ModLength, nu1CndBase, nu1XBase, nu1RBase

37.3.5.1.11 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

PUKCL(Specific).CarryIn = 0;
PUKCL(Specific).GF2n = ...;

PUKCL(u2Option) =...;

// Depending on the option specified, not all fields must be filled
PUKCL_RedMod(nu1ModBase) = <Base of the ram location of N>;
PUKCL_RedMod(u2ModLength) = <Length of N>;
PUKCL_RedMod(nu1CnsBase) = <Base of the ram location of Cns>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(RedMod,pvPUKCLParam);
if (PUKCL_Param.Status == PUKCL_OK)
            {
            // operation has correctly been performed
            ...
            }
else // Manage the error
```

37.3.5.1.12 Constraints

Depending on the options chosen the lengths of the R area and Cns area differ:

- For the Setup:
  - RLength = 64bytes
  - CnsLength = u2ModLength +12
- For the Fast Reduction and Normalize:
  - RLength = u2ModLength +4
  - CnsLength = u2ModLength +8
- For the BigRedMod:
  - RLength = u2ModLength +4
  - CnsLength =64

The following combinations of input values must be avoided in the case of a modular reduction ‘alone’, meaning that it has not been requested as an option of any other command:

- nu1ModBase, nu1CnsBase, nu1RBase, nu1XBase are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2CnsLength}, {nu1XBase, 2*u2XLength + 8 + s} or {nu1RBase, u2RLength} are not in Crypto RAM
- u2ModLength is either: < 4, > 0xffc or not a 32-bit length
- Overlaps exist between two or more of the areas: {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2CnsLength}, {nu1XBase, 2*u2XLength + 8 + s} or {nu1RBase, u2RLength}

Note: Overlaps between {nu1RBase, RLength} and {nu1XBase, 2*u2XLength + 8} are forbidden; but if the operation is the Fast, Normalized or Big Modular Reduction, the equality between nu1RBase and nu1XBase is authorized.

37.3.5.1.13 Status Returned Values

Table 37-49. RedMod Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly
PUKCL_DIVISION_BY_ZERO	Severe	When computing an Euclidean division, the Modulus was found to be zero. This occurs ONLY when either reducing using an Euclidean division or computing the reduction constant usable for a Fast or Normalize Reduction.
PUKCL_UNEXPLOITABLE_OPTIONS	Severe	A bad combination of options has been detected.
PUKCL_MALFORMED_MODULUS	Severe	The Msw of the modulus is not zero.

37.3.5.2 Modular Exponentiation (Without CRT)

37.3.5.2.1 Purpose

This service is used to perform the Modular Exponentiation computation. This service processes integers in GF(p) only.

The options available for this service are:

- Fast implementation
- Regular implementation
- Exponent is located in Crypto RAM or not in Crypto RAM
- Exponent window size

37.3.5.2.2 How to Use the Service

37.3.5.2.3 Description

Important: Before using these functions, ensure that the constant Cns has been calculated with the Setup of the Modular Reductions service.

This service processes the following operation:

The service name for this operation is `ExpMod`.

$R = X^{Exp} mod(N)$

In this computation, the following parameters need to be provided:

- X: input number (pointed by {nu1XBase,u2ModLength +16})
- N: modulus (pointed by {nu1ModBase,u2ModLength +4}).
- Exp: exponent (pointed by {pfu1ExpBase,u2ExpLength +4})
- Cns: Fast Modular Constant (pointed by {nu1CnsBase,u2ModLength +8})
- Precomp: precomputation workspace (pointed by{nu1PrecompBase,PrecompLen})
- Blinding: exponent blinding value (provided inu1Blinding)

The length PrecompLen depends on the lengths and options chosen; its calculus is detailed in Options below.

Note: The minimum value for u2ModLength is 12 bytes. Therefore, the significant length of N must be at least three 32-bit words.

37.3.5.2.4 Parameters Definition

Table 37-50. ExpMod Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	–	–	Options (see below)	Options (see below)
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of N	Base of N untouched
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns untouched
u2ModLength	u2	I	–	–	Length of N	Length of N
nu1XBase(1)	nu1	I	Crypto RAM	u2ModLength + 16	Base of X	Base of X Filled with the result
nu1PrecompBase	nu1	I	Crypto RAM	See below	Base of Precomp as a workspace	Base of Precomp workspace corrupted
pfu1ExpBase(2)	pfu1	I	Any place(3)	u2ExpLength + 4	Base of the Exponent	Base of the Exponent untouched
u2ExpLength(4)	u2	I	–	–	Significant length of Exponent	Significant length of Exponent
u1Blinding(5)	u1	I	–	–	Exponent unblinding value	Exponent unblinding value untouched

Notes:

1. This zone contains the number to be exponentiated (u2ModLength bytes) and is used during the computations as a workspace (four 32-bit words longer than the number to be exponentiated). At the end of the computation, it contains the correct result of the operation.
2. The exponent must be given with a supplemental word on the LSB side (low addresses). This word shall be set to zero.
3. If the PUKCL_EXPMOD_EXPINPUKCCRAM option is not set, the location of the exponent MUST NOT be the Crypto RAM, even partially.
4. The u2ExpLength parameter does not take into account the supplemental word needed on the LSB side of the exponent.
5. It is possible to mask the exponent in memory using an 8-bits XOR mask value. Be aware that not only the exponent, but also the supplemental word has to be masked. If masking is not desired, then this parameter must be set to 0.

37.3.5.2.5 Options

The options are set by the u2Options input parameter, which is composed of:

- the mandatory Calculus Mode Option described in Table 37-51
- the mandatory Window Size Option described in Table 37-52
- the indication of the presence of the exponent in Crypto RAM

Note: Please check precisely if one part of the exponent is in Crypto RAM. If this is the case the PUKCL_EXPMOD_EXPINPUKCCRAM must be used.

The u2Options number is calculated by an “Inclusive OR” of the options. Some examples in C language are:

- Operation:Fast Modular Exponentiation with the window size equal to 1 and with no part of the Exponent in the Crypto RAM

```
PUKCL(u2Options) = PUKCL_EXPMOD_FASTRSA | PUKCL_EXPMOD_WINDOWSIZE_1;
```

- Operation: Regular Modular Exponentiation with the window size equal to 2 and with one part of the Exponent in the Crypto RAM

```
PUKCL(u2Options) = PUKCL_EXPMOD_REGULARRSA | PUKCL_EXPMOD_WINDOWSIZE_2 |
PUKCL_EXPMOD_EXPINPUKCCRAM;
```

There is no difference on the final result when using any of the options for this service. The choice has to be made according to the available resources (RAM, Time) and also taking into account the expected security level.

For this service, two exclusive Calculus Modes are possible. The following table describes the Calculus Mode Options.

Table 37-51. ExpMod Service Calculus Mode Option

Option	Explanation
PUKCL_EXPMOD_FASTRSA	Performs a Fast computation
PUKCL_EXPMOD_REGULARRSA	Performs a Regular computation, slower than the Fast version, but using Regular calculus methods

For this service, four window sizes are possible. The window size in bits is those of the windowing method used for the exponent.

The choice of the window size is a balance between the size of the parameters and the computation time:

- Increasing the window size increases the precomputation workspace.
- Increasing the window size reduces the computation time (may not be relevant for very small exponents).

The following table details the size of the precomputation workspace, depending on the chosen window size option.

Table 37-52. ExpMode Service Window Size Options and Precomputation Space Size

Option specified	Size of the PrecompBase Workspace (bytes)	Content of the Workspace
PUKCL_EXPMOD_WINDOWSIZE_1	3*(u2ModLength + 4) + 8	x
PUKCL_EXPMOD_WINDOWSIZE_2	4*(u2ModLength + 4) + 8	$x\ x^3$
PUKCL_EXPMOD_WINDOWSIZE_3	6*(u2ModLength + 4) + 8	$x\ x^3\ x^5\ x^7$
PUKCL_EXPMOD_WINDOWSIZE_4	10*(u2ModLength + 4) + 8	$x\ x^3\ x^5\ x^7\ x^9\ x^{11}\ x^{13}\ x^{15}$

The exponent can be located in RAM or in the data space. If one part of the exponent is in Crypto RAM this must be mandatory signaled by using the option PUKCL_EXPMOD_EXPINPUKCCRAM.

The following table describes this option.

Table 37-53. ExpMod Service Exponent in Crypto RAM Option

Option	Purpose
PUKCL_EXPMOD_EXPINPUKCCRAM	The exponent can be read from any data space of memory, including Flash, RAM or even Crypto RAM. When at least one word the exponent is in Crypto RAM, this option has to be set.

37.3.5.2.6 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;


PUKCL(u2Option) =...;

// Depending on the option specified, not all fields must be filled
PUKCL_ExpMod(nu1ModBase) = <Base of the ram location of N>;
PUKCL_ExpMod(u2ModLength) = <Length of N>;
PUKCL_ExpMod(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL_ExpMod(nu1XBase) = <Base of the ram location of X>;
PUKCL_ExpMod(nu1PrecompBase) = <Base of the ram location of Precomp>;
PUKCL_ExpMod(pfu1ExpBase) = <Base of the location of Exp>;
PUKCL_ExpMod(u2ExpLength) = <Length of Exp>;
...
```

```
// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ExpMod, pvPUKCLParam);
if (PUKCL_Param.Status == PUKCL_OK)
            {
            // operation has been performed correctly
            ...
            }
else // Manage the error
```

37.3.5.2.7 Constraints

The following combinations of input values must be avoided in the case of a modular reduction 'alone', meaning that it has not been requested as an option of any other command:

- nu1ModBase,nu1CnsBase, nu1XBase,nu1PrecompBase,nu1ExpBase are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1XBase, u2ModLength +16},{nu1PrecompBase, <PrecompLength>} are not in Crypto RAM
- {nu1ExpBase,u2ExpLength + 4} has no part in Crypto RAM and PUKCL_EXPMOD_EXPINPUKCCRAM is specified
- u2ModLength or u2ExpLength are either: < 4, > 0xffc or not a 32-bit length
- None or both PUKCL_EXPMOD_REGULARRSA and PUKCL_EXPMOD_FASTRSA are specified.
- {nu1PrecompBase,<PrecompLength>} overlaps with either: {nu1ModBase, u2ModLength +4}, {nu1CnsBase, u2ModLength + 8} {nu1XBase, u2ModLength + 16} or {nu1ExpBase, u2ExpLength + 4}
- {nu1XBase,u2ModLength + 16} overlaps with either: {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8} or {nu1ExpBase, u2ExpLength + 4}
- {nu1ModBase, u2ModLength + 4} overlaps {nu1CnsBase, u2ModLength +8}

37.3.5.2.8 Maximum Sizes for the Modular Exponentiation

The following table provides the maximum sizes for the Modular Exponentiation, depending on the window size and the presence of the exponent in Crypto RAM.

- The figures below are calculated supposing that u2ExpLength =u2ModLength.
- In case of the PUKCL_EXPMOD_EXPINPUKCCRAM option is specified, for the computation of the maximum acceptable size, it is assumed the Exponent is entirely in the Crypto RAM and its length is equal to the Modulus one.
- Otherwise, the Exponent is entirely out of the Crypto RAM and so the computation do not depend on its length.

Table 37-54. Maximum Exponentiation Sizes

Option Specified	Maximum Modulus Size (bytes)	Maximum Modulus Size (bits)
Exponent in Crypto RAM, 1 bit window	576	4608
Exponent in Crypto RAM, 2 bits window	504	4032
Exponent in Crypto RAM, 3 bits window	400	3200
Exponent in Crypto RAM, 4 bits window	284	2272
Exponent not in Crypto RAM, 1 bit window	672	5376
Exponent not in Crypto RAM, 2 bits window	576	4608
Exponent not in Crypto RAM, 3 bits window	448	3584
Exponent not in Crypto RAM, 4 bits window	308	2464

37.3.5.2.9 Status Returned Values

Table 37-55. ExpMod Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	Service functioned correctly

37.3.5.3 Probable Prime Generation (Using Rabin-Miller)

37.3.5.3.1 Purpose

This service is used to perform probable prime generation or test. This service processes integers in GF(p) only.

The options available for this service are:

- Choice of the number of iterations of the Rabin-Miller test
- Generation or Test of a probable prime number
- Fast Implementation
- Regular Implementation
- Exponent Window Size

37.3.5.3.2 Additional Information

The Rabin-Miller test is a probable-primality testing algorithm. As a consequence, the primality of the generated number is not guaranteed at 100%, however, numerous publications have been issued explaining how to estimate the probability of getting a composite number, giving the size of the number and the number of iterations (the T parameter).

Useful information can be found in the *"Handbook of Applied Cryptography (Discrete Mathematics and Its Applications" by Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone*, in the following sections:

- 4.2.3. "Rabin-Miller Test"
- 4.4. "Prime Number Generation"

37.3.5.3.3 How to Use the Service

37.3.5.3.4 Description

This service processes a test for probable primality or a generation of a probable prime number.

Note: When using this service be sure to follow the directives given for the RNG on the chip you use (particularly initialization, seeding) and compulsorily start the RNG.

This service processes one of the following operations: CheckProbablePrimality(N)

or

N = GenerateProbablePrimeFromSeed (NSeed)

In this computation, the following parameters need to be provided:

- N the input number (pointed by {nu1NBase,u2NLength +4})
  - If the requested operation is a test, it is untouched after the operation.
  - If the requested operation is a generation and a probable prime number was found before reaching the Maximum Increment, it contains the resulting probable prime after the operation.
  - If the requested operation was a generation and Maximum Increment was reached before a probable prime number was found, it contains no relevant information.
- Cns as a workspace (pointed by {nu1CnsBase,u2NLength +12})
- Rnd as a workspace (pointed by {nu1RndBase,u2NLength +16})

- Precomp the precomputation workspace (pointed by{nu1PrecompBase,PrecompLen})
- Exp as a workspace (pointed by {pfu1ExpBase,u2ExpLength +4})
- u1MillerRabinIterations the number of Miller Rabin Iterations requested
- u2MaxIncrement, maximum increment of the number in case of probable prime generation

The length PrecompLen depends on the lengths and options chosen; its calculus is detailed in Options below.

The service name for this operation is `PrimeGen`.

37.3.5.3.5 Parameters Definition

Table 37-56. PrimeGen Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1NBase(1)	nu1	I	Crypto RAM	u2NLength + 4	Base of N Number to test or Seed for the generation	Base of N unchanged if test or generation result(2)
nu1CnsBase	nu1	I	Crypto RAM	u2NLength + 12	Base of Cns as a workspace	Base of Cns workspace corrupted
u2NLength	u2	I	–	–	Length of N	Length of N
nu1RndBase	nu1	I	Crypto RAM	Max (u2NLength + 16,64)	Internal Workspace	Internal Workspace corrupted
nu1PrecompBase	nu1	I	Crypto RAM	See Options below	Base of Precomp workspace	Base of Precomp workspace corrupted
nu1RBase(2)	nu1	–	Crypto RAM	–	–	–
nu1ExpBase(3)	nu1	I	Crypto RAM	u2NLength + 4	Base of Exponent (R)	Base of Exponent (R)
u1MillerRabin-Iterations	u1	I	–	–	Miller Rabin's T parameter	Miller Rabin's T parameter
u2MaxIncrement	u2	I	–	–	Maximum Increment(4)	Maximum Increment

Notes:

1. This zone contains the number to be either tested or used as a seed for generation. It has to be provided with one zero word on the MSB side. This area has supplementary constraints (see the following Important note).

1. This parameter does not have to be provided and is used as an internal value for computing the reduction's constant.
2. The area {nu1ExpBase, u2NLength + 4} must be entirely in the Crypto RAM.
3. The generation starts from the number in {nu1NBase,u2NLength + 4} and increments it until a number is found as probable prime. However, the generation may stop for two reasons: The number has been incremented in a way it is bigger than <u2NLength> bytes, or the original number has been incremented by more than <u2MaxIncrement>.

In case of probable prime generation, ensure that the addition of NSeed and Maximum Increment is not a number with more bytes than u2NLength, as this would produce an overflow.

Important:
One additional word is used on the LSB side of the NBase parameter; this word is restored at the end of the calculus. As a consequence, the parameter nu1NBase must never be at the beginning of the Crypto RAM, but at least at one word from the beginning.

One additional word is used on the MSB side of the NBase parameter; this word is not corrupted. As a consequence the Area {nu1NBase, u2NLength} must not be at the end of the Crypto RAM but at least at one word from the end.

Prime numbers of a size lower than 96 bits (three 32-bit words) cannot be generated or tested by this service.

37.3.5.3.6 Options

Some of the Prime Generation options configure the Modular Exponentiation steps and so are very similar to the Modular Exponentiation options.

The options are set by the u2Options input parameter, which is composed of:

- the mandatory Operation Option described in Table 37-57
- the mandatory Calculus Mode Option described in Table 37-58
- the mandatory Window Size Option described in Table 37-59

The u2Options number is calculated by an “Inclusive OR” of the options. Some Examples in C language are:

- Operation: Probable Prime Testing with Fast Modular Exponentiation and the window size equal to 1
  ```
  PUKCL(u2Options) = PUKCL_PRIMEGEN_TEST | PUKCL_EXPMOD_FASTRSA |
  PUKCL_EXPMOD_WINDOWSIZE_1;
  ```
- Operation: Probable Prime Generate with Regular Modular Exponentiation and the window size equal to 2
  ```
  PUKCL(u2Options) = PUKCL_EXPMOD_REGULARRSA | PUKCL_EXPMOD_WINDOWSIZE_2;
  ```

The following table describes the PrimeGen service features available from the various options.

Table 37-57. PrimeGen Service Options

Option	Method Used
PUKCL_PRIMEGEN_TEST	This option is used to specify that only tests will be made on the provided number. When this option is not specified, a prime generation algorithm is selected, starting from the given seed and incrementing it.
PUKCL_EXPMOD_WINDOWSIZE_1,2,3 or 4	Depending on this option, different bit-window sizes will be used. For long exponents, the bigger the window, the faster the computation. However, this has also an impact on the size of the precomputations table.

For this service, two exclusive Calculus Modes are possible. The following table describes the Calculus Mode Options.

Table 37-58. PrimeGen Service Calculus Mode Options

Option	Explanation
PUKCL_EXPMOD_FASTRSA	Perform a Fast computation.
PUKCL_EXPMOD_REGULARRSA	Performs a Regular computation, slower than the Fast version, but using regular calculus methods.

The length of the Precomp area depends on the window size W and u2NLength. The Precomp area length is:

PrecompLen = max(2*(u2NLength + 4) + 2W-1 * (u2NLength + 4), u2NLength + 8 + 64) + 8

Note: Please calculate precisely the length PrecompLen with the formula and the `max()` macro, which takes a maximum of two values.

The following table shows the size of the precomputation workspace (PrecompLen), depending on the chosen window size option.

Table 37-59. PrimeGen Service Precomputation Space Size

Option Specified	Size of the PrecompBase Workspace (bytes)	Content of the Workspace
PUKCL_EXPMOD_WINDOWSIZE_1	max(3*(u2NLength + 4), u2NLength + 72) + 8	x
PUKCL_EXPMOD_WINDOWSIZE_2	max(4*(u2NLength + 4), u2NLength + 72) + 8	$x\ x^3$
PUKCL_EXPMOD_WINDOWSIZE_3	max(6*(u2NLength + 4), u2NLength + 72) + 8	$x\ x^3\ x^5\ x^7$
PUKCL_EXPMOD_WINDOWSIZE_4	max(10*(u2NLength + 4) u2NLength + 72) + 8	$x\ x^3\ x^5\ x^7\ x^9\ x^{11}\ x^{13}\ x^{15}$

The following table provides the maximum sizes for the Prime Generation depending on the window size.

Table 37-60. PrimeGen Service Maximum Sizes

Characteristics of the Operation	Maximum Prime Sizes (bits)
1 bit window	4608
2 bits window	4032
3 bits window	3200
4 bits window	2272

37.3.5.3.7 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
// ! following the directives given for the RNG on the chip PUKCL(u2Option) =...;
// Depending on the option specified, not all fields must be filled
PUKCL_PrimeGen(nu1NBase) = <Base of the ram location of N>;
PUKCL_PrimeGen(u2NLength)  =  <Length  of  N>;
PUKCL_PrimeGen(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL_PrimeGen(nu1PrecompBase) = <Base of the ram location of Precomp>;
PUKCL_PrimeGen(pfu1ExpBase) = <Base of the location of Exp>;
PUKCL_PrimeGen(u2ExpLength) = <Length of Exp>;
PUKCL_PrimeGen(u1MillerRabinIterations) = <Number of iterations>;
PUKCL_PrimeGen(u2MaxIncrement) = <Maximum Increment>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(PrimeGen, pvPUKCLParam);
if (PUKCL_Param.Status == PUKCL_NUMBER_IS_PRIME)
            {
            // The number is probably prime
            ...
            }
else if (PUKCL_Param.Status == PUKCL_NUMBER_IS_NOT_PRIME)
            {
            // The number is not prime
            ...
            }
else // Manage the error
```

37.3.5.3.8 Constraints

The following combinations of input values must be avoided in the case of a modular reduction ‘alone’, meaning that it has not been requested as an option of any other service:

- nu1NBase,nu1CnsBase, nu1RndBase,nu1PrecompBase,nu1ExpBase are not aligned on 32-bit boundaries
- {nu1NBase, u2NLength + 4}, {nu1CnsBase, u2NLength + 12}, {nu1RndBase, u2NLength +12}, {nu1PrecompBase, <PrecompLength>} are not in Crypto RAM
- u2NLength is either: < 12, > 0xffc or not a 32-bit length
- Both PUKCL_EXPMOD_REGULARRSA and PUKCL_EXPMOD_FASTRSA are specified.
- {nu1PrecompBase,<PrecompLength>} overlaps with either: {nu1NBase, u2NLength + 4}, {nu1CnsBase, u2NLength + 12} {nu1RndBase, u2NLength + 12} or {nu1ExpBase, u2ExpLength + 4}
- {nu1RndBase,3*u2NLength + 24} overlaps with either: {nu1NBase, u2NLength + 4},{nu1CnsBase, u2NLength + 12} {nu1XBase, u2NLength + 12} or {nu1ExpBase, u2ExpLength + 4}
- {nu1NBase, u2NLength + 4} overlaps {nu1CnsBase, u2NLength +12}

37.3.5.3.9 Status Returned Values

Table 37-61. PrimeGen Service Return Codes

Returned Status	Importance	Meaning
PUKCL_NUMBER_IS_PRIME	Information	The generated or tested number has been detected as probably prime.
PUKCL_NUMBER_IS_NOT_PRIME	Information	The generated or tested number has been detected as composite.

37.3.5.4 Modular Exponentiation (With CRT)

37.3.5.4.1 Purpose

The purpose of this service is to perform the Modular Exponentiation with the Chinese Remainders Theorem (CRT). This service processes integers in GF(p) only.

The options available for this service are:

- Fast implementation
- Regular implementation
- Exponent is located in Crypto RAM or not
- Exponent window size

37.3.5.4.2 How to Use the Service

37.3.5.4.3 Description

This service processes a Modular Exponentiation with the Chinese Remainder Theorem:

$R = X^{D} mod(N)$ *with* $N = P * Q$

> **Important:** For this service, be sure to follow the directives given for the RSA implementation on the chip you use.

This service requires that the modulus N is the product of two co-primes P and Q and that the decryption exponents D is co-prime with the product ((P-1)*(Q-1)).

The Input data are P, Q, EP, EQ, Rvalue, and X. P and Q are the co-primes so that N = P*Q.

X is the number to exponentiate.

EP, EQ and Rval are calculated as follows:

$EP = Dmod(P - 1)$ $EQ = Dmod(Q - 1)$ $Rval = P^{-1} mod(Q)$

In some cases, the decryption exponent D may not be available and the encryption exponent E may be available instead. The possibilities to calculate the parameters are:

- Calculate D from E with the formula:
  $D = E^{-1} mod((P - 1) \times (Q - 1))$
- Calculate the parameters from E:
  $EP = E^{-1} mod(P - 1)$ $EQ = E^{-1} mod(Q - 1)$ $Rval = P^{-1} mod(Q)$

In this computation, the following parameters need to be provided:

- X the input number (pointed by {nu1XBase,2*u2ModLength +16})
- P and Q the primes (pointed by {nu1ModBase,2*u2ModLength +8}).
- EP and EQ the reduced exponents (pointed by {pfu1ExpBase,2*u2ExpLength +8})
- Rval and Precomp (pointed by{nu1PrecompBase,RAndPrecompLen})
- Blinding the exponent blinding value (provided inu1Blinding)

The length RAndPrecompLen depends on the lengths and options chosen; its calculus is detailed in Options below.

The service for this operation is CRT.

Note: The minimum value for u2ModLength is 12 bytes. Therefore, the significant length of P or Q must be at least three 32-bit words.

37.3.5.4.4 Parameters Definition

The following table shows the parameter block for the CRT service.

Many parameters have complex placement in memory; therefore, detailed figures are provided in CRT Service Placement below.

Table 37-62. CRT Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Options	u2	I	–	–	Options (see below)	Options (see below)
nu1ModBase	nu1	I	Crypto RAM	2*u2ModLength + 8	Base of P, Q	Base of P, Q untouched
u2ModLength	u2	I	–	–	Length of P or Q greater than or equal to 12	Length of P or Q
nu1XBase(1)	nu1	I	Crypto RAM	2*u2ModLength + 16	Base of X	Base of X Filled with the result
nu1PrecompBase	nu1	I	Crypto RAM	See Options below	Base of Rvalue and Pre computations workspace	Corrupted
pfu1ExpBase(2)	pfu1	I	Any place	2*u2ExpLength + 8	Base of EP, EQ	Base of EP, EQ untouched
u2ExpLength	u2	I	–	–	Significant length of EP or EQ	Significant length of EP or EQ
u1Blinding(3)	u4	I	–	–	Exponent unblinding value	Exponent unblinding value

Notes:

1. This zone contains the number to be exponentiated (u2ModLength bytes) and is used during the computations as a workspace (four 32-bit words longer than the number to be exponentiated). At the end of the computation, it contains the correct result of the operation.
2. If the PUKCL_EXPMOD_EXPINPUKCCRAM option is not set, the location of the exponent MUST NOT be placed in the Crypto RAM, even partially.
3. It is possible to mask the exponent in memory using a 32-bit XOR mask value. Be aware that not only the exponent, but also the supplemental spill word has to be masked. If masking is not desired, the parameter must be set to 0.

37.3.5.4.5 Options

Most of the CRT options configure the Modular Exponentiation steps of the CRT and so are very similar to the Fast Modular Exponentiation options.

The options are set by the u2Options input parameter, which is composed of:

- the mandatory Calculus Mode Option described in Table 37-63
- the mandatory Window Size Option described in Table 37-64
- the indication of the presence of the exponent in Crypto RAM

Important: Please check precisely if one part of the exponent area (containing EP and EQ) is in Crypto RAM. If this is the case, the PUKCL_EXPMOD_EXPINPUKCCRAM option must be used.

The u2Options number is calculated by an “Inclusive OR” of the options. Some Examples in C language are:

- Operation: CRT using the Fast Modular Exponentiation with the window size equal to 1 and with no part of the Exponent area in the Crypto RAM
  ```
  PUKCL(u2Options) = PUKCL_EXPMOD_FASTRSA | PUKCL_EXPMOD_WINDOWSIZE_1;
  ```
- Operation:CRT using the Regular Modular Exponentiation with the window size equal to 2 and with one part the Exponent area in the Crypto RAM
  ```
  PUKCL(u2Options) = PUKCL_EXPMOD_REGULARRSA | PUKCL_EXPMOD_WINDOWSIZE_2 |
  PUKCL_EXPMOD_EXPINPUKCCRAM;
  ```

For this service, two exclusive Calculus Modes for the Modular Exponentiation steps of the CRT are possible. The following table describes the Calculus Mode Options.

Table 37-63. CRT Service Calculus Mode Options

Option	Explanation
PUKCL_EXPMOD_FASTRSA	Perform a Fast computation.
PUKCL_EXPMOD_REGULARRSA	Performs a Regular computation, slower than the Fast version, but using regular calculus methods.

For this service, four window sizes for the Modular Exponentiation Steps are possible. The window size in bits is those of the windowing method used for the exponent.

The choice of the window size is a balance between the size of the parameters and the computation time:

- Increasing the window size increases the precomputation workspace.
- Increasing the window size reduces the computation time (may not be relevant for very small exponents). The length of the Rval and Precomp area depends on the window size W and u2ModLength.

The Rval and Precomp area length is:

$$RandPrecompLen = 4 * (u2ModLength + 4) + \max(64 , 2^{(W-1)} * (u2ModLength + 4)) + 8$$

Important: Please calculate precisely the length RandPrecompLen with the formula and the `max()` macro, which takes the maximum of two values.

The following table shows the size of the Rval and Precomp area, depending on the chosen window size option.

Table 37-64. CRT Service Window Size Options and Rval and Precomp Area Size

Option Specified	Size of the Rval and Precomp Area (bytes)	Precomputation Values
PUKCL_EXPMOD_WINDOWSIZE_1	4*(u2ModLength + 4) + max(64 , (u2ModLength + 4)) + 8	x
PUKCL_EXPMOD_WINDOWSIZE_2	4*(u2ModLength + 4) + max(64 , 2*(u2ModLength + 4)) + 8	$x\ x^3$
PUKCL_EXPMOD_WINDOWSIZE_3	4*(u2ModLength + 4) + max(64 , 4*(u2ModLength + 4)) + 8	$x\ x^3\ x^5\ x^7$
PUKCL_EXPMOD_WINDOWSIZE_4	10*(u2ModLength + 4) + max(64 , 8*(u2ModLength + 4)) + 8	$x\ x^3\ x^5\ x^7\ x^9\ x^{11}\ x^{13}\ x^{15}$

The exponent area can be located in RAM or in the data space. If one part of the exponent area is in Crypto RAM this must be mandatory signaled by using the PUKCL_EXPMOD_EXPINPUKCCRAM option.

The following table describes this option.

Table 37-65. CRT Service Crypto RAM Option Exponent Area

Option	Purpose
PUKCL_EXPMOD_EXPINPUKCCRAM	The exponent area can be read from any data space of memory, including Crypto RAM. When at least one word the exponent is in Crypto RAM, this option has to be set.

37.3.5.4.6 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;


PUKCL(u2Option) =...;

// Depending on the option specified, not all fields must be filled PUKCL_CRT(nu1ModBase) =
<Base of the ram location of P and Q>; PUKCL_CRT(u2ModLength) = <Length of P or Q>;
PUKCL_CRT(nu1XBase) = <Base of the ram location of X>;
PUKCL_CRT(nu1PrecompBase) = <Base of the ram location of RVal and Precomp>;
PUKCL_CRT(pfu1ExpBase) = <Base of the ram location of EP and EQ>;
PUKCL_CRT(u2ExpLength) = <Length of EP or EQ>;
PUKCL_CRT(u1Blinding) = <Blinding value>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(CRT, pvPUKCLParam);
if (PUKCL_Param.Status == PUKCL_OK)
            {
            // operation has been performed correctly
            ...
            }
else // Manage the error
```

37.3.5.4.7 Constraints

The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1XBase, nu1PrecompBase, pfu1ExpBase are not aligned on 32-bit boundaries
- {nu1XBase, 2*u2ModLength + 16}, {nu1ModBase, 2*u2ModLength + 8}, {nu1PrecompBase,<PrecompLength>} are not in Crypto RAM
- {nu1ExpBase,2*u2ExpLength + 8} is not in Crypto RAM and PUKCL_EXPMOD_EXPINPUKCCRAM is specified
- u2ModLength or u2ExpLength are either: < 4, > 0xffc or not a 32-bit length
- None or both PUKCL_EXPMOD_REGULARRSA and PUKCL_EXPMOD_FASTRSA are specified.
- {nu1XBase,2*u2ModLength + 16} overlaps with either: {nu1ModBase, 2*u2ModLength +8}, {nu1PrecompBase, <PrecompLength>} or {pfu1ExpBase, 2*u2ExpLength + 8}
- {nu1ModBase,2*u2ModLength + 8} overlaps with either: {nu1PrecompBase, <PrecompLength>} or {pfu1ExpBase, 2*u2ExpLength + 8}
- {nu1PrecompBase, <PrecompLength>} overlaps {pfu1ExpBase, 2*u2ExpLength +8}

37.3.5.4.8 CRT Service Parameter Placement

The parameters' placements are described in detail in the following figures.

Figure 37-2. Modulus P and Q in {nu1ModBase, 2*u2ModLength + 8}

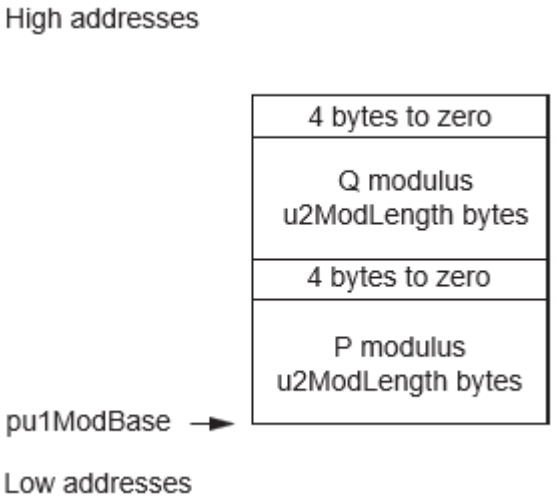


Figure 37-3. Value X in {nu1XBase, 2*u2ModLength + 16}

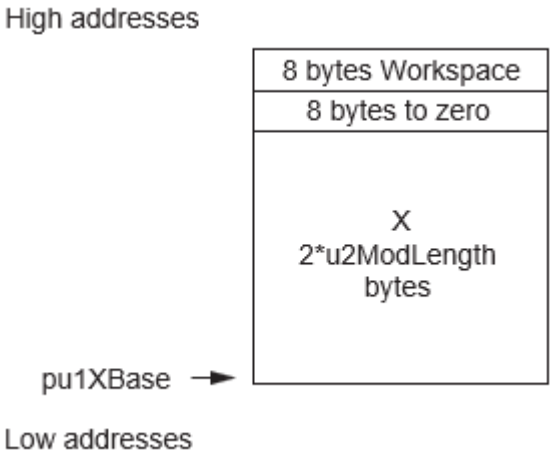


Figure 37-4. Exponents EP and EQ in {fnu1ExpBase, 2*u2ExpLength + 8}

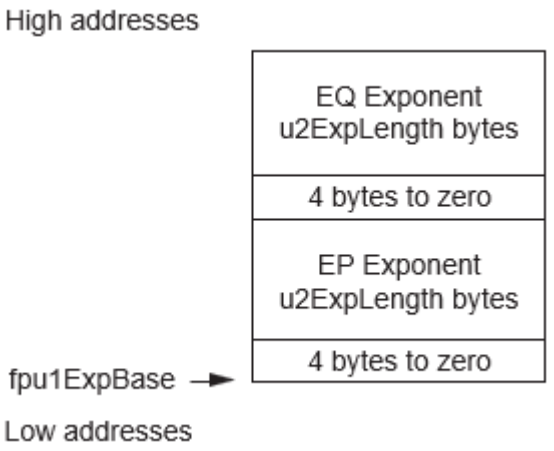
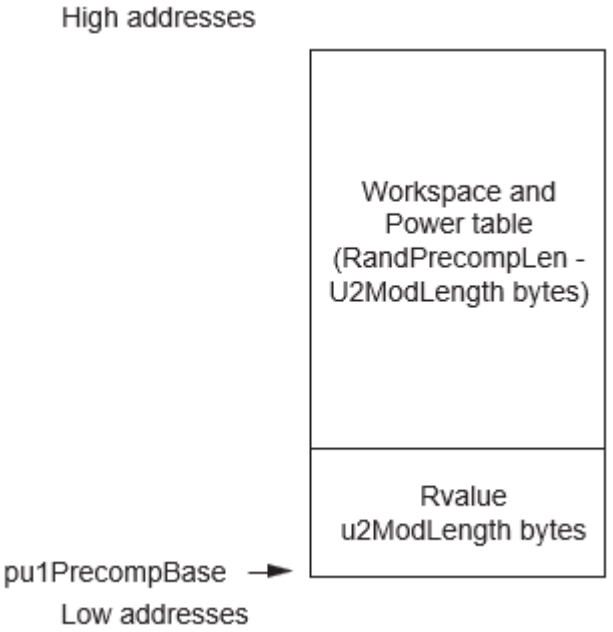


Figure 37-5. Value Rval and Precomp in {nu1PrecompBase, RandPrecompLen}



37.3.5.4.9 CRT Service Modular Exponentiation Maximum Size

The following table details the maximum size in bits of P or Q, of N and of EP or EQ.

- The maximum size in bits of P or Q equals:
  <Max Size Bits P> = <Max Size Bits Q> = 8 * <Max u2ModLength bytes>
- The maximum size in bits of N=P*Q equals:
  <Max Size Bits N> = 2 * <Max Size Bits P>
- The maximum size in bits of EP or EQ equals:
  <Max Size Bits EP> = <Max Size Bits EQ> = 8 * <Max u2ExpLength bytes>
- In case of the PUKCL_EXPMOD_EXPINPUKCCRAM option is specified, for the computation of the maximum acceptable size, it is assumed the Exponent is entirely in the Crypto RAM and its length equal the Modulus one.

- Otherwise, the Exponent is entirely out of the Crypto RAM and so the computation do not depend on its length.

Table 37-66. CRT Service Maximum Sizes

Characteristics of the Operation	P or Q Max Bit Sizes	N Max Bit Sizes	EP or EQ Max Bit Sizes
Exponent in Crypto RAM, 1 bit window	2912	5824	2912
Exponent in Crypto RAM, 2 bits window	2688	5376	2688
Exponent in Crypto RAM, 3 bits window	2464	4928	2464
Exponent in Crypto RAM, 4 bits window	2304	4608	2304
Exponent not in Crypto RAM, 1 bit window	3584	7168	<application dependent>
Exponent not in Crypto RAM, 2 bits window	3232	6464	<application dependent>
Exponent not in Crypto RAM, 3 bits window	2912	5824	<application dependent>
Exponent not in Crypto RAM, 4 bits window	2688	5376	<application dependent>

37.3.5.4.10 Status Returned Values

Table 37-67. CRT Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	Information	Service functioned correctly

37.3.6 Elliptic Curves Over GF(p) Services

This section provides a complete description of the currently available elliptic curve over Prime Fields services. These services process integers in GF(p) only.

The offered services cover the basic operations over elliptic curves such as:

- Adding two points over a curve
- Doubling a point over a curve
- Multiplying a point by an integral constant
- Converting a point's projective coordinates (resulting from a doubling or an addition) to the affine coordinates, and oppositely converting a point's affine coordinates to the projective coordinates.
- Testing the point presence on the curve.

Additionally, some higher level services covering the needs for signature generation and verification are offered:

- Generating an ECDSA signature (compliant with FIPS186-2)
- Verifying an ECDSA signature (compliant with FIPS186-2) The supported curves use the following curve equation:

$Y^2 = X^3 + aX + b$

37.3.6.1 Coordinate Systems

Related Links

37.3.5.1. Modular Reduction

37.3.6.1.1 General Considerations

In this implementation, several choices have been made related to the coordinate systems managed by the elliptic curve primitives.

There are two systems currently managed by the library:

- Affine Coordinates System where each curve point has two coordinates (X, Y)
- Projective Coordinates System where each point is represented with three coordinates (X,Y, Z)

Converting from the affine coordinates system to a projective coordinates system is performed by extending its representation with Z = 1:

$(X, Y) \Rightarrow (X, Y, Z= 1)$

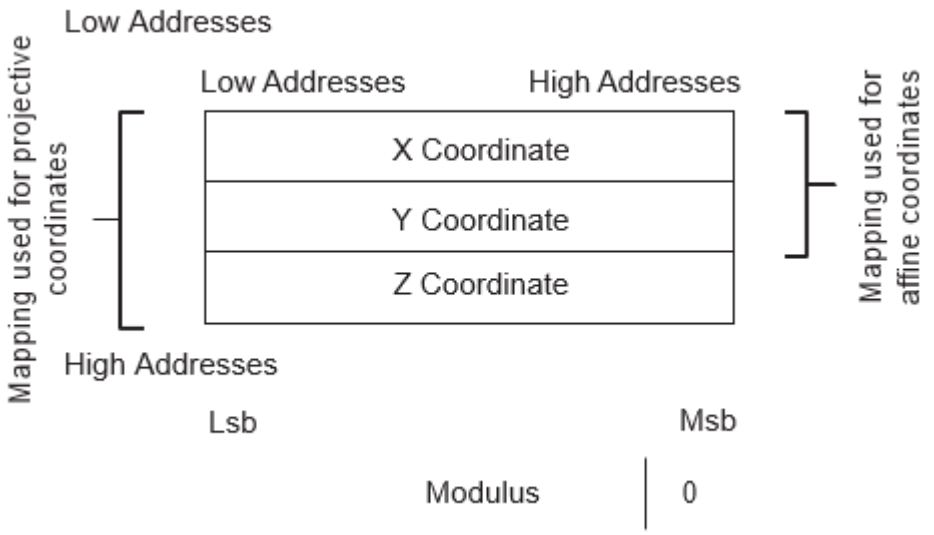
Converting from a projective coordinate to an affine one is a service offered by the PUKCL. The formula to perform this conversion is:

$(X, Y, Z) \Rightarrow (X / Z^2, Y / Z^3)$

37.3.6.1.2 Points Representations

Depending on the representation (Projective or Affine), points are represented tn memory, as shown in the following figure.

Figure 37-6. Points Representation in Memory



In this figure, the modulus is represented as a reference, and to show that coordinates are always to be provided on the length of the modulus plus one 32-bit word.

The different types of representations are as follows:

Affine representation $Pt = \begin{bmatrix} X_{Affine} < P \times 2^{15} \\ Y_{Affine} < P \times 2^{15} \end{bmatrix}$

Projective representation $Pt = \begin{bmatrix} X_{Projective} < P \times 2^{15} \\ Y_{Projective} < P \times 2^{15} \\ Z_{Projective} < P \times 2^{15} \end{bmatrix}$

Notes:

1. The minimum value for u2ModLength is 12 bytes. Therefore, the significant length of the modulus must be at least three 32-bit words.
2. In some cases the point can be the infinite point. In this case, it is represented with its Z coordinates equal or congruent to zero.

37.3.6.1.3 Modulus and Modular Constant Parameters

In most of the services the following parameters must be provided:

- P the Modulus (often pointed by {nu1ModBase,u2ModLength + 4}): This parameter contains the Modulus Integer prime P defining the Galois Field used in points coordinates computations. The Modulus must be u2ModLength bytes long, while having a supplemental zeroed 32-bit word on the MSB side.
  Note: Most of the Elliptic Curve computations are reduced modulo P. In many functions the reductions are made with the Fast Reduction.
- Cns the Modular Constant (often pointed by {nu1CnsBase,u2ModLength + 12}): This parameter contains the Modular Constant associated to the Modulus

Important: The Modular Constant must be calculated before using the GF(p) Elliptic Curves functions by a call to the Setup for Modular Reductions with the GF(p) option (see *Modular Reduction Setup* in the *Modular Reduction* from Related Links).

37.3.6.2 Point Addition

37.3.6.2.1 Purpose

This service is used to perform a point addition, based on a given elliptic curve over GF(p). Please note that:

- This service is not intended to add the same point twice. In this particular case, use the doubling service
  (see 37.3.6.4. Fast Point Doubling).

37.3.6.2.2 How to Use the Service

37.3.6.2.3 Description

The operation performed is:

$Pt_C = Pt_A + Pt_B$

In this computation, the following parameters need to be provided:

- A the input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointABase,3*u2ModLength + 12}). This point can be the Infinite Point.
- B the input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointBBase,3*u2ModLength + 12}). This point can be the Infinite Point.
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength +8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength +4})
- The workspace not initialized (pointed by {nu1WorkSpace, 5*u2ModLength +32}

The resulting C point is represented in projective coordinates (X,Y,Z) and is stored at the very same place than the input point A. This Point can be the Infinite Point.

The service name for this operation is `ZpEccAddFast`. This service uses Fast mode and Fast Modular Reduction for computations.

Important: Before using this service, ensure that the constant Cns has been calculated with the Setup of the Modular Reduction functions.

37.3.6.2.4 Parameters Definition

Table 37-68. ZpEccAddFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of Modulus P	Base of Modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulo	Length of modulo
nu1PointABase	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Input point A (projective coordinates)	Resulting point C (projective coordinates)
nu1PointBBase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point B (projective coordinates)	Input point B
nu1Workspace	nu1	I	Crypto RAM	5*u2ModLength + 32	–	Corrupted workspace

37.3.6.2.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

PUKCL (u2Option) = 0;

PUKCL _ZpEccAdd(nu1ModBase) = <Base of the ram location of P>;
PUKCL _ZpEccAdd(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _ZpEccAdd(u2ModLength) = <Byte length of P>;
PUKCL _ZpEccAdd(nu1PointABase) = <Base of the ram location of the A point>;
PUKCL _ZpEccAdd(nu1PointBBase) = <Base of the ram location of the B point>;
PUKCL _ZpEccAdd(nu1Workspace) = <Base of the ram location of the workspace>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEccAddFast,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.6.2.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1PointBBase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1PointBBase, 3*u2ModLength + 12}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8},{nu1PointABase, 3*u2ModLength + 12}, {nu1PointBBase, 3*u2ModLength + 12} and {nu1Workspace, 5*u2ModLength + 32}

37.3.6.2.7 Status Returned Values

Table 37-69. ZpEccAddFast Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.6.3 Point Addition and Subtraction

37.3.6.3.1 Purpose

This service is used to perform a point addition and point subtraction, based on a given elliptic curve over GF(p). Please note that:

- This service is not intended to add the same point twice. In this particular case, use the doubling service (see 37.3.6.4. Fast Point Doubling).

37.3.6.3.2 How to Use the Service

37.3.6.3.3 Description

The operation performed is:

$Pt_C = Pt_A \pm Pt_B$

In this computation, the following parameters need to be provided:

- A the input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointABase,3*u2ModLength + 12}). This point can be the Infinite Point.
- B the input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointBBase,3*u2ModLength + 12}). This point can be the Infinite Point.
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength +8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength +4})
- The workspace not initialized (pointed by {nu1WorkSpace, 5*u2ModLength +32}
- The operator filled with the operation to perform (Addition or Subtraction)

The resulting C point is represented in projective coordinates (X,Y,Z) and is stored at the very same place than the input point A. This Point can be the Infinite Point.

The service name for this operation is `ZpEccAddSubFast`. This service uses Fast mode and Fast Modular Reduction for computations.

Note: Before using this service, ensure that the constant Cns has been calculated with the setup of the modular reduction functions.

37.3.6.3.4 Parameters Definition

Table 37-70. ZpEccAddSubFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of Modulus P	Base of Modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulo	Length of modulo
nu1PointABase	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Input point A (projective coordinates)	Resulting point C (projective coordinates)
nu1PointBBase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point B (projective coordinates)	Input point B
u2Operator	u2	I	-	-	Addition or Subtraction	Addition or Subtraction
nu1Workspace	nu1	I	Crypto RAM	5*u2ModLength + 32	–	Corrupted workspace

37.3.6.3.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;


PUKCL (u2Option) = 0;

PUKCL _ZpEccAddSub(nu1ModBase) = <Base of the ram location of P>;
```

```
PUKCL _ZpEccAddSub(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _ZpEccAddSub(u2ModLength) = <Byte length of P>;
PUKCL _ZpEccAddSub(nu1PointABase) = <Base of the ram location of the A point>;
PUKCL _ZpEccAddSub(nu1PointBBase) = <Base of the ram location of the B point>;
PUKCL _ZpEccAddSub(nu1Workspace) = <Base of the ram location of the workspace>;
PUKCL _ZpEccAddSub(u2Operator) = <Operation to perform (PUKCL_ZPECCADD or PUKCL_ZPECCSUB)>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEccAddSubFast,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
        {
        ...
        }
else // Manage the error
```

37.3.6.3.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1PointBBase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1PointBBase, 3*u2ModLength + 12}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8},{nu1PointABase, 3*u2ModLength + 12}, {nu1PointBBase, 3*u2ModLength + 12} and {nu1Workspace, 5*u2ModLength + 32}

37.3.6.3.7 Status Returned Values

Table 37-71. ZpEccAddFast Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.6.4 Fast Point Doubling

37.3.6.4.1 Purpose

This service is used to perform a Point Doubling, based on a given elliptic curve over GF(p).

37.3.6.4.2 How to Use the Service

37.3.6.4.3 Description

These two services process the Point Doubling:

$Pt_C = 2 \times Pt_A$

In this computation, the following parameters need to be provided:

- A the input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointABase,3*u2ModLength + 12}). This point can be the Infinite Point.
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength +8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength +4})
- The workspace not initialized (pointed by {nu1WorkSpace, 4*u2ModLength +28}
- The a parameter relative to the elliptic curve (pointed by {nu1ABase,u2ModLength +4})
- The resulting C point is represented in projective coordinates (X,Y,Z) and is stored at the same location than the input point A. This point can be the Infinite Point.

The service name for this operation is `ZpEccDblFast`. This service uses Fast mode and Fast Modular Reduction for computations.

Important: Before using this service, ensure that the constant Cns has been calculated with the setup of the Fast Modular Reduction service.

37.3.6.4.4 Parameters Definition

Table 37-72. ZpEccDblFastService

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1ABase	u2	I	Crypto RAM	u2ModLength + 4	Parameter a of the elliptic curve	Parameter a of the elliptic curve
nu1PointABase	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Input point A (projective coordinates)	Resulting point C (projective coordinates)
nu1Workspace	nu1	I	Crypto RAM	4*u2ModLength + 28	–	Corrupted workspace

37.3.6.4.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;


PUKCL (u2Option) = 0;

PUKCL _ZpEccDbl(nu1ModBase) = <Base of the ram location of P>;
PUKCL _ZpEccDbl(u2ModLength) = <Byte length of P>;
PUKCL _ZpEccDbl(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _ZpEccDbl(nu1PointABase) = <Base of the ram location of the A point>;
PUKCL _ZpEccDbl(nu1ABase) = <Base of the a parameter of the elliptic curve>;
PUKCL _ZpEccDbl(nu1Workspace) = <Base of the ram location of the workspace>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEccDblFast,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.6.4.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1ABase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength+ 12}, {nu1ABase, u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1ABase, u2ModLength + 4} and {nu1Workspace, 4*u2ModLength + 28}

37.3.6.4.7 Status Returned Values

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.6.5 Fast Multiplying by a Scalar Number of a Point

37.3.6.5.1 Purpose

This service is used to multiply a point by an integral constant K on a given elliptic curve over GF(p).

37.3.6.5.2 How to Use the Service

37.3.6.5.3 Description

These two services process the Multiplying by a scalar number:

$Pt_C = K \times Pt_A$

In this computation, the following parameters need to be provided:

- A the input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointABase,3*u2ModLength + 12}). This point can be the Infinite Point.
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength +8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength +4})
- The workspace not initialized (pointed by {nu1WorkSpace, 8*u2ModLength +44}
- The a parameter relative to the elliptic curve (pointed by {nu1ABase,u2ModLength +4})
- K the scalar number (pointed by {nu1ScalarNumber,u2ScalarLength +4})

The resulting C point is represented in projective coordinates (X,Y,Z) and is stored at the very same place than the input point A. This point can be the Infinite Point.

The service name for this operation is `ZpEccMulFast`. This service uses Fast mode and Fast Modular Reduction for computations.

Note: Before using this service, ensure that the constant Cns has been calculated with the setup of the Fast Modular Reduction service.

37.3.6.5.4 Parameters Definition

Table 37-73. ZpEccMulFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1KBase	nu1	I	Crypto RAM	u2KLength	Scalar number used to multiply the point A	Unchanged
u2KLength	u2	I	–	–	Length of scalar K	Length of scalar K
nu1PointABase	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Input point A (projective coordinates)	Resulting point C (projective coordinates)
nu1ABas	nu1	I	Crypto RAM	u2ModLength + 4	Parameter a of the elliptic curve	Unchanged
nu1Workspace	nu1	I	Crypto RAM	8*u2ModLength + 44	–	Corrupted workspace

37.3.6.5.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;


PUKCL (u2Option) = 0;
```

```
PUKCL _ZpEccMul(nu1ModBase) = <Base of the ram location of P>;
PUKCL _ZpEccMul(u2ModLength) = <Byte length of P>;
PUKCL _ZpEccMul(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _ZpEccMul(nu1PointABase) = <Base of the ram location of the A point>;
PUKCL _ZpEccMul(nu1ABase) = <Base of the ram location of the parameter A of the elliptic
curve>;
PUKCL _ZpEccMul(nu1KBase) = <Base of the ram location of the scalar number>;
PUKCL _ZpEccMul(nu1Workspace) = <Base of the ram location of the workspace>;
PUKCL_ZpEccMul(u2KLength) = <Byte length of the Scalar Number K>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEccMulFast,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.6.5.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase,nu1CnsBase, nu1PointABase, nu1ABase, nu1ScalarNumber, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength+ 12}, {nu1ABase, u2ModLength + 4}, {nu1ScalarNumber, u2ScalarLength} or {nu1Workspace, 8*u2ModLength + 44} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1ABase, u2ModLength + 4}, {nu1ScalarNumber, u2ScalarLength} and {nu1Workspace, 8*u2ModLength + 44}

37.3.6.5.7 Status Returned Values

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.6.6 Quick Dual Multiplying by Two Scalar Numbers and Two Points

37.3.6.6.1 Purpose

This service is used to multiply two points by two integral constants K1 and K2, and then provide the addition of these multiplications results.

Important: This service has a quick implementation without additional security.

37.3.6.6.2 How to Use the Service

37.3.6.6.3 Description

This service processes the dual Multiplying by two scalar numbers:

$PtC = K_1 \times Pt_A + K_2 \times Pt_B$

In this computation, the following parameters need to be provided:

- A the first input point is filled in projective coordinates (X,Y,Z) (pointed by {pu1PointABase, (3*(u2ModLength + 4)) * (2(WA-2))}). This point can be the Infinite Point.

- B the 2nd input point is filled in projective coordinates (X,Y,Z) (pointed by {pu1PointBBase, (3*(u2ModLength + 4)) * (2(WB-2))}). This point can be the Infinite Point.
- P the modulus filled and Cns the Fast Modular Constant filled (pointed by {pu1ModCnsBase,2*u2ModLength + 16})
- The a parameter filled and the workspace not initialized (pointed by {pu1AWorkBase, 9*u2ModLength +48}
- KAB the scalar numbers (pointed by {pu1KABBase, 2*u2KLength +8})
- The options are set by the u2Options input parameter, which is composed of:
  - wA: Size of window for Point A between 2 and15
  - wB: Size of window for Point B between 2 and15
  - PUKCL_ZPECCMUL_SCAL_IN_CLASSIC_RAM flag: to set only if the scalars are entirely in Classic RAM with no part in PUKCC RAM

The resulting C point is represented in projective coordinates (X,Y,Z) and is stored at (pu1AWorkBase + u2ModLength + 4). This point can be the Infinite Point.

Important: Before using this service, ensure that the constant Cns has been calculated with the setup of the Fast Modular Reduction service.

37.3.6.6.4 Parameters Definition

WA is the Point A window size and WB is the Point B window size (see Options below for details).

Important: Please calculate precisely the length of areas with the formulas. Ensure that the pu1 type is a pointer on 4 bytes and contains the full address (see 37.3.3.4. Aligned Significant Length).

Table 37-74. ZpEccQuickDualMulFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
pu1ModCnsBase	pu1	I	Crypto RAM	2 * u2ModLength + 16	Base of modulus P, Base of Cns	Base of modulus P, Base of Cns
u2Option	u2	I	–	–	Option related to the called service (see below)	–
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
pu1KABBase	pu1	I	Any RAM	2 * u2KLength + 8	Scalar numbers used to multiply the points A and B	Unchanged
u2KLength	u2	I	–	–	Length of scalars KA and KB	Length of scalars KA and KB
pu1PointABase	pu1	I/O	Crypto RAM	(3*(u2ModLength + 4)) * $(2^{(WA-2)})$ (1)	Input point A (projective coordinates)	Unchanged
pu1PointBBase	pu1	I	Crypto RAM	(3*(u2ModLength + 4)) * $(2^{(WB-2)})$ (2)	Input point B (projective coordinates)	Unchanged
pu1AWorkBase	pu1	I	Crypto RAM	9*u2ModLength + 48	Parameter a of the elliptic curve	Resulting point C (projective coordinates) in pu1AWorkBase Base + u2ModLength + 4

Notes:

1. The precalculus table size for the point A is calculated from chosen window size “WA”.
2. The precalculus table size for the point B is calculated from chosen window size “WB”.

37.3.6.6.5 Options

The options are set by the u2Options input parameter, which is composed of:

- the mandatory windows sizes WA and WB
- the indication of the presence of the scalars in system RAM

Note: Please check precisely if one part of the scalars is in Crypto RAM. If this is the case, the PUKCL_ZPECCMUL_SCAL_IN_CLASSIC_RAM option must not be used.

The u2Options number is calculated by an “Inclusive OR” of the options. Some Examples in C language are:

- ```
  // Scalars are in system RAM
  // The Point A window size is 3
  // The Point B window size is 4
  PUKCL(u2Options) = PUKCL_ZPECCMUL_SCAL_IN_CLASSIC_RAM |
  PUKCL_ZPECCMUL_WINSIZE_A_VAL_TO_OPT(3) |
  PUKCL_ZPECCMUL_WINSIZE_B_VAL_TO_OPT(4);
  ```
- ```
  // Scalars are in the PUKCC Cryptographic RAM
  // The Point A window size is 2
  // The Point B window size is 5
  PUKCL(u2Options) = PUKCL_ZPECCMUL_WINSIZE_A_VAL_TO_OPT(2) |
  PUKCL_ZPECCMUL_WINSIZE_B_VAL_TO_OPT(5);
  ```

For this service, many window sizes are possible. The window sizes in bits are those of the windowing method used for the scalar multiplying.

The choice of the window sizes is a balance between the size of the parameters and the computation time:

- Increasing the window size increases the precomputation table size.
- Increasing the window size to the optimum reduces the computation time.

The following table details the size of the point and the precomputation table, depending on the chosen window size option.

Table 37-75. ZpEccQuickDualMulFast Service Window Size Options and Precomputation Table Size

Option Specified	Size of the Point and the Precomputation Table
PUKCL_ZPECCMUL_WINSIZE_A_VAL_TO_OPT(WA) WA in [2, 15]	(3*(u2ModLength + 4)) * ($2^{(WA-2)}$)
PUKCL_ZPECCMUL_WINSIZE_B_VAL_TO_OPT(WB) WB in [2, 15]	(3*(u2ModLength + 4)) * ($2^{(WB-2)}$)

The scalars can be located in PUKCC RAM or in system RAM. If both scalars are entirely in system RAM with no part in PUKCC RAM this can be signaled by using the option PUKCL_ZPECCMUL_SCAL_IN_CLASSIC_RAM. In all other cases this option must not be used.

The following table describes this option.

Table 37-76. ZpEccQuickDualMulFast Service System RAM Scalar Options

Option	Purpose
PUKCL_ZPECCMUL_SCAL_IN_CLASSIC_RAM	The scalars can be located in Crypto RAM or in system RAM. If both scalars are entirely in system RAM with no part in Crypto RAM this can be signaled by using this option . In all other cases this option must not be used.

37.3.6.6.6 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

PUKCL(u2Option) = <Configure scalar numbers location and windows sizes>;
PUKCL_ZpEccQuickDualMulFast(pu1ModCnsBase) = <Base of the ram location of P and Cns>;
PUKCL_ZpEccQuickDualMulFast(u2ModLength) = <Byte length of P>;
PUKCL_ZpEccQuickDualMulFast(u2KLength) = <Byte length of scalars>;
PUKCL_ZpEccQuickDualMulFast(pu1PointABase) = <Base of the ram location of the A point>;
PUKCL_ZpEccQuickDualMulFast(pu1PointBBase) = <Base of the ram location of the B point>;
PUKCL_ZpEccQuickDualMulFast(pu1AWorkBase) = <Base of the ram location of the parameter A of
the elliptic curve and workspace>;
PUKCL_ZpEccQuickDualMulFast(pu1KABBase) = <Base of the ram location of the scalar numbers KA
and KB>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEccQuickDualMulFast, pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.6.6.7 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- pu1ModCnsBase,pu1PointABase, pu1PointBBase, pu1AWorkBase, pu1KABBase are not aligned on 32-bit boundaries
- {pu1ModCnsBase, 2*u2ModLength + 16}, {pu1PointABase, (3*(u2ModLength + 4)) *($2^{(WA-2)}$)}, {pu1PointBBase, (3*(u2ModLength + 4)) * ($2^{(WB-2)}$)} or { pu1AWorkBase, 9*u2ModLength + 48} are not in PUKCC RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- Alloverlapping between {pu1ModCnsBase, 2*u2ModLength + 16}, {pu1PointABase, (3*(u2ModLength + 4)) * ($2^{(WA-2)}$)}, {pu1PointBBase, (3*(u2ModLength + 4)) * ($2^{(WB-2)}$)} or {pu1AWorkBase, 9*u2ModLength + 48}.

37.3.6.6.8 Parameters Placement

The parameters' placement is described in the following figures.

Figure 37-7. Modulus P and Cns{pu1ModCnsBase, 2*u2ModLength + 16}

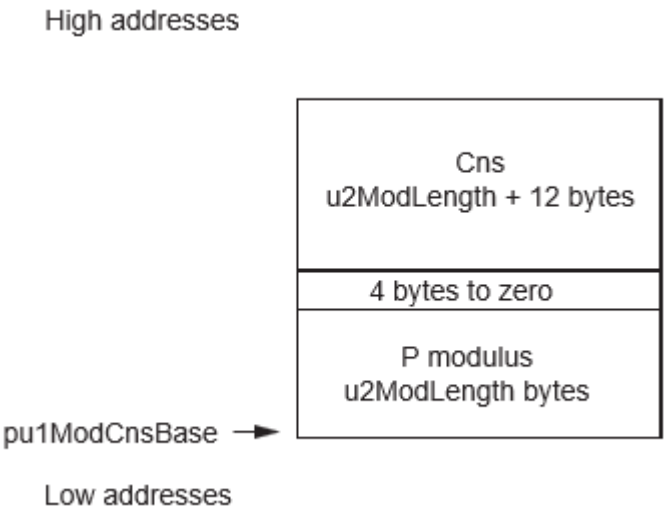


Figure 37-8. Points A and B {pu1PointABase, [(3*(u2ModLength + 4)) * ($2^{(WA-2)}$)] Or [(3*(u2ModLength + 4)) * ($2^{(WB-2)}$)]}

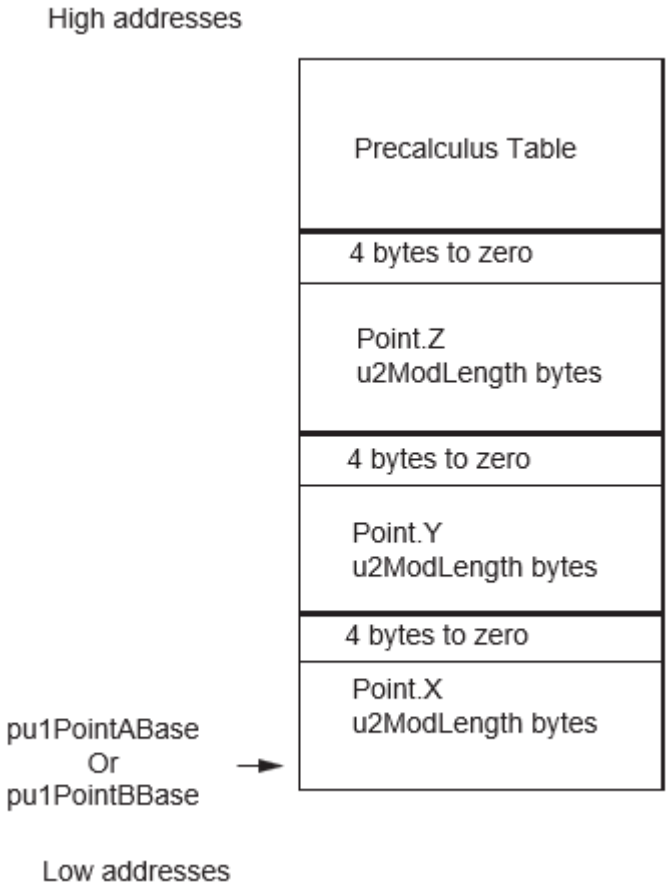


Figure 37-9. Scalars KA and KB {pu1KABBase, 2 * u2KLength + 8}

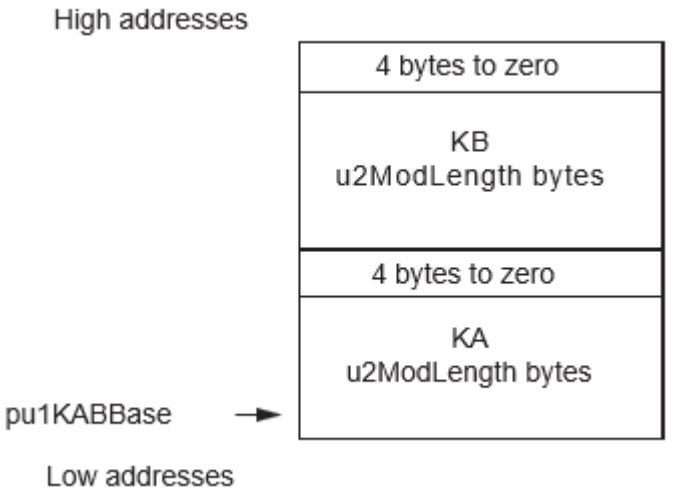
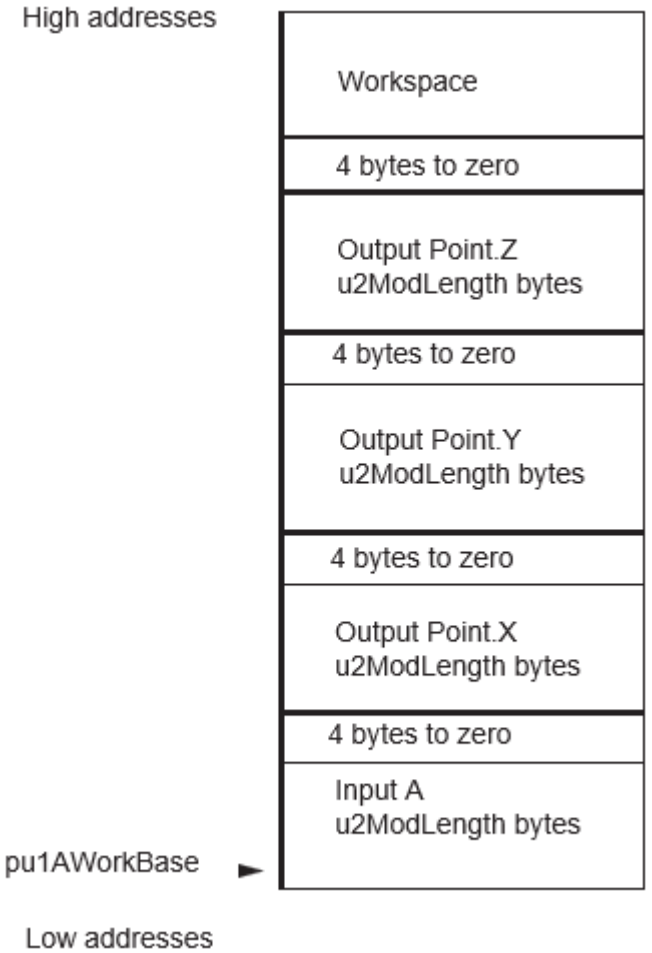


Figure 37-10. The a parameter and Workspace {pu1AWorkBase, 9*u2ModLength + 48}



37.3.6.6.9 Status Returned Values

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.6.7 Projective to Affine Coordinates Conversion

37.3.6.7.1 Purpose

This service is used to perform a point coordinates conversion from projective representation to affine.

37.3.6.7.2 How to Use the Service

37.3.6.7.3 Description

The operation performed is:

$$Pt_{X\ Affine\ \ coordinate} = \left[\frac{Pt_{X\text{Projective coordinate}}}{\left(Pt_{Z\ \text{Projective coordinate}}\right)2}\right]$$

$$Pt_{Y\ Affine\ \ coordinate} = \left[\frac{Pt_{Y\ \text{Projective coordinate}}}{\left(Pt_{Z\ \text{Projective coordinate}}\right)3}\right]$$

In this computation, the following parameters need to be provided:

- A the input point is filled in projective coordinates (X,Y,Z) or affine coordinates for X and Y, and setting Z to 1(pointed by {nu1PointABase,3*u2ModLength + 12}). The Point A can be the point at infinity. In this case, the u2Status returned is PUKCL_POINT_AT_INFINITY.
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength +8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength +4})
- The workspace not initialized (pointed by {nu1WorkSpace, 4*u2ModLength +48}

The result is the point A with its (X,Y) coordinates converted to affine, and the Z coordinate set to 1. The service for this operation is ZpEcConvProjToAffine.

Important: Before using this service, ensure that the constant Cns has been calculated with the Setup of the fast Modular Reductions service.

37.3.6.7.4 Parameters Definition

Table 37-77. ZpEccConvAffineToProjective Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1PointABase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point A	Resulting point A in affine coordinates
nu1Workspace	nu1	I	Crypto RAM	4*u2ModLength + 48	–	Workspace

37.3.6.7.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;


PUKCL (u2Option) = 0;
```

```
PUKCL _ZpEcConvProjToAffine(nu1ModBase) = <Base of the ram location of P>;
PUKCL _ZpEcConvProjToAffine(u2ModLength) = <Byte length of P>;
PUKCL _ZpEcConvProjToAffine(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _ZpEcConvProjToAffine(nu1PointABase) = <Base of the ram location of the A point>;
PUKCL _ZpEcConvProjToAffine(nu1Workspace) = <Base of the ram location of the workspace>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEcConvProjToAffine,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.6.7.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8},{nu1PointABase, 3*u2ModLength+ 12}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8},

{nu1PointABase, 3*u2ModLength + 12} and {nu1Workspace, 4*u2ModLength + 48}

37.3.6.7.7 Status Returned Values

Table 37-78. ZpEccConvAffineToProjective Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.
PUKCL_POINT_AT_INFINITY	Warning	The input point has its Z equal to zero, so it's a representation of the infinite point.

37.3.6.8 Affine to Projective Coordinates Conversion

37.3.6.8.1 Purpose

This service is used to perform a point coordinates conversion from an affine point representation to projective.

37.3.6.8.2 How to Use the Service

37.3.6.8.3 Description

The operation performed is:

affine(*Xa*, *Ya*) → *projective*(*Xp*, *Yp*, *Zp*)

In this computation, the following parameters need to be provided:

- A the input point is filled in affine coordinates for X and Y, and setting Z to 1 (pointed by {nu1PointABase,3*u2ModLength + 4}).
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength +8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength +4})
- The workspace not initialized (pointed by {nu1WorkSpace, 2*u2ModLength +16}

The result is the point A with its (X,Y,Z) projective coordinates.

The service for this operation is ZpEcConvAffineToProjective

Important: Before using this service, ensure that the constant Cns has been calculated with the setup of the Fast Modular Reductions service.

37.3.6.8.4 Parameters Definition

Table 37-79. ZpEccConvAffineToProjective Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1PointABase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point A	Resulting point A in affine coordinates
nu1Workspace	nu1	I	Crypto RAM	2*u2ModLength + 16	–	Workspace

37.3.6.8.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;


PUKCL (u2Option) = 0;

PUKCL _ZpEcConvAffineToProjective(nu1ModBase) = <Base of the ram location of P>;
PUKCL _ZpEcConvAffineToProjective(u2ModLength) = <Byte length of P>;
PUKCL _ZpEcConvAffineToProjective(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _ZpEcConvAffineToProjective(nu1PointABase) = <Base of the ram location of the A point>;
PUKCL _ZpEcConvAffineToProjective(nu1Workspace) = <Base of the ram location of the workspace>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEcConvAffineToProjective,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.6.8.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength+ 12}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8}, {nu1PointABase, 3*u2ModLength + 12}, and {nu1Workspace, 2*u2ModLength + 16}

37.3.6.8.7 Status Returned Values

Table 37-80. ZpEccConvAffineToProjective Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.6.9 Randomize a Coordinate

37.3.6.9.1 Purpose

This service is used to convert the projective representation of a point to another projective representation.

37.3.6.9.2 How to Use the Service

37.3.6.9.3 Description

The operation performed is:

Projective(X_1, Y_1, Z_1) → *Projective*(X_2, Y_2, Z_2)

In this computation, the following parameters need to be provided:

- The input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointBase,3*u2ModLength + 12}). This Point must not be the point at infinity.
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength +8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength +4})
- The workspace not initialized (pointed by {nu1WorkSpace, 3*u2ModLength +28}
- The random number (pointed by {nu1RandomBase, u2ModLength +4}).

The result is the point nu1PointBase with its (X,Y,Z) coordinates randomized.

The service for this operation is ZpEcRandomiseCoordinate.

Important: Before using this service:

- Ensure that the constant Cns has been calculated with the setup of the Modular Reduction service.
- Be sure to follow the directives given for the RNG on the chip you use (particularly initialization, seeding) and compulsorily start the RNG

.

37.3.6.9.4 Parameters Definition

Table 37-81. ZpEccRandomiseCoordinate Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1PointBase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point	Resulting point
nu1RandomBase	nu1	I	Crypto RAM	u2ModLength + 4	Random	Corrupted
nu1Workspace	nu1	I	Crypto RAM	3*u2ModLength + 28	–	Workspace

37.3.6.9.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
// ! following the directives given for the RNG on the chip

PUKCL (u2Option) = 0;

// Depending on the option specified, not all fields must be filled
PUKCL _ZpEccRandomiseCoordinate(nu1ModBase) = <Base of the ram location of P>;
PUKCL _ZpEccRandomiseCoordinate(u2ModLength) = <Byte length of P>;
```

```
PUKCL _ZpEccRandomiseCoordinate(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL_ZpEccRandomiseCoordinate(nu1RandomBase) = <Base of the ram location where the the RNG
is stored>;
PUKCL _ZpEccRandomiseCoordinate(nu1PointBase) = <Base of the ram location of the point>;
PUKCL _ZpEccRandomiseCoordinate(nu1Workspace) = <Base of the ram location of the workspace>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEccRandomiseCoordinate,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.6.9.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1RandomBase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1RandomBase, u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1RandomBase, u2ModLength + 4} and {nu1Workspace, 3*u2ModLength + 28}

37.3.6.9.7 Status Returned Values

Table 37-82. ZpEccRandomiseCoordinate Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.6.10 Point is on Elliptic Curve

37.3.6.10.1 Purpose

This service is used to test whether or not the point is on the curve.

37.3.6.10.2 How to Use the Service

37.3.6.10.3 Description

The operation performed is:

Status = *IsPointOnCurve*(*X*, *Y*, *Z*)

In this computation, the following parameters need to be provided:

- The input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointBase,3*u2ModLength + 4}). This Point can be the point at infinity.
- AParam and BParam are the Elliptic Curve Equation parameters. (pointed by{nu1AParam, u2ModLength+4} and {nu1BParam, u2ModLength+4}).
- Cns the Fast Modular Constant filled (pointed by{nu1CnsBase,u2ModLength+8}).
- P the modulus filled (pointed by {nu1ModBase,u2ModLength +4}).
- The workspace not initialized (pointed by {nu1WorkSpace, 4*u2ModLength +28}.

The result is the status of the point (X,Y,Z) regarding the Elliptic Curve Equation.

The service name for this operation is `ZpEcPointIsOnCurve`.

Note: Before using this service, ensure that the constant Cns has been calculated with the setup of the Fast Modular Reduction service.

37.3.6.10.4 Parameters Definition

Table 37-83. ZpEcPointIsOnCurve Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1PointBase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point	unchanged
nu1AParam	nu1	I	Crypto RAM	u2ModLength + 4	The parameter a	The parameter a
nu1BParam	nu1	I	Crypto RAM	u2ModLength + 4	The parameter b	The parameter b
nu1Workspace	nu1	I	Crypto RAM	4*u2ModLength + 28	–	Workspace

37.3.6.10.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;


PUKCL (u2Option) = 0;

PUKCL _ZpEcPointIsOnCurve(nu1ModBase) = <Base of the ram location of P>;
PUKCL _ZpEcPointIsOnCurve(u2ModLength) = <Byte length of P>;
PUKCL _ZpEcPointIsOnCurve(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _ZpEcPointIsOnCurve(nu1AParam) = <Base of the ram location of the parameter a>;
PUKCL _ZpEcPointIsOnCurve(nu1BParam) = <Base of the ram location of the parameter b>;
PUKCL _ZpEcPointIsOnCurve(nu1PointBase) = <Base of the ram location of the point>;
PUKCL _ZpEcPointIsOnCurve(nu1Workspace) = <Base of the ram location of the workspace>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEcPointIsOnCurve,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.6.10.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1AParam, nu1BParam, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength+4}, {nu1CnsBase, u2ModLength+8}, {nu1PointABase, 3*u2ModLength +12}, {nu1AParam, u2ModLength + 4}, {nu1BParam, u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM.
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length.
- All overlapping between {nu1ModBase, u2ModLength+4}, {nu1CnsBase,u2ModLength+8}, {nu1PointABase, 3*u2ModLength+12}, {nu1AParam, u2ModLength+4}, {nu1AParam, u2ModLength + 4} and {nu1Workspace, 4*u2ModLength+28}.

37.3.6.10.7 Status Returned Values

Table 37-84. ZpEcPointIsOnCurve Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The point is on the curve.
PUKCL_POINT_IS_NOT_ON_ CURVE	Warning	The point is not on the curve.

.....continued		
Returned Status	**Importance**	**Meaning**
PUKCL_POINT_AT_INFINITY	Warning	The input point has its Z equal to zero, so it's a representation of the infinite point.

37.3.6.11 Generating an ECDSA Signature (Compliant with FIPS 186-2)

37.3.6.11.1 Purpose

This service is used to generate an ECDSA signature following the FIPS 186-2. It performs the second step of the Signature Generation. A hash value *(HashVal)* must be provided as input, it has to be previously computed from the message to be signed using a secure hash algorithm.

A scalar number must be provided too as described in the FIPS 186-2. The result (R,S) is computed by this service.

37.3.6.11.2 How to Use the Service

37.3.6.11.3 Description

The operation performed is:

(*R*, *S*) = *EcDsaSign*(Pt_A, *HashVal*, *k*, *CurveParameters*, *PrivateKey*)

This service processes the following checks:

- If the Scalar Number k is out of the range [1, PointOrder -1], the calculus is stopped and the status is set to PUKCL_WRONG_SELECT_NUMBER.
- If R equals zero, the calculus is stopped and the status is set to PUKCL_WRONG_SELECT_NUMBER.
- If S equals zero, the calculus is stopped and the status is set to PUKCL_WRONG_SELECT_NUMBER.

In this computation, the following parameters need to be provided:

- A the input point is filled in "mixed" coordinates (X,Y) with the affine values and Z = 1 (pointed by {nu1PointABase,3*u2ModLength + 12})
- Cns the working space for the Fast Modular Constant not initialized (pointed by {nu1CnsBase,u2ScalarLength + 8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength + 4})
- The workspace not initialized (pointed by {nu1WorkSpace, 8*u2ModLength + 44}
- The a parameter relative to the elliptic curve (pointed by {nu1ABase, u2ModLength + 4})
- The order of the Point A on the elliptic curve (pointed by {nu1OrderPointBase, u2ScalarLength + 4})
- k the input Scalar Number beforehand generated and filled (pointed by{nu1ScalarNumber,u2ScalarLength + 4})
- HashVal the hash value beforehand generated and filled (pointed by {nu1HashBase, u2ScalarLength + 4})
- The Private Key (pointed by {nu1PrivateKey, u2ScalarLength +4})
- Generally, u2ScalarLength is equal to (u2ModLength) or (u2ModLength + 4)

Important:
For the ECDSA signature generation be sure to follow the directives given for the RNG on the chip you use (particularly initialization, seeding) and compulsorily start the RNG.

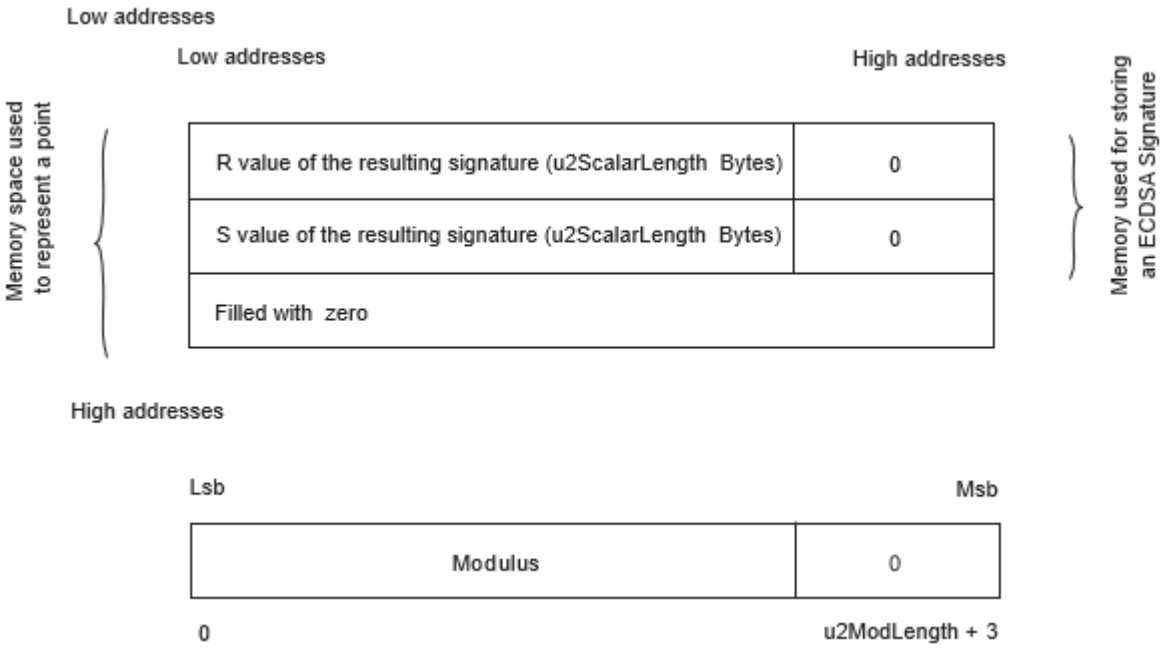
The scalar number k must be selected at random. This random must be generated before the call of the ECDSA signature. For this random generation be sure to follow the directives given for the RNG on the chip you use (particularly initialization, seeding) and compulsorily start the RNG.

The operation performed is:

- Compute the ECDSA (R,S) as described in FIPS 186-2, but leaving the user the role of computing the input Hash Value, thus leaving the freedom of using any other algorithm than SHA-1.
- Compute a R value using the input A point and the scalar number.
- Compute a S value using R, the scalar number, the private key and the provided hash value. Note that the resulting signature (R,S) is stored at the place of the input A point.
- If all is correct and S is different from zero, the status is set to PUKCL_OK. If all is correct and S equals zero,the status is set to PUKCL_WRONG_SELECT_NUMBER. If an error occurs, the status is set to the corresponding error value (see Status Returned Values below).

The service name for this operation is `ZpEcDsaGenerateFast`. This service uses Fast mode and Fast Modular Reduction for computation.

- The signature (R,S), when resulting from a computation is given back at address of the A point:
  - R output is at offset 0 and has length (u2ScalarLength + 4)bytes.
  - S output is at offset (u2ScalarLength + 4) bytes and has length (u2ScalarLength + 4) bytes.
  - The MSB 4 zero bytes may be suppressed to get the R and S values on u2ScalarLength bytes



37.3.6.11.4 Parameters Definition

Table 37-85. ZpEcDsaGenerateFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ScalarLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1ScalarNumber	nu1	I	Crypto RAM	u2ScalarLength + 4	Scalar Number used to multiply the point A	Unchanged
nu1OrderPointBase	nu1	I	Crypto RAM	u2ScalarLength + 4	Order of the Point A in the elliptic curve	Unchanged
nu1PrivateKey	nu1	I/O	Crypto RAM	u2ScalarLength + 4	Base of the Private Key	Unchanged
nu1HashBase(1)	nu1	I	Crypto RAM	u2ScalarLength + 4	Base of the hash value resulting from the previous SHA	Unchanged
u2ScalarLength	u2	I	–	–	Length of scalar (same length as the length of order)	Length of scalar
nu1PointABase(2)	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Input point A (three coordinates (X,Y) affine and Z = 1)	Resulting signature (R,S,0)
nu1ABase	nu1	I	Crypto RAM	u2ModLength + 4	Parameter a of the elliptic curve	Unchanged
nu1Workspace	nu1	I	Crypto RAM	8*u2ModLength + 44	–	Corrupted workspace

Notes:

1. The hash value calculus is defined by the ECDSA norm and depends on the elliptic curve domain parameters. To construct the input parameter, the 4 Most Significant Bytes must be set to zero.
2. The resulting signature format is different from the point A format (see Description above for information on the point A format).

37.3.6.11.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
// ! following the directives given for the RNG on the chip


PUKCL (u2Option) = 0;

// Depending on the option specified, not all fields must be filled
PUKCL _ZpEcDsaGenerate(nu1ModBase) = <Base of the ram location of P>; PUKCL
_ZpEcDsaGenerate(u2ModLength) = <Byte length of P>;
PUKCL _ZpEcDsaGenerate(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _ZpEcDsaGenerate(nu1PointABase) = <Base of the A point>;
PUKCL _ZpEcDsaGenerate(nu1PrivateKey) = <Base of the Private Key>;
PUKCL _ZpEcDsaGenerate(nu1ScalarNumber) = <Base of the ScalarNumber>;
PUKCL _ZpEcDsaGenerate(nu1OrderPointBase) = <Base of the order of A point>;
PUKCL _ZpEcDsaGenerate(nu1ABase) = <Base of the a parameter of the curve>;
PUKCL _ZpEcDsaGenerate(nu1Workspace) = <Base of the workspace>;
PUKCL _ZpEcDsaGenerate(nu1HashBase) = <Base of the SHA resulting hash>;
PUKCL_ZpEcDsaGenerate(u2ScalarLength)    = < Length of ScalarNumber>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEcDsaGenerateFast, pvPUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
```

```
            }
else // Manage the error
```

37.3.6.11.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1PrivateKey, nu1ScalarNumber, nu1OrderPointBase,nu1ABase, nu1Workspace or nu1HashBase are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength+ 12},{nu1PrivateKey, u2ScalarLength + 4},{nu1ScalarNumber, u2ScalarLength + 4},{nu1OrderPointBase, u2ScalarLength + 4}, {nu1ABase, u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} or {nu1HashBase, u2ScalarLength + 4} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1PrivateKey, u2ScalarLength + 4}, {nu1ScalarNumber, u2ScalarLength + 4}, {nu1OrderPointBase, u2ScalarLength + 4}, {nu1ABase, u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} and {nu1HashBase, u2ScalarLength + 4}

37.3.6.11.7 Status Returned Values

Table 37-86. ZpEcDsaGenerateFast Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem. The signature is the good one.
PUKCL_WRONG_SELECTNUMBER	Warning	The given value for nu1ScalarNumber is not good to perform this signature generation.

37.3.6.12 Verifying an ECDSA Signature (Compliant with FIPS186-2)

37.3.6.12.1 Purpose

This service is used to verify an ECDSA signature following the FIPS 186-2. It performs the second step of the Signature Verification.

A hash value (HashVal) must be provided as input, it has to be previously computed from the message to be signed using a secure hash algorithm.

As second significant input, the Signature is provided to be checked. This service checks the signature and fills the status accordingly.

37.3.6.12.2 How to Use the Service

37.3.6.12.3 Description

The operation performed is:

Verify = *EcDsaVerifySignature*(Pt_A, *HashVal*, *Signature*, *CurveParameters*, *PublicKey*)

The points used for this operation are represented in different coordinate systems. In this computation, the following parameters need to be provided:

- A the input point is filled with the affine values (X,Y) and Z = 1 (pointed by{nu1PointABase,3*u2ModLength + 12})
- Cns the working space for the Fast Modular Constant not initialized (pointed by {nu1CnsBase,u2ScalarLength + 8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength + 4})
- The workspace not initialized (pointed by {nu1WorkSpace, 8*u2ModLength + 44}
- The a parameter relative to the elliptic curve (pointed by {nu1ABase,u2ModLength + 4})

- The order of the Point A on the elliptic curve (pointed by {nu1OrderPointBase,u2ScalarLength + 4})
- HashVal the hash value is generated prior and filled (pointed by {nu1HashBase,u2ScalarLength + 4})
- The Public Key point is filled in “mixed” coordinates (X,Y) with the affine values and Z = 1 (pointed by {nu1PointPublicKeyGen, 3*u2ModLength + 12})
- The input signature (R,S), even if it is not a Point, is represented in memory like a point in affine coordinates (X,Y) (pointed by {nu1PointSignature, 2*u2ScalarLength + 8})
  Note: For the ECDSA signature verification be sure to follow the directives given for the RNG on the chip you use (particularly initialization, seeding) and compulsorily start the RNG.
- The operation consists in obtaining a V value with all these input parameters and checking that V equals the provided R. If all is correct and the signature is the good one, the status is set to PUKCL_OK. If all is correct and the signature is wrong, the status is set to PUKCL_WRONG_SIGNATURE. If an error occurs, the status is set to the corresponding error value (see Status Returned Values below).

37.3.6.12.4 Parameters Definition

Table 37-87. ZpEcDsaVerifyFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ScalarLength + 12	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1OrderPointBase	nu1	I	Crypto RAM	u2ScalarLength + 4	Order of the Point A in the elliptic curve	Unchanged
nu1PointSignature	nu1	I	Crypto RAM	2*u2ScalarLength + 8	Signature(r, s)	Corrupted
nu1HashBase(1)	nu1	I	Crypto RAM	u2ScalarLength + 4	Base of the hash value resulting from the previous SHA	Corrupted
u2ScalarLength	u2	I	–	–	Length of scalar	Length of scalar
nu1PointABase	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Generator point	Corrupted
nu1PointPublicKeyGen	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Public point	Corrupted
nu1ABase	nu1	I	Crypto RAM	u2ModLength + 4	Parameter a of the elliptic curve	Unchanged
nu1Workspace	nu1	I	Crypto RAM	8*u2ModLength + 44	–	Corrupted workspace

Note:

1. The hash value calculus is defined by the ECDSA norm and depends on the elliptic curve domain parameters. To construct the input parameter, the 4 Most Significant Bytes must be set to zero.

37.3.6.12.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
// ! following the directives given for the RNG on the chip

PUKCL(u2Option) = 0;

// Depending on the option specified, not all fields must be filled
PUKCL_ZpEcDsaVerify(nu1ModBase) = <Base of the ram location of P>;
PUKCL_ZpEcDsaVerify(u2ModLength) = <Byte length of P>;
PUKCL_ZpEcDsaVerify(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL_ZpEcDsaVerify(nu1PointABase) = <Base of the A point>;
```

```
PUKCL_ZpEcDsaVerify(nu1PrivateKey) = <Base of the Private Key>;
PUKCL_ZpEcDsaVerify(nu1ScalarNumber) = <Base of the ScalarNumber>;
PUKCL_ZpEcDsaVerify(nu1OrderPointBase) = <Base of the order of A point>;
PUKCL_ZpEcDsaVerify(nu1ABase) = <Base of the a parameter of the curve>;
PUKCL_ZpEcDsaVerify(nu1Workspace) = <Base of the workspace>;
PUKCL_ZpEcDsaVerify(nu1HashBase) = <Base of the SHA resulting hash>;
 PUKCL_ZpEcDsaVerify(u2ScalarLength)    = < Length of ScalarNumber>;

...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEcDsaVerifyFast, pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
            {
            ...
            }ou
else
            if(PUKCL(u2Status) == PUKCL_WRONG_SIGNATURE)
                        {
                        ...
                        }
            else // Manage the error
```

37.3.6.12.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1PointPublicKeyGen, nu1PointSignature, nu1OrderPointBase,nu1ABase, nu1Workspace or nu1HashBase are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength+ 12}, {nu1PointPublicKeyGen, 3*u2ModLength + 12}, {nu1PointSignature,2*u2ScalarLength + 8}, {nu1OrderPointBase, u2ScalarLength + 4}, {nu1ABase, u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} or {nu1HashBase, u2ScalarLength + 4} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1PointPublicKeyGen, 3*u2ModLength + 12}, {nu1PointSignature, 2*u2ScalarLength + 8}, {nu1OrderPointBase, u2ScalarLength + 4}, {nu1ABase, u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} and {nu1HashBase, u2ScalarLength + 4}

37.3.6.12.7 Status Returned Values

Table 37-88. ZpEcDsaVerifyFast Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem. The signature is the good one.
PUKCL_WRONG_SIGNATURE	Warning	The signature is wrong.

37.3.6.13 Quick Verifying an ECDSA Signature (Compliant with FIPS 186-2)

37.3.6.13.1 Purpose

This service is used to verify an ECDSA signature following the FIPS 186-2. It performs the second step of the Signature Verification using Quick Dual Multiplying to perform computation.

A hash value (HashVal) must be provided as input, it has to be previously computed from the message whose signature is verified using a secure hash algorithm.

As second significant input, the Signature is provided to be checked.

This service checks the signature and fills the status accordingly.

Important: This service has a quick implementation without additional security.

37.3.6.13.2 How to Use the Service

37.3.6.13.3 Description

The operation performed is:

Verify = *EcDsaVerifySignature*(Pt_A, *HashVal*, *Signature*, *CurveParameters*, *PublicKey*)

The points used for this operation are represented in different coordinate systems.

In this computation, the following parameters need to be provided (such that u2MaxLength = max(u2ModLength, u2ScalarLength)):

- A the input point is filled with the affine values (X,Y) and Z = 1 (pointed by {pu1PointABase, (3*(u2ModLength + 4)) * ($2^{(WA-2)}$)})
- P the modulus filled and Cns the working space for the Fast Modular Constant not initialized (pointed by {pu1ModBase, u2ModLength + u2MaxLength + 16})
- The a parameter relative to the elliptic curve filled and workspace not initialized (pointed by {pu1AWorkBase,8*u2MaxLength + u2ModLength + 48})
- The order of the Point A on the elliptic curve (pointed by {pu1OrderPointBase,u2ScalarLength +4})
- HashVal the hash value beforehand generated and filled (pointed by {pu1HashBase,u2MaxLength +4})
- The Public Key point is filled in “mixed” coordinates (X,Y) with the affine values and Z = 1 (pointed by {nu1PointPublicKeyGen, (3*(u2ModLength + 4)) * ($2^{(WB-2)}$)})
- The input signature (R,S), even if it is not a Point, is represented in memory like a point in affine coordinates (X,Y) (pointed by {nu1PointSignature, 2*u2ScalarLength + 8})

The operation consists of obtaining a V value with all input parameters and checks that V equals the provided R. If all is correct and the signature is the good one, the status is set to PUKCL_OK. If all is correct and the signature is wrong, the status is set to PUKCL_WRONG_SIGNATURE. If an error occurs, the status is set to the corresponding error value (see Status Returned Values below).

37.3.6.13.4 Parameters Definition

To place the parameters correctly the maximum of u2ModLength and u2ScalarLength must be calculated: u2MaxLength = max(u2ModLength, u2ScalarLength)

WA is the Point A window size and WB is the Point Public Key window size (see Options below for details).

Important: Please calculate precisely the length of areas with the formulas and the `max()` service which takes the maximum of two values. Ensure that the pu1 type is a pointer on 4 bytes and contains the full address (see 37.3.3.4. Aligned Significant Length for details).

Table 37-89. ZpEcDsaQuickVerify Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
pu1ModCnsBase	pu1	I	Crypto RAM	u2ModLength + 4 + u2MaxLength + 12	Base of modulus P	Base of modulus P

.....continued

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
u2Option	u2	I	–	–	Option related to the called service (see below)	–
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
pu1OrderPointBase	pu1	I	Crypto RAM	u2ScalarLength + 4	Order of the Point A in the elliptic curve	Unchanged
pu1PointSignature	pu1	I	Any RAM	2*u2ScalarLength + 8	Signature(r, s)	Corrupted
pu1HashBase (see **Note 1**)	pu1	I	Crypto RAM	u2MaxLength + 4	Base of the hash value resulting from the previous SHA	Corrupted
u2ScalarLength	u2	I	–	–	Length of scalar	Length of scalar
pu1PointABase	pu1	I/O	Crypto RAM	(3*u2ModLength + 12) * (2(WA-2))	Generator point	Corrupted
pu1PointPublicKeyGen	pu1	I/O	Crypto RAM	(3*u2ModLength + 12) * (2(WB-2))	Public Key point	Corrupted
pu1AWorkBase	pu1	I	Crypto RAM	(u2ModLength + 4) + (8*u2MaxLength + 44)	Parameter a of the elliptic curve and Workspace	Corrupted

Note:

1. 1. The hash value calculus is defined by the ECDSA norm and depends on the elliptic curve domain parameters. To construct the input parameter, the 4 Most Significant Bytes must be set to zero.

A suggested parameters placement in Crypto RAM is:

- ModCnsBase
- OrderPointBase
- Signature may be placed here or in Classical RAM
- HashBase
- PointABase
- PointPublicKeyGen
- AWorkBase

37.3.6.13.5 Options

The options are set by the u2Options input parameter, which is composed of:

- the mandatory windows sizes WA (window for Point A) and WB (window for Point Public Key)
- the indication of the presence of the Point Signature in system RAM

Important: Please check precisely if the Point Signature is in Crypto RAM. If this is the case the PUKCL_ZPECCMUL_SCAL_IN_CLASSIC_RAM must not be used.

The u2Options number is calculated by an “Inclusive OR” of the options. Some Examples in C language are:

- ```
  // Point Signature in system RAM
  // The Point A window size is 3
  // The Point Public Key window size is 4
  ```
  ```

```
PUKCL(u2Options) = PUKCL_ZPECCMUL_SCAL_IN_CLASSIC_RAM |
PUKCL_ZPECCMUL_WINSIZE_A_VAL_TO_OPT(3) |
PUKCL_ZPECCMUL_WINSIZE_B_VAL_TO_OPT(4);
```

- ```
  // Point Signature in the Cryptographic RAM
  // The Point A window size is 2
  // The Point Public Key window size is 5
  PUKCL(u2Options) = PUKCL_ZPECCMUL_WINSIZE_A_VAL_TO_OPT(2) |
  PUKCL_ZPECCMUL_WINSIZE_B_VAL_TO_OPT(5);
  ```

For this service, many window sizes are possible. The window sizes in bits are those of the windowing method used for the scalar multiplying.

The choice of the window sizes is a balance between the size of the parameters and the computation time:

- Increasing the window size increases the precomputation table size.
- Increasing the window size to the optimum reduces the computation time.

The following table details the estimated windows WA and WB optimum and possible for some curves.

Table 37-90. ZpEcDsaQuickVerify Service Estimated WA and WB Window Size

Curve Size (bits)	Optimum Window size	Possible Window Sizes (WA, WB) or (WB, WA)
192	5	5, 5
256	5	5, 5
384	6	5, 5
521	6	4, 5

The following table details the size of the point and the precomputation table, depending on the chosen window size option.

Table 37-91. ZpEcDsaQuickVerify Service Window Size and Precomputation Table Size Options

Option Specified	Point and Precomputation Table Size
PUKCL_ZPECCMUL_WINSIZE_A_VAL_TO_OPT(WA) WA in [2, 15]	(3*(u2ModLength + 4)) * ($2^{(WA-2)}$)
PUKCL_ZPECCMUL_WINSIZE_B_VAL_TO_OPT(WB) WB in [2, 15]	(3*(u2ModLength + 4)) * ($2^{(WB-2)}$)

The Point Signature can be located in PUKCC RAM or in system RAM. If the Point Signature is entirely in system RAM with no part in PUKCC RAM this can be signaled by us ing the option PUKCL_ZPECCMUL_SCAL_IN_CLASSIC_RAM. In all other cases this option must not be used.

The following table describes this option.

Table 37-92. ZpEcDsaQuickVerify Service Point Signature in Classical RAM Option

Option	Purpose
PUKCL_ZPECCMUL_SCAL_IN_CLASSIC_RAM	The Point Signature can be located in Crypto RAM or in system RAM. If the Point Signature is entirely in system RAM with no part in PUKCC RAM this can be signaled by using this option. In all other cases this option must not be used.

37.3.6.13.6 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;
PUKCL(u2Option) = <Point Signature location and windows sizes>;
PUKCL_ZpEcDsaQuickVerify(pu1ModCnsBase) = <Base of the ram location of P and Cns>;
PUKCL_ZpEcDsaQuickVerify(u2ModLength) = <Byte length of P>;
PUKCL_ZpEcDsaQuickVerify(pu1PointABase) = <Base of the ram location of the A point>;
PUKCL_ZpEcDsaQuickVerify(pu1PointPublicKeyGen) = <Base of the Public Key>;
PUKCL_ZpEcDsaQuickVerify(pu1PointSignature) = <Base of the Signature (r, s)>;
```
```

```
PUKCL_ZpEcDsaQuickVerify(pu1OrderPointBase) = <Base of the order of the A point>;
PUKCL_ZpEcDsaQuickVerify(pu1AWorkBase) = <Base of the ram location of the parameter A of the
elliptic curve and workspace>;
PUKCL_ZpEcDsaQuickVerify(pu1HashBase) = <Base of the SHA resulting hash>;
PUKCL_ZpEcDsaQuickVerify(u2ScalarLength) = <Byte length of R and S in Point Signature>;
. . .
// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(ZpEcDsaQuickVerify, pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
                {
                ...
                }
else
                if ( PUKCL(u2Status) = PUKCL_WRONG_SIGNATURE )
                                {
                                ...
                                }
                else // Manage the error
```

37.3.6.13.7 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- pu1ModCnsBase, pu1PointABase, pu1PointPublicKeyGen, pu1PointSignature,pu1OrderPointBase, pu1AWorkBase or pu1HashBase are not aligned on 32-bit boundaries
- {pu1ModCnsBase, u2ModLength + 4 + u2MaxLength + 12}, {pu1PointABase, (3 * u2ModLength + 12)* ($2^{(WA-2)}$)}, {pu1PointPublicKeyGen, (3 * u2ModLength + 12) * ($2^{(WPub-2)}$)}, {pu1OrderPointBase, u2ScalarLength + 4}, {nu1ABase, u2ModLength + 4}, {pu1AWorkBase, (u2ModLength + 4) + (8 * u2MaxLength + 44)} or {nu1HashBase, u2ScalarLength + 4} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {pu1ModCnsBase, u2ModLength + 4 + u2MaxLength + 12}, {pu1PointABase, (3 * u2ModLength + 12) * ($2^{(WA-2)}$)}, {pu1PointPublicKeyGen, (3 * u2ModLength + 12) *($2^{(WPub-2)}$)}, {pu1OrderPointBase, u2ScalarLength + 4}, {pu1PointSignature, 2 * u2ScalarLength + 8}, {nu1ABase, u2ModLength + 4}, {pu1AWorkBase, (u2ModLength + 4) + (8 * u2MaxLength + 44)} and {nu1HashBase, u2ScalarLength + 4}

37.3.6.13.8 Status Returned Values

Table 37-93. ZpEcDsaQuickVerify Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem. The signature is the good one.
PUKCL_WRONG_SIGNATURE	Warning	The signature is wrong.

37.3.6.13.9 Parameter Placement

The parameters' placement is described in detail in the following figures.

Figure 37-11. Modulus P and Cns{pu1ModCnsBase, u2ModLength + 4 + u2MaxLength + 12}

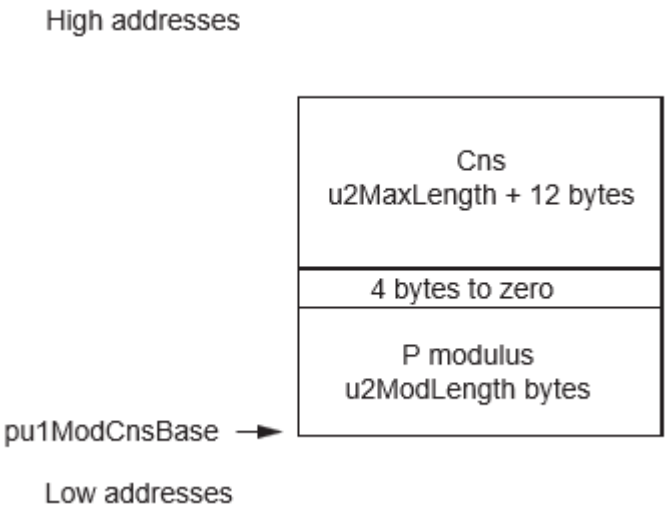


Figure 37-12. Points A {pu1PointABase, (3*(u2ModLength + 4)) * ($2^{(WA-2)}$)} and Public Key Gen {pu1PointPublicKeyGen, (3*(u2ModLength + 4)) * ($2^{(WB-2)}$)}

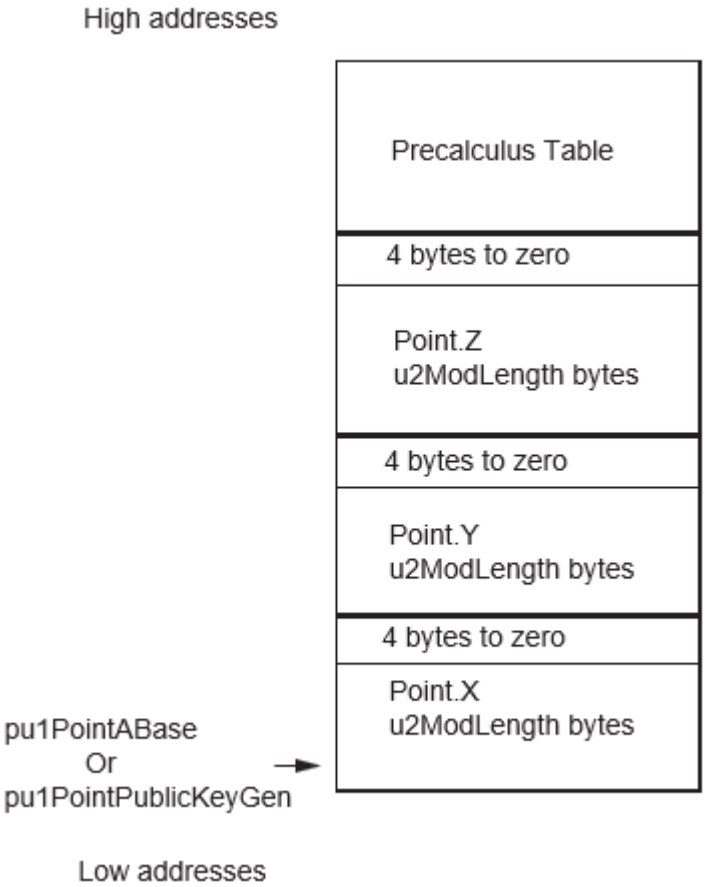


Figure 37-13. PointSignature {pu1PointSignature, 2 * u2ScalarLength + 8}

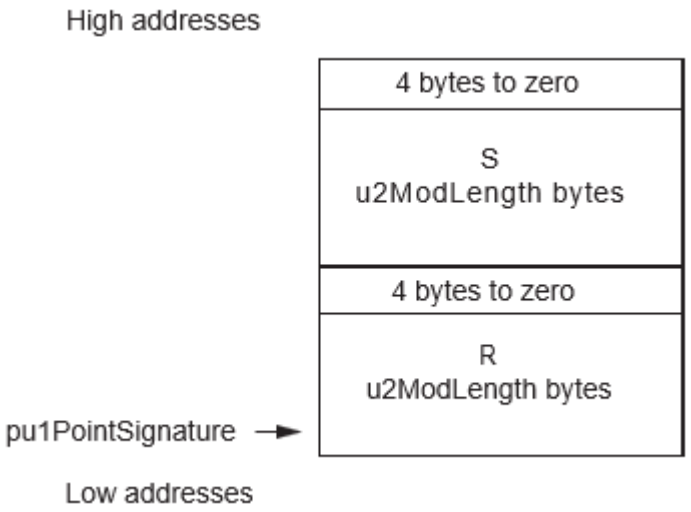
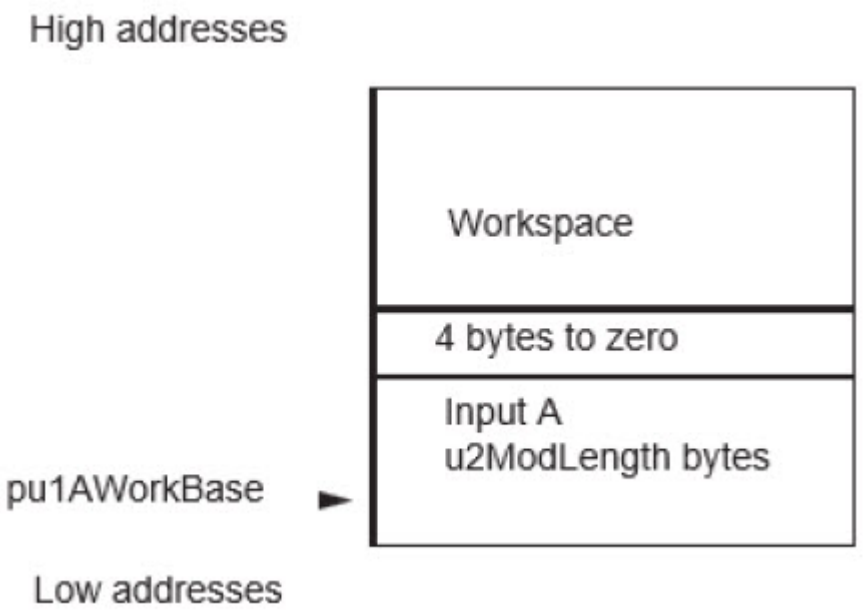


Figure 37-14. The a parameter and Workspace {pu1AWorkBase, 9*u2ModLength + 48}



37.3.7 Elliptic Curves Over GF(2^n) Services

This section provides a complete description of the currently available elliptic curve over Polynomials in GF(2^n) services.

These services process Polynomials in GF(2^n) only.

The offered services cover the basic operations over elliptic curves such as:

- Adding two points over a curve
- Doubling a point over a curve
- Multiplying a point by an integral constant
- Converting a point's projective coordinates (resulting from a doubling or an addition) to the affine coordinates, and oppositely converting a point's affine coordinates to the projective coordinates.

- Testing the point presence on the curve.

Additionally, some higher level services covering the needs for signature generation and verification are offered:

- Generating an ECDSA signature (compliant with FIPS186-2)
- Verifying an ECDSA signature (compliant with FIPS 186-2) The supported curves use the following curve equation in GF(2^n):

$Y^2 + XY = X^3 + aX + b$

37.3.7.1 Parameters Format

Related Links

37.3.5.1. Modular Reduction
37.3.3.4. Aligned Significant Length

37.3.7.1.1 Polynomials in GF(2^n)

Polynomials in GF(2^n) are binary polynomials reduced modulo the polynomial P[X]. This polynomial is called the modulus and may be abbreviated to P in this document. The storage of these polynomials in memory area is described in Aligned Significant Length (see *Aligned Significant Length* from Related Links).

For notation simplicity the comparison signs "<" or ">" may be used for polynomials, this is to be interpreted as a comparison between the degree of the polynomials.

In GF(2^n) fully reduced polynomials are of degree strictly lower than degree(P[X]). In many cases the polynomials used in this library are only partially reduced and so have a degree higher or equal than degree(P[X]), but this degree is maintained strictly lower than (degree(P[X]) + 15).

37.3.7.1.2 Coordinates System

In this implementation, several choices have been made related to the coordinate systems managed by the elliptic curve primitives.

There are two systems currently managed by the library:

- Affine Coordinates System where each curve point has two coordinates (X,Y)
- Projective Coordinates System where each point is represented with three coordinates (X,Y,Z)

Converting from the affine coordinates system to a projective coordinates system and is performed by extending its representation having Z = 1:

$(X,Y) \Rightarrow (X,Y, Z= 1)$

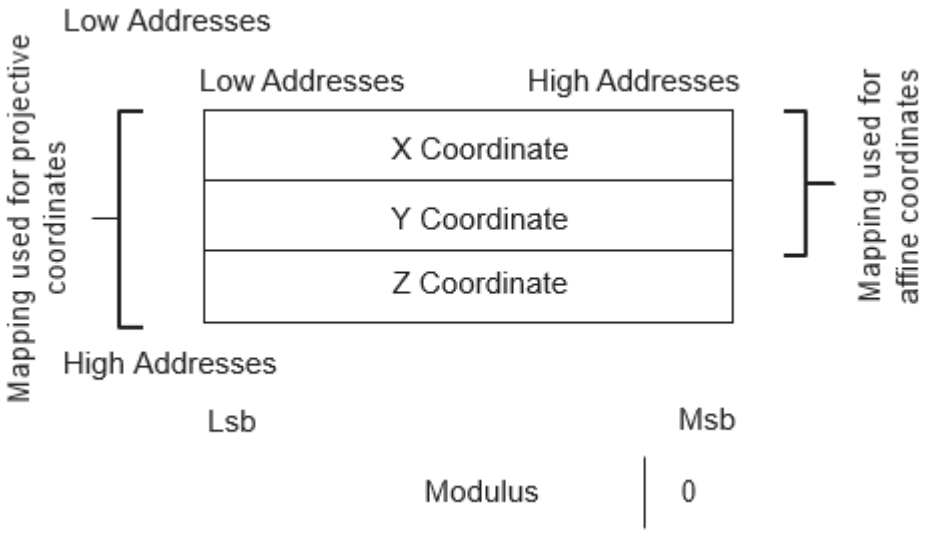
Converting from a projective coordinate to an affine one is a service offered by the library. The formula to perform this conversion is:

$(X,Y, Z) \Rightarrow (X \Rightarrow Z, Y/Z^2)$

37.3.7.1.3 Points Representation in Memory

Depending on the representation (Projective or Affine), points are represented in memory as shown in the following figure.

Figure 37-15. Point Representation in Memory



In this figure, the modulus is represented as a reference, and to show that coordinates are always to be provided on the length of the modulus plus one 32-bit word.

Different types of representations are listed here:

Affine representation: $Pt = \begin{bmatrix} X_{Affine < P \times X^{15}} \\ Y_{Affine < P \times X^{15}} \end{bmatrix}$

Projective representation: $Pt = \begin{bmatrix} X_{\mathrm{Projective} < P \times X^{15}} \\ Y_{\mathrm{Projective} < P \times X^{15}} \\ Z_{\mathrm{Projective} < P \times X^{15}} \end{bmatrix}$

Notes:

1. The minimum value for u2ModLength is 12 bytes. Therefore, the significant length of the modulus must be at least three 32-bit words.
2. In some cases the point can be the infinite point. In this case it is represented with its Z coordinates equal or congruent to zero.

37.3.7.1.4 Modulus and Modular Constant Parameters

In most of the services the following parameters must be provided:

- P the Modulus (often pointed by {nu1ModBase,u2ModLength + 4}): This parameter contains the Modulus Polynomial P[X] defining the Galois Field used in points coordinates computations. The Modulus must be u2ModLength bytes long, while having a supplemental zeroed 32-bit word on the MSB side.
  Note: Most of the Elliptic Curve computations are reduced modulo P. In many functions the reductions are made with the Fast Reduction.
- Cns the Modular Constant (often pointed by {nu1CnsBase,u2ModLength + 12}): This parameter contains the Modular Constant associated to the Modulus.

Important: The Modular Constant must be calculated before using the GF(2^n) Elliptic Curves functions by a call to the Setup for Modular Reductions with the GF(2^n) option (see *Modular Reduction* from Related Links).

37.3.7.1.5 Curve Parameters in Memory

Some services need one or both of the Elliptic Curve Equation Parameters a and b. In this case these values are organized in memory as follows:

- The a Parameter relative to the Elliptic Curve Equation (often pointed by {nu1ABase,u2ModLength +4}). The a Parameter is written in a classical way in memory. It is u2ModLength bytes long and has a supplemental zeroed 32-bit word on the MSB side.
- The a and b Parameters relative to the Elliptic Curve Equation (often pointed by {nu1ABBase,2*u2ModLength + 8}):
  - The a Parameter is written in memory on u2ModLength bytes long, with a supplemental zeroed 32-bit word on the MSB side.
  - The b Parameter is written in memory after the a Parameter at an offset of (u2ModLength + 4) bytes. It is written in memory on u2ModLength bytes long, with a supplemental zeroed 32-bit word on the MSB side.

37.3.7.2 Point Addition

37.3.7.2.1 Purpose

This service is used to perform a point addition, based on a given elliptic curve over GF(2^n).

Please note that this service is not intended to add the same point twice. In this particular case, use the doubling service (see 37.3.7.3. Point Doubling).

37.3.7.2.2 How to Use the Service

37.3.7.2.3 Description

The operation performed is:

$Pt_C = Pt_A + Pt_B$

In this computation, the following parameters need to be provided:

- Point A the input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointABase,3*u2ModLength + 12}). This point can be the Infinite Point.
- Point B the input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointBBase,3*u2ModLength + 12}). This point can be the Infinite Point.
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength + 12})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength + 4})
- The a parameter relative to the elliptic curve equation (pointed by {nu1ABase,u2ModLength + 4})
- The workspace not initialized (pointed by {nu1WorkSpace, 7*u2ModLength + 40}

The resulting C point is represented in projective coordinates (X,Y,Z) and is stored at the same place than the input point A. This Point can be the Infinite Point.

The services for this operation are:

- Service GF2NEccAddFast: The fast mode is used, the fast modular reduction is used in the computations.

Important: Before using this service, ensure that the constant Cns has been calculated with the setup of the Modular Reductions service.

37.3.7.2.4 Parameters Definition

Table 37-94. GF2NEccAddFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of Modulus P	Base of Modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 12	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulo	Length of modulo
nu1PointABase	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Input point A (projective coordinates)	Resulting point C (projective coordinates)
nu1PointBBase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point B (projective coordinates)	Input point B
nu1ABBase	nu1	I	Crypto RAM	u2ModLength + 4	Parameter a of the elliptic curve	Unchanged
nu1Workspace	nu1	I	Crypto RAM	7*u2ModLength + 40	–	Corrupted workspace

37.3.7.2.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;
//Depending on the function the Random Number Generator
//must be initialized and started
//following the directives given for the RNG on the chip
PUKCL(u2Option) = 0;
PUKCL_GF2NEccAdd(nu1ModBase) = <Base of the ram location of P>;
PUKCL_GF2NEccAdd(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL_GF2NEccAdd(u2ModLength) = <Byte length of P>;
PUKCL_GF2NEccAdd(nu1PointABase) = <Base of the ram location of the A point>;
PUKCL_GF2NEccAdd(nu1PointBBase) = <Base of the ram location of the B point>;
PUKCL_GF2NEccAdd(nu1ABBase) = <Base of the ram location of the a Parameter>;
PUKCL_GF2NEccAdd(nu1Workspace) = <Base of the ram location of the workspace>;
. . .
// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(GF2NEccAddFast, pvPUKCLParam);
if (PUKCL(u2Status) == PUKCL_OK)
                {
                ...
                }
else // Manage the error
```

37.3.7.2.6 Constraints

No overlapping between either input and output are allowed The following conditions must be avoided to ensure the service works correctly:

- nu1ModBase,nu1CnsBase, nu1PointABase, nu1PointBBase, nu1ABBase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength+ 12}, {nu1PointBBase, 3*u2ModLength + 12}, {nu1ABase,u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1PointBBase, 3*u2ModLength + 12}, {nu1ABase,u2ModLength + 4} and {nu1Workspace, 5*u2ModLength + 32}

37.3.7.2.7 Status Returned Values

Table 37-95. GF2NEccAddFast Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without errors.

37.3.7.3 Point Doubling

37.3.7.3.1 Purpose

This service is used to perform a Point Doubling, based on a given elliptic curve over GF(2^n).

37.3.7.3.2 How to Use the Service

37.3.7.3.3 Description

The operation performed is:

$Pt_C = 2 \times Pt_A$

In this computation, the following parameters need to be provided:

- A the input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointABase,3*u2ModLength + 12}). This point can be the Infinite Point.
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength +8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength +4})
- The workspace not initialized (pointed by {nu1WorkSpace, 4*u2ModLength +28}
- The a and b Parameters relative to the Elliptic Curve Equation (pointed by {nu1ABBase,2*u2ModLength+ 8})
- The resulting C point is represented in projective coordinates (X,Y,Z) and is stored at the very same place than the input point A. This point can be the Infinite Point.

The service name for this operation is `GF2NEccDblFast`. This service uses Fast mode and Fast Modular Reduction for computation.

Important: Before using this service, ensure that the constant Cns has been calculated with the setup of the Fast Modular Reductions service.

37.3.7.3.4 Parameters Definition

Table 37-96. GF2NEccDblFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 12	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1ABBase	u2	I	Crypto RAM	2*u2ModLength + 8	Parameters a and b of the elliptic curve	Parameter a and b of the elliptic curve
nu1PointABase	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Input point A (projective coordinates)	Resulting point C (projective coordinates)
nu1Workspace	nu1	I	Crypto RAM	4*u2ModLength + 28	–	Corrupted workspace

37.3.7.3.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

PUKCL (u2Option) = 0;

PUKCL _GF2NEccDbl(nu1ModBase) = <Base of the ram location of P>;
PUKCL _GF2NEccDbl(u2ModLength) = <Byte length of P>;
PUKCL _GF2NEccDbl(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _GF2NEccDbl(nu1PointABase) = <Base of the ram location of the A point>;
PUKCL _GF2NEccDbl(nu1ABBase) = <Base of the a and b parameters of the elliptic curve>;
PUKCL _GF2NEccDbl(nu1Workspace) = <Base of the ram location of the workspace>;
...
```

```
// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(GF2NEccDblFast,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.7.3.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1ABBase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength+ 12}, {nu1ABBase, 2*u2ModLength + 8}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1ABase, u2ModLength + 4} and {nu1Workspace, 4*u2ModLength + 28}

37.3.7.3.7 Status Returned Values

Table 37-97. GF2NEccDblFast Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.7.4 Scalar Point Multiply

37.3.7.4.1 Purpose

This service is used to multiply a point by an integral constant K on a given elliptic curve over GF(2^n).

37.3.7.4.2 How to Use the Service

37.3.7.4.3 Description

The operation performed is:

$Pt_C = K \times Pt_A$

In this computation, the following parameters need to be provided:

- A the input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointABase,3*u2ModLength + 12}). This point can be the Infinite Point.
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength + 8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength + 4})
- The workspace not initialized (pointed by {nu1WorkSpace, 8*u2ModLength + 44}
- The a and b parameters relative to the elliptic curve (pointed by {nu1ABBase,2*u2ModLength + 8})
- K the scalar number (pointed by {nu1ScalarNumber,u2ScalarLength + 4})

The resulting C point is represented in projective coordinates (X,Y,Z) and is stored at the very same place than the input point A. This point can be the Infinite Point.

The service name for this operation is `GF2NEccMulFast`. This service uses Fast mode and Fast Modular Reduction for computation.

Important: Before using this service, ensure that the constant Cns has been calculated with the setup of the Fast Modular Reductions service.

37.3.7.4.4 Parameters Definition

Table 37-98. GF2NEccMulFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 12	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1KBase	nu1	I	Crypto RAM	u2KLength	Scalar number used to multiply the point A	Unchanged
u2KLength	u2	I	–	–	Length of scalar K	Length of scalar K
nu1PointBase	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Input point A (projective coordinates)	Resulting point C (projective coordinates)
nu1ABase	nu1	I	Crypto RAM	2*u2ModLength + 8	Parameters a and b of the elliptic curve	Unchanged
nu1Workspace	nu1	I	Crypto RAM	8*u2ModLength + 44	–	Corrupted workspace

37.3.7.4.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

PUKCL (u2Option) = 0;

PUKCL _GF2NEccMul(nu1ModBase) = <Base of the ram location of P>;
PUKCL _GF2NEccMul(u2ModLength) = <Byte length of P>;
PUKCL _GF2NEccMul(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _GF2NEccMul(nu1PointBase) = <Base of the ram location of the A point>;
PUKCL _GF2NEccMul(nu1ABase) = <Base of the ram location of the parameters a and b of the
elliptic
curve>;
PUKCL _GF2NEccMul(nu1KBase) = <Base of the ram location of the scalar number>;
PUKCL _GF2NEccMul(nu1Workspace) = <Base of the ram location of the workspace>;
PUKCL _GF2NEccMul(u2KLength) = <Length of the ram location of the scalar number>;

...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(GF2NEccMulFast,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.7.4.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointBase, nu1ABase, nu1KBase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointBase, 3*u2ModLength+ 12}, {nu1ABase, 2*u2ModLength + 8}, {nu1KBase, u2KLength} or {nu1Workspace, 8*u2ModLength + 44} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length

- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8}, {nu1PointBase, 3*u2ModLength + 12}, {nu1ABase, 2*u2ModLength + 8}, {nu1KBase, u2KLength} and {nu1Workspace, 8*u2ModLength + 44}

37.3.7.4.7 Status Returned Values

Table 37-99. GF2NEccMulFast Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.7.5 Projective to Affine Coordinates Conversion

37.3.7.5.1 Purpose

This service is used to perform a point coordinates conversion from a projective representation to an affine.

37.3.7.5.2 How to Use the Service

37.3.7.5.3 Description

The operation performed is:

$$Pt_{X\ Affine\ \ coordinate} = \left[\frac{Pt_{X\text{Projective coordinate}}}{\left(Pt_{Z\ \text{Projective coordinate}}\right)}\right]$$

$$Pt_{Y\ Affine\ \ coordinate} = \left[\frac{Pt_{Y\ \text{Projective coordinate}}}{\left(Pt_{Z\ \text{Projective coordinate}}\right)2}\right]$$

In this computation, the following parameters need to be provided:

- A the input point is filled in projective coordinates (X,Y,Z) or affine coordinates for X and Y, and setting Z to 1 (pointed by {nu1PointABase,3*u2ModLength + 12}). The Point A can be the point at infinity. In this case, the u2Status returned is PUKCL_POINT_AT_INFINITY.
- Cns the Modular Constant filled (pointed by {nu1CnsBase,u2ModLength + 8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength + 4})
- The workspace not initialized (pointed by {nu1WorkSpace, 4*u2ModLength + 48}

The result is the point A with its (X,Y) coordinates converted to affine, and the Z coordinate set to 1.

The service name for this operation is `GF2NEcConvProjToAffine`.

Important: Before using this service, ensure that the constant Cns has been calculated with the setup of the Fast Modular Reductions service.

37.3.7.5.4 Parameters Definition

Table 37-100. GF2NEcConvProjToAffine Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 12	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1PointABase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point A	Resulting point A in affine coordinates
nu1Workspace	nu1	I	Crypto RAM	4*u2ModLength + 48	–	Workspace

37.3.7.5.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
// ! following the directives given for the RNG on the chip

PUKCL (u2Option) = 0;

PUKCL _GF2NEcConvProjToAffine(nu1ModBase) = <Base of the ram location of P>;
PUKCL _GF2NEcConvProjToAffine(u2ModLength) = <Byte length of P>;
PUKCL _GF2NEcConvProjToAffine(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _GF2NEcConvProjToAffine(nu1PointABase) = <Base of the ram location of the A point>;
PUKCL _GF2NEcConvProjToAffine(nu1Workspace) = <Base of the ram location of the workspace>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(GF2NEcConvProjToAffine,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.7.5.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8},{nu1PointABase, 3*u2ModLength + 12}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12} and {nu1Workspace, 4*u2ModLength + 48}

37.3.7.5.7 Status Returned Values

Table 37-101. GF2NEcConvProjToAffine Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.
PUKCL_POINT_AT_INFINITY	Warning	The input point has its Z equal to zero, so it is a representation of the infinite point.

37.3.7.6 Affine to Projective Coordinates Conversion

37.3.7.6.1 Purpose

This service is used to perform a point coordinates conversion from an affine point representation to projective.

37.3.7.6.2 How to Use the Service

37.3.7.6.3 Description

The operation performed is:

affine(*Xa*, *Ya*) → *projective*(*Xp*, *Yp*, *Zp*)

In this computation, the following parameters need to be provided:

- A the input point is filled in affine coordinates for X and Y, and setting Z to 1 (pointed by {nu1PointABase,3*u2ModLength + 4}).
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength + 8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength + 4})

- The workspace not initialized (pointed by {nu1WorkSpace, 2*u2ModLength +16} The result is the point A with its (X,Y,Z) projective coordinates.

The service name for this operation is `GF2NEcConvAffineToProjective`.

Important: Before using this service, ensure that the constant Cns has been calculated with the setup of the Fast Modular Reductions service.

37.3.7.6.4 Parameters Definition

Table 37-102. GF2NEcConvAffineToProjective Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1PointABase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point A	Resulting point A in affine coordinates
nu1Workspace	nu1	I	Crypto RAM	2*u2ModLength + 16	–	Workspace

37.3.7.6.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
// ! following the directives given for the RNG on the chip

PUKCL (u2Option) = 0;

PUKCL _GF2NEcConvAffineToProjective(nu1ModBase) = <Base of the ram location of P>;
PUKCL _GF2NEcConvAffineToProjective(u2ModLength) = <Byte length of P>;
PUKCL _GF2NEcConvAffineToProjective(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _GF2NEcConvAffineToProjective(nu1PointABase) = <Base of the ram location of the A
point>;
PUKCL _GF2NEcConvAffineToProjective(nu1Workspace) = <Base of the ram location of the
workspace>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(GF2NEcConvAffineToProjective,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.7.6.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, and {nu1Workspace, 2*u2ModLength + 16}

37.3.7.6.7 Status Returned Values

Table 37-103. GF2NEcConvAffineToProjective Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.7.7 Randomize Coordinate

37.3.7.7.1 Purpose

This service is used to convert the Projective representation of a point to another Projective representation.

37.3.7.7.2 How to Use the Service

37.3.7.7.3 Description

The operation performed is:

Projective(X_1, Y_1, Z_1) → *Projective*(X_2, Y_2, Z_2)

In this computation, the following parameters need to be provided:

- The input point is filled in projective coordinates (X,Y,Z) (pointed by {nu1PointBase,3*u2ModLength + 12}). This Point must not be the point at infinity.
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase,u2ModLength + 8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength + 4})
- The workspace not initialized (pointed by {nu1WorkSpace, 3*u2ModLength + 28}
- The random number (pointed by {nu1RandomBase, u2ModLength + 4}) The result is the point nu1PointBase with its (X,Y,Z) coordinates randomized. The service for this operation is GF2NEcRandomiseCoordinate.

Important:
Before using this service:

- Ensure that the constant Cns has been calculated with the Setup of the fast Modular Reductions service.
- Be sure to follow the directives given for the RNG on the chip you use (particularly initialization, seeding) and compulsorily start the RNG.

37.3.7.7.4 Parameters Definition

Table 37-104. GF2NEcRandomiseCoordinate Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1PointBase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point	Resulting point
nu1RandomBase	nu1	I	Crypto RAM	u2ModLength + 4	Random	Corrupted
nu1Workspace	nu1	I	Crypto RAM	3*u2ModLength + 28	–	Workspace

37.3.7.7.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
```

```
// ! following the directives given for the RNG on the chip

PUKCL (u2Option) = 0;

// Depending on the option specified, not all fields must be filled
PUKCL _GF2NEcRandomiseCoordinate(nu1ModBase) = <Base of the ram location of P>;
PUKCL _GF2NEcRandomiseCoordinate(u2ModLength) = <Byte length of P>;
PUKCL _GF2NEcRandomiseCoordinate(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL_GF2NEcRandomiseCoordinate(nu1RandomBase) = <Base of the ram location where the the rng
is stored>;
PUKCL _GF2NEcRandomiseCoordinate(nu1PointBase) = <Base of the ram location of the point>;
PUKCL _GF2NEcRandomiseCoordinate(nu1Workspace) =
<Base of the ram location of the workspace>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(GF2NEcRandomiseCoordinate,&PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.7.7.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1RandomBase, nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1RandomBase, u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1RandomBase, u2ModLength + 4} and {nu1Workspace, 3*u2ModLength + 28}

37.3.7.7.7 Status Returned Values

Table 37-105. GF2NEcRandomiseCoordinate Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.

37.3.7.8 Point is on Elliptic Curve

37.3.7.8.1 Purpose

This service is used to test whether the point is on the curve.

37.3.7.8.2 How to Use the Service

37.3.7.8.3 Description

The operation performed is:

Status = IsPointOnCurve(X, Y, Z);

In this computation, the following parameters need to be provided:

- The input points filled in projective coordinates (X, Y, Z) (pointed by {nu1PointBase, 3*U2ModLength + 4}). This point can be point at infinity.
- AParam and BParam are the Elliptic Curve Equation parameters (pointed by {nu1AParam, u2ModLength+ 4} and {nu1BParam, u2ModLength + 4}).
- Cns the Fast Modular Constant filled (pointed by {nu1CnsBase, u2ModLength + 8})

- P the modulus filled (pointed by {nu1ModBase, u2ModLength + 8})
- The workspace not initialized (pointed by {nu1WorkSpace, 4*u2ModLength + 28})

The service name for this operation is `GF2NEcPointIsOnCurve`.

Important: Before using this service, the constant Cns must have been calculated with the Fast Modular Reduction service.

37.3.7.8.4 Parameters Definition

Table 37-106. GF2NEcPointIsOnCurve Service Parameters

Parameter	Type	Dir.	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ModLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1PointBase	nu1	I	Crypto RAM	3*u2ModLength + 12	Input point	Unchanged
nu1AParam	nu1	I	Crypto RAM	u2ModLength + 4	The parameter a	Unchanged
nu1BParam	nu1	I	Crypto RAM	u2ModLength + 4	The parameter b	Unchanged
nu1Workspace	nu1	I	Crypto RAM	4*u2ModLength + 28	N/A	Workspace

37.3.7.8.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
// ! following the directives given for the RNG on the chip

PUKCL (u2Option) = 0;

// Depending on the option specified, not all fields must be filled
PUKCL _GF2NEcPointIsOnCurve(nu1ModBase) = <Base of the ram location of P>;
PUKCL _GF2NEcPointIsOnCurve(u2ModLength) = <Byte length of P>;
PUKCL _GF2NEcPointIsOnCurve(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _GF2NEcPointIsOnCurve(nu1PointABase) = <Base of the A point>;
PUKCL _GF2NEcPointIsOnCurve(nu1AParam) = <Base of the ram location of the parameter a>;
PUKCL _GF2NEcPointIsOnCurve(nu1BParam) = <Base of the ram location of the parameter b>;
PUKCL _GF2NEcPointIsOnCurve(nu1PointBase) = <Base of the ram location of the point>;
PUKCL _GF2NEcPointIsOnCurve(nu1Workspace) = <Base of the ram location of the workspace>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKC L_Process(GF2NEcPointIsOnCurve,
pvPUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.7.8.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure that the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1AParam, nu1BParam and nu1Workspace are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1AParam, u2ModLength + 4}, {nu1BParam, u2ModLength + 4}, {nu1Workspace, 4*u2ModLength + 28} are not in Crypto RAM

- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1AParam, u2ModLength + 4}, {nu1BParam, u2ModLength + 4} and {nu1Workspace, 4*u2ModLength + 28}

37.3.7.8.7 Status Returned Values

Table 37-107. GF2NEcPointIsOnCurve Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The point is on the curve.
PUKCL_POINT_IS_NOT_ON_CURVE	Warning	The point is not on the curve.
PUKCL_POINT_AT_INFINITY	Warning	The input point has its Z equal to zero, so it's a representation of the infinite point.

37.3.7.9 Generating an ECDSA Signature (Compliant with FIPS 186-2)

37.3.7.9.1 Purpose

This service is used to generate an ECDSA signature following the FIPS 186-2. It performs the second step of the Signature Generation. A hash value (HashVal) must be provided as input, it has to be previously computed from the message to be signed using a secure hash algorithm.

A scalar number must be provided, as described in the FIPS 186-2.

The result (R,S) is computed by this service. If S equals zero, the status is set to PUKCL_WRONG_SELECT_NUMBER.

37.3.7.9.2 How to Use the Service

37.3.7.9.3 Description

The operation performed is:

(*R*, *S*) = *EcDsaSign*(Pt_A, *HashVal*, *k*, *CurveParameters*, *PrivateKey*)

This service processes the following checks:

- If the Scalar Number k is out of the range [1, PointOrder -1], the calculus is stopped and the status is set to PUKCL_WRONG_SELECT_NUMBER.
- If R equals zero, the calculus is stopped and the status is set to PUKCL_WRONG_SELECT_NUMBER.
- If S equals zero, the calculus is stopped and the status is set to PUKCL_WRONG_SELECT_NUMBER. In this computation, the following parameters need to be provided:
- A the input point is filled in "mixed" coordinates (X,Y) with the affine values and Z = 1 (pointed by {nu1PointABase,3*u2ModLength + 12})
- Cns the working space for the Fast Modular Constant not initialized (pointed by {nu1CnsBase,u2ScalarLength + 8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength + 4})
- The workspace not initialized (pointed by {nu1WorkSpace, 8*u2ModLength + 44}
- The a and b parameters relative to the elliptic curve equation (pointed by {nu1ABBase, 2*u2ModLength + 8})
- The order of the Point A on the elliptic curve (pointed by {nu1OrderPointBase, u2ScalarLength + 4})
- k the input Scalar Number beforehand generated and filled (pointed by{nu1ScalarNumber,u2ScalarLength + 4})
- HashVal the hash value beforehand generated and filled (pointed by {nu1HashBase, u2ScalarLength +4})

- The Private Key (pointed by {nu1PrivateKey, u2ScalarLength +4})
- Generally u2ScalarLength is equal to (u2ModLength) or (u2ModLength + 4)

Important:
For the ECDSA signature generation be sure to follow the directives given for the RNG on the chip you use (particularly initialization, seeding) and compulsorily start the RNG.

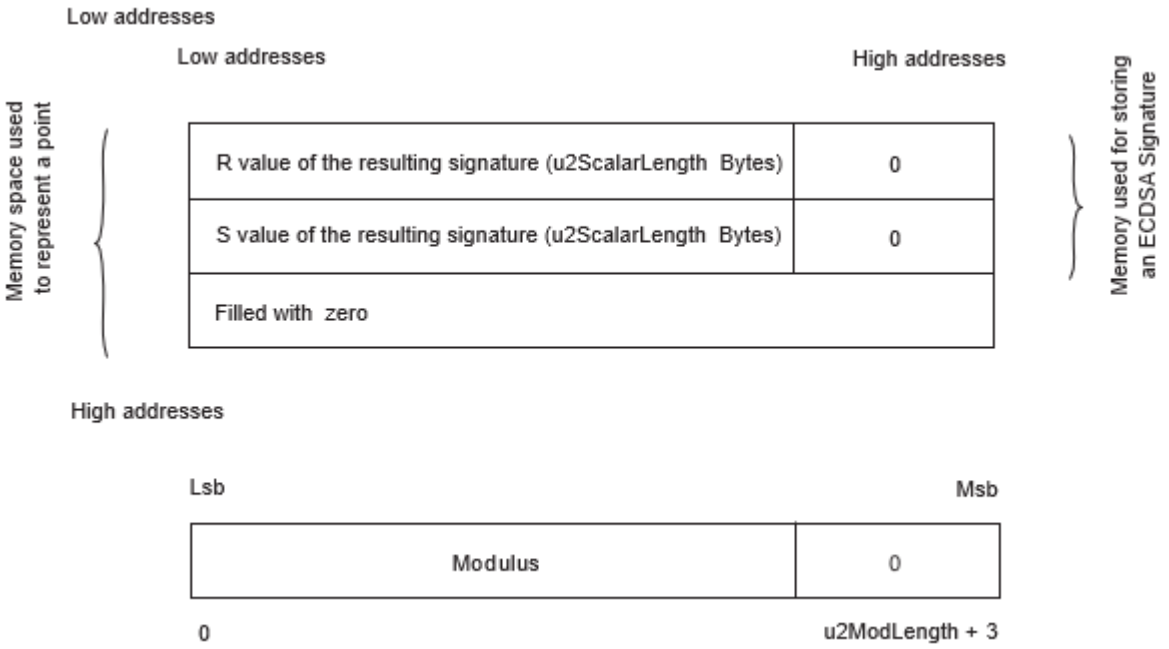
The scalar number k must be selected at random. This random must be generated before the call of the ECDSA signature. For this random generation be sure to follow the directives given for the RNG on the chip you use (particularly initialization, seeding) and compulsorily start the RNG.

The operation performed is:

- Compute the ECDSA (R,S) as described in FIPS 186-2, but leaving the user the role of computing the input Hash Value, thus leaving the freedom of using any other algorithm than SHA-1.
- Compute a R value using the input A point and the scalar number.
- Compute a S value using R, the scalar number, the private key and the provided hash value. Note that the resulting signature (R,S) is stored at the place of the input A point.
- If all is correct and S is different from zero, the status is set to PUKCL_OK. If all is correct and S equals zero,the status is set to PUKCL_WRONG_SELECT_NUMBER. If an error occurs, the status is set to the corresponding error value (see Status Returned Values below).

The service name for this operation is GF2NEcDsaGenerateFast. The fast mode is used, the fast modular reduction is used in the computations.

- The signature (R,S), when resulting from a computation is given back at address of the A point:
  - The R value result with u2ModLength + 4 bytes (padded with zeros).
  - The S value result with u2ModLength + 4 bytes (padded with zeros)
  - The u2NLength + 4 following bytes (space for the third coordinate of A) are filled with zeros.



37.3.7.9.4 Parameters Definition

Table 37-108. GF2NEcDsaGenerateFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ScalarLength + 12	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1ScalarNumber	nu1	I	Crypto RAM	u2ScalarLength + 4	Scalar Number used to multiply the point A	Unchanged
nu1OrderPointBase	nu1	I	Crypto RAM	u2ScalarLength + 4	Order of the Point A in the elliptic curve	Unchanged
nu1PrivateKey	nu1	I/O	Crypto RAM	u2ScalarLength + 4	Base of the Private Key	Unchanged
nu1HashBase(1)	nu1	I	Crypto RAM	u2ScalarLength + 4	Base of the hash value resulting from the previous SHA	Unchanged
u2ScalarLength	u2	I	–	–	Length of scalar (same length as the length of order)	Length of scalar
nu1PointABase	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Input point A (three coordinates (X,Y) affine and Z = 1)	Resulting signature (R,S,0)
nu1ABase	nu1	I	Crypto RAM	2*u2ModLength + 8	Parameter a of the elliptic curve	Unchanged
nu1Workspace	nu1	I	Crypto RAM	8*u2ModLength + 44	–	Corrupted workspace

Note:

1. Whatever the chosen SHA, the resulting hash value may have a length inferior or equal to the modulo length and be padded with zeros until its total length is u2ModLength + 4.

37.3.7.9.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
// ! following the directives given for the RNG on the chip

PUKCL (u2Option) = 0;

// Depending on the option specified, not all fields must be filled
PUKCL _GF2NEcDsaGenerate(nu1ModBase) = <Base of the ram location of P>;
PUKCL _GF2NEcDsaGenerate(u2ModLength) = <Byte length of P>;
PUKCL _GF2NEcDsaGenerate(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _GF2NEcDsaGenerate(nu1PointABase) = <Base of the A point>;
PUKCL _GF2NEcDsaGenerate(nu1PrivateKey) = <Base of the Private Key>;
PUKCL _GF2NEcDsaGenerate(nu1ScalarNumber) = <Base of the ScalarNumber>;
PUKCL _GF2NEcDsaGenerate(nu1OrderPointBase) = <Base of the order of A point>;
PUKCL _GF2NEcDsaGenerate(nu1ABase) = <Base of the a parameter of the curve>; PUKCL
_GF2NEcDsaGenerate(nu1Workspace) = <Base of the workspace>;
PUKCL _GF2NEcDsaGenerate(nu1HashBase) = <Base of the SHA resulting hash>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(GF2NEcDsaGenerateFast, pvPUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else // Manage the error
```

37.3.7.9.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1PrivateKey, nu1ScalarNumber, nu1OrderPointBase,nu1ABase, nu1Workspace or nu1HashBase are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength+ 12},{nu1PrivateKey, u2ScalarLength + 4},{nu1ScalarNumber, u2ScalarLength + 4},{nu1OrderPointBase, u2ScalarLength + 4}, {nu1ABase, u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} or {nu1HashBase, u2ScalarLength + 4} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1PrivateKey, u2ScalarLength + 4}, {nu1ScalarNumber, u2ScalarLength + 4}, {nu1OrderPointBase, u2ScalarLength + 4}, {nu1ABase, u2ModLength + 4}, {nu1Workspace, <WorkspaceLength>} and {nu1HashBase, u2ScalarLength + 4}

37.3.7.9.7 Status Returned Values

Table 37-109. GF2NEcDsaGenerate Fast Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without problem.
PUKCL_WRONG_SELECTNUMBER	Warning	The given value for nu1ScalarNumber is not good to perform this signature generation.

37.3.7.10 Verifying an ECDSA Signature (Compliant with FIPS 186-2)

37.3.7.10.1 Purpose

This service is used to verify an ECDSA signature following the FIPS 186-2. It performs the second step of the Signature Verification.

A hash value (HashVal) must be provided as input, it has to be previously computed from the message to be signed using a secure hash algorithm.

As second significant input, the Signature is provided to be checked. This service checks the signature and fills the status accordingly.

37.3.7.10.2 How to Use the Service

37.3.7.10.3 Description

The operation performed is:

Verify = *EcDsaVerifySignature*(Pt_A, *HashVal*, *Signature*, *CurveParameters*, *PublicKey*)

The points used for this operation are represented in different coordinate systems. In this computation, the following parameters need to be provided:

- A the input point is filled with the affine values (X,Y) and Z = 1 (pointed by{nu1PointABase,3*u2ModLength + 12})
- Cns the working space for the Fast Modular Constant not initialized (pointed by {nu1CnsBase,u2ScalarLength + 8})
- P the modulus filled (pointed by {nu1ModBase,u2ModLength +4})
- The workspace not initialized (pointed by {nu1WorkSpace, 8*u2ModLength +44} The a and b parameters relative to the elliptic curve (pointed by {nu1ABase,2*u2ModLength + 8})
- The order of the Point A on the elliptic curve (pointed by {nu1OrderPointBase,u2ScalarLength +4})

- HashVal the hash value beforehand generated and filled (pointed by {nu1HashBase,u2ScalarLength +4})
- The Public Key point is filled in “mixed” coordinates (X,Y) with the affine values and Z = 1 (pointed by {nu1PointPublicKeyGen, 3*u2ModLength + 12})
- The input signature (R,S), even if it is not a Point, is represented in memory like a point in affine coordinates (X,Y) (pointed by {nu1PointSignature, 2*u2ScalarLength + 8})

Important: For the ECDSA signature verification be sure to follow the directives given for the RNG on the chip you use (particularly initialization, seeding) and compulsorily start the RNG.

- The operation consists in obtaining a V value with all these input parameter and check that V equals the provided R. If all is correct and the signature is the good one, the status is set to PUKCL_OK. If all is correct and the signature is wrong, the status is set to PUKCL_WRONG_SIGNATURE. If an error occurs, the status is set to the corresponding error value (see Status Returned Values below).

The service name for this operation is `GF2NEcDsaVerifyFast`. This service uses Fast mode and Fast Modular Reduction for computation.

37.3.7.10.4 Parameters Definition

Table 37-110. GF2NEcDsaVerifyFast Service Parameters

Parameter	Type	Direction	Location	Data Length	Before Executing the Service	After Executing the Service
nu1ModBase	nu1	I	Crypto RAM	u2ModLength + 4	Base of modulus P	Base of modulus P
nu1CnsBase	nu1	I	Crypto RAM	u2ScalarLength + 8	Base of Cns	Base of Cns
u2ModLength	u2	I	–	–	Length of modulus P	Length of modulus P
nu1OrderPointBase	nu1	I	Crypto RAM	u2ScalarLength + 4	Order of the Point A in the elliptic curve	Unchanged
nu1PointSignature	nu1	I	Crypto RAM	2*u2ScalarLength + 8	Signature(r, s)	Corrupted
nu1HashBase(1)	nu1	I	Crypto RAM	u2ScalarLength + 4	Base of the hash value resulting from the previous SHA	Corrupted
u2ScalarLength	u2	I	–	–	Length of scalar	Length of scalar
nu1PointABase	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Generator point	Corrupted
nu1PointPublicKeyGen	nu1	I/O	Crypto RAM	3*u2ModLength + 12	Public point	Corrupted
nu1ABase	nu1	I	Crypto RAM	2*u2ModLength + 8	Parameter a and b of the elliptic curve	Unchanged
nu1Workspace	nu1	I	Crypto RAM	8*u2ModLength + 44	–	Corrupted workspace

Note:

1. Whatever the chosen SHA, the resulting hash value may have a length inferior or equal to the modulo length and be padded with zeros until its total length is u2ModLength + 4.

37.3.7.10.5 Code Example

```
PUKCL_PARAM PUKCLParam;
PPUKCL_PARAM pvPUKCLParam = &PUKCLParam;

// ! The Random Number Generator must be initialized and started
// ! following the directives given for the RNG on the chip

PUKCL (u2Option) = 0;
```

```
// Depending on the option specified, not all fields must be filled PUKCL
_GF2NEcDsaVerify(nu1ModBase) = <Base of the ram location of P>;
PUKCL _GF2NEcDsaVerify(u2ModLength) = <Byte length of P>;
PUKCL _GF2NEcDsaVerify(nu1CnsBase) = <Base of the ram location of Cns>;
PUKCL _GF2NEcDsaVerify(nu1PointABase) = <Base of the A point>;
PUKCL _GF2NEcDsaVerify(nu1PrivateKey) = <Base of the Private Key>;
PUKCL _GF2NEcDsaVerify(nu1ScalarNumber) = <Base of the ScalarNumber>;
PUKCL _GF2NEcDsaVerify(nu1OrderPointBase) = <Base of the order of A point>;
PUKCL _GF2NEcDsaVerify(nu1ABase) = <Base of the a parameter of the curve>; PUKCL
_GF2NEcDsaVerify(nu1Workspace) = <Base of the workspace>;
PUKCL _GF2NEcDsaVerify(nu1HashBase) = <Base of the SHA resulting hash>;
...

// vPUKCL_Process() is a macro command, which populates the service name
// and then calls the library...
vPUKCL_Process(GF2NEcDsaVerifyFast, &PUKCLParam);
if (PUKCL (u2Status) == PUKCL_OK)
            {
            ...
            }
else
            if(PUKCL(u2Status) == PUKCL_WRONG_SIGNATURE)
            {
            ...
            }
else // Manage the error
```

37.3.7.10.6 Constraints

No overlapping between either input and output are allowed. The following conditions must be avoided to ensure the service works correctly:

- nu1ModBase, nu1CnsBase, nu1PointABase, nu1PointPublicKeyGen, nu1PointSignature, nu1OrderPointBase,nu1ABBase, nu1Workspace or nu1HashBase are not aligned on 32-bit boundaries
- {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength + 8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1PointPublicKeyGen, 3*u2ModLength + 12}, {nu1PointSignature,2*u2ScalarLength + 8}, {nu1OrderPointBase, u2ScalarLength + 4}, {nu1ABBase, 2*u2ModLength + 8}, {nu1Workspace, <WorkspaceLength>} or {nu1HashBase, u2ScalarLength + 4} are not in Crypto RAM
- u2ModLength is either: < 12, > 0xffc or not a 32-bit length
- All overlapping between {nu1ModBase, u2ModLength + 4}, {nu1CnsBase, u2ModLength +8}, {nu1PointABase, 3*u2ModLength + 12}, {nu1PointPublicKeyGen, 3*u2ModLength + 12}, {nu1PointSignature, 2*u2ScalarLength + 8}, {nu1OrderPointBase, u2ScalarLength + 4}, {nu1ABBase, 2*u2ModLength + 8}, {nu1Workspace, <WorkspaceLength>} and {nu1HashBase, u2ScalarLength + 4}

37.3.7.10.7 Status Returned Values

Table 37-111. GF2NEcDsaVerifyFast Service Return Codes

Returned Status	Importance	Meaning
PUKCL_OK	–	The computation passed without errors. The signature is correct.
PUKCL_WRONG_SIGNATURE	Warning	The signature is incorrect.

37.3.8 PUKCL Requirements and Performance

37.3.8.1 Services Stack Usage

This library is using the main core to execute its computations, and therefore is also sharing some resources with the application.

It may be important for the application to know RAM usage by the library functions and to be aware that the library does not use any global variables.

The following table provides the minimum number of bytes used by the library that have to be available on the stacks to ensure that the functionality can be executed correctly. In some cases, the library may use less bytes than the specified number for some options. This table contains estimated values.

Table 37-112. Services Stack Usage

PUKCL Service	STACK Usage (Bytes)
SelfTest	112
ClearFlags	0
Swap	8
Fill	8
CondCopy	24
FastCopy	16
Smult	16
Smult (with reduction)	88
Comp	8
Fmult	24
Fmult (with reduction)	96
Square	16
Square (with reduction)	88
Div	144
GCD	136
RedMod (Setup)	160
RedMod (using fast reduction)	80
RedMod (randomize)	80
RedMod (Normalize)	80
RedMod (Using Division)	184
ExpMod	200
PrimeGen	416
CRT	304
ZpEccAddFast	104
ZpEccAddSubFast	92
ZpEcConvProjToAffine	280
ZpEcConvAffineToProjective	64
ZpEccDblFast	96
ZpEccMulFast	168
ZpEccQuickDualMulFast	216
ZpEcDsaGenerateFast	392
ZpEcDsaVerifyFast	456
ZpEcDsaQuickVerify	368
ZpEcRandomiseCoordinate	56
GF2NEccAddFast	128
GF2NEcConvProjToAffine	264
GF2NEcConvAffineToProjective	56
GF2NEccDblFast	136
GF2NEccMulFast	208
GF2NEcDsaGenerateFast	376
GF2NEcDsaVerifyFast	440

.....continued	
PUKCL Service	**STACK Usage (Bytes)**
GF2NEcRandomiseCoordinate	56

37.3.8.2 Parameter Size Limits for Different Services

The following table lists parameter size limits for different services.

For the services ModExp, PrimeGen, and CRT, additional details are available in the service description.

Table 37-113. Parameter Size Limits

API	Min/Max Sizes	Comments
SelfTest	—	—
ClearFlags	—	—
Swap	4 bytes to 2044 bytes	Per block to be swapped
Fill	4 bytes to 4088 bytes	—
Fast Copy/Clear	4 bytes to 2044 bytes	Supposing Length(R) = Length(X)
Conditional Copy/Clear	4 bytes to 2044 bytes	Supposing Length(R) = Length(X)
Smult	4 bytes to 2040 bytes	Supposing Length(R) = Length(X) + 4 Bytes, No Z Parameter, No Reduction
Compare	4 bytes to 2044 bytes	Supposing Length(X) = Length(Y)
FMult	Input: 4 bytes to 1020 bytes Output: 4bytes to 2040 bytes	Supposing Length(Y) = Length(X), No Z Parameter, No Reduction
Square	Input: 4 bytes to 1020 bytes Output: 4 bytes to 2040 bytes	Supposing No Z Parameter, No Reduction
Euclidean Division	Divider: 8 to 1016 bytes Num.: 8 to 2032 bytes	Supposing Length(Num) = 2*Length(Divider)
Mod. inv. / GCD	8 to 1012 bytes	—
ModRed	Modulus: 12 to 1016 bytes Input: 24 to 2032 bytes	Supposing RBase = XBase
Fast ModExp Exp in Crypto RAM	12 to 576 bytes (96 to 4608 bits)	Supposing Length(Exponent) = Length(Modulus), Window Size = 1 With the Exponent in Crypto RAM
Fast ModExp Exp not in Crypto RAM	12 to 672 bytes (96 to 5376 bits)	Supposing Length(Exponent) = Length(Modulus), Window Size = 1 With the Exponent not in Crypto RAM
Prime Gen.	Prime Number: 12 to 448 bytes (96 to 3584 bits)	Supposing Window Size = 1
CRT	Modulus = Two Primes: Size of one prime from 24 to 448 bytes Modulus = from 48 to 896 bytes (384 to 7168 bits)	Supposing Length(Exponent) = Length(Modulus), Window Size = 1
ECC Addition qnd Subtraction GF(p)	Modulus: 12 to 308 bytes	—
ECC Doubling GF(p)	Modulus: 12 to 400 bytes	—
ECC Multiplication GF(p)	Modulus: 12 to 264 bytes	Supposing Length(Scalar) = Length(Modulus)
ECC Quick Dual Multiplication GF(p)	Modulus: 12 to 152 bytes	—

.....continued		
API	Min/Max Sizes	Comments
ECDSA Generate GF(p)	Modulus: 12 to 220 bytes (up to 521 bits for common curves)	Supposing Length(Scalar) = Length(Modulus)
ECDSA Verify GF(p)	Modulus: 12 to 188 bytes (up to 521 bits for common curves)	Supposing Length(Scalar) = Length(Modulus)
ECC Addition GF(2n)	Modulus: 12 to 248 bytes	—
ECC Doubling GF(2n)	Modulus: 12 to 364 bytes	—
ECC Multiplication GF(2n)	Modulus: 12 to 250 bytes	Supposing Length(Scalar) = Length(Modulus)
ECDSA Generate GF(2n)	Modulus: 12 to 208 bytes (up to 571 bits for common curves)	Supposing Length(Scalar) = Length(Modulus)
ECDSA Verify GF(2n)	Modulus: 12 to 180 bytes (up to 571 bits for common curves)	Supposing Length(Scalar) = Length(Modulus)
ECDSA Quick Verify GF(2n)	Modulus: 12 to 140 bytes (up to 571 bits for common curves)	Supposing Length(Scalar) = Length(Modulus)

37.3.8.3 Service Timing

The values in the following tables are estimated performances for CPU clock of 64 MHz. The CPU and PUKCC are operated at the same frequency. Due to possible change in the parameters values, the measurements show approximated values.

Other test conditions:

- PUKCL library data in Crypto RAM
- Test code and test data in SRAM
- ICache and DCache are disabled

37.3.8.3.1 Service Timing for RSA

RSA uses the ExpMod service for encryption and decryption. Following tables show service timing, where 'W' indicates window size.

Table 37-114. RSA1024

Operation	Clock Cycles	Timing one block
RSA 1024 decryption / signature generation. No CRT, Regular implementation, W=4	3.05 MCycles	47.799 ms
RSA 1024 decryption / signature generation. With CRT, Regular implementation, W=4	1.09 MCycles	17.109 ms
RSA 1024 encryption / signature verification. No CRT, Fast implementation, W=1 Exponent=3	0.07 MCycles	1.141 ms
RSA 1024 encryption / signature verification. No CRT, Fast implementation, W=1 Exponent=0x10001	0.07 MCycles	1.129 ms

Table 37-115. RSA2048

Operation	Clock Cycles	Timing One block
RSA 2048 decryption / signature generation. No CRT, Regular implementation, W=4	21.6 MCycles	338.249 ms
RSA 2048 decryption / signature generation. With CRT, Regular implementation, W=4	6.36 MCycles	99.408 ms
RSA 2048 encryption / signature verification. No CRT, Fast implementation, W=1 Exponent=3	0.24 MCycles	3.843 ms

.....continued		
Operation	**Clock Cycles**	**Timing One block**
RSA 2048 encryption / signature verification. No CRT, Fast implementation, W=1 Exponent=0x10001	0.24 MCycles	3.827 ms

Table 37-116. RSA4096

Operation	Clock Cycles	Timing One block
RSA 4096 Decryption / signature generation. No CRT, Regular implementation, W=1	209 MCycles	3.2742s
RSA 4096 Decryption / signature generation. With CRT, Regular implementation, W=3	46.1 MCycles	720.95 ms
RSA 4096 encryption / signature verification. No CRT, Fast implementation, W=1 Exponent=3	0.91 MCycles	14.346 ms
RSA 4096 encryption / signature verification. No CRT, Fast implementation, W=1 Exponent=0x10001	0.91 MCycles	14.337 ms

37.3.8.3.2 Service Timing for Prime Generation

Prime generation uses the PrimeGen service.

Table 37-117. Prime Generation

Operation	Clock Cycles	Timing One Block
Regular Generation of two primes, Prime_Length=512 bits, W=4, Rabin Miller Iterations Number = 3, (average of 200 samples)	Mean = 47.4 MCycles	Mean = 0.4s
Regular Generation of two primes, Prime_Length=512 bits, W=4, Rabin Miller Iterations Number = 3, (Standard Deviation for 200 samples)	Std Dev = 30.3 Mcycles	Std Dev = 0.47s
Regular Generation of two primes, Prime_Length=1024 bits, W=4, Rabin Miller Iterations Number = 3, (average of 200 samples)	Mean = 419.71 MCycles	Mean = 6.558s
Regular Generation of two primes, Prime_Length=1024 bits, W=4, Rabin Miller Iterations Number = 3, (Standard Deviation for 200 samples)	Std Dev = 294 Mcycles	Std Dev = 4.59s
Regular Generation of two primes, Prime_Length=2048 bits, W=4, Rabin Miller Iterations Number = 3, (average of 200 samples)	Mean = 4.78 GCycles	Mean = 74.68s
Regular Generation of two primes, Prime_Length=2048 bits, W=4, Rabin Miller Iterations Number = 3, (Standard Deviation for 200 samples)	Std Dev = 3.05 GCycles	Std Dev = 47.65s

37.3.8.3.3 Service Timing for ECDSA on Prime Field

In the following table, ECDSA signature generation uses the ZpEcDsaGenerateFast service and signature verification uses ZpEcDsaQuickVerify

Table 37-118. ECDSA GF(p)

Operation	Clock Cycles	Timing One block
ECDSA GF(p) 256 Generate Fast	2.67 MCycles	41.864 ms
ECDSA GF(p) 256 Verify Quick W=(4,4) Scalar in PUKCC RAM	1.84 MCycles	28.888 ms
ECDSA GF(p) 384 Generate Fast	6.18 MCycles	96.712 ms
ECDSA GF(p) 384 Verify Quick W=(4,4) Scalar in PUKCC RAM	4.15 MCycles	64.868 ms
ECDSA GF(p) 521 Generate Fast	13.36 MCycles	208.869 ms
ECDSA GF(p) 521 Verify Quick W=(4,4) Scalar in PUKCC RAM	8.81 MCycles	137.711 ms

37.3.8.3.4 Service Timing for ECDSA on Binary Field

In the following table, ECDSA signature generation uses the GF2NEcDsaGenerateFast service and signature verification uses GF2NEcDsaVerifyFast

Table 37-119. ECDSA GF(2^n)

Operation	CPU Cycles	Timing One block
ECDSA GF(2^n) B283 Generate Fast	3.21 MCycles	50.301 ms
ECDSA GF(2^n) B283 Verify	6.40 MCycles	100.150 ms
ECDSA GF(2^n) B409 Generate Fast	6.94 Mcycles	108.554 ms
ECDSA GF(2^n) B409 Verify	13.73 Mcycles	214.571 ms
ECDSA GF(2^n) B571 Generate Fast	15.08 Mcycles	235.704 ms
ECDSA GF(2^n) B571 Verify	30.07 MCycles	469.972 ms

38. Analog-to-Digital Converter (ADC)

38.1 Overview

The PIC32CX-BZ2 12-bit High Speed Successive Approximation Register (SAR) Analog-to-Digital Converter (ADC) includes the following features:

- 12-bit resolution
- One ADC module, up to 2 Msps conversion rate
- Single-ended and/or differential input
- Supported in Sleep mode
- Two digital comparators
- Two digital filters supporting two modes:
  - Oversampling mode
  - Averaging mode
- Designed for motor control, power conversion and general purpose applications

The PIC32CX-BZ2 has one shared ADC module. This ADC module incorporates a multiplexer on the input to facilitate a group of inputs and provides a flexible automated scanning option through the input scan logic.

For the ADC module, the analog inputs are connected to the Sample and Hold (S&H) capacitor. The ADC module performs the conversion of the input analog signal based on the configurations set in the registers. When the conversion is complete, the final result is stored in the result buffer for the specific analog input and is passed to the digital filter and digital comparator if configured to use data from this particular sample.

Equation 38-1. ADC Throughput Rate

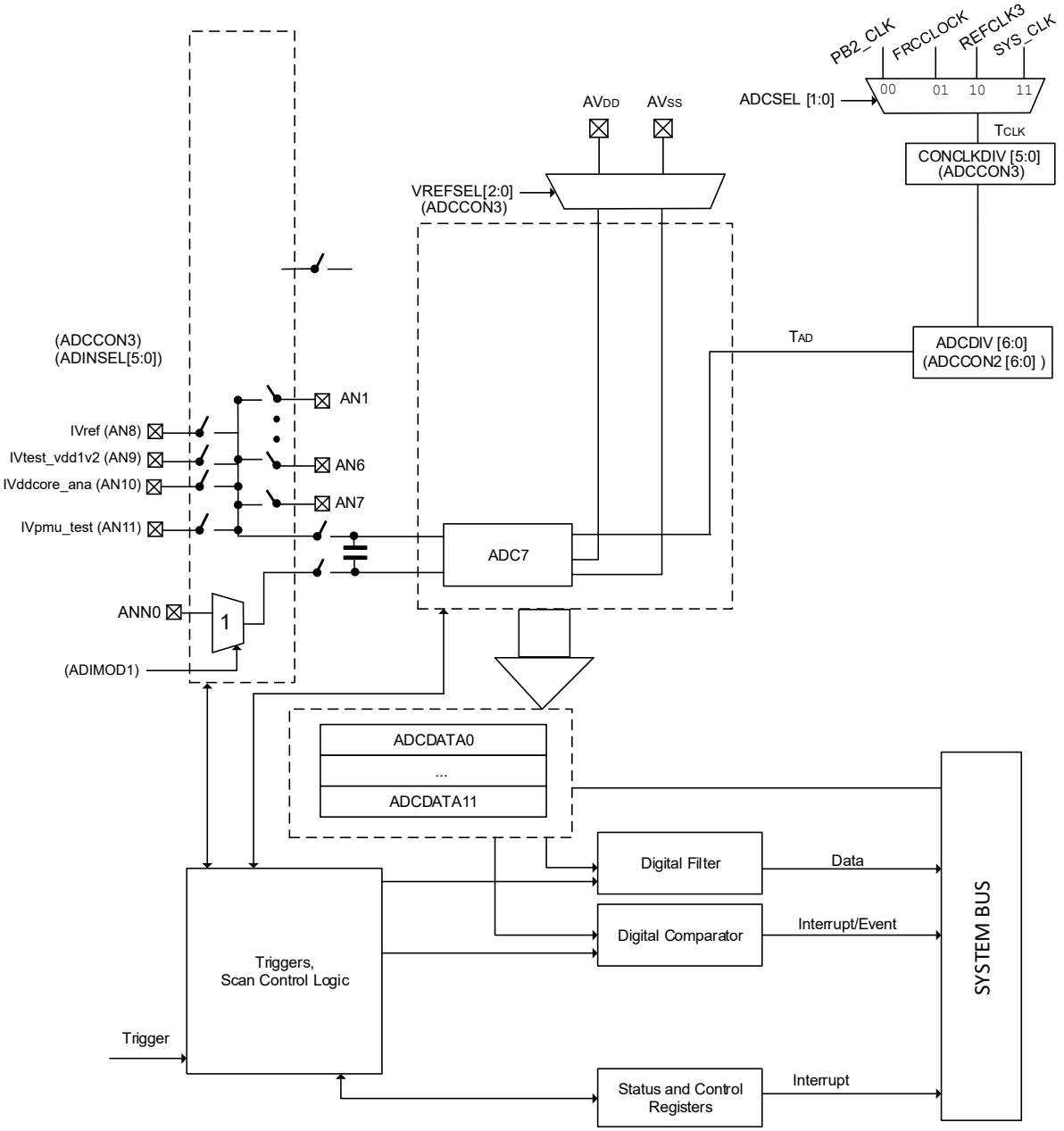
$$FTP = \frac{T_{AD}}{T_{SAMP} + T_{CONV}}$$

Where,

- T_{AD} = The frequency of the individual ADC module.

A block diagram of the ADC module is illustrated in the following figure.

Figure 38-1. ADC Block Diagram



38.2 ADC Operation

The High Speed Successive Approximation Register (SAR) ADC is designed to support power conversion and motor control applications and consists of one shared ADC module. The shared ADC module has multiple analog inputs connected to its S&H circuit through a multiplexer. Multiple analog inputs share this ADC; therefore, it is termed the shared ADC module. The shared ADC module is used to measure analog signals of lower frequencies and signals that are static in nature (in other words, do not change significantly with time). However, this ADC module is capable of up to 2 Msps sample rate.

The analog inputs connected to the shared ADC module are Class 2 and Class 3 inputs. The number of inputs designated for each class depends on the specific device. For the PIC32CX-BZ2, the following arrangement is provided.

- Class 2 = AN0 to AN5
- Class 3 = AN6 to AN7

The property of each class of analog input is described in the following table.

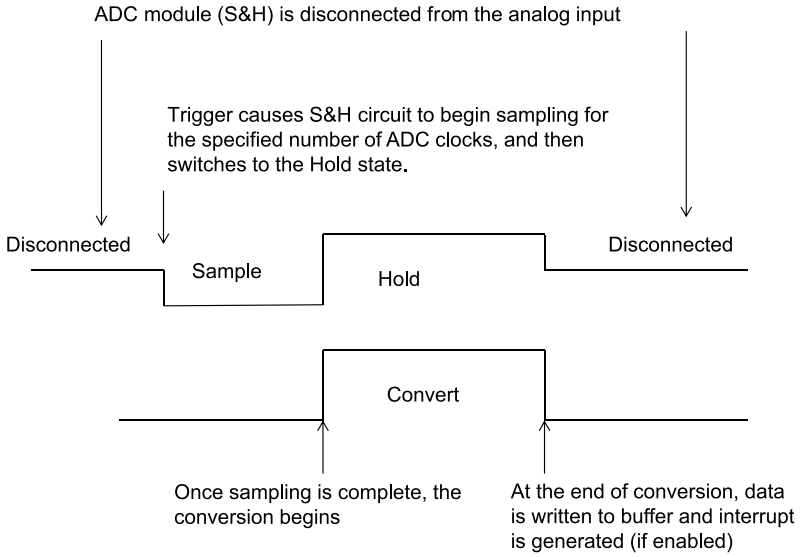
Table 38-1. Analog Input Class

ADC Module	Analog Input Class	Trigger	Trigger Action
Shared ADC module	Class 2	Individual trigger source or scan trigger	Starts sampling sequence or begins scan sequence
Shared ADC module with input scan	Class 3	Scan trigger	Starts scan sequence

Class 2 and Class 3 analog input properties:

- Class 2 inputs are used on the shared ADC module, either individually triggered or as part of a scan list. When used individually, they are triggered by their unique trigger selected by the ADCTRGx register.
- The analog inputs on the shared ADC have a natural order of priority (for example, AN6 has a higher priority than AN7).
- Class 3 inputs are used exclusively for scanning and share a common trigger source (scan trigger).
- Class 3 analog inputs share both the ADC module and the trigger source; therefore, the only method possible to convert them is to scan them sequentially for each incoming scan trigger event, where scanning occurs in the natural order of priority.
- The arrival of a trigger in the shared ADC module only starts the sampling. When the trigger arrives, the ADC module goes into sampling mode for the sampling time decided by the SAMC[9:0] bits (ADCCON2[25:16]). At the end of sampling, the ADC starts conversion. Upon completion of conversion, the ADC module is used to convert the next in line Class 2 or Class 3 inputs according to the natural order of priority. When a shared analog input (Class 2 or Class 3) has completed all conversion and no trigger is pending, the ADC module is disconnected from all analog inputs

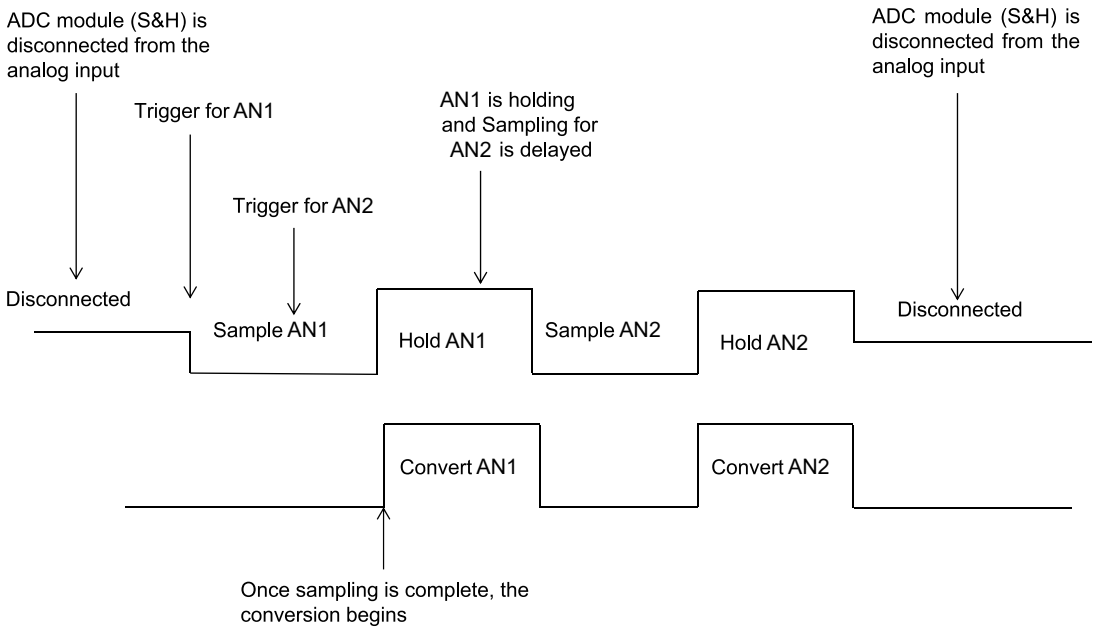
Figure 38-2. Sample and Conversion Sequence for Shared ADC Modules



38.2.1 Class 2 Triggering

When a single Class 2 input is triggered, it is sampled and converted by the shared S&H using the sequence illustrated in Sample and Conversion Sequence for the Shared ADC Modules figure; see *Sample and Conversion Sequence for Shared ADC Modules* figure in the *ADC Operation* from Related Links. When multiple Class 2 inputs are triggered, it is important to understand the consequences of trigger timing. If a conversion is underway and another Class 2 trigger occurs, then the sample-hold-conversion for the new trigger is stalled until the in-process, sample-hold cycle is complete, as shown in the following figure.

Figure 38-3. Multiple Independent Class 2 Trigger Conversion Sequence



When multiple inputs to the shared S&H are triggered simultaneously, the processing order is determined by their natural priority (the lowest numbered input has the highest priority). As an example, if AN1, AN2 and AN3 are triggered simultaneously, AN1 is sampled and converted first, followed by AN2 and finally, AN3. When using the independent Class 2 triggering on the shared S&H, the SAMC[9:0] bits (ADCCON2[25:16]) determine the sample time for all inputs while the appropriate TRGSRC[4:0] bits in the ADCTRGx Register (see *ADCTRG1* register from Related Links) determine the trigger source for each input.

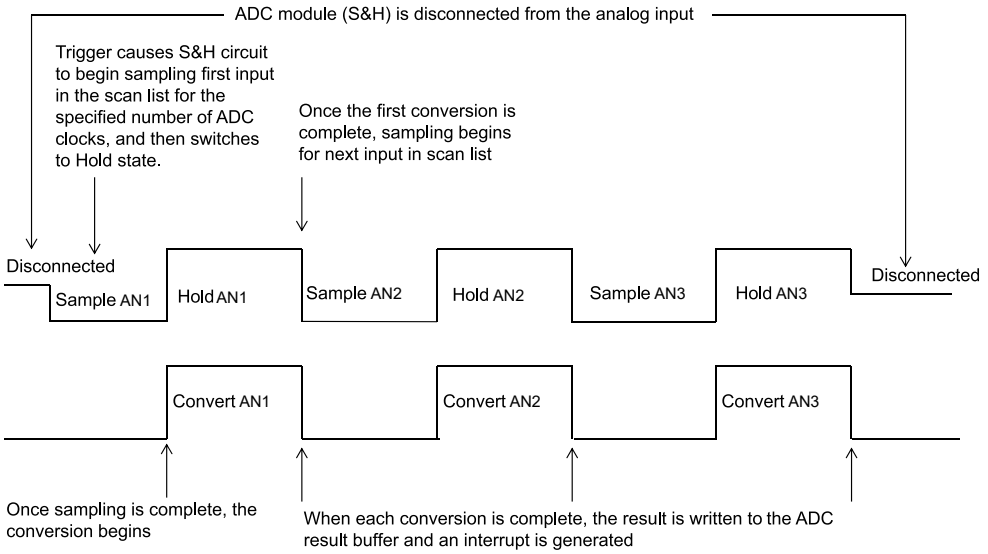
Related Links

38.11.15. ADCTRG1
38.2. ADC Operation

38.2.2 Input Scan

Input scanning is a feature that allows an automated scanning sequence of multiple Class 2 or Class 3 inputs. All Class 2 and Class 3 inputs are scanned using the single shared S&H. The selection of analog inputs for scanning is done with the CSSx bits of the ADCCSS1 registers. Class 2 inputs are triggered using STRIG selection in the ADCTRGx register, and Class 3 inputs are triggered using the STRGSRC[4:0] of the ADCCON1[20:16] register. When a trigger occurs for Class 2 or Class 3 inputs, the sampling and conversion occur in the natural input order is used; lower number inputs are sampled before higher number inputs.

Figure 38-4. Input Scan Conversion Sequence for Three Class 2 Inputs

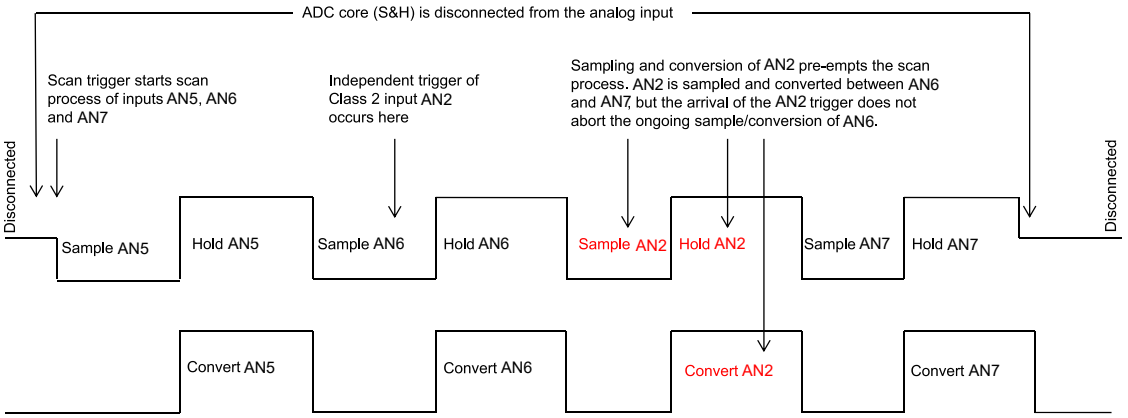


When using the shared analog inputs in scan mode, the SAMC[9:0] bits in the ADC Control Register 2 (ADCCON2[25:16]) determine the sample time for all inputs, while the Scan Trigger Source Selection bits (STRGSRC[4:0]) in the ADC Control Register 1 (ADCCON1[20:16]) determine the trigger source.

To ensure predictable results, a scan must not be retriggered until a sampling of all inputs is complete. Ensure system design to preclude retriggering a scan while a scan is in progress.

Individual Class 2 triggers that occur during a scan preempts the scan sequence if they are a higher priority than the sample currently being processed. In the following figure, a scan of AN5, AN6 and AN7 is underway when an independent trigger of Class 2 input AN2 takes place. The scan is interrupted for the sampling and conversion of AN2.

Figure 38-5. Scan Conversion Pre-empted by Class 2 Input Trigger



38.3 ADC Module Configuration

Operation of the ADC module is directed through bit settings in the specific registers. The following instructions summarize the actions and the settings. The options and details for each configuration step are provided in the subsequent sections.

To configure the ADC module, perform the following steps:

1. Configure the analog port pins as described in 38.3.1. Configuring the Analog Port Pins.
2. Select the analog inputs to the ADC multiplexers as described in 38.3.2. Selecting the ADC Multiplexer Analog Inputs.
3. Select the format of the ADC result as described in 38.3.3. Selecting the Format of the ADC Result.
4. Select the conversion trigger source as described in 38.3.4. Selecting the Conversion Trigger Source.
5. Select the voltage reference source as described in 38.3.5. Selecting the Voltage Reference Source.
6. Select the scanned inputs as described in 38.3.6. Selecting the Scanned Inputs.
7. Select the analog-to-digital conversion clock source and prescaler as described in 38.3.7. Selecting the Analog-to-Digital Conversion Clock Source and Prescaler.
8. Specify any additional acquisition time (if required) as described in 38.9. ADC Sampling Requirements.
9. Turn on the ADC module as described in 38.3.8. Turning ON the ADC.
10. Poll (or wait for the interrupt) for the voltage reference to be ready as described in 38.3.5. Selecting the Voltage Reference Source.
11. Enable the analog and bias circuit for the required ADC modules, and, after the ADC module wakes up, enable the digital circuit as described in 38.6.3. Low-Power Mode.
12. Configure the ADC interrupts (if required) as described in 38.5. Interrupts.

38.3.1 Configuring the Analog Port Pins

The ANSELx registers for the I/O ports associated with the analog inputs are used to configure the corresponding pin as an analog or a digital pin. A pin is configured as an analog input when the corresponding ANSELx bit = '1'. When the ANSELx bit = '0', the pin is set to digital control. The ANSELx registers are set when the device comes out of Reset, causing the ADC input pins to be configured as analog inputs by default.

The TRISx registers control the digital function of the port pins. The port pins that are required as analog inputs must have their corresponding bit set in the specific TRISx register, configuring the pin as an input. If the I/O pin associated with an ADC input is configured as an output by clearing the TRISx bit, the port's digital output level (V_{OH} or V_{OL}) is converted. After a device Reset, all of the TRISx bits are set. For more information on port pin configuration, see *I/O Ports and Peripheral Pin Select (PPS)* from Related Links.

Note: When reading a PORT register that shares pins with the ADC, any pin configured as an analog input reads as '0' when the PORT latch is read. Analog levels on any pin that is defined as a digital input but not configured as an analog input, may cause the input buffer to consume the current that exceeds the device specification.

Related Links

6. I/O Ports and Peripheral Pin Select (PPS)

38.3.2 Selecting the ADC Multiplexer Analog Inputs

The ADC module has two inputs, referred to as the positive and negative inputs. Input selection options vary as described in the following sections.

38.3.2.1 Selection of Positive Inputs

For the shared ADC module, the positive input is shared among all Class 2 and Class 3 inputs. Input connection of the analog input ANx to the shared ADC is automatic for either the Class 2 input trigger or during a scan of Class 2 and or Class 3 inputs. Selecting inputs for scanning is described in *Selecting the Scanned Inputs* from Related Links.

Related Links

38.3.6. Selecting the Scanned Inputs

38.3.2.2 Selection of Negative Inputs

Negative input selection is determined by the setting of the DIFFx bit of the ADCIMCON1 register. The DIFFx bit allows the inputs to be rail-to-rail and either single-ended or differential. The SIGNx and DIFFx bits in the ADCIMCON1 register scale the internal ADC analog inputs and reference voltages and configure the digital result to align with the expected full-scale output range.

For the shared ADC module, the analog inputs have individual settings for the DIFFx bit. Therefore, the user has the ability to select certain inputs as single-ended and others as differential while being connected to the same shared ADC module. While sampling, the signal changes on-the-fly as single-ended or differential according to its corresponding DIFFx bit setting.

Table 38-2. Negative Input Selection

ADCIMCON1		Input Configuration	Input Voltage		Output
DIFFx	SIGNx				
1	1	Differential 2's complement	Minimum input	$V_{IN}P - V_{IN}N = -V_{REF}$	-2048
			Maximum input	$V_{IN}P - V_{IN}N = V_{REF}$	+2047
1	0	Differential unipolar	Minimum input	$V_{IN}P - V_{IN}N = -V_{REF}$	0
			Maximum input	$V_{IN}P - V_{IN}N = V_{REF}$	+4095
0	1	Single-ended 2's complement	Minimum input	$V_{IN}P = V_{REF}$	-2048
			Maximum input	$V_{IN}P - V_{IN}N = V_{REF}$	+2047
0	0	Single-ended unipolar	Minimum input	$V_{IN}P = V_{REF}$	0
			Maximum input	$V_{IN}P - V_{IN}N = V_{REF}$	+4095

Legend:

- $V_{IN}P$ = Positive S&H input
- $V_{IN}N$ = Negative S&H input
- $V_{REF} = V_{REFH} - V_{REFL}$

Note: For proper operation and to prevent device damage, input voltage levels must not exceed the limits listed in the Electrical Specifications.

38.3.3 Selecting the Format of the ADC Result

The data in the ADC Result register can be read in any of the four supported data formats. The user can select from unsigned integer, signed integer, unsigned fractional or signed fractional. Integer data is right-justified and fractional data is left-justified.

- The integer or fractional data format selection is specified globally for all analog inputs using the Fractional Data Output Format bit, FRACT (ADCCON1[23]).
- The signed or unsigned data format selection can be independently specified for each individual analog input using the SIGNx bits in the ADCIMCONx registers

  .

The following table provides how a result is formatted.

Table 38-3. ADC Result Format

FRACT	SIGNx	Description	32-bit Output Data Format			
0	0	Unsigned integer	0000	0000	0000	0000
			0000	dddd	dddd	dddd

.....continued						
FRACT	SIGNx	Description	32-bit Output Data Format			
0	1	Signed integer	ssss	ssss	ssss	ssss
			ssss	sddd	dddd	dddd
1	0	Fractional	dddd	dddd	dddd	0000
			0000	0000	0000	0000
1	1	Signed fractional	sddd	dddd	dddd	dddd
			0000	0000	0000	0000

The following code is an example for ADC Class 2 configuration and fractional format.

```
int main(int argc, char** argv) {
int result[3];

/* Configure ADCCON1 */
ADCCON1bits.FRACT = 1;    // use Fractional output format ADCCON1bits.SELRES = 3;   // ADC
resolution is 12 bits ADCCON1bits.STRGSRC = 0;  // No scan trigger.

/* Configure ADCCON2 */
ADCCON2bits.SAMC = 5;         // ADC sampling time = 5 * TAD7
ADCCON2bits.ADCDIV = 1;       // ADC clock freq is half of control clock = TAD7

/* Initialize warm up time register */ ADCANCON = 0;
ADCANCONbits.WKUPCLKCNT = 5;  // Wakeup exponent = 32 * TADx

/* Clock setting */ ADCCON3 = 0;
ADCCON3bits.ADCSEL = 0;       // Select input clock source
ADCCON3bits.CONCLKDIV = 1;    // Control clock frequency is half of input clock
ADCCON3bits.VREFSEL = 0;      // Select AVDD and AVSS as reference source

/* No selection for dedicated ADC modules, no presync trigger, not sync sampling */
ADCTRGMODEbits = 0;

/* Select ADC input mode */
ADCIMCON1bits.SIGN7 = 0;  // unsigned data format ADCIMCON1bits.DIFF7 = 0;  // Single ended
mode ADCIMCON1bits.SIGN8 = 0;  // unsigned data format ADCIMCON1bits.DIFF8 = 0;  // Single
ended mode ADCIMCON1bits.SIGN9 = 0;  // unsigned data format ADCIMCON1bits.DIFF9 = 0;  //
Single ended mode

/* Configure ADCGIRQENx */
ADCGIRQEN1 = 0;           // No interrupts are used
ADCGIRQEN2 = 0;

/* Configure ADCCSSx */
ADCCSS1 = 0;              // No scanning is used
ADCCSS2 = 0;

/* Configure ADCCMPCONx */
ADCCMPCON1 = 0;           // No digital comparators are used. Setting the ADCCMPCONx
ADCCMPCON2 = 0;           // register to '0' ensures that the comparator is disabled.
ADCCMPCON3 = 0;           // Other registers are “don't care”.
ADCCMPCON4 = 0;
ADCCMPCON5 = 0; ADCCMPCON6 = 0;

/* Configure ADCFLTRx */
ADCFLTR1 = 0;             // No oversampling filters are used. ADCFLTR2 = 0;
ADCFLTR3 = 0; ADCFLTR4 = 0; ADCFLTR5 = 0; ADCFLTR6 = 0;
/* Set up the trigger sources */
ADCTRGSNSbits.LVL7 = 0;   // Edge trigger ADCTRGSNSbits.LVL8 = 0;   // Edge trigger
ADCTRGSNSbits.LVL9 = 0;   // Edge trigger
ADC1TRG2bits.TRGSRC7 = 1; // Set AN7 to trigger from software
ADC2TRG3bits.TRGSRC8 = 1; // Set AN8 to trigger from software
ADC2TRG3bits.TRGSRC9 = 1; // Set AN9 to trigger from software
/* Early interrupt */
ADCEIEN1 = 0;               // No early interrupt
ADCEIEN2 = 0;

/* Turn the ADC on */ ADCCON1bits.ON = 1;

/* Wait for voltage reference to be stable */
```

```
while(!ADCCON2bits.BGVRRDY); // Wait until the reference voltage is ready
while(ADCCON2bits.REFFLT);   // Wait if there is a fault with the reference voltage

/* Enable clock to analog circuit */
ADCANCONbits.ANEN7 = 1;       // Enable the clock to analog bias

/* Wait for ADC to be ready */
while(!ADCANCONbits.WKRDY7); // Wait until ADC7 is ready

/* Enable the ADC module */ ADCCON3bits.DIGEN7 = 1;       // Enable ADC7

while (1) {
/* Trigger a conversion */ ADCCON3bits.GSWTRG = 1;

/* Wait the conversions to complete */
while (ADCDSTAT1bits.ARDY7 == 0);
/* fetch the result */
result[0] = ADCDATA7;

while (ADCDSTAT1bits.ARDY8 == 0);
/* fetch the result */
result[1] = ADCDATA8;

while (ADCDSTAT1bits.ARDY9 == 0);
/* fetch the result */
result[2] = ADCDATA9;

/*
* Process results here
*
* Note 1: Loop time determines the sampling time since all inputs are Class 2.
* If the loop time happens is small and the next trigger happens before the
* completion of set sample time, the conversion will happen only after the
* sample time has elapsed.
*
* Note 2: Results are in fractional format
*
*/
}
return (1);
}
```

38.3.4 Selecting the Conversion Trigger Source

Class 2 inputs to the ADC module can be triggered for conversion either individually or as part of a scan sequence. Class 3 inputs can only be triggered as part of a scan sequence. Individual or scan triggers can originate from an on-board timer or output compare peripheral event, from external digital circuits connected to INT0, from external analog circuits connected to an analog comparator or through software by setting a trigger bit in an SFR.

Note: When conversion triggers for multiple Class 2 analog inputs occur simultaneously, they are prioritized according to a natural order priority scheme based on the analog input used. AN6 has the highest priority, AN7 has the next highest priority and so on.

38.3.4.1 Trigger Selection Class 2 Inputs

For each one of the Class 2 inputs, the user application can independently specify a conversion trigger source. The individual trigger source for an analog input 'x' is specified by the TRGSRC[4:0] bits located in registers ADCTRG1 through ADCTRG3. For example, these trigger sources may include:

- **General Purpose (GP) Timers**: When a period match occurs for the 32-bit timer, Timer3/2 or Timer5/4, or the 16-bit Timer1, Timer3 or Timer5, a special ADC trigger event signal is generated by the timer. This feature does not exist for other timers. For more information, see *Timer/Counter (TC)* from Related Links.
- **Output Compare**: The Output Compare peripherals, OC1, OC3 and OC5, can be used to generate an ADC trigger, then the output transitions from a low to high state. For more information, see *Timer/Counter (TC)* from Related Links.

- **Comparators**: The analog Comparators can be used to generate an ADC trigger when the output transitions from a low state to a high state. For more information, see *Digital Comparator* from Related Links.
- **External INT0 Pin Trigger**: In this mode, the ADC module starts a conversion on an active transition on the INT0 pin. The INT0 pin may be programmed for either a rising edge input or a falling edge input to trigger the conversion process.
- **Global Software Trigger**: The ADC module can be configured for manually triggering a conversion for all inputs that have selected this trigger option. The user can manually trigger a conversion by setting the Global Software Trigger bit, GSWTRG (ADCCON3[6]).

Related Links

38.4.1. Digital Comparator
40. Timer/Counter (TC)

38.3.4.2 Conversion Trigger Sources and Control

The following are the possible sources for each trigger signal:

- External trigger selection through the TRGSRCx[4:0] bits in the ADCTRGx registers. This capability is supported only for Class 2 analog inputs. Typically, the user specifies a particular trigger source to initiate a conversion for specific input. All of the analog inputs may select the same trigger source if desired. In such an event, the result resembles a “scanned conversion”, which has its order of completion enforced by the priority of the inputs associated with the same trigger source. The first trigger selection is `00000` (no trigger), which amounts to temporarily disabling that particular trigger and, consequently, temporarily disabling that analog input from being converted. The next two selections for trigger source (GSWTRG and GLSWTRG) are software-generated trigger sources. The second software-generated trigger selection is the Global Software Trigger (GSWTRG). This trigger links to the GSWTRG bit in the ADCCON3 register, which may be used to enable the user application to initiate a single conversion. GSWTRG is a self-clearing bit; therefore, it clears itself on the next ADC clock cycle after being set by the user application. The third software-generated trigger selection is the Global Level Software Trigger (GLSWTRG), which is linked to the GLSWTRG bit in the ADCCON3 register. This trigger may be used by the user application to initiate a burst of consecutive samples as the GLSWTRG bit is not self-clearing. The fourth trigger selection is a special selection, the Scan Trigger selection, which allows the Class 2 analog inputs to be included as members of a global scan of all inputs.
- Scanned trigger selection via the STRGSRC[4:0] bits in the ADCCON1 register and select bits in the ADCCSS1 registers. This mode is typically used to initiate the conversion of a group of analog inputs. This capability works for 2 and 3 analog inputs but is typically used for Class 3 inputs because they do not have individual associated TRGSRC bits. One of the trigger selections is the GSWTRG bit in the ADCCON3 register, which may be used to enable the user software to initiate a conversion.
- User initiated trigger via the ADINSEL[5:0] bits and the RQCNVRT bit in the ADCCON3 register. This mode enables the user application to create an individual conversion trigger request for a specified analog input. Using this mode enables the user application to trigger the conversion of an input without changing the trigger source configuration of the ADC. This is useful in handling error situations where another software module wants ADC information without disrupting the normal operation of the ADC. This is also the preferred method to generate the initial trigger to start a digital filter sequence.
- User-controlled sampling of Class 2 and Class 3 inputs via the ADINSEL[5:0] bits and the SAMP bit in the ADCCON3 register. Setting the SAMP bit causes the Class 2 and Class 3 inputs to be in Sampling mode while ignoring the selection of the SAMC[9:0] bits. This mode is also useful in software conversion of ADC with software-selectable sample time.
- External module (such as PTG) may specify an analog input for conversion via the setting of the ECRIEN bit in the ADCCON2 register. This method operates independently of the normal TRGSRC

and STRGSRC methods. External modules may still use individual trigger signals and initiate conversions via the normal TRGSRC and STRGSRC methods.

38.3.4.3 User-Requested Individual Conversion Trigger (Software ADC Conversion) (Only for Class 2 and Class 3 Inputs)

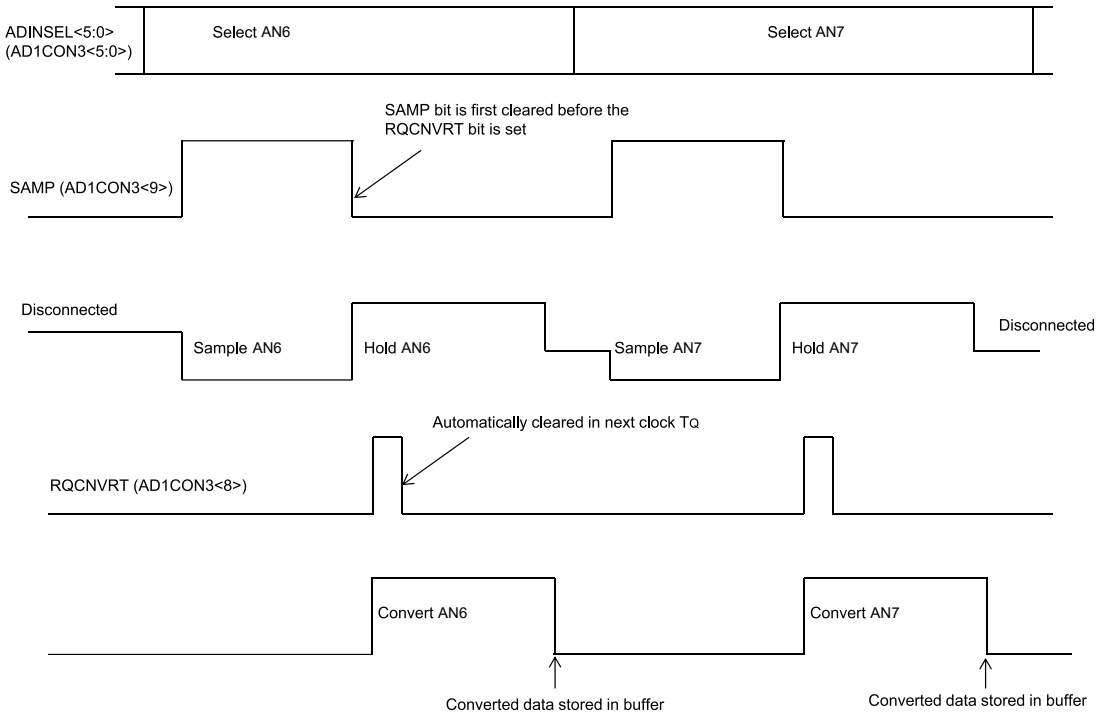
The user can explicitly request a single conversion (by software) of any selected analog input at any time during program execution without changing the trigger source configuration of the ADC.

The steps to be followed for conversion are as follows:

1. The analog input ID to be converted is specified by the ADC Input Select bits, ADINSEL[5:0] (ADCCON3[5:0]).
2. The sampling of analog input is started by setting the SAMP bit (ADCCON3[9]).
3. After the required sampling time (time delay), the SAMP bit is cleared.
4. The conversion of sampled signal is started by setting the RQCNVRT bit (ADCCON3[8]).
5. Once the conversion is complete, the ARDYx bit of the ADCDSTATx register is set. The data can be read from the ADCDATAx register.

The following figure illustrates the conversion process in graphical form.

Figure 38-6. Individual Conversion Trigger Process



38.3.5 Selecting the Voltage Reference Source

The user application can select the voltage reference for the ADC module, which can be internal or external. The Voltage Reference Input Selection bits, VREFSEL[2:0] (ADCCON3[15:13]), select the voltage reference for analog-to-digital conversions. The upper voltage reference (V_{REFH}) and the lower voltage reference (V_{REFL}) may be the internal AV_{DD} and AV_{SS} voltage rails or the band gap reference generator or the external V_{REFH}+ and V_{REF}- input pins. When the voltage reference and band gap reference are ready, the BGVRRDY (ADCCON2[31]) bit is set. If a Fault occurs in the voltage

reference (such as a brown-out), the REFFLT bit (ADCCON2[30]) is set. The BGVRRDY and REFFLT bits can also generate interrupts if the BGVRIEN bit (ADCCON2[15]) and REFFLTIEN bit (ADCCON2[14]) are set, respectively.

The voltages applied to the external reference pins must comply with certain specifications. See *Electrical Characteristics* from Related Links.

The Analog Input Charge Pump Enable bit, AICPMPEN (ADCCON1[12]), must be set when the difference between the selected reference voltages (V_{REFH} - V_{REFL}) is less than 0.65 * (AV_{DD} - AV_{SS}). Setting this bit does not increase the magnitude of the reference voltage; however, setting this bit reduces the series source resistance to the sampling capacitors. This maximizes the SNR for analog-to-digital conversions using small reference voltage rails.

Related Links

43. Electrical Characteristics

38.3.6 Selecting the Scanned Inputs

All available analog inputs can be configured for scanning. Class 2 and Class 3 inputs are sampled using the shared ADC module. A single conversion trigger source is selected for all of the inputs selected for scanning using the STRGSRC[4:0] bits (ADCCON1[20:16]). On each conversion trigger, the ADC module starts converting (in the natural priority) all inputs specified in the user-specified scan list (ADCCSS1 or ADCCSS2). For Class 2 and Class 3 inputs, the trigger initiates a sequential sample/conversion process in the natural priority order.

An analog input belongs to the scan if it is:

- A Class 3 input. For Class 3 inputs, scan is the only mechanism for conversion.
- A Class 2 input that has the scan trigger selected as the trigger source by selecting the STRIG option in the TRGSRCx[4:0] bits located in the ADCTRG1 through ADCTRG8 registers.

The trigger options available for scan are identical to those available for independent triggering of Class 2 inputs. Any Class 2 inputs that are part of the scan must have the STRIG option selected as their trigger source in the TRGSRCx[4:0] bits.

Note: The end-of-scan (EOS) is generated only if the last shared input conversion has completed. Until this condition is met, the scan sequence is still in effect. Therefore, the EOS Interrupt can be used for any scan sequence with any combination of input types.

The following code is an example for ADC scanning multiple inputs.

```
int main(int argc, char** argv) {
int result[3];

/* Configure ADCCON1 */
ADCCON1 = 0;              // No ADCCON1 features are enabled including: Stop-in-Idle, turbo,
// CVD mode, Fractional mode and scan trigger source. ADCCON1bits.SELRES = 3;   // ADC7
resolution is 12 bits
ADCCON1bits.STRGSRC = 1;  // Select scan trigger.

/* Configure ADCCON2 */
ADCCON2bits.SAMC = 5;     // ADC7 sampling time = 5 * TAD7
ADCCON2bits.ADCDIV = 1;   // ADC7 clock freq is half of control clock = TAD7

/* Initialize warm up time register */ ADCANCON = 0;
ADCANCONbits.WKUPCLKCNT = 5; // Wakeup exponent = 32 * TADx

/* Clock setting */
ADCCON3bits.ADCSEL = 0;       // Select input clock source
ADCCON3bits.CONCLKDIV = 1;    // Control clock frequency is half of input clock
ADCCON3bits.VREFSEL = 0;      // Select AVDD and AVSS as reference source

ADC0TIMEbits.ADCDIV = 1;      // ADC0 clock frequency is half of control clock = TAD0
ADC0TIMEbits.SAMC = 5;        // ADC0 sampling time = 5 * TAD0
ADC0TIMEbits.SELRES = 3;      // ADC0 resolution is 12 bits

/* Select analog input for ADC modules, no presync trigger, not sync sampling */
```

```
ADCTRGMODEbits.SH0ALT = 0;    // ADC0 = AN0

/* Select ADC input mode */
ADCIMCON1bits.SIGN0 = 0;      // unsigned data format ADCIMCON1bits.DIFF0 = 0;      //
Single ended mode ADCIMCON1bits.SIGN8 = 0;      // unsigned data format ADCIMCON1bits.DIFF8
= 0;      // Single ended mode ADCIMCON1bits.SIGN40 = 0;     // unsigned data format
ADCIMCON1bits.DIFF40 = 0;     // Single ended mode

/* Configure ADCGIRQENx */
ADCGIRQEN1 = 0;               // No interrupts are used. ADCGIRQEN2 = 0;

/* Configure ADCCSSx */
ADCCSS1 = 0;                 // Clear all bits
ADCCSS2 = 0;
ADCCSS1bits.CSS0 = 1;         // AN0 (Class 1) set for scan ADCCSS1bits.CSS8 = 1;         //
AN8 (Class 2) set for scan ADCCSS2bits.CSS40 = 1;        // AN40 (Class 3) set for scan

/* Configure ADCCMPCONx */
ADCCMPCON1 = 0;               // No digital comparators are used. Setting the ADCCMPCONx
ADCCMPCON2 = 0;               // register to '0' ensures that the comparator is disabled.
ADCCMPCON3 = 0;               // Other registers are 'don't care'.
ADCCMPCON4 = 0;
ADCCMPCON5 = 0; ADCCMPCON6 = 0;

/* Configure ADCFLTRx */
ADCFLTR1 = 0;                // No oversampling filters are used. ADCFLTR2 = 0;
ADCFLTR3 = 0; ADCFLTR4 = 0; ADCFLTR5 = 0; ADCFLTR6 = 0;
/* Set up the trigger sources */
ADCTRG1bits.TRGSRC0 = 3;      // Set AN0 (Class 1) to trigger from scan source
ADCTRG3bits.TRGSRC8 = 3;      // Set AN8 (Class 2) to trigger from scan source
// AN40 (Class 3) always uses scan trigger source

/* Early interrupt */
ADCEIEN1 = 0;                // No early interrupt
ADCEIEN2 = 0;

/* Turn the ADC on */ ADCCON1bits.ON = 1;

/* Wait for voltage reference to be stable */
while(!ADCCON2bits.BGVRRDY); // Wait until the reference voltage is ready
while(ADCCON2bits.REFFLT);   // Wait if there is a fault with the reference voltage

/* Enable clock to analog circuit */
ADCANCONbits.ANEN0 = 1;       // Enable the clock to analog bias ADC0
ADCANCONbits.ANEN7 = 1;       // Enable, ADC7

/* Wait for ADC to be ready */
while(!ADCANCONbits.WKRDY0);     // Wait until ADC0 is ready while(!
ADCANCONbits.WKRDY7);     // Wait until ADC7 is ready

/* Enable the ADC module */
ADCCON3bits.DIGEN0 = 1;          // Enable ADC0
ADCCON3bits.DIGEN7 = 1;          // Enable ADC7

while (1) {
/* Trigger a conversion */ ADCCON3bits.GSWTRG = 1;

/* Wait the conversions to complete */
while (ADCDSTAT1bits.ARDY0 == 0);
/* fetch the result */
result[0] = ADCDATA0;

while (ADCDSTAT1bits.ARDY8 == 0);
/* fetch the result */
result[1] = ADCDATA8;

while (ADCDSTAT2bits.ARDY40 == 0);
/* fetch the result */
result[2] = ADCDATA40;

/*
* Process results here
*
*
*/
}
```

```
return (1);
}
```

38.3.7 Selecting the Analog-to-Digital Conversion Clock Source and Prescaler

The ADC module can use the internal Fast RC (FRC) oscillator output, system clock (SYSCLK), reference clock (REFCLK3) or peripheral bus clock (PBCLK) as the conversion clock source (T_Q). See *ADCCON3* register from Related Links.

When the ADCSEL[1:0] bits (ADCCON2[31:30]) = '01', the internal FRC oscillator is used as the ADC clock source. When using the internal FRC oscillator, the ADC module can continue to function in Sleep and Idle modes.

Note: It is recommended that applications that require precise timing of ADC acquisitions use SYSCLK as the clock source for the ADC.

For correct analog-to-digital conversions, the conversion clock limits must not be exceeded. Clock frequencies from 1 MHz to 28 MHz are supported by the ADC module.

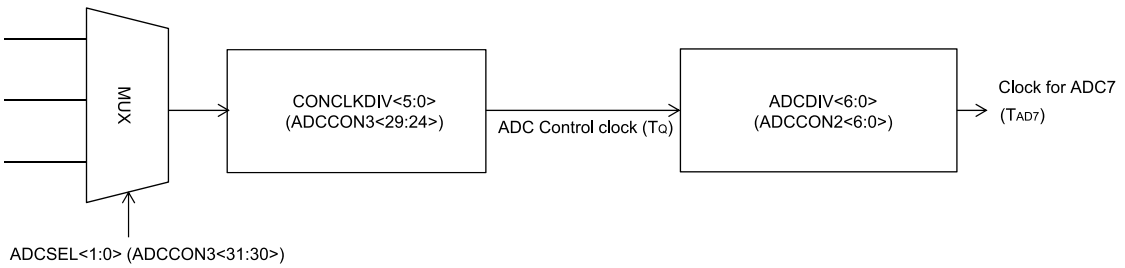
The maximum rate that analog-to-digital conversions may be completed by the ADC module (effective conversion throughput) is 2 Msps. However, the maximum rate that a single input can be converted is dependent on the sampling time requirements. In addition, the sampling time depends on the output impedance of the analog signal source. For more information on sampling time, see *ADC Sampling Requirements* from Related Links.

The input clock source for the ADC is selected using the ADCSEL[1:0] bits (ADCCON3[31:30]). The input clock is further divided by the control clock divider CONCLKDIV[5:0] bits (ADCCON3[29:24]). The output clock is called the "ADC control clock" with a time period of T_Q.

The ADC control clock is divided by the ADCDIV[6:0] bits (ADCxTIME[22:16]). This acts as the clock source for the respective dedicated ADC modules with a time period of T_{ADX}.

The ADC control clock is divided before it is used for the shared ADC by the ADCDIV[6:0] bits (ADCCON2[6:0]). The time period for this clock is denoted as T_{AD7}.

Figure 38-7. Clock Derivation for Shared ADC Modules



Equation 38-2. Sample Time for the Shared ADC Module

$$t_{SAMC} = ADCCON2 < 25:16 > T_{AD}$$

$$t_{conversion} = 2 + ADCCON2 < 22:21 > T_{AD}$$

Related Links

38.11.4. ADCCON3

38.9. ADC Sampling Requirements

38.3.8 Turning ON the ADC

Turning ON the ADC module involves the following procedure.

When the ADC module enable bit, ON (ADCCON1[15]), is set to '1', the module is in Active mode and is fully powered and functional. When the ON bit is '0', the ADC module is disabled. Once disabled, the digital and analog portions of the ADC are turned off for maximum current savings. In addition to setting the ON bit, the analog and digital circuits of ADC must be turned ON. See *Low-power Mode* from Related Links.

Note: Writing to the ADC control bits that control the ADC clock, input assignments, scanning, voltage reference selection, S&H circuit operating modes and interrupt configuration is not recommended while the ADC module is enabled.

Related Links

38.6.3. Low-Power Mode

38.3.9 ADC Status Bits

The ADC module includes the WKRDYx/WKRDY7 status bit in the ADCANCON register, which indicates the current state of ADC Analog and bias circuit. The user application must not perform any ADC operations until this bit is set.

38.4 Additional ADC Functions

This section describes some additional features of the ADC module, which includes:

- Digital comparator
- Oversampling filter

38.4.1 Digital Comparator

The ADC module features digital comparators that can be used to monitor selected analog input conversion results and generate interrupts when a conversion result is within the user-specified limits. Conversion triggers are still required to initiate conversions. The comparison occurs automatically once the conversion is complete. This feature is enabled by setting the Digital Comparator Module Enable bit, ENDCMP (ADCCMPCONx[7]).

The user application makes use of an interrupt that is generated when the analog-to-digital conversion result is higher or lower than the specified high and low limit values in the ADCCMPx register. The high and low limit values are specified in the DCMPHI[15:0] bits (ADCCMPx[31:16]) and the DCMPLO[15:0] bits (ADCCMPx[15:0]).

The CMPEx bits ('x' = 0 through 31) in the ADCCMPENx registers are used to specify which analog inputs are monitored by the digital comparator (for the first 8 analog inputs, ANx, where 'x' = 0 through 31). The ADCCMPCONx register specifies the comparison conditions that generates an interrupt, as follows:

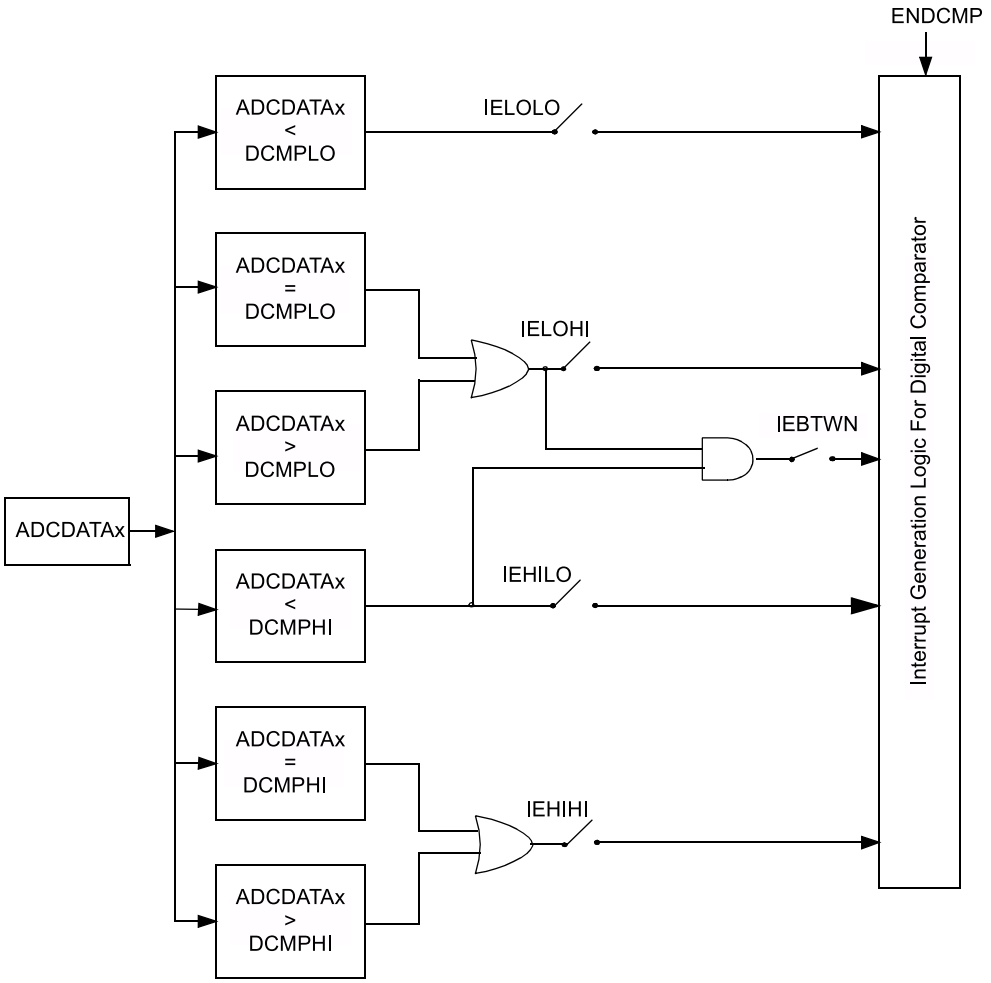
- When IEBTWN = 1, an interrupt is generated when DCMPLO ≤ ADCDATA < DCMPHI
- When IEHIHI = 1, an interrupt is generated when DCMPHI ≤ ADCDATA
- When IEHILO = 1, an interrupt is generated when ADCDATA < DCMPHI
- When IELOHI = 1, an interrupt is generated when DCMPLO ≤ ADCDATA
- When IELOLO = 1, an interrupt is generated when ADCDATA < DCMPLO

The comparator event generation is illustrated in the following figure. When the ADC module generates a conversion result, the conversion result is provided to the comparator. The comparator uses the DIFFx and SIGNx bits of the ADCIMCONx register (depending on the analog input used) to determine the data format used and to appropriately select whether the comparison must be signed or unsigned. The global ADC setting, which is specified by the FRACT bit (ADCCON1[23]), is also used to set the fractional or integer format. The digital comparator compares the ADC result with the high and low limit values (depending on the selected comparison criteria) in the ADCCMPx register.

Depending on the comparator results, a digital comparator interrupt event may be generated. If a comparator event occurs, the Digital Comparator Interrupt Event Detected status bit, DCMPED (ADCCMPCONx[5]), is set, and the Analog Input Identification (ID) bits, AINID[4:0] (ADCCMPCONx[12:8]), are automatically updated so that the user application knows which analog input generated the interrupt event.

Note: The user software must format the values contained in the ADCCMPx registers to match converted data format as either signed or unsigned, and fractional or integer.

Figure 38-8. Digital Comparator



The following code is an example for ADC digital comparator.

```
int main(int argc, char** argv) {
int result = 0, eventFlag = 0;

/* Configure ADCCON1 */
ADCCON1 = 0;                 // No ADCCON1 features are enabled including: Stop-in-Idle,
// turbo, CVD mode, Fractional mode and scan trigger source. ADCCON1bits.SELRES = 3;       //
ADC resolution is 12 bits
ADCCON1bits.STRGSRC = 0;      // No scan trigger.

/* Configure ADCCON2 */
ADCCON2bits.SAMC = 5;         // ADC7 sampling time = 5 * TAD7
ADCCON2bits.ADCDIV = 1;       // ADC7 clock freq = TAD7

/* Initialize warm up time register */ ADCANCON = 0;
```

```
/* No selection for dedicated ADC modules, no presync trigger, not sync sampling */
ADCTRGMODEbits = 0;

/* Select ADC input mode */
ADCIMCON1bits.SIGN8 = 0;      // unsigned data format
ADCIMCON1bits.DIFF8 = 0;      // Single ended mode

/* Configure ADCGIRQENx */
ADCGIRQEN1 = 0;               // No interrupts are used
ADCGIRQEN2 = 0;

/* Configure ADCCSSx */
ADCCSS1 = 0;                 // No scanning is used
ADCCSS2 = 0;

/* Configure ADCCMPCONx */
ADCCMP1 = 0;                 // Clear the register ADCCMP1bits.DCMPHI = 0xC00;  // High
limit is a 3072 result. ADCCMP1bits.DCMPLO = 0x500;  // Low limit is a 1280 result.
ADCCMPCON1bits.IEBTWN = 1;    // Create an event when the measured result is
// >= low limits and < high limit. ADCCMPEN1 = 0;                // Clear all enable bits
ADCCMPEN1bits.CMPE8 = 1;      // set the bit corresponding to AN8
ADCCMPCON1bits.ENDCMP = 1;    // enable comparator
ADCCMPCON2 = 0; ADCCMPCON3 = 0; ADCCMPCON4 = 0; ADCCMPCON5 = 0; ADCCMPCON6 = 0;

/* Configure ADCFLTRx */
ADCFLTR1 = 0;                // No oversampling filters are used. ADCFLTR2 = 0;
ADCFLTR3 = 0; ADCFLTR4 = 0; ADCFLTR5 = 0; ADCFLTR6 = 0;
/* Set up the trigger sources */
ADCTRG3bits.TRGSRC8 = 3;         // Set AN8 (Class 2) to trigger from scan source

/* Early interrupt */
ADCEIEN1 = 0;                // No early interrupt
ADCEIEN2 = 0;

/* Turn the ADC on */ ADCCON1bits.ON = 1;

/* Wait for voltage reference to be stable */
while(!ADCCON2bits.BGVRRDY);     // Wait until the reference voltage is ready
while(ADCCON2bits.REFFLT);       // Wait if there is a fault with the reference voltage

/* Enable clock to analog circuit */
ADCANCONbits.ANEN7 = 1;          // Enable the clock to analog bias

/* Wait for ADC to be ready */
while(!ADCANCONbits.WKRDY7);     // Wait until ADC7 is ready

/* Enable the ADC module */
ADCCON3bits.DIGEN7 = 1;          // Enable ADC7

while (1) {
/* Trigger a conversion */ ADCCON3bits.GSWTRG = 1;

while (ADCDSTAT1bits.ARDY8 == 0);
/* fetch the result */
result = ADCDATA8;

/* Note: It is not necessary to fetch the result for the digital
* comparator to work. In this example we are triggering from
* software so we are using the ARDY8 to gate our loop. Reading the
* data clears the ARDY bit.
*/
/* See if we have a comparator event*/
if (ADCCMPCON1bits.DCMPED == 1) {
eventFlag = 1;
/*
* Process results here
*/
}
}
```

```
return (1);
}
```

38.4.2 Oversampling Digital Filter

The ADC module supports two oversampling digital filters. The oversampling digital filter consists of an accumulator and a decimator (down-sampler), which function together as a low-pass filter. By sampling an analog input at a higher-than-required sample rate, then processing the data through the oversampling digital filter, the effective resolution of the ADC module can be increased at the expense of decreased conversion throughput.

To obtain 'x' bits of extra resolution, the number of samples required (over and above the Nyquist rate) = $(2^x)^2$:

- 4x oversampling yields one extra bit of resolution (total 13 bits resolution)
- 16x oversampling yields two extra bits of resolution (total 14 bits resolution)
- 64x oversampling provides three extra bits of resolution (total 15 bits resolution)
- 256x oversampling provides four extra bits of resolution (total 16 bits resolution)

The digital filter also has an averaging mode, where it accumulates the samples and divides it by the number of samples.

Note:

1. Only Class 2 analog inputs can engage the digital filter. Therefore, the CHNLID[2:0] bits are 3 bits wide (0 to 7).
2. During the burst conversion process (repeated trigger until all required data for oversampling is obtained), in the case of filtering Class 2 input using the shared ADC module, higher priority ADC inputs may still process conversions; lower priority ADC conversion requests are held waiting until the filter burst sequence is completed.
3. If higher priority requests occur during the digital filter sequence, they delay the completion of the filtering process. This delay may affect the accuracy of the result because the multiple samples cannot be contiguous. The user must arrange the initiation trigger for the oversampling filters to occur while there are no expected interruptions from higher priority ADC conversion requests.

The user application must configure the following bits to perform an oversampling conversion:

- Select the amount of oversampling through the Oversampling Filter Oversampling Ratio (OVRSAM[2:0]) bits in the ADC Filter register (ADCFLTRx[28:26]).
- Set the filter mode to either Oversampling mode or Averaging mode using the DFMODE bit(ADCFLTRx[29]).
- If the filter is set to Averaging mode and the data format is set to fractional (FRACT bit), set or clear the DATA16EN bit (ADCFLTRx[30]) to set the output resolution.
- Set the sample time for subsequent samples:
  - If using Class 2 inputs, select the sample time using the SAMC[9:0] bits (ADC- CON2[25:16]).
- Select the specific analog input to be oversampled by configuring the Analog Input ID Selection bits, CHNLID[4:0] (ADCFLTRx[20:16]).
- If needed, include the oversampling filter interrupt event in the global ADC interrupt by setting the Accumulator Filter Global Interrupt Enable bit, AFGIEN (ADCFLTRx[25]).
- Enable the oversampling filter by setting the Oversampling Filter Accumulator Enable bit, AFEN (ADCFLTRx[31]).

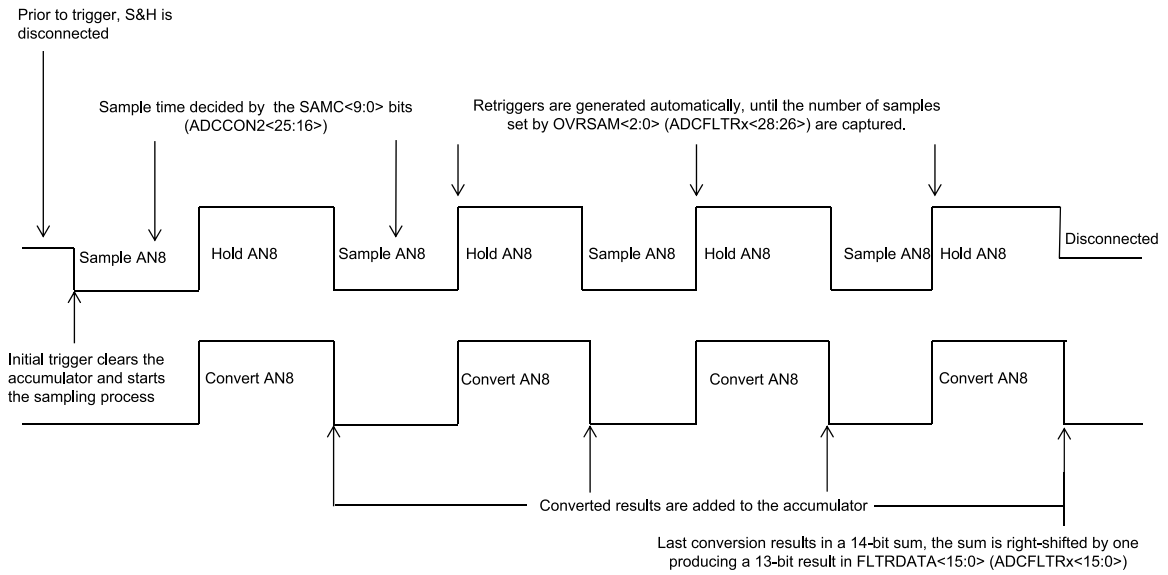
When the digital filter module is configured, the filter's control logic waits for an external trigger to initiate the process. The trigger signal for the analog input to be oversampled causes the accumulator to be cleared and initiates the first conversion. The trigger also forces the trigger sensitivity into level mode and forces the trigger itself to 1 as long as the filter needs to acquire

the user-specified number of samples via the OVRSAM[2:0] bits (ADCFLTRx[28:26]). The time delay between each acquired sample is decided by the set sample time in the SAMC[9:0] bits in the ADCCON2 register for Class 2 and the time for conversion. When the required number set by OVRSAM[2:0] are received and processed, the data stored in the FLTRDATA[15:0] bit (ADCFLTRx[15:0]) and the AFRDY bit (ADCFLTRx[24]) is set and the interrupt is generated (if enabled).

The following figure illustrates 4x oversampling using a Class 2 input. Triggering a Class 2 input initiates sampling for the length of time defined by the SAMC[9:0] bits. Retriggers generated by the oversampling logic use the SAMC[9:0] bits to set the sample time.

Class 2 inputs use the shared S&H; therefore, oversampling blocks lower priority Class 2 and Class 3 triggers. Higher priority Class 2 triggers completely disrupt the oversampling process; therefore, they must be avoided completely. The same priority rule applies to two Class 2 inputs that use two digital filters. In such a case, the higher priority input also uses the shared ADC module in Burst mode and prevents the lower priority input from using the shared ADC. Only after all required samples are obtained by the higher priority input can the lower priority input use the shared ADC to acquire samples for its own digital filtering.

Figure 38-9. 4x Oversampling of a Class 2 Input

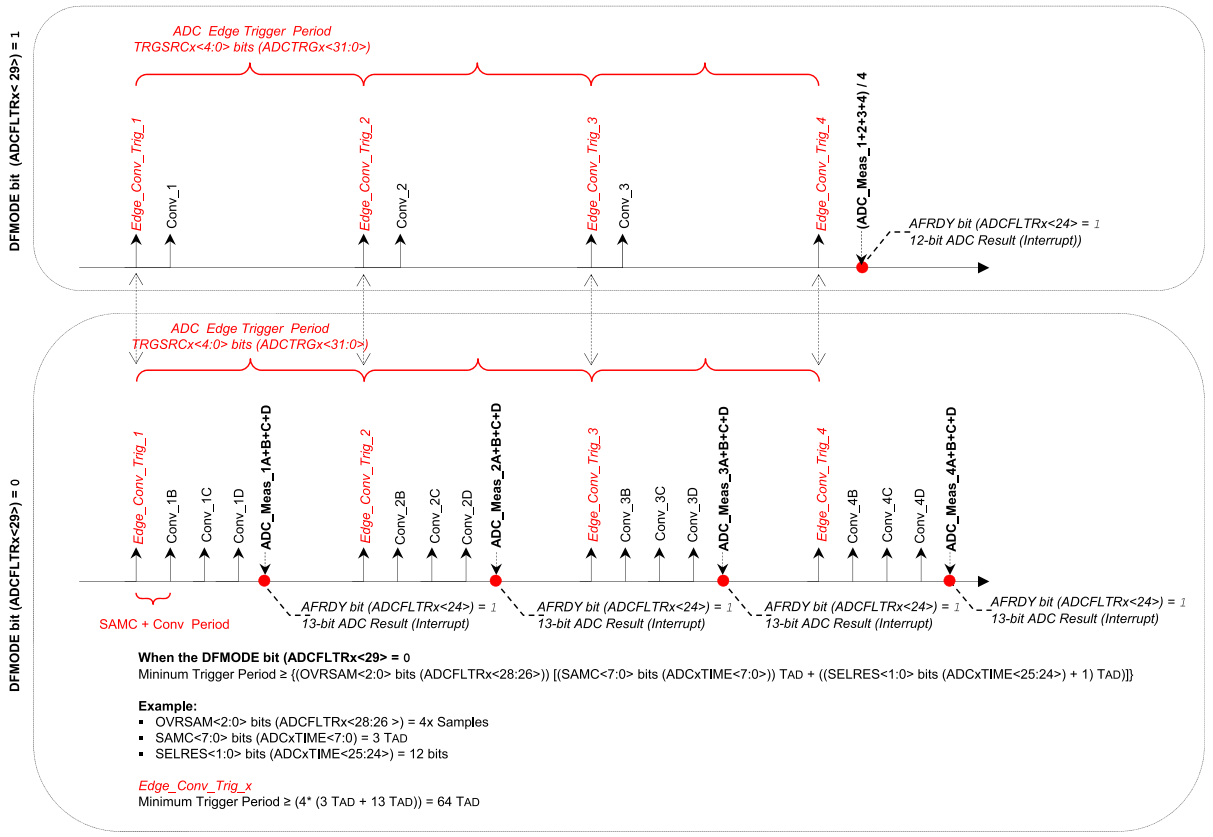


The following code is an example for ADC digital oversampling filter.

```
int main(int argc, char** argv) {
int result;

/* Configure ADCCON1 */
ADCCON1 = 0;          // No ADCCON1 features are enabled including: Stop-in-Idle, turbo,
// CVD mode, Fractional mode and scan trigger source.

/* Configure ADCCON2 */
ADCCON2 = 0;          // Since, we are using only the Class 1 inputs, no setting is
// required for ADCDIV

/* Initialize warm up time register */ ADCANCON = 0;
ADCANCONbits.WKUPCLKCNT = 5;  // Wake-up exponent = 32 * TADx

/* Clock setting */ ADCCON3 = 0;
ADCCON3bits.ADCSEL = 0;       // Select input clock source
ADCCON3bits.CONCLKDIV = 1;    // Control clock frequency is half of input clock
ADCCON3bits.VREFSEL = 0;      // Select AVDD and AVSS as reference source
```

```
ADC0TIMEbits.ADCDIV = 1;      // ADC0 clock frequency is half of control clock = TAD0
ADC0TIMEbits.SAMC = 5;        // ADC0 sampling time = 5 * TAD0
ADC0TIMEbits.SELRES = 3;      // ADC0 resolution is 12 bits

/* Select analog input for ADC modules, no presync trigger, not sync sampling */
ADCTRGMODEbits.SH0ALT = 0;    // ADC0 = AN0

/* Select ADC input mode */
ADCIMCON1bits.SIGN0 = 0;      // unsigned data format
ADCIMCON1bits.DIFF0 = 0;      // Single ended mode

/* Configure ADCGIRQENx */
ADCGIRQEN1 = 0;               // No interrupts are used
ADCGIRQEN2 = 0;

/* Configure ADCCSSx */
ADCCSS1 = 0;                  // No scanning is used
ADCCSS2 = 0;
/* Configure ADCCMPCONx */
ADCCMPCON1 = 0;               // No digital comparators are used. Setting the ADCCMPCONx
ADCCMPCON2 = 0;               // register to '0' ensures that the comparator is disabled.
ADCCMPCON3 = 0;               // Other registers are 'don't care'.
ADCCMPCON4 = 0; ADCCMPCON5 = 0; ADCCMPCON6 = 0;

/* Configure ADCFLTRx */
ADCFLTR1 = 0;                 // Clear all bits ADCFLTR1bits.CHNLID = 0;      // Use AN0 as
the source ADCFLTR1bits.OVRSAM = 3;      // 16x oversampling ADCFLTR1bits.DFMODE = 0;      //
Oversampling mode ADCFLTR1bits.AFEN = 1;        // Enable filter 1
ADCFLTR2 = 0;                 // Clear all bits
ADCFLTR3 = 0; ADCFLTR4 = 0; ADCFLTR5 = 0; ADCFLTR6 = 0;

/* Set up the trigger sources */ ADCTGSNSbits.LVL0 = 0;        // Edge trigger
ADCTRG1bits.TRGSRC0 = 1;     // Set AN0 to trigger from software.

/* Turn the ADC on */ ADCCON1bits.ON = 1;

/* Wait for voltage reference to be stable */
while(!ADCCON2bits.BGVRRDY); // Wait until the reference voltage is ready
while(ADCCON2bits.REFFLT);   // Wait if there is a fault with the reference voltage

/* Enable clock to analog circuit */
ADCANCONbits.ANEN0 = 1;       // Enable the clock to analog bias and digital control

/* Wait for ADC to be ready */
while(!ADCANCONbits.WKRDY0); // Wait until ADC0 is ready

/* Enable the ADC module */ ADCCON3bits.DIGEN0 = 1;       // Enable ADC0

while (1) {
/* Trigger a conversion */ ADCCON3bits.GSWTRG = 1;

/* Wait for the oversampling process to complete */
while (ADCFLTR1bits.AFRDY == 0);
/* fetch the result */
result = ADCFLTR1bits.FLTRDATA;

/*
* Process result Here
*
* Note 1: Loop time determines the sampling time for the first sample.
* remaining samples sample time is determined by set sampling + conversion time.
*
* Note 2: The first 5 samples may have reduced accuracy.
*
*/
}
return (1);
}
```

Figure 38-10. ADC Filter Comparisons Example



38.5 Interrupts

The ADC module supports interrupts triggered from a variety of sources that can be processed individually or globally. An early interrupt feature is also available to compensate for interrupt servicing latency.

After an enabled interrupt is generated, the CPU jumps to the vector assigned to that interrupt. The CPU begins executing code at the vector address. The user software at this vector address must perform the required operations, such as processing the data results, clearing the interrupt flag, then exiting. See *Nested Vector Interrupt Controller (NVIC)* from Related Links for more information on interrupts and the vector address table details.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)

38.5.1 Interrupt Sources

The ADC is capable of generating interrupts from the events listed in the following table.

Table 38-4. ADC Interrupt Sources

Interrupt Event	Description	Interrupt Enable Bit	Interrupt Status Bit
ANx Data Ready Event	Interrupt is generated upon a completion of a conversion from an analog input source (ANx). Each of the ARDYx bits is capable of generating a unique interrupt when set using the ADCBASE register.	AGIENx of ADCGIRQEN1	ARDYx of ADCDSTAT1 register

.....continued			
Interrupt Event	**Description**	**Interrupt Enable Bit**	**Interrupt Status Bit**
Digital Comparator Event	When an conversion's comparison criteria are met by a configured and enabled digital comparator. Each of the digital comparators is capable of generating a unique interrupt when its DCMPED bit is set.	DCMPGIEN of ADCCMPCONx register	DCMPED of ADCCMPCONx register
Oversampling Filter Data Ready Event	When an oversampling filter has completed the accumulation/decimation process and has stored the result.	AFGIEN of ADCFLTRx register	AFRDY of ADCFLTRx register
Both Band Gap Voltage and ADC Reference Voltage Ready Event	Interrupt is generated when both band gap voltage and ADC reference voltage are ready.	BGVRIEN of ADCCON2 register	BGVRRDY of ADCCON2 register
Band Gap Fault/ Reference Voltage Fault/ AV_{DD} Brown-out Fault Event	Interrupt is generated when Band Gap Fault/ Reference Voltage Fault/AV_{DD} Brown-out occurs.	REFFLTIEN of ADCCON2 register	REFFLT of ADCCON2 register
ADC Module Wake-up Event	Interrupt is generated when ADC wakes up after being enabled.	WKIEN0 of ADCANCON register	WKRDY0 of ADCANCON register
Update Ready Event	Interrupt is generated when ADC SFRs are ready to be (and can be safely) updated with new values.	UPDIEN of ADCCON3 register	UPDRDY of ADCCON3 register

38.5.2 ADC Base Register (ADCBASE) Usage

After conversion of ADC is complete, if the interrupt is vectored to a function that is common to all analog inputs, it takes some significant time to find the ADC input by evaluating the ARDYx bits in the ADCDSTATx. To avoid this time spent, the ADCBASE register is provided, which contains the base address of the user's ADC ISR jump table. When read, the ADCBASE register provides a sum of the contents of the ADCBASE register plus an encoding of the ARDYx bits set in the ADCDSTATx registers. This use of the ADCBASE register supports the creation of an interrupt vector address that can be used to improve the performance of an ISR.

The ARDYx bits are binary priority encoded with ARDY1 being the highest priority and ARDY8 being the lowest priority. The encoded priority result is, then, shifted left the amount specified by the number of bit positions specified by the IRQVS[2:0] bits in the ADCCON1 register, then added to the contents of the ADCBASE register. If there are no ARDYx bits set, then reading the ADCBASE register equals the value written into the ADCBASE register.

The ADCBASE register is typically loaded with the base address of a jump table that contains the address of the appropriate ISR. The k^{th} interrupt request is enabled via the AGIENx bit (1-8) in one of ADCGIRQENx SFRs ('x' = 1 or 2).

The following codes are examples for the ADCBASE register usage.

Case 1:

```
ADCBASE = 0x1234; // Set the address
ADCCON1bits.IRQVS = 2; // left shift by 2
ADCGIRQEN1bits.AGIEN0 = 1; // enable interrupt when AN0 completion is done.
```

When the ADC conversion for AN0 is complete, bit `0` of ADCDSTAT1 = ARDY0 is set.

Read value of ADCBASE = 0x1234 + (0 << 2) = 0x1234.

Therefore, the ISR must be placed at address `0x1234` for AN0.

Case 2:

```
ADCBASE = 0x1234; // Set the address
ADCCON1bits.IRQVS = 2; // left shift by 2
ADCGIRQEN1bits.AGIEN0 = 2; // enable interrupt when AN2 completion is done.
```

When the ADC conversion for AN2 is complete, bit 2 of ADCDSTAT1 = ARDY2 is set.

Read value of ADCBASE = 0x1234 + (2 << 2) = 0x123C.

Therefore, the ISR must be placed at address 0x123C for AN2.

Note: The contents of the ADCBASE register are not altered. Summation is performed when the ADCBASE register is read and the summation result is the returned read value from the ADCBASE SFR.

38.5.3 Interrupt Enabling, Priority and Vectoring

Each of the ADC events previously mentioned generates an interrupt when its associate Interrupt Enable bit, IE, is set. Each of the ADC events previously listed also has an associated interrupt vector. See *Nested Vector Interrupt Controller (NVIC)* from Related Links for more information on the vector location and control/status bits associated with each individual interrupt.

Related Links

10.2. Nested Vector Interrupt Controller (NVIC)

38.5.4 Individual and Global Interrupts

The use of the individual interrupts previously listed can significantly optimize the servicing of multiple ADC events by keeping each ISR focused on efficiently handling a specific event. In addition, different ISRs can be easily segregated according to the tasks performed, thereby making user software easier to implement and maintain. There may be cases where it is desirable to have a single ISR service multiple interrupt events. To facilitate this, each ADC event can be logically “ORed” to create a single global ADC interrupt. When an ADC event is enabled for a global interrupt, it vectors to a single interrupt routine. It is the responsibility of this single global ISR to determine the source of the interrupt through polling and process it accordingly.

Use of the Global Interrupt requires configuration of its own unique IE, IF, IP and IS bits as well as configuration of its interrupt vector as described in Interrupt Enabling, Priority and Vectoring. See *Interrupt Enabling, Priority and Vectoring* from Related Links.

Interrupts for the ADC can be configured as individual or global, or utilized as both where some are processed individually and others in the global ISR.

Related Links

38.5.3. Interrupt Enabling, Priority and Vectoring

38.6 Power-Saving Modes of Operation

The Power-Saving, Sleep and Idle modes are useful for reducing the conversion noise by minimizing the digital activity of the CPU, buses and other peripherals.

38.6.1 Sleep Mode

When the device enters Sleep mode, the system clock (SYCCLK) is halted. If an ADC module selects SYSCLK as its clock source or selects REFCLK3 as its clock source (REFCLK3 is generated from SYSCLK), the ADC enters the Sleep mode.

When the SYSCLK is the source (directly or indirectly) and Sleep mode occurs during a conversion, the conversion is aborted. The converter cannot resume a partially completed conversion on exiting from Sleep mode. The ADC register contents are not affected by the device entering or leaving Sleep mode. The ADC module can operate during Sleep mode if the ADC clock source is derived from a source other than SYSCLK that is active during Sleep mode. The FRC clock source is a logical choice

for operation during Sleep; however, the REFCLK3 clock source can also be used, provided it has an input clock that is operational during Sleep mode.

ADC operation during Sleep mode reduces the digital switching noise from the conversion. When the conversion is completed, the ARDYx status bit for that analog input is set and the result is loaded into the corresponding ADC Result register (ADCDATAx).

If any of the ADC interrupts are enabled, the device is woken up from Sleep mode when the ADC interrupt occurs. The program execution resumes at the ADC ISR if the ADC interrupt is greater than the current CPU priority. Otherwise, execution continues from the instruction after the WAIT instruction that placed the device in Sleep mode.

To minimize the effects of digital noise on the ADC module operation, the user must select a conversion trigger source that ensures that the analog-to-digital conversion take places in Sleep mode. For example, the external interrupt pin (INT0) conversion trigger option (TRGSRC[4:0] = `00100`) can be used for performing sampling and conversion while the device is in Sleep mode.

Note: For the ADC module to operate in Sleep mode, the ADC clock source must be set to Internal FRC (ADCSEL[1:0] bits (ADCCON2[31:30]) = `01`). Alternately, the REFCLK3 source can be used; however, the clock source used for REFCLK3 must operate during Sleep mode. Any changes to the ADC clock configuration require that the ADC be disabled.

38.6.2 Operation During Idle Mode

For the ADC, the stop in the Idle Mode bit, SIDL (ADCCON1[13]), specifies whether the ADC module stops on Idle or continues on Idle. If SIDL = 0, the ADC module continues normal operation when the device enters the Idle mode. If any of the ADC interrupts are enabled, the device wakes up from the Idle mode when the ADC interrupt occurs. The program execution resumes at the ADC ISR if the ADC interrupt is greater than the current CPU priority. Otherwise, execution continues from the instruction after the WAIT instruction that placed the device in the Idle mode.

If SIDL = 1, the ADC module stops in the Idle mode. If the device enters the Idle mode during a conversion, the conversion is aborted. The converter cannot resume a partially completed conversion on exiting from the Idle mode.

38.6.3 Low-Power Mode

The ADC module can be placed in a low-power state by disabling the digital circuit for individual ADC modules that are not running. This is possible by clearing the DIGENx bits and the DIGEN7 bit in the ADCCON3 register. (See *ADCCON3* register from Related Links.)

An even lower power state is possible by disabling the analog and bias circuit for individual ADC modules that are not running. This is possible by clearing the ANENx bits and the ANEN7 bit in the ADCANCON register. (See *ADCANCON* register from Related Links.) Disabling the digital circuit to achieve Low-Power mode provides a significantly faster module restart compared to disabling and re-enabling the analog and bias circuit of the ADC module. This is because disabling and re-enabling the analog and bias circuit using the ANENx bits and the ANEN7 bit requires a wake-up time (typical minimum wake-up time of 20 µs) for the ADC module before it can be used. Refer to the Electrical Specifications in the specific device data sheet for more information on the stabilization time.

When the analog and bias circuit for an ADC module is enabled, the wake-up must be polled (or through an interrupt) using the wake-up ready bits, WKRDY6:WKRDY0 and WKRDY7, which must be equal to '`1`'.

Related Links

38.11.4. ADCCON3
38.11.24. ADCANCON
43. Electrical Characteristics

38.7 Effects of Reset

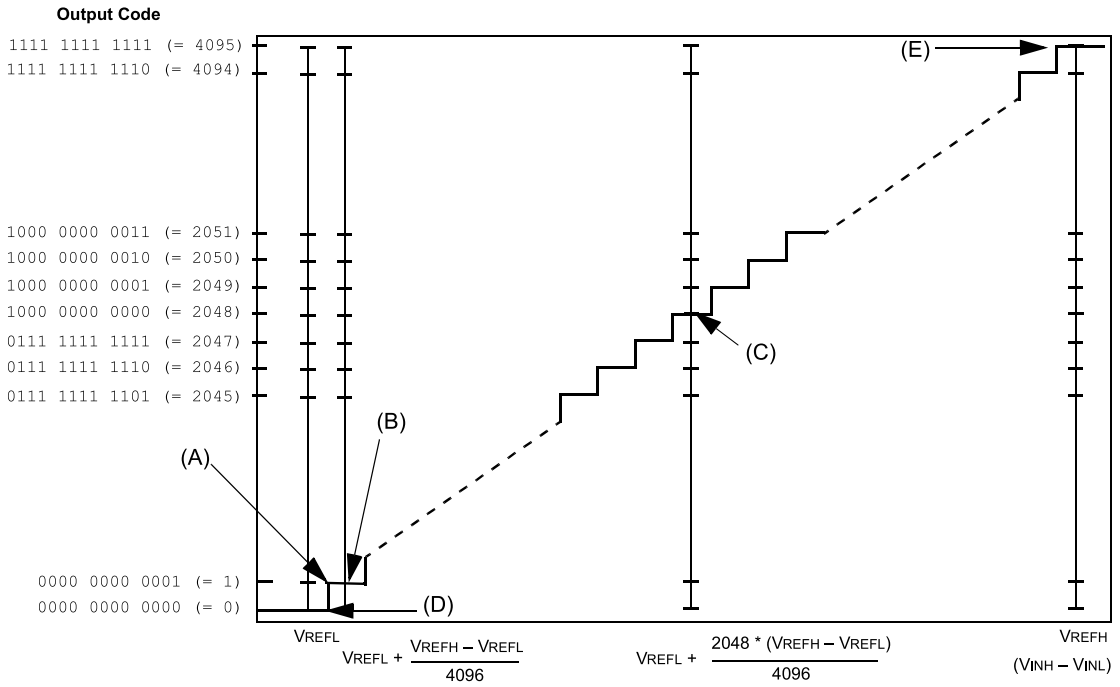
Following any Reset event, all the ADC control and status registers are reset to their default values with control bits in a non-active state. This disables the ADC module and sets the analog input pins to Analog Input mode. Any conversion that was in progress terminates, and the result cannot be written to the result buffer. The values in the ADCDATAx registers are initialized to '0x00000000' during a device Reset. The bias circuits are also turned OFF, so the ADC resuming operations wait for the bias circuits to stabilize by polling (or requesting to be interrupted by) the BGVRRDY bit (ADCCON2 register).

38.8 Transfer Function

A typical transfer function of the 12-bit ADC is illustrated in the following figure. The difference of the input voltages (V_{INH} - V_{INL}) is compared with the reference (V_{REFH} - V_{REFL}).

- The first code transition (A) occurs when the input voltage is (V_{REFH} - V_{REFL}/8192) or 0.5 LSb.
- The 0000 0000 0001 code is centered at (V_{REFH} - V_{REFL}/4096) or 1.0 LSb (B).
- The 1000 0000 0000 code is centered at (2048 * (V_{REFH} - V_{REFL})/4096) (C).
- An input voltage less than (1 * (V_{REFH} - V_{REFL})/8192) converts as 0000 0000 0000 (D).
- An input greater than (8192 * (V_{REFH} - V_{REFL})/8192) converts as 1111 1111 1111 (E).

Figure 38-11. Analog-to-Digital Transfer Function



38.9 ADC Sampling Requirements

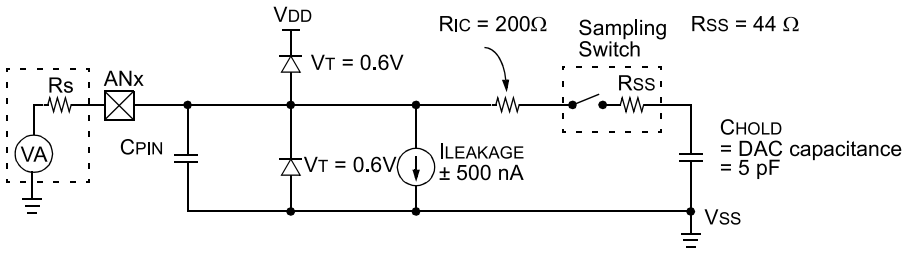
The analog input model of the 12-bit ADC is illustrated in the following figure. The total acquisition time for the analog-to-digital conversion is a function of the internal circuit settling time and the holding capacitor charge time.

For the ADC module to meet its specified accuracy, the charge holding capacitor (C_{HOLD}) must be allowed to fully charge to the voltage level on the analog input pin. The analog output source impedance (R_S), the interconnect impedance (R_{IC}) and the internal sampling switch (R_{SS}) impedance combine to directly affect the time required to charge the C_{HOLD}. The combined impedance of the analog sources must, therefore, be small enough to fully charge (to within one-fourth LSB of

the desired voltage) the holding capacitor within the selected sample time. The internal holding capacitor is in the discharged state prior to each sample operation.

At least 1 T_{AD} time period must be allowed between conversions for the acquisition time. Refer to the *Electrical Characteristics* from the Related Links.

Figure 38-12. 12-bit ADC Analog Input Model



Note: The C_{PIN} value depends on the device package and is not tested. The effect of the C_{PIN} is negligible if Rs 5 k.

Legend:

- C_{PIN} = Input capacitance
- R_{SS} = Sampling switch resistance
- R_S = Source resistance
- $I_{LEAKAGE}$ = Leakage current at the pin due to various junctions
- V_T = Threshold voltage
- R_{IC} = Interconnect resistance
- C_{HOLD} = Sample/hold capacitance

Related Links

43. Electrical Characteristics

38.10 Connection Considerations

Because the analog inputs employ Electrostatic Discharge (ESD) protection, they have diodes to V_{DD} and V_{SS}; therefore, the analog input must be between V_{DD} and V_{SS}. If the input voltage exceeds this range by greater than 0.3V (either direction), one of the diodes becomes forward biased, and it may damage the device if the input current specification is exceeded.

An external RC filter is sometimes added for antialiasing of the input signal. The R (resistive) component must be selected to ensure that the acquisition time is met. Any external components connected (through high-impedance) to an analog input pin (capacitor, Zener diode and so on) must have very little leakage current at the pin.

38.11 Register Description

Notes: The following conventions are used in the following registers:

- R = Readable bit
- W = Writable bit
- U = Unimplemented bit, read as '0'
- 1= Bit is set0= Bit is cleared
- x = Bit is unknown
- -n = Value at POR
- HS = Hardware Set
- HC = Hardware Cleared

Note: CLR/SET/INV registers for each register are located at offset <register offset> + 0x04, 0x08, 0x0C, respectively.

38.11.1 Register Summary

The PIC32CX-BZ2 12-bit High Speed SAR ADC module has the following Special Function Registers (SFRs):

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00 ... 0x13FF	Reserved									
0x1400	ADCCON1	7:0		IRQVS[2:0]			STRGLVL	DMABL[2:0]		
		15:8	ON	FRZ	SIDL			FSYDMA	FSYUPB	SCANEN
		23:16	FRACT	SELRES[1:0]		STRGSRC[4:0]				
		31:24								
0x1404 ... 0x140F	Reserved									
0x1410	ADCCON2	7:0		ADCDIV[6:0]						
		15:8	BGVRIEN	REFFLTIEN	EOSIEN		ENXCNVRT			
		23:16	SAMC[7:0]							
		31:24	BGVRRDY	REFFLT	EOSRDY				SAMC[9:8]	
0x1414 ... 0x141F	Reserved									
0x1420	ADCCON3	7:0	GLSWTRG	GSWTRG	ADINSEL[5:0]					
		15:8		VREFSEL[2:0]		TRGSUSP	UPDIEN	UPDRDY	SAMP	RQCNVRT
		23:16	CHN_EN_SHR							
		31:24	ADCSEL[1:0]		CONCLKDIV[5:0]					
0x1424 ... 0x143F	Reserved									
0x1440	ADCIMCON1	7:0	DIFF3	SIGN3	DIFF2	SIGN2	DIFF1	SIGN1	DIFF0	SIGN0
		15:8	DIFF7	SIGN7	DIFF6	SIGN6	DIFF5	SIGN5	DIFF4	SIGN4
		23:16	DIFF11	SIGN11	DIFF10	SIGN10	DIFF9	SIGN9	DIFF8	SIGN8
		31:24								
0x1444 ... 0x147F	Reserved									
0x1480	ADCGIRQEN1	7:0	AGIEN7	AGIEN6	AGIEN5	AGIEN4	AGIEN3	AGIEN2	AGIEN1	AGIEN0
		15:8					AGIEN11	AGIEN10	AGIEN9	AGIEN8
		23:16								
		31:24								
0x1484 ... 0x149F	Reserved									
0x14A0	ADCCSS1	7:0	CSS7	CSS6	CSS5	CSS4	CSS3	CSS2	CSS1	CSS0
		15:8					CSS11	CSS10	CSS9	CSS8
		23:16								
		31:24								
0x14A4 ... 0x14BF	Reserved									
0x14C0	ADCDSTAT1	7:0	ARDY7	ARDY6	ARDY5	ARDY4	ARDY3	ARDY2	ARDY1	ARDY0
		15:8					ARDY11	ARDY10	ARDY9	ARDY8
		23:16								
		31:24								
0x14C4 ... 0x14DF	Reserved									
0x14E0	ADCCMPEN1	7:0	CMPEx[7:0]							
		15:8								
		23:16								
		31:24								

.....continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x14E4 ... 0x14EF	Reserved									
0x14F0	ADCCMP1	7:0	DCMPLO[7:0]							
		15:8	DCMPLO[15:8]							
		23:16	DCMPHI[7:0]							
		31:24	DCMPHI[15:8]							
0x14F4 ... 0x14FF	Reserved									
0x1500	ADCCMPEN2	7:0	CMPEx[7:0]							
		15:8								
		23:16								
		31:24								
0x1504 ... 0x150F	Reserved									
0x1510	ADCCMP2	7:0	DCMPLO[7:0]							
		15:8	DCMPLO[15:8]							
		23:16	DCMPHI[7:0]							
		31:24	DCMPHI[15:8]							
0x1514 ... 0x159F	Reserved									
0x15A0	ADCFLTR1	7:0	FLTRDATA[7:0]							
		15:8	FLTRDATA[15:8]							
		23:16				CHNLID[4:0]				
		31:24	AFEN	DATA16EN	DFMODE	OVRSAM[2:0]			AFGIEN	AFRDY
0x15A4 ... 0x15AF	Reserved									
0x15B0	ADCFLTR2	7:0	FLTRDATA[7:0]							
		15:8	FLTRDATA[15:8]							
		23:16				CHNLID[4:0]				
		31:24	AFEN	DATA16EN	DFMODE	OVRSAM[2:0]			AFGIEN	AFRDY
0x15B4 ... 0x15FF	Reserved									
0x1600	ADCTRG1	7:0				TRGSRC0[4:0]				
		15:8				TRGSRC1[4:0]				
		23:16				TRGSRC2[4:0]				
		31:24				TRGSRC3[4:0]				
0x1604 ... 0x160F	Reserved									
0x1610	ADCTRG2	7:0				TRGSRC4[4:0]				
		15:8				TRGSRC5[4:0]				
		23:16				TRGSRC6[4:0]				
		31:24				TRGSRC7[4:0]				
0x1614 ... 0x167F	Reserved									
0x1680	ADCCMPCON1	7:0	ENDCMP	DCMPGIEN	DCMPED	IEBTWN	IEHIHI	IEHILO	IELOHI	IELOLO
		15:8			AINID[5:0]					
		23:16								
		31:24								
0x1684 ... 0x168F	Reserved									

.....continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x1690	ADCCMPCON2	7:0	ENDCMP	DCMPGIEN	DCMPED	IEBTWN	IEHIHI	IEHILO	IELOHI	IELOLO
		15:8				AINID[4:0]				
		23:16								
		31:24								
0x1694 ... 0x16FF	Reserved									
0x1700	ADCBASE	7:0	ADCBASE[7:0]							
		15:8	ADCBASE[15:8]							
		23:16								
		31:24								
0x1704 ... 0x170F	Reserved									
0x1710	ADCDMASTAT	7:0								RAF0
		15:8	DMACNTEN							RAF0IEN
		23:16	WROVRERR							RBF0
		31:24	DMAEN							RBF0IEN
0x1714 ... 0x171F	Reserved									
0x1720	ADCCNTB	7:0	ADCCNTB[7:0]							
		15:8	ADCCNTB[15:8]							
		23:16	ADCCNTB[23:16]							
		31:24	ADCCNTB[31:24]							
0x1724 ... 0x172F	Reserved									
0x1730	ADCDMAB	7:0	ADDMAB[7:0]							
		15:8	ADDMAB[15:8]							
		23:16	ADDMAB[23:16]							
		31:24	ADDMAB[31:24]							
0x1734 ... 0x173F	Reserved									
0x1740	ADCTRGSNS	7:0	LVL7	LVL6	LVL5	LVL4	LVL3	LVL2	LVL1	LVL0
		15:8								
		23:16								
		31:24								
0x1744 ... 0x17FF	Reserved									
0x1800	ADCANCON	7:0	ANEN7							ANEN0
		15:8	WKRDY7							WKRDY0
		23:16	WKIEN7							WKIEN0
		31:24					WKUPCLKCNT[3:0]			
0x1804 ... 0x1AFF	Reserved									
0x1B00	ADCSYSCFG0	7:0	AN[7:0]							
		15:8	AN[15:8]							
		23:16					AN[19:16]			
		31:24								
0x1B04 ... 0x1DFF	Reserved									
0x1E00	ADCDATAx	7:0	DATA[7:0]							
		15:8	DATA[15:8]							
		23:16	DATA[23:16]							
		31:24	DATA[31:24]							

38.11.2 ADCCON1 – ADC Control Register 1

Name: ADCCON1
Offset: 0x1400
Reset: 0x00601000
Property: -

This register controls the basic operation of the ADC module, including behavior in Sleep and Idle modes, and data formatting. This register also specifies the vector shift amounts for the Interrupt Controller. Additional ADCCON1 functions include the RAM buffer length in DMA mode.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	FRACT	SELRES[1:0]		STRGSRC[4:0]				
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	1	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	ON	FRZ	SIDL			FSYDMA	FSYUPB	SCANEN
Access	R/W	R/W	R/W			R/W	R/W	R/W
Reset	0	0	0			0	0	0

Bit	7	6	5	4	3	2	1	0
		IRQVS[2:0]			STRGLVL	DMABL[2:0]		
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit 23 – FRACT Fractional Data Output Format bit

Value	Description
0	Integer
1	Fractional

Bits 22:21 – SELRES[1:0] Shared ADC (ADC2) Resolution bits

Note: Changing the resolution of the ADC does not shift the result in the corresponding ADCDATAx register. The result occupies 12 bits, with the corresponding lower unused bits set to '0'. For example, a resolution of 6 bits results in ADCDATAx[5:0] being set to '0' and ADCDATAx[11:6] holding the result.

Value	Description
11	12 bits (default)
10	10 bits
01	8 bits
00	6 bits

Bits 20:16 – STRGSRC[4:0] ScanTrigger Source Select bits

Value	Description
10001 – 11111	Reserved
10000	EVSYS_47
01111	EVSYS_46
01110	EVSYS_45
01101	EVSYS_44
01100	EVSYS_43

Value	Description
01011	EVSYS_42
01010	EVSYS_41
01001	EVSYS_40
01000	EVSYS_39
00111	EVSYS_38
00110	EVSYS_37
00101	EVSYS_36
00100	INT0 External interrupt
00011	Reserved
00010	Global level software trigger (GLSWTRG)
00001	Global software edge trigger (GSWTRG)
00000	No Trigger

Bit 15 – ON ADC Module Enable bit

Note: The ON bit must be set only after the ADC module is configured.

Value	Description
0	ADC module is disabled
1	ADC module is enabled

Bit 14 – FRZ Freeze in Debug Mode

Value	Description
0	Do not freeze in Debug mode
1	Freeze in Debug mode

Bit 13 – SIDL Stop in Idle Mode bit

Value	Description
0	Continue module operation in Idle mode
1	Discontinue module operation when device enters Idle mode

Bit 10 – FSYDMA Fast Synchronous DMA System Clock bit

Value	Description
0	Fast synchronous DMA system clock is disabled
1	Fast synchronous DMA system clock is enabled

Bit 9 – FSYUPB Fast Synchronous UPB Clock bit

Value	Description
0	Fast synchronous UPB clock is disabled
1	Fast synchronous UPB clock is enabled

Bit 8 – SCANEN SCAN Enable bit

Bits 6:4 – IRQVS[2:0] Interrupt Vector Shift bits

To determine the interrupt vector address, this bit specifies the amount of left-shift done to the ARDYx status bits in the ADCDSTAT1 and ADCDSTAT2 registers prior to adding with the ADCBASE register.

Interrupt Vector Address = Read Value of ADCBASE, and Read Value of ADCBASE = Value written to ADCBASE + x << IRQVS[2:0], where 'x' is the smallest active input ID from the ADCDSTAT1 or ADCDSTAT2 registers (which has highest priority).

Value	Description
111	Shift x left 7 bit position
110	Shift x left 6 bit position
101	Shift x left 5 bit position
100	Shift x left 4 bit position
011	Shift x left 3 bit position
010	Shift x left 2 bit position

Value	Description
001	Shift x left 1 bit position
000	Shift x left 0 bit position

Bit 3 – STRGLVL ScanTrigger High Level/Positive Edge Sensitivity bit

Value	Description
0	Scan trigger is positive edge sensitive. Once STRIG mode is selected (TRGSRCx[4:0] in the ADCTRGx register), only a single scan trigger is generated, which completes the scan of all selected analog inputs.
1	Scan trigger is high level sensitive. Once STRIG mode is selected (TRGSRCx[4:0] in the ADCTRGx register), the scan trigger continues for all selected analog inputs, until the STRIG option is removed.

Bits 2:0 – DMABL[2:0] DMA to System RAM Buffer Length Size

Defines the number of locations in system memory allocated per analog input for DMA interface use. As each output data is 16-bit wide, one location consists of 2 bytes. Therefore, the actual size reserved in the system RAM follows the formula: RAM Buffer Length in bytes = $2_{(DMABL+1)}$.

38.11.3 ADCCON2 – ADC Control Register 2

Name: ADCCON2
Offset: 0x1410
Reset: 0x00000000
Property: -

This register controls the reference selection for the ADC module, the sample time for the shared ADC module, interrupt enable for reference, early interrupt selection and clock division selection for the shared ADC.

Bit	31	30	29	28	27	26	25	24
	BGVRRDY	REFFLT	EOSRDY				SAMC[9:8]	
Access	R/HS/HC	R/HS/HC	R/HS/HC				R/W	R/W
Reset	0	0	0				0	0

Bit	23	22	21	20	19	18	17	16
	SAMC[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	BGVRIEN	REFFLTIEN	EOSIEN		ENXCNVRT			
Access	R/W	R/W	R/W		R/W			
Reset	0	0	0		0			

Bit	7	6	5	4	3	2	1	0
		ADCDIV[6:0]						
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit 31 – BGVRRDY Band Gap Voltage/ADC Reference Voltage Status bit
Data processing is valid only after BGVRRDY is set by hardware, so the application code must check that the BGVRRDY bit is set to ensure data validity. This bit set to '0' when ON (ADCCON1[15]) = 0.

Value	Description
0	Either or both band gap voltage and ADC reference voltages (V_{REF}) are not ready
1	Both band gap voltage and ADC reference voltages (V_{REF}) are ready

Bit 30 – REFFLT Band Gap/V_{REF}/A_{VDD} BOR Fault Status bit
This bit is cleared when the ON bit (ADCCON1[15]) = 0 and the BGVRRDY bit = 1.

Value	Description
0	Band gap and V_{REF} voltage are working properly
1	Fault in band gap or the V_{REF} voltage while the ON bit (ADCCON1[15]) was set. Most likely a band gap or V_{REF} fault is caused by a BOR of the analog V_{DD}supply.

Bit 29 – EOSRDY End of Scan Interrupt Status bit
This bit is cleared when ADCCON2[31:24] are read in software.

Value	Description
0	Scanning has not completed
1	All analog inputs are considered for scanning through the scan trigger (all analog inputs specified in the ADCCSS1 and ADCCSS2 registers) have completed scanning

Bits 25:16 – SAMC[9:0] SampleTime for the Shared ADC (ADC2) bits
Where T_{AD7} = Period of the ADC conversion clock for the Shared ADC (ADC2) controlled by the ADCDIV[6:0] bits.

Value	Description
1111111111	1025 T_{AD7}
...	
0000000001	3 T_{AD7}
0000000000	2 T_{AD7}

Bit 15 – BGVRIEN Band Gap/V_{REF} Voltage Ready Interrupt Enable bit

Value	Description
0	No interrupt is generated when the BGVRRDY bit is set
1	Interrupt is generated when the BGVRDDY bit is set

Bit 14 – REFFLTIEN Band Gap/V_{REF} Voltage Fault Interrupt Enable bit

Value	Description
0	No interrupt is generated when the REFFLT bit is set
1	Interrupt is generated when the REFFLT bit is set

Bit 13 – EOSIEN End of Scan Interrupt Enable bit

Value	Description
0	No interrupt is generated when the EOSRDY bit is set
1	Interrupt is generated when the EOSRDY bit is set

Bit 11 – ENXCNVRT Enable External Conversion Request Interface

Setting this bit enables another module (such as the PTG) to specify and request conversion of an ADC input.

Note: The external module (such as the PTG) is responsible for asserting only the proper trigger signals. This ADC module has no method to block specific trigger requests from the external module.

Bits 6:0 – ADCDIV[6:0] Division Ratio for the Shared SAR ADC Core Clock bits

The ADCDIV[6:0] bits divide the ADC control clock (T_Q) to generate the clock for the shared SAR ADC.

Value	Description
1111111	254 * T_Q = T_{AD2}
...	
0000011	6 * T_Q = T_{AD2}
0000010	4 * T_Q = T_{AD2}
0000001	2 * T_Q = T_{AD2}
0000000	Reserved

38.11.4 ADCCON3 – ADC Control Register 3

Name: ADCCON3
Offset: 0x1420
Reset: 0x00000000
Property: -

This register enables ADC clock selection, enables/disables the digital feature for the shared ADC module and controls the manual (software) sampling and conversion.

Bit	31	30	29	28	27	26	25	24
	ADCSEL[1:0]		CONCLKDIV[5:0]					
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	CHN_EN_SHR							
Access	R/W							
Reset	0							

Bit	15	14	13	12	11	10	9	8
	VREFSEL[2:0]			TRGSUSP	UPDIEN	UPDRDY	SAMP	RQCNVRT
Access	R/W	R/W	R/W	R/W	R/W	R/HS/HC	R/W	R/HS/HC
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	GLSWTRG	GSWTRG	ADINSEL[5:0]					
Access	R/W	R/W, HC	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:30 – ADCSEL[1:0] Analog-to-Digital Clock Source (T_{CLK}) bits

Value	Description
00	Peripheral Bus Clock
01	FRC Clock
10	REFO3 Clock Output
11	System Clock (SYS_CLK)

Bits 29:24 – CONCLKDIV[5:0] Analog-to-Digital Control Clock (T_Q) Divider bits

Value	Description
111111	64 * T_{CLK}= T_Q
...	
000011	4 * T_{CLK}= T_Q
000010	3 * T_{CLK}= T_Q
000001	2 * T_{CLK}= T_Q
000000	T_{CLK}= T_Q

Bit 23 – CHN_EN_SHR Shared ADC Digital Enable bit

Value	Description
1	ADC is digital enabled
0	ADC is digital disabled

Bits 15:13 – VREFSEL[2:0] Voltage Reference (V_{REF}) Input Selection bits

Table 38-5.

VREFSEL[2:0]	AD_{REF+}	AD_{REF-}
000	AV_{DD}	AV_{SS}
001-111	RESERVED FOR FUTURE USE	

Bit 12 – TRGSUSP Trigger Suspend bit

Value	Description
1	Triggers are blocked from starting a new analog-to-digital conversion, but the ADC module is not disabled
0	Triggers are not blocked

Bit 11 – UPDIEN Update Ready Interrupt Enable bit

Value	Description
1	Interrupt is generated when the UPDRDY bit is set by hardware
0	No interrupt is generated

Bit 10 – UPDRDY ADC Update Ready Status bit

Note: This bit is only active while the TRGSUSP bit is set and there are no more running conversions of any ADC modules.

Value	Description
1	ADC SFRs can be updated
0	ADC SFRs cannot be updated

Bit 9 – SAMP Class 2 and Class 3 Analog Input Sampling Enable bit(1,2,3,4)

Value	Description
1	The ADC S&H amplifier is sampling
0	The ADC S&H amplifier is holding

Bit 8 – RQCNVRT Individual ADC Input Conversion Request bit

This bit and its associated ADINSEL[5:0] bits enable the user to individually request an analog-to-digital conversion of an analog input through software.

Note: This bit is automatically cleared in the next ADC clock cycle.

Value	Description
1	Trigger the conversion of the selected ADC input as specified by the ADINSEL[5:0] bits
0	Do not trigger the conversion

Bit 7 – GLSWTRG Global Level Software Trigger bit

Value	Description
1	Trigger conversion for ADC inputs that have selected the GLSWTRG bit as the trigger signal, either through the associated TRGSRC[4:0] bits in the ADCTRGx registers or through the STRGSRC[4:0]bits in the ADCCON1 register
0	Do not trigger an analog-to-digital conversion

Bit 6 – GSWTRG Global Software Trigger bit

This bit is automatically cleared in the next ADC clock cycle.

Value	Description
0	Trigger conversion for ADC inputs that have selected the GSWTRG bit as the trigger signal, either through the associated TRGSRC[4:0] bits in the ADCTRGx registers or through the STRGSRC[4:0]bits in the ADCCON1 register
1	Do not trigger an analog-to-digital conversion

Bits 5:0 – ADINSEL[5:0] Analog Input Select bits

These bits select the analog input to be converted when the RQCNVRT bit is set.

Note:

1. The SAMP bit has the highest priority and setting this bit keeps the S&H circuit in Sample mode until the bit is cleared. Also, usage of the SAMP bit causes settings of SAMC[9:0] bits (ADCCON2[25:16]) to be ignored.
2. The SAMP bit only connects Class 2 and Class 3 analog inputs to the shared ADC.
3. The SAMP bit is not a self-clearing bit and it is the responsibility of application software to first clear this bit and, only after setting the RQCNVRT bit, to start the analog-to-digital conversion.
4. Normally, when the SAMP and RQCNVRT bits are used by software routines, all TRGSRCx[4:0] bits and STRGSRC[4:0] bits must be set to '00000' to disable all external hardware triggers and prevent them from interfering with the software-controlled sampling command signal SAMP and with the software-controlled trigger RQCNVRT.

Value	Description
111111	Reserved
...	
001011	PMU Test Output
001010	VddCore (Internal)
001001	CP_Test_1.2V (Internal)
001000	BandGap Reference (Internal)
000111	AN7 is being monitored
...	
000001	AN1 is being monitored
000000	AN0 is being monitored

38.11.5 ADCIMCON1 – ADC Input Mode Control Register 1

Name: ADCIMCON1
Offset: 0x1440
Reset: 0x00000000
Property: -

This register enables the user to select between single-ended and differential operation as well as select between signed and unsigned data format.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	DIFF11	SIGN11	DIFF10	SIGN10	DIFF9	SIGN9	DIFF8	SIGN8
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	DIFF7	SIGN7	DIFF6	SIGN6	DIFF5	SIGN5	DIFF4	SIGN4
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DIFF3	SIGN3	DIFF2	SIGN2	DIFF1	SIGN1	DIFF0	SIGN0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 23 – DIFF11 AN11 Mode bit

Value	Description
1	AN11 is using Differential mode
0	AN11 is using Single-ended mode

Bit 22 – SIGN11 AN11 Signed Data Mode bit

Value	Description
1	AN11 is using Signed Data mode
0	AN11 is using Unsigned Data mode

Bit 21 – DIFF10 AN10 Mode bit

Value	Description
1	AN10 is using Differential mode
0	AN10 is using Single-ended mode

Bit 20 – SIGN10 AN10 Signed Data Mode bit

Value	Description
1	AN10 is using Signed Data mode
0	AN10 is using Unsigned Data mode

Bit 19 – DIFF9 AN9 Mode bit

Value	Description
1	AN9 is using Differential mode
0	AN9 is using Single-ended mode

Bit 18 – SIGN9 AN9 Signed Data Mode bit

Value	Description
1	AN9 is using Signed Data mode
0	AN9 is using Unsigned Data mode

Bit 17 – DIFF8 AN8 Mode bit

Value	Description
1	AN8 is using Differential mode
0	AN8 is using Single-ended mode

Bit 16 – SIGN8 AN8 Signed Data Mode bit

Value	Description
1	AN8 is using Signed Data mode
0	AN8 is using Unsigned Data mode

Bit 15 – DIFF7 AN7 Mode bit

Value	Description
1	AN7 is using Differential mode
0	AN7 is using Single-ended mode

Bit 14 – SIGN7 AN7 Signed Data Mode bit

Value	Description
1	AN7 is using Signed Data mode
0	AN7 is using Unsigned Data mode

Bit 13 – DIFF6 AN6 Mode bit

Value	Description
1	AN6 is using Differential mode
0	AN6 is using Single-ended mode

Bit 12 – SIGN6 AN6 Signed Data Mode bit

Value	Description
1	AN6 is using Signed Data mode
0	AN6 is using Unsigned Data mode

Bit 11 – DIFF5 AN5 Mode bit

Value	Description
1	AN5 is using Differential mode
0	AN5 is using Single-ended mode

Bit 10 – SIGN5 AN5 Signed Data Mode bit

Value	Description
1	AN5 is using Signed Data mode
0	AN5 is using Unsigned Data mode

Bit 9 – DIFF4 AN4 Mode bit

Value	Description
1	AN4 is using Differential mode
0	AN4 is using Single-ended mode

Bit 8 – SIGN4 AN4 Signed Data Mode bit

Value	Description
1	AN4 is using Signed Data mode
0	AN4 is using Unsigned Data mode

Bit 7 – DIFF3 AN3 Mode bit

Value	Description
1	AN3 is using Differential mode

Value	Description
0	AN3 is using Single-ended mode

Bit 6 – SIGN3 AN3 Signed Data Mode bit

Value	Description
1	AN3 is using Signed Data mode
0	AN3 is using Unsigned Data mode

Bit 5 – DIFF2 AN2 Mode bit

Value	Description
1	AN2 is using Differential mode
0	AN2 is using Single-ended mode

Bit 4 – SIGN2 AN2 Signed Data Mode bit

Value	Description
1	AN2 is using Signed Data mode
0	AN2 is using Unsigned Data mode

Bit 3 – DIFF1 AN1 Mode bit

Value	Description
1	AN1 is using Differential mode
0	AN1 is using Single-ended mode

Bit 2 – SIGN1 AN1 Signed Data Mode bit

Value	Description
1	AN1 is using Signed Data mode
0	AN1 is using Unsigned Data mode

Bit 1 – DIFF0 AN0 Mode bit

Value	Description
1	AN0 is using Differential mode
0	AN0 is using Single-ended mode

Bit 0 – SIGN0 AN0 Signed Data Mode bit

Value	Description
1	AN0 is using Signed Data mode
0	AN0 is using Unsigned Data mode

38.11.6 ADCGIRQEN1 – ADC Global Interrupt Enable Register 1

Name: ADCGIRQEN1
Offset: 0x1480
Reset: 0x00000000
Property: -

This register specifies which of the individual input conversion interrupts can generate the global ADC interrupt.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
					AGIEN11	AGIEN10	AGIEN9	AGIEN8
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
	AGIEN7	AGIEN6	AGIEN5	AGIEN4	AGIEN3	AGIEN2	AGIEN1	AGIEN0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 – AGIEN ADC Global Interrupt Enable bits

Value	Description
1	Interrupts are enabled for the selected analog input. The interrupt is generated after the converted data is ready (indicated by the ARDYx bit ('x' = 8-1) of the ADCDSTAT1 register)
0	Interrupts are disabled

38.11.7 ADCCSS1 – ADC Common Scan Select Register 1

Name: ADCCSS1
Offset: 0x14A0
Reset: 0x00000000
Property: -

This register specifies the analog inputs to be scanned by the common scan trigger.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
					CSS11	CSS10	CSS9	CSS8
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
	CSS7	CSS6	CSS5	CSS4	CSS3	CSS2	CSS1	CSS0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 – CSS Analog Common Scan Select bits

Notes:

1. In addition to setting the appropriate bits in this register, Class 2 analog inputs must select the STRIG input as the trigger source if they are to be scanned through the CSSx bits. Refer to the bit descriptions in the ADCTRGx registers for selecting the STRIG option.
2. If a Class 2 input is included in the scan by setting the CSSx bit to '1' and by setting the TRGSRCx[4:0] bits to STRIG mode (0b11), the user application must ensure that no other triggers are generated for that input using the RQCNVRT bit in the ADCCON3 register or the hardware input or any digital filter. Otherwise, the scan behavior is unpredictable.

Value	Description
1	Select ANx for input scan
0	Skip ANx for input scan

38.11.8 ADCDSTAT1 – ADC Data Ready Status Register 1

Name: ADCDSTAT1
Offset: 0x14C0
Reset: 0x00000000
Property: -

This register contains the interrupt status of the individual analog input conversions. Each bit represents the data-ready status for its associated conversion result.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
					ARDY11	ARDY10	ARDY9	ARDY8
Access					R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
	ARDY7	ARDY6	ARDY5	ARDY4	ARDY3	ARDY2	ARDY1	ARDY0
Access	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC
Reset	0	0	0	0	0	0	0	0

Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 – ARDY Conversion Data Ready for Corresponding Analog Input Ready bits

Value	Description
1	This bit is set when converted data is ready in the data register
0	This bit is cleared when the associated data register is read

38.11.9 ADCCMPEN1 – ADC Digital Comparator 1 Enable Register

Name: ADCCMPEN1
Offset: 0x14E0
Reset: 0x00000000
Property: -

These registers select which analog input conversion results is processed by the digital comparator.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	CMPEx[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – CMPEx[7:0] ADC Digital Comparator ‘x’ Enable bits
Note: CMPEx = where "x" stands for bit value from 0 to 7.

These bits enable conversion results corresponding to the analog input to be processed by the digital comparator. CMPE0 enables AN0, CMPE1 enables AN1 and so on.
Notes:

1. CMPEx = ANx, where ‘x’ = 0-31 (Digital Comparator inputs are limited to AN0 through AN31).
2. Changing the bits in this register while the Digital Comparator is enabled (ENDCMP = 1) can result in unpredictable behavior.

38.11.10 ADCCMPEN2 – ADC Digital Comparator 2 Enable Register

Name: ADCCMPEN2
Offset: 0x1500
Reset: 0x00000000
Property: -

These registers select which analog input conversion results is processed by the digital comparator.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
	CMPEx[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – CMPEx[7:0] ADC Digital Comparator 'x' Enable bits
Note: CMPEx = where 'x' stands for bit value from 0 to 7.

These bits enable conversion results corresponding to the analog input to be processed by the digital comparator. CMPE0 enables AN0, CMPE1 enables AN1 and so on.
Notes:

1. CMPEx = ANx, where 'x' = 0-31 (Digital Comparator inputs are limited to AN0 through AN31).
2. Changing the bits in this register while the Digital Comparator is enabled (ENDCMP = 1) can result in unpredictable behavior.

38.11.11 ADCCMP1 – ADC Digital Comparator 1 Limit Value Register

Name: ADCCMP1
Offset: 0x14F0
Reset: 0x00000000
Property: -

These registers contain the high and low digital comparison values for use by the digital comparator.
Notes:

1. Changing theses bits while the Digital Comparator is enabled (ENDCMP = 1) can result in unpredictable behavior.
2. The format of the limit values must match the format of the ADC converted value in terms of sign and fractional settings.
3. For Digital Comparator 0 used in CVD mode, the DCMPHI[15:0] and DCMPLO[15:0] bits must always be specified in signed format as the CVD output data is differential and is always signed.

Bit	31	30	29	28	27	26	25	24
	DCMPHI[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	DCMPHI[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	DCMPLO[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DCMPLO[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:16 – DCMPHI[15:0] Digital Comparator 'x' High Limit Value bits(1,2,3)
These bits store the high limit value, which is used by digital comparator for comparisons with ADC converted data.

Bits 15:0 – DCMPLO[15:0] Digital Comparator 'x' Low Limit Value bits(1,2,3)
These bits store the low limit value, which is used by digital comparator for comparisons with ADC converted data.

38.11.12 ADCCMP2 – ADC Digital Comparator 2 Limit Value Register

Name: ADCCMP2
Offset: 0x1510
Reset: 0x00000000
Property: -

These registers contain the high and low digital comparison values for use by the digital comparator.
Notes:

1. Changing theses bits while the Digital Comparator is enabled (ENDCMP = 1) can result in unpredictable behavior.
2. The format of the limit values must match the format of the ADC converted value in terms of sign and fractional settings.
3. For Digital Comparator 0 used in CVD mode, the DCMPHI[15:0] and DCMPLO[15:0] bits must always be specified in signed format as the CVD output data is differential and is always signed.

Bit	31	30	29	28	27	26	25	24
	DCMPHI[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	DCMPHI[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	DCMPLO[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DCMPLO[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:16 – DCMPHI[15:0] Digital Comparator 'x' High Limit Value bits(1,2,3)
These bits store the high limit value, which is used by digital comparator for comparisons with ADC converted data.

Bits 15:0 – DCMPLO[15:0] Digital Comparator 'x' Low Limit Value bits(1,2,3)
These bits store the low limit value, which is used by digital comparator for comparisons with ADC converted data.

38.11.13 ADCFLTR1 – ADC Digital Filter 1 Register

Name: ADCFLTR1
Offset: 0x15A0
Reset: 0x00000000
Property: -

These registers provide control and status bits for the oversampling filter accumulator, and also includes the 16-bit filter output data.

Bit	31	30	29	28	27	26	25	24
	AFEN	DATA16EN	DFMODE	OVRSAM[2:0]			AFGIEN	AFRDY
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
				CHNLID[4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	FLTRDATA[15:8]							
Access	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	FLTRDATA[7:0]							
Access	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC	R/HS/HC
Reset	0	0	0	0	0	0	0	0

Bit 31 – AFEN Digital Filter ‘x’ Enable bit

Value	Description
1	Digital filter is enabled
0	Digital filter is disabled and the AFRDY status bit is cleared

Bit 30 – DATA16EN Filter Significant Data Length bit

Note: This bit is significant only if DFMODE = 1 (Averaging Mode) and FRACT (ADCCON1[23]) = 1 (Fractional Output Mode).

Value	Description
1	All 16 bits of the filter output data are significant
0	Only the first 12 bits are significant, followed by four zeros

Bit 29 – DFMODE ADC Filter Mode bit

Value	Description
1	Filter ‘x’ works in Averaging mode
0	Filter ‘x’ works in Oversampling Filter mode (default)

Bits 28:26 – OVRSAM[2:0] Oversampling Filter Ratio bits

Value	Description
	If DFMODE is ‘0’
111	128 samples (shift sum 3 bits to right, output data is in 15.1 format)
110	32 samples (shift sum 2 bits to right, output data is in 14.1 format)
101	8 samples (shift sum 1 bit to right, output data is in 13.1 format)
100	2 samples (shift sum 0 bits to right, output data is in 12.1 format)
011	256 samples (shift sum 4 bits to right, output data is 16 bits)

Value	Description
010	64 samples (shift sum 3 bits to right, output data is 15 bits)
001	16 samples (shift sum 2 bits to right, output data is 14 bits)
000	4 samples (shift sum 1 bit to right, output data is 13 bits)
	If DFMODE is '1'
111	256 samples (256 samples to be averaged)
110	128 samples (128 samples to be averaged)
101	64 samples (64 samples to be averaged)
100	32 samples (32 samples to be averaged)
011	16 samples (16 samples to be averaged)
010	8 samples (8 samples to be averaged)
001	4 samples (4 samples to be averaged)
000	2 samples (2 samples to be averaged)

Bit 25 – AFGIEN Digital Filter 'x' Interrupt Enable bit

Value	Description
1	Digital filter interrupt is enabled and is generated by the AFRDY status bit
0	Digital filter is disabled

Bit 24 – AFRDY Digital Filter 'x' Data Ready Status bit

Note: This bit is cleared by reading the FLTRDATA[15:0] bits or by disabling the Digital Filter module (by setting AFEN to '0').

Value	Description
1	Data is ready in the FLTRDATA[15:0] bits
0	Data is not ready

Bits 20:16 – CHNLID[4:0] Digital Filter Analog Input Selection bits

Note: Only the first 12 analog inputs, Class 2 (AN0 -AN11), can use a digital filter.

These bits specify the analog input to be used as the oversampling filter data source.

Value	Description
11111	Reserved
...	
...	
...	
01100	Reserved
01011	AN11
...	
...	
...	
00010	AN2
00001	AN1
00000	AN0

Bits 15:0 – FLTRDATA[15:0] Digital Filter 'x' Data Output Value bits

The filter output data is as per the fractional format set in the FRACT bit (ADCCON1[23]). The FRACT bit must not be changed while the filter is enabled. Changing the state of the FRACT bit after the operation of the filter ended must not update the value of the FLTRDATA[15:0] bits to reflect the new format.