Mach-NX SFB Hardware Usage Guide

Technical Note

FPGA-TN-02222-1.0

October 2021

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults and associated risk the responsibility entirely of the Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
AES	Advanced Encryption Standard
AHB	Advanced High Performance
APB	Advanced Peripheral Bus
CPU	Central Processing Unit
CSR	Control and Status Registers
EAR	Extended Address Register
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
eSPI	Enhanced Serial Peripheral Interface
FIFO	First In, First Out
GPIO	General Purpose Input/Output
HMAC	Hash Message Authentication Code
I^2C	Inter-Integrated Circuit
OOB	Out-of-Band
PFR	Platform Firmware Resiliency
PKC	Public Key Cryptography
PLD	Programmable Logic Device
QSPI	Quad Serial Peripheral Interface
RISC	Reduced Instruction Set Computer
RoT	Root of Trust
SCI	System Configuration Interface
SFB	SoC Function Block
SHA	Secure Hash Algorithm
SoC	System on Chip
SPI	Serial Peripheral Interface
TRNG	True Random Number Generator

Register Access Definitions

Access Type	Behavior on Read Access	Behavior on Write Access
RO	Returns register value	Ignores write access
WO	Returns 0	Updates register value
RW	Returns register value	Updates register value
RW1C	Returns register value	Writing 1’b1 on register bit clears the bit to 1’b0. Writing 1’b0 on register bit is ignored.
RSVD	Returns 0	Ignores write access

1. SoC Function Block Overview

The Mach™-NX device family is the next generation of Lattice Semiconductor Low Density PLDs including enhanced security features and a Platform Firmware Resiliency SoC Function Block (SFB). The enhanced security features include Advanced Encryption Standard (AES) AES-128/256, Secure Hash Algorithm (SHA) SHA-256/384, Elliptic Curve Digital Signature Algorithm (ECDSA), Elliptic Curve Integrated Encryption Scheme (ECIES), Hash Message Authentication Code (HMAC) HMAC-SHA256/384, Public Key Cryptography, True Random Number Generator (TRNG) and Unique Secure ID.

The Mach-NX family is a Root-of-Trust hardware solution that can easily scale to protect the whole system with its enhanced bitstream security and user mode functions. With Lattice Mach-NX device, you can implement a Platform Firmware Resiliency (PFR) solution in your system, as described in NIST Special Publication 800-193. The purpose of this document is to describe the individual IP in the Mach-NX SoC Function Block.

1.1. Root of Trust

The Lattice Mach-NX FPGA can serve as the Root of Trust and provide the following services:

- Image Authentication – On system power-up or reset, the Mach-NX device holds the protected devices in reset while it authenticates their boot images in SPI flash. After each signature authentication passes, the Mach-NX device releases each resets, and those devices can boot from their authenticated SPI flash image. Image authentication can also be requested at any time through the I^2C Out of Band (OOB) communication path.
- Image Recovery – If a flash image becomes corrupted for any reason, it fails to be authenticated. The Mach-NX device can restore it to a known good state by copying from an authenticated backup image.
- SPI Flash Monitoring and Protection – The Mach-NX device can monitor multiple SPI/QSPI buses for unauthorized activity and block unauthorized accesses using external SPI quick switches. The monitors can be configured to watch for specific SPI flash commands and address ranges defined by the system designer and designate them as authorized (whitelisted) or unauthorized (blacklisted).
- Event Logging – The Mach-NX device logs security events, such as unauthorized flash accesses and notifies the BMC.
- I^2C Monitoring – The Mach-NX device can monitor an I^2C bus for unauthorized activity and block unauthorized transactions by disabling the I^2C bus. The monitor can be configured with multiple whitelist or blacklist filters to watch for specific byte or bit patterns defined by the system designer and designate them as authorized or unauthorized I^2C transactions.

1.2. SoC Function Block Diagram

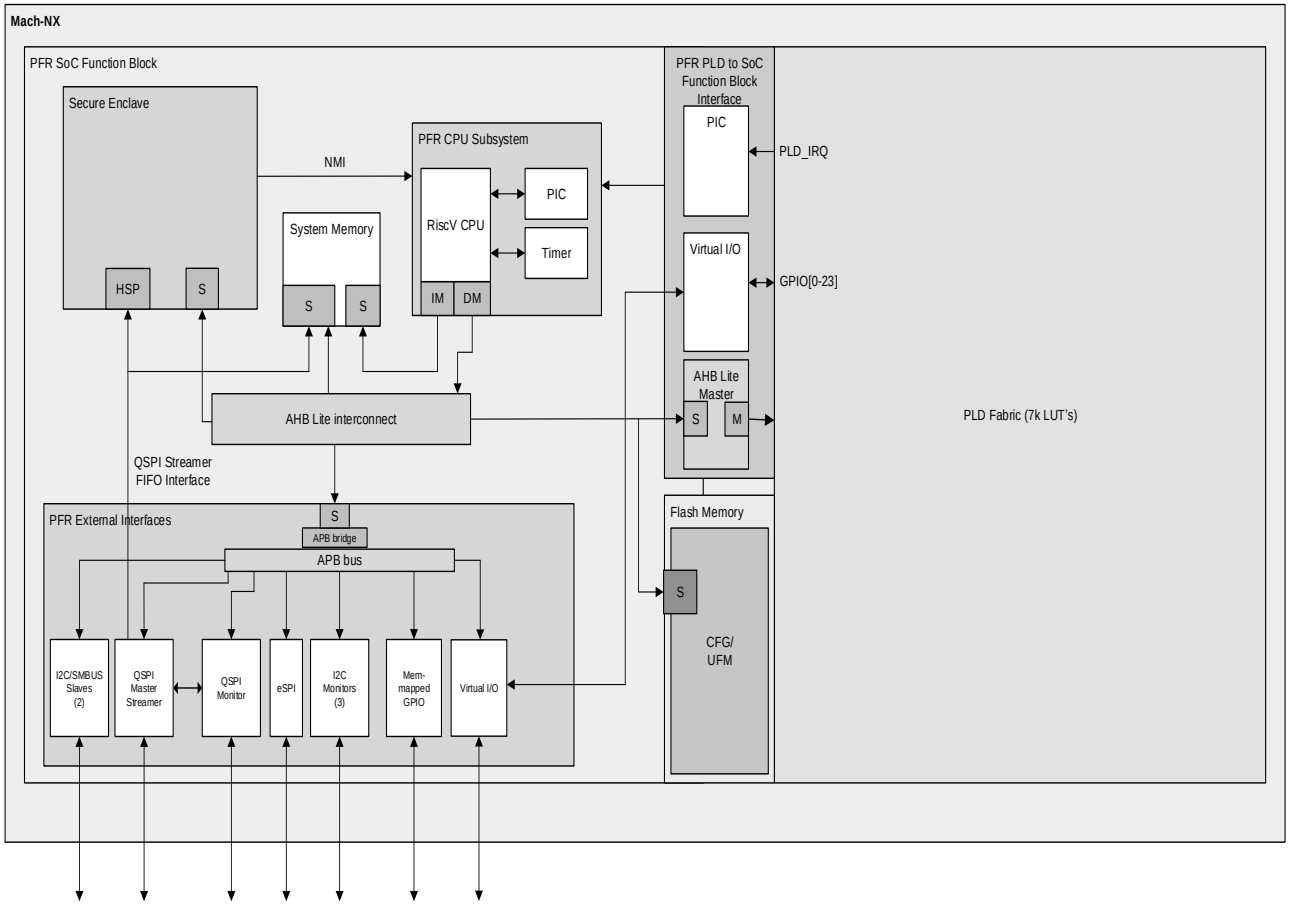


Figure 1.1. Mach-NX SoC Function Block Diagram

1.3. SoC Function Block Memory Map

Table 1.1. SoC Function Block Memory Map

Subsystem	Base Address	End Address	Block
PFR	00000000	0001FFFF	CPU Instruction RAM and Data RAM
	00020000	0007FFFF	RESERVED
	00080000	000807FF	CPU PIC TIMER
	00080800	000BFFFF	RESERVED
	000C0000	000C1FFF	Memory Mapped GPIO
	000C2000	000C3FFF	I^2C Monitor
	000C4000	000C7FFF	QSPI Monitor
	000C8000	000C9FFF	QSPI Master Streamer
	000CA000	000CBFFF	RESERVED
	000CC000	000CFFFF	eSPI
	000D0000	000DFFFF	I^2C/SMBus Slave
	000E0000	000FFFFF	RESERVED
FPGA	00100000	0013FFFF	RESERVED
	00140000	001400FF	PLD Logic
	00140100	001FFFFF	RESERVED
Security	00200000	003BFFFF	Security Enclave
	003C0000	FFFFFFFF	RESERVED

2. CPU Subsystem

2.1. Overview

The RISC-V MC processes data and instruction by considering the timer interrupt and external interrupt. As shown in Figure 2.1, the CPU core module has a 32-bit processor core and optional submodules. It uses two interfaces, one AHB-L interface (Read-Only) for instruction and one AHB-L interface (Read/Write Access) for data memory. RISC-V core, PIC, Timer, and AHB-L multiplex are run in the system clock domain. The RISC-V core debug runs in two clock domains: the system clock domain and the JTAG clock domain.

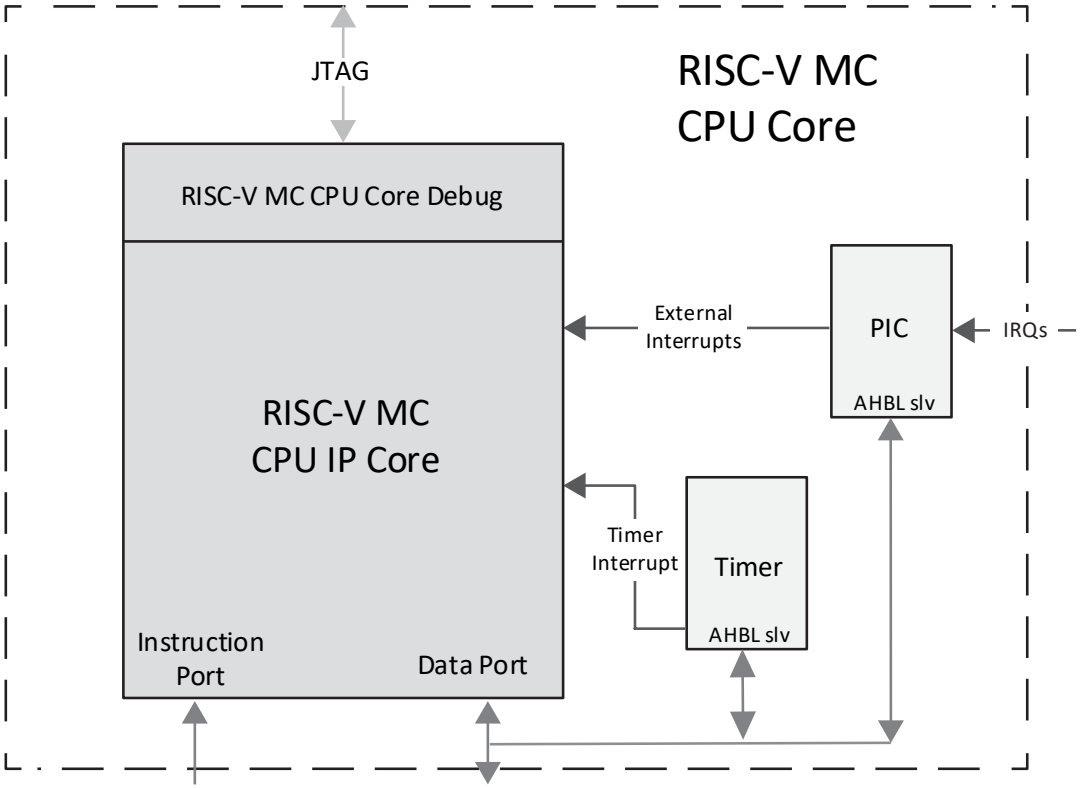


Figure 2.1. RISC-V MC CPU Diagram

2.2. Modules Description

2.2.1. RISC-V Processor Core

The processor core follows the RV32I instruction set.

2.2.1.1. Interrupt

Whenever an interrupt occurs, it has to remain in its active level until it is cleared by the processor core interrupt service routine.

If an interrupt occurs before jumping to the interrupt service routine, the processor core stops the prefetch stage and waits for all instructions in the later pipeline stages to complete their execution.

2.2.1.2. Exception

If an exception occurs, the processor core stops the corresponding instruction, flushes all previous instructions, and waits until the terminated instruction reaches the writeback stage before jumping to the exception service routine.

2.2.1.3. Debug

The processor core supports the IEEE-1149.1 JTAG debug logic with two hardware breakpoints.

2.2.1.4. Control and status registers

The processor core supports the Control and Status Registers (CSR) listed in Table 2.1.

Table 2.1. RISC-V Processor Core Control and Status Registers

addr	CSR Name	Access	Reset Value	Fields
0x300	mstatus (machine status register)	RW	0x0	bits[12:11] mpp: privilege mode before entering trap , should always be 2'b11 (m mode) bit [7] mpie: updated to mie value when entering to trap bit [3] mie: global interrupt enable
0x304	mie (machine interrupt enable register)	RW	0x0	bit[11] meie: m mode external interrupt enable bit[7] mtie: m mode timer interrupt enable bit[3] msie: m mode software interrupt enable
0x341	mepc (machine exception program counter)	RO	0x0	bits[31:0]: When trap in taken into m mode, mepc is used to store the address of the instruction that encountered exception.
0x342	mcause (machine cause register)	RO	0x0	bit[31]: 1'b1: interrupt 1'b0: exception bit[3:0]: exception code for interrupt: • 3-machine software interrupt • 7-machine timer interrupt • 11-machine external interrupt For exception: 0 – instruction address misaligned 1 – instruction access fault 2 – illegal instruction 4 – load address misaligned 5 – load access fault
0x343	mtval (machine trap value register)	RO	0x0	bits[31:0]: When a hardware breakpoint is triggered, or an instruction-fetch, load, or store address is misaligned or access exception occurs, mtval is written with the faulting address. It may also be written with illegal instruction when an illegal instruction exception occurs.
0x344	mip (machine interrupt pending register)	RO	0x0	bit[11] meip: m mode external interrupt pending bit[7] mtip: m mode timer interrupt pending bit[3] msip: m mode software interrupt pending

2.2.2. Submodule (PIC/Timer)

The CPU contains submodules: PIC and Timer. The PIC and Timer share the same start address in memory map and a fixed 2 kB address range is allocated if any of them are enabled.

2.2.2.1. PIC

The PIC aggregates up to eight external interrupt inputs (IRQs) into one interrupt output to processor core. The interrupt status register and can be used to read the values of IRQs. Individual IRQs can be configured by programming the corresponding enable and polarity registers. Table 2.2 provides the descriptions of PIC registers.

Table 2.2. PIC Registers

<table>
<tr><th>Offset</th><th>Name</th><th>Access</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>0x000</td><td>PIC_ISRC</td><td>WO</td><td>0x0</td><td>Interrupt Status Register Clear
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[7]</td><td>PIC_ISRC [7]</td><td>W</td><td>1</td><td>0x0</td></tr>
<tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
<tr><td>[1]</td><td>PIC_ISRC [1]</td><td>W</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_ISRC [0]</td><td>W</td><td>1</td><td>0x0</td></tr>
</table></td></tr>
<tr><td>0x004</td><td>PIC_ISRS</td><td>WO</td><td>0x0</td><td>Interrupt Status Register Set
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[7]</td><td>PIC_ISRS [7]</td><td>W</td><td>1</td><td>0x0</td></tr>
<tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
<tr><td>[1]</td><td>PIC_ISRS [1]</td><td>W</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_ISRS [0]</td><td>W</td><td>1</td><td>0x0</td></tr>
</table></td></tr>
<tr><td>0x008</td><td>PIC_ISR</td><td>RO</td><td>0x0</td><td>Interrupt Status Register
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[7]</td><td>PIC_ISR [7]</td><td>R</td><td>1</td><td>0x0</td></tr>
<tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
<tr><td>[1]</td><td>PIC_ISR [1]</td><td>R</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_ISR [0]</td><td>R</td><td>1</td><td>0x0</td></tr>
</table></td></tr>
<tr><td>0x010</td><td>PIC_IERC</td><td>WO</td><td>0x0</td><td>Interrupt Enable Register Clear
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[7]</td><td>PIC_IERC[7]</td><td>W</td><td>1</td><td>0x0</td></tr>
<tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
<tr><td>[1]</td><td>PIC_IERC[1]</td><td>W</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_IERC[0]</td><td>W</td><td>1</td><td>0x0</td></tr>
</table>
PIC_IERC[i]
Enable or Disable the interrupt request (irq[i]) port from the aggregation of interrupts (core_meip).
• 0 – disable irq[i]
• 1 – enable irq[i]</td></tr>
</table>

<table>
<tr><th>Offset</th><th>Name</th><th>Access</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>0x014</td><td>PIC_IERS</td><td>WO</td><td>0x0</td><td>Interrupt Enable Register Set

<table><tr><td>Field</td><td>Name</td><td>Access</td><td>Width</td><td>Reset</td></tr><tr><td>[7]</td><td>PIC_IERS[7]</td><td>W</td><td>1</td><td>0x0</td></tr><tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr><tr><td>[1]</td><td>PIC_IERS[1]</td><td>W</td><td>1</td><td>0x0</td></tr><tr><td>[0]</td><td>PIC_IERS[0]</td><td>W</td><td>1</td><td>0x0</td></tr></table></td></tr>
<tr><td>0x018</td><td>PIC_IER</td><td>RO</td><td>0x0</td><td>Interrupt Enable Register

<table><tr><td>Field</td><td>Name</td><td>Access</td><td>Width</td><td>Reset</td></tr><tr><td>[7]</td><td>PIC_IERS[7]</td><td>R</td><td>1</td><td>0x0</td></tr><tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr><tr><td>[1]</td><td>PIC_IERS[1]</td><td>R</td><td>1</td><td>0x0</td></tr><tr><td>[0]</td><td>PIC_IERS[0]</td><td>R</td><td>1</td><td>0x0</td></tr></table></td></tr>
<tr><td>0x020</td><td>PIC_POLC</td><td>WO</td><td>0x0</td><td>PIC Interrupt Polarity Register Clear

<table><tr><td>Field</td><td>Name</td><td>Access</td><td>Width</td><td>Reset</td></tr><tr><td>[7]</td><td>PIC_POLC [7]</td><td>W</td><td>1</td><td>0x0</td></tr><tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr><tr><td>[1]</td><td>PIC_POLC l[1]</td><td>W</td><td>1</td><td>0x0</td></tr><tr><td>[0]</td><td>PIC_POLC [0]</td><td>W</td><td>1</td><td>0x0</td></tr></table></td></tr>
<tr><td>0x024</td><td>PIC_POLS</td><td>WO</td><td>0x0</td><td>PIC Interrupt Polarity Register Set

<table><tr><td>Field</td><td>Name</td><td>Access</td><td>Width</td><td>Reset</td></tr><tr><td>[7]</td><td>PIC_POLC [7]</td><td>W</td><td>1</td><td>0x0</td></tr><tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr><tr><td>[1]</td><td>PIC_POLC l[1]</td><td>W</td><td>1</td><td>0x0</td></tr><tr><td>[0]</td><td>PIC_POLC [0]</td><td>W</td><td>1</td><td>0x0</td></tr></table></td></tr>
<tr><td>0x028</td><td>PIC_POL</td><td>RO</td><td>0x0</td><td>PIC Interrupt Polarity register

<table><tr><td>Field</td><td>Name</td><td>Access</td><td>Width</td><td>Reset</td></tr><tr><td>[7]</td><td>PIC_POLC [7]</td><td>R</td><td>1</td><td>0x0</td></tr><tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr><tr><td>[1]</td><td>PIC_POLC l[1]</td><td>R</td><td>1</td><td>0x0</td></tr><tr><td>[0]</td><td>PIC_POLC [0]</td><td>R</td><td>1</td><td>0x0</td></tr></table>

PIC_POLC[i]
Indicates the polarity of interrupt request (irq[i]) port.
• 0 – irq[i] is active high
• 1 – irq[i] is active low</td></tr>
</table>

2.2.2.2. Timer

The Timer module provides a 64-bit real-time counter register (mtime) and time compare register (mtimecmp). An output interrupt signal notices the RISC-V processor core when the value of mtime is greater than or equal to the value of mtimecmp. Table 2.3 provides the descriptions of Timer registers.

Table 2.3. Timer Registers

Offset	Name	Access	Reset Value	Description
mtime A 64-bit real-time counter register. You must set the register to a non-zero value to start the counting process.				
0x400	TIMER_CNT_L	RW	0x0	Lower 32 bits of *mtime* register
0x404	TIMER_CNT_H	RW	0x0	Higher 32 bits of *mtime* register
mtimecmp This register is used to generate or clear the timer interrupt (mtip). When the value of mtime register is greater than or equal to the value of mtimecmp register, the cpu_mtip_o is asserted and remains asserted until it is cleared by writing to mtimecmp register. Lower 32 bit for Timer time compare register				
0x410	TIMER_CMP_L	RW	0x0	Lower 32-bit for *mtimecmp* register.
0x414	TIMER_CMP_H	RW	0x0	Higher 32-bit for *mtimecmp* register

3. System Memory

3.1. Overview

The System Memory is used for code execution and temporary data storage.

3.2. Features

The key features of the System Memory are:

- 128 KB SRAM
- Dual AHB Lite Slave interface
- FIFO interface connected to QSPI Master Streamer

3.3. Block Diagram

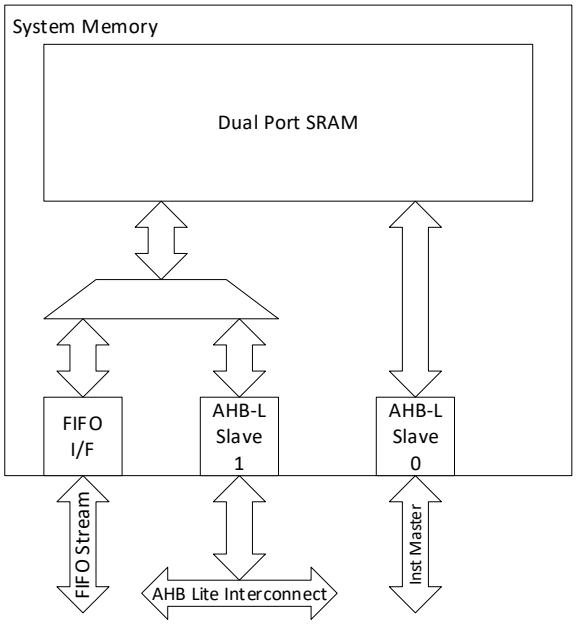


Figure 3.1. System Memory Block Diagram

3.3.1. AHB-Lite Interface

The System Memory has two AHB Lite slave interfaces. Slave 0 interface is read only and connected directly to the Instruction Master of the CPU. Slave 1 interface is connected to the SoC Function Block AHB Lite Interconnect.

3.3.2. FIFO Interface

The dedicated FIFO interface is shared with the AHB-L port S1. This interface is used by the QSPI Master Streamer to upload firmware values to the core memory.

3.3.3. System Memory Timing Information

When a port reads and the other port writes on the same address, the read transaction completes first and the old data is propagated into the output before the new data is written on the selected address. After which, the new data is made available on both ports and can be read on the next read transaction.

4. QSPI Monitor

4.1. Overview

The QSPI Monitor is an SPI access and command monitoring module that can monitor up to three SPI, DSPI, or QSPI buses for unauthorized activity and prevent transactions from completing by controlling internal or external switches. In addition to monitoring, the QSPI Monitor connects the external SPI/DSPI/QSPI buses to internal QSPI Master Streamer through a programmable mux/demux block.

4.2. Features

The key features of the QSPI monitor are:

- Supports three external SPI, DSPI, or QSPI buses to monitor illegal activity
- Enable/disable dynamically the flash initialization commands per monitor
- Flash commands (program, read, erase) are monitored based on address ranges
- Supports up to eight dynamically configurable address ranges for filtering per monitor
- Supports both 24-bit and 32-bit flash addressing modes/commands
- Supports single and dual flash configurations
- Supports internal and external switching

4.3. Block Diagram

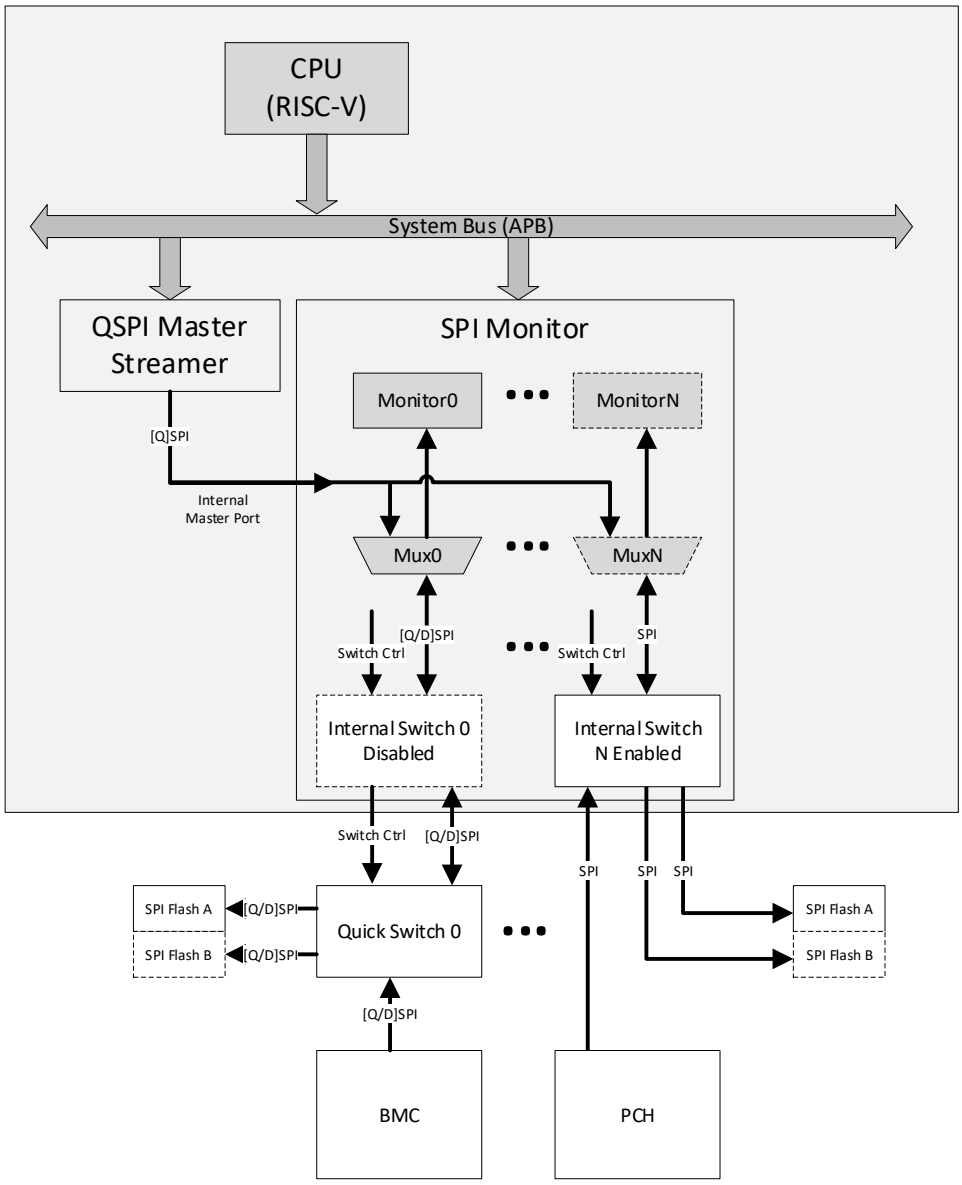


Figure 4.1. QSPI Monitor Block Diagram

4.4. Signal Description

Table 4.1. QSPI Monitor Signal Description

Signal	Direction	Description
QSPI Monitor External Signal		
QSPI_MONx_CLK	Bidir	SPI/QSPI clock (High Impedance during monitoring)
QSPI_MONx_CSN	Output	Chip select (High Impedance during monitoring)
QSPI_MONx_DIS_A	Output	Quick Switch Disable Flash A (0=enabled, 1=disabled)
QSPI_MONx_DIS_B	Output	Quick Switch Disable Flash B (0=enabled, 1=disabled)
QSPI_MONx_DQ0	Bidir	SPI: MOSI QSPI: serial data input and output (High Impedance during monitoring)
QSPI_MONx_DQ1	Bidir	SPI: MISO QSPI: serial data input and output (High Impedance during monitoring)
QSPI_MONx_DQ2	Bidir	SPI: unused QSPI: serial data input and output (High Impedance during monitoring)
QSPI_MONx_DQ3	Bidir	SPI: unused QSPI: serial data input and output (High Impedance during monitoring)
QSPI_MONx_PRE_CSN	Input	QSPI/SPI Chip select before quick switch
QSPI_MONx_RST_O	Output	Reset
QSPI_MONx_SWI_EN	Output	Quick Switch Output Enable (0=disabled, 1=enabled). This signal is enabled when the QSPI Monitor is protecting the SPI Flash and when the QSPI Monitor is switched to the internal master.
QSPI_MONx_SWI_ISO	Output	Quick Switch Isolation (0=disabled, 1=enabled), this optional signal is used when a flash has switching logic to select between multiple SPI Masters (BMC and PCH). This signal is enabled when the QSPI Monitor is switched to the internal master.

4.5. QSPI Command List

The allowed QSPI commands are shown in Table 4.2. All other commands are blocked.

Table 4.2. QSPI Command List Table

Command	Default	Description
Initialization Command 0	01 (WRSR)	Initialization Command 0
Initialization Command 1	04 (WRDI)	Initialization Command 1
Initialization Command 2	05 (RDSR)	Initialization Command 2
Initialization Command 3	06 (WREN)	Initialization Command 3
Initialization Command 4	50 (WRSR_EN)	Initialization Command 4
Initialization Command 5	9F (RDID)	Initialization Command 5
Page Program Command	02	Page Program Command Command/Address/Data widths are all 1-bit in SPI mode, 4-bit in QSPI mode.
Page Program Quad Address Quad Data Command	38	Page Program Quad Address Quad Data Command Command width is 1-bit. Address and Data widths are 4-bit.
Erase 4KB Command	20	Erase 4 KB Command
Erase 32KB Command	52	Erase 32 KB Command
Erase 64KB Command	D8	Erase 64 KB Command
Read Command	03	Read Command.

Command	Default	Description
Fast Read Command	0B	Fast Read Command Command/Address/Data widths are all 1-bit in SPI mode, 4-bit in QSPI mode.
Read Dual Data	3B	Read Dual Data Command Command/Address widths are 1-bit. Data width is 2-bit.
Read Dual Address Dual Data Command	BB	Read Dual Address Dual Data Command Command width is 1-bit. Address and Data widths are 2-bit.
Read Quad Data Command	6B	Read Quad Data Command Command/Address widths are 1-bit in SPI mode, 4-bit in QSPI mode. Data width is 4-bit.
Read Quad Address Quad Data Command	EB	Read Quad Address Quad Data Command Command width is 1-bit in SPI mode, 4-bit in QSPI mode. Address and Data widths are 4-bit.
Quad SPI Mode Enter Command	35	Quad SPI Mode Enter Command
Quad SPI Mode Exit Command	F5	Quad SPI Mode Exit Command
4-byte Mode Enter Command	B7	4-Byte Mode Enter Command
4-byte Mode Exit Command	E9	4-Byte Mode Exit Command
4-byte Read Extended Address Command	C8	4-Byte Read Extended Address Register Command
4-byte Write Extended Address Command	C5	4-Byte Write Extended Address Register Command
4-byte Page Program Command	12	4-Byte Page Program Command
4-byte Page Program Quad Address Quad Data Command	3E	4-Byte Page Program Quad Address Quad Data Command.
4-byte Erase 4KB Command	21	4-Byte Erase 4 KB Command
4-byte Erase 32KB Command	5C	4-Byte Erase 32 KB Command
4-byte Erase 64KB Command	DC	4-Byte Erase 64 KB Command
4-byte Read Command	13	4-Byte Read Command
4-byte Fast Read Command	0C	4-Byte Fast Read Command Command/Address/Data widths are all 1-bit in SPI mode, 4-bit in QSPI mode.
4-byte Read Dual Data Command	3C	4-Byte Read Dual Data Command Command/Address widths are 1-bit. Data width is 2-bit.
4-byte Read Dual Address Quad Data Command	BC	4-Byte Read Dual Address Dual Data Command Command width is 1-bit. Address and Data widths are 2-bit.
4-byte Read Quad Data Command	6C	4-Byte Read Quad Data Command Command/Address widths are 1-bit in SPI mode, 4-bit in QSPI mode. Data width is 4-bit.
4-byte Read Quad Address Quad Data Command	EC	4-Byte Read Quad Address Quad Data Command Command width is 1-bit in SPI mode, 4-bit in QSPI mode. Address and Data widths are 4-bit.

4.6. Register Description

A summary of the QSPI Monitor Core Registers is shown in Table 4.4. Global registers are mapped to offsets 0x000–0x0FC and per-monitor registers are mapped to 0xN00–0xNFF, where *N* corresponds to the monitor number, in the range of 1 to 3. For example, registers of the first monitor are at offsets 0x100–0x1FC and registers of the second monitor are at 0x200–0x2FC. The registers for Address Space 7 are mapped from 0xM00–0xMFF, where *M* is equal to (5 + N), see Table 4.3.

Table 4.3. QSPI Monitor Address Space Mapping for each Monitor

Monitor	Register Offsets for Address Spaces 0 to 6	Register Offsets for Address Space 7	N	M
Monitor0	0x100-0x1FF	0x600-0x6FF	1	6
Monitor1	0x200-0x2FF	0x700-0x7FF	2	7
Monitor2	0x300-0x3FF	0x800-0x8FF	3	8

Table 4.4. QSPI Monitor Core Registers

Offset	Register Name	Access	Reset Value	Description
0x000	MONITOR_CFG	RO	0x03	num_bus_monitors[1:0] – Number of bus monitors reserved[31:42]
0x004	MONITOR_CTRL	RW	0x00	monitor0_en[0] – Enable/disable Monitor0 monitor1_en[1] – Enable/disable Monitor1 monitor2_en[2] – Enable/disable Monitor2 reserved[31:3]
0x008	MONITOR_SPI_MODE	RW	0x00	monitor0_spi_mode[1:0] – Monitor0 SPI Mode (0 or 3) reserved[3:2] monitor1_spi_mode[5:4] – Monitor1 SPI Mode (0 or 3) reserved[7:6] monitor2_spi_mode[9:8] – Monitor2 SPI Mode (0 or 3) reserved[31:10]
0x010	INT_STATUS	RW	0x00	Interrupt Status Interrupt status: illegal_op0_int[0] – Bus 0 Illegal Operation interrupt illegal_op0_overflow_int[1] – Bus 0 Illegal Operation Overflow interrupt reserved[3:2] illegal_op1_int[4] – Bus 1 Illegal Operation interrupt illegal_op1_overflow_int[5] – Bus 1 Illegal Operation Overflow interrupt reserved[7:6] illegal_op2_int[8] – Bus 2 Illegal Operation interrupt illegal_op2_overflow_int[9] – Bus 2 Illegal Operation Overflow interrupt reserved[31:10] Writing 1 to a bit clears that interrupt.

Offset	Register Name	Access	Reset Value	Description
0x014	INT_ENABLE	RW	0x00	Interrupt Enable: illegal_op0_en[0] – Enable Bus 0 Illegal Operation interrupt illegal_ op0_overflow_en[1] – Enable Bus 0 Illegal Operation Overflow interrupt reserved[3:2] illegal_op1_en[4] – Enable Bus 1 Illegal Operation interrupt illegal_op1_overflow_en[5] – Enable Bus 1 Illegal Operation Overflow interrupt reserved[7:6] illegal_op2_en[8] – Enable Bus 2 Illegal Operation interrupt illegal_op2_overflow_en[9] – Enable Bus 2 Illegal Operation Overflow interrupt reserved[31:10]
0x018	INT_SET	RW	0x00	Interrupt Set: illegal_op0_set[0] – Set Bus 0 Illegal Operation interrupt illegal_op0_overflow_set[1] – Set Bus 0 Illegal Operation Overflow interrupt reserved[3:2] illegal_op1_set[4] – Set Bus 1 Illegal Operation interrupt illegal_op1_overflow_set[5] – Set Bus 1 Illegal Operation Overflow interrupt reserved[7:6] illegal_op2_set[8] – Set Bus 2 Illegal Operation interrupt illegal_op2_overflow_set[9] – Set Bus 2 Illegal Operation Overflow interrupt reserved[31:10] Writing 1 to a bit sets that interrupt
0x*N*00	CONTROL	RW	0x00	mux_sel[3:0] – Select which internal client is connected to the external SPI/QSPI pins 0: SPI/QSPI Monitor 1: Internal master interface 0 2-7: reserved flash_a_en[4] – Flash A is disabled (0) or enabled (1) flash_b_en[5] – Flash B is disabled (0) or enabled (1) reserved[7:6] init_cmd_filter[8] – Block initialization commands allow_4byte_addr[9] – Allow 4-byte addressing commands reserved[31:10]
0x*N*04	SPACE_EN	RW	0x00	Space monitoring enable bits space0_en[0] – Disable (0) or enable (1) monitoring of space 0 space1_en[1] – Disable (0) or enable (1) monitoring of space 1 space2_en[2] – Disable (0) or enable (1) monitoring of space 2 space3_en[3] – Disable (0) or enable (1) monitoring of space 3 space4_en[4] – Disable (0) or enable (1) monitoring of space 4 space5_en[5] – Disable (0) or enable (1) monitoring of space 5 space6_en[6] – Disable (0) or enable (1) monitoring of space 6 space7_en[7] – Disable (0) or enable (1) monitoring of space 7 reserved[31:8]
0x*N*08	READ_DUMMY_NUM	RW	0x08	Number of dummy cycles in an SPI flash read The minimum allowed value is 1. See the flash device data sheet for details. num_dummy_cycles[4:0] reserved[31:5]

Offset	Register Name	Access	Reset Value	Description
0xN10	MAXIMUM_ADDRESS	RW	0xFFFFFFFF	max_addr[31:0] – SPI transaction starting addresses and incremental addresses are masked with this value before comparison with address space ranges.
0xN14	COMMAND_DISABLE0	RW	0x00000000	command_disable[31:0] – When set to 1, this field disables individual command checking. Each bit corresponds to a specific parameter command. See Table 4.5 for details on each bit field.
0xN18	COMMAND_DISABLE1	RW	0x00000000	command_disable[8:0] – When set to 1, this field disables individual command checking. Each bit corresponds to a specific parameter command. See Table 4.5 for details on each bit field. reserved[31:9]
0x*N*20	SPACE0_FILTER_CTRL	RW	0x03	prg_cmd_allow[0] – Allow (whitelist) program commands in space 0 erase_cmd_allow[1] – Allow (whitelist) erase commands in space 0 read_cmd_block[2] – Block (blacklist) read commands in space 0 reserved[31:3]
0x*N*24	SPACE0_START_ADDR	RW	0x00000000	page_start_addr[31:8] – Start address for space 0, aligned to 256-byte page boundary reserved[7:0] – Writes are ignored; Reads return 0
0x*N*28	SPACE0_END_ADDR	RW	0x000000FF	page_end_addr[31:8] – End address for space 0, aligned to 256-byte page boundary reserved_ff[7:0] – Writes are ignored; Reads return 0xFF.
0x*N*40	SPACE1_FILTER_CTRL	RW	0x03	prg_cmd_allow[0] – Allow (whitelist) program commands in space 1 erase_cmd_allow[1] – Allow (whitelist) erase commands in space 1 read_cmd_block[2] – Block (blacklist) read commands in space 1 reserved[31:3]
0x*N*44	SPACE1_START_ADDR	RW	0x00000000	page_start_addr[31:8] – Start address for space 1, aligned to 256-byte page boundary reserved[7:0] – Writes are ignored; Reads return 0.
0x*N*48	SPACE1_END_ADDR	RW	0x000000FF	page_end_addr[31:8] – End address for space 1, aligned to 256-byte page boundary reserved_ff[7:0] – Writes are ignored; Reads return 0xFF.
0x*N*60	SPACE2_FILTER_CTRL	RW	0x03	prg_cmd_allow[0] – Allow (whitelist) program commands in space 2 erase_cmd_allow[1] – Allow (whitelist) erase commands in space 2 read_cmd_block[2] – Block (blacklist) read commands in space 2 reserved[31:3]
0x*N*64	SPACE2_START_ADDR	RW	0x00000000	page_start_addr[31:8] – Start address for space 2, aligned to 256-byte page boundary. reserved[7:0] – Writes are ignored; Reads return 0.
0x*N*68	SPACE2_END_ADDR	RW	0x000000FF	page_end_addr[31:8] – End address for space 2, aligned to 256-byte page boundary. reserved_ff[7:0] – Writes are ignored; Reads return 0xFF.

Offset	Register Name	Access	Reset Value	Description
0x*N*80	SPACE3_FILTER_CTRL	RW	0x03	prg_cmd_allow[0] – Allow (whitelist) program commands in space 3 erase_cmd_allow[1] – Allow (whitelist) erase commands in space 3 read_cmd_block[2] – Block (blacklist) read commands in space 3 reserved[31:3]
0x*N*84	SPACE3_START_ADDR	RW	0x00000000	page_start_addr[31:8] – Start address for space 3, aligned to 256-byte page boundary reserved[7:0] – Writes are ignored; Reads return 0.
0x*N*88	SPACE3_END_ADDR	RW	0x000000FF	page_end_addr[31:8] – End address for space 3, aligned to 256-byte page boundary reserved_ff[7:0] – Writes are ignored; Reads return 0xFF.
0x*N*F0	ILLEGAL_CMD	RO	0x00	illegal_cmd[7:0]: Illegal operation command reserved[31:8]
0x*N*F4	ILLEGAL_ADDR	RO	0x00000000	Illegal operation address

4.7. Initialization Command Filtering

When initialization command filtering is enabled, the QSPI Monitor watches for all of the Initializations commands (see Table 4.2). If one of these commands is detected, the transaction is terminated immediately, the command is recorded in the illegal_cmd register, illegal_addr is set to 0, and an illegal operation interrupt is sent.

By default, filtering for initialization commands is disabled. In a typical use case, initialization commands are allowed for a certain period of time (such as during boot up) and then filtering can be enabled through the register interface.

4.8. Address Filtering

The QSPI Monitor can filter program, erase, and read commands based on address ranges. Up to four address ranges (also called spaces) can be monitored, and filtering can be enabled independently for program, erase, and read commands for each space. Each space consists of a start address, end address, and whitelist/blacklist indicators for each type of command. Address spaces are aligned on 256-byte page boundaries. The default setting for all spaces is to allow (whitelist) program, erase, and read operations in that space. The settings for each space can be modified to block (blacklist) program, erase, or read operations. Each type of operation (program, erase, or read) has a separate whitelist/blacklist setting.

Program/erase operations are considered illegal for all addresses except spaces that have been whitelisted.

- If a program operation starts from a page address that is not inside a whitelisted address space, it is considered illegal.
- If an erase operation starts from an address that is not inside a whitelisted address space, or starts from an address inside a whitelisted address space but the address range goes outside the whitelisted address space, it is considered illegal.

Read operations are allowed for all addresses except spaces that have been blacklisted.

- If a read operation starts from an address that is inside a blacklisted address space, or starts from an address outside a blacklisted address space and the incremental address crosses into a blacklisted address space, it is considered illegal.

When an illegal operation is detected, the transaction is terminated immediately, the command and address are saved in the illegal_cmd and illegal_addr registers, and an illegal operation interrupt is generated.

Because program/erase operations are blacklisted by default and read operations are whitelisted by default, the recommended usage model is to only define whitelist areas for program/erase operations and blacklist areas for read operations as address spaces.

Overlapping program/erase whitelist and read blacklist address spaces should be avoided because it can lead to unintended consequences, such as an address range being writable but not readable. This prevents common use cases such as the host verifying data written to flash by reading it back.

4.9. Command Disable

Filtering for individual commands can be disabled by writing to the COMMAND_DISABLE0 and COMMAND_DISABLE1 register (see Table 4.5). By default, all commands are enabled.

Table 4.5. QSPI Monitor Command Disable Register Fields

Command Register	Field Index	Command
COMMAND_DISABLE0	0	Initialization Command 0
	1	Initialization Command 1
	2	Initialization Command 2
	3	Initialization Command 3
	4	Initialization Command 4
	5	Initialization Command 5
	6	Initialization Command 6
	7	Initialization Command 7
	8	Initialization Command 8
	9	Initialization Command 9
	10	Page Program Command
	11	Page Program Quad Address Quad Data Command
	12	Erase 4KB Command
	13	Erase 32KB Command
	14	Erase 64KB Command
	15	Read Command
	16	Fast Read Command
	17	Read Dual Data
	18	Read Dual Address Dual Data Command
	19	Read Quad Data Command
	20	Read Quad Address Quad Data Command
	21	Quad SPI Mode Enter Command
	22	Quad SPI Mode Exit Command
	23	4-byte Mode Enter Command
	24	4-byte Mode Exit Command
	25	4-byte Read Extended Address Command
	26	4-byte Write Extended Address Command
	27	4-byte Page Program Command
	28	4-byte Page Program Quad Address Quad Data Command
	29	4-byte Erase 4KB Command
	30	4-byte Erase 32KB Command
	31	4-byte Erase 64KB Command
COMMAND_DISABLE1	0	4-byte Read Command
	1	4-byte Fast Read Command
	2	4-byte Read Dual Data Command
	3	4-byte Read Dual Address Dual Data Command
	4	4-byte Read Quad Data Command
	5	4-byte Read Quad Address Quad Data Command
	Others	Reserved

4.9.1. 24/32-Bit Addressing

Flash devices larger than 128 Mb (16 MB) provide three separate mechanisms for addressing beyond the traditional 24-bit address space:

- Commands to enter/exit 4-byte mode (EN4B/EX4B)
  When the flash is in 4-byte mode, commands which normally take a 3-Byte address (read, erase, program, and others) expect 4-byte addresses instead of 3-byte addresses. The default is 3-byte mode.
- Extended Address Register (EAR)
  The EAR is an 8-bit register in the flash, which can be read and written using special commands (RDEAR/WREAR). When the flash is in 3-byte mode, the EAR is used to select which 128 Mbit segment is addressed by the 3-byte address. In other words, the value in EAR is used as the upper 8 bits of the 32-bit flash address (flash_addr[31:0] = {EAR, addr[23:16], addr[15:8], addr[7:0]}). The EAR default value is 0.
- 4-byte Address Commands
  The 4-byte commands, such as READ4B, FAST_READ4B, are separate commands from the standard 3-byte commands, such as READ, FAST_READ. The 4-byte commands always take 4-byte addresses, regardless of whether the flash is in 4-byte or 3-byte mode, and do not use the EAR.

When the monitor is configured to allow 32-bit addressing, the monitor internally tracks the addressing status of the flash (3-byte/4-byte mode, EAR) based on commands observed on the SPI/QSPI bus and uses this information to filter addresses observed on the bus. When the flash is in 4-byte mode or a 4-byte command is detected, the monitor compares the 32-bit address on the bus with the configured address spaces to determine if the operation is illegal or allowed. When the flash is in 3-byte mode, the monitor compares the 32-bit value comprised of EAR and the 24-bit address on the bus with the configured address spaces to determine if the operation is illegal or allowed.

All address comparisons are performed with the full 32-bits to prevent aliasing between 24-bit and 32-bit addresses which could result in security holes or false illegal operation detection.

When the monitor is configured to not allow 32-bit addressing (allow_4byte_addr = 0), the monitor is set to 3-byte mode, EAR is set to 0, and all of the 4-byte commands defined in the QSPI Command List Table are considered illegal operations. If one of these commands is detected, the transaction is terminated immediately, the command and address are recorded in the illegal_cmd and illegal_addr registers, and an illegal operation interrupt is sent.

4.10. Unrecognized Command Filtering

If a command is detected that does not match any of the commands defined in the QSPI Command List Table (see Table 4.2), the transaction is terminated immediately, the command is recorded in the illegal_cmd register, illegal_addr is set to 0, and an illegal operation interrupt is sent.

4.11. Timing Sequence

4.11.1. Illegal Command Blocking

If one of the illegal commands is detected (Figure 4.2), the transaction is terminated immediately by extending chip select and adding a clock pulse to confuse the SPI flash.

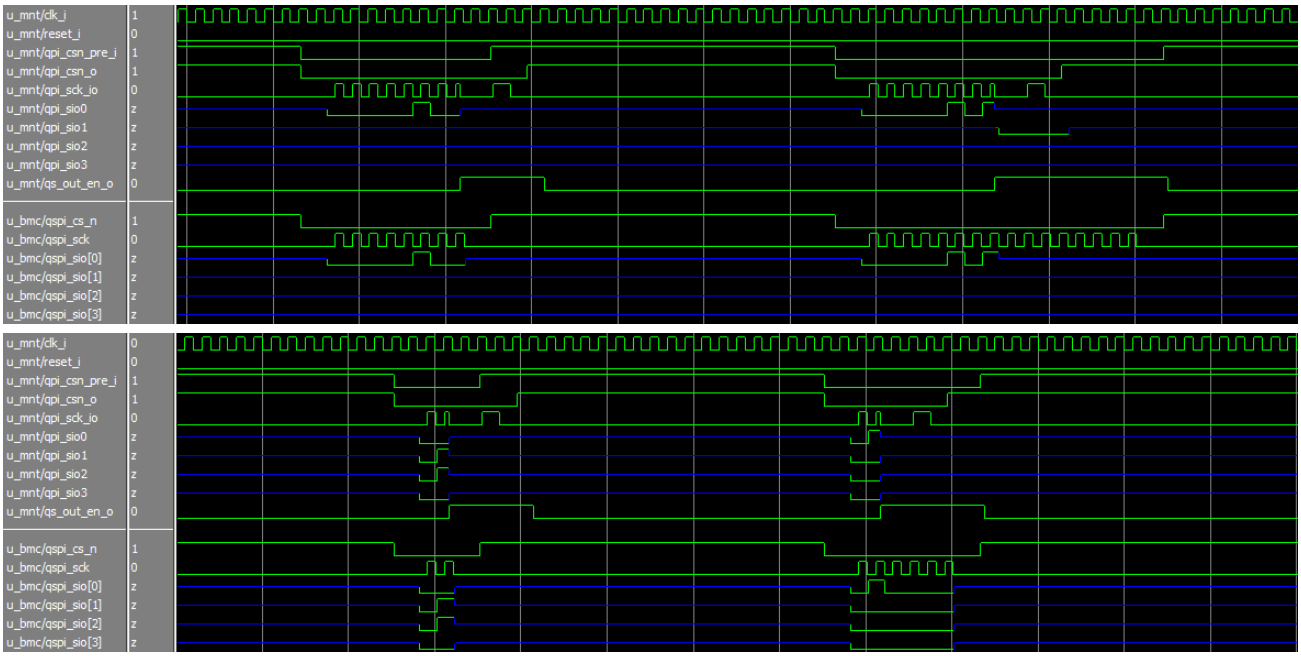
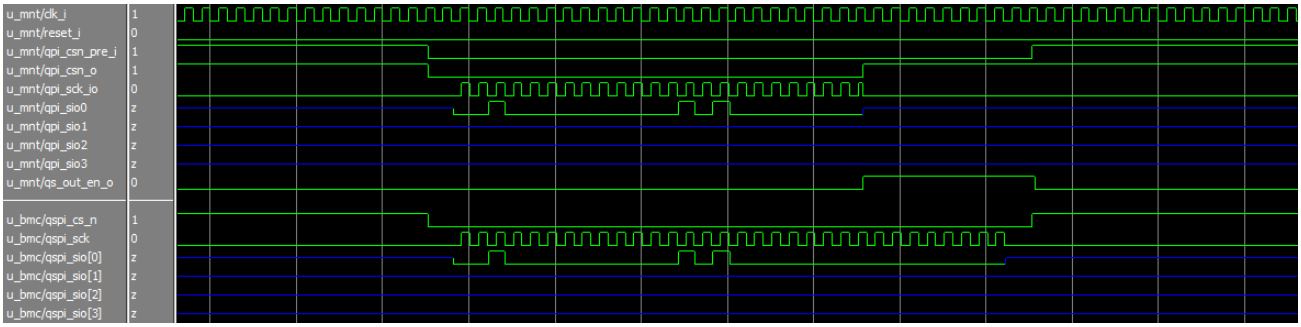


Figure 4.2. One Illegal Command

4.11.2. Illegal Erase Command Breaking (3-Byte Address)

If an illegal erase command is detected (Figure 4.3), the transaction is terminated immediately by driving chip select high.



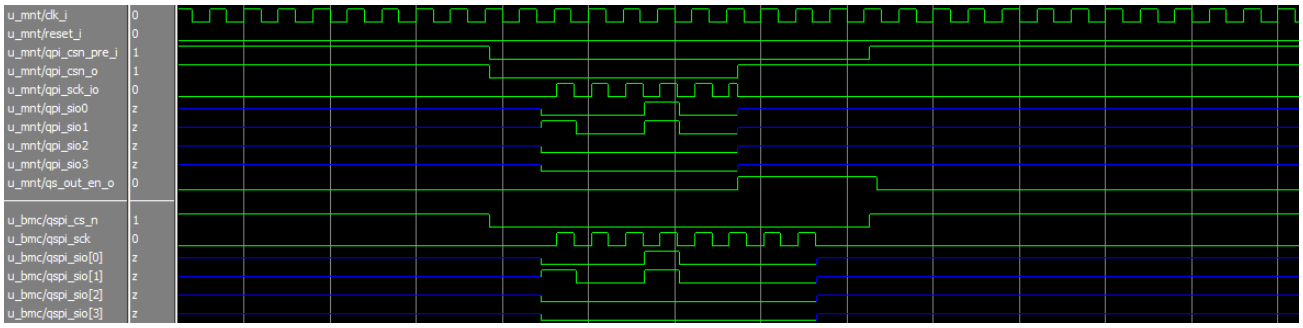


Figure 4.3. Illegal Erase Command

4.11.3. Illegal Program Command Breaking (3-Byte Address, Illegal Start Address)

If an illegal program command is detected (Figure 4.4), the transaction is terminated immediately by driving chip select high.

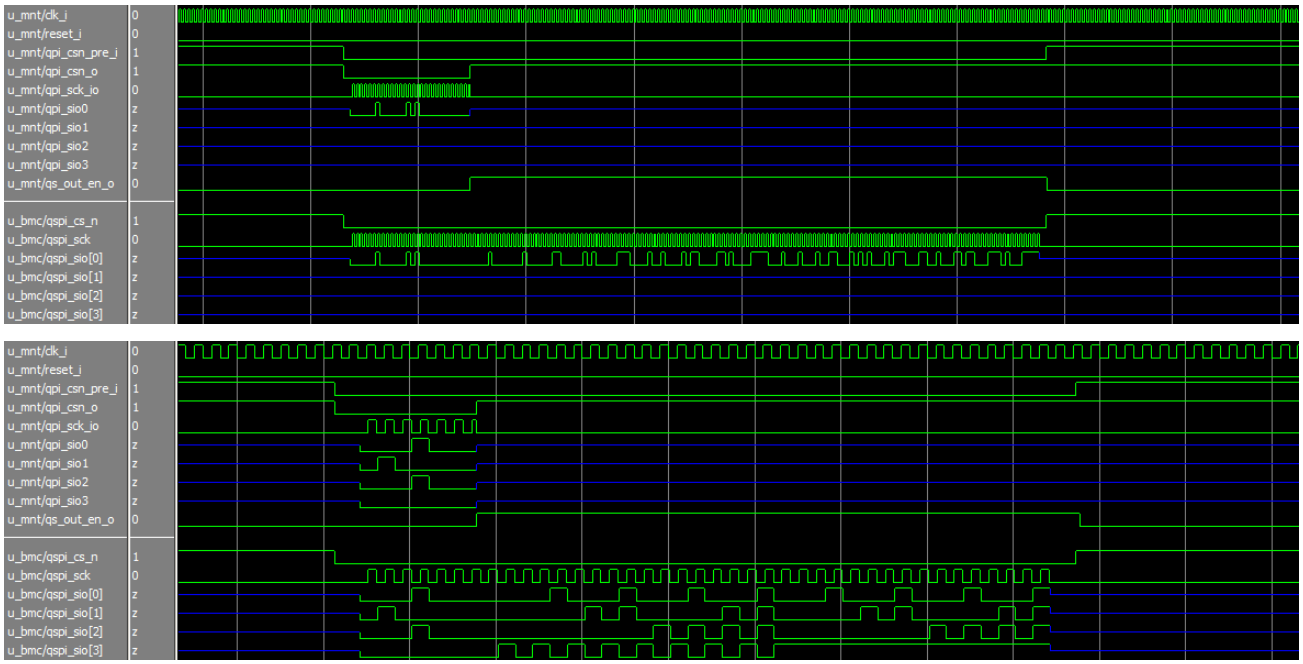


Figure 4.4. Illegal Program Command (3-Byte Address, Illegal Start Address)

4.11.4. Illegal Read Command Breaking (3-Byte Address, Illegal Start Address)

If an illegal read command is detected (Figure 4.5), the transaction is terminated immediately by driving chip select high.

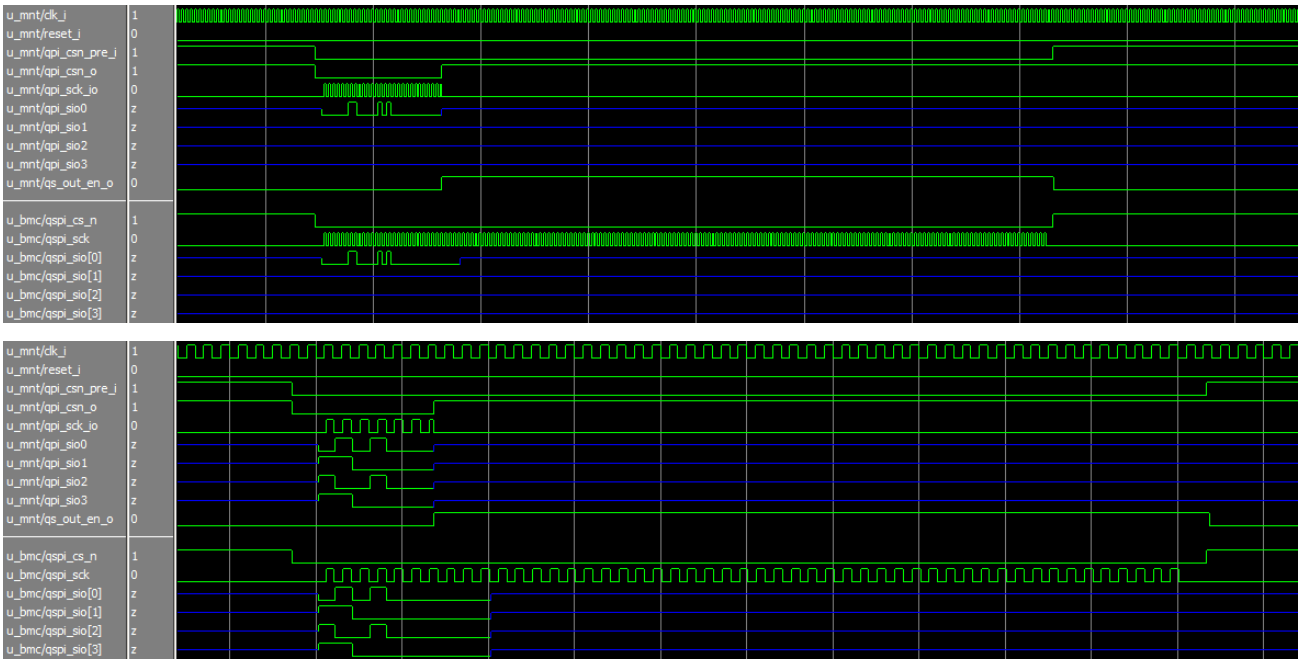


Figure 4.5. Illegal Read Command (3-Byte Address, Illegal Start Address)

4.11.5. Illegal Read Command Breaking (3-Byte Address, Incremental Address Overflow)

If a read command incremental address overflow is detected (Figure 4.6), the transaction is terminated immediately by driving chip select high.

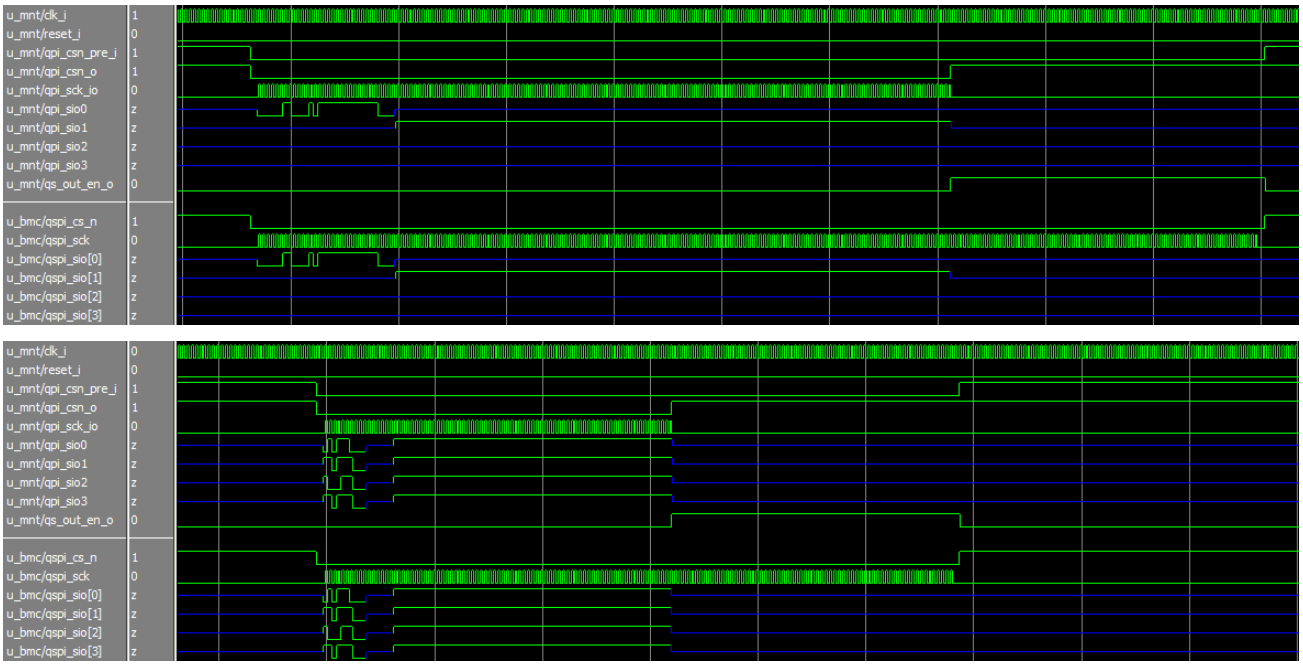


Figure 4.6. Illegal Read Command (3-Byte Address, Incremental Address Overflow)

4.11.6. Illegal 4-Byte Command Breaking

If a 4-byte command is disabled and a 4-byte command is detected (Figure 4.7), the transaction is terminated immediately by driving chip select high.

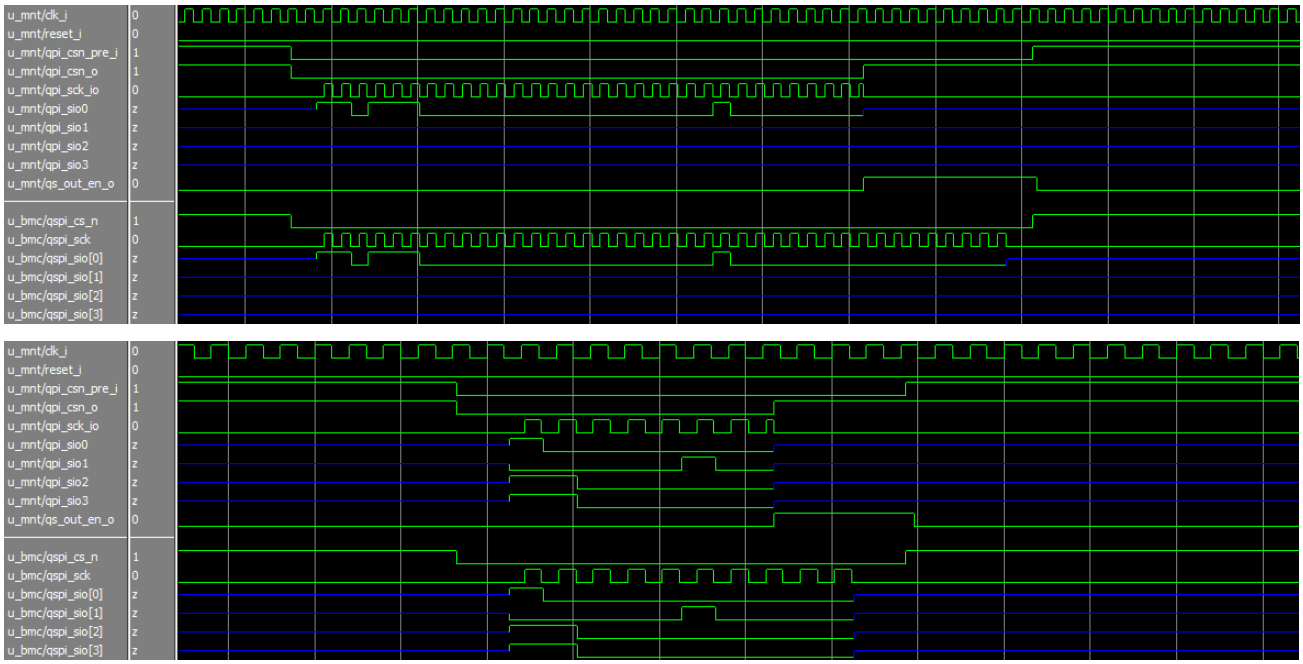


Figure 4.7. Illegal 4-Byte Command Breaking

4.12. Mux/Demux Functionality

Each external SPI/QSPI bus can be connected either to its corresponding monitor, or to the QSPI Master Streamer through a mux/demux block. This allows the QSPI Master Streamer to disable the monitor and access the external flash. Each bus/monitor/mux combination is independent of the others. It is the responsibility of the firmware to manage the muxes appropriately to prevent the internal SPI/QSPI master from being connected to more than one external bus at a time.

4.13. Internal Switching

When internal switching is enabled, the switch which connects the external SPI Master (BMC, PCH, and others) to the SPI flash is implemented inside the FPGA soft logic instead of being implemented externally on the board with a quick switch device. Regardless of whether the switch is internal or external, the SPI bus is still monitored by the QSPI Monitor. When any illegal activity is detected on the SPI bus, the internal switch is disabled, disconnecting the external SPI master from the slave. The internal switch only supports SPI communications; DSPI and QSPI are not supported. For DSPI and QSPI, an external quick switch device is required.

Figure 4.8 shows the diagram for the internal switch.

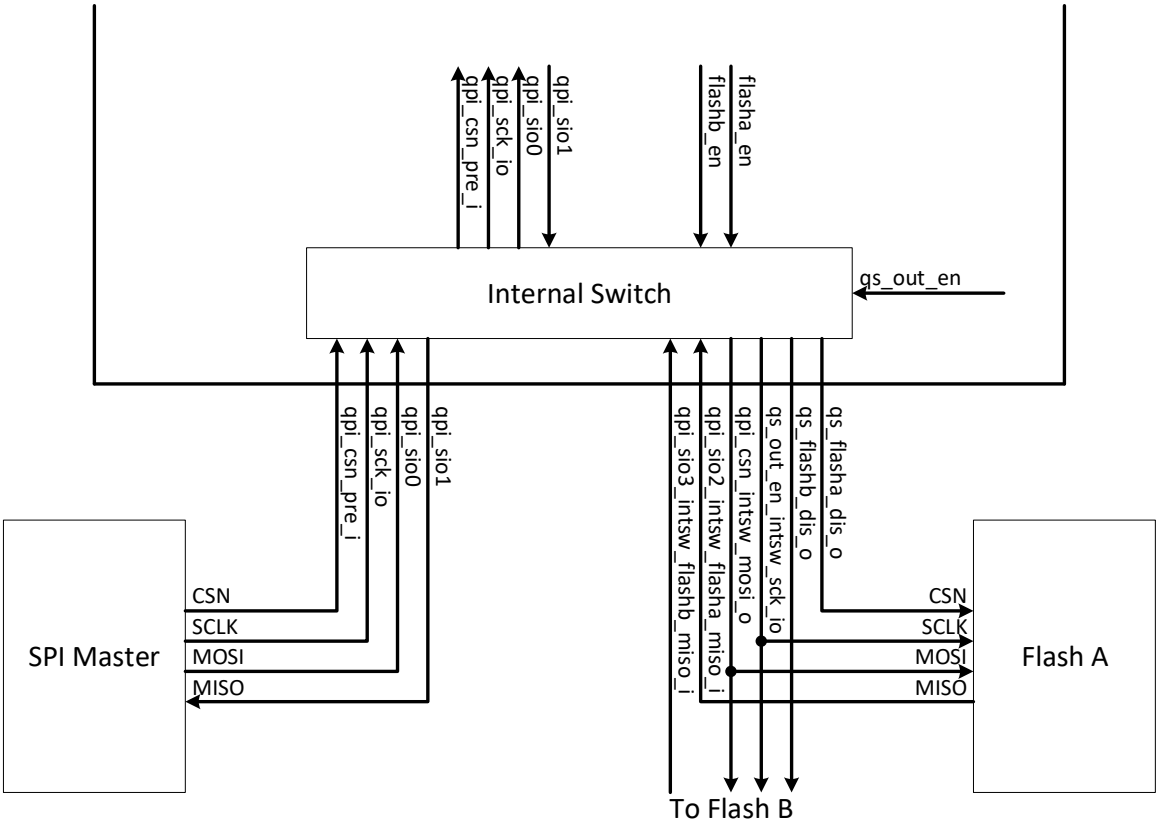


Figure 4.8. QSPI Internal Switch

5. QSPI Master Streamer

The QSPI Master Streamer is a configurable SPI master, which can support SPI, DSPI, and QSPI slaves. It contains FIFOs for Tx and Rx data, which support page read and page program (256 bytes). It also provides an external Rx FIFO output interface (8-bit) which is connected to the Secure Enclave and System Memory.

The QSPI Master Streamer provides significant performance improvement by supporting data read and write transactions of programmable length, allowing an entire SPI flash device to be read in one SPI transaction. The Secure Enclave FIFO output interface (8-bit) also enables direct transmission of input data from the SPI slave to the High Speed Port of the Secure Enclave, without tying up the CPU or system bus.

5.1. Features

The key features of the QSPI Master Streamer include:

- Generation of SPI, DSPI, and QSPI transactions
- Support for long SPI transactions (up to 256-byte write and 4 Gb read) with no CPU interactions
- Programmable transaction type and length
- Provision of external 8-bit FIFO interface for connecting to other blocks

5.2. Block Diagram

QSPI Master Streamer Block Diagram is shown in Figure 5.1. There are Tx and Rx FIFOs with each having a 32-bit access port for the system bus (APB) and an 8-bit access port for the SPI Master state machine. 8-bit data is packed or unpacked into 32-bit chunks as it enters or leaves the FIFOs.

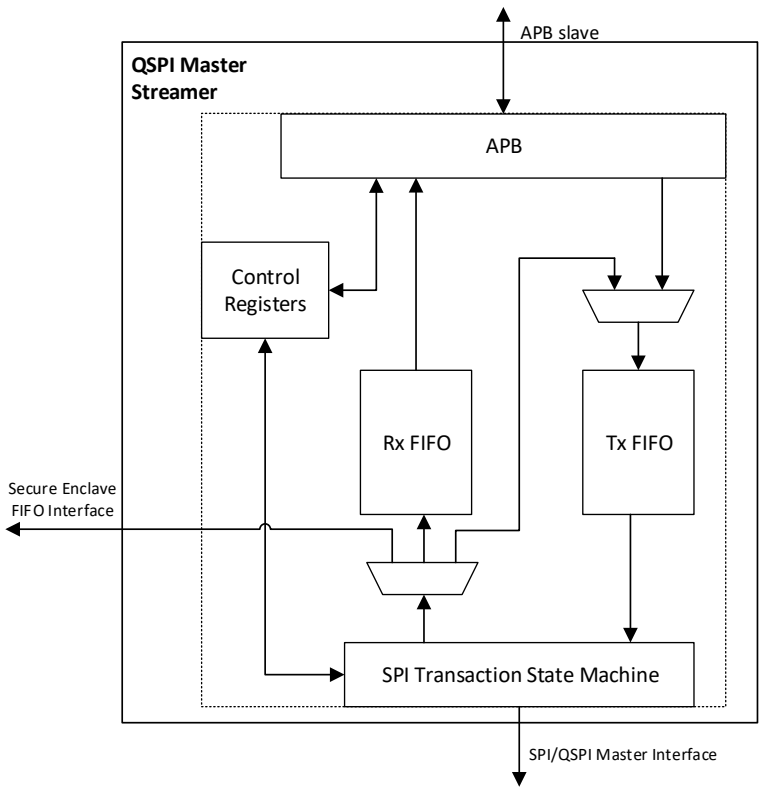


Figure 5.1. QSPI Master Streamer Block Diagram

5.3. FIFO Configuration

The QSPI Master Streamer FIFO configuration is shown in Table 5.1.

Table 5.1. QSPI Streamer FIFO Configuration

Attribute	Configuration	Notes
Tx FIFO Size	512	—
Tx FIFO Almost Full Flag	256	—
Tx FIFO Almost Empty Flag	4	—
Tx FIFO Endianness	Big	APB Tx FIFO Data: 31:24, 23:16, 15:8, 7:0 Big endian: 0, 1, 2, 3
Rx FIFO Size	256	—
Rx FIFO Almost Full Flag	252	—
Rx FIFO Almost Empty Flag	4	—
Rx FIFO Endianness	Big	Received bytes from SPI are packed in this order (from 0-3): APB Rx FIFO Data: 31:24, 23:16, 15:8, 7:0 Big endian: 0, 1, 2, 3

5.4. Register Description

The QSPI Master Streamer IP core register map is shown in the Table 5.2.

Table 5.2. QSPI Master Streamer IP Core Registers

Offset	Name	Access	Reset Value	Description
0x00	QSPI_CTRL	RW	0x00000004	spi_mode[1:0] • 00: SPI mode 0 • 01: reserved • 10: reserved • 11: SPI mode 3 sck_div[4:2]: • 0: Fqpi_sck_o = Fclk_i • 1: Fqpi_sck_o = Fclk_i/2 • 2: Fqpi_sck_o = Fclk_i/4 • 3: Fqpi_sck_o = Fclk_i/8 • 4: Fqpi_sck_o = Fclk_i/16 • 5: Fqpi_sck_o = Fclk_i/32 reserved[30:5] soft_reset[31] Writing 1 to this bit resets all of the internal logic, flushes the FIFOs (resets the read/write pointers), and restores all registers to their default settings. Reads return 0. Intended for error recovery.
0x04	CMD_DATA	RW	0x0	Command data to transmit in transaction phase 1 (always big endian)
0x08	TX_FIFO_DATA	WO	0x0	Data to transmit in transaction phase 2 When the Tx FIFO is full, register writes to this address is blocked until the FIFO is no longer full. Tx FIFO status is available in the fifo_ctrl and int_status registers.
0x0C	RX_FIFO_DATA	RO	0x0	Data received in transaction phase 4 If the Rx FIFO contains less than four bytes when a 32-bit read is received on the system bus and there is a SPI transaction currently in progress, the read is blocked until 4 bytes are received or the SPI transaction completes.

Offset	Name	Access	Reset Value	Description
0x10	TRANSACTION_CTRL1	RW	0x0	ph1_num_bytes[2:0] – Number of bytes from cmd_data to transmit in transaction phase 1 (legal values: 0-4) ph2_num_bytes[11:3] – Number of bytes from Tx FIFO to transmit in transaction phase 2 (legal values: 0- *Tx FIFO Size*) ph3_dummy_cycles[16:12] – Number of dummy cycles to transmit in transaction phase 3 ph1_mode[18:17] – Transmit phase 1 data in: • 0: SPI mode • 1: DSPI mode • 2: QSPI mode • 3: reserved ph2_mode[20:19] – Transmit phase 2 data in: • 0: SPI mode • 1: DSPI mode • 2: QSPI mode • 3: reserved ph3_mode[22:21] – Transmit phase 3 dummy cycles in: • 0: SPI mode • 1: DSPI mode • 2: QSPI mode • 3: reserved ph4_mode[24:23] – Receive phase 4 data in: • 0: SPI mode • 1: DSPI mode • 2: QSPI mode • 3: reserved rxfifo_last_en[25] – Enable(1)/Disable(0) assertion of rxfifo_last_o for the last received byte of the SPI transaction reserved[30:26] start[31] – Write 1 to start an SPI transaction (reads return 0)
0x14	TRANSACTION_CTRL2	RW	0x0	ph4_num_bytes[31:0] – Number of bytes to receive in transaction phase 4
0x18	STATUS	RO	0x0	tx_fifo_empty[0] – Tx FIFO is empty tx_fifo_almost_empty[1] – Tx FIFO is not empty and has less than *Tx FIFO Almost Empty Flag* bytes tx_fifo_almost_full[2] – Tx FIFO is not full and has more than *Tx FIFO Almost Full Flag* bytes tx_fifo_full[3] – Tx FIFO is full rx_fifo_empty[4] – Rx FIFO is empty rx_fifo_almost_empty[5] – Rx FIFO is not empty and has less than *Rx FIFO Almost Empty Flag* bytes rx_fifo_almost_full[6] – Rx FIFO is not full and has more than *Rx FIFO Almost Full Flag* bytes reserved[30:8] busy[31] – SPI transaction is in progress
0x1C	FIFO_CTRL	RW	0x0	reserved[6:0] tx_fifo_flush[7] – Flush contents of Tx FIFO (reset read and write pointers) rx_fifo_dest[9:8]: • 0: internal Rx FIFO • 1: external Rx FIFO interface • 2: reserved • 3: internal Tx FIFO

Offset	Name	Access	Reset Value	Description
				reserved[14:10] rx_fifo_flush[15]: flush contents of Rx FIFO (reset read and write pointers) reserved[31:16]
0x20	INT_STATUS	RW	0x0	Interrupt status: done_int[0] – Done interrupt (SPI transaction completed) tx_fifo_empty_int[1] – Tx FIFO Empty interrupt tx_fifo_almost_empty_int[2] – Tx FIFO Almost Empty interrupt tx_fifo_almost_full_int[3] – Tx FIFO Almost Full interrupt tx_fifo_full_int[4] – Tx FIFO Full interrupt rx_fifo_empty_int[5] – Rx FIFO Empty interrupt rx_fifo_almost_empty_int[6] – Rx FIFO Almost Empty interrupt rx_fifo_almost_full_int[7] – Rx FIFO Almost Full interrupt rx_fifo_full_int[8] – Rx FIFO Full interrupt reserved[31:9] Writing 1 to a bit clears that interrupt FIFO interrupts are triggered on the rising edge of the corresponding FIFO condition (empty, full, etc.) and stay asserted until cleared by writing a 1 to this register to clear the interrupt. Current status of the FIFO conditions is always available in the status register.
0x24	INT_ENABLE	RW	0x0	Interrupt enable: done_en[0] – Enable Done interrupt (SPI transaction completed) tx_fifo_empty_en[1] – Enable Tx FIFO Empty interrupt tx_fifo_almost_empty_en[2] – Enable Tx FIFO Almost Empty interrupt tx_fifo_almost_full_en[3] – Enable Tx FIFO Almost Full interrupt tx_fifo_full_en[4] – Enable Tx FIFO Full interrupt rx_fifo_empty_en[5] – Enable Rx FIFO Empty interrupt rx_fifo_almost_empty_en[6] – Enable Rx FIFO Almost Empty interrupt rx_fifo_almost_full_en[7] – Enable Rx FIFO Almost Full interrupt rx_fifo_full_en[8] – Enable Rx FIFO Full interrupt reserved[31:9]
0x28	INT_SET	RW	0x0	Interrupt set: done_set[0] – Set Done interrupt (SPI transaction completed) tx_fifo_empty_set[1] – Set Tx FIFO Empty interrupt tx_fifo_almost_empty_set[2] – Set Tx FIFO Almost Empty interrupt tx_fifo_almost_full_set[3] – Set Tx FIFO Almost Full interrupt tx_fifo_full_set[4] – Set Tx FIFO Full interrupt rx_fifo_empty_set[5] – Set Rx FIFO Empty interrupt rx_fifo_almost_empty_set[6] – Set Rx FIFO Almost Empty interrupt rx_fifo_almost_full_set[7] – Set Rx FIFO Almost Full interrupt rx_fifo_full_set[8] – Set Rx FIFO Full interrupt reserved[31:9]

5.5. Secure Enclave FIFO Interface

The Secure Enclave FIFO interface supports the transfer of large streams of data from SPI flash to the Secure Enclave and System Memory. This allows firmware images to be loaded in the High Speed Port of the Security Enclave for faster authentication and the CPU firmware image being loaded into System Memory.

5.6. Operation

5.6.1. Transaction Phases

The QSPI Master Streamer generates an SPI or a QSPI transaction in multiple phases, as shown in Figure 5.2. Each phase is controlled by separate register settings. In the typical usage model, the CPU programs all of the transaction phase registers with the settings for the desired transaction, then write 1 to bit[31] of the TRANSACTION_CTRL1 register to initiate SPI transactions. For transactions which use data, the CPU should write data to the FIFO before starting the transaction (see examples sequence below for details).

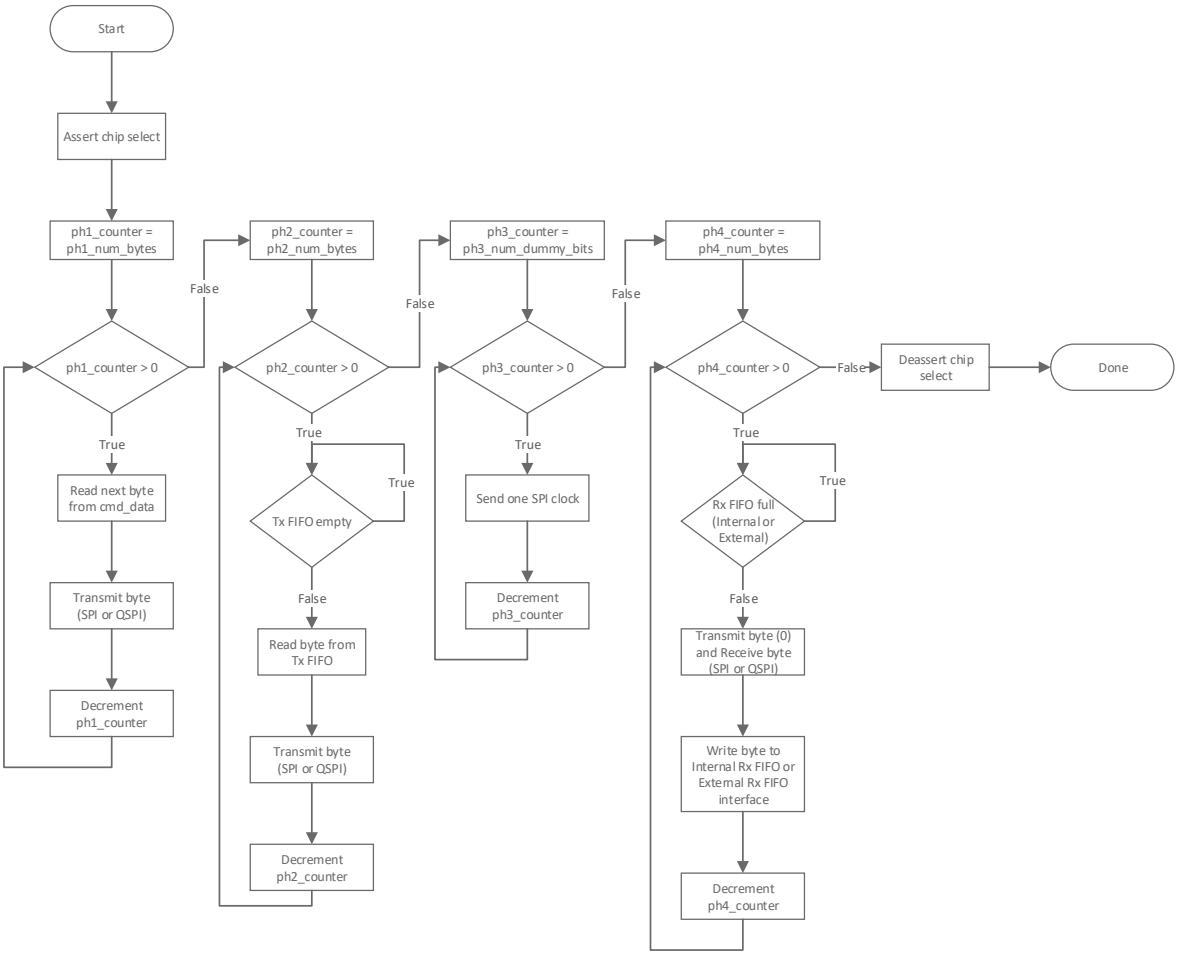


Figure 5.2. QSPI Master Streamer Programmable Phases

Phase 1: Transmit ph1_num_bytes (0-4) bytes from cmd_data register

- For SPI flash devices, this normally includes 1 command byte and 0 or 3 address bytes.
- Data is transmitted in SPI mode, DSPI mode, or QSPI mode depending on the ph1_mode setting in transaction_ctrl1.
- Serial data input is ignored

Phase 2: Transmit ph2_num_bytes (0-1028) bytes from Tx FIFO

- For SPI flash devices, this is normally used for page program data and/or 4 byte addressing.
- Data is transmitted in SPI mode DSPI mode, or QSPI mode depending on the ph2_mode setting in transaction_ctrl1.
- Serial data input is ignored.

Phase 3: Transmit ph3_num_dummy_bits (0-15) bits

- For SPI flash devices, this is normally used to generate dummy cycles for read data commands.
- Dummy data (0) is transmitted in SPI mode, DSPI mode, or QSPI mode depending on the ph3_mode setting.
- Serial data input is ignored.

Phase 4: Receive ph4_num_bytes (0-4GB) bytes and send to Rx FIFO

- For SPI flash devices, this is normally used for read commands.
- Data is received in SPI mode, DSPI mode, or QSPI mode depending on the ph4_mode setting.
- Received data is stored in Rx FIFO or sent out the External Rx FIFO interface depending on the rx_fifo_dest.
- Serial data output is 0 for SPI or high impedance for QSPI.
- SPI slave ignores the data.

SPI Flash Page Program example:

cmd_data = 0x02xxxxxx (where xxxxxx = 24-bit Flash address).

Tx FIFO contains DataByte1…DataByte16 values

ph1_num_bytes = 4, ph1_mode = 0

ph2_num_bytes = N, ph2_mode = 0 (N=16 in example)

ph3_num_dummy_bits = 0, ph3_mode = 0

ph4_num_bytes = 0, ph4_mode = 0

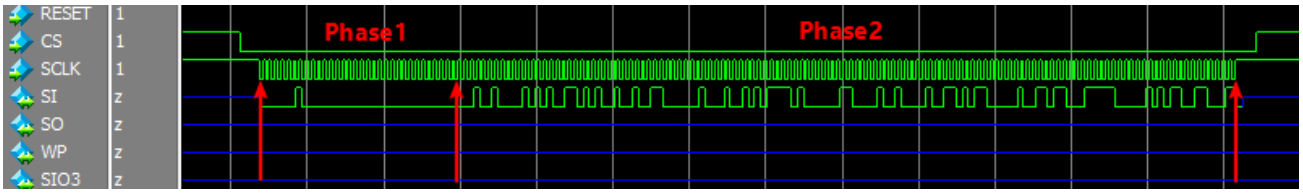


Figure 5.3. Example for PP Program Sequence

SPI Flash FAST_READ example:

cmd_data = 0x0Bxxxxxx (where xxxxxx = 24-bit address).

ph1_num_bytes = 4, ph1_mode = 0

ph2_num_bytes = 0, ph2_mode = 0

ph3_num_dummy_bits = N, ph3_mode = 0 (N=8 in example)

ph4_num_bytes = M, ph4_mode = 0 (M=16 in example)

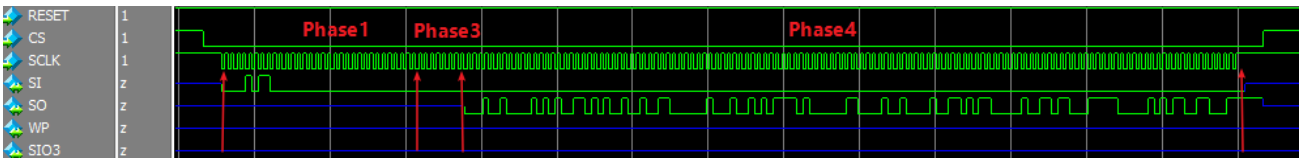


Figure 5.4. Example for FAST_READ Sequence

SPI RDID example:

cmd_data = 0x9F000000

ph1_num_bytes = 1, ph1_mode = 0

ph2_num_bytes = 0, ph2_mode = 0

ph3_num_dummy_bits = 0, ph3_mode = 0

ph4_num_bytes = 3, ph4_mode = 0

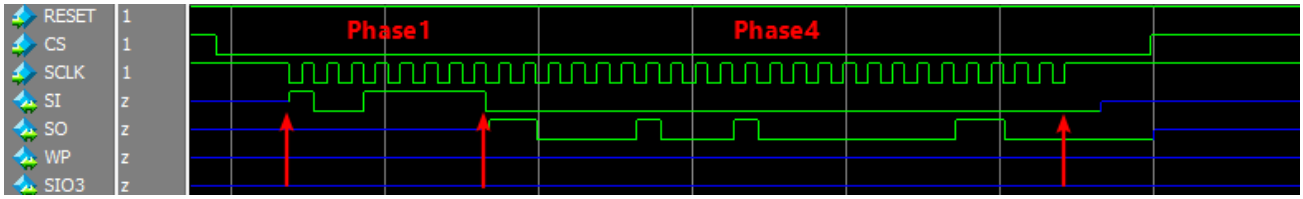


Figure 5.5. Example for RDID Sequence

SPI Flash QREAD4B example:

cmd_data = 0x6C000000

Tx FIFO contains 4-byte Read Address

ph1_num_bytes = 1, ph1_mode = 0

ph2_num_bytes = 4, ph2_mode = 0

ph3_num_dummy_bits = N, ph3_mode = 0 (N=8 in example)

ph4_num_bytes = M, ph4_mode = 2 (M=64 in example)

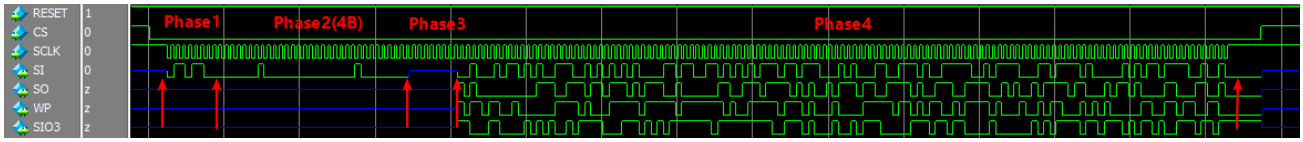


Figure 5.6. Example for QREAD4B Sequence

5.6.2. Width Conversion

Each Tx and Rx FIFO has a 32-bit access port for the system bus (APB) and an 8-bit access port for the SPI Master state machine. The 8-bit data is packed or unpacked into 32-bit chunks as it enters or leaves the FIFOs. The endianness of the 32-bit data is big endian, see Table 5.1.

Wherever possible, the implementation should avoid stalling the system bus while doing width conversions. For example, on the Tx FIFO, the 32-bit write value should be stored in a local register and the system bus write cycle should be terminated before doing the four 8-bit writes to the Tx FIFO. On the Rx FIFO, the logic should read bytes from the Rx FIFO into a local 32-bit register whenever the Rx FIFO is not empty, so that the 32-bit value can be returned immediately whenever a system bus read is received. This avoids tying up the system bus and stalling the CPU while the width conversions are being performed.

5.6.3. FIFO Empty/Full Behavior

The recommended usage model is for the CPU to write all of the data for a transaction to the Tx FIFO (for example, a full 256-byte page) before starting the transaction so that the Tx FIFO does not become empty in the middle of a transaction.

If the Rx FIFO indicates that it is full before the transaction is completed, then the SPI/QSPI state machine stalls until the Rx FIFO is no longer full. When this stall occurs, qpi_csn_o is held asserted but the SPI/QSPI clock is gated off (held in the inactive state). When the Rx FIFO is not full, the clock is gated back on and data is received over SPI/QSPI.

5.6.4. Typical Flash Read/Program Flow

The typical flash (MX25L12845G, MACRONIX, CO, Ltd) read/program flow is shown in Figure 5.7.

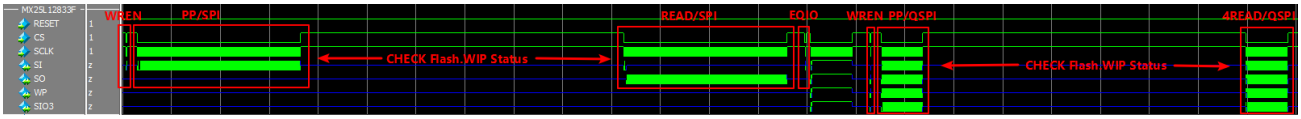


Figure 5.7. Typical Flash Read/Program Flow

6. I²C Monitor

The I²C Monitor is an I²C access and command module that can monitor the traffic on the I²C bus to identify potential illegal traffic based on a pre-defined library. Once illegal traffic is detected, this I²C Monitor informs the host through the status flag and/or interrupt. With user option, the current communication can be disrupted by disabling the I²C bus.

The I²C Monitor compares the first eight bytes of I²C bus traffic immediately after the slave I²C address with the pre-defined filters in the database. The filters are set up by the system host through the APB Bus. Once a matching event is detected, it informs the host using status sampling through the APB Bus or dedicated interrupt. Meanwhile, according to user setup, it disables the current I²C communication to prevent catastrophic damage to the system.

The block diagram of the I²C Monitor is shown in Figure 6.1.

6.1. Features

The key features of the I²C Monitor include:

- Monitoring of traffic on an I²C bus for illegal activity based on programmable filter conditions
- Protects bus from illegal activity by driving bus low

6.2. Block Diagram

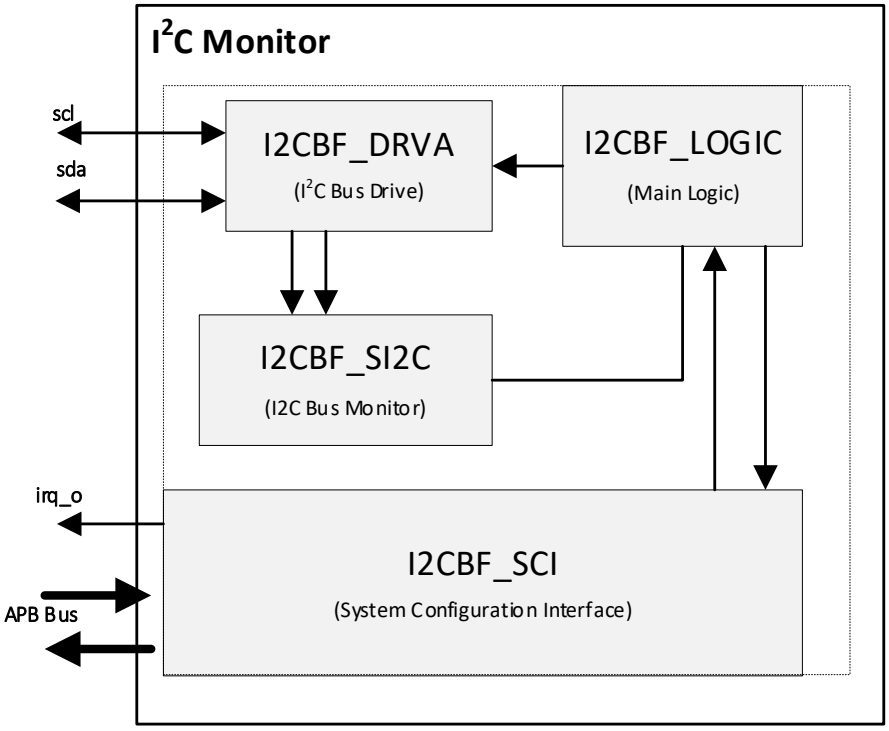


Figure 6.1. I²C Monitor Block Diagram

6.3. Signal Description

Table 6.1. I²C Monitor Signal Description

Signal	Direction	Description
I²C Monitor		
I2C_MONx_SCL	Bidir	Clock (Input during monitor, drives low when exception)
I2C_MONx_SDA	Bidir	Data (Input during monitor, drives low when exception)

6.4. Register Description

The register address map specifies the available I²C Monitor core registers and which are accessible through APB.

Table 6.2. I²C Monitor Core Registers

Offset	Register Name	Access	Default value	Description
0x1FC	I2CBF_CR	R/W	32'H00000000	Control Register • Bit [31:8]: Unused • Bit [7]: i2cbf_en – This bit enables the I2CBF IP to perform the I2C bus traffic event detection. • Bit [6]: bus_stop – If asserted, this bit allows the host to unconditionally disable the I2C bus by driving both SCL and SDA low. The bus is released after this bit is written with 0. • Bit [5]: bus_dis_en – This bit enables the I2CBF IP to disable the I2C bus in case an event is detected. • Bit [4:0]: total_number_of_entry These five bits allow the host to inform the I2CBF IP of the number entries defined for the event detection. The current maximum number is 20.
0x1F8	I2CBF_INTENR	R/W	32'H00000000	Interrupt Enable Register • Bit [31:8]: Unused • Bit[7:6]: Reserved • Bit[5]: int_en – This bit enables the interrupt from the I2CBF IP to the system host in case an I²C traffic event is detected. • Bit [4:0]: Reserved
0x1F4	I2CBF_INTSETR	R/W	32'H00000000	Interrupt Set Register • Bit [31:8]: Unused • Bit[7:6]: Reserved Bit[5]: int_set – This bit enables the interrupt from the I2CBF IP to the system host in case an I²C traffic event is detected. • Bit [4:0]: Reserved
0x1F0	I2CBF_SR	R	32'H00000000	Status Register • Bit [31:8]: Unused • Bit [7:6]: Reserved • Bit [5]: event_detected – This bit indicates that there is an event detected from the I²C bus traffic based on the entry table setup. • Bit [4:0]: entry_number_for_current_event These four bits indicate the entry number for the current event. It is stable once an event is detected until the I2CBF IP is reset by the host.

Offset	Register Name	Access	Default value	Description
0x140	Reserved	—	—	—
0x13C	ENTRY20_D	R/W	32'H00000000	ENTRY 20 [127:96] • Bit [31]: entry_enable – This bit enables this particular entry for the event detection. • Bit [30]: 10_bits_address – This bit enables the 10 bits address for I^2C slave address checking. • Bit [29:27]: i2c_10_bits_address_msb – These bits are the MSB 3 bits of the 10 bits address if the bit [126] is set. • Bit [26:20]: 7_bits_address – These bits are the I^2C Slave Address for I^2C bus traffic checking. • Bit [19]: rw – This bit is the RW checking bit for I^2C bus traffic monitoring. The same as the I^2C Bus Specification, 1 is for read and 0 for write. • Bit [18]: rw_nc – This bit is used to disable the RW bit checking. Once set, the RW bit is ignored for I^2C bus traffic checking, hence, both I^2C bus read and write are monitored. • Bit [17:16]: check_mode – These two bits are used to set up the checking mode for this entry. The provided modes are shown in Table 6.3. • Bit [15:8]: detection_enable_mask – These eight bits are the active High Mask to enable corresponding received byte(s) among the eight bytes immediately after the I^2C slave address for event detection. • Bit [7:0]: bit_wide_operation_selection_mask – These eight bits are the active High Mask to alter corresponding byte event detection from pattern matching to bit detection against the corresponding Check Data Byte. See the detailed format in Table 6.4.
0x138	ENTRY20_C	R/W	32'H00000000	ENTRY 20 [95:64] Bit [31:0]: check_data_mask_byte – These four bytes are the checking data for the event detection or Bit Mask bytes depending on the entry check mode setup. For Mode 00 and Mode 01, these four bytes serve as Bit Mask bytes for bitwise checking selection. The mask bytes order corresponds to the position of the SET bit (1) within the Bit-Wide Operation Selection Mask (Entry Bit [103:96]). The larger mask number index corresponds to the MSB side of the Bit-Wide Operation Selection Mask. A maximum of four Bit Mask bytes can be selectively enabled per entry. For Mode 10 and Mode 11, these four bytes are part of the check data bytes (total of 12 bytes). See the detailed format in Table 6.4.
0x134	ENTRY20_B	R/W	32'H00000000	ENTRY20 [63:32] Bit [31:0]: check_data_byte – These first four bytes of data are the checking data for the event detection. See the detailed format in Table 6.4.
0x130	ENTRY20_A	R/W	32'H00000000	ENTRY20 [31:0] Bit [31:0]: check_data_byte – These next four bytes of data are the checking data for the event detection. See the detailed format in Table 6.4.

Offset	Register Name	Access	Default value	Description
0x12C	ENTRY19_D	R/W	32'H00000000	Refer to Entry20 Register
0x128	ENTRY19_C	R/W	32'H00000000	Refer to Entry20 Register
0x124	ENTRY19_B	R/W	32'H00000000	Refer to Entry20 Register
0x120	ENTRY19_A	R/W	32'H00000000	Refer to Entry20 Register
—	—	—	—	ENTRY2_A – ENTRY18_D Refer to Entry20 Register
—	—	—	—	
—	—	—	—	
—	—	—	—	
0x0C	ENTRY1_D	R/W	32'H00000000	Refer to Entry20 Register
0x08	ENTRY1_C	R/W	32'H00000000	Refer to Entry20 Register
0x04	ENTRY1_B	R/W	32'H00000000	Refer to Entry20 Register
0x00	ENTRY1_A	R/W	32'H00000000	Refer to Entry20 Register

Table 6.3. Check Mode Table

Check Mode	Description	Detection Enable Mask	Bit-Wide Operation Selection Mask	Data Byte Utilized
2'b00 Check Mode 1	• Multiple byte checking against Data Byte 1–8 based on the Detection Enable Mask and Bit-Wide Operation Selection Mask setting • Event trigger by ALL enabled bytes matching to the corresponding receiving bytes	Utilized, Multiple Hot	Utilized, Max 4 Hot	Data Byte 1–8 Bit Mask 1–4
2'b01 Check Mode 2	• Multiple byte checking against Data Byte 1–8 based on the Detection Enable Mask and Bit-Wide Operation Selection Mask setting • Event trigger by ANY enabled bytes matching to the corresponding receiving bytes	Utilized, Multiple Hot	Utilized, Max 4 Hot	Data Byte 1–8 Bit Mask 1–4
2'b10 Check Mode 3	• Single byte checking against entry Data Byte 1–12, based on the Detection Enable Mask setting • Event is triggered if the specified receiving byte DOES NOT match any byte among the entry Data Byte 1–12.	Utilized, One Hot	—	Data Byte 1–12
2'b11 Check Mode 4	• Single byte checking against the entry Data Byte 1–12 based on the Detection Enable Mask Setting. • Event is triggered if the specified receiving byte DOES match one of the entry Data Byte 1–12.	Utilized, One Hot	—	Data Byte 1–12

Each Entry Data is 128 bits in length. The field's assignments for each I²C Monitor Entry Data is shown Table 6.4. The APB host, through the 32 bits APB Bus, can read/write this True Dual Port memory.

Table 6.4. Data Entry Format

Bits	**127**	**126**	**125**	**124**	**123**	**122**	**121**	**120**	**119**	**118**	**117**	**116**	**115**	**114**	**113**	**112**
Name	Entry Enable	10 Bits Address	10 Bits Address MSB			I²C 7 Bits Address							RW	RW NC	Check Mode	
Bits	**111**	**110**	**109**	**108**	**107**	**106**	**105**	**104**	**103**	**102**	**101**	**100**	**99**	**98**	**97**	**96**
Name	Detection Enable Mask								Bit-Wide Operation Selection Mask							
Bits	**95**	**94**	**93**	**92**	**91**	**90**	**89**	**88**	**87**	**86**	**85**	**84**	**83**	**82**	**81**	**80**
Name	Check Data Byte 12/Bit Mask 4								Check Data Byte 11/Bit Mask 3							
Bits	**79**	**78**	**77**	**76**	**75**	**74**	**73**	**72**	**71**	**70**	**69**	**68**	**67**	**66**	**65**	**64**
Name	Check Data Byte 10/Bit Mask 2								Check Data Byte 9/Bit Mask 1							
Bits	**63**	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**
Name	Check Data Byte 8								Check Data Byte 7							
Bits	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Name	Check Data Byte 6								Check Data Byte 5							
Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Name	Check Data Byte 4								Check Data Byte 3							
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	Check Data Byte 2								Check Data Byte 1							

6.5. Module Description

6.5.1. I2CBF_SCI

I2CBF_SCI is the System Configuration Interface for the I²C Monitor, which contains the configurable memory for the IP. The system host can access this memory through the 32 bits APB bus. The memory/registers are:

- Control register, read and write access.
- Status register, read-only.
- True Dual Port memory for the entry data for event detection, which can be read/written by the host through the 32-bit APB bus, and read by the I2CBF_LOGIC through the 128-bit internal data bus.

6.5.2. I2CBF_SI2C

The I2CBF_SI2C module monitors the I²C bus activities, detects the START/STOP conditions, examines the I²C slave address, and fetches the first eight data bytes. It provides the received address, RW bits, received data bytes along with the data byte count number to the I2CBF_LOGIC module for event detection. Also, for some application scenarios, it can provide the receiving bit count and acknowledge information to I2CBF_DRVX block for I²C bus disable control.

6.5.3. I2CBF_LOGIC

The I2CBF_LOGIC module gets the current I²C bus activities from the I2CBF_SI2C module and the entry data from the I2CBF_SCI module. It then performs event detection based on the control register and entry data settings for every data byte received from the I²C bus traffic. Once an event is detected, it sets up corresponding status register bit and sends out interrupt to the host if control register setting allows.

6.5.4. I2CBF_DRVA

This I2CBF_DRVA module is controlled by the output of the I2CBF_LOGIC module to disable the I²C bus by driving both SCL and SDA low. This disrupts the I²C communication. Once the I²C bus is disabled, it could only be released by writing to the status register of the I²C Monitor from the host or by cycling the power supply. To only disrupt the I²C communication on the slave side without disabling the whole I²C bus, a different module, such as I2CBF_DRVB with bus multiplexer, can be deployed to replace the I2CBF_DRVA module.

6.6. Programming Flow

During the system initialization phase, the system host should download the Entry Table into the True Dual Port Memory inside the I2CBF_SCI block through the APB bus. To start the I²C bus monitoring process, the host should write the I2CBF_CR through the APB bus with the I2CBF_EN bit set to 1. The host should receive interrupt if an event is detected, if the INT_EN bit is set inside I2CBF_CR, or the host has to pull the I2CBF_SR to identify the I²C bus status.

If the BUS_DIS_EN bit inside the I2CBF_CR is set, the I²C bus is disabled for further communication between master and slave. To resume the I²C bus communication, the host should perform a write operation to the I2CBF_SR through the APB bus.

To stop I²C bus monitoring, the host should write the I2CBF_CR with the I2CBF_EN bit set to 0.

6.6.1. Example Data Alignment for Check Mode 1 and Mode 2

For Check Mode 1 and Check Mode 2, the I²C Monitor checks every time a new byte (for the first eight bytes immediately after the I²C Slave Address) is received for all available entries. For each entry, the Data and Mask alignment is shown in Figure 6.2.

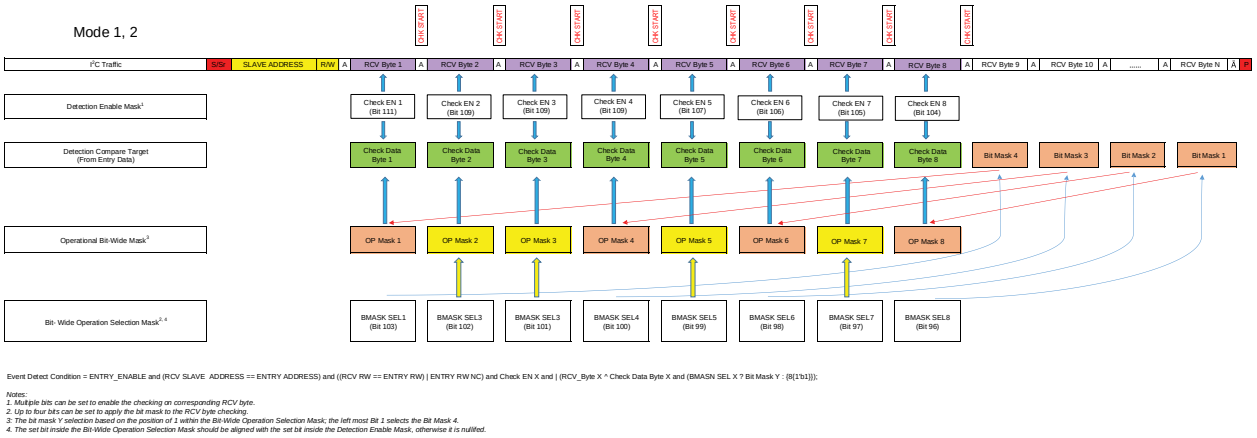


Figure 6.2. Check Mode 1 and Mode 2 Data Alignment

6.6.2. Example Data Alignment for Check Mode 3 and Mode 4

For Check Mode 3 and Check Mode 4, the I²C Monitor checks every time a new byte (for the first eight bytes immediately after the I²C Slave Address) is received for all available entries. However, the data arrangement is different. For each entry, the data alignment in Check Mode 3 and Check Mode 4 are demonstrated in Figure 6.3.

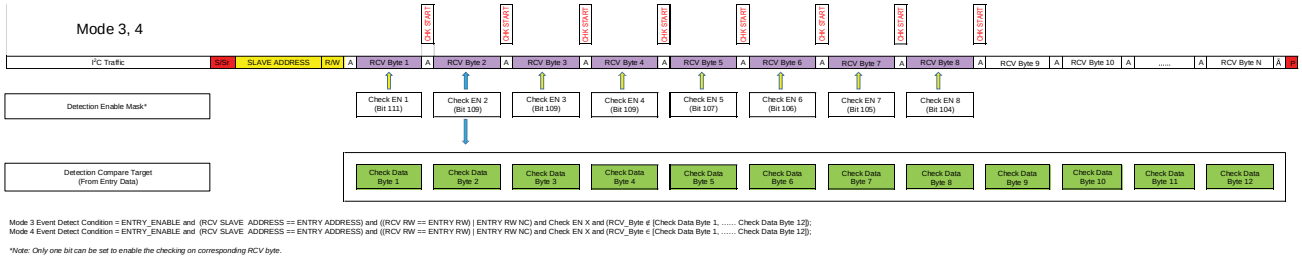


Figure 6.3. Check Mode 3 and Mode 4 Data Alignment

7. I²C/SMBus Slave

7.1. Overview

The I²C Slave provides device addressing, read/write operation and an acknowledgement mechanism.

7.2. Features

The key features of I²C Slave include:

- Supports 7-bit and 10-bit Addressing Mode
- Supports the following bus speeds:
  - Standard-mode (Sm) – up to 100 kbit/s
  - Fast-mode (Fm) – up to 400 kbit/s
  - Fast-mode Plus (Fm+) – up to 1 Mbit/s
- Supports Clock stretching
- Configurable ACK/NACK response on address and data phases
- Integrated Pull-up
- Integrated Glitch filter
- Polling and Out-of-band Interrupt Modes
- 8-byte Tx FIFO
- 16-byte Rx FIFO
- SMBus Support

7.3. Signal Description

Table 7.1. I²C Slave IP Core Signal Description

Signal	Direction	Description
I²C/SMBus Slave		
SMBUSx_INT	Output	SMBus Alert signal/I²C Interrupt
SMBUSx_SCL	Input	SMBus/I²C Serial Clock
SMBUSx_SDA	Bidir	SMBus/I²C Data

7.4. Register Description

7.4.1. Overview

The I²C Slave Core configuration registers are located at the addresses shown in Table 7.2.

Table 7.2. I²C Slave Registers Address Map

Offset	Register Name	Access Type	Description
0x00	RD_DATA_REG	RO	Read Data Register
0x00	WR_DATA_REG	WO	Write Data Register
0x04	SLVADR_L_REG	R/W	Slave Address Lower Register
0x08	SLVADR_H_REG	R/W	Slave Address Higher Register
0x0C	CONTROL_REG	R/W	Control Register
0x10	TGT_BYTE_CNT_REG	R/W	Target Byte Count Register
0x14	INT_STATUS1_REG	RW1C	Interrupt Status First Register
0x18	INT_ENABLE1_REG	R/W	Interrupt Enable First Register
0x1C	INT_SET1_REG	WO	Interrupt Set First Register
0x20	INT_STATUS2_REG	RW1C	Interrupt Status Second Register
0x24	INT_ENABLE2_REG	R/W	Interrupt Enable Second Register
0x28	INT_SET2_REG	WO	Interrupt Set Second Register
0x2C	FIFO_STATUS_REG	RO	FIFO Status Register
0x30–0x3C	Reserved	RSVD	Reserved Write access is ignored and 0 is returned on read access.

The RD_DATA_REG and WR_DATA_REG share the same offset. Write access to this offset goes to WR_DATA_REG while read access goes to RD_DATA_REG.

7.4.2. Write Data Register (WR_DATA_REG)

Table 7.3 shows the Write Data Register. This is the interface to Transmit FIFO. Writing to WR_DATA_REG pushes a word to Transmit FIFO. When writing to WR_DATA_REG, the host should ensure that Transmit FIFO is not full. This can be done by reading FIFO_STATUS_REG. Data is popped WR_DATA_REG during I²C read transaction. When reset is performed, the contents of Transmit FIFO are not reset but the FIFO control logic is reset. Thus, content is not guaranteed after reset.

Table 7.3. Write Data Register

Field	Name	Access	Width	Reset
[7:0]	tx_fifo	WO	8	not guaranteed

7.4.3. Read Data Register (RD_DATA_REG)

Table 7.4 shows the Read Data register. This is the interface to Receive FIFO. After a data is received from I²C bus during I²C write transaction, the received data is pushed to Receive FIFO. Reading from RD_DATA_REG pops a word from Receive FIFO. The host should ensure that Receive FIFO has data before reading RD_DATA_REG, data is not guaranteed when this register is read during Receive FIFO empty condition. On the other hand, if Receive FIFO is full but I²C Slave continues to receive data, new data is lost. Read FIFO_STATUS_REG to determine the status of Receive FIFO. Similar to Transmit FIFO, the reset value of Receive FIFO is also not guaranteed after reset.

Table 7.4. Read Data Register

Field	Name	Access	Width	Reset
[7:0]	rx_fifo	RO	8	not guaranteed

7.4.4. Slave Address Registers (SLAVE_ADDRL_REG, SLAVE_ADDRH_REG)

The Slave Address Lower Register (SLAVE_ADDRL_REG) shown in Table 7.5 is a 7-bit Slave address. This is used for 7-bit and 10-bit addressing mode as follows:

- For 7-bit Addressing Mode, it is the Slave address
- For 10-bit Addressing Mode, it is the lower 7 bits of the Slave address

Table 7.5. Slave Address Lower Register

Field	Name	Access	Width	Reset
[7]	Reserved	RSVD	1	–
[6:0]	slave_addr_l_reg	RW	7	0x51

The Slave Address Higher Register (SLAVE_ADDRH_REG) shown in Table 7.6 is the upper 3 bits of 10-bit Slave address. This is not used in 7-bit addressing mode.

Table 7.6. Slave Address Higher Register

Field	Name	Access	Width	Reset
[7:3]	Reserved	RSVD	5	–
[2:0]	slave_addr_h_reg	RW	3	0x0

7.4.5. Control Register (CONTROL_REG)

Table 7.7 shows the summary of Control Register. This each bit of this register controls the behavior of I^{2}C Slave Core.

Table 7.7. Control Register

Field	Name	Access	Width	Reset
[7:5]	Reserved	RSVD	3	—
[4]	nack_data	RW	1	1'b0
[3]	nack_addr	RW	1	1'b0
[2]	Reset	WO	1	1'b0
[1]	clk_stretch_en	RW	1	1'b0
[0]	addr_10bit_en	RW	1	1'b0

- nack_data

  NACK on Data Phase. Specifies ACK/NACK response on I^{2}C data phase.
  1'b0 – Sends ACK to received data
  1'b1 – Sends NACK to received data

- nack_addr

  NACK on Address Phase. Specifies ACK/NACK response on I^{2}C address phase.
  1'b0 – Sends ACK to received address if it matches the programmed slave address
  1'b1 – Sends NACK to received data

- reset

  Reset. Resets I^{2}C Slave Core for 1 clock cycle. The registers and APB interface are not affected by this reset. This is write-only bit because it has auto clear feature; it is cleared to 1'b0 after 1 clock cycle.
  1'b0 – No action.
  1'b1 – Resets I^{2}C Slave Core.

- clk_stretch_en

  Clock Stretch Enable. Enables clock stretching on ACK bit of data.
  1'b0 – I^{2}C Slave Core releases SCL signal

1'b1 – I²C Slave Core pulls down SCL signal on the next ACK bit of data phase and keeps pulling down until the host writes 1'b0 on this bit.

- addr_10bit_en

  10-bit Address Mode Enable. Enables the reception of 10-bit I²C address.
  1'b0 – I²C Slave Core rejects the 10-bit I²C address, it sends NACK.
  1'b1 – I²C Slave Core responds to 10-bit I²C address. If SLAVE_ADDRH_REG.slave_addr_h_reg is 3'h0, it also responds to 7-bit address.

7.4.6. Target Byte Count Register (TGT_BYTE_CNT_REG)

Table 7.8 shows the summary of Target Byte Count Register. The desired number of bytes to transfer (read/write) in I²C bus should be written to this register. This is used for Transfer Complete interrupt generation – asserts when the target byte count is achieved.

Table 7.8. Target Byte Count Register

Field	Name	Access	Width	Reset
[7:0]	byte_cnt	RSVD	8	8'h00

7.4.7. Interrupt Status Registers (INT_STATUS1_REG, INT_STATUS2_REG)

Table 7.9 and Table 7.10 show the Interrupt Status Register (INT_STATUS1_REG and INT_STATUS2_REG) which contains all the interrupts currently pending in the I²C Slave Core. When an interrupt bit asserts, it remains asserted until it is cleared by the host by writing 1'b1 to the corresponding bit.

The interrupt status bits are independent of the interrupt enable bits; in other words, status bits may indicate pending interrupts, even though those interrupts are disabled in the Interrupt Enable Register, see the Interrupt Enable Registers (INT_ENABLE1_REG, INT_ENABLE2_REG) section for details. The logic which handles interrupts should mask (bitwise and logic) the contents of INT_STATUS1_REG and INT_ENABLE1_REG registers as well as INT_STATUS2_REG and INT_ENABLE2_REG to determine the interrupts to service. The int_o interrupt signal is asserted whenever both an interrupt status bit and the corresponding interrupt enable bits are set.

Table 7.9. Interrupt Status First Register

Field	Name	Access	Width	Reset
[7]	tr_cmp_int	RW1C	1	1'b0
[6]	stop_det_int	RW1C	1	1'b0
[5]	tx_fifo_full_int	RW1C	1	1'b0
[4]	tx_fifo_aempty_int	RW1C	1	1'b0
[3]	tx_fifo_empty_int	RW1C	1	1'b0
[2]	rx_fifo_full_int	RW1C	1	1'b0
[1]	rx_fifo_afull_int	RW1C	1	1'b0
[0]	rx_fifo_ready_int	RW1C	1	1'b0

- tr_cmp_int

  Transfer Complete Interrupt Status. This interrupt status bit asserts when the number of bytes transferred in I^2C interface is equal to TGT_BYTE_CNT.byte_cnt.
  1'b0 – No interrupt
  1'b1 – Interrupt pending

- stop_det_int

  STOP Condition Detected Interrupt Status. This interrupt status bit asserts when STOP condition is detected after an ACK/NACK bit.
  1'b0 – No interrupt
  1'b1 – Interrupt pending

- tx_fifo_full_int

  Transmit FIFO Full Interrupt Status. This interrupt status bit asserts when Transmit FIFO changes from not full state to full state.
  1'b0 – No interrupt
  1'b1 – Interrupt pending

- tx_fifo_aempty_int

  Transmit FIFO Almost Empty Interrupt Status. This interrupt status bit asserts when the amount of data words in Transmit FIFO changes from 3 to 2.
  1'b0 – No interrupt
  1'b1 – Interrupt pending

- tx_fifo_empty_int

  Transmit FIFO Empty Interrupt Status. This interrupt status bit asserts when the last data in Transmit FIFO is popped-out, causing the FIFO to become empty.
  1'b0 – No interrupt
  1'b1 – Interrupt pending

- rx_fifo_full_int

  Receive FIFO Full Interrupt Status. This interrupt status bit asserts when RX FIFO full status changes from not full to full state.
  1'b0 – No interrupt
  1'b1 – Interrupt pending

- rx_fifo_afull_int

Receive FIFO Almost Full Interrupt Status. This interrupt status bit asserts when the amount of data words in Receive FIFO changes from 13 to 14.

1'b0 – No interrupt
1'b1 – Interrupt pending

- rx_fifo_ready_int

  Receive FIFO Ready Interrupt Status. This interrupt status bit asserts when Receive FIFO is empty and receives a data word from I^2C interface.

  1'b0 – No interrupt
  1'b1 – Interrupt pending

Table 7.10. Interrupt Status Second Register

Field	Name	Access	Width	Reset
[7:2]	reserved	RSVD	6	—
[1]	stop_err_int	RW1C	1	1'b0
[0]	start_err_int	RW1C	1	1'b0

- stop_err_int

  STOP Condition Error Interrupt Status. This interrupt status bit asserts after detecting a STOP condition when it is not expected. STOP condition is expected to occur only after the ACK/NACK bit. The stop_err_int and stop_det_int do not assert at the same time.

  1'b0 – No interrupt
  1'b1 – Interrupt pending

- start_err_int

  START Condition Error Interrupt Status. This interrupt status bit asserts after detecting a START condition when it is not expected. START condition is expected to occur only when I^2C bus is idle and after receiving an ACK or a NACK (repeated START condition).

  1'b0 – No interrupt
  1'b1 – Interrupt pending

7.4.8. Interrupt Enable Registers (INT_ENABLE1_REG, INT_ENABLE2_REG)

Table 7.11 and Table 7.12 show the summary of Interrupt Enable Registers that corresponds to interrupts status bits in INT_STATUS1_REG and INT_STATUS2_REG. They do not affect the contents of the INT_STATUS1_REG and INT_STATUS2_REG. If one of the INT_STATUS1_REG/ INT_STATUS2_REG bits asserts, and the corresponding bit of INT_ENABLE1_REG/INT_ENABLE2_REG is 1'b1, the interrupt signal int_o asserts.

Table 7.11. Interrupt Enable First Register

Field	Name	Access	Width	Reset
[7]	tr_cmp_en	RW	1	1'b0
[6]	stop_det_en	RW	1	1'b0
[5]	tx_fifo_full_en	RW	1	1'b0
[4]	tx_fifo_aempty_en	RW	1	1'b0
[3]	tx_fifo_empty_en	RW	1	1'b0
[2]	rx_fifo_full_en	RW	1	1'b0
[1]	rx_fifo_afull_en	RW	1	1'b0
[0]	rx_fifo_ready_en	RW	1	1'b0

- tr_cmp_en

  Transfer Complete Interrupt Enable. Interrupt enable bit corresponded to Transfer Complete Interrupt Status.

1'b0 – Interrupt disabled
1'b1 – Interrupt enabled

- stop_det_en

  STOP Condition Detected Interrupt Enable. Interrupt enable bit corresponded to STOP Condition Detected Interrupt Status.
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

- tx_fifo_full_en

  Transmit FIFO Full Interrupt Enable. Interrupt enable bit corresponded to Transmit FIFO Full Interrupt Status.
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

- tx_fifo_aempty_en

  Transmit FIFO Almost Empty Interrupt Enable. Interrupt enable bit corresponded to Transmit FIFO Almost Empty Interrupt Status.
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

- tx_fifo_empty_en

  Transmit FIFO Empty Interrupt Enable. Interrupt enable bit corresponded to Transmit FIFO Empty Interrupt Status.
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

- rx_fifo_full_en

  Receive FIFO Full Interrupt Enable. Interrupt enable bit corresponded to Receive FIFO Full Interrupt Status.
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

- rx_fifo_afull_en

  Receive FIFO Almost Full Interrupt Enable. Interrupt enable bit corresponded to Receive FIFO Almost Full Interrupt Status.
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

- rx_fifo_ready_en

  Receive FIFO Ready Interrupt Enable. Interrupt enable bit corresponded to Receive FIFO Ready Interrupt Status.
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

Table 7.12. Interrupt Enable Second Register

Field	Name	Access	Width	Reset
[7:2]	reserved	RSVD	6	–
[1]	stop_err_en	RW	1	1'b0
[0]	start_err_en	RW	1	1'b0

- stop_err_en

  STOP Condition Error Interrupt Enable. Interrupt enable bit corresponded to STOP Condition Error Interrupt Status.
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

- start_err_en

  START Condition Error Interrupt Enable. Interrupt enable bit corresponded to START Condition Error Interrupt Status.
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

7.4.9. Interrupt Set Registers (INT_SET1_REG, INT_SET2_REG)

Table 7.13 and Table 7.14 show the summary of Interrupt Set Registers. Writing 1'b1 to a register bit in INT_SET1_REG or INT_SET2_REG asserts the corresponding interrupts status bit in INT_STATUS1_REG or INT_STATUS2_REG while writing 1'b0 is ignored. This is intended for testing purposes only.

Table 7.13. Interrupt Set First Register

Field	Name	Access	Width	Reset
[7]	tr_cmp_set	WO	1	1'b0
[6]	stop_det_set	WO	1	1'b0
[5]	tx_fifo_full_set	WO	1	1'b0
[4]	tx_fifo_aempty_set	WO	1	1'b0
[3]	tx_fifo_empty_set	WO	1	1'b0
[2]	rx_fifo_full_set	WO	1	1'b0
[1]	rx_fifo_afull_set	WO	1	1'b0
[0]	rx_fifo_ready_set	WO	1	1'b0

- tr_cmp_set

  Transfer Complete Interrupt Set. Interrupt set bit corresponded to Transfer Complete Interrupt Status.
  1'b0 – No action
  1'b1 – Asserts INT_STATUS1_REG.tr_cmp_int

- stop_det_set

  STOP Condition Detected Interrupt Set. Interrupt set bit corresponded to STOP Condition Detected Interrupt Status.
  1'b0 – No action
  1'b1 – Asserts INT_STATUS1_REG.stop_det_int

- tx_fifo_full_set

  Transmit FIFO Full Interrupt Set. Interrupt set bit corresponded to Transmit FIFO Full Interrupt Status.
  1'b0 – No action
  1'b1 – Asserts INT_STATUS1_REG.tx_fifo_full_int

- tx_fifo_aempty_set

  Transmit FIFO Almost Empty Interrupt Set. Interrupt set bit corresponded to Transmit FIFO Almost Empty Interrupt Status.
  1'b0 – No action
  1'b1 – Asserts INT_STATUS1_REG.tx_fifo_aempty_int

- tx_fifo_empty_set

  Transmit FIFO Empty Interrupt Set. Interrupt set bit corresponded to Transmit FIFO Empty Interrupt Status.
  1'b0 – No action
  1'b1 – Asserts INT_STATUS1_REG.tx_fifo_empty_int

- rx_fifo_full_set

  Receive FIFO Full Interrupt Set. Interrupt set bit corresponded to Receive FIFO Full Interrupt Status.
  1'b0 – No action
  1'b1 – Asserts INT_STATUS1_REG.rx_fifo_full_int

- rx_fifo_afull_set

  Receive FIFO Almost Full Interrupt Set. Interrupt set bit corresponded to Receive FIFO Almost Full Interrupt Status.
  1'b0 – No action
  1'b1 – Asserts INT_STATUS1_REG.rx_fifo_afull_int

- rx_fifo_ready_set

  Receive FIFO Ready Interrupt Set. Interrupt set bit corresponded to Receive FIFO Ready Interrupt Status.
  1'b0 – No action
  1'b1 – Asserts INT_STATUS1_REG.rx_fifo_ready_int

Table 7.14. Interrupt Set Second Register

Field	Name	Access	Width	Reset
[7:2]	reserved	RSVD	6	–
[1]	stop_err_set	WO	1	1'b0
[0]	start_err_set	WO	1	1'b0

- stop_err_set

  STOP Condition Error Interrupt Set. Interrupt set bit corresponded to STOP Condition Error Interrupt Status.
  0 – No action.
  1 – Asserts INT_STATUS2_REG.stop_err_set.

- start_err_set

  START Condition Error Interrupt Set. Interrupt set bit corresponded to START Condition Error Interrupt Status.
  0 – No action.
  1 – Asserts INT_STATUS2_REG.start_err_set.

7.4.10. FIFO Status Register (FIFO_STATUS_REG)

FIFO Status Register reflects the status of Transmit FIFO and Receive FIFO as shown in Table 7.15.

Table 7.15. FIFO Status Register

Field	Name	Access	Width	Reset
[7:6]	reserved	RSVD	2	—
[5]	tx_fifo_full	RO	1	1'b0
[4]	tx_fifo_aempty	RO	1	1'b1
[3]	tx_fifo_empty	RO	1	1'b1
[2]	rx_fifo_full	RO	1	1'b0
[1]	rx_fifo_afull	RO	1	1'b0
[0]	rx_fifo_empty	RO	1	1'b1

- tx_fifo_full

  Transmit FIFO Full. This bit reflects the full condition of Transmit FIFO.
  1'b0 – Transmit FIFO is not full
  1'b1 – Transmit FIFO is full

- tx_fifo_aempty

  Transmit FIFO Almost Empty. This bit reflects the almost empty condition of Transmit FIFO.
  1'b0 – Data words in Transmit FIFO is greater than *TX FIFO Almost Empty Flag* attribute
  1'b1 – Data words in Transmit FIFO is less than or equal to *TX FIFO Almost Empty Flag* attribute

- tx_fifo_empty

  Transmit FIFO Empty. This bit reflects the empty condition of Transmit FIFO.
  1'b0 – Transmit FIFO is not empty – has at least 1 data word
  1'b1 – Transmit FIFO is empty

- rx_fifo_full

  Receive FIFO Full. This bit reflects the full condition of Receive FIFO.
  1'b0 – Receive FIFO is not full
  1'b1 – Receive FIFO is full

- rx_fifo_afull

  Receive FIFO Full. This bit reflects the almost full condition of Receive FIFO.
  1'b0 – Data words in Receive FIFO is less than *RX FIFO Almost Full Flag* attribute
  1'b1 – Data words in Receive FIFO is greater than or equal to *RX FIFO Almost Full Flag* attribute

- rx_fifo_empty

  Receive FIFO Full. This bit reflects the empty condition of Receive FIFO.
  1'b0 – Receive FIFO is not empty – has at least 1 data word
  1'b1 – Receive FIFO is empty

7.5. Operations Details

7.5.1. General I²C Operation

In the I²C bus, the transaction is always initiated by the master. A slave may not transmit data unless it has been addressed by the master. Each device on the I²C bus has a specific device address to differentiate between other devices that are on the same I²C bus. Data transfer is initiated only when the bus is idle. A bus is considered idle if both SDA and SCL lines are high after a STOP condition.

The general procedure for an I²C transaction is as follows:

1. Master wants to send data to a slave:
   - Master-transmitter sends a START condition and addresses the slave-receiver
   - Master-transmitter sends data to slave-receiver
   - Master-transmitter terminates the transfer with a STOP condition
2. Master wants to receive/read data from a slave:
   - Master-receiver sends a START condition and addresses the slave-transmitter
   - Master-receiver sends the requested register to read to slave-transmitter
   - Master-receiver receives data from the slave-transmitter
   - Master-receiver terminates the transfer with a STOP condition

I²C communication is initiated by the master sending a START condition and terminated by the master sending a STOP condition. Normal data on the SDA line must be stable during the high level of the SCL line. The High or Low state of the data line can only change when SCL is Low. The Start condition is a unique case and is defined by a High-to-Low transition on the SDA line while SCL is High. The Stop condition is a unique case and is defined by a Low-to-High transition on the SDA line while SCL is High. These are shown in Figure 7.1.

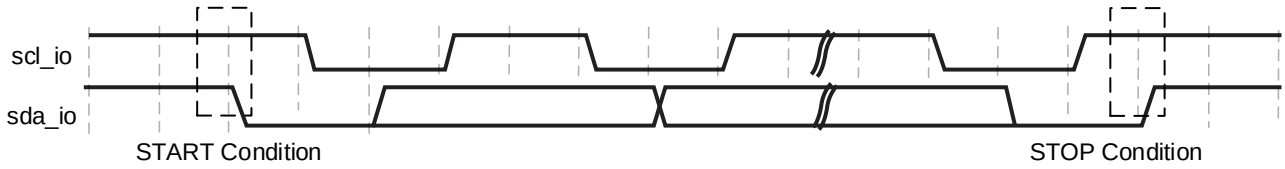


Figure 7.1. START and STOP Conditions

Each data packet on the I²C bus consists of eight bits of data followed by an acknowledge bit (ACK) so one complete data byte transfer requires nine clock pulses. Data is transferred with the most significant bit (MSB) first. The transmitter releases the SDA line during the ACK bit and the receiver of the data transfer must drive the SDA line low during the ACK bit to acknowledge the data receipt. If a Slave-receiver does not drive the SDA line low during the ACK bit, this indicates that the Slave-receiver was unable to accept the data and the Master can then generate a Stop condition to abort the transfer. If the Master-receiver does not generate an ACK, this indicates to the Slave-transmitter that this byte was the last byte of the transfer.

For more information on I²C bus, refer to I²C Bus Specification and User Manual.

7.5.2. Glitch Filter

I²C Slave IP Core has integrated glitch filter to remove 50ns noise/spike as recommended by the I²C Bus Spec for Standard Mode, Fast Mode, and Fast Mode Plus. The glitch filter is applied to both the SCL and SDA signals before they are fed to internal logic. Thus, the I²C signals seen by the IP Core is delayed by a number of clock cycles (~50 ns +1 clock cycle). The filter depth is automatically adjusted based on the *System Clock Frequency* attribute.

7.5.3. Clock Stretching

Clock Stretching allows the I²C slave to pause a transaction by holding the SCL line Low. The transaction cannot continue until the line is released high again. On the byte level, a slave device may be able to receive bytes of data at a fast rate, but needs more time to store a received byte or prepare another byte to be transmitted. Slaves can then hold the SCL line Low after reception and acknowledgment (ACK bit) of a byte to force the master into a wait state until the slave is ready for the next byte transfer.

I²C Slave Core performs clock stretching on the byte level (during ACK/NACK bit) if CONTROL_REG.clk_stretch_en is set to 1. Clock stretching is only performed during data phase. Clock stretching is normally performed when the host need more time before it can address the request of I²C master.

7.5.4. ACK/NACK Response

I²C Slave Core can be configured to send an ACK or a NACK based on settings of CONTROL_REG.nack_data and CONTROL_REG.nack_addr, refer to Control Register (CONTROL_REG) section for details. If the host would like to temporarily disable the access to I²C Slave Core, it should set CONTROL_REG.nack_addr = 1'b1. In this case, I²C Slave Core sends NACK when it is addressed by the external I²C Master.

If the host would like to terminate an on-going I²C write transaction to the I²C Slave Core, it should set CONTROL_REG.nack_data = 1'b1. In this case, I²C Slave Core sends NACK on the next ACK bit for a data byte. Note that the ACK bit is always sent by the receiver, the CONTROL_REG.nack_data has no effect on I²C read transaction.

7.6. Programming Flow

7.6.1. Initialization

To perform initialization, load the appropriate registers of the I²C Slave Controller namely:

- SLAVE_ADDRL_REG, SLAVE_ADDRH_REG – This step is optional. In most cases, initial value set in I²C Slave Addresses attribute of the user interface does not need to be changed.
- CONTROL_REG
- TGT_BYTE_CNT_REG – It is recommended to set this if the size of the data is known. Set this to 8'h00 if the number of bytes to transfer is not known, that is receiving unknown amount of data.
- INT_ENABLE1_REG – It is recommended to enable only the following interrupts when receiving commands from master.
  - Transfer Complete Interrupt – If the size of data is known
  - Receive FIFO Data Interrupt – if the size of data is unknown
- INT_ENABLE2_REG – it is recommended to enable both error interrupts

7.6.2. Data Transfer in response to I²C Master Read

The following are the recommended steps to perform data transfer in response to read request of I²C Master. This assumes that the amount of data to send is known.

To perform data transfer in response to read request of I²C Master:

1. Write data to WR_DATA_REG, amounting to <= FIFO Depth.
2. Enable only Transfer Complete Interrupt. If transmit data is > FIFO Depth, enable also TX FIFO Almost Empty interrupt. If no more data to transfer, otherwise, proceed to step 7.
3. Wait for TX FIFO Almost Empty Interrupt.
   If polling mode is desired, read INT_STATUS1_REG until tx_fifo_aempty_int asserts.
   If interrupt mode is desired, simply wait for interrupt signal to assert, then read INT_STATUS1_REG and check that tx_fifo_aempt_int is asserted.
   Read INT_STATUS2_REG also to check that no error occurred.
4. Clear TX FIFO Almost Empty Interrupt, it also okay to clear all interrupts.
5. Write data byte to WR_DATA_REG, amounting to less than or equal to (FIFO Depth - TX FIFO Almost Empty Setting).

6. If there are remaining data to transfer, go back to Step 3, otherwise, disable TX FIFO Almost Empty Interrupt.
7. Wait for Transfer Complete Interrupt
   If polling mode is desired, read INT_STATUS1_REG until tr_cmp_int asserts.
   If interrupt mode is desired, simply wait for interrupt signal to assert, then read INT_STATUS1_REG and check that tr_cmp_int is asserted.
   Read INT_STATUS2_REG also to check that no error occurred.
8. Clear all interrupts.

7.6.3. Data Transfer in response to I²C Master Write

The following are the recommended steps to perform data transfer in response to write request of I²C Master. This assumes that the amount of data to receive is known.

To perform data transfer in response to write request of I²C Master:

1. Enable only Transfer Complete Interrupt. If data to receive is > FIFO Depth, enable also RX FIFO Almost Full interrupt. If data to receive is <= FIFO Depth, proceed to Step 7.
2. Wait for RX FIFO Almost Full Interrupt.
   If polling mode is desired, read INT_STATUS2_REG until rx_fifo_afull_int asserts.
   If interrupt mode is desired, simply wait for interrupt signal to assert, then read INT_STATUS2_REG and check that rx_fifo_afull_int is asserted.
   Read INT_STATUS2_REG also to check that no error occurred.
3. Clear RX FIFO Almost Full Interrupt, it also okay to clear all interrupts.
4. Read data byte from RD_DATA_REG, amounting to less than or equal to (FIFO Depth - TX FIFO Almost Empty Setting).
5. If there are remaining data to receive, go back to Step 2, otherwise, disable RX FIFO Almost Full Interrupt.
6. Wait for Transfer Complete Interrupt
   If polling mode is desired, read INT_STATUS1_REG until tr_cmp_int asserts.
   If interrupt mode is desired, simply wait for interrupt signal to assert, then read INT_STATUS1_REG and check that tr_cmp_int is asserted.
   Read INT_STATUS2_REG also to check that no error occurred.
7. Clear all interrupts.
8. Read all data from RD_DATA_REG.

7.7. SMBus Slave Support

The I²C Slave Core provides SMBus support by including the smb_alert signal.

7.7.1. SMBus Control and Status Register

Table 7.16. SMBus Register Address Map

Offset	Register Name	Access Type	Description
0x30	SMB_CONTROL_REG	RW	SMBus control and status register

Table 7.17. SMB Control and Status Register

Field	Name	Access	Width	Reset
[7:1]	Reserved	RSVD	7	–
[0]	smb_alert	RW	1	1'b0

- smb_alert

  Transmits the alert interrupt to SMBus Master

  1'b0 – No interrupt to Master

  1'b1 – SMBus slave sent alert interrupt to Master

7.7.2. Operation Details

7.7.2.1. SMBAlert Operation

A Slave device can signal the Master through SMBUSx_INTinterrupt line that it wants to talk. The Master processes the interrupt and simultaneously accesses all the smbalert devices through the Alert Response Address. Only the Slave device which pulled SMBUSx_INTLow acknowledges the Alert Response Address (0001 100b). The host performs a modified Receive Byte operation. The 7-bit device address provided by the Slave transmit device is placed in the seven most significant bits of the byte. The eighth bit can be zero or one.

If more than one device pulls SMBUSx_INTLow, the highest priority device (lowest address) device wins the communication rights.

After receiving an acknowledge (ACK) from the Master in response to its address, the device stops pulling down the SMBUSx_INTsignal. If the Master still sees the SMBUSx_INTLow when the message transfer is complete, the same process repeats again. The SMBus Slave controller monitors the data bus to see if any other Slave is responding to the Alert Response Address. This can be achieved by checking the input and output of SMBUSx_SDA. When there is match, the smb_alert register bit is cleared and the controller generates an interrupt signal to the RISC-V processor.

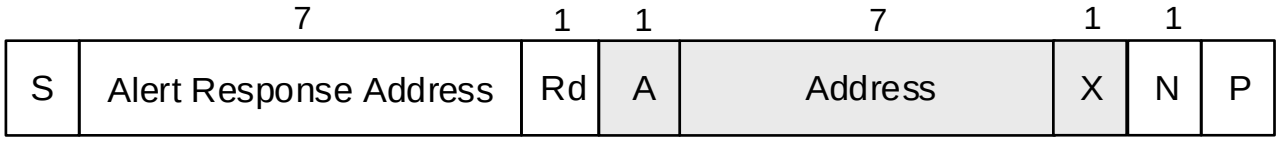


Figure 7.2. SMBus 7-Bit Addressable Device Response

8. eSPI Slave

The Enhanced Serial Peripheral Interface (eSPI) is a synchronous serial interface compatible with SPI. The eSPI Slave includes an ALERT pin to inform the eSPI Master that the eSPI Slave needs to be serviced. The eSPI slave is a configurable Slave which supports multiple channels. The eSPI Slave also allows Slave triggered transactions.

8.1. Features

The key features of the eSPI Slave are:

- Compliant with eSPI base specification as defined in Enhanced Serial Peripheral Interface Specification rev.1.0
- Supports Single, Dual and Quad modes
- Supports Slave triggered transaction
- Supports the following channels:
- Peripheral Channel
- Virtual Wires Channel
- OOB Message (Tunneled SMBus) Channel
- Run-time Flash Access Channel
- Supports CRC Checking

8.2. Block Diagram

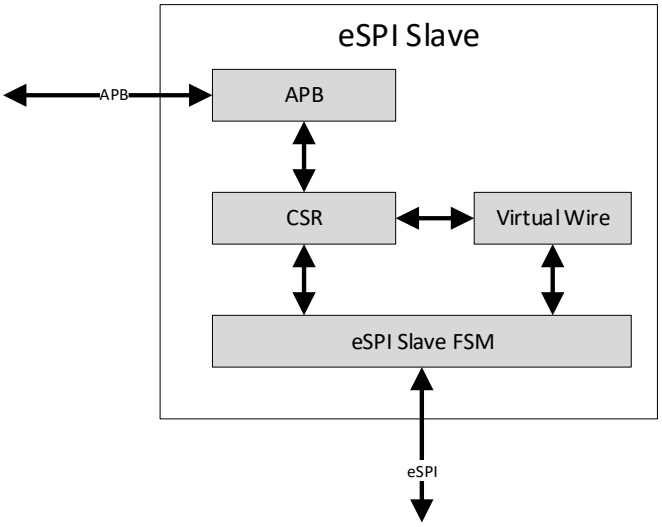


Figure 8.1. eSPI Slave Block Diagram

8.2.1. CSR

CSR contains the configuration and status registers which are written either by the system Master through the APB Interface or by the eSPI Master through the eSPI bus.

8.2.2. Virtual Wire

The virtual wire channel is used to communicate the state of the sideband pins or GPIO tunneled through eSPI as in-band messages.

8.2.3. eSPI Slave FSM

This blocks contains the main FSM of the eSPI Slave and the other channels supported by the Slave. The FIFO for these channels are also contained in this block.

8.3. Signal Description

Table 8.1. eSPI Slave External Signal Description

Port	Width	Direction	Description
eSPI Slave I/O Interface			
espi_clk	1	Input	eSPI Clock input driven by eSPI Master
espi_data	4	Bidir	eSPI serial data
espi_cs	1	Input	eSPI chip select input driven by eSPI Master
espi_alert	1	Output	eSPI Slave alert output

8.4. Channel FIFOs

The eSPI Slave provides RX and TX FIFO's for the different channels. The FIFO sizes are shown in Table 8.2.

Table 8.2. Channel FIFO Size Table

Peripheral Channel FIFO	
Posted completion RX FIFO depth	128 bytes
Posted completion TX FIFO depth	128 bytes
Non posted RX FIFO depth	16 bytes
Non posted TX FIFO depth	16 bytes
OOB Channel FIFO	
OOB RX FIFO depth	256 bytes
OOB TX FIFO depth	256 bytes
Flash Channel FIFO	
Flash non posted RX FIFO depth	256 bytes
Flash completion RX FIFO depth	256 bytes
Flash non posted TX FIFO depth	256 bytes
Flash completion TX FIFO depth	256 bytes

8.5. Register Description

The eSPI slave registers are shown in Table 8.3.

Table 8.3. Summary of eSPI Slave Registers

Offset	Register Name	Access	Reset Value	Description
0x00	ESPI_DEVICE_ID	RO	0x0000_0001	reserved[31:8] version_id[7:0] – Indicates compliance to specific eSPI specification revision. Slaves compliant to this revision of the specification must advertise a value of *01h* in this field. The value for this register is loaded from PARAMETER
0x04	ESPI_GEN_CAP_CONFIG	RW	0x0000_0000	crc_check_en[31] –This bit is set to *1* by eSPI master to enable the CRC checking on the eSPI bus. By default, CRC checking is disabled. resp_modifier_en[30] – This bit is set to *1* to enable the use of Response Modifier by eSPI slave to append either a peripheral (channel 0) completion, a virtual wire (channel 1) packet or a flash access (channel 3) completion to the GET_STATUS response phase. When this bit is a *0*, eSPI slave must only use the Response Modifier of *00*, which is no append. By default, the Response Modifier is disabled. reserved[29]

Offset	Register Name	Access	Reset Value	Description
				alert_mode[28] – Configures the Alert mechanism used by the slave to initiate a transaction on the eSPI interface. • 0b: I/O[1] pin is used to signal the Alert event • 1b: Alert# pin is used to signal the Alert event io_mode_sel[27:26] – eSPI master programs this field to enable the appropriate mode of operation, which takes effect at the deassertion edge of the Chip Select. The I/O Mode configured in this field must be supported by both the master and the slave. Single I/O mode is supported by default. • 2'b00 Single I/O • 2'b01 Dual I/O • 2'b10 Quad IO • 2'b11 Reserved cfg_io_mode[25:24] – Indicates the I/O modes supported by the slave. • 2'b00 Single I/O • 2'b01 Single and Dual I/O • 2'b10 Single and Quad I/O • 2'b11 Single, Dual and Quad open_drain_alert_sel[23] – Set to *1* by eSPI master to configure the Alert pin as an open drain output. By default, Alert pin operates as a driven output. This bit must only be programmed to *1* if open drain Alert pin is supported by the slave. The bit must be valid when Alert Mode bit is a *1* indicating Alert pin is used for signaling the Alert event. operating_freq[22:20] – Identifies the frequency of operation. • 3'b000 20 MHz • 3'b001 25 MHz • 3'b010 33 MHz • 3'b011 50MHz • 3'b100 66MHz • Others Reserved cfg_open_drain_alert[19] – Indicates the support of the Alert# pin as an open drain output by the slave. cfg_max_freq[18:16] – Identifies the maximum frequency of operation supported by the slave. • 3'b000 20 MHz • 3'b001 25 MHz • 3'b010 33 MHz • 3'b011 50 MHz • 3'b100 66 MHz • Others Reserved max_wait_state[15:12] – eSPI master sets the maximum WAIT STATE allowed before the slave must respond with an ACCEPT, DEFER, NON FATAL ERROR or FATAL ERROR response code. This is a 1-based field in the granularity of byte time. When *0*, it indicates a value of 16-byte time. A byte time corresponds to eight serial clocks in the Single I/O mode, four serial clocks in the Dual I/O mode or two serial clocks in the Quad I/O mode. reserved[11:8] cfg_channel[7:0] – Indicates that the corresponding

Offset	Register Name	Access	Reset Value	Description
				channel is supported by the slave. • Bits 0 Peripheral Channel • Bits 1 Virtual Wire Channel • Bits 2 OOB Message Channel • Bits 3 Flash Access Channel • - Bits 7:4 Reserved for platform specific channels
0x08	ESPI_CH0_CAP_CONFIG	RW	0x0000_1101	reserved[31:15] peri_ch_max_read_req_size[14:12] – Maximum read request size for the Peripheral channel. The length of the read request must not cross the naturally aligned address boundary of the corresponding Maximum Read Request Size. • 000b Reserved • 001b 64 bytes address aligned • 010b 128 bytes address aligned • 011b 256 bytes address aligned • 100b 512 bytes address aligned • 101b 1024 bytes address aligned • 110b 2048 bytes address aligned • 111b 4096 bytes address aligned reserved[11] peri_ch_max_payload_size_sel[10:8] – Maximum payload size for the Peripheral channel. The value set by the eSPI master must never be more than the value advertised in the Max Payload Size Supported field. The payload of the transaction must not cross the naturally aligned address boundary of the corresponding Maximum Payload Size. • 000b: Reserved • 001b: 64 bytes address aligned max payload size • 010b: 128 bytes address aligned max payload size • 011b: 256 bytes address aligned max payload size • 100b 111b: Reserved reserved[7] cfg_peri_max_payload[6:4] – This field advertises the Maximum Payload Size supported by the slave. • 000b Reserved • 001b 64 bytes address aligned max payload size • 010b 128 bytes address aligned max payload size • 011b 256 bytes address aligned max payload size • 100b 111b : Reserved reserved[3] peri_ch_bus_master_en[2] – When this bit is a *0*, it disables the slave from generating bus mastering cycles on the Peripheral channel. When this bit is a *1*, it allows the slave to generate bus mastering cycles on the Peripheral channel. Prior to clearing the Bus Master Enable bit from *1* to *0*, there must be no outstanding non posted cycle pending completion from the slave. cfg_peri_channel_ready[1] – When this bit is a *1*, it indicates that the slave is ready to accept transactions on the Peripheral

Offset	Register Name	Access	Reset Value	Description
				channel. eSPI master should poll this bit after the channel is enabled before running any transaction on this channel to the slave. peri_ch_en[0] – The channel is by default enabled after the eSPI Reset. This bit is cleared to *0* by eSPI master to disable the Peripheral channel. Besides, clearing this bit from *1* to *0* triggers a reset to the Peripheral channel. The channel remains disabled until this bit is set to *1* again. Prior to disabling the Peripheral channel, the Bus Master Enable bit should be cleared to *0* to disable the bus mastering cycles.
0x0C	ESPI_CH1_CAP_CONFIG	RW	0x0000_0000	reserved[31:22] vw_max_count[21:16] – The maximum number of Virtual Wire groups that can be sent in a single Virtual Wire packet. This is a 0 based count. The default value of 0 indicates count of 1. The value configured in this field must never be more than the value advertised in the Maximum Virtual Wire Count Supported field. reserved[15:14] cfg_max_vw_count[13:8] – Advertises the Maximum Virtual Wire Count supported by the slave. If the slave supports different count value as initiator and as receiver of the Virtual Wires, this field indicates the lower of the two. The Virtual Wire Count specifies the maximum number of Virtual Wire groups being communicated in a single Virtual Wire packet. eSPI slave must advertise a value of *000111b* or more in this field to indicate the support of at least 8 Virtual Wire groups being communicated in a single Virtual Wire packet. This is a 0-based count. reserved[7:2] cfg_vw_channel_ready[1] – When this bit is a *1*, it indicates that the slave is ready to accept transactions on the Virtual Wire channel. eSPI master should poll this bit after the channel is enabled before running any transaction on this channel to the slave. vw_ch_en[0] – This bit is set to *1* by eSPI master to enable the Virtual Wire channel. Clearing this bit from *1* to *0* do not reset the Virtual Wire channel whereby the state of all the Virtual Wires must continue to be maintained internally. When this bit is *0*, no transaction shall occur on the Virtual Wire channel. The channel is by default disabled after the eSPI Reset.
0x10	ESPI_CH2_CAP_CONFIG	RW	0x0000_0100	reserved[31:11] oob_msg_ch_max_payload_size_sel[10:8] – eSPI master sets the maximum payload size for the OOB Message channel. The value set by the eSPI master must never be more than the value advertised in the Max Payload Size Supported field. The Maximum Payload Size applies to the actual payload of the protocol embedded in the OOB packet. • 000b – Reserved • 001b – 64 bytes max payload size • 010b – 128 bytes max payload size

Offset	Register Name	Access	Reset Value	Description
				• 011b – 256 bytes max payload size • 100b - 111b – Reserved reserved[7] cfg_oob_max_payload[6:4] – Advertises the Maximum Payload Size supported by the slave. The Maximum Payload Size applies to the actual payload of the protocol embedded in the OOB packet. • 000b – Reserved • 001b – 64 bytes max payload size • 010b – 128 bytes max payload size • 011b – 256 bytes max payload size • 100b - 111b – Reserved reserved[3:2] cfg_oob_channel_ready[1] –When this bit is a *1*, it indicates that the slave is ready to accept transactions on the OOB Message channel. eSPI master should poll this bit after the channel is enabled before running any transaction on this channel to the slave. oob_msg_ch_en[0] – This bit is set to *1* by eSPI master to enable the OOB Message channel. Clearing this bit from *1* to *0* triggers a reset to the OOB Message channel such as during error handling. The channel remains disabled until this bit is set to *1* again. The channel is by default disabled after the eSPI Reset.
0x14	ESPI_CH3_CAP_CONFIG	RW	0x0000_1104	reserved[31:15] flash_ch_max_read_req_size[14:12] – eSPI master sets the maximum read request size for the Flash Access channel. The length of the read request must not exceed the corresponding Maximum Read Request Size with no address alignment requirement. • 000b: Reserved. • 001b: 64 bytes max read request size • 010b: 128 bytes max read request size • 011b: 256 bytes max read request size • 100b: 512 bytes max read request size • 101b: 1024 bytes max rea d request size • 110b: 2048 bytes max read request size • 111b: 4096 bytes max read request size cfg_flash_sharing_mode[11] – When Flash Access channel is supported, this bit advertises the flash sharing scheme intended by the slave. • 0b: Master attached flash sharing • 1b: Slave attached flash sharing This bit is a Read Only ‘0’ in the base specification as only master attached flash sharing is defined. flash_ch_max_payload_size_sel[10:8] – eSPI master sets the maximum payload size for the Flash Access channel. The value set by the eSPI master must never be more than the value advertised in the Max Payload Size Supported field. • 000b: Reserved. • 001b: 64 bytes max payload size • 010b: 128 bytes max payload size

Offset	Register Name	Access	Reset Value	Description
				• 011b: 256 bytes max payload size • 100b 111b: Reserved cfg_flash_max_payload[7:5] – Advertises the Maximum Payload Size supported by the slave. • 000b: Reserved • 001b: 64 bytes max payload size • 010b: 128 bytes max payload size • 011b: 256 bytes max payload size • 100b 111b: Reserved flash_ch_block_erase_size[4:2] – eSPI master sets this field to communicate the block erase size to the slave. This field is applicable only to master attached flash sharing scheme. • 000b: Reserved • 001b: 4 Kbytes • 010b: 64 Kbytes • 011b: Both 4 Kbytes and 64 Kbytes are supported • 100b: 128 Kbytes • 101b: 256 Kbytes • 110b 111b: Reserved cfg_flash_channel_ready[1] – When this bit is a *1*, it indicates that the slave is ready to accept transactions on the Flash Access channel. eSPI master should poll this bit after the channel is enabled before running any transaction on this channel to the slave. flash_ch_en[0] – This bit is set to *1* by eSPI master to enable the Flash Access channel. Clearing this bit from *1* to *0* triggers a reset to the Flash Access channel such as during error handling. The channel remains disabled until this bit is set to *1* again. The channel is by default, disabled after the eSPI Reset.
0x18	ESPI_CH3_CAP_CONFIG_2	RO	0x0000_0000	reserved[31:22] cfg_flash_trgt_rpmc[21:16] – Indicates the total number of Replay Protected Monotonic Counters (RPMC) supported by the Slave. It is a 1-based field. • 0h: Slave does not support RPMC • 1h: Slave supports up to 1 RPMC • 2h: Slave supports up to 2 RPMC … • 3Fh: Slave supports up to 63 RPMC cfg_flash_trgt_blk_erase_size[15:8] – Indicates the sizes of the erase commands the master may issue. If multiple bits are set then the master may issue an erase using any of the indicated sizes. If multiple regions are accessible by the master, this field advertises the common erase block sizes for these regions. This field is only applicable when slave attached flash sharing scheme is selected. • Bit 0: Reserved • Bit 1: Reserved • Bit 2: 4 Kbytes EBS supported • Bit 3: Reserved • Bit 4: Reserved • Bit 5: 32 Kbytes EBS supported • Bit 6: 64 Kbytes EBS supported • Bit 7: 128 Kbytes EBS supported

Offset	Register Name	Access	Reset Value	Description
				reserved[7:3] cfg_flash_trgt_max_rd_size[2:0] – Indicates the maximum read request size supported by the slave as the Target on the Flash Access channel. This field is only applicable when slave attached flash sharing scheme is selected. • 000b, 001b: 64 bytes max read request size. • 010b: 128 bytes max read request size • 011b: 256 bytes max read request size • 100b: 512 bytes max read request size • 101b: 1024 bytes max read request size • 110b: 2048 bytes max read request size • 111b: 4096 bytes max read request size
0x1C	PUT_PC_RX_FIFO_DATA	RO	0x0000_0000	reserved[31:8] put_pc_rx_fifo_data[7:0] – FIFO to store incoming Peripheral PC receive packets
0x20	GET_PC_TX_FIFO_DATA	WO	0x0000_0000	reserved[31:8] get_pc_tx_fifo_data[7:0] – FIFO to store outgoing Peripheral PC transmit packets
0x24	PUT_NP_RX_FIFO_DATA	RO	0x0000_0000	reserved[31:8] put_np_rx_fifo_data[7:0] – FIFO to store incoming Peripheral NP receive packets
0x28	GET_NP_TX_FIFO_DATA	WO	0x0000_0000	reserved[31:8] get_np_tx_fifo_data[7:0] – FIFO to s tore outgoing Peripheral NP transmit packets
0x2C	PUT_OOB_RX_FIFO_DATA	RO	0x0000_0000	reserved[31:8] put_oob_rx_fifo_data[7:0] – FIFO to store incoming OOB receive packets
0x30	GET_OOB_TX_FIFO_DATA	WO	0x0000_0000	reserved[31:8] get_oob_tx_fifo_data[7:0] – FIFO to store outgoing OOB transmit packets
0x34	GET_FLASH_NP_TX_FIFO	WO	0x0000_0000	reserved[31:8] get_flash_np_tx_fifo[7:0] – FIFO to store TX flash NP packets
0x38	PUT_FLASH_C_RX_FIFO	RO	0x0000_0000	reserved[31:8] put_flash_c_rx_fifo[7:0] – FIFO to store RX flash Completion packets
0x3C	PUT_FLASH_NP_RX_FIFO	RO	0x0000_0000	reserved[31:8] put_flash_np_rx_fifo[7:0] – FIFO to store RX flash NP packets
0x40	GET_FLASH_C_TX_FIFO	WO	0x0000_0000	reserved[31:8] get_flash_c_tx_fifo[7:0] – FIFO to store T X flash Completion packets
0x44	IRQ_ENABLE1	RW	0x0000_0000	This register controls the masking of interrupt. When the particular bit in this register is *0,* then the corresponding Interrupt in the IRQ_STATUS1 register is masked. See Table 8.4 for details on the fields.
0x48	IRQ_STATUS1	RW1C	0x0000_0000	See Table 8.4 for details on the fields.
0x4C	IRQ_ENABLE2	RW	0x0000_0000	This register controls the masking of interrupt. When the particular bit in this register is *0*, then the corresponding interrupt in the IRQ_STATUS2 register is masked. See Table 8.5 for details on the fields.
0x50	IRQ_STATUS2	RW1C	0x0000_0000	See Table 8.5 for details on the fields.

Offset	Register Name	Access	Reset Value	Description
0x54	TX_CHN_AVAIL_REQ	WO	0x0000_0000	Writing this register triggers the IP to generate the alert request to get the attention of Master. These bits are cleared by the IP on the reception of corresponding GET_* commands. reserved[31:5] flash_np_tx_avail_req[4] – GET FLASH NP Avail request flash_c_tx_avail_req[3] – GET FLASH C Avail request oob_tx_avail_req[2] – GET OOB Avail request np_tx_avail_req[1] – GET NP Avail request pc_tx_avail_req[0] – GET PC Avail request
0x58	ESPI_QUEUE_STATUS	RO	0x0000_0000	reserved[31:11] fl_np_avail[10] – Available Alert for Flash NP commands fl_c_avail[9] – Available Alert for Flash NP commands fl_np_free[8] – Free Alert for Flash NP commands fl_c_free[7] – Free Alert for Flash Completion commands oob_avail[6] – Available Alert for OOB commands vw_avail[5] – Available Alert for Virtual wire commands np_avail[4] – Available Alert for Peripheral NP commands pc_avail[3] – Available Alert Peripheral PC command oob_free[2] – Free Alert for OOB commands np_free[1] – Free Alert for Peripheral NP command pc_free[0] – Free Alert for Peripheral PC commands
0x5C	FIFO_STATUS	RO	0x0000_0000	reserved[31:20] get_flash_c_tx_fifo_empty[19] – GET Flash C TX FIFO Empty indication put_flash_np_rx_fifo_empty[18] – PUT Flash NP RX FIFO Empty indication put_flash_c_rx_fifo_empty[17] – PUT Flash C RX FIFO Empty indication get_flash_np_tx_fifo_empty[16] – GET Flash NP TX FIFO Empty indication get_oob_tx_fifo_empty[15] – OOB TX FIFO Empty indication put_oob_rx_fifo_empty[14] – OOB RX FIFO Empty indication get_np_tx_fifo_empty[13] – Peripheral NP RX FIFO Empty indication put_np_rx_fifo_empty[12] – Peripheral NP RX FIFO Empty indication get_pc_tx_fifo_empty[11] – Peripheral PC TX FIFO Empty indication put_pc_rx_fifo_empty[10] – Peripheral PC RX FIFO Empty indication get_flash_c_tx_fifo_full[9] – Flash C TX FIFO Full indication put_flash_np_rx_fifo_full[8] – PUT Flash NP RX FIFO Full indication put_flash_c_rx_fifo_full[7] – PUT Flash C RX FIFO Full indication get_flash_np_tx_fifo_full[6] – Flash NP TX FIFO Full indication get_oob_tx_fifo_full[5] – OOB TX FIFO Full indication put_oob_rx_fifo_full[4] – OOB RX FIFO Full indication get_np_tx_fifo_full[3] – Peripheral PC TX FIFO Full indication put_np_rx_fifo_full[2] – Peripheral NP RX FIFO Full

Offset	Register Name	Access	Reset Value	Description
				indication get_pc_tx_fifo_full[1] – Peripheral PC TX FIFO Full indication put_pc_rx_fifo_full[0] – Peripheral PC RX FIFO Full indication
0x60	FIFO_FLUSH	WO	0x0000_0000	reserved[31:10] get_flash_c_tx_fifo_flush[9] – Flush the GET_FLASH_C_TX_FIFO put_flash_np_rx_fifo_flush[8] – Flush the PUT_FLASH_NP_RX_FIFO put_flash_c_rx_fifo_flush[7] – Flush the PUT_FLASH_C_RX_FIFO get_flash_np_tx_fifo_flush[6] – Flush the GET_FLASH_NP_TX_FIFO get_oob_tx_fifo_flush[5] – Flush the GET_OOB_TX_FIFO put_oob_rx_fifo_flush[4] – Flush the PUT_OOB_RX_FIFO get_np_tx_fifo_flush[3] – Flush the GET NP TX FIFO put_np_rx_fifo_flush[2] – Flush the PUT NP RX FIFO get_pc_tx_fifo_flush[1] – Flush the GET PC TX FIFO put_pc_rx_fifo_flush[0] – Flush the PUT PC RX FIFO
0x64	ERROR_STATUS	RW1C	0x0000_0000	reserved[31:9] page_err[8] – Register to indicate Address Page Error mal_vw_channel[7] – Register to indicate Malformed Error detected in VW channel mal_peri_channel[6] – Register to indicate Malformed Error detected in Peripheral channel get_wout_avail[5] – Register to indicate GET Without Available Error is detected put_wout_free[4] – Register to indicate PUT Without Error is detected unexpec_cs_deassert[3] – Register to indicate Unexpected CS Deassert Error is detected invalid_crc[2] – Register to indicate Invalid CRC Error is detected invalid_ctype[1] – Register to indicate Invalid Cycle type Error is detected invalid_cmd[0] – Register to indicate Invalid Command Error is detected
0x68	VW_IDX0_TX_DATA	RW	0x0000_0000	vw_idx0_tx_data[31:0] – Register for VW_IDX0_TX_DATA
0x6C	VW_IDX1_TX_DATA	RW	0x0000_0000	vw_idx1_tx_data[31:0] – Register for VW_IDX1_TX_DATA
0x70	VW_IDX4_TX_DATA	RW	0x0000_0000	vw_idx4_tx_data[31:0] – Register for VW_IDX4_TX_DATA
0x74	VW_IDX5_TX_DATA	RW	0x0000_0000	vw_idx5_tx_data[31:0] – Register for VW_IDX5_TX_DATA
0x78	VW_IDX6_TX_DATA	RW	0x0000_0000	vw_idx6_tx_data[31:0] – Register for VW_IDX6_TX_DATA
0x7C	VW_IDX2_RX_DATA	RO	0x0000_0000	vw_idx2_rx_data[31:0] – Register for VW_IDX2_RX_DATA
0x80	VW_IDX3_RX_DATA	RO	0x0000_0000	vw_idx3_rx_data[31:0] – Register for VW_IDX3_RX_DATA
0x84	VW_IDX7_RX_DATA	RO	0x0000_0000	vw_idx7_rx_data[31:0] – Register for VW_IDX7_RX_DATA
0x88	VW_IDX64_RX_DATA	RO	0x0000_0000	vw_idx64_rx_data[31:0] – Register for VW_IDX64_RX_DATA

Table 8.4. Interrupt Bit Fields for IRQ_ENABLE1 and IRQ_STATUS1

Bit Index	Field Name	Description
31	IRQ_GET_FLASH_C_CMD_DONE	Enable register to indicate get flash np command done
30	IRQ_GET_FLASH_NP_CMD_DONE	Enable register to indicate FLASH NP Command done
29	IRQ_FLASH_C_RX_FIFO_URUN	Enable register to indicate FLASH C RX FIFO Underrun detected
28	IRQ_FLASH_C_RX_FIFO_ORUN	Enable register to indicate FLASH C RX FIFO Overrun detected
27	IRQ_PUT_FLASH_C_CMD_DONE	Enable register to indicate PUT Flash C Command done
26	IRQ_FLASH_NP_RX_FIFO_URUN	Enable register to indicate FLASH NP RX FIFO Underrun detected
25	IRQ_FLASH_NP_RX_FIFO_ORUN	Enable register to indicate FLASH NP RX FIFO Overrun detected
24	IRQ_PUT_FLASH_NP_CMD_DONE	Enable register to indicate PUT Flash NP command done
23	IRQ_GET_VW_CMD_DONE	Enable register to indicate get vw command done
22	IRQ_PUT_VW_CMD	Enable register to indicate put vw command done
21	IRQ_OOB_TX_FIFO_URUN	Enable register to indicate OOB TX FIFO Underrun detected
20	IRQ_OOB_TX_FIFO_ORUN	Enable register to indicate OOB TX FIFO Overrun detected
19	IRQ_GET_OOB_CMD_DONE	Enable register to indicate GET oob command done
18	IRQ_OOB_RX_FIFO_URUN	Enable register to indicate OOB RX FIFO Underrun detected
17	IRQ_OOB_RX_FIFO_ORUN	Enable register to indicate OOB RX FIFO Overrun detected
16	IRQ_PUT_OOB_CMD_DONE	Enable register to indicate Bit for indicating put oob command done
15	IRQ_NP_TX_FIFO_URUN	Enable register to indicate NP TX FIFO Underrun condition detected
14	IRQ_NP_TX_FIFO_ORUN	Enable register to indicate NP TX FIFO Overrun condition detected
13	IRQ_GET_NP_CMD	Enable register to indicate GET NP Command is received
12	IRQ_PC_TX_FIFO_URUN	Enable register to indicate PC TX FIFO Underrun condition detected
11	IRQ_PC_TX_FIFO_ORUN	Enable register to indicate PC TX FIFO Overrun condition detected
10	IRQ_GET_PC_CMD	Enable register to indicate GET PC Command is received
9	IRQ_NP_RX_FIFO_URUN	Enable register to indicate NP RX FIFO Underrun detected
8	IRQ_NP_RX_FIFO_ORUN	Enable register to indicate NP RX FIFO Overrun detected
7	IRQ_PUT_MEMRD32_SHORT	Enable register to indicate PUT MEMRD32 RD Short command received
6	IRQ_PUT_IORD_SHORT	Enable register to indicate PUT IO RD Short command received
5	IRQ_PUT_IOWR_SHORT	Enable register to indicate PUT IO WR Short command received
4	IRQ_PUT_NP_CMD	Enable register to indicate PUT NP Command is received
3	IRQ_SHORT_MEMWR	Enable register to indicate SHORT IO Command is received
2	IRQ_PC_RX_FIFO_URUN	Enable register to indicate PC RX FIFO Underrun interrupt
1	IRQ_PC_RX_FIFO_ORUN	Enable register to indicate PC RX FIFO Overrun interrupt
0	IRQ_PUT_PC_CMD	Enable register to indicate PUT PC Command is received

Table 8.5. Interrupt Bit Fields for IRQ_ENABLE2 and IRQ_STATUS2

Bit Index	Field Name	Description
31:5	RESERVED	Reserved fields
4	IRQ_ESPI_ERR_DET	Enable register to indicate IRQ eSPI Error detection
3	IRQ_FLASH_C_TX_FIFO_URUN	Enable register to indicate FLASH C TX FIFO Underrun detected
2	IRQ_FLASH_C_TX_FIFO_ORUN	Enable register to indicate FLASH C TX FIFO Overrun detected
1	IRQ_FLASH_NP_TX_FIFO_URUN	Enable register to indicate FLASH NP TX FIFO Underrun detected
0	IRQ_FLASH_NP_TX_FIFO_ORUN	Enable register to indicate FLASH NP TX FIFO Overrun detected

9. GPIO

The GPIO provides a dedicated interface to configure each GPIO as either an input or an output. When configured as an input, it can detect the state of a GPIO by reading the state of the associated register. When configured as an output, it takes the value written into the associated register and controls the state of the controlled GPIO. The SoC Function Block provides two types of GPIO, see Table 9.1. The Memory Mapped GPIO are registered based and controlled by the CPU. The Virtual GPIO are controlled by the PLD logic, see Table 9.2.

The GPIO core consists of registers for reading and writing the GPIO channel. It also includes the necessary logic to identify an interrupt event, when the port input changes.

Table 9.1. External GPIO Signal Descriptions

Signal	Direction	Description
Memory Mapped GPIO		
GPIO_MMxx	Bidir	16 General Purpose Memory Mapped I/O
Virtual GPIO		
GPIO_xx	Bidir	24 General Purpose I/O controlled from the PLD

Table 9.2. PLD Interface Signal Descriptions

Signal	Direction	Description
gpio_input[23:0]	Output	Read data from GPIO
gpio_output[23:0]	Input	Write data to GPIO
gpio_direction[23:0]	Input	Set direction of the Virtual GPIO as input (0) or output (1)
ready_gpio	Output	When high, the Virtual I/O are ready to read gpio_direction and gpio_output inputs. When a change has occurred on the gpio_direction and gpio_output inputs, the ready_gpio signal goes low until the changes have been updated and returns to high when the changes are complete.
reset_n_gpio	Input	Active low reset When asserted, the reset_n_gpio signal places the Virtual GPIO in reset condition (tri-stated inputs) and deasserts ready_gpio.

9.1. GPIO Features

The GPIO IP features are:

- Setting or clearing an output through a single register to allow parallel control of the outputs
- Setting or clearing an output by writing Set Data and Clear Data registers
- Output register reflects the output driven status
- Input register reflects the input status
- All inputs may be configured as an interrupt source with configurable edge or level detection

9.2. Register Description

Table 9.3 shows the summary of GPIO registers.

Table 9.3. Register Address Map

Offset	Register Name	Access Type	Description
0x00	RD_DATA_REG	R	Read Data Register
0x04	WR_DATA_REG	R/W	Write Data Register
0x08	SET_DATA_REG	W	Set Data Register
0x0C	CLEAR_DATA_REG	W	Clear Data Register
0x10	DIRECTION_REG	R/W	Direction Control Register
0x14	INT_TYPE_REG	R/W	Interrupt Type Configure Register

Offset	Register Name	Access Type	Description
0x18	INT_METHOD_REG	R/W	Interrupt Method Configure Register
0x1C	INT_STATUS_REG	R/W	Interrupt Status Register
0x20	IN_ENABLE_REG	R/W	Interrupt Enable Register
0x24	INT_SET_REG	W	Interrupt Set Register

9.2.1. Read Data Register (RD_DATA_REG)

Reading the Read Data Register returns the data from the input pins, see Table 9.4. Reset value is not observable because value is updated immediately after reset.

Table 9.4. Read Data Register

Name	Access	Width	Reset
rd_data	R	16	NA

9.2.2. Write Data Register (WR_DATA_REG)

Writing in the Write Data Register changes the data of the output pins, see Table 9.5.

Table 9.5. Write Data Register

Name	Access	Width	Reset
wr_data	R/W	*16*	*0*

9.2.3. Set Data Register (SET_DATA_REG)

If any bit of the Set Data Register is set to *1*, the corresponding bit of wr_data gets set to *1*, see Table 9.6.

Table 9.6. Set Data Register

Name	Access	Width	Reset
set_data	W	*16*	0

9.2.4. Clear Data Register (CLEAR_DATA_REG)

If any bit of the Clear Data Register is set to *1*, the corresponding bit of wr_data gets cleared set to *0*, see Table 9.7.

Table 9.7. Clear Data Register

Name	Access	Width	Reset
clear_data	W	*16*	0

9.2.5. Direction Register (DIRECTION_REG)

The Direction Register determines the direction of pins. If any bit of this register is set to *0*, the corresponding pin is configured as an input, otherwise as an output, see Table 9.8.

Table 9.8. Direction Register

Name	Access	Width	Reset
direction_reg	R/W	*16*	0

9.2.6. Interrupt Type Register (INT_TYPE_REG)

The Interrupt Type Registers sets the type as edge (*0*) or level (*1*), see Table 9.9.

Table 9.9. Interrupt Type Register

Name	Access	Width	Reset
int_type	R/W	*16*	0

9.2.7. Interrupt Method Register (INT_METHOD_REG)

The Interrupt Method Registers set the mode as rising (*1*) or falling (*0*) in for edge type interrupt or high (*1*) or low (*0*) for level type interrupt, see Table 9.10.

Table 9.10. Interrupt Method Register

Name	Access	Width	Reset
int_method	R/W	*16*	0

9.2.8. Interrupt Status Register (INT_STATUS_REG)

The Interrupt Status Register (see Table 9.11) shows the interrupt status for each input, regardless of whether it is enabled or not. If any bit of this register is set to *1* and the corresponding bit of INT_ENABLE_REG is set as well, interrupt happens on the corresponding input. In order to clear interrupt, you must write *1* to the corresponding bit.

Table 9.11. Interrupt Status Register

Name	Access	Width	Reset
int_status	R/W	*16*	0

9.2.9. Interrupt Enable Register (INT_ENABLE_REG)

In the Interrupt Enable Register (see Table 9.12), each bit that is set to *1* enables interrupt for the corresponding port when it is configured as an input.

Table 9.12. Interrupt Enable Register

Name	Access	Width	Reset
int_enable	R/W	*16*	0

9.2.10. Interrupt Set Register (INT_SET_REG)

In the Interrupt Set Register (see Table 9.13), you can generate interrupt by writing *1* to the corresponding bit of this register. This also sets the corresponding bit of the int_status register to *1*.

Table 9.13. Interrupt Set Register

Name	Access	Width	Reset
int_set	W	*16*	0

9.3. Programming Flow

9.3.1. Initialization

Initial values for all registers come from the user interface. To change default configuration, the following GPIO registers should be set properly before performing Read or Write operation:

- Direction Register
- Interrupt Type Register
- Interrupt Method Register
- Interrupt Enable Register

In case any of the interrupts are enabled, these must first be cleared by writing *1*s to the corresponding bits of the Interrupt Status Register.

9.3.2. Data Transfer (Transmit/Receive Operation)

Assuming that the module is not currently performing any operation, below are recommended steps for performing a GPIO transaction.

- To read from inputs, read the Read Data Register.
- To write to outputs, write to the Write Data Register.

If an interrupt occurs and you want to clear that interrupt, write *1*s to corresponding bits of the Interrupt Status Register.

10. Secure Enclave

For information on the Secure Enclave, please contact your local sales representative.

References

For more information, refer to the following documents:

- Lattice Propel 1.0 User Guide
- Lattice Diamond Software 3.11 User Guide

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Revision 1.0, October 2021

Section	Change Summary
All	Production release.

Revision 0.82, June 2021

Section	Change Summary
SoC Function Block Memory Map	In Table 1.1, changed PFR Base Address to 00080800 for RESERVED Block, and PFR End Address to 000807FF for CPU PIC TIMER Block.
CPU Subsystem	In Table 2.3, changed the last Offset to 0x414, TIMER_CMP_H.

QSPI Monitor	• In the overview description, changed the QSPI monitor definition to an SPI access and command monitoring module.
	• In Table 4.4:
	  • changed Register Name to SPACE2_END_ADDR for 0xN68 Offset;
	  • changed Register Name to SPACE3_FILTER_CTRL for 0xN80 Offset.

QSPI Master Streamer	Modified the description regarding initiating the SPI transactions In the Transaction Phases section.
I^2C Monitor	Changed the I^2C monitor definition to an I^2C access and command module in the introduction.

Revision 0.81, February 2021

Section	Change Summary
All	• Changed document title to *Mach-NX SFB Hardware Usage Guide*. • Updated document to change all SoC reference to *SoC Function Block*.
Acronyms in This Document	Updated content.

Revision 0.80, December 2020

Section	Change Summary
All	Preliminary release.

www.latticesemi.com