Lattice Sentry 2.2 Mach-NX SoC Function Block Hardware User Guide

Technical Note

FPGA-TN-02360-1.0

April 2024

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 all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. 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. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. 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.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
AES	Advanced Encryption Standard
AHB	Advanced High-Performance Bus
APB	Advanced Peripheral Bus
BMC	Baseboard Management Controller
CPU	Central Processing Unit
CSR	Control and Status Registers
DSPI	Dual Serial Peripheral Interface
EAR	Extended Address Register
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
FIFO	First In, First Out
GPIO	General Purpose Input/Output
HMAC	Hash Message Authentication Code
HSP	High-Speed Data Port
I^2C	Inter-Integrated Circuit
IRQ	Interrupt Request
OOB	Out-of-Band
PCH	Platform Controller Hub
PFR	Platform Firmware Resiliency
PKC	Public Key Cryptography
PLD	Programmable Logic Device
QSPI	Quad Serial Peripheral Interface
RISC-V	Reduced Instruction Set Computer-V (five)
RoT	Root of Trust
Rx	Receiver
SCI	System Configuration Interface
SFB	SoC Function Block
SFDP	Serial Flash Discoverable Parameter
SHA	Secure Hash Algorithm
SMBus	System Management Bus
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
TRNG	True Random Number Generator
Tx	Transmitter

1. System on Chip (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 IPs in the Mach-NX SoC Function Block.

1.1. Register Access Definitions

Access Type	Behavior on Read Access	Behavior on Write Access
RO – Read-Only Register	Returns register value.	Ignores write access.
WO – Write-Only Register	Returns 0.	Updates register value.
RW – Read Write Register	Returns register value.	Updates register value.
RW1C – Read Write 1 to Clear Register	Returns register value.	Writing 1’b1 on register bit clears the bit to 1’b0. Writing 1’b0 on register bit is ignored.
RC – Read Clear Register	Clears register value to 0.	Ignores write access.
RSVD – Reserved Register	Returns 0.	Ignores write access.

1.2. 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 reset, 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 or unauthorized.
- Event Logging – The Mach-NX device logs security events, such as unauthorized flash accesses and notifies the BMC.
- I^2C/SMBus Filter – The Mach-NX device can monitor an I^2C bus for unauthorized activity and block unauthorized write transactions by disabling the I^2C bus. It acts as pass-through relay from the point of view of both SMBus Controller and SMBus Target devices on the bus. It directly attaches to the controller port and protects all target devices against malicious traffic generated from the controller port based on the list of allowable and non-allowable write commands. All read transactions are allowed.

1.3. SoC Function Block Diagram

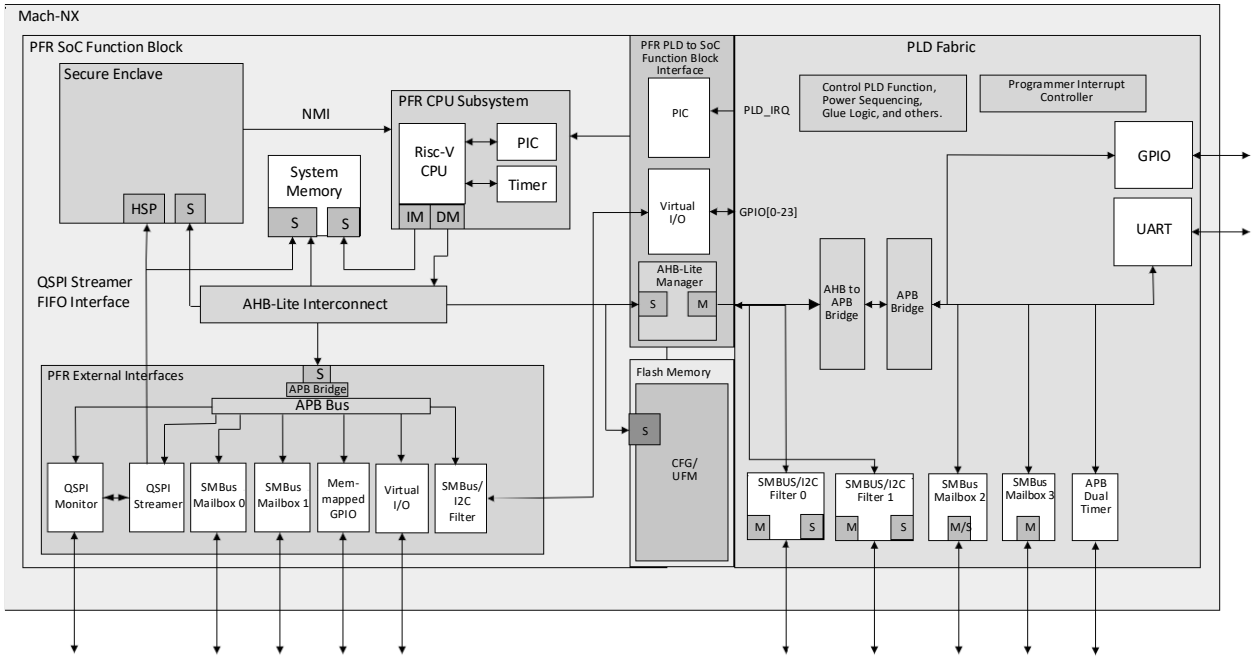


Figure 1.1. Mach-NX SoC Function Block Diagram

1.4. SoC Function Block Memory Map

Table 1.1. SoC Function Block Memory Map

Block	Start Address (HEX)	End Address (HEX)	Size (KB)
CPU Instruction and Data RAM 0	00000000	0001FFFF	128
CPU Instruction and Data RAM 1	00020000	0002BFFF	48
Reserved	00030000	0007FFFF	320
CPU PIC Timer	00080000	000807FF	2
Reserved	00080800	000BFFFF	254
GPIO	000C0000	000C03FF	1
Reserved	000C0800	000C3FFF	14
QSPI Streamer	000C8000	000C83FF	1
Reserved	000C8400	000CFFFF	31
SMBus Mailbox 0	000D0000	000D7FFF	32
Reserved	000D8000	000DFFFF	32
SMBus Mailbox 1	000E0000	000E7FFF	32
General Purpose Timer	000E8000	000E83FF	1
SMBus Filter	000E9000	000E9FFF	4
Reserved	000EA000	000FFFFF	95
UAB	00100000	0012FFFF	192
Customer PLD Logic	00130000	0017FFFF	320
QSPI Monitor	000C4000	000C4FFF	4
Reserved	00200000	002BFFFF	768
Crypto256	00380000	003BFFFF	256
Crypto384 (registers_intf)	002C0000	002FFFFF	256
Reserved	00300000	003FFFFF	1024

2. CPU Subsystem

2.1. Overview

The RISC-V MC processes data and instructions 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 read-only AHB-L interface for instruction and one AHB-L interface with 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.

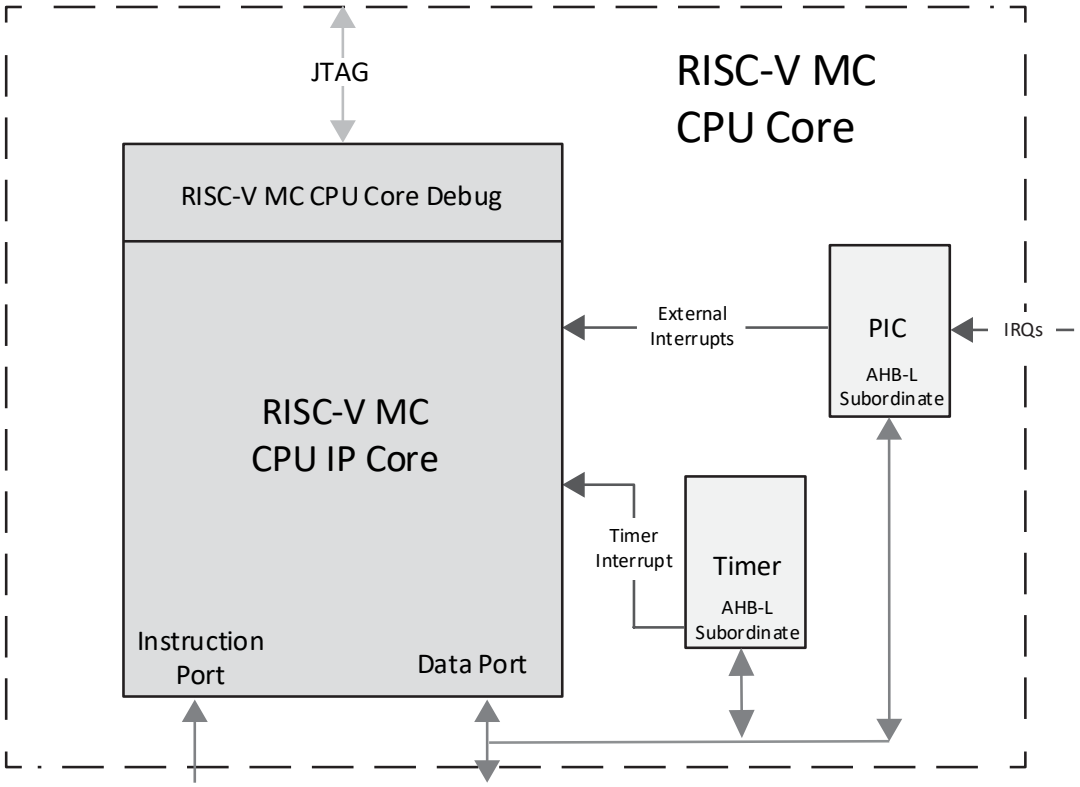


Figure 2.1. RISC-V MC CPU Diagram

2.2. Module Descriptions

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 Register

CSR No.	CSR Name	Access	Fields
0x300	mstatus (machine status register)	read/write	bit[12:11]: mpp, privilege mode before entering a trap, should always be 2'b11 in machine mode in this CPU core. bit[7]: mpie, mie before entering a trap, updates to mie value when entering a trap. bit[3]: mie, global interrupt enable.
0x304	mie (machine interrupt enable register)	read/write	bit[11]: meie, machine mode external interrupt enable. bit[7]: mtie, machine mode timer interrupt enable. bit[3]: msie, machine mode software interrupt enable.
0x305	mtvec	read/write (initialized to 0x20)	bit[31:2]: trap vector base address, 4-byte aligned. bit[0]: trap vector mode, all traps set the program counter to the base address in RISC-V MC CPU core. Bit[1] is not supported. Only 1'0 - direct mode and 1'b1 - vectored mode are available.
0x340	mscratch	read/write	bit[31:0]: in machine mode, it is used to hold a pointer to a machine-mode hart-local context space and swapped with a user register upon entry to a machine mode trap handler.
0x341	mepc (machine exception program counter)	read/write	bit[31:0]: when a trap is taken into machine mode, mepc is used to store the address of the instruction that encounters the exception.
0x342	mcause (machine cause register)	read-only	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)	read-only	bit[31:0]: When a hardware breakpoint is triggered, or an instruction fetch, load, or store address is misaligned, or an access exception occurs, mtval is written with the fault address. It may also be written with illegal instruction when an illegal instruction occurs.
0x344	mip (machine interrupt pending register)	read/write	bit[11]: meip, machine mode external interrupt pending, read-only. bit[7]: mtip, machine mode timer interrupt pending, read-only. bit[3]: msip, machine mode software interrupt pending, readable and writable.

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 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>Description</th></tr>
<tr><td>0x000</td><td>PIC_STATUS</td><td>Interrupt Status Register
Access: read-write
Parameterizable width: min=2, max=8
Indicates the pending interrupt at corresponding interrupt request port(irq[i] at top level).

<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[N-1]</td><td>PIC_STATUS [N-1]</td><td>RW</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_STATUS [1]</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_STATUS [0]</td><td>RW</td><td>1</td><td>0x0</td></tr>
</table>

PIC_STATUS[i]:
Read
• 0 – no interrupt at irq[i]
• 1 – interrupt pending at irq[i]
Write
• 0 – no effect
• 1 – clear interrupt status for irq[i]</td></tr>
<tr><td>0x004</td><td>PIC_ENABLE</td><td>Interrupt Enable Register
Access: read-write
Parameterizable width: min=2, max=8
Indicates whether the processor responds to the interrupt from corresponding interrupt request port (irq[i]) or not.

<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[N-1]</td><td>PIC_ENABLE[N-1]</td><td>RW</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_ENABLE[1]</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_ENABLE[0]</td><td>RW</td><td>1</td><td>0x0</td></tr>
</table>

PIC_ENABLE[i]:
Read
• 0 – irq[i] disabled
• 1 – irq[i] enabled
Write
• 0 – disable irq[i]
• 1 – enable irq[i]</td></tr>
</table>

<table>
<tr><th>Offset</th><th>Name</th><th>Description</th></tr>
<tr><td>0x008</td><td>PIC_SET</td><td>Interrupt Set Register
Access: write-only
Parameterizable width: min=2, max=8
Sets the interrupt status for corresponding interrupt request port(irq[i]).

<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[N-1]</td><td>PIC_SET [N-1]</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_SET [1]</td><td>W</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_SET [0]</td><td>W</td><td>1</td><td>0x0</td></tr>
</table>

PIC_SET[i]:
Read
• Invalid operation gets 0.
Write
• 0 – no effect
• 1 – set interrupt status for irq[i] (set PIC_STATUS[i])</td></tr>
<tr><td>0x00C</td><td>PIC_POL</td><td>Interrupt Polarity Register
Access: read-write
Parameterizable width: min=2, max=8
Indicates the polarity of corresponding interrupt request (irq[i]) port.

<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[N]</td><td>PIC_POL [N]</td><td>RW</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_POL l[1]</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_POL [0]</td><td>RW</td><td>1</td><td>0x0</td></tr>
</table>

PIC_POL[i]:
Read
• 0 – irq[i] is active high
• 1 – irq[i] is active low
Write
• 0 – Set irq[i] active high
• 1 – Set irq[i] 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

Offset	Name	Access	Reset Value	Description
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 bits 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:

- 176 KB SRAM
- Dual AHB-Lite subordinate interface
- FIFO interface connected to QSPI Streamer

3.3. Block Diagram

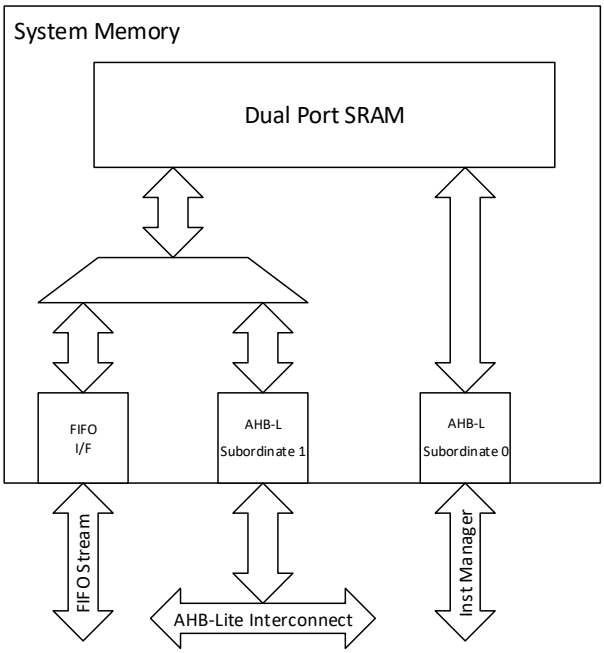


Figure 3.1. System Memory Block Diagram

3.3.1. AHB-Lite Interface

The System Memory has two AHB-Lite subordinate interfaces. Subordinate 0 interface is read-only and is connected directly to the Instruction Manager of the CPU. Subordinate 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-Lite port S1. This interface is used by the QSPI 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 that, 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 two 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 Streamer through a programmable mux/demux block.

4.2. Features

The key features of the QSPI monitor are:

- Supports two external SPI, DSPI, or QSPI buses to monitor illegal activity, each bus has two channels with two chip select pins
- Enable/disable dynamically the flash initialization commands per monitor
- Supports non-volatile configure commands recording
- Supports SPI, DSPI, or QSPI bus transfer mode initialization
- Supports Extended Address Register (EAR) address initialization
- Flash commands, including program, read, and 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 chip select switching

4.3. Block Diagram

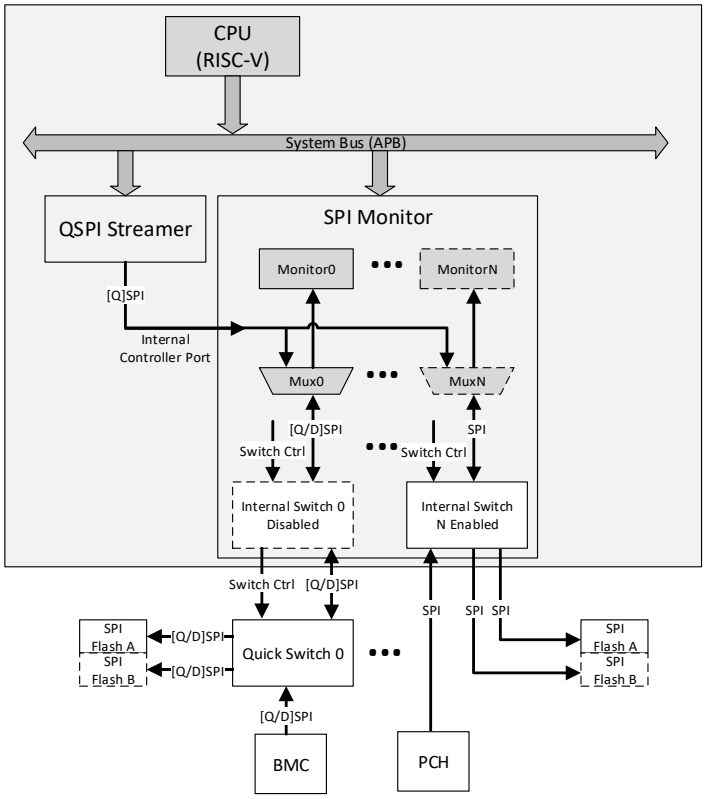


Figure 4.1. QSPI Monitor Block Diagram

4.4. Signal Description

Table 4.1. External 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 controller.
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 controllers, for example, BMC and PCH. This signal is enabled when the QSPI Monitor is switched to the internal controller.

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)	Write Status Register
Initialization Command 1	04 (WRDI)	Write Disable
Initialization Command 2	05 (RDSR)	Read Status Register
Initialization Command 3	06 (WREN)	Write Enable
Initialization Command 4	50 (WRSR_EN)	Volatile Status Register Write Enable

Command	Default	Description
Initialization Command 5	9F (RDID)	Read Identification
Initialization Command 6	5A	Read Serial Flash Discoverable Parameter (SFDP) Mode
Initialization Command 7	70	Read Flag Status Register, refer to related SPI flash data sheet.
Initialization Command 8	36	Individual Block Lock, refer to related SPI flash data sheet.
Initialization Command 9	FFFF	Not supported/defined
Page Program Command	02	Page Program
Page Program Quad Address Quad Data Command	FFFF	Not supported/defined
Erase 4 KB Command	20	Sector Erase (4 KB)
Erase 32 KB Command	FFFF	Not supported/defined
Erase 64 KB Command	D8	Block Erase (64 KB)
Read Command	03	Read
Fast Read Command	0B	Fast Read
Read Dual Data	3B	Dual Output Read
Read Dual Address Dual Data Command	FFFF	Not supported/defined
Read Quad Data Command	6B	Quad Read
Read Quad Address Quad Data Command	EB	Quad Address Quad Read
Quad SPI Mode Enter Command	35	Enable QSPI
Quad SPI Mode Exit Command	F5	Disable QSPI
4-byte Mode Enter Command	B7	Enable 4-byte address mode
4-byte Mode Exit Command	E9	Disable 4-byte address mode
4-byte Read Extended Address Command	FFFF	Not supported/defined
4-byte Write Extended Address Command	FFFF	Not supported/defined
4-byte Page Program Command	12	Page Program 4-byte address
4-byte Page Program Quad Address Quad Data Command	3E	Page Program 4-byte Quad address Quad data
4-byte Erase 4 KB Command	21	Sector Erase (4 KB) 4-byte address
4-byte Erase 32 KB Command	FFFF	Not supported/defined
4-byte Erase 64 KB Command	DC	Block Erase (64 KB) 4-byte address
4-byte Read Command	13	Read 4-byte address
4-byte Fast Read Command	0C	Fast Read 4-byte address
4-byte Read Dual Data Command	3C	Dual Output Read 4-byte address
4-byte Read Dual Address Quad Data Command	FFFF	Not supported/defined
4-byte Read Quad Data Command	6C	Fast Read Quad Data 4-byte address
4-byte Read Quad Address Quad Data Command	EC	Fast Read Quad Address Quad Data 4-byte address

Note: Default value of FFFF means command is disabled/not defined. Usage of this command is blocked by the QSPI Monitor.

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 • 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.
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]

Offset	Register Name	Access	Reset Value	Description
				• 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.

0xN00	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 controller 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]

0xN04	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]
0xN08	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]
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]
0xN20	SPACE0_FILTER_CTRL	RW	0x03	• prg_cmd_allow[0] – Allow program commands in space 0 • erase_cmd_allow[1] – Allow erase commands in space 0 • read_cmd_block[2] – Block read commands in space 0 • reserved[31:3]
0xN24	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.

Offset	Register Name	Access	Reset Value	Description
0xN28	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.
0xN40	SPACE1_FILTER_CTRL	RW	0x03	• prg_cmd_allow[0] – Allow program commands in space 1 • erase_cmd_allow[1] – Allow erase commands in space 1 • read_cmd_block[2] – Block read commands in space 1 • reserved[31:3]
0xN44	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.
0xN48	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.
0xN60	SPACE2_FILTER_CTRL	RW	0x03	• prg_cmd_allow[0] – Allow program commands in space 2 • erase_cmd_allow[1] – Allow (erase commands in space 2 • read_cmd_block[2] – Block read commands in space 2 • reserved[31:3]
0xN64	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.
0xN68	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.
0xN80	SPACE3_FILTER_CTRL	RW	0x03	• prg_cmd_allow[0] – Allow program commands in space 3 • erase_cmd_allow[1] – Allow erase commands in space 3 • read_cmd_block[2] – Block read commands in space 3 • reserved[31:3]
0xN84	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.
0xN88	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.
0xNF0	ILLEGAL_CMD	RO	0x00	• illegal_cmd[7:0]: Illegal operation command • reserved[31:8]
0xNF4	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 Initialization commands (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 can be monitored. Address ranges are also called spaces 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 allowlist/blocklist indicators for each type of command. Address spaces are aligned on 256-byte page boundaries. The default setting for all spaces is to allow program, erase, and read operations in that space. The settings for each space can be modified to

block program, erase, or read operations. Each type of operation (program, erase, or read) has a separate allowlist/blocklist setting.

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

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

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

- If a read operation starts from an address that is inside a blocked address space, or starts from an address outside a blocked address space and the incremental address crosses into a blocked 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 blocked by default and read operations are allowed by default, the recommended usage model is to only define allowed areas for program/erase operations and blocked areas for read operations as address spaces.

Overlapping program/erase allowed and read blocked 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

Command Register	Field Index	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 Mbit 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, for example, read, erase, program, and so on, 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 or 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 and 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, that is, 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 detected does not match any of the commands defined in the QSPI Command List Table (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 which is interpreted by the SPI Flash as illegal command and transaction is aborted.

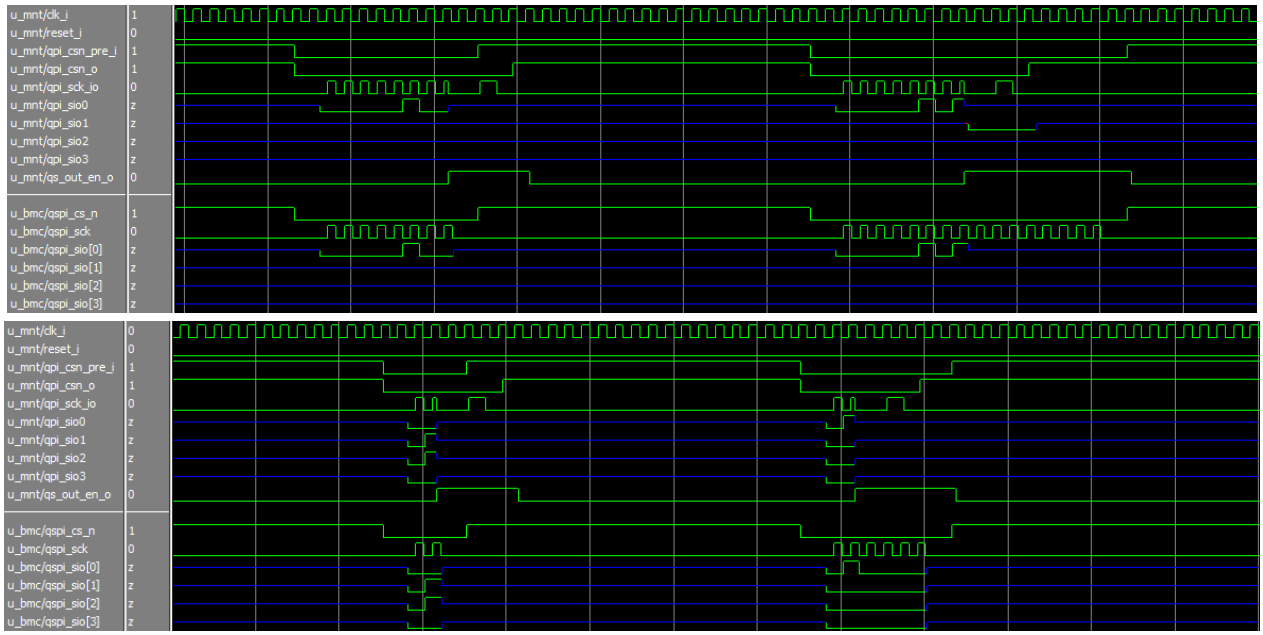
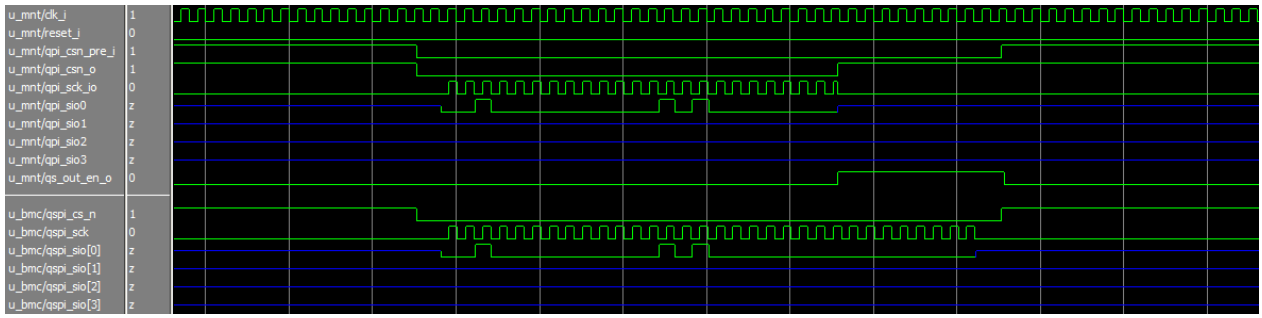


Figure 4.2. One Illegal Command

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.



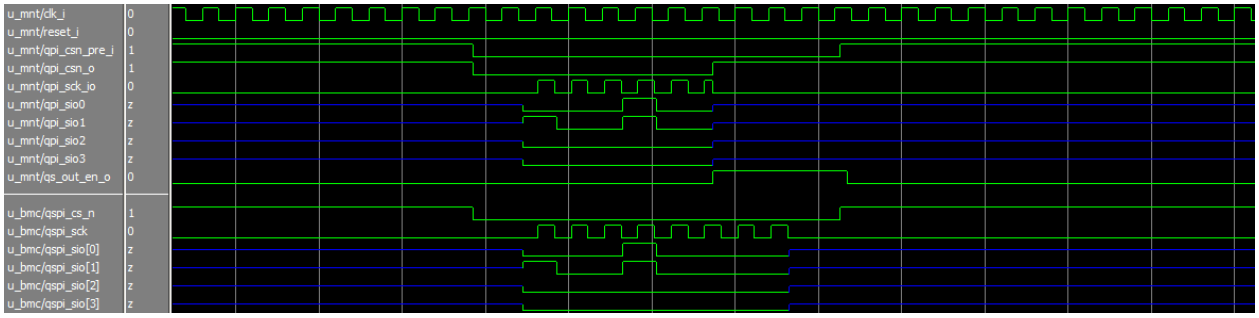


Figure 4.3. Illegal Erase Command

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.

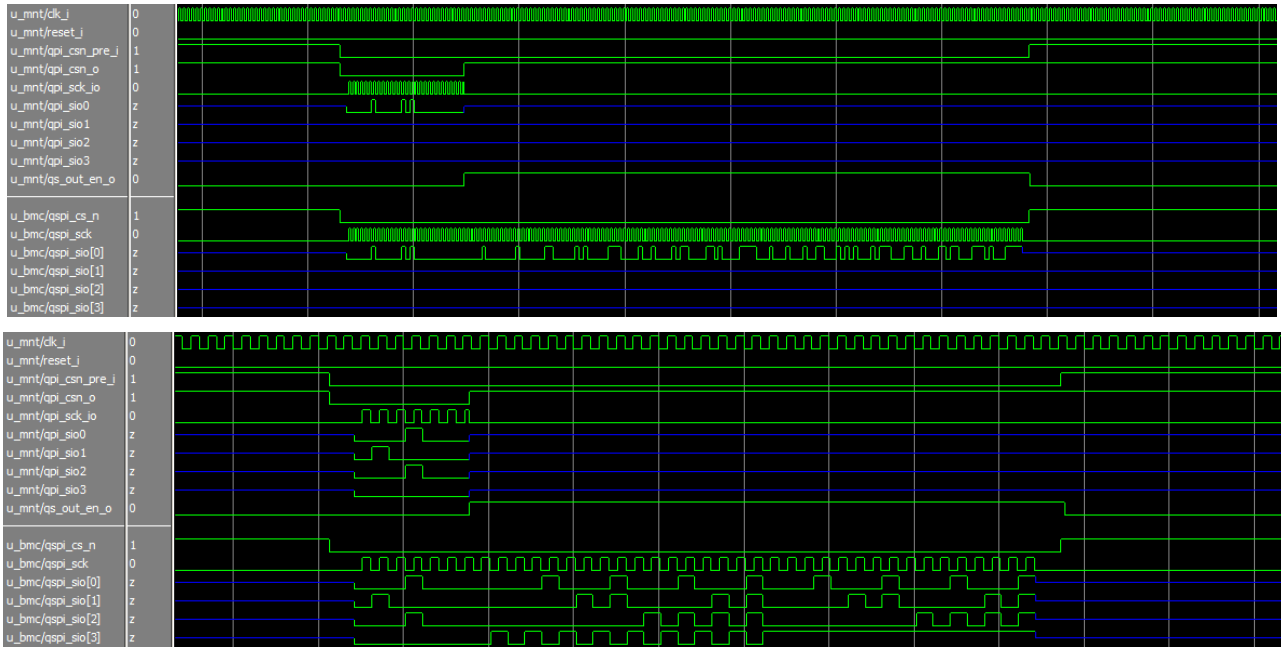
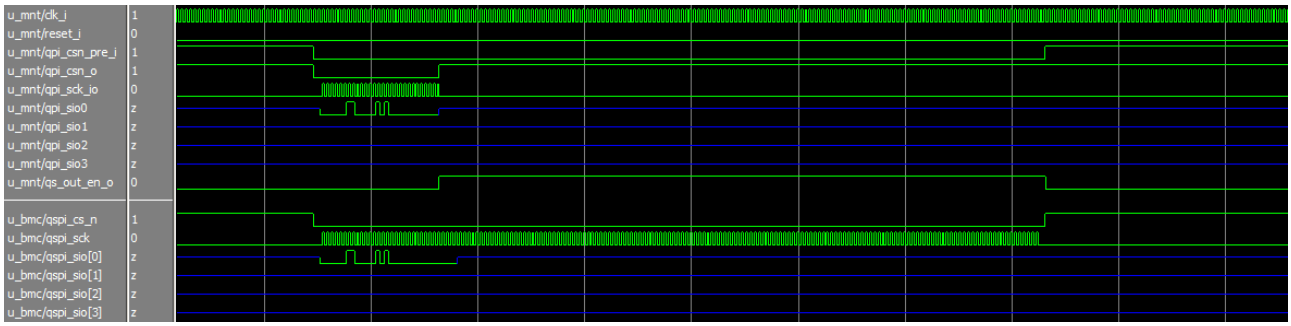


Figure 4.4. Illegal Program Command, 3-Byte Address, Illegal Start Address

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.



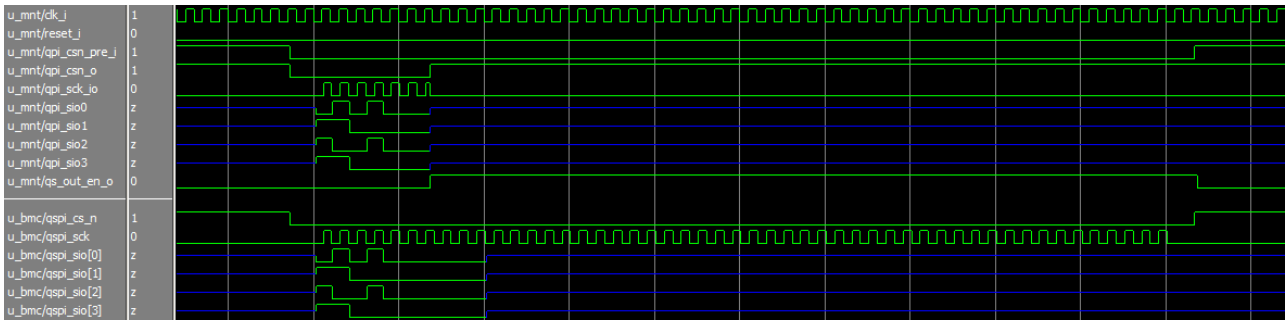


Figure 4.5. Illegal Read Command, 3-Byte Address, Illegal Start Address

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.

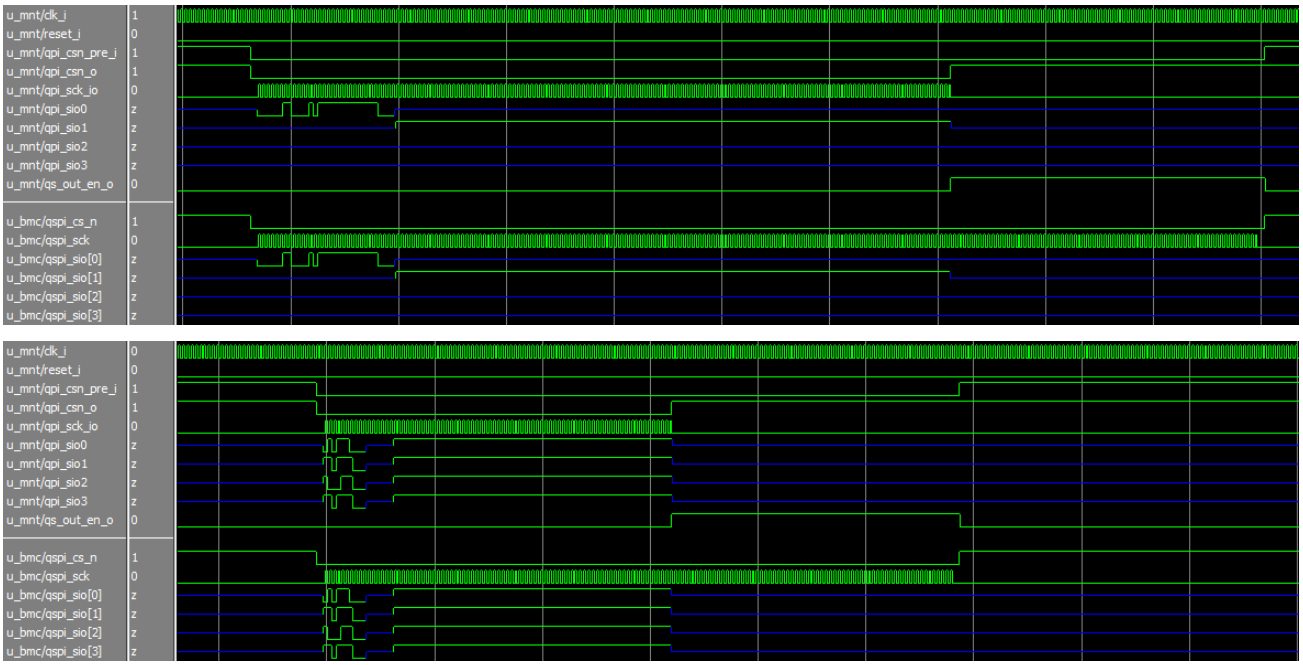
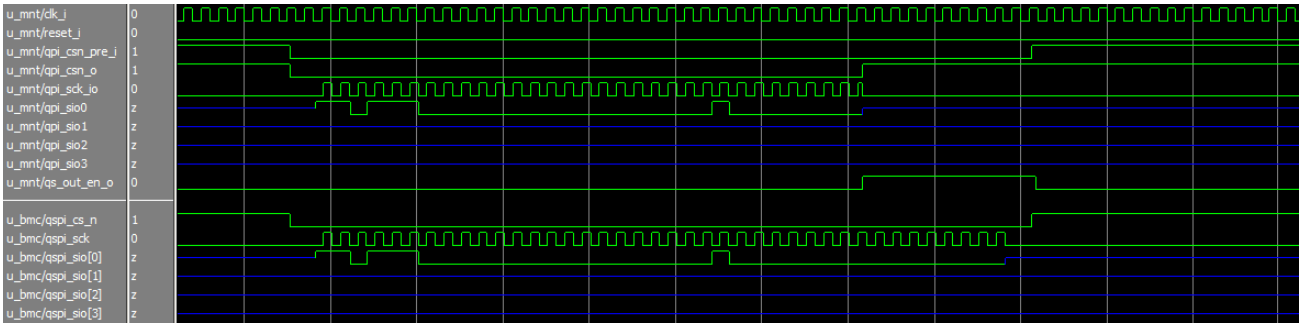


Figure 4.6. Illegal Read Command, 3-Byte Address, Incremental Address Overflow

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.



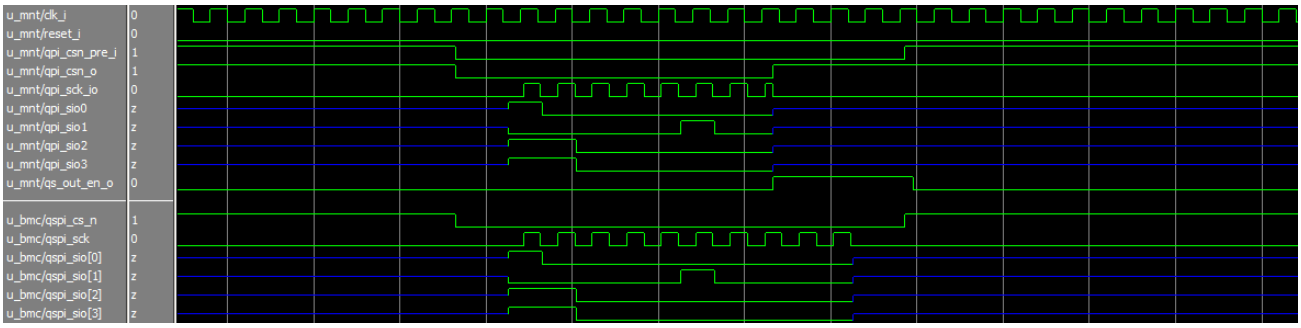


Figure 4.7. Illegal 4-Byte Command Breaking

4.12. Mux/Demux Functionality

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

5. QSPI Streamer

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

The QSPI 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 8-bit Secure Enclave FIFO output interface also enables direct transmission of input data from the SPI target 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 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 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 APB system bus and an 8-bit access port for the SPI controller state machine. 8-bit data is packed or unpacked into 32-bit chunks as it enters or leaves the FIFOs.

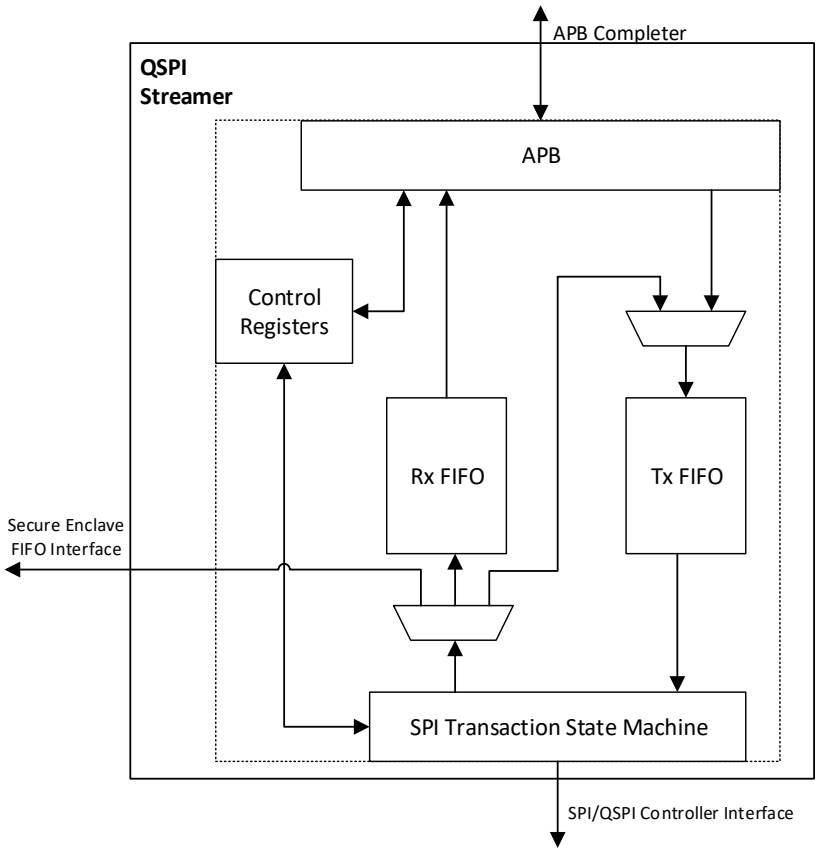


Figure 5.1. QSPI Streamer Block Diagram

5.3. FIFO Configuration

The QSPI 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	<table><tr><th>APB Tx FIFO Data</th><th>31:24</th><th>23:16</th><th>15:8</th><th>7:0</th></tr><tr><td>Big endian</td><td>0</td><td>1</td><td>2</td><td>3</td></tr></table>
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):<table><tr><th>APB Rx FIFO Data</th><th>31:24</th><th>23:16</th><th>15:8</th><th>7:0</th></tr><tr><td>Big endian</td><td>0</td><td>1</td><td>2</td><td>3</td></tr></table>

5.4. Register Description

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

Table 5.2. QSPI Streamer IP Core Registers

Offset	Name	Access	Reset Value	Description

0x00	QSPI_CTRL	RW	0x0	• spi_mode[1:0]
				  • 00: SPI mode 0
				  • 01: reserved
				  • 10: reserved
				  • 11: SPI mode 3
				• reserved[29:2]
				• pll_reset[30]
				  • Write 1 to assert the PLL reset.
				  • Write 1 again to release the PLL reset.
				  • Reads return 0.
				• 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 an SPI transaction currently in progress. The read is blocked until four 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	0x33	• tx_fifo_empty[0] – Tx FIFO is empty. • tx_fifo_almost_empty[1] – asserted Tx FIFO is empty or has less than Tx FIFO Almost Empty Flag bytes. • tx_fifo_almost_full[2] – asserted Tx FIFO is full or 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] – asserted Rx FIFO is empty or has less than Rx FIFO Almost Empty Flag bytes. • rx_fifo_almost_full[6] – asserted Rx FIFO is full or has more than Rx FIFO Almost Full Flag bytes. • rx_fifo_full[7] – Rx FIFO is full. • reserved[10:8] • [11] – if external rx_fifo is enabled, it is ext_rxfifo_full_i, otherwise 0. • [12] – if external rx_fifo is enabled, ext_rxfifo_almost_empty_i, otherwise 0. • reserved[29:13] • [30] – PLL lock state, 1 for lock, 0 for unlock. • busy[31] – SPI transaction is in progress.

Offset	Name	Access	Reset Value	Description

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
				• 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, such as empty, full, and so on 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

Offset	Name	Access	Reset Value	Description
				• 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]
0x30	PLL_PHASE_SHIFT	RW	6'b0	• Phasedir[0] – Dynamic Phase adjustment direction. 0 stands for delayed, 1 for advanced. • Phasesel[3:1] – Dynamic Phase select signal. Only 001: CLKOS2 is supported. • Phaseloadreg[4] – Dynamic Phase load signal. Each pulse of the PHASELOADREG signal generates a phase shift. • Phasestep[5] – Dynamic Phase step signal. The VCO phase changes on the negative edge of the PHASESTEP input after four VCO cycles. • reserved[31:6]*
0x100	PLL access	RW	—	Bypass to PLL interface*

***Note:** Refer to sysCLOCK-PLL-Design-and-Usage-Guide-for-Nexus-Platform (FPGA-TN-02095) for more details.

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 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 example sequence below for details.

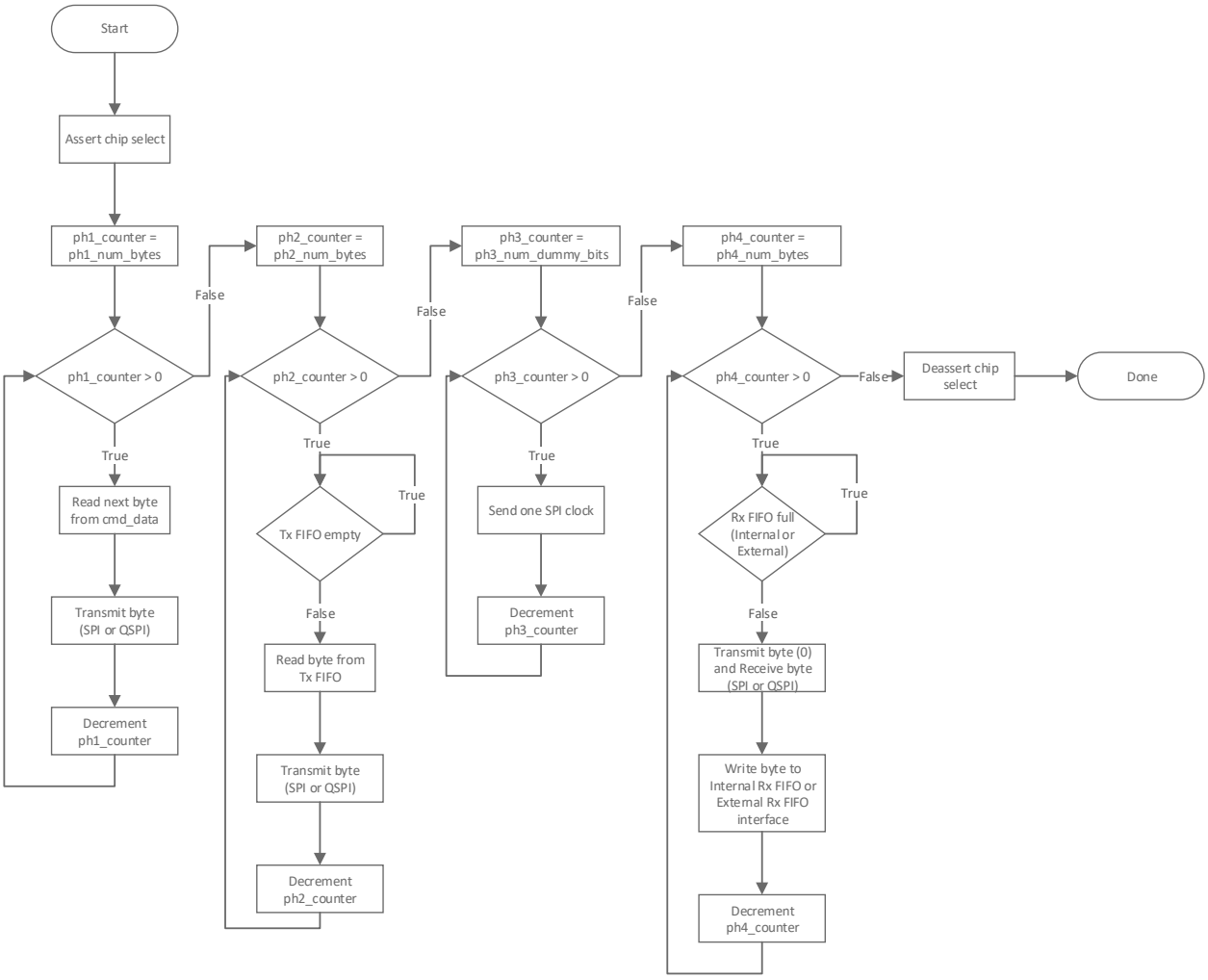


Figure 5.2. QSPI Streamer Programmable Phases

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 target ignores the data.

- SPI Flash Page Program example (Figure 5.3):
  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 this example)
  ph3_num_dummy_bits = 0, ph3_mode = 0
  ph4_num_bytes = 0, ph4_mode = 0

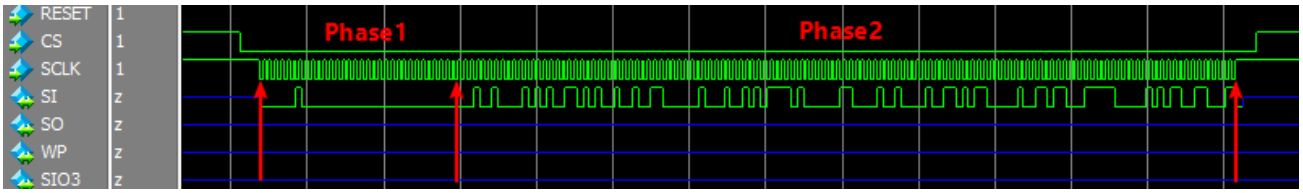


Figure 5.3. Example for Page Program Sequence

- SPI Flash FAST_READ example (Figure 5.4):
  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 this example)
  ph4_num_bytes = M, ph4_mode = 0 (M=16 in this example)

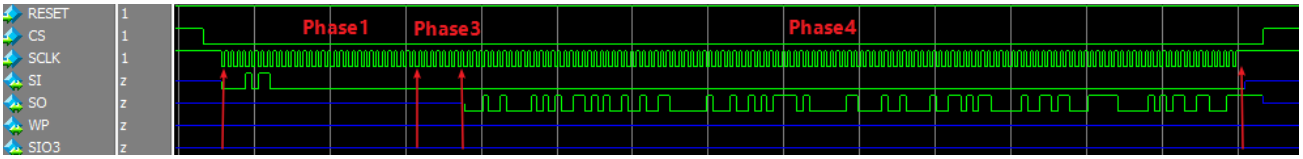


Figure 5.4. Example for FAST_READ Sequence

- SPI RDID example (Figure 5.5):
  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

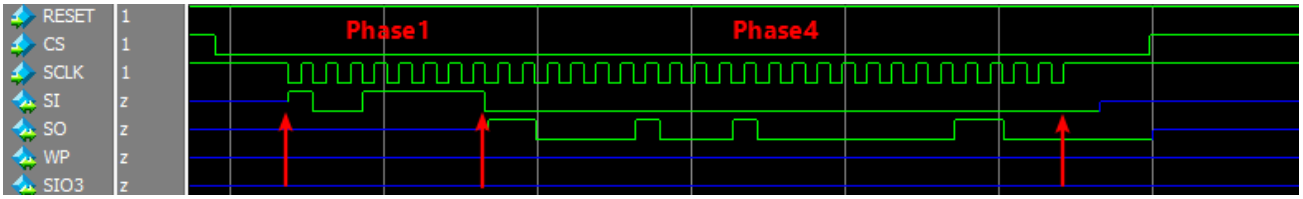


Figure 5.5. Example for RDID Sequence

- SPI Flash QREAD4B example (Figure 5.6):
  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 this example)
  ph4_num_bytes = M, ph4_mode = 2 (M=64 in this example)

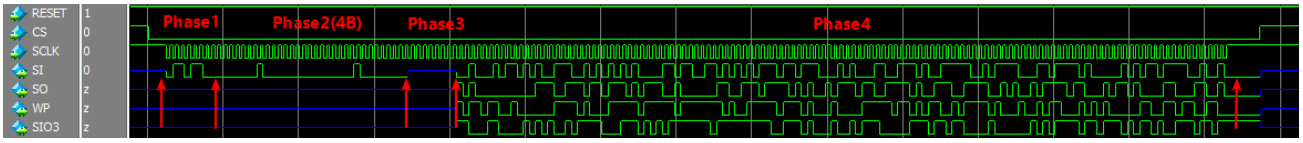


Figure 5.6. Example for QREAD4B Sequence

5.6.2. Width Conversion

Each Tx and Rx FIFO has a 32-bit access port for the system bus and an 8-bit access port for the SPI controller 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 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, that is, 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.

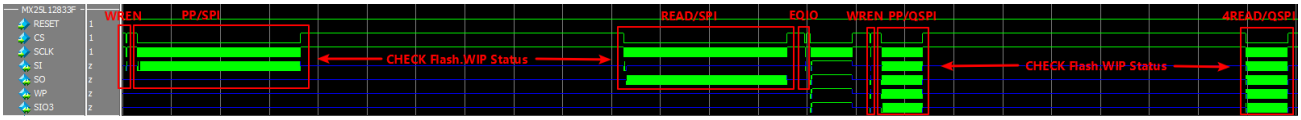


Figure 5.7. Typical Flash Read/Program Flow

6. SMBus Mailbox – Target Mode

6.1. Overview

The I²C Target provides device addressing, read/write operation and an acknowledgement mechanism. It supports two operations:

- Normal SMBus transfer as a target device
- Register file transfer as SMBus Mailbox

6.2. Features

The key features of I²C Target 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 Target Support

6.3. Signal Description

Table 6.1. I²C Target IP Core Signal Description

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

6.4. Register Description

6.4.1. Overview

The I²C Target Core configuration registers are located at the addresses shown in Table 6.2.

Table 6.2. I²C Target 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	Target Address Lower Register
0x08	SLVADR_H_REG	R/W	Target 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

Offset	Register Name	Access Type	Description
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
0x2C	FLUSH_FIFO	WO	Flush FIFO Register
0x30	SMB_CONTROL_REG	RW	SMBus Control and Status Register
0x30–0x3C	Reserved	RSVD	Reserved Write access is ignored and 0 is returned on read access.
0x2000-0x23FF	Registers File	RW	Registers File 256x32 bits for SMBus Mailbox Storage

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.

6.4.2. Write Data Register (WR_DATA_REG)

Table 6.3 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 I^2C 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 6.3. Write Data Register

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

6.4.3. Read Data Register (RD_DATA_REG)

Table 6.4 shows the Read Data register. This is the interface to Receive FIFO. After a data is received from I^2C bus during I^2C 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 I^2C Target 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 6.4. Read Data Register

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

6.4.4. Target Address Registers (SLAVE_ADDRL_REG, SLAVE_ADDRH_REG)

The Target Address Lower Register (SLAVE_ADDRL_REG) shown in Table 6.5 is a 7-bit Target address. This is used for 7-bit and 10-bit addressing mode as follows:
for 7-bit Addressing Mode, it is the Target address;
for 10-bit Addressing Mode, it is the lower 7 bits of the Target address.

Table 6.5. Target Address Lower Register

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

The Target Address Higher Register (SLAVE_ADDRH_REG) shown in Table 6.6 is the upper three bits of 10-bit Target address. This is not used in 7-bit addressing mode.

Table 6.6. Target Address Higher Register

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

6.4.5. Control Register (CONTROL_REG)

Table 6.7 shows the summary of Control Register. Each bit of this register controls the behavior of I²C Target Core.

Table 6.7. Control Register

Field	Name	Access	Width	Reset
[7:6]	Reserved	RSVD	3	—
[5]	dat_src_sw	RW	1	1'b0
[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

- dat_src_sw

  Data source switch. Select data source when external controller read routine.
  - 1'b0 – selects register file for mailbox.
  - 1'b1 – selects tx_fifo for normal external read.
- nack_data

  NACK on Data Phase. Specifies ACK/NACK response on I²C 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 I²C address phase.
  - 1'b0 – Sends ACK to received address if it matches the programmed target address.
  - 1'b1 – Sends NACK to received data.
- reset

  Reset. Resets I²C Target Core for one 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 one clock cycle.
  - 1'b0 – No action.
  - 1'b1 – Resets I²C Target Core.
- clk_stretch_en

  Clock Stretch Enable. Enables clock stretching on ACK bit of data.
  - 1'b0 – I²C Target Core releases SCL signal.
  - 1'b1 – I²C Target 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 Target Core rejects the 10-bit I²C address. It sends NACK.
  - 1'b1 – I²C Target 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.

6.4.6. Target Byte Count Register (TGT_BYTE_CNT_REG)

Table 6.8 shows the summary of Target Byte Count Register. The desired number of bytes to transfer (read/write) in I^2C bus should be written to this register. This is used for Transfer Complete interrupt generation – asserts when the target byte count is achieved.

Table 6.8. Target Byte Count Register

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

6.4.7. Interrupt Status Registers (INT_STATUS1_REG, INT_STATUS2_REG)

Table 6.9 and Table 6.10 show the Interrupt Status Register (INT_STATUS1_REG and INT_STATUS2_REG) which contains all the interrupts currently pending in the I^2C Target 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 6.9. 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 6.10. 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

6.4.8. Interrupt Enable Registers (INT_ENABLE1_REG, INT_ENABLE2_REG)

Table 6.11 and Table 6.12 show the summary of Interrupt Enable Registers that correspond 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 6.11. 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 corresponds to Transfer Complete Interrupt Status.
  - 1'b0 – Interrupt disabled
  - 1'b1 – Interrupt enabled
- stop_det_en

  STOP Condition Detected Interrupt Enable. Interrupt enable bit corresponds 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 corresponds 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 corresponds 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 corresponds 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 corresponds 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 corresponds 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 corresponds to Receive FIFO Ready Interrupt Status.
  - 1'b0 – Interrupt disabled
  - 1'b1 – Interrupt enabled

Table 6.12. 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 corresponds 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 corresponds to START Condition Error Interrupt Status.
  - 1'b0 – Interrupt disabled
  - 1'b1 – Interrupt enabled

6.4.9. Interrupt Set Registers (INT_SET1_REG, INT_SET2_REG)

Table 6.13 and Table 6.14 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 6.13. 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 corresponds 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 corresponds 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 corresponds 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 corresponds 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 corresponds 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 corresponds 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 corresponds 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 6.14. 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 corresponds 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 corresponds to START Condition Error Interrupt Status.
  - 0 – No action
  - 1 – Asserts INT_STATUS2_REG.start_err_set

6.4.10. FIFO Status Register (FIFO_STATUS_REG)

FIFO Status Register reflects the status of Transmit FIFO and Receive FIFO as shown in Table 6.15.

Table 6.15. 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 one 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 one data word.
  - 1'b1 – Receive FIFO is empty.

6.4.11. Flush FIFO Register (FLUSH_FIFO)

Table 6.16. Flush FIFO Register

Field	Name	Access	Width	Reset
[7:2]	reserved	RSVD	6	—
[1]	Rxfifo_flush	WO	1	1'b0
[0]	txfifo_flush	WO	1	1'b0

- rxfifo_flush
  - 0: No action
  - 1: Flush RX FIFO data to empty
- txfifo_flush
  - 0: No action
  - 1: Flush TX FIFO data to empty

6.4.12. Register File

The external SMBus controller initiates an SMBus Mailbox read transaction. The read byte data message is routed to the Register File. The external SMBus controller cannot write message to the Register File. The external SMBus controller writes message to the RX_FIFO. The host reads that message data from the RX_FIFO and writes to the Register File through offset address 0x2000.

6.5. Operations Details

6.5.1. General I²C Operation

In the I²C bus, the transaction is always initiated by the controller. A target may not transmit data unless it has been addressed by the controller. 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. Controller wants to send data to a target.
   a. Controller-transmitter sends a START condition and addresses the target-receiver.
   b. Controller-transmitter sends data to target-receiver.
   c. Controller-transmitter terminates the transfer with a STOP condition.
2. Controller wants to receive/read data from a target.
   a. Controller-receiver sends a START condition and addresses the target-transmitter.
   b. Controller-receiver sends the requested register to read to target-transmitter.
   c. Controller-receiver receives data from the target-transmitter.
   d. Controller-receiver terminates the transfer with a STOP condition.

I²C communication is initiated by the controller sending a START condition and terminated by the controller 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 6.1.

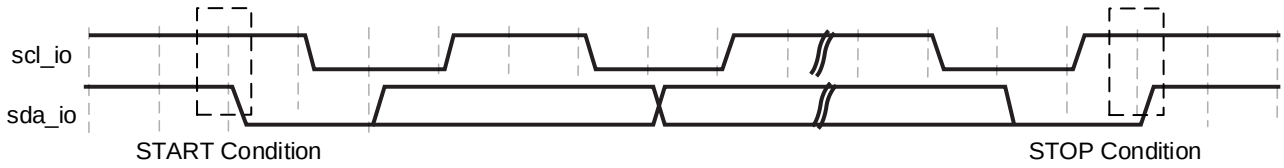


Figure 6.1. START and STOP Conditions

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 target-receiver does not drive the SDA line low during the ACK bit, this indicates that the target-receiver was unable to accept the data and the controller can then generate a Stop condition to abort the transfer. If the controller-receiver does not generate an ACK, this indicates to the target-transmitter that this byte is the last byte of the transfer.

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

6.5.2. Glitch Filter

I²C Target IP Core has integrated glitch filter to remove 50 ns 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.

6.5.3. Clock Stretching

Clock Stretching allows the I²C target 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 target 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. Targets can then hold the SCL line Low after reception and acknowledgment (ACK bit) of a byte to force the controller into a wait state until the target is ready for the next byte transfer.

I²C Target 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 needs more time before it can address the request of I²C controller.

6.5.4. ACK/NACK Response

I²C target 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 the Control Register (CONTROL_REG) section for details. If the host would like to temporarily disable the access to I²C Target Core, it should set CONTROL_REG.nack_addr = 1'b1. In this case, I²C target core sends NACK when it is addressed by the external I²C controller.

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

6.6. Programming Flow

6.6.1. Initialization

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

- SLAVE_ADDRL_REG, SLAVE_ADDRH_REG – This step is optional. In most cases, initial value set in I²C Target Addresses attribute of the user interface does not need to be changed. Read access to the address by external controller is routed to Register File, while write access to the address is routed to internal RX_FIFO.
- 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, which is receiving unknown amount of data.
- INT_ENABLE1_REG – It is recommended to enable only the following interrupts when receiving commands from controller.
  - 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.

6.6.2. Data Transfer in response to I²C Controller Read

As mentioned, the two target addresses for the IP are the normal SMBus target device data transfer and the SMBus mailbox Register File access. According to the accessed address, there are two ways to respond to the external controller read.

6.6.2.1. Normal SMBus Target Device Read Data Transfer

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

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

1. Write data to WR_DATA_REG, amounting to <= FIFO Depth.
2. Enable only Transfer Complete Interrupt. If transmit data is > FIFO Depth. Also enable TX FIFO Almost Empty interrupt if there are 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.
Also read INT_STATUS2_REG to check that no error occurred.

4. Clear TX FIFO Almost Empty Interrupt. It is 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.
   Also read INT_STATUS2_REG to check that no error occurred.
8. Clear all interrupts.

6.6.2.2. SMBus Mailbox Register File Read Data Transfer

If the accessed address is Register File, the SMBus Mailbox IP outputs the addressed data in Register File automatically. The data format is shown in Figure 6.2.

Figure 6.2. SMBus Mailbox Read Byte Message

6.6.3. Data Transfer in Response to I²C Controller Write

Similarly, the external SMBus controller can initiate controller write transaction to two target addresses. One is routed to internal RX_FIFO logic and the other is to Register File through the RISC-V host.

6.6.3.1. Normal SMBus Target Device Write Data Transfer

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

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

1. Enable only Transfer Complete Interrupt. If data to receive is > FIFO Depth. Also enable 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.
   Also read INT_STATUS2_REG to check that no error occurs.
3. Clear RX FIFO Almost Full Interrupt. It is 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.
   Also read INT_STATUS2_REG to check that no error occurs.
7. Clear all interrupts.
8. Read all data from RD_DATA_REG.

6.6.3.2. SMBus Mailbox Register File Write Data Transfer

If the accessed address is Register File, the external controller write data firstly inputs to RX_FIFO. The host reads out the data and writes it to the Register File according to the Register File address. The data format is shown in Figure 6.3.

Figure 6.3. SMBus Mailbox Write Byte Message

6.7. SMBus Target Support

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

6.7.1. SMBus Control and Status Register

Table 6.17. SMBus Register Address Map

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

Table 6.18. 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 controller
  - 1'b0 – No interrupt to controller
  - 1'b1 – SMBus target sent alert interrupt to controller

6.7.2. Operation Details

6.7.2.1. SMBAlert Operation

A target device can signal the controller through SMBUSx_INT interrupt line that it wants to talk. The controller processes the interrupt and simultaneously accesses all the SMBAlert devices through the Alert Response Address. Only the target device which pulls SMBUSx_INT low acknowledges the Alert Response Address, 0001 100b. The host performs a modified Receive Byte operation. The 7-bit device address provided by the target 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_INT low, the highest priority device, which has the lowest address, wins the communication rights.

After receiving an acknowledge (ACK) from the controller in response to its address, the device stops pulling down the SMBUSx_INT signal. If the controller still sees the SMBUSx_INT low when the message transfer is complete, the same process repeats again. The SMBus target controller monitors the data bus to see if any other target 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.

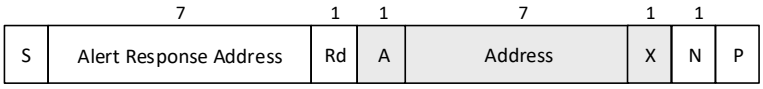


Figure 6.4. SMBus 7-Bit Addressable Device Response

7. SMBus Mailbox – Controller Mode

SMBus controller core is used to initiate SMBus transfer to access other SMBus targets. It supports multi-controller on one bus.

7.1. Signal Description

Table 7.1. External Signals of SMBus Controller

Signal	Direction	Description
I²C/SMBus Target		
scl_io	Bidir	SMBus/I²C Serial Clock
sda_io	Bidir	SMBus/I²C Data

7.2. Register Description

Offset	Register Name	Access	Reset	Description
0x400	PRERlo	R/W	[31:8] RSVD [7:0] 8'hff	Clock prescale register low-byte 5×SCL frequency = clk_i / (PRERhi<<8 + PRERlo)
0x404	PRERhi	R/W	[31:8] RSVD [7:0] 8'hff	Clock prescale register high-byte 5×SCL frequency = clk_i / (PRERhi<<8 + PRERlo)
0x408	CTR	R/W	[31:8] RSVD [7:6] 2'h00 [5:0] RSVD	Control Register **Field** / **Description** [7] EN – Controller enable bit 1 = Core is enabled. 0 = Core is disabled. [6] IEN – Interrupt enable bit 1 = Interrupt is enabled. 0 = Interrupt is disabled.
0x40c	TXR	WO	[31:8] RSVD [7:0] 8'h00	Transmit Register **Field** / **Description** [7:1] Next byte to be transmitted via SMBus Controller [0] a) The byte's LSB. b) RW bit during target address transfer 1 = Reading from target 0 = Writing to target
0x40c	RXR	RO	[31:8] RSVD [7:0] 8'h00	Receive Register Last byte received through the SMBus Controller
0x410	CR	WO	[31:8] RSVD [7:0] 8'h00	Command Register **Field** / **Name** [7] STA – Generate (repeated) start condition [6] STO – Generate stop condition [5] RD – Read from target [4] WR – Write to target [3] ACK, when a receiver, sent ACK (ACK = 0) or NACK (ACK = 1) [2] When set, clears the timeout status [1] Reserved [0] IACK – Interrupt acknowledge When set, clears a pending interrupt.

<table>
<tr><th>Offset</th><th>Register Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td>0x410</td><td>SR</td><td>RO</td><td>[31:8] RSVD
[7:0] 8’h00</td><td>Status Register
<table>
<tr><th>Field</th><th>Name</th></tr>
<tr><td>[7]</td><td>RxACK – Receives acknowledge from addressed target.
1 = No acknowledge received.
0 = Acknowledge received.</td></tr>
<tr><td>[6]</td><td>Busy – Indicates that the SMBus bus is busy.
1 = bus is busy.
0 = bus is idle.</td></tr>
<tr><td>[5]</td><td>AL – Arbitration lost. This bit is set when the core loses arbitration. Arbitration is lost when:
A STOP signal is detected but not requested.
A START signal is detected but not requested.
The controller drives SDA high, but SDA is low.
Controller drives SCL high, but SCL is low. Not clock stretch.</td></tr>
<tr><td>[4]</td><td>Reserved</td></tr>
<tr><td>[3]</td><td>Timeout
1 = SCL and SDA line have been high for 50 us.</td></tr>
<tr><td>[2]</td><td>Timeout
1 = SCL line has been low for 200 ms.</td></tr>
<tr><td>[1]</td><td>TIP – Transfer in Progress
1 = Transferring data
0 = Transfer is complete</td></tr>
<tr><td>[0]</td><td>IF – Interrupt Flag. This bit is set when an interrupt is pending. The int_o signal is asserted if the IEN bit is set. The Interrupt Flag is set when:
One byte transfer has been completed.
Target NACK.
Arbitration is lost.
Controller changes from busy to idle.
smbalert_n_i signal falling edge is detected.</td></tr>
</table></td></tr>
</table>

SMBus register offset address starts at 0x400.

The prescale register, offset = 0x400 and 0x404, is used to prescale the SCL clock line based on the controller clock. This design uses an internal clock enable signal, clk_en, to generate the SCL clock frequency. The frequency of clk_en is calculated by the equation [system_clock frequency / (Prescale Register + 1)] and this frequency is five times SCL frequency. The contents of the prescale register can only be modified when the core is not enabled.

The control register, offset = 0x408, has only two bits used for this design. The MSB of this register is the most critical one because it enables or disables the entire SMBus core. The core does not respond to any command unless this bit is set.

The transmit register and the receive register share the same address, offset = 0x40C, depending on the direction of data transfer. The data to be transmitted through the SMBus is stored in the transmit register, while the byte received through the SMBus is available in the receive register.

The status register and the command register share the same address, offset = 0x410. The status register allows the monitoring of the SMBus operations, while the command register stores the next command for the next SMBus operation. Unlike the rest of the registers, the bits in the command register are cleared automatically after each operation. Therefore, this register must be written for each start, write, read, or stop of the SMBus operation.

7.3. Programming Flow

7.3.1. Initialization

Write the appropriate data to the prescale register based on the frequency of SCL through the AHB-Lite bus S02. The SCL frequency meets the equation: 5×SCL frequency = clk_i/(PRERhi<<8 + PRERlo).

7.3.2. SMBus Controller Operation Flow

Figure 7.1 shows the SMBus controller program flow in interrupt mode. The controller can also be used in polling mode. The polling mode is the same as interrupt mode except that the polling mode needs to poll the SR bit 0 instead of being interrupted by int_o to check status. In the polling mode, set the CTR to 0x80.

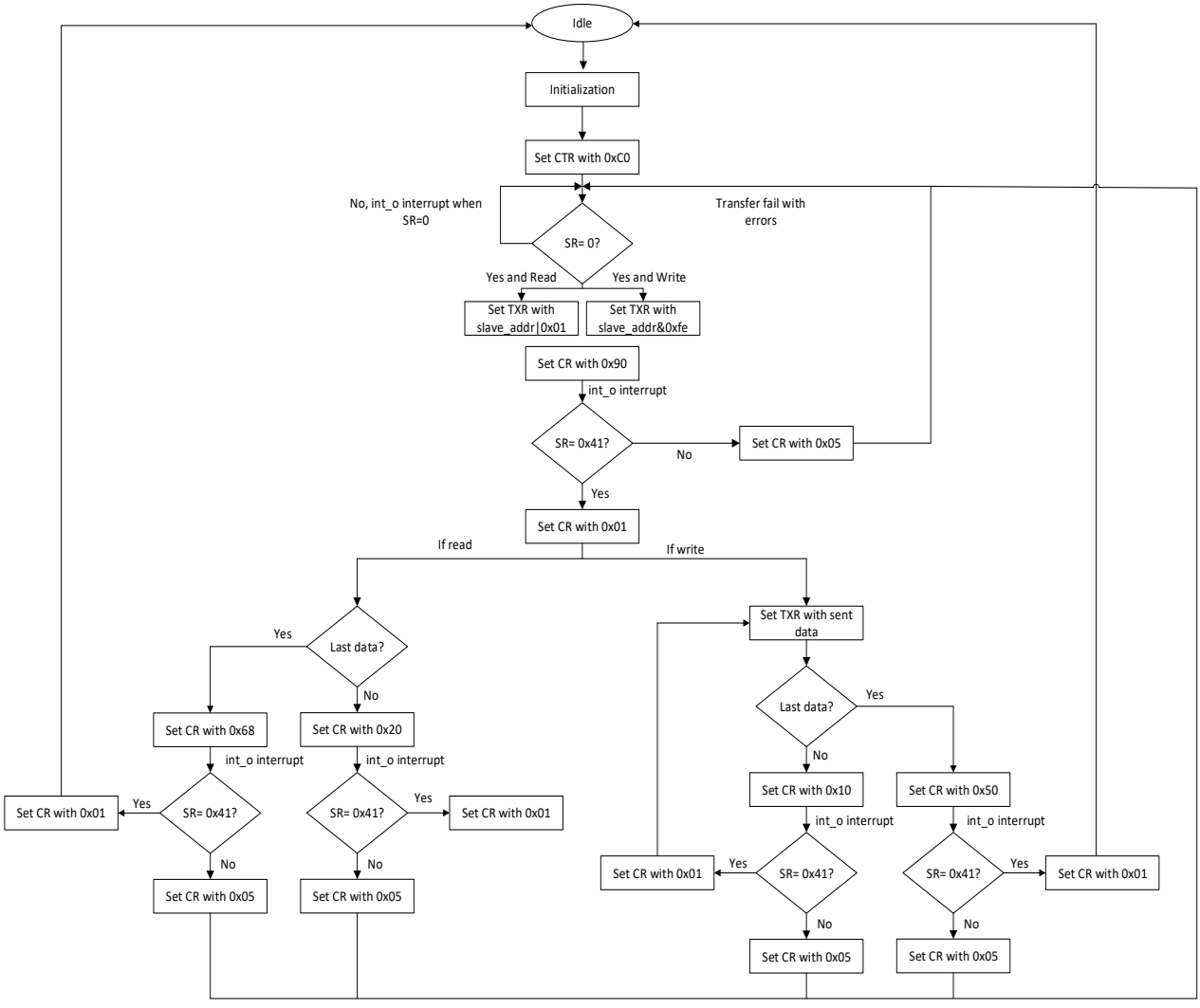


Figure 7.1. SMBus Controller Program Flow Interrupt Mode

7.3.3. Write Data to the SMBus Target

1. Write 0x80 to the control register (CTR) to enable the SMBus Controller through the AHB-Lite bus. If enable interrupt, the write data is 0xC0.
2. Read the status register (SR) through the AHB-Lite bus until all bits of the status register is 0.
3. Write the SMBus target address and write bit to the transmit register (TXR) through the AHB-Lite bus.
4. Write 0x90 to the command register (CR) through the AHB-Lite bus to start the SMBus write operation.
5. When using polling mode, read the status register (SR) until bit 0 of the status register is set and check if other bits except bit 6 are 0s. When using interrupt mode, if the host is interrupted by int_o signal, read the status register (SR) and check if other bits except bit 0 and bit 6 are 0s. Both modes need to write 0x1 to CR to clear bit 0 of SR. If other bits except bit 0 and bit 6 are not 0s, there is an error, write 0x5 to CR to clear SR and go back to step 2.
6. Write the byte which is sent to the SMBus target to the transmit register (TXR) through the AHB-Lite bus.
7. Write 0x10 to the CR through the AHB-Lite bus to set SMBus write operation.
8. When using polling mode, read the status register (SR) until bit 0 of the status register is set and check if other bits except bit 6 are 0s. When using interrupt mode, if the host is interrupted by int_o signal, read the status register (SR) and check if other bits except bit 0 and bit 6 are 0s. Both modes need to write 0x1 to CR to clear bit 0 of SR. If other bits except bit 0 and bit 6 are not 0s, there is an error, write 0x5 to CR to clear SR and go back to step 2. If there is no error and there are other data to write, go back to step 6.
9. When sending the last byte, write 0x50 to the command register (CR) through the AHB-Lite bus to write the last byte and stop the SMBus write operation.
10. When using polling mode, read the status register (SR) until bit 0 of the status register is set and check if other bits except bit 6 are 0s. Bit 6 is set when other controllers use the bus at this time. Otherwise, it also should be 0. When using interrupt mode, if the host is interrupted by the int_o signal, read the status register (SR) and check if other bits except bit 0 and bit 6 are 0s. Both modes need to write 0x1 to CR to clear bit 0 of SR. If other bits except bit 0 and bit 6 are not 0s, there is an error. Write 0x5 to CR to clear SR and go back to step 9.

7.3.4. Read Data from the SMBus Target

1. Write 0x80 to the control register (CTR) to enable the SMBus Controller through the AHB-Lite bus. If enable interrupt, the write data is 0xC0.
2. Read the status register (SR) through the AHB-Lite bus until all bits of the status register is 0s.
3. Write the SMBus target address and the read bit to the transmit register (TXR) through the AHB-Lite bus.
4. Write 0x90 to the command register (CR) through the AHB-Lite bus to start the SMBus read operation.
5. When using polling mode, read the status register (SR) until bit 0 of the status register is set and check if other bits except bit 6 are 0s. When using interrupt mode, if the host is interrupted by the int_o signal, read the status register (SR) and check if other bits except bit 0 and bit 6 are 0s. Both modes need to write 0x1 to CR to clear bit 0 of SR. If other bits except bit 0 and bit 6 are not 0s, there is an error. Write 0x5 to CR to clear SR and go back to step 2.
6. Write 0x20 to command register (CR) through the AHB-Lite bus to read data from the target. If it is the last byte to read, write 0x28 to command register (CR) to NACK last byte.
7. When using polling mode, read the status register (SR) until bit 0 of the status register is set and check if other bits except bit 6 are 0s. When using interrupt mode, if host is interrupted by int_o signal, read the status register (SR) and check if other bits except bit 0 and bit 6 are 0s. Both modes need to write 0x1 to CR to clear bit 0 of SR. If other bits except bit 0 and bit 6 are not 0s, there is an error, write 0x5 to CR to clear SR and go back to step 2.
8. Read data from the receive register (RXR) through the AHB-Lite bus. If there is no error and there are other data to read, go back to step 6.
9. When reading the last byte, write 0x68 to the command register (CR) through the AHB-Lite bus to read the last byte and stop the SMBus read operation.
10. When using polling mode, read the status register (SR) until bit 0 of the status register is set and check if other bits except bit 6 are 0s. Bit 6 is set when other controllers use the bus at the same time. Otherwise, it also should be 0. When using interrupt mode, if the host is interrupted by the int_o signal, read the status register (SR) and check if

other bits except bit 0 and bit 6 are 0s. Both modes need to write 0x1 to CR to clear bit 0 of SR. If other bits except bit 0 and bit 6 are not 0s, there is an error. Write 0x5 to CR to clear SR and go back to step 9.

8. I²C/SMBus Filter

SMBus Relay with filter, named as I²C filter in this document, is designed to function as an invisible relay from the point of view of both Controller and Target devices on the bus. It is meant to be directly attached to the controller port and protect all target devices against malicious traffic generated from the controller port based on an allowlist of allowable commands set by the host, such as CPU, FPGA RoT design, and others. The filter IP is the subset of the SMBus protocol. SMBSUS# and SMBALERT# are not supported.

The design is implemented in Verilog HDL. It can be configured and generated using the Lattice Propel™ Builder software. It can be implemented using the Lattice Radiant™ software and the Lattice Diamond® Place and Route tool. The module registers are accessed by the host using a 32-bit AHB-Lite interface. The host can be a CPU, FPGA RoT design, and others.

8.1. Features

The I²C filter soft IP has the following features:

- Provides four interfaces, namely, AHB-Lite, SMBus controller, SMBus Target, and Interrupt.
- Connected between a single controller and multiple target devices.
- Protects the secondary devices from malicious traffic generated from the controller.
- Does not violate SMBus protocol and is transparent between the Primary and Secondary devices.
- Allows all the Read access.
- Verifies all the write access against an allowlist of allowable opcodes (SMBus command) set inside the memory.
- The opcodes in the memory can be initialized and/or written by the host, such as CPU, FPGA RoT design, and others, through the AHB-Lite bus.
- Samples the commands from controller with a high frequency system clock before passing it to the secondary devices.
- Supports clock stretching from both primary and secondary devices.
- Supports glitch filter from both primary and secondary devices.
- Each bit in the memory represents one opcode (SMBus command) equivalent to the address. Write is allowed if the bit value is 1 and not-allowed if the bit value is 0. 256 bits are required to be supported for each target device.
- Supports up to 128 target-devices on the bus, with 7-bit addressing supported only.
- When a block event occurs, the write transaction is halted, an interrupt is sent to the host, and the status register is updated with the blocked command for the corresponding target address.

8.2. Conventions

The nomenclature used in this document is based on Verilog HDL.

8.3. Functional Description

8.3.1. Overview

The Relay logic is required to pass the communication between controller and target devices that does not violate SMBus protocol and is transparent from both controller and target devices attached on the bus. This is achieved through a mix of oversampling the bus as well as comprehension of direction changes during transmitting of SMBus frames. The Filter logic allows all read accesses. Any write access is required to be verified against an allowlist of allowable opcodes. Based on SMBus protocol, the first byte after start bit is the desired 7-bit target address along with the operation bit: 0 = write, 1 = read. The second byte is the SMBus command. The following bytes are data. The write access is verified by checking the command byte according to the target's allowlist.

As shown in Figure 8.1, I²C filter uses one AHB-Lite interface for filter allowlist configuration and register access. The SMBus controller port is used to connect SMBus controller while SMBus target port is used to connect SMBus target. There is only one SMBus controller and up to 128 targets on one bus.

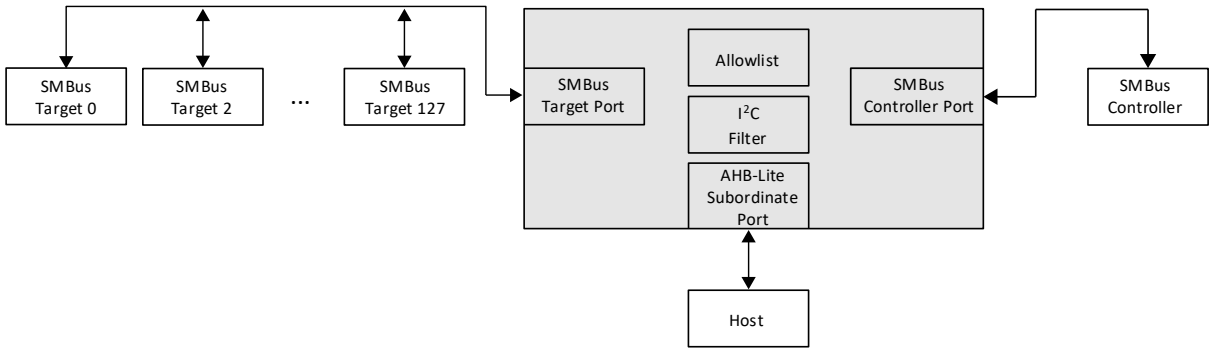


Figure 8.1. I²C Filter Topology

8.3.2. Read Transaction

Figure 8.2 shows an example of a read transaction relayed through the component. The skew visible between SMBus controller device to I²C filter controller port and I²C filter target port to SMBus target device lines is the result of passing through the Relay and is within margins not violating SMBus signaling. Read transaction is always allowed by the I²C filter, ignoring the allowlist.

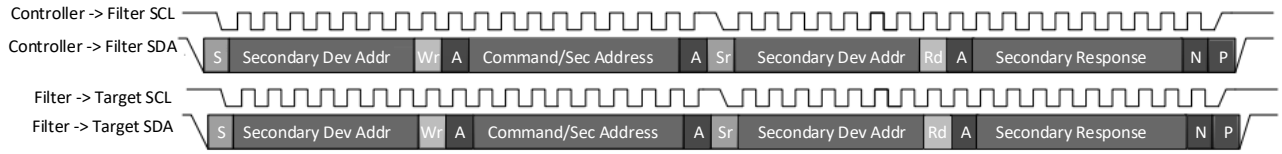


Figure 8.2. I²C Filter Read Transaction

8.3.3. Non-blocked Write Transaction

Figure 8.3 shows an example of a valid write transaction relayed through the component. The write transaction is not blocked because the command transferred is in the allowlist.

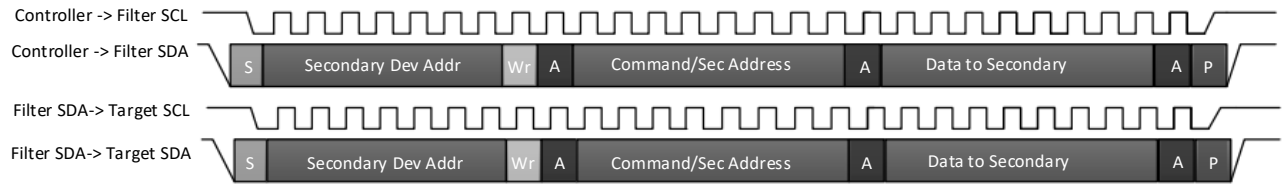


Figure 8.3. I²C Filter Non-blocked Write Transaction

8.3.4. Blocked Write Transaction

Figure 8.4. below shows an example of invalid write transaction filtered out through the component, in this case the I²C filter asserts SMBus Stop condition (P) leading to an abort on the target device, while also responding with a NACK to the controller device signaling an error in the write transaction. Stop target device occurs at command phase while NACK to controller device occurs at first byte data phase. The write transaction is blocked due to the command transferred is not in the allowlist.

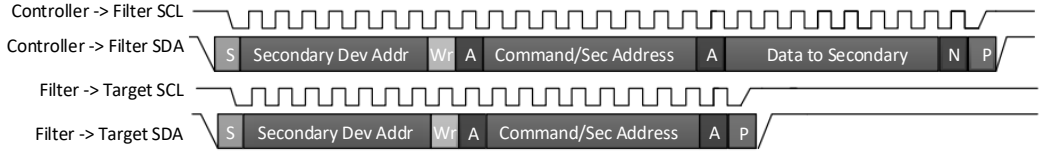


Figure 8.4. I²C Filter Blocked Write Transaction

8.3.5. Signals Description

Table 8.1. Interface Signal Description

Signal Name	Width	Direction	Description
Clock and Reset			
clk_i	1	Input	System clock
rstn_i	1	Input	System reset. The reset assertion can be asynchronous but reset negation should be synchronous. This signal is active low. When asserted, output ports and registers are forced to their reset values.
SCL Clock Speed			
scl_speed_i	2	Input	Indicates the SCL clock speed to be sent to the target devices. 2'b00: Reserved 2'b01: 100 kHz 2'b10: 400 kHz 2'b11: 1000 kHz
AHB-Lite Bus			
ahbl_hsel_slv_i	1	Input	AHB-L Select Signal Indicates that the target device is selected and a data transfer is required.
ahbl_haddr_slv_i	32	Input	The system address bus.
ahbl_hburst_slv_i	3	Input	3'b000: SINGLE Single burst 3'b001: INCR Incrementing burst of undefined length (NOT supported) 3'b010: WRAP4 4-bit wrapping burst 3'b011: INCR4 4-bit incrementing burst 4'b100: WRAP8 8-bit wrapping burst 3'b101: INCR8 8-bit incrementing burst 8'b110: WRAP16 16-bit wrapping burst 3'b111: INCR16 16-bit incrementing burst
ahbl_hprot_slv_i	4	Input	ahbl_hprot_slv_i [0]: 1'b0 – opcode fetch; 1'b1 – data access ahbl_hprot_slv_i [1]: 1'b0 – user access; 1'b1 – privileged access ahbl_hprot_slv_i [2]: 1'b0 – non-bufferable, 1'b1 – bufferable ahbl_hprot_slv_i [3]: 1'b0 – non-cacheable; 1'b1 – cacheable
ahbl_hsize_slv_i	3	Input	3'b000: one byte 3'b001: two bytes 3'b010: four bytes
ahbl_htrans_slv_i	2	Input	Indicates the transfer type of the current transfer. This can be: 2'b00: IDLE 2'b01: BUSY 2'b10: NONSEQUENTIAL 2'b11: SEQUENTIAL
ahbl_hwdata_slv_i	32	Input	The write data bus
ahbl_hwrite_slv_i	1	Input	When HIGH, this signal indicates a write transfer and when LOW, it indicates a read transfer.
ahbl_hready_slv_i	1	Input	This signal should come from AHB-L Interconnect. When set to 1, this indicates the previous transfer is complete.
ahbl_hrdata_slv_o	32	Output	The read data bus
ahbl_hreadyout_slv_o	1	Output	When HIGH, this signal indicates that a transfer has finished on the bus. This signal can be driven LOW to extend a transfer.
ahbl_hresp_slv_o	1	Output	When LOW, this signal indicates that the transfer status is OKAY. When HIGH, it indicates that the transfer status is ERROR.
Interrupt Signal			
irq	1	Output	Interrupt to host (CPU), reset value is 1'b0.

Signal Name	Width	Direction	Description
SMBus Signal (Remove Tri-State Buffer = 0)			
scl_m	1	Input/Output	SMbus clock connect to controller
sda_m	1	Input/Output	SMbus data connect to controller
scl_s	1	Input/Output	SMbus clock connect to target
sda_s	1	Input/Output	SMbus data connect to target
SMBus Signal (Remove Tri-State Buffer = 1)			
scl_m_in	1	Input	SMbus clock input from controller
scl_m_outen	1	Output	SMbus clock output enable to controller 0: Buffer should work as output (tri-state should get scl_m_out) 1: Buffer should work as input
scl_m_out	1	Output	SMbus clock output to controller
sda_m_in	1	Input	SMbus data input from controller
sda_m_outen	1	Output	SMbus data output enable to controller 0: Buffer should work as output (tri-state should get sda_m_out) 1: Buffer should work as input
sda_m_out	1	Output	SMbus data output to controller
scl_s_in	1	Input	SMbus clock input from target
scl_s_outen	1	Output	SMbus clock output enable to target 0: Buffer should work as output (tri-state should get scl_s_out) 1: Buffer should work as input
scl_s_out	1	Output	SMbus clock output to target
sda_s_in	1	Input	SMbus data input from target
sda_s_outen	1	Output	SMbus data output enable to target 0: Buffer should work as output (tri-state should get sda_s_out) 1: Buffer should work as input
sda_s_out	1	Output	SMbus data output to target

8.3.6. Register Description

The register address map, shown in Table 8.2, specifies the available IP core registers. The offset of each register increments by four to allow easy interfacing with the processor and system buses. In this case, each register is 32-bit wide.

Table 8.2. Register Address Map

Offset	Register Name	Access	Reset	Description
Allowlist (EBR)				
0x00	Allowlist number (range 0 ~ 59). Specify allowlist number corresponding to target address. Target addresses vary from 7'd0 to 7'd7f for 7 bits address. Allowlist numbers may be 0~59 for this design. But the maximum number could extend to 255 in the future. One target	WO	32'h0	Target address 7'd3, 7'd2, 7'd1, 7'd0 allowlist number. bit31:24 -> address 7'd3 allowlist number bit23:16 -> address 7'd2 allowlist number bit15:8 -> address 7'd1 allowlist number bit7:0 -> address 7'd0 allowlist number
0x04				Target address 7'd7, 7'd6, 7'd5, 7'd4 allowlist number. bit31:24 -> address 7'd7 allowlist number bit23:16 -> address 7'd6 allowlist number bit15:8 -> address 7'd5 allowlist number bit7:0 -> address 7'd4 allowlist number
0x08				Target address 7'd11, 7'd10, 7'd9, 7'd8 allowlist number. bit31:24 -> address 7'd11 allowlist number bit23:16 -> address 7'd10 allowlist number bit15:8 -> address 7'd9 allowlist number bit7:0 -> address 7'd8 allowlist number

Offset	Register Name	Access	Reset	Description
0x0C	only has one allowlist number. But one allowlist number can be applied to all targets.			Target address 7'd15, 7'd14, 7'd13, 7'd12 allowlist number. bit31:24 -> address 7'd15 allowlist number bit23:16 -> address 7'd14 allowlist number bit15:8 -> address 7'd13 allowlist number bit7:0 -> address 7'd12 allowlist number
0x10				Target address 7'd19, 7'd18, 7'd17, 7'd16 allowlist number. bit31:24 -> address 7'd19 allowlist number bit23:16 -> address 7'd18 allowlist number bit15:8 -> address 7'd17 allowlist number bit7:0 -> address 7'd16 allowlist number
0x14				Target address 7'd23, 7'd22, 7'd21, 7'd20 allowlist number. bit31:24 -> address 7'd23 allowlist number bit23:16 -> address 7'd22 allowlist number bit15:8 -> address 7'd21 allowlist number bit7:0 -> address 7'd20 allowlist number
0x18				Target address 7'd27, 7'd26, 7'd25, 7'd24 allowlist number. bit31:24 -> address 7'd27 allowlist number bit23:16 -> address 7'd26 allowlist number bit15:8 -> address 7'd25 allowlist number bit7:0 -> address 7'd24 allowlist number
0x1c				Target address 7'd31, 7'd30, 7'd29, 7'd28 allowlist number. bit31:24 -> address 7'd31 allowlist number bit23:16 -> address 7'd30 allowlist number bit15:8 -> address 7'd29 allowlist number bit7:0 -> address 7'd28 allowlist number
0x20				Target address 7'd35~7'd32 allowlist number. Bitmap refer to above.
0x24				Target address 7'd39~7'd36 allowlist number.
0x28				Target address 7'd43~7'd40 allowlist number. Bitmap refer to above.
0x2c				Target address 7'd47~7'd44 allowlist number. Bitmap refer to above.
0x30				Target address 7'd51~7'd48 allowlist number. Bitmap refer to above.
0x34				Target address 7'd55~7'd52 allowlist number. Bitmap refer to above.
0x38				Target address 7'd59~7'd56 allowlist number. Bitmap refer to above.
0x3c				Target address 7'd63~7'd60 allowlist number. Bitmap refer to above.
0x40				Target address 7'd67~7'd64 allowlist number. Bitmap refer to above.
0x44				Target address 7'd71~7'd68 allowlist number. Bitmap refer to above.
0x48				Target address 7'd75~7'd72 allowlist number. Bitmap refer to above.
0x4c				Target address 7'd79~7'd76 allowlist number. Bitmap refer to above.
0x50				Target address 7'd83~7'd80 allowlist number. Bitmap refer to above.
0x54				Target address 7'd87~7'd84 allowlist number. Bitmap refer to above.
0x58				Target address 7'd91~7'd88 allowlist number. Bitmap refer

<table>
<tr><th>Offset</th><th>Register Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td></td><td rowspan="10"></td><td rowspan="10"></td><td rowspan="10"></td><td>to above.</td></tr>
<tr><td>0x5c</td><td>Target address 7’d95~7’d92 allowlist number. Bitmap refer to above.</td></tr>
<tr><td>0x60</td><td>Target address 7’d99~7’d96 allowlist number. Bitmap refer to above.</td></tr>
<tr><td>0x64</td><td>Target address 7’d103~7’d100 allowlist number. Bitmap refer to above.</td></tr>
<tr><td>0x68</td><td>Target address 7’d107~7’d104 allowlist number. Bitmap refer to above.</td></tr>
<tr><td>0x6c</td><td>Target address 7’d111~7’d108 allowlist number. Bitmap refer to above.</td></tr>
<tr><td>0x70</td><td>Target address 7’d115~7’d112 allowlist number. Bitmap refer to above.</td></tr>
<tr><td>0x74</td><td>Target address 7’d119~7’d116 allowlist number. Bitmap refer to above.</td></tr>
<tr><td>0x78</td><td>Target address 7’d123~7’d120 allowlist number. Bitmap refer to above.</td></tr>
<tr><td>0x7c</td><td>Target address 7’d127~7’d124 allowlist number. Bitmap refer to above.</td></tr>
<tr><td>0x80</td><td>allowlist number 0</td><td rowspan="31">WO</td><td rowspan="31">256’h0</td><td rowspan="31">There are 256 bits in this range. Every bit represents a command allowlist. For example, bit 9 is the SMBus command 0x9 allowlist. If the bit is 1, command 0x9 is passed in write transaction. If the bit is 0, command 0x9 is blocked in write transaction.

<table>
<tr><th>Offset</th><th>Byte3</th><th>Byte2</th><th>Byte1</th><th>Byte0</th></tr>
<tr><td>0x00</td><td>Bit31:24</td><td>Bit23:16</td><td>Bit15:8</td><td>Bit7:0</td></tr>
<tr><td>0x04</td><td>Bit63:56</td><td>Bit55:48</td><td>Bit47:40</td><td>Bit39:32</td></tr>
<tr><td>0x08</td><td>Bit95:88</td><td>Bit87:80</td><td>Bit79:72</td><td>Bit71:64</td></tr>
<tr><td>0x0c</td><td>Bit127:120</td><td>Bit119:112</td><td>Bit111:104</td><td>Bit103:96</td></tr>
<tr><td>0x10</td><td>Bit159:152</td><td>Bit151:144</td><td>Bit143:136</td><td>Bit135:128</td></tr>
<tr><td>0x14</td><td>Bit191:184</td><td>Bit183:176</td><td>Bit175:168</td><td>Bit167:160</td></tr>
<tr><td>0x18</td><td>Bit223:216</td><td>Bit215:208</td><td>Bit207:200</td><td>Bit199:192</td></tr>
<tr><td>0x1c</td><td>Bit255:248</td><td>Bit247:240</td><td>Bit239:232</td><td>Bit231:224</td></tr>
</table>
</td></tr>
<tr><td>0xA0</td><td>allowlist number 1</td></tr>
<tr><td>0xC0</td><td>allowlist number 2</td></tr>
<tr><td>0xE0</td><td>allowlist number 3</td></tr>
<tr><td>0x100</td><td>allowlist number 4</td></tr>
<tr><td>0x120</td><td>allowlist number 5</td></tr>
<tr><td>0x140</td><td>allowlist number 6</td></tr>
<tr><td>0x160</td><td>allowlist number 7</td></tr>
<tr><td>0x180</td><td>allowlist number 8</td></tr>
<tr><td>0x1A0</td><td>allowlist number 9</td></tr>
<tr><td>0x1C0</td><td>allowlist number 10</td></tr>
<tr><td>0x1E0</td><td>allowlist number 11</td></tr>
<tr><td>0x200</td><td>allowlist number 12</td></tr>
<tr><td>0x220</td><td>allowlist number 13</td></tr>
<tr><td>0x240</td><td>allowlist number 14</td></tr>
<tr><td>0x260</td><td>allowlist number 15</td></tr>
<tr><td>0x280</td><td>allowlist number 16</td></tr>
<tr><td>0x2A0</td><td>allowlist number 17</td></tr>
<tr><td>0x2C0</td><td>allowlist number 18</td></tr>
<tr><td>0x2E0</td><td>allowlist number 19</td></tr>
<tr><td>0x300</td><td>allowlist number 20</td></tr>
<tr><td>0x320</td><td>allowlist number 21</td></tr>
<tr><td>0x340</td><td>allowlist number 22</td></tr>
<tr><td>0x360</td><td>allowlist number 23</td></tr>
<tr><td>0x380</td><td>allowlist number 24</td></tr>
<tr><td>0x3A0</td><td>allowlist number 25</td></tr>
<tr><td>0x3C0</td><td>allowlist number 26</td></tr>
<tr><td>0x3E0</td><td>allowlist number 27</td></tr>
<tr><td>0x400</td><td>allowlist number 28</td></tr>
<tr><td>0x420</td><td>allowlist number 29</td></tr>
<tr><td>0x440</td><td>allowlist number 30</td></tr>
</table>

Offset	Register Name	Access	Reset	Description
0x460	allowlist number 31			
0x480	allowlist number 32			
0x4A0	allowlist number 33			
0x4C0	allowlist number 34			
0x4E0	allowlist number 35			
0x500	allowlist number 36			
0x520	allowlist number 37			
0x540	allowlist number 38			
0x560	allowlist number 39			
0x580	allowlist number 40			
0x5A0	allowlist number 41			
0x5C0	allowlist number 42			
0x5E0	allowlist number 43			
0x600	allowlist number 44			
0x620	allowlist number 45			
0x640	allowlist number 46			
0x660	allowlist number 47			
0x680	allowlist number 48			
0x6A0	allowlist number 49			
0x6C0	allowlist number 50			
0x6E0	allowlist number 51			
0x700	allowlist number 52			
0x720	allowlist number 53			
0x740	allowlist number 54			
0x760	allowlist number 55			
0x780	allowlist number 56			
0x7A0	allowlist number 57			
0x7C0	allowlist number 58			
0x7E0	allowlist number 59			
0x800	Interrupt Enable	R/W	8'bxx0x0000	Bit 0: enable "slave no-acked address" interrupt. • 1: enable • 0: disable Bit 1: enable "slave no-acked command" interrupt. • 1: enable • 0: disable Bit 2: enable "slave no-acked data" interrupt. • 1: enable • 0: disable Bit 3: enable "master no-acked data" interrupt. • 1: enable • 0: disable Bit 5: enable "command is blocked" interrupt. • 1: enable • 0: disable Others: not used, ignore.
0x804	Interrupt Status	R/W1C*	8'bxx0x0000	Bit 0: slave no-acked address • 1: active • 0: inactive Bit 1: slave no-acked command • 1: active

Offset	Register Name	Access	Reset	Description
				• 0: inactive Bit 2: slave no-acked data • 1: active • 0: inactive Bit 3: master no-acked data • 1: active • 0: inactive Bit 5: command is blocked. • 1: active • 0: inactive Others: not used, ignore.
0x808	Interrupt Set	WO	8'bxx0x0000	Bit 0: set "slave no-acked address" interrupt bit. Set 1 to trigger the interrupt. Bit 1: Set "slave no-acked command" interrupt bit. Set 1 to trigger the interrupt. Bit 2: set "slave no-acked data" interrupt bit. Set 1 to trigger the interrupt. Bit 3: set "master no-acked data" interrupt bit. Set 1 to trigger interrupt. Bit 5: set "command is blocked" interrupt bit. Set 1 to trigger interrupt. Others: not use, ignore.
0x808	Blocked Address	RO	32'b0	Bit 31: • 1: address is valid; • 0: address is invalid. bit7:0: blocked address
0x80C	Blocked Command	RO	32'b0	bit31: • 1: command is valid; • 0: command is invalid. bit7:0: blocked command

***Note:** R/W1C, readable and write 1 to clear relevant bit.

8.3.7. Program Flow

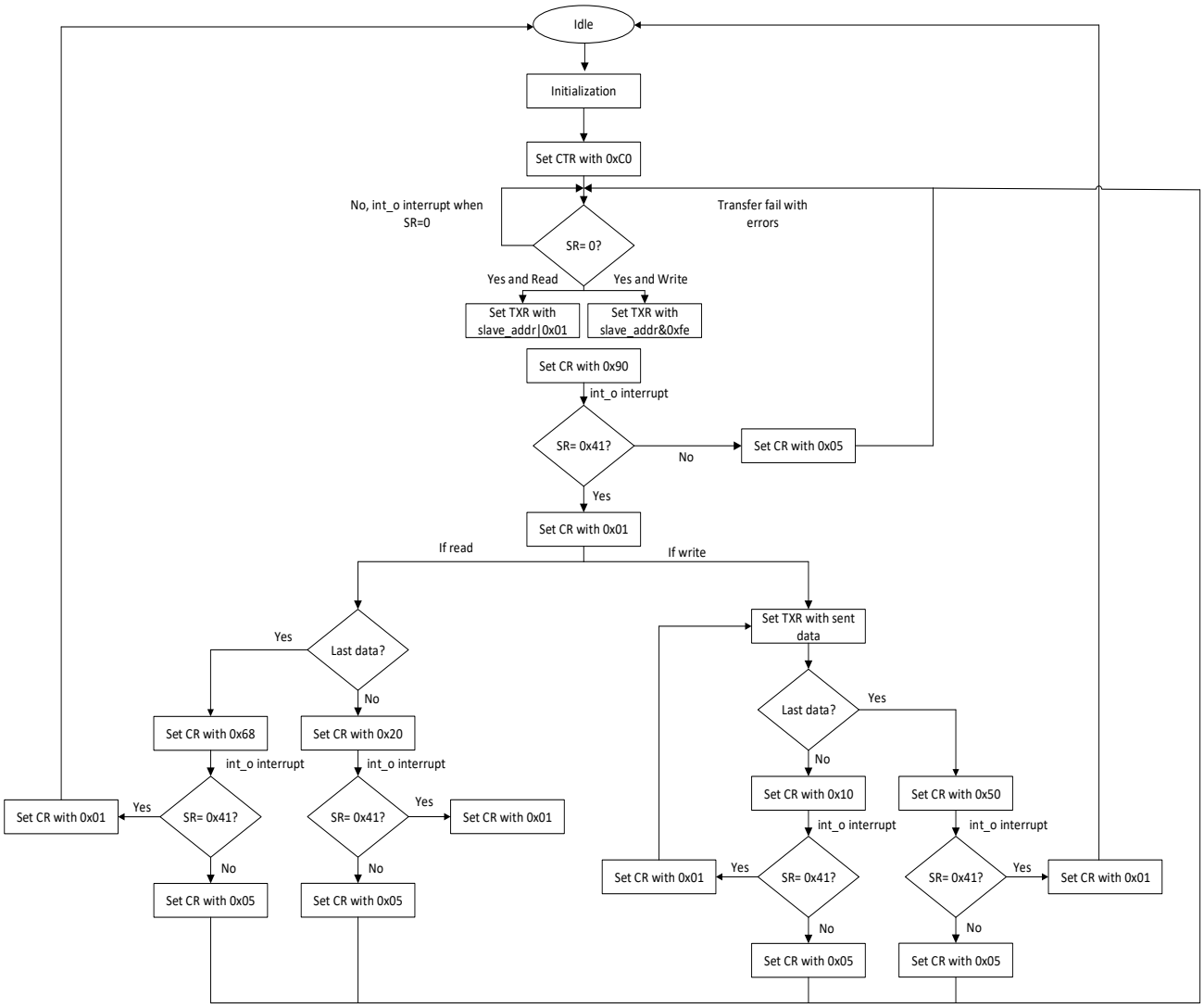


Figure 8.5. SMBus IP Program Flow

8.3.8. Initialization

When utilizing the IP, the register should be configured first. The targets' allowlist number and the allowlist contents should be configured through ABH-Lite port. There are two ways of using the IP: by Interrupt mode or Polling mode. If using interrupt mode, the interrupt enable register bit should be set accordingly.

8.3.9. Interrupt Mode

If using interrupt mode, when an interrupt condition is triggered, an interrupt signal (irq) asserts. The host responses to the interrupt, reads the *Interrupt Status* register and writes 1s to clear it. If this is a blocked command event interrupt, the host reads the Blocked Address and Blocked Command registers to check the blocked address and command. The two registers are valid when bit 31 is set to 1. Note that new status overrides the current status. So the host needs to handle the interrupt in a timely manner.

8.3.10. Polling Mode

The SMBus IP can also be used in polling mode. In this mode, after initialization, the host needs to poll the Interrupt Status register timely. If a status bit is set, the host also needs to write 1s to clear the relevant bit. Similarly, when a “command is blocked” status is set, the host can read the Blocked Address and Blocked Command registers to check the blocked address and command. The two registers are valid when the bit 31 is set to 1. Note that the new status overrides the current status. So the host needs to read out the status in a timely manner.

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, as shown in Table 9.1. The Memory Mapped GPIO are register based and controlled by the CPU. The Virtual GPIO are controlled by the PLD logic, as shown in 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 is ready to read gpio_direction and gpio_output inputs. When a change occurs on the gpio_direction and gpio_output inputs, the ready_gpio signal goes low until the change has been updated and returns to high when the change is 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
0x18	INT_METHOD_REG	R/W	Interrupt Method Configure Register

Offset	Register Name	Access Type	Description
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 (Table 9.4). Reset value is not observable because the 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 (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 is set to 1 (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 is cleared set to 0 (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, it is configured as an output (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), as shown in 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, as shown in 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 (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 (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 (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 1s 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.

1. To read from inputs, read the Read Data Register.
2. To write to outputs, write to the Write Data Register.
3. If an interrupt occurs and you want to clear that interrupt, write 1s to corresponding bits of the Interrupt Status Register.

10. Secure Enclave

The Secure Enclave provides a set of security services for the Mach-NX device. The Secure Enclave has two interfaces for sending and receiving data: a register interface and a FIFO-based High Speed Data Port (HSP). The Secure Enclave provides the following major functions:

- Secure Hash Algorithm (SHA) — 256/384 bits
- Elliptic Curve Digital Signature Algorithm (ECDSA) — generation and verification
- Message Authentication Codes (MACs) — Hash-based MAC (HMAC)
- Elliptic Curve Diffie-Hellman (ECDH) Scheme
- Elliptic Curve Cryptography (ECC) Key Pair Generation — public Key/private Key
- Elliptic Curve Integrated Encryption Scheme (ECIES) encryption/decryption
- True Random Number Generator (TRNG)
- Advanced Encryption Standard (AES) — 128/256 bits
- Authentication controller for configuration engine
- AHB-Lite interface to user logic
- High Speed Port (HSP) for FIFO-based streaming data transfer
- Unique Secure ID

All these security services are provided through Crypto-256/384 API. For detailed information, refer to Lattice Sentry 3.0 PFR IP API Reference (FPGA-TN-02336).

References

For more information, refer to:

- Lattice Propel 1.1 SDK User Guide (FPGA-UG-02115)
- Lattice Propel 1.1 Builder User Guide (FPGA-UG-02116)
- Lattice Diamond 3.12 User Guide
- Lattice Sentry Solution Stack web page
- Mach-NX Devices web page
- Boards, Demos, IP Cores, and Reference Designs for Mach-NX Devices
- Lattice Insights for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

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

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.0, April 2024

Section	Change Summary
All	Production release.

www.latticesemi.com