SMBus Mailbox IP – Lattice Propel Builder

IP Version: 1.2.0

User Guide

FPGA-IPUG-02165-1.2

December 2025

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
AHB-Lite	Advanced High-performance Bus – Lite
API	Application Programming Interface
CPU	Central Processing Unit
EBR	Embedded Block RAM
FIFO	First In First Out
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
I2C	Inter-Integrated Circuit
IP	Intellectual Property
LMMI	Lattice Memory Mapped Interface
LUT	Look-Up Table
MCTP	Management Component Transport Protocol
RISC-V	Reduced Instruction Set Computer – V (Five)
RX	Receive
RTL	Register Transfer Level
SCL	Serial Clock Line
SDA	Serial Data Line
SMBus	System Management Bus
Tx	Transmit

1. Introduction

The System Management Bus (SMBus) is a two-wire interface through which simple system and power management devices can communicate with the rest of the system. The protocol is compatible with the I2C bus protocol and is often found in monitoring power conditions, temperature, and other sensors on a board. While the SMBus is derived from I2C, there are several major differences existing between the specifications of the two buses. The device that initiates the transmission on the SMBus is commonly known as the Controller, while the device being addressed is called the Target.

SMBus bus protocols support many kinds of formats, such as SMBus write byte, SMBus write word, SMBus read byte, SMBus read word, SMBus write block, SMBus read block and so on. See SMBus Specification for more information.

SMBus Mailbox is an SMBus target, which is designed to communicate with SMBus host mailbox. It responds to the standard SMBus Write Byte and Read Byte format messages, as shown in Figure 1.1 and Figure 1.2.

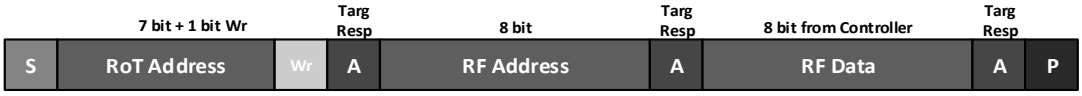

Figure 1.1. SMBus Mailbox Write Byte Message

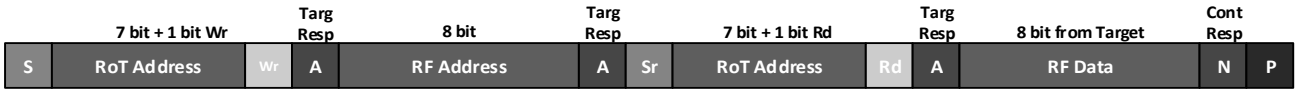

Figure 1.2. SMBus Mailbox Read Byte Message

The MCTP over SMBus/I2C transport binding defines how the MCTP packets are delivered over a physical SMBus or I2C medium using SMBus transactions. All MCTP transactions are based on the SMBus Block Write bus protocol. The first eight bytes make up the packet header. The first three fields—Destination Target Address, Command Code, and Length—map directly to SMBus functional fields. The remaining header and payload fields map to SMBus Block Write *Data Byte* fields, as indicated in Figure 1.1. Hence, the inclusion of the Source Target Address in the header is specified by MCTP rather than SMBus. This is done to facilitate addressing required for establishing communications back to the message originator.

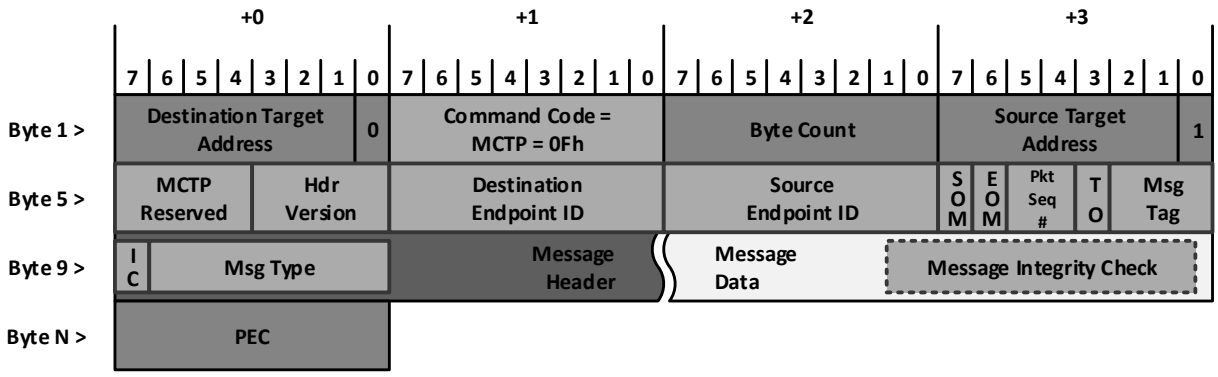

Figure 1.3. MCTP over SMBus Packet Format

1.1. Features

The soft IP has the following features:

- Compatible with SMBus Specification
- Compatible with AHB-Lite Specification
- Supports SMBus controller
- Supports SMBus target
- Supports SMBus mailbox
- Supports MCTP over SMBus
- Supports 7-bit/10-bit addressing modes
- Supports *Fairness arbitration* SMBus arbitration mechanism
- Clock stretching and wait state generation
- Timeout monitor for the Controller
- Interrupt flag generation
- Arbitration lost interrupt, with automatic transfer cancellation
- Bus busy detection
- Integrated glitch filter

1.2. Conventions

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

2. Functional Descriptions

2.1. Overview

The functional diagram of this IP is shown in Figure 2.1, which contains two main blocks: one is the Target Core and the other is the Controller Core.

The Target Core performs two operations:

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

The Controller Core can initiate SMBus transfer as an SMBus Controller.

The SMBus interface is connected to the external bus through SDA/SCL signals. The Controller Core connects through the P1 interface and the Target Core connects through the P0 interface. Therefore, a switch/mux is needed between Controller and Target Cores. The switch is implemented in SMBus interface by the following method:

- If the Controller Core does not initiate a transfer, P0 is routed to the SMBus interface and P1 is switched off. The SMBus target core can then exchange data using the SDA/SCL signals.
- If the Controller Core initiates a transfer, P1 is routed to the SMBus interface and P0 is switched off. The SMBus Controller Core can then exchange data using the SDA/SCL signals.

The Controller Core can initiate SMBus transfer to access other SMBus Targets. The MCTP transfer is also controlled by the Controller Core logic. The Controller Core supports multi-controller on one bus simultaneously. All the controllers obey fairness arbitration rules to avoid any bus conflicts.

The Target Core makes sure the IP serves as an SMBus target device on the bus. The external controller writing messages are routed to RX_FIFO. Reading messages may come from two data sources, TX_FIFO and Register File according to register configuration. The Register File can only be updated by the external host writing through the AHB-Lite port S01. The IP responds to two target addresses: the configured I2C Target Address from IP attributes, which can also be changed dynamically by the host, and the SMBus Device Default Address (7'b1100-001) as specified by the SMBus Specification.

The host accesses the IP through the AHB-Lite port. From the system level, it only has one AHB-Lite subordinate port, which can be connected to the host or another AHB-Lite Interconnect. The host can access Controller Core Register, Target Core Register, and Register File by the different addresses.

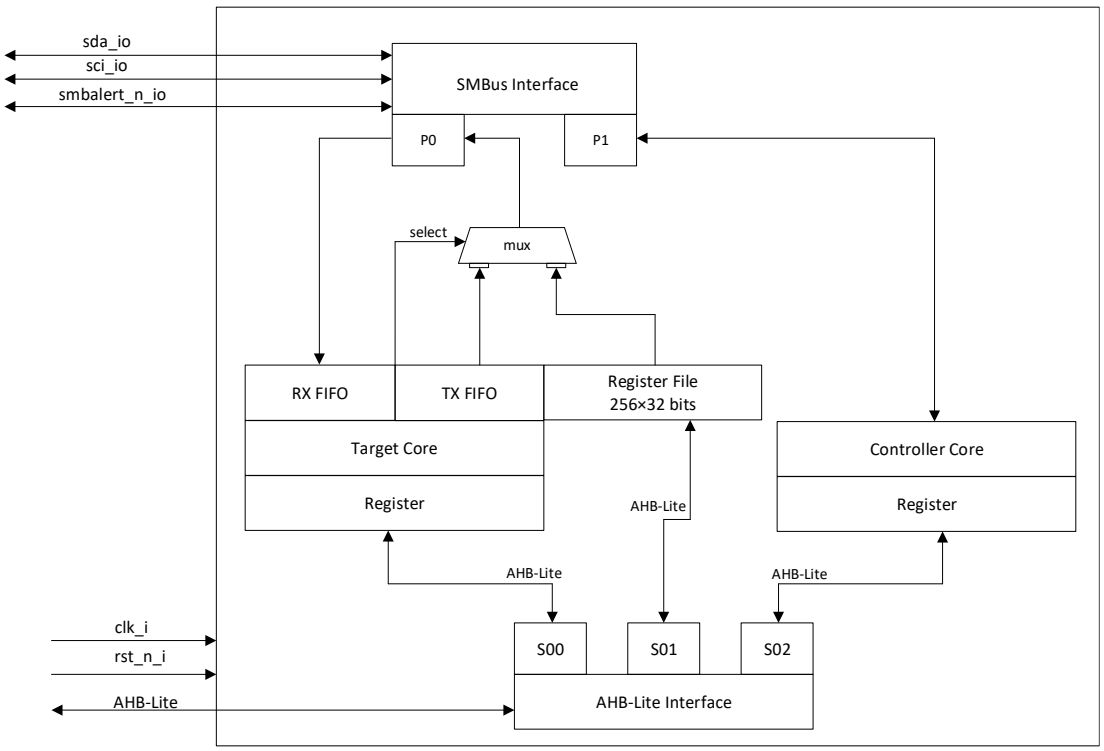

Figure 2.1. SMBus IP Core Functional Diagram

2.2. Signals Description

Table 2.1. Interface Signal Description

Signal Name	Width	Direction	Reset	Description
Clock and Reset				
clk_i	1	input	—	System Clock
rst_n_i	1	input	—	System Reset. The reset assertion can be asynchronous but reset negation should be synchronous. This is an active low signal. When asserted, output ports and registers are forced to their reset values.
AHB-Lite Bus				
ahbl_hsel_slv_i	1	input	—	AHB-Lite Select Signal Indicates that the subordinate 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

Signal Name	Width	Direction	Reset	Description
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: 1 byte 3'b001: 2 bytes 3'b010: 4 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 this signal is HIGH, it indicates a write transfer and when it is LOW, it indicates a read transfer.
ahbl_hready_slv_i	1	input	0	This signal should come from AHB-Lite Interconnect. When it is set to 1, it indicates the previous transfer is complete.
ahbl_hrdata_slv_o	32	output	0	The read data bus
ahbl_hreadyout_slv_o	1	output	0	When this signal is HIGH, it 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	0	When this signal is LOW, it indicates that the transfer status is OKAY. When it is HIGH, it indicates that the transfer status is ERROR.
Interrupt Signal				
int_o	1	output	0	Interrupt to host (CPU). The reset value is 1’b0.
SMBus Signal				
scl_io	1	inout	Reset value is weak high (pull-up).	SMBus Serial Clock
sda_io	1	inout	Reset value is weak high (pull-up).	SMBus data signal
smbalert_n_o	1	output	0	SMBus alert (active low)

2.3. Attributes

The configurable attributes of the IP Core are shown in Table 2.2. The attributes can be configured through the IP Catalog’s Module/IP Wizard of the Lattice Propel™ Builder.

Table 2.2. Attributes Description

Parameter	Value Range	Description
Enable Controller Function	Checkbox: Check, Uncheck	Switch on/off controller function. Check: enable the Controller Core; Uncheck: disable the Controller Core. Target-only mode.
SMBus Mailbox Base Address	0–{AHB-Lite Address assignment designation}	Specify IP base address on AHB-Lite bus.
AHB-Lite Address bus number of bits	[1–32]	AHB-Lite bus address width
AHB-Lite Data Bus number of bits	[8–32]	AHB-Lite bus data width
Addressing Mode	7-bit, 10-bit	7-bit or 10-bit addressing selection
I2C Target Address: 0x	7-bit: 0x0–0x7f, 10-bit: 0x0–3ff	As a target device, it is the target address for external controller to address.
Device Architecture	['LFMNX', 'MachXO3D']	FPGA Architecture
System Clock Frequency (MHz)	[40–100]	System Clock Frequency
I2C Frequency (kHz)	[100, 400, 1000]	I2C Bus Frequency

2.4. Register Description

The register address map, shown in Table 2.3, 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 2.3. Registers Address Map

<table>
<tr><th>Offset</th><th>Register Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td colspan="5">AHB-Lite S00: Each register is 32-bit wide wherein the upper bits [31:8] are reserved and the lower 8 bits [7:0] are used.</td></tr>
<tr><td>0x00</td><td>RD_DATA_REG</td><td>RO</td><td>Not guaranteed.</td><td>Read Data Register</td></tr>
<tr><td>0x00</td><td>WR_DATA_REG</td><td>WO</td><td>Not guaranteed.</td><td>Write Data Register</td></tr>
<tr><td>0x04</td><td>SLVADR_L_REG</td><td>R/W</td><td>[7] RSVD
[6:0] I2C Target Address[6:0]</td><td>Target Address Lower Register, same as the I2C Target Address attribute.</td></tr>
<tr><td>0x08</td><td>SLVADR_H_REG</td><td>R/W</td><td>[7:3] RSVD
[2:0] I2C Target Address[9:7]</td><td>Target Address Higher Register, same as the I2C Target Address attribute.</td></tr>
<tr><td>0x0C</td><td>CONTROL_REG</td><td>R/W</td><td>[7:6] RSVD
[5:1] 0
[0] See Addressing Mode in Table 2.2.</td><td>Control Register
<table>
<tr><th>Field</th><th>Name</th><th>Access</th></tr>
<tr><td>[7:6]</td><td>RSVD</td><td>RSVD</td></tr>
<tr><td>[5]</td><td>dat_src_sw</td><td>R/W</td></tr>
<tr><td>[4]</td><td>nack_data</td><td>R/W</td></tr>
<tr><td>[3]</td><td>nack_addr</td><td>R/W</td></tr>
<tr><td>[2]</td><td>reset</td><td>WO</td></tr>
<tr><td>[1]</td><td>clk_stretch_en</td><td>R/W</td></tr>
<tr><td>[0]</td><td>addr_10bit_en</td><td>R/W</td></tr>
</table></td></tr>
<tr><td>0x10</td><td>TGT_BYTE_CNT_REG</td><td>R/W</td><td>8’h00</td><td>Target Byte Count Register</td></tr>
</table>

<table>
<tr><th>Offset</th><th>Register Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td>0x14</td><td>INT_STATUS1_REG</td><td>RW1C</td><td>8’h00</td><td>Interrupt Status First Register
<table>
<tr><th>Field</th><th>Name</th><th>Access</th></tr>
<tr><td>[7]</td><td>tr_cmp_int</td><td>RW1C</td></tr>
<tr><td>[6]</td><td>stop_det_int</td><td>RW1C</td></tr>
<tr><td>[5]</td><td>tx_fifo_full_int</td><td>RW1C</td></tr>
<tr><td>[4]</td><td>tx_fifo_aempty_int</td><td>RW1C</td></tr>
<tr><td>[3]</td><td>tx_fifo_empty_int</td><td>RW1C</td></tr>
<tr><td>[2]</td><td>rx_fifo_full_int</td><td>RW1C</td></tr>
<tr><td>[1]</td><td>rx_fifo_afull_int</td><td>RW1C</td></tr>
<tr><td>[0]</td><td>rx_fifo_ready_int</td><td>RW1C</td></tr>
</table></td></tr>
<tr><td>0x18</td><td>INT_ENABLE1_REG</td><td>R/W</td><td>8’h00</td><td>Interrupt Enable First Register
<table>
<tr><th>Field</th><th>Name</th><th>Access</th></tr>
<tr><td>[7]</td><td>tr_cmp_en</td><td>R/W</td></tr>
<tr><td>[6]</td><td>stop_det_en</td><td>R/W</td></tr>
<tr><td>[5]</td><td>tx_fifo_full_en</td><td>R/W</td></tr>
<tr><td>[4]</td><td>tx_fifo_aempty_en</td><td>R/W</td></tr>
<tr><td>[3]</td><td>tx_fifo_empty_en</td><td>R/W</td></tr>
<tr><td>[2]</td><td>rx_fifo_full_en</td><td>R/W</td></tr>
<tr><td>[1]</td><td>rx_fifo_afull_en</td><td>R/W</td></tr>
<tr><td>[0]</td><td>rx_fifo_ready_en</td><td>R/W</td></tr>
</table></td></tr>
<tr><td>0x1C</td><td>INT_SET1_REG</td><td>WO</td><td>8’h00</td><td>Interrupt Set First Register
<table>
<tr><th>Field</th><th>Name</th><th>Access</th></tr>
<tr><td>[7]</td><td>tr_cmp_set</td><td>WO</td></tr>
<tr><td>[6]</td><td>stop_det_set</td><td>WO</td></tr>
<tr><td>[5]</td><td>tx_fifo_full_set</td><td>WO</td></tr>
<tr><td>[4]</td><td>tx_fifo_aempty_set</td><td>WO</td></tr>
<tr><td>[3]</td><td>tx_fifo_empty_set</td><td>WO</td></tr>
<tr><td>[2]</td><td>rx_fifo_full_set</td><td>WO</td></tr>
<tr><td>[1]</td><td>rx_fifo_afull_set</td><td>WO</td></tr>
<tr><td>[0]</td><td>rx_fifo_ready_set</td><td>WO</td></tr>
</table></td></tr>
<tr><td>0x20</td><td>INT_STATUS2_REG</td><td>RW1C</td><td>8’h00</td><td>Interrupt Status Second Register
<table>
<tr><th>Field</th><th>Name</th><th>Access</th></tr>
<tr><td>[7]</td><td>reserved</td><td>RSVD</td></tr>
<tr><td>[6]</td><td>scl_h_to</td><td>RW1C</td></tr>
<tr><td>[5]</td><td>scl_l_to</td><td>RW1C</td></tr>
<tr><td>[4]</td><td>SR_check_value</td><td>RW1C</td></tr>
<tr><td>[3]</td><td>SR_check_valid</td><td>RW1C</td></tr>
<tr><td>[2]</td><td>arp_cmd_det</td><td>RW1C</td></tr>
<tr><td>[1]</td><td>stop_err_int</td><td>RW1C</td></tr>
<tr><td>[0]</td><td>start_err_int</td><td>RW1C</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Offset</th><th>Register Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td>0x24</td><td>INT_ENABLE2_REG</td><td>R/W</td><td>[7:2] RSVD
[1:0] 2’b00</td><td>Interrupt Enable Second Register<table><tr><th>Field</th><th>Name</th><th>Access</th></tr><tr><td>[7]</td><td>reserved</td><td>RSVD</td></tr><tr><td>[6]</td><td>scl_h_to_en</td><td>R/W</td></tr><tr><td>[5]</td><td>scl_l_to_en</td><td>R/W</td></tr><tr><td>[4]</td><td>reserved</td><td>RSVD</td></tr><tr><td>[3]</td><td>SR_valid_en</td><td>R/W</td></tr><tr><td>[2]</td><td>arp_cmd_en</td><td>R/W</td></tr><tr><td>[1]</td><td>stop_err_en</td><td>R/W</td></tr><tr><td>[0]</td><td>start_err_en</td><td>R/W</td></tr></table></td></tr>
<tr><td>0x28</td><td>INT_SET2_REG</td><td>WO</td><td>[7:2] RSVD
[1:0] 2’b00</td><td>Interrupt Set Second Register<table><tr><th>Field</th><th>Name</th><th>Access</th></tr><tr><td>[7]</td><td>reserved</td><td>RSVD</td></tr><tr><td>[6]</td><td>sch_h_to_set</td><td>WO</td></tr><tr><td>[5]</td><td>sch_l_to_set</td><td>WO</td></tr><tr><td>[4]</td><td>reserved</td><td>RSVD</td></tr><tr><td>[3]</td><td>SR_valid_set</td><td>WO</td></tr><tr><td>[2]</td><td>arp_cmd_set</td><td>WO</td></tr><tr><td>[1]</td><td>stop_err_set</td><td>WO</td></tr><tr><td>[0]</td><td>start_err_set</td><td>WO</td></tr></table></td></tr>
<tr><td>0x2C</td><td>FIFO_STATUS_REG</td><td>RO</td><td>[7:6] RSVD
[5:0] 6’b011001</td><td>FIFO Status Register<table><tr><th>Field</th><th>Name</th><th>Access</th></tr><tr><td>[7:6]</td><td>RSVD</td><td>RSVD</td></tr><tr><td>[5]</td><td>tx_fifo_full</td><td>RO</td></tr><tr><td>[4]</td><td>tx_fifo_aempty</td><td>RO</td></tr><tr><td>[3]</td><td>tx_fifo_empty</td><td>RO</td></tr><tr><td>[2]</td><td>rx_fifo_full</td><td>RO</td></tr><tr><td>[1]</td><td>rx_fifo_afull</td><td>RO</td></tr><tr><td>[0]</td><td>rx_fifo_empty</td><td>RO</td></tr></table></td></tr>
<tr><td>0x2C</td><td>FLUSH_FIFO</td><td>WO</td><td>[7:2] RSVD
[1:0] 2’b0</td><td>Interrupt Enable Second Register<table><tr><th>Field</th><th>Name</th><th>Access</th></tr><tr><td>[7:2]</td><td>reserved</td><td>RSVD</td></tr><tr><td>[1]</td><td>rxfifo_flush</td><td>WO</td></tr><tr><td>[0]</td><td>txfifo_flush</td><td>WO</td></tr></table></td></tr>
<tr><td>0x30</td><td>SMB_CONTROL_REG</td><td>R/W</td><td>[7:1] RSVD
[0] 1’b0</td><td>SMBus Control and Status Register
[7:1] RSVD
[0] smb_alert: Transmits the alert interrupt to SMBus Controller.
1’b0 – No interrupt to Controller.
1’b1 – SMBus target sends alert interrupt to Controller.</td></tr>
<tr><td>0x34-
0x3C</td><td>Reserved</td><td>RSVD</td><td>RSVD</td><td>Reserved
Write access is ignored and 0 is returned on read access.</td></tr>
</table>

<table>
<tr><th>Offset</th><th>Register Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td colspan="5">AHB-Lite S01</td></tr>
<tr><td>0x2000-
0x23ff</td><td>Registers File</td><td>R/W</td><td>All ‘0’s</td><td>Registers File 256x32bits, SMBus Mailbox Registers File.<table><tr><th>Offset</th><th>Dword Number</th></tr><tr><td>0x00</td><td>0</td></tr><tr><td>0x04</td><td>1</td></tr><tr><td>0x08</td><td>2</td></tr><tr><td>…</td><td>…</td></tr><tr><td>0x3fc</td><td>255</td></tr></table></td></tr>
<tr><td colspan="5">AHB-Lite S02: each register is 32-bit wide wherein the upper bits [31:8] are reserved and the lower 8 bits [7:0] are used.</td></tr>
<tr><td>0x400</td><td>PRERlo</td><td>R/W</td><td>[31:8] RSVD
[7:0] 8’hff</td><td>Clock prescale register low-byte
5×SCL frequency = clk_i / (PRERhi<<8 + PRERlo)</td></tr>
<tr><td>0x404</td><td>PRERhi</td><td>R/W</td><td>[31:8] RSVD
[7:0] 8’hff</td><td>Clock prescale register high-byte
5×SCL frequency = clk_i / (PRERhi<<8 + PRERlo)</td></tr>
<tr><td>0x408</td><td>CTR</td><td>R/W</td><td>[31:8] RSVD
[7:6] 2’h00
[5:0] RSVD</td><td>Control Register<table><tr><th>Field</th><th>Description</th></tr><tr><td>[7]</td><td>EN –Controller Core enable bit.
1 = Core is enabled
0 = Core is disabled</td></tr><tr><td>[6]</td><td>IEN – interrupt enable bit.
1 = Interrupt is enabled
0 = Interrupt is disabled</td></tr></table></td></tr>
<tr><td>0x40c</td><td>TXR</td><td>WO</td><td>[31:8] RSVD
[7:0] 8’h00</td><td>Transmit Register<table><tr><th>Field</th><th>Description</th></tr><tr><td>[7:1]</td><td>Next byte to be transmitted via SMBus Controller Core</td></tr><tr><td>[0]</td><td>a) The byte’s LSB.
b) RW bit during target address transfer
1 = Reading from target
0 = Writing to target</td></tr></table></td></tr>
<tr><td>0x40c</td><td>RXR</td><td>RO</td><td>[31:8] RSVD
[7:0] 8’h00</td><td>Receive Register
Last byte received through the SMBus Controller Core.</td></tr>
<tr><td>0x410</td><td>CR</td><td>WO</td><td>[31:8] RSVD
[7:0] 8’h00</td><td>Command Register<table><tr><th>Field</th><th>Name</th></tr><tr><td>[7]</td><td>STA – Generate (repeated) start condition</td></tr><tr><td>[6]</td><td>STO – Generate stop condition</td></tr><tr><td>[5]</td><td>RD – Read from target</td></tr><tr><td>[4]</td><td>WR – Write to target</td></tr><tr><td>[3]</td><td>ACK, when a receiver sends ACK (ACK = 0) or NACK (ACK = 1).</td></tr><tr><td>[2]</td><td>When set, clears the timeout status</td></tr><tr><td>[1]</td><td>Reserved</td></tr><tr><td>[0]</td><td>IACK – Interrupt acknowledge. When set, clears a pending interrupt.</td></tr></table></td></tr>
</table>

<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 – Received 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 drive 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 25 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.</td></tr>
</table></td></tr>
</table>

Table 2.4. Access Type Definition

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

2.4.1. Target Core Register

AHB-Lite S00 is the port to access Target Core Register. The offset address is 0. The detailed description for every register is shown below.

The RD_DATA_REG and WR_DATA_REG share the same offset. The Write access to this offset goes to WR_DATA_REG, while the Read access goes to RD_DATA_REG. Write Data Register is the interface to Transmit FIFO (TX_FIFO). Writing to WR_DATA_REG pushes a word to Transmit FIFO (TX_FIFO). When writing to WR_DATA_REG, the host should ensure that Transmit FIFO (TX_FIFO) is not full. This can be done by reading FIFO_STATUS_REG. Data is popped from WR_DATA_REG during I2C read transaction. When reset is performed, the contents of Transmit FIFO (TX_FIFO) are not reset but the FIFO control logic is reset. Thus, the content is not guaranteed after reset. The Read Data register is the interface to Receive FIFO (RX_FIFO). After a data is received from I2C bus during I2C write transaction, the received data is pushed to Receive FIFO (RX_FIFO). Reading from RD_DATA_REG pops a word from Receive FIFO (RX_FIFO). The host should ensure that Receive FIFO (RX_FIFO) has data before reading RD_DATA_REG, data is not guaranteed when this register is read during Receive FIFO (RX_FIFO) empty condition. On the other hand, if Receive FIFO (RX_FIFO) is full but I2C target continues to receive data, new data is lost. The Read FIFO_STATUS_REG to determine the status of Receive FIFO (RX_FIFO). Similar to Transmit FIFO (TX_FIFO), the reset value of Receive FIFO (RX_FIFO) is also not guaranteed after reset.

The Target Address Lower Register (TARGET_ADDRL_REG) 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 seven bits of the Target address. The Target Address Higher Register (TARGET_ADDRH_REG) is the upper three bits of 10-bit target address. This is not used in 7-bit addressing mode. The reset values of TARGET_ADDRL_REG and TARGET_ADDRH_REG is set by the I2C Target Address attribute, as shown in Table 2.2.

Each bit of the Control Register (CONTROL_REG) controls the behavior of the Target Core.

- 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 address.
- reset
  Reset. Resets I2C Target IP Core for one clock cycle. The registers and LMMI interface are not affected by this reset. This is write-only bit because it has the auto-clear feature. It is cleared to 1’b0 after one clock cycle.
  - 1’b0 – No action.
  - 1’b1 – Resets I2C Target IP Core.
- clk_stretch_en
  Clock Stretch Enable. Enables clock stretching on ACK bit of data.
  - 1’b0 – I2C Target IP Core releases SCL signal.
  - 1’b1 – I2C Target IP 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 IP Core rejects the 10-bit I2C address. It sends NACK.
  - 1’b1 – I2C Target IP Core responds to 10-bit I2C address. If TARGET_ADDRH_REG is 3’h0, it also responds to the 7-bit address.

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

The Interrupt Status Register (INT_STATUS1_REG and INT_STATUS2_REG) contains all the interrupts currently pending in the 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 description of Interrupt Enable Registers (INT_ENABLE1_REG, INT_ENABLE2_REG) below 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.

The corresponding bits of INT_STATUS1_REG are shown below:

- tr_cmp_int
  Transfer Complete Interrupt Status. This interrupt status bit asserts when the number of bytes transferred in I2C interface is equal to TGT_BYTE_CNT_REG.
  - 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 (TX_FIFO) Full Interrupt Status. This interrupt status bit asserts when Transmit FIFO (TX_FIFO) changes from not full state to full state.
  - 1'b0 – No interrupt.
  - 1'b1 – Interrupt pending.
- tx_fifo_aempty_int
  Transmit FIFO (TX_FIFO) Almost Empty Interrupt Status. This interrupt status bit asserts when the amount of data words in Transmit FIFO (TX_FIFO) changes from 'TX FIFO Almost Empty Flag' – 1 to 'TX FIFO Almost Empty Flag'.
  - 1'b0 – No interrupt.
  - 1'b1 – Interrupt pending.
- tx_fifo_empty_int
  Transmit FIFO (TX_FIFO) Empty Interrupt Status. This interrupt status bit asserts when the last data in Transmit FIFO (TX_FIFO) is popped-out, causing the FIFO to become empty.
  - 1'b0 – No interrupt.
  - 1'b1 – Interrupt pending.
- rx_fifo_full_int
  Receive FIFO (RX_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 (RX_FIFO) Almost Full Interrupt Status. This interrupt status bit asserts when the amount of data words in Receive FIFO (RX_FIFO) changes from 'RX FIFO Almost Full Flag' – 1 to 'RX FIFO Almost Full Flag'.
  - 1'b0 – No interrupt.
  - 1'b1 – Interrupt pending.

- rx_fifo_ready_int
  Receive FIFO (RX_FIFO) Ready Interrupt Status. This interrupt status bit asserts when Receive FIFO (RX_FIFO) is empty and receives a data word from I2C interface.
  - 1'b0 – No interrupt.
  - 1'b1 – Interrupt pending.

The corresponding bit of INT_STATUS2_REG is shown below:

- scl_h_to
  Timeout flag when SCL and SDA lines have been high for 50 us in transfer mode.
  - 1'b0 – No interrupt.
  - 1'b1 – Interrupt pending.
- scl_l_to
  Timeout flag when SCL line has been low for 25 ms.
  - 1'b0 – No interrupt.
  - 1'b1 – Interrupt pending.
- arp_cmd_det
  Flag when 7'h61 is addressed.
  - 1'b0 – No interrupt.
  - 1'b1 – Interrupt pending.
- SR_check_valid: repeat start check valid.
- SR_check_value: repeat start value.

  When SR_check_valid is valid (=1), then you know whether repeat start happens or not according to SR_check_value (=1, happen; =0 does not happen).
- stop_err_int

  STOP Condition Error Interrupt Status. This interrupt status bit asserts after detecting a STOP condition when it is not expected. A 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 I2C bus is idle and after receiving an ACK or a NACK (repeated START condition).
  - 1'b0 – No interrupt.
  - 1'b1 – Interrupt pending.

INT_ENABLE1_REG/INT_ENABLE2_REG corresponds to interrupts status bits in INT_STATUS1_REG and INT_STATUS2_REG. They do not affect the contents of the INT_STATUS1_REG and INT_STATUS2_REG. If one of the INT_STATUS1_REG/INT_STATUS2_REG bits assert and the corresponding bit of INT_ENABLE1_REG/ INT_ENABLE2_REG is 1'b1, the interrupt signal int_o asserts.

The corresponding bits of INT_ENABLE1_REG are shown below:

- 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.

The corresponding bits of INT_ENABLE2_REG are shown below:

- scl_h_to_en

  Enable interrupt when SCL and SDA line have been high for 50 us in transfer mode.
  - 1'b0 – Interrupt disabled.
  - 1'b1 – Interrupt enabled.
- scl_l_to_en

  Enable interrupt when SCL line has been low for 25 ms.
  - 1'b0 – Interrupt disabled.
  - 1'b1 – Interrupt enabled.
- SR_valid_en

  Enable interrupt when repeat start check finish.
  - 1'b0 – Interrupt disabled.
  - 1'b1 – Interrupt enabled.
- arp_cmd_en

  Enable interrupt when 7'h61 is addressed.
  - 1'b0 – Interrupt disabled.
  - 1'b1 – Interrupt enabled.
- 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.

INT_ SET1_REG/INT_SET2_REG corresponds to interrupts status bits in INT_STATUS1_REG and INT_STATUS2_REG. 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.

The corresponding bit of INT_ SET1_REG shows below:

- 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 corresponds to Receive FIFO Ready Interrupt Status.
  - 1'b0 – No action.
  - 1'b1 – Asserts INT_STATUS1_REG.rx_fifo_ready_int.

The corresponding bits of INT_ SET2_REG are shown below:

- scl_h_to_set
  SCL and SDA line high timeout interrupt set. Interrupt set bit corresponds to scl_h_to bit in INT_STATUS2_REG.
  - 0 – No action.
  - 1 – Asserts INT_STATUS2_REG. scl_h_to.
- scl_l_ to_set
  SCL line low timeout interrupt set. Interrupt set bit corresponds to scl_l_to bit in INT_STATUS2_REG.
  - 0 – No action.
  - 1 – Asserts INT_STATUS2_REG. scl_l_to.
- arp_cmd_det
  7'h61 is addressed interrupt set. Interrupt set bit corresponds to arp_cmd_det bit in INT_STATUS2_REG.
  - 0 – No action.
  - 1 – Asserts INT_STATUS2_REG. arp_cmd_det.
- 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.

FIFO Status Register reflects the status of Transmit FIFO and Receive FIFO as shown below.

- 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 the TX FIFO Almost Empty Flag attribute.
  - 1'b1 – Data words in Transmit FIFO is less than or equal to the 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 the RX FIFO Almost Full Flag attribute.
  - 1'b1 – Data words in Receive FIFO is greater than or equal to the 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.

The corresponding bits of FLUSH_FIFO are shown below:

- rxfifo_flush
  Flush RX FIFO data.
  - 0 – No action.
  - 1 – Flush RX FIFO data to empty.
- txfifo_flush
  Flush TX FIFO data.
  - 0 – No action.
  - 1 – Flushes TX FIFO data to empty.

2.4.2. Registers 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 messages to the RX_FIFO. The host reads that message data from the RX_FIFO and write to the Register File through the AHB-Lite port S01. The Host can read and write to the Register File from the ABH_Lite port S01. The offset address is 0x2000.

2.4.3. Controller Core Register

AHB-Lite S02 is the port to access Controller Core Register. The offset address is 0x400.

The prescale register (offset = 0x00 and 0x04) is used to prescale the SCL clock line based on the system 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 [clk_i 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.

Only two bits of the control register (offset = 0x08) are 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. If the bit is set, the Target Core to SMBus interface route is disabled.

The transmit register and the receive register share the same address (offset = 0x0C) 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 = 0x10). 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.

2.5. SMBus Alert Signal

The SMBus alert signal provides interrupt signal to the SMBus controller when pulled Low. A target device can signal the controller through smbalert_n_o 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 pulled smbalert_n_o 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 smbalert_n_o low, the highest priority device (lowest address) device wins the communication rights.

After receiving an acknowledge (ACK) from the controller in response to its address, the device stops pulling down the smbalert_n_o signal. If the controller still sees the smbalert_n_o low when the message transfer is complete, the same process repeats again. The SMBus target core 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 smbdat_io. When there is match, the smb_alert register bit is cleared and the controller generates an interrupt signal to the host.

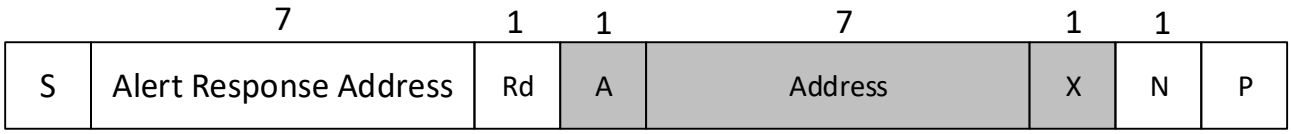

Figure 2.2. 7-bit Addressable Device Response

3. Program Flow

The SMBus mailbox IP can be used as an SMBus controller and SMBus target simultaneously. But the SMBus controller function also can be disabled by unchecking the Enable Controller function attribute box when configuring the IP in Lattice Propel Builder.

If both SMBus controller and SMBus target are enabled, when SMBus controller initiates a transfer, SMBus target logic is halted, and it cannot receive external controller's messages. When SMBus controller logic is complete and halts, the SMBus target logic wakes up and is available for receiving messages from any controller.

The SMBus mailbox IP needs to be initialized for both SMBus controller and SMBus target core blocks to enable normal operation.

3.1. SMBus Target Core Initialization

To perform initialization, load the following appropriate registers of the Target Core:

- TARGET_ADDRL_REG, TARGET_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, that 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.

3.2. SMBus Controller Core 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).

3.3. SMBus Target Core Operation Flow

3.3.1. Data Transfer in Response to External Controller Read

As mentioned, the two target address 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.

3.3.1.1. Normal SMBus Target Device Read Data Transfer

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

To perform data transfer in response to read request of SMBus Controller:

1. Write data to WR_DATA_REG, amounting to ≤ FIFO Depth.
2. Enable only Transfer Complete Interrupt. If transmit data > FIFO Depth, also enable TX FIFO Almost Empty interrupt if there is 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 occurs.
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.

3.3.1.2. 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 1.2.

3.3.2. Data Transfer in Response to External Controller Write

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

3.3.2.1. Normal SMBus Target Device Write Data Transfer

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

To perform data transfer in response to write request of the SMBus Controller:

1. Enable only Transfer Complete Interrupt. If data to receive > FIFO Depth, also enable RX FIFO Almost Full interrupt. If data to receive ≤ 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 the 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 also okay to clear all interrupts.
4. Read data byte from RD_DATA_REG, amounting to less than or equal to FIFO Depth – TX FIFO Almost Empty Setting.
5. If there are remaining data to receive, go back to Step 2. Otherwise, disable RX FIFO Almost Full Interrupt.
6. Wait for Transfer Complete Interrupt. If polling mode is desired, read INT_STATUS1_REG until tr_cmp_int asserts.

   If interrupt mode is desired, simply wait for the 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.

3.3.2.2. 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 write it to the Register File according to the Register File address. The data format is shown in Figure 1.1.

3.4. SMBus Controller Core Operation Flow

Figure 3.1 shows the SMBus controller program flow in interrupt mode. The Controller Core 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.

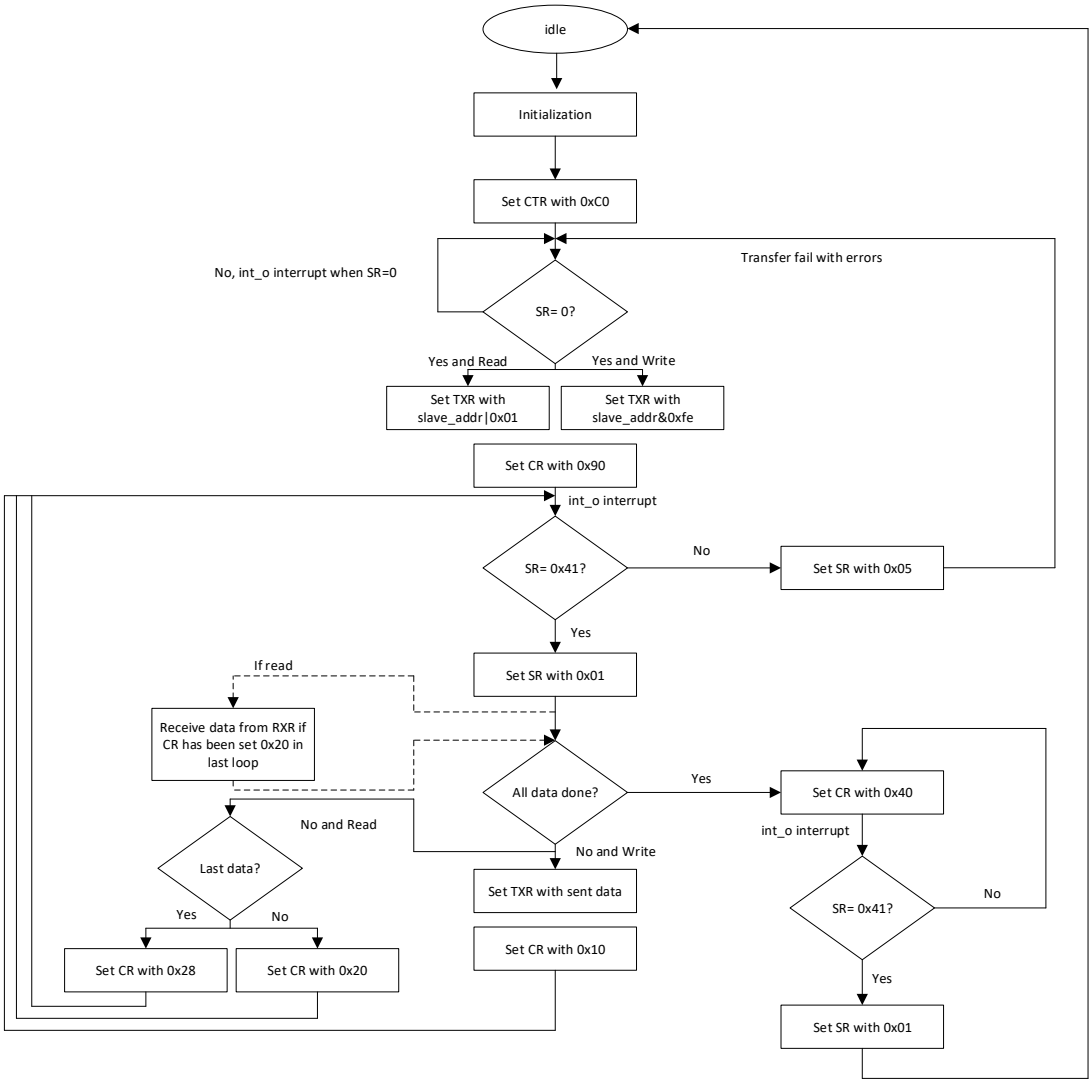

Figure 3.1. SMBus Controller Program Flow Interrupt Mode

3.4.1. 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 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 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 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 is another data to write, go back to step 6.

9. When all the bytes have been sent, write 0x40 to the command register (CR) through the AHB-Lite bus to 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.

3.4.2. 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 no error and have another data to read, go back to step 6.
9. When the read operation is finished, write 0x40 to the command register (CR) through the AHB-Lite bus to 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. Bit6 is set when other controllers use the bus at this time, otherwise it also should be 0. 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 9.

3.5. C Code API

Refer to the IP driver for the details.

4. Generating the SMBus Mailbox IP

This section provides information on how to generate the SMBus Mailbox IP Core module using Lattice Propel Builder.

To generate the SMBus Mailbox IP Core module:

1. In Lattice Propel Builder, create a new design. Select the SMBus Mailbox IP in IP Catalog.
2. Enter the component name as shown in Figure 4.1. Click **Next**.

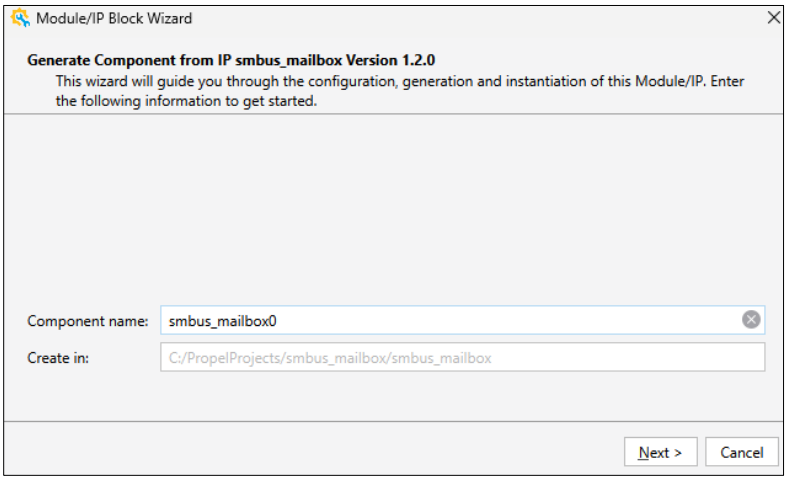

Figure 4.1. Module/IP Block Wizard

3. Configure the parameters as shown in Figure 4.2. Click **Generate**.

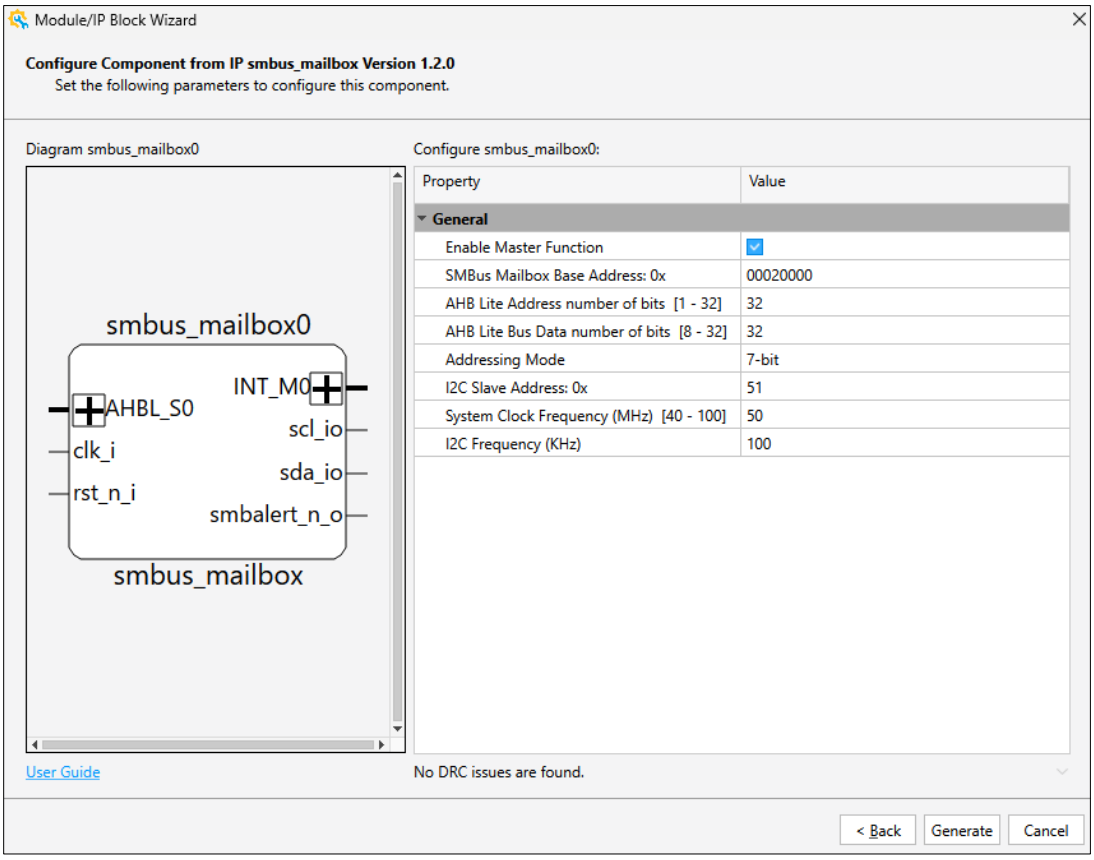

Figure 4.2. Configuring Parameters

4. Verify the information. Click **Finish**.

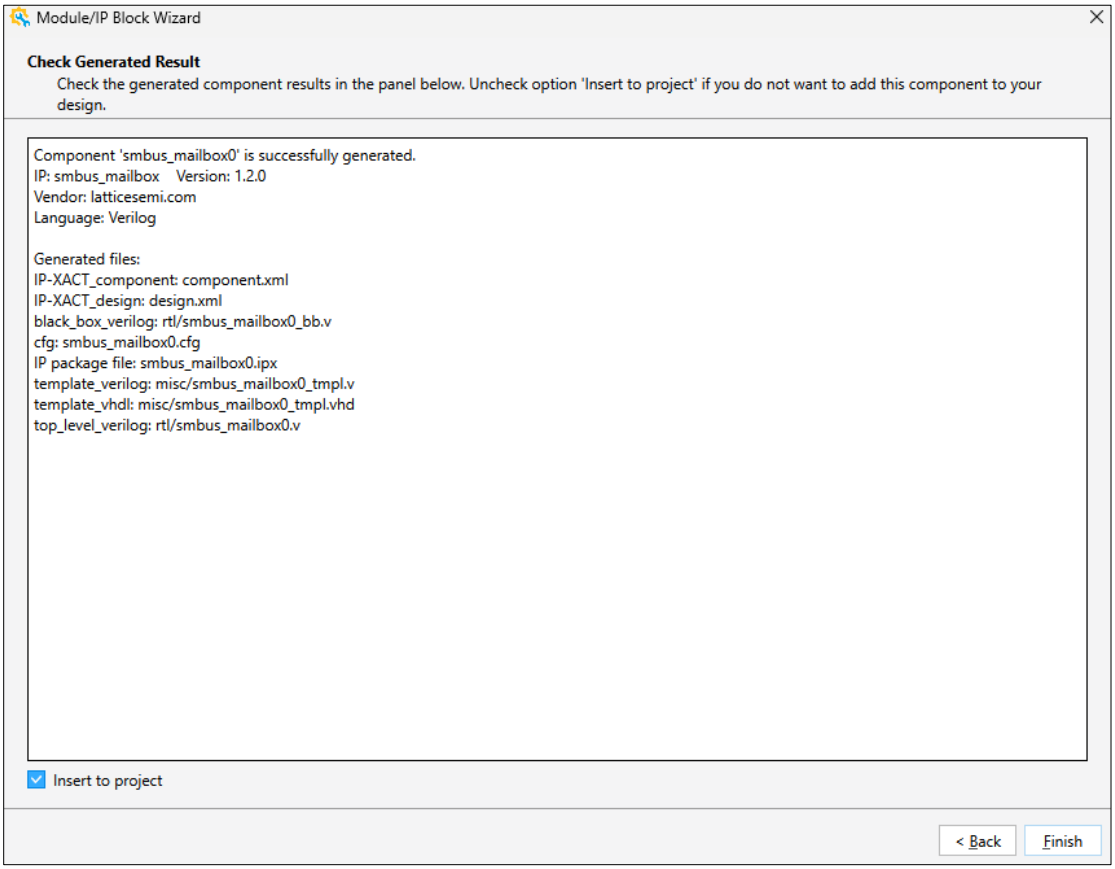

Figure 4.3. Verifying Results

5. Confirm or modify the module instance name. Click **OK**.

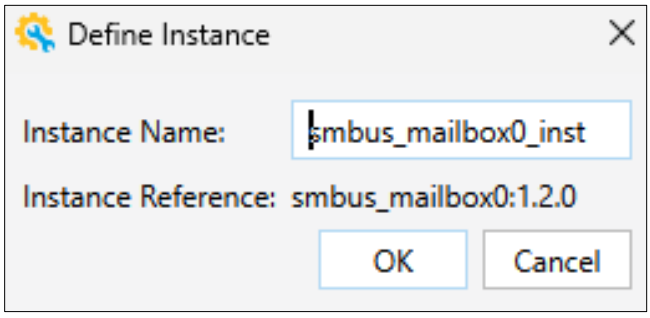

Figure 4.4. Specifying Instance Name

The CPU IP instance is successfully generated, as shown in Figure 4.5.

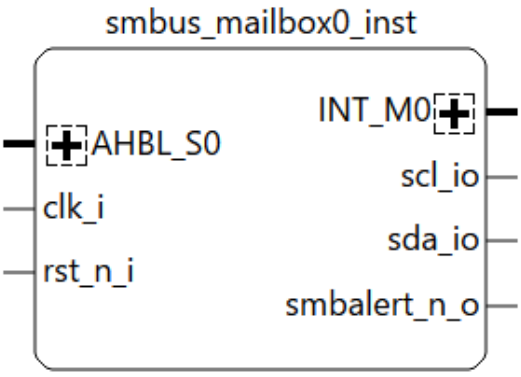

Figure 4.5. Generated Instance

5. Applicable Devices

- MachXO3D™
- MachXO5™-NX
- CertusPro™-NX
- MachXO4™

Appendix A. Resource Utilization

Table A.1 shows the SMBus Mailbox resource utilization for LFMXO4-050HE-5BBG256A device using Synplify Pro of Lattice Radiant Software 2025.2.

Table A.1. Resource Utilization for LFMXO4-050HE-5BBG256A

Configuration	clk Fmax (MHz)[1]	Registers	LUTs	EBRs
Default	124.15	387	778	0
Enable Master Function = Unchecked, Others = Default	132.34	251	540	0
Addressing Mode = 10 bit, System Clock Frequency = 100MHz, I2C Frequency = 1000kHZ, Others = Default	131.22	399	816	0
Enable Master Function = Unchecked, Addressing Mode = 10 bit, System Clock Frequency = 100MHz, I2C Frequency = 1000kHZ, Others = Default	131.06	248	493	0

Notes:

1. Fmax is generated when the FPGA design only contains the SMBus Mailbox module, and the target frequency is 100 MHz. These values may be reduced when a user logic is added to the FPGA design.

References

- SMBus Mailbox IP Release Notes (FPGA-RN-02108)
- SMBus Mailbox IP web page
- CertusPro-NX Devices web page
- MachXO3D Devices web page
- Mach-XO4 Devices web page
- MachXO5-NX Devices web page
- AMBA 3 AHB-Lite Protocol Specification
- SMBus Specification
- Lattice Radiant Software web page
- Lattice Propel Design Environment web page
- 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

Note: In some instances, the IP may be updated without changes to the user guide. The user guide may reflect an earlier IP version but remains fully compatible with the later IP version. Refer to the IP Release Notes for the latest updates.

Revision 1.2, IP v1.2.0, December 2025

Section	Change Summary
All	• Updated the document title from *Lattice Sentry SMBus Mailbox IP Core- Propel Builder* to *SMBus Mailbox IP – Lattice Propel Builder*. • Added the IP version to the cover page.
Abbreviations in This Document	• Updated the section name to its current. • Updated the abbreviation list.
Applicable Devices	• Removed *Mach-NX*. • Added *MachXO5™-NX*, *CertusPro™-NX*, and *MachXO4™*.
Appendix A. Resource Utilization	Added this section.
References	Updated this section.
Revision History	Added the following note: *In some instances, the IP may be updated without changes to the user guide. The user guide may reflect an earlier IP version but remains fully compatible with the later IP version. Refer to the IP Release Notes for the latest updates*.

Revision 1.1, August 2024

Section	Change Summary
All	Changed *master* to *controller* and *slave* to *target* globally.
Introduction	Updated Figure 1.1. SMBus Mailbox Write Byte Message, Figure 1.2. SMBus Mailbox Read Byte Message, and Figure 1.3. MCTP over SMBus Packet Format to reflect inclusive language.
Functional Description	• Updated Figure 2.1. SMBus IP Core Functional Diagram to reflect inclusive language. • In Table 2.1. Interface Signal Description, updated Reset from *O* to *0* for the following signals: ahbl_hrdata_slv_o, ahbl_hreadyout_slv_o, ahbl_hresp_slv_o, int_o, and smbalert_n_o. • In Table 2.3. Registers Address Map, updated the Reset value of 0x0C from *[7:6] RSVD [4:1] 0 [0] See Addressing Mode in Table 2.2* to *[7:6] RSVD [5:1] 0 [0] See Addressing Mode in Table 2.2*.
Generating the SMBus Mailbox IP	Updated Figure 4.1. Module/IP Block Wizard, Figure 4.2. Configuring Parameters, Figure 4.3. Verifying Results, Figure 4.4. Specifying Instance Name, and Figure 4.5. Generated Instance.
Technical Support Assistance	Added the link to Lattice Answer Database.

Revision 1.0, December 2021

Section	Change Summary
All	Initial release.

www.latticesemi.com