Lattice Sentry SMBus Mailbox IP Core - Propel Builder

User Guide

FPGA-IPUG-02165-1.0

December 2021
Disclaimers

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

Acronyms in This Document .................................................................................................................. 5
1. Introduction........................................................................................................................................ 6
   1.1. Features ........................................................................................................................................ 7
   1.2. Conventions .................................................................................................................................. 7
2. Functional Descriptions ...................................................................................................................... 8
   2.1. Overview ...................................................................................................................................... 8
   2.2. Signals Description ....................................................................................................................... 9
   2.3. Attributes ..................................................................................................................................... 10
   2.4. Register Description .................................................................................................................... 11
      2.4.1. Slave Controller Register ...................................................................................................... 15
      2.4.2. Registers File .......................................................................................................................... 22
      2.4.3. Master Controller Register .................................................................................................... 22
   2.5. SMBus Alert Signal ...................................................................................................................... 22
3. Program Flow ...................................................................................................................................... 23
   3.1. SMBus Slave Controller Initialization ......................................................................................... 23
   3.2. SMBus Master Controller Initialization ....................................................................................... 23
   3.3. SMBus Slave Controller Operation Flow ..................................................................................... 23
      3.3.1. Data Transfer in Response to External Master Read ............................................................... 23
      3.3.2. Data Transfer in Response to External Master Write ............................................................. 24
   3.4. SMBus Master Controller Operation Flow .................................................................................... 24
      3.4.1. Write Data to the SMBus Slave .............................................................................................. 25
      3.4.2. Read Data from the SMBus Slave .......................................................................................... 26
   3.5. C Code API .................................................................................................................................... 26
4. Generating the SMBus Mailbox IP ..................................................................................................... 27
5. Applicable Devices ............................................................................................................................. 29
References ............................................................................................................................................... 30
Technical Support Assistance ................................................................................................................ 31
Revision History ................................................................................................................................. 32
Figures

Figure 1.1. SMBus Mailbox Write Byte Message .................................................................6
Figure 1.2. SMBus Mailbox Read Byte Message .................................................................6
Figure 1.3. MCTP over SMBus Packet Format ................................................................6
Figure 2.1. SMBus IP Core Functional Diagram .................................................................9
Figure 2.2. 7-bit Addressable Device Response .................................................................22
Figure 3.1. SMBus Master Program Flow Interrupt Mode ..................................................25
Figure 4.1. Module/IP Block Wizard ...................................................................................27
Figure 4.2. Configuring Parameters ....................................................................................27
Figure 4.3. Verifying Results .................................................................................................28
Figure 4.4. Specifying Instance Name ..................................................................................28
Figure 4.5. Generated Instance ...........................................................................................28

Tables

Table 2.1. Interface Signal Description ................................................................................9
Table 2.2. Attributes Description .........................................................................................10
Table 2.3. Registers Address Map .......................................................................................11
Table 2.4. Access Type Definition ........................................................................................15
## Acronyms in This Document

A list of acronyms used in this document.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AHB-Lite</td>
<td>Advanced High-performance Bus – Lite</td>
</tr>
<tr>
<td>APB</td>
<td>Advanced Peripheral Bus</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>I²C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>LMMI</td>
<td>Lattice Memory Mapped Interface</td>
</tr>
<tr>
<td>RoT</td>
<td>Root of Trust</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Level</td>
</tr>
<tr>
<td>SMBus</td>
<td>System Management Bus</td>
</tr>
</tbody>
</table>
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 Master, while the device being addressed is called the Slave.

SMBus bus protocols support many kinds of formats, such as SMBus write byte, SMBus write word, SMBus read word, SMBus write block, SMBus read block and so on. See SMBus Specification for more information. SMBus Mailbox is a SMBus slave, 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](image1)

![Figure 1.2. SMBus Mailbox Read Byte Message](image2)

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 8 bytes make up the packet header. The first three fields—Destination Slave 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 Slave 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](image3)
1.1. Features

The soft IP has the following features:

- Compatible with SMBus Specification
- Compatible with AHB-Lite Specification
- Supports SMBus master
- Supports SMBus slave
- 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 master
- 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 the two main blocks: one is Slave Controller and the other is Master Controller.

The Slave Controller performs two operations:
- Normal SMBus transfer as a Slave
- Register file transfer as SMBus Mailbox

The Master Controller can initiate SMBus transfer as a SMBus master.

The SMBus interface connected to external bus through SDA/SCL signals. The Master Controller connects through the P1 interface and the Slave Controller connects through the P0 interface. Therefore, a switch/mux is needed between Master and Slave Controllers. The switch is implemented in SMBus interface by the following method:
- If Master Controller does not initiate transfer, P0 is routed to the SMBus interface and P1 is switched off. The SMBus slave controller can then exchange data using the SDA/SCL signals.
- If Master Controller initiates transfer, P1 is routed to the SMBus interface and P0 is switched off. The SMBus Master Controller can then exchange data using the SDA/SCL signals.

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

The Slave Controller makes sure the IP serves as a SMBus slave device on the bus. The external master 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. There existing two slave address for the IP are the configured I\textsuperscript{2}C Slave Address from IP attributes which can also be changed dynamically by the host and the SMBus Device Default Address, 7\textsuperscript{b}1100-001, according to the SMBus Specification.

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

Table 2.1. Interface Signal Description

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Width</th>
<th>Direction</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clock and Reset</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clk_i</td>
<td>1</td>
<td>input</td>
<td>—</td>
<td>System clock</td>
</tr>
<tr>
<td>rst_n_i</td>
<td>1</td>
<td>input</td>
<td>—</td>
<td>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.</td>
</tr>
<tr>
<td><strong>AHB-Lite Bus</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ahbl_hsel_slv_i</td>
<td>1</td>
<td>input</td>
<td>—</td>
<td>AHBL Select signal. Indicates that the slave device is selected and a data transfer is required.</td>
</tr>
<tr>
<td>ahbl_haddr_slv_i</td>
<td>32</td>
<td>input</td>
<td>—</td>
<td>The system address bus.</td>
</tr>
<tr>
<td>ahbl_hburst_slv_i</td>
<td>3</td>
<td>input</td>
<td>—</td>
<td>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</td>
</tr>
<tr>
<td>Signal Name</td>
<td>Width</td>
<td>Direction</td>
<td>Reset</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------</td>
<td>-----------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ahbl_hprot_slv_i</td>
<td>4</td>
<td>input</td>
<td>—</td>
<td>ahbl_hprot_slv_i [0] : 1'b0 – opcode fetch; 1'b1 - data access.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ahbl_hprot_slv_i [1]: 1'b0 – user access; 1'b1 - privileged access.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ahbl_hprot_slv_i [2]: 1'b0 – non-bufferable, 1'b1 - bufferable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ahbl_hprot_slv_i [3]: 1'b0 – non-cacheable; 1'b1 - cacheable.</td>
</tr>
<tr>
<td>ahbl_hsize_slv_i</td>
<td>3</td>
<td>input</td>
<td>—</td>
<td>3'b000: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3'b001: 2 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3'b010: 4 bytes</td>
</tr>
<tr>
<td>ahbl_htrans_slv_i</td>
<td>2</td>
<td>input</td>
<td>—</td>
<td>Indicates the transfer type of the current transfer. This can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2'b00: IDLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2'b01: BUSY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2'b10: NONSEQUENTIAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2'b11: SEQUENTIAL</td>
</tr>
<tr>
<td>ahbl_hwdata_slv_i</td>
<td>32</td>
<td>input</td>
<td>—</td>
<td>The write data bus</td>
</tr>
<tr>
<td>ahbl_hwrite_slv_i</td>
<td>1</td>
<td>input</td>
<td>—</td>
<td>When HIGH, this signal indicates a write transfer and when LOW a read transfer.</td>
</tr>
<tr>
<td>ahbl_hready_slv_i</td>
<td>1</td>
<td>input</td>
<td>0</td>
<td>This signal should come from AHBL Interconnect. When set to 1, this indicates the previous transfer is complete.</td>
</tr>
<tr>
<td>ahbl_hrdyout_slv_o</td>
<td>1</td>
<td>output</td>
<td>O</td>
<td>When HIGH, this signal indicates that a transfer has finished on the bus. This signal can be driven LOW to extend a transfer.</td>
</tr>
<tr>
<td>ahbl_hresp_slv_o</td>
<td>1</td>
<td>output</td>
<td>O</td>
<td>When LOW, this signal indicates that the transfer status is OKAY. When HIGH, it indicates that the transfer status is ERROR.</td>
</tr>
</tbody>
</table>

### Interrupt Signal

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int_o</td>
<td>output</td>
<td>Interrupt to host (CPU), reset value is 1'b0.</td>
</tr>
</tbody>
</table>

### SMBus Signal

<table>
<thead>
<tr>
<th>Signal</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scl_io</td>
<td>inout</td>
<td>SMBus Serial Clock</td>
</tr>
<tr>
<td>sda_io</td>
<td>inout</td>
<td>SMBus data signal</td>
</tr>
<tr>
<td>smbalert_n_o</td>
<td>output</td>
<td>SMBus alert (active low)</td>
</tr>
</tbody>
</table>

## 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Master Function</td>
<td>Checkbox: check, uncheck</td>
<td>Switch on/off master function. Check: enable master controller; uncheck: disable master controller. Slave only mode.</td>
</tr>
<tr>
<td>SMBus Mailbox Base Address</td>
<td>0 ~ (AHB Lite Address assignment designation)</td>
<td>Specify IP base address on AHB-Lite bus.</td>
</tr>
<tr>
<td>AHB Lite Address bus number of bits</td>
<td>[1–32]</td>
<td>AHB-Lite bus address width</td>
</tr>
</tbody>
</table>
### 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>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 Slave Address[6:0]</td>
<td>Slave Address Lower Register, same to I2C Slave Address attribute</td>
</tr>
<tr>
<td>0x08</td>
<td>SLVADR_H_REG</td>
<td>R/W</td>
<td>[7:3] RSVD [2:0] I2C Slave Address[9:7]</td>
<td>Slave Address Higher Register, same to I2C Slave Address attribute</td>
</tr>
<tr>
<td>0x0C</td>
<td>CONTROL_REG</td>
<td>R/W</td>
<td>[7:6] RSVD [4:1] 0 [0] See Addressing Mode in Table 2.2</td>
<td>Control Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[7:6]</td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[7]</td>
<td>RSVD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[5]</td>
<td>dat_src_sw</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[4]</td>
<td>nack_data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[3]</td>
<td>nack_addr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[1]</td>
<td>clk_stretch_en</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[0]</td>
<td>addr_10bit_en</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>
<tr>
<td>0x14</td>
<td>INT_STATUS1_REG</td>
<td>RW1C</td>
<td>8'h00</td>
<td>Interrupt Status First Register</td>
</tr>
</tbody>
</table>

© 2021 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal. All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.
<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x18</td>
<td>INT_ENABLE1_REG</td>
<td>R/W</td>
<td>8'h00</td>
<td>Interrupt Enable First Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Field Name</td>
</tr>
<tr>
<td></td>
<td>[7]</td>
<td>tr_cmp_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[6]</td>
<td>stop_det_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[5]</td>
<td>tx_fifo_full_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[4]</td>
<td>tx_fifo_empty_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3]</td>
<td>tx_fifo_aempty_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[2]</td>
<td>rx_fifo_full_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>rx_fifo_full_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[0]</td>
<td>rx_fifo_full_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>0x1C</td>
<td>INT_SET1_REG</td>
<td>WO</td>
<td>8'h00</td>
<td>Interrupt Set First Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Field Name</td>
</tr>
<tr>
<td></td>
<td>[7]</td>
<td>tr_cmp_set</td>
<td>WO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>rx_fifo_full_set</td>
<td>WO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[0]</td>
<td>rx_fifo_full_set</td>
<td>WO</td>
<td></td>
</tr>
<tr>
<td>0x20</td>
<td>INT_STATUS2_REG</td>
<td>RW1C</td>
<td>8'h00</td>
<td>Interrupt Status Second Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Field Name</td>
</tr>
<tr>
<td></td>
<td>[7]</td>
<td>reserved</td>
<td>RSVD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[6]</td>
<td>scl_h_to</td>
<td>RW1C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[5]</td>
<td>scl_l_to</td>
<td>RW1C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[4]</td>
<td>SR_check_value</td>
<td>RW1C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3]</td>
<td>SR_check_valid</td>
<td>RW1C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[2]</td>
<td>arp_cmd_det</td>
<td>RW1C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>stop_err_int</td>
<td>RW1C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[0]</td>
<td>start_err_int</td>
<td>RW1C</td>
<td></td>
</tr>
<tr>
<td>0x24</td>
<td>INT_ENABLE2_REG</td>
<td>R/W</td>
<td>[7:2] RSVD</td>
<td>2'b00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Field Name</td>
</tr>
<tr>
<td></td>
<td>[7]</td>
<td>reserved</td>
<td>RSVD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[6]</td>
<td>scl_h_to_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[5]</td>
<td>scl_l_to_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[4]</td>
<td>reserved</td>
<td>RSVD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[2]</td>
<td>arp_cmd_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>stop_err_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[0]</td>
<td>start_err_en</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>0x28</td>
<td>INT_SET2_REG</td>
<td>WO</td>
<td>[7:2] RSVD</td>
<td>2'b00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Field Name</td>
</tr>
<tr>
<td></td>
<td>[7]</td>
<td>reserved</td>
<td>RSVD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[6]</td>
<td>sch_h_to_set</td>
<td>WO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[4]</td>
<td>reserved</td>
<td>RSVD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>stop_err_set</td>
<td>WO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[0]</td>
<td>start_err_set</td>
<td>WO</td>
<td></td>
</tr>
</tbody>
</table>
### Lattice Sentry SMBus Mailbox IP Core

#### Propeller Builder User Guide

© 2021 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal. All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

---

**Register Name**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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</td>
</tr>
</tbody>
</table>

#### Field Access:

<table>
<thead>
<tr>
<th>Field</th>
<th>Name</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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</td>
</tr>
</tbody>
</table>

#### Field Access:

<table>
<thead>
<tr>
<th>Field</th>
<th>Name</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x30</td>
<td>SMB_CONTROL_REG</td>
<td>R/W</td>
<td>SMBus control and status register</td>
</tr>
</tbody>
</table>

#### Field Access:

<table>
<thead>
<tr>
<th>Field</th>
<th>Name</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7:1]</td>
<td>RSVD</td>
<td>RSVD</td>
</tr>
<tr>
<td>[0]</td>
<td>smb_alert</td>
<td>RSVD</td>
</tr>
</tbody>
</table>

- `1'b0` – No interrupt to Master
- `1'b1` – SMBus slave sent alert interrupt to Master

---

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x34</td>
<td>Reserved</td>
<td>RSVD</td>
<td>Write access is ignored and 0 is returned on read access.</td>
</tr>
</tbody>
</table>

---

**AHB-Lite S01**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2000-0x23ff</td>
<td>Registers File</td>
<td>R/W</td>
<td>Registers File 256x32bits, SMBus Mailbox Registers File.</td>
</tr>
</tbody>
</table>

#### Field Access:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Dword number</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

---

**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**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x400</td>
<td>PRERlo</td>
<td>R/W</td>
<td>Clock prescale register low-byte</td>
</tr>
</tbody>
</table>

#### Field Access:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>RSVD</td>
</tr>
<tr>
<td>[7:0]</td>
<td>8’hff</td>
</tr>
</tbody>
</table>

- `5xSCL frequency = clk_i / (PRERhi<<8 + PRERlo)`

---

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x404</td>
<td>PRERhi</td>
<td>R/W</td>
<td>Clock prescale register high-byte</td>
</tr>
</tbody>
</table>

#### Field Access:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>RSVD</td>
</tr>
<tr>
<td>[7:0]</td>
<td>8’hff</td>
</tr>
</tbody>
</table>

- `5xSCL frequency = clk_i / (PRERhi<<8 + PRERlo)`

---

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x408</td>
<td>CTR</td>
<td>R/W</td>
<td>Control register</td>
</tr>
</tbody>
</table>

#### Field Access:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>RSVD</td>
</tr>
<tr>
<td>[7:6]</td>
<td>2’h00</td>
</tr>
<tr>
<td>[5:0]</td>
<td>RSVD</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40c</td>
<td>TXR</td>
<td>WO</td>
<td>Transmit register</td>
</tr>
</tbody>
</table>

#### Field Access:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>RSVD</td>
</tr>
<tr>
<td>[7:0]</td>
<td>8’h00</td>
</tr>
</tbody>
</table>

- `a) The byte’s LSB.`
- `b) RW bit during slave address transfer`
- `1 = Reading from slave`
- `0 = Writing to slave`
<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>Access</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 Master Controller</td>
</tr>
<tr>
<td>0x410</td>
<td>CR</td>
<td>WO</td>
<td>[31:8] RSVD [7:0] 8'h00</td>
<td>Command register</td>
</tr>
<tr>
<td></td>
<td>Field</td>
<td>Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7]</td>
<td>STA – Generate (repeated) start condition</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[6]</td>
<td>STO – Generate stop condition</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td>RD – Read from slave</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4]</td>
<td>WR – Write to slave</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td>ACK, when a receiver, sent ACK (ACK = 0) or NACK (ACK = 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>When set, clears the timeout status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[0]</td>
<td>IACK – Interrupt acknowledge. When set, clears a pending interrupt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x410</td>
<td>SR</td>
<td>RO</td>
<td>[31:8] RSVD [7:0] 8'h00</td>
<td>Status register</td>
</tr>
<tr>
<td></td>
<td>Field</td>
<td>Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7]</td>
<td>RxACK – Received acknowledge from addressed slave. 1 = No acknowledge received 0 = Acknowledge received</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[6]</td>
<td>Busy – Indicates that the SMBus bus is busy. 1 = bus is busy 0 = bus is idle</td>
<td></td>
<td></td>
<td></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 master drives SDA high, but SDA is low • Master drive SCL high, but SCL is low. Not clock stretch.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4]</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td>Timeout 1 = SCL and SDA line have been high for 50us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>Timeout 1 = SCL line has been low for 25ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>TIP – Transfer in Progress 1 = Transferring data 0 = Transfer is complete</td>
<td></td>
<td></td>
<td></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:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 2.4.1. Slave Controller Register

AHB-Lite S00 is the port to access Slave Controller core register. The offset address is 0. The detail 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 WR_DATA_REG during I^2C 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 I^2C bus during I^2C 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 I^2C Slave 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 Slave Address Lower Register (SLAVE_ADDRL_REG) is a 7-bit Slave address. This is used for 7-bit and 10-bit addressing mode as follows: for 7-bit Addressing Mode, it is the Slave address; for 10-bit Addressing Mode, it is the lower 7 bits of the Slave address. The Slave Address Higher Register (SLAVE_ADDRH_REG) is the upper 3 bits of 10-bit Slave address. This is not used in 7-bit addressing mode. The reset values of SLAVE_ADDRL_REG and SLAVE_ADDRH_REG is set by the I^2C Slave Address attribute as shown in Table 2.2.

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

- **dat_src_sw**
  Data source switch. Select data source when external master read routine.
  - 1'b0 select register file for mailbox
  - 1'b1 select tx_fifo for normal external read

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

---

**Table 2.4. Access Type Definition**

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Behaviour on Read Access</th>
<th>Behaviour on Write Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>RO</td>
<td>Returns register value</td>
<td>Ignores write access</td>
</tr>
<tr>
<td>WO</td>
<td>Returns 0</td>
<td>Updates register value</td>
</tr>
<tr>
<td>RW</td>
<td>Returns register value</td>
<td>Updates register value</td>
</tr>
<tr>
<td>RW1C</td>
<td>Returns register value</td>
<td>Writing 1'b1 on register bit clears the bit to 1'b0. Writing 1'b0 on register bit is ignored.</td>
</tr>
<tr>
<td>RSVD</td>
<td>Returns 0</td>
<td>Ignores write access</td>
</tr>
</tbody>
</table>
• **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 slave address
  1'b1 – Sends NACK to received address

• **reset**
  Reset. Resets I²C Slave 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 auto clear feature; it is cleared to 1'b0 after 1 clock cycle.
  1'b0 – No action.
  1'b1 – Resets I²C Slave IP Core.

• **clk_stretch_en**
  Clock Stretch Enable. Enables clock stretching on ACK bit of data.
  1'b0 – I²C Slave IP Core releases SCL signal
  1'b1 – I²C Slave 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 Slave IP Core rejects the 10-bit I²C address, it sends NACK.
  1'b1 – I²C Slave IP Core responds to 10-bit I²C address. If SLAVE_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 Slave Controller 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.

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_almost_full_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_timeout
  Timeout flag when SCL and SDA line have been high for 50us in transfer mode
  
  1'b0 – No interrupt
  1'b1 – Interrupt pending

• scl_l_timeout
  Timeout flag when SCL line has been low for 25ms
  
  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 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. 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 corresponded to Transfer Complete Interrupt Status.
  
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

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

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

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

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

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

• **rx_fifo_afull_en**
  
  Receive FIFO Almost Full Interrupt Enable. Interrupt enable bit corresponded to Receive FIFO Almost Full Interrupt Status.
  
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled
• rx_fifo_ready_en
  
  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 50us in transfer mode
  
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

• scl_l_to_en
  enable interrupt when SCL line has been low for 25ms
  
  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 corresponded to STOP Condition Error Interrupt Status.
  
  1'b0 – Interrupt disabled
  1'b1 – Interrupt enabled

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

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

• stop_det_set
  STOP Condition Detected Interrupt Set. Interrupt set bit corresponded to STOP Condition Detected Interrupt Status.
  
  1'b0 – No action
  1'b1 – Asserts INT_STATUS1_REG.stop_det_int
- `tx_fifo_full_set`
  Transmit FIFO Full Interrupt Set. Interrupt set bit corresponded to Transmit FIFO Full Interrupt Status.
  - 1'b0 – No action
  - 1'b1 – Asserts INT_STATUS1_REG.tx_fifo_full_int

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

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

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

- `rx_fifo_afull_set`
  - 1'b0 – No action
  - 1'b1 – Asserts INT_STATUS1_REG.rx_fifo_afull_int

- `rx_fifo_ready_set`
  - 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 corresponded 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 corresponded 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 corresponded 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 corresponded to STOP Condition Error Interrupt Status.
  - 0 – No action.
  - 1 – Asserts INT_STATUS2_REG.stop_err_set.
- start_err_set
  START Condition Error Interrupt Set. Interrupt set bit corresponded to START Condition Error Interrupt Status.
  0 – No action.
  1 – Asserts INT_STATUS2_REG.start_err_set.
FIFO Status Register reflects the status of Transmit FIFO and Receive FIFO as shown blow.
- tx_fifo_full
  Transmit FIFO Full. This bit reflects the full condition of Transmit FIFO.
  1'b0 – Transmit FIFO is not full
  1'b1 – Transmit FIFO is full
- tx_fifo_aempty
  Transmit FIFO Almost Empty. This bit reflects the almost empty condition of Transmit FIFO.
  1'b0 – Data words in Transmit FIFO is greater than ‘TX FIFO Almost Empty Flag’ attribute
  1'b1 – Data words in Transmit FIFO is less than or equal to ‘TX FIFO Almost Empty Flag’ attribute
- tx_fifo_empty
  Transmit FIFO Empty. This bit reflects the empty condition of Transmit FIFO.
  1'b0 – Transmit FIFO is not empty – has at least 1 data word
  1'b1 – Transmit FIFO is empty
- rx_fifo_full
  Receive FIFO Full. This bit reflects the full condition of Receive FIFO.
  1'b0 – Receive FIFO is not full
  1'b1 – Receive FIFO is full
- rx_fifo_afull
  Receive FIFO Full. This bit reflects the almost full condition of Receive FIFO.
  1'b0 – Data words in Receive FIFO is less than ‘RX FIFO Almost Full Flag’ attribute
  1'b1 – Data words in Receive FIFO is greater than or equal to ‘RX FIFO Almost Full Flag’ attribute
- rx_fifo_empty
  Receive FIFO Empty. This bit reflects the empty condition of Receive FIFO.
  1'b0 – Receive FIFO is not empty – has at least 1 data word
  1'b1 – Receive FIFO is empty
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 – flush tx fifo data to empty.
2.4.2. Registers File

The external SMBus master initiates an SMBus Mailbox read transaction. The read byte data message is routed to the Register File. The external SMBus master cannot write message to the Register File. The external SMBus master writes message 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 AHB_Lite port S01. The offset address is 0x2000.

The contents of the Register File is byte aligned. Every byte has a dedicated meaning. See Mach-NX User Guide Specification for more information.

2.4.3. Master Controller Register

AHB-Lite S02 is the port to access Master 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 master 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 \((\text{clk_i frequency} / (\text{Prescale Register} + 1))\) 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 Slave Controller 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 master when pulled Low. A slave device can signal the master through smbalert_n_o interrupt line that it wants to talk. The master processes the interrupt and simultaneously accesses all the smbalert devices through the Alert Response Address. Only the slave device, which pulled 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 slave transmit device is placed in the 7 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 master in response to its address, the device stops pulling down the smbalert_n_o signal. If the master still sees the smbalert_n_o low when the message transfer is complete, the same process repeats again. The SMBus slave controller monitors the data bus to see if any other slave is responding to the Alert Response address. This can be achieved by checking the input and output of smdat_io. When there is match, the smb_alert register bit is cleared and the controller generates an interrupt signal to the host.

<table>
<thead>
<tr>
<th>S</th>
<th>Alert Response Address</th>
<th>Rd</th>
<th>A</th>
<th>Address</th>
<th>X</th>
<th>N</th>
<th>P</th>
</tr>
</thead>
</table>

Figure 2.2. 7-bit Addressable Device Response
3. Program Flow

The SMBus mailbox IP can be used as a SMBus master and SMBus slave simultaneously. But, the SMBus master function also can be disabled by unchecking the Enable Master function attribute box when configuring the IP in the Lattice Propel Builder.

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

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

3.1. SMBus Slave Controller Initialization

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

- SLAVE_ADDR low_REG, SLAVE_ADDR high_REG – This step is optional. In most cases, initial value set in i²C Slave Addresses attribute of the user interface does not need to be changed. Read access to the address by external master 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 master.
  - Transfer Complete Interrupt – If the size of data is known.
  - Receive FIFO Data Interrupt – if the size of data is unknown.
- INT_ENABLE2_REG – It is recommended to enable both error interrupts.

3.2. SMBus Master Controller 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 / (PRERhigh<<8 + PRERlow).

3.3. SMBus Slave Controller Operation Flow

3.3.1. Data Transfer in Response to External Master Read

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

3.3.1.1. Normal SMBus Slave Device Read Data Transfer

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

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

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

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 Master Write
Similarly, the external SMBus master can initiate master write transaction to two slave address. One is routed to 
internal RX_FIFO logic and the other is to Register File through RISCV host.

3.3.2.1. Normal SMBus Slave Device Write Data Transfer
The following are the recommended steps to perform data transfer in response to write request of SMBus Master. This 
assumes that the amount of data to receive is known.
To perform data transfer in response to write request of SMBus Master:
1. Enable only Transfer Complete Interrupt. If data to receive is > FIFO Depth, enable also RX FIFO Almost Full 
   interrupt. If data to receive is ≤ FIFO Depth, proceed to Step 7.
2. Wait for RX FIFO Almost Full Interrupt. If polling mode is desired, read INT_STATUS2_REG until rx_fifo_afull_int 
   asserts. If interrupt mode is desired, simply wait for interrupt signal to assert, then read INT_STATUS2_REG and 
   check that rx_fifo_afull_int is asserted. Read INT_STATUS2_REG also to check that no error occurred.
3. Clear RX FIFO Almost Full Interrupt, it also okay to clear all interrupts.
4. Read data byte from RD_DATA_REG, amounting to less than or equal to (FIFO Depth - TX FIFO Almost Empty 
   Setting).
5. If there are remaining data to receive, go back to Step 2, otherwise, disable RX FIFO Almost Full Interrupt.
6. Wait for Transfer Complete Interrupt. If polling mode is desired, read INT_STATUS1_REG until tr_cmp_int asserts. 
   If interrupt mode is desired, simply wait for interrupt signal to assert, then read INT_STATUS1_REG and check that 
   tr_cmp_int is asserted. Read INT_STATUS2_REG also to check that no error occurred.
7. Clear all interrupts.
8. Read all data from RD_DATA_REG.

3.3.2.2. SMBus Mailbox Register File Write data transfer
If the accessed address is Register File, the external master 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 Master Controller Operation Flow
Figure 3.1 shows the SMBus master program flow in interrupt mode. The master controller can also be used in polling 
mode. The polling mode is same as interrupt mode except that the polling mode needs to poll the SR bit 0 instead of 
interrupted by int_o to check status. In the polling mode, set the CTR to 0x80.
### 3.4.1. Write Data to the SMBus Slave

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 slave 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 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 slave 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 no error and have 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. Bit6 is set when other master 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.

3.4.2. Read Data from the SMBus Slave

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 slave 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 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 0x20 to command register (CR) through the AHB-Lite bus to read data from the slave. 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 master 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 Propel Builder.

To generate the SMBus Mailbox IP Core module:

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

![Figure 4.1. Module/IP Block Wizard](image)

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

![Figure 4.2. Configuring Parameters](image)
4. Verify the information. Click Finish.

![Figure 4.3. Verifying Results](image)

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

![Figure 4.4. Specifying Instance Name](image)

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

![Figure 4.5. Generated Instance](image)
5. Applicable Devices

- MachXO3D™
- Mach™-NX
References

- AMBA 3 AHB-Lite Protocol Specification
- SMBus Specification
Technical Support Assistance
Submit a technical support case through www.latticesemi.com/techsupport.
## Revision History

**Revision 1.0, December 2021**

<table>
<thead>
<tr>
<th>Section</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>