



I2C Controller IP

IP Version: v2.4.0

User Guide

FPGA-IPUG-02071-2.1

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

Contents	3
Acronyms in This Document	6
1. Introduction	7
1.1. Overview of the IP	7
1.2. Quick Facts	7
1.3. IP Support Summary	7
1.4. Features	8
1.5. Licensing Information	8
1.6. Hardware Support	8
1.7. Minimum Device Requirements	8
1.8. Naming Conventions	8
1.8.1. Nomenclature	8
1.8.2. Signal Names	8
1.8.3. Attribute Names	9
2. Functional Description	10
2.1. IP Architecture Overview	10
2.2. Clocking and Reset	10
2.3. Operations Details	11
2.3.1. General I2C Operation	11
2.3.2. Clock Generation and Synchronization	11
2.3.3. Glitch Filter	12
2.3.4. Clock Stretching and SCL Timeout	12
2.3.5. Multi-controller Arbitration	12
2.4. User Interfaces	12
2.4.1. Selectable Memory-Mapped Interface	12
2.5. Programming Flow	13
2.5.1. Initialization	13
2.5.2. Write to a Target Device	13
2.5.3. Read from a Target Device	14
2.5.4. Read from a Target Device with a Specific Register or Command (Repeated Start)	14
3. IP Parameter Description	16
4. Signal Description	17
5. Register Description	19
5.1. Overview	19
5.2. Write Data Register (WR_DATA_REG)	20
5.3. Read Data Register (RD_DATA_REG)	20
5.4. Target Address Registers (TARGET_ADDRL_REG, TARGET_ADDRH_REG)	20
5.5. Control Register (CONTROL_REG)	20
5.6. Target Byte Count Register (TGT_BYTE_CNT_REG)	21
5.7. Mode Register (MODE_REG)	22
5.8. Clock Prescaler Low Register (CLK_PRESCAL_REG)	22
5.9. Interrupt Status Registers (INT_STATUS1_REG, INT_STATUS2_REG)	22
5.10. Interrupt Enable Registers (INT_ENABLE1_REG, INT_ENABLE2_REG)	24
5.11. Interrupt Set Registers (INT_SET1_REG, INT_SET2_REG)	26
5.12. FIFO Status Register (FIFO_STATUS_REG)	27
5.13. SCL Timeout Register (SCL_TIMEOUT_REG)	28
6. Example Design	29
6.1. Example Design Supported Configuration	29
6.2. Overview of the Example Design and Features	29
6.3. Design Components Example	31
6.4. Generating the Example Design	31
6.5. Hardware Testing	34

6.5.1.	Hardware Testing Setup	34
6.5.2.	Expected Output	34
7.	Designing with the IP	35
7.1.	Generating and Instantiating the IP	35
7.1.1.	Generated Files and File Structure	37
7.2.	Design Implementation	37
7.2.1.	Device Constraint Editor	37
7.2.2.	Manual PDC File Creation	37
7.3.	Timing Constraints	37
7.4.	Running Functional Simulation	38
Appendix A. Resource Utilization		40
References		42
Technical Support Assistance		43
Revision History		44

Figures

Figure 2.1. I2C Controller IP Functional Diagram.....	10
Figure 2.2. START and STOP Conditions	11
Figure 6.1. I2C Controller IP in Propel SoC Project	30
Figure 6.2. Sample C-Code Test Routine.....	30
Figure 6.3. I2C Controller Example Design Block Diagram	31
Figure 6.4. Create SoC Project	32
Figure 6.5. Define Instance	32
Figure 6.6. Build SoC Project Result.....	33
Figure 6.7. Lattice C/C++ Design Project.....	33
Figure 6.8. Build C/C++ Project Result	34
Figure 6.9 Sample I2C Read Command Sent by I2C Controller.....	34
Figure 7.1. Module/IP Block Wizard	35
Figure 7.2. Configure User Interface of I2C Controller IP	36
Figure 7.3. Check Generating Result.....	36
Figure 7.4. Simulation Wizard.....	38
Figure 7.5. Add and Reorder Source	38
Figure 7.6. Simulation Waveform	39

Tables

Table 1.1. Summary of the I2C Controller IP	7
Table 1.2. I2C Controller IP Support Readiness	7
Table 3.1. Attributes Table	16
Table 3.2. Attributes Descriptions	16
Table 4.1. I2C Controller IP Signal Description	17
Table 5.1. Registers Address Map.....	19
Table 5.2. Access Type Definition	19
Table 5.3. Write Data Register.....	20
Table 5.4. Read Data Register	20
Table 5.5. Target Address Lower Register	20
Table 5.6. Target Address Higher Register	20
Table 5.7. Control Register	20
Table 5.8. Target Byte Count Register	21
Table 5.9. Mode Register.....	22
Table 5.10. Clock Prescaler Low Register	22
Table 5.11. Interrupt Status First Register	23
Table 5.12. Interrupt Status Second Register	24
Table 5.13. Interrupt Enable First Register	24
Table 5.14. Interrupt Enable Second Register	25
Table 5.15. Interrupt Set First Register.....	26
Table 5.16. Interrupt Set Second Register	27
Table 5.17. FIFO Status Register	27
Table 5.18. SCL Timeout Register	28
Table 6.1. I2C Controller IP Configuration Supported by the Example Design	29
Table 7.1. Generated File List	37
Table A.1. LIFCL-40-9BG400I Device Resource Utilization	40
Table A.2. LAV-AT-E70-1LFG1156I Device Resource Utilization	40
Table A.3. LN2-CT-20-1CBG484C Device Resource Utilization	40
Table A.4. LFMXO4-110HC-5BBG484I Device Resource Utilization.....	41

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
APB	Advanced Peripheral Bus
AMBA	Advanced Microcontroller Bus Architecture
DUT	Design Under Test
EBR	Embedded Block RAM
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
I/O	Input/Output
I2C	Inter-Integrated Circuit
IP	Intellectual Property
LMMI	Lattice Memory Mapped Interface
LSE	Lattice Synthesis Engine
LUT	Look-Up Table
PIC	Programmable Interrupt Controller
PLL	Phase-Locked Loop
RAM	Random Access Memory
RISC-V	Reduced Instruction Set Computer Five
RTL	Register Transfer Level
RX	Receiver
SCL	Serial Clock
SDA	Serial Data
SoC	System on Chip
SRAM	Static Random Access Memory
TX	Transmitter
UART	Universal Asynchronous Receiver/Transmitter

1. Introduction

1.1. Overview of the IP

The I2C (Inter-Integrated Circuit) bus is a simple, low-bandwidth, short-distance protocol. It is often seen in systems with peripheral devices that are accessed intermittently. It is commonly used in short-distance systems, where the number of traces on the board should be minimized. The device that initiates the transmission on the I2C bus is commonly known as the Controller, while the device being addressed is called the Target.

Lattice Semiconductor general-purpose I2C Controller IP offers an effective way to control an I2C bus. The programmable nature of FPGA provides you with flexibility of configuring the I2C Controller device to your need, thus allowing you to customize the I2C Controller to meet your specific design requirements.

1.2. Quick Facts

Table 1.1. Summary of the I2C Controller IP

IP Requirements	Supported Devices	CrossLink™-NX, Certus™-NX, Certus-NX-RT, CertusPro™-NX, CertusPro-NX-RT, MachXO5™-NX, Lattice Avant™, Certus-N2, and MachXO4™.
	IP Changes ¹	For a list of changes to the IP, refer to the I2C Controller IP Release Notes (FPGA-RN-02027) .
Resource Utilization	Supported User Interfaces	Lattice Memory Mapped Interface (LMMI) and Advanced Peripheral Bus (APB)
	Resources	Refer to Appendix A. Resource Utilization
Design Tool Support	Lattice Implementation	IP Core v2.4.0 – Lattice Radiant™ Software 2025.2 and Lattice Propel™ Builder 2025.2
	Synthesis	Lattice Synthesis Engine Synopsys Synplify Pro® for Lattice
	Simulation	For a list of supported simulators, see the Lattice Radiant Software User Guide .

Note:

1. In some instances, the IP may be updated without changes to the user guide. This 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.

1.3. IP Support Summary

Table 1.2. I2C Controller IP Support Readiness

Device Family	System Clock Frequency (MHz) ¹	Supported Feature	Radiant Timing Model	Hardware Validated
Crosslink-NX	50	SCL at 100 kHz: I2C Read-Write	Final	Yes
Certus-NX	50	SCL at 100 kHz: I2C Read-Write	Final	No
Certus-NX-RT	50	SCL at 100 kHz: I2C Read-Write	Final	No
CertusPro-NX	50	SCL at 100 kHz: I2C Read-Write	Final	Yes
CertusPro-NX-RT	50	SCL at 100 kHz: I2C Read-Write	Final	No
MachXO5-NX	50	SCL at 100 kHz: I2C Read-Write	Final	Yes
		SCL at 400 kHz: I2C Read-Write	Final	Yes
		SCL at 1 MHz: I2C Read-Write	Final	Yes
	10	SCL at 100 kHz: I2C Read-Write	Final	Yes
		SCL at 400 kHz: I2C Read-Write	Final	Yes
		SCL at 1 MHz: I2C Read-Write	Final	Yes
	200	SCL at 100 kHz: I2C Read-Write	Final	Yes

Device Family	System Clock Frequency (MHz) ¹	Supported Feature	Radiant Timing Model	Hardware Validated
		SCL at 400 kHz: I2C Read-Write	Final	Yes
		SCL at 1 MHz: I2C Read-Write	Final	Yes
Avant	50	SCL at 100 kHz: I2C Read-Write	Preliminary	No
Certus-N2	50	SCL at 100 kHz: I2C Read-Write	Preliminary	No

Note:

1. This is the system clock frequency used during hardware validation. For the actual frequency supported by the IP, refer to [Appendix A. Resource Utilization](#).

1.4. Features

The I2C Controller IP supports the following key features:

- Supports 7-bit and 10-bit Addressing Mode
- Programmable SCL frequency, supporting the following bus speeds:
 - Standard-mode (Sm) – up to 100 kbit/s
 - Fast-mode (Fm) – up to 400 kbit/s
 - Fast-mode Plus (Fm+) – up to 1 Mbit/s
- Integrated Pull-up
- Integrated Glitch filter
- Arbitration lost detection in multi-controller system
- Polling and Out-of-band Interrupt Modes
- Selectable LMMI or APB interface
- Supports Clock stretching

1.5. Licensing Information

The I2C Controller IP is provided at no additional cost with the Lattice Radiant software.

1.6. Hardware Support

Refer to the [Example Design](#) section for more information on the boards used.

1.7. Minimum Device Requirements

There is no limitation in device speed grade for I2C Controller IP. See [Appendix A. Resource Utilization](#) for minimum required resources to instantiate this IP and maximum clock frequency supported.

1.8. Naming Conventions

1.8.1. Nomenclature

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

1.8.2. Signal Names

Signal Names that end with:

- `_n` are active low
- `_i` are input signals
- `_o` are output signals
- `_io` are bi-directional input/output signals

1.8.3. Attribute Names

Attribute names in this document are formatted in title case and italicized (*Attribute Name*).

2. Functional Description

This section provides a detailed functional description of the I2C Controller IP which includes information regarding clock and reset handling, and user interfaces.

2.1. IP Architecture Overview

The I2C Controller IP accepts commands from LMMI/APB interface via register programming. These commands are decoded into I2C read/write transactions to the external I2C target device. The I2C bus transactions can be configured to be 1 to 256 bytes in length.

The I2C Controller can operate in interrupt or polling mode. This means that you can choose to poll the I2C Controller for a change in status at periodic intervals (Polling Mode) or wait to be interrupted by the I2C Controller when data needs to be read or written (Interrupt Mode).

Functional diagram of the I2C Controller is presented in [Figure 2.1](#).

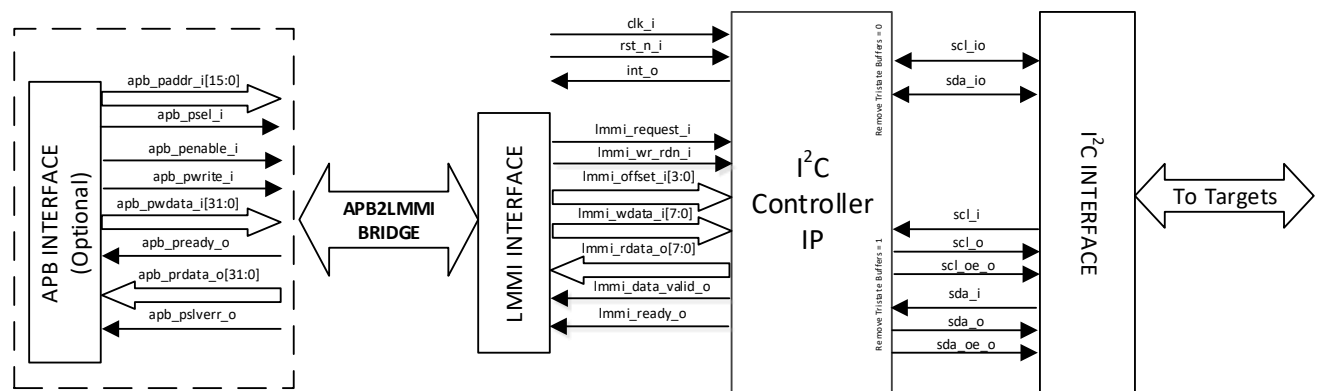


Figure 2.1. I2C Controller IP Functional Diagram

2.2. Clocking and Reset

The I2C Controller IP requires a system clock: `clk_i`. You need to provide this clock via an external source and route it to an appropriate pin on the FPGA. The system clock (externally sourced) will provide a timing reference and be used to operate the IP's internal logic.

The I2C Controller IP contains an asynchronous active low reset: `rst_n_i`. When asserted, the output ports and registers are forced to their default values. The reset assertion can be asynchronous but reset negation should be synchronous. The IP contains internal logic to synchronously de-assert the internal reset once `rst_n_i` is de-asserted, so you do not need to implement your own de-assertion logic.

Refer to the [Clock Generation and Synchronization](#) section for more information on clock and reset signal requirements.

2.3. Operations Details

2.3.1. General I2C Operation

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

The general procedure for an I2C transaction is as follows:

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

I2C communication is initiated by the controller sending a START condition and terminated by the controller sending a STOP condition. START condition is defined as a high-to-low transition on the SDA line while the SCL is high. STOP condition is defined as a low-to-high transition on the SDA line while the SCL is high. Also during transaction, SDA line is toggled only during SCL is low and is stable when SCL is high. These are shown in [Figure 2.2](#). For more information on the I2C bus, refer to [I2C Bus Specification and User Manual](#).

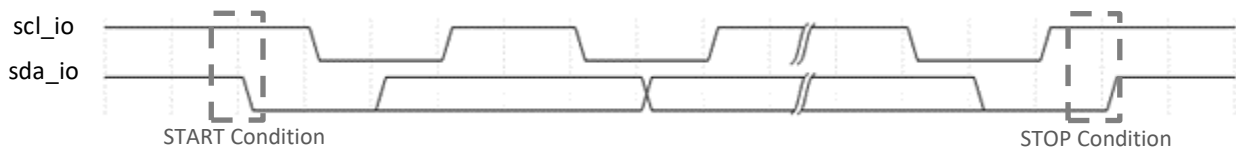


Figure 2.2. START and STOP Conditions

2.3.2. Clock Generation and Synchronization

I2C Controller IP generates an I2C clock signal, `scl_io`, based on the system clock (`clk_i`) and Clock Prescaler Register (`MODE_REG.clk_presc_high`, `CLK_PRESCAL_REG.clk_presc_low`). The initial value of Clock Prescaler Register is set by *Clock Prescaler* attribute. You can change the `scl_io` frequency by programming the Clock Prescaler registers. The $1/2$ period of `scl_io` is equal to N cycles of `clk_i`, where N is the Clock Prescaler value. The I2C Controller IP can be programmed to generate an I2C clock that runs in the Standard Mode (up to 100 kbits/sec) and the Fast Mode (up to 400 kbits/sec) and Fast Mode Plus (up to 1 MHz).

In multi-controller system, the actual SCL clock that is seen by all devices on the bus may not be running at the same frequency that the Controller generates. This is because of the clock synchronization that occurs in SCL line as follows:

- Low period in SCL line is set by the slowest controller driving the I2C clock.
- High period in SCL line is set by the fastest controller driving the I2C clock.

All the I2C Controllers in the bus start counting its own SCL Low period when SCL is driven low by at least one controller. When all controllers have counted off their Low period, they release their own clock line. However, due to the wired AND logic nature of the SCL line, it only goes to High logic when all controllers have released their own SCL signal. When SCL line goes High, all the I2C Controllers driving bus starts counting their own High periods. The first device to complete its High period again pulls the SCL line Low.

2.3.3. Glitch Filter

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

2.3.4. Clock Stretching and SCL Timeout

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

In unusual event, the target may hold the SCL line Low for a very long time. I2C Controller IP can detect this event through the SCL timeout feature. Refer to [SCL Timeout Register \(SCL_TIMEOUT_REG\)](#) section for more information.

2.3.5. Multi-controller Arbitration

The I2C Controller IP can operate in a multi-controller environment where two or more I2C bus controllers share the same bus. Arbitration occurs when multiple controllers attempt to initiate a transaction simultaneously. This arbitration process is performed bit by bit:

- While SCL is high, each controller verifies whether the SDA level matches the bit it transmits.
- Arbitration may span multiple bits, for example, when two controllers access the same target device.
- If both controllers transmit identical data, they complete the entire transaction without error.
- The first time a controller attempts to send a High (1) bit but detects Low (0) on SDA, it knows it has lost arbitration. At this point, it disables its SDA output driver, allowing the winning controller to complete its transaction.

When the I2C Controller IP loses arbitration, it asserts the Arbitration Loss Interrupt Status (INT_STATUS2_REG.arb_lost_int). The IP does not automatically retry the transaction; you must initiate retransmission manually, as shown in the following procedure.

For Write Transactions

1. Perform a software reset by writing 1'b1 to CONTROL_REG.reset and CONTROL_REG.tx_fifo_reset to stop the transmission and clear the Transmit FIFO.
2. Reprogram the controller for retransmission.
3. Start the transaction by writing 1'b1 to CONTROL_REG.start.

For Read Transactions

Arbitration loss can occur during the address phase or while sending read acknowledgments.

1. Write 1'b1 to CONTROL_REG.reset and CONTROL_REG.rx_fifo_reset to stop the transaction and clear the Receive FIFO.
2. Reprogram the controller for retransmission.
3. Restart by writing 1'b1 to CONTROL_REG.start.

2.4. User Interfaces

2.4.1. Selectable Memory-Mapped Interface

The memory-mapped interface of I2C Controller IP is selected by the *APB Mode Enable* attribute, which can be APB or LMMI. Register access is done via the selected memory-mapped interface. These interfaces are not described in this user guide, references to their respective specifications are provided below.

For LMMI interface, refer to the [Lattice Memory Mapped Interface and Lattice Interrupt Interface \(FPGA-UG-02039\)](#) document for information and timing diagram of the LMMI. Take note of the following information when checking LMMI timing diagram in the said document:

- Immi_ready_o is always asserted; thus, write and read transactions have no wait state.
- Read latency is one clock cycle.

For APB interface, refer to [AMBA 3 APB Protocol v1.0 Specification](#) for information and timing diagram of the APB interface. Take note of the following information when checking APB timing diagram in the said document:

- Write transaction has one wait state.
- Read transaction has one wait state (IP Core v1.0.2 or earlier has two wait states).

2.5. Programming Flow

2.5.1. Initialization

The following I2C Controller registers can be set outside of the actual transaction sequence. They should be set properly before starting an I2C transaction:

- TARGET_ADDRL_REG, TARGET_ADDRH_REG – Set the address of the target Target Device.
- CLK_PRESCL_REG – Set based on target scl_io frequency. The upper bits, MODE_REG.clk_presc_high are set during transaction because they are grouped with mode register.
- SCL_TIMEOUT_REG – Set to 8'h00 if you do not want to check SCL timeout or set to desired timeout value.
- INT_ENABLE2_REG – it is recommended to enable all interrupts in this register to check for error/unexpected event.

When accessing multiple devices, the TARGET_ADDRL_REG or TARGET_ADDRH_REG registers should be set prior to transaction.

2.5.2. Write to a Target Device

Below are the recommended steps for performing I2C write transaction. This assumes that the module is not currently performing any operation and initialization is completed.

To perform I2C write transaction:

1. Set the following MODE_REG fields according to the desired transfer mode: bus_speed_mode, addr_mode, clk_presc_high. Set the trx_mode field to 1'b0 for write transaction.
2. Set TGT_BYTE_CNT_REG according to the number of bytes to transfer.
3. Write data to WR_DATA_REG, amounting to \leq FIFO Depth.
4. Set CONTROL_REG.start to 1'b1 to start the I2C transaction.
5. Optional: If interrupt mode is desired, enable target interrupts in INT_ENABLE1_REG.
If number of words to transfer is \leq FIFO Depth, set tr_cmp_en = 1'b1.
If number of words to transfer is $>$ FIFO Depth, set the following: tx_fifo_aempty_en = 1'b1 and tr_cmp_en = 1'b1.
Other interrupts in this register are disabled.
6. If total number of bytes to transfer $>$ FIFO Depth, wait for Transmit FIFO Almost Empty Interrupt.
If polling mode is desired (interrupts are disabled), 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_aempty_int is asserted.
In both cases, read also INT_STATUS2_REG to ensure that the transfer is good.
7. Clear Transmit Buffer Almost Empty Interrupt by writing 1'b1 to INT_STATUS1_REG.tx_fifo_aempty_int. Clearing all interrupts in this register by writing 8'hFF to INT_STATUS1_REG is also okay, since we are not interested in other interrupts for this recommended sequence.
8. Write data to WR_DATA_REG, amounting to less than or equal to (FIFO Depth – TX FIFO Almost Empty Flag).
9. If there is remaining data to transfer, go back to Step 6.

10. Wait for Transfer Complete Interrupt.

If polling mode is desired (interrupts are disabled), read INT_STATUS1_REG until tr_cmp_int asserts.

If interrupt mode is desired, set INT_ENABLE1_REG = 8'h80, then wait for interrupt signal to assert. Then, read INT_STATUS1_REG and check that tr_cmp_int is asserted.

11. Clear all interrupts by writing 8'hFF to INT_STATUS1_REG.

2.5.3. Read from a Target Device

Below are the recommended steps for performing I2C read transaction. This assumes that the module is not currently performing any operation and initialization is completed.

To perform I2C read transaction:

1. Set the following MODE_REG fields according to the desired transfer mode: bus_speed_mode, addr_mode, clk_presc_high. Set the trx_mode field to 1'b1 for read transaction.
2. Set TGT_BYTE_CNT_REG according to the number of bytes to transfer.
3. Set CONTROL_REG.start to 1'b1 to start the I2C transaction.
4. Optional: If interrupt mode is desired, enable target interrupts in INT_ENABLE1_REG.
If number of words to transfer is \leq FIFO Depth, set tr_cmp_en = 1'b1.
If number of words to transfer is $>$ FIFO Depth, set the following: rx_fifo_afull_en = 1'b1 and tr_cmp_en = 1'b1.
Other interrupts in this register are disabled.
5. If total number of bytes to receive $>$ FIFO Depth, wait for Receive FIFO Almost Full Interrupt.
If polling mode is desired (interrupts are disabled), read INT_STATUS1_REG until rx_fifo_afull_int asserts.
If interrupt mode is desired, simply wait for interrupt signal to assert. Then, read INT_STATUS1_REG and check that rx_fifo_afull_int is asserted.
In both cases, read also INT_STATUS2_REG to ensure that the transfer is good.
6. Clear Receive FIFO Almost Full Interrupt by writing 1'b1 to INT_STATUS1_REG.rx_fifo_afull_int. Clearing all interrupts in this register by writing 8'hFF to INT_STATUS1_REG is also okay, since we are not interested in other interrupts for this recommended sequence.
7. Read all data from RD_DATA_REG. It is expected the amount of received data is less than or equal to (FIFO Depth – TX FIFO Almost Empty Flag). Read FIFO_STATUS_REG to confirm if Receive FIFO is emptied.
8. If there is remaining data to receive, go back to Step 5.
9. Wait for Transfer Complete Interrupt.
If polling mode is desired (interrupts are disabled), read INT_STATUS1_REG until tr_cmp_int asserts.
If interrupt mode is desired, set INT_ENABLE1_REG = 8'h80, then wait for interrupt signal to assert. Then, read INT_STATUS1_REG and check that tr_cmp_int is asserted.
10. Clear all interrupts by writing 8'hFF to INT_STATUS1_REG.
11. Read all the remaining data from RD_DATA_REG.

2.5.4. Read from a Target Device with a Specific Register or Command (Repeated Start)

Below are the recommended steps for performing a combined I2C read transaction. This assumes that the module is not currently performing any operation and initialization is completed.

To perform write transaction of a combined I2C transaction:

1. Set the following MODE_REG fields according to the desired transfer mode: bus_speed_mode, addr_mode, clk_presc_high. Set the trx_mode field to 1'b0 for write transaction.
2. Set TGT_BYTE_CNT_REG according to the number of bytes to transfer.
3. Write data to WR_DATA_REG, amounting to \leq FIFO Depth. Write the target register and commands needed.
4. Set CONTROL_REG.repeated_start to 1'b1 and CONTROL_REG.start to 1'b1 to start the I2C transaction.
5. Optional: If interrupt mode is desired, enable target interrupts in INT_ENABLE1_REG.

If number of words to transfer is \leq *FIFO Depth*, set `tr_cmp_en` = 1'b1.

If number of words to transfer is $>$ *FIFO Depth*, set the following: `tx_fifo_aempty_en` = 1'b1 and `tr_cmp_en` = 1'b1.

Other interrupts in this register are disabled.

6. If total number of bytes to transfer $>$ *FIFO Depth*, wait for Transmit FIFO Almost Empty Interrupt.
If polling mode is desired (interrupts are disabled), 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.
In both cases, read also `INT_STATUS2_REG` to ensure that the transfer is good.
 7. Clear Transmit Buffer Almost Empty Interrupt by writing 1'b1 to `INT_STATUS1_REG.tx_fifo_aempty_int`. Clearing all interrupts in this register by writing 8'hFF to `INT_STATUS1_REG` is also okay since we are not interested in other interrupts for this recommended sequence.
 8. Write data to `WR_DATA_REG`, amounting to less than or equal to (*FIFO Depth* – *TX FIFO Almost Empty Flag*).
 9. If there is remaining data to transfer, go back to Step 6.
 10. Wait for Transfer Complete Interrupt.
If polling mode is desired (interrupts are disabled), read `INT_STATUS1_REG` until `tr_cmp_int` asserts.
If interrupt mode is desired, set `INT_ENABLE1_REG` = 8'h80 then wait for interrupt signal to assert. Then, read `INT_STATUS1_REG` and check that `tr_cmp_int` is asserted.
 11. Clear all interrupts by writing 8'hFF to `INT_STATUS1_REG`.
- To perform read transaction of a combined I2C transaction:
12. Set the following `MODE_REG` fields according to the desired transfer mode: `bus_speed_mode`, `addr_mode`, `ack_mode`, `clk_presc_high`. Set the `trx_mode` field to 1'b1 for read transaction.
 13. Set `TGT_BYTE_CNT_REG` according to the number of bytes to transfer.
 14. Set `CONTROL_REG.repeated_start` to 1'b0 and `CONTROL_REG.start` to 1'b1 to start the I2C transaction.
 15. Optional: If interrupt mode is desired, enable target interrupts in `INT_ENABLE1_REG`.
If number of words to transfer is \leq *FIFO Depth*, set `tr_cmp_en` = 1'b1.
If number of words to transfer is $>$ *FIFO Depth*, set the following: `rx_fifo_afull_en` = 1'b1 and `tr_cmp_en` = 1'b1.
Other interrupts in this register are disabled.
 16. If total number of bytes to receive $>$ *FIFO Depth*, wait for Receive FIFO Almost Full Interrupt.
If polling mode is desired (interrupts are disabled), read `INT_STATUS1_REG` until `rx_fifo_afull_int` asserts.
If interrupt mode is desired, simply wait for interrupt signal to assert. Then, read `INT_STATUS1_REG` and check that `rx_fifo_afull_int` is asserted.
In both cases, read also `INT_STATUS2_REG` to ensure that the transfer is good.
 17. Clear Receive FIFO Almost Full Interrupt by writing 1'b1 to `INT_STATUS1_REG.rx_fifo_afull_int`. Clearing all interrupts in this register by writing 8'hFF to `INT_STATUS1_REG` is also okay since we are not interested in other interrupts for this recommended sequence.
 18. Read all data from `RD_DATA_REG`. It is expected the amount of received data is less than or equal to (*FIFO Depth* – *TX FIFO Almost Empty Flag*). Read `FIFO_STATUS_REG` to confirm if Receive FIFO is emptied.
 19. If there is remaining data to receive, go back to Step 16.
 20. Wait for Transfer Complete Interrupt.
If polling mode is desired (interrupts are disabled), read `INT_STATUS1_REG` until `tr_cmp_int` asserts.
If interrupt mode is desired, set `INT_ENABLE1_REG` = 8'h80 then wait for interrupt signal to assert. Then read `INT_STATUS1_REG` and check that `tr_cmp_int` is asserted.
 21. Clear all interrupts by writing 8'hFF to `INT_STATUS1_REG`.
 22. Read all the remaining data from `RD_DATA_REG`.

3. IP Parameter Description

The configurable attributes of the I2C Controller IP are shown in [Table 3.1](#) and are described in [Table 3.2](#). The attributes can be configured through the IP Catalog Module/IP wizard of the Lattice Radiant software.

Table 3.1. Attributes Table

Attribute	Selectable Values	Default	Dependency on Other Attributes
General			
APB Mode Enable	Checked, Unchecked	Checked	—
Remove Tristate Buffers	Checked, Unchecked	Unchecked	—
FIFO			
FIFO Width	8	8	—
FIFO Depth	16, 32, 64, 128, 256	16	—
Implementation of FIFO	EBR, LUT	LUT	—
TX FIFO Almost Empty Flag	1 – 256	2	Less than or equal to FIFO Depth.
RX FIFO Almost Full Flag	1 – 256	14	Less than or equal to FIFO Depth.
Clock			
System Clock Frequency (MHz)	10 - 200	50	—
Desired SCL Frequency (kHz)	100 - 1000	100	—
Clock Prescaler	1 - 2047	247	Calculated based on <i>System Clock Frequency</i> and <i>Desired SCL Frequency</i> .
Actual SCL Frequency (kHz)	Output Value	100	Calculated based on <i>System Clock Frequency</i> and <i>Clock Prescaler</i> .

Table 3.2. Attributes Descriptions

Attribute	Description
General	
APB Mode Enable	Selects memory-mapped interface for register that you access. The unselected interfaces are not available in the generated IP. Checked: APB I/F is enabled. Unchecked: LMMI I/F is enabled.
Remove Tristate Buffers	Removes the tristate buffer to allow you to include additional logic between the IP and the buffers. Checked: Tristate buffers will not be instantiated inside the IP. Unchecked: Tristate buffers will be instantiated inside the IP. This is the default option.
FIFO	
FIFO Width	Specifies the bit width of each data word of the internal FIFO. This is not editable and is fixed to 8 because each I2C data is 8 bits.
FIFO Depth	Specifies the number of FIFO levels, only power of 2 values are allowed.
Implementation of FIFO	Selects the FPGA resource that are used to implement the FIFO: EBR or LUT.
TX FIFO Almost Empty Flag	Specifies the trigger level for asserting INT_STATUS_REG.tx_fifo_aempty_int. Refer to the Mode Register (MODE_REG) section for more details.
RX FIFO Almost Full Flag	Specifies the trigger level for asserting INT_STATUS_REG.rx_fifo_afull_int. Refer to the Mode Register (MODE_REG) section for more details.
Clock	
System Clock Frequency (MHz)	Specifies the frequency of system clock.
Desired SCL Frequency (kHz)	Used for calculating <i>Clock Prescaler</i> value.
Clock Prescaler	Sets frequency of scl_o signal by specifying the reset value of <i>Clock Prescaler</i> registers. This is calculated by dividing <i>System Clock Frequency</i> with $2 \times \text{Desired SCL Frequency}$.
Actual SCL Frequency (kHz)	The frequency of scl_io. This is calculated by dividing <i>System Clock Frequency</i> with $2 \times \text{Clock Prescaler}$.

4. Signal Description

Table 4.1 lists the input and output signals for the I2C Controller IP.

Table 4.1. I2C Controller IP Signal Description

Port Name	I/O	Width	Description
Clock and Reset			
clk_i	In	1	System clock
rst_n_i	In	1	Asynchronous active low reset The reset assertion can be asynchronous but reset negation should be synchronous. When asserted, output ports and registers are forced to their reset values.
Interrupt Port			
int_o	Out	1	Interrupt signal Reset value is 1'b0.
I2C Interface (Remove Tristate Buffers = 0)			
scl_io	In/Out	1	I2C Serial Clock Generated by the I2C Controller to synchronize data transfers. Selected from the user interface. Reset value is weak high (pull-up).
sda_io	In/Out	1	I2C data signal Reset value is weak high (pull-up).
I2C Interface (Remove Tristate Buffers = 1)			
scl_i	In	1	I2C serial clock input Input from tristate buffer.
scl_o	Out	1	I2C serial clock output Output to tristate buffer. This signal is fixed to 1'b0.
scl_oe_o	Out	1	I2C serial clock output enable Active low signal to enable output of a tristate buffer. Reset value is 1'b1.
sda_i	In	1	I2C data signal input Input from tristate buffer.
sda_o	Out	1	I2C data signal output Output to tristate buffer. This signal is fixed to 1'b0.
sda_oe_o	Out	1	I2C data signal output enable Active low signal to enable output of a tristate buffer. Reset value is 1'b1.
LMMI Interface*			
lmmi_request_i	In	1	Start transaction
lmmi_wr_rdn_i	In	1	Write = 1'b1, Read = 1'b0
lmmi_offset_i	In	4	Register offset, starting at offset 0
lmmi_wdata_i	In	8	Input data bus
lmmi_rdata_o	Out	8	Output data bus Reset value is 0.
lmmi_rdata_valid_o	Out	1	Read transaction is complete and lmmi_rdata_o contains valid data. Reset value is 1'b0.
lmmi_ready_o	Out	1	IP is ready to receive a new transaction. This is always asserted (tied to 1'b1).
APB Interface*			
apb_psel_i	In	1	APB Select signal Indicates that the device is selected and a data transfer is required.
apb_paddr_i	In	6	APB Address signal
apb_pwdata_i	In	32	APB Write data signal Bits [31:8] are not used.

Port Name	I/O	Width	Description
apb_pwrite_i	In	1	APB Direction signal Write = 1, Read = 0.
apb_penable_i	In	1	APB Enable signal Indicates the second and subsequent cycles of an APB transfer.
apb_pready_o	Out	1	APB Ready signal Indicates transfer completion. IP uses this signal to extend an APB transfer. Reset value is 1'b0.
apb_pslverr_o	Out	1	APB Error signal Indicates a transfer failure. This signal is tied to 1'b0.
apb_prdata_o	Out	32	APB Read data signal Bits [31:8] are not used. Reset value is 0.

***Note:** Only one of the two interfaces are available as selected by *Interface* attribute.

5. Register Description

5.1. Overview

You can control the I2C Controller IP by writing to and reading from the configuration registers. The I2C Controller IP configuration registers can be performed at the run-time.

Table 5.1 lists the address map and specifies the registers available to you. The offset of each register is dependent on attribute *APB Mode Enable* setting as follows:

- *APB Mode Enable* is Unchecked – the offset increments by 1.
- *APB Mode Enable* is Checked – the offset increments by 4 to allow easy interfacing with the Processor and System Buses. In this mode, each register is 32-bit wide wherein the upper bits [31:8] are reserved and the lower 8 bits [7:0] are described in the next section.

Table 5.1. Registers Address Map

Offset LMMI	Offset APB	Register Name	Access Type	Description
0x0	0x00	WR_DATA_REG	WO	Write Data Register
0x0	0x00	RD_DATA_REG	RO	Read Data Register
0x1	0x04	TARGET_ADDRL_REG	RW	Target Address Lower Register
0x2	0x08	TARGET_ADDRH_REG	RW	Target Address Higher Register
0x3	0x0C	CONTROL_REG	WO	Control Register
0x4	0x10	TGT_BYTE_CNT_REG	RW	Byte Count Register
0x5	0x14	MODE_REG	RW	Mode Register
0x6	0x18	CLK_PRESCAL_REG	RW	Clock Prescaler Low Register
0x7	0x1C	INT_STATUS1_REG	RW1C	First Interrupt Status Register
0x8	0x20	INT_ENABLE1_REG	RW	First Interrupt Enable Register
0x9	0x24	INT_SET1_REG	WO	First Interrupt Set Register
0xA	0x28	INT_STATUS2_REG	RW1C	Second Interrupt Status Register
0xB	0x2C	INT_ENABLE2_REG	RW	Second Interrupt Enable Register
0xC	0x30	INT_SET2_REG	WO	Second Interrupt Set Register
0xD	0x34	FIFO_STATUS_REG	RO	FIFO Status Register
0xE	0x38	SCL_TIMEOUT_REG	RW	SCL Timeout Register
0xF	0x3C	Reserved	RSVD	Reserved Write access is ignored and 0 is returned on read access.

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

The behavior of registers to write and read access is defined by its access type, which is defined in Table 5.2.

Table 5.2. Access Type Definition

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

5.2. Write Data Register (WR_DATA_REG)

Table 5.3 presents the Write Data Register. This is the interface to Transmit FIFO. Writing to WR_DATA_REG pushes a word to Transmit FIFO. When writing to WR_DATA_REG, you should ensure that Transmit FIFO is not full. This can be done by reading FIFO_STATUS_REG. When reset is performed, the contents of Transmit FIFO are not reset but the FIFO control logic is reset. Thus, content is not guaranteed after reset.

Table 5.3. Write Data Register

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

5.3. Read Data Register (RD_DATA_REG)

Table 5.4 presents the Read Data Register. This is the interface to Receive FIFO. After a data is received from I2C bus, the received data is pushed to Receive FIFO. Reading from RD_DATA_REG pops a word from Receive FIFO. You should ensure that Receive FIFO has data before reading RD_DATA_REG, data is not guaranteed when this register is read during Receive FIFO empty condition. On the other hand, if Receive FIFO is full but I2C Controller continues to receive data, new data is lost. Read FIFO_STATUS_REG to determine the status of Receive FIFO. Similar to Transmit FIFO, the reset value of Receive FIFO is also not guaranteed after reset.

Table 5.4. Read Data Register

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

5.4. Target Address Registers (TARGET_ADDRL_REG, TARGET_ADDRH_REG)

The Target Address Lower Register (TARGET_ADDRL_REG) shown in Table 5.5 is a 7-bit Target address. This is used for 7-bit and 10-bit addressing mode as follows:

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

Table 5.5. Target Address Lower Register

Field	Name	Access	Width	Reset
[7]	reserved	RSVD	1	—
[6:0]	target_addr_l_reg	RW	7	7'h00

The Target Address Higher Register (TARGET_ADDRH_REG) shown in Table 5.6 is the upper 3 bits of 10-bit Target address. This is not used in 7-bit addressing mode.

Table 5.6. Target Address Higher Register

Field	Name	Access	Width	Reset
[7:3]	reserved	RSVD	5	—
[2:0]	target_addr_h_reg	RW	3	3'h0

5.5. Control Register (CONTROL_REG)

Table 5.7 presents a summary of Control Register.

Table 5.7. Control Register

Field	Name	Access	Width	Reset
[7]	reserved	RSVD	1	—
[6]	rx_fifo_reset	WO	1	1'b0
[5]	tx_fifo_reset	WO	1	1'b0
[4]	reserved	RSVD	1	—

Field	Name	Access	Width	Reset
[3]	repeated_start	RW	1	1'b0
[2]	reset	WO	1	1'b0
[1]	abort	WO	1	1'b0
[0]	start	WO	1	1'b0

- **rx_fifo_reset**
Resets the Receive FIFO logic, effectively flushing the contents. This is write-only bit because it has an auto clear feature; it is cleared to 1'b0 after 1 clock cycle.
1'b0 – No action.
1'b1 – Resets the Receive FIFO.
- **tx_fifo_reset**
Resets the Transmit FIFO logic, effectively flushing the contents. This is write-only bit because it has an auto clear feature; it is cleared to 1'b0 after 1 clock cycle.
1'b0 – No action.
1'b1 – Resets the Transmit FIFO.
- **repeated_start**
Repeated Start. Causes I2C Controller to omit the generation of a stop condition following the completion of a transaction. Subsequent transaction is a repeated start.
1'b0 – No action.
1'b1 – Next transaction uses a repeated start.
- **reset**
Reset. Resets I2C Controller IP. The registers and LMMI interface are not affected by this reset.
1'b0 – No action.
1'b1 – Resets I2C Controller IP.
- **abort**
Abort. Stops an I2C transaction in progress. After aborted process, INT_STATUS2_REG.abort_ack asserts.
1'b0 – No action.
1'b1 – Stops an I2C transaction in progress.
- **start**
Start. Starts an I2C transaction. This bit is written when all registers are programmed for the transaction.
1'b0 – No action.
1'b1 – Starts an I2C transaction.

5.6. Target Byte Count Register (TGT_BYTE_CNT_REG)

Table 5.8 presents a summary of Byte Count Register. The desired number of bytes to transfer (read/write) in I2C bus should be written to this register. This is used for Transfer Complete interrupt generation – asserts when the target byte count is achieved.

Table 5.8. Target Byte Count Register

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

5.7. Mode Register (MODE_REG)

Table 5.9 presents a summary of Mode Register (MODE_REG).

Table 5.9. Mode Register

Field	Name	Access	Width	Reset
[7:6]	bus_speed_mode	RW	2	2'b00
[5]	addr_mode	RW	1	1'b0
[4]	reserved	RSVD	1	–
[3]	trx_mode	RW	1	1'b0
[2:0]	clk_presc_high	RW	3	Clock Prescaler[10:8]

- **bus_speed_mode**
Bus Speed Mode. Specifies the bus speed mode. This information is used for generating the setup time for repeated START condition ($t_{SU;STA}$) and setup time for STOP condition ($t_{SU;STO}$). The values below are estimated based on *System Clock Frequency* attribute.
2'b00 – Standard mode: $t_{SU;STA} = t_{SU;STO} \sim 4.7 \mu s$
2'b01 – Fast mode: $t_{SU;STA} = t_{SU;STO} \sim 1.3 \mu s$
2'b10 – Fast mode Plus: $t_{SU;STA} = t_{SU;STO} \sim 0.5 \mu s$
- **addr_mode**
Address Mode. Selects the Addressing Mode.
1'b0 – 7-bit address mode
1'b1 – 10-bit address mode
- **trx_mode**
Transmit/Receive Mode. Sets the read or write operation on the I2C bus.
1'b0 – Write I2C transaction
1'b1 – Read I2C transaction
- **clk_presc_high**
Clock Prescaler High Register. The upper three bits of the Clock Prescaler High Register.

5.8. Clock Prescaler Low Register (CLK_PRESC_L_REG)

Clock Prescaler Low Register (CLK_PRESC_L_REG) shown in Table 5.10 sets the lower byte of the Clock Prescaler that is used to generate SCL from the clock. The upper three bits are located in the Mode Register. The reset value is set by the *Clock Prescaler* attribute.

Table 5.10. Clock Prescaler Low Register

Field	Name	Access	Width	Reset
[7:0]	clk_presc_low	RW	8	Clock Prescaler[7:0]

5.9. Interrupt Status Registers (INT_STATUS1_REG, INT_STATUS2_REG)

Table 5.11 and Table 5.12 present Interrupt Status Registers (INT_STATUS1_REG and INT_STATUS2_REG). They contain all the interrupts currently pending in the I2C Controller IP. When an interrupt bit asserts, it remains asserted until you clear it by writing 1'b1 to the corresponding bit.

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

Table 5.11. Interrupt Status First Register

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

- **tr_cmp_int**
Transfer Complete Interrupt Status. This interrupt status bit asserts when the number of bytes transferred in I2C interface is equal to TGT_BYTE_CNT.byte_cnt.
1'b0 – No interrupt
1'b1 – Interrupt pending
- **tx_fifo_full_int**
Transmit FIFO Full Interrupt Status. This interrupt status bit asserts when Transmit FIFO changes from not full state to full state.
1'b0 – No interrupt
1'b1 – Interrupt pending
- **tx_fifo_aempty_int**
Transmit FIFO Almost Empty Interrupt Status. This interrupt status bit asserts when the amount of data words in Transmit FIFO changes from '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 Empty Interrupt Status. This interrupt status bit asserts when the last data in Transmit FIFO is popped-out, causing the FIFO to become empty.
1'b0 – No interrupt
1'b1 – Interrupt pending
- **rx_fifo_full_int**
Receive FIFO Full Interrupt Status. This interrupt status bit asserts when RX FIFO full status changes from not full to full state.
1'b0 – No interrupt
1'b1 – Interrupt pending
- **rx_fifo_afull_int**
Receive FIFO Almost Full Interrupt Status. This interrupt status bit asserts when the amount of data words in Receive FIFO changes from '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 Ready Interrupt Status. This interrupt status bit asserts when Receive FIFO is empty and receives a data word from I2C interface.
1'b0 – No interrupt
1'b1 – Interrupt pending

Table 5.12. Interrupt Status Second Register

Field	Name	Access	Width	Reset
[7:4]	reserved	RSVD	4	—
[3]	nack_error_int	RW1C	1	1'b0
[2]	abort_ack_int	RW1C	1	1'b0
[1]	arb_lost_int	RW1C	1	1'b0
[0]	timeout_int	RW1C	1	1'b0

- **nack_error_int**
NACK Error Interrupt Status. This interrupt status bit asserts when NACK is received, expecting an ACK during data phase and address phase.
1'b0 – No interrupt
1'b1 – Interrupt pending
- **abort_ack_int**
Transaction Abort Acknowledgement Interrupt Status. This interrupt status bit asserts when an I2C Controller IP aborts an ongoing transaction because you write 1'b1 to CONTROL_REG.abort. The interrupt asserts when NACK has been issued.
1'b0 – No interrupt
1'b1 – Interrupt pending
- **arb_lost_int**
Arbitration Lost Interrupt Status. This interrupt status bit asserts when I2C Controller IP loses arbitration.
1'b0 – No interrupt
1'b1 – Interrupt pending
- **timeout_int**
SCL Timeout Interrupt Status. This interrupt status of bit asserts when scl_io is hold for SCL_TIMEOUT_REG.timeout_val times the programmed SCL low period.
1'b0 – No interrupt
1'b1 – Interrupt pending

5.10. Interrupt Enable Registers (INT_ENABLE1_REG, INT_ENABLE2_REG)

Table 5.13 and Table 5.14 present the summary of Interrupt Enable Registers. They corresponds to interrupts status bits in INT_STATUS1_REG and INT_STATUS2_REG. They do not affect the contents of the INT_STATUS1_REG and INT_STATUS2_REG. If one of the INT_STATUS1_REG/ INT_STATUS2_REG bits asserts, and the corresponding bit of INT_ENABLE1_REG/ INT_ENABLE2_REG is 1'b1, the interrupt signal int_o asserts.

Table 5.13. Interrupt Enable First Register

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

- **tr_cmp_en**
Transfer Complete Interrupt Enable. Interrupt enabled bit corresponded to Transfer Complete Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled

- **tx_fifo_full_en**
Transmit FIFO Full Interrupt Enable. Interrupt enabled bit corresponded to Transmit FIFO Full Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled
- **tx_fifo_aempty_en**
Transmit FIFO Almost Empty Interrupt Enable. Interrupt enabled bit corresponded to Transmit FIFO Almost Empty Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled
- **tx_fifo_empty_en**
Transmit FIFO Empty Interrupt Enable. Interrupt enabled bit corresponded to Transmit FIFO Empty Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled
- **rx_fifo_full_en**
Receive FIFO Full Interrupt Enable. Interrupt enabled bit corresponded to Receive FIFO Full Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled
- **rx_fifo_afull_en**
Receive FIFO Almost Full Interrupt Enable. Interrupt enabled bit corresponded to Receive FIFO Almost Full Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled
- **rx_fifo_ready_en**
Receive FIFO Ready Interrupt Enable. Interrupt enabled bit corresponded to Receive FIFO Ready Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled

Table 5.14. Interrupt Enable Second Register

Field	Name	Access	Width	Reset
[7:4]	reserved	RSVD	4	—
[3]	nack_error_en	RW	1	1'b0
[2]	abort_ack_en	RW	1	1'b0
[1]	arb_lost_en	RW	1	1'b0
[0]	timeout_en	RW	1	1'b0

- **nack_error_en**
NACK Error Interrupt Enable. Interrupt enabled bit corresponded to NACK Error Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled
- **abort_ack_en**
Transaction Abort Acknowledgement Interrupt Enable. Interrupt enabled bit corresponded to Transaction Abort Acknowledgement Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled
- **arb_lost_en**
Arbitration Lost Interrupt Enable. Interrupt enabled bit corresponded to Arbitration Lost Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled

- **timeout_en**
SCL Timeout Interrupt Enable. Interrupt enabled bit corresponded to SCL Timeout Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled

5.11. Interrupt Set Registers (INT_SET1_REG, INT_SET2_REG)

Table 5.15 and Table 5.16 present the summary of Interrupt Set Registers. Writing 1'b1 to a register bit in INT_SET1_REG or INT_SET2_REG asserts the corresponding interrupts status bit in INT_STATUS1_REG or INT_STATUS2_REG while writing 1'b0 is ignored. This is intended for testing purposes only.

Table 5.15. Interrupt Set First Register

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

- **tr_cmp_set**
Transfer Complete Interrupt Set. Interrupt set bit corresponded to Transfer Complete Interrupt Status.
1'b0 – No action
1'b1 – Asserts INT_STATUS1_REG.tr_cmp_int
- **tx_fifo_full_set**
Transmit FIFO Full Interrupt Set. Interrupt set bit corresponded to Transmit FIFO Full Interrupt Status.
1'b0 – No action
1'b1 – Asserts INT_STATUS1_REG.tx_fifo_full_int
- **tx_fifo_aempty_set**
Transmit FIFO Almost Empty Interrupt Set. Interrupt set bit corresponded to Transmit FIFO Almost Empty Interrupt Status.
1'b0 – No action
1'b1 – Asserts INT_STATUS1_REG.tx_fifo_aempty_int
- **tx_fifo_empty_set**
Transmit FIFO Empty Interrupt Set. Interrupt set bit corresponded to Transmit FIFO Empty Interrupt Status.
1'b0 – No action
1'b1 – Asserts INT_STATUS1_REG.tx_fifo_empty_int
- **rx_fifo_full_set**
Receive FIFO Full Interrupt Set. Interrupt set bit corresponded to Receive FIFO Full Interrupt Status.
1'b0 – No action
1'b1 – Asserts INT_STATUS1_REG.rx_fifo_full_int
- **rx_fifo_afull_set**
Receive FIFO Almost Full Interrupt Set. Interrupt set bit corresponded to Receive FIFO Almost Full Interrupt Status.
1'b0 – No action
1'b1 – Asserts INT_STATUS1_REG.rx_fifo_afull_int
- **rx_fifo_ready_set**
Receive FIFO Ready Interrupt Set. Interrupt set bit corresponded to Receive FIFO Ready Interrupt Status.
1'b0 – No action
1'b1 – Asserts INT_STATUS1_REG.rx_fifo_ready_int

Table 5.16. Interrupt Set Second Register

Field	Name	Access	Width	Reset
[7:4]	reserved	RSVD	4	—
[3]	nack_error_set	WO	1	1'b0
[2]	abort_ack_set	WO	1	1'b0
[1]	arb_lost_set	WO	1	1'b0
[0]	timeout_set	WO	1	1'b0

- **nack_error_set**
NACK Error Interrupt Set. Interrupt set bit corresponded to NACK Error Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled
- **abort_ack_set**
Transaction Abort Acknowledgement Interrupt Set. Interrupt set bit corresponded to Transaction Abort Acknowledgement Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled
- **arb_lost_set**
Arbitration Lost Interrupt Set. Interrupt set bit corresponded to Arbitration Lost Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled
- **timeout_set**
SCL Timeout Interrupt Set. Interrupt set bit corresponded to SCL Timeout Interrupt Status.
0 – Interrupt disabled
1 – Interrupt enabled

5.12. FIFO Status Register (FIFO_STATUS_REG)

FIFO Status Register reflects the status of Transmit FIFO and Receive FIFO as shown in [Table 5.17](#).

Table 5.17. FIFO Status Register

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

- **tx_fifo_full**
Transmit FIFO Full. This bit reflects the full condition of Transmit FIFO.
1'b0 – Transmit FIFO is not full
1'b1 – Transmit FIFO is full
- **tx_fifo_aempty**
Transmit FIFO Almost Empty. This bit reflects the almost empty condition of Transmit FIFO.
1'b0 – Data words in Transmit FIFO is greater than 'TX FIFO Almost Empty Flag' attribute
1'b1 – Data words in Transmit FIFO is less than or equal to 'TX FIFO Almost Empty Flag' attribute

- **tx_fifo_empty**
Transmit FIFO Empty. This bit reflects the empty condition of Transmit FIFO.
1'b0 – Transmit FIFO is not empty – has at least 1 data word
1'b1 – Transmit FIFO is empty
- **rx_fifo_full**
Receive FIFO Full. This bit reflects the full condition of Receive FIFO.
1'b0 – Receive FIFO is not full
1'b1 – Receive FIFO is full
- **rx_fifo_afull**
Receive FIFO Full. This bit reflects the almost full condition of Receive FIFO.
1'b0 – Data words in Receive FIFO is less than 'RX FIFO Almost Full Flag' attribute
1'b1 – Data words in Receive FIFO is greater than or equal to 'RX FIFO Almost Full Flag' attribute
- **rx_fifo_empty**
Receive FIFO Full. This bit reflects the empty condition of Receive FIFO.
1'b0 – Receive FIFO is not empty – has at least 1 data word
1'b1 – Receive FIFO is empty

5.13. SCL Timeout Register (SCL_TIMEOUT_REG)

Table 5.18 presents the SCL Timeout Register (CLK_TIMEOUT_REG). SCL timeout occurs when scl_io is held low for the number of clock cycles approximately equal to SCL_TIMEOUT_REG.timeout_val multiplied by Clock Prescaler Register (MODE_REG.clk_presc_high, CLK_PRESCAL_REG.clk_presc_low). SCL Timeout Interrupt Status bit asserts when SCL timeout occurs. Timeout detection is disabled when CLK_TIMEOUT_REG.timeout_val is set to 8'h00 or 8'h01.

Table 5.18. SCL Timeout Register

Field	Name	Access	Width	Reset
[7:0]	timeout_val	RW	8	8'h00

6. Example Design

The I2C Controller example design allows you to compile, simulate, and test the I2C Controller IP on the following Lattice evaluation boards:

- [CertusPro-NX Evaluation Board](#)
- [MachXO5-NX Development Board](#)

6.1. Example Design Supported Configuration

Note: In the table below, ✓ refers to a checked option in the I2C Controller IP example design.

Table 6.1. I2C Controller IP Configuration Supported by the Example Design

Attribute	I2C Controller
General	
APB Mode Enable	Checked
Remove Tristate Buffers	Unchecked
FIFO	
FIFO Width	8 (Display only)
FIFO Depth	16
Implementation of FIFO	EBR
TX FIFO Almost Empty Flag	2
RX FIFO Almost Full Flag	14
Clock	
System Clock Frequency (MHz)	50
Desired SCL Frequency (kHz)	100
Clock Prescaler	247 (Display only)
Actual SCL Frequency (kHz)	100.20040080160321 (Display only)

6.2. Overview of the Example Design and Features

The example design discussed in this section is created using the *RISC-V MC SoC Project* template in the [Lattice Propel Design Environment](#). The generated project includes the following components:

- Processor – RISC-V (MC w/ PIC/T)
- GPIO
- Asynchronous SRAM
- UART – Serial port
- PLL
- Glue Logic

I2C Controller and I2C Target are instantiated and connected in the project as shown in [Figure 6.1](#). In this example, I2C Controller and I2C Target are instantiated in the same system. In actual hardware or use case, you can connect the I2C Controller to external I2C Target devices.

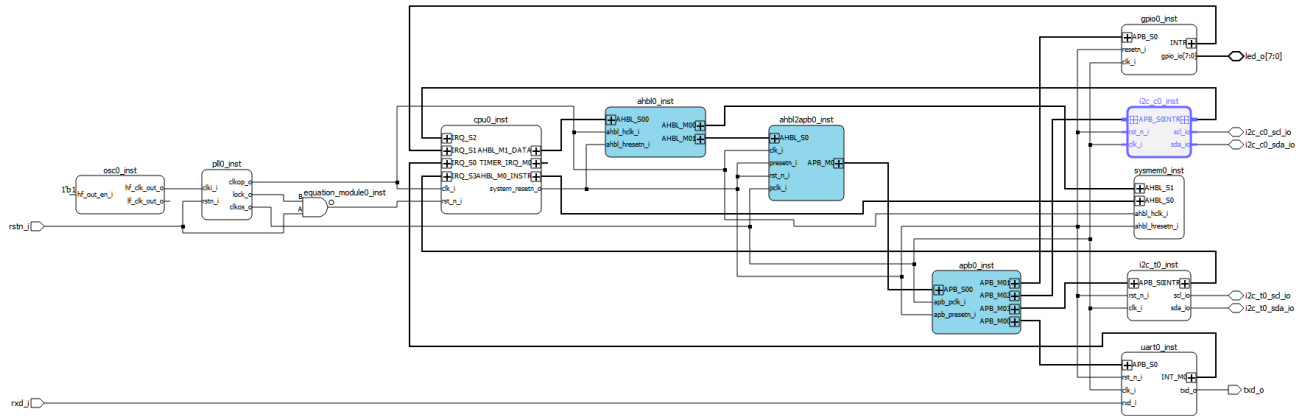


Figure 6.1. I2C Controller IP in Propel SoC Project

An Embedded C/C++ Project is also created in the Propel software to enable developing and debugging application code for different IP features. I2C Controller features can be tested by sending I2C Commands from the I2C Controller to the I2C Target. Runtime configuration of IP and feature testing can be done through C-Code Test Routine. Figure 6.2 shows an example routine to send write data from I2C Controller. This sample test routine can be found in the driver file included in the generated IP.

```

378 uint8_t i2c_master_write(struct i2cm_instance * this_i2cm,
379                          uint16_t address,
380                          uint8_t data_size, uint8_t * data_buffer)
381 {
382     uint8_t data_count = 0;
383     uint8_t i2c_status = 0;
384     uint8_t status = 0;
385     uint8_t i2c_int2 = 0;
386     uint8_t fifo_status = 0;
387
388     if (NULL == this_i2cm || NULL == data_buffer )
389     {
390         return 1;
391     }
392
393     // config the register before issue the transaction
394     reg_8b_write(this_i2cm->base_address | REG_BYTE_CNT, data_size);
395
396     reg_8b_write(this_i2cm->base_address | REG_SLAVE_ADDR_LOW,
397                 address & 0x7F);
398
399     if(this_i2cm->addr_mode==I2CM_ADDR_10BIT_MODE) // 10-bit mode
400     {
401         reg_8b_write(this_i2cm->base_address | REG_SLAVE_ADDR_HIGH,
402                     (address>>8) & 0x03);
403     }
404
405     // set to write mode
406     reg_8b_modify(this_i2cm->base_address | REG_MODE, I2C_TXRX_MODE, 0);
407
408     // clear status bits
409     reg_8b_read(this_i2cm->base_address | REG_INT_STATUS1, &status);
410     reg_8b_write(this_i2cm->base_address | REG_INT_STATUS1, status);
411     reg_8b_read(this_i2cm->base_address | REG_INT_STATUS2, &i2c_int2);
412     reg_8b_write(this_i2cm->base_address | REG_INT_STATUS2, i2c_int2);
413
414     while (data_count < data_size)
415     {
416         // check tx fifo level,
417         reg_8b_read(this_i2cm->base_address | REG_INT_STATUS1, &status);
418
419         // if tx fifo is full, stop loading fifo for now, resume in interrupt or polling loop
420         if ( (status & TX_FIFO_FULL_MASK) != 0 ) {
421             break;
422         }
423
424         reg_8b_write(this_i2cm->base_address | REG_DATA_BUFFER, *data_buffer); // push the data into tx buffer
425
426         // update the counter and data buffer pointer
427         data_buffer++;
428         data_count++;
429     }
430
431     if(this_i2cm->state == I2CM_STATE_IDLE)
432     {
433         // start the transaction
434         this_i2cm->state = I2CM_STATE_WRITE;
435         reg_8b_write(this_i2cm->base_address | REG_CONFIG, I2C_START);
436     }

```

Figure 6.2. Sample C-Code Test Routine

6.3. Design Components Example

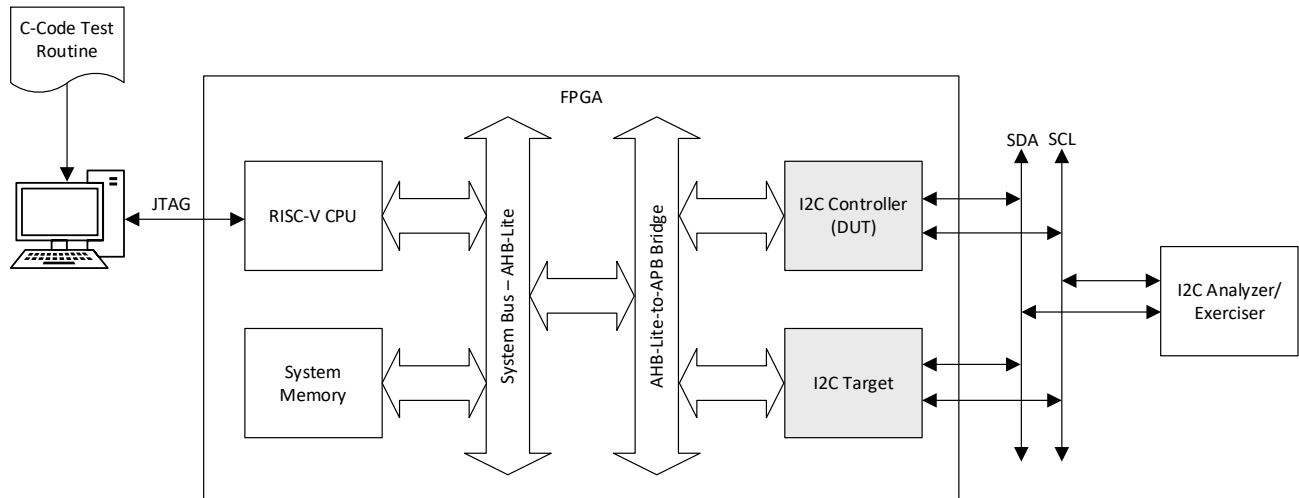


Figure 6.3. I2C Controller Example Design Block Diagram

The I2C Controller example design includes the following blocks:

- RISC-V CPU – Passes the C-Code Test Routine from system memory to system bus. Handles interrupts.
- Memory – Contains commands for testing.
- System Bus – AHB-Lite systems bus for transfers between memory and IP
- I2C Controller IP – IP instance connected to I2C bus (SCL and SDA)
- I2C Target Device(s) – I2C Target IP instance and I2C Analyzer/Exerciser that acts as the Target

6.4. Generating the Example Design

Refer to the [Lattice Propel SDK User Guide](#) for more details on the Lattice Propel software.

1. Launch Lattice Propel software and set your workspace directory.
2. In the Propel software, create a new Lattice SoC Design Project by navigating to **File > New > Lattice SoC Design Project**.
3. The **Create SoC Project** window will open. In the **Device Select** section, indicate the correct details of the device or board that you are using. In [Figure 6.4](#), device is set to LFMX05-25-9BBG400C since MachXO5-NX Development Board is used in the hardware testing. In **Template Design** section, choose **RISC-V MC SoC Project**. Click **Finish**.

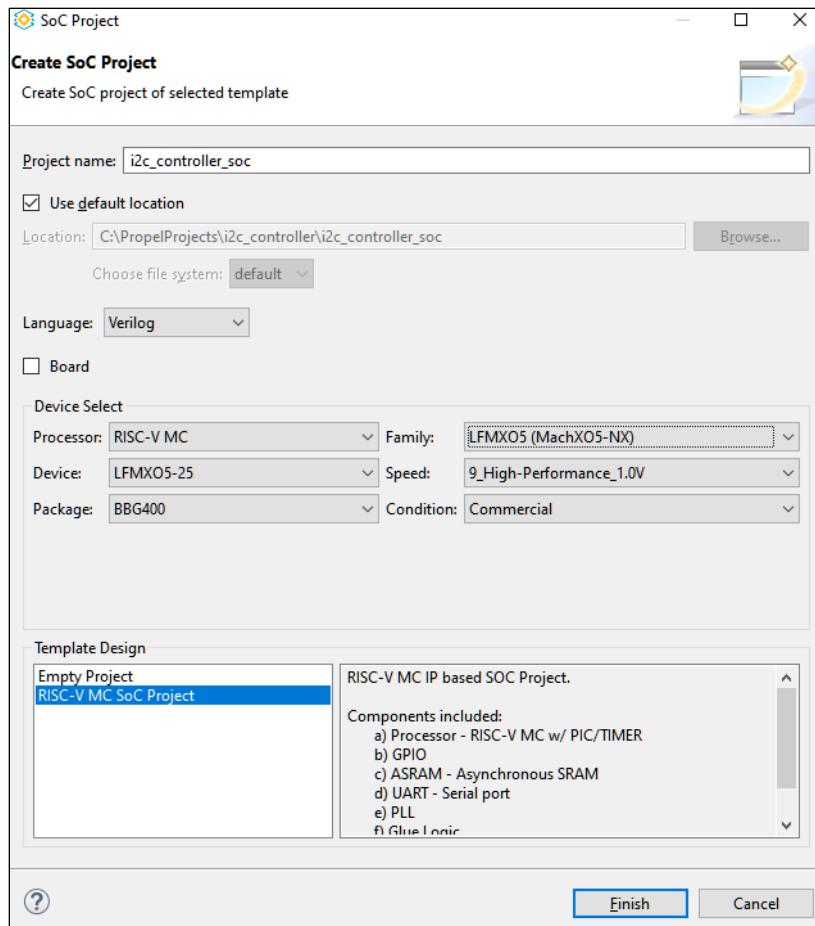



Figure 6.4. Create SoC Project

4. Run Propel Builder by clicking the  icon or navigate to **LatticeTools > Open Design** in Propel Builder. The Propel Builder will open and load the design template.
5. In the **IP Catalog** tab, instantiate the I2C Controller IP. Refer to the [Generating and Instantiating the IP](#) section for more details. In this example, there is one instance of the I2C Controller IP:
See the [Example Design Supported Configuration](#) section for the corresponding parameter settings.
6. After generating the IP, the **Define Instance** window will open. Modify the instance name if needed, then click **OK**.

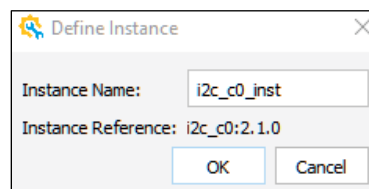



Figure 6.5. Define Instance

7. Connect the instantiated IPs to the system. Refer to [Figure 6.1](#) for the connections used in this IP. You will need to update other components of the system for clock and reset sources, interrupt, and bus interface.
8. Click the  icon or navigate to **Design > Run Radiant** to launch the Lattice Radiant Software.
9. Update your constraints file accordingly and generate the programming file.

10. In the Lattice Propel software, build your SoC project to generate the system environment needed for the embedded C/C++ project. Select your SoC project then navigate to **Project > Build Project**.
11. Check the build result from the **Console** view.

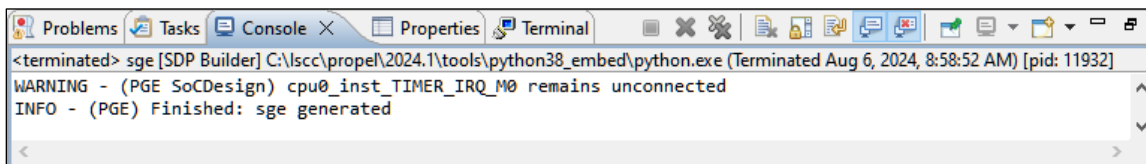


Figure 6.6. Build SoC Project Result

12. Generate a new Lattice C/C++ project by navigating to **File > New > Lattice C/C++ Project**. Update your **Project name**, click **Next**, and then click **Finish**.

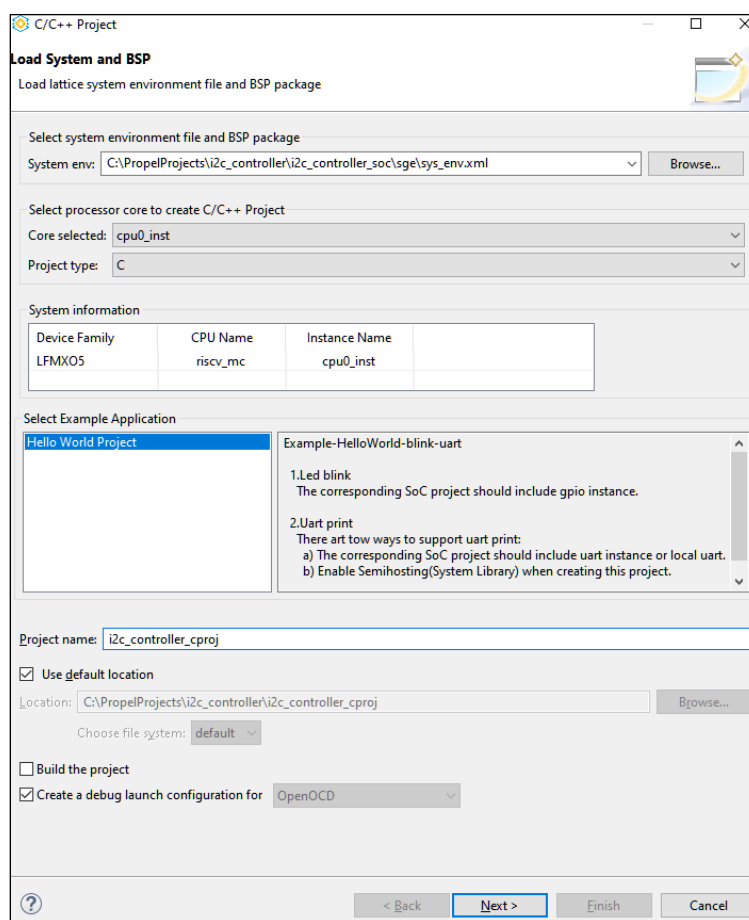
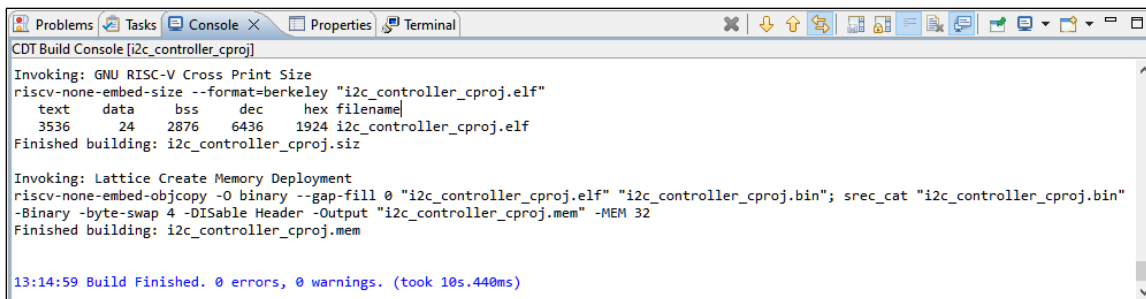


Figure 6.7. Lattice C/C++ Design Project

13. Select your C/C++ project then select **Project > Build**.
14. Check the build result from the **Console** view.



```
CDT Build Console [i2c_controller_cproj]
Invoking: GNU RISC-V Cross Print Size
riscv-none-embed-size --format=berkeley "i2c_controller_cproj.elf"
  text    data    bss     dec      hex filename
  3536     24    2876    6436    1924 i2c_controller_cproj.elf
Finished building: i2c_controller_cproj.siz

Invoking: Lattice Create Memory Deployment
riscv-none-embed-objcopy -O binary --gap-fill 0 "i2c_controller_cproj.elf" "i2c_controller_cproj.bin"; srec_cat "i2c_controller_cproj.bin"
-Binary -byte-swap 4 -DISable Header -Output "i2c_controller_cproj.mem" -MEM 32
Finished building: i2c_controller_cproj.mem

13:14:59 Build Finished. 0 errors, 0 warnings. (took 10s.440ms)
```

Figure 6.8. Build C/C++ Project Result

15. This environment is now ready for running your tests on the device. Refer to the *Running Demo on MachXO3D Breakout Board – Hello World* section of the [Lattice Propel SDK User Guide](#) for step-by-step guide.

6.5. Hardware Testing

6.5.1. Hardware Testing Setup

Download the generated bitstream file from the [Generating the Example Design](#) section to the MachXO5-NX Development Board via the Lattice Radiant Programmer.

6.5.2. Expected Output

Below is a sample waveform captured via I2C Analyzer/Exerciser tool.

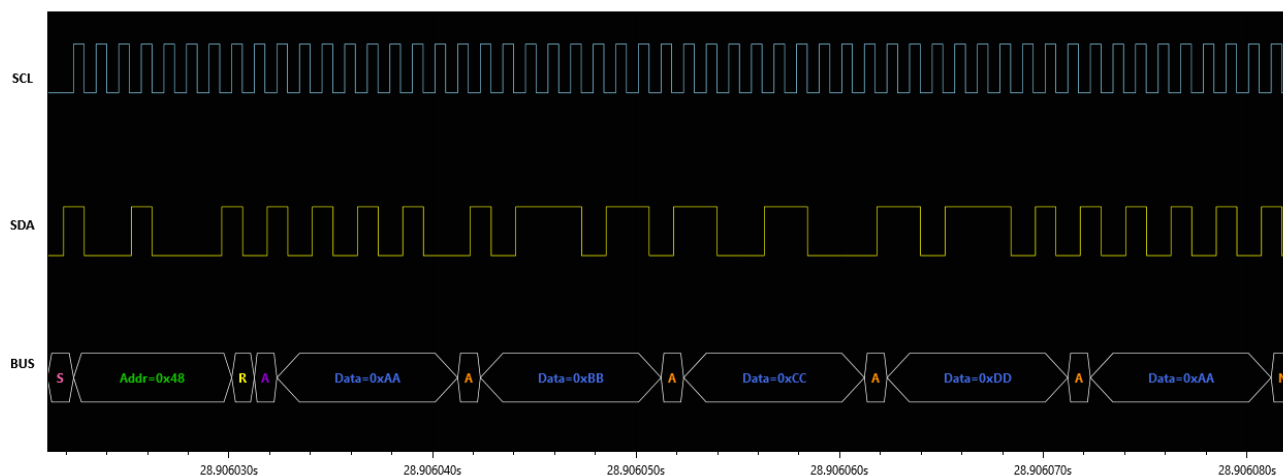


Figure 6.9 Sample I2C Read Command Sent by I2C Controller

7. Designing with the IP

This section provides information on how to generate the IP Core using the Lattice Radiant software and how to run simulation and synthesis. For more details on the Lattice Radiant software, refer to the [Lattice Radiant Software User Guide](#).

Note: The screenshots provided are for reference only. Details may vary depending on the version of the IP or software being used. If there have been no significant changes to the GUI, a screenshot may reflect an earlier version of the IP.

7.1. Generating and Instantiating the IP

The Lattice Radiant software allows you to customize and generate modules and IPs and integrate them into the device architecture. The procedure for generating the I2C Controller IP in Lattice Radiant software is described below.

To generate the I2C Controller IP:

1. Create a new Lattice Radiant software project or open an existing project.
2. In the **IP Catalog** tab, double-click on **I2C_Controller** under **IP, Processors, Controllers, and Peripherals** category. The **Module/IP Block Wizard** opens as shown in [Figure 7.1](#). Enter values in the **Component name** and the **Create in** fields and click **Next**.

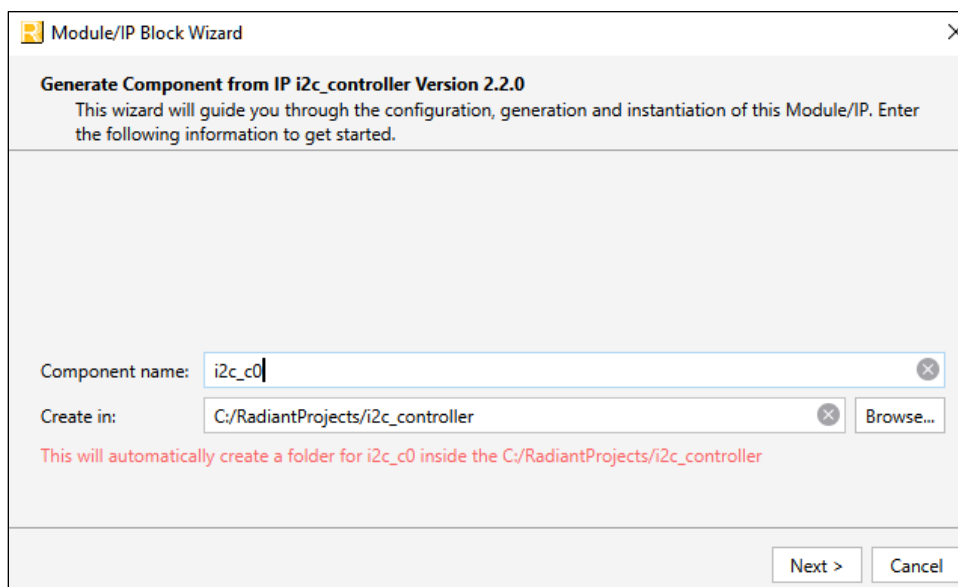


Figure 7.1. Module/IP Block Wizard

3. In the module dialog box of the **Module/IP Block Wizard** window, customize the selected I2C Controller IP using drop-down menus and check boxes. As a sample configuration, see [Figure 7.2](#). For configuration options, see the [IP Parameter Description](#) section.

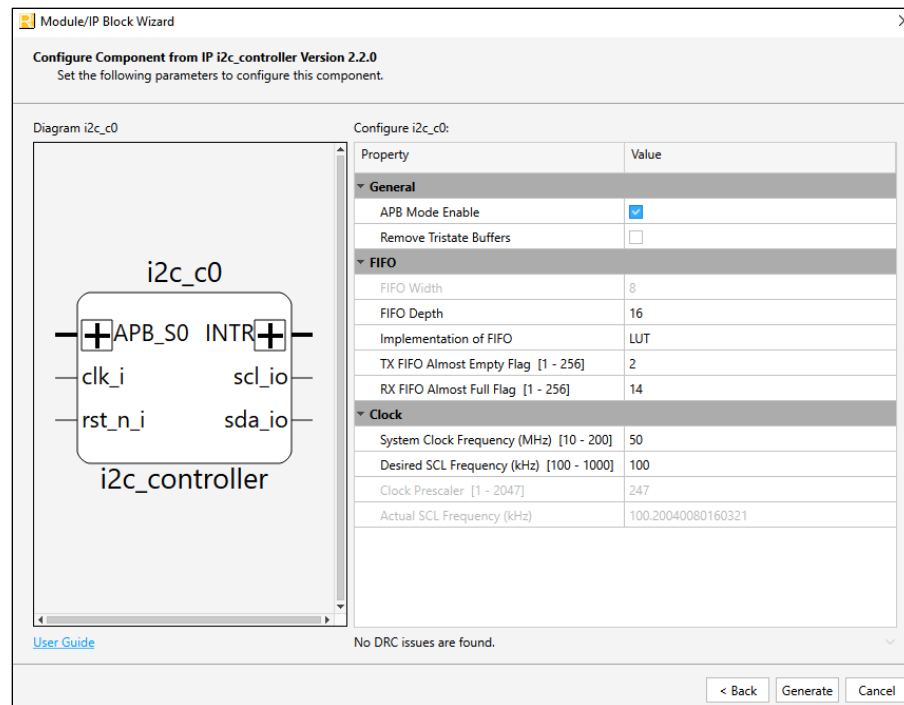


Figure 7.2. Configure User Interface of I2C Controller IP

- Click **Generate**. The **Check Generating Result** dialog box opens, showing design block messages and results as shown in Figure 7.3.

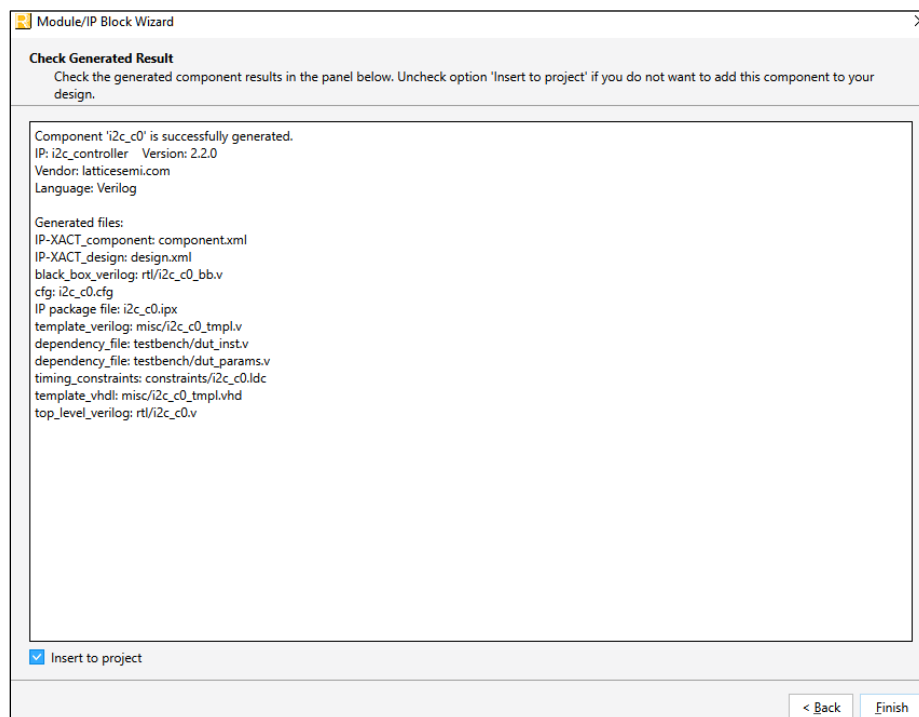


Figure 7.3. Check Generating Result

- Click the **Finish** button. All the generated files are placed under the directory paths in the **Create in** and the **Component name** fields shown in Figure 7.1.

7.1.1. Generated Files and File Structure

The generated I2C Controller IP package includes the closed-box (<Component name>_bb.v) and instance templates (<Component name>_tmpl.v/vhd) that can be used to instantiate the IP in a top-level design. An example RTL top-level reference source file (<Component name>.v) that can be used as an instantiation template for the IP is also provided. You may also use this top-level reference as the starting template for the top-level for their complete design. The generated files are listed in [Table 7.1](#).

Table 7.1. Generated File List

Attribute	Description
<Component name>.ipx	This file contains the information on the files associated to the generated IP.
<Component name>.cfg	This file contains the parameter values used in IP configuration.
component.xml	Contains the ipxact:component information of the IP.
design.xml	Documents the configuration parameters of the IP in IP-XACT 2014 format.
rtl/<Component name>.v	This file provides an example RTL top file that instantiates the IP.
rtl/<Component name>_bb.v	This file provides the synthesis closed-box.
misc/<Component name>_tmpl.v misc /<Component name>_tmpl.vhd	These files provide instance templates for the IP.

7.2. Design Implementation

Completing your design includes additional steps to specify analog properties, pin assignments, and timing and physical constraints. You can add and edit the constraints using the Device Constraint Editor or by manually creating a PDC File.

A post-synthesis constraint file (.pdc) contains both timing and non-timing constraint. Constraints that are added using the Device Constraint Editor are saved to the active .pdc file. The active post-synthesis design constraint file is then used as input for post-synthesis processes.

7.2.1. Device Constraint Editor

Refer to the [Lattice Radiant Software](#) User Guide for more information on how to use the device constraint editor.

7.2.2. Manual PDC File Creation

Add the post-synthesis constraint file (.pdc) in the Lattice Radiant software and define the I/O pins according to the schematic design for ports defined in your design. You can define different types of constraints such as pins, clocks, and other timing paths.

7.3. Timing Constraints

The timing constraints are based on the clock frequency used. The timing constraints for the IP are defined in relevant constraint files. The example below shows the IP timing constraints generated for I2C Controller IP.

You need to provide proper timing and physical design constraints to ensure that your design meets the desired performance goals on the FPGA. Add the content of the following IP constraint file to your design constraints:

```
<IP_Instance_Path>/<IP_Instance_Name>/constraints/<IP_Instance_Name>.ldc
```

The constraint file has been verified during IP evaluation with the IP instantiated directly in the top-level module. You can modify the constraints in this file with thorough understanding of the effect of each constraint.

To use this constraint file:


1. Copy the contents of <IP_Instance_Name>.ldc to the top-level design constraint for post-synthesis.
2. Remove create_clock constraints if the I2C Controller is instantiated in another module.
3. Remove ldc_set_port for scl_io and sda_io if REMOVE_TRISTATE buffer is enabled.

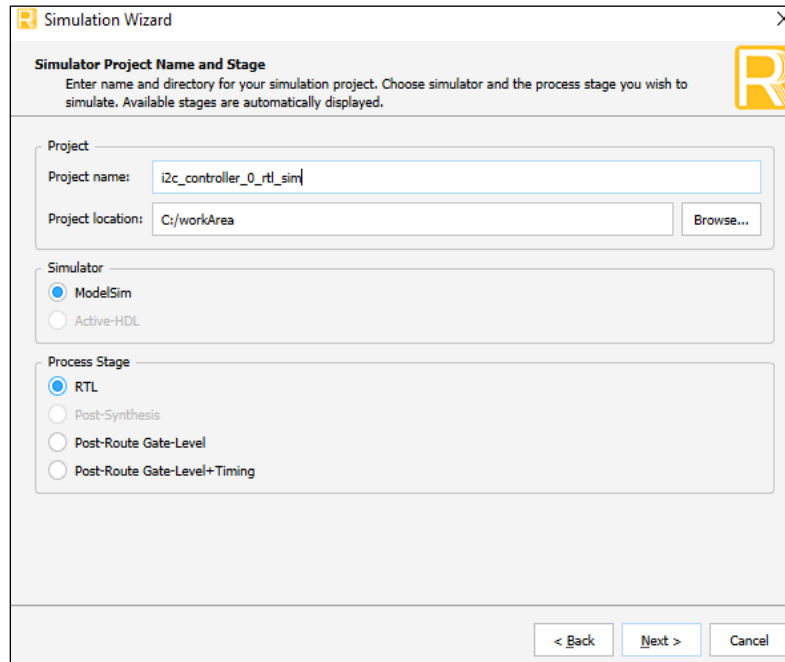
Refer to [Lattice Radiant Timing Constraints Methodology \(FPGA-AN-02059\)](#) for details on how to constraint your design.

7.4. Running Functional Simulation

You can run functional simulation after the IP is generated.

To run functional simulation:

1. Click the  button located on the **Toolbar** to initiate the **Simulation Wizard** shown in [Figure 7.4](#).



Simulation Wizard

Simulator Project Name and Stage
Enter name and directory for your simulation project. Choose simulator and the process stage you wish to simulate. Available stages are automatically displayed.

Project
Project name:
Project location:

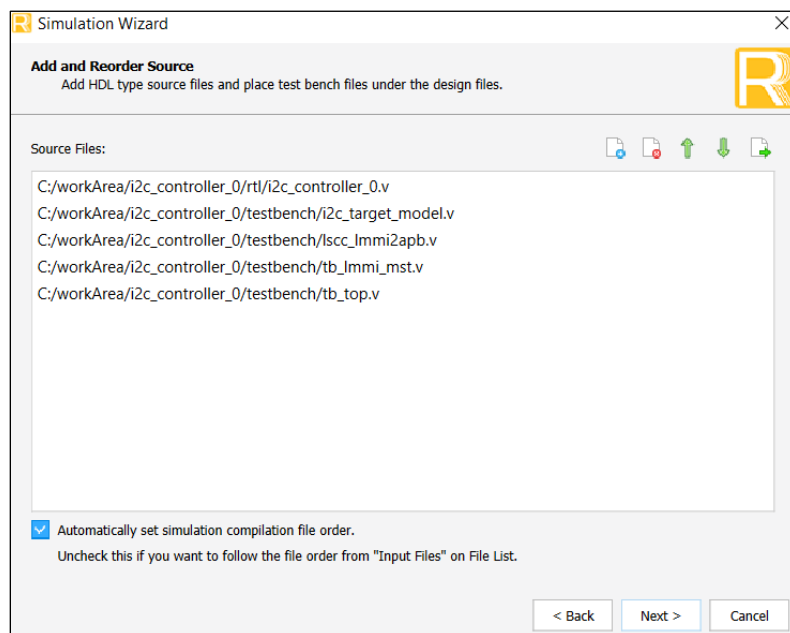
Simulator
☒ ModelSim
☐ Active-HDL

Process Stage
☒ RTL
☐ Post-Synthesis
☐ Post-Route Gate-Level
☐ Post-Route Gate-Level+Timing

< Back Next > Cancel

Figure 7.4. Simulation Wizard

2. Click **Next** to open the **Add and Reorder Source** window as shown in [Figure 7.5](#).



Simulation Wizard

Add and Reorder Source
Add HDL type source files and place test bench files under the design files.

Source Files:

- C:/workArea/i2c_controller_0/rtl/i2c_controller_0.v
- C:/workArea/i2c_controller_0/testbench/i2c_target_model.v
- C:/workArea/i2c_controller_0/testbench/lscclmmi2apb.v
- C:/workArea/i2c_controller_0/testbench/tb_lmmi_mst.v
- C:/workArea/i2c_controller_0/testbench/tb_top.v

☒ Automatically set simulation compilation file order.
Uncheck this if you want to follow the file order from "Input Files" on File List.

< Back Next > Cancel

Figure 7.5. Add and Reorder Source

3. Click **Next**. The **Summary** window is shown. Click **Finish** to run the simulation.

Note: It is necessary to follow the procedure above until it is fully automated in the Lattice Radiant software suite.

The results of the simulation in our example are provided in [Figure 7.6](#).

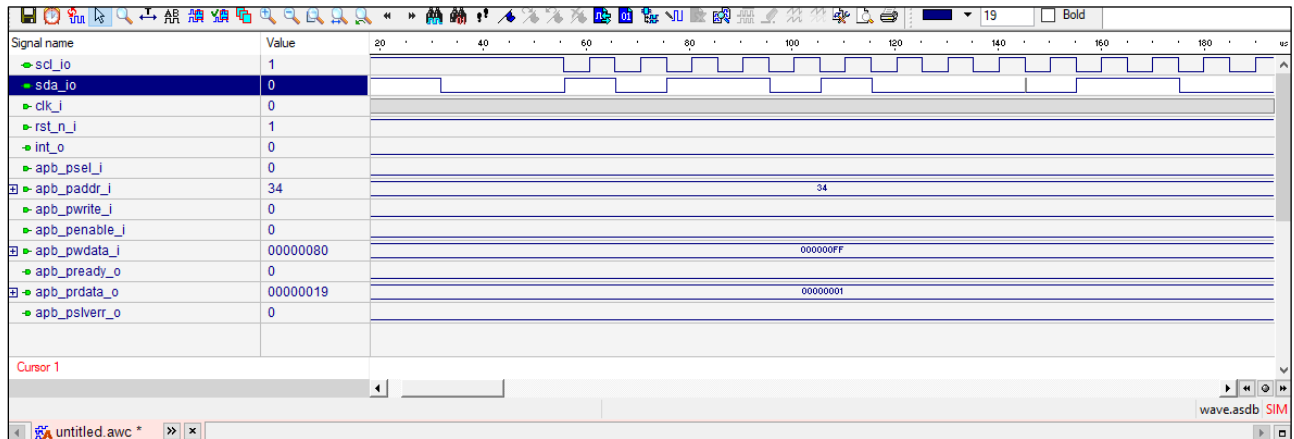


Figure 7.6. Simulation Waveform

Appendix A. Resource Utilization

[Table A.1](#) shows configuration and resource utilization for the LIFCL-40-9BG400I device using Synplify Pro of the Lattice Radiant Software 2.1.

Table A.1. LIFCL-40-9BG400I Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Slice Registers	LUTs	EBRs
Default	179.662	508	609	0
APB Mode Enable is Unchecked, Others = Default	200.000	489	652	0
Implementation of FIFO = EBR, Others = Default	190.876	494	584	2
FIFO Depth = 256, RX FIFO Almost Full Flag = 254, Others = Default	137.893	623	1388	0
FIFO Depth = 256, RX FIFO Almost Full Flag = 254, Implementation of FIFO = EBR, Others = Default	190.949	604	697	2

Note:

1. Fmax is generated when the FPGA design contains only I2C Controller IP and the target Frequency is 50 MHz. These values may be reduced when user logic is added to the FPGA design.

[Table A.2](#) shows configuration and resource utilization for the LAV-AT-E70-1LFG1156I device using Synplify Pro of the Lattice Radiant Software 2022.1.

Table A.2. LAV-AT-E70-1LFG1156I Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Slice Registers	LUTs	EBRs
Default	115.168	508	587	0
APB Mode Enable is Unchecked, Others = Default	249.813	491	576	0
Implementation of FIFO = EBR, Others = Default	116.239	492	559	2
FIFO Depth = 256, RX FIFO Almost Full Flag = 254, Others = Default	99.780	620	1164	0
FIFO Depth = 256, RX FIFO Almost Full Flag = 254, Implementation of FIFO = EBR, Others = Default	97.809	604	651	2

Note:

1. Fmax is generated when the FPGA design contains only I2C Controller IP and the target Frequency is 50 MHz. These values may be reduced when user logic is added to the FPGA design.

[Table A.3](#) shows configuration and resource utilization for the LN2-CT-20-1CBG484C device using Synplify Pro of the Lattice Radiant Software 2024.2.

Table A.3. LN2-CT-20-1CBG484C Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Slice Registers	LUTs	EBRs
Default	115.741	487	588	0
APB Mode Enable is Unchecked, Others = Default	250.000	483	587	0
Implementation of FIFO = EBR, Others = Default	124.626	471	494	2
FIFO Depth = 256, RX FIFO Almost Full Flag = 254, Others = Default	117.082	599	1165	0
FIFO Depth = 256, RX FIFO Almost Full Flag = 254, Implementation of FIFO = EBR, Others = Default	130.056	583	654	2

Note:

1. Fmax is generated when the FPGA design contains only I2C Controller IP and the target Frequency is 50 MHz. These values may be reduced when user logic is added to the FPGA design.

Table A.4 shows configuration and resource utilization for the LFMXO4-110HC-5BBG484I device using Synplify Pro of the Lattice Radiant Software 2025.2.

Table A.4. LFMXO4-110HC-5BBG484I Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Slice Registers	LUTs	EBRs
Default	84.10	712	721	0
APB Mode Enable is Unchecked, Others = Default	96.99	695	724	0
Implementation of FIFO = EBR, Others = Default	79.28	440	556	2
FIFO Depth = 256, RX FIFO Almost Full Flag = 254, Others = Default	53.49	4624	3343	0
FIFO Depth = 256, RX FIFO Almost Full Flag = 254, Implementation of FIFO = EBR, Others = Default	73.20	512	638	2

Note:

1. Fmax is generated when the FPGA design contains only I2C Controller IP and the target Frequency is 50 MHz. These values may be reduced when user logic is added to the FPGA design.

References

For more information, refer to:

- [Lattice Memory Mapped Interface and Lattice Interrupt Interface \(FPGA-UG-02039\)](#)
- [AMBA 3 APB Protocol v1.0 Specification](#)
- [I2C Bus Specification and User Manual](#)
- [Lattice Radiant Software 2023.1 User Guide](#)
- [Lattice Radiant Timing Constraints Methodology \(FPGA-AN-02059\)](#)
- [Reveal User Guide for Radiant Software](#)
- [I2C Controller IP Release Notes \(FPGA-RN-02027\)](#)
- [I2C Target IP Core](#) web page
- [Lattice Propel Design Environment](#) web page
- [Lattice Radiant Software](#) web page
- [Lattice Solutions IP Cores](#) web page
- [Lattice Solutions Reference Designs](#) web page
- [Avant-E](#) web page
- [Avant-G](#) web page
- [Avant-X](#) web page
- [Certus-N2](#) web page
- [Certus-NX](#) web page
- [CertusPro-NX](#) web page
- [CrossLink-NX](#) web page
- [MachXO4](#) web page
- [MachXO5-NX](#) web page
- [CertusPro-NX Evaluation Board](#) web page
- [MachXO5-NX Development Board](#) web page
- [Lattice Insights](#) web page 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/en/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 2.1, IP v2.4.0, December 2025

Section	Change Summary
All	<ul style="list-style-type: none"> Updated the IP version information on the cover page. Added a note on the IP version in the <i>Quick Facts</i> and <i>Revision History</i> sections. Made editorial fixes.
Introduction	<ul style="list-style-type: none"> In Table 1.1. Summary of the I2C Controller IP: <ul style="list-style-type: none"> Added <i>MachXO4</i> to <i>Supported Devices</i>. Updated <i>Lattice Implementation</i>. In Table 1.2. I2C Controller IP Support Readiness: <ul style="list-style-type: none"> Added a table note. Removed the <i>IP</i> column. Added the Attribute Names section.
Functional Description	Updated the Multi-controller Arbitration section.
Signal Description	Updated the descriptions for <i>scl_oe_o</i> and <i>sda_oe_o</i> in Table 4.1. I2C Controller IP Signal Description .
Designing with the IP	Added a note on screenshots in this section.
Resource Utilization	Added the MachXO4 device resource utilization.
References	Added <i>I2C Target IP Core</i> , <i>MachXO4</i> , <i>MachXO5-NX</i> , <i>Lattice Propel Design Environment</i> , and <i>Lattice Solutions Reference Designs</i> web pages.

Revision 2.0, IP v2.3.0, July 2025

Section	Change Summary
All	Updated the IP version information on the cover page.
Introduction	In Table 1.1. Summary of the I2C Controller IP : <ul style="list-style-type: none"> Updated <i>Supported FPGA Family</i> to <i>Supported Devices</i> and rearranged devices order. Removed <i>Targeted Devices</i>. Updated <i>Lattice Implementation</i>.

Revision 1.9, IP v2.2.0, December 2024

Section	Change Summary
All	<ul style="list-style-type: none"> Added the IP version information on the cover page. Updated <i>I²C</i> to <i>I2C</i>. Removed the <i>Debugging</i> section. Made editorial fixes.
Acronyms in This Document	<ul style="list-style-type: none"> Added <i>Design Under Test (DUT)</i>, <i>Embedded Block RAM (EBR)</i>, <i>First In, First Out (FIFO)</i>, <i>General Purpose Input/Output (GPIO)</i>, <i>Input/Output (I/O)</i>, <i>Intellectual Property (IP)</i>, <i>Lattice Synthesis Engine (LSE)</i>, <i>Look-Up Table (LUT)</i>, <i>Programmable Interrupt Controller (PIC)</i>, <i>Phase-Locked Loop (PLL)</i>, <i>Random Access Memory (RAM)</i>, <i>Receiver (RX)</i>, <i>Reduced Instruction Set Computer Five (RISC-V)</i>, <i>Serial Clock (SCL)</i>, <i>Static Random Access Memory (SRAM)</i>, <i>System on Chip (SoC)</i>, <i>Serial Data (SDA)</i>, <i>Transmitter (TX)</i>, and <i>Universal Asynchronous Receiver/Transmitter (UART)</i>.
Introduction	<ul style="list-style-type: none"> Updated Table 1.1. Summary of the I2C Controller IP: <ul style="list-style-type: none"> Added the <i>Certus-N2</i> device family to <i>Supported FPGA Family</i>. Removed <i>IP Version</i> and added <i>IP Changes</i>. Added the <i>LFD2NX-9</i>, <i>LFD2NX-28</i>, <i>LFCPNX-50</i>, <i>LFMXO5-55T</i>, <i>LFMXO5-100T</i>, <i>LAV-AT-G70</i>, <i>LAV-AT-X70</i>, and <i>LN2-CT-20</i> devices to <i>Targeted Devices</i>.

Section	Change Summary
	<ul style="list-style-type: none"> Updated <i>Resources</i> and <i>Lattice Implementation</i>. Updated the Licensing Information and Minimum Device Requirements sections. Replaced the <i>IP Validation Summary</i> section with the IP Support Summary section. Removed the <i>1.7.3. Host</i> and <i>1.7.4. Attribute</i> sections.
Functional Description	<ul style="list-style-type: none"> Removed <i>ack_mode</i> from step 1 in the Write to a Target Device, Read from a Target Device, and Read from a Target Device with a Specific Register or Command (Repeated Start) sections. Updated the <i>scl_io</i> description in the Clock Generation and Synchronization section. Added (<i>interrupts are disabled</i>) to the following: <ul style="list-style-type: none"> Steps 6 and 10 in the Write to a Target Device section Steps 5 and 9 in the Read from a Target Device section Steps 6, 10, 16, and 20 in the Read from a Target Device with a Specific Register or Command (Repeated Start) section. Replaced <i>Target Register</i> with <i>Specific Register</i> in the Read from a Target Device with a Specific Register or Command (Repeated Start) section header.
IP Parameter Description	In Table 3.1. Attributes Table, updated the <i>Selectable Values</i> field for the <i>TX FIFO Almost Empty Flag</i> and <i>RX FIFO Almost Full Flag</i> attributes.
Example Design	Added this section.
Designing with the IP	<ul style="list-style-type: none"> Updated the paragraph in the Designing with the IP section. Updated Figure 7.1. Module/IP Block Wizard, Figure 7.2. Configure User Interface of I2C Controller IP, and Figure 7.3. Check Generating Result.
Resource Utilization	Added resource utilizations for the Lattice Radiant software version 2024.2.
References	Added the <i>Lattice Solutions IP Cores</i> web page, <i>Avant-G</i> web page, <i>Avant-X</i> web page, <i>Certus-N2</i> web page, <i>CertusPro-NX Evaluation Board</i> web page, <i>MachXO5-NX Development Board</i> web page, and <i>I2C Controller IP Release Notes (FPGA-RN-02027)</i> .

Revision 1.8, January 2024

Section	Change Summary
All	<ul style="list-style-type: none"> Renamed the document from <i>I²C Controller IP Core – Lattice Radiant Software</i> to <i>I²C Controller IP</i>. Updated the instances of <i>I²C Controller IP Core</i> to <i>I²C Controller IP</i>. Made editorial fixes. Minor adjustments to ensure that the document is consistent with Lattice Semiconductor's inclusive language policy. Updated the device name from <i>LAV-AT-500E</i> to <i>LAV-AT-E70</i>.
Disclaimers	Updated boilerplate.
Inclusive Language	Added boilerplate.
Introduction	<ul style="list-style-type: none"> Moved the first two paragraph of the 1. Introduction section to 1.1. Overview of the IP section. Updated the heading number for 1.2. Quick Facts and 1.3. Features sections. Updated Table 1.1 caption from <i>Quick Facts</i> to <i>Summary of the I²C Controller IP</i>. Moved the previous 3.1. <i>Licensing the IP</i> and 3.4. <i>IP Evaluation</i> sections to 1.4. Licensing Information section. Moved the previous 3.5. <i>Hardware Validation</i> section to 1.5. IP Validation Summary section. Added the 1.6. Minimum Device Requirements section. Updated the previous header from 1.3. <i>Conventions</i> to 1.7. <i>Naming Conventions</i>.
Functional Description	<ul style="list-style-type: none"> Updated the previous header from 2.1. <i>Overview</i> to 2.1. <i>IP Architecture Overview</i>. Added the 2.2. Clocking and Reset section. Moved the previous 2.5. <i>Operations Details</i> section to 2.3. Operations Details section. Moved the previous 2.6. <i>Selectable Memory-Mapped Interface</i> to 2.4.1. Selectable

Section	Change Summary
	<p>Memory-Mapped Interface section.</p> <ul style="list-style-type: none"> Moved the previous 2.7. <i>Programming Flow</i> section to 2.5. <i>Programming Flow</i> section.
IP Parameter Description	Moved the previous 2.3. <i>Attributes Summary</i> section to 3. <i>IP Parameter Description</i> section.
Signal Description	Moved the previous 2.2. <i>Signal Description</i> section to 4. <i>Signal Description</i> section.
Register Description	Moved the previous 2.4. <i>Register Description</i> section to 5. <i>Register Description</i> section.
Designing with the IP	<ul style="list-style-type: none"> Moved the previous section 3.2. <i>Generation and Synthesis</i> to 6.1. <i>Generating and Instantiating the IP</i> section. Added 6.2. <i>Design Implementation</i> and 6.3. <i>Timing Constraints</i> sections. Moved the previous section 3.3. <i>Running Functional Simulation</i> to 6.4. <i>Running Functional Simulation</i> section.
Debugging	Added this section.
References	<ul style="list-style-type: none"> Updated the names of existing references. Added references to <i>Lattice Memory Mapped Interface and Lattice Interrupt Interface (FPGA-UG-02039)</i>, <i>AMBA 3 APB Protocol v1.0 Specification</i>, <i>I²C Bus Specification and User Manual</i>, <i>Lattice Radiant Software 2023.1 User Guide</i>, <i>Lattice Radiant Timing Constraints Methodology (FPGA-AN-02059)</i>, <i>Reveal User Guide for Radiant Software</i>, <i>Avant-E web page</i>, and <i>Lattice Insights web page</i>.

Revision 1.7, August 2023

Section	Change Summary
All	<ul style="list-style-type: none"> Changed the word 'Master' to 'Controller'. Changed the word 'Slave' to 'Target'.
Introduction	<ul style="list-style-type: none"> Added MachX05-NX, Certus-NX-RT and CertusPro-NX-RT to Supported FPGA Family. Added LIFCL-33, LFMX05-25, UT24C40, and UT24CP100 to Targeted Devices. Updated Lattice Implementation to 'IP Core v2.x.x - Lattice Radiant Software 2022.1 or later and Lattice Propel Builder 2022.1 or later'.
Functional Description	<ul style="list-style-type: none"> Updated Figure 2.1. I2C Controller IP Core Functional Diagram. Updated Table 2.1. I2C Controller IP Core Signal Description. Updated Table 2.2. Attributes Table. Updated Table 2.3. Attributes Descriptions. Updated the Register Description section. Updated the Operations Details section.
Core Generation, Simulation, and Validation	<ul style="list-style-type: none"> Updated Figure 3.1. Module/IP Block Wizard. Updated Figure 3.2. Configure User Interface of I2C Controller IP Core. Updated Figure 3.3. Check Generating Result. Updated Figure 3.4. Simulation Wizard. Updated Figure 3.5. Add and Reorder Source.
References	<p>Added reference links to the following:</p> <ul style="list-style-type: none"> Lattice Radiant Software website CrossLink-NX FPGA website Certus-NX FPGA website CertusPro-NX FPGA website MachX05-NX FPGA website Lattice Avant-E FPGA website
Technical Support Assistance	Added link to the Lattice Answer Database.

Revision 1.6, November 2022

Section	Change Summary
Introduction	<ul style="list-style-type: none"> In Table 1.1. Summary of the I2C Controller IP: <ul style="list-style-type: none"> added Lattice Avant to Supported FPGA Family;

Section	Change Summary
	<ul style="list-style-type: none"> added LAV-AT-500E to Targeted Devices.
Functional Description	<ul style="list-style-type: none"> Updated Figure 2.1. I2C Controller IP Functional Diagram. In the Control Register (CONTROL_REG) section: <ul style="list-style-type: none"> updated Table 5.7. Control Register; added description for the following new register fields: rx_fifo_reset, tx_fifo_reset and repeated_start. Newly added the Read from a Target Device with a Target Register or Command (Repeated Start) section.
Core Generation, Simulation, and Validation	<ul style="list-style-type: none"> Updated Figure 6.1. Module/IP Block Wizard, Figure 6.2. Configure User Interface of I2C Controller IP, and Figure 6.3. Check Generating Result. Changed the section name to IP Evaluation for section 3.4. Changed the default value of the hardware evaluation capability in the Strategy dialog box to disabled in the IP Evaluation section. Newly added the Hardware Validation section.
Appendix A. Resource Utilization	Newly added Table A.2. Resource Utilization for LAV-AT-E70-1LFG1156I Device.

Revision 1.5, August 2022

Section	Change Summary
Functional Description	<ul style="list-style-type: none"> In Table 5.1. Registers Address Map: <ul style="list-style-type: none"> changed INT_ENABLE2_REG register Access Type to RW; changed INT_ENABLE1_REG register Access Type to RW. In Table 5.13. Interrupt Enable First Register: <ul style="list-style-type: none"> changed Name, Access, Width and Reset for Field [6]. In Table 5.15. Interrupt Set First Register: <ul style="list-style-type: none"> changed Name, Access, Width and Reset for Field [6]. In the Interrupt Set Registers (INT_SET1_REG, INT_SET2_REG) section: <ul style="list-style-type: none"> removed the description for <i>stop_det_set</i>.

Revision 1.4, October 2021

Section	Change Summary
Revision	Corrected previous revision to 1.3.

Revision 1.3, June 2021

Section	Change Summary
Introduction	<ul style="list-style-type: none"> Removed last paragraph. Updated Table 1.1. Quick Facts. <ul style="list-style-type: none"> Revised Supported FPGA Families Revised Targeted Devices Revised Lattice Implementation Revised reference to the Lattice Radiant Software User Guide.
Core Generation, Simulation, and Validation	<ul style="list-style-type: none"> Removed reference to the Lattice Radiant Software User Guide. Replaced CrossLink-NX devices with Lattice FPGA devices built on the Lattice Nexus platform In the Hardware Evaluation section.
References	Updated this section.

Revision 1.2, June 2020

Section	Change Summary
Introduction	Updated Table 1.1. Quick Facts.

Section	Change Summary
	<ul style="list-style-type: none"> Added Certus-NX as supported FPGA family. Added LFD2NX-40 as targeted device. Changed Synplify version to Pro for Lattice.
Functional Description	<ul style="list-style-type: none"> Updated Table 2.4 and Table 2.11. Changed Read Transaction to <i>The wait state of APB read transaction has been reduced to one</i> in Selectable Memory-Mapped Interface section.
Appendix A. Resource Utilization	Updated Table A.1.
References	Updated this section.

Revision 1.1, February 2020

Section	Change Summary
Introduction	Updated Table 1.1. Quick Facts. <ul style="list-style-type: none"> Changed LIFCL to CrossLink-NX as supported FPGA family. Added LIFCL-17 as targeted device.
Core Generation, Simulation, and Validation	Updated user interface item to <i>IP, Processors, Controllers, and Peripherals</i> category.

Revision 1.0, December 2019

Section	Change Summary
All	Changed document status from Preliminary to final.
Introduction	Updated Table 1.1. Quick Facts. <ul style="list-style-type: none"> Added Resource information. Updated Lattice Implementation information.
References	Removed reference to the FPGA device web page.
Appendix A. Resource Utilization	Added resource utilization information.
All	Minor editorial changes

Revision 0.80, October 2019

Section	Change Summary
All	Preliminary release.



www.latticesemi.com