Generic Soft I^{2}C Master Controller

Reference Design

FPGA-RD-02201-1.0

December 2020

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

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
I^2C	Inter-Integrated Circuit
SCL	Serial Clock Line
SDA	Serial Data Line

1. Introduction

I²C or Inter-Integrated Circuit is a popular serial interface protocol that is widely used in many electronic systems. The I²C interface is a two-wire interface capable of half-duplex serial communication at moderate to high speeds of up to a few megabits per second. There are thousands of I²C peripherals on the market today, ranging from data converters to video processors. The I²C bus is a good choice for designs that need to communicate with low-speed peripherals due to its simplicity and low cost.

This reference design implements an I²C Master Module on any Lattice FPGA using Lattice Diamond® 3.11 and Lattice Radiant® 2.1. It follows the I²C specification to provide device addressing, read/write operation, and an acknowledgement mechanism. It adds an instant I²C compatible interface to any component in the system. The programmable nature of FPGA devices provides you with the flexibility of configuring the I²C master device to any legal slave address. This avoids the potential slave address collision on an I²C bus with multiple slave devices.

2. Features

- Supports a wide array of Lattice FPGAs such as MachXO2™, MachXO3™, LatticeECP3™, ECP5™, CrossLink™, CrossLink™-NX, and iCE40 UltraPlus™
- Supports 7-bit and 10-bit slave address
- Supports operation at 100 kHz* (Standard Mode) and 400 kHz* (Fast Mode)
- Supports repeated start operations
- Interrupt generation logic
- Verilog RTL, test bench
- Byte-wide clock stretching

***Note:** Verified in both simulation and hardware.

3. Functional Description

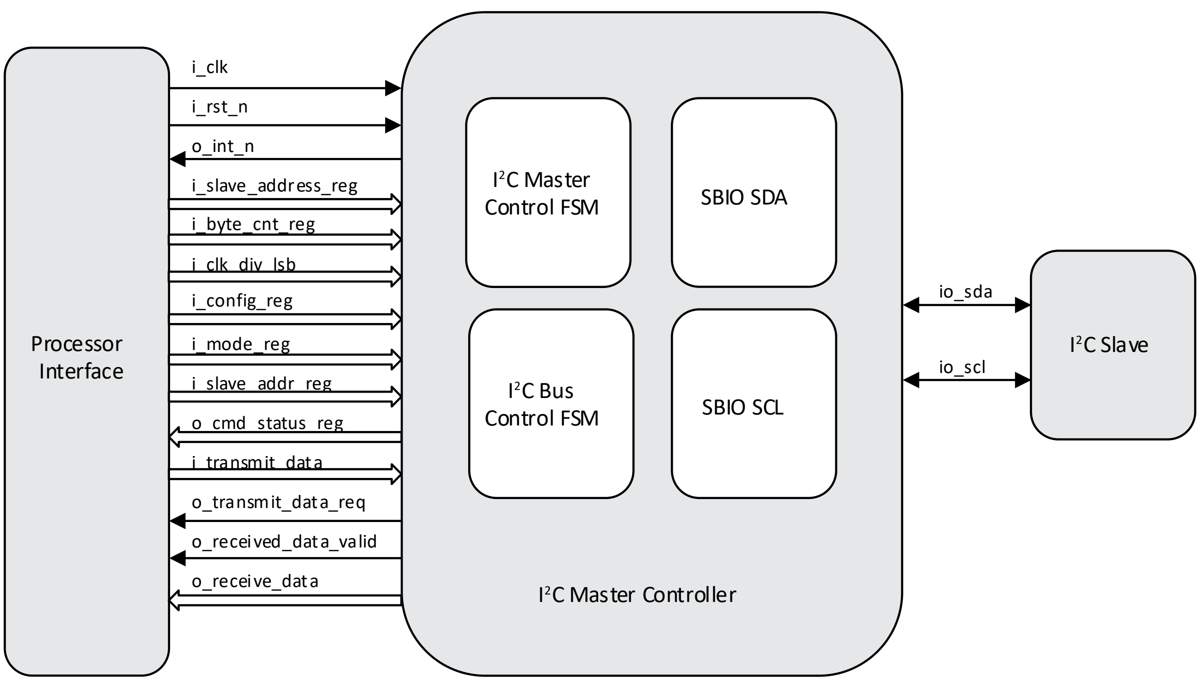

Figure 3.1. Block Diagram

4. Pin Descriptions

Table 4.1. Pin Descriptions

Signal	Width	Type	Description
i_clk	1	Input	System clock operating at 24 MHz
i_rst_n	1	Input	Asynchronous active-low system reset
o_int_n	1	Output	Active-low processor interrupt
i_slave_addr_reg	10	Input	10-bit I^2C slave address. If 7-bit addressing mode is enabled, then the controller takes only i_slave_addr_reg[6:0].
i_byte_cnt_reg	8	Input	Sets the number of data bytes to be read or written for the I^2C transaction
i_clk_div_lsb	8	Input	Sets the lower byte of the clock divider that is used to generate SCL from CLK. The upper three bits are located in the mode register.
i_config_reg	6	Input	This is used to configure the I^2C Master Controller (see Table 7.1).
i_mode_reg	8	Input	Sets the various modes of operation like speed, read/write (see Table 7.1).
o_cmd_status_reg	8	Output	Indicates the status of the operation, I^2C bus (see Table 7.1).
o_start_ack	1	Output	Acknowledge to the start bit provided by the user through i_config_reg
i_transmit_data	8	Input	Data to be transmitted over the SDA line to the I^2C slave
o_transmit_data_requested	1	Output	Indicates that transmit data is required
o_received_data_valid	1	Output	A *1* corresponds to valid data availability on the o_receive_data line.
o_receive_data	8	Output	Received data bus
io_scl	1	Inout	I^2C clock line
io_sda	1	Inout	I^2C data line

5. Design Modules

The design includes the modules shown in Figure 5.1.

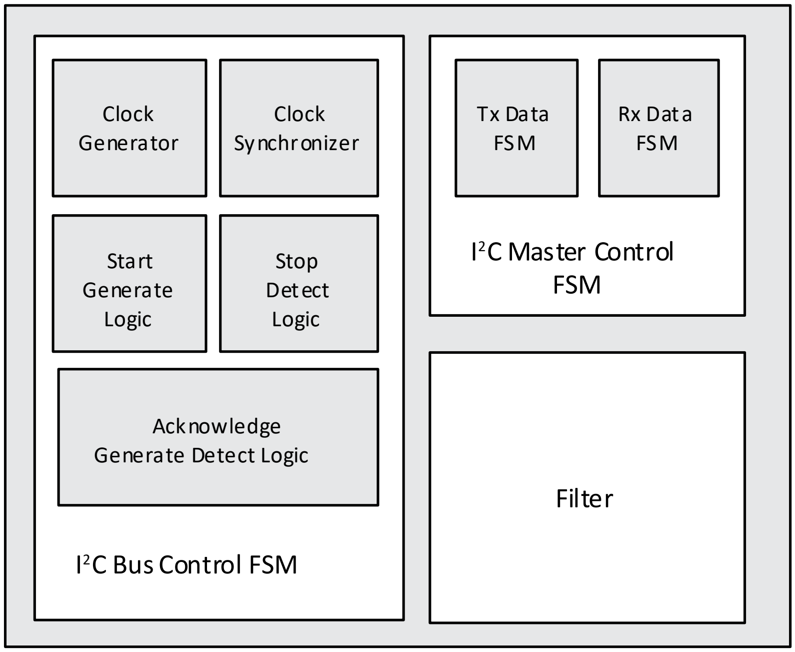

Figure 5.1. Functional Block Diagram

5.1. I^{2}C Bus Control FSM

The I^{2}C Bus Control FSM is comprised of the Clock Generator/Synchronizer, Start/Stop generate/detect logic, and Acknowledge generate/detect logic.

The Clock Generation and Synchronization logic generates the I^{2}C clock signal SCL based on the system clock and clock divide factors configured by the processor. Due to the nature of the I^{2}C bus, the actual SCL clock that is seen by all devices on the bus may not be running at the same frequency that the master generates. This module starts counting its SCL low period when the current master drives SCL low. Once a device's clock goes low, the master holds the SCL line low until the clock high state is reached. When all devices count off their LOW period, the clock line is released and goes HIGH. There is no difference between the device clocks and the state of the SCL line, and all the devices start counting their HIGH periods. The first device to complete its HIGH period pulls again the SCL line LOW. In this way, a synchronized SCL clock is generated with its LOW period determined by the device with the longest clock LOW period, and its HIGH period determined by the one with the shortest clock HIGH period.

The start/stop logic generates and detects start and stop events on the I^{2}C bus. The detection of start and stop events is necessary to determine whether or not the I^{2}C bus is in use by another master on the bus when the primary master gets a START signal from the processor. When the I^{2}C bus is idle, both SCL and SDA are pulled high by passive pull-ups. A start condition is signaled by transitioning SDA from high to low while SCL is still high. Likewise, a stop condition is signaled by transitioning SDA from low to high while SCL is high.

5.2. I^{2}C Master Control FSM

For controlling data transfer, the I^{2}C master makes use of a control FSM, along with counters for controlling the bits and bytes. The byte counter is an 8-bit counter that keeps track of the number of bytes that are written or read during the I^{2}C transaction. This counter increments after each byte is written to or read from the external I^{2}C slave device. The count is then compared with the byte count register. If the value is a match, the I^{2}C Master Controller considers the transaction complete, issues a stop signal on the I^{2}C bus, asserts the RXTX_DONE flag, and waits for the next transaction to be initiated from the processor. This counter is fully controlled by the main control FSM.

6. Internal Register Map

The I²C Master Controller configuration can be performed on run-time. Table 6.1 lists the available registers.

Table 6.1. Register List

Port/Bit	9	8	7	6	5	4	3	2	1	0
i_slave_addr_reg	SADR[9]	SADR[8]	SADR[7]	SADR[6]	SADR[5]	SADR[4]	SADR[3]	SADR[2]	SADR[1]	SADR[0]
i_byte_cnt_reg			BCNT[7]	BCNT[6]	BCNT[5]	BCNT[4]	BCNT[3]	BCNT[2]	BCNT[1]	BCNT[0]
i_clk_div_lsb			DIV[7]	DIV[6]	DIV[5]	DIV[4]	DIV[3]	DIV[2]	DIV[1]	DIV[0]
i_config_reg					RESET	ABORT	TX_IE	RX_IE	INT_CLR	START
i_mode_reg			BPS[1]	BPS[0]	ADR_MOD		RW_MODE	DIV[10]	DIV[9]	DIV[8]
o_cmd_status_reg			I2C_BUSY	TX_DONE	RX_DONE	TX_ERR	RX_ERR	ABORT_ACK		

Undefined

7. Register Bit Descriptions

Table 7.1. Register Bit Descriptions

Register Bit	Description
i_slave_addr_reg[9:0]	10-bit slave address. If 10-bit addressing mode is disabled (i_mode_reg[5] = 1'b1), then the controller takes only i_slave_addr_reg[6:0].
I_byte_cnt_reg[7:0]	Sets the number of data bytes to be written or read for the I2C transaction. For example, set the register to 8 to transfer eight data bytes.
i_clk_div_lsb[7:0] - DIV[7:0]	Sets the lower byte of the clock divider that is used to generate SCL from CLK. The upper three bits are located in i_mode_reg[2:0]. Note that DIV[0] is not used since only even DIV values are supported.
i_config_reg[5] – RESET –	Writing a 1 resets this I2C Master Controller.
i_config_reg[4] – ABORT –	Writing a 1 stops the current I2C transaction in progress. This bit is cleared by the ABORT_ACK status bit in the Command Status Register.
i_config_reg[3] – TX_IE –	Set this bit high to enable interrupt generation on o_int_n output after completing a transmission (I^2C Master Write) and a STOP condition in the I^2C bus has been issued.
i_config_reg[2] – RX_IE	Set this bit high to enable interrupt generation on o_int_n output when receiving has completed (I^2C Master Read) and a STOP condition in the I^2C bus has been issued.
i_config_reg[1] – INT_CLR	Writing a 1 clears all bits in the *o_cmd_status_reg* output except the I2C_BUSY bit.
i_config_reg[0] – START	Write a 1 to start an I^2C transaction. This bit is auto-cleared after the master successfully arbitrates and acquires the I^2C bus.
i_mode_reg[7:6] – BPS[1:0]	Selects the I^2C speed mode. (2'b00 = standard, 2'b01 = fast, others are reserved)
i_mode_reg[5] – ADR_MOD	Selects the I^2C address mode. (1'b0 = 7-bit Addressing, 1'b1 = 10-bit Addressing)
i_mode_reg[3] – RW_MODE	Sets the read or write operation on the I^2C bus. (0 = write, 1 = read)
i_mode_reg[2:0] – DIV[10:8]	The upper three bits of the clock divider factor.
o_cmd_status_reg[7] – I2C_BUSY	This read-only status bit indicates that the bridge is busy performing a data transaction and a STOP is not issued. This bit reflects the state of the I^2C bus and cannot be cleared by the user.
o_cmd_status_reg[6] – TX_DONE	This read-only status bit indicates that the I^2C write operation issued is completed, but the STOP condition may still be in progress.
o_cmd_status_reg[5] – RX_DONE	This read-only status bit indicates that the I^2C read operation issued is completed, but the STOP condition may still be in progress.
o_cmd_status_reg[4] – TX_ERR	This read-only status bit indicates an error during the I^2C write operation.
o_cmd_status_reg[3] – RX_ERR	This read-only status bit indicates an error during the I^2C read operation.
o_cmd_status_reg[2] – ABORT_ACK	This read-only status bit indicates that the ABORT command is completed. You should clear the proper FIFO and status bits afterwards.

Note: All status bits, except I2C_BUSY, are cleared by writing a *1* to the INTR_CLR bit in the configuration 3 register.

8. Timing Diagram

8.1. I²C Master Write Timing Diagram

The following describes how a Processor Interface should control this reference design when an I²C Write Transaction is desired.

1. The I²C Master Controller waits for *i_config_reg[0]* to be asserted to 1 to begin a transaction.
2. The Processor Interface requests an I²C transaction by applying the necessary signal values to the inputs below:

Table 8.1. Configuration and Mode Bit Requirements for I²C Write Transaction

Register Bit	Value	Function
i_config_reg[0]	1'b1	Starts an I²C transaction
i_config_reg[3]	1'b1	Generates an interrupt to o_int_n on transmit completion
i_mode_reg[3]	1'b0	Defines that the I²C transaction is a write operation

Note: For simplicity, some of the bits for i_config_reg and i_mode_reg are not shown in Figure 8.1. Refer to Table 7.1 for the complete list of options.

3. The Processor Interface waits for *o_start_ack* to go HIGH before it sets *i_config_reg[5:0]* input to 6'b000000.
4. At this point, the I²C bus transaction had already begun and a positive *o_transmit_data_request* strobe is generated each time a transmit data is required in the *i_transmit_data* input on the next clock cycle.
5. When the total number of bytes defined in the *i_byte_cnt_reg* input is reached, an *o_int_n* interrupt is generated which also tells the Processor Interface that a stop condition is generated. At this point, the Processor Interface should assert *i_config_reg[1]* to 1 to clear the bits of *o_cmd_status_reg* and puts the I²C Master Controller in idle state. When a new transaction is intended, i_config_reg[1] should be deasserted to 0.

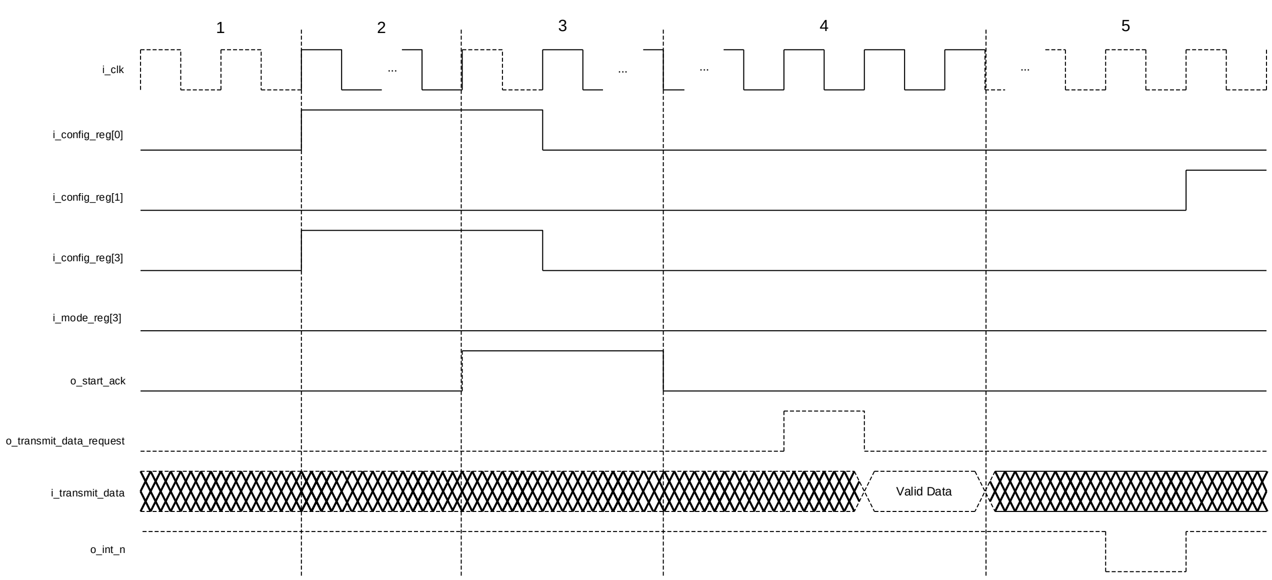

Figure 8.1. I²C Write Timing

8.2. I²C Master Read Timing Diagram

The following describes how a Processor Interface should control this reference design when an I²C Read Transaction is desired.

1. The I²C Master Controller waits for *i_config_reg[0]* to be asserted to 1 to begin a transaction.
2. The Processor Interface requests an I²C transaction by applying the necessary signal values to the inputs below:

Table 8.2. Configuration and Mode Bit Requirements for I²C Read Transaction

Register Bit	Value	Function
i_config_reg[0]	1'b1	Starts an I²C transaction
i_config_reg[2]	1'b1	Generates an interrupt to o_int_n on receive completion
i_mode_reg[3]	1'b1	Defines that the I²C transaction is a read operation

Note: For simplicity, some of the bits for i_config_reg and i_mode_reg are not shown in Figure 8.2. Refer to Table 7.1 for the complete list of options.

3. The Processor Interface waits for *o_start_ack* to go HIGH before it sets *i_config_reg[5:0]* inputs to 6'b000000.
4. At this point, the I²C bus transaction had already begun and a positive *o_received_data_valid* strobe is generated each time a received data is valid in the *o_receive_data* output.
5. When the total number of bytes defined in the *i_byte_cnt_reg* input has been reached, an *o_int_n* interrupt is generated which also tells the Processor Interface that a stop condition has been generated. At this point, the Processor Interface should then assert *i_config_reg[1]* to 1 to clear the bits of *o_cmd_status_reg* and puts the I²C Master Controller in idle state. When a new transaction is intended, i_config_reg[1] should be deasserted to 0.

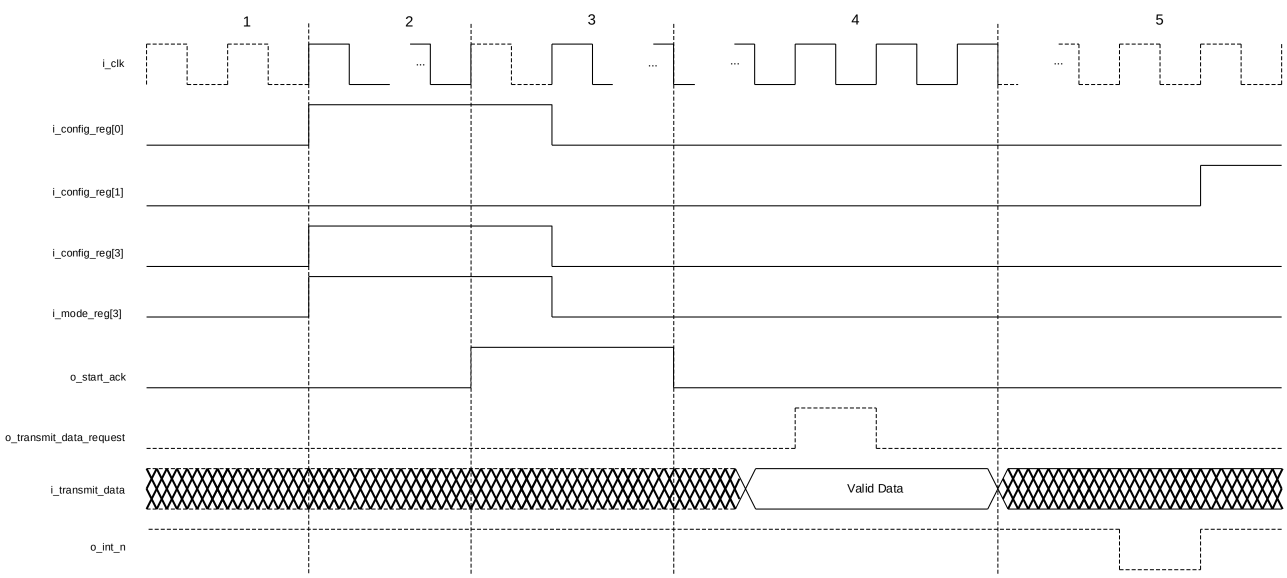

Figure 8.2. I²C Read Timing

9. Operation Sequence

9.1. 7-Bit Addressing Mode

9.1.1. Single/Multi-Byte Write Operation

Figure 9.1 shows a Master Write operation in 7-bit addressing mode. The master generates the START bit and sends the 7-bit slave address, followed by the eighth bit which is a data direction read/write bit (R/W). *0* is sent for this WRITE operation. The master sends the data followed by an acknowledgment (A) from the slave. The slave generates an acknowledgment for every byte of data from the master. The processor can either STOP the transaction by sending a STOP bit, or the slave can respond with a NACK (A') so that the master stops the data write by generating a STOP condition to terminate the data transfer.

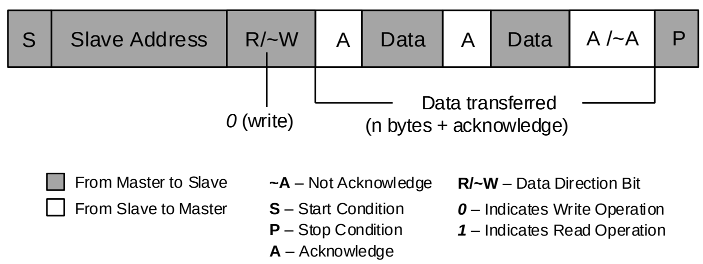

Figure 9.1. Data Format for Master Write Operation Using 7-Bit Address Mode

9.1.2. Single/Multi-Byte Read Operation

Figure 9.2 shows a Master Read operation in 7-bit addressing mode. The master generates a START bit, transmits a 7-bit slave address, followed by an eighth bit which is a data direction bit (R/W). A *1* is sent for this READ operation. The slave acknowledges this by a positive acknowledgment (A). The slave transmits a byte of data, which the master should acknowledge (A) for further data transactions to continue. The master generates a Not Acknowledge (A) before generating a STOP condition to terminate the data transfer.

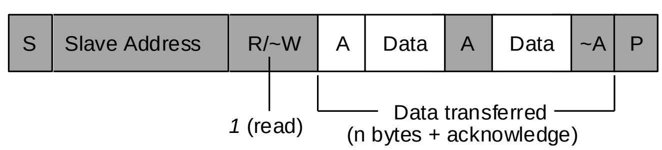

Figure 9.2. Data Format for Master Read Operation Using a 7-Bit Address Mode

9.1.3. Write with Repeated Start

Figure 9.3 shows a Master Write with Repeated Start. The master generates a START bit and sends a 7-bit slave address plus the eighth R/W bit as *0* for the write transaction. The slave acknowledges this request. The master then sends one or more data byte followed by an acknowledgment from the slave. Instead of generating a STOP condition, the master generates another START (that is Repeated START) and repeats the process again. You can define how many times the process is repeated before generating a STOP condition

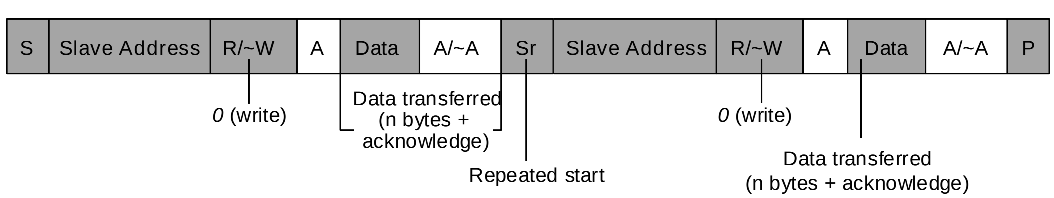

Figure 9.3. Data Format for Master Write Operation with Repeat Start Using a 7-Bit Address Mode

9.2. 10-Bit Addressing Mode

9.2.1. Single/Multi-Byte Write Operation

Figure 9.4 shows a Master Write operation in 10-bit addressing mode. The master generates the START condition and sends the first seven bits of the first byte. The first seven bits are *11110XX*, of which the last two bits (XX) are the two Most-Significant Bits (MSBs) of the 10-bit address, followed by a *0* R/W eighth bit. Slaves supporting 10-bit mode and matching the two MSB address bits respond with an acknowledgment (A1). The master sends the second byte of the slave address and which is acknowledged (A2) by the matching slave. Hereafter, the write data transfer is similar to conventional 7-bit addressing mode.

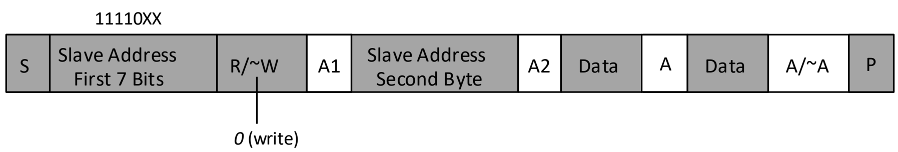

Figure 9.4. Data Format for Master Write Operation Using a 10-Bit Address Mode

9.2.2. Single/Multi-Byte Read Operation

Figure 9.5 shows the Master Read operation in 10-bit addressing mode. The master generates the START condition and sends the first seven bits of the first byte. The first seven bits are *11110XX* of which the last two bits (XX) are the two Most-Significant Bits (MSBs) of the 10-bit address, followed by a *0* R/W eighth bit. Slaves supporting 10-bit mode and matching the two MSB address bits respond with an acknowledgment (A1). The master sends the second byte of the slave address which is acknowledged (A2) by the matching slave. The master generates a *Repeated START* and sends the same first byte of the address followed by a *1* on the R/W bit. The slave generates a positive acknowledgement (A3). Hereafter, the read data transaction is similar to conventional 7-bit addressing mode.

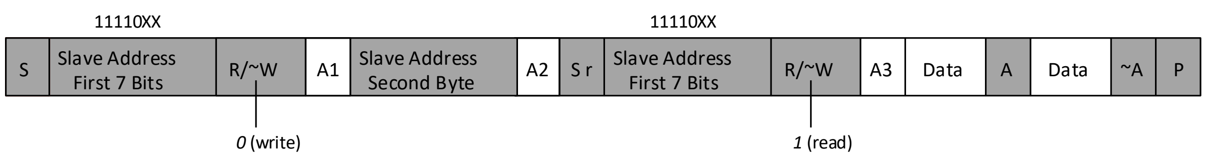

Figure 9.5. Data Format for Master Read Operation Using a 10-Bit Address Mode

9.2.3. Single/Multi-Byte Write Operation with Repeat Start

Figure 9.6 shows a Write with Repeated Start using a 10-bit addressing mode. The master generates the START condition and sends the first seven bits of the first byte. The first seven bits are *11110XX*, of which the last two bits (XX) are the two Most-Significant Bits (MSBs) of the 10-bit address, followed by a 0 R/W eighth bit. Slaves supporting 10-bit mode and matching the two MSB address bits respond with an acknowledgment (A). The master sends the second byte of the slave address and which is acknowledged (A) by the matching slave. Hereafter, the write data transfer is similar to conventional 10-bit addressing mode but instead of generating a STOP condition, the master generates another START (that is Repeated START) and repeats the process again. You can define how many times the process is repeated before generating a STOP condition.

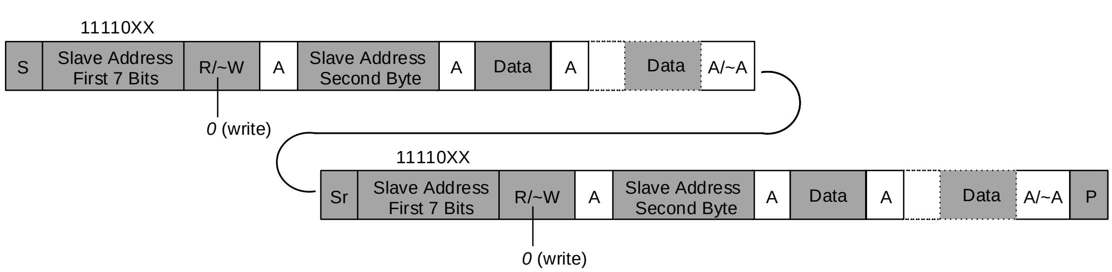

Figure 9.6. Data Format for Master Write Operation with Repeat Start Using a 10-Bit Address Mode

9.3. Clock Stretching

Clock stretching pauses a transaction when the slave holds the SCL line LOW. The transaction cannot continue until the line is released HIGH again. On the byte level, a 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. The slave can then hold the SCL line LOW after receipt and acknowledgment of a byte to force the master into a wait state until the slave is ready for the next byte transfer in a handshake procedure.

You can test the clock stretching capability of this reference design by uncommenting *stretch_test* in the *tb_defines.v* source file. For more info, refer to the Customization section.

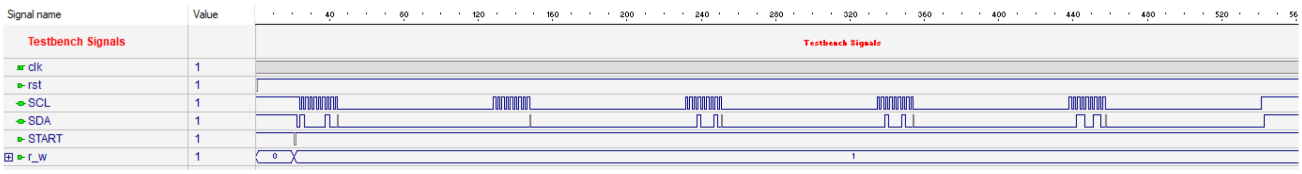

Figure 9.7. Simulation Waveform Showing a Four-Byte I²C Write Transaction with Clock Stretching

10. Customization

To customize the testbench files of this reference design, a file named *tb_defines.v* contains all the compiler directives that you can modify. This includes device selection, slave addresses settings, clock source, clock speed, and others. Table 10.1 shows the complete list of compiler directives. Figure 10.1 shows an example of customization implemented in the *tb_defines.v* file.

Table 10.1. Compiler Directives Options

Category	Compiler Directives	Remarks
Device Selection	ECP3™	Uncomment only one to enable the selected device.
	ECP5™	
	LIFMD	
	LIFCL	
	MachXO2™	
	MachXO3™	
	iCE40 Ultraplus™	
Slave Addresses	SLAVE_ADDRESS1	Define 10-bit slave addresses to be accessed by the master. If 10-bit address mode is disabled, then the controller takes only SLAVE_ADDRESSX[6:0].
	SLAVE_ADDRESS2	
Clock Speed Selection	CLK_12MHZ	Uncomment only one to enable the selected clock speed.*
	CLK_24MHZ	
	CLK_32MHZ	
Clock Stretching Test	stretch_test	Uncomment to enable clock stretching test in the slave testbench files.
Clock Stretching Value	stretch_value	Define the duration of the stretch in decimal value.
I²C Mode Selection	STD	Uncomment only one to enable Standard or Fast Mode.
	FSTMD	

***Note:** If the desired clock speed is not in the selection, other clock speeds can still be used. However, you should manually modify the i_clk_div_lsb and i_mode_reg[2:0] values to allow proper generation of io_scl clock line.

```
// **********************************************
// Device Selection
// (Uncomment the selected device.)
// **********************************************
//`define ECP3
//`define ECP5
//`define LIFMD
//`define LIFCL
//`define XO2
`define XO3
//`define Ultraplus


// **********************************************
// Slave Addresses
// (Define 10-bit slave addresses to be accessed by the master)
// **********************************************
`define SLAVE_ADDRESS1      10'b111_100_0001 // 0x3C1 or 0x41 depending on whether 10-bit addre
`define SLAVE_ADDRESS2      10'b111_100_0011 // 0x3C3 or 0x43 depending on whether 10-bit addre

// **********************************************
// Clock Speed Selection
// (Uncomment the selected clock speed.)
// **********************************************
//`define CLK_12MHZ
`define CLK_24MHZ
//`define CLK_32MHZ


// **********************************************
// Clock Stretching Test
// (Uncomment to enable clock stretching test.)
// **********************************************
//`define stretch_test

// **********************************************
// Clock Stretching Value
// (Uncomment to enable clock stretching test.)
// **********************************************
`define stretch_value   2000 // Maximum is 4095


// **********************************************
// I2C Mode Selection
// (Uncomment only one to enable Standard or Fast Mode)
// **********************************************
//`define STD   //Standard Mode
`define FSTMD   //Fast Mode
```

Figure 10.1. Compiler Directive Customization Example

11. HDL Simulation and Verification

This Generic I^{2}C Master module (*i2c_master_controller_top.v*) is simulated using a top-level testbench file *tb.v* that acts as the processor interface mentioned in Figure 3.1. It also allows access to two slave addresses 0x41 and 0x3C3 (7-bit and 10-bit address modes respectively) from the two instantiations of the testbench I^{2}C Slave module (*i2c_ebr_slave_top.v*).

Whenever the I^{2}C Master performs a write transaction, the I^{2}C slave's simple RAM allows it to store the data sent by the I^{2}C Master. The same data will then be sent by the I^{2}C Slave whenever the I^{2}C Master requests for an I^{2}C read transaction. For simplicity, only selected signals are shown in the figures below. The following lists the testbench flow:

1. The I^{2}C Master sends a 4-byte I^{2}C write command.
    a. As shown in Figure 11.1, the first byte (in this case 0x00) is treated by the testbench as the starting address of the RAM. The succeeding bytes 0x11, 0x22, and 0x33 are the actual data sent by the I^{2}C Master.
    b. Figure 11.2 shows a zoomed-in view during momentary assertion of the *o_transmit_data_request* port. At the falling edge of this port, the Master Controller fetches the data from the *i_transmit_data* port (0x11) so it can be sent to the slave through the I2C bus.
2. The I^{2}C Master sends a 3-byte I^{2}C read command.
    a. As shown in Figure 11.3, the data bytes 0x11, 0x22, and 0x33 was read back from the I^{2}C Slave.
    b. Figure 11.4 shows a zoomed-in view during momentary assertion of the *o_received_data_valid* port. During the same period, the processor interface can fetch the data from the *o_receive_data* port. As shown in Figure 6.4, 0x11 has been successfully received by the I^{2}C Master from the I^{2}C Slave.
3. The I^{2}C Master sends a 2-byte I^{2}C write command twice with a Repeat Start in between.
4. For easier analysis, the top-level testbench file *tb.v* implements display tasks ($display) showing the simulation activity and in what timeline a certain task is performed. After the above I^{2}C transactions, three more similar transactions are made but uses 10-bit address mode.

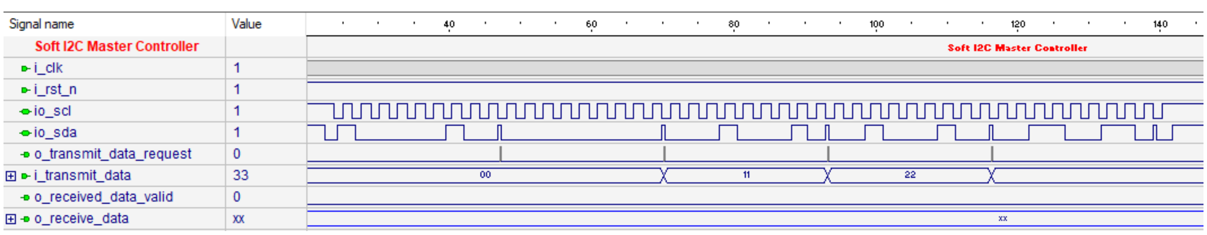

Figure 11.1. 4-Byte I^{2}C Write with Starting Address = 0x00

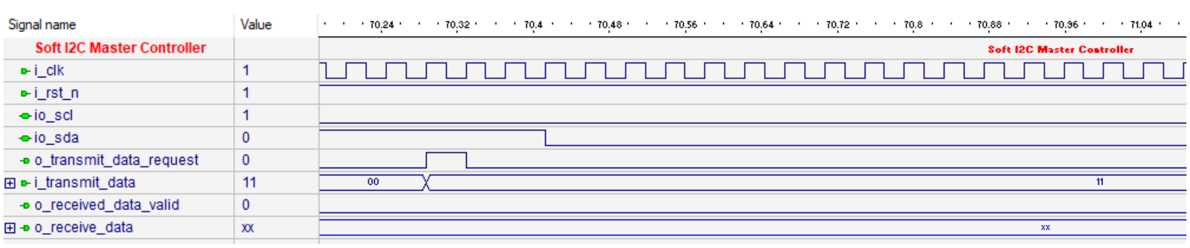

Figure 11.2. Zoomed-In View When 0x11 is Received by the Slave

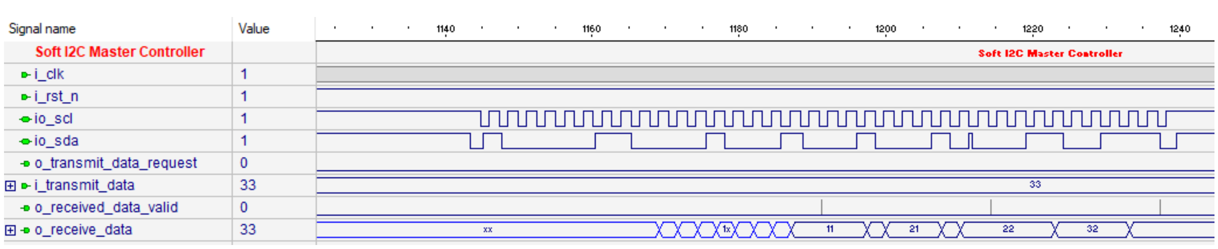

Figure 11.3. 3-Byte I^{2}C Read

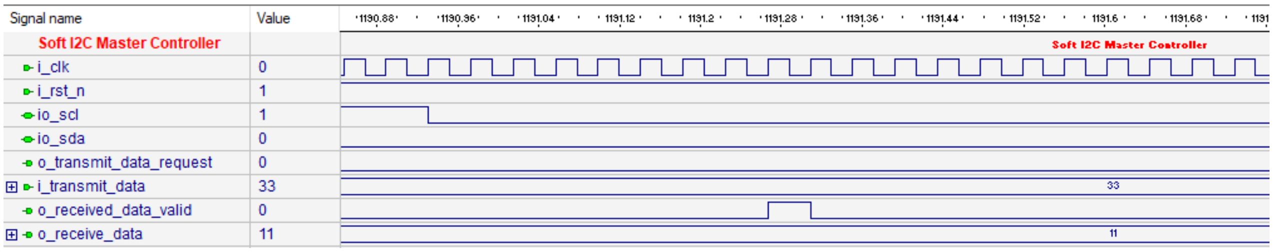

Figure 11.4. Zoomed-In View When 0x11 is Fetched by the I²C Slave Module

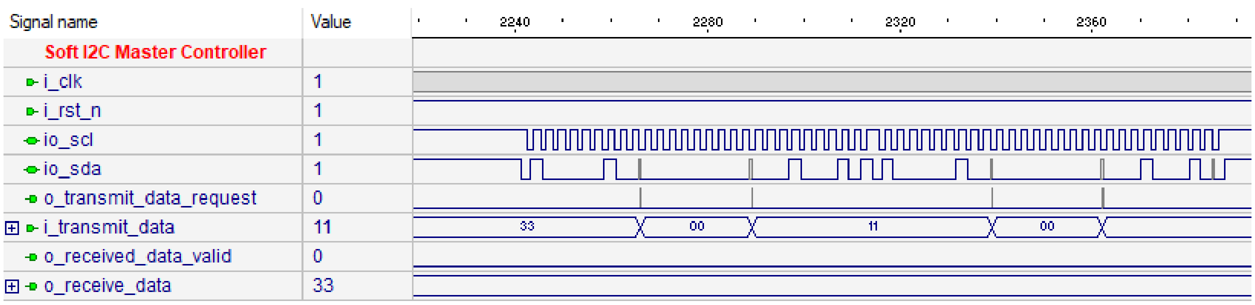

Figure 11.5. Repeated 2-Byte I²C Write Command

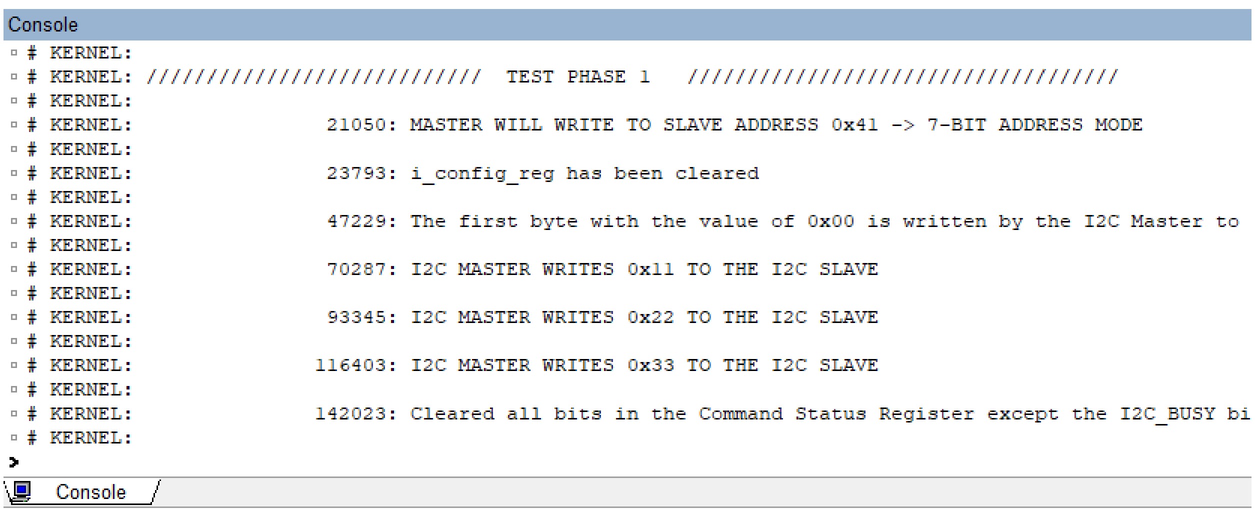

Figure 11.6. Aldec Active-HDL Console View

12. Packaged Design

The reference design folder (Generic_Soft_I2C_Master) contains five subfolders: Docs, Project, Simulation, Source, and Testbench. The details of each subfolder are as follows:

- Project – contains subfolders for each FPGA Family. Each of these subfolders contains either a Diamond or a Radiant project file (.LDF and .RDF).
- Simulation – contains subfolders for each FPGA Family. Each of these subfolders contains the simulation file (.DO) used to run RTL simulation on Aldec Active-HDL.
- Source – contains all the Generic I^2C Master RTL files.
- Testbench – contains all the testbench source files.

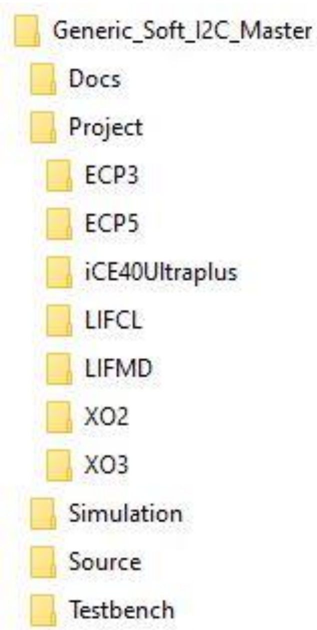

Figure 12.1. Packaged Design Directory Structure

12.1. Using the Simulation File (.DO)

To use the simulation file, perform the following steps:

1. Open the DO file on a text editor and replace the text **<ENTER simulation DIRECTORY PATH HERE>** from Line 1 with the directory path of the simulation file. An example is seen on Line 4 of the file.

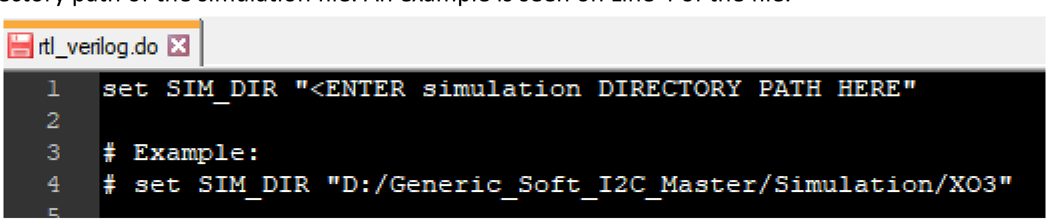

Figure 12.2. Changing the Simulation Directory

2. Run the file on Aldec Active-HDL by selecting *Execute macro…* under the **Tools** option.

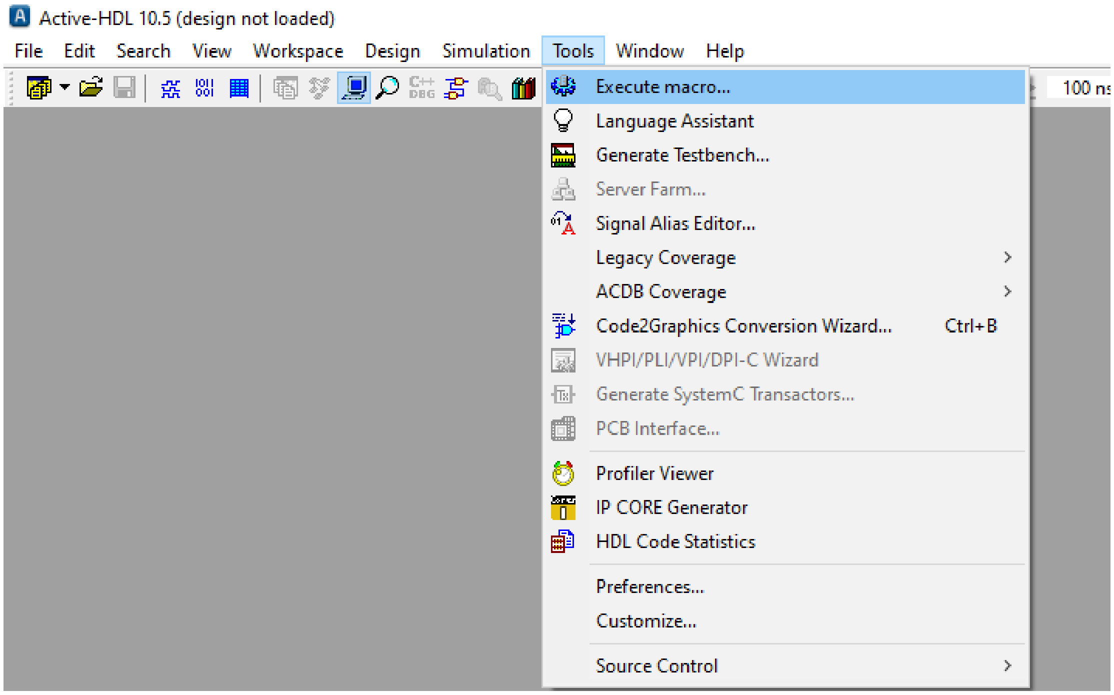

Figure 12.3. Running the Simulation File

13. Hardware Validation

This reference design was hardware validated using a MachXO3- 9400 Development Board (LCMXO3LF-9400C-ASC-B-EVN) and an iCE40 UltraPlus Breakout Board (iCE40UP5K-B-EVN). A companion demo was also created to allow the user to perform actual hardware validation on most Lattice FPGA. Refer to the Generic Soft I^{2}C Master and Slave Write-Read Demo (FPGA-UG-02122).

14. Implementation

This design is implemented in Verilog. When using this design in a different device or strategy settings, density, speed/grade, performance, and utilization may vary. Due to the limitations of the I/O pin count of iCE40 UltraPlus and CrossLink devices, the included two projects for these fail during Map. However, if most of the ports for this reference design are only used internally, Map succeeds like in the case of the companion demo, Generic Soft I^2C Master and Slave Write-Read Demo (FPGA-UG-02122).

Table 14.1. Resource Utilization

<table>
<tr><th>Device Family</th><th>Language</th><th>Utilization (LUTs)</th><th>f_{MAX} (MHz)</th><th>I/O</th></tr>
<tr><td>Lattice ECP3[1]</td><td>Verilog</td><td>290</td><td>>32</td><td rowspan="7">This Reference Design has a total of 69 ports. The hardware validated companion demo mentioned in this document is only using nine I/O since most of the ports are only used internally.</td></tr>
<tr><td>ECP5[2]</td><td>Verilog</td><td>289</td><td>>32</td></tr>
<tr><td>CrossLink™[3]</td><td>Verilog</td><td>~263[8]</td><td>>32</td></tr>
<tr><td>CrossLink™-NX[4]</td><td>Verilog</td><td>290</td><td>>32</td></tr>
<tr><td>iCE40 UltraPlus[5]</td><td>Verilog</td><td>282[9]</td><td>>32</td></tr>
<tr><td>MachXO2[6]</td><td>Verilog</td><td>292</td><td>>32</td></tr>
<tr><td>MachXO3[7]</td><td>Verilog</td><td>292</td><td>>32</td></tr>
</table>

Notes:

1. Performance and utilization characteristics are generated using LFE3-35EA-8FN484C with Lattice Diamond 3.11 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro®.
2. Performance and utilization characteristics are generated using LFE5U-85F-8BG381C with Lattice Diamond 3.11 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
3. Performance and utilization characteristics are generated usingiCE40UP5K-SG48I with Lattice Radiant 2.1 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
4. Performance and utilization characteristics are generated using LIFCL-40-7BG400I with Lattice Lattice Radiant 2.1 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
5. Performance and utilization characteristics are generated using LFE5U-85F-8BG381C with Lattice Diamond 3.11 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
6. Performance and utilization characteristics are generated using LCMXO2-7000HE-6TG144C with Lattice Diamond 3.11 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
7. Performance and utilization characteristics are generated using LCMXO3LF-9400C-6BG484C with Lattice Diamond 3.11 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
8. Approximation only. Selected CrossLink device does not meet the required 69 I/O for this reference design. However, if some of the ports are going to be utilized internally, this reference design can still be used.
9. Total LUT count came from the *Map Resource Usage* section of Lattice Radiant software's report browser after compiling the design using another top-level unit of the companion demo that instantiates the *i2c_master_controller_top* module.

References

For more information, refer to the following documents:

- LatticeECP3 EA Family Data Sheet (DS1021)
- ECP5 and ECP5-5G Family Data Sheet (FPGA-DS-02012)
- CrossLink Family Data Sheet (FPGA-DS-02007)
- MachXO2 Family Data Sheet (DS1035)
- MachXO3 Family Data Sheet (FPGA-DS-02032)
- iCE40 UltraPlus Family Data Sheet (FPGA-DS-02008)
- Generic Soft I^2C Master and Slave Write-Read Demo (FPGA-UG-02122)

Technical Support Assistance

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

Revision History

Revision 1.0, December 2020

Section	Change Summary
All	Initial release.