Generic Soft I^{2}C Slave/Peripheral

Reference Design

FPGA-RD-02193-1.3

January 2023

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
EBR	Embedded Block RAM
I²C	Inter-Integrated Circuit
IP	Intellectual Property
SCL	Serial Clock Line
SDA	Serial Data Line

1. Introduction

The I^{2}C, or Inter-Integrated Circuit, is a two-wire interface capable of half-duplex serial communication at moderate to high speeds of up to a few megabits per second. The I^{2}C incorporates an addressing system to identify the multiple I^{2}C slaves on the I^{2}C bus. The system utilizes two bidirectional lines, which are the SDA (Serial Data) and SCL (Serial Clock).

This reference design implements an I^{2}C slave module on any Lattice FPGA using Lattice Diamond® 3.12 and Lattice Radiant™ software 3.1. It follows the I^{2}C specification to provide device addressing, read/write operation and an acknowledgement mechanism. It adds an instant I^{2}C compatible interface to any component in the system. The programmable nature of FPGA devices provides you with the flexibility of configuring the I^{2}C slave device to any legal slave address. This avoids the potential slave address collision on an I2C bus with multiple slave devices.

2. Features

The following specifications are tested and verified using Lattice Diamond 3.12 and Radiant 3.1 on both simulation and hardware, unless stated otherwise.

- Supports a wide array of Lattice FPGAs such as MachXO2™, MachXO3™, LatticeECP3™, ECP5™, CrossLink™, CrossLink™-NX, and iCE40 UltraPlus™[1]
- Supports 7-bit and 10-bit slave addressing
- Software programmable slave address
- Supports clock stretching
- Supports repeated start condition
- Supports I^{2}C SCL range of up to 1 MHz with the following I^{2}C speed modes tested:
    - Standard Mode – 100 kHz
    - Fast Mode – 400 kHz
    - Fast Mode Plus – 1 MHz[2]

Notes:

1. Lattice Radiant 3.1 is used for iCE40 UltraPlus and Crosslink-NX. Lattice Diamond 3.12 is used for the rest of the mentioned devices.
2. Verified using simulation only.

3. Functional Description

3.1. Functional Block Diagram

Figure 3.1 shows an overview of the reference design, which has two interfaces: Peripheral (right side arrows), which connects an internal user module, and I^{2}C (left side arrows) which can be connected to an external I^{2}C Master. The user module may be internally (configured on the same FPGA device) or externally (another device) connected to the I^{2}C Slave module.

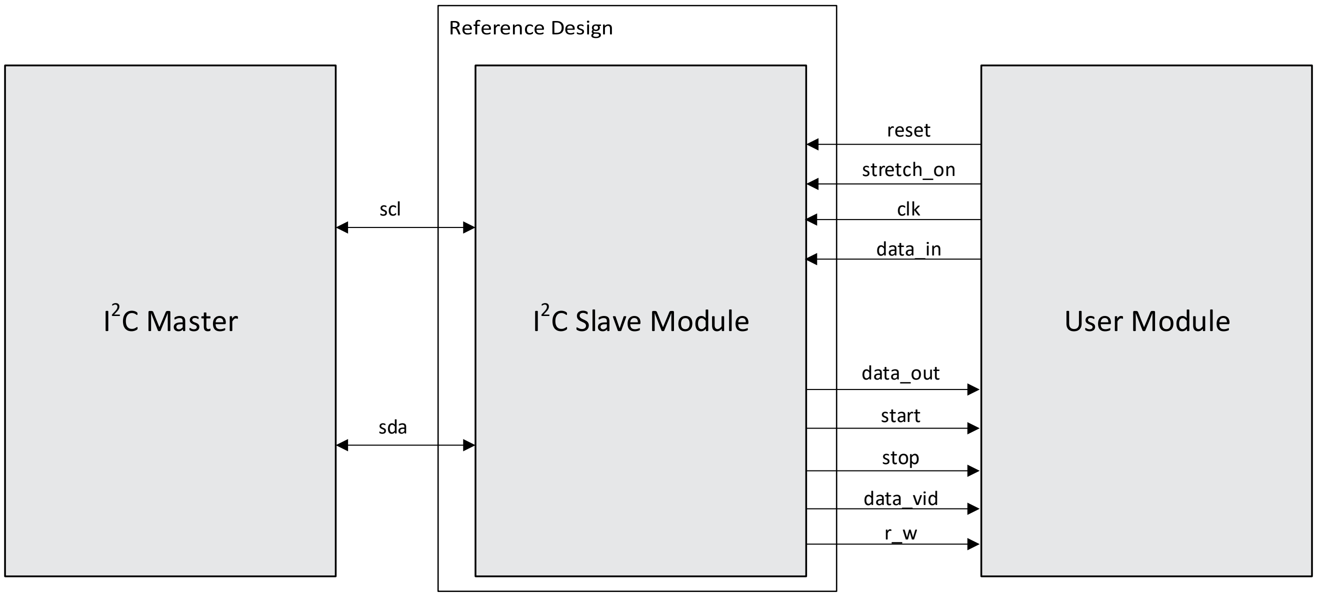


Figure 3.1. Functional Block Diagram

3.2. Signal Descriptions

Table 3.1 provides a detailed description of each signal indicated on Figure 3.1, relative to the I²C Slave module.

Table 3.1. Signal Descriptions

Signal Name	I/O Type	Width	Interface	Description
scl	BIDI	1	I²C	Serial Clock Line of the I²C core
sda	BIDI	1	I²C	Serial Data Line of the I²C core
reset	IN	1	Peripheral	Active-HIGH reset that signals the I²C Slave module to initialize.
stretch_on	IN	1	Peripheral	0 – Allows the user module to run without clock stretching. 1 – Allows byte-level clock stretching.
clk	IN	1	Peripheral	12 MHz system clock or higher
data_in	IN	8	Peripheral	8-bit input data port used to receive data from the user module that is transmitted to the I²C Master during I²C Read.
data_out	OUT	8	Peripheral	8-bit output data port used to transmit data to the user module that is transmitted by the I²C Master during I²C Write. This remains 0 all throughout and is only updated when data_vld is high.
start	OUT	1	Peripheral	Active-HIGH strobe Indicates that an I²C START or REPEAT START condition is generated by the I²C master.
stop	OUT	1	Peripheral	Active-HIGH strobe Indicates either of the following: • An I²C stop condition is generated by the master. • The I²C slave address sent by the I²C Master does not match the I²C Slave module's address (RD default: 0x41).
data_vld	OUT	1	Peripheral	Active-HIGH strobe. Indicates either of the following: • If r_w = 0, the I²C Master requested a write transaction. The User Module can fetch the data from the data_out port. • If r_w = 1, the I²C Master requested a read transaction. The User Module should feed the data to the data_in port. The data should be held at the same value until the next data_vld strobe.
r_w	OUT	1	Peripheral	0 – Indicates an I²C Write Operation (user module to receive). 1 – Indicates an I²C Read Operation (user module to transmit).

3.3. I²C Write Timing

The following describes the timing for the peripheral interface, as shown in Figure 3.2, when an I²C write command is received from the I²C Master:

1. The User Module waits for a positive strobe at the *start* port of the I²C Slave module (i2cslave_controller_top.v), which means that a START condition is generated by an external I²C Master.
2. If clock stretching is not desired, *stretch_on* can be held LOW by the User Module to allow continuous data transmission from the I²C Master without any interruption. At every positive strobe from *data_vld*, the data sent by the I²C Master to the I²C Slave should be fetched by the User Module from the *data_out* port.
3. If clock stretching is desired, the User Module can set the logic level of *stretch_on* to HIGH and holds it at that level. During this time, SCL is held LOW by the I²C Slave Module and stretches it indefinitely until *stretch_on* is released to LOW by the User Module.
    - The port *stretch_on* can be asserted HIGH at any time but the actual stretching is automatically applied at the negative edge of the ninth SCL clock cycle of every I²C frame.
4. Upon releasing *stretch_on* by the User Module, the I²C Slave Module would generate further *data_vld* strobe if more data are to be sent by the I²C Master. If a positive strobe at the *stop* port has been generated, it means that the I²C Master generated a stop condition signifying the end transaction and the I²C Slave returns to an idle state.

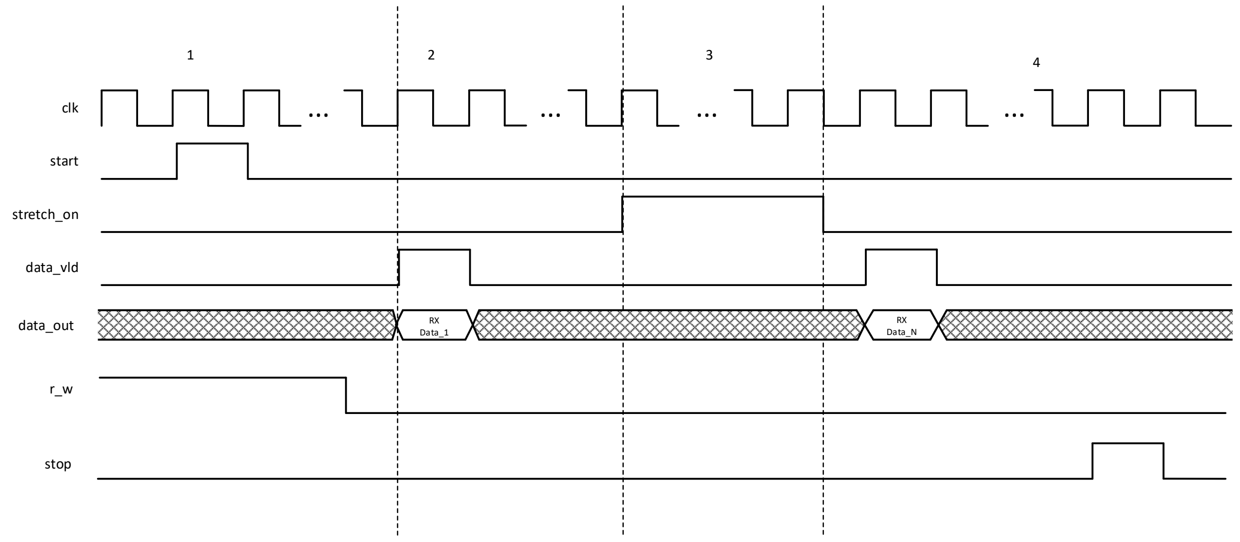


Figure 3.2. I²C Write Timing Diagram

3.4. I²C Read Timing

The following describes the timing for the peripheral interface, as illustrated by Figure 3.3, when an I²C read command is received from the I²C Master:

1. The User Module waits for a positive strobe at the *start* port of the I²C Slave module (i2cslave_controller_top.v) which means that an external I²C Master has generated a START condition.
2. If clock stretching is not desired, *stretch_on* can be held LOW by the user module to allow continuous data transmission to the I²C Master without any interruption. At every positive strobe from *data_vld*, the data to be sent by the I²C Slave to the I²C Master should be sent to the *data_in* port and be held there until the next *data_vld* strobe.
3. If clock stretching is desired, the User Module can set the logic level of *stretch_on* to HIGH and holds it at that level. During this time, SCL is held LOW by the I²C Slave Module after each data byte and stretches it indefinitely until *stretch_on* is released to LOW by the User Module.
   - The port *stretch_on* can be asserted HIGH at any time but the actual stretching is automatically applied at the negative edge of the ninth SCL clock cycle of every I²C frame.
4. Upon releasing *stretch_on* by the User Module, the I²C Slave Module would generate further *data_vld* strobe if more data are requested by the I²C Master. If a positive strobe at the stop port has been generated, it means that the I²C Master generated a stop condition signifying the end transaction and the I²C Slave returns to an idle state.

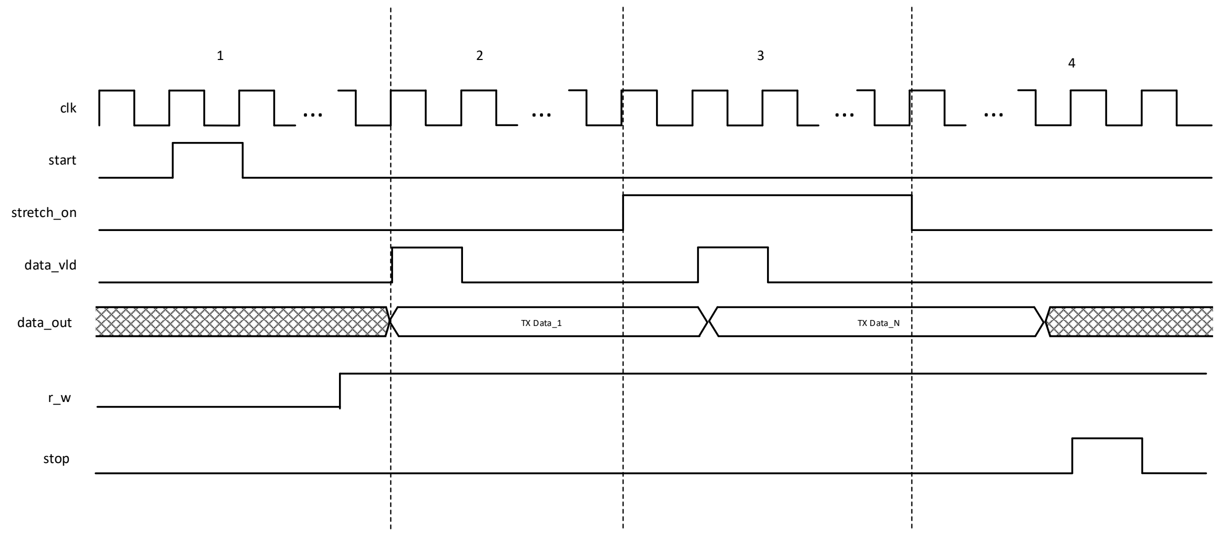


Figure 3.3. I²C Read Timing Diagram

4. Operation Sequence

4.1. 7-bit Addressing Mode

4.1.1. Single/Multi-Byte Write Operation

Figure 4.1 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 I^{2}C Master can either STOP the transaction by sending a STOP condition, 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.

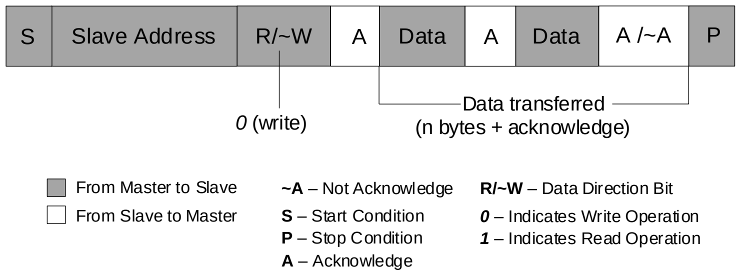


Figure 4.1. Data Format for Master Write Operation Using 7-bit Address Mode

4.1.2. Single/Multi-Byte Read Operation

Figure 4.2 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.

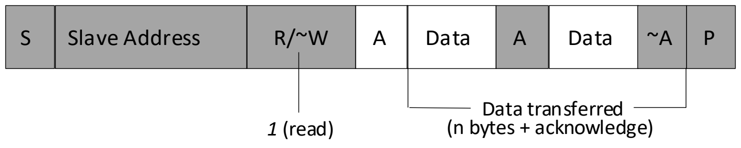
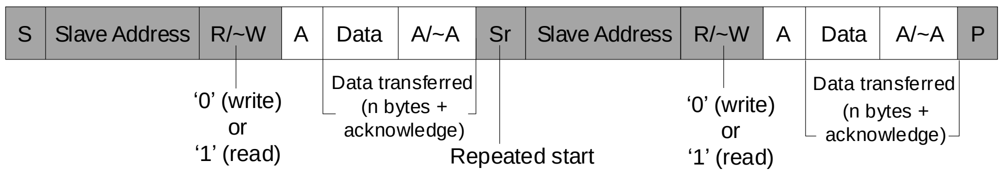


Figure 4.2. Data Format for Master Read Operation Using 7-Bbit Address Mode

4.1.3. Combined Format

Figure 4.3 shows a Write or Read Operation 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, and 1 for the read transaction. After the slave acknowledges this request, the Slave then sends or receives one or more data byte followed by an acknowledgement bit either from the I^{2}C Master or Slave depending on the type of transaction. Instead of generating a STOP condition, the master generates another START (*Repeated START*) and repeats the process again. The I^{2}C Master can define the number of times the process is repeated before generating a STOP condition.


S
Slave Address
R/~W
A
Data
A/~A
Sr
Slave Address
R/~W
A
Data
A/~A
P
'0' (write) or '1' (read)
Data transferred (n bytes + acknowledge)
Repeated start
'0' (write) or '1' (read)
Data transferred (n bytes + acknowledge)

*Transfer direction of data and acknowledge bits depends on the R/~W bits.

Figure 4.3. Data Format for Master Write Operation with Repeat Start Using a 7-bit Address Mode

4.2. 10-bit Addressing Mode

4.2.1. Single/Multi-Byte Write Operation

Figure 4.4 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.

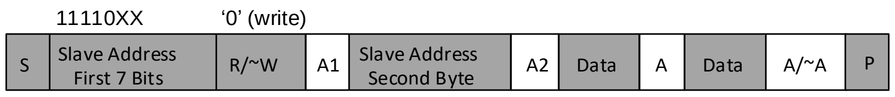


Figure 4.4. Data Format for Master Write Operation Using a 10-bit Address Mode

4.2.2. Single/Multi-Byte Read Operation

Figure 4.5 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.

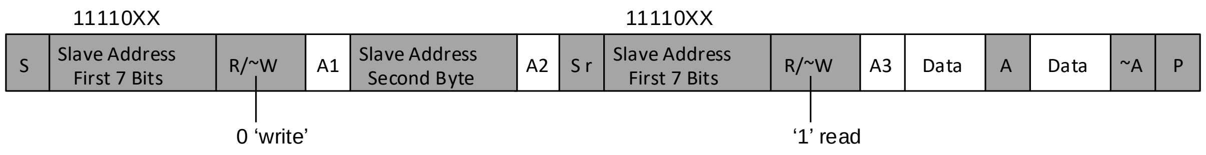

11110XX
11110XX
S
Slave Address First 7 Bits
R/~W
A1
Slave Address Second Byte
A2
S r
Slave Address First 7 Bits
R/~W
A3
Data
A
Data
~A
P
0 'write'
'1' read

Figure 4.5. Data Format for Master Read Operation Using a 10-bit Address Mode

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

Figure 4.6 shows a Write with Repeated Start using 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. The I^{2}C Master can define the number of times the process is repeated before generating a STOP condition.

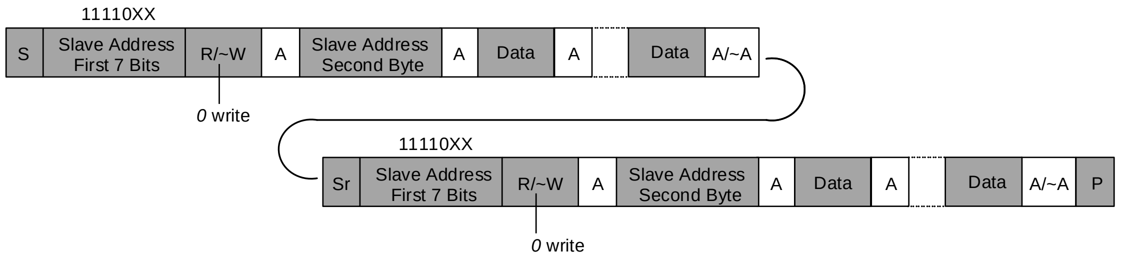

11110XX
S
Slave Address First 7 Bits
R/~W
A
Slave Address Second Byte
A
Data
A
Data
A/~A
0 write
11110XX
Sr
Slave Address First 7 Bits
R/~W
A
Slave Address Second Byte
A
Data
A
Data
A/~A
P
0 write

Figure 4.6. Data Format for Master Write Operation with Repeat Start Using a 10-bit Address Mode

4.3. Clock Stretching

Clock Stretching pauses a transaction by holding 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 reception 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.

To enable clock stretching, the user module can assert stretch_on to HIGH within each data byte and holds it at that level. During this time, SCL is held LOW by the I^{2}C Slave module after each data byte and stretches it indefinitely until stretch_on is released to LOW by the user module. Figure 4.7 shows a simulation example without clock stretching, while Figure 4.8 shows the same transaction with clock stretching.

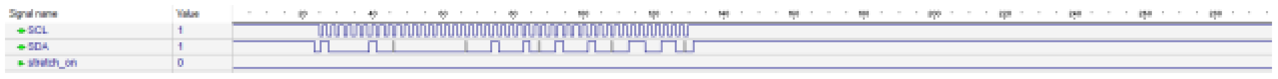


Figure 4.7. Clock Stretch Off

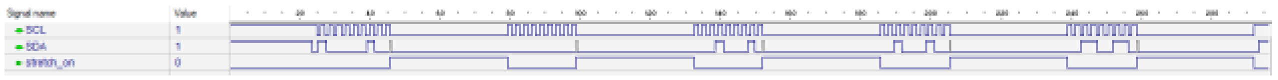
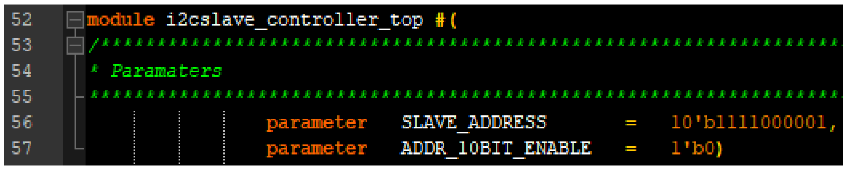


Figure 4.8. Clock Stretch On

5. Customization

The slave address can be customized in the user module by overriding the SLAVE_ADDRESS and ADDR_10BIT_ENABLE parameters during instantiation of the *i2cslave_controller_top* module. Figure 5.1 illustrates an example if it is done in the testbench module source file named *tb.v*. For simplicity, this reference design includes a file named *tb_defines.v* that contains all the compiler directives that the user can modify. This includes device selection, slave addresses settings, clock speed, and others. Table 5.1 shows the complete list of compiler directives. Figure 5.2 shows an example of customization implemented in the *tb_defines.v* file.

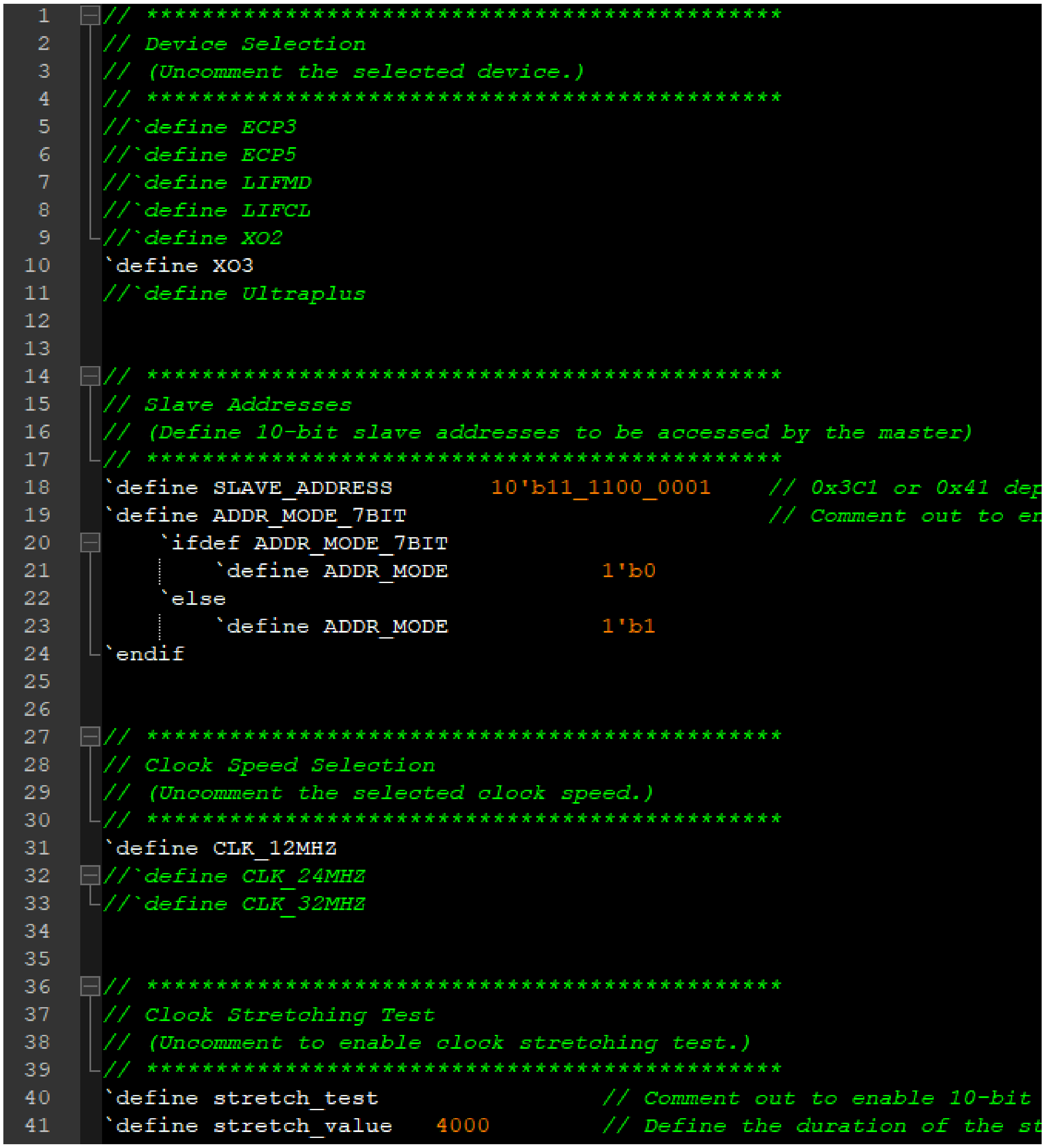


```
module i2cslave_controller_top #(
/***********************************************************************
 * Paramaters
 ***********************************************************************
            parameter   SLAVE_ADDRESS       =   10'b1111000001,
            parameter   ADDR_10BIT_ENABLE   =   1'b0)
```

Figure 5.1. Slave Address Customization

Table 5.1. Compiler Directives Option

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_ADDRESS	Define the 10-bit slave address. If 10-bit address mode is disabled, then the controller takes only SLAVE_ADDRESSX[6:0].
	ADDR_MODE_7BIT	Uncomment to enable 7-bit address mode and disable 10-bit address mode.
Clock Speed Selection	CLK_12MHZ	Uncomment only one to enable the selected clock speed. If the desired clock speed is not in the selection, the user can still manually define it in the testbench file *tb.v*.
	CLK_24MHZ	
	CLK_32MHZ	
Clock Stretching Test	stretch_test	Uncomment to test the clock stretching capability.
	stretch_value	Define the duration of the stretch in decimal value.



```
// ***********************************************
// 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_ADDRESS       10'b11_1100_0001    // 0x3C1 or 0x41 dep
`define ADDR_MODE_7BIT                          // Comment out to en
    `ifdef ADDR_MODE_7BIT
        `define ADDR_MODE           1'b0
    `else
        `define ADDR_MODE           1'b1
`endif


// ***********************************************
// 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                // Comment out to enable 10-bit
`define stretch_value   4000        // Define the duration of the st
```

Figure 5.2. Compiler Directive Customization Example

6. HDL Simulation and Verification

The I²C slave module (*i2c_slave_controller_top.v*) is simulated using a top-level testbench file *tb.v*. An I²C Master module (*i2c_master.v*) is also instantiated which has built-in functions to perform simple I²C Write and Read transactions. A simple RAM is also found in the top-level testbench to allow the I²C Slave to store the data sent by the I²C Master. The same data is sent by the I²C Slave whenever the I²C Master requests for an I²C Read transaction. The following lists the testbench flow:

1. The I²C Master sends a 4-byte I²C write command to the slave address 0x41..
    a. As shown in Figure 6.1, 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 I²C Master.
    b. Figure 6.2 shows a zoomed-in view during momentary assertion of the *data_vld* port. During the same period, *r_w* is already LOW and *data_out* port is updated. This means that 0x11 has been successfully received by the I²C Slave and can be fetched by the User Module from the *data_out port*.
2. The I²C Master sends a 3-byte I²C read command to the slave address 0x41.
    a. As shown in Figure 6.3, the I²C Master writes the register byte 0x00 before generating a Repeat Start condition. Afterward, the slave address 0x41 was resent and the data bytes 0x11, 0x22, and 0x33 was read back from the I²C Slave.
    b. Figure 6.4 shows a zoomed-in view during momentary assertion of the *data_vld* port. During the same period, *r_w* is already HIGH. This also means that the data from the *data_in* port should be fetched by the I²C Slave module beginning at each data_vld strobe and should be held at the same value until the next data_vld strobe.
3. 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 I²C transactions, three more similar transactions are made but uses 10-bit address mode and clock stretching.

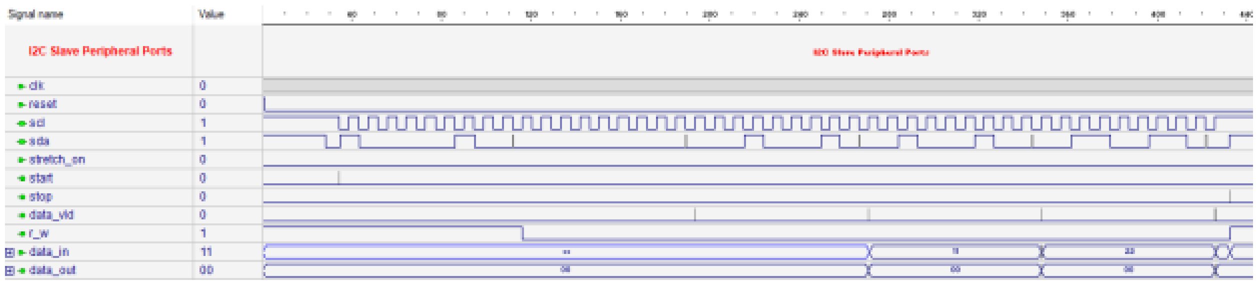


Figure 6.1. 4-Byte I²C Write with Starting Address = 0x00

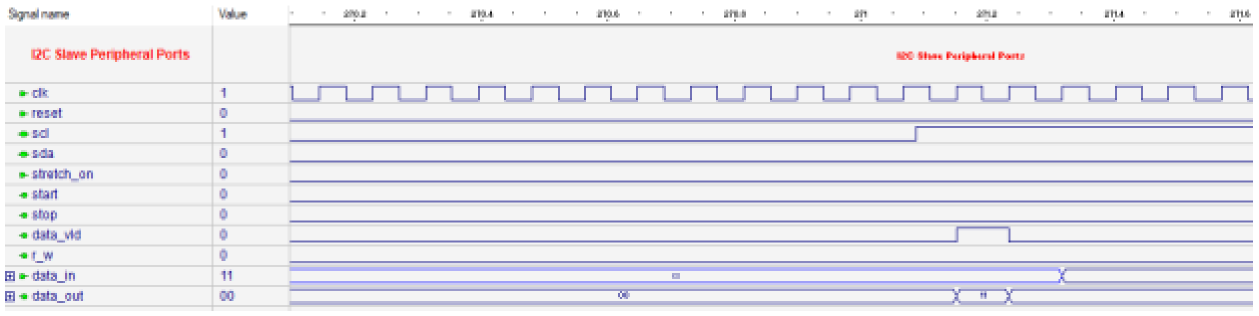


Figure 6.2. Zoomed-In View When 0x11 is Received by the Slave

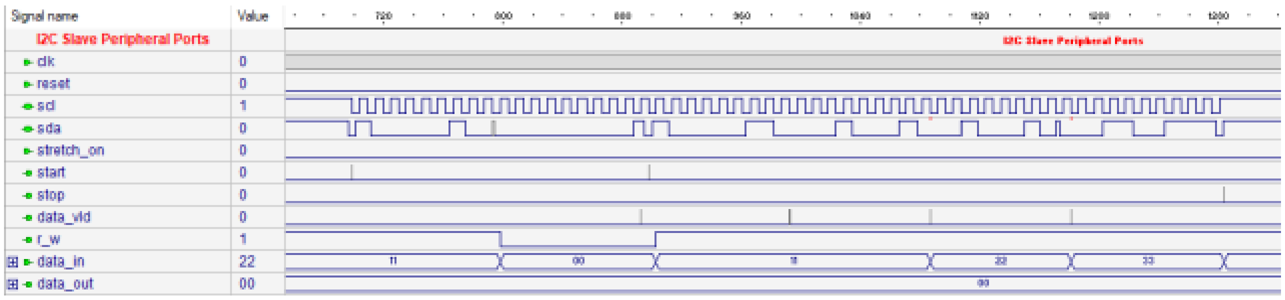


Figure 6.3. 3-Byte I^{2}C Read

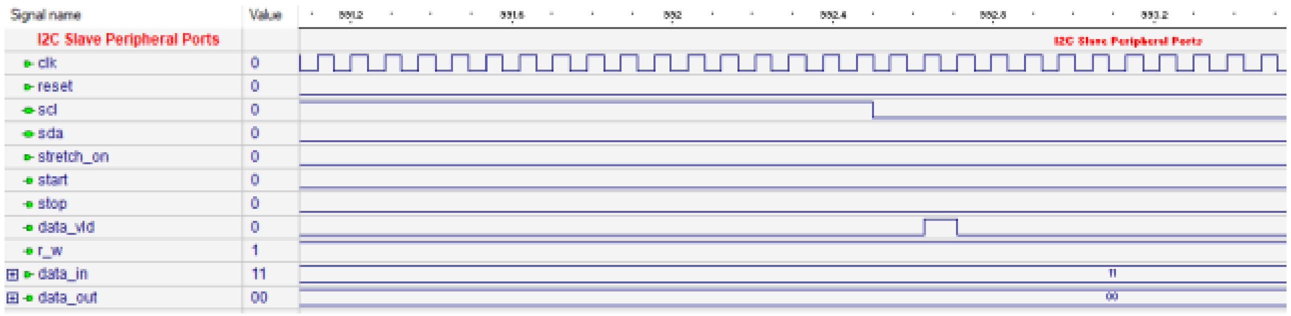
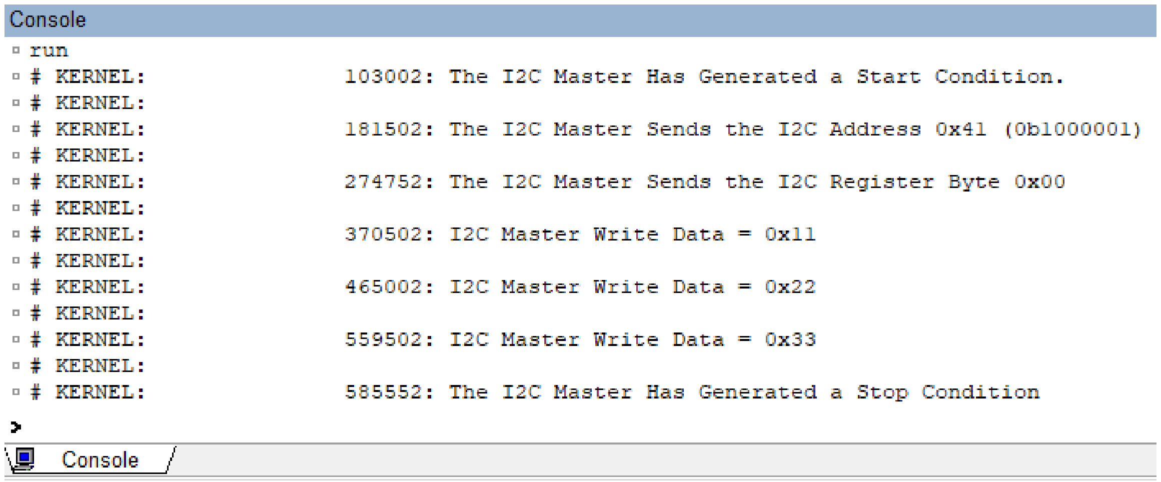


Figure 6.4. Zoomed-In View When 0x11 is Fetched by the I^{2}C Slave Module



```
Console
run
# KERNEL:              103002: The I2C Master Has Generated a Start Condition.
# KERNEL:
# KERNEL:              181502: The I2C Master Sends the I2C Address 0x41 (0b1000001)
# KERNEL:
# KERNEL:              274752: The I2C Master Sends the I2C Register Byte 0x00
# KERNEL:
# KERNEL:              370502: I2C Master Write Data = 0x11
# KERNEL:
# KERNEL:              465002: I2C Master Write Data = 0x22
# KERNEL:
# KERNEL:              559502: I2C Master Write Data = 0x33
# KERNEL:
# KERNEL:              585552: The I2C Master Has Generated a Stop Condition
>
Console
```

Figure 6.5. Aldec Active-HDL Console View

7. Packaged Design

The reference design folder (Soft_I2C_Slave_Peripheral) 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 Lattice Radiant project file (.LDF and .RDF).
- Simulation – contains subfolders for each FPGA Family. Each of these subfolders contains the simulation files (.DO) used to run RTL simulation on Aldec Active-HDL and Mentor Modelsim.
- Source – contains the I^{2}C slave peripheral RTL files (i2cslave_controller.v and i2cslave_controller_top.v).
- Testbench – contains subfolders for each FPGA family. Each of these subfolders contains the testbench I^{2}C Master (i2c_master), compiler directives file (tb_defines.v) and the top-level testbench file (tb.v).

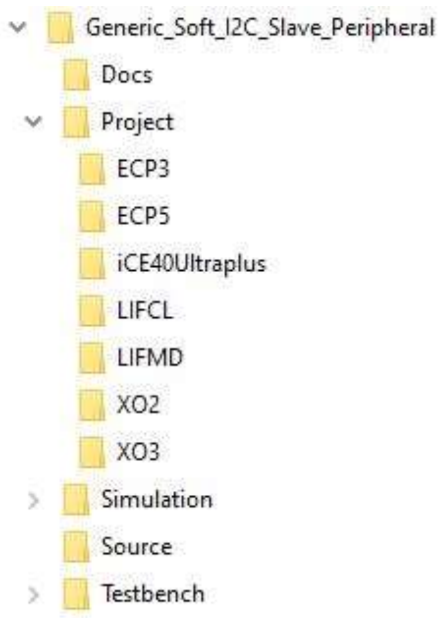
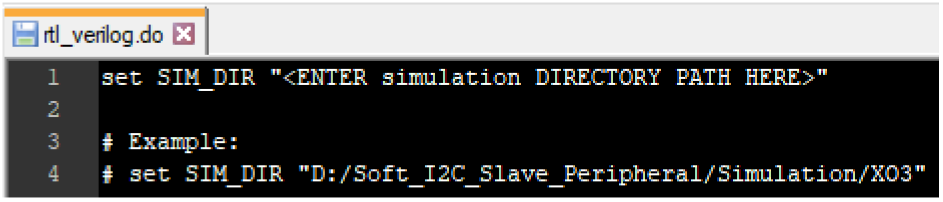


Figure 7.1. Packaged Design Directory Structure

7.1. Using the Simulation File (.DO)

To use the simulation file:

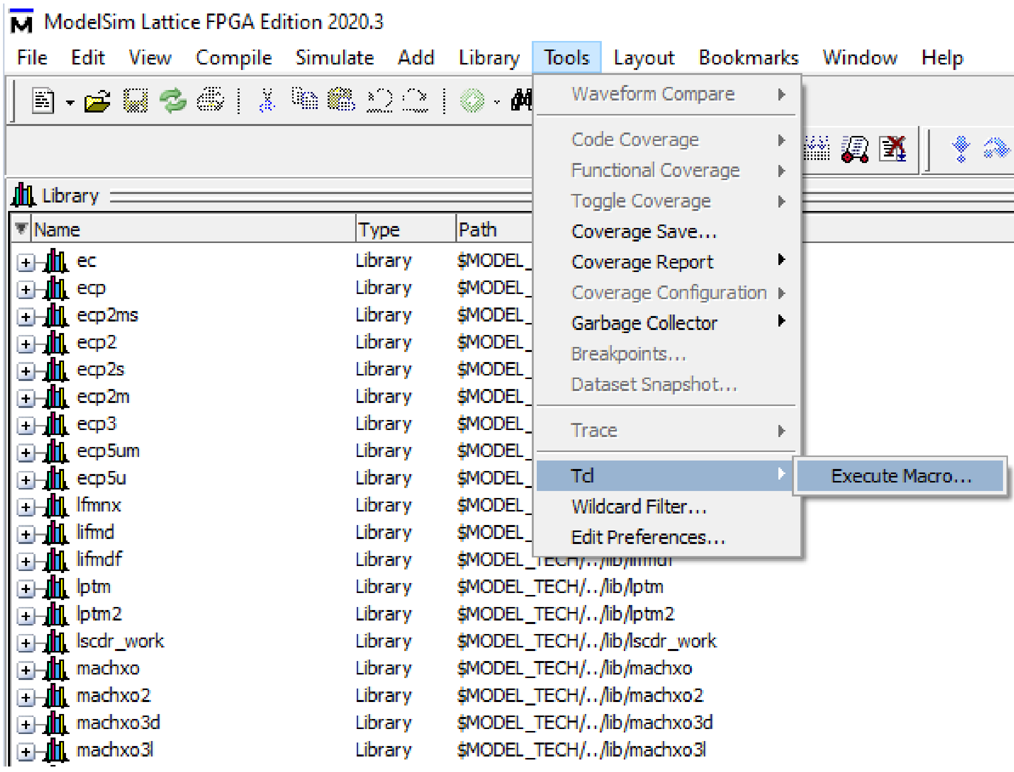
1. Open the DO file on a text editor and replace the text **<ENTER simulation DIRECTORY PATH HERE>** from Line 1 with the directory path of the simulation file. An example is seen on Line 4 of the file.



```
1  set SIM_DIR "<ENTER simulation DIRECTORY PATH HERE>"
2
3  # Example:
4  # set SIM_DIR "D:/Soft_I2C_Slave_Peripheral/Simulation/XO3"
```

Figure 7.2. Changing the Simulation Directory

2. Run the simulation script file on Mentor ModelSim as shown in Figure 7.3.



Name	Type	Path
ec	Library	$MODEL_
ecp	Library	$MODEL_
ecp2ms	Library	$MODEL_
ecp2	Library	$MODEL_
ecp2s	Library	$MODEL_
ecp2m	Library	$MODEL_
ecp3	Library	$MODEL_
ecp5um	Library	$MODEL_
ecp5u	Library	$MODEL_
lfmnx	Library	$MODEL_
lifmd	Library	$MODEL_
lifmdf	Library	$MODEL_TECH/../lib/lifmdf
lptm	Library	$MODEL_TECH/../lib/lptm
lptm2	Library	$MODEL_TECH/../lib/lptm2
lscdr_work	Library	$MODEL_TECH/../lib/lscdr_work
machxo	Library	$MODEL_TECH/../lib/machxo
machxo2	Library	$MODEL_TECH/../lib/machxo2
machxo3d	Library	$MODEL_TECH/../lib/machxo3d
machxo3l	Library	$MODEL_TECH/../lib/machxo3l

Figure 7.3. Running the Simulation Script for Mentor ModelSim

8. 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).

9. Implementation

This design is implemented in Verilog. When using this design in a different device or strategy setting, the 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 fails 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 I²C Master and Slave Write-Read Demo (FPGA-UG-02122).

Table 9.1. Resource Utilization

Device Family	Language	Utilization (LUTs)	f_{MAX} (MHz)	I/O
LatticeECP3[1]	Verilog	149	>50	25
ECP5[2]	Verilog	149	>50	25
CrossLink[3]	Verilog	155	>50	25
CrossLink-NX[4]	Verilog	157	>50	25
iCE40 UltraPlus[5]	Verilog	240	>50	25
MachXO2[6]	Verilog	148	>50	25
MachXO3[7]	Verilog	148	>50	25

Notes:

1. Performance and utilization characteristics are generated using LFE3-35EA-8FN484C with Lattice Diamond 3.12 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 3.12 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
3. Performance and utilization characteristics are generated using LIF-MD6000-6MG81I with Lattice Diamond 3.12 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 3.1 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
5. Performance and utilization characteristics are generated using iCE40UP5K-SG48I with Lattice Radiant 3.1 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 3.12 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 3.12 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.

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 (FPGA-DS-02056)
- MachXO3 Family Data Sheet (FPGA-DS-02032)
- iCE40 UltraPlus Family Data Sheet (FPGA-DS-02008)
- Generic Soft I^{2}C Master Controller (FPGA-RD-02201)
- Generic Soft I^{2}C 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 1.3, January 2023

Section	Change Summary
All	Global update to Lattice Diamond 3.12 and Lattice Radiant 3.1.
Customization	• Table 5.1. Compiler Directives Option: • updated to ADDR_MODE_7BIT as well as the remarks; • newly added stretch_test and its remarks; • removed the original I2C Mode Selection category. • Updated Figure 5.2. Compiler Directive Customization Example.
Packaged Design	Removed the Aldec Active-HDL related contents including the original Figure 7.3. Running the Simulation Script for Aldec Active-HDL from the Using the Simulation File (.DO) section.
Implementation	Table 9.1. Resource Utilization: • updated utilization (LUTs) for CrossLink-NX and iCE40 UltraPlus families; • updated Note 3 and Note 5 contents changing the devices; • Removed the original Note 8.

Revision 1.2, August 2021

Section	Change Summary
All	Minor adjustments in formatting.
Functional Description	• Updated content in I2C Write Timing and I2C Read Timing section. • Updated Table 3.1.
Operation Sequence	• Changed Write with Repeated Start to Combined Format. • Updated Figure 4.1, Figure 4.2, and Figure 4.3.
Customization	Updated Figure 5.1 and Figure 5.2.
HDL Simulation and Verification	Updated section content including Figure 6.3 to Figure 6.5.
Packaged Design	Updated content including Figure 7.3 and Figure 7.4.

Revision 1.1, December 2020

Section	Change Summary
All	Updated Lattice Radiant version to 2.1 across the document.
Acronyms in This Document	Added Embedded Block RAM as definition.
Features	Updated content to add CrossLink-NX.
Functional Description	• Updated content in Functional Block Diagram, I2C Write Timing, and I2C Read Timing section. • Updated Table 3.1. • Updated Figure 3.1, Figure 3.2, and Figure 3.3.
Operation Sequence	• Added Single/Multi-Byte Write Operation, Single/Multi-Byte Write Operation with Repeat Start, and Clock Stretching section. • Updated Figure 4.3, Figure 4.4, Figure 4.5, and Figure 4.6.
Customization	• Updated section content. • Updated Figure 5.1 and Figure 5.2. • Added Table 5.1.
HDL Simulation and Verification	Updated section content, including figures.
Packaged Design	Updated Figure 7.1.
Hardware Validation	Updated section content.
Implementation	Updated content and footnotes in Table 9.1.
References	Updated content to add Generic Soft I^{2}C Master Controller and Soft I^{2}C Master and Slave Demo reference.

Revision 1.0, May 2020

Section	Change Summary
All	Initial release