TAMPERE UNIVERSITY OF TECHNOLOGY

**THANG LUONG CAO**

**Serial bus adapter design for FPGA**

Master of Science Thesis

Examiner: Prof. Timo D. Hämäläinen
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
On May 6, 2015.

# ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Electrical Engineering
**THANG, LUONG CAO: Serial bus adapter design for FPGA**
Master of Science Thesis, 63 pages
April 2015
Major: Communication Circuits and Systems
Examiner: Prof. Timo D. Hämäläinen
Keywords: Serial bus, $I^2C$, adapter, SoC, FPGA, HPS, simulation

In recent years, FPGAs (Field Programmable Gate Arrays) have become a popular platform for testing and implementing hardware designs by increasing their capacity and cost efficiency in the competition with Application Specific Integrated Circuits (ASICs). Processors can be used for any problem but they have not been optimized for specific problems. The design of ASIC is an extremely complex task, very time consuming and expensive; they are used for mass products. FPGA is an intermediate solution between general purpose processors and ASICs. Altera Cyclone V 28nm is a *System On Chip (SoC),* which integrates a *Hard Processor Core (HPS),* peripherals, and memory controller with the FPGA fabric. However, HPS consist of only one-directional serial data (SDA) buses and serial clock (SCL) buses and provides support for a communication link only between integrated circuits on a board. It is necessary to build an $I^2C$ serial bus adapter in order to communicate between HPS and other devices outside the board.

$I^2C$ serial bus adapter is implemented and tested in this thesis. It adapts the communication from one-directional serial data line of hard processor system to bi-directional data line. In order to test the $I^2C$ adapter in both writing data operation and reading data operation, Signal Generator blocks to generate testing signals are implemented and $I^2C$ Slave block from OpenCores to detect and display data to LEDs is used. All the blocks are implemented in *VHSIC Hardware Description Language (VHDL).*

The verifications for $I^2C$ Adapter, Signal Generator and $I^2C$ Slave are inspected by waveforms on Modelsim SE 10.2c simulator. The block implementations are compiled and programmed by Quartus II 13.1 to DE1-SoC FPGA development board. DE1-SoC board buttons and LEDs are used to test the $I^2C$ adapter operation by a user. The results show that the adapter works as specified.

# PREFACE

The research through my Master's Thesis was conducted during academic year 2014-2015 at the Department of Pervasive Computing at Tampere University of Technology.

I would like to thank Prof. Timo D. Hämäläinen for the opportunity to work this project. I am especially grateful to Dr. Tech. Erno Salminen for his time and dedication provided to the completion of my thesis, because of his long design meetings, guidance and advices.

I also would like to thank to all of my friends in Finland who were being with me during time in this beautiful country, my good friend Stefanus Arinno for helping me a lot and Julio Cesar for his motivation pictures at my work station office forcing me to finish my thesis all the time.

Last but not the least; I am grateful for my family, my parents Long and Nguyen, my younger brother Loc for their support throughout my study and life.

Tampere, April 2015

Luong Cao Thang

# CONTENTS


1. INTRODUCTION ..... 1
2. DEVICES AND I$^2$C PROTOCOL ..... 6
   - 2.1 Platform ..... 6
   - 2.2 Utilized tools ..... 7
   - 2.3 I$^2$C Protocol ..... 8
     - 2.3.1 Introduction ..... 8
     - 2.3.2 Protocol ..... 8
     - 2.3.3 Start and stop conditions ..... 10
     - 2.3.4 Byte format ..... 10
     - 2.3.5 Acknowledge (ACK) and not acknowledge (NACK) ..... 11
     - 2.3.6 R/***W*** bit ..... 12
3. IMPLEMENTATION ON FPGA ..... 13
   - 3.1 Block Diagrams ..... 13
     - 3.1.1 Writing data process ..... 13
     - 3.1.2 Reading data process ..... 14
   - 3.2 Signal generator ..... 15
     - 3.2.1 Signal generator to write ..... 15
     - 3.2.2 Signal generator to read ..... 17
   - 3.3 I$^2$C Adapter ..... 19
     - 3.3.1 I$^2$C Adapter Block ..... 19
     - 3.3.2 Finite State Machine ..... 20
     - 3.3.3 How I$^2$C Adapter work ..... 22
   - 3.4 I$^2$C Slave ..... 27
   - 3.5 Top Level Design ..... 29
4. VERIFICATION AND RESULT ..... 30
   - 4.1 Writing Data Process Verification ..... 30
     - 4.1.1 Verification of Signal Generator To Write in Writing Data Process ..... 30
     - 4.1.2 Verification of I$^2$C Adapter in Writing Data Process ..... 35
     - 4.1.3 Verification of I$^2$C Slave in Writing Data Process ..... 40
   - 4.2 Reading Data Process Verification ..... 45
     - 4.2.1 Verification of Signal Generator To Read in Reading Data Process ..... 45
     - 4.2.2 Verification of I$^2$C Adapter in Reading Data Process ..... 49
     - 4.2.3 Verification of I$^2$C Slave in Reading Data Process ..... 54
   - 4.3 Whole System Verification ..... 59

4.3.1 Register Transfer Level ..... 59
4.3.2 Assign Pins and Display Result by DE1-SoC ..... 60
5. CONCLUSIONS ..... 62

# LIST OF FIGURES

*Figure 1.1 Altera SoC FPGA Device Block Diagram* . ..... *2*
*Figure 1.2 Board Block Diagram* . ..... *2*
*Figure 1.3 Block Diagram on DE1-SoC.* ..... *4*
*Figure 1.4 Block Diagram on DE1-SoC for Writing Data Operation.* ..... *5*
*Figure 1.5 Block Diagram on DE1-SoC for Reading Data Operation.* ..... *5*

*Figure 2.1 DE1-SoC Board* . ..... *6*
*Figure 2.2 $I^2C$ bus configuration* . ..... *8*
*Figure 2.3 Start and Stop conditions* . ..... *10*
*Figure 2.4 Data transfer on the $I^2C$ bus* . ..... *11*
*Figure 2.5 Acknowledgement on $I^2C$ bus* ..... *11*
*Figure 2.6 A complete data transfer* . ..... *12*

*Figure 3.1 Writing data process block diagram.* ..... *13*
*Figure 3.2 Reading data process block diagram.* ..... *14*
*Figure 3.3 Signal Generator To Write Block.* ..... *15*
*Figure 3.4 Address and data signals waveform generated by Signal Generator To Write.* ..... *16*
*Figure 3.5 Signal Generator To Read Block.* ..... *17*
*Figure 3.6 Address and data signals waveform generated by Signal Generator To Read.* ..... *18*
*Figure 3.7 I2C Adapter Block.* ..... *19*
*Figure 3.8 Finite state machine of I2C Adapter block.* ..... *21*
*Figure 3.9 I2C Slave Block* ..... *27*
*Figure 3.10 Write Data Top Design Block* ..... *29*
*Figure 3.11 Read Data Top Design block* ..... *29*

*Figure 4.1 Modelsim testbench for reset, start and stop of Signal Generator To Write in Writing Data Process.* ..... *31*
*Figure 4.2 Modelsim testbench for address and data transmission of Signal Generator To Write in Writing Data Process.* ..... *32*
*Figure 4.3 Signal Generator To Write Block on RTL Viewer of Quartus II in Writing Data Process.* ..... *33*
*Figure 4.4 Inside Signal Generator To Write Block on RTL Viewer of Quartus II in Writing Data Process.* ..... *34*
*Figure 4.5 Modelsim testbench for reset, start and stop of I2C Adapter in Writing Data Process.* ..... *35*

*Figure 4.6 Modelsim testbench for address transmission of I2C Adapter in Writing Data Process.* ..... *36*
*Figure 4.7 Modelsim testbench for data transmission of I2C Adapter in Writing Data Process.* ..... *37*
*Figure 4.8 I2C Adapter Block on RTL Viewer of Quartus II in Writing Data Process.* ..... *38*
*Figure 4.9 Inside I2C Adapter Block on RTL Viewer of Quartus II in Writing Data Process.* ..... *39*
*Figure 4.10 Modelsim testbench for reset, start and stop of I2C Slave in Writing Data Process.* ..... *40*
*Figure 4.11 Modelsim testbench for address transmission of I2C Slave in Writing Data Process.* ..... *41*
*Figure 4.12 Modelsim testbench for data transmission of I2C Slave in Writing Data Process.* ..... *42*
*Figure 4.13 I2C Slave Block on RTL Viewer of Quartus II in Writing Data Process.* ..... *43*
*Figure 4.14 Inside I2C Slave Block on RTL Viewer of Quartus II in Writing Data Process.* ..... *44*
*Figure 4.15 Modelsim testbench for reset, start and stop of Signal Generator To Read in Reading Data Process.* ..... *45*
*Figure 4.16 Modelsim testbench for address and data transmission of Signal Generator To Read in Reading Data Process.* ..... *46*
*Figure 4.17 Signal Generator To Read Block on RTL Viewer of Quartus II in Reading Data Process.* ..... *47*
*Figure 4.18 Inside Signal Generator To Read Block on RTL Viewer of Quartus II in Reading Data Process.* ..... *48*
*Figure 4.19 Modelsim testbench for reset, start and stop of I2C Adapter in Reading Data Process.* ..... *49*
*Figure 4.20 Modelsim testbench for address transmission of I2C Adapter in Reading Data Process.* ..... *50*
*Figure 4.21 Modelsim testbench for data transmission of I2C Adapter in Reading Data Process.* ..... *51*
*Figure 4.22 I2C Adapter Block on RTL Viewer of Quartus II in Reading Data Process.* ..... *52*
*Figure 4.23 Inside I2C Adapter Block on RTL Viewer of Quartus II in Writing Data Process.* ..... *53*
*Figure 4.24 Modelsim testbench for reset, start and stop of I2C Slave in Reading Data Process.* ..... *54*
*Figure 4.25 Modelsim testbench for address transmission of I2C Slave in Reading Data Process.* ..... *55*
*Figure 4.26 Modelsim testbench for data transmission of I2C Slave in Reading Data Process.* ..... *56*

*Figure 4.27 I2C Slave Block on RTL Viewer of Quartus II in Reading Data Process.* ..... *57*
*Figure 4.28 Inside I2C Slave Block on RTL Viewer of Quartus II in Reading Data Process.* ..... *58*
*Figure 4.29 Top Level Design for Writing Data Process on RTL Viewer of Quartus II.* ..... *59*
*Figure 4.30 Top Level Design for Reading Data Process on RTL Viewer of Quartus II.* ..... *59*
*Figure 4.31 Assignment Editor for Top Level Design.* ..... *60*
*Figure 4.32 Output of Top Level Design Block displaying on Leds of DE1-SoC in Writing Data Process.* ..... *61*
*Figure 4.33 Output of Top Level Design Block displaying on Leds of DE1-SoC in Reading Data Process.* ..... *61*

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| FPGAs | Field Programmable Gate Arrays |
| ASICs | Application Specific Integrated Circuits |
| SoC | System on Chip |
| HPS | Hard Processor System |
| PCB | Printed Circuit Board |
| SDRAM | Synchronous dynamic random access memory |
| DAC | Digital to analog converter |
| ADC | Analog to digital converter |
| I2C | I squared C |
| SDA | Serial Data Line |
| SCL | Serial Clock Line |
| ACK | Acknowledge |
| NACK | Not Acknowledge |
| SPI | Serial Peripheral Interface |
| UART | Universal Asynchronous Receiver Transmitter |
| CAN | Control Area Network |
| USB | Universal Serial Bus |
| Rst | Reset |
| Rst_n | Negative Reset |
| Clk | Clock |
| Clk_50 | Clock 50 MHz |
| Scl_ex | Serial clock line extra |
| Sda_ex | Serial data line extra |
| Hps_scli | Hard processor system serial clock line input |
| Hps_sdi | Hard processor system serial data line input |
| Hps_sclo | Hard processor system serial clock line output |
| Hps_sdo | Hard processor system serial data line output |
| Data_in | Data input |
| Wr | Write state |
| Rd | Read state |
| Rw_wr | Read/Write of Write State |

| | |
|---|---|
| Rw_rd | Read/Write of Read State |
| Ack_wr | Acknowledge of Write State |
| Ack_rd | Acknowledge of Read State |
| ICs | Integrated Circuits |
| MSB | Most Significant Bit |
| LSB | Least Significant Bit |
| I/O | Input/output |
| RTL | Register Transfer Level |
| FSM | Finite State Machine |
| SPD | Serial Presence Detect |
| EEPROMs | Electrically Erasable Programmable Read-Only Memory |
| NVRAM | Non-volatile Random Access Memory |

# 1. INTRODUCTION

A *Field Programmable Gate Array (FPGA)* is an integrated circuit designed to be configured by a customer or designer after manufacturing. The basic blocks in an FPGA device are *Logic Elements (LE)* and the interconnections between them are programmable to communicate each other [1]. Reprogrammable logic device provides a fast and cost efficient way for testing and implementing custom digital designs. A variety of reusable *Intellectual Property (IP)* components allows the designer to create complex designs in reasonable time and synthesizable soft-core processors providing the possibility to implement functionality using software. It is often easier and faster to implement complex functionality using software rather than implementing the same functionality on hardware logic [2]. The *field programmable* can be understood as the ability to program it in the field or programming can be done by the end-user [3]. In compare to processors [4] and *Application Specific Integrated Circuits (ASICs)* [5], FPGAs are inexpensive and they can even outperform the others, since designers can develop application specific logic that take advantage of the inner parallelism of the given problem; FPGA are designed to provide good performance for any application, whereas ASIC are just designed for a given problem. Beside the advantages of FPGA over ASIC in terms of flexible reprogrammable ability, early testing stages, shorter time-to-market, it cannot get over ASIC by area, delay, power consumption and unit price in high volume products. There is a measurement done by Kuon and Rose in their research about the gap between FPGAs and ASICs [6].

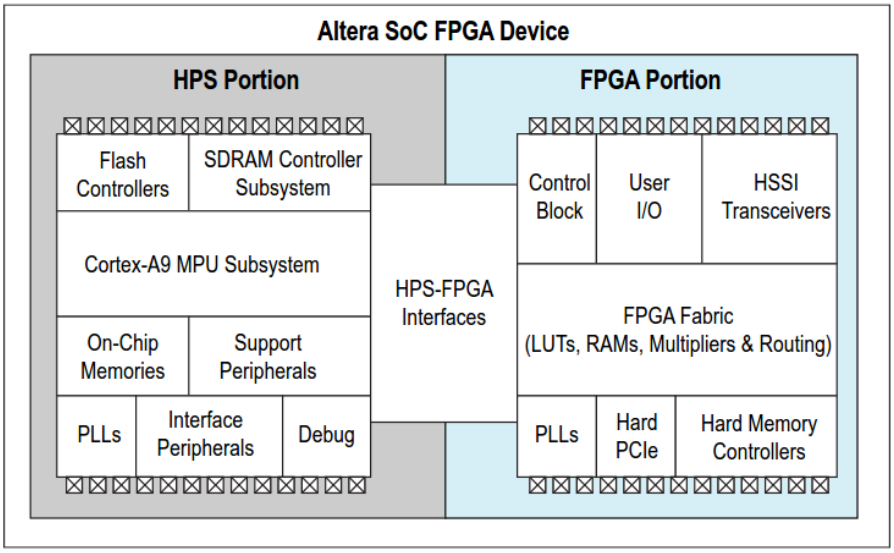


***Figure 1.1*** *Altera SoC FPGA Device Block Diagram* [8].

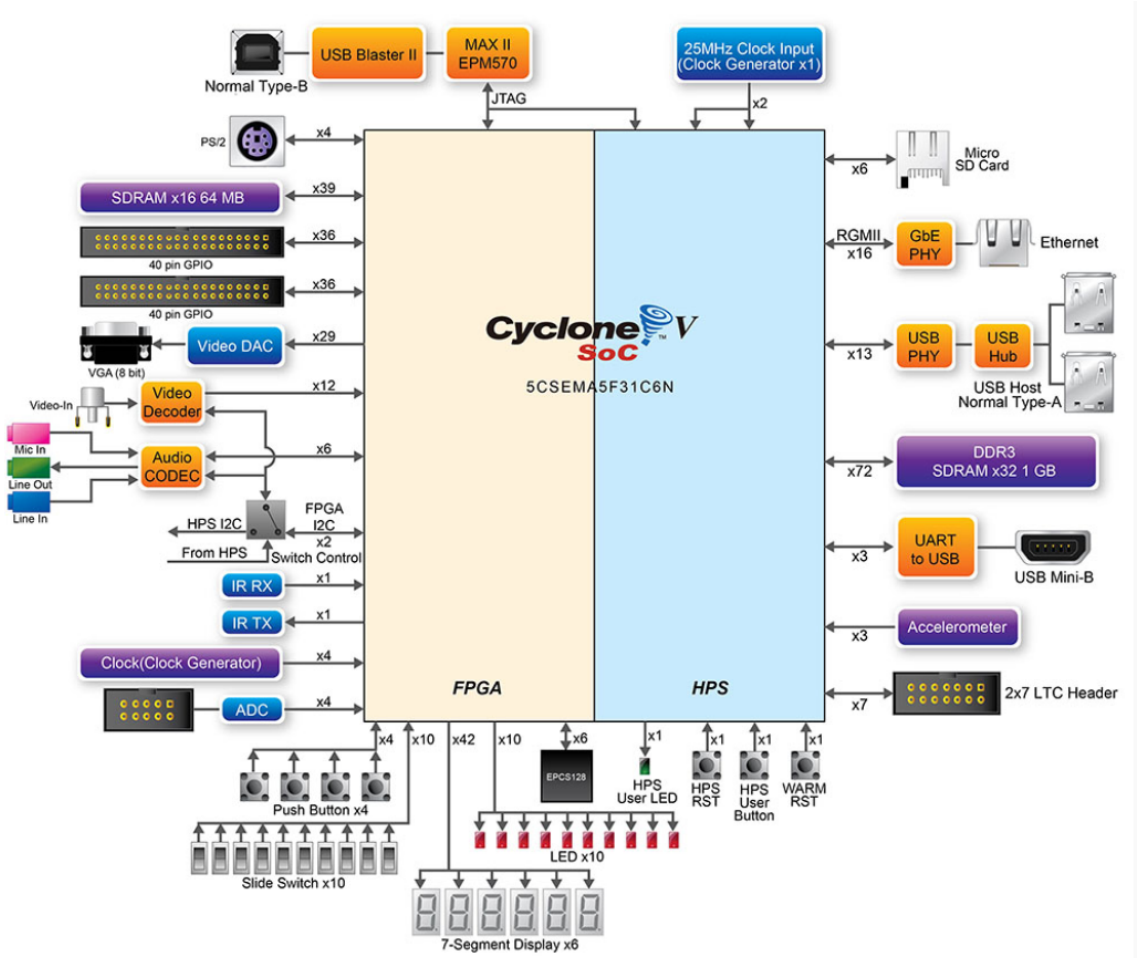


***Figure 1.2*** *Board Block Diagram* [9].

Altera Cyclone V 28nm is a *System On Chip (SoC),* which integrates a *Hard Processor Core (HPS),* peripherals, and memory controller with the FPGA fabric using a high-bandwidth interconnect backbone [7][8]. Cyclone V is available on DE1-SoC FPGA development board. The Altera SoC FPGA Device Block Diagram is shown in Figure 1.1 and Figure 1.2 illustrates the DE1-SoC board block diagram for Cyclone V in a FPGA development board.

Inter-Integrated Circuit, abbreviated as $I^2C$ is a serial bus short distance protocol invented by Philips Semiconductor to transfer data among ICs. Because of advantages in simplicity and low manufacturing cost, $I^2C$ is nowadays one of the most popular serial bus communication protocols in the market together with other serial bus communication protocols such as SPI [26], UART [27], CAN [28], USB [29], and so on. There are many devices, which have I/O $I^2C$ interface and communicate with other devices following the $I^2C$ protocol. Examples of $I^2C$ compatible devices are Analog to Digital Converter, Digital to Analog Converter, EEPROM, Real Time Clock [30],Real Time Calendar [31], Temperature Sensor [32], LCD multimedia color touch panel from TerasIC, and so on.

The $I^2C$ protocol is applied to $I^2C$ compatible devices which have bi-directional signals serial data (SDA) and serial clock (SCL). However, the Hard Processor System on Altera Cyclone V has only one-directional $I^2C$ buses, one serial clock and one serial data for signals coming in to HPS, and one serial clock and one serial data for signals coming out from HPS. In order to adapt the communication between one-directional serial data line of HPS and bi-directional serial data line following the $I^2C$ protocol, it is necessary to implement a serial bus adapter as illustrated in Figure 1.3.

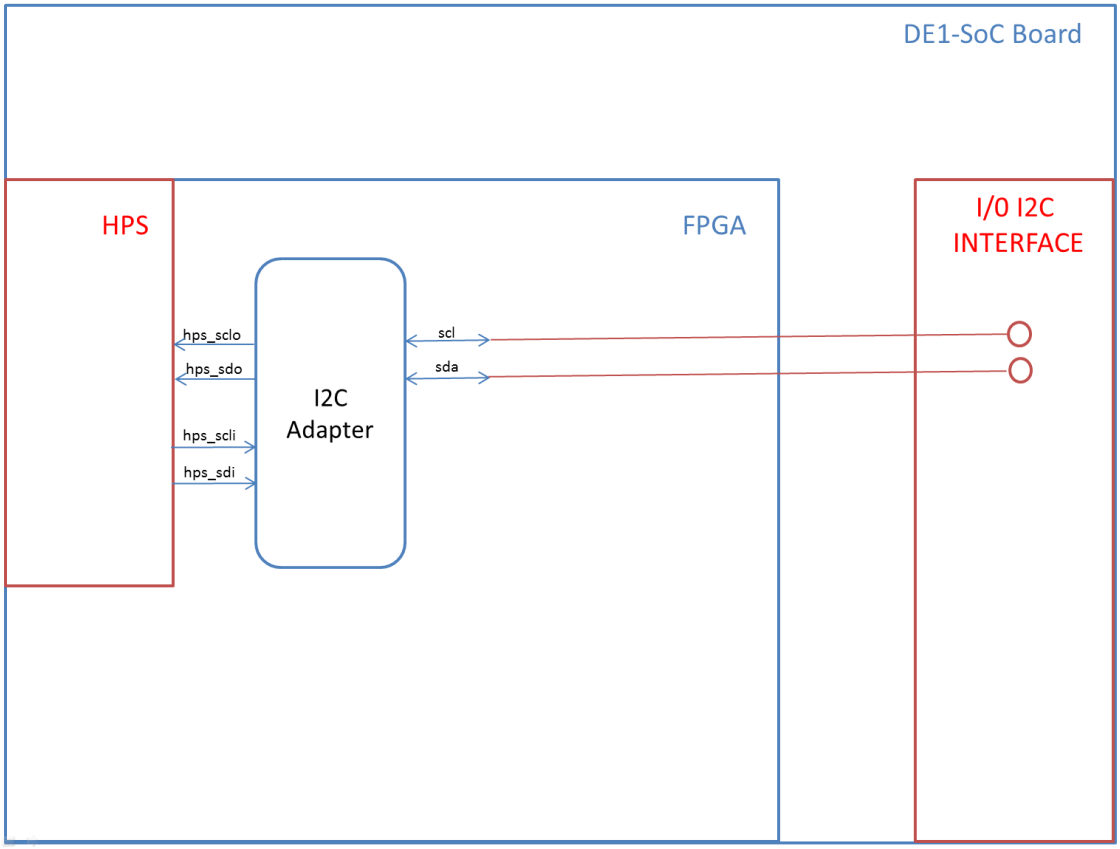


***Figure 1.3*** *Block Diagram on DE1-SoC.*

For this thesis, a serial bus Adapter to make communication between HPS Portion and FPGA Portion following $I^2C$ protocol in Altera SoC FPGA Device is created. In order to test the $I^2C$ Adapter, there is also Signal Generator to produce testing signals and an $I^2C$ Slave from OpenCores is involved. The data generated by the Signal Generator after going through $I^2C$ Adapter is detected at $I^2C$ Slave and displayed to the LEDs of the DE1-SoC Board for Writing Data Operation and Reading Data Operation as shown in Figure 1.4 and Figure 1.5 respectively. The thesis is divided into the following chapters. Chapter 2 introduces devices and tools used in the work. The implementation of blocks is presented in Chapter 3. Chapter 4 shows verification and results of block implementations by software as well as by compiling and programming the FPGA device. Chapter 5 is the conclusions for this thesis.

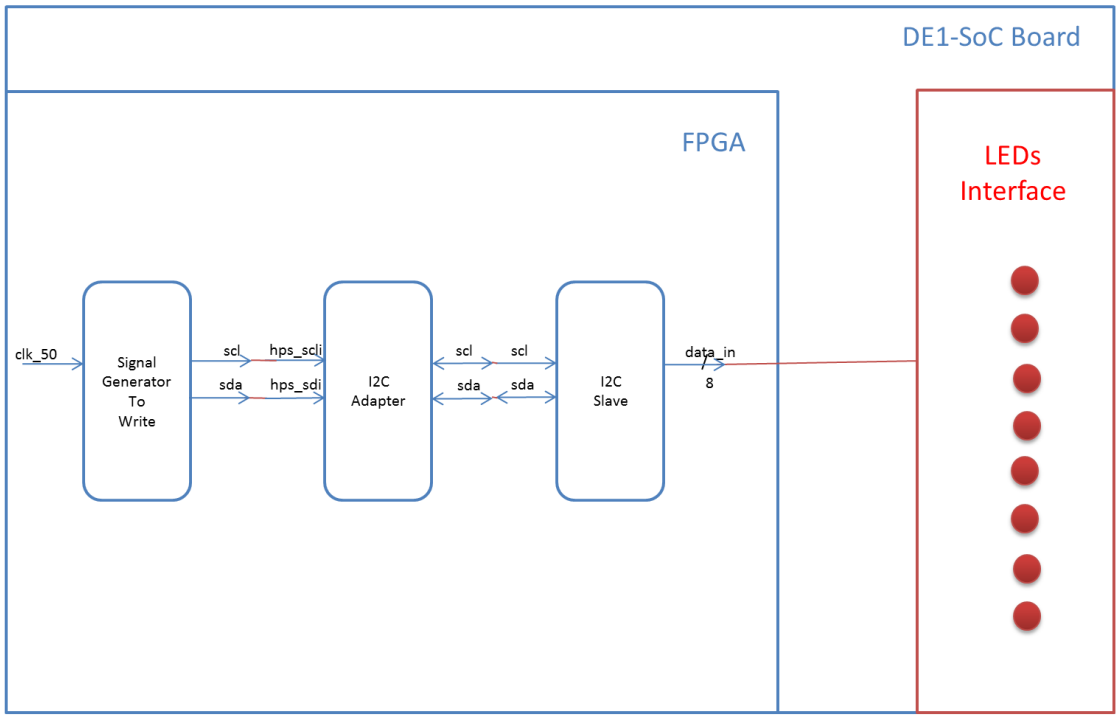


***Figure 1.4*** *Block Diagram on DE1-SoC for Writing Data Operation.*

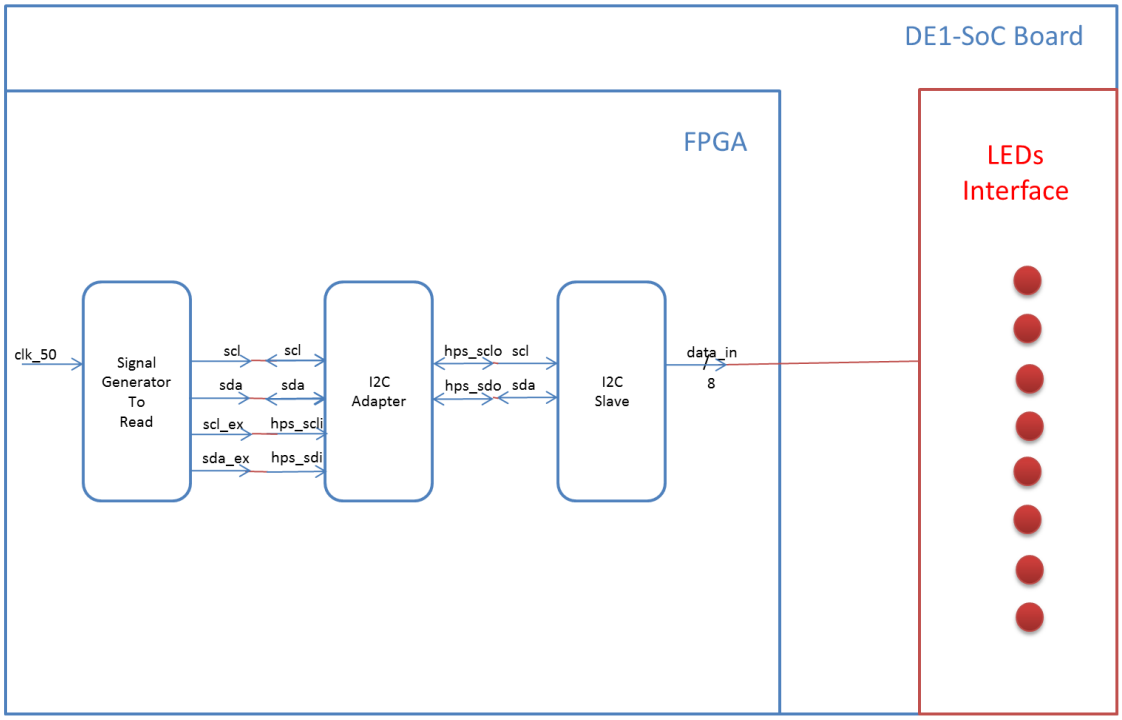


***Figure 1.5*** *Block Diagram on DE1-SoC for Reading Data Operation.*

# 2. DEVICES AND I²C PROTOCOL

This Chapter presents an introduction to the I²C protocol. Devices and utilized tools used in this work are described.

## 2.1 Platform

The platform used in this work is DE1-SoC FPGA development board. The DE1-SoC board has many features that allow users to implement a wide range of designed circuits, from simple circuits to various multimedia projects [10].

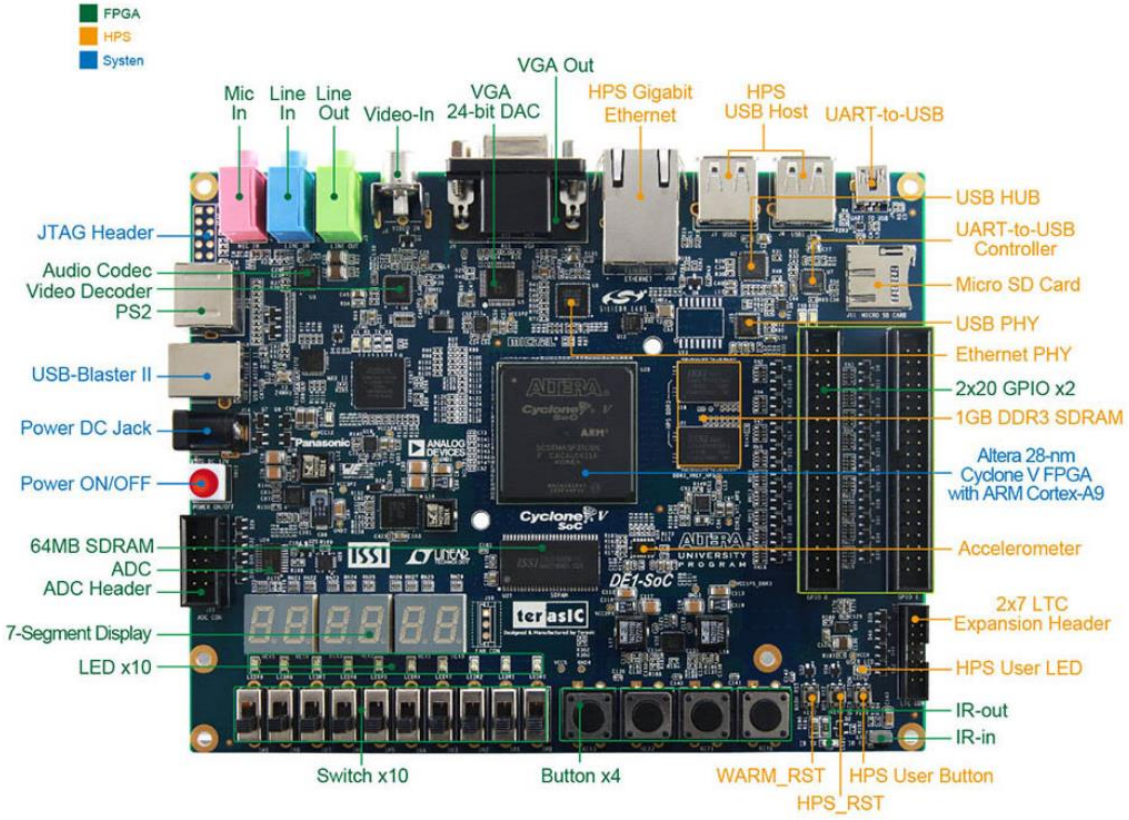


***Figure 2.1*** *DE1-SoC Board* [10].

An overview of the specification of DE1-SoC Board is following [10]:

FPGA Device:

- Cyclone V SoC 5CSEMA5F31C6 Device
- Duo-core ARM Cortex-A9 (HPS)
- 85K Programmable Logic Elements

- 44500 Kbits embedded memory
- 6 Fractional PLLs
- 2 Hard Memory Controllers

Configuration and Debug:

- Serial Configuration device – EPCS128 on FPGA
- On-Board USB Blaster II (Normal Type B USB connector)

Connectors:

- Two 40-pin Expansion Headers (voltage levels: 3.3V)
- One 10-pin ADC Input Header
- One LTC connector (One Serial Peripheral Interface (SPI) Master, one I2C and one GPIO interface)

Switches, Button and Indicators:

- 4 User Keys (FPGA x4)
- 10 User switches (FPGA x10)
- 11 User LEDs (FPGA x10; HPS x1)
- 2 HPS Reset Buttons (HPS_RST_n and HPS_WARM_RST_n)
- Six 7-segment displays

## 2.2 Utilized tools

The following tools were used in this work:

- Altera Quartus II [11]
  Quartus II is used for analysis and synthesis of the HDL design of the project. Quartus II enables the developer to compile designs, perform timing analysis, examine RLT diagrams, simulate a design's response to stimulation, and configure the target device with the programmer. The version used in the project is Quartus II 13.1 (64-bit).
- Modelsim [13]
  Modelsim is a hardware simulation and debug environment by Mentor Graphics, primarily targeted at smaller ASIC and FPGA designs. Modelsim is used to verify and simulate for VHDL design of project. The version used in the project is Modelsim SE 10.2 c.

## 2.3 $I^2C$ Protocol

### 2.3.1 Introduction

From 1980s, Philips Semiconductors Company created the $I^2C$ interface which is used for data transfer among ICs at the Printed Circuit Board (PCB) level. The concept is connecting all the $I^2C$ bus compatible devices which have an $I^2C$ interface. This concept allows devices communicate directly with each other devices via $I^2C$ bus [14].

In $I^2C$, designs proceed rapidly from block diagram to final schematic and interconnections are minimized that ICs have fewer pins. With simplicity and low manufacturing cost, $I^2C$ is common in many applications such as reading configuration data on SDRAM [15], supporting systems management for PCI cards [16], accessing low speed DACs [17] and ADCs [18], and display data channel. $I^2C$ is now implemented in over 1000 different ICs [19] and broadly adopted by many leading chip design companies like Intel, Texas Instrument, Analog Devices, *etc*.

### 2.3.2 Protocol

In $I^2C$, only two signal lines are required; a serial data line (SDA) and a serial clock line (SCL). Each device connected to the bus is software addressable by a unique address and simple master/slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers. The device that initiates communication is called the Master, and at that time, all the other devices on the bus are considered Slaves.

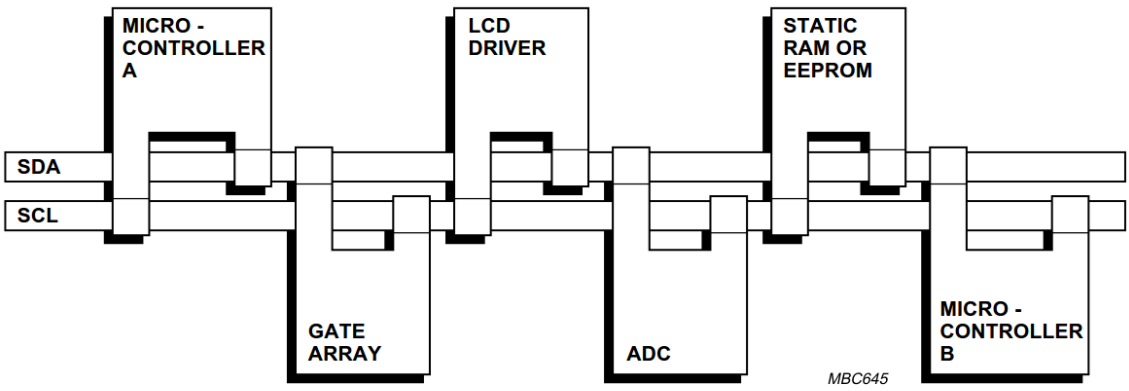


***Figure 2.2*** *$I^2C$ bus configuration* [20].

Figure 2.2 illustrates I2C bus configuration and Table 2.1 describe some basic I2C bus terminology. First, consider when microcontroller A wants to send information to microcontroller B, microcontroller A is master and addresses microcontroller B is the slave. Microcontroller A (master-transmitter) sends data to microcontroller B (slave-receiver) and microcontroller A terminates the transfer. When a Master wants to initiate a communication, it issues a "START" condition. At that time, Slave has to listen to the bus for incoming data. After the "START" is issued, the Master sends the "ADDRESS" of the Slave that it wishes to communicate with along with a bit to indicate the direction of the data transfer (either read or write). Slave will then compare its address with the address received on the bus. If the address matches, the Slave will send an "ACKNOWLEDGEMENT" (ACK) to the Master. Slave whose address does not match will not send an ACK. Once communication is established, the two lines are busy. No other device is allowed to control the lines except the Master and the Slave which was selected. When the Master wants to terminate communication, it will issue a "STOP" signal. After that, both SCL line and SDA line are released and free.

So far we have introduced the "START", "ADDRESS", "ACKNOWLEDGEMENT" and "STOP" signals. We will discuss these signals in more detail later. Terms used in $I^2C$ bus are summarized in the Table 2.1.

***Table 2.1*** *Definition of $I^2C$ bus terminology* [21].

| **Term** | **Description** |
|---|---|
| Transmitter | the device which sends data to bus |
| Receiver | the device which receives data from bus |
| Master | the device which initiates a transfer, generate clock signals and terminates a transfer |
| Slave | the device addressed by a master |

### 2.3.3 Start and stop conditions

When a Master wants to initiate a data transfer, it issues a START condition and when it wants to terminate the transfer, a STOP condition will be initiated. There can be multiple STARTs during once transaction called a repeated START. The Master can then release the STOP condition whenever it wants to.

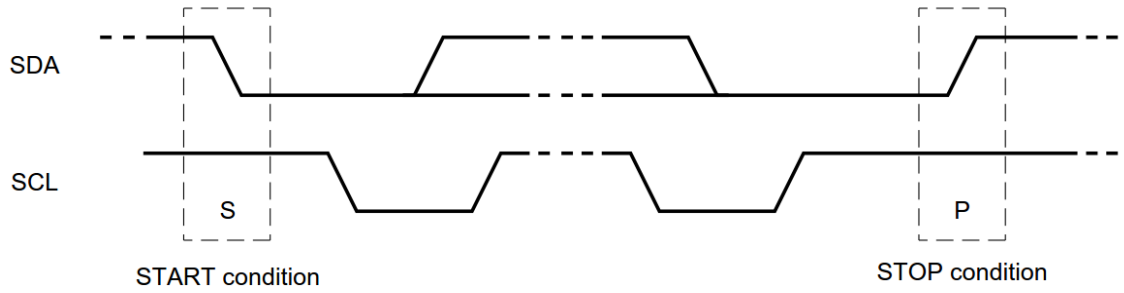


***Figure 2.3*** *Start and Stop conditions* [21].

As you can see in Figure 2.3, a START is issued by bringing the SDA line low while the SCL line is high. A STOP condition is implemented by transitioning the SDA line high while the SCL line is high. START and STOP conditions are always generated by the Master. The bus is considered to be busy after the START condition. The bus stays busy if a repeated START is generated instead of a STOP condition. In this respect, the STARTS and repeated START conditions are functionally identical. After that the Master controls the SCL line and can generate clock signals.

### 2.3.4 Byte format

The $I^2C$ bus is a byte-oriented protocol. After signaling Slave by the START condition, the Master sends "starting byte" to the Slave. There are two components that make us the "starting bytes": Slave address and data direction (Read or Write). The Master sends the MSB (Most Significant Bit) first and the LSB (Least Significant Bit) last. There are two addressing modes in the $I^2C$ protocol: the 7-bit and 10-bit address modes.

We will first consider the 7-bit addressing mode. Every byte put on the SDA line must be eight bits long. The number of bytes that can be transmitted per transfer is unrestricted. Each byte must be followed by an Acknowledge bit. If a Slave cannot receive or transmit another complete byte of data until it has performed some other function, for example servicing an internal interrupt, it can hold the clock line SCL LOW to force the master into a wait state. Data transfer then continues when the Slave is ready for another byte of data and releases clock line SCL. Data transfer on the $I^2C$ bus is illustrated in the Figure 2.4.

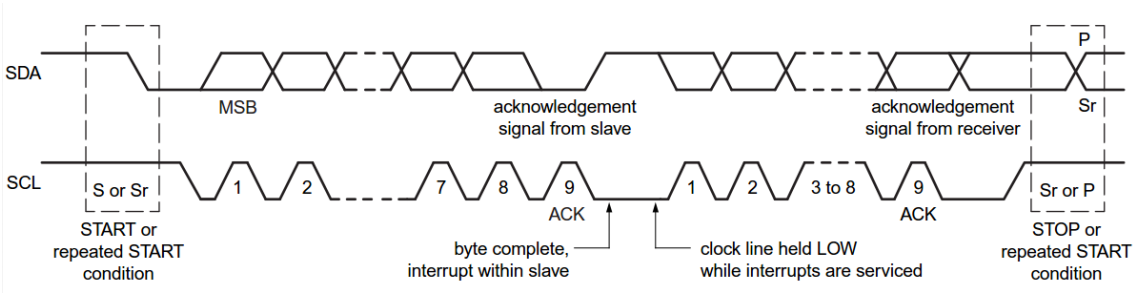


***Figure 2.4*** *Data transfer on the $I^2C$ bus* [21].

With the 10-bit addressing mode, when the $I^2C$ bus became more popular, it was recognized that the number of available addresses in the 7-bit addressing mode is too small. Therefore, a new addressing mode (the 10-bit mode) was developed. The new addressing mode also supports the old one. Devices with 7-bit addresses can be connected with devices with 10-bit addresses on the same mode. In this mode, the first two bytes are dedicated for address and data direction. The format of the first byte is 11110xx; the last two bits of the first byte, combined with eight bits in the second byte form the 10-bit address.

### 2.3.5 Acknowledge (ACK) and not acknowledge (NACK)

Acknowledgement is obligatory in order to inform the transmitter that data has been successfully transmitted. Figure 2.5 illustrates the acknowledgement mechanism. The Master generates the acknowledge-related clock pulse and the transmitter releases the SDA line (HIGH) during the acknowledge clock pulse so that the receiver can take control of the SDA line. IF the receiver does not acknowledge, leaving the SDA line high, the transfer must be aborted. If acknowledged by pulling the SDA line low, the transmitter knows that data has been successfully received, so it keeps sending data to the receiver.

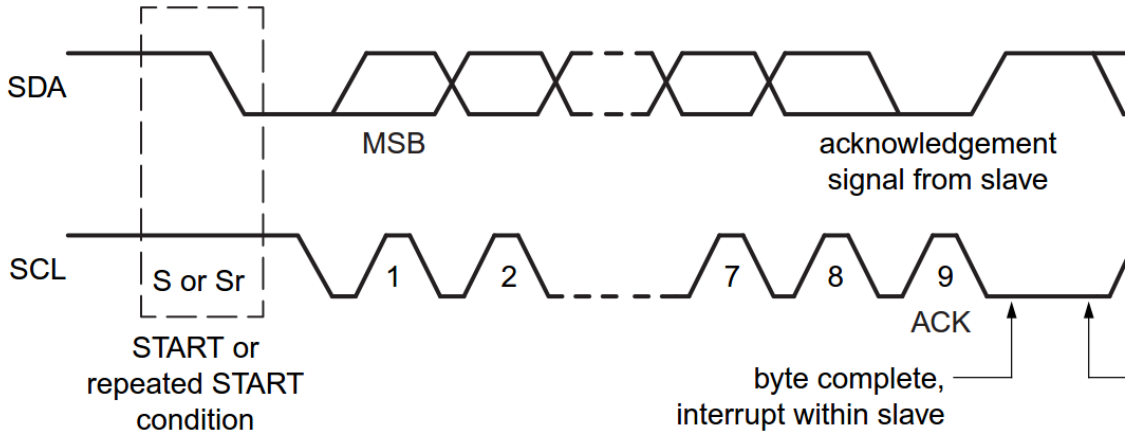


***Figure 2.5*** *Acknowledgement on $I^2C$ bus* [21].

The acknowledge takes place after every bytes. The acknowledge bit allows the receiver to signal the transmitter that the byte successfully received and another byte may be sent. The Master generates all clock pulses, including acknowledge of the ninth clock pulse. When SDA remains HIGH during this ninth clock pulse, this is defined as Not Acknowledge signal. The Master can then generate either a STOP condition to abort the transfer, or a repeated START condition to start a new transfer. There are five conditions that lead to the generation of a NACK [21]:

1. No receiver is present on the bus with the transmitted address so there is no device to respond with an ACKNOWLEDGE.
2. The receiver is unable to receive or transmit because it is performing some real-time function and is not ready to start communication with the Master.
3. During the transfer, the receiver gets data or commands that it does not understand.
4. During the transfer, the receiver cannot receive any more data bytes.
5. A master-receiver must signal the end of the transfer to the slave transmitter.

### 2.3.6 R/$\overline{W}$ bit

After the START condition (S), a Slave address is sent. This address is the first 7 bits, the eighth bit is a data direction bit (R/$\overline{W}$). If the direction bit is '0', it indicates a transmission (or WRITE). IF the bit is '1', it indicates a request for data (or READ). Figure 2.6 is a complete data transfer including the direction bit.

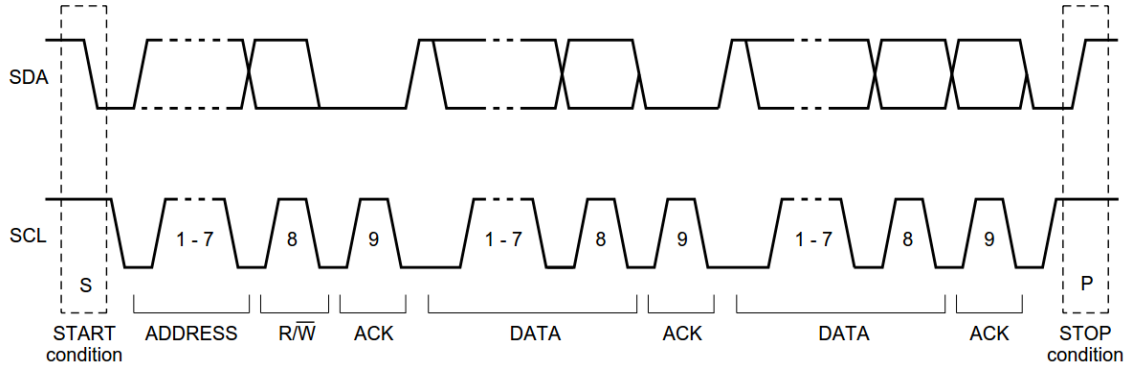


***Figure 2.6*** *A complete data transfer* [21].

# 3. IMPLEMENTATION ON FPGA

In this chapter, we describe the implement action of a Signal Generator, an $I^2C$ Adapter and an $I^2C$ Slave on DE1-SoC FPGA development board manufactured by Terasic. All the Signal Generator, $I^2C$ Adapter and $I^2C$ Slave are described in VHDL.

## 3.1 Block Diagrams

### 3.1.1 Writing data process

The writing data process includes three blocks, which are Signal Generator to Write, I2C Adapter and I2C Slave. All the $I^2C$ interface of blocks is connected by $I^2C$ buses.

Figure 3.1 illustrated data transmission from Signal Generator To Write through the I2C Adapter and terminated at I2C Slave. Signals generated by the Signal Generator To Write include address signals and data signals, that are serial signals. Address serial data signals is transmitted by the SDA bus sampling at frequency of SCL bus to I2C Adapter first. Data signals are transmitted to the I2C Adapter after address signals finish transmission. Address signals and data signals come in the I2C Adapter by one-directional port hps_sdi and come out the I2C Adapter by bi-directional port sda. At the end, I2C slave detects data signals from I2C Adapter and transfer it into 8 bits parallel signal at data_in port.

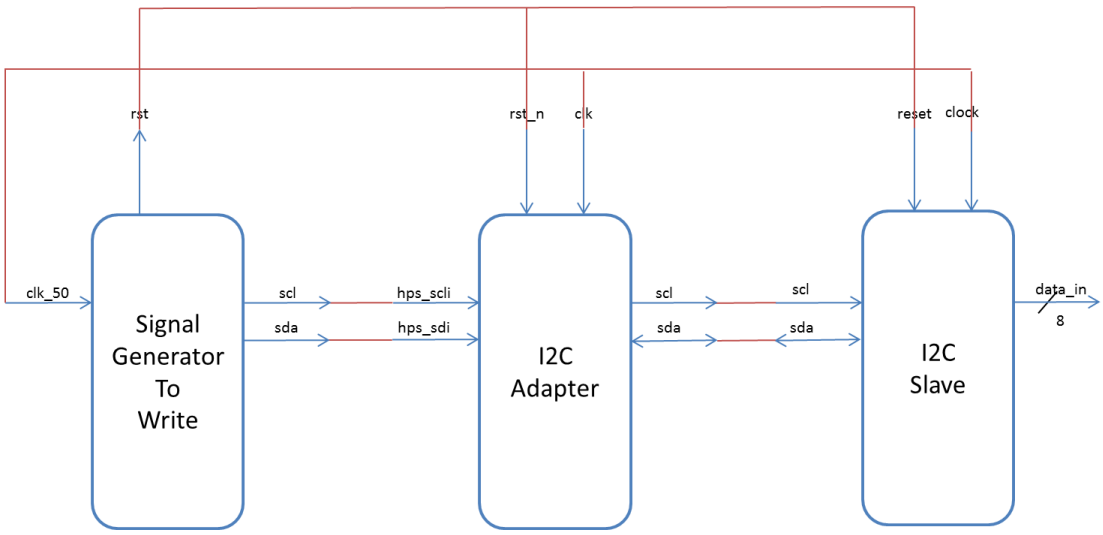


***Figure 3.1*** *Writing data process block diagram.*

### 3.1.2 Reading data process

The reading data process includes three blocks, which are Signal Generator to Read, I2C Adapter and I2C Slave. All the $I^2C$ interface of blocks is connected by $I^2C$ buses.

Figure 3.2 illustrated data transmission from Signal Generator To Read through the I2C Adapter and terminated at I2C Slave. Signals generated by the Signal Generator To Read include address signals and data signals, that are serial signals. Address serial signals is transmitted by the SDA bus sampling at frequency of SCL bus from port sda_ex of Signal Generator To Read to port hps_sdi of I2C Adapter first. Data serial signals are transmitted from port sda of Signal Generator To Read to port sda of I2C Adapter after address signals to finish transmission. Address signals come in the I2C Adapter by one-directional port hps_sdi and come out the I2C Adapter by one-directional port hps_sdo. Data signals come in the I2C Adapter by bi-directional port sda and come out the I2C Adapter by one-directional port hps_sdo. At the end, I2C slave detects data signals from I2C Adapter and transfer it into 8 bits parallel signal at data_in port.

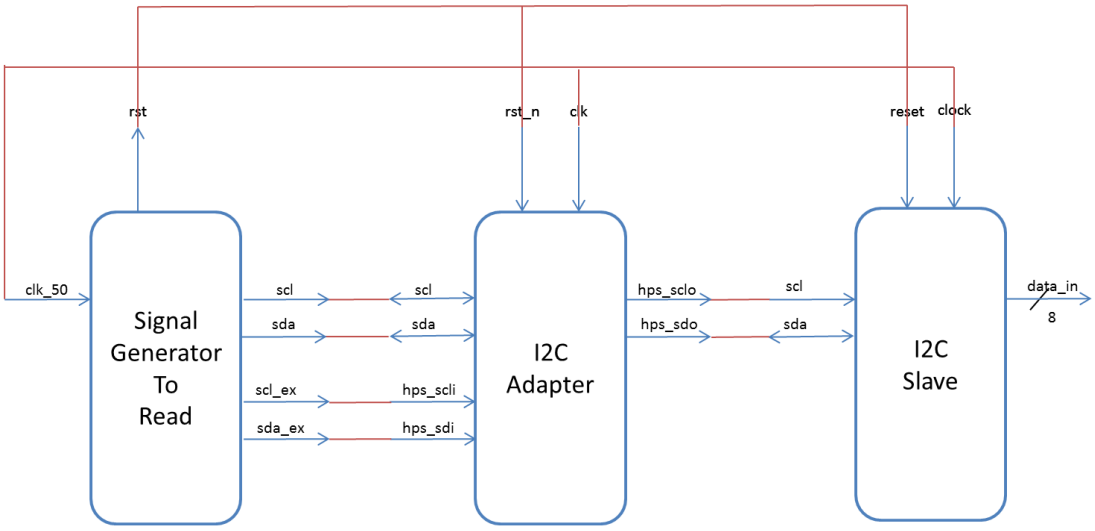


***Figure 3.2*** *Reading data process block diagram.*

## 3.2 Signal generator

### 3.2.1 Signal generator to write

Signal Generator To Write has one input port and three output ports as shown in the Figure 3.3. The input port clk_50 is the sampling clock signal and is assigned to the 50 MHz frequency pin of DE1-SoC FPGA development board. The clk_50 is operated at frequency 50 MHz inside the Signal Generator To Write block. Output port rst is reset signal to reset the data transmission process. Output port scl is sampling clock to data signals transmission. The last output port which is sda using to transmit address signals as data signals from Signal Generator To Write block to I2C Adapter block.

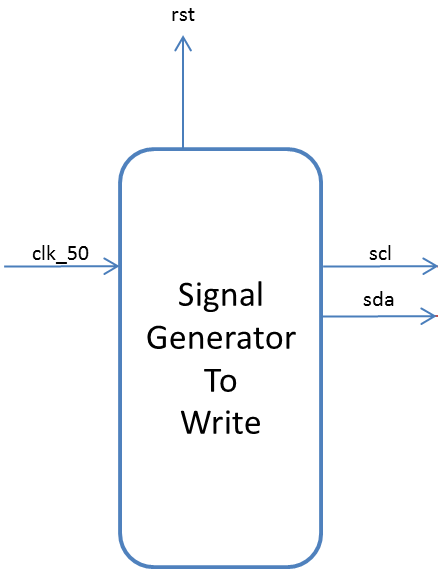


***Figure 3.3*** *Signal Generator To Write Block.*

Figure 3.4 illustrates the waveform of signals generated from Signal Generator To Write block. As you can see from Figure 3.4, the Signal Generator To Write Block generates an active low reset signal, which makes a falling edge at the end of phase 0. Reset signal is remained to be 0 for the rest of transmission process. The scl signal is sampling data clock signal and operating at 200 Hz. The sda signal has falling edge, which is a transition from high to low while scl is high at phase 2; it makes a start condition following the start condition definition in $I^2C$ protocol. The sda then transmits a serial in 8 bits binary number 00000000 of address from phase 3 to 10 with the last bit is 0 for write decision process. The sda signal has value 0 at phase 11 standing for the acknowledge bit that correct address transmission. Serial 8 bits binary number 10101010 of data is transmitted from phase 12 to 19 with the last bit is 0 for the next byte writing decision process. The sda signal has value 0 at phase 20 standing for the acknowledge bit that complete data transmission. The sda signal has rising edge, which is a transition from low to high while scl is high at phase 21; it makes a stop condition following the stop condition definition in $I^2C$ protocol.

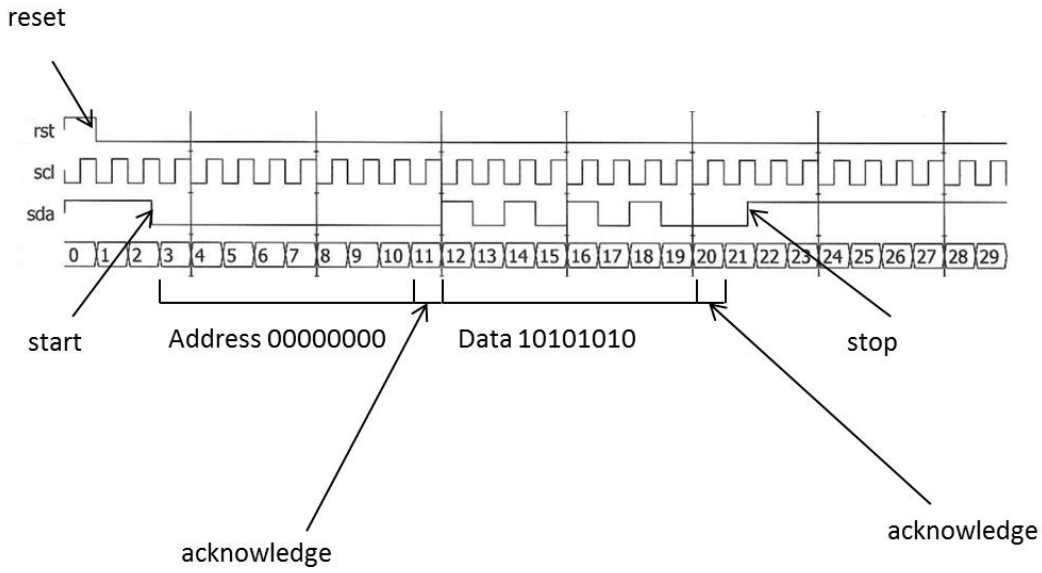


***Figure 3.4*** *Address and data signals waveform generated by Signal Generator To Write.*

### 3.2.2 Signal generator to read

Signal Generator To Write has one input port and five output ports as shown in the Figure 3.5. The input port clk_50 is the sampling clock signal and is assigned to the 50 MHz frequency pin of DE1-SoC FPGA development board. The clk_50 is operated at 50 MHz frequency inside the Signal Generator To Write block. Output port rst is reset signal aiming to reset the data transmission process. Output port scl is sampling clock to data signals transmission. Output port sda is used to transmit data signals from Signal Generator To Write block to I2C Adapter block. Output port scl_ex is sampling clock to address signals transmission. Output port sda_ex is used to transmit address signals from Signal Generator To Write block to I2C Adapter block.

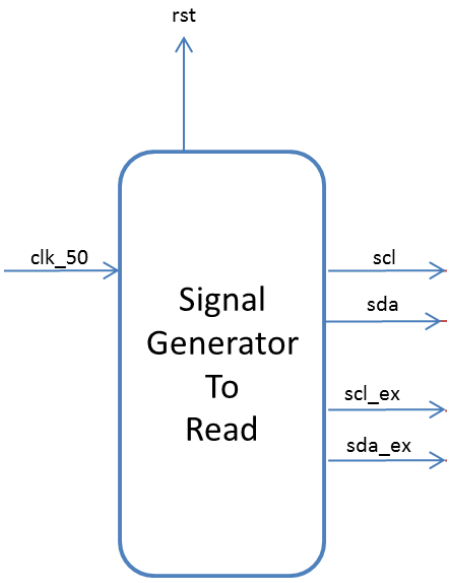


***Figure 3.5*** *Signal Generator To Read Block.*

Figure 3.6 illustrates the waveform of signals generated from Signal Generator To Write block. As you can see from the figure, the Signal Generator To Write Block generates an active low reset signal which makes a falling edge at the end of phase 0. The reset signal is remained to be 0 for the rest of transmission process. The scl and scl_ex signals are sampling data clock signal and sampling address clock signal respectively, they are both operated at 200 Hz. The sda_ex signal has falling edge which is a transition from high to low while scl_ex is high at phase 2; it makes a start condition following the start condition definition in $I^2C$ protocol. The sda_ex is then transmit a serial 8 bits binary number 00000001 of address from phase 3 to 10 with the last bit is 1 for read decision process. The sda signal has value 0 at phase 11 standing for the acknowledge bit that correct address transmission. Serial 8 bits binary number 10101011 of data is transmitted from phase 12 to 19 with the last bit is 1 for next byte reading decision process. The sda_ex signal has value 0 at phase 20 standing for the acknowledge bit that correct data transmission. The sda_ex signal has rising edge which is a transition from low to high while scl_ex is high at phase 21; it makes a stop condition following the stop condition definition in $I^2C$ protocol.

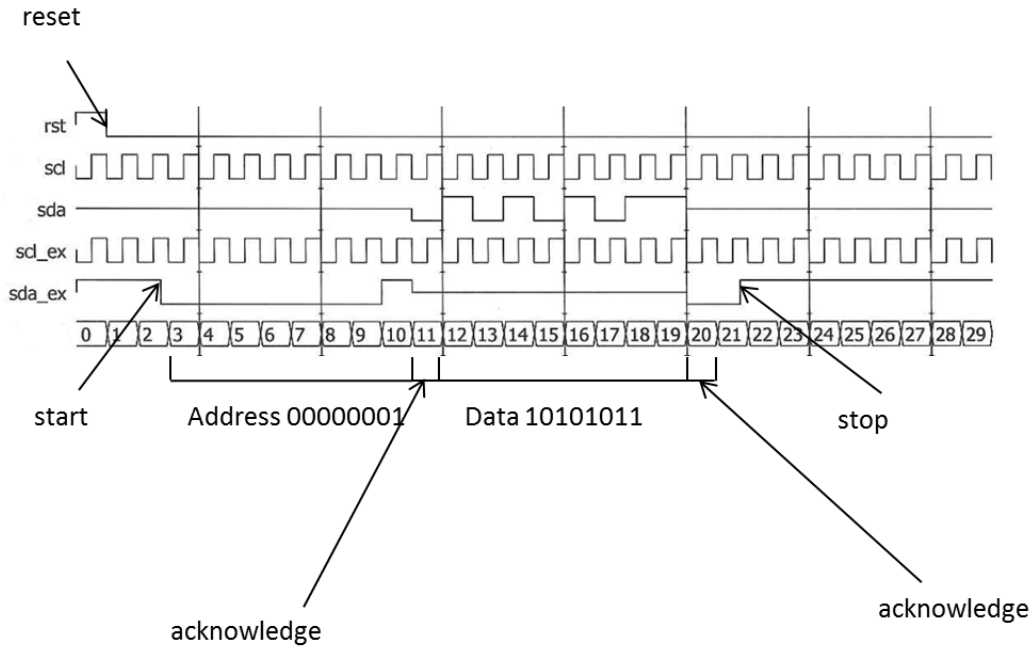


***Figure 3.6*** *Address and data signals waveform generated by Signal Generator To Read.*

## 3.3 I²C Adapter

### 3.3.1 I²C Adapter Block

The I2C Adapter has four input ports, two output ports and two inout ports as shown in the Figure 3.7. The input port clk is the sampling clock signal and is assigned to the 50 MHz frequency pin of DE1-SoC FPGA development board. The clk is operated at 50 MHz frequency inside the I2C Adapter block. Input port rst_n is reset signal aiming to reset the data transmission process inside I2C Adapter; it is an active low reset. Inout port scl is output in writing data process, it is used for sampling clock to address signals and data signals as well. Inout port scl is input in reading data process, it is used for sampling clock to data signals. Inout port sda is output in writing data process, it is used to transmit address signals and data signals as well. Inout sda is input in reading data process and it is used to transmit data signals. Input port hps_scli is sampling clock to address signals as data signals in writing data transmission and sampling clock to address signals in reading data transmission. Input port hps_sdi is used to transmit address signals as data signals in writing data transmission and transmit address signals in reading data transmission. Output port hps_sclo is sampling clock to data signals in reading data transmission. Output port hps_sdo is used to transmit data signals in reading data transmission. Table 3.1 describes the I2C Adapter ports.

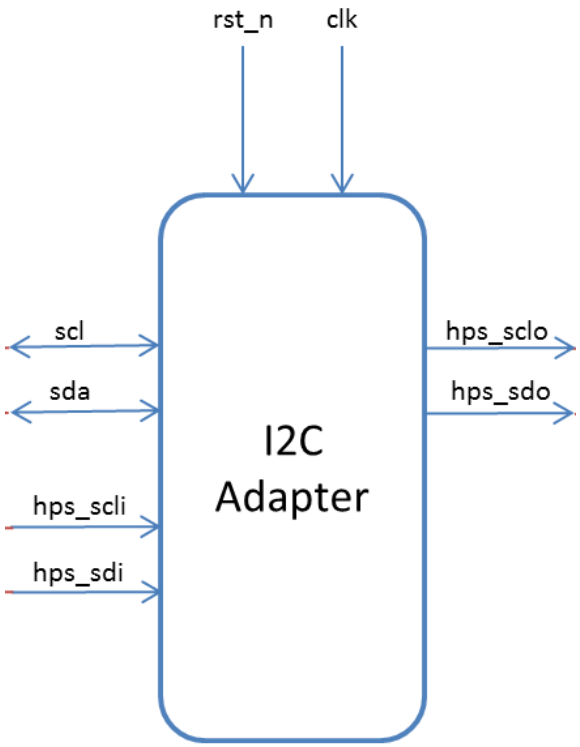


***Figure 3.7*** *I2C Adapter Block.*

***Table 3.1*** *I2C Adapter ports description.*

| Port | Width | Mode | Data Type | Description |
|---|---|---|---|---|
| clk | 1 | in | std_logic | system clock |
| rst_n | 1 | in | std_logic | asynchronous active low reset |
| scl | 1 | inout | std_logic | serial clock line of I2C bus |
| sda | 1 | inout | std_logic | serial data line of I2C bus |
| hps_scli | 1 | in | std_logic | serial clock input line of I2C bus |
| hps_sdi | 1 | in | std_logic | serial data line input of I2C bus |
| hps_sclo | 1 | out | std_logic | serial clock output line of I2C bus |
| hps_sdo | 1 | out | std_logic | serial data output line of I2C bus |

### 3.3.2 Finite State Machine

The $I^2C$ Adapter uses the state machine depicted in Figure 3.8 to implement the I2C bus protocol. Upon start-up, the component immediately enters the idle state. It follows the condition for each state as described and stops when finishing data transmission with the stop condition. The explanation for each state is described more clearly in part 3.3.3 how adapter work.

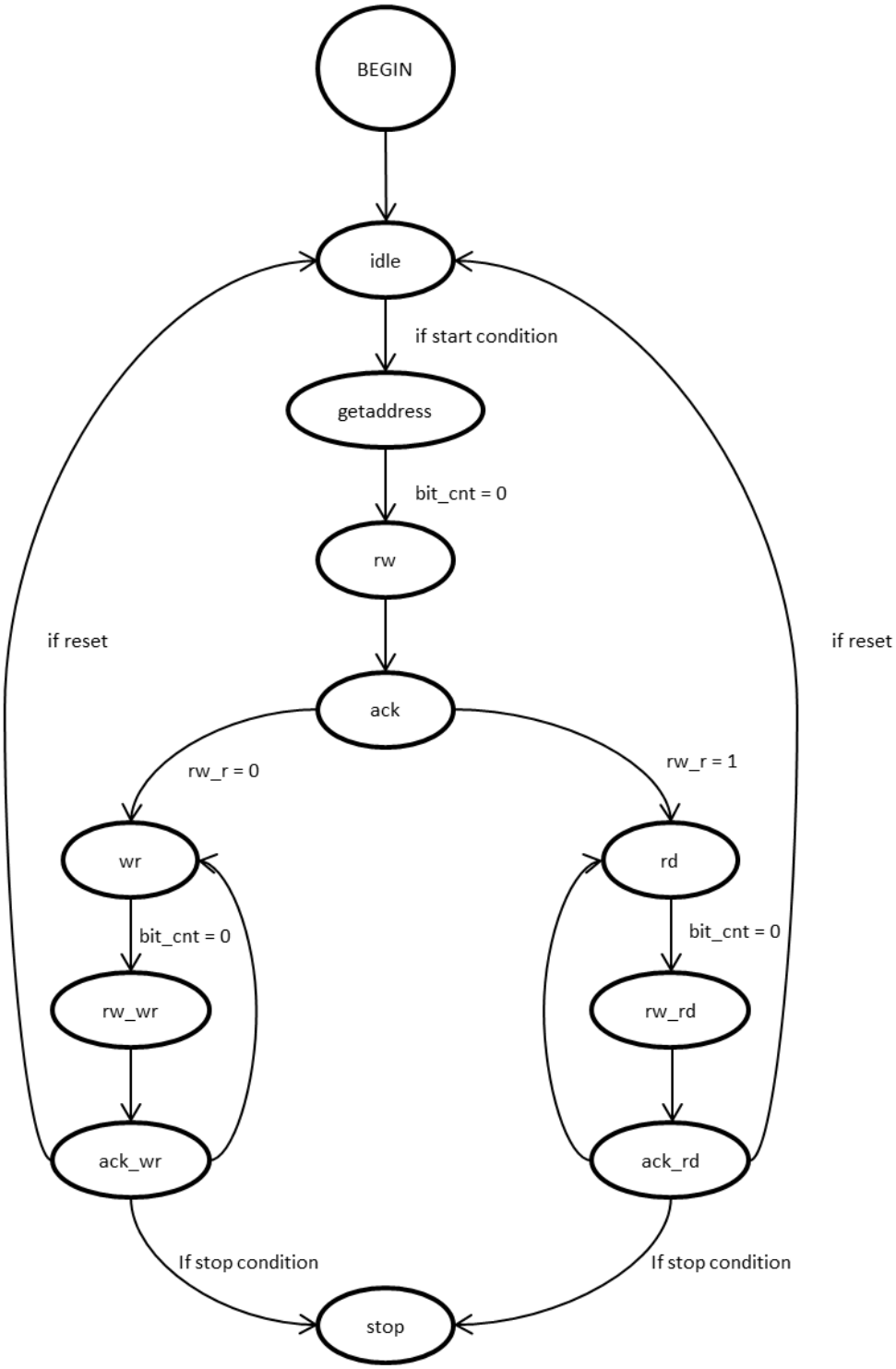


***Figure 3.8*** *Finite state machine of I2C Adapter block.*

### 3.3.3 How I²C Adapter work

As the description for I2C Finite State Machine in the Figure 3.8, it is easy to follow by dividing it into separated processes as Start Process and Stop Process, Reset Process, Getting Address Process, Writing Data Process and Reading Data Process. Each process can be more understandable by looking the description of the code for it.

**Start Process and Stop Process**

Program 3.1 shows how the Start Process and Stop Process work inside the I2C Adapter. At the Start Process, hps_sda_r keeps the previous value of data signal and hps_sdi is the current data signal. When there is a transition in data value from high to low as hps_sda_r = 1 and hps_sdi = 0 while the sampling clock signal is high as hps_scli = 1, the start condition happens and marking as start_edge = 1. Without falling edge of data signal during high period of sampling clock signal, start_edge = 0. At the Stop Process, hps_sda_r keeping previous value of data signal and hps_sdi is the current data signal. When there is a transition in data value from low to high as hps_sda_r = 0 and hps_sdi = 1 while the sampling clock signal is high as hps_scli = 1, the stop condition happens and marking as stop_edge = 1. Without rising edge of data signal during high period of sampling clock signal, stop_edge = 0.

```
53  start_process: PROCESS (hps_scli, hps_scl_r, hps_sdi)
54  BEGIN
55     IF (hps_scli = '1' and hps_sda_r='1' and hps_sdi ='0')   THEN  --start condition, falling edge of hps_sdi
56        start_edge <= '1';                                          --make a tick for start_edge signal
57     ELSE
58        start_edge <= '0';                                          --start_edge = 0 without falling edge of hps_sdi
59     END IF;
60
61  END PROCESS;
62
63
64  stop_process: PROCESS (hps_scli, hps_scl_r, hps_sdi)
65  BEGIN
66     IF (hps_scli = '1' and hps_sda_r='0' and hps_sdi ='1')   THEN  --stop condition, rising edge of hps_sdi
67        stop_edge <= '1';                                           --make a tick for stop_edge signal
68     ELSE
69        stop_edge <= '0';                                           --stop_edge = 0 without falling edge of hps_sdi
70     END IF;
```

***Program 3.1*** *Start Process and Stop Process Program.*

**Reset Process**

Program 3.2 shows how the Reset Process works inside the I2C Adapter. At the Reset Process, rst_n stands for active low reset signal and it is assigned to be 1 at the beginning. When rst_n = 1, I2C Adapter is in idle state which keeping the default values for output signals as scl = 0, sda= 0, hps_sclo = 0 and hps_sdo = 0. The read/write bit register rw_r, the acknowledge register from hard processor core ack_hps and the acknowledge register from slave ack_cypress are also kept at 0 during idle state. When rst_n = 0, output signals hps_sclo, scl, sda as well as register hps_scl_r hps_sda_r are assigned to input signals hps_scli, hps_scli, hps_sdi, hps_scli and hps_sdi respectively. There is always a waiting for start condition or stop condition when rst_n = 0. If the start condition happens, I2C Adapter turns to getting address state as start_edge = 1. If the stop condition happens, I2C Adapter turns to idle state as stop_edge = 1.

```
76      IF (rst_n = '1') THEN                       --reset asserted
77
78         state       <= idle;                     --state idle waiting for start condition
79         bit_cnt     <= 8;                        --keep value of counter at 8 bits for
80                                                  --trasmitting address signals and data signals
81         rw_r        <= '0';                      --read/write register asserted
82
83         scl         <= '0';                      --serial clock to/from slave asserted
84         sda         <= '0';                      --serial data to/from slave asserted
85
86         hps_sclo    <= '0';                      --serial clock output to hps asserted
87         hps_sdo     <= '0';                      --serial data output to hps asserted
88
89         hps_scl_r   <= '0';                      --serial clock register inside I2C Adapter asserted
90         hps_sda_r   <= '0';                      --serial data register inside I2C Adatper asserted
91
92         ack_cypress <= '0';                      --acknowledge from slave register asserted
93         ack_hps     <= '0';                      --acknowledge from hps register asserted
94
95      ELSIF (clk'EVENT and clk = '1') THEN        --reset is actived
96
97         hps_sclo    <= hps_scli;                 --assert serial clock output
98                                                  --to be the same as serial clock input
99         scl         <= hps_scli;                 --assert serial clock to/from slave
100                                                 --to be same as serial clock input
101        sda         <= hps_sdi;                  --assert serial data to/from slave
102                                                 --to be same as data input
103        hps_scl_r   <= hps_scli;                 --hps clock register is asserted as serial clock input
104        hps_sda_r   <= hps_sdi;                  --hps data register is asserted as data input
105
106        IF start_edge = '1' THEN                 --start edge = 1 when start condition
107           state <= getaddress;                  --transaction start with getting the address
108
109        ELSIF stop_edge = '1' THEN               --stop edge = 1 when stop condition
110           state <= idle;                        --transaction complete, go to next state
```

***Program 3.2*** *Reset Process Program.*

### Getting Address Process

Program 3.3 shows how the Getting Address Process works inside the I2C Adapter. At the Getting Address Process, scl_edge stands for the synchronous clock edge and is used to synchronizing address signals transmission as data signals transmission. In the idle state, I2C turns to getaddress in the next state when start condition happens. Bit counter bit_cnt is counted down from 8 in idle state to 7 in getaddress state. In the getaddress state, bit counter is continued to count down until it equals 0 to finish address bits signal transmission. When bit_cnt = 0, I2C Adapter get into read/write selection rw state. The read/write selection bit is asserted by the last bit of address bits, the eighth bit; then I2C Adapter turns to acknowledge for address transmission state ack. At acknowledge for address transmission state ack, acknowledge bit is taken at the middle of ninth clock and it is the acknowledge bit from the slave. Bit counter is reset to 7 after the address transaction finishing. If the acknowledge from slave is 0, I2C Adapter continue to next process Writing Data Process or Reading Data Process, which is decided by the read/write selection bit register rw_r. If the acknowledge from slave is 1, it means wrong address, I2C Adapter turns to take the address again.

```
117             WHEN idle =>                        --idle state waiting for start condition
118                IF (scl_edge = '1') THEN         --synchronous clock edge
119                   IF start_edge = '1' THEN      --start_edge = 1 when start condition
120                      bit_cnt  <= bit_cnt - 1;   --bit_cnt = 7 to go to getaddress state
121                      state    <= getaddress;    --getting bit of address
122                   END IF;
123                END IF;
124
125
126             WHEN getaddress =>                  --getaddress state getting 8 bits address
127                IF (bit_cnt = 0) THEN            --address bits transmission finished
128                      state   <= rw;             --go to slave acknowledge
129                ELSE                             --next clock cycle
130                   IF (scl_edge = '1') THEN      --synchronous clock edge
131                      bit_cnt <= bit_cnt - 1;    --keep track of counting down number of bits receiving
132                   END IF;
133                END IF;
134
135
136             WHEN rw =>                          --read/write selection bit
137                sda <= 'Z';
138                rw_r <= hps_sda_r;               --read/write asserted by last bit of address
139                IF (scl_edge = '1') THEN         --synchronous clock edge
140                   bit_cnt  <= bit_cnt - 1;      --keep track of counting down number of bits receiving
141                   state    <= ack;              --next state is acknowledge when bit_cnt = -1
142                END IF;
143
144
145             WHEN ack =>                         --acknowledge for address transmission
146                sda <= 'Z';
147                IF (bit_cnt2 = 12) THEN          --acknowledge bit is taken at the middle of ninth clock
148                   ack_cypress <= sda;           --acknowledge bit from slave
149                END IF;
150                IF (scl_edge = '1') THEN         --synchronous clock edge
151                   bit_cnt     <= 7;             --bit counter reset after address transaction
152                   IF (ack_cypress = '0') THEN   --acknowledge following i2c protocol
153                      IF (rw_r = '0') THEN       --read/write selection bit
154                         state <= wr;            --go to write state if read/write selection bit = 0
155                      ELSE
156                         state <= rd;            --go to read state if read/write selection bit = 1
157                      END IF;
158                   ELSE                          --notacknowledge if ack_cypress = 1
159                      state    <= getaddress;    --if notacknowlege, go to getting address again
160                   END IF;
161                END IF;
```

***Program 3.3*** *Getting Address Process Program.*

### Writing Data Process

Program 3.4 shows how the Writing Data Process works inside the I2C Adapter. At the Writing Data Process, scl_edge stands for the synchronous clock edge and is used to synchronizing address signals transmission as data signals transmission. In the write state wr, bit counter is counted down at the sampling synchronous clock edge until it equals 0 to finish writing data bits signal transmission. When bit_cnt = 0, I2C Adapter gets into read/write selection rw_wr state. The read/write selection bit is asserted by the last bit of data bits, the eighth bit; then I2C Adapter turns to acknowledge for writing data transmission state ack_wr. At acknowledge for writing data transmission state ack_wr, acknowledge bit is taken at the middle of ninth clock and it is the acknowledge bit from the slave. Bit counter is reset to 7 after the writing data transaction finishing. If the acknowledge from slave is 0, I2C Adapter continue to next process Writing Data Process wr or Reading Data Process rd, which is decided by the read/write selection bit register rw_r. If the acknowledge from slave is 1, it means wrong data, I2C Adapter turns to take the address again and start a new whole process.

```
164             WHEN wr =>                          --write byte state getting 8 bits data
165                sda <= hps_sdi;                  --output serial data asserted by input serial data
166                IF (bit_cnt = 0) THEN            --reset counter when data bits transmission finished
167                   state   <= rw_wr;             --go to slave acknowledge
168                ELSE                             --next clock cycle
169                   IF (scl_edge = '1') THEN      --synchronous clock edge
170                      bit_cnt <= bit_cnt - 1;    --keep track of counting down number of bits receiving
171                   END IF;
172                END IF;
173
174
175             WHEN rw_wr =>                       --read/write selection bit
176                rw_r <= hps_sda_r;               --read/write asserted by last bit of address
177                IF (scl_edge = '1') THEN         --synchronous clock edge
178                   bit_cnt <= bit_cnt - 1;       --keep track of counting down number of bits receiving
179                   state <= ack_wr;              --next state is acknowledge when bit_cnt = -1
180                END IF;
181
182
183             WHEN ack_wr =>                      --acknowledge for byte transfer
184             sda <= 'Z';
185                IF (bit_cnt2 = 12) THEN          --acknowledge bit is taken at the middle of ninth clock
186                   ack_cypress <= sda;           --acknowledge bit from slave
187                END IF;
188                IF (scl_edge = '1') THEN         --synchronous clock edge
189                   bit_cnt <= 7;                 --bit counter reset after data transaction
190                   IF (ack_cypress = '0') THEN   --acknowledge following i2c protocol
191                      IF (rw_r = '0') THEN       --read/write selection bit
192                         state <= wr;            --continue to write if read/write selection bit = 0
193                      ELSE
194                         state <= rd;            --if read/write selection bit = 1, go to read state
195                      END IF;
196                   ELSE                          --notacknowledge if ack_cypress = 1
197                      state <= getaddress;       --if notacknowlege, go to getting address again
198                   END IF;
199                END IF;
```

***Program 3.4*** *Writing Data Process Program.*

**Reading Data Process**

Program 3.5 shows how the Reading Data Process works inside the I2C Adapter. At the Reading Data Process, scl_edge stands for the synchronous clock edge and is used to synchronizing address signals transmission as data signals transmission. In the read state rd, bit counter is counted down at the sampling synchronous clock edge until it equals 0 to finish reading data bits signal transmission. When bit_cnt = 0, I2C Adapter gets into read/write selection rw_rd state. The read/write selection bit is asserted by the last bit of data bits, the eighth bit; then I2C Adapter turns to acknowledge for reading data transmission state ack_rd. At acknowledge for reading data transmission state ack_rd, acknowledge bit is taken at the middle of ninth clock and it is the acknowledge bit from the hard processor core. Bit counter is reset to 7 after the reading data transaction finishing. If the acknowledge from slave is 0, I2C Adapter continue to next process Reading Data Process rd or Writing Data Process wr, which is decided by the read/write selection bit register rw_r. If the acknowledge from slave is 1, it means wrong data, I2C Adapter turns to take the address again and start a new whole process.

```
202             WHEN rd =>                          --read byte state getting 8 bits data
203                sda <= 'Z';
204                hps_sdo  <= sda;                 --output serial data asserted by input serial data
205                IF (bit_cnt = 0) THEN            --reset counter when data bits transmission finished
206                   state   <= rw_rd;             --go to hps acknowledge
207                ELSE                             --next clock cycle
208                   IF (scl_edge = '1') THEN      --synchronous clock edge
209                      bit_cnt <= bit_cnt - 1;    --keep track of counting down number of bits receiving
210                   END IF;
211                END IF;
212
213
214             WHEN rw_rd =>                       --read/write selection bit
215                sda <= 'Z';
216                rw_r <= sda;                     --read/write asserted by last bit of address
217                IF (scl_edge = '1') THEN         --synchronous clock edge
218                   bit_cnt <= bit_cnt - 1;       --keep track of counting down number of bits receiving
219                   state <= ack_rd;              --next state is acknowledge when bit_cnt = -1
220                END IF;
221
222
223             WHEN ack_rd =>                      --acknowledge for byte transfer
224                sda <= 'Z';
225                IF (bit_cnt2 = 12) THEN          --acknowledge bit is taken at the middle of ninth clock
226                   ack_hps <= hps_sdi;           --acknowledge bit from hps
227                END IF;
228                IF (scl_edge = '1') THEN         --synchronous clock edge
229                   bit_cnt <= 7;                 --bit counter reset after data transaction
230                   IF (ack_hps = '0') THEN       --acknowledge following i2c protocol
231                      IF (rw_r = '0') THEN       --read/write selection bit
232                         state <= wr;            --if read/write selection bit = 0, go to write state
233                      ELSE
234                         state <= rd;            --continue to write if read/write selection bit = 1
235                      END IF;
236                   ELSE                          --notacknowledge if ack_cypress = 1
237                      state <= getaddress;       --if notacknowlege, go to getting address again
238                   END IF;
239                END IF;
```

***Program 3.5*** *Reading Data Process Program.*

## 3.4 I²C Slave

The I2C Slave code is taken from OpenCores respecting the copyright. The I2C Slave has four input ports, seven output ports and one inout port as shown in the Figure 3.9. The input port clock is the sampling clock signal and is assigned to the 50 MHz frequency pin of DE1-SoC FPGA development board. The clock is operated at frequency 50 MHz inside the I2C Slave block. Input port reset is reset signal aiming to reset the data transmission process at I2C Slave; it is active low reset. Input port scl is used for sampling clock to address signals and data signals as well. Inout port sda is input to read address signal as data signal from I2C Adapter. Output port start_detected is true if start condition is detected at I2C Slave. Output port transfer_started is true if a valid address was received and acknowledged. Output port read_mode is true if master wants to read device. Output port stop_detected is true if stop condition is detected ad I2C Slave. Input port data_out is used to send byte to master. Output port data_out_requested is true if data write has to be filled to send the next byte. Output port data_in is the last received byte from master. Output data_in_valid is true if the master sent a byte.

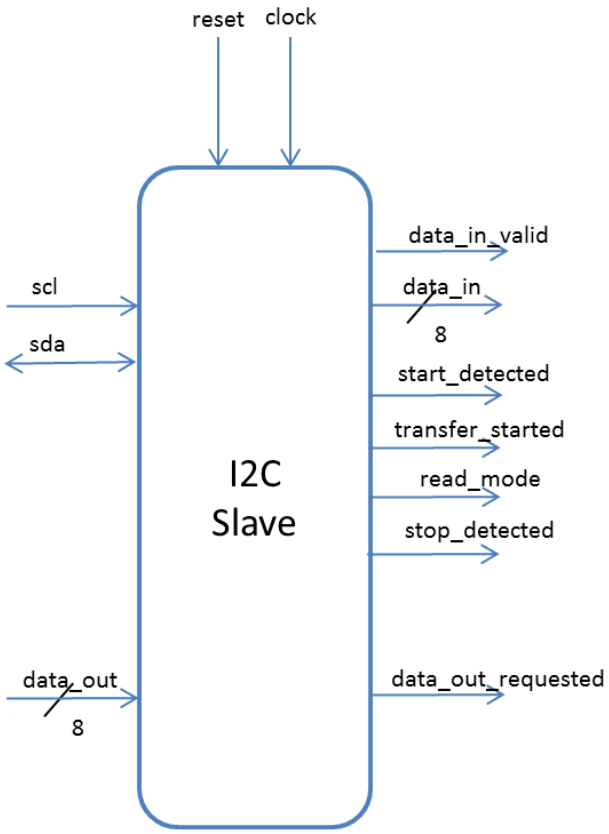


***Figure 3.9*** *I2C Slave Block.*

The I2C Slave includes three main processes, which are Control Process, Read Process and Write Process. The Control Process is used to process for receiving and sending bytes to I2C bus with the proper acknowledge generators and detection. The Read Process provides 2 functions which are: if read_byte is set to true, then 8 bits are read from the I2C bus into input_shift; if read_ack is set to true, then one bit will be read from the I2C bus for the acknowledge from master into input_shift(0). The Write Process provides 2 functions which are: if write_byte is set to true then 8 bits are written from data_out to the I2C bus; if write_ack is set to true, then one 0 bit will be written to the I2C bus for the acknowledge from slave. Start or stop bit detection resets the state machine for both Read Process and Write Process. In this project, in order to display the signal from Signal Generator To Write and Signal Generator To Read through I2C Adapter on the LEDs of DE1-SoC FPGA development board, we need to use only Control Process and Read Process. It also means we do not need to use input port data_out and output port data_out_requested in this project.

## 3.5 Top Level Design

The Top Level Design is divided into two projects, which are one project for writing data and one for reading data through the I2C Adapter. Figure 3.10 illustrates the Write Data Top Design Block, which includes three other clocks inside: Signal Generator To Write, I2C Adapter and I2C Slave. The connection of blocks inside Write Data Top Design is as Figure 3.1. The input port clk_50 is the sampling clock signal and is assigned to the 50 MHz frequency pin of DE1-SoC FPGA development board. The clk_50 is operated at frequency 50 MHz inside the Write Data Top Design Block. The output port output is 8 bits width and assigned to LEDs pin of DE1-SoC FPGA development board.


***Figure 3.10*** *Write Data Top Design Block.*

Figure 3.11 illustrates the Read Data Top Design Block, which includes three other clocks inside: Signal Generator To Read, I2C Adapter and I2C Slave. The connection of blocks inside Read Data Top Design is as Figure 3.2. The input port clk_50 is the sampling clock signal and is assigned to the 50 MHz frequency pin of DE1-SoC FPGA development board. The clk_50 is operated at frequency 50 MHz inside the Write Data Top Design Block. The output port output is 8 bits width and assigned to LEDs pin of DE1-SoC FPGA development board.

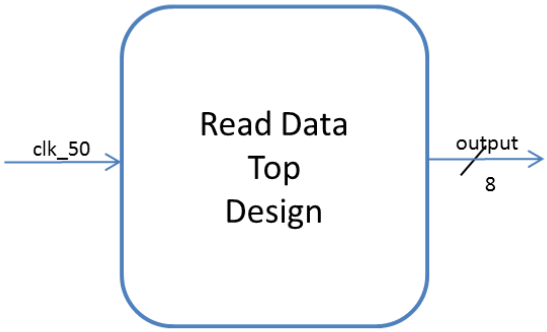


***Figure 3.11*** *Read Data Top Design block.*

# 4. VERIFICATION AND RESULT

Chapter 3 shows how to implement the Signal Generator, $I^2C$ Adapter and $I^2C$ Slave. Chapter 4 presents verification and result for those blocks by simulation in Modelsim SE 10.2c. The implementations on board are verified by the display of LEDs on DE1-SoC FPGA development board.

## 4.1 Writing Data Process Verification

### 4.1.1 Verification of Signal Generator To Write in Writing Data Process

As described in 3.2.1, the Signal Generator To Write has to generate the sequence of signals making reset, start condition, stop condition, address transmission and data transmission following $I^2C$ protocol. Figure 4.1 illustrates reset, start condition and stop condition waveforms of signals generated by Signal Generator To Write in a testbench. As can be seen from Figure 4.1, Signal Generator To Write generates an active low reset at the end of phase 0 when signal rst changes from high to low and remains for the rest of transmission. Start condition happens at phase 2 when the sda signal has a falling edge while the scl signal is high. Stop condition happens at phase 21 when the sda signal has a rising edge while the scl signal is high.

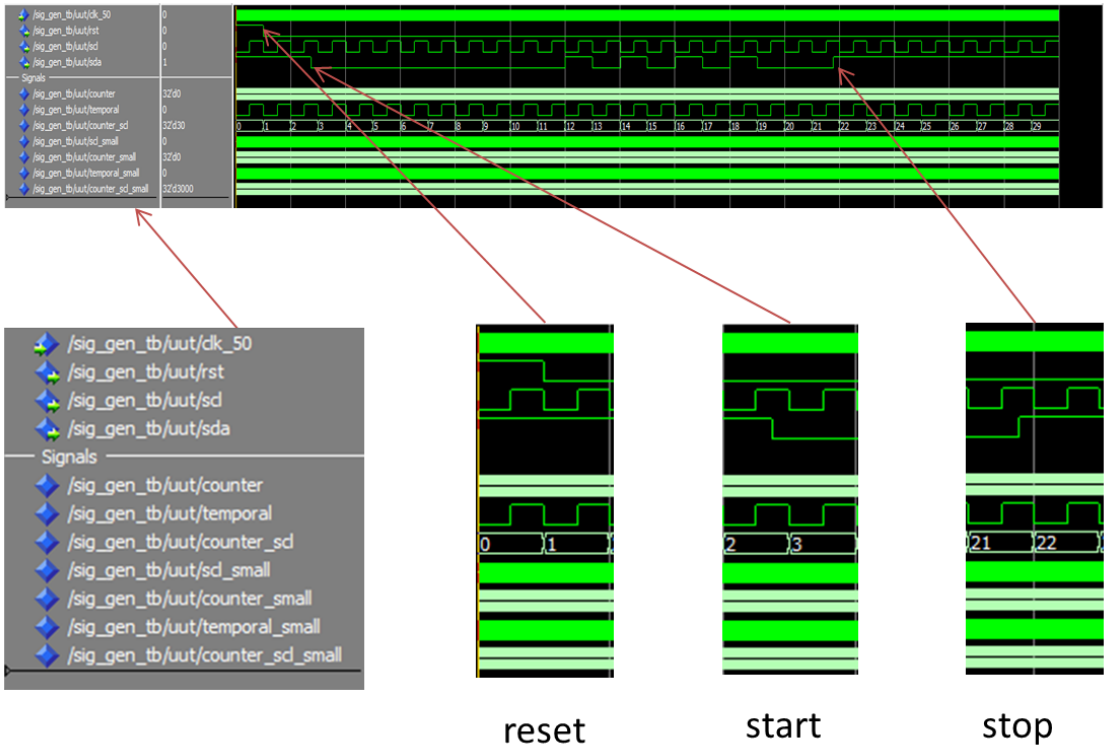


***Figure 4.1*** *Modelsim testbench for reset, start and stop of Signal Generator To Write in Writing Data Process.*

Figure 4.2 illustrates the address transmission and the data transmission waveforms of signals generated by Signal Generator To Write in a testbench. As can be seen from Figure 4.2, address bits are 0000 0000 with the last bit for read/write selection is generated by Signal Generator To Write from phase 3 to 10. Bit value 0 at phase 11 is used for acknowledge. Data bits are 1010 1010 with last bit for read/write selection is generated by Signal Generator To Write from phase 12 to phase 19. Bit value 0 at phase 20 is used for acknowledge.

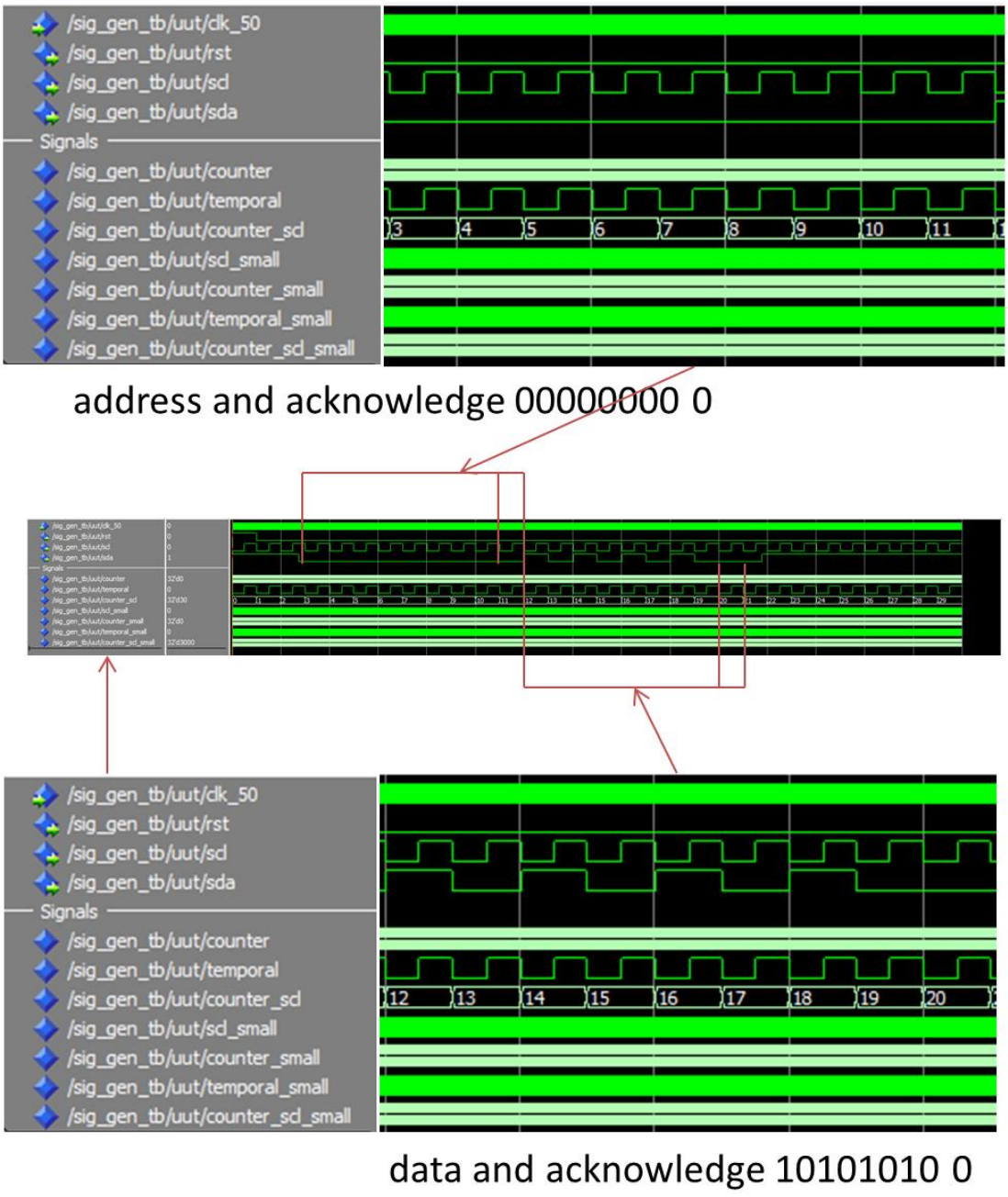


***Figure 4.2*** *Modelsim testbench for address and data transmission of Signal Generator To Write in Writing Data Process.*

Figure 4.3 and Figure 4.4 show Signal Generator To Write Block and components inside it respectively in Writing Data Process. Signal Generator To Write Block after Compile Design process by Quartus II and programing to FPGA device can be seen from RTL Viewer.

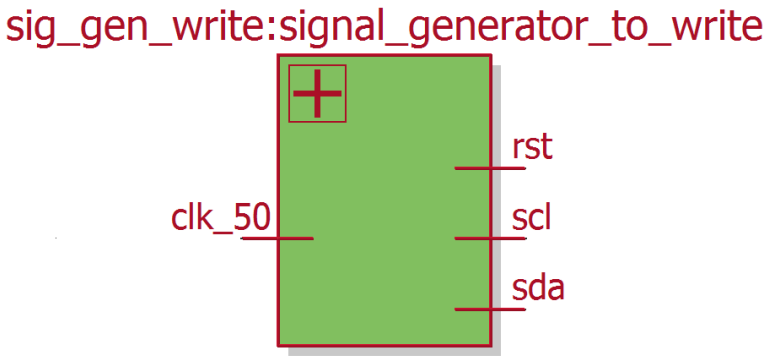


***Figure 4.3*** *Signal Generator To Write Block on RTL Viewer of Quartus II in Writing Data Process.*

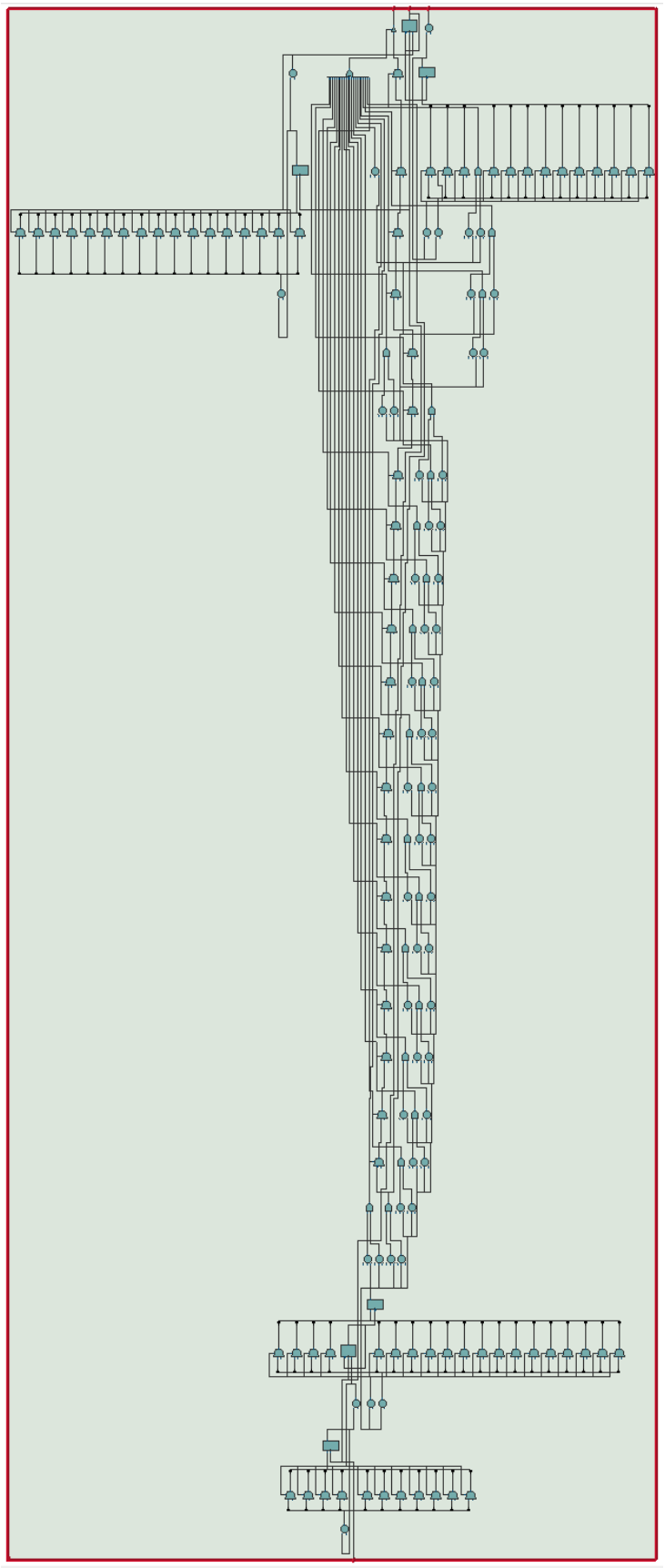

***Figure 4.4*** *Inside Signal Generator To Write Block on RTL Viewer of Quartus II in Writing Data Process.*

### 4.1.2 Verification of I²C Adapter in Writing Data Process

As described in 3.3, the I2C Adapter is used to transmit data from the hard processor core ARM Cortex-A9 to a slave in Writing Data Process following I²C protocol. Figure 4.5 illustrates reset, start condition and stop condition detection waveforms of signal transmission through I2C Adapter in Writing Data Process in a testbench. As can be seen from Figure 4.5, I2C Adapter can detect the active low reset, start condition as falling edge of hps_sdi during high hps_scli period and stop condition as rising edge of hps_sdi during high hps_scli period.

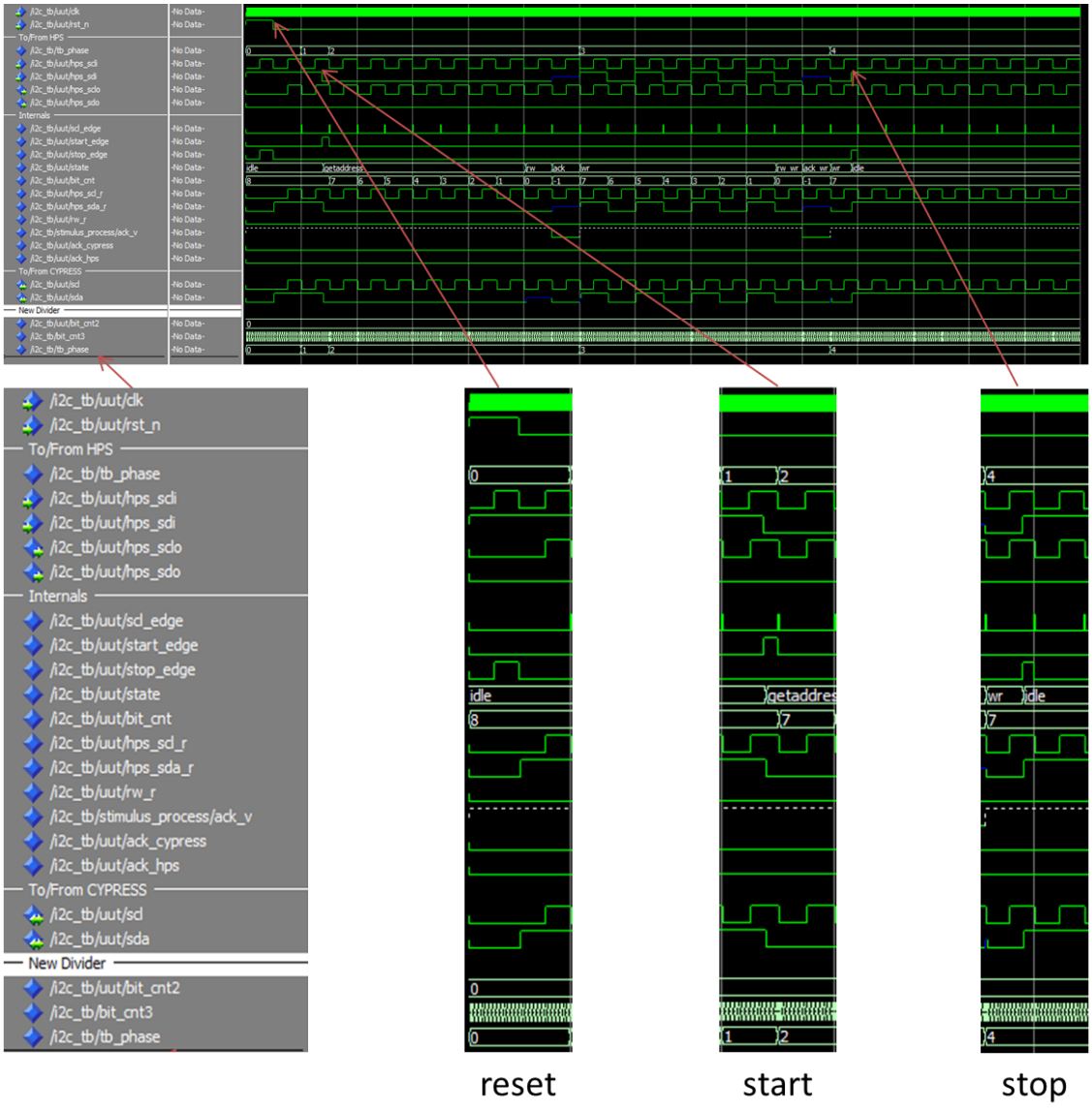


***Figure 4.5*** *Modelsim testbench for reset, start and stop of I2C Adapter in Writing Data Process.*

Figure 4.6 illustrates the address transmission waveforms of signals through I2C Adapter in a testbench. As can be seen from Figure 4.6, address bits are 0000 0000 with the first seven bits in getaddress state and the last bit for read/write selection in rw state. Address bits come in from input port hps_sdi of I2C Adapter. Inout port sda, which is output in Writing Data Process, shows exact value of address bits following input port hps_sdi. Acknowledge from slave as from port sda in Writing Data Process shows in ack state.

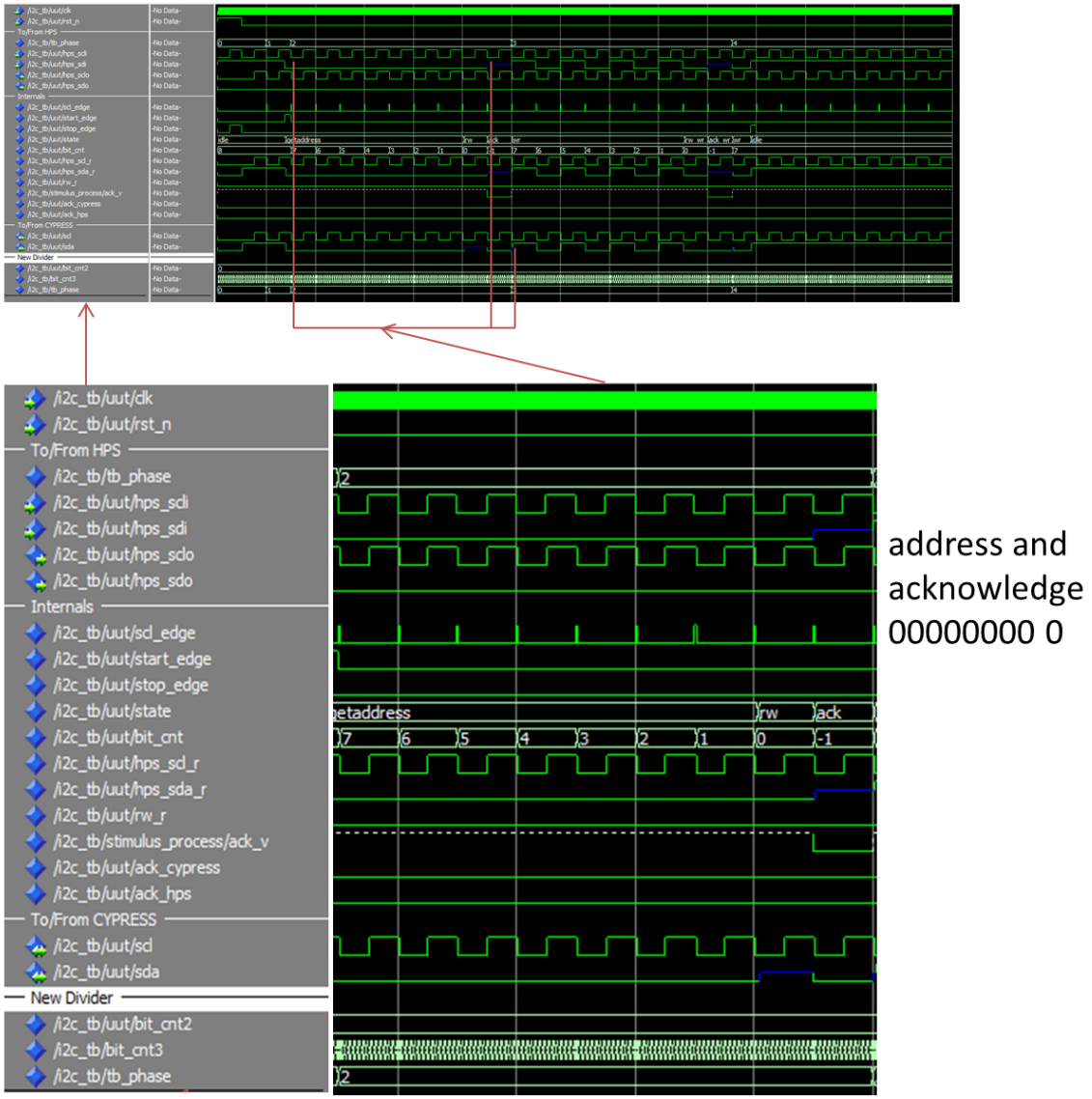


***Figure 4.6*** *Modelsim testbench for address transmission of I2C Adapter in Writing Data Process.*

Figure 4.7 illustrates the data transmission waveform of signals through I2C Adapter in a testbench. As can be seen from Figure 4.7, data bits are 1010 1010 with the first seven bits in wr state and the last bit for read/write selection in rw_wr state. Data bits come in from input port hps_sdi of I2C Adapter. Inout port sda which is output in Writing Data Process shows exact value of data bits following input port hps_sdi. Acknowledge from slave as from port sda in Writing Data Process shows in ack_wr state.

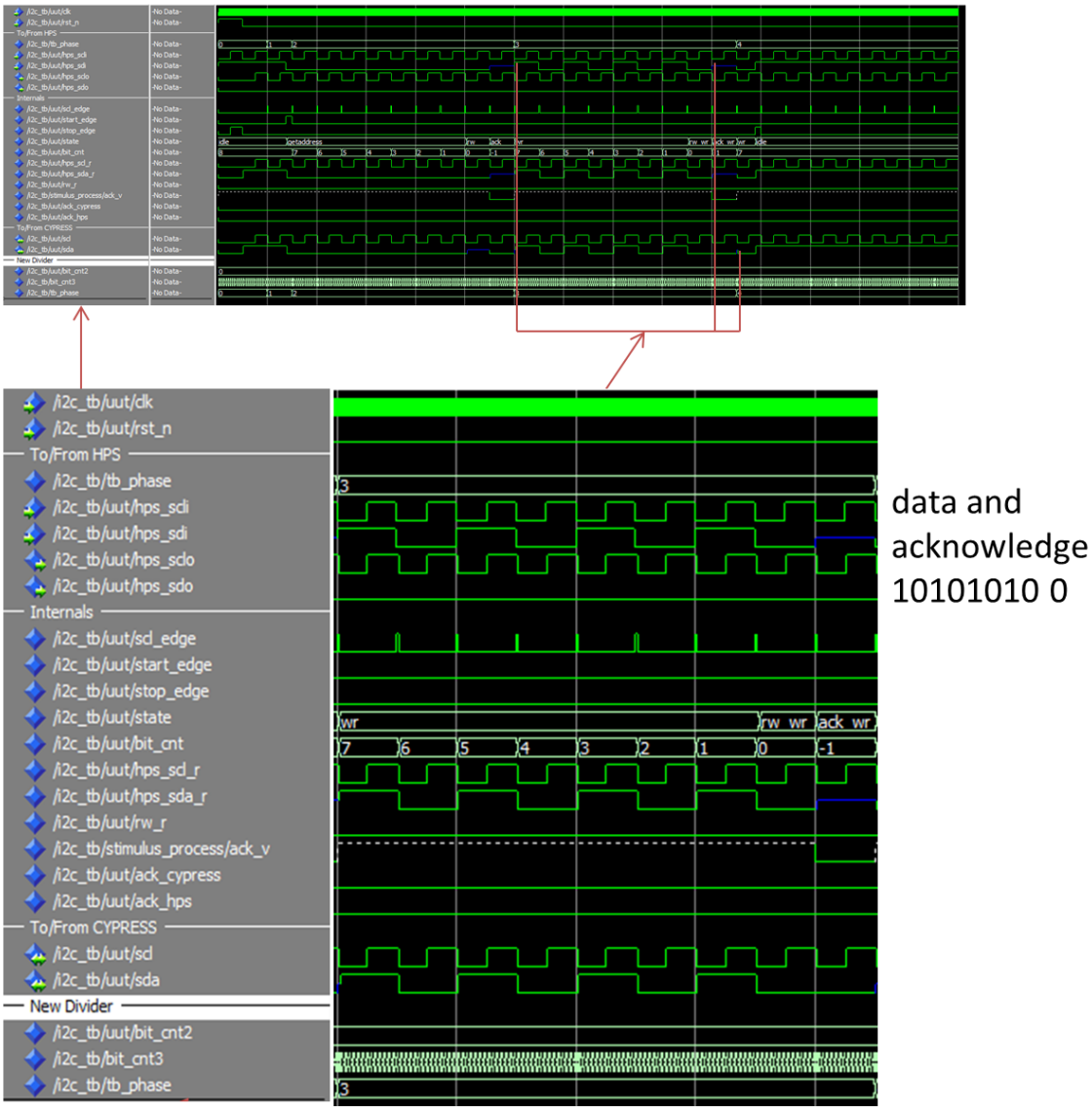


***Figure 4.7*** *Modelsim testbench for data transmission of I2C Adapter in Writing Data Process.*

Figure 4.8 and Figure 4.9 show I2C Adapter Block and components inside it respectively in Writing Data Process. I2C Adapter Block after Compile Design process by Quartus II and programing to FPGA device can be seen from RTL Viewer.

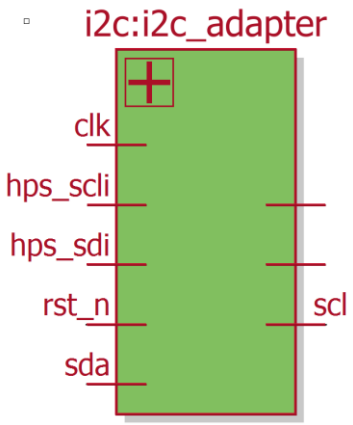


***Figure 4.8*** *I2C Adapter Block on RTL Viewer of Quartus II in Writing Data Process.*

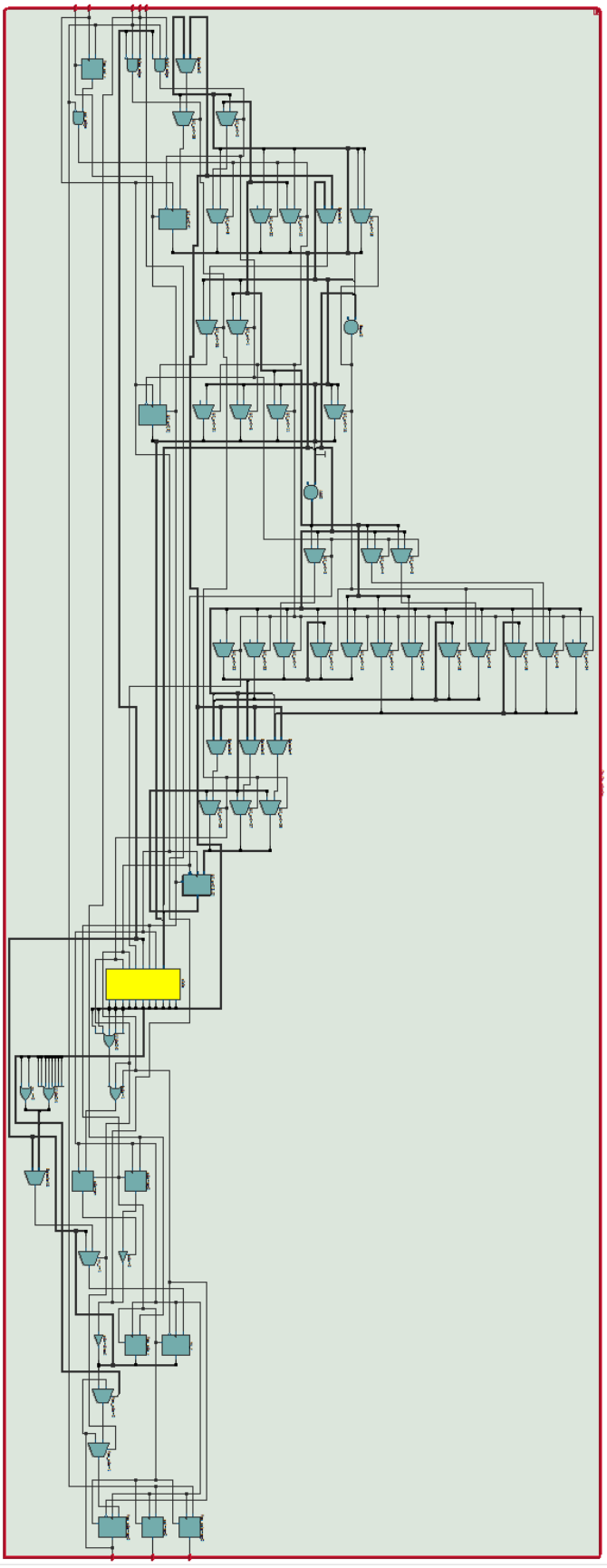

***Figure 4.9*** *Inside I2C Adapter Block on RTL Viewer of Quartus II in Writing Data Process.*

### 4.1.3 Verification of I²C Slave in Writing Data Process

As described in part 3.4, the I2C Slave is used to detect the data transmission from I2C Adapter and display to the LEDs of DE1-SoC FPGA development board by 8 bits output port in Writing Data Process following I²C protocol. Figure 4.10 illustrates reset, start condition and stop condition detection waveforms of signal transmission at I2C Slave in Writing Data Process in a testbench. As can be seen from Figure 4.10, I2C Slave can detect the active low reset, start condition as falling edge of sda during high scl period and stop condition as rising edge of sda during high scl period.

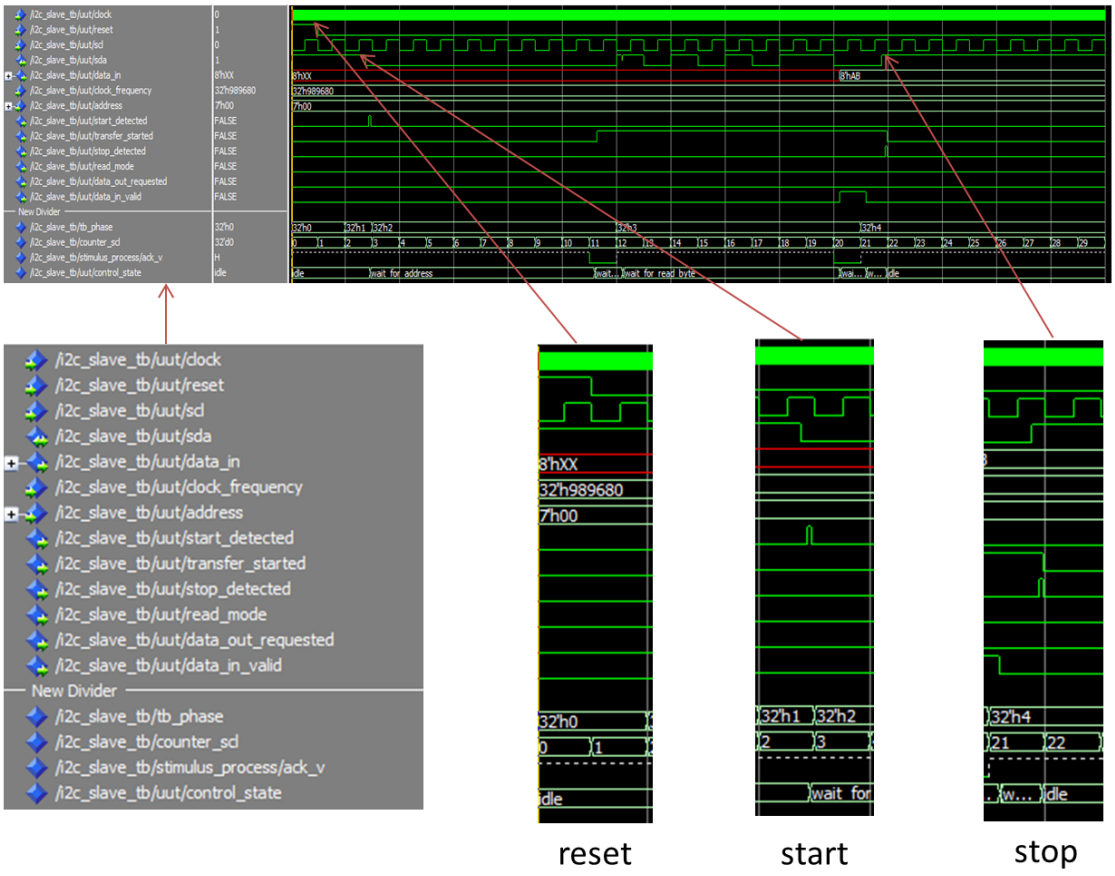


***Figure 4.10*** *Modelsim testbench for reset, start and stop of I2C Slave in Writing Data Process.*

Figure 4.11 illustrates the address transmission waveforms of signals at I2C Slave in a testbench. As can be seen from Figure 4.11, address bits are 0000 0000 with the first seven bits from phase 3 to 9 and the last bit for read/write selection in phase 10. Address bits 0000 0000 is correct and there is high signal for transfer_started at phase 11. Acknowledge from slave in Writing Data Process is given in phase 11.

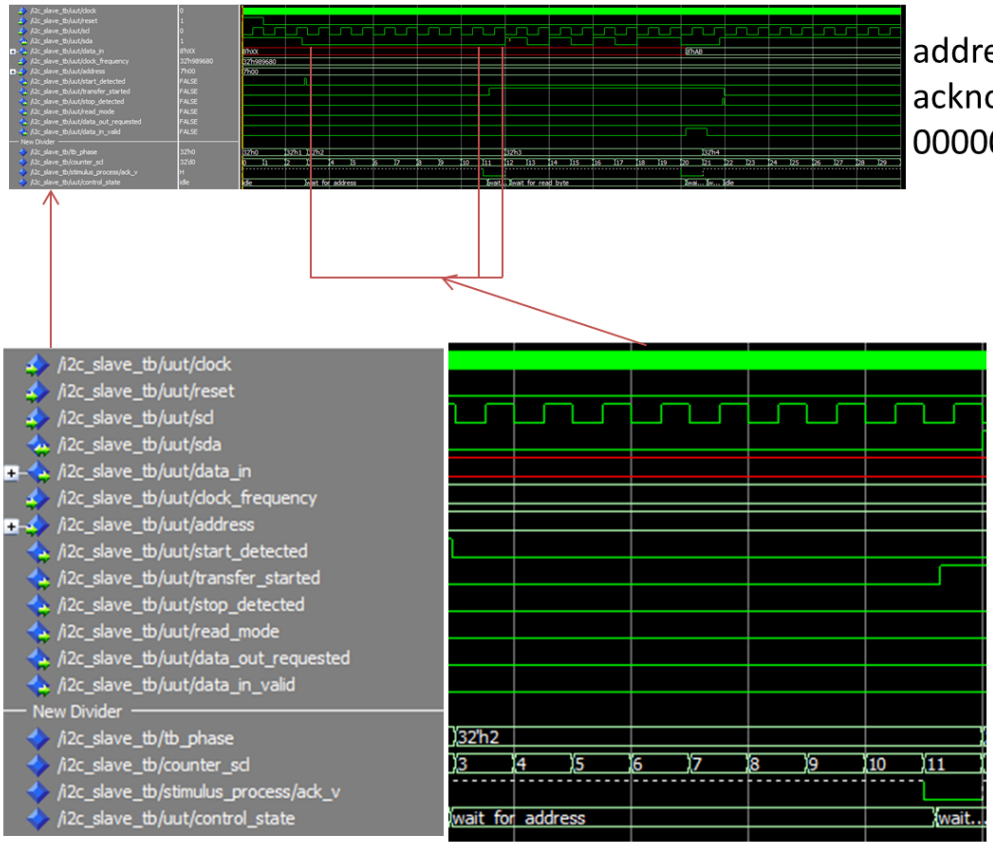


***Figure 4.11*** *Modelsim testbench for address transmission of I2C Slave in Writing Data Process.*

Figure 4.12 illustrates the data transmission waveforms of signals at I2C Slave in a testbench. As can be seen from Figure 4.12, data bits are 1010 1010 with the first seven bits from phase 12 to 18 and the last bit for read/write selection in phase 19. Data bits 1010 1010 is shown at output port data_in when there is high signal for data_in_valid at phase 20. Acknowledge from slave in Writing Data Process is given in phase 20.


***Figure 4.12*** *Modelsim testbench for data transmission of I2C Slave in Writing Data Process.*

Figure 4.13 and Figure 4.14 show I2C Slave Block and components inside it respectively in Writing Data Process. I2C Slave Block after Compile Design process by Quartus II and programing to FPGA device can be seen from RTL Viewer.

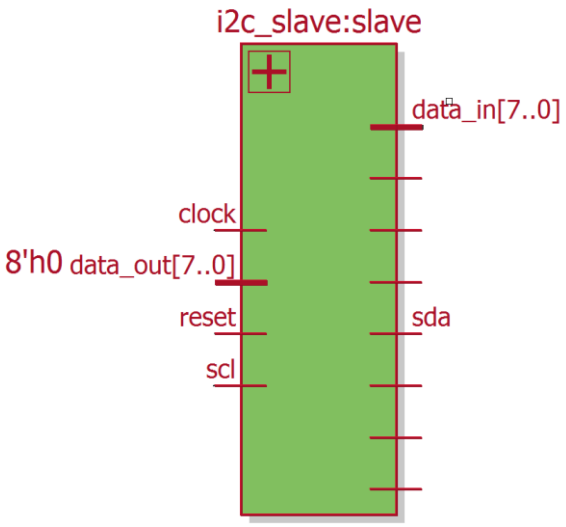


***Figure 4.13*** *I2C Slave Block on RTL Viewer of Quartus II in Writing Data Process.*

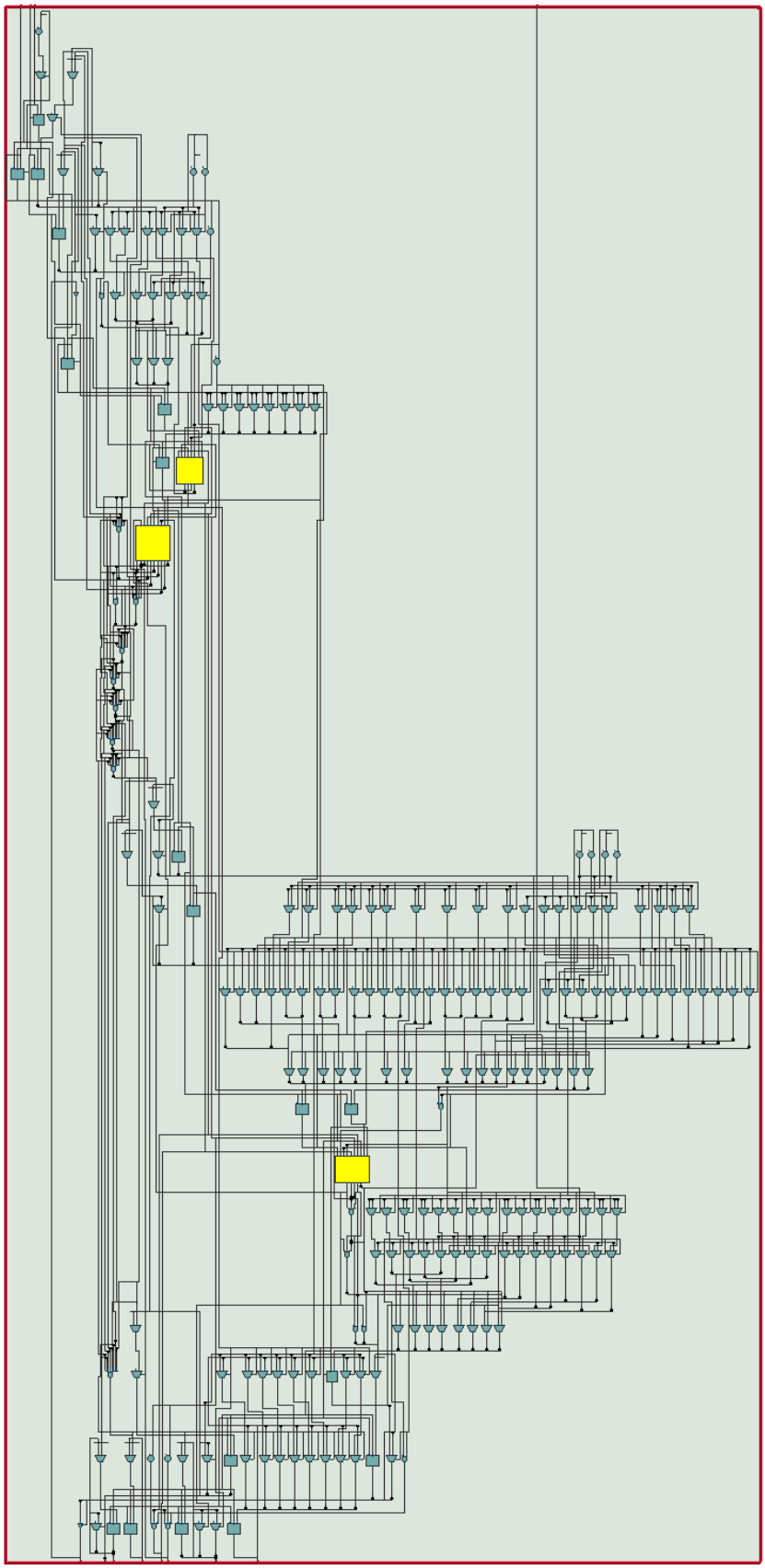

***Figure 4.14*** *Inside I2C Slave Block on RTL Viewer of Quartus II in Writing Data Process.*

## 4.2 Reading Data Process Verification

### 4.2.1 Verification of Signal Generator To Read in Reading Data Process

As described in part 3.2.2, the Signal Generator To Read has to generate the sequence of signals making reset, start condition, stop condition, address transmission and data transmission following $I^2C$ protocol. Figure 4.15 illustrates reset, start condition and stop condition waveforms of signals generating by Signal Generator To Read in a testbench. As can be seen from Figure 4.15, Signal Generator To Read generates an active low reset at the end of phase 0 when signal rst changes from high to low and remains for the rest of transmission. Start condition happens at phase 2 when the sda_ex signal has a falling edge while the scl_ex signal is high. Stop condition happens at phase 21 when the sda_ex signal has a rising edge while the scl_ex signal is high.

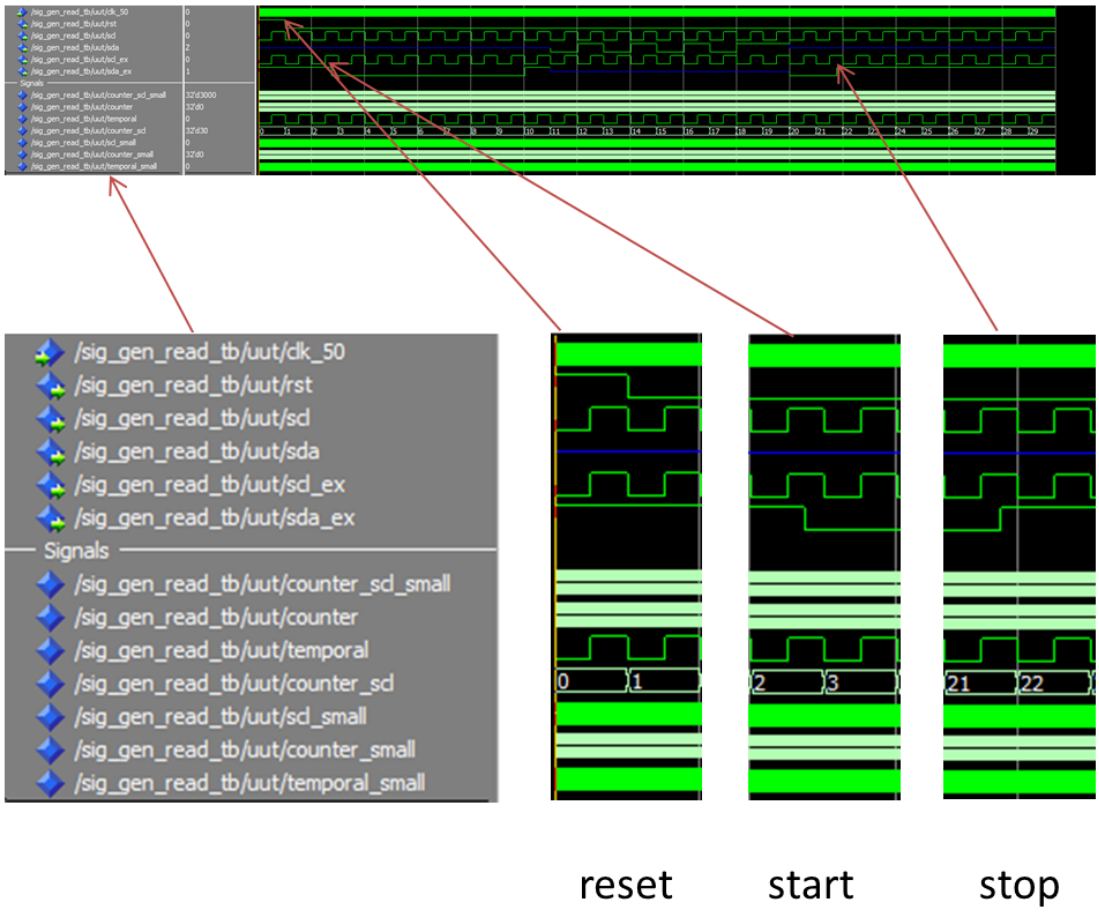


***Figure 4.15*** *Modelsim testbench for reset, start and stop of Signal Generator To Read in Reading Data Process.*

Figure 4.16 illustrates the address transmission and the data transmission waveforms of signals generating by Signal Generator To Read in a testbench. As can be seen from Figure 4.2, address bits are 0000 0000 in sda_ex with the last bit for read/write selection is generated by Signal Generator To Read from phase 3 to 10. Bit value 0 at phase 11 is used for acknowledge. Data bits are 1010 1010 in sda_ex with last bit for read/write selection is generated by Signal Generator To Read from phase 12 to 19. Bit value 0 at phase 20 is used for acknowledge.

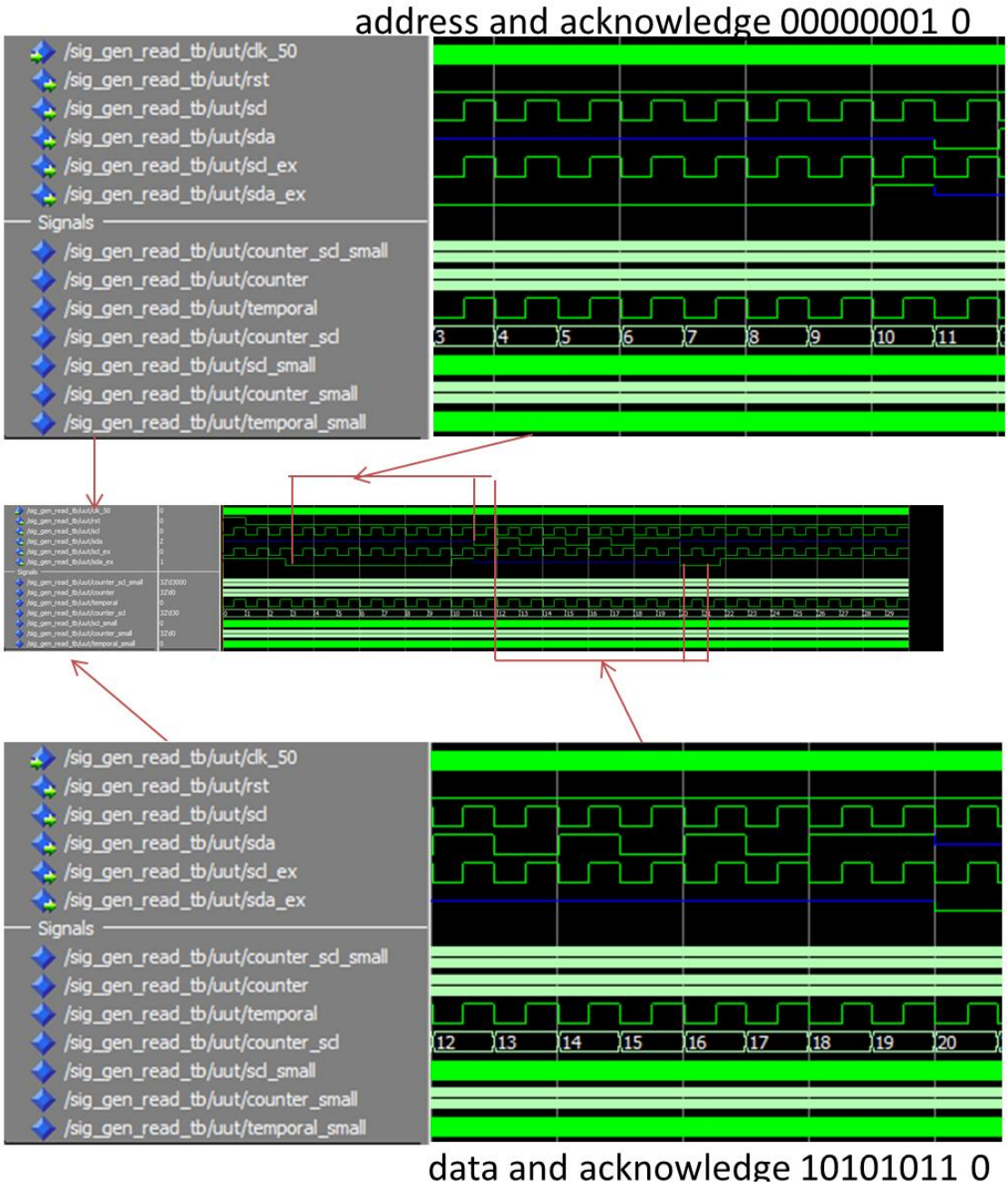


***Figure 4.16*** *Modelsim testbench for address and data transmission of Signal Generator To Read in Reading Data Process.*

Figure 4.17 and Figure 4.18 show Signal Generator To Read Block and components inside it respectively Reading Data Process. Signal Generator To Read Block after Compile Design process by Quartus II and programing to FPGA device can be seen from RTL Viewer.

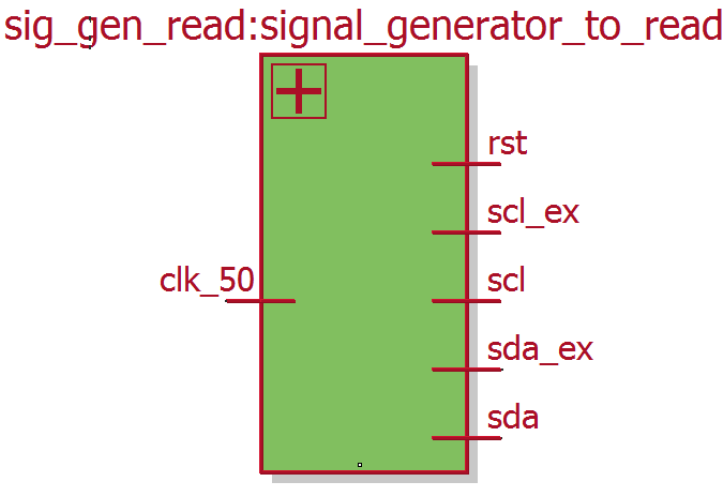


***Figure 4.17*** *Signal Generator To Read Block on RTL Viewer of Quartus II in Reading Data Process.*

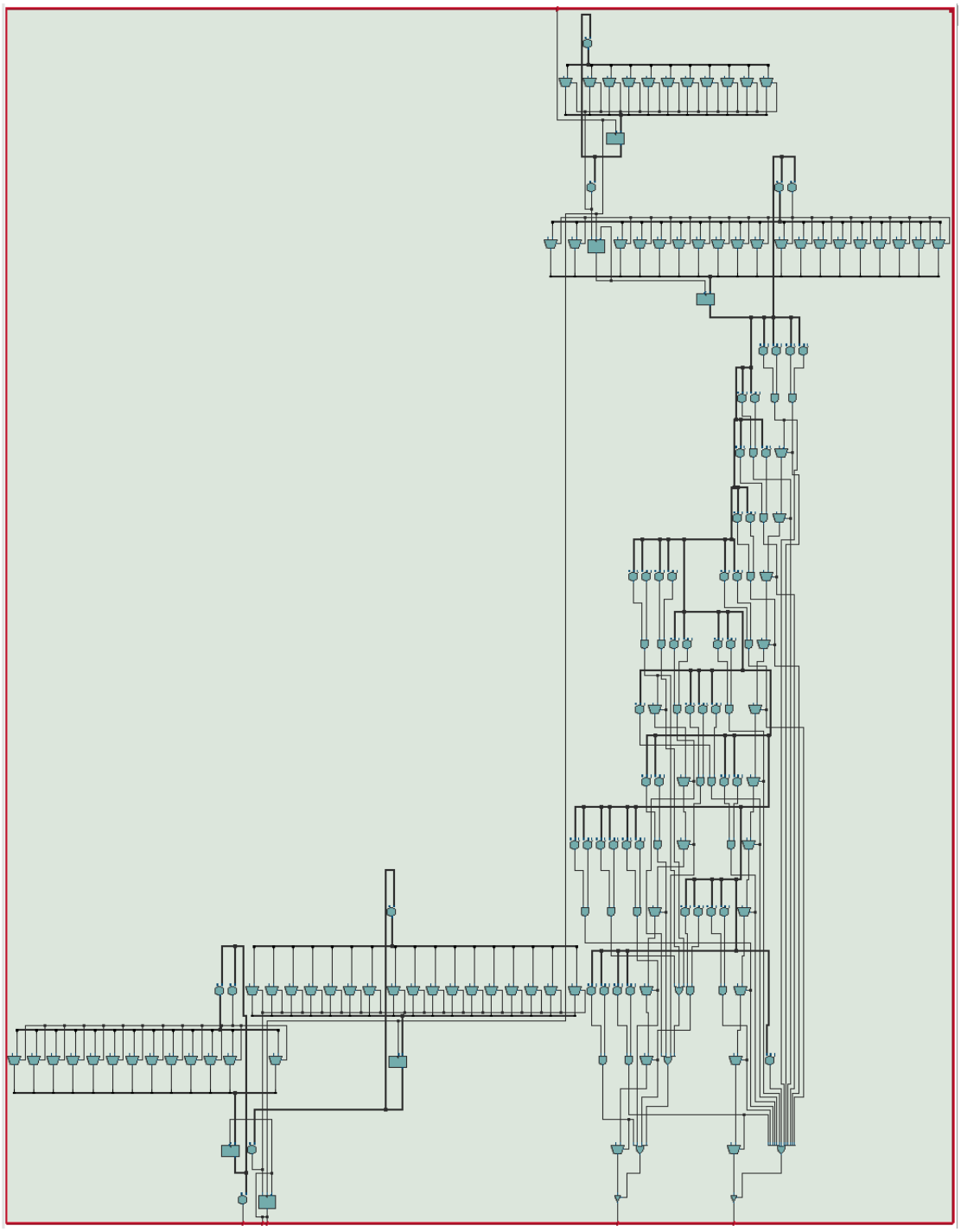

***Figure 4.18*** *Inside Signal Generator To Read Block on RTL Viewer of Quartus II in Reading Data Process.*

### 4.2.2 Verification of I²C Adapter in Reading Data Process

As described in part 3.3, the I2C Adapter is used to transmit data from a slave to the hard processor core ARM Cortex-A9 in Reading Data Process following I²C protocol. Figure 4.19 illustrates the reset, start condition and stop condition detection waveforms of signal transmission through I2C Adapter in Reading Data Process in a testbench. As can be seen from Figure 4.19, I2C Adapter can detect the active low reset, start condition as falling edge of sda during high scl period and stop condition as rising edge of sda during high scl period.

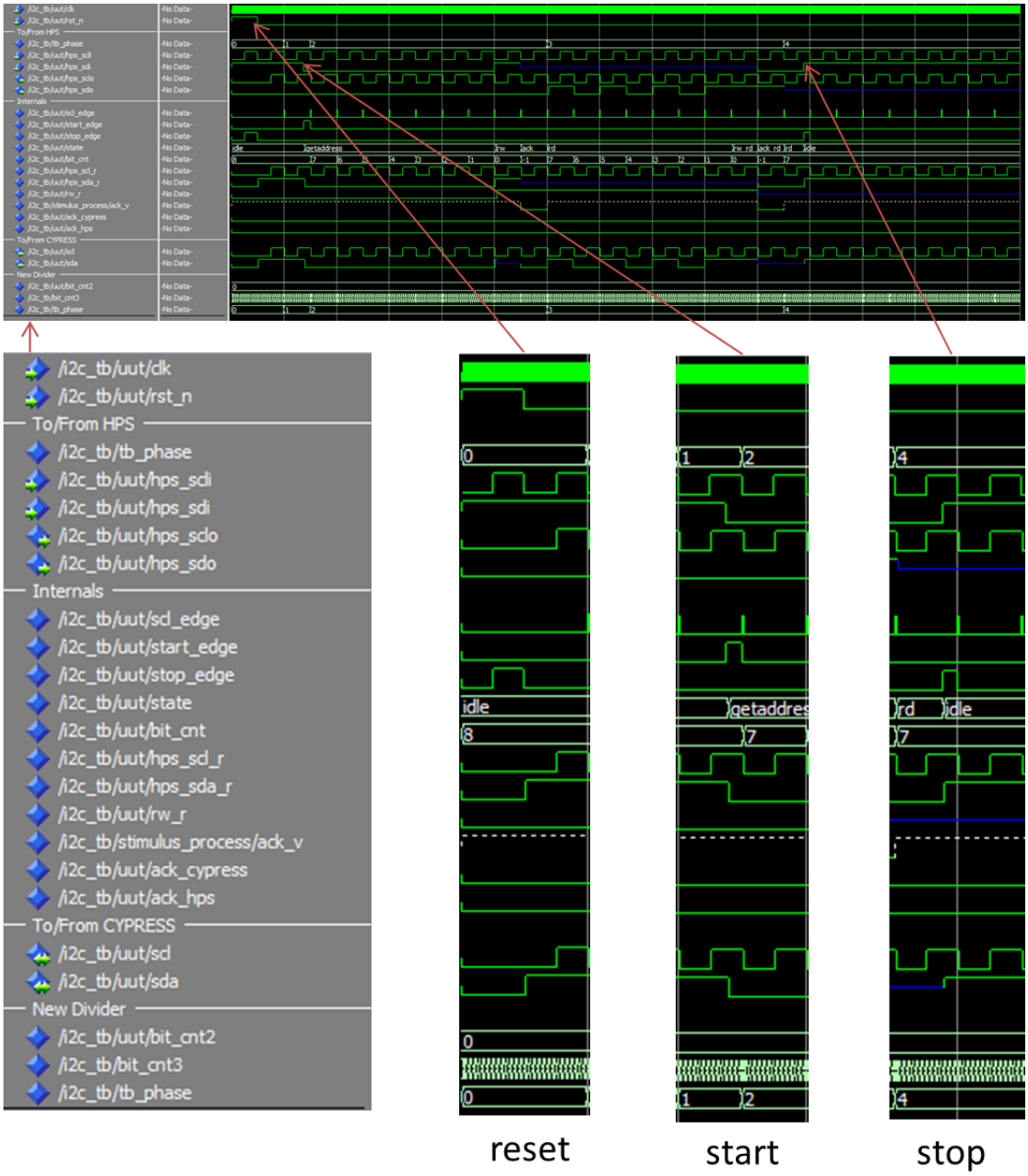


***Figure 4.19*** *Modelsim testbench for reset, start and stop of I2C Adapter in Reading Data Process.*

Figure 4.20 illustrates the address transmission waveforms of signals through I2C Adapter in a testbench. As can be seen from Figure 4.20, address bits are 0000 0001 with the first seven bits in getaddress state and the last bit for read/write selection in rw state. Address bits come in from input port hps_sdi of I2C Adapter. Inout port sda which is output in transmitting address to slave in Reading Data Process shows exact value of address bits following input port hps_sdi. Acknowledge for address bits from slave core as from port sda in Reading Data Process shows in ack state.

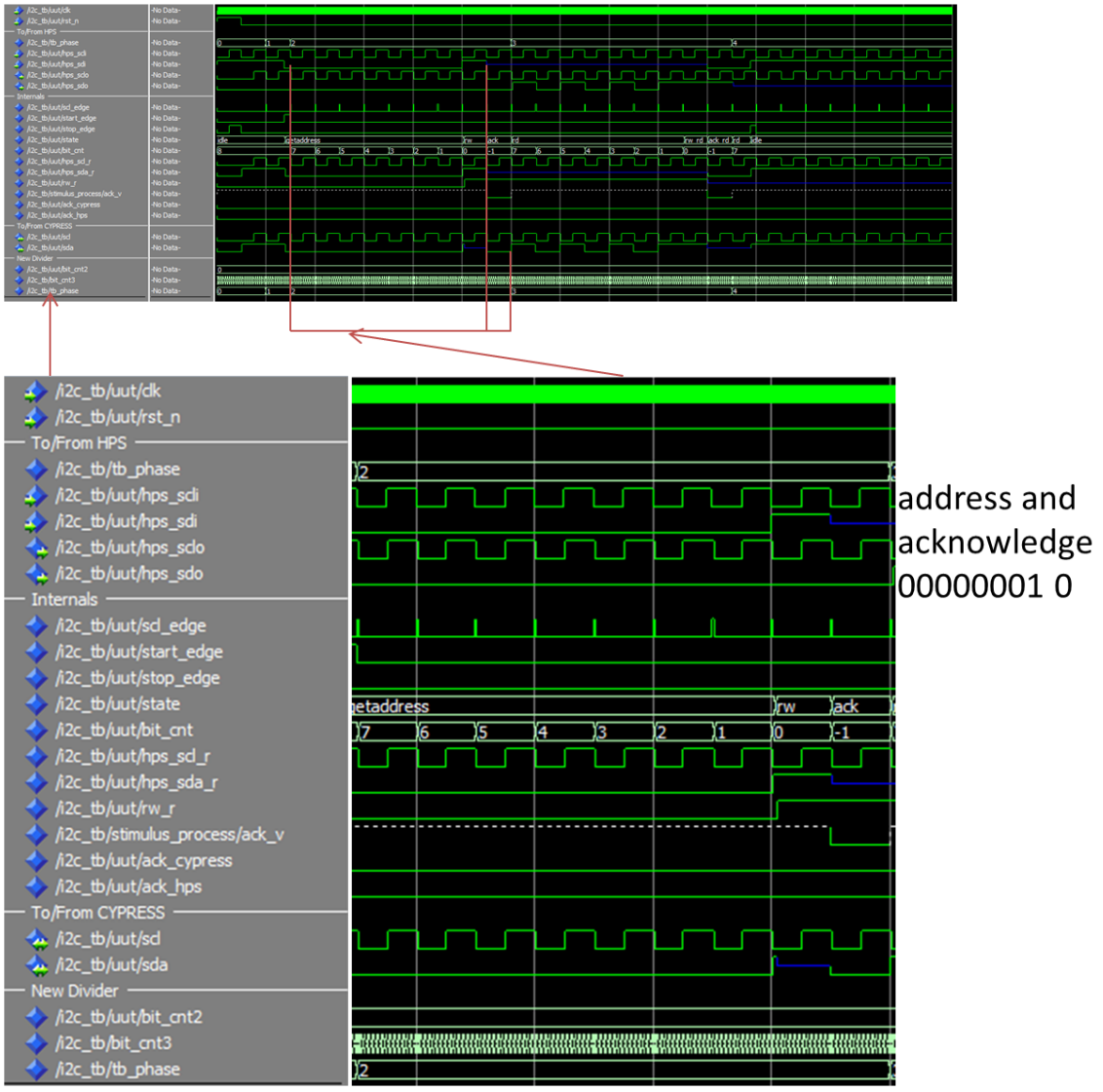


***Figure 4.20*** *Modelsim testbench for address transmission of I2C Adapter in Reading Data Process.*

Figure 4.21 illustrates the data transmission waveforms of signals through I2C Adapter in a testbench. As can be seen from Figure 4.21, data bits are 1010 1011 with the first seven bits in rd state and the last bit for read/write selection in rw_rd state. Data bits come in from inout port sda of I2C Adapter which is input for transmitting data in Reading Process. Output port hps_sdo shows exact data bits following port sda. Acknowledge from the hard processor core as from port hps_sdi in Reading Data Process shows in ack_rd state.

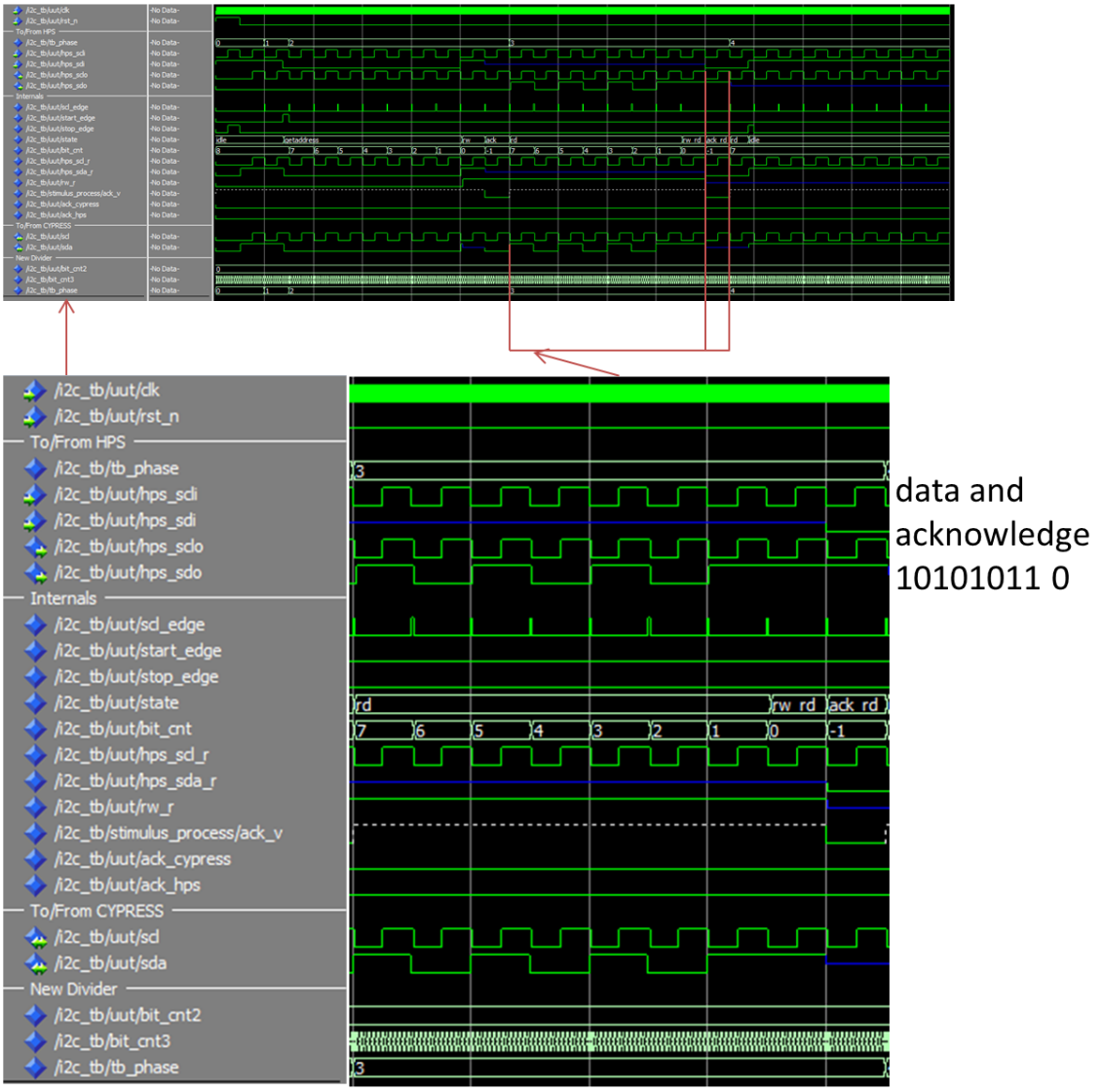


***Figure 4.21*** *Modelsim testbench for data transmission of I2C Adapter in Reading Data Process.*

Figure 4.22 and Figure 4.23 show I2C Adapter Block and components inside it respectively in Reading Data Process. I2C Adapter Block after Compile Design process by Quartus II and programing to FPGA device can be seen from RTL Viewer.

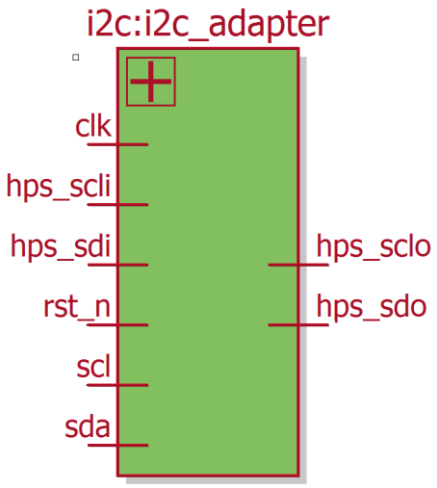


***Figure 4.22*** *I2C Adapter Block on RTL Viewer of Quartus II in Reading Data Process.*

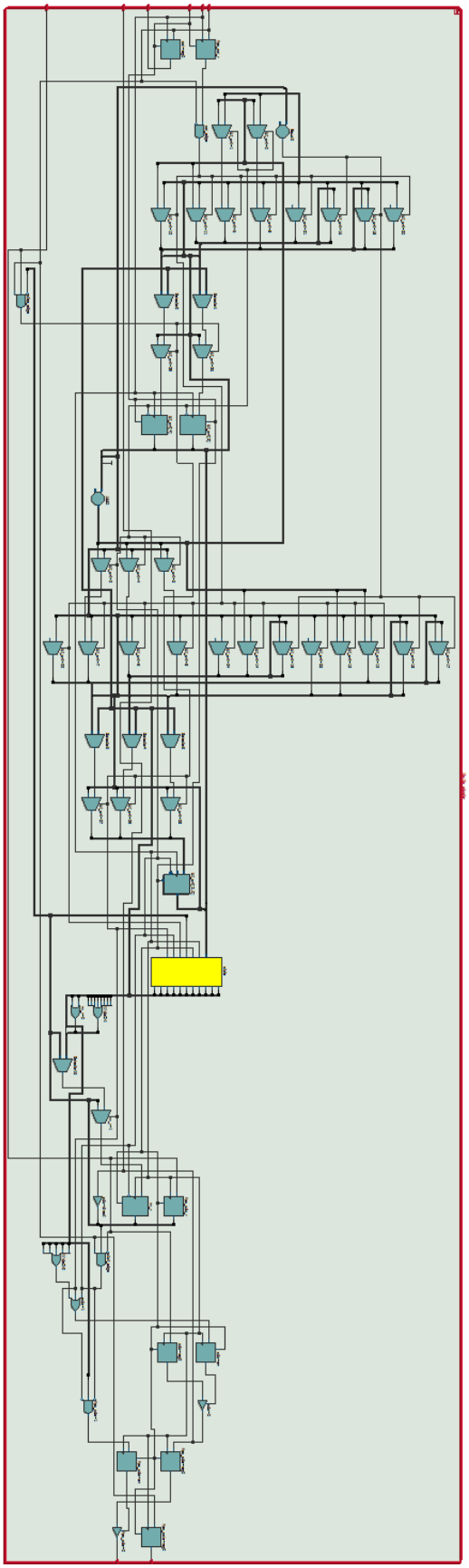

***Figure 4.23*** *Inside I2C Adapter Block on RTL Viewer of Quartus II in Writing Data Process.*

### 4.2.3 Verification of I²C Slave in Reading Data Process

As described in part 3.4, the I2C Slave is used to detect the data transmission from I2C Adapter and display to the LEDs of DE1-SoC FPGA development board by 8 bits output port in Reading Data Process following I²C protocol. Figure 4.24 illustrates reset, start condition and stop condition detection waveforms of signal transmission at I2C Slave in Reading Data Process in a testbench. As can be seen from Figure 4.24, I2C Slave can detect the active low reset, start condition as falling edge of sda during high scl period and stop condition as rising edge of sda during high scl period.

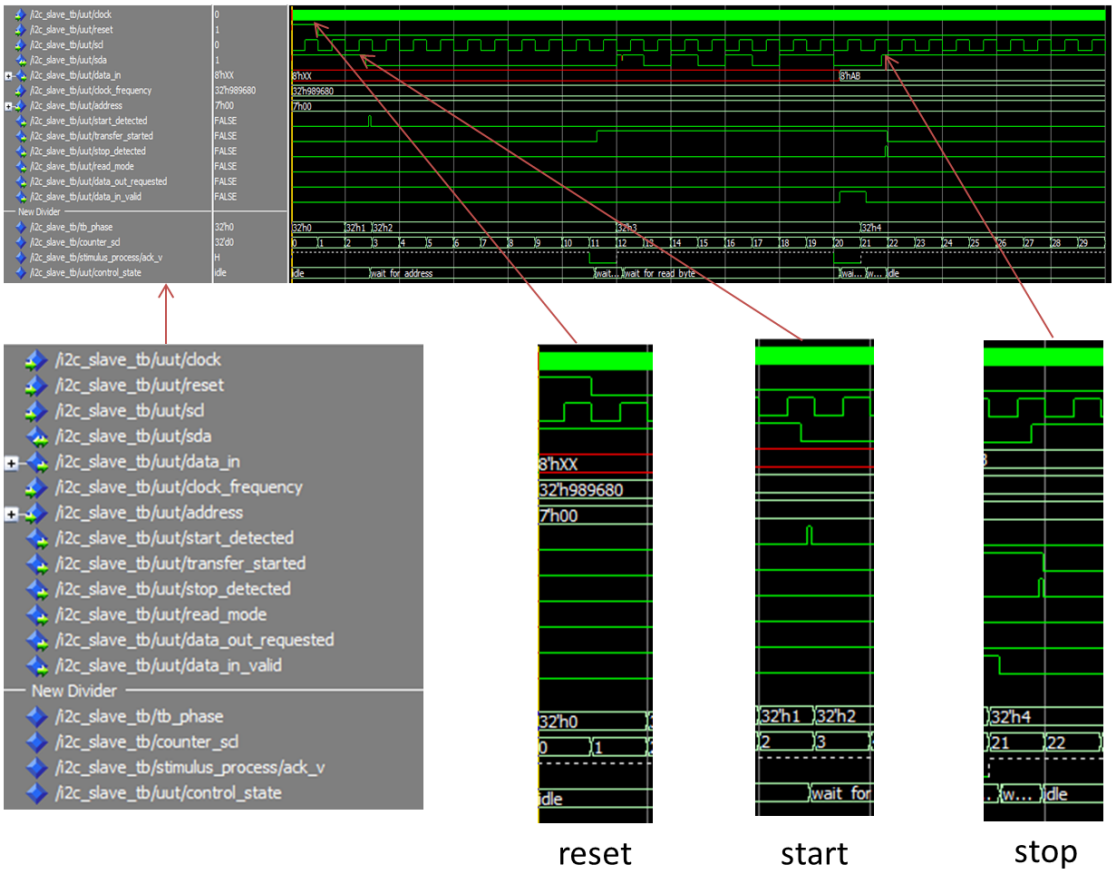


***Figure 4.24*** *Modelsim testbench for reset, start and stop of I2C Slave in Reading Data Process.*

Figure 4.25 illustrates the address transmission waveforms of signals at I2C Slave in a testbench. As can be seen from Figure 4.25, address bits are 0000 0000 with the first seven bits from phase 3 to 9 and the last bit for read/write selection in phase 10. Address bits 0000 0000 is correct and there is high signal for transfer_started at phase 11. Acknowledge from slave in Reading Data Process is given in phase 11.

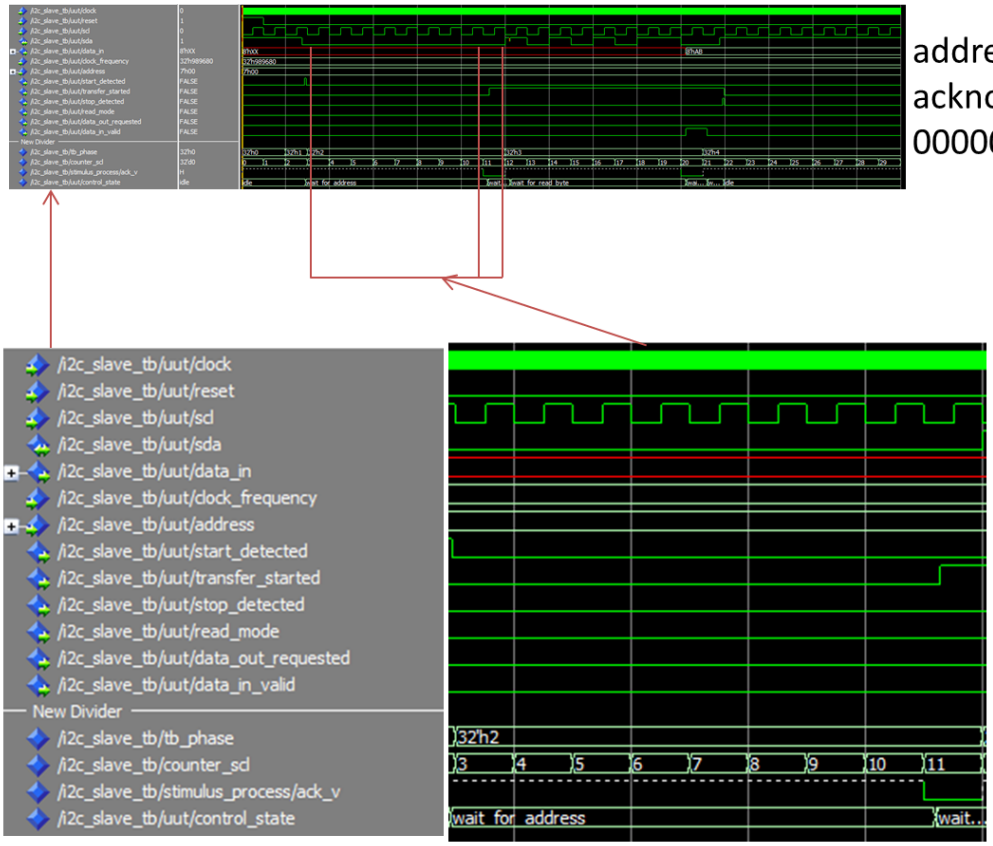


***Figure 4.25*** *Modelsim testbench for address transmission of I2C Slave in Reading Data Process.*

Figure 4.26 illustrates the data transmission waveform of signals at I2C Slave in a testbench. As can be seen from Figure 4.26, data bits are 1010 1011 with the first seven bits from phase 12 to 18 and the last bit for read/write selection in phase 19. Data bits 1010 1011 is shown at output port data_in when there is high signal for data_in_valid during phase 20. Acknowledge from slave in Reading Data Process is given in phase 20.

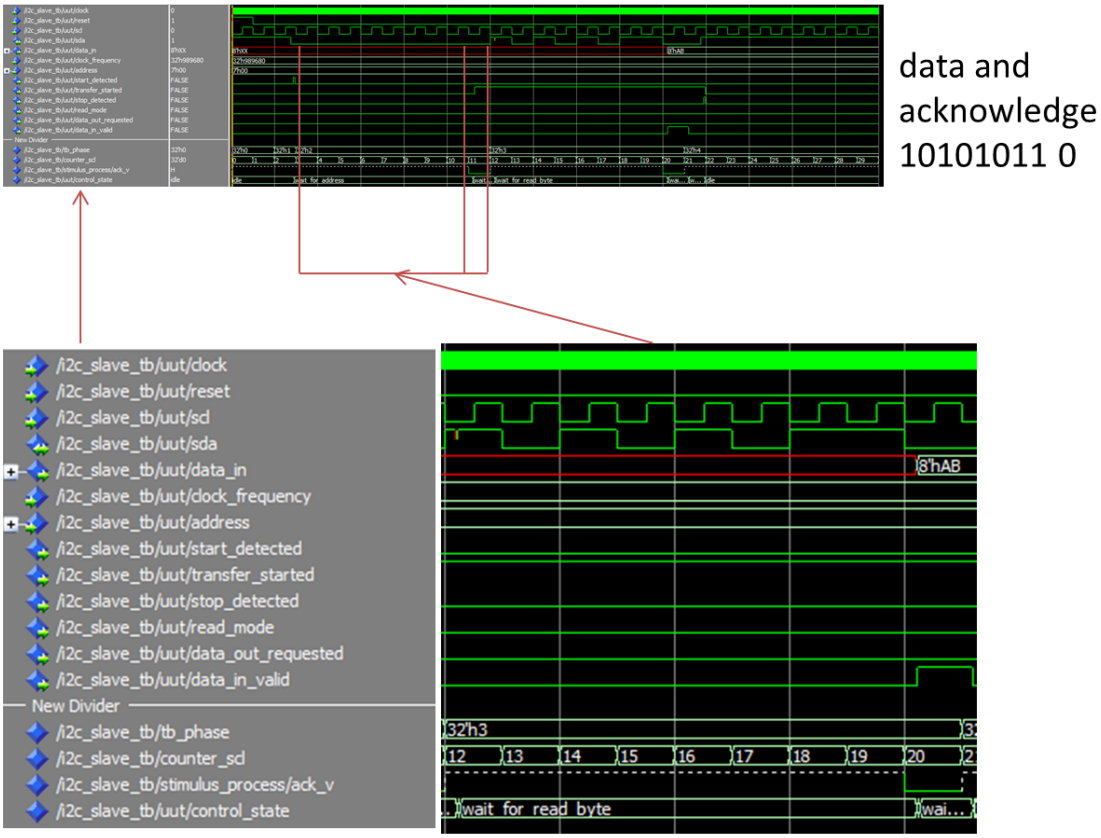


***Figure 4.26*** *Modelsim testbench for data transmission of I2C Slave in Reading Data Process.*

Figure 4.13 and Figure 4.14 show I2C Slave Block and components inside it respectively in Reading Data Process. I2C Slave Block after Compile Design process by Quartus II and programing to FPGA device can be seen from RTL Viewer.

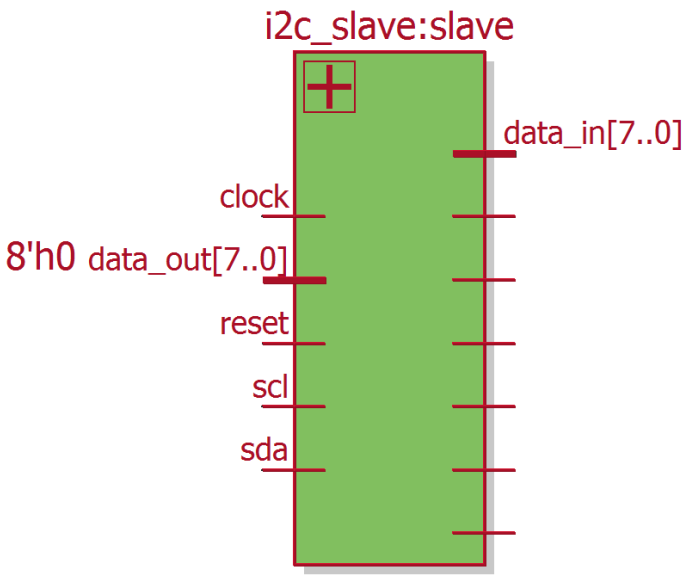


***Figure 4.27*** *I2C Slave Block on RTL Viewer of Quartus II in Reading Data Process.*

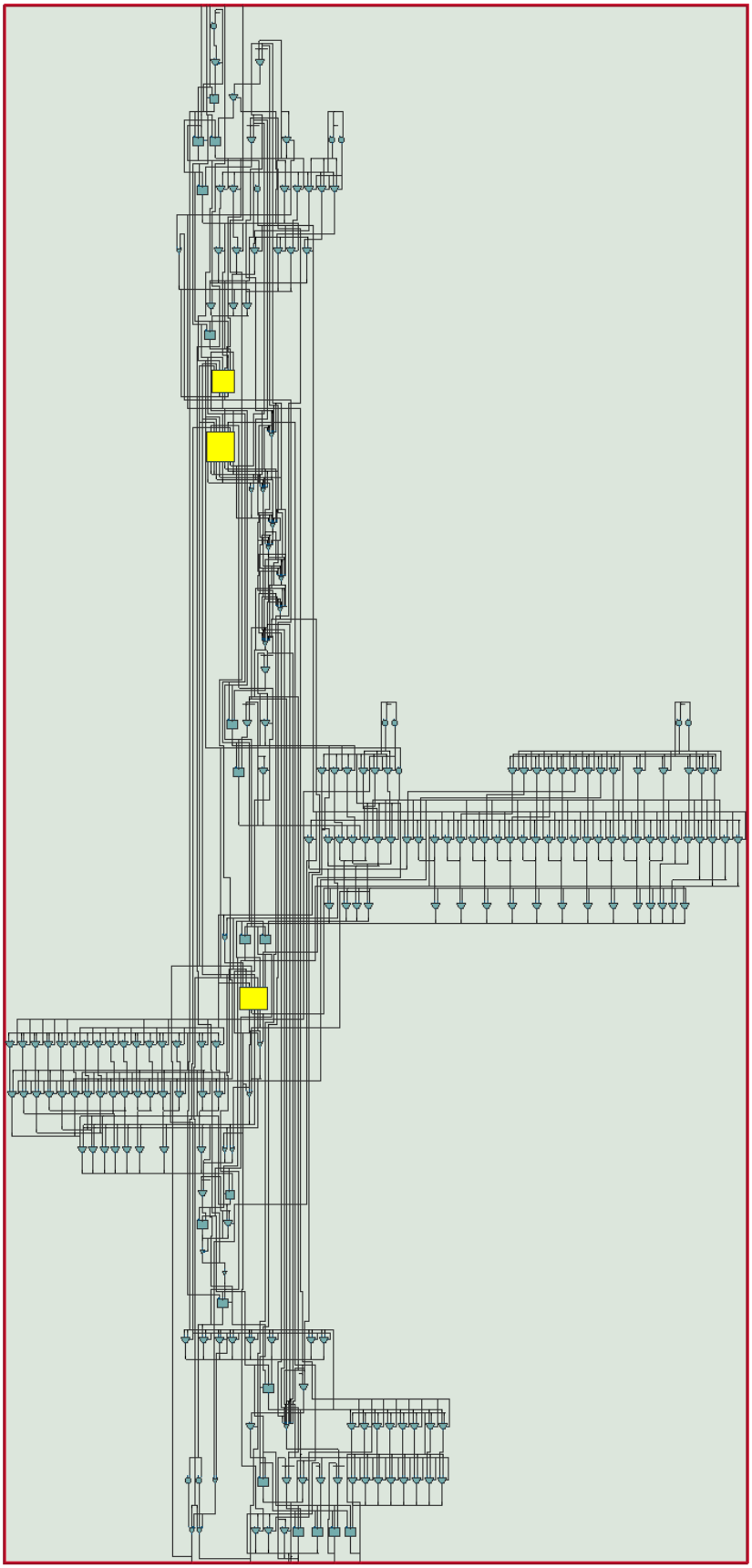

***Figure 4.28*** *Inside I2C Slave Block on RTL Viewer of Quartus II in Reading Data Process.*

## 4.3 Whole System Verification

### 4.3.1 Register Transfer Level

Register Transfer Level is the flow of digital signals between hardware registers and logical operations, which models a synchronous digital circuit. The implementations for Top Level Design of Writing Data Process and Reading Data Process after Compile Design process and Programmer to DE1-SoC FPGA development board can be seen from RTL Viewer of Quartus II. Figure 4.29 illustrates blocks and connection between them in an FPGA device following Writing Data Process as in 3.1.1. Figure 4.30 illustrates blocks and connection between them in an FPGA device following Reading Data Process as in 3.1.2.

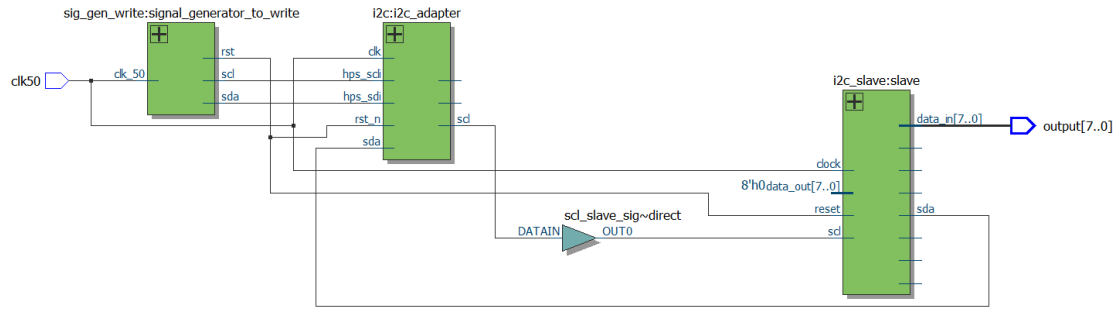


***Figure 4.29*** *Top Level Design for Writing Data Process on RTL Viewer of Quartus II.*

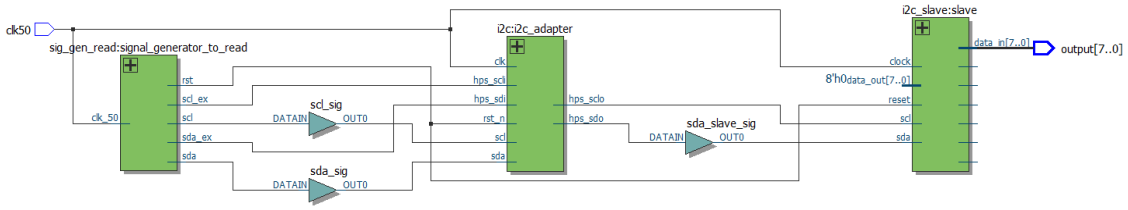


***Figure 4.30*** *Top Level Design for Reading Data Process on RTL Viewer of Quartus II.*

### 4.3.2 Assign Pins and Display Result by DE1-SoC

In order to check the transmission of the whole Writing Data Process and Reading Data Process, it is necessary to assign the output of Top Level Design to the LEDs of DE1-SoC FPGA development board. Table 4.1 and Table 4.2 show the Pin Assignment with FPGA Pin numbers for 50 MHz frequency and LEDRs in DE1-SoC FPGA development board.

***Table 4.1*** *Pin Assignment of Clock Inputs.*

| *Signal Name* | *FPGA Pin No.* | *Description* | *I/O Standard* |
|---|---|---|---|
| CLOCK_50 | PIN_AF14 | 50 MHz clock input | 3.3V |
| CLOCK2_50 | PIN_AA16 | 50 MHz clock input | 3.3V |
| CLOCK3_50 | PIN_Y26 | 50 MHz clock input | 3.3V |
| CLOCK4_50 | PIN_K14 | 50 MHz clock input | 3.3V |
| HPS_CLOCK1_25 | PIN_D25 | 25 MHz clock input | 3.3V |
| HPS_CLOCK2_25 | PIN_F25 | 25 MHz clock input | 3.3V |

***Table 4.2*** *Pin Assignment of LEDs.*

| *Signal Name* | *FPGA Pin No.* | *Description* | *I/O Standard* |
|---|---|---|---|
| LEDR[0] | PIN_V16 | LED [0] | 3.3V |
| LEDR[1] | PIN_W16 | LED [1] | 3.3V |
| LEDR[2] | PIN_V17 | LED [2] | 3.3V |
| LEDR[3] | PIN_V18 | LED [3] | 3.3V |
| LEDR[4] | PIN_W17 | LED [4] | 3.3V |
| LEDR[5] | PIN_W19 | LED [5] | 3.3V |
| LEDR[6] | PIN_Y19 | LED [6] | 3.3V |
| LEDR[7] | PIN_W20 | LED [7] | 3.3V |
| LEDR[8] | PIN_W21 | LED [8] | 3.3V |
| LEDR[9] | PIN_Y21 | LED [9] | 3.3V |

Figure 4.31 illustrates the Assignment Editor for Top Level Design after Compile Design process and Programmer process by Quartus II to DE1-SoC FPGA development board. Pin Assignment is made by Pin Planner of Quartus II software.

| | Status | From | To | Assignment Name | Value | Enabled |
|---|---|---|---|---|---|---|
| 1 | Ok | | clk50 | Location | PIN_AF14 | Yes |
| 2 | Ok | | output[7] | Location | PIN_W20 | Yes |
| 3 | Ok | | output[6] | Location | PIN_Y19 | Yes |
| 4 | Ok | | output[5] | Location | PIN_W19 | Yes |
| 5 | Ok | | output[4] | Location | PIN_W17 | Yes |
| 6 | Ok | | output[3] | Location | PIN_V18 | Yes |
| 7 | Ok | | output[2] | Location | PIN_V17 | Yes |
| 8 | Ok | | output[1] | Location | PIN_W16 | Yes |
| 9 | Ok | | output[0] | Location | PIN_V16 | Yes |

***Figure 4.31*** *Assignment Editor for Top Level Design.*

Figure 4.32 and Figure 4.33 show output of Top Level Design for Writing Data Process and Reading Data Process respectively by LEDRs of DE1-SoC FPGA development board. The data output for Writing Data Process is 1010 1010 as displaying from LEDR [7] to LEDR [0] as the picture caption in Figure 4.32.

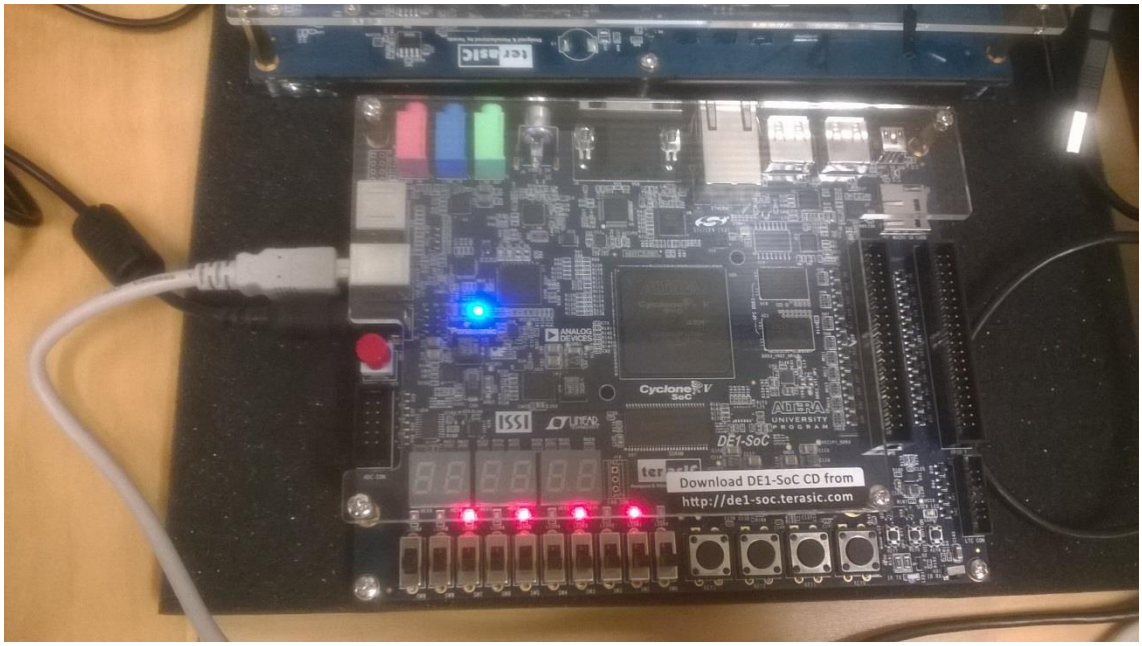


***Figure 4.32*** *Output of Top Level Design Block displaying on Leds of DE1-SoC in Writing Data Process.*

The data output for Reading Data Process is 1010 1011 as displaying from LEDR [7] to LEDR [0] of DE1-SoC FPGA development board as the picture caption in Figure 4.33.

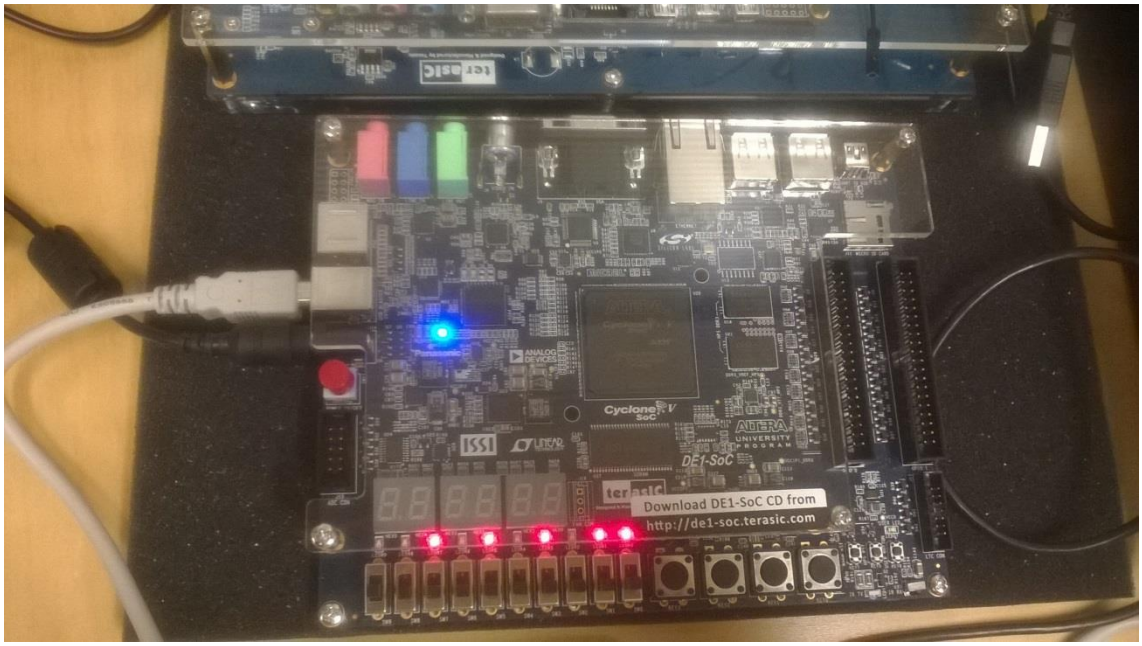


***Figure 4.33*** *Output of Top Level Design Block displaying on Leds of DE1-SoC in Reading Data Process.*

# 5. CONCLUSIONS

The thesis presented an $I^2C$ adapter for the new Altera Cyclone V SoC-FGPA. The $I^2C$ Adapter was created between the HPS part and FPGA part of the Cyclone V.

DE1-SoC FPGA development board [22] was used and the sampling frequency for the whole system verification is 50 MHz frequency. The operating frequency inside Signal Generator, $I^2C$ Adapter and $I^2C$ Slave is 200 Hz. In this thesis, the purpose is testing the communication of $I^2C$ Adapter. The 7-bit addressing mode of $I^2C$ protocol is applied for getting address of $I^2C$ Slave. As the results shown in Chapter 4 Verification And Result, address transaction and data transaction is correct. The waveforms action of Signal Generator, $I^2C$ Adapter and $I^2C$ Slave is shown by Modelsim SE 10.2c simulator. After compiling the design and programming by Altera Quartus II 13.1, LEDs on DE1-SoC FPGA development board shows correct data from Signal Generator in both Writing Data Process and Reading Data Process through the $I^2C$ Adapter. It can be seen from the Flow Summary of Quartus II that, in the Writing Data Process, the total number of logic utilization (in ALMs) used is 121, total number of registers is 143, and total number of pins is 9. In the Reading Data Process, the total number of logic utilization (in ALMs) used is 135, the total number of registers is 158, and the total number of pins is 9.

The goals of the thesis are reached by implementing $I^2C$ Adapter and verifying data transactions going through it properly. The testing with Hard Processor System will be future work. Using and applying the $I^2C$ Adapter built in this thesis; users can take access data input to HPS from real devices. The $I^2C$ bus is popular and when the number of available addresses in the 7-bit addressing mode is recognized too small, the new addressing mode (the 10-bit mode) will be necessary. The future improvement can be done by modifying the $I^2C$ Adapter to 10-bit addressing mode. The new addressing mode also supports the old one. Devices with 7-bit addresses can be connected with devices with 10-bit addresses on the same mode. In this mode, the first two bytes are dedicated for address and data direction. The format of the first byte is 11110xx; the last two bits of the first byte, combined with eight bits in the second byte from the 10-bit address. After testing communication with data input from dual-core ARM processor and modifying to 10-bit addressing mode, $I^2C$ can be utilized in any device that need to communicate with HPS such as the LCD multimedia color touch panel from TerasIC [23], SPD EEPROMs [24] on SDRAM or NVRAM chips [25]. For this thesis, the default address of $I^2C$ Slave is 00000000 for Writing data operation, and 00000001 for Reading data operation. The address of $I^2C$ Slave can be modified to the real device

address; the user also can connect $I^2C$ Adapter to the slaves other than $I^2C$ Slave from this thesis. Modification and verification data transmission through other slaves and real devices will be future work.

# REFERENCES


[1] I. Kuon, R. Tessier, and J. Rose. FPGA architecture: Survey and challenges. *Found. Trends Electron. Des. Autom.,* 2:135-253, Feb.2008.

[2] O. Esko, ASIP Integration and Verification Flow for FPGA. Master's thesis, Tampere University of Technology, Finland, June 2011.

[3] S.D. Brown. An overview of technology, architecture and CAD tools for programmable logic devices. In *Proc. IEEE Custom Integrated Cir. Conf.,* pages 69-76, May 1994.

[4] C. Riviere, Supercomputers for all, the next frontier for high performance computing. Special Report, 2013, available at: http://www.prace-ri.eu/IMG/pdf/prace_report_october_2013.pdf.

[5] P. Franzon, S. Perelstein, A. Hurst. Introduction to ASIC Design, spring 1999, available at: http://www.ece.ncsu.edu/asic/tutorials/tutor1/tutor1.pdf.

[6] I. Kuon and J. Rose. Measuring the Gap between FPGAs and ASICs. *IEEE Trans. Computer-Aided Design Integrated Circ. Syst.,* 26(2):203-215, Feb. 2007.

[7] Altera Corporation. *Cyclone V Overview*, available at: https://www.altera.com/products/fpga/cyclone-series/cyclone-v/overview.html, Referenced 2.4.2015.

[8] Altera Corporation. *Introduction to Cyclone V Hard Processor System*, February 2014.

[9] Terasic Corporation. *DE1-SoC Board*, available at: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836&PartNo=2, Referenced 2.4.2015.

[10] Terasic Corporation. *DE1-SoC Board Overview*, available at: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836. Referenced 5.4.2015.

[11] Terasic Corporation. *DE1-SoC Board Specification*, available at: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836. Referenced 5.4.2015.

[12] Altera Corporation. *Introduction to the Quartus II Software, Version 10.0,* available at: https://www.altera.com/en_US/pdfs/literature/manual/archives/intro_to_quartus2.pdf. Referenced 5.4.2015

[13] Mentor Graphics Corporation. *ModelSim SE Tutorial, Software Version 10.1,* available at: http://cs.colby.edu/courses/S15/cs232/labs/lab01/modelsim_se_tut.pdf.

[14] Philips Semiconductors Corporation. *$I^2C$ Manual,* March 24, 2003, available at: http://www.nxp.com/documents/application_note/AN10216.pdf.

[15] B. Akesson. An introduction to SDRAM and memory controllers, available at: http://www.es.ele.tue.nl/premadona/files/akesson01.pdf.

[16] Security Standards Council. *Payment Card Industry Security Standards,* available at: https://www.pcisecuritystandards.org/pdfs/pcissc_overview.pdf.

[17] H. Zumbahlen, *Basic Linear Design.* Chapter 6: Converter, Section 6.1, Analog Devices, 2007.

[18] H. Zumbahlen, *Basic Linear Design.* Chapter 6: Converter, Section 6.2, Analog Devices, 2007.

[19] M. Rabaey, *Digital Integrated Circuit.* Prentice Hall, 1995.

[20] Philips Semiconductors Corporation. *The $I^2C$ Bus Specification, Version 2.1,* January, 2000, available at: http://i2c2p.twibright.com/spec/i2c.pdf.

[21] NXP Semiconductors Corporation. *$I^2C$ bus specification and user manual,* Rev. 6-4 April 2014, available at: http://www.nxp.com/documents/user_manual/UM10204.pdf.

[22] Terasic Corporation. *DE1-SoC User Manual,* June 2014, available at: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836&PartNo=4.

[23] Terasic Corporation. *VEEK-MT-SoCKit Overview,* available at: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=882.

[24] Microchip Technology Inc. *Basic Serial EEPROM Operation,* 1993, available at: http://ecee.colorado.edu/~mcclurel/man536.pdf.

[25] STMicroelectronics More Intelligent Solutions. *Non-Volatile RAM and RTC*, Selection Guide, available at: http://www.mouser.com/catalog/supplier/library/pdf/STNVRAM.pdf.

[26] Microchip Technology Inc. *Section 20. Serial Peripheral Interface (SPI)*, available at: http://ww1.microchip.com/downloads/en/DeviceDoc/70067E.pdf, Referenced 7.4.2015.

[27] Microchip Technology Inc. *Section 21. UART*, available at: http://ww1.microchip.com/downloads/en/DeviceDoc/39708B.pdf, Reference 7.4.2015.

[28] Renesas Electronics. *Introduction to CAN.* Application Note, available at: http://documentation.renesas.com/doc/products/mpumcu/apn/rej05b0804_m16cap.pdf, Referenced 7.4.2015.

[29] R. Kamal. *Embedded Systems. Chapter 3: Serial Bus Communication Protocols – USB.* McGraw-Hill Education, 2008.

[30] Texas Instruments. *Real-Time Clock (RTC),* November 2010, available at: http://www.ti.com/lit/ds/symlink/bq32000.pdf. Reference 7.4.2015.

[31] NXP Semiconductors. *Real-time clock/calendar,* Rev. 10 – 3 April 2012, available at: http://www.nxp.com/documents/data_sheet/PCF8563.pdf.

[32] D. Gilliam. *Temperature Sensors,* March 2003, available at: http://coecsl.ece.illinois.edu/ge423/sensorprojects/gilliam%20-%20temp%20sensors.pdf. Referenced 7.4.2015.