ISLAMIC UNIVERSITY OF TECHNOLOGY

REMOTE DATA ACQUISITION SYSTEM USING FPGA

By

Husnain Al Bustam (082405)

Nafiz Ur Rahman (082427)

Ferdous Ibna Idrish (082467)

Supervised by

Dr. Md. Fokhrul Islam

Co-Supervised by

Md. Shahzamal

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF SCIENCE IN ELECTRICAL AND ELECTRONIC ENGINEERING

AT

ISLAMIC UNIVERSITY OF TECHNOLOGY

DHAKA, BANGLADESH

12th SEPTEMBER, 2012

ISLAMIC UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF

ELECTRICAL AND ELECTRONIC ENGINEERING

The thesis entitled "**Remote Data Acquisition System Using FPGA**", by **Husnain Al Bustam, Nafiz Ur Rahman, Ferdous Ibna Idrish** has been accepted in partial fulfillment of the requirement for the degree of **Bachelor of Science in Electrical and Electronic Engineering** on 12.09.2012.

Approved by
Dr. Md. Fokhrul Islam

Dr. Md. Fokhrul Islam
Project Supervisor
Department of Electrical & Electronic Engineering (EEE)

Md. Shahzamal
Project Co-Supervisor
Bangladesh Atomic Energy Commission, Bangladesh.

Prof. Dr. Md. Shahid Ullah
Head of the department
Electrical and Electronic Engineering (EEE)

ISLAMIC UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF

ELECTRICAL AND ELECTRONIC ENGINEERING

Date: 12th September, 2012

It is hereby declared that neither this thesis nor any part thereof has been submitted elsewhere for the award of any degree or diploma.

Author 1	**Husnain Al Bustam**
Author 2	**Nafiz Ur Rahman**
Author 3	**Ferdous Ibna Idrish**
Supervisor	**Dr. Md. Fokhrul Islam**
Co Supervisor	**Md. Shahzamal**

TO OUR PARENTS

ABSTRACT

Single cost-efficient VLSI chip nowadays turns out to be a solution to microelectronic design problems. In these designs embedded system architectures are used which are comprised of software programmable components accompanied with dedicated hardware processing modules integrated into the VLSI chip [1]. Also emerging new designs which are based on heterogeneous embedded system architectures, offer flexible low-cost design in a short design cycle, integrating multiple software programmable components together with dedicated hardware components into a single cost-efficient VLSI chip. With the help of new technology it has now become possible not only to introduce programmability in these single chip architectures but also maintaining most of the advantages of customized VLSI solutions [1].Once some applications which were too difficult for computing hardware, now with impressive advances in Very Large Scale Integration (VLSI) technology are becoming more feasible[2]. Utilizing the prebuilt logic blocks and programmable routing resources engineers and designers now configure FPGAs (FPGA stands for Field Programmable Gate Array) to implement custom hardware. The configuration of FPGA is specified with the help of hardware description language. FPGA technology is now adopting among the engineers scientists at all levels of expertise as higher level tools evolve to deliver the fruits of reprogrammable silicon [2]. The "*Remote Data Acquisition System Using FPGA*" has been entirely implemented on the FPGA of the Spartan-3E Starter Kit. DAQs are complex systems which are the basis for building and monitoring tools that enable the supervision of local and remote systems. Data is transferred from the data loggers via radio telemetry, cellular, serial ports, Ethernet, satellite peripherals etc. The mobility and flexibility of the data loggers and sensors make the applications of this technology limitless. In everywhere we can see the magical touch of science and engineering in the present world which enables human being to have the maximum comfort and security of life. In that contrast our developed data acquisition system is such a device that could be used for biomedical applications, nuclear research, or data acquisition from such a place where the presence of human being is risky as well as harmful, for the example in the radioactive areas. The

developed system will communicate through the SMSC LAN83C185 Ethernet PHY with remote pc via internet. Of course the Ethernet port of the Spartan-3E starter kit has to be interfaced with the personal computers as well as that pc should be connected with the remote pc through internet. The configuration of the PCs used in this R and D work- Intel® Desktop Board DG41WV mother board which has LAN support Gigabit (10/100/1000 Mbits/sec) LAN subsystem using the Realtek* RTL8111D Gigabit Ethernet Controller and Intel® Core™2 Duo Processor. The data transfer rate of the Ethernet port of Spartan-3E starter kit is 10/100 Mbits/Sec. So our developed system could transfer data at the rate of 10/100 Mbits/Sec. National instrument DAQ interfaced with the ADC of the Spartan-3E starter kit to acquire analog signal from the outer world. The ADC process the digital data to FPGA, therefore those data is processed through the SMSC LAN83C185 Ethernet PHY to the personal computer of the concerned person who intended to acquire and send that data through internet in a remote place. The graphical control of the system has been developed using the LAB VIEW program.

ACKNOWLEDGEMENTS

We would like to thank Dr. Md. Fokhrul Islam, our supervisor, for his many suggestions and constant support during this research. We are also thankful to Md. Shahzamal, our co supervisor for his guidance in the theoretical aspects of the research when we were in chaos and confusion. Mr. Tnaveer Ahmed Bhuiyan, an Alumni of AIUB currently researching in University of Texas, Dallas, shared with us his knowledge of speech enhancement and provided many useful references and friendly encouragement. Of course, we are grateful to our parents for their patience and love. Without them this work would never have come into existence (literally).

Finally, we wish to thank the following: Mr. Mahbubul Hoq (Director-IE,AERE), Dr. Carlos *Leitão (*Researcher, GoLP is a research unit, University of Lisbon, Portugal) and all the other friends who gave us love and support.

The Authors

IUT,OIC, Bangladesh

September 2012

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENT....VII

LIST OF FIGURES....XII

LIST OF TABLES....XVI

LIST OF SYMBOLS AND ABBREVIATION....XVII

CHAPTERS

1. Introduction
   1.1. Background....02
   1.2. Motivation....02
   1.3. The Research Goal....03
2. Data Acquisition System (DAQ)
   2.1. Introduction....06
   2.2. Types of DAQ....08
      2.2.1. Wireless Data Acquisition Systems....09
      2.2.2. Serial Communication Data Acquisition Systems....09
      2.2.3. USB Data Acquisition Systems....09
      2.2.4. Ethernet Data Acquisition Systems....10
   2.3. Data Acquisition Plug-in Boards....10
   2.4. Switch Boxes....11
   2.5. Stand-alone Data Loggers....11
   2.6. Application of DAQs....12
      2.6.1. Wireless DAQ....13
      2.6.2. Ethernet DAQ....14
      2.6.3. USB DAQ....14
   2.7. Remote DAQ....15
      2.7.1. Available Remote DAQs....16
      2.7.2. Retriever....16

2.7.3. Wireless Modem Models……………………………………………………………………....16

3. CISC Architecture

3.1. Introduction………………………………………………………………………………………….20

3.2. History of CISC Architecture…………………………………………………………………20

3.3. Attributes of CISC Architecture……………………………………………………………...22

3.4. Recent Development of CISC Architecture…………………………………………….....24

3.5. Advantages of CISC Architecture…………………………………………………………..24

3.6. Disadvantages of CISC Architecture……………………………………………………....25

3.7. Application of CISC Architecture…………………………………………………………...26

4. FPGA

4.1. Introduction………………………………………………………………………………………….28

4.2. History……………………………………………………………………………………………......28

4.3. FPGA Structures……………………………………………………………………………………29

4.3.1. Basic Structure…………………………………………………………………………………..29

4.4. Configuring a FPGA………………………………………………………………………………..30

4.5. Advantages of FPGA……………………………………………………………………………….30

4.6. Disadvantages of FPGA…………………………………………………………………………...31

5. Hardware and Software

5.1. Introduction………………………………………………………………………………………….33

5.2. Hardware……………………………………………………………………………………………..33

5.2.1. Spartan-3E-starter kit………………………………………………………………………...33

5.3. USB TO RS232 SERIAL Adapter CABLE DB9 PIN PL2303…………………………….....42

5.3.1. USB TO RS232 SERIAL Adapter CABLE……………………………………………….42

5.4. Software

5.4.1. VHDL……………………………………………………………………………………………….43

5.4.2. Lab VIEW………………………………………………………………………………………….44

5.4.3. Xilinx 13.2 ISE…………………………………………………………………………………...45

6. Remote Data Acquisition System

6.1. Interfacing the Peripherals of Spartan-3E Starter Kit…………………………………....47

6.1.1. DAC………………………………………………………………………………………………….47

6.1.2. SPI Communication…………………………………………………………………………….48

6.1.3. Communication Protocol……48
6.1.4. Character LCD Screen……50
6.1.5. Rotary Push-Button Switch……50
6.2. Working Principle……52
6.3. ADC……56
6.4. Programmable Pre-Amplifier……58
6.4.1. Programmable Gain……58
6.4.2. Interface……59
6.4.3. SPI Control Interface……60
6.5. Analog to Digital Converter (ADC) ……61
6.5.1. Interface……61
6.5.2. SPI Control Interface……61
6.6. Interfacing of the Ethernet Controller……62
6.6.1. Design Strategy……62
6.6.2. Open Core Ethernet Controller……63
6.6.3. Design……63
6.6.4. Host……65
6.7. Remote Data Acquisition System……66
7. Result Analysis and Discussion
7.1. Result Analysis……72
7.1.1. DAC Implementation on FPGA……72
7.1.2. Tables and Figures……72
7.1.3. Simulation Results……76
7.2. Measure Light Intensity using Photodiode……78
7.2.1. Photodiode BPW34 as light sensor……78
7.2.2. Build a Light Intensity Logger……79
7.3. Measure Distance using the Ultrasonic Sensor……80
7.4. Temperature Simulation and Calibration……82
7.5. Optical Heart Rate Monitor……84
7.6. Discussion
7.6.1. DAC……86

7.6.2. DAQs……………………………………………………………………………………………86

REFERENCES……………………………………………………………………………………………87
APPENDIX-I (XILINX-13.2 ISE TUTORIAL) ……………………………………………………………94
APPENDIX-II (VHDL TUTORIAL) ………………………………………………………………………121
APPENDIX-III (CODES) ………………………………………………………………………………134

LIST OF FIGURES

Figure-2.1. Components of data acquisition system……………………………………………………..07

Figure-3.1. Intel 8086 Architecture, the 1st member of x86 family……………………………….23

Figure-4.1. Basic structure of FPGA…………………………………………………………………………..30

Figure-5.1. Spartan-3E Starter Kit Board-.............…………………………………………………………36

Figure-5.2. Spartan-3E Starter Kit Board-.………………………………………………………………….36

Figure-5.3. Standard USB Type A/Type B Cable…………………………………………………………..37

Figure-5.4. Connect the USB Type B Connector to the Starter Kit Board Connector………....37

Figure-5.5. 10/100 Ethernet PHY with RJ-45 Connector……………………………………………….38

Figure-5.6. Character LCD Interface…………………………………………………………………………....38

Figure-5.7. Four Slide Switches and Eight LEDs……………………………………………………………..39

Figure-5.8. Push-Button Switches Require Internal Pull-up Resistor in FPGA Input Pin……39

Figure-5.9. RS-232 Serial Ports……………………………………………………………………………………..40

Figure-5.10. Digital-to-Analog Converter and Associated Header………………………………...40

Figure-5.11. Analog to Digital Converter (Two-Channel Analog Capture Circuit)……………..41

Figure-5.12. Connections to Intel StrataFlash Flash Memory………………………………………..41

Figure-5.13. FPGA Interface to Micron 512 Mbit DDR SDRAM……………………………………...42

Figure-5.14._USB TO RS232 Serial cable adapter…………………………………………………….....43

Figure-6.1. DAC connection schematics. ……………………………………………………………………...47

Figure-6.2. SPI communication waveform. ………………………………………………………………….48

Figure-6.3. SPI communication protocol………………………………………………………………………..49

Figure-6.4. Character LCD interface………………………………………………………………………………50

Figure-6.5. Rotary Switch Shaft Encoder Circuitry…………………………………………………………51

Figure-6.6. Output from Rotary Shaft Encoder Circuitry…………………………………………….....52

Figure-6.7. Block Diagram of the Developed System……………………………………………………....53

Figure-6.8. Block diagram of the system (Software) ……………………………………………………54

Figure-6.9. State diagram of the DAC……………………………………………………………………………..55

Figure-6.10. State diagram of the dis_value component………………………………………………..56

Figure-6.11. Two-Channel Analog Capture Circuit……57
Figure-6.12. Detailed View of Analog Capture Circuit……58
Figure-6.13. SPI Serial Interface to Amplifier……60
Figure-6.14. SPI Timing When Communicating with Amplifier……60
Figure-6.15. Analog-to-Digital Conversion Interface……62
Figure-6.16. Detailed SPI Timing to ADC……62
Figure-6.17. Design Diagram……64
Figure-6.18. Remote High Speed Data Acquisition System……67
Figure-6.19. Network Architecture……68
Figure-6.20. Simulation Using the CISCO Packet Tracer-5.3……70
Figure-7.1. Experimental Setup……72
Figure-7.2. For 100mV……74
Figure-7.3. For 10mV……75
Figure-7.4. Simulation……76
Figure-7.5. Behavioral……76
Figure-7.6. RTL Schematic……77
Figure-7.7. Design Summary……77
Figure-7.8. Photodiode BPW34……79
Figure-7.9. Develop a LabVIEW VI that reads the voltage across the resistor and converts the voltage to lux. ……79
Figure-7.10. Front Panel View (Light Intensity Logger) ……80
Figure-7.11. Front View (Ultrasonic Sensor) ……81
Figure-7.12. Block Diagram (Ultrasonic Sensor) ……82
Figure-7.13. Front Panel View (Temperature) ……83
Figure-7.14. Block Diagram (Temperature) ……83
Figure-7.15. Optical heart rate monitor schematic……84
Figure-7.16. IR emitter and detector encased in Velcro……85
Figure-A1. ISE Project Navigator (0.61xd) ……88
Figure-A2. Project Navigator Window……89
Figure-A3. Creating New Project……90
Figure-A4. Creating New Project (Cont.) ……90

Figure-A5. Creating New Project (Cont.) 91
Figure-A6. Creating New Project (Completed) 92
Figure-A7. Creating New Source...92
Figure-A8.New Source Wizard...93
Figure-A9.New Source Wizard (Continue) 93
Figure-A10. Project Navigator Window for the Project "alarm" 94
Figure-A11. Checking Syntax (Successful) 97
Figure-A12. New Source Wizard (Test Bench) 97
Figure-A13. New Source Wizard (Test Bench) 98
Figure-A14. Design (Simulation) 100
Figure-A15. Simulation...101
Figure-A16. Isim Window..101
Figure-A17. Expanded User Constraints...102
Figure-A18. Unconstrained Clocks..102
Figure-A19. Profile Generated for Clock "CLK" 103
Figure-A20. Profile Generated for Clock "REMOTE" 103
Figure-A21. Constrained Clocks...104
Figure-A22. Profile Generated After "I/O Planning (Plan Ahead) Post Synthesis"..............105
Figure-A23. Planning I/O Ports constraints.. 105
Figure-A24. ISE iMPACT Window..106
Figure-A25. ISE Design FlowFigure-A26. Process Window.......................................107
Figure-A26. Process Window...108
Figure-A27. Set RTL/Tech Viewer Startup Mode...108
Figure-A28. "Create RTL Schematic"-Wizard..109
Figure-A29. "Create RTL Schematic"-Wizard (Continue) 109
Figure-A30. RTL Schematic..110
Figure-A31. Detail View of RTL Schematic..110
Figure-A32. Set RTL/Tech Viewer Startup Mode..111
Figure-A33. "Create Technology Schematic"-Wizard..111
Figure-A34. "Create Technology Schematic"-Wizard (Continue) 112
Figure-A35. Technology Schematic...112

Figure-A36. View of Technology Schematic (Double) ………………………………………………..113
Figure-B1. DFF with Asynchronous Reset……………………………………………………………..115
Figure-B2. When rst= ‘0’, then q<=d…………………………………………………………………….118
Figure-B3. When rst = ‘1’, then q <= ‘0’………………………………………………………………….119
Figure-B4. Counter of Example-2…………………………………………………………………………119
Figure-B5. Simulation Result…………………………………… ……………………………………………122
Figure-B6. 4-input Multiplexer………………………………………………………………………….....123
Figure-B7. Y <= A WHEN SEL = “00” ………………………………………………………………………125
Figure-B8. Y <= A WHEN SEL = “01” ………………………………………………………………………125
Figure-B9. Y <= A WHEN SEL = “10” ………………………………………………………………………126
Figure-B10. Y <= A WHEN SEL = “11” ……………………………………………………………….....126

LIST OF TABLES

Table-1. Programmable Gain Settings for Pre-Amplifier……………………………………..……….59

Table-2. AMP Interface Signals…………………………………………………………………………59

Table-3. ADC INTERFACE SIGNALS……………………………………………………………......61

Table.4. Network Design……………………………………………………………………………….69

TABLE -5. FOR 100mV Step……………………………..………………………………………….72

TABLE -6. FOR 10mV Step…………………………………………………………………………..74

LIST OF SYMBOLS AND ABBREVIATION

ADC	Analog to Digital Converter
ASIC	Application Specific Integrated Circuits.
CISC	Complex Instruction Set Computer.
CLBs	Configurable Logic Blocks
CPI	Clock Per Instruction
CPLD	Complex Programmable Logic Device
DAC	Digital to Analog Converter
DAQ/DAS	Data Acquisition System
DDS	Direct Digital Synthesis
DDR SDRAM	Double Data Rate Synchronous Dynamic Random Access Memory
DSP	Digital Signal Processing
EEPROM	Electrically Erasable Programmable Read-Only Memory
FSM	Finite State Machine
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
ICON	Integrated Controller
IEEE	Institute of Electrical and Electronic Engineers
ILA	Integrated Logic Analyzer
IP Core	Intellectual Property Core
ISE	Integrated Software Environment
Lab VIEW	Libratory Virtual Instrument Workbench.
LAN	Local Area Network
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LUT	Lookup Table

PLD	Programmable Logic Devices
RAM	Random Access Memory
ROM	Read-Only Memory
RTL	Resistor-Transistor logic
SPI	Serial Peripheral Interface
SPLDs	Simple PLDs
SRAM	Static Random Access Memory
UCF	User Constraints File
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VIO	Virtual Input and Output

CHAPTER 1

INTRODUCTION

1.1. Background

Digital system which is also known by other name such as logic design, digital logic, switching circuits etc. can process only digital signal dealing with only a limited numbers of discreet values [3]. Digital system is characterized by its manipulation of discreet elements of information such as electric impulses, the decimal digits, arithmetic operation, punctuation marks or any other set of meaningful symbols [4]. The subject matter of digital design is the design of digital electronic circuits such as logic gates, flip-flops, shift registers, counters etc. The main application areas of digital design are the digital computers, control systems, data communications and many other applications that require electronic digital hardware. The HDL (Hardware Description Language) is used to describe the behavior of a digital system. As part of course work in "Digital Logic Design" we were given idea about the VHDL (A Hardware Description Language) that raises our interest to work in the field of digital system design on FPGA. While doing industrial training in Institute of Electronics, Bangladesh Atomic Energy Commission we came to know about the FPGA and how to program it. At the end of year, 2011 there was short course on "Digital System Designing on FPGA" in Bangladesh Atomic Energy Commission where for the first time we were introduced with the "Spartan-3E Starter Kit". Our first task was to interface the peripherals such as ADC, DAC, LCD, Rotary Switch, and Push-Button Switch of the Spartan-3E Starter Kit. After that we started thinking of a project and according to the advice of our supervisor and co-supervisor we undertook a project which is almost similar to this project that is, "Data Acquisition System Using FPGA". The main difference between this project and that DAQ is that medium of communication with Personal Computer (PC). This project concerned about the communication using the SMSC LAN 83C185 Ethernet PHY of Spartan-3E Starter Kit while that DAQ communicates via RS-232 Serial Ports.

1.2. Motivation

The Nuclear Electronics Lab, Institute of Electronics (IE), AERE (Atomic Energy Research Establishment) is responsible for developing all electronic equipment for nuclear research. Nuclear research areas are always health threatening in some extents. So in that kind of environment for effective research if we use a data acquisition system it would be worthwhile. This is the primary motivation of our. Besides there are several projects are going on in Energy Institute, AERE which requires data acquisition remotely. There are facilities of Cancer Therapy and many radioactive tests for detection of diseases in AERE which most often require data acquisition. Recently IE has established a VLSI lab which is very first time in Bangladesh opens the new horizon in microelectronic research. So considering all these stated circumstances we have taken this as our undergraduate project. But the main challenge of this work is that we have make the system portable as well as the data communication should be fast. Again Bangladesh is third world country so we think that if we could make data acquisition system locally that will be beneficial for the people of country as well as huge amount money could be saved that we usually spend buying DAQs from outside.

1.3. The Research Goal

The demand of portable, flexible and high speed data communication DAQs are growing rapidly. In this era of science and engineering data transfer rate or how fast the data communication is the main concern of researchers and scientists. We have split up this project in to three parts. In the first part we have interfaced the necessary peripherals; in the second step we have implemented such a DQA that could exchange data at a rate 1 Mbits/Sec through RS232 serial ports of “Spartan-3E Starter Kit”. But as we have stated earlier speed is the crucial issue, therefore there is Ethernet port in “Spartan-3E Starter Kit” over which data could be transferred at a rate 10/100 Mbits/Sec. Which is much faster than the earlier one. So in third steps we have implemented a DAQ which could transfer data using Ethernet port. The working principle of both the DAQs are same but the difference between them is the modes of communications. We have interfaced the peripherals and RS232 serial ports with FPGA embedded in “Spartan-3E Starter Kit” using the VHDL language for both cases. The control of the system have been implemented using the LabVIEW program. Our implemented DAQs are the discretised systems in a sense as both of these have to acquire data from outer world in terms of analogue signal using the NI-DAQ which is processed to FPGA using the ADC embedded in starter kit. This kind of system could be on research level but the main concern of engineering design is the flexibility and sustainability of the products, moreover should be user friendly in nature. We can easily understand this fact why the Windows OS has the largest number of desktop users, this because of user friendliness. So our future goal regarding this project to make “Single Chip Solution” which will be flexible, reliable and user friendly in nature. As we have mentioned earlier, in AERE a VLSI lab is established recently with modern equipments. So our main target is to reduce the size of this DAQ into a single cost-effective VLSI chip.

CHAPTER 2

DATA ACQUISITION SYSTEM (DAQ)

2.1. Introduction

Data acquisition system (DAQ) as the name implies is defined as the products or processes of collecting data or information on an electrical phenomenon such as voltage, current or physical phenomenon like pressure, temperature, sound into a computer for storage, processing, analysis, inspection or further data manipulation. With the advancement of technology, the type of data acquisition system has been simplified and made more accurate, versatile, and reliable through electronic equipment like PC, sensors, transducers, filters etc. Data acquisition systems equipments vary with the requirement of the specific field of its utilization. Basic equipments are

- Transducer
- Amplifier
- Active filter
- Analog multiplexer
- Sample hold
- Programmer sequencer
- A/D converter
- Computer

The input to the system is an electrical or physical parameter such as temperature, pressure, flow, acceleration, and position, which are analog quantities. If the input is physical parameter then it is first converted into an electrical signal by means of a transducer; further processing or manipulation of data is done by electrical circuits. Electrical signal does not need to be converted again with the transducers. Using an amplifier the output signal of transducer is magnified. This output signal of transducer is very small say in microvolt or millivolt level, magnified up to 1v or 10v level using the amplifier. In addition, the transducer output signal may be comprised of a signal with high impedance, a differential signal along with common-mode noise, a current signal, a signal superimposed on a high voltage, or a combination of the signals stated above. Several specialized types of amplifier can be used for the amplification of the low level signals to the high level signals for desired operation [6,11].

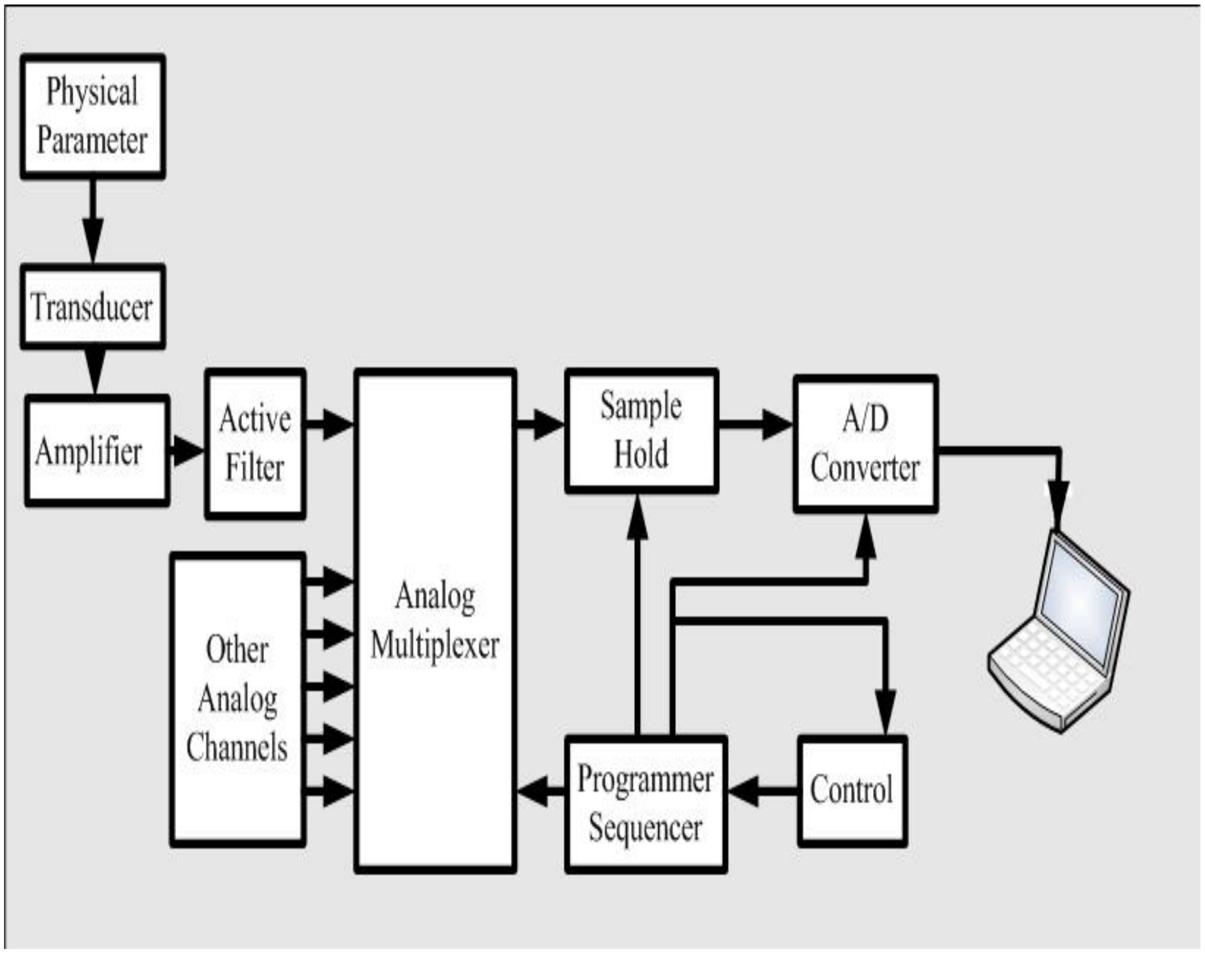


Figure-2.1. Components of data acquisition system[5]

The magnified output is then transmitted to the low-pass active filter which get rid of the unwanted high frequency component or electrical component which give rise to the noise. Sometimes the high level output signal from the amplifier undergoes the operation of squiring, multiplication, log conversion, RMS conversion, division, linearization etc. with the help of special nonlinear analog function circuit. The processed analog signal is then followed by a analog multiplexer. What it does is to switch sequentially among a number of different analog input channel. The connection between each input and output for a specified time is controlled by the multiplexer switch. When the output is connected to the input in the meantime the sample hold circuit acquires the signal voltage level and the A/D converter becomes operational, converting the analog values into digital values. The digitized values next transmitted to the data bus of the computer or used as a input the digital circuit. Time sharing is occurred with the A/D converter along with the analog multiplexer and sample hold circuit among a number of input channels. The timing and control of the complete DAQ is done by a digital circuit called a programmer sequencer, which in turn is under the control of the computer. The computer itself also used to control the overall data acquisition system sometimes. This is perhaps the most commonly used DAQ configuration, there are also other ones. Instead of multiplexing high-level signals, low-level multiplexing is sometimes used with the amplifier following the multiplexer. In such cases, just one amplifier is required, but its gain may have to be changed from one channel to the next another during multiplexing. Another method is to amplify and convert the signal into digital form at the transducer location and then send the digital information in serial form to the computer. In that case the parallel form of digital data is acquired by suitable conversion and then in the computer data bus these data are multiplexed[5,9,13]

2.2. Types of DAQ

Now-a-days there are various types of data acquisition system available. Various novel technology is emerging, demands data acquisition systems capable of pleasant control, application of these technologies. Though each data acquisition system is a bit different from another due to its desired application but the main components of the data acquisition systems remain same. Some of the data acquisition systems are

- Wireless Data Acquisition Systems
- Serial Communication Data Acquisition Systems
- USB Data Acquisition Systems
- Ethernet Data Acquisition Systems
- Data Acquisition Plug-in Boards
- Switch Boxes

- Stand-alone Data Loggers

2.2.1. Wireless Data Acquisition Systems

Wireless data acquisition systems can be used to eliminate the costly and time consuming installation of cable runs for wiring of sensors and other instruments. These system consist of one or more wireless transmitters which sends data back to a wireless receiver connected to a remote computer. Now-a-days for the measurement of ambient temperature and relative humidity, barometric pressure, line pressure, infrared temperature, thermocouples, RTDs, pH, pulse output sensors and wireless transmitters are available. Also transmitters ranging from 4 to 20 mA and voltage output transducers are available from different brand in the market. Receivers which are to be connected to the USB or Ethernet port of the PC are also available [7].

2.2.2. Serial Communication Data Acquisition Systems

Serial communication data acquisition systems are a good choice when the measurement needs to be made at a location which is remote from the computer. There are several different communication standards, RS232 is the most common but it only supports point to point communication and relatively short distances. RS485 supports transmission distances to 1500 meters with a single or 2-pair cable and also allows up to 32 devices to share the same bus[7].

2.2.3. USB Data Acquisition Systems

The Universal Serial Bus (USB) is a new measure for connecting PCs to peripheral devices such as printers, monitors, modems, pen drives and data acquisition devices. USB offers several advantages over conventional serial and parallel connections, which include higher bandwidth (up to 12 MBits/s) and the ability to provide power to the peripheral device. For data acquisition application USB based DAQs are ideal. Since USB connections supply power, only one cable is required to establish the link between the data acquisition device to the PC, which most likely has at least one USB port[7,9].

2.2.4. Ethernet Data Acquisition Systems

Ethernet based data acquisition systems are now-a-days a popular option for many users due to its specific advantages. Most industrial and commercial premises have Ethernet network cables installed, which allow a distributed data acquisition system to be integrated without the cost of additional wiring. One of the special kind of Ethernet devices employs a built in web server in which they transmit their acquired data for further processing or manipulation. This in turns provide with the facility that the user can check the system condition without installing any software just by simply browsing the web server with a standard browser. Even the data can be checked through the mobile or smart phones as almost all the smart phone has a built in standard browser supporting HTML5. So it offers flexibility and feasibility with control. Another advantage of using Ethernet is that the data can easily be shared among users on the local network and also via the Internet to authorized users around the world [7,9,11].

2.3. Data Acquisition Plug-in Boards

These kind of data acquisition boards can be plugged directly into the computer data bus. Advantages of using boards are they offer speed (because they are connected directly to the bus) and lower cost (because the overhead of packaging and power is provided by the computer). Generally for IBM pc and available computers of modern days boards are offered. Features provided by the cards may vary due to number and type of inputs (voltage, thermocouple, digital), outputs, speed and other functions provided. Each board installed, is addressed at a unique Input/Output map location in the computer. The I/O map in the computer provides the address locations which are used by the processor to gain access to the specific device as required by its program[13].

2.4. Switch Boxes

Due to various design verification and product testing applications, switch boxes are included as a type of data acquisition system. Mainly a switch box is a intermediary between the testing device and other instrumentation such as oscilloscope, counter, digital multimeters, power supplies etc. They route the signals between device and instrumentations. Switch boxes are available that can switch signal levels from a few micro volts to several hundred volts, and from DC to several gigahertz. In addition to basic switching, some switchboxes add simple control measures. For example, some manufacturers have added digital input/output capabilities, analogue output control, and isolated actuators for controlling high-power devices.

2.5. Stand-alone Data Loggers

Data loggers are initially used to monitor signals over a period of time in order to identify if there are irregularities that may require attention. Most data loggers also provide with a way to graph and analyze the data, which is further collected through a PC connection. Although it is used primarily in the up-front design verification stage of product development, data loggers also are used in-house for environmental chamber monitoring, component inspection, bench top testing, and process trouble-shooting. Since they are typically used in single-instrument applications, data loggers also make great portable field-testing instruments. They are some data loggers which are capable of performing mathematical operation on the acquired data and to compare the data with the appropriate limit defined by the user for suitable control operation. For instance, in case of measurement of temperature before using the acquired data, signal conditioning or linearization must be applied. The data loggers capable of performing such operation mentioned above can be a replacement of computer sometimes. To transfer the measured data in a PC most data loggers have a communication interface like GPIB, USB or LAN. Some of the applications require the acquired data to be downloaded on the PC in that case the continuous monitoring is needed which is not feasible. Because in that case the stand-alone benefit of data loggers will be lost. [8] If the data logger has its own internal data storage capability, or access to an external storage device such as a disk drive, then one can download the measurements for analysis at a later time. For general application a data logger with 20 channel or fewer with a low scan rate is decent. For greater flexibility and functionality, one should select a data logger that can operate as a standalone instrument, can be easily upgraded, and can be connected to a PC. A data logger should also have plugin slots and the ability to measure different types of input signals without external signal conditioning[9].

2.6. Application of DAQs

Applications of DAQs are very wide spread. Starting from small control system for car, DAQs are widely used for communication purpose, controlling sophisticated research system, telemetry, space technology, radar and positioning system, robotics, life science, many fields of engineering like electrical, mechanical, civil, computer science, geological etc. Some applications are discussed here.

2.6.1. Wireless DAQ

Wireless data acquisition systems are basically used when the user or monitoring system is distant from the environment form where data is gathered. It's also used to avoid extra cost and complexity for wiring. Here the wireless data acquisition system for testing rocket engines is discussed. This is a prototype wireless data-acquisition system which is developed for a potential replacement of a wired data-acquisition system to be used in testing rocket engines. The traditional use of wires to connect sensors, signal-conditioning circuits, and data acquisition circuitry is time-consuming and prone to error, especially when there are many sensors used in a test. In this system there is one master node with several slave nodes. Communication with the master node via pc is done by an Ethernet connection. Rechargeable batteries supply power to the slave nodes. These batteries are enclosed in such an enclosure so that the changing weather will not affect them. A time modulated ultra-wide-band radio transceiver is used in the master unit and also in each of the slave unit. What it does is by means of transmitting extremely low power of pulses with narrow width, it spread its RF energy over several Giga-hertz. Out of six sensors connected to each slave nodes two of them are connected directly to ADC and the other four is connected to ADC via signal conditioner indirectly. Each sensor has the maximum sampling rate of 5 kHz for streaming data. To accommodate the data collecting from the sensors in the five slave node the bandwidth of one channel of the communication system should be adequate. Comparing traditional sinusoidal wave based radios TM-UWB radios offer much higher spatial capacity. Also with existing wireless transmission TM-UWB radios works fine without any interference[15,16]. For this system, approximately 50ft (15m) is the maximum radio communication range between the master and the slave node. Small dipole antennas are used in this case. By using larger antenna or greater transmission power the range can be increased. Batteries for the slave nodes are used in such a way so that they can serve the desired operation. Operational lifetime can be increased by using extra batteries. Power consumption is controlled by radio transceiver.

The software performs the following operations:

- Controlling the operating schedule of the slave nodes.
- In order to maximize the utilization of the system to manage the sampling rate and latencies of the readings.
- To ensure synchronization among all operational nodes.[17]

2.6.2. Ethernet DAQ

Ethernet data acquisition systems are used widely now-a-days for industrial, commercial and may other purposes. Here an Ethernet IO data acquisition system is discussed. The industrial Ethernet I/O data acquisition systems discussed here is developed by Intelligent Instrument. They are using open architecture standards for easy development and deployment into existing and future systems. By using the efficiencies of standard technology, it offers a lower total cost of ownership for monitoring and control applications. With the help Ethernet Data Acquisition System EDAS CE, one can take advantage of the existing network infrastructure, lower cost network components. Also the development of networking and application is possible. Isolated digital and analog I/O, 100BaseT Ethernet, Windows CE development tools, all in an industrial modular package, make EDAS CE the ideal solution for a wide range of monitoring and control applications[9].

2.6.3. USB DAQ

USB Data Acquisition Systems are used both for remote purpose as well as stand-alone purpose. There are many type of USB Data Acquisition System available. One is NI USB-6009 [6].Its applications are

- Robot Specific Uses

- Loop refinement of PID
- Measurement of frame strain
- Diagnostics of electrical system
- Simulation of sensors

- Non-Robot Specific Uses
    - Dynamometer
    - Ball velocity measurement
    - Understanding FRC Electronics

2.7. Remote DAQ

Remote data acquisition is defined as the collection of information about a system or a process which located in a distant place. The data which is collected is used to study various parameters of a desired system and the implications of it would have on the dependent processes/systems. The systems that enable supervision of remote processes for data collection are termed remote data acquisition or remote data collection systems. These systems are usually designed using PCs and other processor-based input/output modules conforming to RS-232 and RS-485 standards. The advantages offered by the remote data acquisition system equipped with data loggers are:

- Installation is easy
- Ownership cost is low
- Reliability and flexibility
- Lowered risk of electrical faults, lightening surges and other hazardous weather conditions

In case of engineering processing the remote data collection plays a crucial role. To improve the efficiency, performance, reliability of the engineering project manipulation of data is necessary which is provided by the remote data acquisition system.

2.7.1. Available Remote DAQs

Many remote data acquisition system are available depending on the application demand. Some of them are based on internet while the others use wireless sensors to acquire data for further processing or manipulation. Some available data acquisition systems are briefly discussed here.

2.7.2. Retriever

It is a solar-powered, wireless data collection module which is used for landfill, remediation, and other sites. The Retriever is not just a simple hardware, but it facilitates a data delivery and reporting system which delivers data of pump flow and well level. These data can be clearly charted and displayed, and securely archived. From the sensors deployed in water flow and level the built in microprocessor take the data reading in an hourly basis and the acquired data is stored. The daily data acquired can be sent to a preferred web address. The received data can be expressed in tabular format or any other format according to the choice of the user. The data is properly backed up so that one can download the necessary data when need arises. Using a standard web browser one can access the web site containing the data which can be modified according to the corresponding date and time. So without being present in the site it is possible to produce a short summary of operation of the site[9,16,17]

2.7.3. Wireless Modem Models

These types of units provide rugged and reliable operation with sophisticated features which make them versatile and easy to use for a wide variety of applications. Their size is compact and easy operation make these units perfect for innumerable mobile/remote data applications. There applications are-

Mobile and Fleet Communications

- Automatic Vehicle Location (AVL)
- Differential GPS Navigation
- Computer Aided Dispatch
- Data Base Access from remote places
- Monitoring and Control of traffic

GPS

- Automatic Vehicle Location (AVL)
- Differential GPS navigation
- Differential and RTK survey corrections

Industrial Data Communications

- Monitoring oil and gas Field
- Management of water and waste water
- Control of irrigation
- Control and monitoring of environment
- Security/Alarm System Management
- Automation of factory.

Two type of module are available from Teledesign Systems. These are TS4000 Radio Modem and TS2000 Mobile Radio Modem. Also products from various brands are available meeting the demand of different types of consumers. [8]

CHAPTER 3

CISC ARCHITECTURE

3. 1. Introduction

CISC architecture referred to a computer architecture in which hundreds of operation can be performed by CPU (Central Processing Unit). CISC stands for complex instruction set computing. It referred to a computer where a single instruction can execute various low level operation or capable of multistage operation or addressing modes within single instructions[18]. This is not similar compared to Reduced Instruction Set Computers (RISC), which support fewer instructions. The computers that support CISC, are capable of accomplishing a wide variety of computing tasks, which makes them the priority choice for the general purpose computer. That's why the majority of personal computer in the market is using this architecture. Early in the 1970s and 1980s, the invention of CISC architecture started a new era where the computer became capable of performing complex operation. As a result it gave birth of complex or a bit sophisticated computer program writing. Or it can be said that instead of writing lengthy program this instruction set provided with the opportunity of writing small program for the same tasks.

3.2. History of CISC Architecture

As with the passage of time high-level languages started to become popular, so the computer architects gave a try to make the machine's capabilities match constructs used by programmers. With the help of complex addressing mode reduction of many instructions to single machine instruction become possible. Memory in the past was relatively slow and expensive so, compacting the program size was one of the desirable goals. Faster loading and more available space for other programs are the two advantages offered by the small program size. For the production of memory chips Intel Corporation was founded by Robert Noyce, Gordon Moore and Arthur Rock in 1968. Robert Noyce, invented the integrated silicon circuit. Intel sold the amount of chips that was worth about $3000 in their first year of operation. Calculators were a bit large electromechanical machines weighing of 20 kg in late 1960s. For a proposed electronic calculator, a Japanese company named Busicom in September 1969, proposed Intel to manufacture 12 custom chips. Ted Hoff the assigned engineer for this project looked this plan from a different viewpoint. He came up with a idea that if he use a 4-bit general purpose CPU on a single chip than the same thing can be done more simply which will also be cheap as well. As a result the 2300-transistor 4004 was born as a first single chip CPU (Faggin et al., 1996). Though Intel had no future plan at the beginning of this project but later on Intel came up with that it would be worthwhile if they use the 4004 in other projects. So Intel bought all the rights of the chip from Busicom at a cost of $60000. After some research in 1972 the 8-bit version of the chip, the 8008 was introduced. At first Intel started to produce the 8008 at low scale as they are not sure that it will arouse the interest. But this chip was able to arouse a large interest among users so Intel went for the design of a new chip which has the memory space of around 16k like the 8008's. So in 1974, a small general purpose CPU, 8080 is emerged. Within a very short time this product took the market like PDP-8. Intel made a good profit by selling these chips in millions. 8086, a more generalized 16-bit CPU was introduced in 1978. Though the design of 8086 is almost similar to the 8080 but they differ in their compatibility. The next member of this family followed by 8086 was 8088 which has the same architecture as 8086. But instead of having 16 bit bus it had only 8-bit bus which made it not only slower but also cheaper than 8086. This chip became the standard for personal computer industry when IBM chose 8088 as the CPU for IBM pc. Limits for addressing memory for both 8086 and 8088 was 1 megabyte. In the subsequent days it becomes a problem so Intel designed a new chip named 80286. The memory organization of 80286 is different from the 8088 and 8086 but the basic instructions were same. In the midrange PS/2 models and in the IBM PC/AT, 80286 had specific application. 80286 was a great success as people accepted it due to its almost like 8088 but faster than 8088. The quest for a 32-bit pc on a single chip became fruitful in 1985 when Intel introduced 80386. This is also more or less compatible with the earlier versions. This backward compatibility was desirable for the users who are running old software but at the same time the users who are enthusiastic for future change had to hold back for some time. 80486 was introduced after four years. The main feature of 80486 was a floating point unit, 8k of cache memory on the chip and obviously it was faster than the previous version. The function of cache memory is to avoid the slow accesses in the main memory by holding memory words which are most commonly used inside or close to CPU. 80486 also provided with the opportunity to build system with several CPUs. By the time Intel lost a trademark infringement lawsuit that the numbers cannot be trademarked like 80486 so they give a new name “Pentium” to the next generation. Pentium chips were faster than the 80486 as Pentium has two internal pipeline compared to one internal pipeline of 80486. As a name Pentium became more popular among the marketing people and the new chip was introduced named as Pentium Pro. Though it was almost same as the previous one but it brought a new change. Pentium Pro was able to execute five instructions at a time and it had a huge change in the internal organization. An important feature of Pentium pro was the introduction of the two-level cache memory. For commonly used instructions the processor had its own 8k memory and for holding commonly used data there was an additional 8k memory. There was a second cache memory of 256k in the same package. Pentium II was introduced followed by Pentium pro which was almost similar to the previous version except it has a multimedia extension which is called MMX. In case of audio and video processing this instructions speeded up the computations so there was no need of an additional multimedia coprocessor. As this is a new feature added by Pentium II so it can be called a new version of Pentium Pro with multimedia instructions [10]. PDP-11, System/360, VAX, Motorola 68000 family, and Intel x86 architecture based processors; all the mentioned microprocessors are CISC architecture based. With the passage of time processors like Pentium III, Pentium IV, Dual Core, Core 2 Duo, Quad Core was introduced by Intel. Also Many processor based on CISC architecture is available in the market[18,21,26,27].

3.3. Attributes of CISC Architecture [20,21,24]

Due to the development of CISC architecture there are some attributes which are more or less common among the CISC based microprocessors. These are discussed below:

- Instructions have source and destination following a 2-operand format. Commands available from the register to register, from memory to register and from register to memory. Indexing of arrays requires multiple addressing modes for memory including specialized modes.
- Variable length instructions where the length often varies according to the addressing mode
- Instructions which require multiple clock cycles to execute.

Other common characteristics among CISC architectures are:

- Complex instruction-decoding logic, driven by the need for a single instruction to support multiple addressing modes.
- A small number of general purpose registers. This results from the directly operating instruction on memory. Also for instruction coding, execution and storage of the code the absence of the required amount of chip is responsible.
- Several special purpose registers. Many CTSC designs set aside special registers for the stack pointer, interrupt handling, and so on. By making the instruction set more complex this can simplify the hardware design.
- A 'Condition code" register which is set as a side-effect of most instructions. The function of this register is to check the last operation and to store or record if there is any error condition.

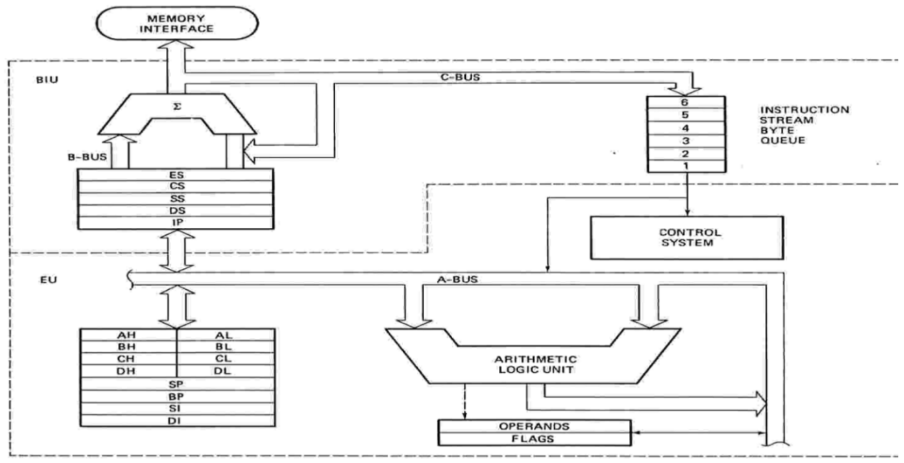


Figure3.1. Intel 8086 Architecture, the 1st member of x86 family [24]

3.4. Recent Development of CISC Architecture

As advancement of science and technology is ongoing, likewise the new era is starting to emerge in this field. New ideas and technological approach made it possible. There are various development ongoing in CISC architecture. These are –

- The terms RISC and CISC have become less meaningful with the continued evolution of both CISC and RISC designs and implementations.
- CISC chips are now able to execute more than one instruction within a single clock. Pipelining is possible due to this and accommodation of many transistors on a single chip is possible.
- Modern x86 processors also decode and split more complex instructions into a series of smaller internal "micro-operations" which can thereby be executed in a pipelined (parallel) fashion, thus achieving high performance.
- Attempts have been made to combine features of both RISC and CISC to develop a new approach
- Intel has teamed up with Hewlett-Packard (HP) to design a new type of ISA. This is known as IA-64 (Intel Architecture 64).

3.5. Advantages of CISC Architecture

There are many advantages of CISC architecture. Some are mentioned below.

- CISC has varying lengths to reduce wasted space in memory.
- Has developed a process to manage power which adjusts clock speed and voltage.
- Uses less instructions to perform similar instructions than RISC.
- Provides programmers with assembly instructions to do a lot with smaller programs.

- Microprogramming is as easy as assembly language to implement, and much less expensive than hardwiring a control unit.
- The ease of micro-coding new instructions allowed designers to make CISC machines upwardly compatible: a new computer could run the same programs as earlier computers because the new computer would include the instruction of the earlier computers as a superset.
- As each instruction became more capable, fewer instructions could be used to implement a given task. So efficient use of slow main memory is possible.
- Because micro-program instruction sets can be written to match the constructs of high-level languages, the compiler does not have to be as complicated.

3.6. Disadvantages of CISC Architecture

Though CISC architecture offers a lot of advantages, it also hyas some disadvantages. These are discussed below.

- CISC chips are relatively slow (compared to RISC chips) per instruction.
- CISC chips require many more transistors than comparable RISC designs.
- Harder to pipeline using CISC architecture.
- Expensive to produce.
- Earlier generations of a processor family generally were contained as a subset in every new version - so instruction set & chip hardware become more complex with each generation of computers.
- So that as many instructions as possible could be stored in memory with the least possible wasted space, individual instructions could be of almost any length - this means for execution different instructions will need different clock time as a result the performance of the machine will be slower.

- Many specialized instructions aren't used frequently enough to justify their existence -approximately 20% of the available instructions are used in a typical program.
- CISC instructions typically set the condition codes as a side effect of the instruction.

3.7. Application of CISC Architecture

CISC architecture is used widely in all kinds of CPU which has wide application as PC or personal computer. Day by day the processors in this PCs' are getting more faster which enables the user with more flexibility in his/her daily use. Texas Instruments calculators like the TI-89, TI-92, and Voyage 200 lines use the 68000 line of processors. CISC based processors also used widely in trainer boards for learning purposes. Palm Pilot series that are running on Palm OS 1.x to 4.x are CISC based and also in the control system of the space shuttle CISC based processors is used. However, they became most well known as the processors powering desktop computers such as the Apple Macintosh, the Commodore Amiga, the Sinclair QL, the Atari ST, purposes [27].

CHAPTER 4

FPGA

4.1. Introduction

FPGA stands for Field-Programmable Gate Arrays. FPGAs are pre-fabricated silicon devices that can be electrically programmed to become almost any kind of digital circuit or system. They have many advantages over Application Specific Integrated Circuits (ASIC). ASICs are designed for specific application using CAD tools and fabricated at a foundry. Developing an ASIC takes very much time and is expensive. Furthermore, it is not possible to correct errors after fabrication. In contrast to ASICs, FPGAs are configured after fabrication and they also can be reconfigured. This is done with a hardware description language (HDL) which is compiled to a bit stream and downloaded to the FPGA. The disadvantages of FPGAs are that the same application needs more space (transistors) on chip and the application runs slower on a FPGA as modern as the ASIC counterpart. Due to the increase of transistor density FPGA were getting more powerful over the years. On the other hand the development of ASICs was getting slower and more expensive. Therefore FPGAs are increasingly applied to high performance embedded system [29].

4.2. History

We'd have a brief look at the history of FPGA and maybe explain why we prefer to prototype this way before going to full ASIC systems. The development of the first Field Programmable Gate Array (FPGA) was by a company called Xilinx in 1985; however the number of gates on the chip was in the low thousands and it wasn't catching on at the time. Realizing the limitations of PROMs and PLDs the US Naval Surface Warfare dept. funded a project to develop a computer with reprogrammable gates, in this case, 600,000 of them. The patent for this technology was given in 1992, after this point the FPGA took off and started to appear initially in telecoms and networking equipment but later in automotive and consumer applications. In 1997, a researcher at the University of Sussex created algorithms that would adapt the FPGA, creating evolvable hardware. By the turn of the century, chips were starting to have millions of reprogrammable gates on them and the market share had gone from a small number of millions to billions of dollars. There are of course some downsides over ASICs, although these are changing. Historically FPGAs have been slower, less energy efficient and not had as much functionality. However, on the upside, they are easily reprogrammable to correct bugs and with the new security procedures being developed to securely load them, and the ability to correct security issues on the FPGA as an upgrade to a system, in the future we may well see FPGAs replacing ASICs more and more on a permanent basis [33].

4.3. FPGA Structures

4.3.1. Basic Structure

In the chapter the basic structure of a FPGA will be described. Xilinx is one of the biggest FPGA manufacturers. A Xilinx FPGA is made up of three basic blocks [38]:

• **PLB:** The programmable logic blocks are the user specific functions being calculated.

• **IOB:** The Input / Output block make it possible to connect the FPGA to the other elements of the application

• **Interconnect:** Interconnect is essential for writing between PLB and from IOBs to PLBs.

The PLBs are located at the center of the chip and the IOBs on the periphery. The interconnect is necessary to implement several designs on the FPGA. The distributed configuration memory controls the behavior of the PLBs and IOBs by storing the program. Next the implementations of PLBs interconnect and IOBs are described more in detail.

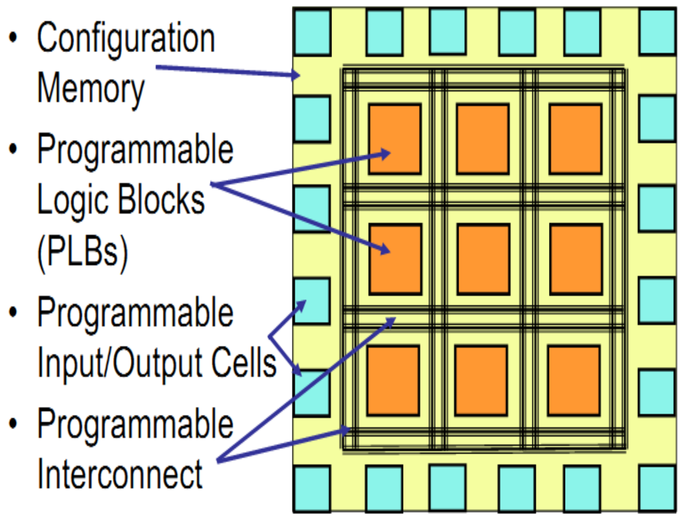

• Configuration Memory
• Programmable Logic Blocks (PLBs)
• Programmable Input/Output Cells
• Programmable Interconnect

Figure 4.1: Basic structure of FPGA [36]

4.4. Configuring a FPGA

FPGAs are not programmed directly. Synthesis tools translate the code into bit stream, which is downloaded to the configuration memory of the FPGA. Commonly, hardware description languages (HDL) are use to configure the device. But resent trends also offer the possibility to high level languages. Furthermore, there are library based solutions which are optimized for a specific device.

4.5. Advantages of FPGA

- Reduction in size (compact)
- Higher resolution
- Lower dead time
- Multifunctional
- Upgradable
- Possible automation of entire acquisition process
- Remote acquisition

4.6. Disadvantages of FPGA

- Signal requires pre-filtering due to ADC limitations
- Complex design procedure (requires knowledge of C/C++, VHDL/Verilog)
- Expensive hardware and software

CHAPTER 5

HARDWARE AND SOFTWARE

5.1. Introduction

From the title of the chapter it is clear that we are here going to brief about the hardware and software that we have used to implement the project ***"Remote High Speed Data Acquisition System".*** The main hardware that we have used to implement the project is "***Spartan-3E Starter Kit*** " and "***NI DAQ (National Instruments Data Acquisition System)***". As we know that without the proper programming the hardware do not work, so to program the FPGA and interfacing the necessary peripheral of Spartan-3E starter kit `we have used the "***VHDL (Very High Speed Integrated Circuits Hardware Description Language)"*** and to implement the control of the system we have used the LabVIEW software (Laboratory Virtual Instrumentation Engineering Workbench). An IDE or Integrated Development Environment is a software program that is designed to help programmers and developers build software. To code VHDL we have used Xilinx-13.2 ISE. It may seem a strange fact that why here we separately describing the "USB to RS232". The reason we have mentioned earlier in the abstract and chapter-1 that we have implement two "***Remote Data Acquisition System***". One communicates using the "*10/100 Ethernet Physical Layer*" and another using "*RS232 Serial Ports*". So we think it is important make the reader aware of the model that we have used while implementing the projects. This chapter will give reader a fair idea about the hardware and software that are needed to implement such kind of digital system. In hardware section we will give a fair idea about the "*Spartan-3E Starter Kit*", "USB to RS232" and "NI DAQ". And in the Software section of this chapter we will brief about the VHDL, LabVIEW and Xilinx-13.2 ISE. Here will just brief about what these things are, not how they are working or how to use them. In the appendix section we have given two tutorials on "Xilinx-13.2 ISE" and "VHDL".

5.2. Hardware

5.2.1. Spartan-3E-starter kit

XC3S500E FPGA is the core of the Spartan-3E Starter kit board which is one of the Spartan-3E families. Its logic is higly optimized. This board provides ideal platform for the implementation and verification of designs for embedded and FPGA applications. The Spartan-3E Starter Kit board highlights the unique features of the Spartan-3E FPGA family and provides a convenient development board for embedded processing applications [35]. The board highlights these features:

- Spartan-3E specific features
- Parallel NOR Flash configuration
- MultiBoot FPGA configuration from Parallel NOR Flash PROM
- SPI serial Flash configuration
- Embedded development
- MicroBlaze™ 32-bit embedded RISC processor
- PicoBlaze™ 8-bit embedded controller
- DDR memory interfaces

The Spartan-3E Starter Kit board includes additional peripherals and memory; user can design multi-purpose systems with Verilog HDL and VHDL. MicroBlaze or PicoBlaze soft-core processor can run on Spartan-3E FPGA, so users can make use of ISE and EDK development kit to design and debug customer system based on MicroBlaze or PicoBlaze quickly [1].

The key features of the Spartan-3E Starter Kit board are [34,35,36]:

- Xilinx XC3S500E Spartan-3E FPGA
- Up to 232 user-I/O pins
- 320-pin FBGA package
- Over 10,000 logic cells
- Xilinx 4 Mbit Platform Flash configuration PROM
- Xilinx 64-macrocell XC2C64A CoolRunner CPLD
- 64 MByte (512 Mbit) of DDR SDRAM, x16 data interface, 100+ MHz

- 16 MByte (128 Mbit) of parallel NOR Flash (Intel StrataFlash)
- FPGA configuration storage
- Micro Blaze code storage/shadowing
- 16 Mbits of SPI serial Flash (STMicro)
- FPGA configuration storage
- Micro Blaze code shadowing
- 2-line, 16-character LCD screen
- PS/2 mouse or keyboard port
- VGA display port
- 10/100 Ethernet PHY (requires Ethernet MAC in FPGA)
- Two 9-pin RS-232 ports (DTE- and DCE-style)
- On-board USB-based FPGA/CPLD download/debug interface
- 50 MHz clock oscillator
- SHA-1 1-wire serial EEPROM for bitstream copy protection
- Hirose FX2 expansion connector
- Three Digilent 6-pin expansion connectors
- Four-output, SPI-based Digital-to-Analog Converter (DAC)
- Two-input, SPI-based Analog-to-Digital Converter (ADC) with programmable-gain pre-amplifier
- ChipScope™ SoftTouch debugging port
- Rotary-encoder with push-button shaft
- Eight discrete LEDs
- Four slide switches
- Four push-button switches
- SMA clock input
- 8-pin DIP socket for auxiliary clock oscillator

Figure-5.1. Spartan-3E Starter Kit Board-1 [36]

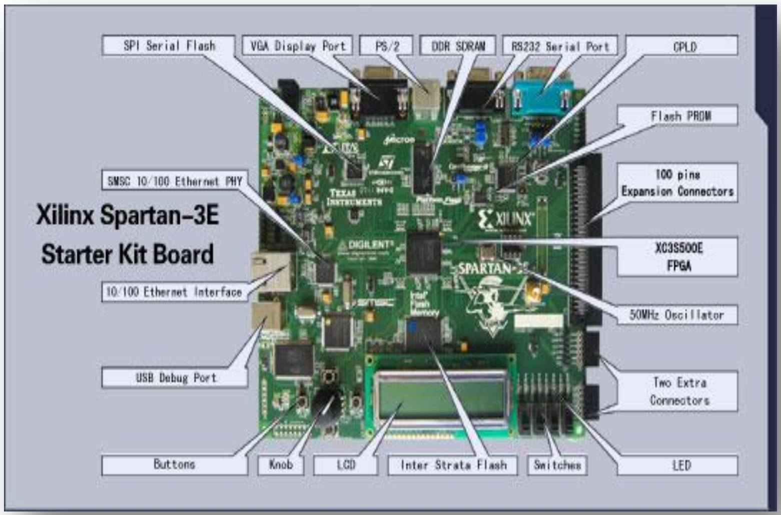


Figure-5.2. Spartan-3E Starter Kit Board-2 [34]

USB Type B Connector
Connects to Starter Kit's USB connector
USB Type A Connector
Connects to computer's USB connector
UG230_c4_04_030306

Figure-5.3. Standard USB Type A/Type B Cable[35]

Figure-5.4. Connect the USB Type B Connector to the Starter Kit Board Connector[35]

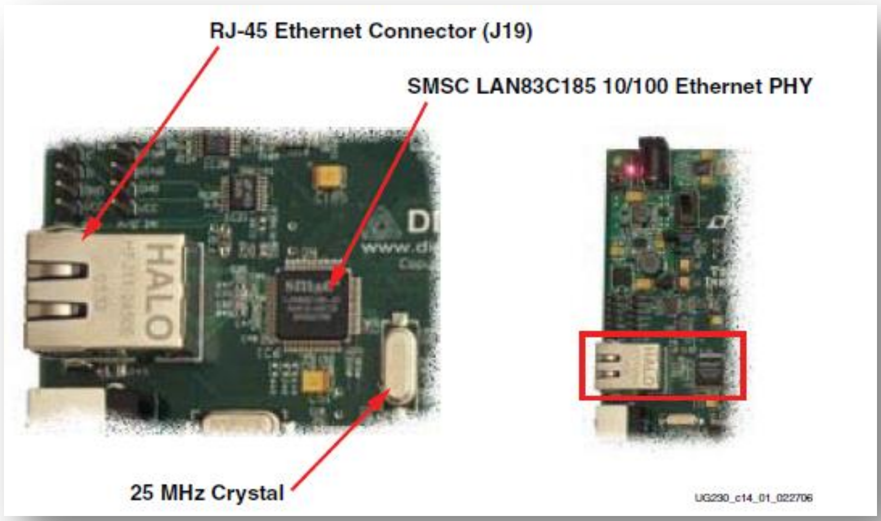

RJ-45 Ethernet Connector (J19)
SMSC LAN83C185 10/100 Ethernet PHY
25 MHz Crystal
UG230_c14_01_022706

Figure-5.5. 10/100 Ethernet PHY with RJ-45 Connector[35]

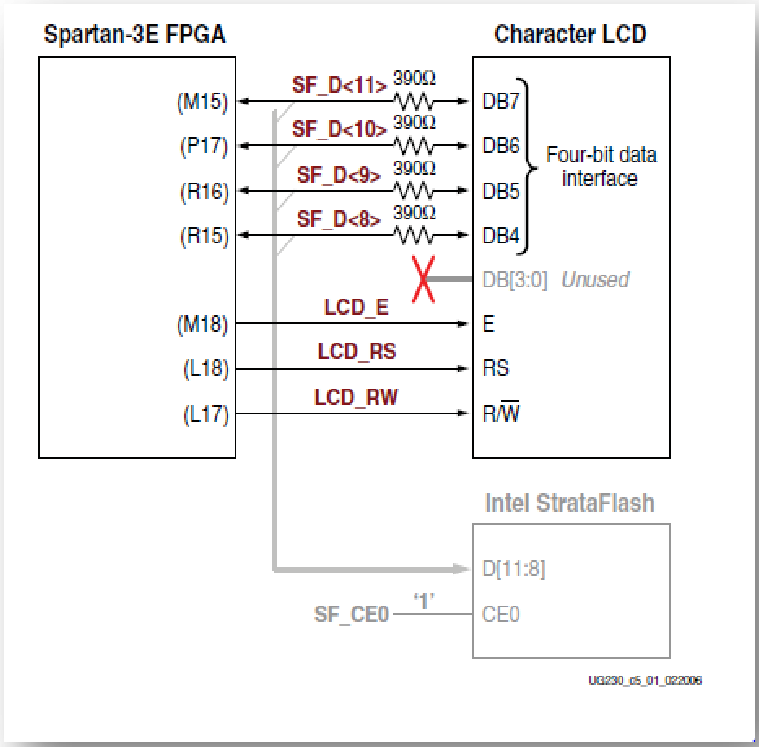


Figure-5.6. Character LCD Interface [35]

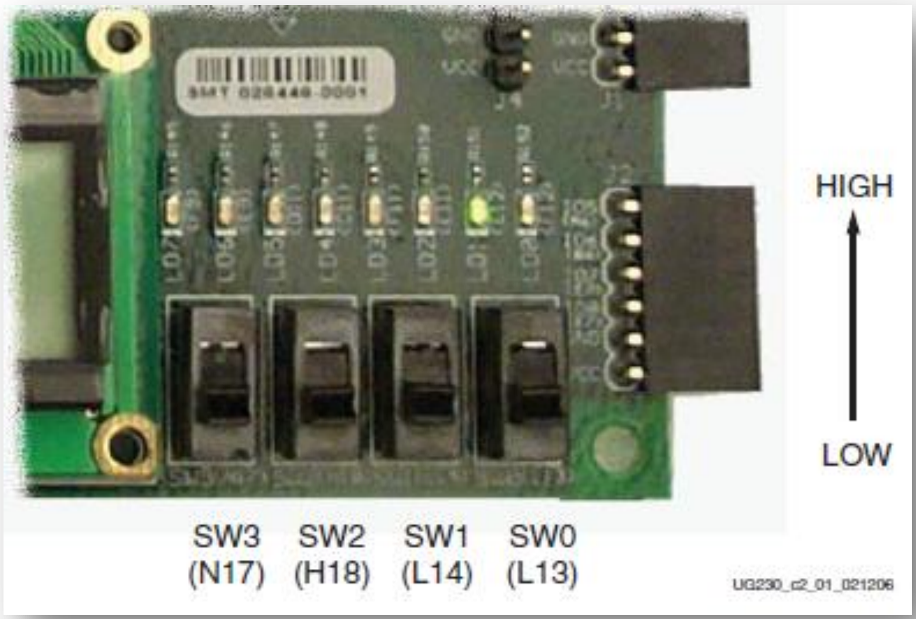


Figure-5.7. Four Slide Switches and Eight LEDs [35].

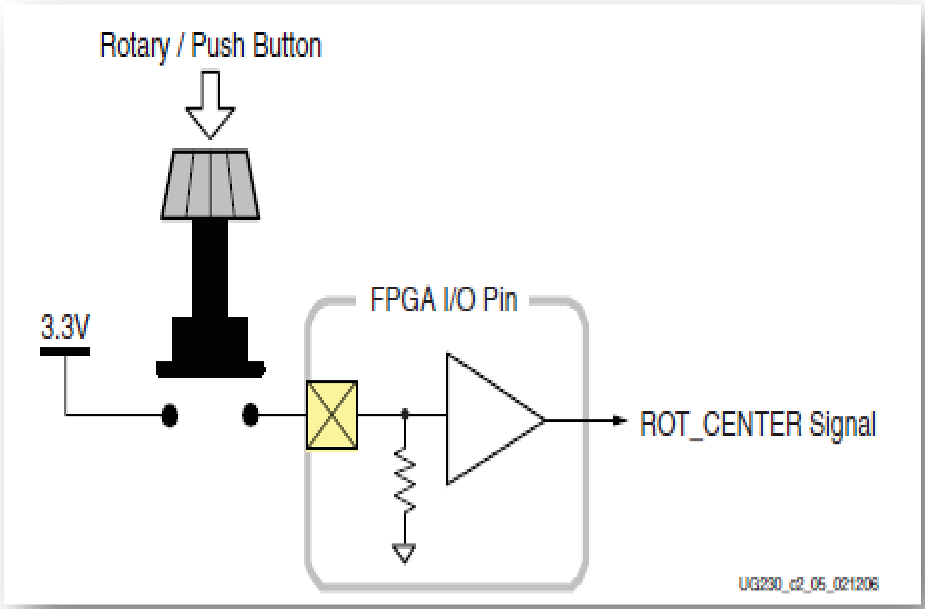


Figure-5.8. Push-Button Switches Require Internal Pull-up Resistor in FPGA Input Pin [35].

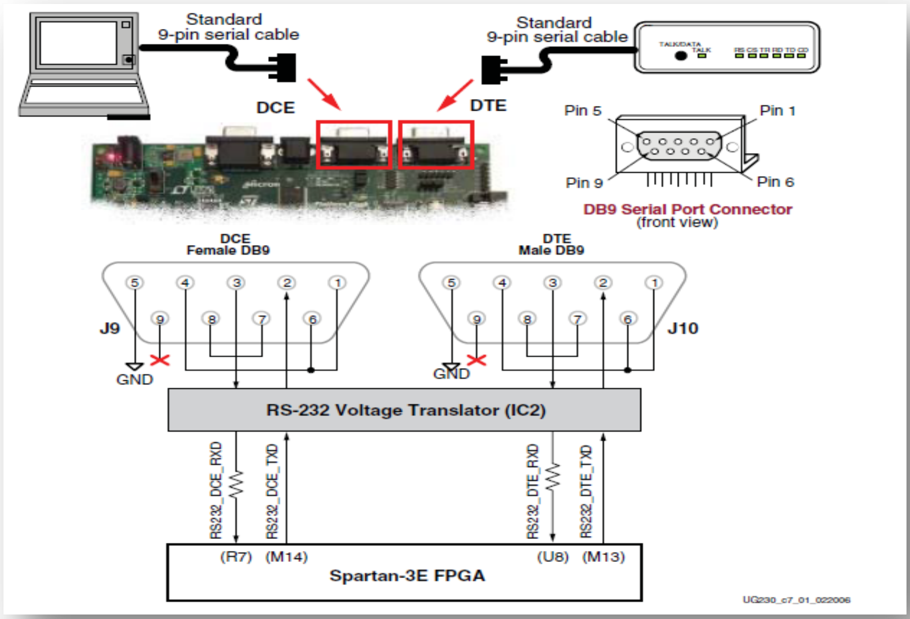


Figure-5.9. RS-232 Serial Ports [35].

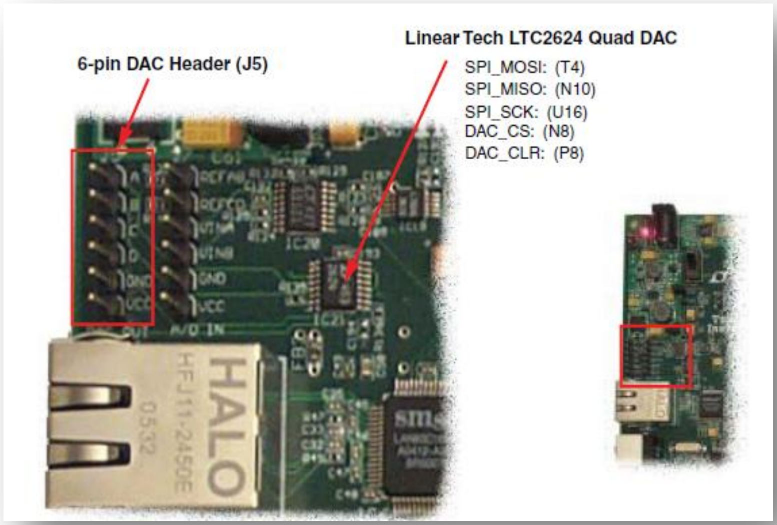


Figure-5.10. Digital-to-Analog Converter and Associated Header [35].

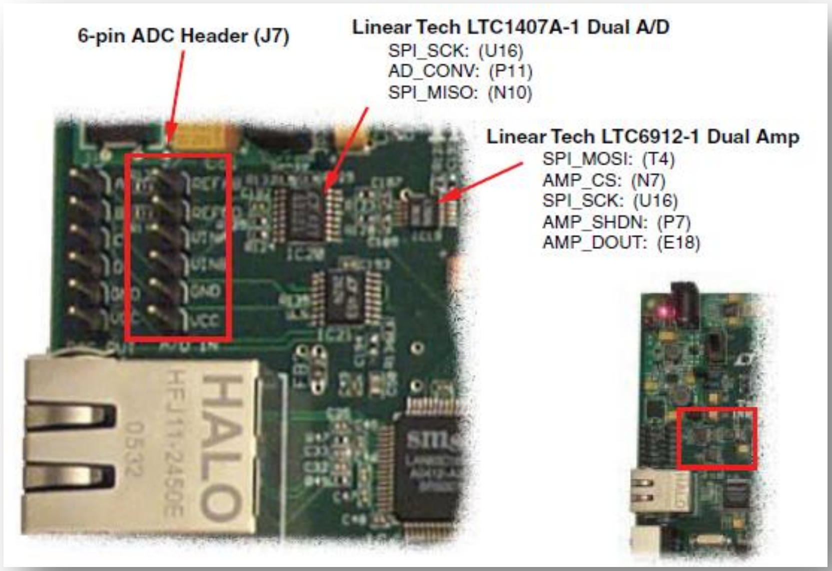


Figure-5.11. Analog to Digital Converter (Two-Channel Analog Capture Circuit) [35]

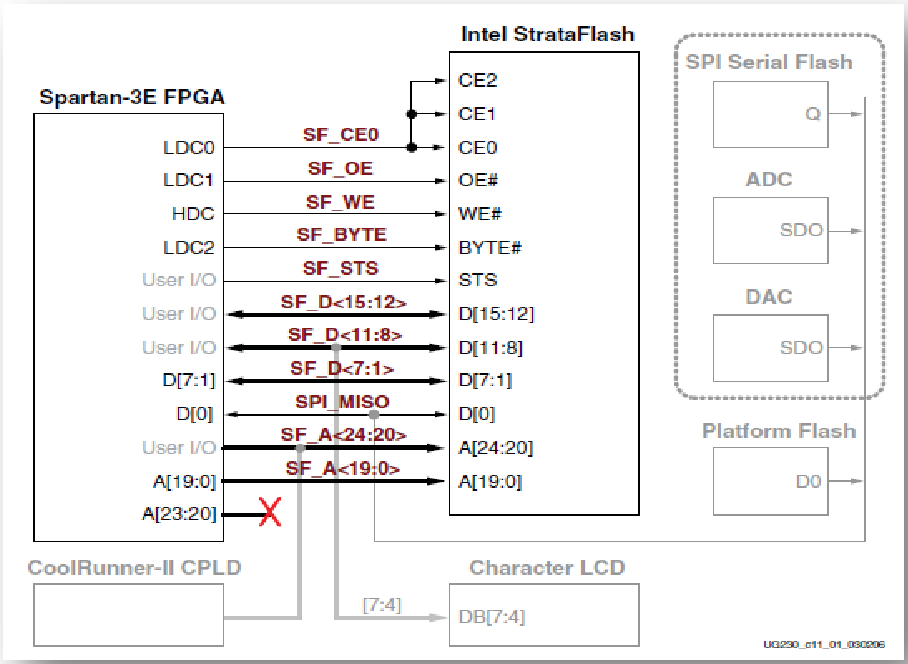


Figure-5.12. Connections to Intel StrataFlash Flash Memory [35]

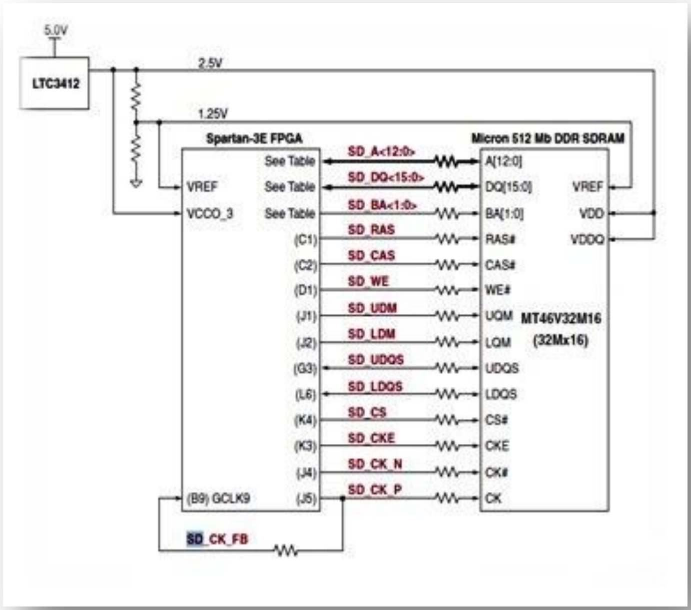


Figure-5.13. FPGA Interface to Micron 512 Mbit DDR SDRAM [35]

Figure 5.3 represents the Commercial Package of Spartan-3E starter kit. Fig 5.4 is the completely indicator diagram of Spartan-3E, means it shows the consisting elements on the board. And from the figure-3 to fig 5.15 are the pictorial representation of the necessary peripherals those we need to interface using the VHDL for the implementation of our project "Remote High Speed Data Acquisition System".

5.3. USB TO RS232 SERIAL Adapter CABLE DB9 PIN PL2303

USB to RS232 serial cable adapter supports over 1Mbps data transfer rate which is enriched with remote Wake-up and power management [34].

5.3.1. USB TO RS232 SERIAL Adapter CABLE DB9 PIN PL2303 Features [38]

High-Speed USB 2.0 To RS232 Serial Pin Cable Adapter-

- USB specification revision 1.1 & 2.0 compliant from this USB Cable Adapter
- High-Speed USB to RS232 Cable support Rs232 serial interface (DB9)
- USB Data Cable Support Windows ME/2000/XP
- Measurement: 81cm/31.8" in length; Weight: 86g

Figure-5.14. USB TO RS232 Serial cable adapter [38]

5.4. Software

In high speed remote data acquisition system VHDL, Lab VIEW, Xilinx-13.2 is used as software. These are going to be described below.

5.4.1. VHDL

VHDL is a Hardware Description Language (HDL) which describes the behavior of an electronic circuit or system. VHDL stands for Very High Speed Integrated Circuits (VHSIC) Hardware Description Language. The development of the VHDL language was funded by the United States Department of Defense (DoD) in the 1980. VHDL's first version was

VHDL-87, later upgraded to the so-called VHDL-93. This is the first and original HDL to be standardized by the IEEE (Institute of Electrical and Electronic Engineers), through the IEEE 1076 standard. An additional standard, the IEEE 1164, was later added to introduce a multi-valued, logic system. VHDL is able to do both circuit synthesis and simulation. VHDL is fully simulatable but not all the constructs are synthesizable. The (automatic) transformation of a less detailed description into a more elaborated one is called synthesis. The main reason behind choosing the VHDL is that, it is a standard, technology or vendor independent language, portable and reusable. The two main applications of VHDL are in the field of Programmable Logic Devices and in the field of ASICs. Programmable Logic Devices are Field Programmable Gate Array (FPGA) and Complex Programmable Logic Devices(CPLD). Once the VHDL code has been written, it can be used either to implement the circuit in a programmable device (Altera, Xilinx, Atmel etc.) or can be submitted to a foundry for fabrication of an AISC chip [1]. VHDL is not sequential language rather it is inherently concurrent (parallel). In concurrent coding the instructions are executed in parallel. Concurrent code is also called the *dataflow code*. This is why VHDL is referred as a “code” rather than a program. Usually WHEN, GENERATE, operators and special kind of assignment, called BLOCK is used for the concurrent coding in VHDL. Purely concurrent code could not be used to implement the synchronous circuits. We can say that, using concurrent code the combinational logic circuit could be built. If we want to implement a sequential circuit then we have to employ the sequential code. But the statement which is placed inside a PROCESS, FUNCTION OR PROCEDURE is executed sequentially [3,39].

5.4.2. Lab VIEW

In the recent days throughout the industry, academia and research labs LabVIEW becomes a common choice of standard as data acquisition and instrument control software. This is a multiplatform instrumentation and analysis softare system which is not only powerful for the required purpose but also provide with flexibiliy. It can be run on Windows , MAC OS X and Linux. The main feature of this software is it can be run on embedded FPGA chips and 32-bit microprocessor which is why it is a popular choice for instrumentation or manipulation of a FPGA based design, Data Acquisition System etc. It has a very user friendly interface which offers feasible writing and execution of programs. Unlike the tradition programming languages LabVIEW features a graphical programming environment with all essential tools for data acquisition, analysis of data and result presentation.It is also able to solve a wide variety of problems in a very short time without any compulsive writing of conventional code in other engineering application.

5.4.3 Xilinx 13.2 ISE

Xilinx is a simulation software for VHDL, Verilog on embedded FPGA based design. The features of logic edition are-

- Complete flow for RTL-based design
- Familiar design flows (pre-7 series devices), with full support of the latest Xilinx 7 series FPGAs and Zynq-7000
- Integrated tool set supporting logic/connectivity, embedded, and DSP designs

The features for embedded edition are-

- Use one tool chain for hard and soft microprocessors
- Reduces board complexity and cost
- Leverage intelligent tools and IP

The features for DSP edition are-

- Leverage tools and IP for varied approaches
- Addresses DSP performance bottlenecks
- Enables leading-edge algorithms

CHAPTER 6

REMOTE DATA ACQUISITION SYSTEM USING FPGA

6.1. Interfacing the Peripherals of Spartan-3E Starter Kit

With recent improvements in the density of field programmable gate array (FPGA) devices, systems with an increasingly higher level of integration are possible. Digital signal processing is an important application area for FPGAs and such systems often require data converters to provide analog outputs from digital domain representations (D. Sillage, 2009). The DAC could be entirely built on FPGA by using the VHDL (Very High Speed Integrated Circuit Hardware Description Language). As it is completely single chip system so it provides a high degree of flexibility and reduces costs. Developed VHDL code has been implemented on the Spartan-3E starter kit.

6.1.1. DAC

The Spartan-3E Starter Kit board includes an SPI-compatible, four-channel, serial Digital to-Analog Converter (DAC). The DAC device is a Linear Technology LTC2624 quad DAC with 12-bit unsigned resolution. The four outputs from the DAC appear on the J5 header, which uses the Digilent 6-pin Peripheral Module format [71].

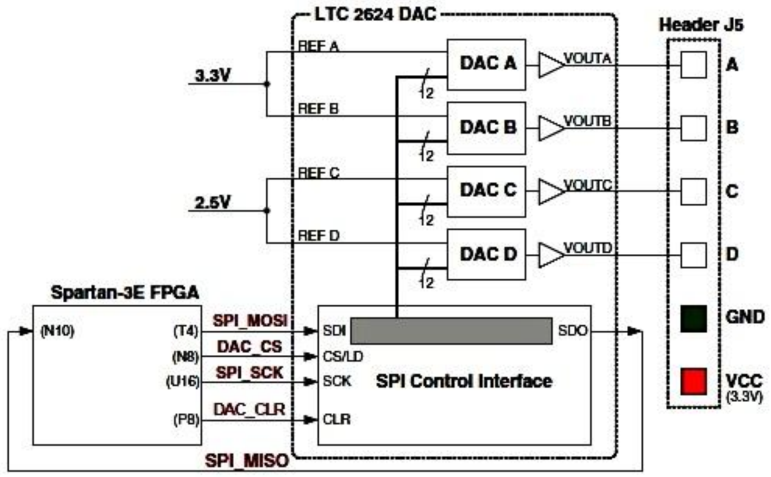


Figure-6.1. DAC connection schematics [71]

6.1.2. SPI Communication

The FPGA uses a Serial Peripheral Interface (SPI) to send digital values to each of the four DAC channels. The SPI bus is a full-duplex, synchronous, character-oriented channel employing a simple four-wire interface. The interface signals between the FPGA and the DAC are the SPI_MOSI, SPI_MISO, and SPI_SCK which are shared with the other devices on the SPI bus. The DAC_CS signal is the active-Low slave select input to the DAC. The DAC_CLR signal is the active-Low, asynchronous reset input to the DAC. As a bus master the FPGA drives the bus clock signal (SPI_SCK) and transmits serial data (SPI_MOSI) to the selected DAC bus slave [72].

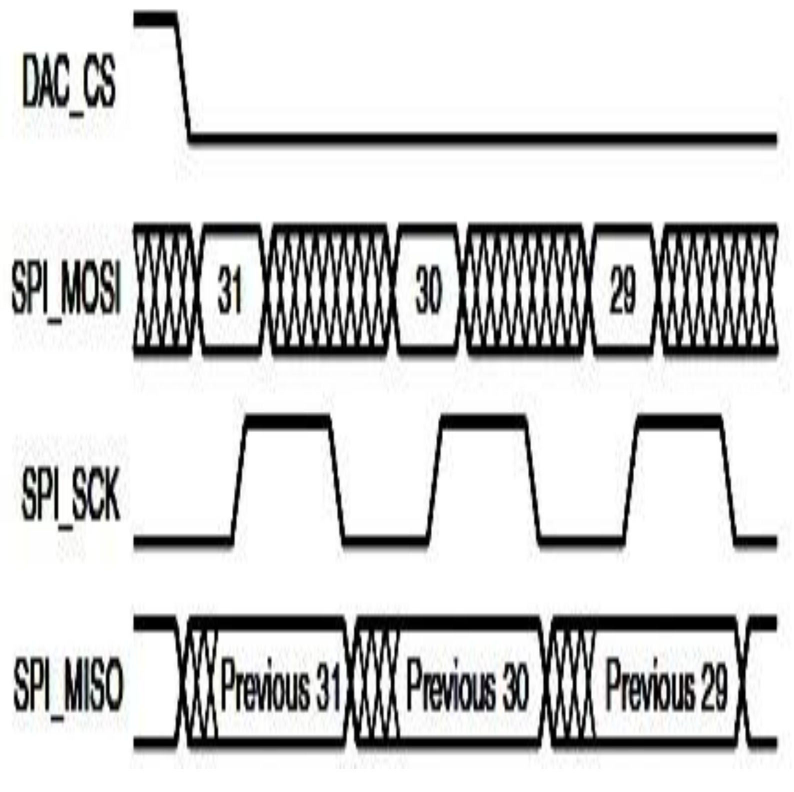


Figure-6.2. SPI communication waveform [72]

6.1.3. Communication Protocol

Figure 3 shows the shows the communications protocol required to interface with the LTC2624 DAC. The DAC supports both a 24-bit and 32-bit protocol. The 32-bit protocol is shown. Inside the D/A converter, the SPI interface is formed by a 32-bit shift register [73].

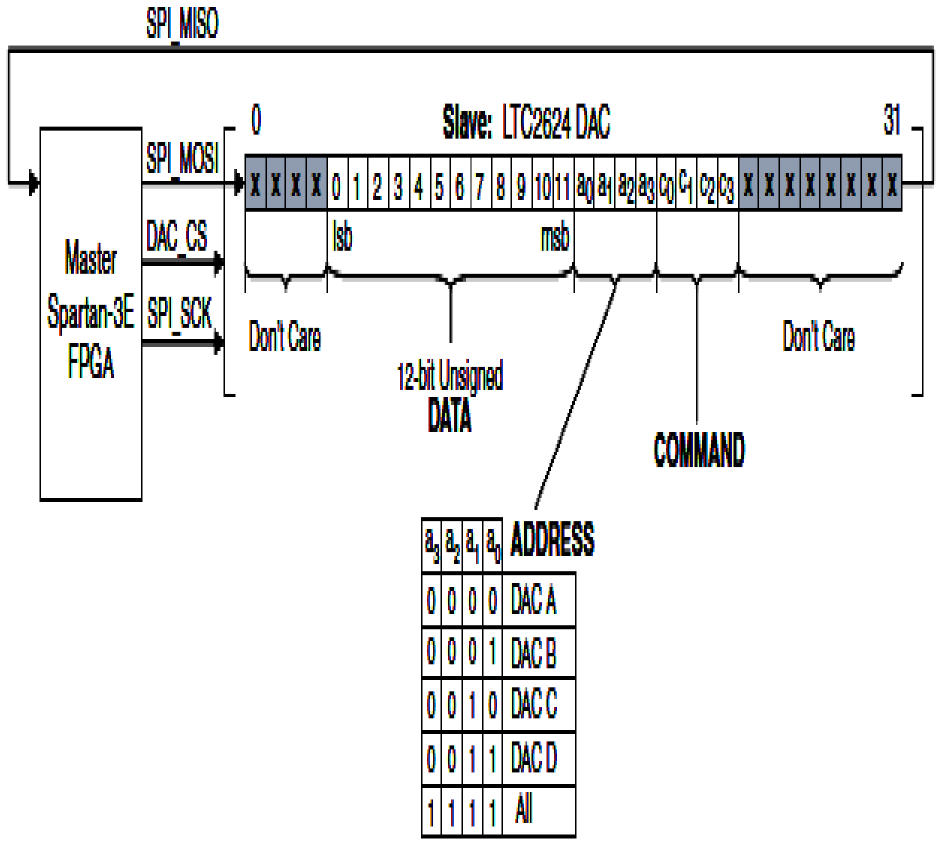


Figure-6.3. SPI communication protocol[73]

Each 32-bit command word consists of a command, an address, followed by data value. The FPGA first sends eight dummy or “don’t care” bits, followed by a 4-bit command. The most commonly used command with the board is COMMAND [3:0] = “0011”, which immediately updates the selected DAC output with the specified data value. Following the command, the FPGA selects one or all the DAC output channels via a 4-bit address field. Following the address field, the FPGA sends a 12-bit unsigned data value that the DAC converts to an analog value on the selected output(s). Finally, four additional dummy or *don’t care* bits pad the 32-bit command word [74].

6.1.4. Character LCD Screen

The Spartan-3E starter kit has 2-line by 16-character LCD. The FPGA controls the LCD by using the 4-bit data interface. The LCD is capable of displaying a variety of information using the standard ASCII and custom characters. The interface character LCD interface signals are the SF_D, SF_D, SF_D, SF_D, LCD_E, LCD_RS and LCD_RW [74].

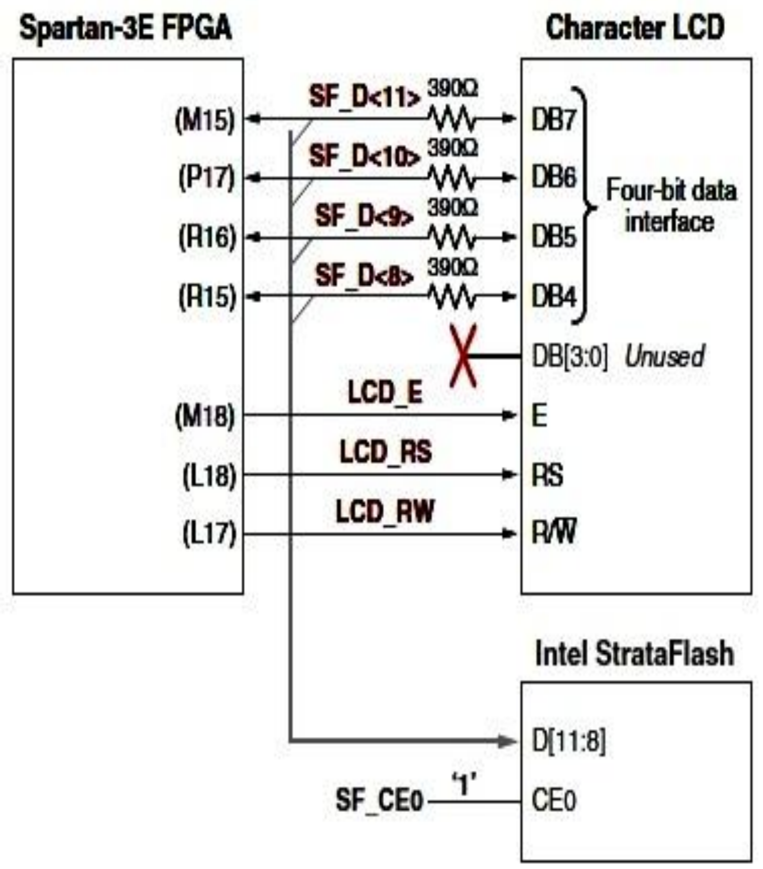


Figure-6.4. Character LCD interface [74].

6.1.5. Rotary Push-Button Switch

The rotary push-button switch is capable of producing three outputs. The two shaft encoder outputs are ROT_A and ROT_B and the center push-button switch is ROT_CENTER. This switch integrates two different factions. The switch shaft rotates and outputs values whenever the shaft turns. The shaft can also be pressed, acting as a push-button switch [74].

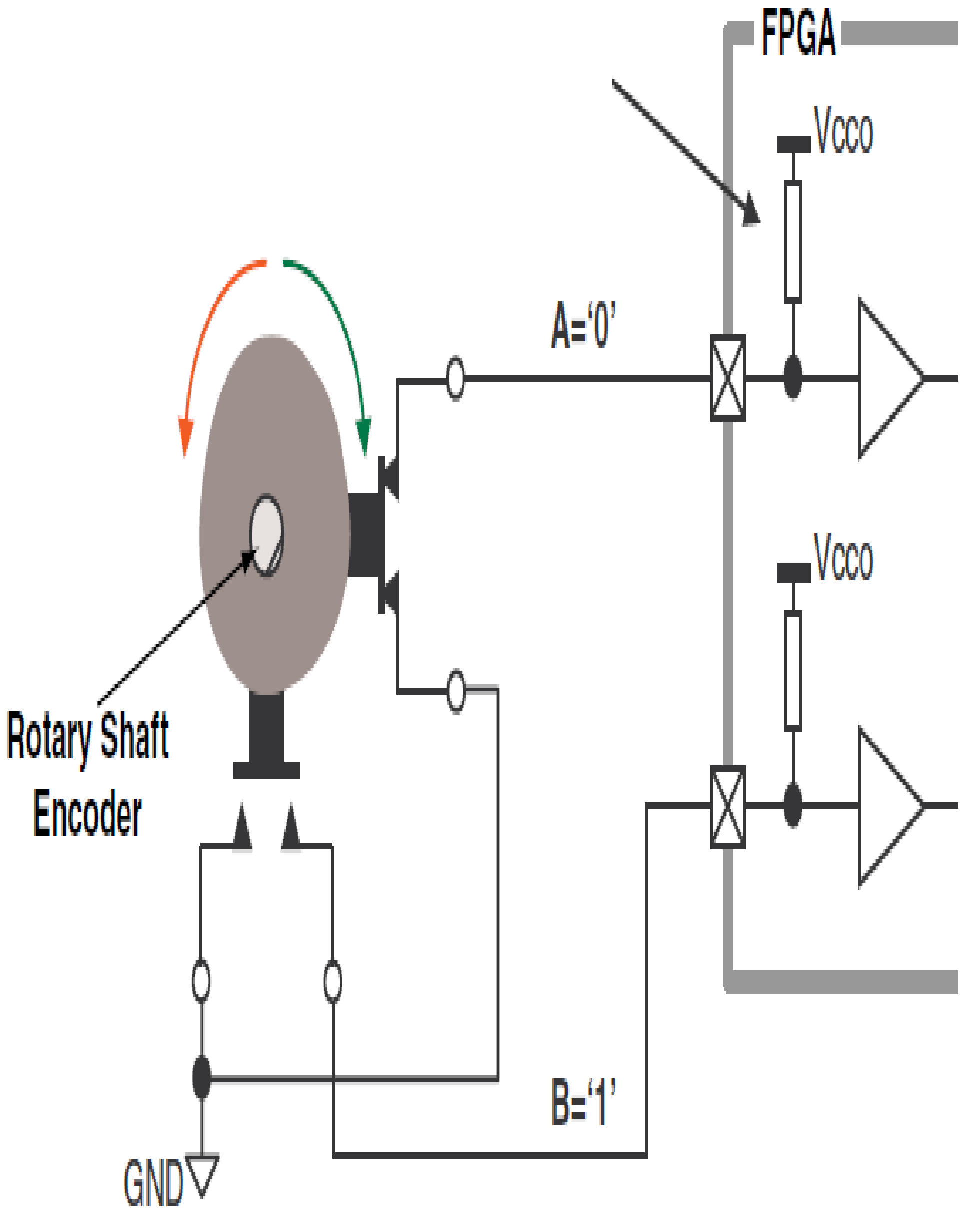


Figure-6.5. Rotary Switch Shaft Encoder Circuitry [73]

Rotating the shaft operates two push-button switches, as shown in 4. Depending on which way the shaft is rotated, one of the switches opens before the other. Likewise, as the rotation continues, one switch closes before the other. However, when the shaft is stationary, also called the detent position, both switches are closed. Closing a switch connects it to ground, generating a logic Low. When the switch is open, a pull-up resistor within the FPGA pin pulls the signal to logic High [72]

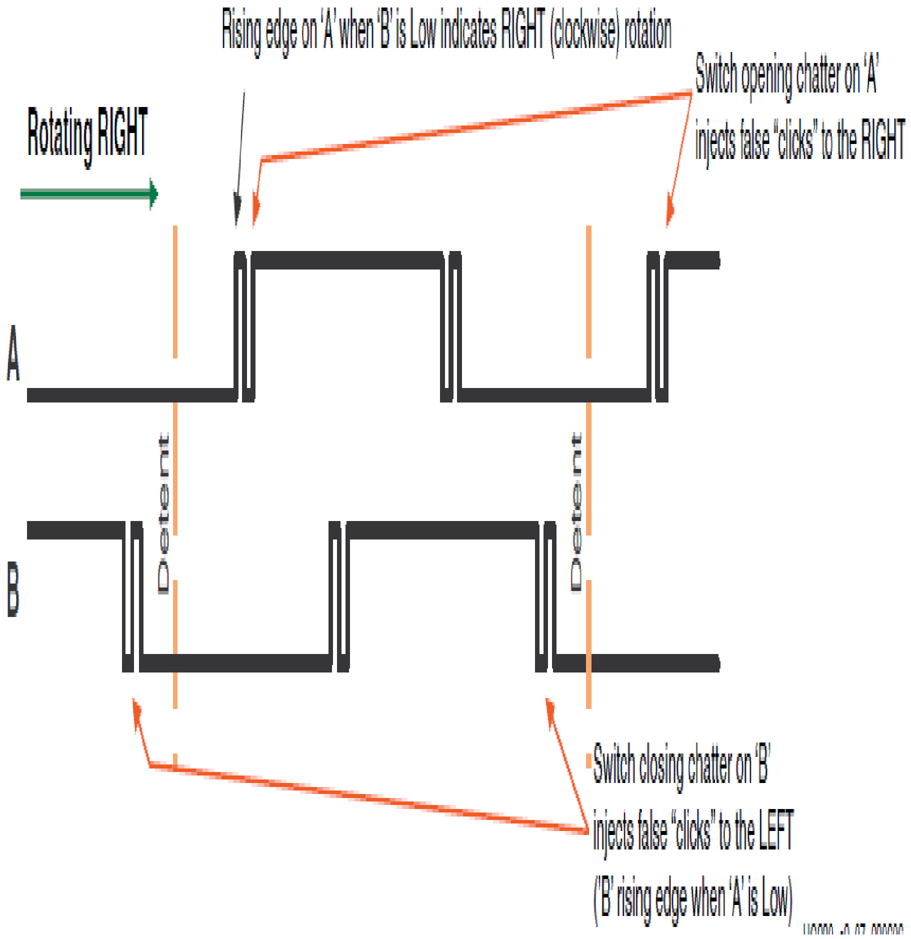

Rising edge on 'A' when 'B' is Low indicates RIGHT (clockwise) rotation
Switch opening chatter on 'A'
injects false "clicks" to the RIGHT
Rotating RIGHT
A
B
Detent
Detent
Switch closing chatter on 'B'
injects false "clicks" to the LEFT
('B' rising edge when 'A' is Low)

Figure-6.6. Output from Rotary Shaft Encoder Circuitry [73].

6.2. Working Principle

This system produces analog voltage at the output of DAC according to the digital value provided by the FPGA . The digital input of the DAC is controlled with the help of rotary switch. The rotation of switch in clockwise direction increase the analog output value and rotation in anticlockwise direction result in the decrement of the analog output value. The step voltage of increment is controlled with a push-button switch. Default step voltage is 100mv. With pressing this switch we can change the step from 100mv to 10mv. For the next push it switches to previous state.

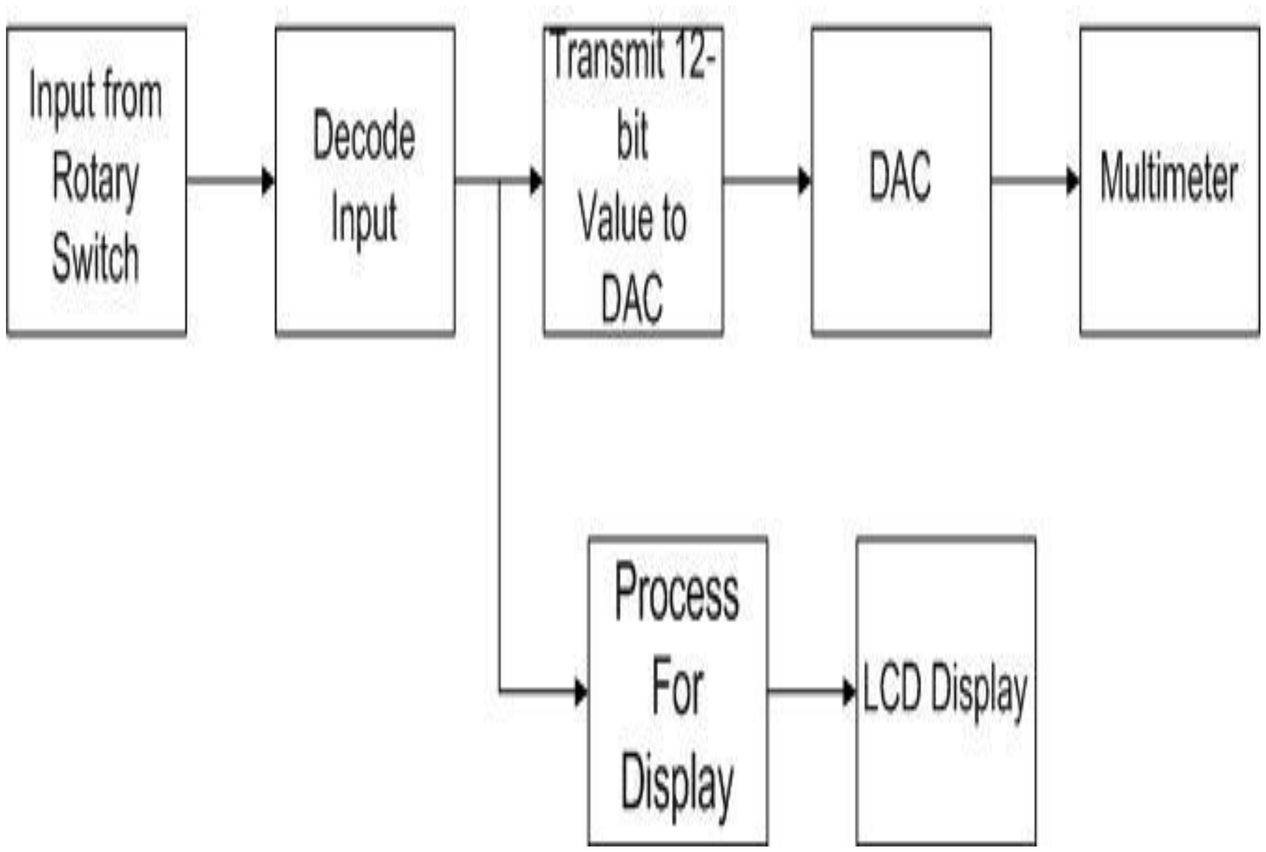


Figure-6.7. Block Diagram of the Developed System

We have developed several VHDL components to implement system within FPGA as shown in Figure-8. Rotary switch component sense the rotary switch and increment and decrement the DAC's digital values. For a single rotation the DAC's digital input value is increased to 164 for 100mv step or 16 for 10 mv. But in reverse rotation it decreases the DAC digital input with equal amount (164 for 100mv step or 16 for 10 mv). The DAC digital input is passed to DAC component.

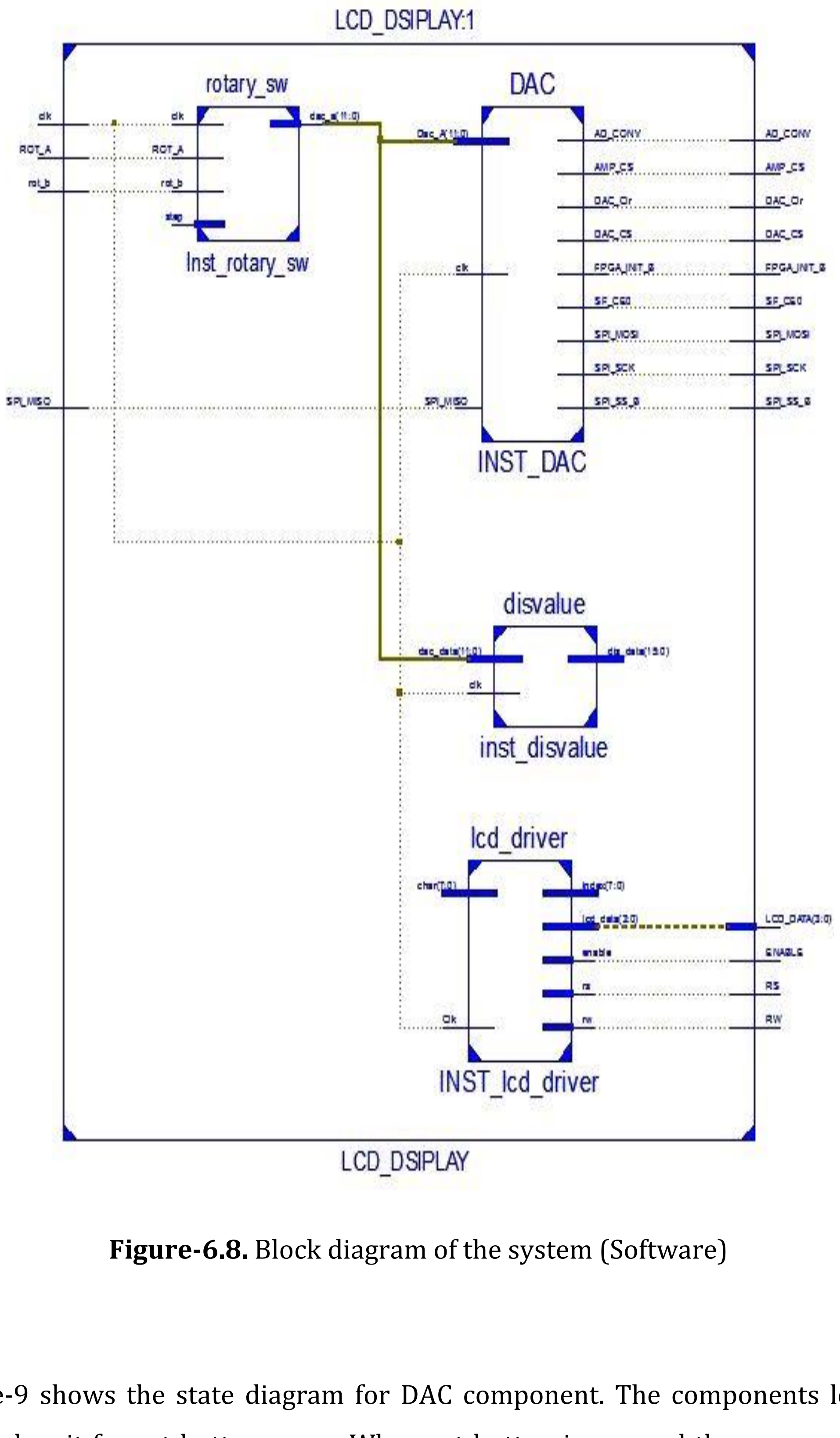


Figure-6.8. Block diagram of the system (Software)

The Figure-9 shows the state diagram for DAC component. The components loop in the state S0 and wait for set button press. When set button is pressed the program go to the next state S1. In the state S1 program will make the DAC_CS = ‘0’ to select the DAC and transfer control to the state S2. In this state 1-bit out of 32-bit data will be sent making the clock SPI_SCK low and transferred to the state S3. In state S3 clock SPI_SCK is made high and condition bitnr < 32 is checked (bitnr is the number of bit transmitted). If the condition bitnr< 32 is true program is transferred to the state S2 otherwise it is transferred to state S4. When the program is in the state S4, switching from this state to other state (S0 or S1) depends upon the set value as depicted in the figure 9.

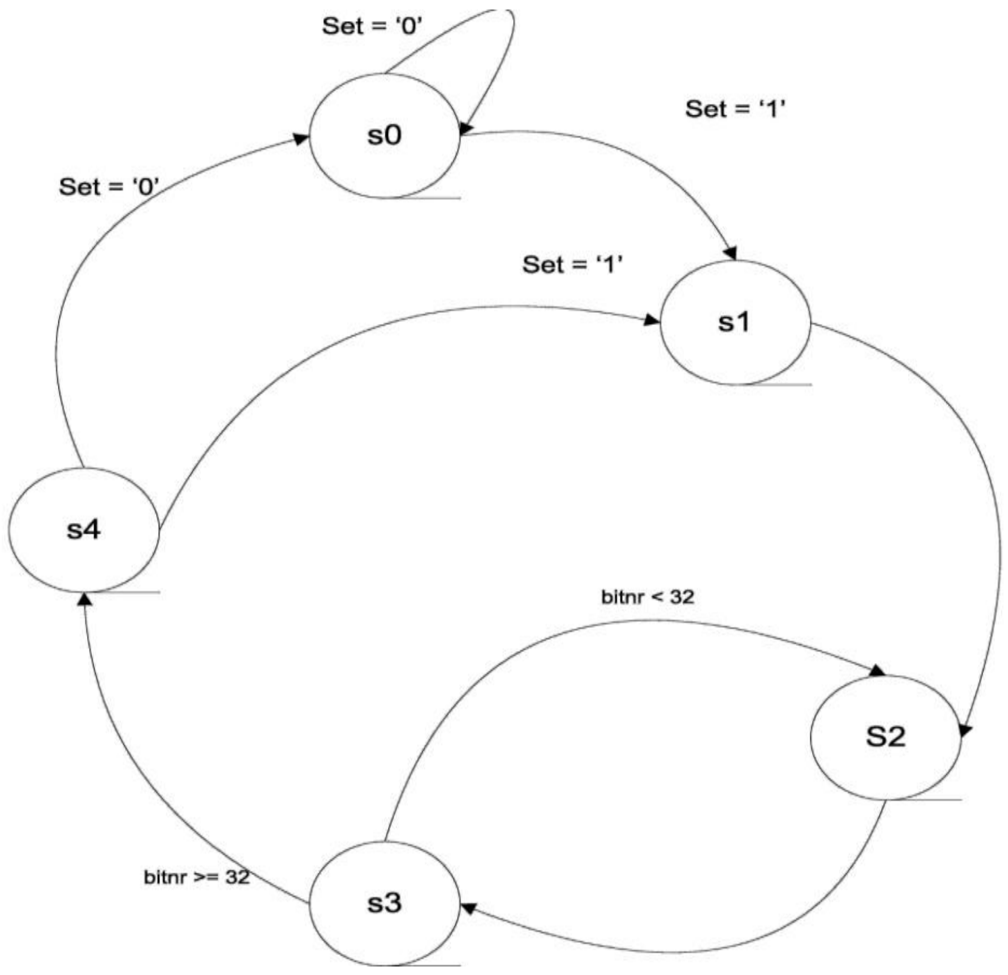


Figure-6.9. State diagram of the DAC

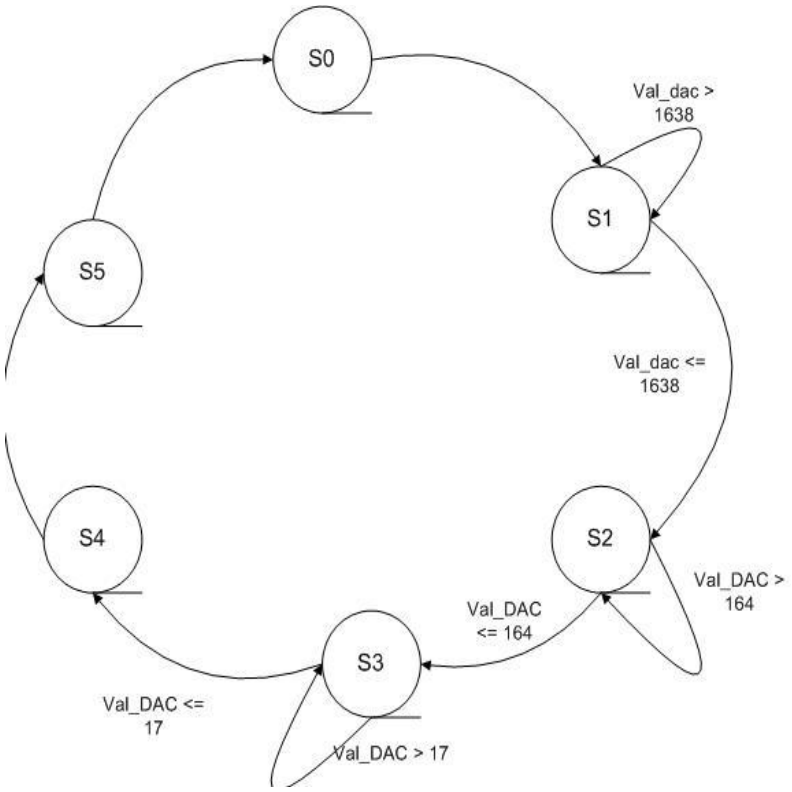


Figure-6.10. State diagram of the dis_value component

The DAC digital input is also sent to dis_value (display value) components. These components process digital input to generate suitable code for representing voltage on the LCD screen. The state diagram for this component is shown in figure-10. In the state S0 val_adc register is updated with DAC digital value. Then in the next state S1, 1638(equivalent to 1 volt) is subtracted from the val_adc. If subtracted value is less than 1638, the program is transferred to the next state S2. Here the val_adc is subtracted by 164(equivalent to .1 volt). In this way this component produces equivalent value for digital input. The generated code is sent to lcd_driver component. The LCD driver component takes the value from dis_value component. The component sends an index to the main program and ask bit pattern for the character to be displayed at this index.

6.3. ADC

ADC stands for Analog to Digital Converter which is essentially a circuit used to convert the analog input values into digital output values. In the Spartan-3E kit board there is a Analog to Digital Converter along with a programmable scaling pre-amplifier, in a two channel analog capture circuit. In the analog capture circuit there is a programmable preamplifier called LTC6912-1. The function of this programmable preamplifier is to scale the analog input signal on header J7 which is embedded on the kit board [76,93].

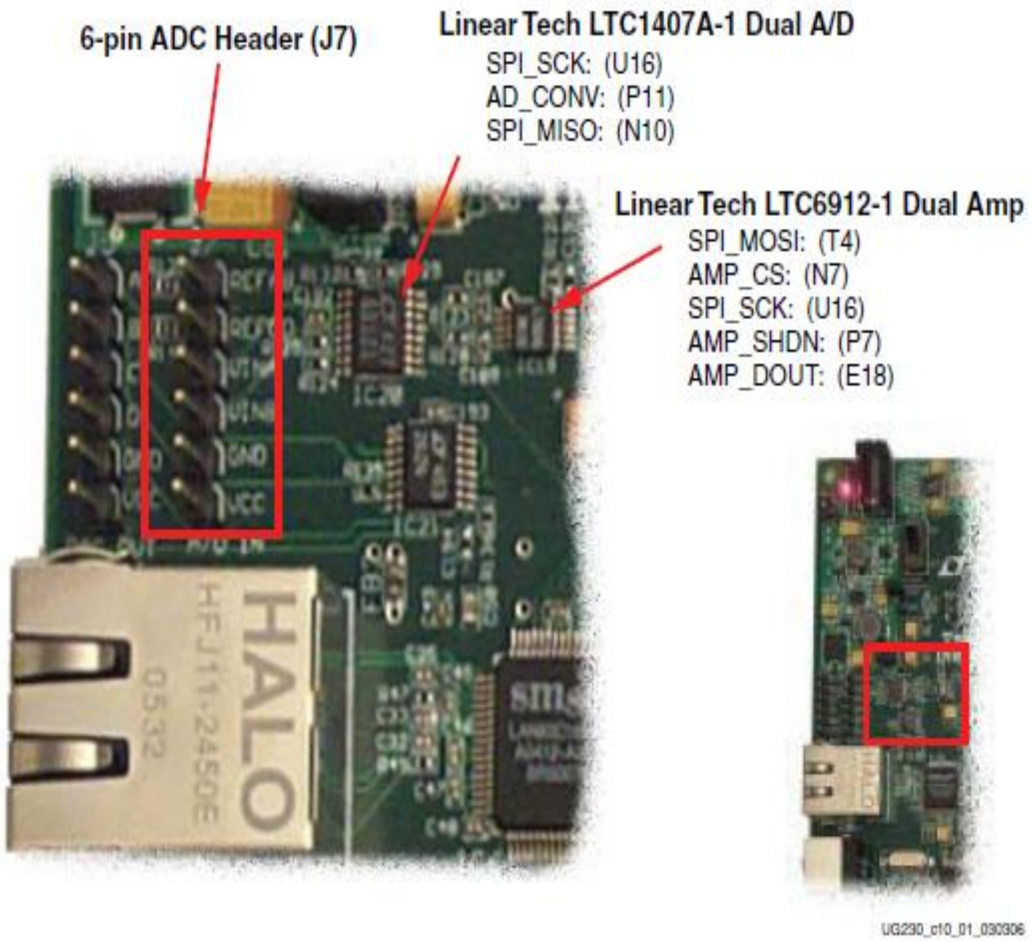


Figure-6.11. Two-Channel Analog Capture Circuit [76,93].

The output of the preamplifier goes as a input to the ADC called LTC1407A-1. FPGA is responsible for the program control of the preamplifier and ADC.

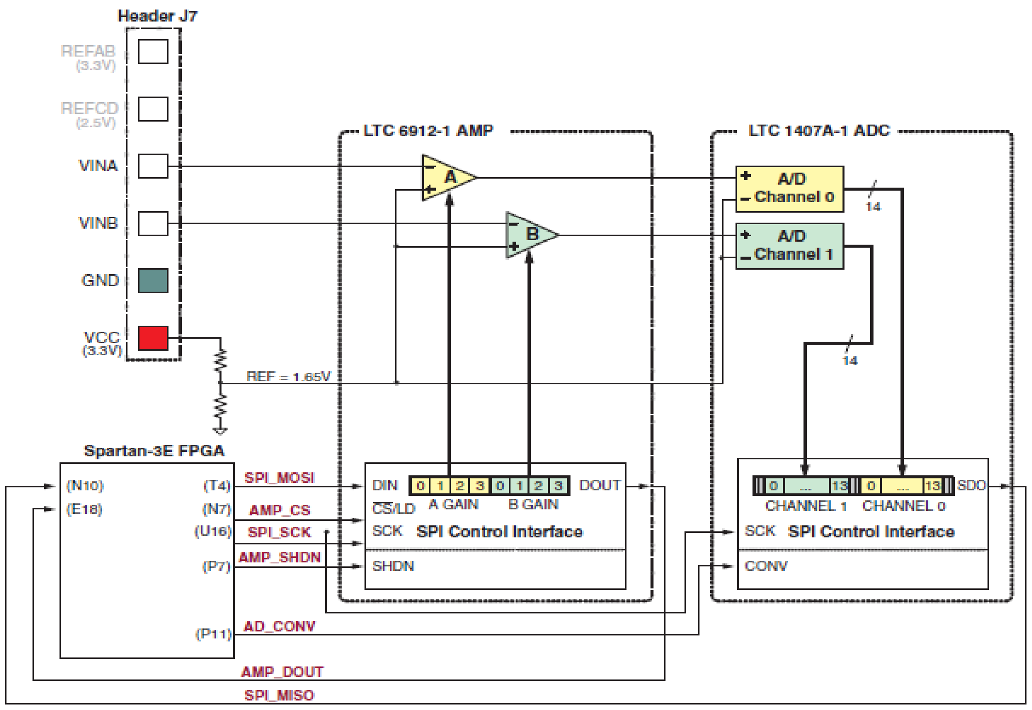


Figure-6.12. Detailed View of Analog Capture Circuit [76,93].

6.4. Programmable Pre-Amplifier

6.4.1. Programmable Gain

The analog voltage applied on VINA and VINB is converted by analog capture circuit which constitutes a digital representation consisting 14-bit. Two independent inverting amplifiers are provided by LTC6912-1 which has programmable gain. Scaling the incoming voltage on VINA or VINB is the main purpose of using the amplifier as a result the maximization of conversion range of DAC will become around 1.65+1.25v. A programmable gain amplifier is associated with each analog channel, as shown in figure. The amplification of analog voltage input in VINA and VINB is done relative to a reference voltage of 1.65. Using a voltage divider of 3.3V of the supply voltage the reference voltage 1.65V is generated. The amplifiers each have a programmable gain range of -1 to -100, as shown in Table 1 [76,93]..

Table-1

Programmable Gain Settings for Pre-Amplifier [76,93].

A3	A2	A1	A0	Gain	Input Voltage Range	
B3	B2	B1	B0		Minimun	Maximum
0	0	0	1	-1	0.4	2.9
0	0	1	0	-2	1.025	2.275
0	0	1	1	-5	1.4	1.9
0	1	0	0	-10	1.525	1.775
0	1	0	1	-20	1.5875	1.7125
0	1	1	0	-50	1.625	1.675
0	1	1	1	-100	1.6375	1.6625

6.4.2. Interface

The interface signals between the FPGA and the amplifier have been presented in the table-2. The SPI_MOSI, SPI_MISO, and SPI_SCK signals are shared with other devices on the SPI bus. The AMP_CS signal is the active-Low slave select input to the amplifier [76,93].

Table-2

AMP Interface Signals[76,91].

Signal	FPGA Pin	Direction
SPI_MOSI	T4	FPGA→AD
AMP_CS	N7	FPGA→AMP
SPI_SCK	U16	FPGA→AMP
AMP_SHDN	P7	FPGA→AMP
AMP_DOUT	E18	FPGA←AMP

6.4.3. SPI Control Interface

Figure-6.13. represents the SPI-based communication interface with amplifier. The gain for each amplifier is sent as an 8-bit command word, consisting of two 4-bit fields [76,93].

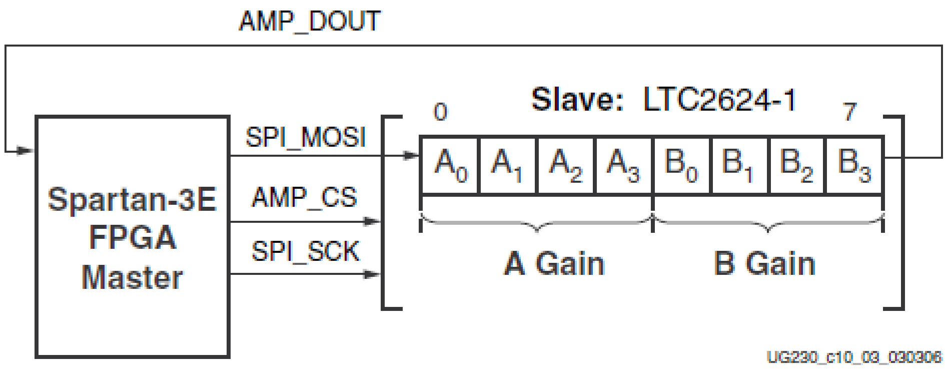


Figure-6.13. SPI Serial Interface to Amplifier [76,93].

The most-significant bit, B3, is sent first. The AMP_DOUT output from the amplifier echoes the previous gain settings. The SPI bus transaction starts when the FPGA asserts AMP_CS Low. The amplifier captures serial data on SPI_MOSI on the rising edge of the SPI_SCK clock signal. The amplifier presents serial data on AMP_DOUT on the falling edge of SPI_SCK [76,93].

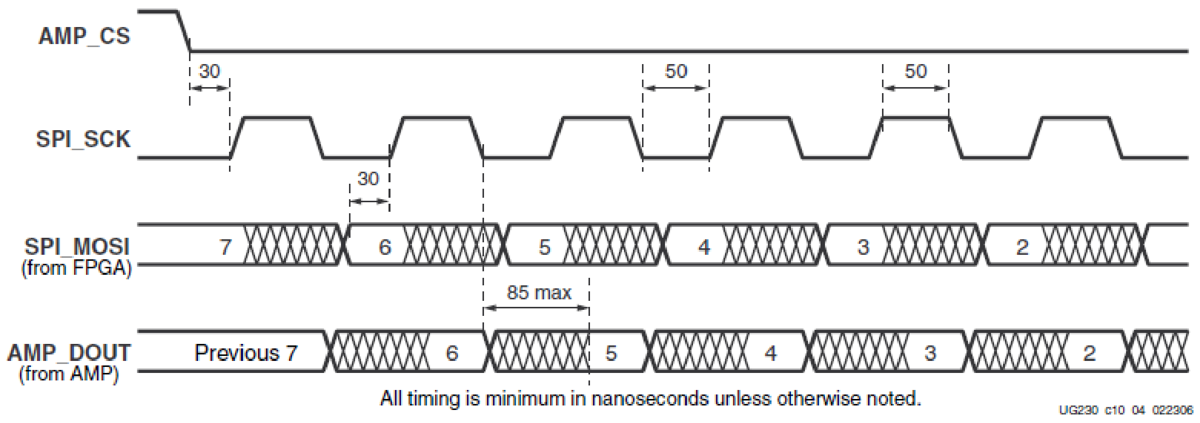


Figure-6.14. SPI Timing When Communicating with Amplifier [76,93].

6.5. Analog to Digital Converter (ADC)

6.5.1. Interface

Table-3 represents the interface signals between the FPGA and the ADC. The SPI_MOSI, SPI_MISO, and SPI_SCK signals are shared with other devices on the SPI bus. The DAC_CS signal is the active-Low slave select input to the DAC. The DAC_CLR signal is the active-Low, asynchronous reset input to the DAC [76,93].

Table-3

ADC INTERFACE SIGNALS [76,93].

Signals	**Pin FPGA**	**Direction**
SPI_SCK	U16	FPGA → ADC
AD_CONV	P11	FPGA → ADC
SPI_MISO	N10	FPGA → ADC

6.5.2. SPI Control Interface

Figure-6.15. represents the SPI bus transaction to the ADC. When the AD_CONV signal goes High, the ADC simultaneously samples both analog channels. The results of this conversion are not presented until the next time AD_CONV is asserted, a latency of one sample. The maxim sample rate is approximately 1.5 MHz. The ADC presents the digital representation of the sampled analog values as a 14-bit, two’s complement binary value [76,93].

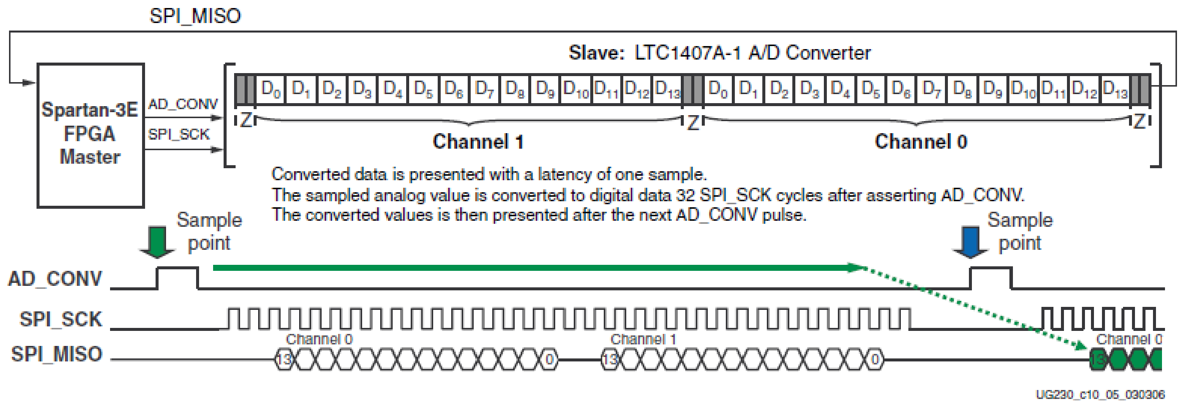

SPI_MISO
Slave: LTC1407A-1 A/D Converter
Spartan-3E FPGA Master
AD_CONV
SPI_SCK
Channel 1
Channel 0
Converted data is presented with a latency of one sample.
The sampled analog value is converted to digital data 32 SPI_SCK cycles after asserting AD_CONV.
The converted values is then presented after the next AD_CONV pulse.
Sample point
Sample point
UG230_c10_05_030306

Figure-6.15. Analog-to-Digital Conversion Interface [76,93].

Figure-6.16. shows detailed transaction timing. The AD_CONV signal is not a traditional SPI slave select enable. Be sure to provide enough SPI_SCK clock cycles so that the ADC leaves the SPI_MISO signal in the high-impedance state. Otherwise, the ADC blocks communication to the other SPI peripherals. The ADC 3-states its data output for two clock cycles before and after each 14-bit data transfer [76,93].

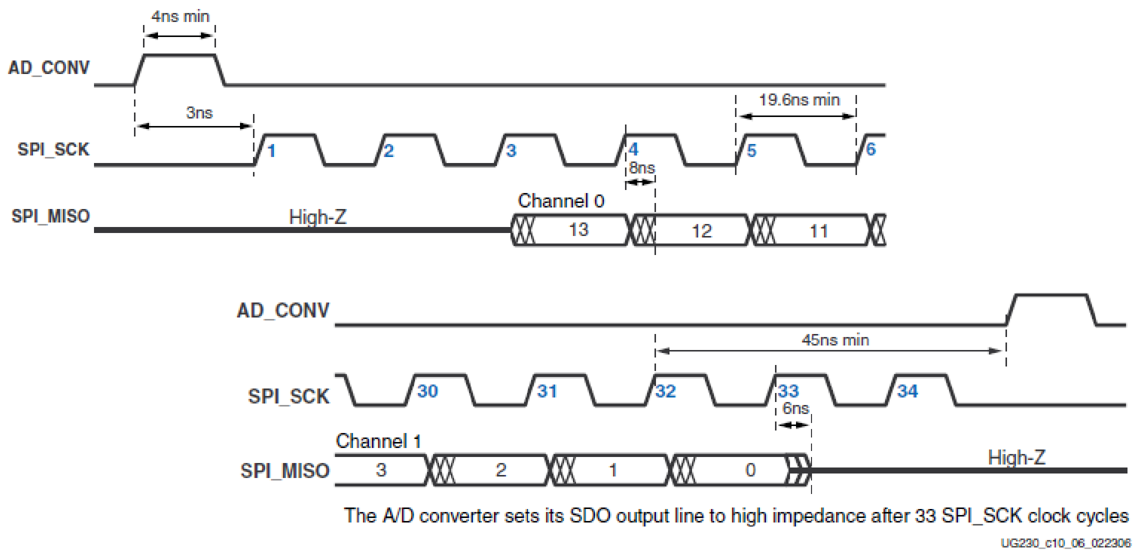


Figure-6.16. Detailed SPI Timing to ADC [76,93].

6.6. Interfacing of the Ethernet Controller [91]

6.6.1. Design Strategy

The chip associated with the Ethernet port of the "*Spartan-3E Starter Kit*" does not implement the Ethernet controller rather than the physical transceiver layer. To transmit or receive data over this Ethernet port we have interface the Ethernet controller for which we have used open core Ethernet controller from the Open Cores. The packet communication through the Ethernet controller is demonstrated by a host module [91].

6.6.2. Open Core Ethernet Controller

The controller presents three interfaces [91]

1. An interface to the PHY
2. An interface to external memory where the packets will be stored
3. A user interface that allows the set-up of the rx/tx buffer descriptors and the read/write of the various registers

To interface the PHY the tri-state i/o has to be converted into the two distinct wires. External memory is used by the controller to read or write packets. The addresses are controller by the user setting up the appropriate pointers in the buffer descriptors. This interface is known as standard wishbone interface. Here the controller is master while the memory is slave. The third interface has logically two parts-one is composed of registers and another is composed buffer descriptors. The Ethernet controller is controlled by the registers. The packet transmission is controlled by the buffer descriptors where in the external memory the packets are stored. Buffer descriptors 8-bytes long of which the first 4-bytes contain the status/control of the packet and the last 4-bytes contain the external memory address of the buffer where the packet is stored. The interface is a standard Wishbone interface. This interface is a slave, and the host is the master [91]..

6.6.3. Design

The design diagram is depicted in the following figure.

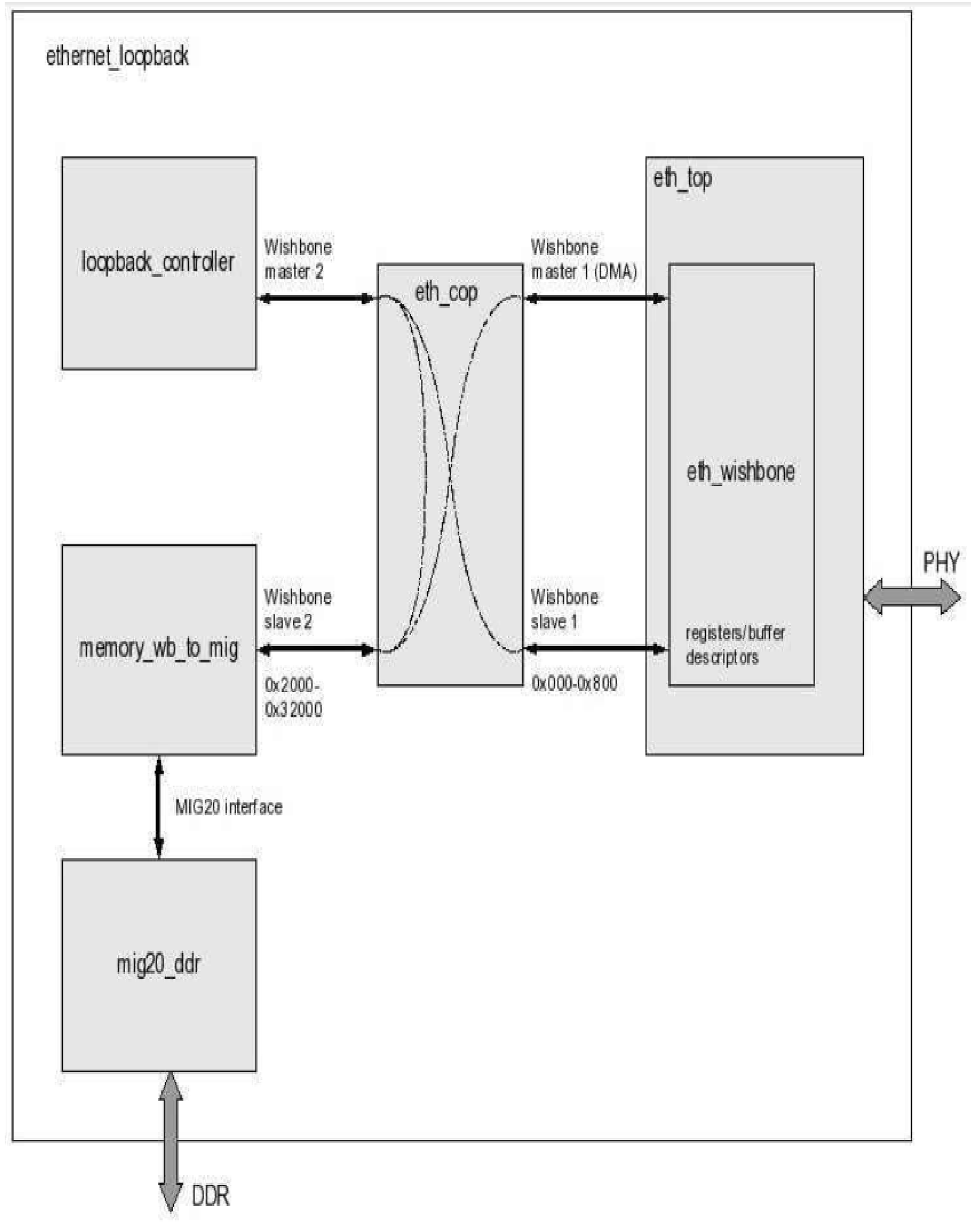


Figure-6.17. Design Diagram [91].

The top level module, “***eth_loopback”***, instantiates the host, the memory and the wishbone arbiter. The DDR SDRAM choosen as external memory for which the Xilinx MIG 2.0 controller is reused. “***memory_wb_to_mig***” module is used to convert the Wishbone to the MIG 2.0 proprietary interface. “***eth_cop***”-a simple arbiter is used to interconnect the two masters and two slaves. In order to be synthesizable the module needs very slight modifications. The arbitration provided by “***eth_cop”*** is based on the address values:

- 0x0000-0x0050: Registers (slave 1)
- 0x0400-0x05ff: Tx buffer descriptors (slave 1)
- 0x0600-0x07ff: Rx buffer descriptors (slave 1)
- 0x2000-0x32000: DDR SDRAM (slave 2)

6.6.4. Host

“loopback_controller” which is a host module reads incoming packets and send them back out without any kinds of modification. The algorithm that works behind loopback is-

```
initialize mode register
forever
initialize one rx buffer descriptor
wait until packet received in corresponding buffer descriptor
initialize tx buffer descriptor w/ same address as rx buffer and same length
endforever
```

For mode register the defaults parameters cannot be used as mode register needs its tx and rx bits set in order to enable receive or transmit. Pointer and control bits constitute the buffer descriptors. The pointer is nothing but the DDR SDRAM address of the packet data. Some control bits are determined by the user while some are determined by the Ethernet core. The most important control bit is the enable bit. In a rx buffer descriptor an enable bit set to 1 indicates to the core that it can write the buffer descriptor and its associated buffer for packet reception. Once the packet is received, the core resets the enable bit, and the user can read the received packet and its status. The tx buffer descriptors contain a similar enable bit. The user sets it to tell the core to send a packet. Once sent, the core resets this bit indicating that the packet was sent [91].

6.7. Remote Data Acquisition System

The block diagram of the "*Remote Data Acquisition System*" has been shown. Our developed DAQ will acquire the data from the surroundings or radioactive areas in terms of voltage, temperature, pressure etc. Then these acquired analog data will be processed to the FPGA through the data conversion from analog domain to digital domain using the ADC. Ethernet port of the "Spartan-3E Starter Kit" has been interfaced in such a way thus the FPGA could process the acquired data to the user's PC who intended to acquire it. The acquired data could be transmitted to remote place using internet. Our main aim of doing this project is to make the use of the "Ethernet Port" of the "Spartan-3E Starter Kit" thus we could use the internet for transmitting the data to a remote place. For checking the workability of the networking of ***Data Acquisition System,*** we have designed a network as depicted in the **Figure-6.18.** which is simulated using the ***CISCO Packet Tracer 5.*** The simulation was carried out in the real time simulation mode in which the clock should run continuously. The simulation results and along with the network design have been represented here respectively.

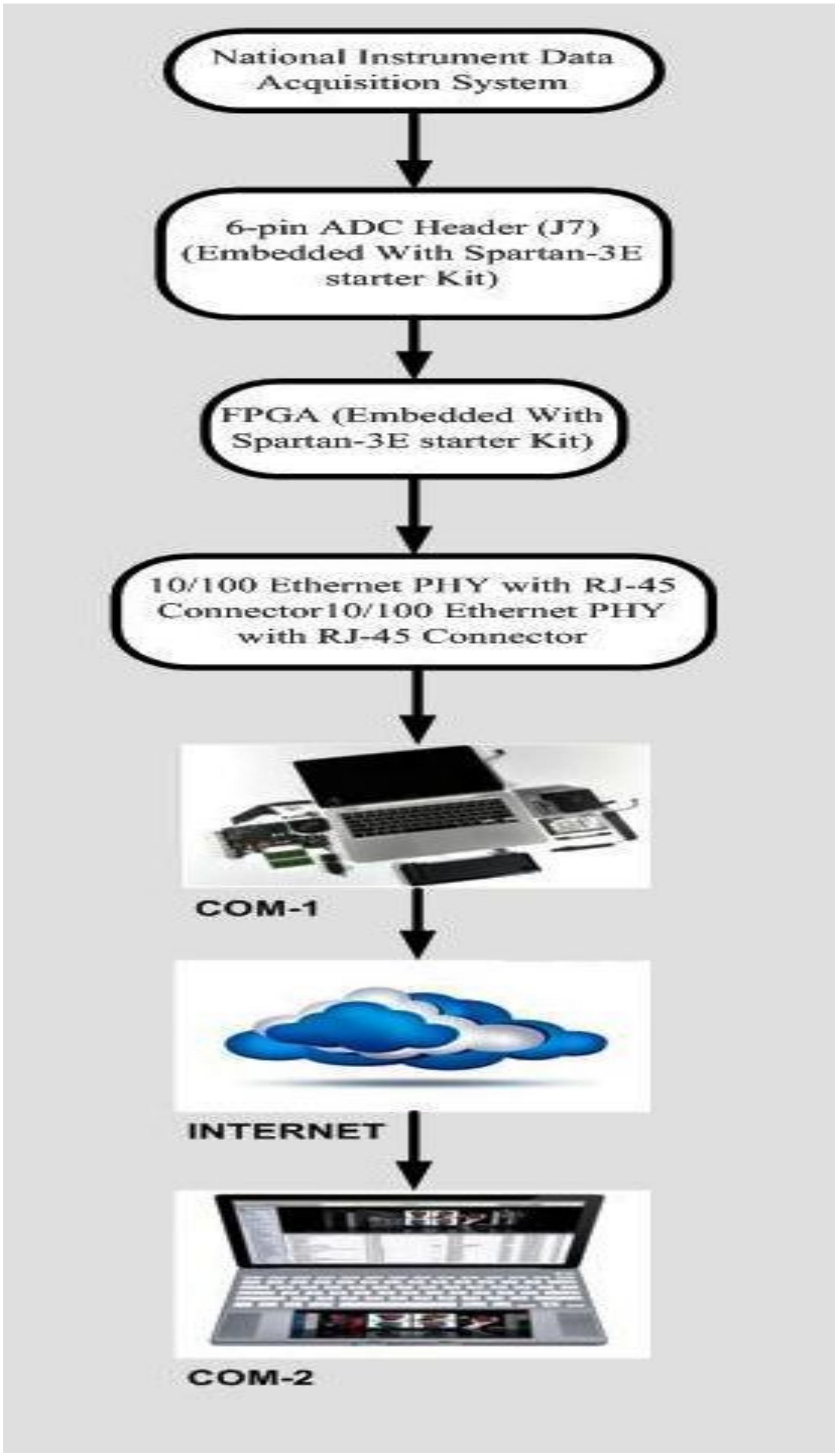

National Instrument Data Acquisition System
6-pin ADC Header (J7) (Embedded With Spartan-3E starter Kit)
FPGA (Embedded With Spartan-3E starter Kit)
10/100 Ethernet PHY with RJ-45 Connector10/100 Ethernet PHY with RJ-45 Connector
COM-1
INTERNET
COM-2

Figure-6.18. Remote High Speed Data Acquisition System

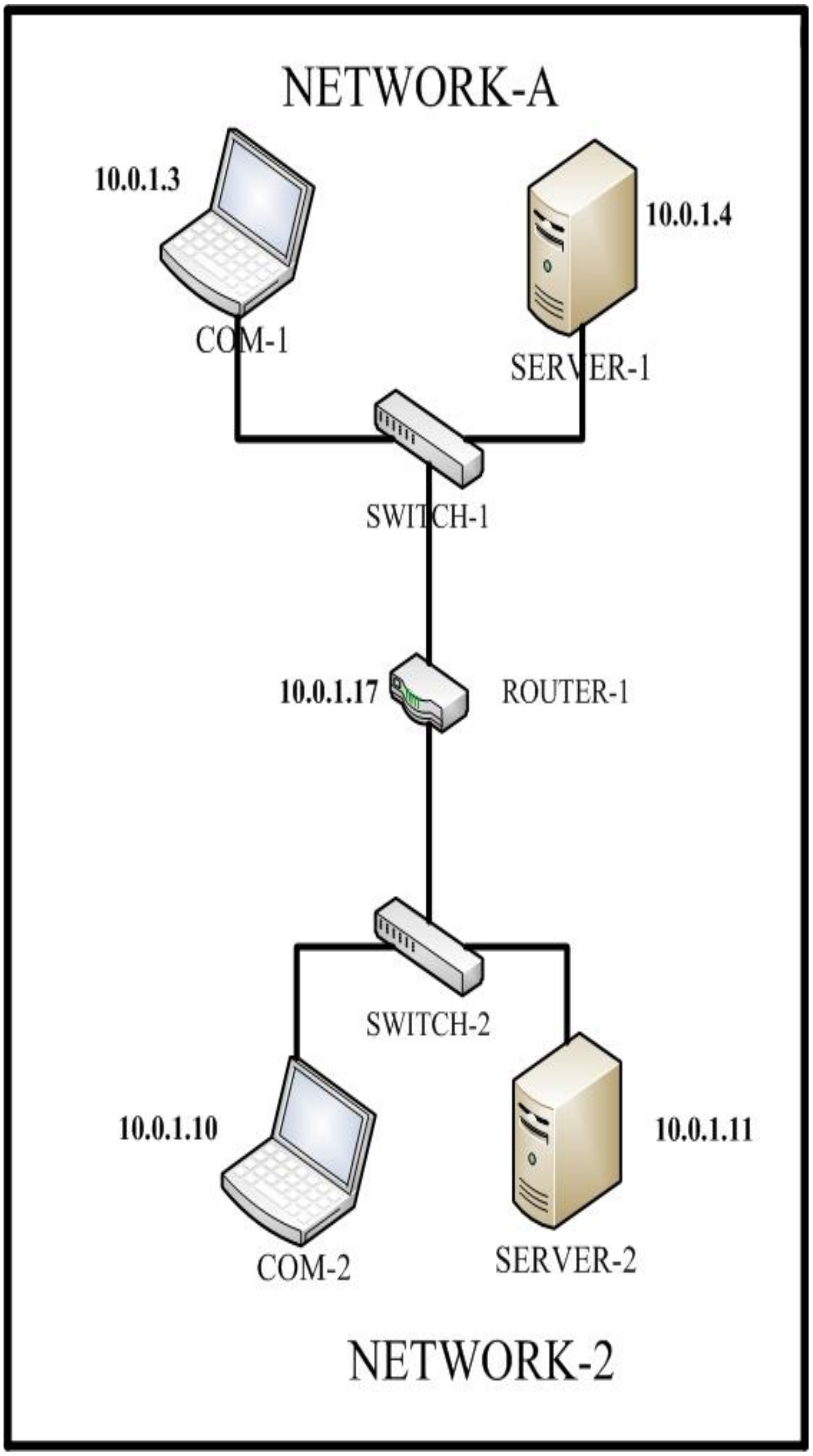
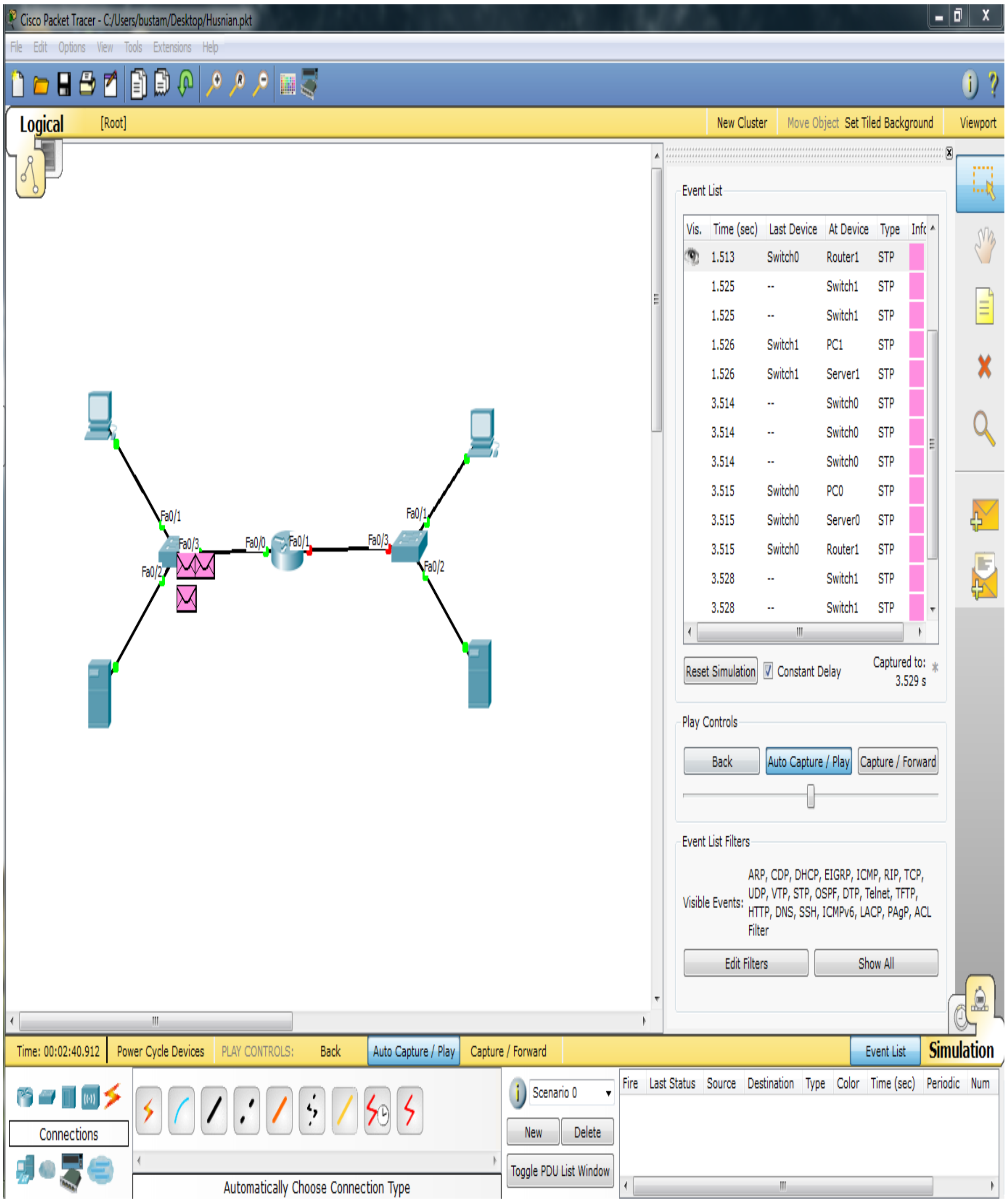


Figure-6.19. Network Architecture

Table.4. Network Design

Subnets	***Subnet address***	***Prefix***	***Mask***	***Hosts***	***Default Gateway (First address)***	***Broadcast (last address)***
NET- 1	10.0.1.0	29	248	8	10.0.1.1	10.0.1.7
NET- 2	10.0.1.8	29	248	8	10.0.1.9	10.0.1.15
Point-to-point (Router)	10.0.1.16	30	252	4	Address usable for point-to point (10.0.1.17 10.0.1.18)	



Vis.	Time (sec)	Last Device	At Device	Type	Info
👁	1.513	Switch0	Router1	STP	🟪
	1.525	--	Switch1	STP	🟪
	1.525	--	Switch1	STP	🟪
	1.526	Switch1	PC1	STP	🟪
	1.526	Switch1	Server1	STP	🟪
	3.514	--	Switch0	STP	🟪
	3.514	--	Switch0	STP	🟪
	3.514	--	Switch0	STP	🟪
	3.515	Switch0	PC0	STP	🟪
	3.515	Switch0	Server0	STP	🟪
	3.515	Switch0	Router1	STP	🟪
	3.528	--	Switch1	STP	🟪
	3.528	--	Switch1	STP	🟪

Captured to: 3.529 s

Visible Events: ARP, CDP, DHCP, EIGRP, ICMP, RIP, TCP, UDP, VTP, STP, OSPF, DTP, Telnet, TFTP, HTTP, DNS, SSH, ICMPv6, LACP, PAgP, ACL Filter

Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num

Figure-6.20. Simulation Using the ***CISCO Packet Tracer-5.3***

CHAPTER 7

RESULT ANALYSIS AND DISCUSSION

7.1. Result Analysis

7.1.1. DAC Implementation on FPGA

The developed VHDL code has been implemented on the Spartan-3E starter kit. The design is synthesized and implemented on the FPGA by using the Xilinx ISE 13.2 software. The kit includes embedded USB based programming logic and an USB end point with a Type B connector [6]. With the help of a USB cable which has connection with the PC and iMPACT programming software we burned the program directly into the FPGA.

Figure-7.1. Experimental Setup

After burning code we have taken different reading using digital multi meter. The taken reading is listed below in the table 4 and table 5. We actually compare the value from the DAC output with the reading displayed on the LCD screen. Following tables show the reading for voltage step 100 mv and 10 mv.

7.1.2. Tables and Figures

TABLE -5

FOR 100mV Step

S/N.	**LCD Value (Volt)**	**Multimeter (Volt)**
1	0.092	0.09
2	0.192	0.19
3	0.292	0.29
4	0.392	0.39
5	0.492	0.48
6	0.592	0.59
7	0.692	0.68
8	0.792	0.78
9	0.892	0.88
10	0.992	0.98
11	1.194	1.17
12	1.294	1.27
13	1.394	1.37
14	1.494	1.47
15	1.594	1.55
16	1.694	1.67
17	1.794	1.77
18	1.894	1.86

19	1.994	1.97
20	2.096	2.06
21	2.196	2.18
22	2.296	2.27
23	2.396	2.36
24	2.496	2.49

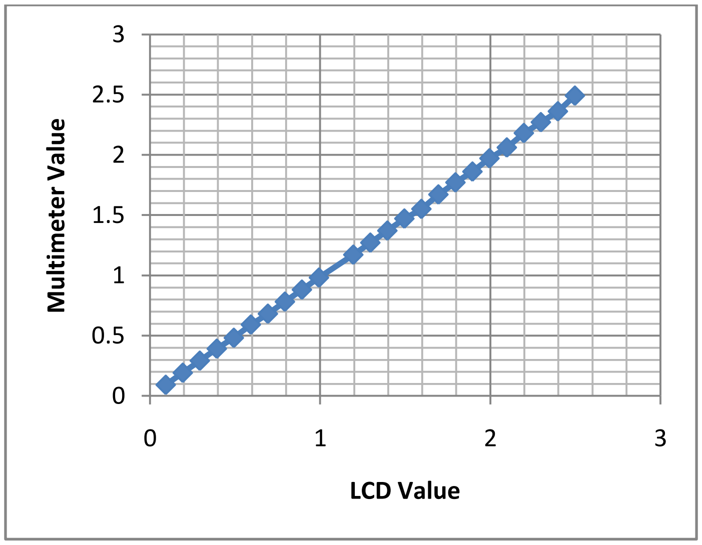


Figure-7.2. For 100mV

TABLE -6

FOR 10mV Step

S/N.	**LCD Value (Volt)**	**Multimeter Value (Volt)**

1	2.408	2.37
2	2.418	2.37
3	2.428	2.40
4	2.436	2.39
5	2.446	2.40
6	2.456	2.41
7	2.464	2.43
8	2.474	2.43
9	2.484	2.44
10	2.494	2.48

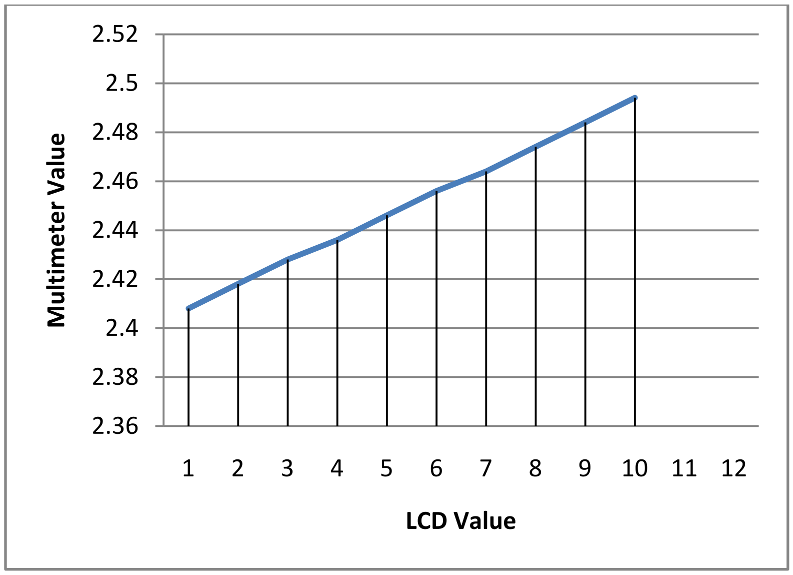
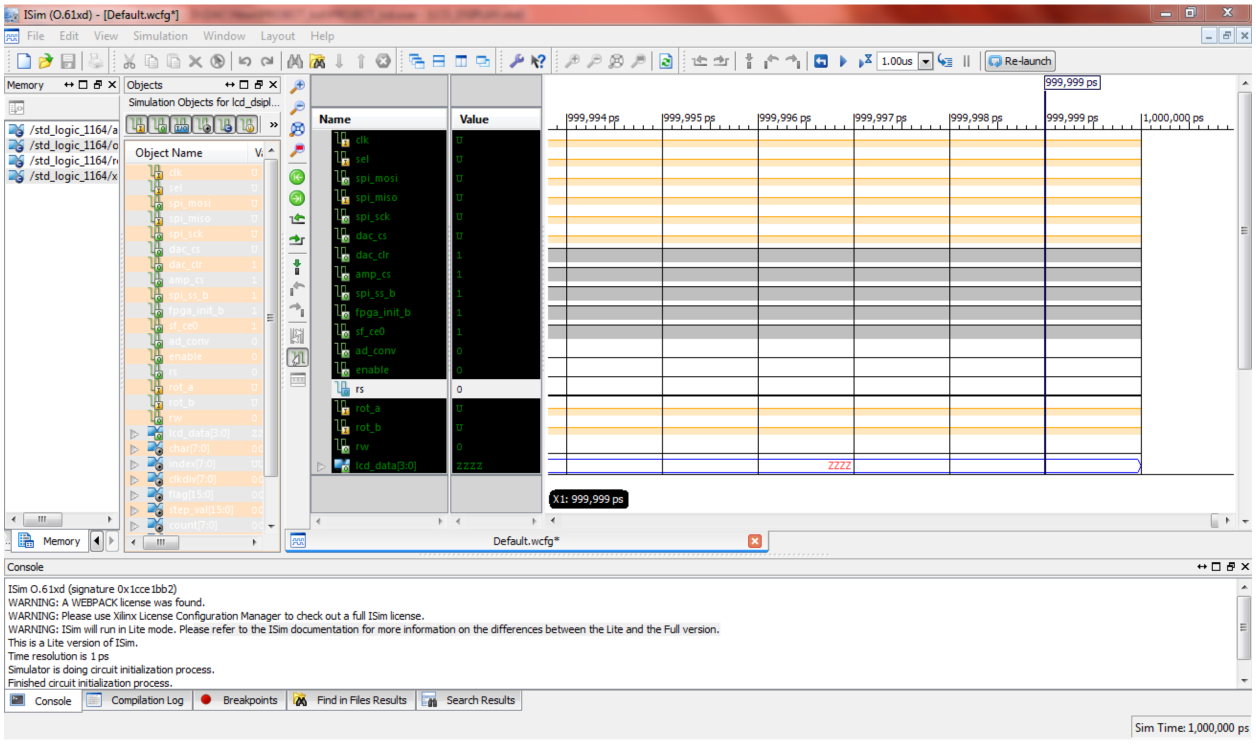


Figure-7.3. For 10mV

We have implemented this system in the Lab. There are some deviations in the reading of the multi meter to the LCD display reading. This may happen in the time of conversion from digital to analog. But the matter of the fact is that the deviation is not great. So we can say that our developed system is almost an accurate and ideal for use for various purposes.

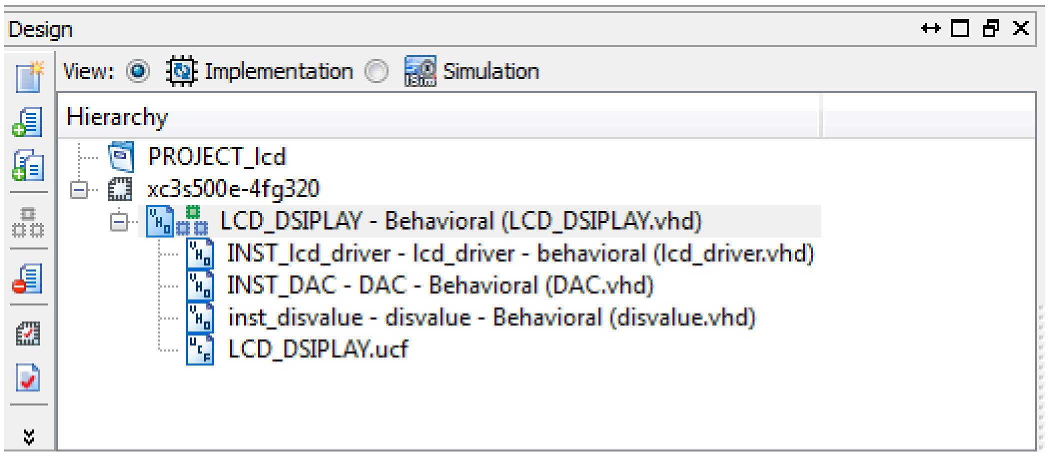
7.1.3. Simulation Results



Name	Value
clk	U
sel	U
spi_mosi	U
spi_miso	U
spi_sck	U
dac_cs	U
dac_clr	1
amp_cs	1
spi_ss_b	1
fpga_init_b	1
sf_ce0	1
ad_conv	0
enable	0
rs	0
rot_a	U
rot_b	U
rw	0
lcd_data[3:0]	ZZZZ

```
ISim O.61xd (signature 0x1cce1bb2)
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

Figure-7.4. Simulation



```
PROJECT_lcd
xc3s500e-4fg320
  LCD_DSIPLAY - Behavioral (LCD_DSIPLAY.vhd)
    INST_lcd_driver - lcd_driver - behavioral (lcd_driver.vhd)
    INST_DAC - DAC - Behavioral (DAC.vhd)
    inst_disvalue - disvalue - Behavioral (disvalue.vhd)
    LCD_DSIPLAY.ucf
```

Figure-7.5. Behavioral

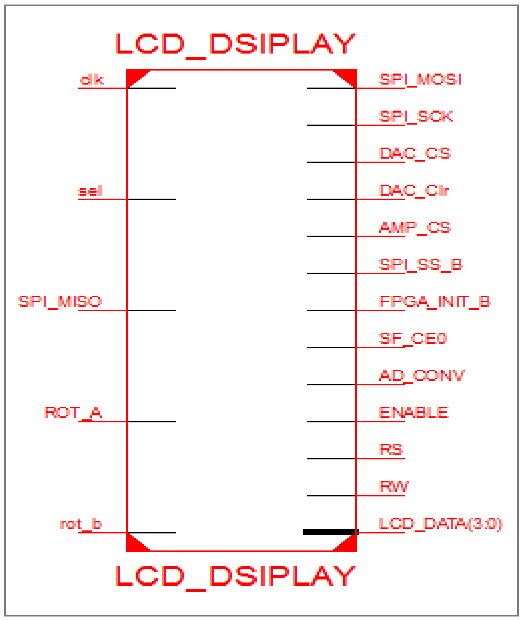
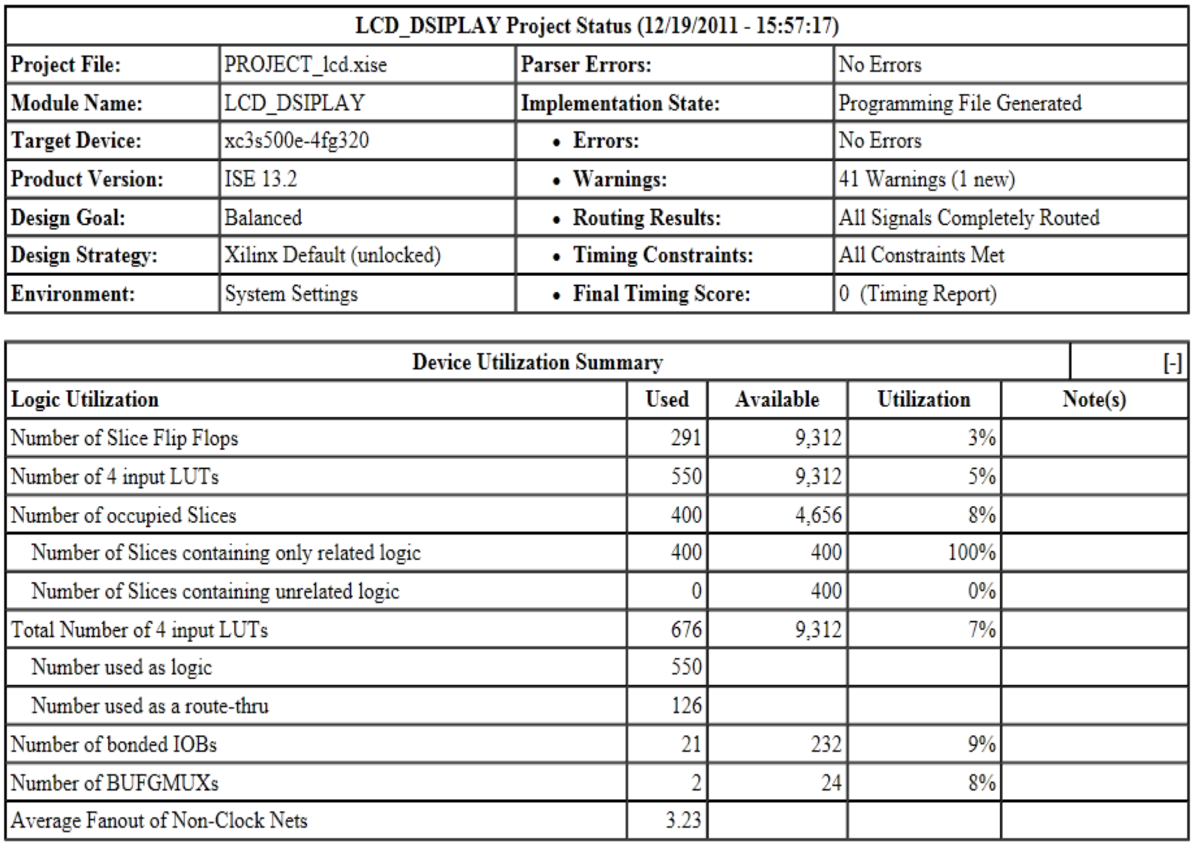


Figure-7.6. RTL Schematic



LCD_DSIPLAY Project Status (12/19/2011 - 15:57:17)			
Project File:	PROJECT_lcd.xise	**Parser Errors:**	No Errors
Module Name:	LCD_DSIPLAY	**Implementation State:**	Programming File Generated
Target Device:	xc3s500e-4fg320	• **Errors:**	No Errors
Product Version:	ISE 13.2	• **Warnings:**	41 Warnings (1 new)
Design Goal:	Balanced	• **Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	All Constraints Met
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	291	9,312	3%	
Number of 4 input LUTs	550	9,312	5%	
Number of occupied Slices	400	4,656	8%	
Number of Slices containing only related logic	400	400	100%	
Number of Slices containing unrelated logic	0	400	0%	
Total Number of 4 input LUTs	676	9,312	7%	
Number used as logic	550			
Number used as a route-thru	126			
Number of bonded IOBs	21	232	9%	
Number of BUFGMUXs	2	24	8%	
Average Fanout of Non-Clock Nets	3.23			

Figure-7.7. Design Summary

7.2. Measure Light Intensity using Photodiode

Light intensity is an important management factor for breeder type poultry. The amount of light received by a plant is an important factor, because plants' growth may suffer if they do not receive sufficient light. Light intensity decreases dramatically with distance from the light source. An object located 15 cm from two 40-watt fluorescent lamps receive about 9300 lux, but only 5400 lux if they are 30 cm from the lamps. The human eye is a very poor "instrument" for measuring light intensity, because the pupil adjusts constantly in response to the amount of light it receives. To accurately measure the light intensity in a given spot, it is best to use a light meter. Light intensity may be measured in lux (metric system) or foot-candles (Imperial system).

1 foot-candle = 10.76 lux

7.2.1. Photodiode BPW34 as light sensor

The Photodiode used is the BPW34. This is a high speed and high sensitive silicon PIN photodiode in a miniature flat plastic package. A photodiode is designed to be responsive to optical input. Due to its water clear epoxy the device is sensitive to visible and infrared radiation. The large active area combined with a flat case gives a high sensitivity at a wide viewing angle.

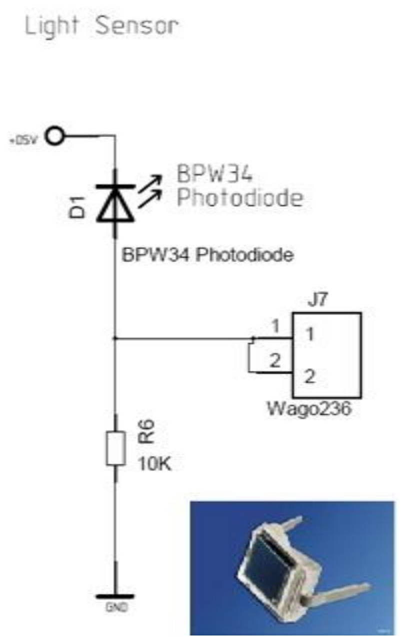


Figure-7.8. Photodiode BPW34

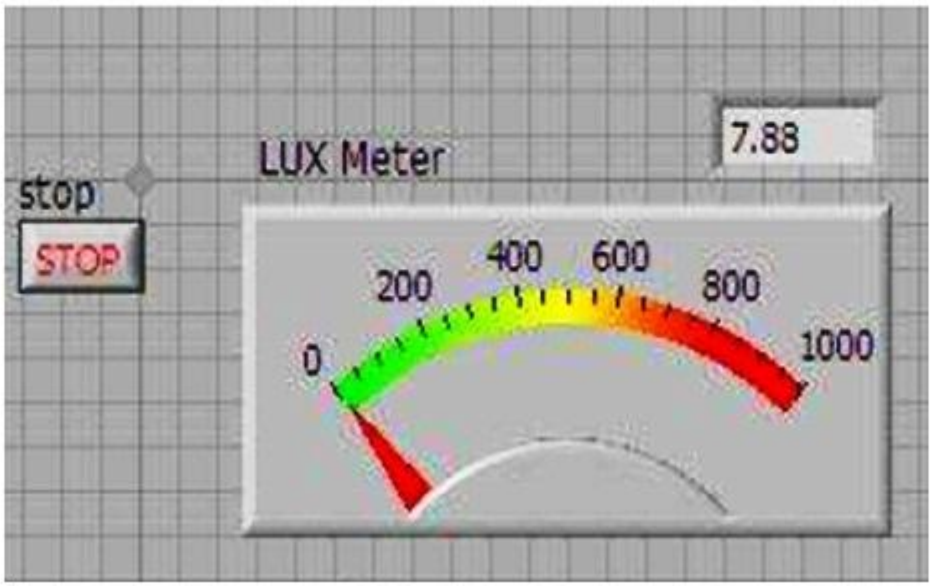


Figure-7.9. Develop a LabVIEW VI that reads the voltage across the resistor and converts the voltage to lux.

7.2.2. Build a Light Intensity Logger

- Reads the voltage across the resistor and converts the voltage to lux
- Adjustable Update Period
- Statistics of Maximum, Minimum and Mean Value
- Intensity plot over time (with Reset)
- A selectable and adjustable Threshold (Limit) and a warning if light is not in Range

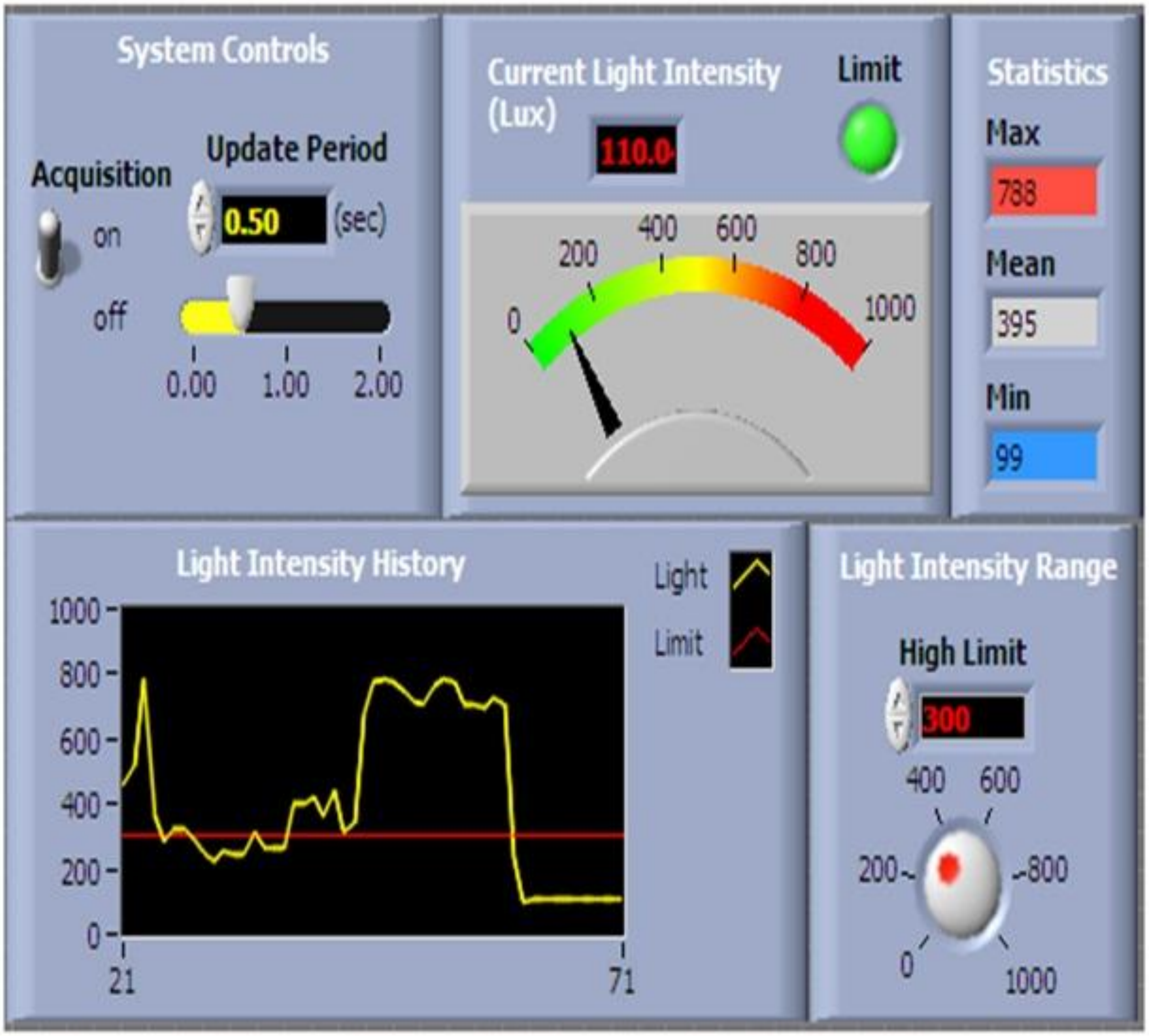
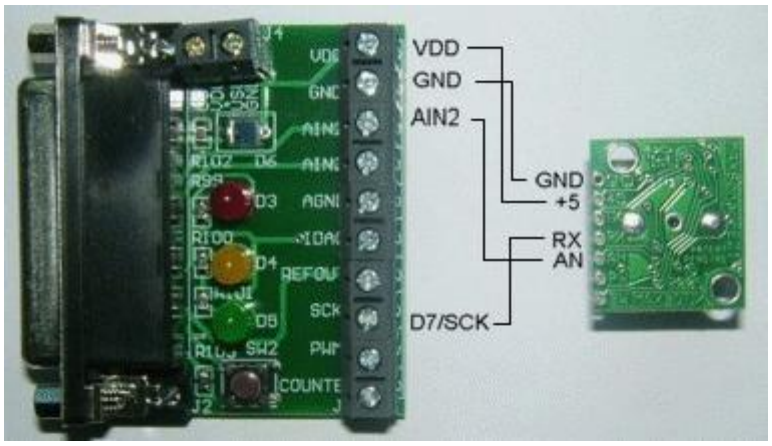


Figure-7.10. Front Panel View (Light Intensity Logger)

7.3. Measure Distance using the Ultrasonic Sensor

The **MaxSonar-EZ1** uses an ultrasonic sensor to detect objects from 6-inches to 254-inches with a 1-inch resolution. The analog output is given as 10mV/inch. You can connect the sensor to the analog IO module as:



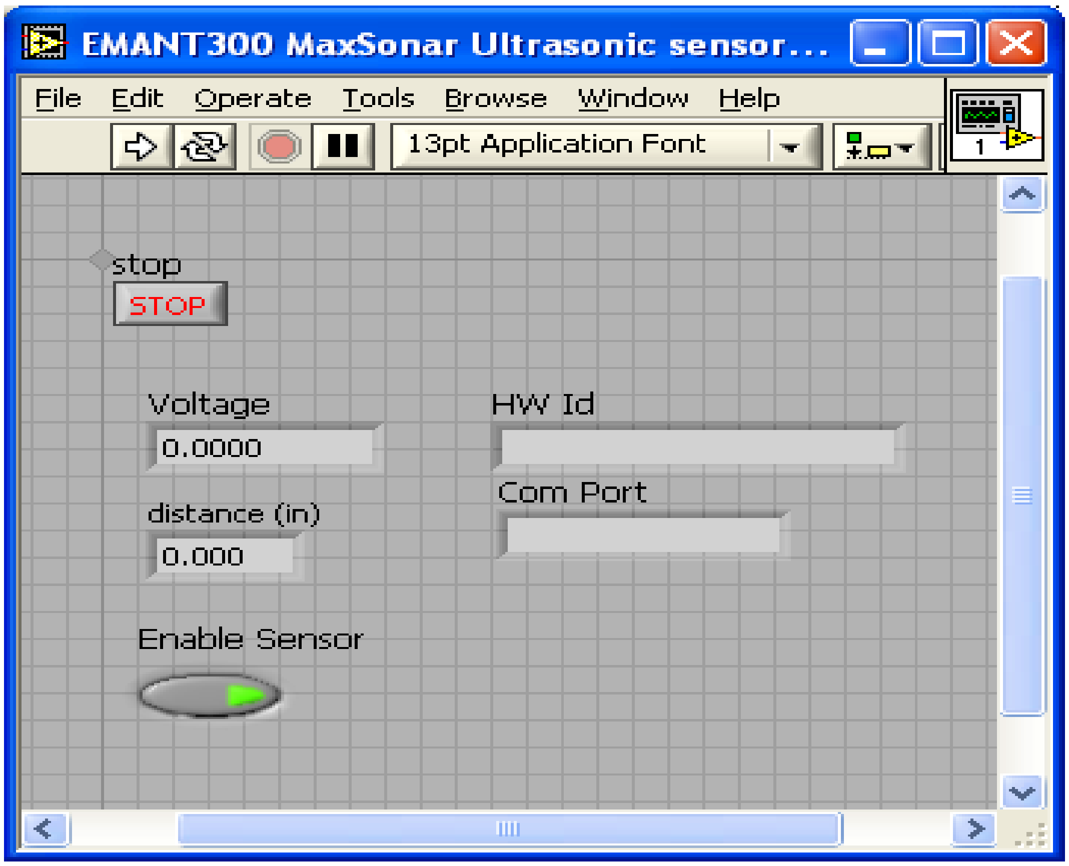


Figure-7.11. Front View (Ultrasonic Sensor)

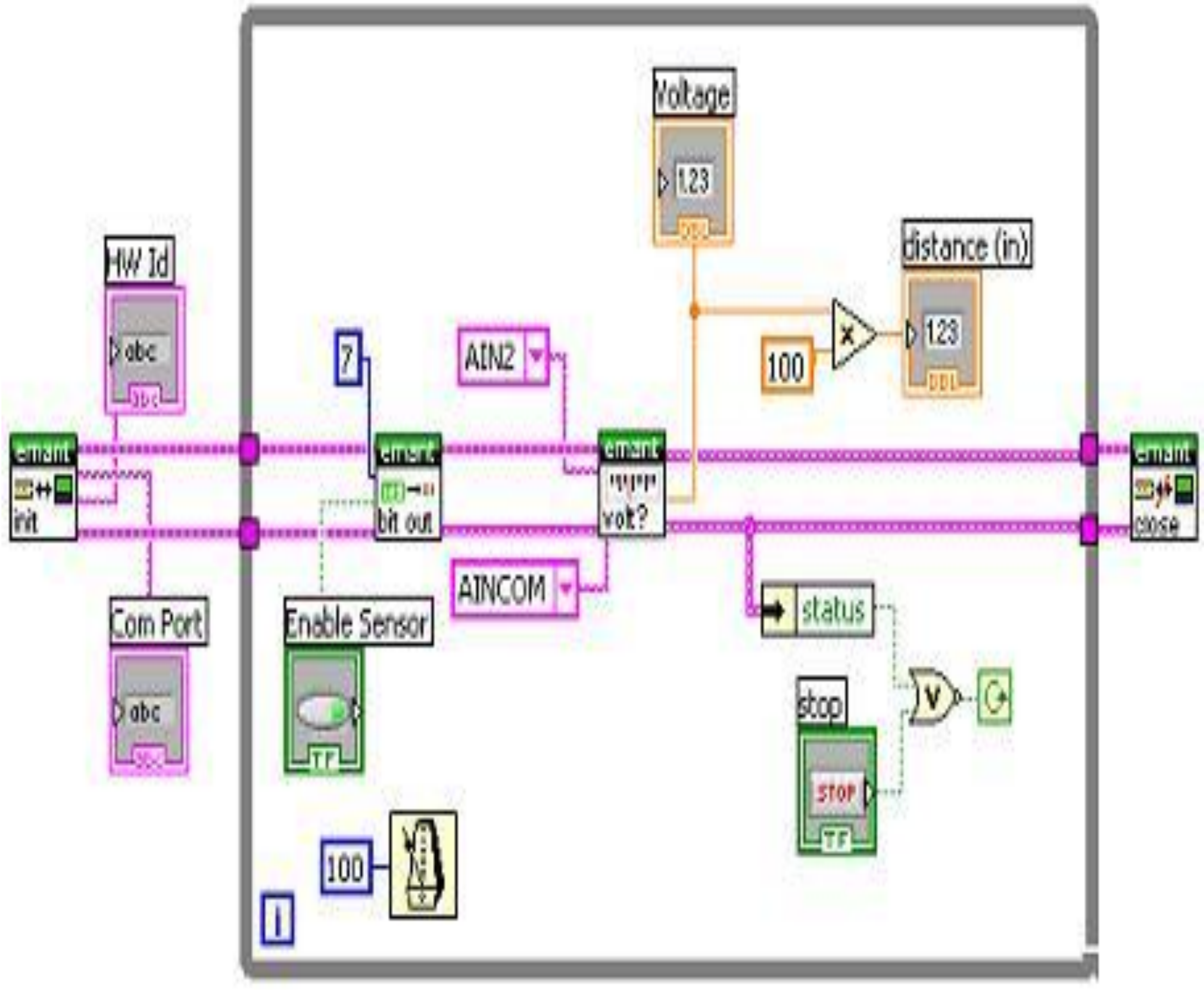


Figure-7.12. Block Diagram (Ultrasonic Sensor)

7.4. Temperature Simulation and Calibration

- Generate a Random Number (Temperature) between 20° to 50 °C
- This number is interpreted as Temperature
- Add a timing (Wait function) for a temperature measurement of 5 Hz
- (→ One temperature every 200 ms)

- Display the Temperature curve in a Chart

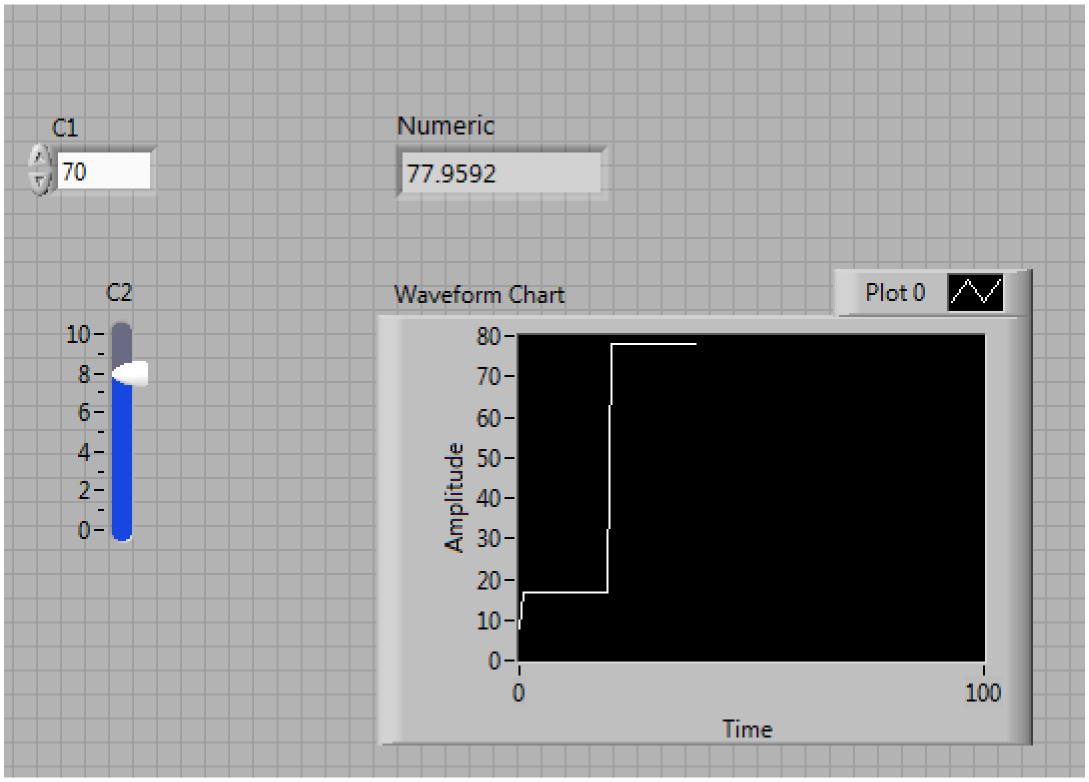


Figure-7.13. Front Panel View (Temperature)

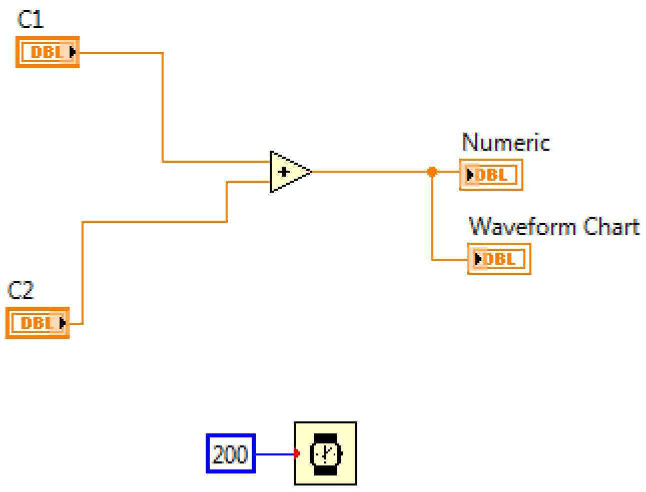


Figure-7.14. Block Diagram (Temperature)

7.5. Optical Heart Rate Monitor

Pulse Oximetry is a non invasive method of measuring a person's oxygenation level which monitors the percentage of haemoglobin (Hb). A probe comprising LEDs and photo detector is attached to the patient's finger. As the light from the two LEDs (red - 660 nm and infrared - 950 nm) pass through the body tissues to a photodetector, it is absorbed by blood and soft tissue. The light absorption rate at the two wavelengths by the hemoglobin is different and depends on the degree of oxygenation. The light level changes as the blood is pumped by the heart. As a consequence, the oximeter also measures the heart rate in beats per minute (BPM). In this application note, we have created a heart rate monitor by using one IR LED & phototransistor pair and observing the waveform at the phototransistor output. This is intended for illustrating a typical light sensor application and not intended for actual medical use.

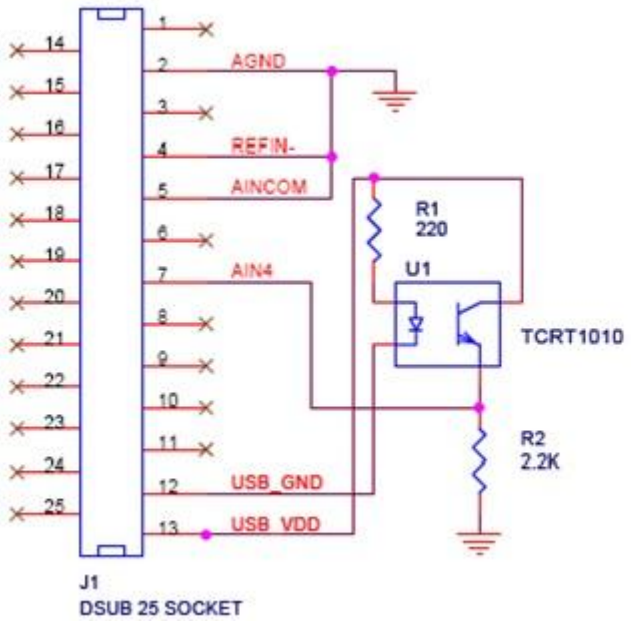
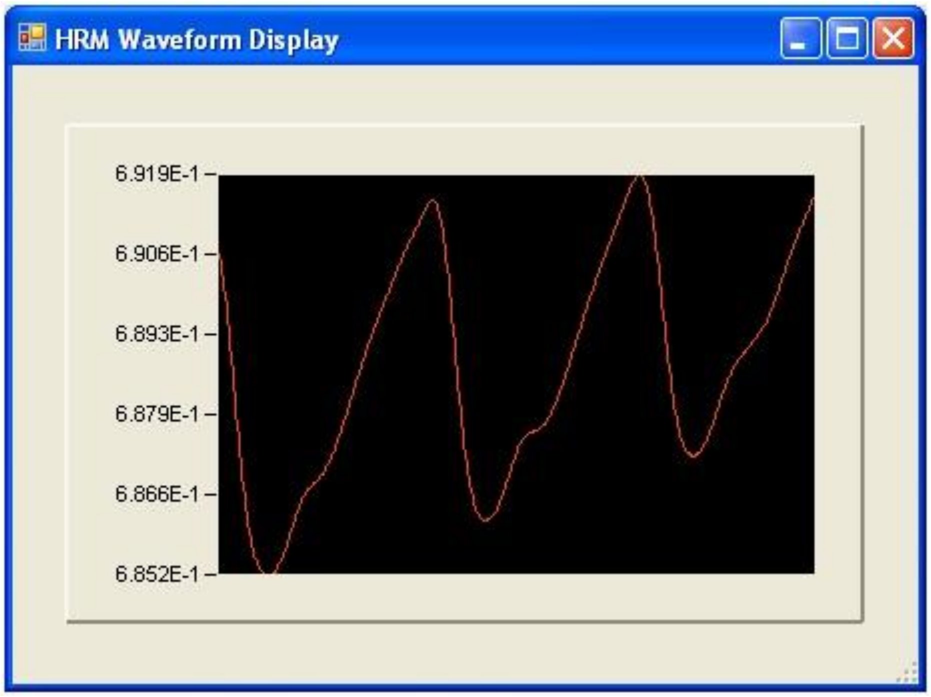


Figure-7.15. Optical heart rate monitor schematic

Figure-7.16. IR emitter and detector encased in Velcro



When the console program is first run, the user to place his/her finger on the sensor. When a finger is placed on the phototransistor, the voltage read rises above 0.3V. If a good contact is made, a sinusoidal type signal is observed. As the signal rides on a fluctuating DC signal, a simple differential signal is created. From maths, we know that when a max or min is reached, its differential value is zero. After this an autocorrelation function is applied and the peaks extracted to obtain the heart rate. When the signal is noisy, a wrong BPM will be calculated. Any count that is obviously wrong is ignored. This is all taken care of by the HRM class.

7.6. Discussion

7.6.1. DAC

We have implemented this system in the Lab. There are some deviations in the reading of the multi meter to the LCD display reading. This may happen in the time of conversion from digital to analog. But the matter of the fact is that the deviation is not great. So we can say that our developed system is almost an accurate and ideal one which could be used for various purposes.

7.6.2. DAQs

Using our developed "*Data Acquisition System*" we have performed four experiments. Those are- Measure Light Intensity using Photodiode, Measure Distance using the Ultrasonic Sensor, Temperature Simulation and Calibration, and Optical Heart Rate Monitor. And in each case we have got the values nearly our expected one.

REFERENCES

[1] M. Michael Vai David R. Martinez, Robert A. Bond. High Performance Embedded Computing Handbook. CRC Press, 2008.

[2] Jonathan Rose Ian Kuon, Russel Tessier. Fpga architecture: Survey and challenges. Foundations and Trends in Electronic Design Automation, 2007.

[3] Volnei A. Pedroni, Circuit Design with VHDL, 1st Edition, MIT Press, Cambridge, MA, U.S.A., ISBN-978-81-203-2683-5.

[4] M. Morris Mano, Digital Logic and Computer Design,Prentice Hall College Div (April 1979), ISBN-13: 978-0132145107.

[5] Peter H. Sydenham and Richard Thorn., Handbook of Measuring System Design © John Wiley & Sons, Ltd, 2005.

[6] http://www.omega.com/das/ , 5th January, 2012.

[7] http://cp.literature.agilent.com/litweb/pdf/5950-3000.pdf, 5th January, 2012.

[8] http://www.techbriefs.com/component/content/article/14608, 7th January, 2012.

[9] J.B. Dixit, Amit Yadav, Intelligent Instrumentation for Engineers , Laxmi Publications Pvt Limited, Sep 1, 2011 - 183 pages.

[10] http://www.ni.com/data-acquisition/, 8th January, 2012.

[11] http://www.qedenv.com/products/remote_wireless_data_acquisition_overview, 8th January, 2012.

[12] http://www.teledesignsystems.com/, 8th January, 2012.

[13] http://www.ti.com/lit/ug/sbou056/sbou056.pdf , 8th January, 2012.

[14] http://www.abs-cbnnews.com/exam-results/05/31/12/marine-engineer-licensure-exams-results-released , 9th January, 2012.

[15] http://www.bartington.com/das1-data-acquisition-system.html, 10th January, 2012.

[16] http://www.home.agilent.com/en/pc-1000000676%3Aepsg%3Apgr/data-acquisition-daq?&cc=BD&lc=eng, 11th January, 2012.

[17] http://books.google.com.bd/embedded_Systems_Design.html?id=Nr_esc=y, 17th January, 2012.

[18] Yi Gao, Shilang Tang, Zhangli Ding, "Comparison between CISC and RISC", 2000.

[19] Steve Heath, "Microprocessor Architectures: RISC, CISC and DSP", Second Edition, 1995.

[20] William Stallings, "Computer Architecture Designing for Performance", "Seventh Edition", 2006.

[21] John L. Hennessy, David A. Patterson, "Computer Architecture A Quantitative Approach", Third Edition, 2006.

[22] http://en.wikipedia.org/wiki/Complex_instruction_set_computing , 19th January, 2012.

[23] http://en.wikipedia.org/wiki/Intel_Core, 19th January, 2012.

[24] Marvin Hobbs, "RISC/CISC Development and Test Support". Prentice Hall (September 1991), **ISBN-13:** 978-0133884142.

[25] http://en.wikibooks.org/wiki/_/Instruction_Set_Architectures , 25th January.

[26] Jeff Prosise, "RISc vs. CISC: The Real Story", PC Magazine, 1995.

[27] Nouf Assad, Microprocessors, History of Computing.

[28] http://en.wikipedia.org/wiki/Field-programmable_gate_array, 2nd February, 2012.

[29] http://www.altera.com/products/fpga.html, 2nd February, 2012.

[30] http://www.webopedia.com/TERM/F/FPGA.html , 3rd February, 2012.

[31] http://www.latticesemi.com/products/fpga/index.cfm, 4th February, 2012.

[32] http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html, 7th February, 2012.

[33] Scott Hauck, André DeHon, Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation, Morgan Kaufmann, 2008 , ISBN 0123705223, 9780123705228.

[34] http://www.up-tech.com/eng/product/detail.asp?id=35, 15th February,2012.

[35] http://www.xilinx.com/products/boards/s3estarter/reference_designs.htm,15th February,2012.

[36] http://parts.digikey.com/1/parts/1366679-kit-starter-spartan-3e-hw-spar3e-sk-uni-g.html, 15th February,2012.

[37] http://www.digilentinc.com/Products/ =2,400,792&Prod=S3EBOARD, 15th February,2012.

[38] http://www.amazon.com/RS232-SERIAL-Adapter-CABLE-PL2303/dp/B00404N0IQ?SubscriptionId=AKIAILPV47KBBQYSR3TQ&tag=fataom0220&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B00404N0IQ , 17th February, 2012.

[39] Peter J. Ashenden, The Designer's Guide to VHDL, 2nd Edition, Elseviewer, ISBN-1-55860-674-2.

[40] Ulrich Heinkel, Martin Padeffke, Werner Haas, Thomas Buerner, Herbert Braisz, Thomas Gentner, Alexander Grassmann, The VHDL Reference, 1st Edition, John Wiley ans Sons. Ltd, ISBN-0-47189972-0.

[41] Yalamanchili S., VHDL Starter's Guid, Englewood Cliffs, NJ, Prentice Hall, 1998.

[42] Yalamanchili S., Introductory VHDL from Simulation to Synthesis, Englewood Cliffs, NJ, Prentice Hall, 2001.

[43] Perry D.L., VHDL, New York, McGraw-Hill, 2nd Edition, 1994.

[44] Pellerin D. and D. Taylor, VHDL Made Easy, Englewood Cliffs, NJ, Prentice Hall, 1997.

[45] Navabi Z., VHDL Analysis and Modeling of Digital System, New York, McGraw-Hill, 2nd Edition, 1993.

[46] Naylor D. and S. jones, VHDL : A Logic Synthesis Approach, London : Chapman and Hall, 1997.

[47] Hamblen J. and M. Furman, Rapid Prototyping of Digital Systems, Boston : Kluwer Academic Publisher, 2nd Edition, 2001.

[48] Chang K. C., Digital System Design with VHDL and Synthesis-An Integrated Approach, Los Alamitos,CA, IEEE Computer Society Press, 1999.

[49] Bhasker J., VHDL Primer, Englewood Cliffs, NJ, Prentice Hall, 3rd Edition, 1999.

[50] Armstrong J. R. and F. G. Gray, VHDL Design Representation and Synthesis, Englewood Cliffs, NJ, Prentice Hall, 2rd Edition, 2000.

[51] Ian Grout, Digital Systems Design with FPGAs and CPLDs, Else viewer, 2008. ISBN-978-81-312-1865-3.

[52] Electronic Industries Association, Standard Data Transfer Format between Data Preparation System and Programmable Logic Device Programmer, JEDEC Standard and JESD3-C, EIA, Washington, DC, 1994.

[53] Peter Lee Stephanie McBader. An fpga implementation of a flexible, parallel image processing architecture suitable for embedded vision system. IEEE, 2003.

[54] Peter Alfke and Bernie New. Implementing state machines in LCA devices. In *The Programmable Logic Data Book*, pages 8–169 – 8–172. Xilinx, Inc., San Jose, CA, 2nd edition, 1994. XAPP 027.001.

[55] Carol A. Fields. Proper use of hierarchy in HDL-based high density FPGA design. In Will Moore and Wayne Luk, editors, *Field-Programmable Logic and Applications: 5th International Workshop, FPL '95*, volume 975 of *Lecture Notes in Computer Science*, pages 168–177, Berlin, Germany, August 1995. Springer Verlag.

[56] IEEE. *IEEE Standard VHDL Language Reference Manual*. IEEE, New York, NY, 1988. IEEE Standard 1076-1987.

[57] IEEE. *IEEE Standard Multi value Logic System for VHDL Model Interoperability (std_logic_1164)*. IEEE, New York, NY, 1993. IEEE Standard 1164-1993.

[58] Manfred Selz. *Untersuchungen zur synthesegerechten Verhaltensbeschreibung mit VHDL*. PhD thesis, Universita ̈t Erlangen-Nu ̈ rnberg, Erlangen, Germany, March 1994.

[59] Synopsys. *Design Compiler Family Reference*. Synopsys, Inc., Mountain View, CA, April 1995. (Version 3.3a).

[60] Synopsys. Finite state machine tutorial source code. April 1995. (Version 3.3a).

[61] Synopsys. *Finite State Machines–Application Note*. Synopsys, Inc., Mountain View, CA, April 1995.

[62] Synopsys. *VHDL Compiler Reference*. Synopsys, Inc., Mountain View, CA, April 1995. (Version 3.3a).

[63] Synopsys. *VSS Reference*. Synopsys, Inc., Mountain View, CA, April 1995. (Version 3.3a).

[64] Viewlogic Systems. *Using Powerview*. Viewlogic Systems, Inc., Marlboro, MA, 1994.

[65] Xilinx. *The Programmable Logic Data Book*. Xilinx, Inc., San Jose, CA, 2nd edition, 1994.

[66] Xilinx. *XACT Reference Guide*. Xilinx, Inc., San Jose, CA, April 1994.

[67] Xilinx. *XACT Xilinx Synopsys Interface FPGA User Guide*. Xilinx, Inc., San Jose, CA, December 1994.

[68] Xilinx. *Floorplanner User Guide*. Xilinx, Inc., San Jose, CA, February 1995. (preliminary version).

[69] D. Sillage, "How to implement DSP algorithms using the Xilinx Spartan-3E starter board",website: http://www.industrialcontroldesignline.com/howto/207001725;jsessionid =YDZGLZ44Z2B3AQSNDLRSKHSCJUNN2JVN?pgno=2, July 2009.

[70] Ray C.C. Cheung and K.P. Pun and Steve C.L. Yuen and K.H. Tsoi and Philip H.W. Leong An FPGA-based Re-configurable 24-bit 96kHz Sigma-Delta Audio *DAC.* FPT 2003, Tokyo, Japan.

[71] http://www.xilinx.com/products/boards/s3estarter/reference_designs.htm

[72] Xilinx, "Spartan-3E Starter Kit Board User Guide", Xilinx Ltd, p.67-71, 2012.

[73] Xilinx, "Spartan-3E Starter Kit Board User Guide", Xilinx Ltd, p.41-51, 2012.

[74] Xilinx, "Spartan-3E Starter Kit Board User Guide", Xilinx Ltd, p.17-19, 2012.

[75] Xilinx,Forums,http://forums.xilinx.com/xlnx/board/message?board.id=XLNXBRD&thread.id=463&view=by_date_ascending&page=1,December.2011

[76] Xilinx, "Spartan-3E Starter Kit Board User Guide", Xilinx Ltd, p.75-81,2006.

[77] K. Chapman, "Picoblaze: amplifier and A/D Converter Control for Spartan-3E Starter Kit", Xilinx Ltd. 2006.

[78] Xilinx, Forums, http://forums.xilinx.com/xlnx/board/message?board.id=XLNXBRD&thread.id=463&view=by_date_ascending&page=1, July 2012.

[79] http://www.elec.york.ac.uk/staff/ajg112.html , 27th May, 2012.

[80] Abhyankar, Y. , Design of a FPGA based data acquisition system for radio astronomy applications, The 16th International Conference on Microelectronics, 2004. ICM 2004 Proceedings.

[81] Sabat, S.L. Reliable High Speed Data Acquisition System Using FPGA, 2nd International Conference on Emerging Trends in Engineering and Technology (ICETET), 2009.

[82] Li Xingguang, A high-speed data acquisition system based on FPGA, International Conference on Test and Measurement, 2009. ICTM '09.

[83] Zhouligong.etc. The ARM Embedded system Software Development Example(1). BeiHang Unviesity Press.

[84] Xuzhijun CPLD/FPGA design. Publishing house of electronics industry.

[85] Daode Zhang etc.Hardware Design of Image Recognition System Based ARM and FPGA.

[86] Howard Johnson,Martin Graham. High-Speed Digital Design.Prentice HALL Press.

[87] Zhao Zecai, The SOC Design Technology Research Based on FPGA, National Defense Science and Technology University Press,Changsha,2006.

[88] Wang Junxiong, The Embedded Applied Research System Based on the FPGA and the NIOS, Southwest Jiaotong University,Xi'an,2006.

[89] Xu Haijun and Ye Weidong, "The Apply of FPGA in the High-performance Data Acquisition System", Computer Technology and Application, 2005, 25(1), pp. 40-43.

[90] Ma Mingjian and Zhou Changcheng, Data Acquisition and Processing Technology, Xi'an Jiaotong University Press, Xi'an,2001.

[91] http://www.whoyouvotefor.info/ethernet.html, 7th May, 2012.

[92] Wu Jihua and Wang Cheng,The Design of The Altera FPGA/CPLD, Posts & Telecom Press,Beijing,2005.

[93] Istiyanto, J.E. , A VHDL-based ADC on FPGA, International Conference on Instrumentation, Communications, Information Technology, and Biomedical Engineering (ICICI-BME), 2009.

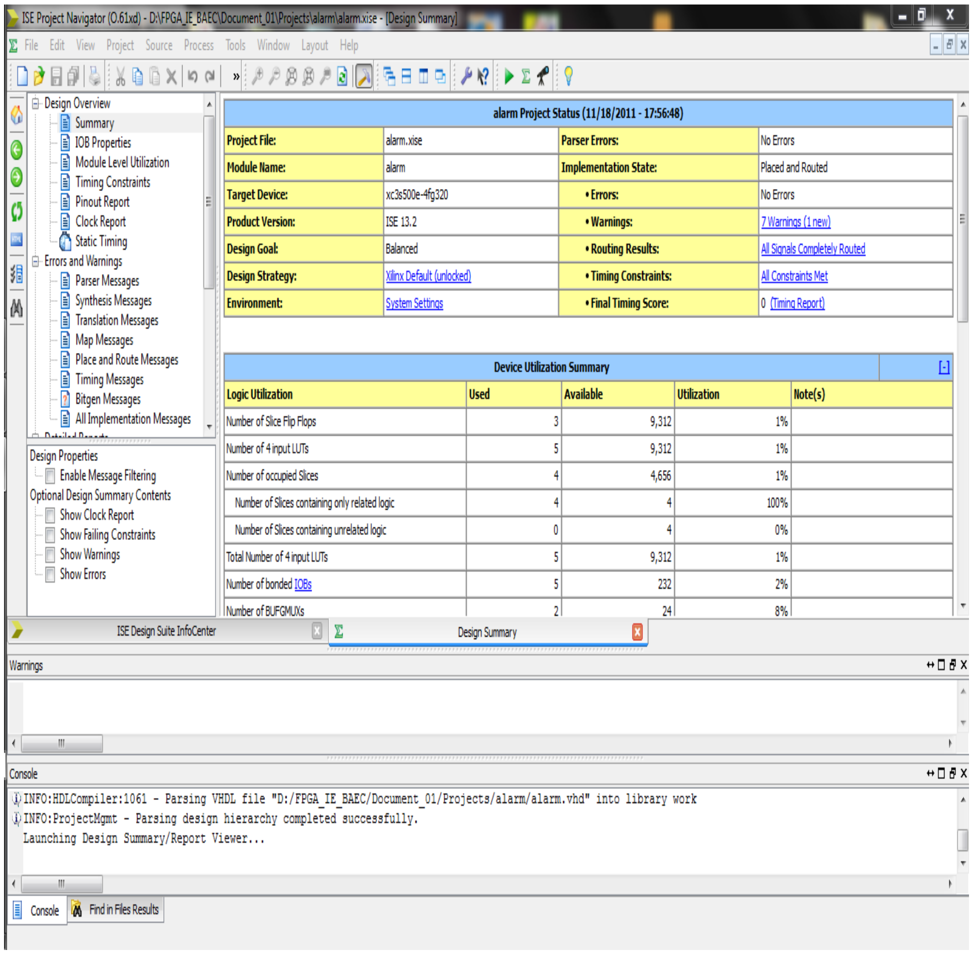
APPENDIX-A

XILINX-13.2 ISE TUTORUAL

XILINX-13.2 ISE TUTORIAL

INTRODUCTION THE XILINX-13.2 ISE

Click on the Icon, 64-bit Project Navigator, to start the ISE Project Navigator (0.61xd) . After that you will find a window like figure-1.

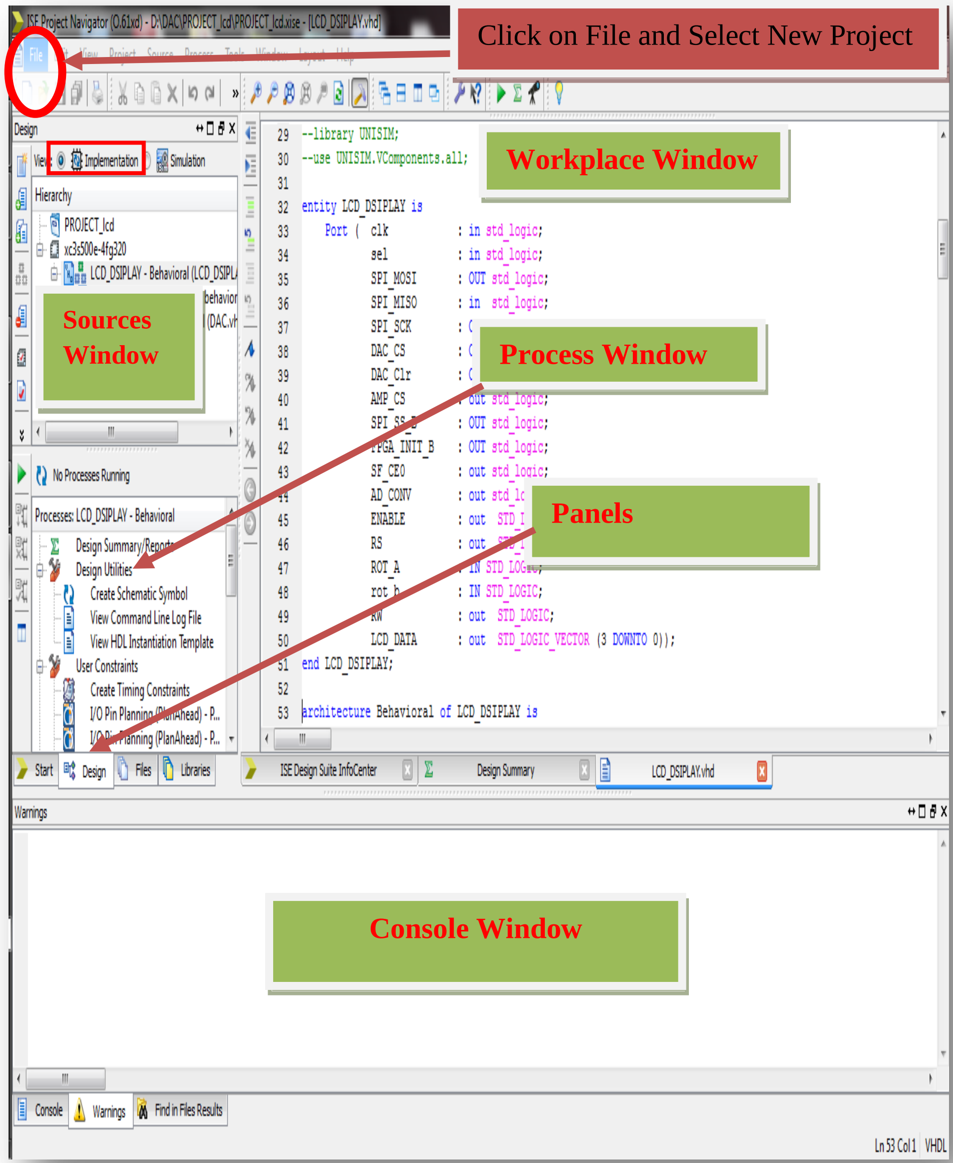


alarm Project Status (11/18/2011 - 17:56:48)			
Project File:	alarm.xise	**Parser Errors:**	No Errors
Module Name:	alarm	**Implementation State:**	Placed and Routed
Target Device:	xc3s500e-4fg320	**• Errors:**	No Errors
Product Version:	ISE 13.2	**• Warnings:**	7 Warnings (1 new)
Design Goal:	Balanced	**• Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	All Constraints Met
Environment:	System Settings	**• Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	3	9,312	1%	
Number of 4 input LUTs	5	9,312	1%	
Number of occupied Slices	4	4,656	1%	
Number of Slices containing only related logic	4	4	100%	
Number of Slices containing unrelated logic	0	4	0%	
Total Number of 4 input LUTs	5	9,312	1%	
Number of bonded IOBs	5	232	2%	
Number of BUFGMUXs	2	24	8%	

```
INFO:HDLCompiler:1061 - Parsing VHDL file "D:/FPGA_IE_BAEC/Document_01/Projects/alarm/alarm.vhd" into library work
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
Launching Design Summary/Report Viewer...
```

Figure-A1. ISE Project Navigator (0.61xd)


Click on File and Select New Project
Workplace Window
Sources Window
Process Window
Panels
Console Window
ISE Project Navigator (O.61xd) - D:\DAC\PROJECT_lcd\PROJECT_lcd.xise - [LCD_DSIPLAY.vhd]
File
Design
Implementation
Simulation
Hierarchy
PROJECT_lcd
xc3s500e-4fg320
LCD_DSIPLAY - Behavioral (LCD_DSIPL
No Processes Running
Processes: LCD_DSIPLAY - Behavioral
Design Summary/Reports
Design Utilities
Create Schematic Symbol
View Command Line Log File
View HDL Instantiation Template
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAhead) - P...
I/O Pin Planning (PlanAhead) - P...
Start
Design
Files
Libraries
ISE Design Suite InfoCenter
Design Summary
LCD_DSIPLAY.vhd
Warnings
Console
Warnings
Find in Files Results
Ln 53 Col 1
VHDL

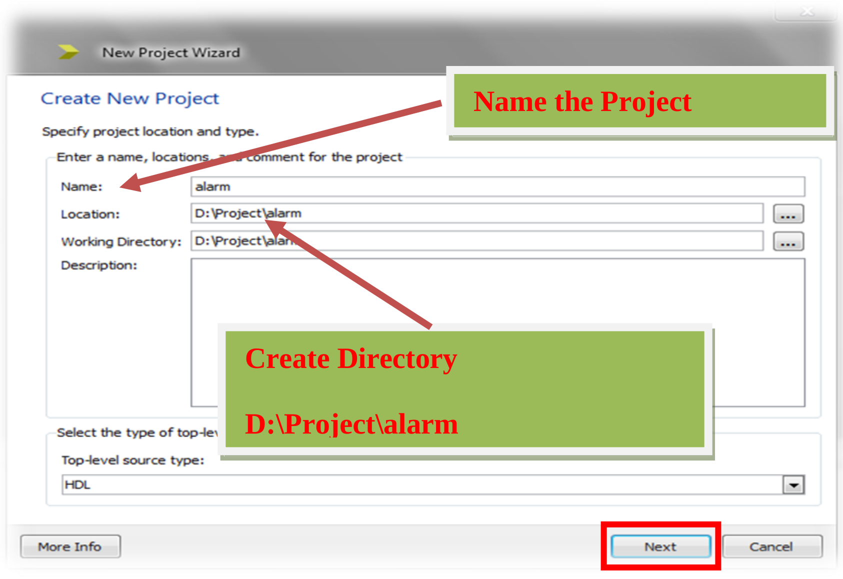
```
--library UNISIM;
--use UNISIM.VComponents.all;

entity LCD_DSIPLAY is
    Port (  clk          : in std_logic;
            sel          : in std_logic;
            SPI_MOSI     : OUT std_logic;
            SPI_MISO     : in  std_logic;
            SPI_SCK      : (
            DAC_CS       : (
            DAC_Clr      : (
            AMP_CS       : out std_logic;
            SPI_SS_B     : OUT std_logic;
            FPGA_INIT_B  : OUT std_logic;
            SF_CE0       : out std_logic;
            AD_CONV      : out std_lo
            ENABLE       : out  STD_L
            RS           : out  STD_L
            ROT_A        : IN STD_LOGIC;
            rot_b        : IN STD_LOGIC;
            RW           : out  STD_LOGIC;
            LCD_DATA     : out  STD_LOGIC_VECTOR (3 DOWNTO 0));
end LCD_DSIPLAY;

architecture Behavioral of LCD_DSIPLAY is
```

Figure-A2. Project Navigator Window

<u>Step 1. Creating New Project</u>

File→New Project Or,

Click on the Icon,



Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

Name: alarm

Location: D:\Project\alarm

Working Directory: D:\Project\alarm

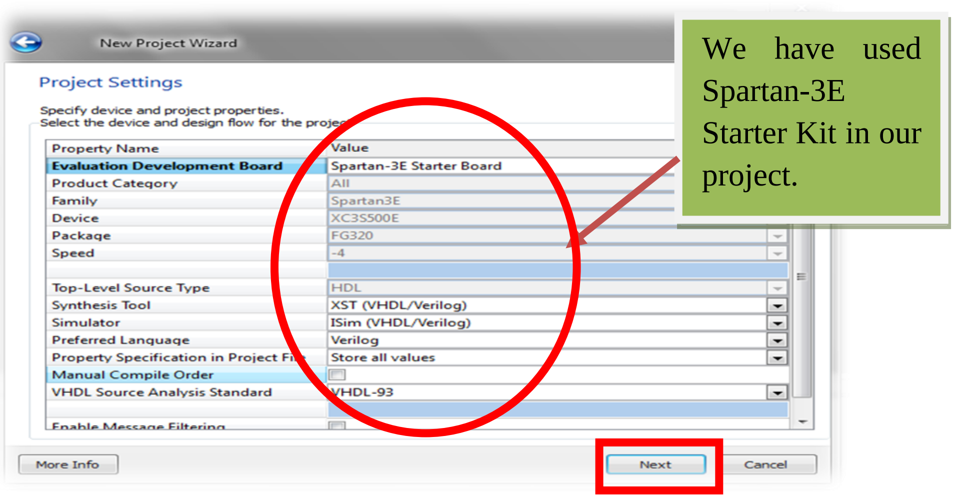
Description:

Select the type of top-le

Top-level source type:

HDL

Figure-A3. Creating New Project



Project Settings

Specify device and project properties.
Select the device and design flow for the projec

Property Name	Value
Evaluation Development Board	Spartan-3E Starter Board
Product Category	All
Family	Spartan3E
Device	XC3S500E
Package	FG320
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project Fi	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

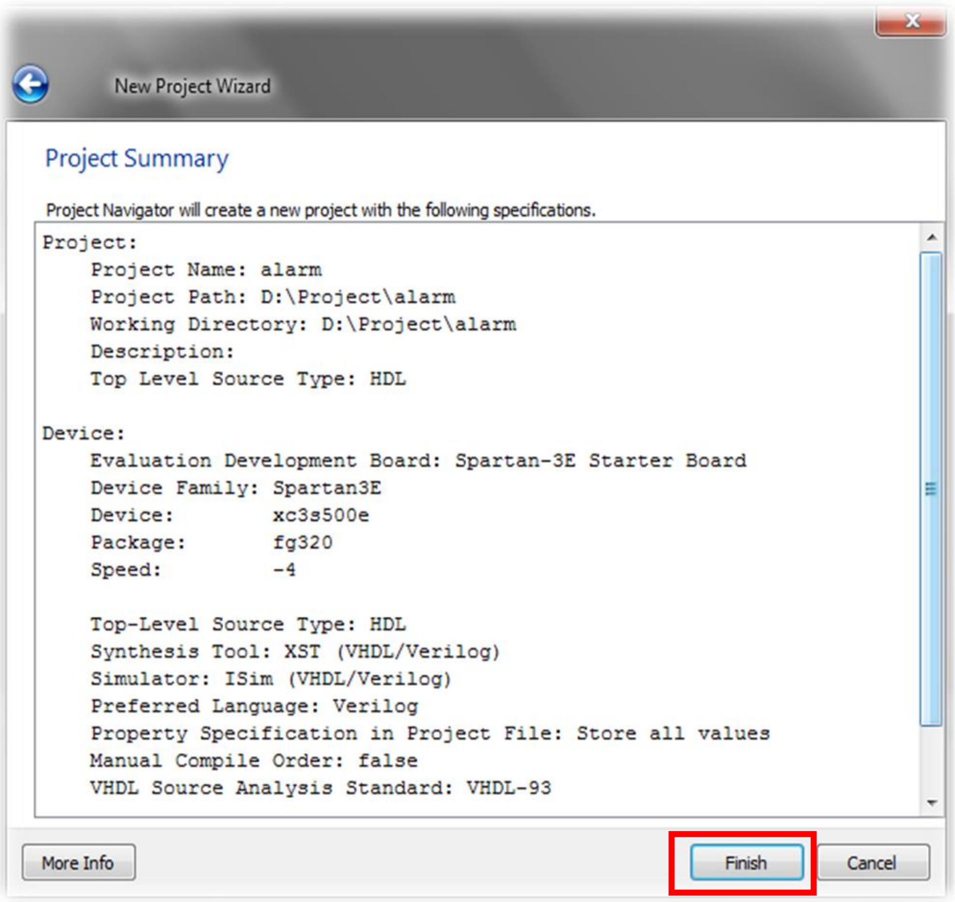
Figure-A4. Creating New Project (Cont.)

In the figure-4 the most important thing is that you have to select “Evaluation Development Board”-type as “Spartan-3E Starter Board”, as we have used this board while developing our system. Xilinx-13.2 ISE supports the following “Evaluation Development Board”.

- Kintex-7 KC705 Evaluation Platform.
- Spartan-3A DSP 1800A Starter Board.
- Spartan-3A DSP 3400A Development Board.
- Spartan-3A Starter Kit.
- Spartan-3AN Starter Kit.

- Spartan-3E 1600E Micro blaze Dev Board.
- Spartan-3E Starter Kit.
- Spartan-6 SP601 Evaluation Platform.
- Spartan-6 Sp605 Evaluation Platform.
- Virtex-4 ML403 Evaluation Platform.
- Virtex-4 ML405 Evaluation Platform.
- Virtex-5 ML505 Evaluation Platform.
- Virtex-5 ML506 Evaluation Platform.
- Virtex-5 ML507 Evaluation Platform.
- Virtex-5 ML510 Evaluation Platform.
- Virtex-6 ML605 Evaluation Platform.

The second important thing is that we have considered here the "**VHDL Source Analysis Standard"-VHDL-93**. Xilinx-13.2 ISE also supports VHDL-200X.



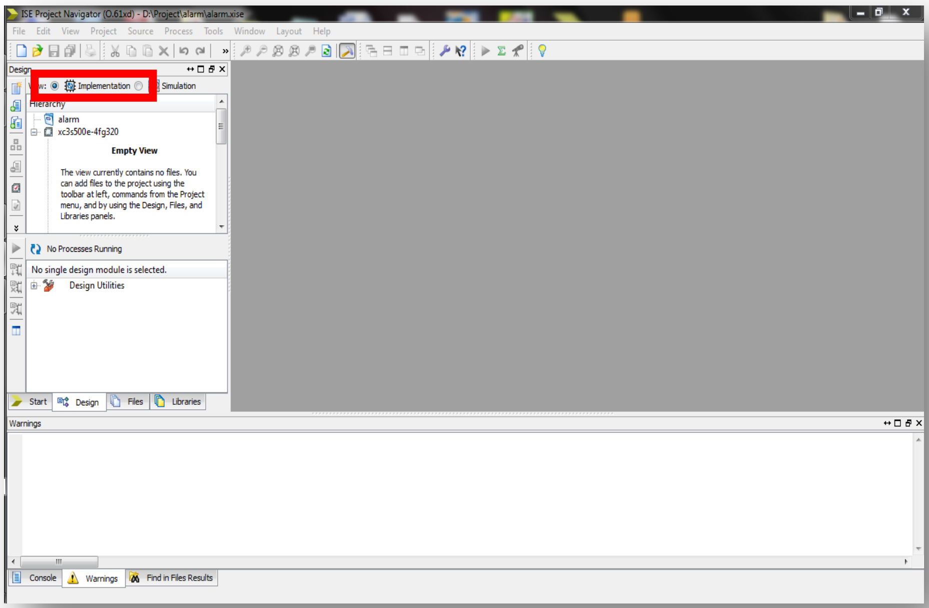
Project Navigator will create a new project with the following specifications.

```
Project:
    Project Name: alarm
    Project Path: D:\Project\alarm
    Working Directory: D:\Project\alarm
    Description:
    Top Level Source Type: HDL

Device:
    Evaluation Development Board: Spartan-3E Starter Board
    Device Family: Spartan3E
    Device:        xc3s500e
    Package:       fg320
    Speed:         -4

    Top-Level Source Type: HDL
    Synthesis Tool: XST (VHDL/Verilog)
    Simulator: ISim (VHDL/Verilog)
    Preferred Language: Verilog
    Property Specification in Project File: Store all values
    Manual Compile Order: false
    VHDL Source Analysis Standard: VHDL-93
```

Figure-A5. Creating New Project (Cont.)

In the last step simply click finish. Then you will get the following window.

☐



Empty View

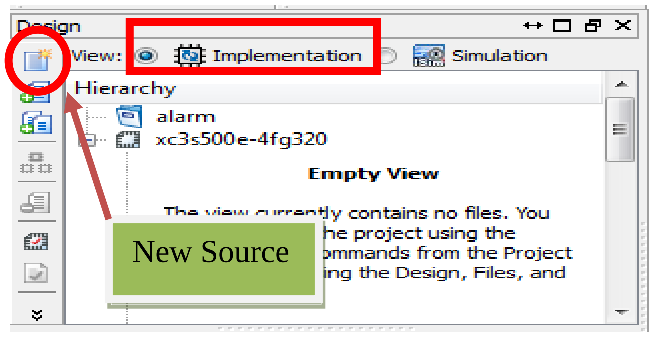
The view currently contains no files. You can add files to the project using the toolbar at left, commands from the Project menu, and by using the Design, Files, and Libraries panels.

No single design module is selected.

Figure-A6. Creating New Project (Completed).

Step-2: Creating New Source

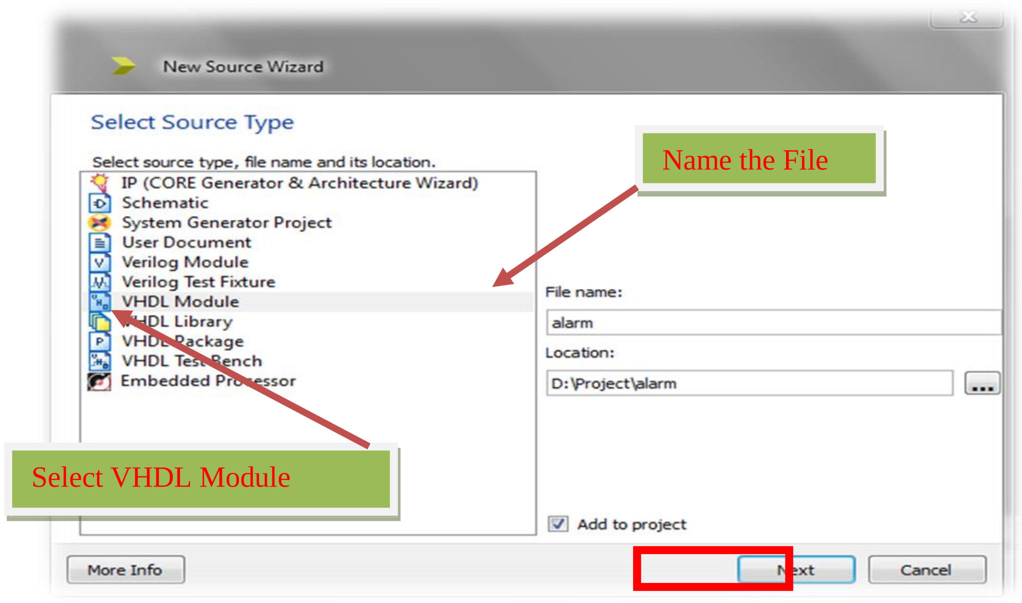
From the design plane, click on the Icon, , which denotes the "New Source". Choose VHDL Module as "Source Type", Type "alarm" in file name and click next.



Empty View

The view currently contains no files. You
he project using the
ommands from the Project
ing the Design, Files, and

Figure-A7. Creating New Source.


New Source Wizard

Select Source Type

Select source type, file name and its location.

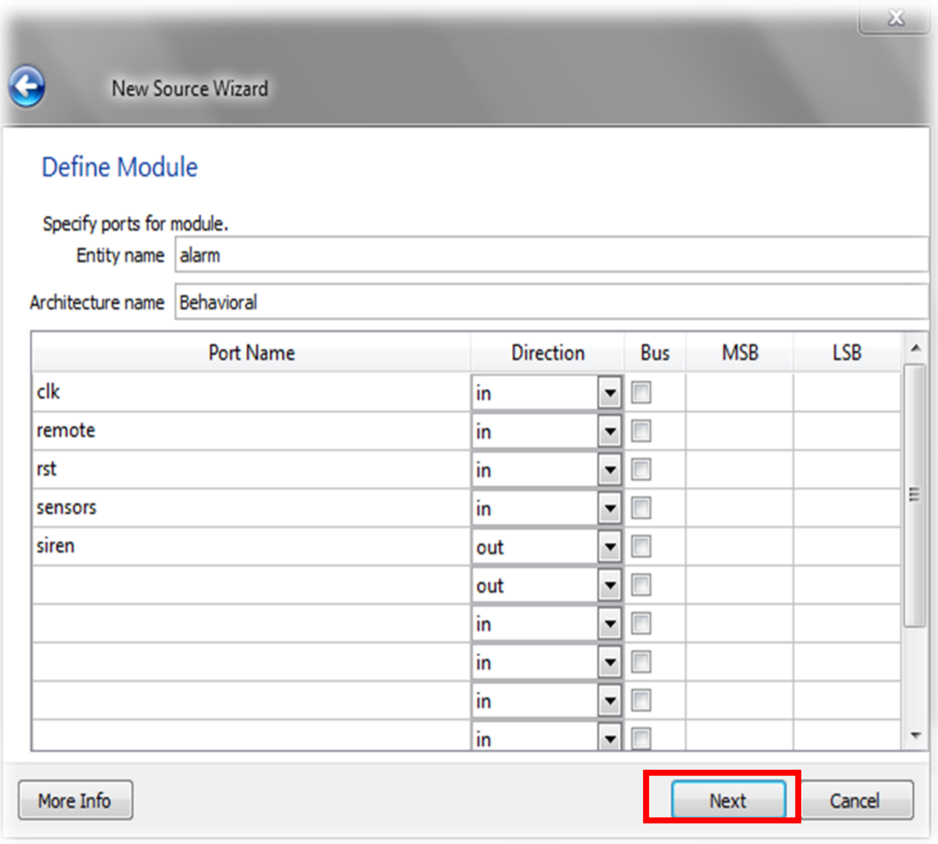
- IP (CORE Generator & Architecture Wizard)
- Schematic
- System Generator Project
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: alarm

Location: D:\Project\alarm

☑ Add to project

Figure-A8. New Source Wizard.


New Source Wizard

Define Module

Specify ports for module.

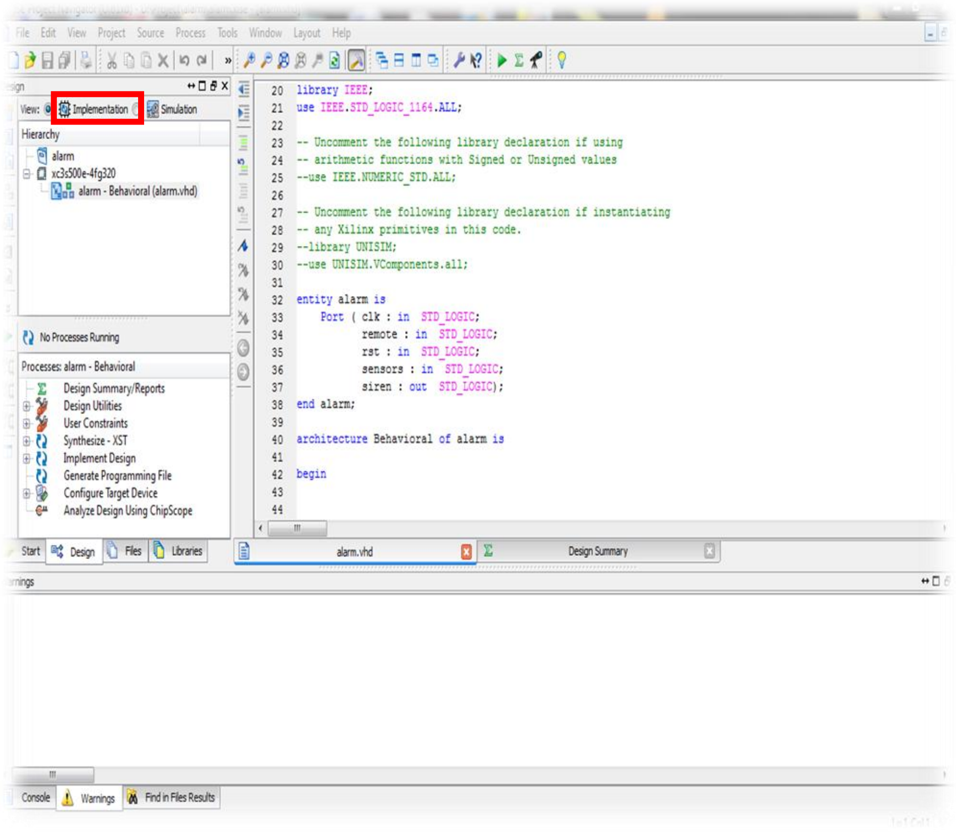
Entity name: alarm

Architecture name: Behavioral

Port Name	Direction	Bus	MSB	LSB
clk	in	☐		
remote	in	☐		
rst	in	☐		
sensors	in	☐		
siren	out	☐		
	out	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		

Figure-A9. New Source Wizard (Continue).

Define the port name and direction as shown in the figure-9.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity alarm is
    Port ( clk : in  STD_LOGIC;
           remote : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           sensors : in  STD_LOGIC;
           siren : out  STD_LOGIC);
end alarm;

architecture Behavioral of alarm is

begin

```

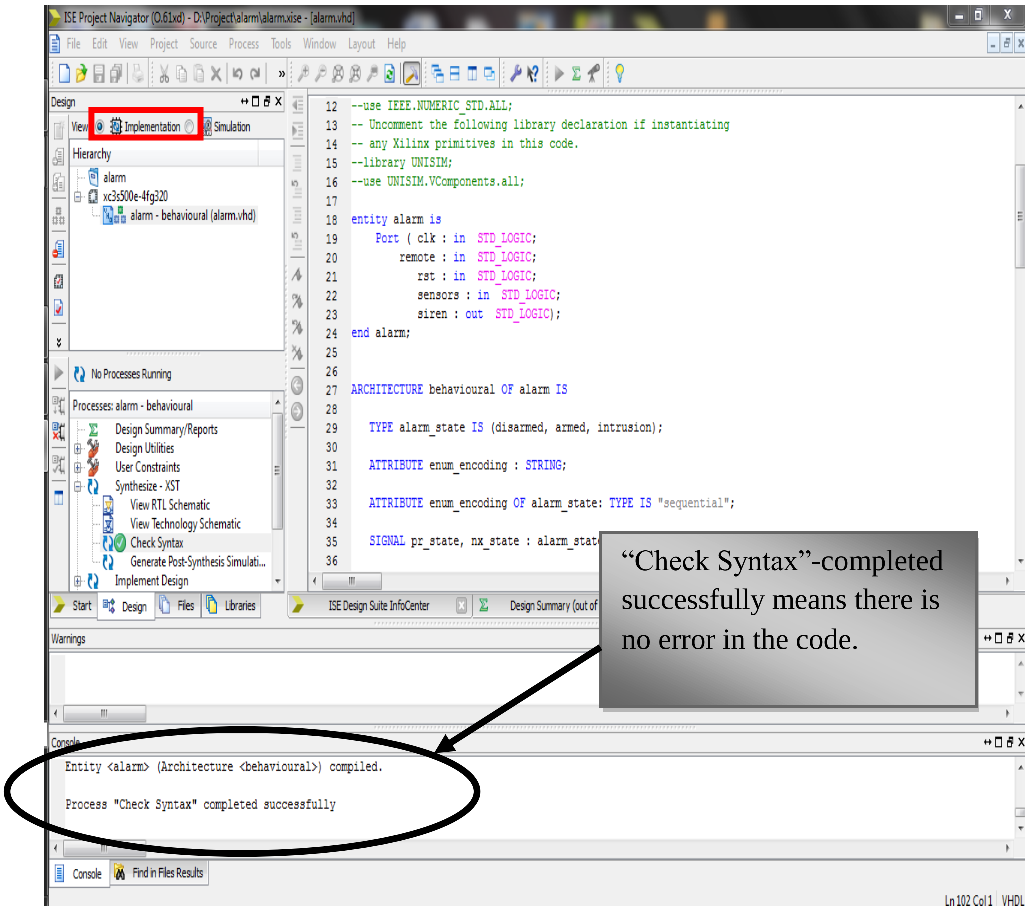
Figure-A10. Project Navigator Window for the Project "alarm".

After that copy the following code in the "**workplace window**". The main objective of this tutorial is to get acquainted the readers with Xilinx-13.2 ISE's use, not to give idea how to write VHDL code. The most important thing about VHDL language is that it is not a case sensitive language like "C". So we have used here the capital letters while programming. This is "alarm.vhd" file.

```
--------------------------------------------------------------------------------
--ENGINEER: HUSNAIN-AL-BUSTAM
-- REVISION 0.01 - FILE CREATED
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-- UNCOMMENT THE FOLLOWING LIBRARY DECLARATION IF USING
-- ARITHMETIC FUNCTIONS WITH SIGNED OR UNSIGNED VALUES
--USE IEEE.NUMERIC_STD.ALL;
-- UNCOMMENT THE FOLLOWING LIBRARY DECLARATION IF INSTANTIATING
-- ANY XILINX PRIMITIVES IN THIS CODE.
--LIBRARY UNISIM;
--USE UNISIM.VCOMPONENTS.ALL;
ENTITY ALARM IS
  PORT ( CLK : IN  STD_LOGIC;
           REMOTE : IN  STD_LOGIC;
      RST : IN  STD_LOGIC;
      SENSORS : IN  STD_LOGIC;
      SIREN : OUT  STD_LOGIC);
END ALARM;
ARCHITECTURE BEHAVIOURAL OF ALARM IS
        TYPE ALARM_STATE IS (DISARMED, ARMED, INTRUSION);
        ATTRIBUTE ENUM_ENCODING : STRING;
        ATTRIBUTE ENUM_ENCODING OF ALARM_STATE: TYPE IS "SEQUENTIAL";
        SIGNAL PR_STATE, NX_STATE : ALARM_STATE;
        SIGNAL FLAG: STD_LOGIC;
        CONSTANT RESET: STD_LOGIC :='0';
        BEGIN
----------------------------- FLAG: ------------------------------------------------
                PROCESS (REMOTE, RST)
                BEGIN
                        IF (RST='1') THEN
                                FLAG <= '0';
                        ELSIF (REMOTE'EVENT AND REMOTE='0') THEN
                        --ELSIF (REMOTE='0') THEN
                                FLAG <= NOT FLAG;
                        END IF;
                END PROCESS;
----- --------------------------LOWER SECTION: ----------------------------------------
                PROCESS (CLK, RST)
                BEGIN
                        IF (RST='1') THEN
                                PR_STATE <= DISARMED;
                        ELSIF (CLK'EVENT AND CLK='1') THEN
                                PR_STATE <= NX_STATE;
                        END IF;
                END PROCESS;
```

```
----- UPPER SECTION: ---------------------------------------
        PROCESS (PR_STATE, FLAG, REMOTE, SENSORS)
            BEGIN
                CASE PR_STATE IS
                WHEN DISARMED =>
                    SIREN <= '0';
                    IF (REMOTE='1' AND FLAG='0') THEN
                        NX_STATE <= ARMED;
                    ELSE
                        NX_STATE <= DISARMED;
                    END IF;
                WHEN ARMED =>
                    SIREN <= '0';
                    IF (SENSORS='1') THEN
                        NX_STATE <= INTRUSION;
                    ELSIF (REMOTE='1' AND FLAG='1') THEN
                        NX_STATE <= DISARMED;
                    ELSE
                        NX_STATE <= ARMED;
                    END IF;
                WHEN INTRUSION =>
                    SIREN <= '1';
                    IF (REMOTE='1' AND FLAG='1') THEN
                        NX_STATE <= DISARMED;
                    ELSE
                    NX_STATE <= INTRUSION;
                        END IF;
                    END CASE;
        END PROCESS;
END BEHAVIOURAL;
-------------------------------------------------------------------------------------------------------
```

Expand the “Synthesize-XST”. Click “Check Syntax”. Correct the error. Then check syntax again.


ISE Project Navigator (O.61xd) - D:\Project\alarm\alarm.xise - [alarm.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
alarm
xc3s500e-4fg320
alarm - behavioural (alarm.vhd)
No Processes Running
Processes: alarm - behavioural
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simulati...
Implement Design
Start Design Files Libraries
ISE Design Suite InfoCenter
Design Summary (out of
Warnings
Console
Find in Files Results
Ln 102 Col 1 VHDL
“Check Syntax”-completed successfully means there is no error in the code.

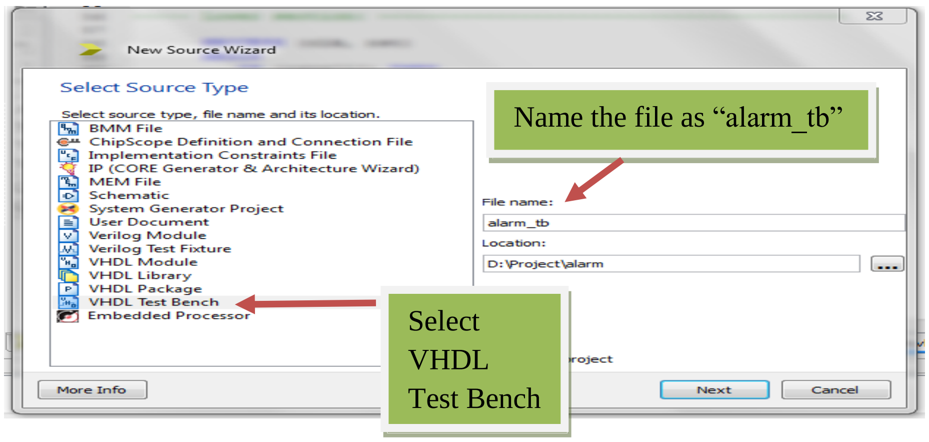
```
--use IEEE.NUMERIC_STD.ALL;
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity alarm is
    Port ( clk : in  STD_LOGIC;
           remote : in  STD_LOGIC;
              rst : in  STD_LOGIC;
              sensors : in  STD_LOGIC;
              siren : out  STD_LOGIC);
end alarm;


ARCHITECTURE behavioural OF alarm IS

   TYPE alarm_state IS (disarmed, armed, intrusion);

   ATTRIBUTE enum_encoding : STRING;

   ATTRIBUTE enum_encoding OF alarm_state: TYPE IS "sequential";

   SIGNAL pr_state, nx_state : alarm_stat
```

```
Entity <alarm> (Architecture <behavioural>) compiled.

Process "Check Syntax" completed successfully
```

Figure-A11. Checking Syntax (Successful).

Step-3. Creating Test Bench File

Click of the Icon, (From the Design Plane-New Source). Choose VHDL Test Bench. File name “alarm_tb”. And click “Next” to create file.



File name: alarm_tb

Location: D:\Project\alarm

Figure-A12. New Source Wizard (Test Bench)

From the "Design Pane" select the "Simulation" and you will get a hierarchy like figure-13. Click on the "alarm_tb behavior " as shown in the figure-13.

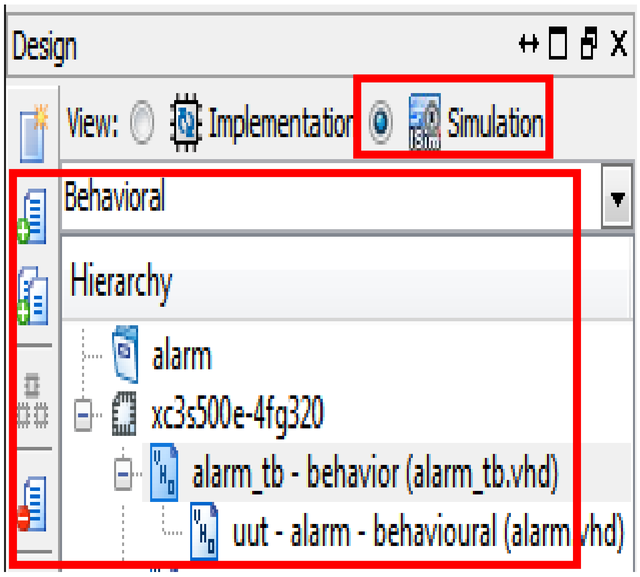

Design
View: Implementation Simulation
Behavioral
Hierarchy
alarm
xc3s500e-4fg320
alarm_tb - behavior (alarm_tb.vhd)
uut - alarm - behavioural (alarm.vhd)

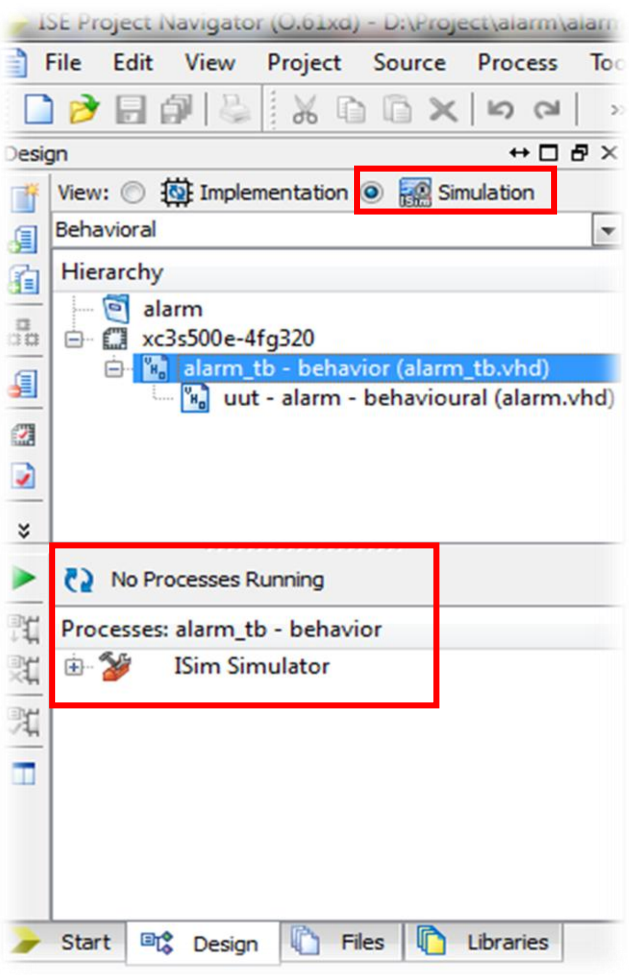
Figure-A13. New Source Wizard (Test Bench).

After that copy the following code in the "workplace window". The following code is for "alarm_tb" file.

```
--------------------------------------------------------------------------------
-- COMPANY: ISLAMIC UNIVERSITY OF TECHNOLOGY, OIC
-- ENGINEER:HUSNAIN-AL-BUSTAM
-- CREATE DATE:   18:08:42 11/18/2011
-- DESIGN NAME:   ALARM
-- PROJECT NAME:  ALARM
-- TARGET DEVICE:  SPARTAN-3E STARTER KIT.
-- VHDL TEST BENCH CREATED BY ISE FOR MODULE: ALARM
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-- UNCOMMENT THE FOLLOWING LIBRARY DECLARATION IF USING
-- ARITHMETIC FUNCTIONS WITH SIGNED OR UNSIGNED VALUES
--USE IEEE.NUMERIC_STD.ALL;
ENTITY ALARM_TB IS
END ALARM_TB;
ARCHITECTURE BEHAVIOR OF ALARM_TB IS
-- COMPONENT DECLARATION FOR THE UNIT UNDER TEST (UUT)
```

```
   COMPONENT ALARM
   PORT(
        CLK : STD_LOGIC;
        REMOTE : IN  STD_LOGIC;
        RST : IN  STD_LOGIC;
        SENSORS : IN  STD_LOGIC;
        SIREN : OUT  STD_LOGIC;
        CLKOUT : OUT STD_LOGIC
       );
   END COMPONENT;
  --INPUTS
  SIGNAL CLK : STD_LOGIC := '0';
  SIGNAL REMOTE : STD_LOGIC := '0';
  SIGNAL RST : STD_LOGIC := '0';
  SIGNAL SENSORS : STD_LOGIC := '0';
  SIGNAL SIREN : STD_LOGIC;
  SIGNAL CLKOUT : STD_LOGIC;
BEGIN
  CLK <= NOT CLK AFTER 20 NS;
-- INSTANTIATE THE UNIT UNDER TEST (UUT)
  UUT: ALARM PORT MAP (
         --CLK => CLK,
         CLKIN => CLK,
         REMOTE => REMOTE,
         RST => RST,
         SENSORS => SENSORS,
         SIREN => SIREN,
         CLKOUT => CLKOUT
       );
  -- CLOCK PROCESS DEFINITIONS
  -- STIMULUS PROCESS
  STIM_PROC: PROCESS
  BEGIN
  WAIT FOR 600 NS;
                        RST <= '1';
                        WAIT FOR 50 NS ;
                        RST <='0';
```

```
                    WAIT FOR 50 NS;
                    REMOTE <='1';
                    WAIT FOR 50 NS;
                    SENSORS<='1';
                    WAIT FOR 100 NS;
                    --REMOTE <='0';
                    --WAIT FOR 50 NS;
                    REMOTE <='1';
                    WAIT FOR 50 NS;
          END PROCESS;
END;
---------------------------------------------------------------------------------------------------------
```



alarm
xc3s500e-4fg320
alarm_tb - behavior (alarm_tb.vhd)
uut - alarm - behavioural (alarm.vhd)

No Processes Running

Processes: alarm_tb - behavior

ISim Simulator

Figure-A14. Design (Simulation)

On the view pane select “Simulation”. In the hiearchy window select the file “alarm_tb”. In the process window first expand the “ISim Simulator” and run “Behavioural Check Syntax”. Click “Simulation Behavioral Model” to start “ISim”. Browse the window.

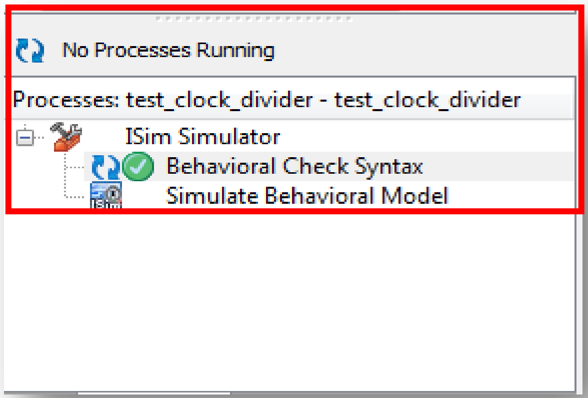

No Processes Running
Processes: test_clock_divider - test_clock_divider
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model

Figure-A15. Simulation.

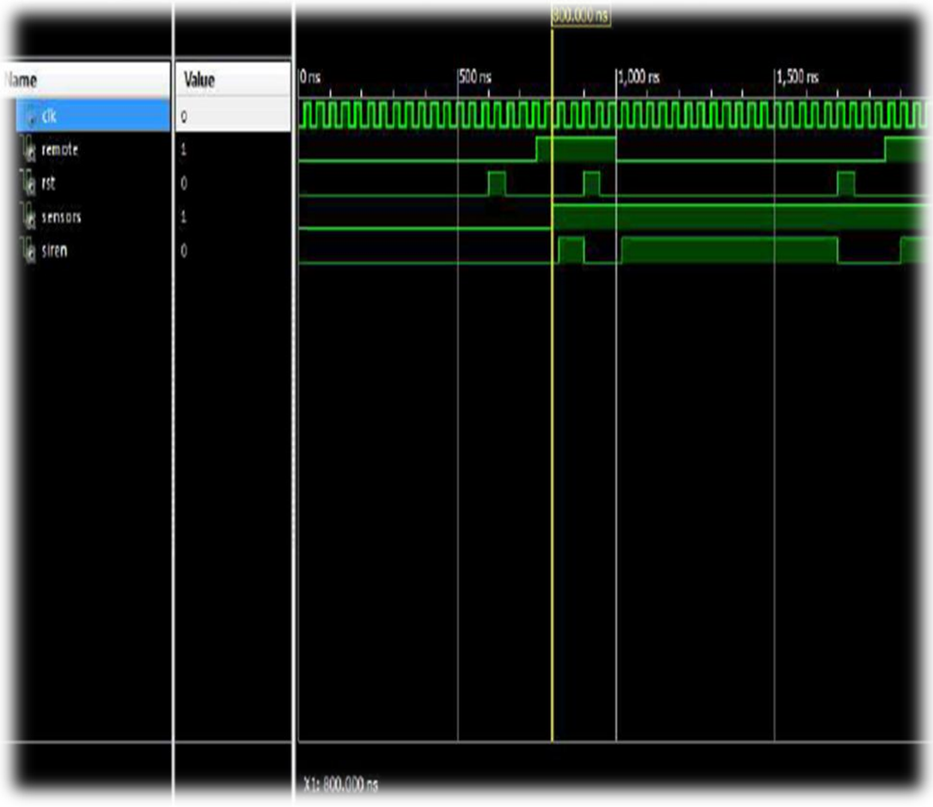


Figure-A16. Isim Window.

STEP-1:

On the view pane now select again the “implementation”. Expand “User Constraints”. Double click on the “Create Timing Constraints”. Then a window will appear before you where you will find “Unconstrained Clocks”. This project has two “Unconstrained Clocks”- “clk” and “remote”. Just double click on them. Each time one windows will appear when you double click on a clock. For the simplicity we haven’t changed any parameter. Click “Ok” to finish. The “Unconstrained Clocks” will be constrained. Save and Exit. The whole process is depicted in the following figures.

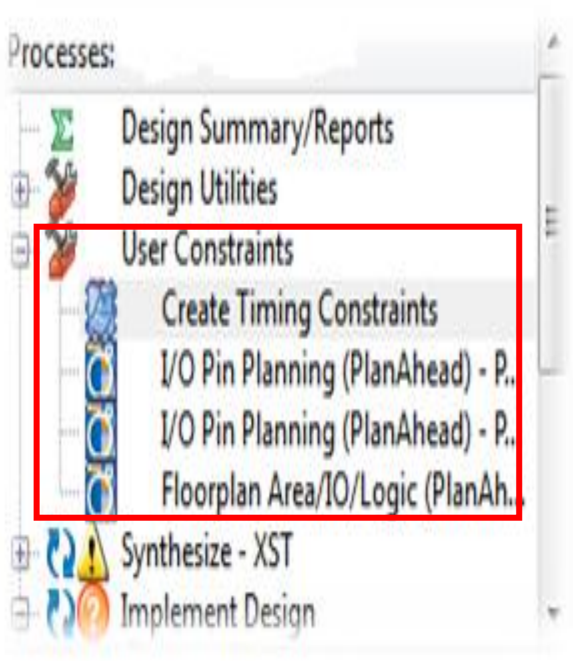
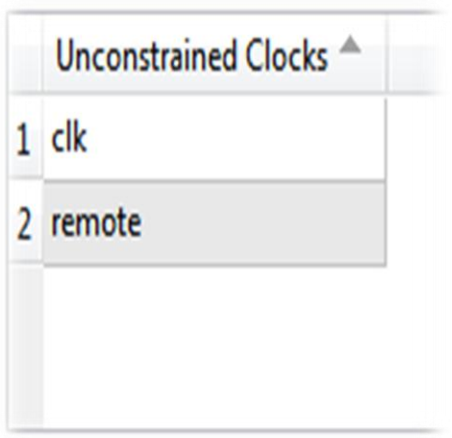
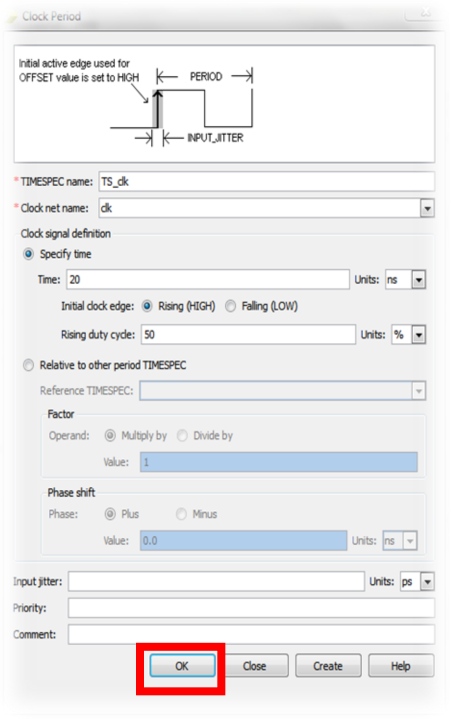

Processes:
Design Summary/Reports
Design Utilities
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAhead) - P..
I/O Pin Planning (PlanAhead) - P..
Floorplan Area/IO/Logic (PlanAh..
Synthesize - XST
Implement Design

Figure-A17. Expanded User Constraints.



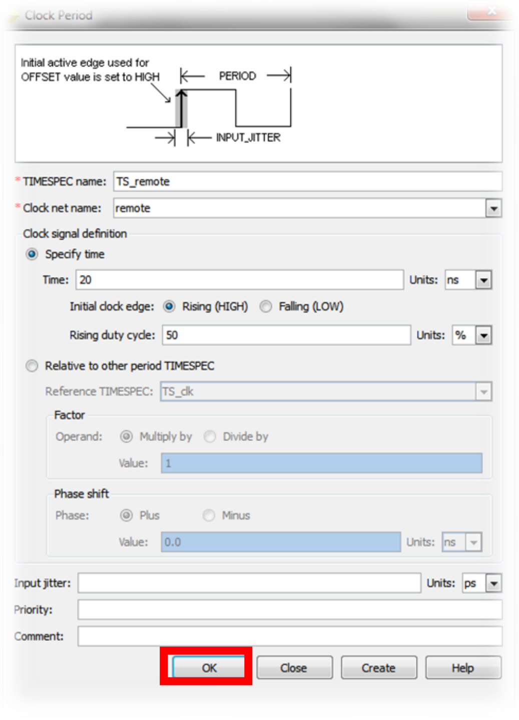
	Unconstrained Clocks
1	clk
2	remote

Figure-A18. Unconstrained Clocks



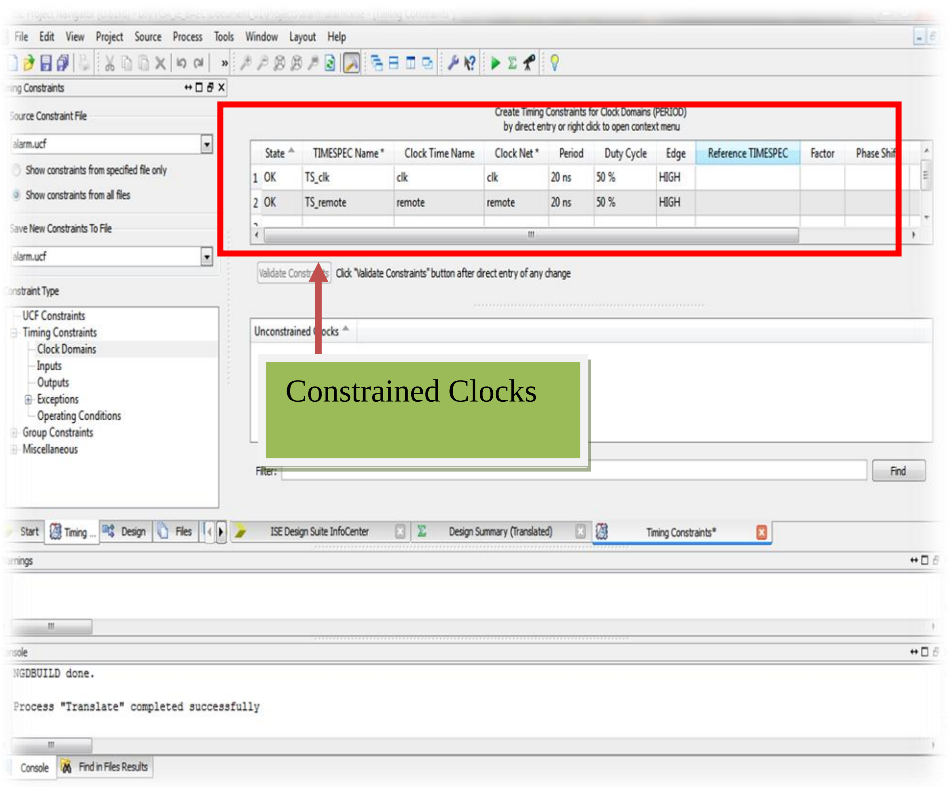
TIMESPEC name: TS_clk
Clock net name: clk
Clock signal definition
[✓ Specify time]
Time: 20 Units: ns
Initial clock edge: [✓ Rising (HIGH)] Falling (LOW)
Rising duty cycle: 50 Units: %
Relative to other period TIMESPEC
Reference TIMESPEC:
Factor
Operand: [✓ Multiply by] Divide by
Value: 1
Phase shift
Phase: [✓ Plus] Minus
Value: 0.0 Units: ns
Input jitter: Units: ps
Priority:
Comment:

Figure-A19. Profile Generated for Clock "CLK"



TIMESPEC name: TS_remote
Clock net name: remote
Clock signal definition
[✓ Specify time]
Time: 20 Units: ns
Initial clock edge: [✓ Rising (HIGH)] Falling (LOW)
Rising duty cycle: 50 Units: %
Relative to other period TIMESPEC
Reference TIMESPEC: TS_clk
Factor
Operand: [✓ Multiply by] Divide by
Value: 1
Phase shift
Phase: [✓ Plus] Minus
Value: 0.0 Units: ns
Input jitter: Units: ps
Priority:
Comment:

Figure-A20. Profile Generated for Clock "REMOTE"


File Edit View Project Source Process Tools Window Layout Help
Source Constraint File
alarm.ucf
Show constraints from specified file only
Show constraints from all files
Save New Constraints To File
alarm.ucf
Constraint Type
UCF Constraints
Timing Constraints
Clock Domains
Inputs
Outputs
Exceptions
Operating Conditions
Group Constraints
Miscellaneous
Create Timing Constraints for Clock Domains (PERIOD)
by direct entry or right click to open context menu
Validate Constraints
Click "Validate Constraints" button after direct entry of any change
Unconstrained Clocks
Constrained Clocks
Filter:
Find
Start
Timing
Design
Files
ISE Design Suite InfoCenter
Design Summary (Translated)
Timing Constraints*
Warnings
Console
Find in Files Results

	State	TIMESPEC Name *	Clock Time Name	Clock Net *	Period	Duty Cycle	Edge	Reference TIMESPEC	Factor	Phase Shif
1	OK	TS_clk	clk	clk	20 ns	50 %	HIGH			
2	OK	TS_remote	remote	remote	20 ns	50 %	HIGH			

```
NGDBUILD done.

Process "Translate" completed successfully
```

Figure-A21. Constrained Clocks

<u>STEP-2:</u>

From the process window select ("User Constraints") select "I/O Planning (Plan Ahead) Post Synthesis" (Figure-14). There you will find a window like "Figure-19". In the "I/O Port" window expand the "Scalar Ports" and assign the values as follows,

- Assign net "clk" to 50 MHz clock.
- Assign "remote" net to the button (V4).
- Assign "rst" net to the button (K17).
- Assign "sensors" net to the slide button (N17).
- Assign "siren" to Led (F9).

Consult "Spartan-3E Starter Kit" for the remaining constraints.

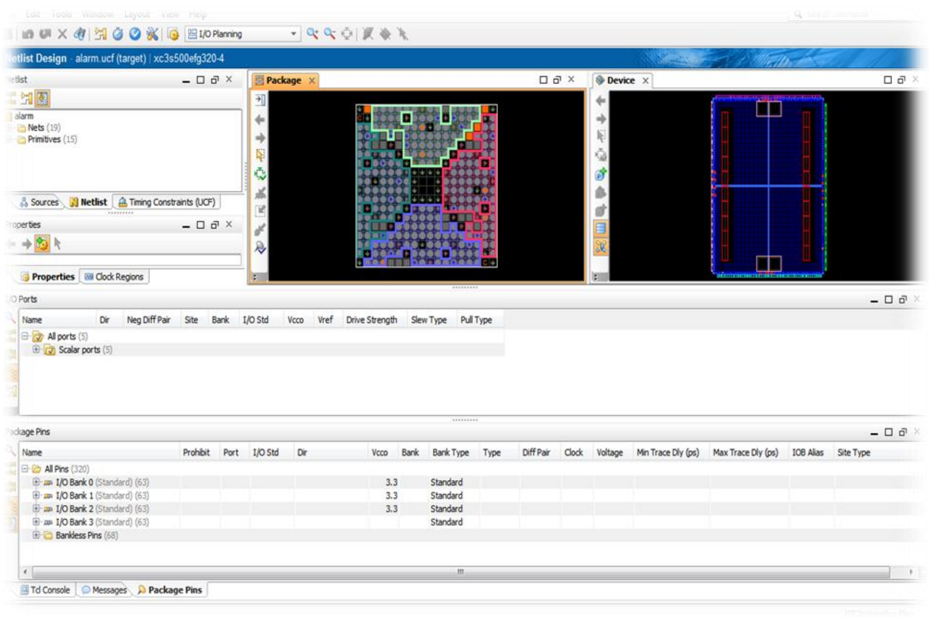
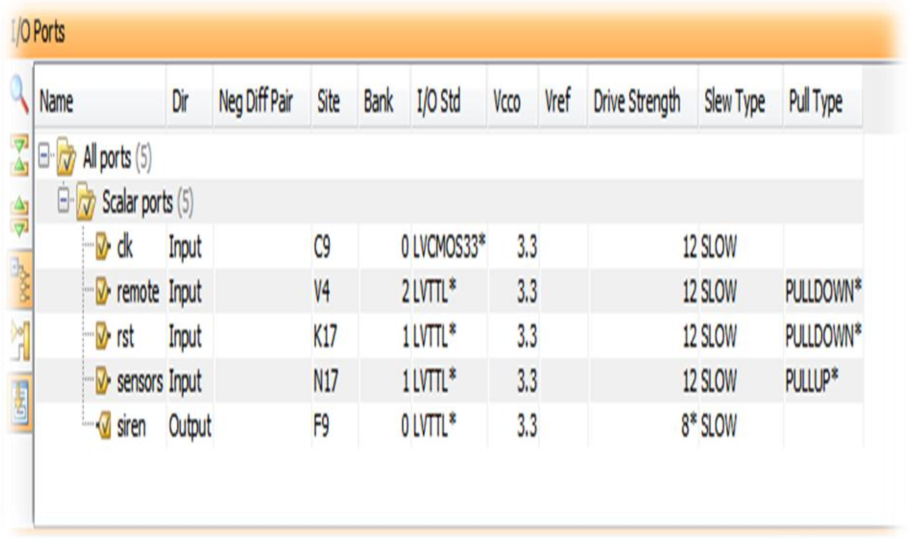


Figure-A22. Profile Generated After "I/O Planning (Plan Ahead) Post Synthesis"



I/O Ports

Name	Dir	Neg Diff Pair	Site	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type
All ports (5)										
Scalar ports (5)										
clk	Input		C9	0	LVCMOS33*	3.3		12	SLOW	
remote	Input		V4	2	LVTTL*	3.3		12	SLOW	PULLDOWN*
rst	Input		K17	1	LVTTL*	3.3		12	SLOW	PULLDOWN*
sensors	Input		N17	1	LVTTL*	3.3		12	SLOW	PULLUP*
siren	Output		F9	0	LVTTL*	3.3		8*	SLOW	

Figure-A23. Planning I/O Ports constraints.

<u>STEP-3: UPLOADING THE CODE INTO THE SPARTAN-3E STARTER KIT</u>

From the process window run the "Synthesize-XST". Then double click to run the "Implement Design". Then run the "Generate Programming File". Expand the "Configure Target Device". Click on "Manage Configuration Project (iMPACT)". A window namely "ISE iMPACT Window" will appear before you, where you have to do a few steps to burn the generated bit file into the target device (FPGA). Then double click on the "Boundary

Scan”. In the “Boundary Scan” window right-click and select “Initialize Chain”. Click “yes”. Select the “alarm.bit” file. Click “Ok” and “Bypass” twice. Then “Ok”.

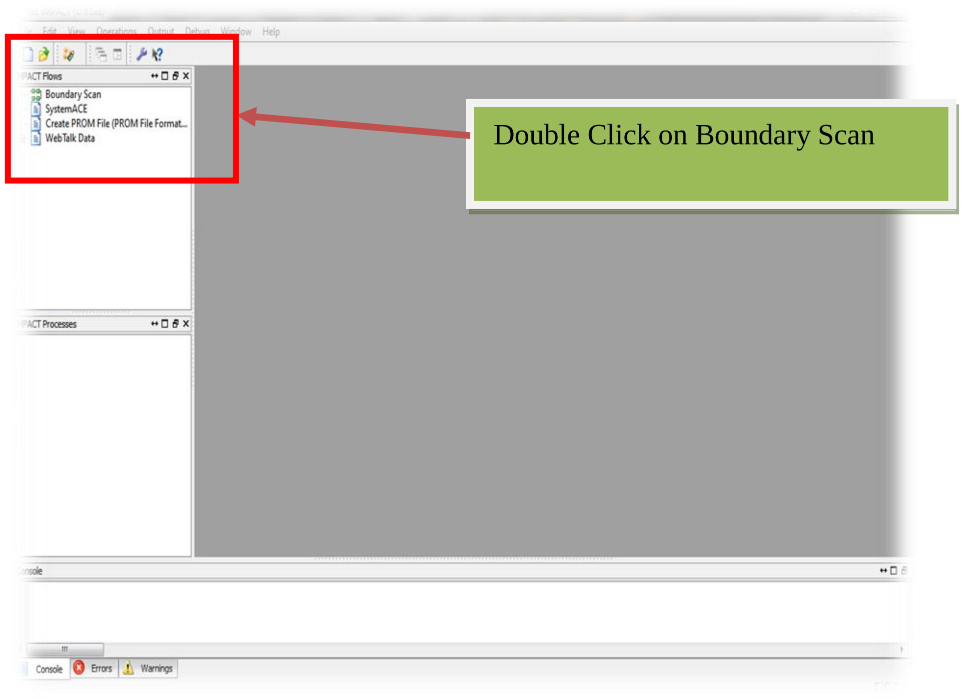


Figure-A24. ISE iMPACT Window

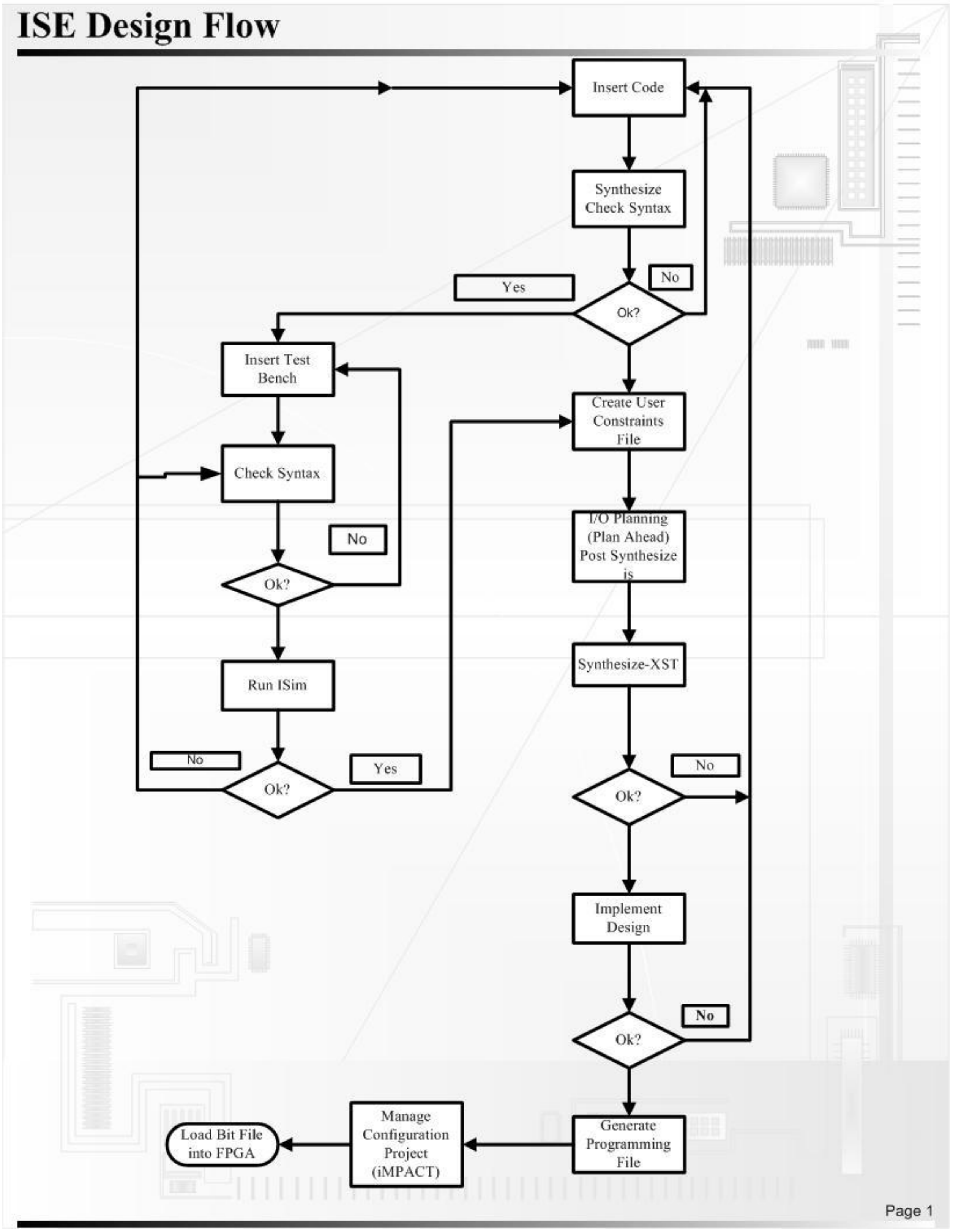


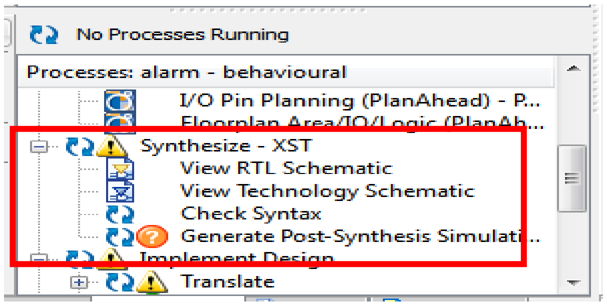
Figure-A25. ISE Design Flow

RTL SCHEMATIC

To see the “RTL Schematic” and “Technology Schematic”, go to the “Process Plane”. And do the following.

Synthesize-SXT→ View RTL Schematic.

Synthesize-SXT→View Technology Schematic.


No Processes Running

Processes: alarm - behavioural

I/O Pin Planning (PlanAhead) - P...

Floorplan Area/IO/Logic (PlanAh...

Synthesize - XST

View RTL Schematic

View Technology Schematic

Check Syntax

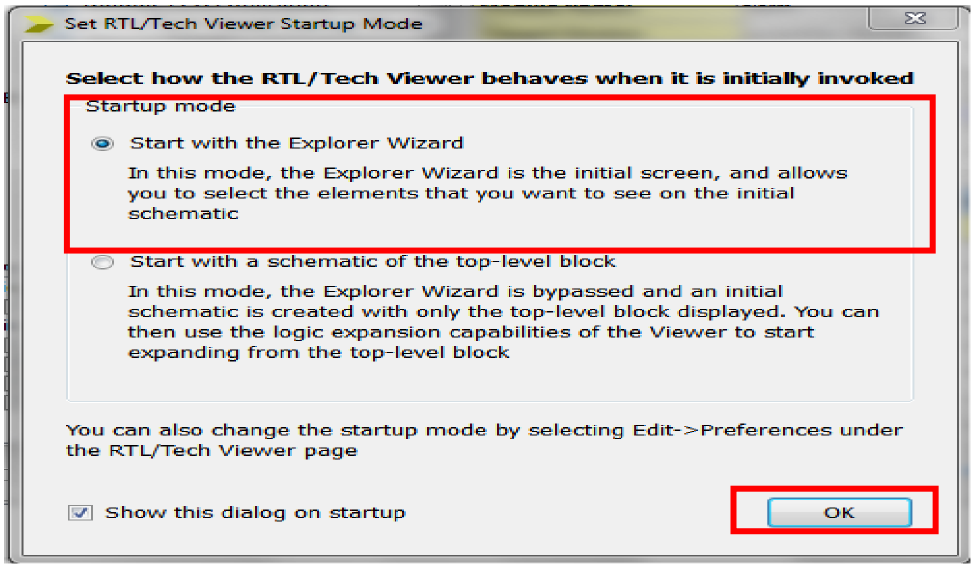
Generate Post-Synthesis Simulati...

Implement Design

Translate

Figure-A26. Process Window.

Suppose you wish to see the “RTL-Schematic” first. Just double click on the “View RTL Schematic”. Then a window will appear before you asking for the “Startup Mode”. Select the “Start with Explorer Wizard”. Click “Ok”.


Set RTL/Tech Viewer Startup Mode

Select how the RTL/Tech Viewer behaves when it is initially invoked

Startup mode

[✓ Start with the Explorer Wizard]

In this mode, the Explorer Wizard is the initial screen, and allows you to select the elements that you want to see on the initial schematic

Start with a schematic of the top-level block

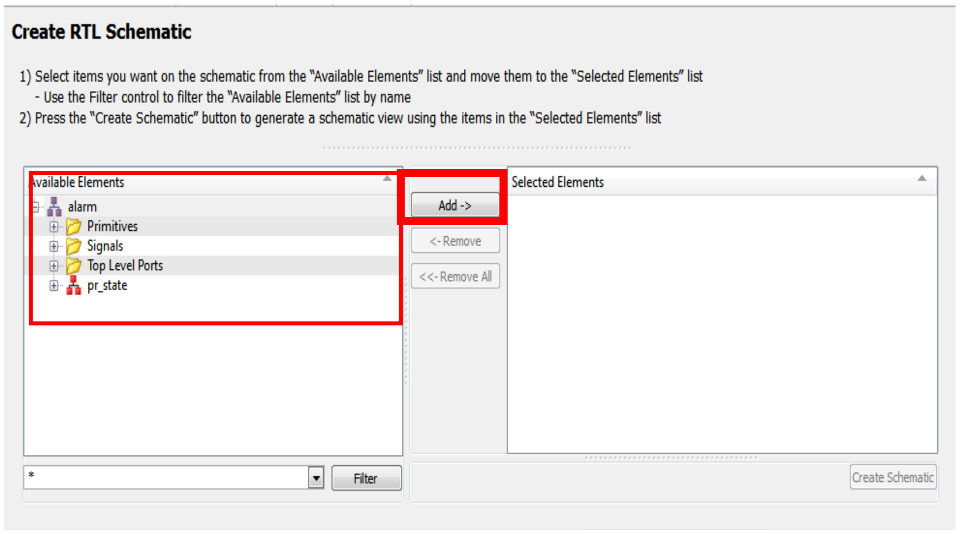
In this mode, the Explorer Wizard is bypassed and an initial schematic is created with only the top-level block displayed. You can then use the logic expansion capabilities of the Viewer to start expanding from the top-level block

You can also change the startup mode by selecting Edit->Preferences under the RTL/Tech Viewer page

[✓] Show this dialog on startup

OK

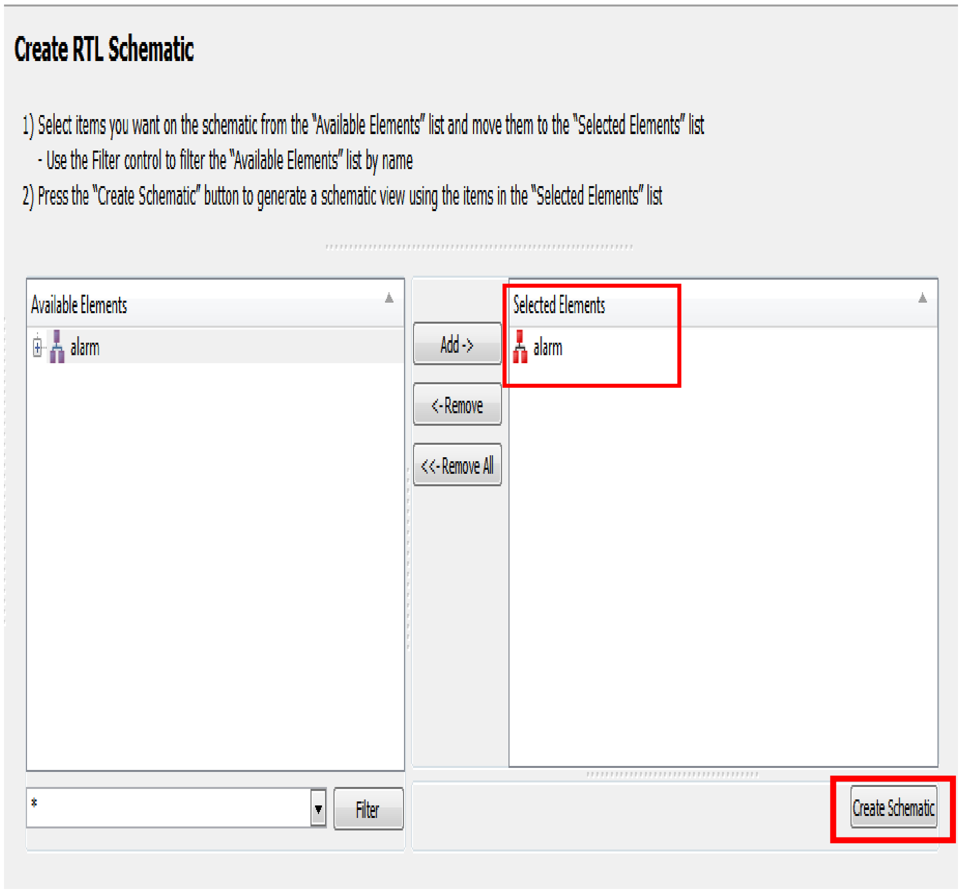
Figure-A27. Set RTL/Tech Viewer Startup Mode.


Create RTL Schematic

1) Select items you want on the schematic from the "Available Elements" list and move them to the "Selected Elements" list
- Use the Filter control to filter the "Available Elements" list by name
2) Press the "Create Schematic" button to generate a schematic view using the items in the "Selected Elements" list

Figure-A28. "Create RTL Schematic"-Wizard

Just Select the "alarm" from the "Available Elements", as shown in the figure-27. And add it to the "Selected Elements". The Click on "Create Schematic". After that you will have a schematic like figure-29.


Create RTL Schematic

1) Select items you want on the schematic from the "Available Elements" list and move them to the "Selected Elements" list
- Use the Filter control to filter the "Available Elements" list by name
2) Press the "Create Schematic" button to generate a schematic view using the items in the "Selected Elements" list

Figure-A29. "Create RTL Schematic"-Wizard (Continue).

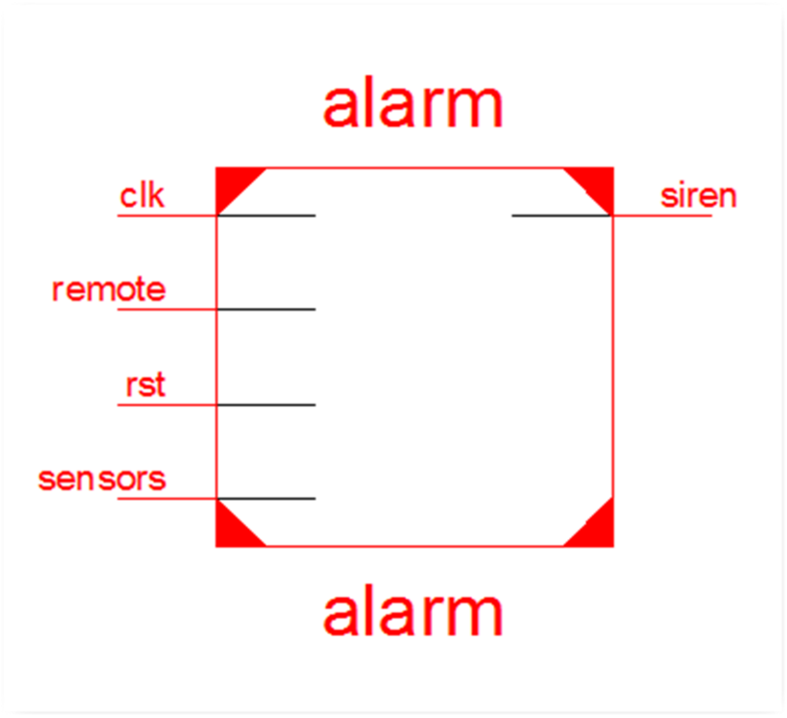


Figure-A30. RTL Schematic.

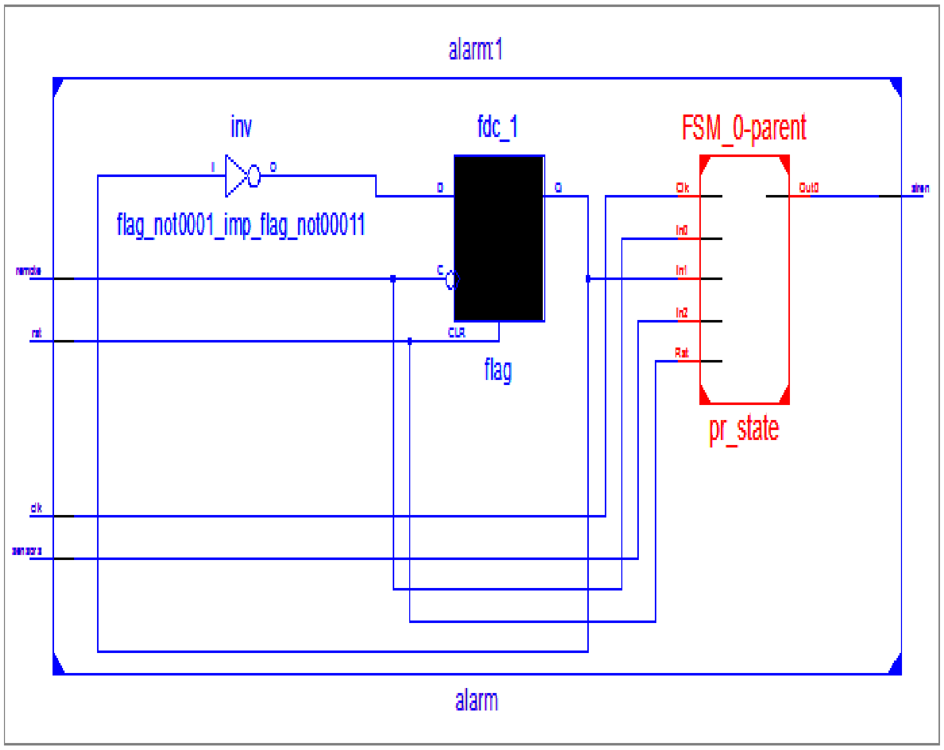
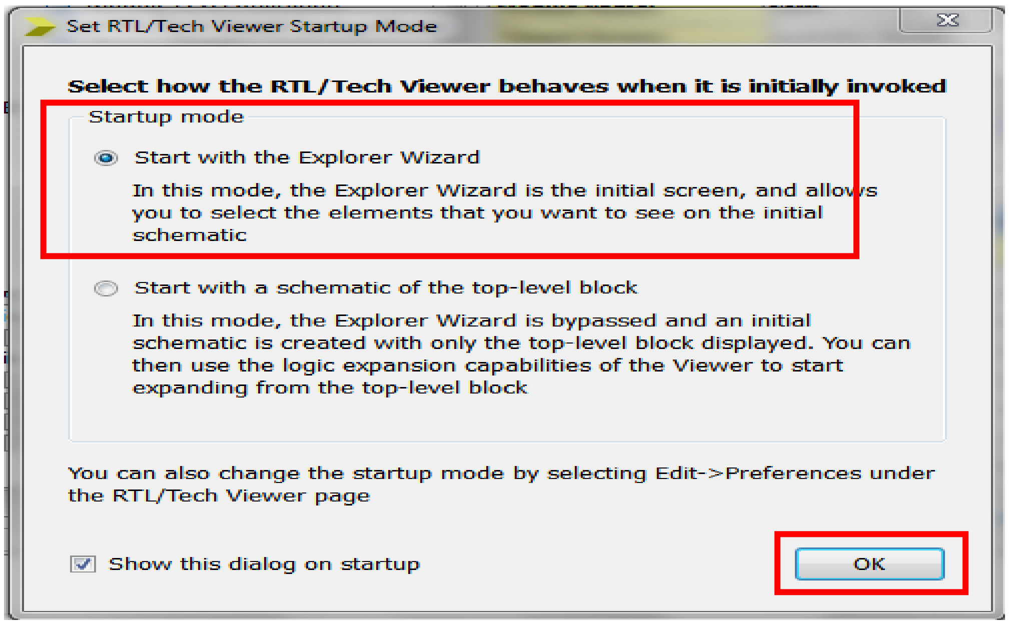


Figure-A31. Detail View of RTL Schematic (Double Click on Figure-29 to get this Schematic)

TECHNOLOGY SCHEMATIC


Set RTL/Tech Viewer Startup Mode

Select how the RTL/Tech Viewer behaves when it is initially invoked

Startup mode

[✓ Start with the Explorer Wizard]

In this mode, the Explorer Wizard is the initial screen, and allows you to select the elements that you want to see on the initial schematic

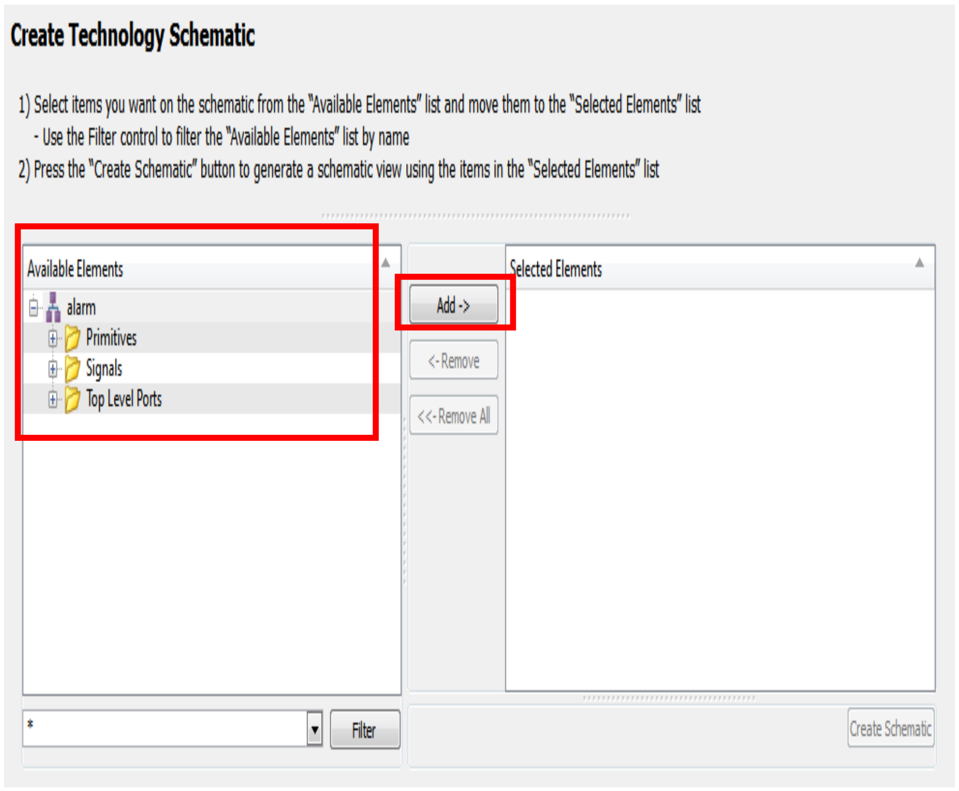
Start with a schematic of the top-level block

In this mode, the Explorer Wizard is bypassed and an initial schematic is created with only the top-level block displayed. You can then use the logic expansion capabilities of the Viewer to start expanding from the top-level block

You can also change the startup mode by selecting Edit->Preferences under the RTL/Tech Viewer page

[✓] Show this dialog on startup

Figure-A32. Set RTL/Tech Viewer Startup Mode.

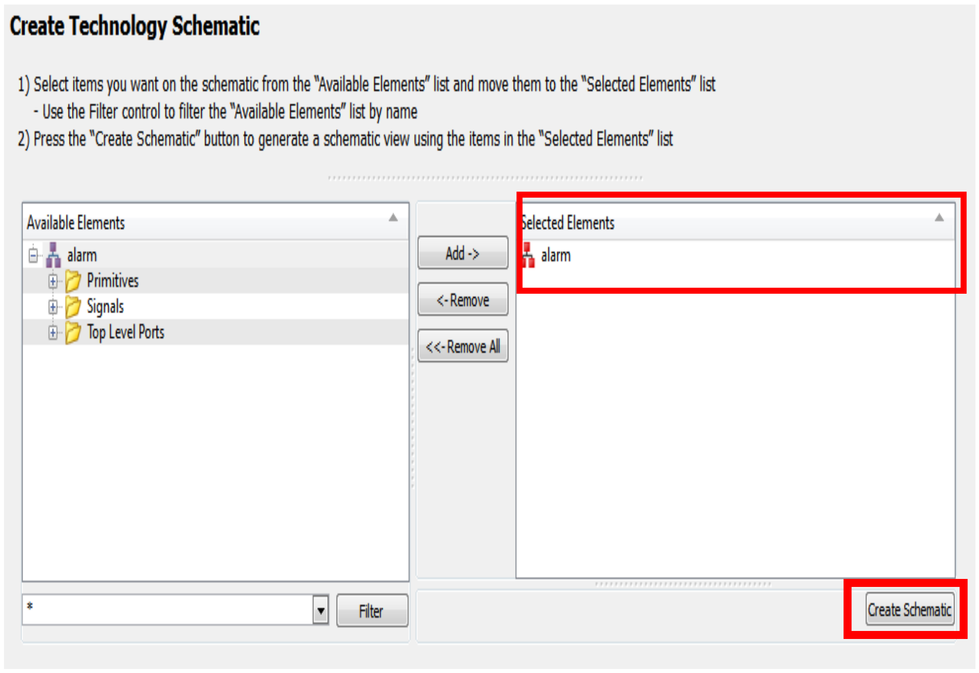

Create Technology Schematic

1) Select items you want on the schematic from the "Available Elements" list and move them to the "Selected Elements" list

- Use the Filter control to filter the "Available Elements" list by name

2) Press the "Create Schematic" button to generate a schematic view using the items in the "Selected Elements" list

Figure-A33. “Create Technology Schematic”-Wizard


Create Technology Schematic

1) Select items you want on the schematic from the "Available Elements" list and move them to the "Selected Elements" list
   - Use the Filter control to filter the "Available Elements" list by name
2) Press the "Create Schematic" button to generate a schematic view using the items in the "Selected Elements" list

Figure-A34. "Create Technology Schematic"-Wizard (Continue).

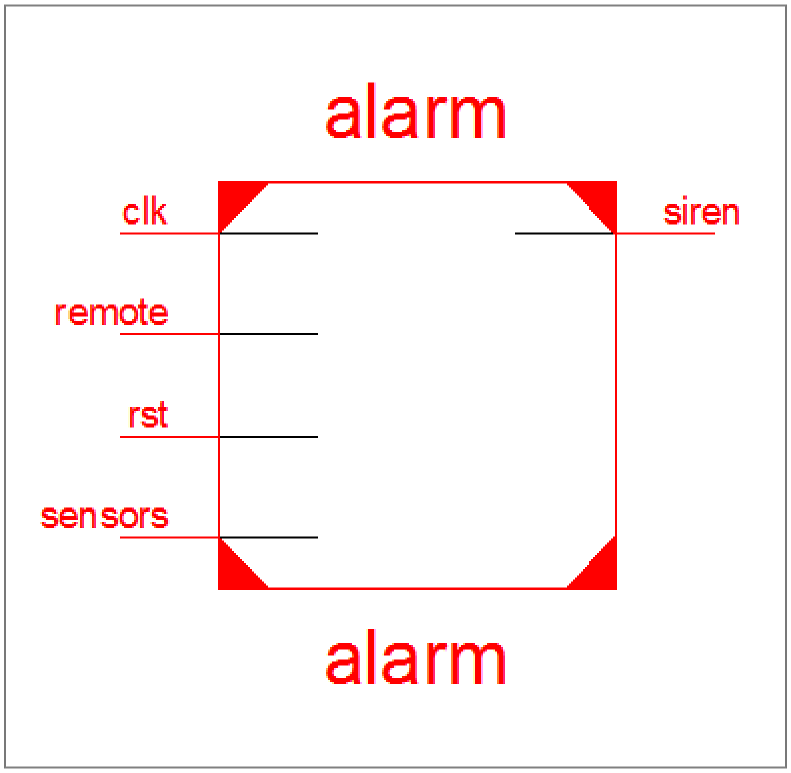


Figure-A35. Technology Schematic.

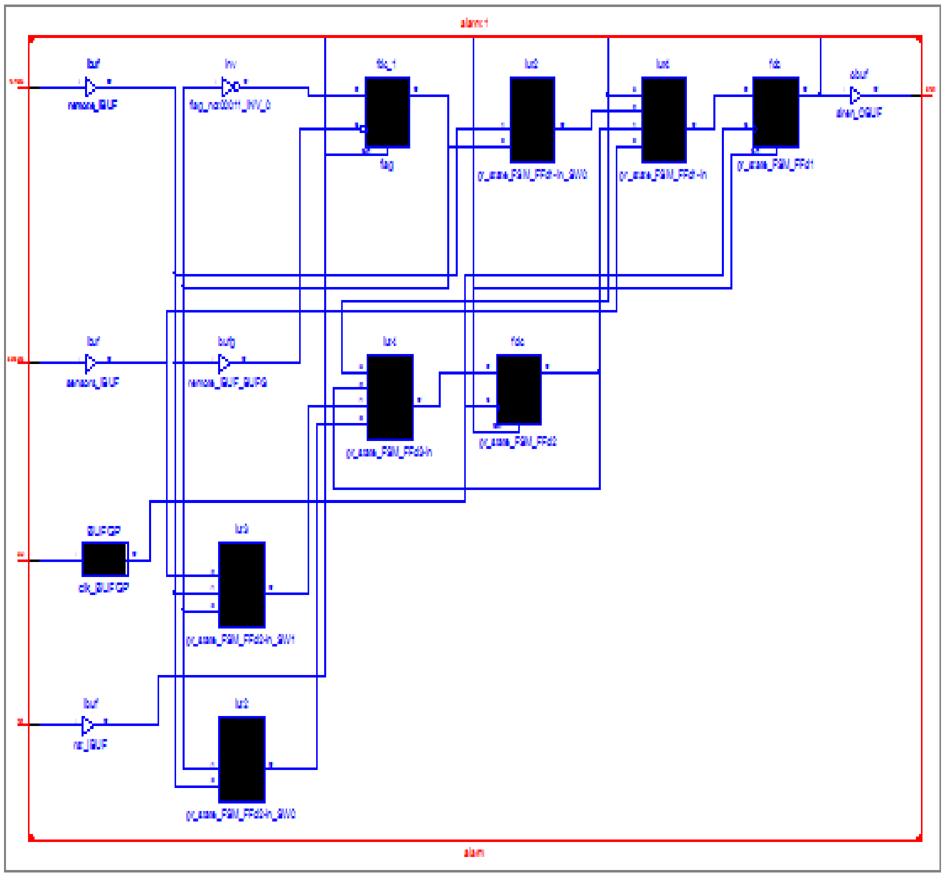


Figure-A36. View of Technology Schematic (Double)

APPENDIX-B

VHDL TUTORUAL

An Introduction to VHDL

VHDL is a Hardware Description Language (HDL) which describes the behavior of an electronic circuit or system. VHDL stands for Very High Speed Integrated Circuits (VHSIC) Hardware Description Language. In this section we will present three initial examples of VHDL which will help to illustrate the fundamental aspects regarding overall code structure.

- EXAMPLE-1 (SEQUENTIAL VHDL CODE)

Figure-1 represents the diagram of a D-type flip-flop (DFF)., triggered at the rising edge of the clock signal (clk) and with an asynchronous reset input (rst). When rst=’1’, the output must be turned low, regardless of clk. Otherwise, the output must copy the input.

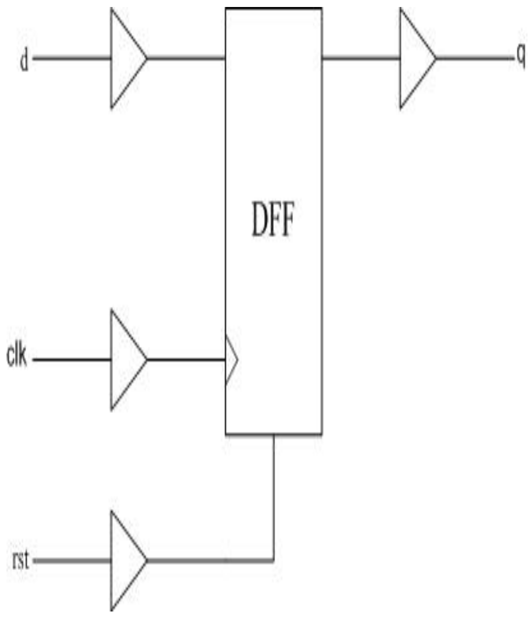
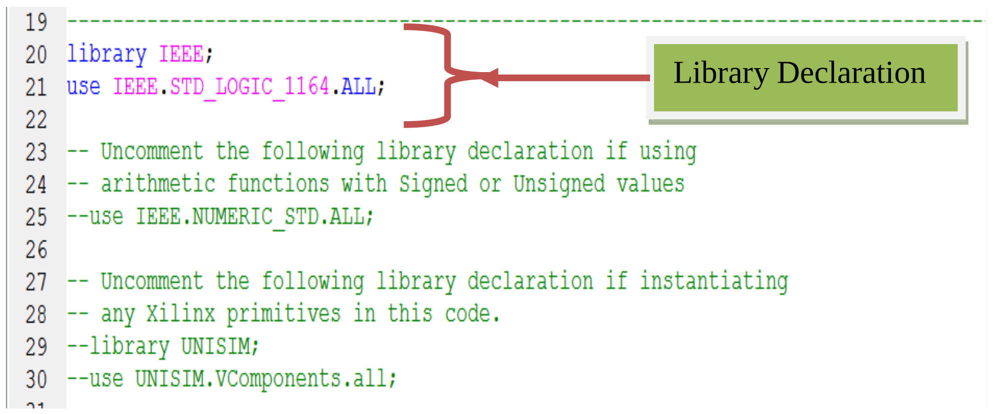
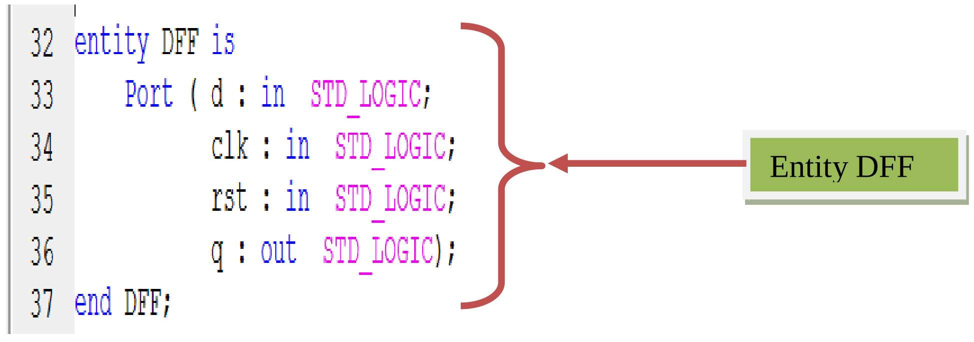


Figure-B1. DFF with Asynchronous Reset

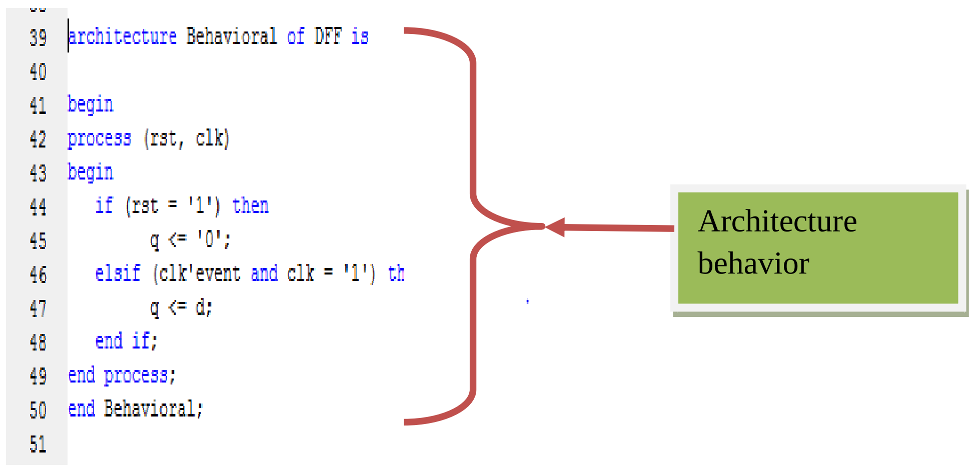
VHDL CODE.



```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```



```
entity DFF is
    Port ( d : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           q : out  STD_LOGIC);
end DFF;
```



```
architecture Behavioral of DFF is

begin
process (rst, clk)
begin
   if (rst = '1') then
        q <= '0';
   elsif (clk'event and clk = '1') th
        q <= d;
   end if;
end process;
end Behavioral;

```

<u>VHDL TEST BENCH CODE</u>

```
--------------------------------------------------------------------------------

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

 ENTITY DFF_TB IS

END DFF_TB;

 ARCHITECTURE BEHAVIOR OF DFF_TB IS
 COMPONENT DFF
   PORT(
        D : IN  STD_LOGIC;
        CLK : IN  STD_LOGIC;
        RST : IN  STD_LOGIC;
        Q : OUT  STD_LOGIC
       );
  END COMPONENT;

   --INPUTS
  SIGNAL D : STD_LOGIC := '1';
  SIGNAL CLK : STD_LOGIC := '0';
  SIGNAL RST : STD_LOGIC := '1';

--OUTPUTS
  SIGNAL Q : STD_LOGIC;
-- CLOCK PERIOD DEFINITIONS
  CONSTANT CLK_PERIOD : TIME := 10 NS;
 BEGIN
 -- INSTANTIATE THE UNIT UNDER TEST (UUT)
  UUT: DFF PORT MAP (
         D => D,
         CLK => CLK,
         RST => RST,
         Q => Q
       );

-- CLOCK PROCESS DEFINITIONS
  CLK_PROCESS :PROCESS
  BEGIN
```

```
                    CLK <= '0';
                    WAIT FOR CLK_PERIOD/2;
                    CLK <= '1';
                    WAIT FOR CLK_PERIOD/2;
 END PROCESS;
-- STIMULUS PROCESS
 STIM_PROC: PROCESS
 BEGIN
-- HOLD RESET STATE FOR 100 NS.
   WAIT FOR 100 NS;
   WAIT FOR CLK_PERIOD*10;
-- INSERT STIMULUS HERE
   WAIT;
 END PROCESS;
END;
--------------------------------------------------------------------------------
```

SIMULATION RESULTS

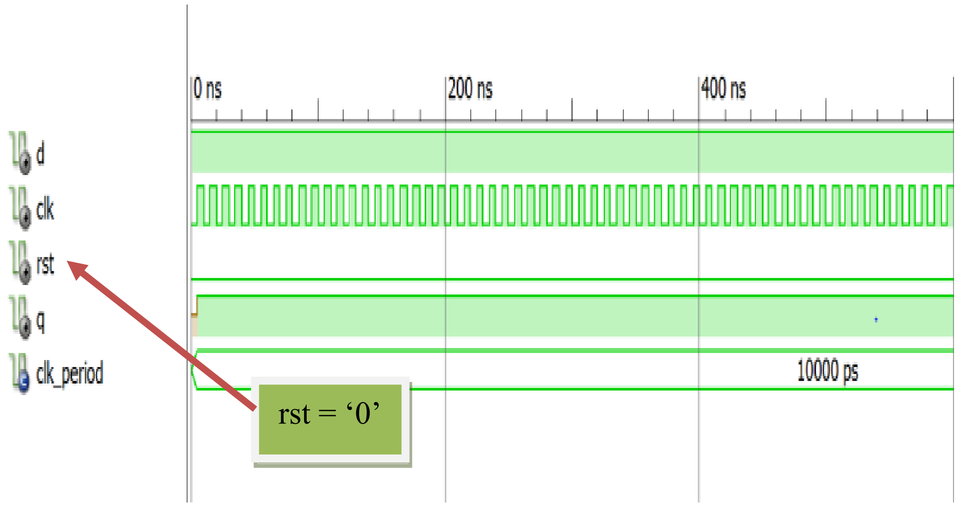


Figure-B2. When rst= '0', then q<=d.

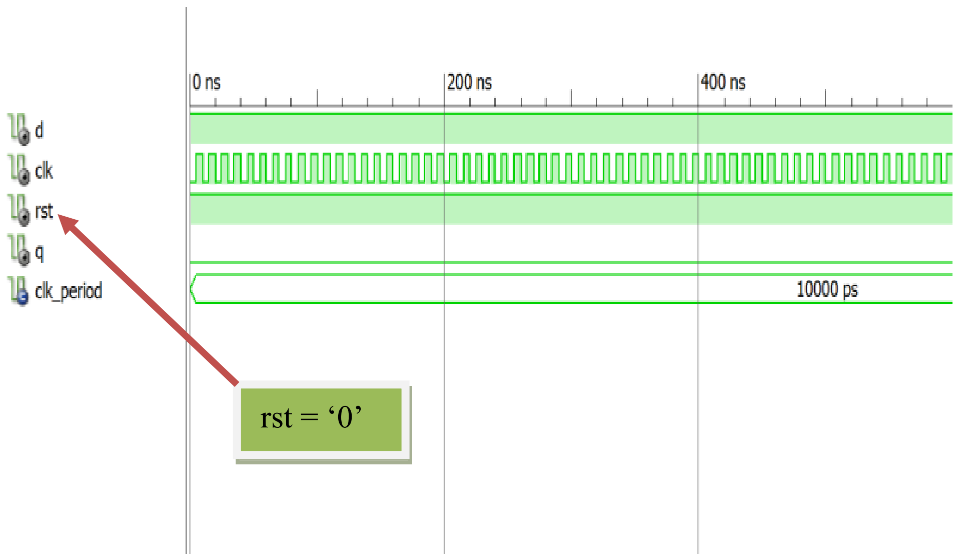


Figure-B3. When rst = ‘1’, then q <= ‘0’.

○ <u>EXAMPLE-2 [1] (SEQUENTIAL VHDL CODE)</u>

In this example we will demonstrate the implementation of a progressive 1-digit decimal counter (0→9→0). A top level diagram of the circuit is shown in the figure-2. It contains a single-bit input (clk) and 4-bit output (digit).

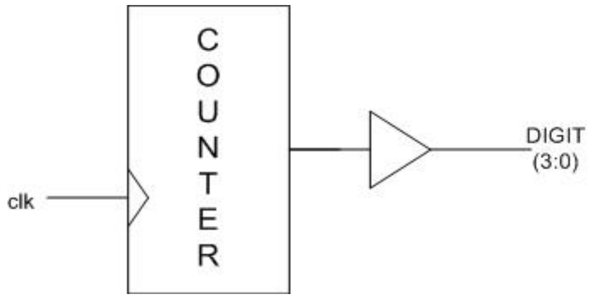
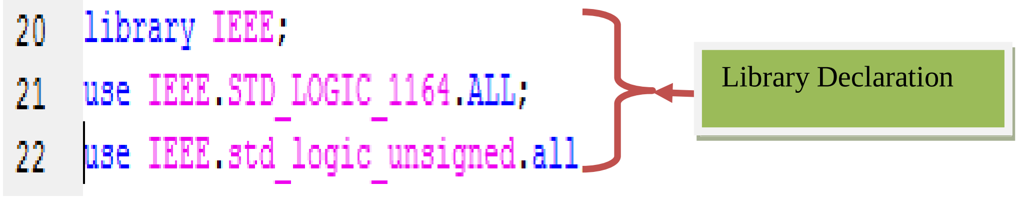
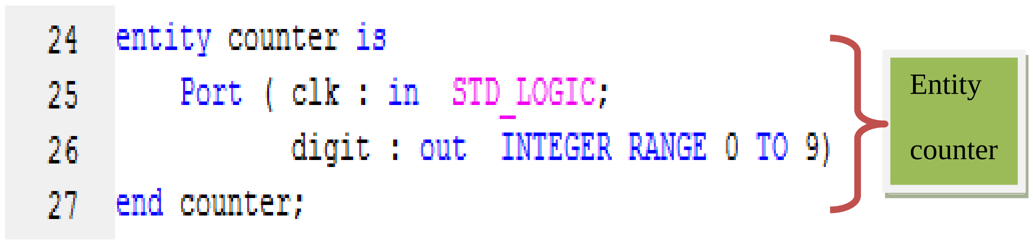


Figure-B4. Counter of Example-2

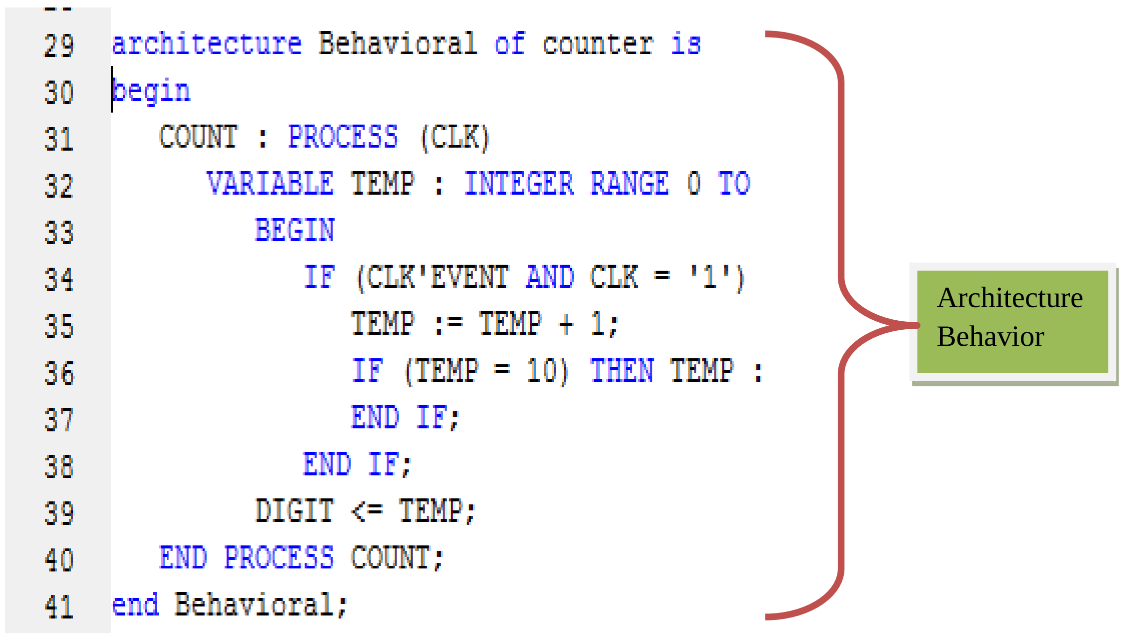
<u>VHDL CODE.</u>



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all
```



```
entity counter is
    Port ( clk : in  STD_LOGIC;
           digit : out  INTEGER RANGE 0 TO 9)
end counter;
```



```
architecture Behavioral of counter is
begin
   COUNT : PROCESS (CLK)
      VARIABLE TEMP : INTEGER RANGE 0 TO
         BEGIN
            IF (CLK'EVENT AND CLK = '1')
               TEMP := TEMP + 1;
               IF (TEMP = 10) THEN TEMP :
               END IF;
            END IF;
         DIGIT <= TEMP;
   END PROCESS COUNT;
end Behavioral;
```

Here in this example a variable namely "temp" is used to create the four flip-flops necessary to store 4-bit output signal.

<u>VHDL TEST BENCH CODE</u>

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY COUNTER_TESTBENCH IS

END COUNTER_TESTBENCH;

ARCHITECTURE BEHAVIOR OF COUNTER_TESTBENCH IS

COMPONENT COUNTER

PORT(

```
      CLK : IN  STD_LOGIC;
      DIGIT : OUT  INTEGER RANGE 0 TO 9
     );
   END COMPONENT;


  --INPUTS
  SIGNAL CLK : STD_LOGIC := '0';


--OUTPUTS
  SIGNAL DIGIT : INTEGER RANGE 0 TO 10;


-- CLOCK PERIOD DEFINITIONS
  CONSTANT CLK_PERIOD : TIME := 10 NS;
 BEGIN
 -- INSTANTIATE THE UNIT UNDER TEST (UUT)
  UUT: COUNTER PORT MAP (
        CLK => CLK,
        DIGIT => DIGIT
      );
-- CLOCK PROCESS DEFINITIONS
  CLK_PROCESS : PROCESS
  BEGIN
                    CLK <= '0';
                    WAIT FOR CLK_PERIOD/2;
                    CLK <= '1';
                    WAIT FOR CLK_PERIOD/2;
  END PROCESS;


-- STIMULUS PROCESS
  STIM_PROC: PROCESS
  BEGIN
    -- HOLD RESET STATE FOR 100 NS.
    WAIT FOR 100 NS;
   WAIT FOR CLK_PERIOD*10;
-- INSERT STIMULUS HERE
   WAIT;
  END PROCESS;
END;
```

<u>SIMULATION RESULT</u>

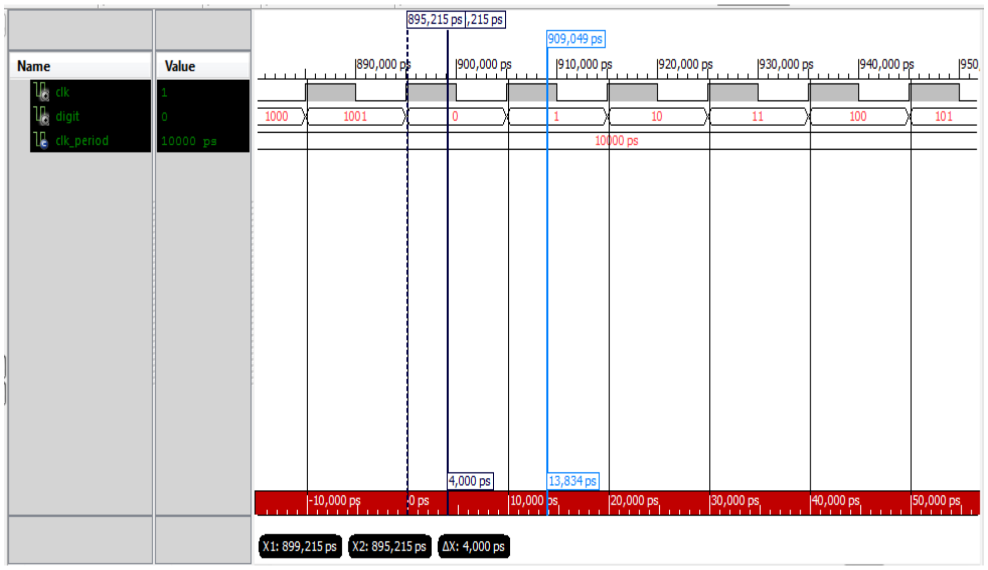


Figure-B5. Simulation Result

○ <u>EXAMPLE-3 [1] (CONCURRENT VHDL CODE)</u>

Figure-6 represents a 4-input, one bit per multiplexer. The output must be equal to the input selected by the selection bits, s1-s0.

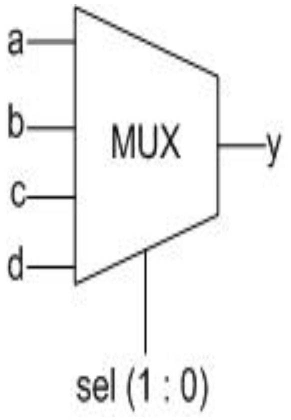
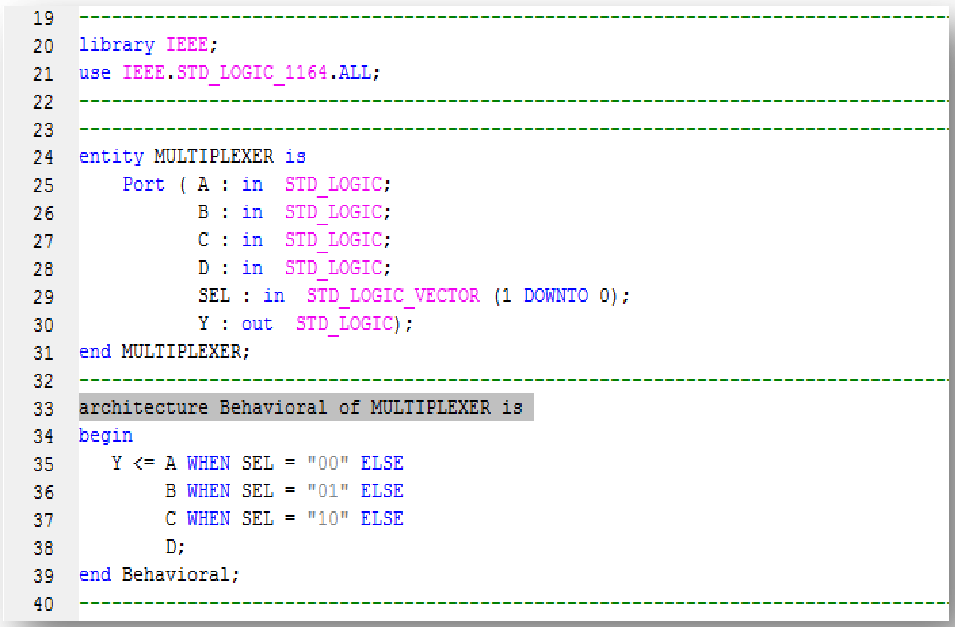


Figure-B6. 4-input Multiplexer.

VHDL CODE.



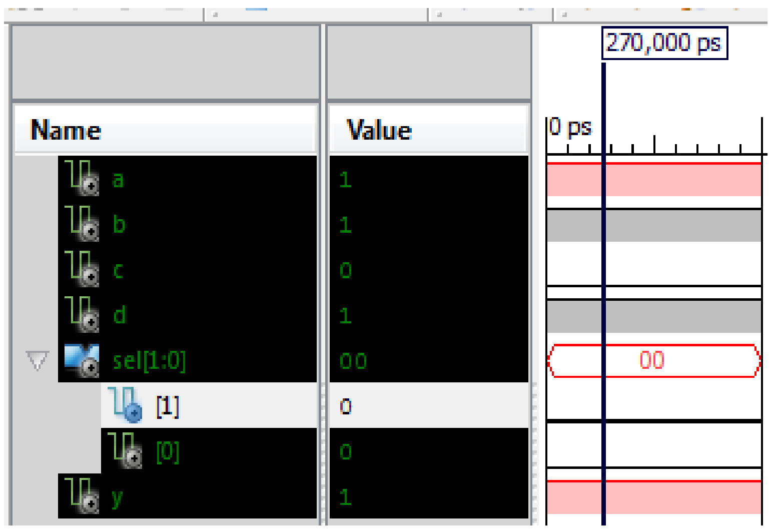
```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
entity MULTIPLEXER is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           C : in  STD_LOGIC;
           D : in  STD_LOGIC;
           SEL : in  STD_LOGIC_VECTOR (1 DOWNTO 0);
           Y : out  STD_LOGIC);
end MULTIPLEXER;
--------------------------------------------------------------------------------
architecture Behavioral of MULTIPLEXER is
begin
   Y <= A WHEN SEL = "00" ELSE
        B WHEN SEL = "01" ELSE
        C WHEN SEL = "10" ELSE
        D;
end Behavioral;
--------------------------------------------------------------------------------
```

VHDL TEST BENCH CODE

```
--------------------------------------------------------------------------------
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MUX_TB IS

END MUX_TB;

ARCHITECTURE BEHAVIOR OF MUX_TB IS
```

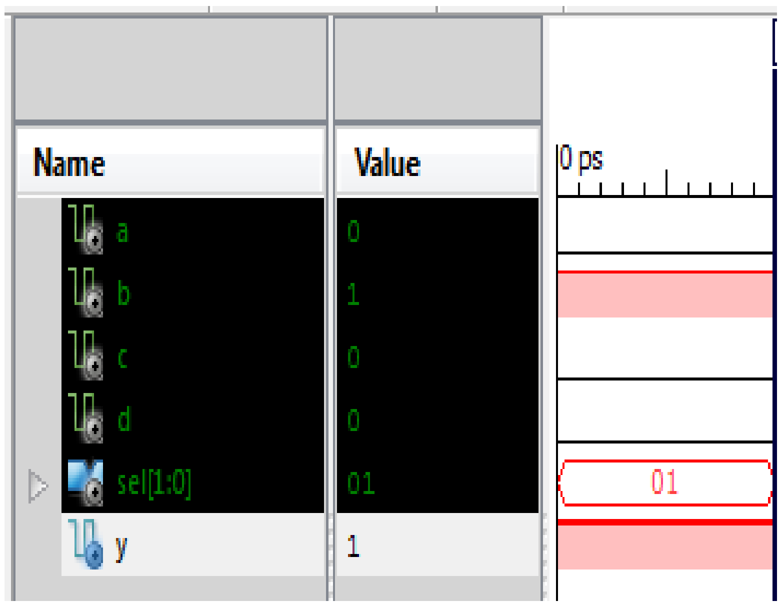
```
COMPONENT MULTIPLEXER
   PORT(
       A : IN  STD_LOGIC;
       B : IN  STD_LOGIC;
       C : IN  STD_LOGIC;
       D : IN  STD_LOGIC;
       SEL : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
       Y : OUT  STD_LOGIC
      );
   END COMPONENT;
  --INPUTS
  SIGNAL A : STD_LOGIC := '0';
  SIGNAL B : STD_LOGIC := '0';
  SIGNAL C : STD_LOGIC := '0';
  SIGNAL D : STD_LOGIC := '0';
  SIGNAL SEL : STD_LOGIC_VECTOR(1 DOWNTO 0) := (OTHERS => '0');
--OUTPUTS
  SIGNAL Y : STD_LOGIC;

BEGIN
UUT: MULTIPLEXER PORT MAP (
        A => A,
        B => B,
        C => C,
        D => D,
        SEL => SEL,
        Y => Y
      );
STIM_PROC: PROCESS
  BEGIN
  WAIT FOR 100 NS;

  WAIT;

  END PROCESS;

END;

--------------------------------------------------------------------------------
```

SIMULATION RESULTS



Name	Value
a	1
b	1
c	0
d	1
sel[1:0]	00
[1]	0
[0]	0
y	1

Figure-B7. Y <= A WHEN SEL = "00"



Name	Value
a	0
b	1
c	0
d	0
sel[1:0]	01
y	1

Figure-B8. Y <= A WHEN SEL = "01"

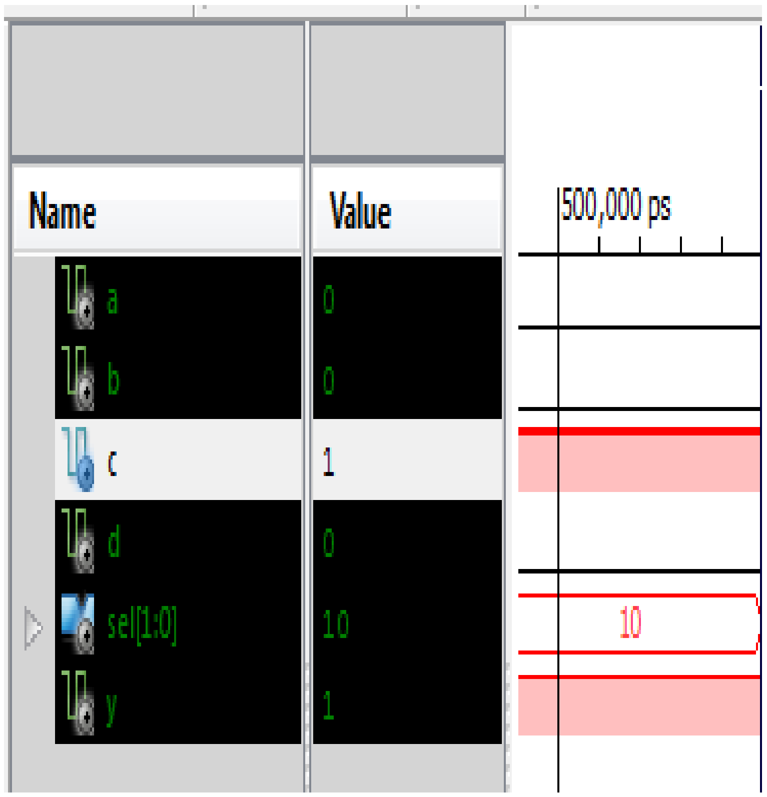


Figure-B9. Y <= A WHEN SEL = “10”

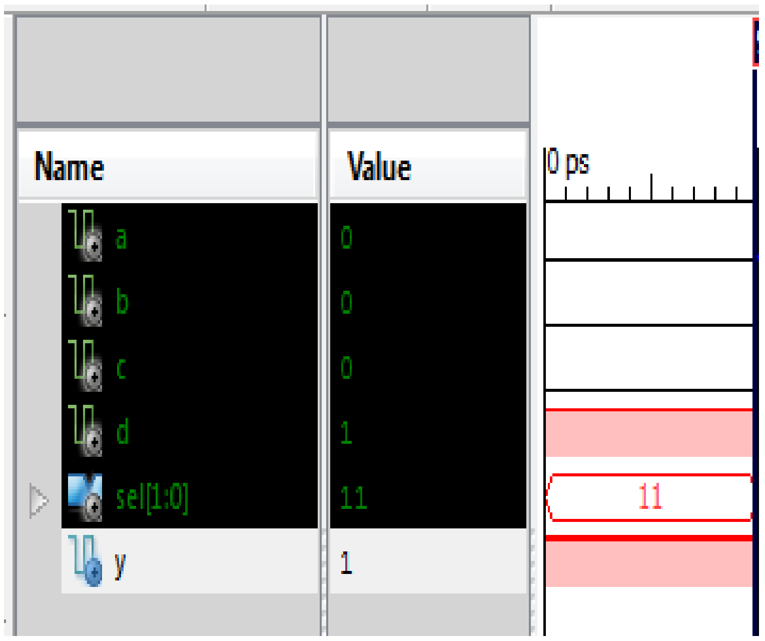


Figure-B10. Y <= A WHEN SEL = “11”

APPENDIX-C

VHDL CODE

C.1. VHDL PROGRAM
C.1.1. Rotary Switch

```
ENTITYROTARY_SW IS
   PORT (              CLK      : IN  STD_LOGIC;
                ROT_A              : IN STD_LOGIC;
                ROT_B              : IN STD_LOGIC;
                 STEP   : IN  STD_LOGIC;
                 DAC_A              : OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
                         );
ENDROTARY_SW;
ARCHITECTURE BEHAVIORAL OF ROTARY_SW IS
SIGNAL CLKDIV  : STD_LOGIC_VECTOR (7 DOWNTO 0):=(OTHERS=>'0');
SIGNAL FLAG    : STD_LOGIC_VECTOR (15 DOWNTO 0):=(OTHERS=>'0');
SIGNAL DAC_DATA  :STD_LOGIC_VECTOR (11 DOWNTO 0):=(OTHERS=>'0');
BEGIN
PSENSE_A: PROCESS(CLK)
BEGIN
IFRISING_EDGE(CLK)THEN
CLKDIV<=CLKDIV+1;
IFCLKDIV= X"00" THEN
FLAG<=FLAG(14 DOWNTO 0) &ROT_A;
IF FLAG= X"8000" THEN
IFROT_B='1' THEN
IF(STEP='1')THEN                   -----CODE FOR 100MV STEP
IF ( DAC_DATA<3931) THEN
                                                DAC_DATA<=DAC_DATA+164;
END IF;
ELSE                              -----CODE FOR 10MV STEP
                                     IF (DAC_DATA<4079) THEN
                                                DAC_DATA<=DAC_DATA+16;
                                     END IF;
END IF;
                         ELSE                                  ----VOLTAGE DECREMENT
IF(STEP='1')THEN                   -----CODE FOR 100MV STEP
                                     IF ( DAC_DATA>164) THEN
                                     DAC_DATA<=DAC_DATA-164;
                                     END IF;
```

```
ELSE                    -----CODE FOR 10MV STEP
                              IF (DAC_DATA>16) THEN
                              DAC_DATA<=DAC_DATA-16;
                                      END IF;
END IF;
END IF;
END IF;
END IF;
DAC_A<=DAC_DATA;
END IF;
END PROCESS;
END BEHAVIORAL;
```

DAC

```
ENTITY DAC IS
  PORT (          CLK        : IN  STD_LOGIC;
                   SPI_MOSI               : OUT STD_LOGIC;
                           SPI_MISO               : IN  STD_LOGIC;
                           SPI_SCK                : OUT STD_LOGIC;
                           DAC_CS                         : OUT STD_LOGIC;
                          DAC_CLR                : OUT STD_LOGIC;
                           AMP_CS                         : OUT STD_LOGIC;
                           SPI_SS_B               : OUT STD_LOGIC;
                           FPGA_INIT_B    : OUT STD_LOGIC;
                           SF_CE0                 : OUT STD_LOGIC;
                           AD_CONV                : OUT STD_LOGIC;
                          DAC_A      : IN   STD_LOGIC_VECTOR (11 DOWNTO 0):=X"FFF"
                           );
END DAC;
ARCHITECTURE BEHAVIORAL OF DAC IS
TYPE            EVENT_TYPE     IS       (S0,  S1,  S2,  S3,  S4 );
SIGNAL  SDACSTATE       : EVENT_TYPE;
SIGNAL  SDATA : STD_LOGIC_VECTOR (31 DOWNTO 0);
BEGIN
        DAC_CLR<= '1';
        SPI_SS_B  <= '1';
        SF_CE0   <= '1';
```

```
            FPGA_INIT_B  <= '1';
            AD_CONV   <= '0';
            AMP_CS <= '1';
            PDAC: PROCESS (CLK, SDACSTATE)
            CONSTANT         KDACA  : STD_LOGIC_VECTOR(7 DOWNTO 0)  := "00110010";                       --
UPDATE IMMEDIATELY
                        VARIABLEBITNR              : INTEGER;
            BEGIN

                                    IFRISING_EDGE (CLK)
                                    THEN
                                    CASESDACSTATE IS
                                    WHEN S0=>DAC_CS<= '1';
                                    SPI_SCK <= '0';
                                    SPI_MOSI<= '0';
                                    SDACSTATE<= S1;
                                    WHEN S1 =>DAC_CS<= '0';
                        BITNR   :=0;
                                    SDATA   <= "00000000" &KDACA&DAC_A& "0000";
                                    SDACSTATE<= S2;
                                    WHEN S2 =>DAC_CS         <= '0';                    -- LOOP: SET DATA
                                    SPI_SCK<= '0';
                                    SPI_MOSI              <= SDATA(31);
                                    SDATA   <= SDATA (30 DOWNTO 0) & '0';
                                    SDACSTATE<=    S3;
                                    WHEN S3 =>SPI_SCK        <= '1';
                                    BITNR := BITNR +1;
                                    IF (BITNR< 32) THEN                      -- SET CLOCK
                                    SDACSTATE<= S2;
                        ELSE
                                    SDACSTATE<= S4;
                END IF;
                                    WHEN S4 =>DAC_CS<= '1';                             -- OK
                                    SPI_SCK <= '0';
SDACSTATE<= S1;
                        END CASE;
                END IF;
```

```
                        END PROCESS;
END BEHAVIORAL;
```

C.1.2. DIS_VALUE

```
ENTITY DISVALUE IS
  PORT ( CLK : IN  STD_LOGIC;
DAC_DATA : IN  STD_LOGIC_VECTOR (11 DOWNTO 0);
DIS_DATA : OUT  STD_LOGIC_VECTOR (15 DOWNTO 0));
END DISVALUE;
ARCHITECTURE BEHAVIORAL OF DISVALUE IS
SIGNALVAL_ADC:STD_LOGIC_VECTOR (11 DOWNTO 0):=X"000";
        TYPE    EVENT_TYPE    IS        (S0,  S1,  S2,  S3,  S4,S5);
        SIGNAL  SDACSTATE       : EVENT_TYPE;
BEGIN
CONV: PROCESS(CLK)
VARIABLE DIG_0:STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
VARIABLE DIG_1:STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
VARIABLE DIG_2:STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
VARIABLE DIG_3:STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
BEGIN
IFRISING_EDGE(CLK) THEN
CASESDACSTATE IS
WHEN S0         =>
                VAL_ADC<=DAC_DATA;
                SDACSTATE<= S1;
        WHEN S1         =>
IF(VAL_ADC>1638) THEN
        DIG_3:=DIG_3+1;
VAL_ADC<=VAL_ADC-1638;
                        ELSE
        SDACSTATE<= S2;
                END IF;
        WHEN S2         =>
IF(VAL_ADC>163)THEN
        DIG_2:=DIG_2+1;
```

```
VAL_ADC<=VAL_ADC-164;
                              ELSE
                    SDACSTATE<= S3;
                              END IF;


          WHEN S3         =>
IF(VAL_ADC>17)THEN
          DIG_1:=DIG_1+1;
VAL_ADC<=VAL_ADC-17;
          ELSE
          SDACSTATE<= S4;
          END IF;
          WHEN S4         =>
IF(VAL_ADC>3)THEN
          DIG_0:=DIG_0+2;
VAL_ADC<=VAL_ADC-3;
                              ELSE
          SDACSTATE<= S5;
          END IF;
WHEN S5=>
DIS_DATA<=DIG_3 & DIG_2 & DIG_1 & DIG_0;
                      DIG_0:="0000";
          DIG_1:="0000";
                     DIG_2:="0000";
          DIG_3:="0000";
                    SDACSTATE<= S0;
          WHEN OTHERS =>SDACSTATE<= S0;
END CASE;
END IF;
END PROCESS;
END BEHAVIORAL;
```

C.1.3. LCD_DRIVER_CODE

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY LCD_DRIVER IS
PORT(   CLK             : IN    STD_LOGIC;
                RS      : OUT   STD_LOGIC;
        RW      : OUT   STD_LOGIC;
        ENABLE          : OUT   STD_LOGIC;
        LCD_DATA        : OUT    STD_LOGIC_VECTOR(3 DOWNTO 0);
                INDEX           : OUT    STD_LOGIC_VECTOR (7 DOWNTO 0);
                CHAR            : IN     STD_LOGIC_VECTOR (7 DOWNTO 0)
        );
END LCD_DRIVER;
-----------------------------------------------------------------
ARCHITECTURE BEHAVIORAL OF LCD_DRIVER IS
TYPE CHARSTATE_TYPE IS (I0, I1, I2, I3, I4, I5, I6, I7, I8, I9, I10, I11, I12, I13, I14, I15, I16, S0, S1, S2, S3, S4,
S5, S6, S7, S8, S9, S10, S11 );
        SIGNAL  CHARSTATE : CHARSTATE_TYPE := I0;
CONSTANT        T_INSTRWAIT     : INTEGER := 100000;            -- 2 MS
CONSTANT        T_WRPULSE       : INTEGER := 12;            -- 0,2 US (200 NS)
CONSTANT        T_SETUPHOLD     : INTEGER := 10;
CONSTANT        T_DATWAIT       : INTEGER := 2500;          -- 50 US
TYPE STATE_TYPE IS (INIT, WAIT_FOR_DATA,WRITE_INITL1,WRITE_ADDRH1,WRITE_ADDRH2,
WRITE_ADDRH3,WRITE_ADDRL1, WRITE_ADDRL2, WRITE_ADDRL3,CHK_BUSYI1,
        CHK_BUSYI2,WRITE_DATAH1,WRITE_DATAH2, WRITE_DATAH3,WRITE_DATAL1, WRITE_DATAL2,
WRITE_DATAL3,CHK_BUSYD1);
SIGNAL STATE : STATE_TYPE := INIT;
SIGNAL  SLCDADR                 : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  SLCDDAT         : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  SLCDWR          : STD_LOGIC;
SIGNAL  SLCDINITWR      :STD_LOGIC;
SIGNAL  SLCDRDY         : STD_LOGIC;


SIGNAL INT_ADDR         : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
SIGNAL INT_DATA         : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
SIGNAL ENRWRS           : STD_LOGIC_VECTOR( 2 DOWNTO 0 );
---------------------------------------------------------------------
```

```
BEGIN
PTEXTOUT:       PROCESS (CLK)
                VARIABLECNT : STD_LOGIC_VECTOR (31 DOWNTO 0);
                        VARIABLE I : STD_LOGIC_VECTOR ( 7 DOWNTO 0);                    --INTEGER
RANGE 0 TO 40;
                                BEGIN
                                INDEX<= I;
                                IFRISING_EDGE (CLK) THEN
                                CASE CHARSTATE IS
                                WHEN I0         =>SLCDWR<= '0';
                                SLCDINITWR<= '0';
                                SLCDADR<= X"01";                        -- WAIT
                                CNT     := (OTHERS => '0');
                                CHARSTATE<= I1;
                                WHEN I1         =>SLCDWR<= '0';
                                CNT := CNT + 1 ;
                        IF (CNT> 800000) THEN
                                CHARSTATE<= I2;
                        END IF;
                                WHEN I2         =>                      -- COMMAND 3
                                SLCDDAT<= X"03";
                                SLCDINITWR<= '1';
                                CNT := (OTHERS => '0');
                                CHARSTATE<= I3;
                                WHEN I3         =>SLCDINITWR  <= '0';
                                CNT := CNT + 1 ;
                        IF (CNT> 250000) THEN
                                CHARSTATE<= I4;
                        END IF;
                                WHEN I4         =>                      -- COMMAND 3
                                SLCDDAT<= X"03";
                                SLCDINITWR<= '1';
                                CNT := (OTHERS => '0');
                                CHARSTATE<= I5;
                                WHEN I5         =>SLCDINITWR  <= '0';
                                CNT := CNT + 1 ;
                        IF (CNT> 5000) THEN
```

```
                    CHARSTATE       <= I6;

                END IF;
                    WHEN I6             =>                              -- COMMAND 4
                    SLCDDAT<= X"03";
                    SLCDINITWR<= '1';
                    CNT := (OTHERS => '0');
                    CHARSTATE<= I7;
                    WHEN I7            =>SLCDINITWR  <= '0';
                    CNT := CNT + 1 ;
                IF (CNT> 5000) THEN
                    CHARSTATE<= I8;
                END IF;
                    WHEN I8             =>                      -- COMMAND 2
                    SLCDDAT<= X"02";
                    SLCDINITWR<= '1';
                    CNT := (OTHERS => '0');
                    CHARSTATE<= I9;
                    WHEN I9            =>SLCDINITWR  <= '0';
                    CNT := CNT + 1 ;
                IF (CNT> 250000) THEN
                    CHARSTATE<= I10;
                END IF;
--------------------------------------------------------------------------------
                    WHEN I10  =>SLCDADR<= X"01";   -- COMMAND 28
                    SLCDDAT<= X"28";
                    SLCDWR<= '1';
                    CNT := (OTHERS => '0');
                    CHARSTATE<= I11;
                    WHEN I11=>SLCDWR<= '0';
                    CNT := CNT + 1 ;
            IF (CNT> 250000) THEN
                    CHARSTATE       <= I12;
            END IF;
                    WHEN I12  =>SLCDADR<= X"01";   -- COMMAND 06
                    SLCDDAT<= X"06";
                SLCDWR<= '1';
                CNT := (OTHERS => '0');
                CHARSTATE<= I13;
                WHEN I13=>SLCDWR<= '0';
                CNT := CNT + 1 ;
            IF (CNT> 250000) THEN
                CHARSTATE      <= I14;
            END IF;
                WHEN I14  =>SLCDADR   <= X"01";  -- COMMAND 0C
                SLCDDAT<= X"0C";
                SLCDWR<= '1';
                CNT := (OTHERS => '0');
                CHARSTATE      <= I15;
                WHEN I15=>SLCDWR<= '0';
                CNT := CNT + 1 ;
            IF (CNT> 250000) THEN
                CHARSTATE<= I16;
            END IF;
                WHEN I16=>CHARSTATE<= S0;
                WHEN S0 =>SLCDWR<= '0';
                SLCDADR<= X"01";                    -- CLEAR DISPLAY
                SLCDDAT<= X"20";  -- SPACE
            IF (SLCDRDY = '1') THEN
                SLCDWR<= '1';
                CNT := (OTHERS => '0');
                CHARSTATE<= S1;
            END IF;
                WHEN S1 =>SLCDWR<= '0';
                CNT := CNT + 1 ;
            IF (CNT>CONV_STD_LOGIC_VECTOR (250000, 16)) THEN                -- 100.000 = 2MS
                CHARSTATE<= S2;
                I := (OTHERS=>'0');
            END IF;
                WHEN S2 => IF (CHAR = "00000000") THEN                   -- NULL CHARACTER
                    IF (I >= X"79") THEN
                                CNT := (OTHERS => '0');
                                CHARSTATE<= S11;
                    ELSE
```

```
                                        I := I + 1;
                                        CHARSTATE<= S2;
                                END IF;
                                ELSE
                                        SLCDWR<= '0';
                                        SLCDADR<= '1' &  I(6 DOWNTO 0);          -- THE CHARACTER
LOCATION
                                        SLCDDAT<= CHAR;
                                IF (SLCDRDY = '1') THEN
                                        SLCDWR<= '1';
                                        CNT := (OTHERS => '0');
                                        CHARSTATE<= S3;
                                END IF;
                                END IF;
                                        WHEN S3 =>SLCDWR<= '0';
                                        CHARSTATE<= S4;
                                        WHEN S4 =>SLCDWR<= '0';
                                IF (SLCDRDY = '1') THEN
                                        CHARSTATE<= S2;
                                        I := I + 1;
                                END IF;
                                WHEN S11=>CNT := CNT + 1;
                        IF (CNT> 10000000) THEN -- WAIT 200 MS : 5 UPDATES /SEC.
        CHARSTATE<= S0;                                                         END IF;
                                WHEN OTHERS =>CHARSTATE<= S0;
                                END CASE;
                        END IF;
                        END PROCESS;
---------------------------------------------------------------
        ENABLE          <= ENRWRS(2);
        RW<= ENRWRS(1);
        RS<= ENRWRS(0);
PLCDINSTR:
        PROCESS (CLK)
        VARIABLE COUNTER : INTEGER RANGE 0 TO 50000000 := 0;
        BEGIN
                IFRISING_EDGE (CLK) THEN
```

```
                    CASE STATE IS
                    WHEN INIT =>
                         COUNTER := COUNTER + 1;
                         IF ( COUNTER >= 500000 ) THEN -- 10 MS
                              COUNTER := 0;
                              STATE<= WAIT_FOR_DATA;
                         END IF;
---- WAIT FOR NEW DATA HERE
                         WHEN WAIT_FOR_DATA =>
                              COUNTER := 0;
                              IF (SLCDWR = '1') THEN
                                   INT_ADDR<= SLCDADR;
                                   INT_DATA<= SLCDDAT;
                                   STATE<= WRITE_ADDRH1; -- CHK_BUSY1;
                              ELSIF (SLCDINITWR ='1') THEN
                                   INT_ADDR<= SLCDADR;
                                   INT_DATA<= SLCDDAT;
                                   STATE<= WRITE_INITL1;
                    END IF;
                         WHEN WRITE_INITL1 =>
                                   COUNTER := COUNTER + 1;
                                   IF ( COUNTER>= T_WRPULSE ) THEN       -- 2 US WR TIME
                                        COUNTER := 0;
                                        STATE<= WAIT_FOR_DATA;
                                   END IF;
-- ADDRESS:  HIGH NIBBLE
                         WHEN WRITE_ADDRH1 =>
                              COUNTER := COUNTER + 1;
                              IF ( COUNTER>= T_WRPULSE ) THEN  -- 2 US WR TIME
                                   COUNTER := 0;
                                   STATE<= WRITE_ADDRH2;
                              END IF;
                         WHEN WRITE_ADDRH2 =>
                              COUNTER := COUNTER + 1;
                              IF (COUNTER >= T_SETUPHOLD) THEN
                                   COUNTER := 0;
                                   STATE<= WRITE_ADDRL1;
                    END IF;
-- ADDRESS LOW NIBBLE
                WHEN WRITE_ADDRL1 =>
                    COUNTER := COUNTER + 1;
                    IF ( COUNTER>= T_WRPULSE ) THEN
                        COUNTER := 0;
                        STATE<= WRITE_ADDRL2;
                    END IF;

                WHEN WRITE_ADDRL2 =>
                    COUNTER := COUNTER + 1;
                    IF (COUNTER >= T_SETUPHOLD) THEN
                        COUNTER := 0;
                        STATE<= CHK_BUSYI1;
                    END IF;
                WHEN CHK_BUSYI1 =>
                    COUNTER := COUNTER + 1;
                    IF ( COUNTER >= T_DATWAIT ) THEN
                        COUNTER := 0;
                        IF (INT_ADDR(7) = '1') THEN
                            STATE<= WRITE_DATAH2;
                        ELSE
                            STATE<= CHK_BUSYI2;
                        END IF;
                END IF;
                WHEN CHK_BUSYI2 =>
                    COUNTER := COUNTER + 1;
                    IF ( COUNTER >= T_INSTRWAIT ) THEN
                        COUNTER := 0;
                        STATE<= WRITE_DATAH2;
                    END IF;
---------- NOW THE DATA -------------------------------------------------
                WHEN WRITE_DATAH2 =>
                    COUNTER := COUNTER + 1;
                    IF ( COUNTER>= T_WRPULSE ) THEN
                        COUNTER := 0;
                        STATE<= WRITE_DATAH3;
                    END IF;
                WHEN WRITE_DATAH3 =>
                    COUNTER := COUNTER + 1;
                    IF (COUNTER >= T_SETUPHOLD) THEN
                        COUNTER := 0;
                        STATE<= WRITE_DATAL2;
                    END IF;
                WHEN WRITE_DATAL2 =>
                    COUNTER := COUNTER + 1;
                    IF ( COUNTER>= T_WRPULSE ) THEN
                        COUNTER := 0;
                        STATE<= WRITE_DATAL3;
                    END IF;
                WHEN WRITE_DATAL3 =>
                    COUNTER := COUNTER + 1;
                    IF (COUNTER >= T_SETUPHOLD) THEN
                        COUNTER := 0;
                        STATE<= CHK_BUSYD1;
                    END IF;
                WHEN CHK_BUSYD1 =>
                    COUNTER := COUNTER + 1;
                    IF ( COUNTER >= T_DATWAIT ) THEN            -- 50 US
                        COUNTER := 0;
                        STATE<= WAIT_FOR_DATA;   --WRITE_DATAH1;
                    END IF;
                WHEN OTHERS => STATE <= INIT;
            END CASE;
        END IF;
END PROCESS;
        SLCDRDY<= '1' WHEN (STATE = WAIT_FOR_DATA) ELSE '0';
        WITH STATE SELECT
        LCD_DATA<="ZZZZ" WHEN INIT,
            INT_DATA(3 DOWNTO 0)        WHEN WRITE_INITL1,
            INT_ADDR(7 DOWNTO 4)        WHEN WRITE_ADDRH1,
            INT_ADDR(7 DOWNTO 4)        WHEN WRITE_ADDRH2,
            INT_ADDR(3 DOWNTO 0)        WHEN WRITE_ADDRL1,
            INT_ADDR(3 DOWNTO 0)        WHEN WRITE_ADDRL2,
                INT_DATA(7 DOWNTO 4)          WHEN WRITE_DATAH1,
                INT_DATA(7 DOWNTO 4)          WHEN WRITE_DATAH2,
                INT_DATA(7 DOWNTO 4)          WHEN WRITE_DATAH3,
                INT_DATA(3 DOWNTO 0)          WHEN WRITE_DATAL1,
                INT_DATA(3 DOWNTO 0)          WHEN WRITE_DATAL2,
                INT_DATA(3 DOWNTO 0)          WHEN WRITE_DATAL3,
                 "ZZZZ"  WHENWAIT_FOR_DATA,
            "ZZZZ"  WHEN OTHERS;
--  ENABLE RW RS
        WITH STATE SELECT
                ENRWRS<= "000"          WHEN INIT,
                        "000"    WHEN WAIT_FOR_DATA,
                        "100"    WHEN WRITE_INITL1,
                        "100"    WHEN WRITE_ADDRH1,
                        "000"    WHEN WRITE_ADDRH2,
                        "100"    WHEN WRITE_ADDRL1,
                        "000"    WHEN WRITE_ADDRL2,
                        "001"    WHEN WRITE_DATAH1,  -- LCDDATA:
                        "101"    WHEN WRITE_DATAH2,             -- OUTPUT LCD DATA: ENABLE = 1;
                        "001"    WHEN WRITE_DATAH3,
                        "001"    WHEN WRITE_DATAL1,  -- LCDDATA:
                        "101"    WHEN WRITE_DATAL2,             -- OUTPUT LCD DATA: ENABLE = 1;
                        "001"    WHEN WRITE_DATAL3,
                        "000"    WHEN OTHERS;    END BEHAVIORAL;
```

C.2. VHDL Code

C.2.1. ADC Implementation

```
--
-- Definition of a dual port ROM for KCPSM2 or KCPSM3 program defined by adc_ctrl.psm
-- and assmbled using KCPSM2 or KCPSM3 assembler.
--
-- Standard IEEE libraries
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--
-- The Unisim Library is used to define Xilinx primitives. It is also used during
-- simulation. The source can be viewed at %XILINX%\vhdl\src\unisims\unisim_VCOMP.vhd
--
library unisim;
use unisim.vcomponents.all;
--
--
entity adc_ctrl is
  Port (     address : in std_logic_vector(9 downto 0);
        instruction : out std_logic_vector(17 downto 0);
         proc_reset : out std_logic;
                clk : in std_logic);
  end adc_ctrl;
--
architecture low_level_definition of adc_ctrl is
--
-- Declare signals internal to this module
--
signal jaddr     : std_logic_vector(10 downto 0);
signal jparity   : std_logic_vector(0 downto 0);
signal jdata     : std_logic_vector(7 downto 0);
signal doa       : std_logic_vector(7 downto 0);
signal dopa      : std_logic_vector(0 downto 0);
signal tdo1      : std_logic;
signal tdo2      : std_logic;
signal update    : std_logic;
signal shift     : std_logic;
signal reset     : std_logic;
signal tdi       : std_logic;
signal sel1      : std_logic;
signal drck1     : std_logic;
signal drck1_buf : std_logic;
signal sel2      : std_logic;
signal drck2     : std_logic;
signal capture   : std_logic;
signal tap5      : std_logic;
signal tap11     : std_logic;
signal tap17     : std_logic;
--
-- Attributes to define ROM contents during implementation synthesis.
-- The information is repeated in the generic map for functional simulation
--
attribute INIT_00 : string;
attribute INIT_01 : string;
attribute INIT_02 : string;
attribute INIT_03 : string;
attribute INIT_04 : string;
attribute INIT_05 : string;
attribute INIT_06 : string;
attribute INIT_07 : string;
attribute INIT_08 : string;
attribute INIT_09 : string;
attribute INIT_0A : string;
attribute INIT_0B : string;
attribute INIT_0C : string;
attribute INIT_0D : string;
attribute INIT_0E : string;
attribute INIT_0F : string;
attribute INIT_10 : string;
attribute INIT_11 : string;
attribute INIT_12 : string;
attribute INIT_13 : string;
attribute INIT_14 : string;
attribute INIT_15 : string;
attribute INIT_16 : string;
attribute INIT_17 : string;
attribute INIT_18 : string;
attribute INIT_19 : string;
attribute INIT_1A : string;
attribute INIT_1B : string;
attribute INIT_1C : string;
attribute INIT_1D : string;
attribute INIT_1E : string;
attribute INIT_1F : string;
attribute INIT_20 : string;
attribute INIT_21 : string;
attribute INIT_22 : string;
attribute INIT_23 : string;
attribute INIT_24 : string;
attribute INIT_25 : string;
attribute INIT_26 : string;
attribute INIT_27 : string;
attribute INIT_28 : string;
attribute INIT_29 : string;
attribute INIT_2A : string;
attribute INIT_2B : string;
attribute INIT_2C : string;
attribute INIT_2D : string;
attribute INIT_2E : string;
attribute INIT_2F : string;
attribute INIT_30 : string;
attribute INIT_31 : string;
attribute INIT_32 : string;
attribute INIT_33 : string;
attribute INIT_34 : string;
attribute INIT_35 : string;
attribute INIT_36 : string;
attribute INIT_37 : string;
attribute INIT_38 : string;
attribute INIT_39 : string;
attribute INIT_3A : string;
attribute INIT_3B : string;
attribute INIT_3C : string;
attribute INIT_3D : string;
attribute INIT_3E : string;
attribute INIT_3F : string;
attribute INITP_00 : string;
attribute INITP_01 : string;
attribute INITP_02 : string;
attribute INITP_03 : string;
attribute INITP_04 : string;
attribute INITP_05 : string;
attribute INITP_06 : string;
attribute INITP_07 : string;
--
-- Attributes to define ROM contents during implementation synthesis.
--
attribute INIT_00 of ram_1024_x_18 : label is
"E0060000020C01B301B301B301B301B3016102110523014E0211051001FB011F";
attribute INIT_01 of ram_1024_x_18 : label is
"E0068001600650164FFF548D2E0454862E014E00C0010FFF4092E005E0040001";
attribute INIT_02 of ram_1024_x_18 : label is
"0520A700A600050600E40127001163016200019960000199600102110520CC080";
attribute INIT_03 of ram_1024_x_18 : label is
"640463016200007401D1052DA7008601E7FFE6FF403B052B54362780017F0211";
attribute INIT_04 of ram_1024_x_18 : label is
"505A440501FC0018505A440301F80030505A440201EC0077505A440101D800EF";
attribute INIT_05 of ram_1024_x_18 : label is
"A7068672A700A600050600E401FF009C505A440101FF0038505A440601FE000C";
attribute INIT_06 of ram_1024_x_18 : label is
"547242FF5472431F407201D1053E546C4200546C43E063016200017802110517";
attribute INIT_07 of ram_1024_x_18 : label is
"01D18530650801D18530650901D1052E01D18530650A01024014007401D1053C";
attribute INIT_08 of ram_1024_x_18 : label is
"C0016004C000409200075492400880016004C000A00001D1052001D185306507";
attribute INIT_09 of ram_1024_x_18 : label is
"01D1053D01D10547021105100122D2000206020602060206620 5E00400015492";
attribute INIT_0A of ram_1024_x_18 : label is
"01D1053254B5400240DF01D1052001D1052001D1053154AC4001600401D1052D";
attribute INIT_0B of ram_1024_x_18 : label is
"54C7400440DF01D1052001D1052001D1053554BE400340DF01D1052001D10520";
attribute INIT_0C of ram_1024_x_18 : label is
"40DF01D1052001D1053001D1053254D0400540DF01D1052001D1053001D10531";
attribute INIT_0D of ram_1024_x_18 : label is
"01AE01D1053001D1053001D1053140DF01D1052001D1053001D1053554D94006";
```

```
attribute INIT_0E of ram_1024_x_18 : label is
"0008010E0A0F198008FF50EC2380080004000500060007004014540DF20054000";
attribute INIT_0F of ram_1024_x_18 : label is
"F680F530D4205D01200154EECA010900080003000206B790B680B53094205CF5";
attribute INIT_10 of ram_1024_x_18 : label is
"03A002000700060015701460A0005504C0018801F480010A08070005A000F790";
attribute INIT_11 of ram_1024_x_18 : label is
"00AEA0005511C1010208030E07000606B5309420411906075916F530D420010D";
attribute INIT_12 of ram_1024_x_18 : label is
"5526C101C008E001020023404301C008E001C2040108C008E008011FA000C008";
attribute INIT_13 of ram_1024_x_18 : label is
"060023804301C008E001C008E0010122C008E010C008E010011FA000C008E008";
attribute INIT_14 of ram_1024_x_18 : label is
"01D10550A0000608070A0608070A0808090A0808090A5539C101090008000700";
attribute INIT_15 of ram_1024_x_18 : label is
"01D1056501D1057A01D1056101D1056C01D1054201D1056F01D1056301D10569";
attribute INIT_16 of ram_1024_x_18 : label is
"057401D1056E01D1056F01D1054301D1052001D1054301D1054401D10541A000";
attribute INIT_17 of ram_1024_x_18 : label is
"0541A00001D1053D01D1054101D10556A00001D1056C01D1056F01D1057201D1";
attribute INIT_18 of ram_1024_x_18 : label is
"101012000194000E000E000E000E1100A00001D1053D01D1054401D1052F01D1";
attribute INIT_19 of ram_1024_x_18 : label is
"A00001D1156001D1152016100188A000803A80075997C00AA00011000194A00F";
attribute INIT_1A of ram_1024_x_18 : label is
"01A90314A00055AAC20101A40219A00055A5C10101A00128A00055A1C001000B";
attribute INIT_1B of ram_1024_x_18 : label is
"C440A4F8A000C440E40101A0C440E401A00055B4C40101AE0414A00055AFC301";
attribute INIT_1C of ram_1024_x_18 : label is
"C44004F001A401BE040604060406040714500 1A001BEC408A4F01450A00001B8";
attribute INIT_1D of ram_1024_x_18 : label is
"04F001A401B8C440040604060407040714500 1A001B8C440C40CA4F01450A000";
attribute INIT_1E of ram_1024_x_18 : label is
"E401400201A0C440E40101A0C440E401450201A0C440E401C440040EA000C440";
attribute INIT_1F of ram_1024_x_18 : label is
"01BE01AE01BE043001AEA00001A4C4400404D500000E000E000E000EA5F0C440";
```

```
attribute INIT_20 of ram_1024_x_18 : label is
"01A901A901C2050101C2050C01C2050601C2052801A401BE042001A401BE01A9";
attribute INIT_21 of ram_1024_x_18 : label is
"E703E602E901E8000133A00001C2C5C0A50FA00001C2C580A50F52172510A000";
attribute INIT_22 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000080010F00";
attribute INIT_23 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_24 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_25 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_26 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_27 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_28 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_29 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_2A of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_2B of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_2C of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_2D of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_2E of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_2F of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_30 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_31 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
```

```
attribute INIT_32 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_33 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_34 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_35 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_36 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_37 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_38 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_39 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_3A of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_3B of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_3C of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_3D of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_3E of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_3F of ram_1024_x_18 : label is
"421B000000000000000000000000000000000000000000000000000000000000";
attribute INITP_00 of ram_1024_x_18 : label is
"D34CD3FCDDF3743C55B0D0D0D0D0D003C50CDF16C0333293774CE88FFFF3CF";
attribute INITP_01 of ram_1024_x_18 : label is
"5776A957A03400F4F333CCCDF3337CCCDF3337CCCDF3334CCCCCAA234F353B34";
attribute INITP_02 of ram_1024_x_18 : label is
"2CCCB33333333332CCCCCCCCCAAAAB6A922088E8D892223A2DAA5ED4000B5B09";
attribute INITP_03 of ram_1024_x_18 : label is
"FCEE0AA20E38388A3EAA3E028FAA3C0B8A38B72DCB72DCB4B30E5D8C0EA8B333";
```

```
attribute INITP_04 of ram_1024_x_18 : label is
"00000000000000000000000000000000000000000000000CAAEC2C36FCCCCF3F";
attribute INITP_05 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_06 of ram_1024_x_18 : label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_07 of ram_1024_x_18 : label is
"C000000000000000000000000000000000000000000000000000000000000000";
--
begin
--
 --Instantiate the Xilinx primitive for a block RAM
 ram_1024_x_18: RAMB16_S9_S18
 --synthesis translate_off
 --INIT values repeated to define contents for functional simulation
 generic map (INIT_00 =>
X"E0060000020C01B301B301B301B301B3016102110523014E0211051001FB011F",
        INIT_01 => X"E0068001600650164FFF548D2E0454862E014E00C0010FFF4092E005E0040001",
        INIT_02 => X"0520A700A600050600E4012700116301620001996000019960010211052CC080",
        INIT_03 => X"6404630162000007401D1052DA7008601E7FFE6FF403B052B54362780017F0211",
        INIT_04 => X"505A440501FC0018505A440301F80030505A440201EC0077505A440101D800EF",
        INIT_05 => X"A7068672A700A600050600E401FF009C505A440101FF0038505A440601FE000C",
        INIT_06 => X"547242FF5472431F407201D1053E546C4200546C43E063016200017802110517",
        INIT_07 => X"01D18530650801D18530650901D1052E01D18530650A01024014007401D1053C",
        INIT_08 => X"C0016004C000409200075492400880016004C000A00001D1052001D185306507",
        INIT_09 => X"01D1053D01D10547021105100122D20002060206020602066205E00400015492",
        INIT_0A => X"01D1053254B5400240DF01D1052001D1052001D1053154AC4001600401D1052D",
        INIT_0B => X"54C7400440DF01D1052001D1052001D1053554BE400340DF01D1052001D10520",
        INIT_0C => X"40DF01D1052001D1053001D1053254D0400540DF01D1052001D1053001D10531",
        INIT_0D => X"01AE01D1053001D1053001D1053140DF01D1052001D1053001D1053554D94006",
        INIT_0E => X"0008010E0A0F198008FF50EC2380008000400050006000700401454DF20054000",
        INIT_0F => X"F680F530D4205D01200154EECA010900080003000206B790B680B53094205CF5",
        INIT_10 => X"03A002000700060015701460A0005504C0018801F480010A08070005A000F790",
        INIT_11 => X"00AEA0005511C1010208030E07000606B5309420411906075916F530D420010D",
        INIT_12 => X"5526C101C008E001020023404301C008E001C2040108C008E008011FA000C008",
        INIT_13 => X"060023804301C008E001C008E0010122C008E010C008E010011FA000C008E008",
        INIT_14 => X"01D10550A0000608070A0608070A0808090A0808090A5539C101090008000700",
        INIT_15 => X"01D1056501D1057A01D1056101D1056C01D1054201D1056F01D1056301D10569",
        INIT_16 => X"057401D1056E01D1056F01D1054301D1052001D1054301D1054401D10541A000",
        INIT_17 => X"0541A00001D1053D01D1054101D10556A00001D1056C01D1056F01D1057201D1",
        INIT_18 => X"101012000194000E000E000E000E1100A00001D1053D01D1054401D1052F01D1",
        INIT_19 => X"A00001D1156001D1152016100188A000803A80075997C00AA00011000194A00F",
        INIT_1A => X"01A90314A00055AAC20101A40219A00055A5C10101A00128A00055A1C001000B",
        INIT_1B => X"C440A4F8A000C440E40101A0C440E401A00055B4C40101AE0414A00055AFC301",
        INIT_1C => X"C44004F001A401BE040604060406040714500１A001BEC408A4F01450A00001B8",
        INIT_1D => X"04F001A401B8C440040604060407040714500１A001B8C440C40CA4F01450A000",
        INIT_1E => X"E401400201A0C440E40101A0C440E401450201A0C440E401C440040EA000C440",
        INIT_1F => X"01BE01AE01BE043001AEA00001A4C4400404D500000E000E000E000EA5F0C440",
        INIT_20 => X"01A901A901C2050101C2050C01C2050601C2052801A401BE042001A401BE01A9",
        INIT_21 => X"E703E602E901E8000133A00001C2C5C0A50FA00001C2C580A50F52172510A000",
        INIT_22 => X"0000000000000000000000000000000000000000000000000000000080010F00",
        INIT_23 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_24 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_25 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_26 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_27 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_28 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_29 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_2A => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_2B => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_2C => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_2D => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_2E => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_2F => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_30 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_31 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_32 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_33 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_34 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_35 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_36 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_37 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_38 => X"0000000000000000000000000000000000000000000000000000000000000000",
        INIT_39 => X"0000000000000000000000000000000000000000000000000000000000000000",
            INIT_3A => X"0000000000000000000000000000000000000000000000000000000000000000",
            INIT_3B => X"0000000000000000000000000000000000000000000000000000000000000000",
            INIT_3C => X"0000000000000000000000000000000000000000000000000000000000000000",
            INIT_3D => X"0000000000000000000000000000000000000000000000000000000000000000",
            INIT_3E => X"0000000000000000000000000000000000000000000000000000000000000000",
            INIT_3F => X"421B000000000000000000000000000000000000000000000000000000000000",
            INITP_00 =>
X"D34CD3FCDDF3743C55B0D0D0D0D0D003C50CDF16C0333293774CE88FFFF3CF",
            INITP_01 => X"5776A957A03400F4F333CCCDF3337CCCDF3337CCCDF3334CCCCCAA234F353B34",
            INITP_02 =>
X"2CCCB33333333332CCCCCCCCCAAAAB6A922088E8D892223A2DAA5ED4000B5B09",
            INITP_03 =>
X"FCEE0AA20E38388A3EAA3E028FAA3C0B8A38B72DCB72DCB4B30E5D8C0EA8B333",
            INITP_04 => X"00000000000000000000000000000000000000000000000CAAEC2C36FCCCCF3F",
            INITP_05 => X"0000000000000000000000000000000000000000000000000000000000000000",
            INITP_06 => X"0000000000000000000000000000000000000000000000000000000000000000",
            INITP_07 => X"C000000000000000000000000000000000000000000000000000000000000000")
 --synthesis translate_on
 port map(    DIB => "0000000000000000",
           DIPB => "00",
            ENB => '1',
            WEB => '0',
           SSRB => '0',
           CLKB => clk,
          ADDRB => address,
            DOB => instruction(15 downto 0),
           DOPB => instruction(17 downto 16),
            DIA => jdata,
           DIPA => jparity,
            ENA => sel1,
            WEA => '1',
           SSRA => '0',
           CLKA => update,
           ADDRA=> jaddr,
            DOA => doa(7 downto 0),
           DOPA => dopa);
--  v2_bscan: BSCAN_VIRTEX2
--  port map(   TDO1 => tdo1,
--         TDO2 => tdo2,
--           UPDATE => update,
--            SHIFT => shift,
--            RESET => reset,
--              TDI => tdi,
--             SEL1 => sel1,
--            DRCK1 => drck1,
--             SEL2 => sel2,
--            DRCK2 => drck2,
--      CAPTURE => capture);
  --buffer signal used as a clock
  upload_clock: BUFG
  port map( I => drck1,
          O => drck1_buf);
  -- Assign the reset to be active whenever the uploading subsystem is active
  proc_reset <= sel1;
  srlC1: SRLC16E
  --synthesis translate_off
  generic map (INIT => X"0000")
  --synthesis translate_on
  port map(   D => tdi,
           CE => '1',
          CLK => drck1_buf,
           A0 => '1',
           A1 => '0',
           A2 => '1',
           A3 => '1',
            Q => jaddr(10),
          Q15 => jaddr(8));
  flop1: FD
  port map ( D => jaddr(10),
           Q => jaddr(9),
           C => drck1_buf);
  srlC2: SRLC16E
  --synthesis translate_off
  generic map (INIT => X"0000")
```

```
--synthesis translate_on
port map(   D => jaddr(8),
           CE => '1',
          CLK => drck1_buf,
           A0 => '1',
           A1 => '0',
           A2 => '1',
           A3 => '1',
            Q => jaddr(7),
          Q15 => tap5);
flop2: FD
port map ( D => jaddr(7),
           Q => jaddr(6),
           C => drck1_buf);
srlC3: SRLC16E
--synthesis translate_off
generic map (INIT => X"0000")
--synthesis translate_on
port map(   D => tap5,
           CE => '1',
          CLK => drck1_buf,
           A0 => '1',
           A1 => '0',
           A2 => '1',
           A3 => '1',
            Q => jaddr(5),
          Q15 => jaddr(3));
flop3: FD
port map ( D => jaddr(5),
           Q => jaddr(4),
           C => drck1_buf);
srlC4: SRLC16E
--synthesis translate_off
generic map (INIT => X"0000")
--synthesis translate_on
port map(   D => jaddr(3),
           CE => '1',
         CLK => drck1_buf,
          A0 => '1',
          A1 => '0',
          A2 => '1',
          A3 => '1',
           Q => jaddr(2),
         Q15 => tap11);
flop4: FD
port map ( D => jaddr(2),
          Q => jaddr(1),
          C => drck1_buf);
srlC5: SRLC16E
--synthesis translate_off
generic map (INIT => X"0000")
--synthesis translate_on
port map(   D => tap11,
          CE => '1',
         CLK => drck1_buf,
          A0 => '1',
          A1 => '0',
          A2 => '1',
          A3 => '1',
           Q => jaddr(0),
         Q15 => jdata(7));
flop5: FD
port map ( D => jaddr(0),
          Q => jparity(0),
          C => drck1_buf);
srlC6: SRLC16E
--synthesis translate_off
generic map (INIT => X"0000")
--synthesis translate_on
port map(   D => jdata(7),
          CE => '1',
         CLK => drck1_buf,
          A0 => '1',
          A1 => '0',
          A2 => '1',
          A3 => '1',
           Q => jdata(6),
         Q15 => tap17);
flop6: FD
port map ( D => jdata(6),
          Q => jdata(5),
          C => drck1_buf);
srlC7: SRLC16E
--synthesis translate_off
generic map (INIT => X"0000")
--synthesis translate_on
port map(   D => tap17,
          CE => '1',
         CLK => drck1_buf,
          A0 => '1',
          A1 => '0',
          A2 => '1',
          A3 => '1',
           Q => jdata(4),
         Q15 => jdata(2));
flop7: FD
port map ( D => jdata(4),
          Q => jdata(3),
          C => drck1_buf);
srlC8: SRLC16E
--synthesis translate_off
generic map (INIT => X"0000")
--synthesis translate_on
port map(   D => jdata(2),
          CE => '1',
         CLK => drck1_buf,
          A0 => '1',
          A1 => '0',
          A2 => '1',
          A3 => '1',
           Q => jdata(1),
          Q15 => tdo1);
  flop8: FD
  port map ( D => jdata(1),
           Q => jdata(0),
           C => drck1_buf);
end low_level_definition;
--
----------------------------------------------------------------------------------
--
-- END OF FILE adc_ctrl.vhd
--
----------------------------------------------------------------------------------
```

C.3. ETH_LOOPBACK

```
MODULE ETH_LOOPBACK
  (
   INPUT  SYS_CLK,
   INPUT  RESET_IN_N,
   // PHY SIGNALS:
   INPUT       E_COL,
   INPUT       E_CRS,
   OUTPUT      E_MDC,
   INOUT       E_MDIO,
   INPUT       E_RX_CLK,
   INPUT       E_RX_DV,
   INPUT [4:0]  E_RXD,
   INPUT       E_TX_CLK,
   OUTPUT      E_TX_EN,
   OUTPUT [4:0] E_TXD,
   // DDR SDRAM SIGNALS:
   INOUT [15:0]  CNTRL0_DDR_DQ,
   OUTPUT [12:0] CNTRL0_DDR_A,
   OUTPUT [1:0]  CNTRL0_DDR_BA,
   OUTPUT       CNTRL0_DDR_CKE,
   OUTPUT       CNTRL0_DDR_CS_N,
   OUTPUT       CNTRL0_DDR_RAS_N,
   OUTPUT       CNTRL0_DDR_CAS_N,
 OUTPUT        CNTRL0_DDR_WE_N,
 OUTPUT [1:0]  CNTRL0_DDR_DM,
 INOUT         CNTRL0_RST_DQS_DIV,
 INOUT [1:0]   CNTRL0_DDR_DQS,
 OUTPUT [0:0]  CNTRL0_DDR_CK,
 OUTPUT [0:0]  CNTRL0_DDR_CK_N,
 // LED SIGNALS
 OUTPUT [7:0]  LED
 );

// WISHBONE MASTER 1: ETHERNET DMA
WIRE [31:0]       M1_WB_ADR_I;
WIRE [31:0]       M1_WB_DAT_I;
WIRE [3:0]        M1_WB_SEL_I;
WIRE              M1_WB_CYC_I;
WIRE              M1_WB_STB_I;
WIRE              M1_WB_WE_I;
WIRE [31:0]       M1_WB_DAT_O;
WIRE              M1_WB_ACK_O;
WIRE              M1_WB_ERR_O;

// WISHBONE MASTER 2: HOST (LOOPBACK CONTROLLER)
WIRE [31:0]       M2_WB_ADR_I;
WIRE [31:0]       M2_WB_DAT_I;
WIRE [3:0]        M2_WB_SEL_I;
WIRE              M2_WB_CYC_I;
WIRE              M2_WB_STB_I;
WIRE              M2_WB_WE_I;
WIRE [31:0]       M2_WB_DAT_O;
WIRE              M2_WB_ACK_O;
WIRE              M2_WB_ERR_O;

// WISHBONE SLAVE 1: ETHERNET REGISTERS AND BUFFER DESCRIPTORS
WIRE [31:0]       S1_WB_DAT_I;
WIRE              S1_WB_ACK_I;
WIRE              S1_WB_ERR_I;
WIRE [31:0]       S1_WB_ADR_O;
WIRE [31:0]      S1_WB_DAT_O;
WIRE [3:0]       S1_WB_SEL_O;
WIRE             S1_WB_WE_O;
WIRE             S1_WB_CYC_O;
WIRE             S1_WB_STB_O;

// WISHBONE SLAVE 2: MEMORY
WIRE [31:0]      S2_WB_DAT_I;
WIRE             S2_WB_ACK_I;
WIRE             S2_WB_ERR_I;
WIRE [31:0]      S2_WB_ADR_O;
WIRE [31:0]      S2_WB_DAT_O;
WIRE [3:0]       S2_WB_SEL_O;
WIRE             S2_WB_WE_O;
WIRE             S2_WB_CYC_O;
WIRE             S2_WB_STB_O;

WIRE             ETHER_MDO;
WIRE             ETHER_MDIO_E;

// FOR THE USER INTERFACE TO THE DDR
WIRE [2:0]       COMMAND;
WIRE [((`DATA_MASK_WIDTH*2)-1):0] DATA_MASK;
WIRE [((`DATA_WIDTH*2)-1):0]     OUTPUT_DATA;
WIRE [((`DATA_WIDTH*2)-1):0]     DDR_INPUT_DATA;
WIRE [((`ROW_ADDRESS +
     `COL_AP_WIDTH +
     `BANK_ADDRESS)-1):0]     INPUT_ADDRESS;
WIRE                       CMD_ACK;
WIRE                       AUTO_REF_REQ;
WIRE                       AR_DONE;
WIRE                       BURST_DONE;

WIRE                       DATA_VALID_OUT;
WIRE                       INIT_DONE;

// FOR THE INFRASTRUCTURE:
```

```
WIRE                WAIT_200US;
WIRE                SYS_RST;
WIRE                SYS_RST90;
WIRE                SYS_RST180;
WIRE [4:0]            DELAY_SEL_VAL;
WIRE                CLK0;
WIRE                CLK90;


// TRI-STATE:
ASSIGN E_MDIO = ETHER_MDIO_E ? ETHER_MDO : 1'BZ;
//ASSIGN LED = 8'B0;


// INSTANTIATE THE ETHERNET MODULE
ETH_TOP ETHERNET
 (
  // WISHBONE COMMON
  .WB_CLK_I(CLK0),
  .WB_RST_I(SYS_RST),

  // WISHBONE SLAVE
  .WB_DAT_I(S1_WB_DAT_O),
  .WB_DAT_O(S1_WB_DAT_I),
  .WB_ADR_I(S1_WB_ADR_O[11:2]),
  .WB_SEL_I(S1_WB_SEL_O),
  .WB_WE_I(S1_WB_WE_O),
  .WB_CYC_I(S1_WB_CYC_O),
  .WB_STB_I(S1_WB_STB_O),
  .WB_ACK_O(S1_WB_ACK_I),
  .WB_ERR_O(S1_WB_ERR_I),

  // WISHBONE MASTER
  .M_WB_ADR_O(M1_WB_ADR_I),
  .M_WB_SEL_O(M1_WB_SEL_I),
  .M_WB_WE_O(M1_WB_WE_I),
  .M_WB_DAT_O(M1_WB_DAT_I),
    .M_WB_DAT_I(M1_WB_DAT_O),
    .M_WB_CYC_O(M1_WB_CYC_I),
    .M_WB_STB_O(M1_WB_STB_I),
    .M_WB_ACK_I(M1_WB_ACK_O),
    .M_WB_ERR_I(M1_WB_ERR_O),

    //TX
    .MTX_CLK_PAD_I(E_TX_CLK),
    .MTXD_PAD_O(E_TXD[3:0]),
    .MTXEN_PAD_O(E_TX_EN),
    .MTXERR_PAD_O(E_TXD[4]),

    //RX
    .MRX_CLK_PAD_I(E_RX_CLK),
    .MRXD_PAD_I(E_RXD[3:0]),
    .MRXDV_PAD_I(E_RX_DV),
    .MRXERR_PAD_I(E_RXD[4]),
    .MCOLL_PAD_I(E_COL),
    .MCRS_PAD_I(E_CRS),

    // MIIM
    .MDC_PAD_O(E_MDC),
    .MD_PAD_I(E_MDIO),
    .MD_PAD_O(ETHER_MDO),
    .MD_PADOE_O(ETHER_MDIO_E),

    .INT_O(ETHER_INT)
//    ,.LED(LED[3:0])
    );



 // INSTANTIATE THE MEMORY
 // (ACTUALLY A WISHBONE INTERFACE TO MIG INTERFACE CONVERTER):
 MEMORY_WB_TO_MIG MEM
   (
    // WB INTERFACE:
    .WB_CLK_I(CLK0),
  .WB_RST_I(SYS_RST),
  .WB_ADR_I(S2_WB_ADR_O),
  .WB_SEL_I(S2_WB_SEL_O),
  .WB_WE_I(S2_WB_WE_O),
  .WB_CYC_I(S2_WB_CYC_O),
  .WB_STB_I(S2_WB_STB_O),
  .WB_ACK_O(S2_WB_ACK_I),
  .WB_ERR_O(S2_WB_ERR_I),
  .WB_DAT_O(S2_WB_DAT_I),
  .WB_DAT_I(S2_WB_DAT_O),
  // MIG INTERFACE:
  // (NOTE THAT IT ALSO PROVIDES THE SYSTEM CLOCK)
  .MIG_CLK90(CLK90),
  .MIG_INIT_DONE(INIT_DONE),
  .MIG_OUTPUT_DATA(OUTPUT_DATA),
  .MIG_INPUT_DATA(DDR_INPUT_DATA),
  .MIG_INPUT_ADDRESS(INPUT_ADDRESS),
  .MIG_COMMAND(COMMAND),
  .MIG_CMD_ACK(CMD_ACK),
  .MIG_DATA_VALID(DATA_VALID_OUT),
  .MIG_BURST_DONE(BURST_DONE),
  .MIG_AR_DONE(AR_DONE),
  .MIG_AUTO_REF_REQ(AUTO_REF_REQ),
  .MIG_WAIT_200US(WAIT_200US),
  .MIG_DATA_MASK(DATA_MASK)
  );


// INSTANTIATE THE MIG MEMORY CONTROLLER:
VLOG_BL2CL25_TOP_0 TOP0
 (
  .AUTO_REF_REQ        (AUTO_REF_REQ),
  .WAIT_200US          (WAIT_200US),
  .RST_DQS_DIV_IN      (CNTRL0_RST_DQS_DIV),
  .RST_DQS_DIV_OUT     (CNTRL0_RST_DQS_DIV),
  .USER_INPUT_DATA     (DDR_INPUT_DATA),
  .USER_OUTPUT_DATA    (OUTPUT_DATA),
 .USER_DATA_VALID      (DATA_VALID_OUT),
 .USER_INPUT_ADDRESS    (INPUT_ADDRESS[((`ROW_ADDRESS + `COL_AP_WIDTH
                                + `BANK_ADDRESS)-1):0]),
 .USER_COMMAND_REGISTER (COMMAND),
 .USER_CMD_ACK         (CMD_ACK),
 .BURST_DONE           (BURST_DONE),
 .INIT_VAL             (INIT_DONE),
 .AR_DONE              (AR_DONE),
 .DDR_DQS              (CNTRL0_DDR_DQS),
 .DDR_DQ               (CNTRL0_DDR_DQ),
 .DDR_CKE              (CNTRL0_DDR_CKE),
 .DDR_CS_N             (CNTRL0_DDR_CS_N),
 .DDR_RAS_N            (CNTRL0_DDR_RAS_N),
 .DDR_CAS_N            (CNTRL0_DDR_CAS_N),
 .DDR_WE_N             (CNTRL0_DDR_WE_N),
 .DDR_BA               (CNTRL0_DDR_BA),
 .DDR_A                (CNTRL0_DDR_A),
 .DDR_DM               (CNTRL0_DDR_DM),
 .USER_DATA_MASK       (DATA_MASK),
 .DDR_CK               (CNTRL0_DDR_CK),
 .DDR_CK_N             (CNTRL0_DDR_CK_N),
 .CLK_INT              (CLK0),
 .CLK90_INT            (CLK90),
 .DELAY_SEL_VAL        (DELAY_SEL_VAL),
 .SYS_RST_VAL          (SYS_RST),
 .SYS_RST90_VAL        (SYS_RST90),
 .SYS_RST180_VAL       (SYS_RST180)
 );


// INSTANTIATE THE DCM AND THE DELAY
VLOG_BL2CL25_INFRASTRUCTURE_TOP INFRASTRUCTURE_TOP0
 (
 .SYS_CLK          (SYS_CLK),
 .RESET_IN_N       (RESET_IN_N),
 .WAIT_200US_ROUT    (WAIT_200US),
 .DELAY_SEL_VAL1_VAL (DELAY_SEL_VAL),
 .SYS_RST_VAL        (SYS_RST),
 .SYS_RST90_VAL      (SYS_RST90),
 .CLK_INT_VAL        (CLK0),
 .CLK90_INT_VAL      (CLK90),
 .SYS_RST180_VAL     (SYS_RST180)
 );

// INSTANTIATE THE LOOPBACK CONTROLLER
LOOPBACK_CONTROL LBC
 (
 .WB_CLK_I(CLK0),
 .WB_RST_I(SYS_RST),
 .M_WB_ADR_O(M2_WB_ADR_I),
 .M_WB_SEL_O(M2_WB_SEL_I),
 .M_WB_WE_O(M2_WB_WE_I),
 .M_WB_DAT_I(M2_WB_DAT_O),
 .M_WB_DAT_O(M2_WB_DAT_I),
 .M_WB_CYC_O(M2_WB_CYC_I),
 .M_WB_STB_O(M2_WB_STB_I),
 .M_WB_ACK_I(M2_WB_ACK_O),
 .M_WB_ERR_I(M2_WB_ERR_O)
 ,.LED(LED[7:0])
 );

// INSTANTIATE THE ETHCOP THAT INTERCONNECTS THE 4 WB INTERFACES
ETH_COP ECOP
 (
 // WISHBONE COMMON
 .WB_CLK_I(CLK0),
 .WB_RST_I(SYS_RST),

 // WISHBONE MASTER 1
 .M1_WB_ADR_I(M1_WB_ADR_I),
 .M1_WB_SEL_I(M1_WB_SEL_I),
 .M1_WB_WE_I(M1_WB_WE_I),
 .M1_WB_DAT_O(M1_WB_DAT_O),
 .M1_WB_DAT_I(M1_WB_DAT_I),
.M1_WB_CYC_I(M1_WB_CYC_I),
.M1_WB_STB_I(M1_WB_STB_I),
.M1_WB_ACK_O(M1_WB_ACK_O),
.M1_WB_ERR_O(M1_WB_ERR_O),

// WISHBONE MASTER 2
.M2_WB_ADR_I(M2_WB_ADR_I),
.M2_WB_SEL_I(M2_WB_SEL_I),
.M2_WB_WE_I(M2_WB_WE_I),
.M2_WB_DAT_O(M2_WB_DAT_O),
.M2_WB_DAT_I(M2_WB_DAT_I),
.M2_WB_CYC_I(M2_WB_CYC_I),
.M2_WB_STB_I(M2_WB_STB_I),
.M2_WB_ACK_O(M2_WB_ACK_O),
.M2_WB_ERR_O(M2_WB_ERR_O),

// WISHBONE SLAVE 1
.S1_WB_ADR_O(S1_WB_ADR_O),
.S1_WB_SEL_O(S1_WB_SEL_O),
.S1_WB_WE_O(S1_WB_WE_O),
.S1_WB_CYC_O(S1_WB_CYC_O),
.S1_WB_STB_O(S1_WB_STB_O),
.S1_WB_ACK_I(S1_WB_ACK_I),
.S1_WB_ERR_I(S1_WB_ERR_I),
.S1_WB_DAT_I(S1_WB_DAT_I),
.S1_WB_DAT_O(S1_WB_DAT_O),

// WISHBONE SLAVE 2
.S2_WB_ADR_O(S2_WB_ADR_O),
.S2_WB_SEL_O(S2_WB_SEL_O),
.S2_WB_WE_O(S2_WB_WE_O),
.S2_WB_CYC_O(S2_WB_CYC_O),
.S2_WB_STB_O(S2_WB_STB_O),
.S2_WB_ACK_I(S2_WB_ACK_I),
.S2_WB_ERR_I(S2_WB_ERR_I),
.S2_WB_DAT_I(S2_WB_DAT_I),
.S2_WB_DAT_O(S2_WB_DAT_O)
```

```
  );

ENDMODULE // ETH_LOOPBACK
```

C.4. memory_wb_to_mig

```
module memory_wb_to_mig
 (
  // wishbone interface:
  input          wb_clk_i,
  input          wb_rst_i,
  input  [31:0]    wb_adr_i,
  input  [31:0]    wb_dat_i,
  input   [3:0]    wb_sel_i,
  input          wb_we_i,
  input          wb_cyc_i,
  input          wb_stb_i,
  output reg       wb_ack_o,
  output reg       wb_err_o,
  output reg [31:0] wb_dat_o,

  // MIG 2.0 interface:
  input          mig_clk90,
  input          mig_init_done,
  input      [31:0] mig_output_data,
  output     [31:0] mig_input_data,
  output     [25:0] mig_input_address,
  output reg [2:0]  mig_command,
  input          mig_cmd_ack,
  input          mig_data_valid,
  output reg       mig_burst_done,
  input          mig_ar_done,
  input          mig_auto_ref_req,
  input          mig_wait_200us,
  output     [3:0]  mig_data_mask
  );
```

```
reg [2:0]         current_init_state;
reg [2:0]         next_init_state;
reg [2:0]         current_state;
reg [2:0]         next_state;
reg [23:0]         cntr;

// user commands:
localparam
  NOP_CMD        = 3'b000,
  PRECHARGE_CMD  = 3'b001,
  INIT_CMD       = 3'b010,
  WRITE_CMD      = 3'b100,
  READ_CMD       = 3'b110;

localparam
  INIT_INIT  = 3'b000,
  WAIT_200us = 3'b001,
  SEND_INIT  = 3'b010,
  WAIT_INIT  = 3'b011,
  INIT_DONE  = 3'b100;

parameter
  IDLE_ST           = 3'b000,
  READ_START_ST     = 3'b001,
  WRITE_START_ST    = 3'b010,
  WAIT_ACK_ST       = 3'b011,
  COMMAND_ACKED_ST  = 3'b100,
  BURST_DONE_ST     = 3'b101,
  WAIT_ACK_N_ST     = 3'b110,
  ACK_N_ST          = 3'b111;


localparam WAIT_BURST_DONE_VALUE = 24'b11;

reg              wb_ack_o_sync;
```

```
assign mig_input_address = {wb_adr_i[20:9],wb_adr_i[8], 1'b0, wb_adr_i[7:0], wb_adr_i[22:21]};
assign mig_data_mask     = ~wb_sel_i; // see ethernet module
assign mig_input_data    = wb_dat_i;


initial begin
  wb_ack_o <= 0;
  wb_ack_o_sync <= 0;
  wb_err_o <= 0;
  cntr <= 0;
  mig_command <= NOP_CMD;
  current_init_state <= 0;
  next_init_state <= 0;
  current_state <= 0;
  next_state <= 0;
end


// process the command
always @ (negedge mig_clk90) begin
  if (current_init_state == SEND_INIT) begin
    mig_command <= INIT_CMD;
  end else if (current_state == READ_START_ST) begin
    mig_command <= READ_CMD;
  end else if (current_state == WRITE_START_ST) begin
    mig_command <= WRITE_CMD;
  end else if (current_init_state == WAIT_INIT ||
             current_state == BURST_DONE_ST) begin
    mig_command <= NOP_CMD;
  end
end


// general counter
always @ (negedge mig_clk90) begin
  if (current_state == WAIT_ACK_ST) begin
    cntr <= WAIT_BURST_DONE_VALUE;
  end else begin
    cntr <= cntr - 1;
  end
end



// set burst_done output to memory
always @ (negedge mig_clk90) begin
  // should be set 3 clocks after a READ or WRITE command has been issued:
  if (current_state == BURST_DONE_ST)
    mig_burst_done <= 1'b1;
  else
    mig_burst_done <= 1'b0;
end



// memory output data to the wishbone interface:
always @ (negedge mig_clk90) begin
  if (mig_data_valid) begin
    case (wb_adr_i[1:0])
      2'b00:      // word access
        wb_dat_o <= mig_output_data;

      2'b10:      // half access
        wb_dat_o <= {16'b0,mig_output_data[15:0]};

      2'b01:      // byte access
        wb_dat_o <= {8'b0,mig_output_data[23:16],16'b0};

      2'b11:
        wb_dat_o <= {14'b0,mig_output_data[7:0]};
    endcase
  end
end



// state processing
```

```
always@ (negedge mig_clk90) begin
  if (wb_rst_i) begin
    current_init_state <= INIT_INIT;
  end else begin
    current_init_state <= next_init_state;
  end
end

always@ (negedge mig_clk90) begin
  if (wb_rst_i) begin
    current_state <= IDLE_ST;
  end else begin
    current_state <= next_state;
  end
end


// synchronizing to the wishbone clock
always @ (negedge mig_clk90) begin
  if (((current_state == WAIT_ACK_N_ST) && !mig_cmd_ack) || current_state == ACK_N_ST) begin
    wb_ack_o_sync <= 1'b1;
  end else begin
    wb_ack_o_sync <= 1'b0;
  end
end


// this is the ack output to the wb interface clocked by the wishbone
// clock
always @ (posedge wb_clk_i) begin
  if (wb_ack_o_sync) begin
    wb_ack_o <= 1'b1;
  end else begin
    wb_ack_o <= 1'b0;
  end
end
```

```
// memory initialization:
always @ (*) begin
  if (wb_rst_i) begin
    next_init_state = INIT_INIT;
  end else begin
    case (current_init_state)
      INIT_INIT:
        if (!mig_wait_200us)
          next_init_state = SEND_INIT;
        else
          next_init_state = INIT_INIT;

      SEND_INIT:
        next_init_state = WAIT_INIT;

      WAIT_INIT:
        if (mig_init_done)
          next_init_state = INIT_DONE;
        else
          next_init_state = WAIT_INIT;

      INIT_DONE:
        next_init_state = INIT_DONE;

      default
        next_init_state = INIT_DONE;
    endcase // case (current_init_state)
  end // else: !if(wb_rst_i)
end // always @ (*)


// state machine after memory initialization
always @ (*) begin
  if (wb_rst_i) begin
    next_state = IDLE_ST;
  end else if (mig_init_done) begin
```

```
case (current_state)
 IDLE_ST:
  if (wb_cyc_i & wb_stb_i) begin
   if (!wb_we_i) begin
    next_state = READ_START_ST;
   end else begin
    next_state = WRITE_START_ST;
   end
  end else begin
   next_state = IDLE_ST;
  end

 READ_START_ST:
  next_state = WAIT_ACK_ST;

 WRITE_START_ST:
  next_state = WAIT_ACK_ST;

 WAIT_ACK_ST:
  if (mig_cmd_ack)
   next_state = COMMAND_ACKED_ST;
  else
   next_state = WAIT_ACK_ST;

 COMMAND_ACKED_ST:
  if (!cntr) begin
   next_state = BURST_DONE_ST;
  end else begin
   next_state = COMMAND_ACKED_ST;
  end

 BURST_DONE_ST:
  next_state = WAIT_ACK_N_ST;

 WAIT_ACK_N_ST:
  if (!mig_cmd_ack) begin
   next_state = ACK_N_ST;
```

```
        end else
          next_state = WAIT_ACK_N_ST;

      ACK_N_ST:
        next_state = IDLE_ST;

      default:
        next_state = IDLE_ST;
    endcase // case (current_state)
  end else begin// if (mig_init_done)
    next_state = IDLE_ST;
  end // else: !if(init_done)
 end

endmodule
```

C.5. module eth_cop

```
`include "../rtl/eth_defines.v"
`include "../rtl/timescale.v"

module eth_cop
 (
  // WISHBONE common
  wb_clk_i, wb_rst_i,

  // WISHBONE MASTER 1
  m1_wb_adr_i, m1_wb_sel_i, m1_wb_we_i,  m1_wb_dat_o,
  m1_wb_dat_i, m1_wb_cyc_i, m1_wb_stb_i, m1_wb_ack_o,
  m1_wb_err_o,

  // WISHBONE MASTER 2
  m2_wb_adr_i, m2_wb_sel_i, m2_wb_we_i,  m2_wb_dat_o,
  m2_wb_dat_i, m2_wb_cyc_i, m2_wb_stb_i, m2_wb_ack_o,
  m2_wb_err_o,
 // WISHBONE slave 1
 s1_wb_adr_o, s1_wb_sel_o, s1_wb_we_o,  s1_wb_cyc_o,
 s1_wb_stb_o, s1_wb_ack_i, s1_wb_err_i, s1_wb_dat_i,
 s1_wb_dat_o,

 // WISHBONE slave 2
 s2_wb_adr_o, s2_wb_sel_o, s2_wb_we_o,  s2_wb_cyc_o,
 s2_wb_stb_o, s2_wb_ack_i, s2_wb_err_i, s2_wb_dat_i,
 s2_wb_dat_o
 );

parameter Tp=1;

// WISHBONE common
input wb_clk_i, wb_rst_i;

// WISHBONE MASTER 1
input  [31:0] m1_wb_adr_i, m1_wb_dat_i;
input   [3:0] m1_wb_sel_i;
input         m1_wb_cyc_i, m1_wb_stb_i, m1_wb_we_i;
output [31:0] m1_wb_dat_o;
output        m1_wb_ack_o, m1_wb_err_o;

// WISHBONE MASTER 2
input  [31:0] m2_wb_adr_i, m2_wb_dat_i;
input   [3:0] m2_wb_sel_i;
input         m2_wb_cyc_i, m2_wb_stb_i, m2_wb_we_i;
output [31:0] m2_wb_dat_o;
output        m2_wb_ack_o, m2_wb_err_o;

// WISHBONE slave 1
input  [31:0] s1_wb_dat_i;
input         s1_wb_ack_i, s1_wb_err_i;
output [31:0] s1_wb_adr_o, s1_wb_dat_o;
output  [3:0] s1_wb_sel_o;
output        s1_wb_we_o,  s1_wb_cyc_o, s1_wb_stb_o;
```

```
 // WISHBONE slave 2
 input  [31:0] s2_wb_dat_i;
 input         s2_wb_ack_i, s2_wb_err_i;
 output [31:0] s2_wb_adr_o, s2_wb_dat_o;
 output  [3:0] s2_wb_sel_o;
 output        s2_wb_we_o,  s2_wb_cyc_o, s2_wb_stb_o;

 reg           m1_in_progress;
 reg           m2_in_progress;
 reg [31:0]    s1_wb_adr_o;
 reg [3:0]     s1_wb_sel_o;
 reg           s1_wb_we_o;
 reg [31:0]    s1_wb_dat_o;
 reg           s1_wb_cyc_o;
 reg           s1_wb_stb_o;
 reg [31:0]    s2_wb_adr_o;
 reg [3:0]     s2_wb_sel_o;
 reg           s2_wb_we_o;
 reg [31:0]    s2_wb_dat_o;
 reg           s2_wb_cyc_o;
 reg           s2_wb_stb_o;

 reg           m1_wb_ack_o;
 reg [31:0]    m1_wb_dat_o;
 reg           m2_wb_ack_o;
 reg [31:0]    m2_wb_dat_o;

 reg           m1_wb_err_o;
 reg           m2_wb_err_o;

 /*
 wire          M1_ADDRESSED_S1 = ( (m1_wb_adr_i >= `ETH_BASE)    & (m1_wb_adr_i < (`ETH_BASE    +
`ETH_WIDTH   )) );
 wire          M1_ADDRESSED_S2 = ( (m1_wb_adr_i >= `ETH_MEMORY_BASE) & (m1_wb_adr_i <
(`ETH_MEMORY_BASE + `ETH_MEMORY_WIDTH)) );
 wire          M2_ADDRESSED_S1 = ( (m2_wb_adr_i >= `ETH_BASE)    & (m2_wb_adr_i < (`ETH_BASE    +
`ETH_WIDTH   )) );
```

```
 wire        M2_ADDRESSED_S2 = ( (m2_wb_adr_i >= `ETH_MEMORY_BASE) & (m2_wb_adr_i <
(`ETH_MEMORY_BASE + `ETH_MEMORY_WIDTH)) );
 */
 wire        M1_ADDRESSED_S1 = ( (m1_wb_adr_i >= 32'h0)    & (m1_wb_adr_i < 32'h800) );
 wire        M1_ADDRESSED_S2 = ( (m1_wb_adr_i >= 32'h2000) & (m1_wb_adr_i < 32'h32000) );
 wire        M2_ADDRESSED_S1 = ( (m2_wb_adr_i >= 32'h0)    & (m2_wb_adr_i < 32'h800) );
 wire        M2_ADDRESSED_S2 = ( (m2_wb_adr_i >= 32'h2000) & (m2_wb_adr_i < 32'h32000) );

 wire        m_wb_access_finished;
 wire        m1_req = m1_wb_cyc_i & m1_wb_stb_i & (M1_ADDRESSED_S1 | M1_ADDRESSED_S2);
 wire        m2_req = m2_wb_cyc_i & m2_wb_stb_i & (M2_ADDRESSED_S1 | M2_ADDRESSED_S2);


 initial begin
   m1_in_progress <= 0;
   m2_in_progress <= 0;
   s1_wb_adr_o    <= 0;
   s1_wb_sel_o    <= 0;
   s1_wb_we_o     <= 0;
   s1_wb_dat_o    <= 0;
   s1_wb_cyc_o    <= 0;
   s1_wb_stb_o    <= 0;
   s2_wb_adr_o    <= 0;
   s2_wb_sel_o    <= 0;
   s2_wb_we_o     <= 0;
   s2_wb_dat_o    <= 0;
   s2_wb_cyc_o    <= 0;
   s2_wb_stb_o    <= 0;
 end


 always @ (posedge wb_clk_i or posedge wb_rst_i)
   begin
     if(wb_rst_i)
       begin
         m1_in_progress <=#Tp 0;
         m2_in_progress <=#Tp 0;
```

```
        s1_wb_adr_o    <=#Tp 0;
        s1_wb_sel_o    <=#Tp 0;
        s1_wb_we_o     <=#Tp 0;
        s1_wb_dat_o    <=#Tp 0;
        s1_wb_cyc_o    <=#Tp 0;
        s1_wb_stb_o    <=#Tp 0;
        s2_wb_adr_o    <=#Tp 0;
        s2_wb_sel_o    <=#Tp 0;
        s2_wb_we_o     <=#Tp 0;
        s2_wb_dat_o    <=#Tp 0;
        s2_wb_cyc_o    <=#Tp 0;
        s2_wb_stb_o    <=#Tp 0;
      end
    else
      begin
        case({m1_in_progress, m2_in_progress, m1_req, m2_req, m_wb_access_finished})  // synopsys_full_case
synopsys_paralel_case
          5'b00_10_0, 5'b00_11_0 :
            begin
              m1_in_progress <=#Tp 1'b1;  // idle: m1 or (m1 & m2) want access: m1 -> m
              if(M1_ADDRESSED_S1)
                begin
                  s1_wb_adr_o <=#Tp m1_wb_adr_i;
                  s1_wb_sel_o <=#Tp m1_wb_sel_i;
                  s1_wb_we_o  <=#Tp m1_wb_we_i;
                  s1_wb_dat_o <=#Tp m1_wb_dat_i;
                  s1_wb_cyc_o <=#Tp 1'b1;
                  s1_wb_stb_o <=#Tp 1'b1;
                end
              else if(M1_ADDRESSED_S2)
                begin
                  s2_wb_adr_o <=#Tp m1_wb_adr_i;
                  s2_wb_sel_o <=#Tp m1_wb_sel_i;
                  s2_wb_we_o  <=#Tp m1_wb_we_i;
                  s2_wb_dat_o <=#Tp m1_wb_dat_i;
                  s2_wb_cyc_o <=#Tp 1'b1;
                  s2_wb_stb_o <=#Tp 1'b1;
                end
              //else
              //$display("(%t)(%m)WISHBONE ERROR: Unspecified address space accessed", $time);
            end
          5'b00_01_0 :
            begin
              m2_in_progress <=#Tp 1'b1;  // idle: m2 wants access: m2 -> m
              if(M2_ADDRESSED_S1)
                begin
                  s1_wb_adr_o <=#Tp m2_wb_adr_i;
                  s1_wb_sel_o <=#Tp m2_wb_sel_i;
                  s1_wb_we_o  <=#Tp m2_wb_we_i;
                  s1_wb_dat_o <=#Tp m2_wb_dat_i;
                  s1_wb_cyc_o <=#Tp 1'b1;
                  s1_wb_stb_o <=#Tp 1'b1;
                end
              else if(M2_ADDRESSED_S2)
                begin
                  s2_wb_adr_o <=#Tp m2_wb_adr_i;
                  s2_wb_sel_o <=#Tp m2_wb_sel_i;
                  s2_wb_we_o  <=#Tp m2_wb_we_i;
                  s2_wb_dat_o <=#Tp m2_wb_dat_i;
                  s2_wb_cyc_o <=#Tp 1'b1;
                  s2_wb_stb_o <=#Tp 1'b1;
                end
              //else
              //$display("(%t)(%m)WISHBONE ERROR: Unspecified address space accessed", $time);
            end
          5'b10_10_1, 5'b10_11_1 :
            begin
              m1_in_progress <=#Tp 1'b0;  // m1 in progress. Cycle is finished. Send ack or err to m1.
              if(M1_ADDRESSED_S1)
                begin
                  s1_wb_cyc_o <=#Tp 1'b0;
                  s1_wb_stb_o <=#Tp 1'b0;
                end
              else if(M1_ADDRESSED_S2)
```

```
            begin
              s2_wb_cyc_o <=#Tp 1'b0;
              s2_wb_stb_o <=#Tp 1'b0;
            end
          end
        5'b01_01_1, 5'b01_11_1 :
          begin
            m2_in_progress <=#Tp 1'b0;  // m2 in progress. Cycle is finished. Send ack or err to m2.
            if(M2_ADDRESSED_S1)
              begin
                s1_wb_cyc_o <=#Tp 1'b0;
                s1_wb_stb_o <=#Tp 1'b0;
              end
            else if(M2_ADDRESSED_S2)
              begin
                s2_wb_cyc_o <=#Tp 1'b0;
                s2_wb_stb_o <=#Tp 1'b0;
              end
          end
      endcase
    end
  end

// Generating Ack for master 1
always @ (*)
  begin
    if(m1_in_progress)
      begin
        if(M1_ADDRESSED_S1) begin
          m1_wb_ack_o <= s1_wb_ack_i;
          m1_wb_dat_o <= s1_wb_dat_i;
        end
        else if(M1_ADDRESSED_S2) begin
          m1_wb_ack_o <= s2_wb_ack_i;
          m1_wb_dat_o <= s2_wb_dat_i;
        end
      end
    else
      m1_wb_ack_o <= 0;
  end


 // Generating Ack for master 2
 always @ (*)
  begin
    if(m2_in_progress)
      begin
        if(M2_ADDRESSED_S1) begin
          m2_wb_ack_o <= s1_wb_ack_i;
          m2_wb_dat_o <= s1_wb_dat_i;
        end
        else if(M2_ADDRESSED_S2) begin
          m2_wb_ack_o <= s2_wb_ack_i;
          m2_wb_dat_o <= s2_wb_dat_i;
        end
      end
    else
      m2_wb_ack_o <= 0;
  end


 // Generating Err for master 1
 // sensitivity list change lll
 //always @ (m1_in_progress or m1_wb_adr_i or s1_wb_err_i or s2_wb_err_i or M2_ADDRESSED_S1 or
M2_ADDRESSED_S2 or
 always @ (*)
  begin
    if(m1_in_progress)  begin
      if(M1_ADDRESSED_S1)
        m1_wb_err_o <= s1_wb_err_i;
      else if(M1_ADDRESSED_S2)
        m1_wb_err_o <= s2_wb_err_i;
    end
    else if(m1_wb_cyc_i & m1_wb_stb_i & ~M1_ADDRESSED_S1 & ~M1_ADDRESSED_S2)
```

```
      m1_wb_err_o <= 1'b1;
    else
      m1_wb_err_o <= 1'b0;
  end


// Generating Err for master 2
always @ (*)
  begin
    if(m2_in_progress)  begin
      if(M2_ADDRESSED_S1)
        m2_wb_err_o <= s1_wb_err_i;
      else if(M2_ADDRESSED_S2)
        m2_wb_err_o <= s2_wb_err_i;
    end
    else if(m2_wb_cyc_i & m2_wb_stb_i & ~M2_ADDRESSED_S1 & ~M2_ADDRESSED_S2)
      m2_wb_err_o <= 1'b1;
    else
      m2_wb_err_o <= 1'b0;
  end


assign m_wb_access_finished = m1_wb_ack_o | m1_wb_err_o | m2_wb_ack_o | m2_wb_err_o;


// Activity monitor
/* lll
 integer cnt;
 always @ (posedge wb_clk_i or posedge wb_rst_i)
 begin
 if(wb_rst_i)
 cnt <=#Tp 0;
 else
 if(s1_wb_ack_i | s1_wb_err_i | s2_wb_ack_i | s2_wb_err_i)
 cnt <=#Tp 0;
 else
 if(s1_wb_cyc_o | s2_wb_cyc_o)
```

```
   cnt <=#Tp cnt+1;
end

   always @ (posedge wb_clk_i)
   begin
   if(cnt==1000) begin
   $display("(%0t)(%m) ERROR: WB activity ??? ", $time);
   if(s1_wb_cyc_o) begin
   $display("s1_wb_dat_o = 0x%0x", s1_wb_dat_o);
   $display("s1_wb_adr_o = 0x%0x", s1_wb_adr_o);
   $display("s1_wb_sel_o = 0x%0x", s1_wb_sel_o);
   $display("s1_wb_we_o = 0x%0x", s1_wb_we_o);
    end
   else if(s2_wb_cyc_o) begin
   $display("s2_wb_dat_o = 0x%0x", s2_wb_dat_o);
   $display("s2_wb_adr_o = 0x%0x", s2_wb_adr_o);
   $display("s2_wb_sel_o = 0x%0x", s2_wb_sel_o);
   $display("s2_wb_we_o = 0x%0x", s2_wb_we_o);
    end

   $stop;
  end
end


   always @ (posedge wb_clk_i)
   begin
   if(s1_wb_err_i & s1_wb_cyc_o) begin
   $display("(%0t) ERROR: WB cycle finished with error acknowledge ", $time);
   $display("s1_wb_dat_o = 0x%0x", s1_wb_dat_o);
   $display("s1_wb_adr_o = 0x%0x", s1_wb_adr_o);
   $display("s1_wb_sel_o = 0x%0x", s1_wb_sel_o);
   $display("s1_wb_we_o = 0x%0x", s1_wb_we_o);
   $stop;
  end
   if(s2_wb_err_i & s2_wb_cyc_o) begin
   $display("(%0t) ERROR: WB cycle finished with error acknowledge ", $time);
   $display("s2_wb_dat_o = 0x%0x", s2_wb_dat_o);
   $display("s2_wb_adr_o = 0x%0x", s2_wb_adr_o);
   $display("s2_wb_sel_o = 0x%0x", s2_wb_sel_o);
   $display("s2_wb_we_o = 0x%0x", s2_wb_we_o);
   $stop;
  end
end

   */

endmodule
```

C.6. eth_defines.v

```
`define ETH_MODER_ADR         8'h0    // 0x0
`define ETH_INT_SOURCE_ADR    8'h1    // 0x4
`define ETH_INT_MASK_ADR      8'h2    // 0x8
`define ETH_IPGT_ADR          8'h3    // 0xC
`define ETH_IPGR1_ADR         8'h4    // 0x10
`define ETH_IPGR2_ADR         8'h5    // 0x14
`define ETH_PACKETLEN_ADR     8'h6    // 0x18
`define ETH_COLLCONF_ADR      8'h7    // 0x1C
`define ETH_TX_BD_NUM_ADR     8'h8    // 0x20
`define ETH_CTRLMODER_ADR     8'h9    // 0x24
`define ETH_MIIMODER_ADR      8'hA    // 0x28
`define ETH_MIICOMMAND_ADR    8'hB    // 0x2C
`define ETH_MIIADDRESS_ADR    8'hC    // 0x30
`define ETH_MIITX_DATA_ADR    8'hD    // 0x34
`define ETH_MIIRX_DATA_ADR    8'hE    // 0x38
`define ETH_MIISTATUS_ADR     8'hF    // 0x3C
`define ETH_MAC_ADDR0_ADR     8'h10   // 0x40
`define ETH_MAC_ADDR1_ADR     8'h11   // 0x44
`define ETH_HASH0_ADR         8'h12   // 0x48
`define ETH_HASH1_ADR         8'h13   // 0x4C
`define ETH_TX_CTRL_ADR       8'h14   // 0x50
`define ETH_RX_CTRL_ADR       8'h15   // 0x54
```

```
`define ETH_MODER_DEF_0        8'h00
`define ETH_MODER_DEF_1        8'hA0
`define ETH_MODER_DEF_2        1'h0
`define ETH_INT_MASK_DEF_0     7'h0
`define ETH_IPGT_DEF_0         7'h12
`define ETH_IPGR1_DEF_0        7'h0C
`define ETH_IPGR2_DEF_0        7'h12
`define ETH_PACKETLEN_DEF_0    8'h00
`define ETH_PACKETLEN_DEF_1    8'h06
`define ETH_PACKETLEN_DEF_2    8'h40
`define ETH_PACKETLEN_DEF_3    8'h00
`define ETH_COLLCONF_DEF_0     6'h3f
`define ETH_COLLCONF_DEF_2     4'hF
`define ETH_TX_BD_NUM_DEF_0    8'h40
`define ETH_CTRLMODER_DEF_0    3'h0
`define ETH_MIIMODER_DEF_0     8'h64
`define ETH_MIIMODER_DEF_1     1'h0
`define ETH_MIIADDRESS_DEF_0   5'h00
`define ETH_MIIADDRESS_DEF_1   5'h00
`define ETH_MIITX_DATA_DEF_0   8'h00
`define ETH_MIITX_DATA_DEF_1   8'h00
`define ETH_MIIRX_DATA_DEF     16'h0000 // not written from WB
`define ETH_MAC_ADDR0_DEF_0    8'h00
`define ETH_MAC_ADDR0_DEF_1    8'h00
`define ETH_MAC_ADDR0_DEF_2    8'h00
`define ETH_MAC_ADDR0_DEF_3    8'h00
`define ETH_MAC_ADDR1_DEF_0    8'h00
`define ETH_MAC_ADDR1_DEF_1    8'h00
`define ETH_HASH0_DEF_0        8'h00
`define ETH_HASH0_DEF_1        8'h00
`define ETH_HASH0_DEF_2        8'h00
`define ETH_HASH0_DEF_3        8'h00
`define ETH_HASH1_DEF_0        8'h00
`define ETH_HASH1_DEF_1        8'h00
`define ETH_HASH1_DEF_2        8'h00
```

```
`define ETH_HASH1_DEF_3        8'h00
`define ETH_TX_CTRL_DEF_0      8'h00
`define ETH_TX_CTRL_DEF_1      8'h00
`define ETH_TX_CTRL_DEF_2      1'h0
`define ETH_RX_CTRL_DEF_0      8'h00
`define ETH_RX_CTRL_DEF_1      8'h00


`define ETH_MODER_WIDTH_0      8
`define ETH_MODER_WIDTH_1      8
`define ETH_MODER_WIDTH_2      1
`define ETH_INT_SOURCE_WIDTH_0  7
`define ETH_INT_MASK_WIDTH_0    7
`define ETH_IPGT_WIDTH_0       7
`define ETH_IPGR1_WIDTH_0      7
`define ETH_IPGR2_WIDTH_0      7
`define ETH_PACKETLEN_WIDTH_0   8
`define ETH_PACKETLEN_WIDTH_1   8
`define ETH_PACKETLEN_WIDTH_2   8
`define ETH_PACKETLEN_WIDTH_3   8
`define ETH_COLLCONF_WIDTH_0    6
`define ETH_COLLCONF_WIDTH_2    4
`define ETH_TX_BD_NUM_WIDTH_0   8
`define ETH_CTRLMODER_WIDTH_0   3
`define ETH_MIIMODER_WIDTH_0    8
`define ETH_MIIMODER_WIDTH_1    1
`define ETH_MIICOMMAND_WIDTH_0  3
`define ETH_MIIADDRESS_WIDTH_0  5
`define ETH_MIIADDRESS_WIDTH_1  5
`define ETH_MIITX_DATA_WIDTH_0  8
`define ETH_MIITX_DATA_WIDTH_1  8
`define ETH_MIIRX_DATA_WIDTH    16 // not written from WB
`define ETH_MIISTATUS_WIDTH     3 // not written from WB
`define ETH_MAC_ADDR0_WIDTH_0   8
`define ETH_MAC_ADDR0_WIDTH_1   8
`define ETH_MAC_ADDR0_WIDTH_2   8
`define ETH_MAC_ADDR0_WIDTH_3   8
```

```
`define ETH_MAC_ADDR1_WIDTH_0   8
`define ETH_MAC_ADDR1_WIDTH_1   8
`define ETH_HASH0_WIDTH_0       8
`define ETH_HASH0_WIDTH_1       8
`define ETH_HASH0_WIDTH_2       8
`define ETH_HASH0_WIDTH_3       8
`define ETH_HASH1_WIDTH_0       8
`define ETH_HASH1_WIDTH_1       8
`define ETH_HASH1_WIDTH_2       8
`define ETH_HASH1_WIDTH_3       8
`define ETH_TX_CTRL_WIDTH_0     8
`define ETH_TX_CTRL_WIDTH_1     8
`define ETH_TX_CTRL_WIDTH_2     1
`define ETH_RX_CTRL_WIDTH_0     8
`define ETH_RX_CTRL_WIDTH_1     8


// Outputs are registered (uncomment when needed)
`define ETH_REGISTERED_OUTPUTS

// Settings for TX FIFO
`define ETH_TX_FIFO_CNT_WIDTH  5
`define ETH_TX_FIFO_DEPTH      16
`define ETH_TX_FIFO_DATA_WIDTH 32

// Settings for RX FIFO
`define ETH_RX_FIFO_CNT_WIDTH  5
`define ETH_RX_FIFO_DEPTH      16
`define ETH_RX_FIFO_DATA_WIDTH 32

// Burst length
`define ETH_BURST_LENGTH       4    // Change also ETH_BURST_CNT_WIDTH
`define ETH_BURST_CNT_WIDTH    3    // The counter must be width enough to count to
ETH_BURST_LENGTH

// WISHBONE interface is Revision B3 compliant (uncomment when needed)
//`define ETH_WISHBONE_B3
```

```
// Following defines are needed when eth_cop.v is used. Otherwise they may be deleted.
`define ETH_BASE            32'hd0000000
`define ETH_WIDTH           32'h800
`define ETH_MEMORY_BASE     32'h2000 // renamed for conflict
`define ETH_MEMORY_WIDTH    32'h30000// renamed for conflict
/*
`define MEMORY_BASE         32'h2000
`define MEMORY_WIDTH        32'h10000
*/
/* Apparently the ISE is not able to do the double translation:
`define M1_ADDRESSED_S1 ( (m1_wb_adr_i >= `ETH_BASE)    & (m1_wb_adr_i < (`ETH_BASE    + `ETH_WIDTH
)) )
`define M1_ADDRESSED_S2 ( (m1_wb_adr_i >= `MEMORY_BASE) & (m1_wb_adr_i < (`MEMORY_BASE +
`MEMORY_WIDTH)) )
`define M2_ADDRESSED_S1 ( (m2_wb_adr_i >= `ETH_BASE)    & (m2_wb_adr_i < (`ETH_BASE    + `ETH_WIDTH
)) )
`define M2_ADDRESSED_S2 ( (m2_wb_adr_i >= `MEMORY_BASE) & (m2_wb_adr_i < (`MEMORY_BASE +
`MEMORY_WIDTH)) )
*/
// Previous defines are only needed for eth_cop.v
```