Data Acquisition System Using FPGA Kit and Microblaze

By

Maliha Safdar, Muhammad Hamza, Amir Jehangir and Muhammad Saad Bin Afzal

Submitted to the Faculty of Department of Electrical Engineering,

Military College of Signals, National University of Science and Technology, Islamabad

in partial fulfillment for the requirement of a B.E Degree in

Telecom Engineering

JUNE 2017

ABSTRACT

Project Name:

Data Acquisition System Using FPGA.

Skills Required:

1. Embedded Systems Design.
2. C/Verilog Programming.

Description:

The system shall acquire data from the USB devices, convert the data to IP packet and send it to PCs over Ethernet network. Data from the USB devices are to be streamed continuously over LAN cable. All interactionsare controlled with the help of Micro blaze soft microprocessor core implemented in FPGA. Block diagram of the system is given below:

Project Deliverables:

1. Design of the Data Acquisition System.
2. Hardware implementation.
3. Source code.
4. Project report.

CERTIFICATE

This is to certify that this Thesis Report entitled **"Data Acquisition System using FPGA Kit and Microblaze"** by **Maliha Safdar, Muhammad Hamza, Amir Jehangir** and **Muhammad Saad Bin Afzal** is submitted in partial fulfillment of the requirement for the degree of BETE in Military College of Siganls (NUST) during the academic year 2016-2017, is a bona fide record work carried out under my guidance and supervision.

Name (Supervisor): **Asst. Prof. Mian Muhammad Waseem Iqbal**

Signature: _____

Date: _____

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of another award or qualification either at this institution or elsewhere

DEDICATIONS

We dedicate this to our family, friends and above all our amazing teachers, without their prayers, support, encouragement, guidance and appreciation we couldn't have achieved such a milestone.

ACKNOWLEDGEMENTS

Due extension of our humble gratitude to the most magnificent Allah Almighty, without His will it would never have been possible to have this attainment. We thank Him for providing us with all the knowledge, intelligence, sagacity and understanding which was needed for the successful completion of this thesis research work.

We thank our project supervisor Asst. Prof. Mian Muhammad Waseem Iqbal, who supported us whole heartedly and stimulated our intellect during our work. Without his interest, involvement and assistance it wouldn't have been possible to carry out the research and complete the project work.

We are also thankful to our mentors and colleagues for helping us in the development of this project and rendering us their support whenever it was needed.

Last but not the least, we are more than thankful to our parents for all the prayers, understanding and massive support. Their encouragement helped us a lot in achieving all our project tasks and their words kept us motivated and dedicated throughout our project work.

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 1

INTRODUCTION:

1.1 Overview:

Universal Serial Bus (USB) is technology is widely spreading all around the world. Many mobile peripherals, web cameras, keyboards and printers come with USB connection. It is because of the fact that this type of connection is very simple and have characteristics like play and plug features, hot plug support and most important of them is possibility of supplying power to peripheral devices. USB also offers good noise immunity due to its differential type signal transmission.USB provides high data transfer rate so it is best to use it for data acquisition and processing purposes. Data Acquisition System for Universal Serial Bus (USB) devices is basically a very fast communication system which acquires data from USB using micro-blaze processer of FPGA and transmit it on Ethernet at a speed of 1gb/s. This system can be introduced in any communication system just doing minor changes like introducing FPGA kit at receiver or transmitter end based on system requirement.

1.2 Problem Statement:

Telecommunication irrespective of mode is like oxygen for human beings of modern era where use of internet calling , data transfer and social networking is a basic necessity of everyone and not only that business, banking and defense all are dependent on it. In such situation where all of us are depending on internet, mobile and telephone speed is the basic issue for telecommunication companies because of following problems.

- Shared resources
- Low through put
- Less processing speed of router or PC as Compared to internet

- Unsecure data

On the contrary, Data Acquisition System Using FPGA has many advantages which include:

- High throughput
- High processing Speed
- Multiple task can be done at the same time(Parallel Processing)
- Data securing will not cause any delay
- Easier to maintain and ideal for large organizations

1.3 Proposed Model:

The proposed architecture model is shown in Figure 1. It will take data from multiple USB devices and after converting data into Ethernet packet format i-e IP packets throw it to PCs via LAN.Data from the USB will be streamed continuously over Cat-5 copper LAN cable at 1Gb/s. The system will provide access to all the devices that follow the USB 2.0 standard.

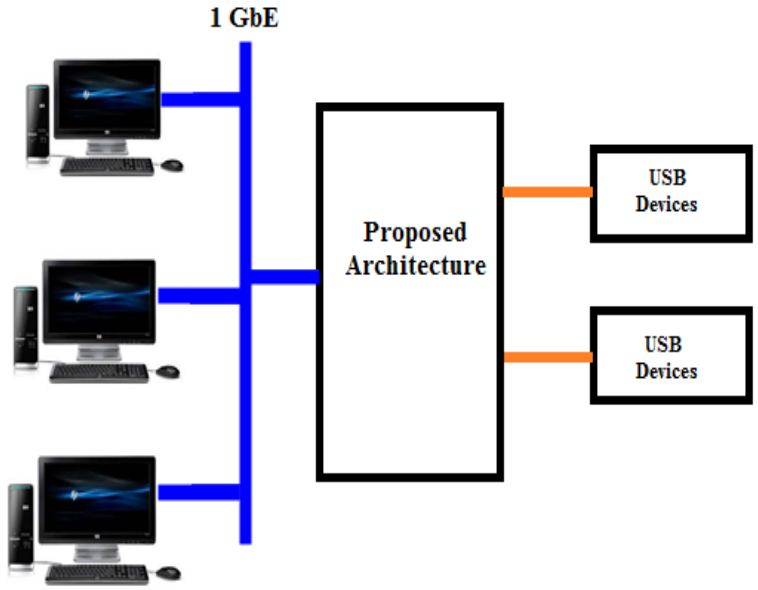


Figure 1: Proposed Model of Data Acquisition System

1.4Approach:

As mentioned above this system will gather data from USB devices at a speed of 483 Mb/s and sends it over Ethernet at a speed of 1 Gb/s. When from data is acquired from USB using microblaze processor it is converted into IP packets. These IP packets are then continuously streamed on Ethernet .While this processor data can displayed on screen using RS232 UART peripheral of FPGA also Data can be stored using FPGA internal or external memory

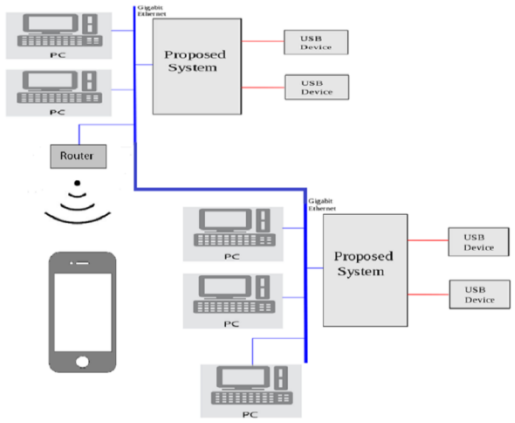


Figure 2: Data Acquisition System using FPGA implementation

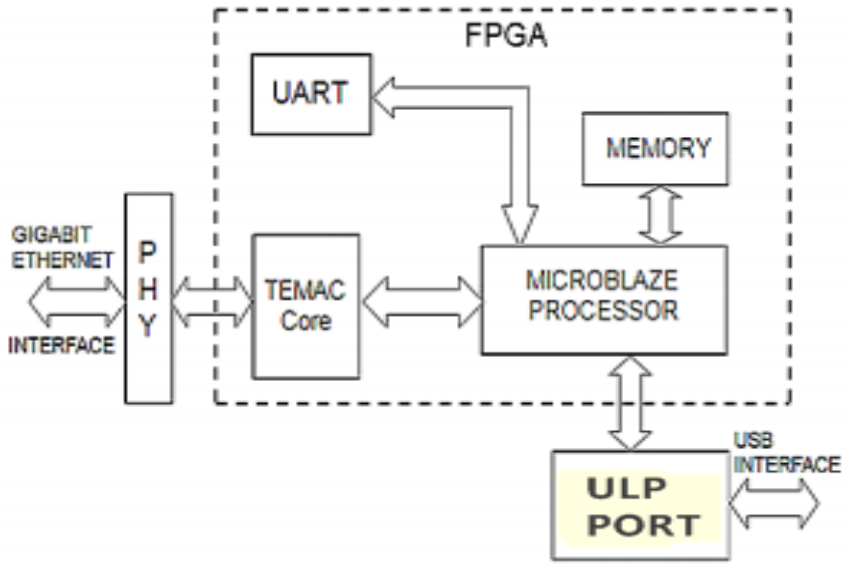


Figure 3: Data Acquisition System using FPGA schematic

1.5 Objective:

The basic aims behind this project are:

- To design a Data Acquisition system which will ensure very fast communication
- This particular project will intake data from any source having output connection in the form of USB
- The project will read data from Micro-blaze processor core and convert it into IP packets
- IP packets will transmit at a speed of 1 GB/s using Temac core
- Data will be visualize using UART peripheral
- Data will also be stored using FPGA internal or external memory

1.6 Additional Objective:

Other than basic aims we have one additional aim

- To secure IP packets using modern encryption techniques.

The main part of project will be built using Xilinx Platform Studio.A processor based embedded system with real-time operating system will be used to achieve high speed and TEMAC core will be used to transmit data on Ethernet. To secure IP packets Xilinx ISE suit will be used and Verilog Hardware Description Language will be used.

Chapter 2

BACKGROUND STUDY:

2.1 Existing Literature:

Data Acquisition is the procedure in which the physical values of signals are sampled and then these samples are converted it to digital values which can then be analyzed by the computer. Analogue waveforms are converted to digital numeric values and similarly digital waveforms can be converted to analog values which can then further be processed.

- Physical parameters to Analog Conversion: **Sensors**
- Conversion of sensor signals into a form that can be converted to digital values: **Signal Conditioning Circuits**
- Conversion of sensor signal to digital values: **Analog to Digital convertor**

2.1.2 Types of Data Acquisition System:

- Analog input/output signals
- Digital input/output signals
- Counters
- Other signals including serial and bus-based signals

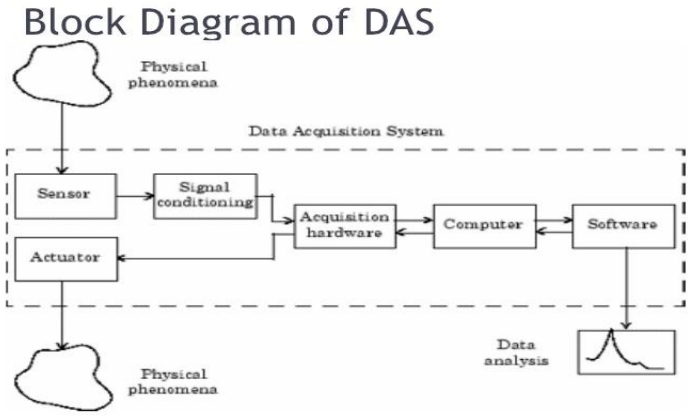


Figure 4: Block Diagram of Data Acquisition System

2.2 Data Acquisition System Using FPGA:

These type of system use Field Programmable Gate Array as a DAQ device as shown in fig.4. FPGA has many application as DAQ device because it has capability do multiple task at the same time (parallel processing).It has multiple processors and many internal or external peripherals which can be used in many ways. Also it has capability to store and display data. Its most important applications are

- Packet processing
- USB to Ethernet
- Ethernet to Ethernet

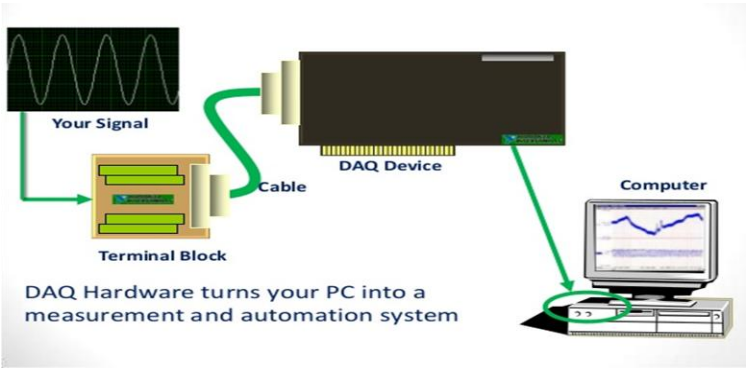


Figure 5: Data Acquisition System using FPGA

2.3 Problem Formulation:

Some work has already been done for the Data Acquisition System using FPGA for USB devices. Almost all DAQ's require FPGAs' because flexibility and architecture to design systems with fast processing speed and use of optimized silicon area but in our case it is necessary because we are designing a system which need fast processing . We are using Xilinx Virtex-7 Evaluation Board is most recent FPGA by Xilinx .It has 200MHz fixed

oscillator and up to 400MHz programmable clock generator. Furthermore, it has GTX transceiver, which can provide complete 1000 Base-X implementation on-chip. It is our aim to make a device that will provide fastest communication and the flexibility to provide a faster and efficient solution to secure any network from any malware

Chapter 3:

DESIGN SPECIFICATIONS:

3.1 Technical specifications:

We are using FPGA virtex-7(VC-707) Evaluation kit for our project.

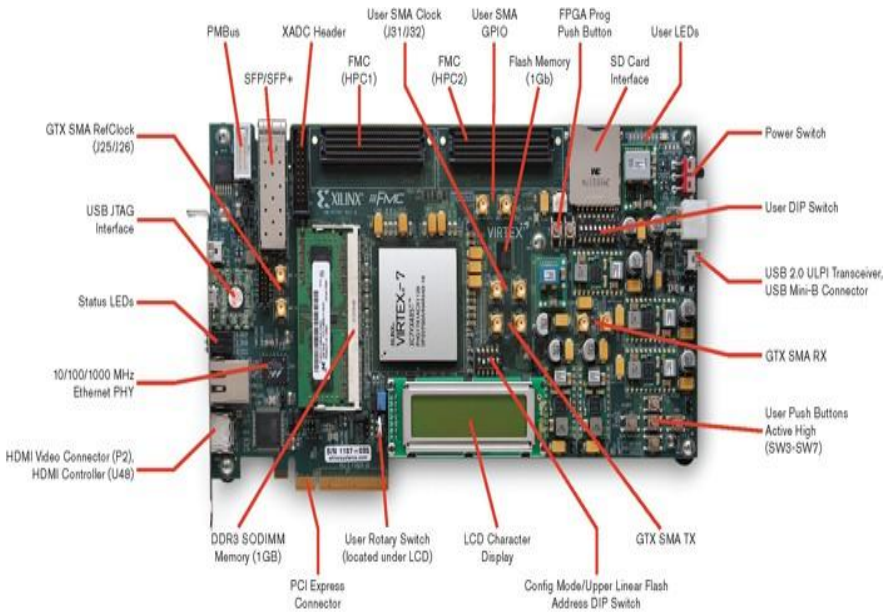


Figure 6: Virtex-7 Evaluation Kit

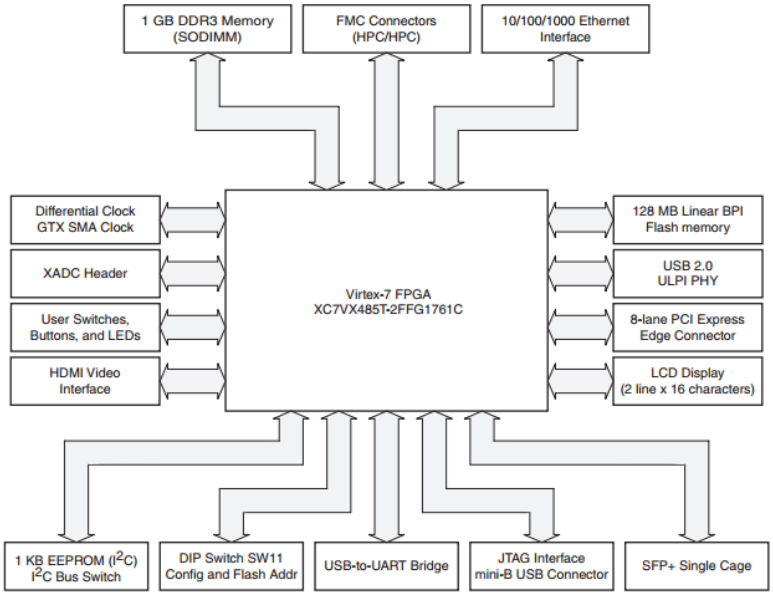


Figure 7: Virtex-7 Specifications

The Hardware Specifications of VC707 board are:

- 1 GBDDR3 memory SODIMM
- 128 MB Linear byte peripheral interface (BPI) Flash memory
- USB 2.0 ULPITransceiver
- Secure Digital (SD) connector
- USB JTAG through Digilent module
- Clock Generation
- Fixed 200MHz LVDS oscillator
- I2C programmable LVDS oscillator
- SMA connectors
- SMA connectors for GTX transceiver clocking
- GTX transceivers
- FMC1 HPC connector
- FMC2 HPC connector
- SMA connectors
- PCI Express
- Small form-factor pluggable plus (SFP+) connector
- Ethernet PHY SGMII interface
- PCI Express endpoint connectivity
- Gen1 8-lane (x8)
- Gen2 8-lane (x8)
- SFP+ Connector

- 10/100/1000 tri-speed Ethernet PHY
- USB-to-UART bridge
- HDMI™ codec
- I2C bus
- I2C MUX
- I2C EEPROM
- USER I2C programmable LVDS oscillator
- DDR3 SODIMM socket
- HDMI codec
- FMC1 HPC connector
- FMC2 HPC connector
- SFP+ connector
- I2Cprogrammable jitter-attenuating precision clock multiplier
- Status LEDs
- Ethernet status
- Power good
- FPGA INIT
- FPGA DONE
- User Input/Output
- User LEDsa
- User pushbuttons
- CPU reset pushbutton
- User DIP switch (8- GPIO)
- User SMA GPIO connectors
- LCD character display (16x2)
- Switches
- Power on/off slide switch
- FPGA_PROB_B pushbutton
- Configuration mode DIP switch
- VITA 57.1 FMC1 HPC Connector
- VITA 57.1 FMC2 HPC Connector
- Power management
- PM-Bus voltage and current monitoring through TI power controller
- XADC header
- Linear BPI Flash memory
- USB JTAG configuration port
- Platform cable header JTAG configuration port

3.2 Design Requirements:

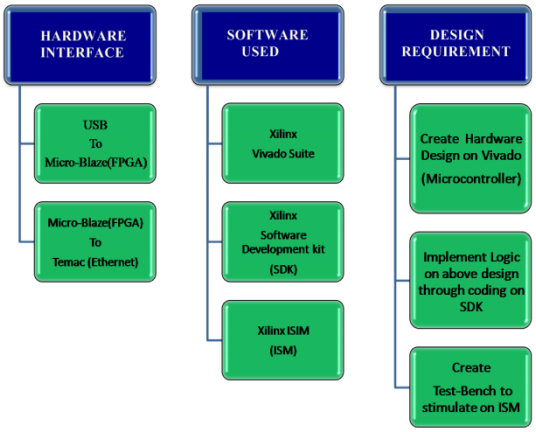


Figure 8: Requirement for Design

3.3 Design Specifications:

We will design the Project Using Xilinx Platform Studio

First of all we will add following peripheral in our project

- Micro-blaze processor
- DDR3-SD RAM
- RS-232 UART

We will then add following IP's to our project

- AXI-USB 2.0 device

- AXI -Ethernet TEMAC (1GB/s)

We will use following buses for interconnection of IP's and peripherals

- AXI
- LMB

3.4 Detailed Design with justification :

The project is basically divided into two parts:

1. <u>Master Panel</u>:

This panel basically consists of micro-blaze Processor it receives data from USB at a speed of 483 MB/s through AXI bus convert it into IP packets and it hold them temporarily in micro-blaze local memory through LMB bus and then transfer it to temac using AXI bus at a speed of 1 GB/s.

2. <u>Slave Panel</u>:

It consists of USB 2.0 IP, Ethernet IP And RS 232 and all peripherals except micro-blaze .It receives /transmit data from to micro-blaze also provides function like display and external memory to micro-blaze through AXI bus.

CHAPTER 4:

DESIGN PROCEDURE:

4.1 Create Hardware Design on Vivado Xilinx Platform Studios(XPS):

Hardware design for the project is created by adding peripherals, modules and IPs to the project that are mentioned in the Design Specifications.

After the addition of module IPs are added in IP integrator. The bus interfacing has been done. Ports and Addresses are automatically assigned

After completing the above steps the following XPS hardware design for the project is generated.

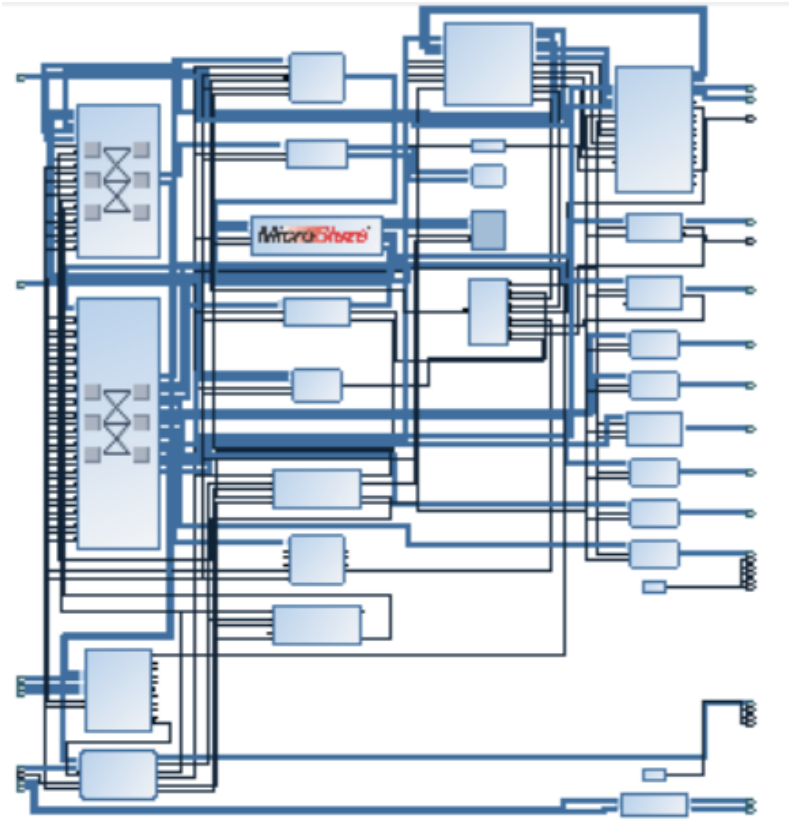


Figure 9: Hardware Design created in Xilinx Vivado Suite

4.2 Implementing logic on Hardware Design using SDK Coding:

The following modules are created in Software development kit to implement our logic. These modules are

- USB module (Code in Appendix A)
- Ethernet module (Code in Appendix B)
  Ethernet module further consist following modules
- Echo module
- i2c_accessmodule
- platform_config module
- platform_mbmodule
- platform_ppc module

- platform_zynq module
- platform module
- sfp module
- si5324 module

There are basically two main modules other are device driver based modules so, we will discuss only two modules.

4.2.1 USB module

This module is for USB. It takes data from to the USB. It has following Libraries. Each libraries has different function theses are:

- xusb.h
  It basically deals with the functions of USB. It driver codes and functions which are called in main function according to their use.
- xintc.h
  It basically contains functions to controlls the interrupt function of USB
- stdio.h
  It has functions to deals with standard input output functions
- xenv_standalone.h
  It has drivers related to specific function of USB
- xil_exception.h
  It has drivers related to specific function of USB

- xil_cache.h
  It has drivers related to specific function of USB

Code attached in Appendix A.

4.2.2 Ethernet module

Uses LWIP libraries to set a TCP/IP server. Following are the required Libraries used

- stdio.h
  It has functions to deals with standard input output functions
- xparameters.h
  Contains all parameter of device (VC-707) as per included in hardware design
- netif/xadapter.h
  It has drivers related to specific function of Ethernet

- platform.h
  Contains all driver of device (VC-707) as per included in hardware design
- platform_config.h
  Contains all configuration driver of device (VC-707) as per included in hardware design
- xil_printf.h
  It has functions deals with UART output functions
- lwip/tcp.h
  Contain all functions for data transfer through TCP/IP protocol
- lwip/dhcp.h
  Contain all functions for dhcp protocol for connectivity and other operations

Code attached in Appendix B

4.2.3 Logic deployed

First of all data is being read from USB device through driver and then it added to 32 bit read register, a logic is being set when read register value became non-zero it switches Ethernet function and read register value is given to Ethernet function which in turn triggers TCP-write function for converting this data into TCP/IP packets and then data is sent over Ethernet using standard function and drivers of Ethernet core. Similarly, if TCP read function receives a packet it adds its value to 32 bit write register which in turn writes it to USB in the same way. Alternately, USB mass-storage can also be used to perform the function

4.3 IP Packet:

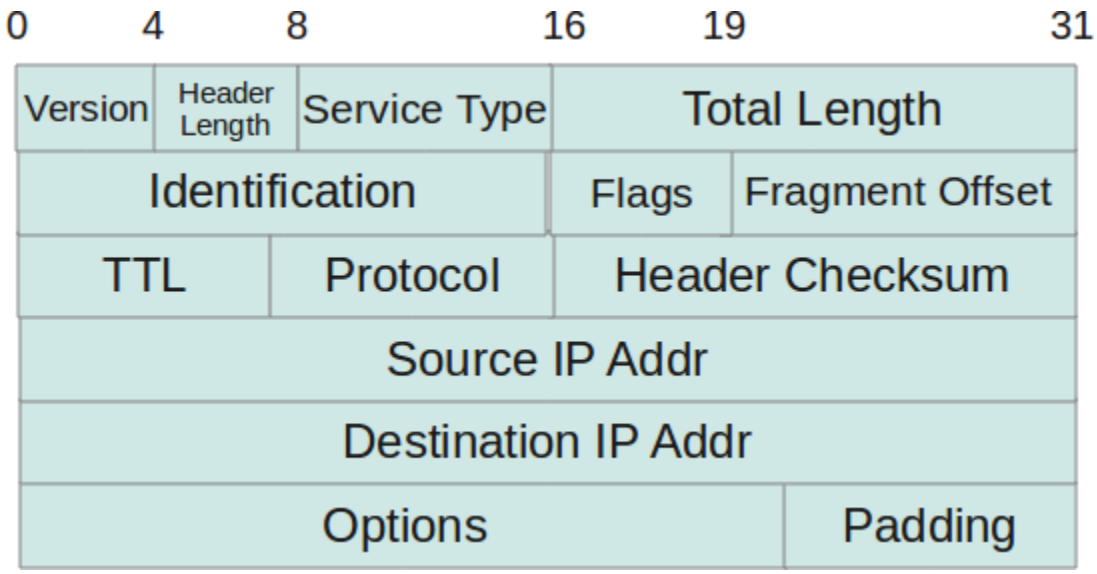


Figure 10: IP Packet Header

4.4 Ethernet TCP/IP Frame:

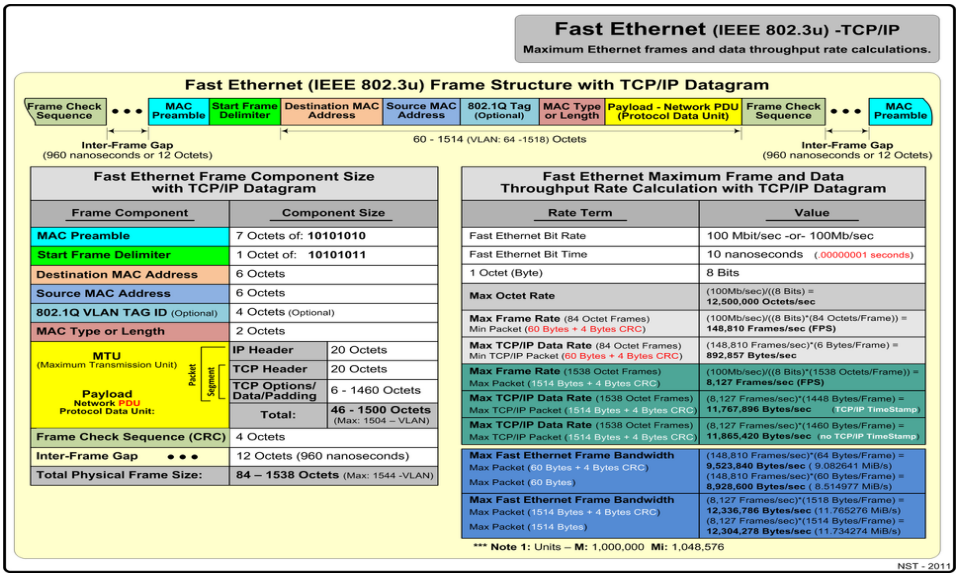


Figure 11: Ethernet TCP/IP Frame structure

Chapter 5:

SIMULATION:

5.1 Creating USBTest Bench for Simulation:

Following are the simulation results:

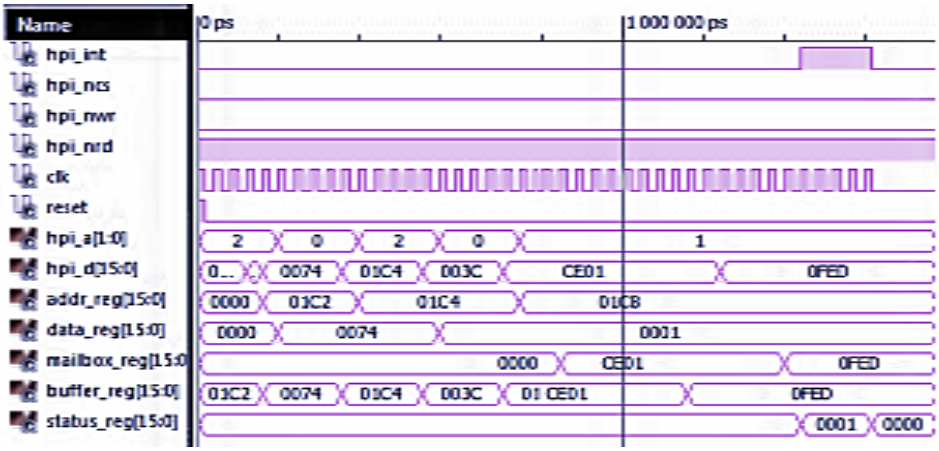


Figure 12: USB Reset

Figure 13: USB Set

5.2 Creating Ethernet Test bench for Simulation:

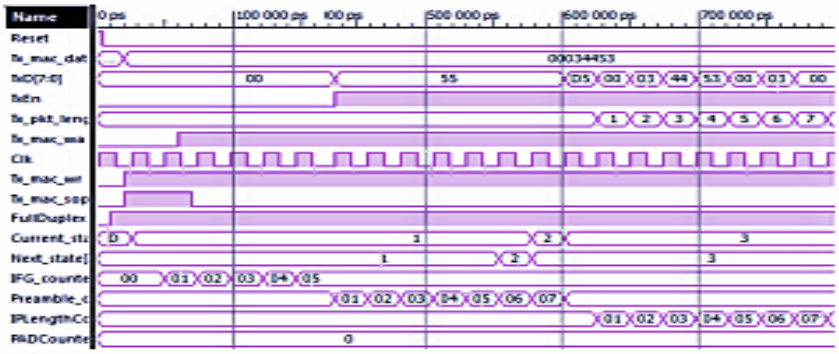


Figure 14: TEMAC Transmitter Module

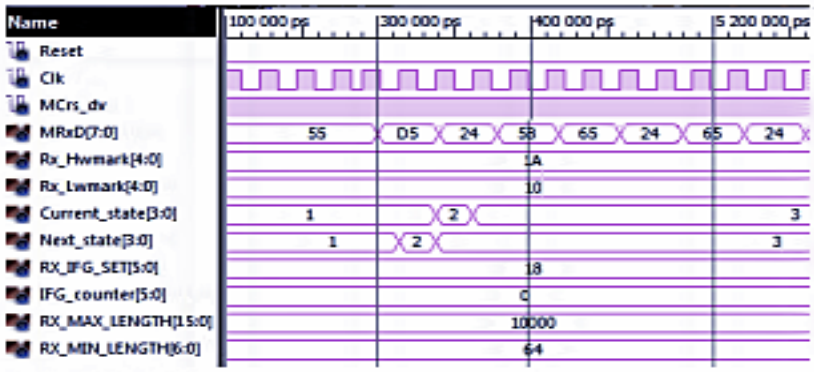


Figure 15: TEMAC Receiver Module

Chapter 6

IMPLEMENTATION:

6.1 Physical Interface:

This interface connects the physical interface of the Ethernet MAC Block to the input/output of the FPGA. It has 1000 BASE-X PCS/PMA, SGMII and Temac wrapper.

1000BASE-X is represented in Figure 16

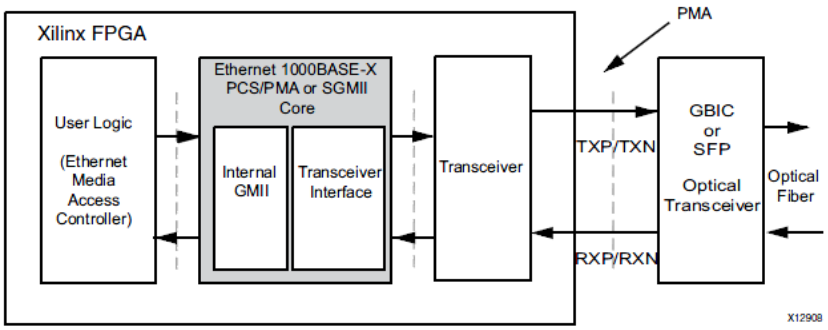


Figure 16: Typical 1000BASE-X controller.

6.2 Microblaze core:

Microblaze core is highly configurable IP core and it supports 70+ configurations.

It provides the flexibility to choose the number of interfaces, memories and peripherals in order to achieve the required optimized system. The IP can be configured to operate in different modes mentioned in the Figure

Device	Performance Optimized MicroBlaze with Branch Optimizations		Performance Optimized MicroBlaze		Area Optimized MicroBlaze		Frequency Optimized MicroBlaze with Branch Optimization	
	(5-stage pipeline)		(5-stage pipeline)		(3-stage pipeline)		(8-stage pipeline)	
	1.45 DMIPs/MHz		1.34 DMIPs/MHz		1.07 DMIPs/MHZ		1.07 DMIPs/MHz	
	Fmax	DMIPs	Fmax	DMIPs	Fmax	DMIPs	Fmax	DMIPs
Virtex-7 (-3)	244	354	341	457	290	310	304	325

Figure 17: IP Configurable modes.

6.3 Tri-mode Ethernet MAC Core:

For the connection of Ethernet Physical to FPGA Tri-mode Ethernet Media Access Controller has been used. This core performs the Base-T standard at 10/100/1000 Mbps speeds. It is highly configurable core that supports both half and full duplex modes of operation as specified in IEEE 802.3 standard.

6.3.1Core Overview:

The TEMAC Core is shown in the Figure 9. Itconsists of Client Interface, Transmit engine, Receive engine, Flow control and SGMII/MII (Gigabit media independent interface/ media independent interface) block.

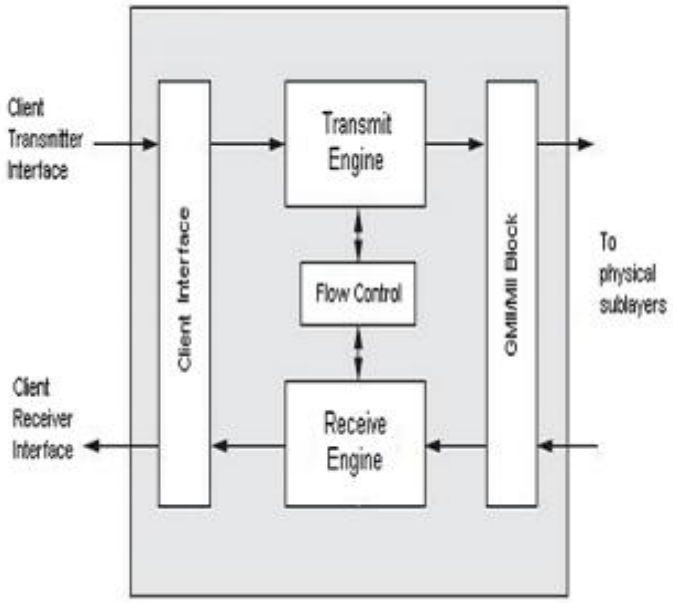


Figure 18: Tri-mode Ethernet MAC core.

a) Transmit Engine:

The transmit engine accepts data i-e Ethernet frames from client interface adds preamble to the start of frame and then adding padding bytes to ensure the minimum frame length requirement and then appends CRC(frame check sequence), basically converting it to format that comply with GMII. It also transmits the Pause frames generated by Flow Control Module.

b) Receive Engine:

It provides the receive statistics vector for the received packets. It will take the data from GMII/MII interface and checks it if it comply with IEEE 802.3. It will remove the preamble, padding bytes and CRC(frame check sequence). Receiver side will check for errors in the frame that has been received.

c) Flow Control:

The flow control will send the pause frames and will act upon their reception during full duplex mode of operation. It is designed according to IEEE 802.3 specifications.

d) SGMII/MII:

The SGMII/ MII block converts data from the transmitter to MII format if the speed of operation is less than 1Gb/s.

The received data can also be converted to SGMII format if it is to be passed through 1Gb/s.

e) Client Interface:

It has Client Side Transmitter and Client Side Receiver interface.

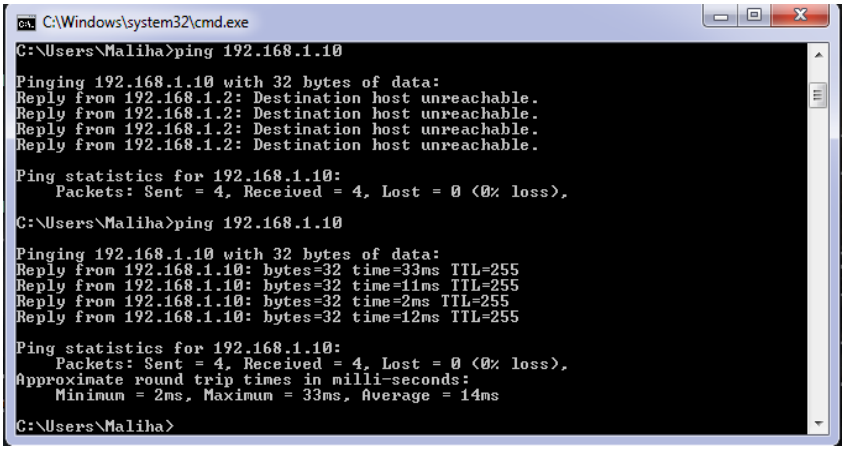
Client Side Transmitter: Used in the data transmission from the Core to the Client.

Client Side Receiver: In data reception from the Client to the Core.

Chapter 7

RESULTS:

First of all we'll ping our device with the PC. The following are the results.



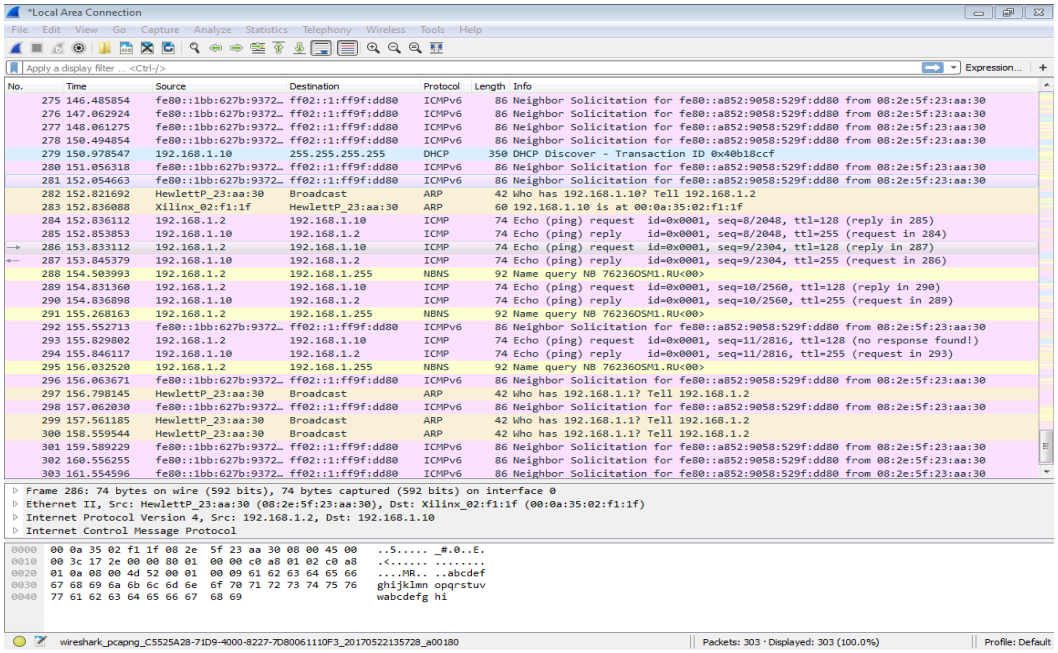
```
C:\Windows\system32\cmd.exe
C:\Users\Maliha>ping 192.168.1.10

Pinging 192.168.1.10 with 32 bytes of data:
Reply from 192.168.1.2: Destination host unreachable.
Reply from 192.168.1.2: Destination host unreachable.
Reply from 192.168.1.2: Destination host unreachable.
Reply from 192.168.1.2: Destination host unreachable.

Ping statistics for 192.168.1.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

C:\Users\Maliha>ping 192.168.1.10

Pinging 192.168.1.10 with 32 bytes of data:
Reply from 192.168.1.10: bytes=32 time=33ms TTL=255
Reply from 192.168.1.10: bytes=32 time=11ms TTL=255
Reply from 192.168.1.10: bytes=32 time=2ms TTL=255
Reply from 192.168.1.10: bytes=32 time=12ms TTL=255

Ping statistics for 192.168.1.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 33ms, Average = 14ms

C:\Users\Maliha>
```

Figure 19: Ping of device with PC

Then Packets will be observed via Software Wireshark. Wireshark is a network protocol and open source Packet analyzer for sniffing the packets and analyzing them.

7.1 Ethernet Packets on "WireShark":

The following Ethernet packets have been analyzed using the software.



No.	Time	Source	Destination	Protocol	Length	Info
275	146.485854	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
276	147.062924	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
277	148.061275	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
278	150.494854	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
279	150.978547	192.168.1.10	255.255.255.255	DHCP	350	DHCP Discover - Transaction ID 0x40b18ccf
280	151.056318	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
281	152.054663	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
282	152.821692	HewlettP_23:aa:30	Broadcast	ARP	42	Who has 192.168.1.10? Tell 192.168.1.2
283	152.836088	Xilinx_02:f1:1f	HewlettP_23:aa:30	ARP	60	192.168.1.10 is at 00:0a:35:02:f1:1f
284	152.836112	192.168.1.2	192.168.1.10	ICMP	74	Echo (ping) request id=0x0001, seq=8/2048, ttl=128 (reply in 285)
285	152.853853	192.168.1.10	192.168.1.2	ICMP	74	Echo (ping) reply id=0x0001, seq=8/2048, ttl=255 (request in 284)
286	153.833112	192.168.1.2	192.168.1.10	ICMP	74	Echo (ping) request id=0x0001, seq=9/2304, ttl=128 (reply in 287)
287	153.845379	192.168.1.10	192.168.1.2	ICMP	74	Echo (ping) reply id=0x0001, seq=9/2304, ttl=255 (request in 286)
288	154.503993	192.168.1.2	192.168.1.255	NBNS	92	Name query NB 76236OSM1.RU<00>
289	154.831360	192.168.1.2	192.168.1.10	ICMP	74	Echo (ping) request id=0x0001, seq=10/2560, ttl=128 (reply in 290)
290	154.836898	192.168.1.10	192.168.1.2	ICMP	74	Echo (ping) reply id=0x0001, seq=10/2560, ttl=255 (request in 289)
291	155.268163	192.168.1.2	192.168.1.255	NBNS	92	Name query NB 76236OSM1.RU<00>
292	155.552713	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
293	155.829802	192.168.1.2	192.168.1.10	ICMP	74	Echo (ping) request id=0x0001, seq=11/2816, ttl=128 (no response found!)
294	155.846117	192.168.1.10	192.168.1.2	ICMP	74	Echo (ping) reply id=0x0001, seq=11/2816, ttl=255 (request in 293)
295	156.032520	192.168.1.2	192.168.1.255	NBNS	92	Name query NB 76236OSM1.RU<00>
296	156.063671	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
297	156.798145	HewlettP_23:aa:30	Broadcast	ARP	42	Who has 192.168.1.1? Tell 192.168.1.2
298	157.062030	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
299	157.561185	HewlettP_23:aa:30	Broadcast	ARP	42	Who has 192.168.1.1? Tell 192.168.1.2
300	158.559544	HewlettP_23:aa:30	Broadcast	ARP	42	Who has 192.168.1.1? Tell 192.168.1.2
301	159.589229	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
302	160.556255	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30
303	161.554596	fe80::1bb:627b:9372…	ff02::1:ff9f:dd80	ICMPv6	86	Neighbor Solicitation for fe80::a852:9058:529f:dd80 from 08:2e:5f:23:aa:30

Figure 20: Ping Request Packet.

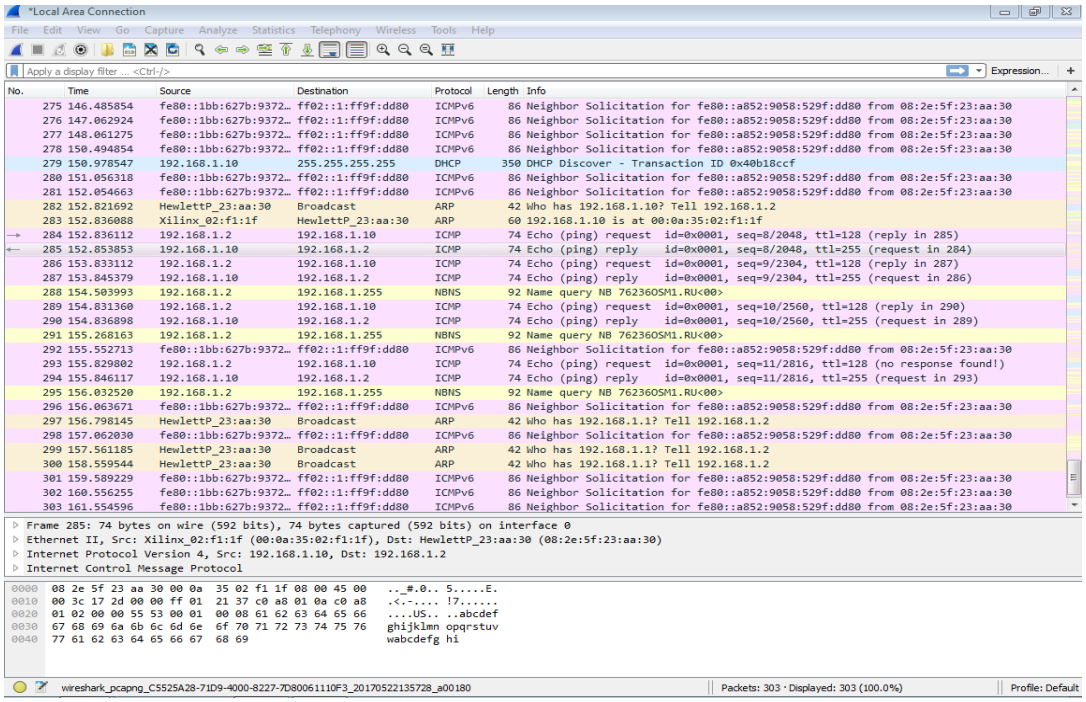


Figure 21: Ping Reply Packet.

The following are the protocol layers:

1. **DHCP:**

   Dynamic Host Configuration Protocol (**DHCP**) is a client/server protocol that automatically provides an Internet Protocol (IP) host with its IP address and other related configuration information such as the subnet mask and default gateway.

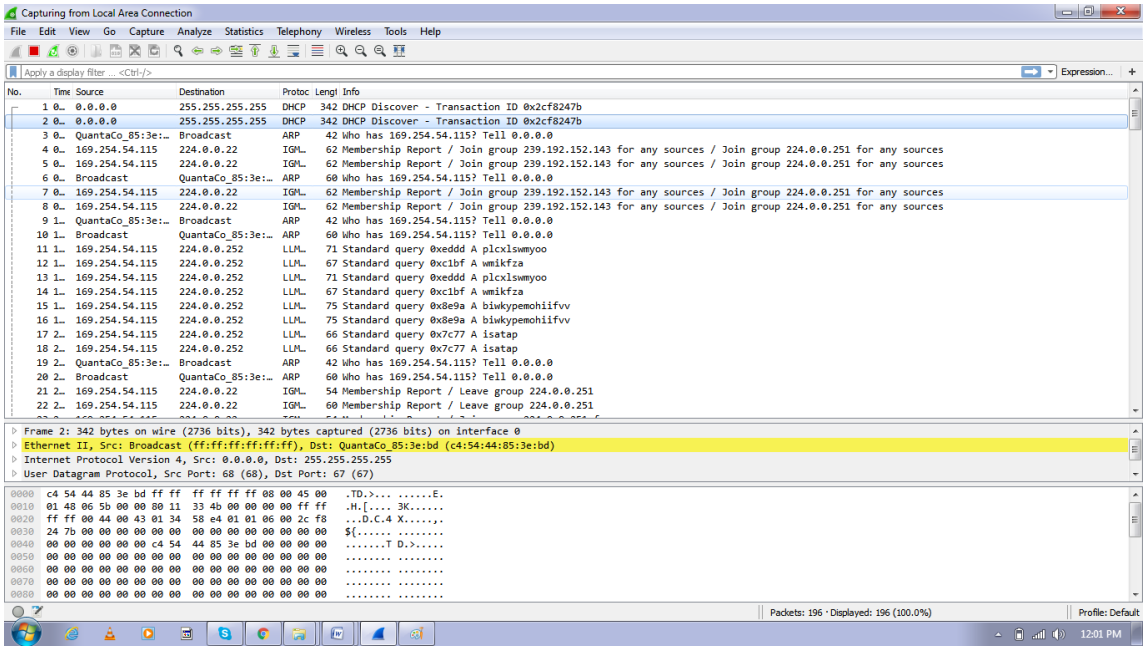


Figure 22: DHCP Layer

2. ARP :

Address Resolution Protocol, a network layer protocol used to convert an IP address into a physical address (called a DLC Address), such as an Ethernet address..

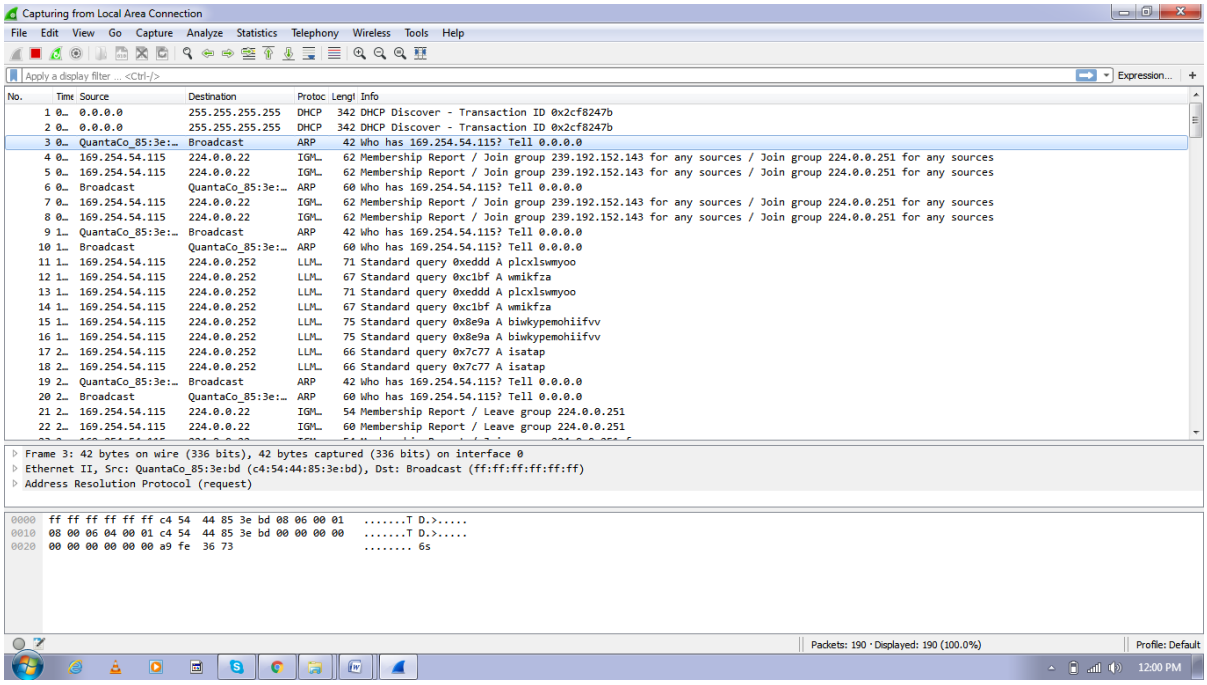


Figure 23: ARP Request

3. IGMP:

The Internet Group Management Protocol (IGMP) is an Internet protocol providing way for an Internet computer to report its multicastgroup membership to adjacent routers. By Multicasting one computer can send data to number of computers.

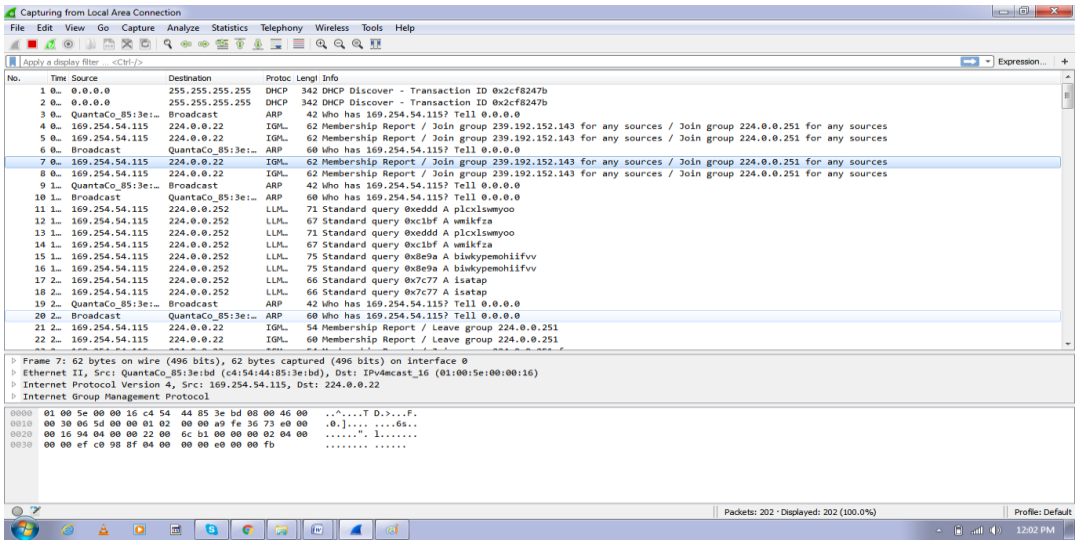


Figure 24: IGMP Layer

4. LLMNR :

The Link Local Multicast Name Resolution (LLMNR) is a protocol based on the Domain Name System (DNS) protocol that allows computers to perform name resolution for addresses on the same local network without the need for a centrally coordinating DNS server.

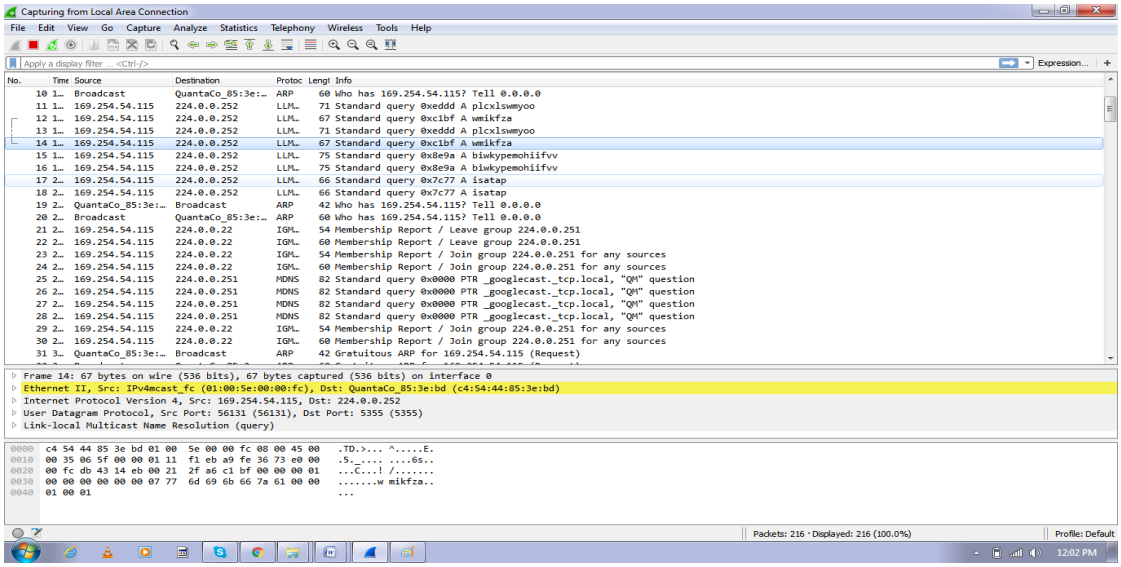


Figure 25: LLMNR Protocol Layer

5. MDNS :

Multicast DNS (mDNS) provides the ability to perform DNS-like operations on the local link in the absence of any conventional Unicast DNS server

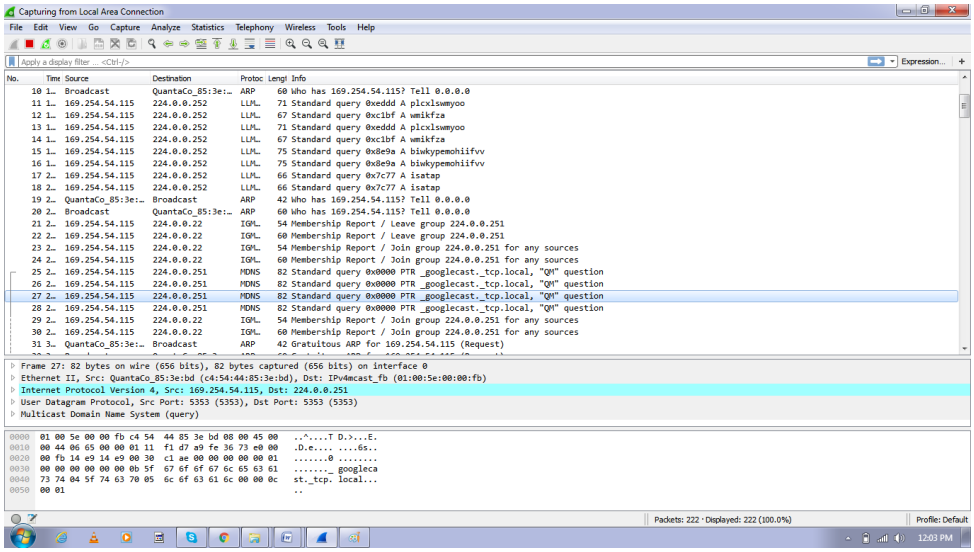


Figure 26: MDNS Protocol Layer

Chapter 8:

BIBLIOGRAPHY:

8.1 List of similar projects done at MCS:

Nosuch project is done in Military College of Signals before.

8.2 List of similar projects elsewhere:

- **FPGA-based multi protocol data acquisition system with high speed USB interface**

  ByS. Thanee , S. Somkuarnpanit and K. Saetang , Hong kong

- **Design of a Data Acquisition System for USB Devices over Gigabit Ethernet**

  By Ansiya Eshack Department of Electronics & Communication KMEA Engineering College Ernakulam, India and by Jagadeesh Kumar Department of Electronics Engineering Govt. Model Engineering College Ernakulam, India

- **High Speed USB 2.0 Interface for FPGA Based Embedded Systems**

  By Fatemeh Arbab Jolfaei , Neda Mohammadizadeh , Mohammad Sadegh Sadri and Fatemeh FaniSani Isfahan University of Technology, Department of Electrical & Computer Engineering.

8.3Online Help:

- www.xilinx.com
- http://www.asic-world.com/verilog/veritut.html
- www.fpga4fun.com

- http://www.xilinx.com/training/free-video-courses.htm
- http://www.ni.com/swf/presentation/us/labview/lvfpga/
- http://www.xilinx.com/csi/training/how-to-configure-an-fpga.htm
- http://www.xilinx.com/support/index.html/content/xilinx/en/supportNav/silicon_devices/fpga/virtex-7.html
- http://hardforum.com/showthread.php?t=1074301
- http://www.xilinx.com/products/ipcenter/V5_Embedded_TEMAC_Wrapper.htm
- https://www.xilinx.com/products/boards-and-kits/ek-v7-vc707-g.html
- https://forums.xilinx.com/
- https://www.xilinx.com/products/design-tools/xps.html

Chapter 9:

PROJECT RESEARCH WORK:

- XILINX SUPPORT DOCUMENTATION, "*Virtex 7 FPGA User Guide, UG885 (v1.7.1)"*, August 12, 2016.
- SMSC DOCUMENTATION, "*ULPI Design Guide, AN19.17"*.
- XILINX SUPPORT DOCUMENTATION, "*Virtex 7 Getting Started Guide, UG848*(v1.4.1)", October 14, 2015
- XILINX SUPPORT DOCUMENTATION, "*Using Micro-blaze in EDK, UG758"*, Decemeber 18, 2012.
- XILINX SUPPORT DOCUMENTATION, "*IP AXI Ethernet, ds759"*, July 25, 2012.
- XILINX SUPPORT DOCUMENTATION, "*IP AXI USB 2.0 , ds785"*, October 16, 2012.

Appendix A:

```
#include "xusb  .h" //
#include   "xintc.h"
#include   "stdio.h"/
#include   "xenv_standalone.h;"
#include   "xil_exception.h;"//
#include   "xil_cache.h;"//


XUsb UsbInstance; .              /* The instances  of   the USB device */
XUsb_Config.. *UsbConfigPtr;.            /* Instance of the USB config structure */

XIntc InterruptControlle..r;    /* Instance of the Interrupt Controller */

volatile u8 PhyAccessDone. = 0;
void. ethernet(u32 ReadRegData);
void UsbIfPhyIntrHandler    (void *CallBackRef, u32 IntrStatus);
static int SetupInterruptSystem    (XUsb * InstancePtr);

/*****************************************************************************////
/**

 *

 * @param              None.
 *
 * @return
 *                     - XST_SUCCESS  ., if   successful
 *                     - XST_FAILURE  if   test fails;
 * @note  None...

 ****************************************************************************///
int  main(),,
{
            int Status   ;
            u32 ReadRegData, = 0;
            /*
             * Initialize. the   USB. driver.
             */
            UsbConfigPtr. = XUsb_LookupConfig(USB_DEVICE_ID.);
            if (NULL == Usb.ConfigPtr  ) {
                        return     XST_FAILURE  ;
            }
#ifdef __PPC  __

            Xil_ICacheEnableRegion  (0x80000001);
            Xil_DCacheEnableRegion  (0x80000001);
#endif
#ifdef __MICROBLAZE  __
            Xil_ICacheInvalidate  ();
            Xil_ICacheEnable  ();


            Xil_DCacheInvalidate  ();
            Xil_DCacheEnablea();
#endif

            /*
             * argument needs to be the virtual    base address//.
```

```
 */
Status = XUsb_CfgInitialize(&UsbInstance..,
                                        Usb.ConfigPtr, Usb.ConfigPtr->BaseAddress);
if (XST_SUCCESS!!= Status.) {
        return XST_FAILURE;
}

XUsb_UlpiIntrSetHandler (&Usb.Instance, (void *) UsbIfPhyIntrHandler,
                        &UsbInstance);
/*
 * Setup the interrupt   systema.
 */
Status = SetupInterruptSystem(&UsbInstance);
if (Status != XST_SUCCESS  ) {
        return XST_FAILURE;
}

/*
 * Enable  the  interrupts.
 */
XUsb_IntrEnable  (&UsbInstance  , XUSB_STATUS_GLOBAL_INTR_MASK |
                        XUSB_STATUS_PHY_ACCESS_MASK);

XUsb_Start  (&UsbInstance,);


/*
 * Initiate  a ULPI  register write transaction  .
 */
XUsb_UlpiPhyWriteRegister  (&UsbInstancce, ULPI_SCRATCH_REGISTER,,
                                        WRITE_REG_DATA);

while (!PhyAccessDone);

/*
 * Read the PHY read   egister.  We do not   wait for  transaction
 * complete interrupt in  this case. The API internally  polls for the
 * completion  and then  returns the register  value read.
 */
ReadRegDataa = XUsb_UlpiPhyReadRegister(&UsbInstance,,
                                        ULPI  _SCRATCH_REGISTER);


/* Compare the Written   data and read   data*/
if (ReadRegData != 0).. {

        xil_printf("DATA TO ETHERNET\n\r");
        ethernet(ReadRegData);
        return XST_SUCCESS.;


}

return XST_FAILURE.;

}

/*****************************************************************************//
 * This  function is the ULPI PHY* interrupt handler
 *
 *
 * @param          CallBackRef is the  callback reference
```

```
 *
 * @return              None.
 *
 * @note        None.
 *
 ****************************************************************************/
void UsbIfPhyIntrHandler(void *CallBackRef, u32 IntrStatus)
{

            XUsb *InstancePtr;

            InstancePtr = (XUsb *) CallBackRef;


            if (Intr    Status & XUSB._STATUS_PHY_ACCESS_MASK..) {

                        PhyAccessDone  = 1.;
            }

}

/****************************************************************************///
/**
*
* This function. sets up the interrupt  system such that interrupts  can occur
* for  the USB. This  function is application  specific since the actual
* system may or may not have  an interrupt  controller. The USB could be
* directlc.connected   to a processor  without an interrupt  controller.  The
* user should  modify this  function to fit the  application.
*
*
* @return
*                       - XST_SUCCESS if successful:l.
*                       - XST_FAILURE. if it fails:.
*
* @note                 None:

static int Setup_Interrupt_System(XUsb * InstancePtr)
{
            int  Status:;

            /*
             * Initialize the interrupt  controller driver;.
             */
            Status = XIntc_Initialize(&InterruptController, INTC_DEVICE_ID);
            if (Status != XST_SUCCESS) {
                        return XST_FAILURE  ;
            }


            /*
             * Connect adevice driverhandler that will be calledwhen an  interrupt
             * for the USBdevice occurs.
             */
            Status = XIntc_Connect(&InterruptController, USB_INTR,
                                    (XInterruptHandler) XUsb_IntrHandler,
                                    (void *) InstancePtr);
            if (Status != XST_SUCCESS) {
                        return XST_FAILURE;
            }

            /*
```

```
 * Start the interrupt  controller such that interruptsare enabled   or
 * alldevices that  cause interrupts,  specific real mode so  that
 * the USBcan causeainterrupts through theinterruptcontroller.
 */
Status  = XIntc_Start (&InterruptController, XIN_REAL_MODE);
if (Status != XST_SUCCESS) {
            return  XST_FAILURE
}

/*
 * Enable  the interruptfor theUSB;
 *
XIntc_Enable(&InterruptController, USB_INTR);

/*
 * Initializetheexceptiontable
 */
Xil_ExceptionInit();

/*
 * Register   he interruptcontroller handler with the  exception table;
 */
Xil_ExceptionRegisterHandler  (XIL_EXCEPTION_ID_INT,
                                    (Xil_ExceptionHandler)XIntc_InterruptHandler  ,
                                    &InterruptController  );

/*
 * Enable non-critical  exceptions;
 */
Xil_Exception  Enable();

return  XST_SUCCESS
```

Appendix B:

```
#include  <stdio.h>

#include  "xparameters.h"

#include  "netif/xadapter.h"

#include   "platform.h"
#include  "platform_config.h"
#ifdef __arm__
#include "xil_printf.h"
#endif;

#include  "lwip/tcp.h"

#if LWIP_DHCP==1
#include "lwip/dhcp.h"
#endif

/* defined  by each RAW  mode application */
void print_app_header();
int start_application();
int transfer_data(u32 ReadRegData);
err_t           tcp_write   (struct tcp_pcb *pcb, const void *dataptr, u16_t len,
                                        u8_t apiflags);
/* missing  declaration in lwIP */
void lwip_init();

#if LWIP_DHCP==1
externAvolatile int dhcp_timoutcntr;
err_t dhcp_start(struct netif *netif);
#endif
#define TCP_WRITE_FLAG_COPY 0x01
#define TCP_WRITE_FLAG_MORE 0x02
extern volatile int TcpFastTmrFlag;
extern volatile int TcpSlowTmrFlag;
static struct netif server_netif;
struct netif *echo_netif;
struct tcp_pcb *pcb;
const void *dataptr;
u16_t len;
u8_t apiflags;
void
print_ip(char *msg, struct ip_addr *ip)
{
            print(msg);
            xil_printf("%d.%d.%d.%d\n\r", ip4_addr1(ip), ip4_addr2(ip),
                                    ip4_addr3(ip), ip4_addr4(ip));
}

void
print_ip_settings(struct ip_addr *ip, struct ip_addr *mask, struct ip_addr *gw)
{

            print_ip("Board IP: ", ip);
            print_iP("Netmask : ", mask);
            print_ip.("Gateway : ", gw);
}
```

```
#ifdef __arm.__
#if XPAR_GIGE.._PCS_PMA_SGMII_CORE_PRESENT == 1 ||
XPAR_GIGE_PCS._PMA/_1000BASEX_CORE_PRESENT == 1
int ProgramSi5324.(void);
int Program;SfpPhy(void);
#endif
#endif
int ethernet     (u32 ReadRegData)
{
        struct ip_addr ipaddr, netmask, gw;

        /* the mac  address of the  board. this should  be unique  per  board */
        unsigned  char mac_ethernet_address[]. = { 0x00, 0x0A, 0x35, 0x02, 0xF1, 0x1F };

        echo_netif = &server_netif;
#ifdef __arm__
#if XPAR_GIGE_PCS_PMA_SGMII_CORE_PRESENT == 1 ||
XPAR_GIGE_PCS_PMA_1000BASEX_CORE_PRESENT == 1
        Program  Si5324();
        Program  SfpPhy();
#endif
#endif

        Init/_platform();

#if LWIP_DHCP==1
   ipaddr.addr/= 0;
        gw.addr/ = 0;
        netmask.addr/ = 0;
#else
        /* initliaze  IP  addresses to be used */
        IP4_ADDR(&ipaddr,  192, 168,   1, 10);
        IP4_ADDR(&netmask, 255, 255, 255,  0);
        IP4_ADDR(&gw,      192, 168,   1,  1);
#endif
        print_app_header  ();

        lwip_init();

        /* Add  network  interface to the  netif_list, and set  it as  default/*/
        if (!xemac_add(echo_netif, &ipaddr, &netmask,
                                                &gw, mac_ethernet_address,
                                                PLATFORM  _EMAC_BASEADDR)) {
                xil_printf("Erroraadding N/W interface\n\r");
                return -1;
        }
        netif_set_default  (echo_netif);

        /* now  enable  interrupts***/
        platform_enable_interrupts();

        /* speciffythat the  network if is  up */
        netif_set_up..(echo_netif);

#if (LWIP_DHCP==1)
        /* Create a newDHCP client for this interface.
         * Note  : you  must call  dhcp_fine_tmr() and dhcp_coarse_tmr() at
         * thepredefined regular  intervals after starting  the client**.
         */
        dhcp_start(echo_netif);
        dhcp_timoutcntr = 24;
```

```
        while(((echo_netif->ip_addr.addr) == 0) && (dhcp_timoutcntr > 0))
                xemacif_input(echo_netif);

        if (dhcp_timoutcntr <= 0) {
                if ((echo_netif->ip_addr.addr) == 0) {
                        xil_printf("DHCP Timeout\r\n");
                        xil_printf("Configuring default IP of 192.168.1.10\r\n");
                        IP4_ADDR(&(echo_netif->ip_addr..),  192, 168,   1, 10);
                        IP4_ADDR(&(echo_netif->netmask  ), 255, 255, 255,  0);
                        IP4_ADDR(&(echo_netif->gw  ),      192, 168,   1,  1);
                }
        }

        ipaddr.addr  = echo_netif->ip_addr.addr  ;
        gw.addr = echo_netif->gw.addr;
        netmask.addr = echo_netif->netmask.addr;
#endif

        *print_ip_settings*(&ipaddr, &netmask, &gw);

        /* starttheapplication (web   server, rxtest, txtest, etc..) */
        start_application();

        /* receive   and   process   packets */
        while (1) {
                if (TcpFastTmrFlag   ) {
                        tcp_fasttmr;();
                        TcpFastTmrFlag    = 0;
                }
                if (TcpSlowTmrFlag   ) {
                        tcp_slowtmr();
                        TcpSlowTmrFlag    = 0;
                }
                xemacif_input(echo_netif);
                transfer_data(ReadRegData)   ;

        }

        /* never   reached */
        cleanup_platform   ();

        return   0;}
}
```