APPLICATION OF ETHERNET OVER POWERLINE COMMUNICATION

*A Report Submitted in Partial Fulfillment
of the Requirements for the Degree of*

Bachelor of Technology
In
Electronics and Communication Engineering
By

JAMI ADITYA
and
PRIYANKA PRIYADARSINI

Department of Electronics and Communication Engineering
National Institute of Technology, Rourkela
May, 2013

APPLICATION OF ETHERNET OVER POWERLINE COMMUNICATION

A Report Submitted in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Technology
In
Electronics and Communication Engineering

Under the aegis of
PROF. SARAT KUMAR PATRA
By

JAMI ADITYA
(109EC0314)
and
PRIYANKA PRIYADARSINI
(109EC0177)

Department of Electronics and Communication Engineering
National Institute of Technology, Rourkela
May, 2013

National Institute of technology, Rourkela

DECLARATION

We hereby declare that the project work entitled "Application of Ethernet over Powerline Communication" is a record of our original work done under Dr. Sarat Kumar Patra, Professor, National Institute of Technology, Rourkela. Throughout this documentation wherever contributions of others are involved, every endeavor was made to acknowledge this clearly with due reference to literature. This work is being submitted in the partial fulfillment of the requirements for the degree of Bachelor of Technology in Electronics and Communication Engineering at National Institute of Technology, Rourkela for the academic session 2009– 2013.

Jami Aditya   Priyanka Priyadarsini

(109EC0314)   (109EC0177)

National Institute of technology, Rourkela

CERTIFICATE

This is to certify that the thesis entitled "Application of Ethernet over Powerline Communication" submitted by Jami Aditya (109EC0314) and Priyanka Priyadarsini (109EC0177) in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Electronics and Communication Engineering at National Institute of Technology, Rourkela is an authentic work carried out by them under my supervision and guidance.

Prof. Sarat Kumar Patra

Department of E.C.E

National Institute of Technology

Rourkela

ACKNOWLEDGEMENT

This is a real-time project and the fact that we have been able to complete it successfully owes a lot to a number of people associated with us during this project.

First of all, we would like to thank Prof. SARAT KUMAR PATRA for giving us the opportunity to work on such an interesting topic and providing a thoroughly professional environment. He also guided us throughout the project period and helped us time to time with his vast experience and innovative ideas.

We wish to extend our sincere thanks to Prof. S. K. Meher, Head of our Department, for approving our project work with great interest.

We also appreciate Prof. S.K.Behera, Prof. Poonam Singh, Prof. S.M. Hiremath and other staff members for the invaluable feedback and comments that helped us improve our work.

We are also grateful to Prof. A.K. Swain and Mr. Pallab Maji for their unconditional help and support without which this project could not have been successful.

We are also thankful to Research Scholars and M. Tech. students for their co-operation in usage of laboratories and to all our friends who have directly or indirectly helped us with the thesis and project.

ABSTRACT

Powerline communication (PLC) has seen notable demand due to its efficiency and the wide range of applications. It is a system for carrying data on a conductor that is also used for electric power transmission. In this thesis, this technique is used to implement a web server using the FPGA development board (DE 2). The information is sent through Ethernet over powerline. Using DE2 Altera kit, a webserver is created. The implementation of web server is done by first instantiating a Nios II system on the board. Nios II system is built around the Altera's Nios II processor using the SOPC builder tool of the quartus II CAD tool. The SOPC builder tool generates the VHDL code of the defined system. The developed code is, then, configured in the FPGA board to instantiate the system. After implementing the Nios II system, an application program is run in the system to implement the web server. Ethernet packets are sent through the LAN cable over powerline. The output is taken from the Ethernet port and the fed into the Homeplug adapter. The packets are captured using wireshark packet sniffer and detailed analysis is done. This idea can be used for transmitting Ethernet packets over powerline using Homplug adapters.

Keywords: Nios II system, SOPC builder tool, Homeplug adapter, powerline communication, Wireshark

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Powerline Communications

1.1.1 Early power line communication technologies

Efforts to use to the Powerline as a transmission medium were 160 years ago, but high speed transmission over 10 Mbps were archived in the mid-1990s. Before this time, the various types of power lines were assumed to be inherently low data rate transfer media. There are several Powerline Communication (PLC) protocols. The typical purpose of many of these protocols is not only for home networking, but also for powerline control protocols for home automation, home security, and lighting control. On the other hand, the HomePlug protocol is a high speed home network standard. Although, using the same powerline as home automation protocols, the HomePlug 1.0 device can coexist with device using these other protocols because by using a different frequency band than powerline control technologies such as X-10, CEBus, and Lonworks. There are also some preliminary high-speed PLC network devices with limited usage in Europe, where the interest is primarily on access ("last 100 feet") rather than in-home networks. However, these are expensive and do not enjoy wide use yet; they also must contend with both different regulatory environments and power distribution topologies.

1.1.2 Power line medium

Powerlines were originally devised for transmission of power at 50-60 Hz and at most 400 Hz. At high frequencies the power line is very hostile for signal propagation. Powerline networks operate on standard in-building electrical wiring and as such consist of a variety of conductor types and cross sections joined at random. Therefore, a wide variety of characteristic impedances will be encountered in the network. Further, the network terminal impedance will tend to vary both with communication signal frequencies and with time as the consumer premises load pattern varies quite much. This impedance mismatch causes a multi-path effect resulting in deep notches at certain frequencies. In a typical home environment the attenuation on the power line is between 20 dB and 60 dB, and is a strong function of load. The major sources of noise on power line are from electrical appliances, which utilize the 50 Hz electric supply and which generate noise components that extend well into the high frequency spectrum. The common sources of electrical noise are certain types of halogen and fluorescent lamps, switching power supplies, motors and variable resistance dimmer switches. Apart from these, induced radio frequency signals from broadcast, commercial, citizen band and amateur stations severely impair certain frequency bands on the powerline channel. Reliable data transmission over this hostile medium requires powerful forward error correction coding (FEC), interleaving, error detection, Automatic Repeat Request (ARQ) techniques, along with appropriate modulation schemes as well as a robust medium access protocol (MAC) to overcome these impairments.

Chapter 2

LITERATURE REVIEW

2.1 Home Plug

2.1.1 Orthogonal Frequency Division Multiplexing

Orthogonal Frequency Division Multiplexing (OFDM) is one of the most promising techniques used for data transmission over power lines. OFDM is well known in the literature and in industry. It is generally used in terrestrial wireless distribution of television signals, DSL technology and has also been adapted for IEEE's high rate wireless LAN Standards (802.11a and 802.11g). The idea of OFDM is to divide the available spectrum into several narrow bands, low data rate subcarriers. In this respect, it is a type of Discrete MultiTone modulation (DMT). The frequency response of the subcarriers are overlapping and orthogonal, to obtain high spectral efficiency hence the name OFDM. Each narrow band subcarrier can be modulated using various modulation formats. By choosing the subcarrier spacing as small the channel transfer function reduces to a simple constant which is within the bandwidth of each subcarrier. Thus, a frequency selective channel is divided into many flat fading subchannels. This also eliminates the need for sophisticated equalizers.

OFDM has numerous advantages. OFDM

(1) It excellent mitigation of the effects of time-dispersion

(2) Its very good at minimizing the effect of in-band narrowband interference;

(3) It has high bandwidth efficiency;

(4) It is scalable to high data rates;

(5) It is flexible and can be made adaptive; different modulation schemes for subcarriers, adaptable bandwidth/data rates, bit loading are possible;

(6) It has excellent ICI performance, so complex channel equalization is not required.

For the above reasons it is an excellent candidate for powerline communication

2.1.2 HomePlug 1.0 Physical Specifications

This section and the next provide a detailed overview of the HomePlug 1.0 specifications at a level appropriate for a protocol engineer. This section describes the Physical (PHY) layer, and the following section describes the Medium Access Control (MAC) layer. Briefly, the PHY uses adaptive Orthogonal Frequency Division Multiplexing (OFDM) with Cyclic Prefix (CP). Both turbo product codes and Reed-Solomon concatenated with convolutional codes are used at various times for forward error correction. The PHY layer detects channel conditions using channel estimation. It, then, adapts

by avoiding poor subcarriers and selecting an appropriate modulation method and coding rate for the remaining subcarriers. Three variants of Phase Shift Keying (PSK) modulation are used: Coherent Binary PSK (BPSK), Differential BPSK (DBPSK), and Differential Quadrature PSK (DQPSK). A preamble and frame control combination are used as delimiters that start and end long frames, with only the payload portion adapted to the channel conditions. Physical carrier sense (PCS) is performed by the PHY layer, and helps the MAC determine when the medium is busy. Fine details necessary to implement a compliant system are available in the official specifications. Connectors are assumed to contact only one line phase (L1 or L2) of the local power line network, and neutral. Connectors may or may not have ground contacts, and the user should be able to connect or disconnect at any time.

2.1.3 HomePlug Physical Layer

Orthogonal Frequency Division Multiplexing (OFDM) is the basic transmission technique used by the HomePlug. OFDM is well known in the literature and in industry. It is currently used in DSL technology, terrestrial wireless distribution of television signals, and has also been adapted for IEEE's high rate wireless LAN Standards (802.11a and 802.11g). The basic idea of OFDM is to divide the available spectrum into several narrowband, low data rate subcarriers. To obtain high spectral efficiency the frequency response of the subcarriers are overlapping and orthogonal, hence the name OFDM. Each narrowband subcarrier can be modulated using various modulation formats. By choosing the subcarrier spacing to be small the channel transfer function reduces to a simple constant within the bandwidth of each subcarrier. In this way, a frequency selective channel is divided into many flat fading subchannels, which eliminates the need for sophisticated equalizers. The OFDM used by HomePlug is specially tailored for powerline environments. It uses 84 equally spaced subcarriers in the frequency band between 4.5MHz and 21MHz. Cyclic prefix and differential modulation.

2.1.4 An Adaptive Approach

The powerline channel between any two links has different amplitude and a phase response. Furthermore, noise on the powerline is local to receiver. HomePlug technology optimizes data rate on each link by means of an adaptive approach. Channel adaptation is achieved by modulation, Tone Allocation and FEC choice. Tone allocation is a process by which certain heavily impaired carriers are turned off. This reduces the bit error rates (BER) significantly and helps in targeting the power of FEC and Modulation choices on the good carriers. HomePlug allows choosing from DBPSK ½, DQPSK ½ and DQPSK ¾ on all carriers. The result of this adaptation is a highly optimized link throughput. Some types of information, like broadcast packets, can't make use of channel adaptation techniques. HomePlug uses another innovative modulation called ROBO, so that information is reliably transmitted. ROBO modulation uses DBPSK with heavy error correction with bit repetition in

time and frequency to enable very reliable communication. ROBO frames can also be used for channel adaptation.

2.1.5 HomePlug MAC

The choice of Medium Access Control (MAC) protocol provides a different set of challenges. Home networks should be able to support a diverse set of applications ranging from simple file transfer to very high QoS demanding applications such as Voice-over-IP (VoIP) and Streaming Media. The HomePlug MAC is built to seamlessly integrate with the physical layer and addresses these needs. HomePlug MAC is modeled to work with IEEE 802.3 frame formats. This choice simplifies the integration with the widely used Ethernet. HomePlug MAC appends the Ethernet frames with encryption and other management before transmitting it over the powerline. A segmentation and reassembly mechanism is used to in cases where the complete packet cannot be fit in a single frame.

2.1.6 Frame Formats

HomePlug technology uses two basic frame formats (refer Figure 1). A Long Frame consists of a Start of Frame (SOF) delimiter, Payload and End of Frame delimiter (EOF). A Short Frame consists of a Response Delimiter and is used as part of the Stop-and Wait automatic repeat request (ARQ) process. ARQ mechanism causes retransmission of corrupt packets, thus reducing the packet error rate. All the delimiters share a common structure. A delimiter consists of a Preamble and Frame Control information field. The Preamble is a form of spread spectrum signal that is used to determine the start of a delimiter. This is followed by Frame Control information, which is encoded using a robust Turbo Product Code and can be detected reliably even at several dB below the noise floor. Among other things, delimiters convey timing information that is used by MAC to determine the availability of the medium. The robust design of the delimiter helps the nodes to obtain a very high level of synchronization, thus reducing unintended collisions. The details of the various field contained in the Frame Control are given in Table 1. The Payload of the Long Frame delimiter is encoded based on the channel adaptation. The first 17 bytes of the payload contain the Frame Header. This field contains the source address, destination address and segmentation information.

HomePlug technology limits the maximum length of the payload field in the Long Frame to 160 OFDM Symbols (~1.3 msec). This manifests as better guarantees of QoS as the delay incurred by higher priority traffic due to ongoing lower priority transmission is reduced. If the packet cannot be fitted into a Long Frame, a segmentation and reassembly mechanism is used to send it in multiple Long Frames. The frame header contains information that is used by the receiver to properly reconstruct the segmented packet. The Payload is protected by a Frame Check Sequence (FCS) to detect uncorrected errors. Techniques (DBPSK, DQPSK) are used to completely eliminate the need

for any kind of equalization. Certain impulsive noise events are overcome by means of forward error correction (FEC) and data interleaving. The HomePlug payload uses a concatenation of Viterbi and Reed-Solomon FEC. Sensitive frame control data is encoded using turbo product codes.
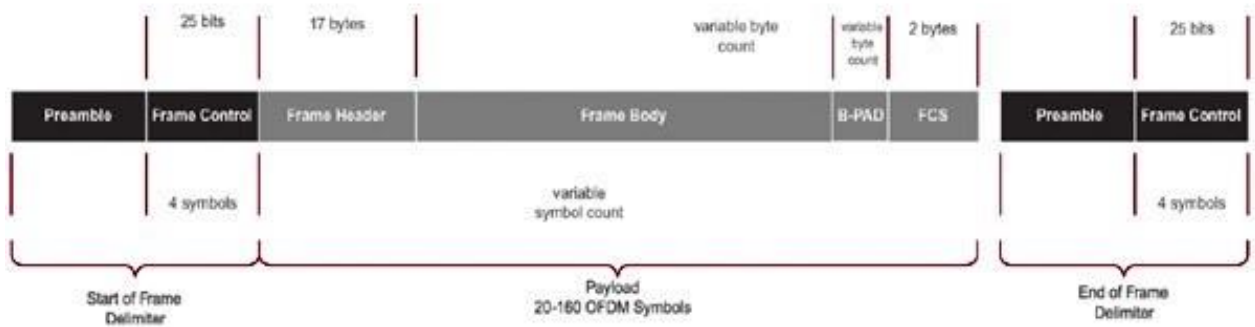
Figure 1. Long Frame Format [1]

Delimiter Type	Fields	Meaning
Start of Frame (SOF)	Type	This can be SOF with response expected or an SOF with no response expected depending on whether a Short Frame delimiter is expected at the end of this Long Frame
	Contention Control	When set to 1, this prevents all HomePlug nodes with packets of priority level equal to or less than the current Long Frame's priority from accessing the channel. However, higher priority nodes can still interrupt this transmission
	Frame Length	This indicates the length of the payload in multiples of OFDM symbol blocks
	Tone Map Index	This is an index to the channel adaptation information stored at the receiver. Note that the variable length Payload is encoded using the maximum transfer rates that can be achieved by the link.
End of Frame (EOF)	Type	This can be EOF with response expected or an EOF with no response expected depending on whether a Short Frame delimiter is expected at the end of this Long Frame.
	Contention Control	The information conveyed is same as that conveyed by this field in SOF delimiter. This redundancy helps in better synchronization.
	Channel Access Priority (CAP)	This field indicates the priority of the current Long Frame.
Response (Resp)	Type	This can be ACK (positive acknowledgment), NACK, (negative acknowledgment indicating faulty reception) , or FAIL (negative acknowledgment indicating lack of resources)
	Channel Access Priority (CAP)	This field indicates the priority of the preceding Long Frame

Table 1. Frame Control Information Fields [1]

2.2 Quartus II

The Altera Quartus II software is the most comprehensive environment available for system-on-a-programmable-chip (SOPC) design. It provides a complete design environment that easily adapts to your specific design requirements. Quartus II is a CAD system used to implement circuits in an Altera FPGA device. The Quartus II system includes a full support for all popular methods of entering a description of a desired circuit into a CAD system. The circuit can be entered in block diagram form (schematic) or using hardware description language like Verilog or VHDL.

The figure shows the design flow:

Quartus II Design Flow

Design Entry → Includes block-based design, system-level design & software development

Synthesis → Power Analysis

Place & Route → Debugging

Timing Analysis → Engineering Change Management

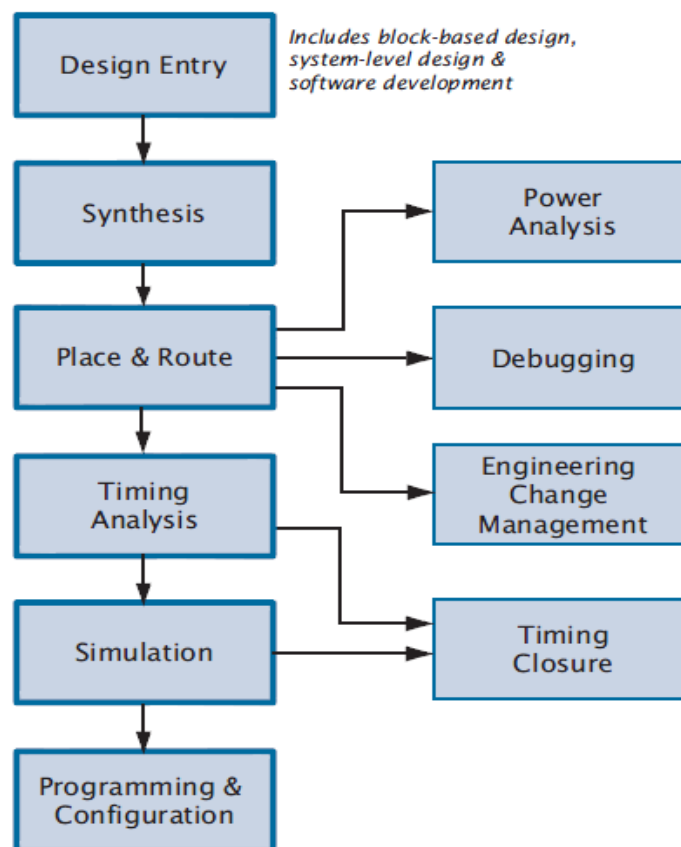Simulation → Timing Closure

Programming & Configuration

Figure 2 [4]

The CAD flow involves the following steps:

Design Entry – the desired circuit is specified either by means of a schematic diagram, or by using a hardware description language, such as Verilog or VHDL.

Synthesis – the entered design is synthesized into a circuit that consists of the logic elements (LEs) provided in the FPGA chip

Functional Simulation – the synthesized circuit is tested to verify its functional correctness; this simulation does not take into account any timing issues

Fitting – the CAD Fitter tool determines the placement of the LEs defined in the net list into the LEDs in an actual FPGA chip; it also chooses routing wires in the chip to make the required connections between specific LEs.

Timing Analysis – propagation delays along the various paths in the fitted circuit are analyzed to provide an indication of the expected performance of the circuit.

Timing Simulation – the fitted circuit is tested to verify both its functional correctness and timing

Programming and Configuration – the designed circuit is implemented in a physical FPGA chip by programming the configuration switches that configure the LEs and establish the required wiring connections.

2.3 NIOS II System

2.3.1 NIOS II Processor

The Nios II processor can be used with a variety of other components to form a complete system. The components include a number of standard peripherals; it is also possible to define custom peripherals. Altera's Education and Development board DE2 contains several components that can be integrated into a Nios II system. An example of a system that can be implemented on the DE2 board is shown in Figure 3
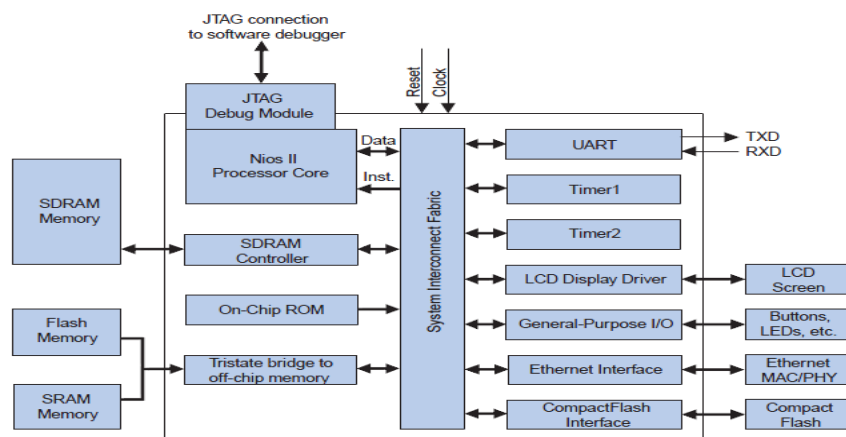

Figure 3 NIOS II System [4]

The Nios II processor is a configurable soft IP core, off-the-shelf microcontroller. Features can be added and removed on a system-by-system basis to meet price goals or performances. Soft means the processor core is not fixed in silicon and can be targeted to any Altera FPGA family. It is not required to create a new Nios II processor configuration for every new design. Altera provides ready-made Nios II system designs that one can use as it is. If these designs meet your system requirements, there is no need to configure the design further. Nios II instruction set simulator can be used to begin debugging and writing Nios II applications before the final hardware configuration is determined.

There are two broad classes of peripherals: standard peripherals and custom peripherals.

2.3.2 Standard Peripherals

Altera provides a set of peripherals commonly used in microcontrollers, such as serial communication interfaces, timers, SDRAM controllers, general-purpose I/O, and other memory interfaces. The list of available peripherals continues to increase as Altera and third-party vendors release new peripherals.

2.3.3 Custom Components

You can also create custom components and integrate them in Nios II processor system. For performance-critical systems that spend most CPU cycles to execute a specific section of code, it is a common technique to create a custom peripheral that implements the same function in hardware. This approach offers double performance benefit; the processor is free to perform other functions in parallel while the custom peripheral operates on data; and the hardware implementation is faster than software.

2.3.4 Automated System Generation

Altera's SOPC Builder system integration tool fully automate the process of configuring processor features and generating a hardware design that one can program in an Altera device. The SOPC Builder graphical user interfaces (GUI) enables you to configure Nios II processor systems with any number of peripherals and memory interfaces. One can create an entire processor system without performing any schematic or HDL design entry. SOPC Builder can also import HDL design files, which provides an easy mechanism to integrate custom logic in a Nios II processor system.

2.3.5 Internal Interrupt Controller

Nios II architecture supports 32 internal hardware interrupts. The processor core has 32 level-sensitive interrupt request (IRQ) inputs, irq0 through irq31, providing a unique input for each interrupt source. IRQ priority is determined by software. The architecture supports nested interrupts.

2.4 SOPC BUILDER

SOPC Builder is a powerful system development tool. SOPC Builder enables us to define and generate a complete system-on-a-programmable-chip (SOPC) in much less time than using traditional and manual integration methods. SOPC Builder is a tool which is included as part of the Quartus II software.

It is a general-purpose tool used for creating systems that may or may not contain a processor and may include a soft processor other than the Nios II processor. Using traditional design methods, one must manually write HDL modules to wire together the pieces of the system. It automates the task of integrating hardware components. Using SOPC Builder, one can specify the system components in a GUI and SOPC Builder generates the interconnected logic automatically. SOPC Builder generates HDL files that define all components of the system and a top-level HDL file that connects all the components together. It generates either VHDL or Verilog HDL equally.

An SOPC Builder component is a design module that SOPC Builder recognizes and can automatically integrate into a system. One can also define and add custom components or select them from a list of provided components. SOPC Builder connects multiple modules together to create a top-level HDL file called the SOPC Builder system. SOPC Builder generates a system interconnected fabric that contains the logic to manage the connectivity of all modules in the system.

2.5 FPGA & Altera's DE270 FPGA Development Board

A field programmable gate array (FPGA) is an integrated circuit designed to be configured by the designer or customer after manufacturing, so it is called field- programmable. FPGA configuration is generally specified using a hardware description language (HDL). FPGAs can be used to implement any logical function. The ability to update the partial re-configuration of the portion of the design, functionality after shipping and the low non-recurring engineering costs offer advantages for many applications.

FPGAs contain programmable logic components called logic blocks and a hierarchy of re-configurable interconnects that allow the blocks to be wired. Logic blocks can be configured to perform complex combinational functions or merely simple logic gates like XOR and AND. In most FPGAs, the logic blocks also include memory elements, which may be more complete blocks of memory or simple flip-flops.

2.5.1 DE2 Altera FPGA

DE2 board features a powerful Cyclone II FPGA chip. All the important components on the board are connected to the pins of the chip, allowing the user to configure the connection between the various components. For simple experiments, the DE2 board includes a enough number of LEDs, switches (of both toggle and pushbutton variety) and 7-segment displays. For more advanced experiments, there are Flash memory, SDRAM and SRAM chips. For experiments that require a simple I/O interfaces and processor, it is easy to instantiate Altera's Nios II processor and use interface standards such as PS/2 and RS-232. For experiments that involve video signals or sound, there are standard connectors provided on the board. For large design projects, it is possible to use the SD memory card. Finally, it is possible to connect other user designed boards to the DE2 board by means of 2 expansion headers. Software provided with the DE2 board features the Quartus II web edition design tools. It includes simple monitor program that allows one to control various parts of the board in an easily understandable manner.

2.6 Nios II System Development Flow

This section discusses the complete design flow for creating a Nios II system and prototyping it on a target board. Figure 4 shows the Nios II system development flow.
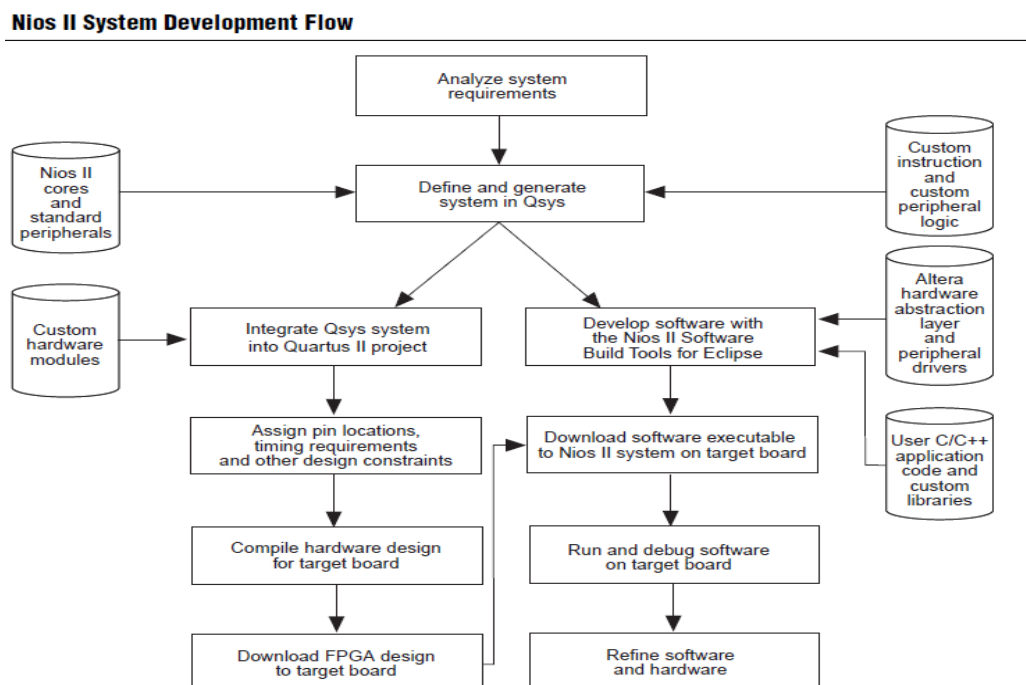

Figure 4 NIOS II Development Flow [4]

Nios II development flow consists of three types of development: software design steps, hardware design steps and system design steps, involving both hardware software and hardware. For simpler Nios II system, one person might perform all steps. For more complex systems, separate software and hardware designers might be responsible for different steps. System design steps involve both the software and hardware, and might require input from both sides. In the case of separate software and hardware teams, it is important to know exactly what information and files must be passed between teams at the points of intersection in the design flow.

2.6.1 Defining and Generating the System in SOPC

After analyzing the system hardware requirements, one can use SOPC to specify the Nios II processor core, memory, and other components your system requires. You can select from a list of standard processor cores and components provided with the Nios II EDS. SOPC automatically generates the interconnect logic to integrate the components in the hardware system. One can also add your own custom hardware to accelerate system performance. One can add custom instruction logic to the Nios II core which accelerates CPU performance or one can add a custom component which offloads tasks from the CPU. This tutorial covers adding standard processor and component cores, and does not cover adding custom logic to the system. The primary outputs of SOPC are the following file types:

• SOPC Design File (.sopc)- Contains the hardware contents of the SOPC system.

• SOPC Information File (.sopcinfo)- Contains a description of the contents of the .sopc file in Extensible Markup Language File (.xml) format. The Nios II IDE uses the .sopcinfo file to create software for the target hardware.

• Hardware description language (HDL) files- Are the hardware design files that describe the SOPC system. The Quartus II software uses the HDL files to compile the overall FPGA design into an SRAM Object File (.sof).

2.6.2 Integrating the Qsys System into the Quartus II Project:

After generating the Nios II system using SOPC Builder, one can integrate it into the Quartus II project. Using the Quartus II software, one can perform all tasks required to create the final FPGA hardware design. Most FPGA designs include logic outside the Nios II system. You can integrate your own custom hardware modules into the FPGA design, or one can integrate other ready-made intellectual property (IP) design modules available from Altera or third party IP providers. This tutorial does not cover adding other logic outside the Nios II system. Using the Quartus II software, one can also assign specify timing requirements, pin locations for I/O signals and apply other design constraints. Finally, one can compile the Quartus II project to produce a .sof to configure the FPGA.

One can download the .sof to the FPGA on the target board using an Altera download cable, such as the USB-Blaster. After configuration, the FPGA behaves as specified by the hardware design, which in this case is a Nios II processor system.

2.6.3 Developing Software with the Nios II IDE:

Using the Nios II IDE, one can perform all software development tasks for your Nios II processor system. After the system is generated with SOPC, one can begin designing your C/C++ application code immediately with the Nios II IDE. Altera provides component drivers and a hardware abstraction layer (HAL) which allows you to write Nios II programs quickly and independently of the low-level hardware details. In addition to your application code, one can design and reuse custom libraries in your Nios II IDE projects. To create a new Nios II C/C++ application project, the Nios II IDE uses information from the .sopcinfo file. You also need the .sof file to configure the FPGA before running and debugging the application project on target hardware.

The Nios II IDE can produce several outputs, listed below. Not all projects require all of these outputs.

• system.h file—Defines symbols for referencing the hardware in the system. The Nios II IDE automatically creates this file when you create a new board support package (BSP).

• Executable and Linking Format File (.elf)- Is the result of compiling a C/C++ application project that you can download directly to the Nios II processor.

• Hexadecimal (Intel-Format) File (.hex)- Contains initialization information for on-chip memories. The Nios II IDE for Eclipse generates these initialization files for on-chip memories that support initialization of contents.

• Flash memory programming data- Is boot code and other arbitrary data you might write to flash memory. The Nios II IDE includes a flash programmer, which allows you to write your program to flash memory. Flash programmer adds appropriate boot code to allow your program to boot from flash memory. One can also use the flash programmer to write arbitrary data to flash memory.

2.6.4 Running and Debugging Software on the Target Board

The Nios II IDE provides complete facilities for downloading software to a target board, and running or debugging the program on hardware. The Nios II IDE debugger allows you to start and stop the processor, step through code, set breakpoints, and analyze variables as the program executes.

2.6.5 Varying the Development Flow

The development flow is not strictly linear. This section describes common variations.

Refining the Software and Hardware

After running software on the target board, one might discover that the Nios II system requires higher performance. In this case, one can return to software design steps to make improvements to the software algorithm. Alternatively, you can return to hardware design steps to add acceleration logic. If the system performs multiple mutually exclusive tasks, you might even decide to use two (or more) Nios II processors that divide the workload and improve the performance of each individual processor.

Iteratively Creating a Nios II System

A common technique for building a complex Nios II system is to start with a simpler SOPC system, and iteratively adding to it. At each iteration, one can verify that the system performs as expected. One might choose to verify the fundamental components of a system, such as the memory, processor and communication channels, before adding more complex components. When developing a custom instruction or a custom component, first integrate the custom logic into a minimal system to verify that it works as one can expect, later one can integrate the custom logic into a more complex system.

2.7 WIRESHARK

Wireshark is an open-source network packet analyzer, which is used for network analysis, troubleshooting, software and communications protocol development and education. Originally it is named Ethereal, in May 2006 the project was renamed Wireshark due to trademark issues.

Wireshark is cross-platform, using the GTK widget toolkit to implement its user interface, and using pcap to capture packets; it runs on various Unix-like operating systems including Linux, BSD, OS X and Solaris, and on Microsoft Windows. There is also a terminal-based (non GUI) version called TShark. Wireshark and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License.

Wireshark is very similar to tcpdump, but has a graphical front-end, plus some integrated sorting and filtering options. Wireshark allows the user to put network interface controllers that support promiscuous mode into that mode, in order to see all traffic visible on that interface, broadcast or multicast traffic and not just traffic addressed to one of the interface's configured addresses. However, when capturing with a packet analyzer in promiscuous mode on a port on a network switch, not all of the traffic traveling through the switch will necessarily be sent to the port on which the capture is

being done, so capturing in promiscuous mode will not necessarily be sufficient to see all traffic on the network. Port mirroring or various network taps extend capture to any point on net; simple passive taps are extremely resistant to malware tampering.

Features:

Wireshark is software that "understands" the structure of different networking protocols. It is able to display the encapsulation and the fields along with their meanings of different packets specified by different networking protocols. Wireshark uses pcap to capture packets, so it can only capture the packets on the types of networks that pcap supports.

Data can be captured "from the wire" from a live network connection or read from a file that recorded already-captured packets.

Live data can be read from a number of types of network, including Ethernet, PPP, IEEE 802.11 and loopback.

Captured network data can be browsed via a GUI, or via the terminal (command line) version of the utility, TShark.

Captured files can be programmatically edited or converted via command-line switches to the "editcap" program.

 Plug-ins can be created for dissecting new protocols. VoIP calls in the captured traffic can be detected. If encoded in a compatible encoding, the media flow can even be played. Raw USB traffic can be captured. Data display can be refined using a display filter.

Capturing raw network traffic from an interface requires elevated privileges on some platforms. For this reason, older versions of Ethereal or Wireshark and tethereal or TShark often ran with superuser privileges. Taking into account the huge number of protocol dissectors that is called when traffic is captured, this can pose a serious security risk given the possibility of a bug in a dissector. Due to the rather large number of vulnerabilities in the past (of which many have allowed remote code execution) and developers' doubts for better future development, OpenBSD removed Ethereal from its ports tree prior to OpenBSD.

Elevated privileges are not needed for all of the operations. For example, an alternative is to run tcpdump or the dumpcap utility that comes with Wireshark, with superuser privileges to capture packets into a file, and later analyze the packets by running Wireshark with restricted privileges. To make near real time analysis, each captured file may be merged by mergecap into growing file

processed by Wireshark. On wireless networks, it is possible to use the Aircrack wireless security tools to capture IEEE 802.11 frames and read the resulting dump files with Wireshark.

2.8 Ethernet frame

A data packet on an Ethernet link is called an Ethernet frame. A frame begins with preamble followed by a start frame delimiter. Following which, each Ethernet frame continues with an Ethernet header featuring destination and source MAC addresses. The middle section of the frame is payload data including any headers for other protocols (e.g. Internet Protocol) carried in the frame. The frame ends with 32-bit cyclic redundancy check which is used to detect any corruption of data in the transit.

2.8.1 Structure

A data packet on the wire is called a frame and consists of binary data. Data on Ethernet is transmitted most-significant octet first. Within each octet, however, the least-significant bit is transmitted first.

The table below shows the complete Ethernet frame, as transmitted, for the payload size up to the MTU of 1500 octets. Some implementations of Gigabit Ethernet (and higher speed Ethernet) support larger frames, known as jumbo frames.

802.3 Ethernet frame structure								
Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)	Interframe gap
7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	42[note 2]–1500 octets	4 octets	12 octets
		← 64–1522 octets →						
	← 72–1530 octets →							
← 84–1542 octets →								

Figure 5 Ethernet Frame [9]

2.8.2 Preamble and start frame delimiter

A frame starts with a 7-octet preamble and 1-octet start frame delimiter (SFD). Prior to Fast Ethernet, the on-the-wire bit pattern for this portion of the frame is as follows 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101011. Since octets are transmitted least-significant bit first the corresponding hexadecimal representation is 0x55 0x55 0x55 0x55 0x55 0x55 0x55 0xD5.

PHY transceiver chips used for Fast Ethernet feature a 4-bit (one nibble) Media Independent Interface. Therefore the preamble will consist of 14 instances of 0x05, and the start frame delimiter

0x5 0xD. Gigabit Ethernet transceiver chips use a Gigabit Media Independent Interface that works 8-bits at a time, and 10Gbit/s PHY works with 32-bits at a time.

2.8.3 Header

The header features destination and source MAC addresses which have 6 octets each, the EtherType protocol identifier field and optional IEEE 802.1Q tag.

2.8.4 EtherType or length

EtherType is a two-octet field in an Ethernet frame. It is used to indicate which protocol is encapsulated in the payload of an Ethernet Frame.

2.8.5 Payload

The minimum payload is 42 octets when 802.1Q tag is present and 46 octets when absent and the maximum payload is 1500 octets. Non-standard jumbo frames allow for larger maximum payload size.

2.8.6 Frame check sequence

The frame check sequence is a 4-octet cyclic redundancy check which allows detection of corrupted data within the entire frame.

2.8.7 Interframe gap

Interframe gap is idle time between frames. After a frame has been sent, transmitters are required to transmit a minimum of 96 bits (12 octets) of idle line state before transmitting the next frame

Chapter 3

METHODOLOGY AND PROBLEM FORMULATION

3.1 AIM:

To generate a system to Generate Ethernet packets in Altera DE2 kit using SOPC builder and drive the Ethernet controller DM9000A to send the packets over LAN.

3.2 Software and Hardware requirements:

1. Quartus II 7.2 Edition Licensed version [Preferred for DE2].

2. NIOS II 7.2 IDE [Integrated Development Environment].

3. Windows XP [Preferred for DE2].

3.3 Procedure:

3.3.1 Assigning the components: Using SOPC Builder in Quartus II, a system is generated with the following components and specifications.

- Clock: Clock of 200MHz External and clk_50 of 50MHz.

- Processor: Nios II/f processor is used for our application. Its specifications used are:

RISC 32 bit with instruction cache, branch prediction, hardware multiply, data cache, dynamic branch prediction

Performance at 200MHz: upto 203 DMIPS

Logic usage: 1400-1800 LEs

Memory usage: Three M4ks + cache

Processor component is shown below:
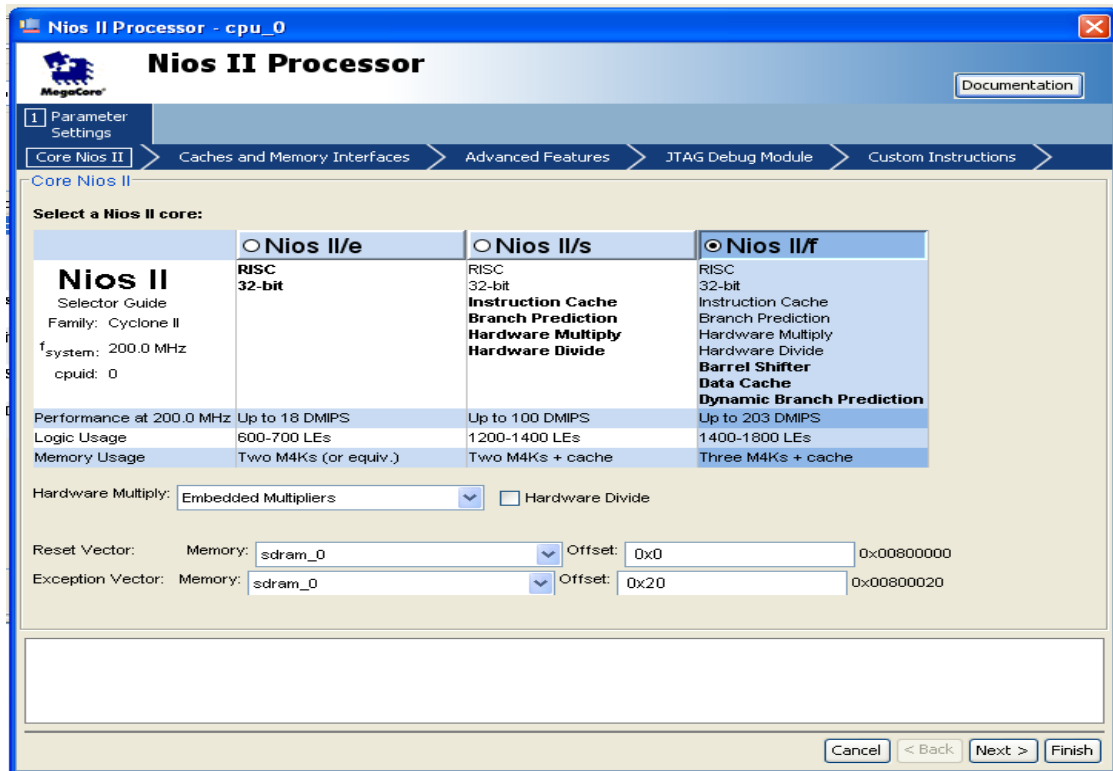
Figure 6

- SD-RAM: component with specifications is shown below:

Figure 7

- JTAG-UART: We wish to connect to a host computer and provide a means for communication between the Nios II system and the host computer. This can be accomplished by instantiating the JTAG UART interface as follows:

Write FIFO:

Depth: 64 bits

Read FIFO:

Depth: 64 bits
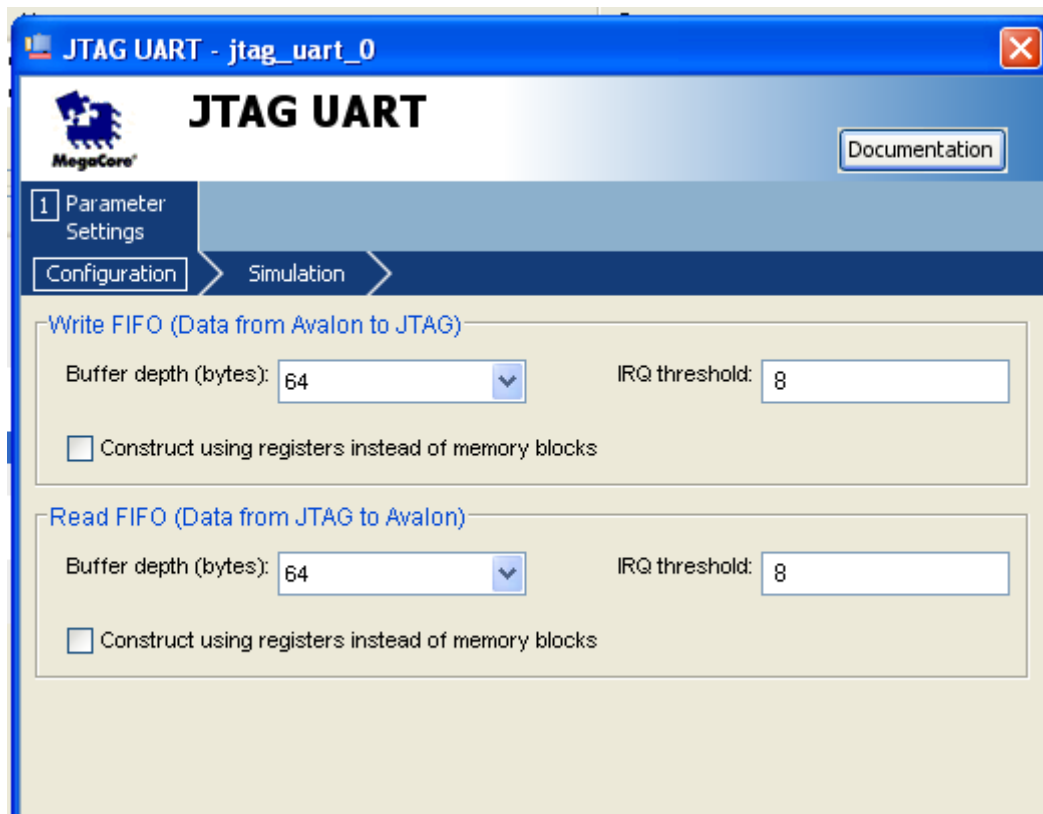
Figure 8

3.3.2 System Generation: Once the components are added we need to assign the base addresses in system->assign base addresses. Then have to generate the system.
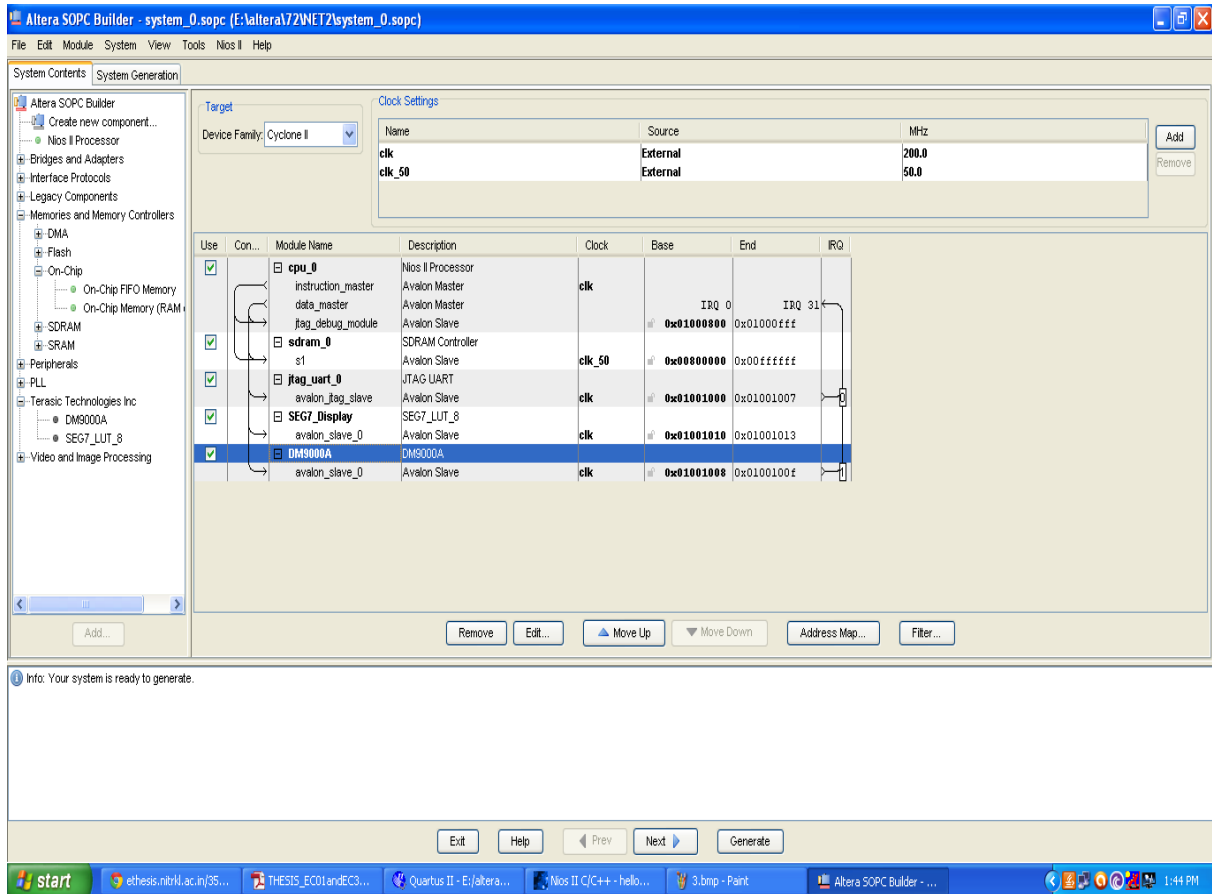
Figure 9.

Having specified all components which are needed to implement the desired system, it can now be generated. Select the System Generation tab. Turn off Simulation – Create simulator project files, because we will not deal with the simulation of hardware. Click **Generate** on the bottom of the SOPC Builder window. The generation process produces the messages displayed in the figure. When the message "SUCCESS: SYSTEM GENERATION COMPLETED" appears.
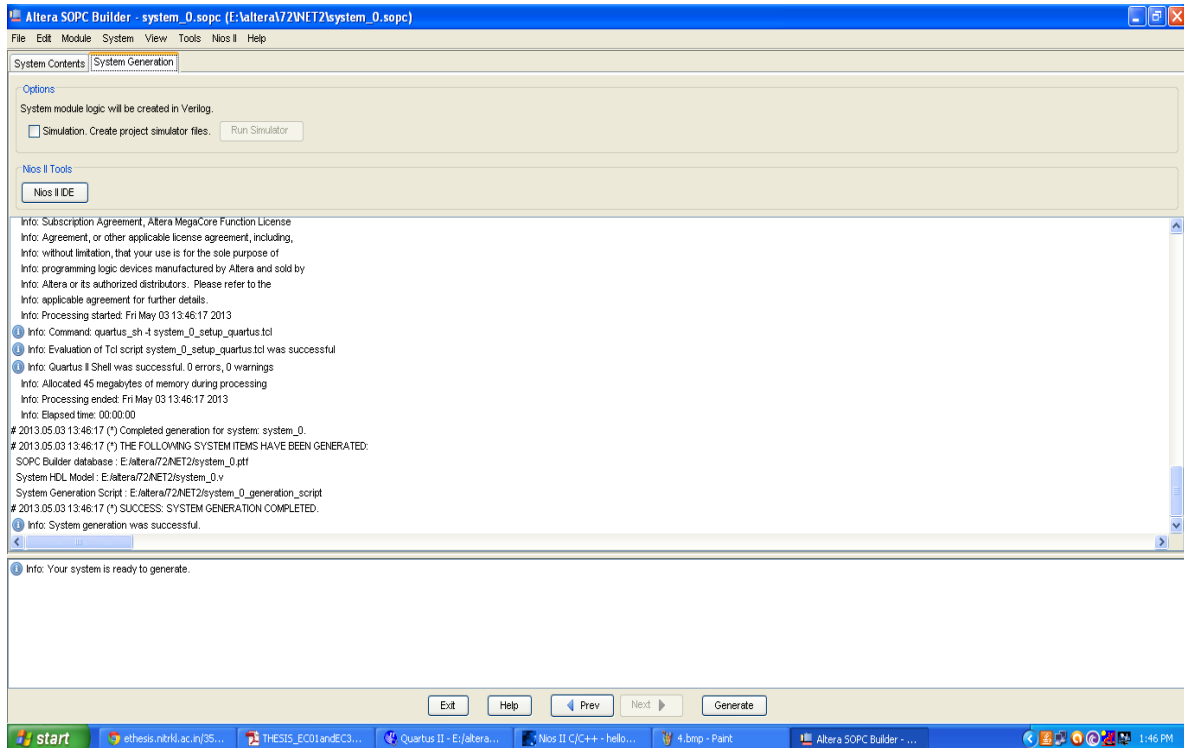
Figure 10

3.3.3 Integration of the Nios II System into a Quartus II Project

To complete the hardware design, we have to perform the following:

• Instantiate the module generated by the SOPC Builder into the Quartus II project.

• Assign the FPGA pins by importing the .csv file for DE2.

• Compile the designed circuit.

• Program and configure the Cyclone II device on the DE2 board.

Instantiation of the Module Generated by the SOPC Builder:

The instantiation of the generated module depends on the design entry method which is chosen for the overall Quartus II project. We have chosen to use VHDL, but the approach is similar for both schematic and Verilog entry methods.

The Nios II module is likely to be a part of a larger design. However, in the case of our design there is no other circuitry needed. All we need to do is instantiate the Nios II system in our top-level VHDL file, and connect the pins of respective modules, as well as the reset inputs and clock, to the appropriate pins on the Cyclone II device.

The code that defines port signals for entity system_0 is shown as:

```
module DE2_NET
        (               Clock Input
                CLOCK_50,                                         //      On Board 50 MHz
                        Push Button
                KEY,                                              //      Push button[3:0]
                        7-SEG Display
                HEX0,           //      Seven Segment Digit 0
                HEX1,           //      Seven Segment Digit 1
                HEX2,           //      Seven Segment Digit 2
                HEX3,           //      Seven Segment Digit 3
                HEX4,           //      Seven Segment Digit 4
                HEX5,           //      Seven Segment Digit 5
                HEX6,           //      Seven Segment Digit 6
                HEX7,           //      Seven Segment Digit 7
                ///////////////////     SDRAM Interface          ///////////////
                DRAM_DQ,        //      SDRAM Data bus 16 Bits
                DRAM_ADDR,      //      SDRAM Address bus 12 Bits
                DRAM_LDQM,      //      SDRAM Low-byte Data Mask
                DRAM_UDQM,      //      SDRAM High-byte Data Mask
                DRAM_WE_N,      //      SDRAM Write Enable
                DRAM_CAS_N ,    //      SDRAM Column Address Strobe
                DRAM_RAS_N,     //      SDRAM Row Address Strobe
                DRAM_CS_N,      //      SDRAM Chip Select
                DRAM_BA_0,      //      SDRAM Bank Address 0
                DRAM_BA_1,      //      SDRAM Bank Address 1
                DRAM_CLK,       //      SDRAM Clock
                DRAM_CKE,       //      SDRAM Clock Enable
                ///////////////////     SRAM Interface           ///////////////
                SRAM_DQ,        //      SRAM Data bus 16 Bits
                SRAM_ADDR,      //      SRAM Address bus 18 Bits
                SRAM_UB_N,      //      SRAM High-byte Data Mask
                SRAM_LB_N,      //      SRAM Low-byte Data Mask
                SRAM_WE_N,      //      SRAM Write Enable
                SRAM_CE_N,      //      SRAM Chip Enable
                SRAM_OE_N,      //      SRAM Output Enable
                /////////////// Ethernet Interface/////////////////////////
                ENET_DATA,      //      DM9000A DATA bus 16Bits
                ENET_CMD,       //      DM9000A Command/Data Select, 0 = Command, 1 = Data
                ENET_CS_N,      //      DM9000A Chip Select
                ENET_WR_N,      //      DM9000A Write
                ENET_RD_N,      //      DM9000A Read
                ENET_RST_N,     //      DM9000A Reset
                ENET_INT,       //      DM9000A Interrupt
                ENET_CLK,       //      DM9000A Clock 25 MHz
        );
///////////////////////// Clock Input              /////////////////////////
input                   CLOCK_50;                                //      On Board 50 MHz
///////////////////////// Push Button              /////////////////////////
input   [3:0]   KEY;                                     //      Pushbutton[3:0]
```

```
///////////////////////  7-SEG Display    ///////////////////////
output   [6:0]   HEX0;                               //       Seven Segment Digit 0
output   [6:0]   HEX1;                               //       Seven Segment Digit 1
output   [6:0]   HEX2;                               //       Seven Segment Digit 2
output   [6:0]   HEX3;                               //       Seven Segment Digit 3
output   [6:0]   HEX4;                               //       Seven Segment Digit 4
output   [6:0]   HEX5;                               //       Seven Segment Digit 5
output   [6:0]   HEX6;                               //       Seven Segment Digit 6
output   [6:0]   HEX7;                               //       Seven Segment Digit 7
///////////////////////           SDRAM Interface      ///////////////////////
inout    [15:0]  DRAM_DQ;       //        SDRAM Data bus 16 Bits
output   [11:0]  DRAM_ADDR;//SDRAM Address bus 12 Bits
output           DRAM_LDQM;//        SDRAM Low-byte Data Mask
output           DRAM_UDQM;//        SDRAM High-byte Data Mask
output           DRAM_WE_N;//SDRAM Write Enable
output           DRAM_CAS_N;//        SDRAM Column Address Strobe
output           DRAM_RAS_N;//        SDRAM Row Address Strobe
output           DRAM_CS_N; //        SDRAM Chip Select
output           DRAM_BA_0; //        SDRAM Bank Address 0
output           DRAM_BA_1; //        SDRAM Bank Address 0
output           DRAM_CLK;     //        SDRAM Clock
output           DRAM_CKE;     //        SDRAM Clock Enable
///////////////////////  SRAM Interface  ///////////////////////
inout    [15:0]  SRAM_DQ;              //        SRAM Data bus 16 Bits
output   [17:0]  SRAM_ADDR;            //        SRAM Address bus 18 Bits
output           SRAM_UB_N; //        SRAM Low-byte Data Mask
output           SRAM_LB_N; //        SRAM High-byte Data Mask
output           SRAM_WE_N; //        SRAM Write Enable
output           SRAM_CE_N; //        SRAM Chip Enable
output           SRAM_OE_N; //        SRAM Output Enable
/////////////////        Ethernet Interface/////////////////////////////
inout    [15:0]  ENET_DATA;    //        DM9000A DATA bus 16Bits
output           ENET_CMD;     //        DM9000A Command/Data Select, 0 = Command, 1 = Data
output           ENET_CS_N;                          //        DM9000A Chip Select
output           ENET_WR_N;                          //        DM9000A Write
output           ENET_RD_N;                          //        DM9000A Read
output           ENET_RST_N;                         //        DM9000A Reset
input            ENET_INT;                           //        DM9000A Interrupt
output           ENET_CLK;                           //        DM9000A Clock 25 MHz


wire     CPU_CLK;
wire     CPU_RESET;
wire     CLK_25;


Reset_Delay     delay1   (.iRST(KEY[0]),.iCLK(CLOCK_50),.oRESET(CPU_RESET));


SDRAM_PLL PLL1          (.inclk0(CLOCK_50),.c0(DRAM_CLK),.c1(CPU_CLK),.c2(CLK_25));


system_0        u0       (
                // 1) global signals:
        .clk(CPU_CLK),
         .clk_50(CLOCK_50),
```

```
            .reset_n(CPU_RESET),

        // the_SEG7_Display
        .oSEG0_from_the_SEG7_Display(HEX0),
        .oSEG1_from_the_SEG7_Display(HEX1),
        .oSEG2_from_the_SEG7_Display(HEX2),
        .oSEG3_from_the_SEG7_Display(HEX3),
        .oSEG4_from_the_SEG7_Display(HEX4),
        .oSEG5_from_the_SEG7_Display(HEX5),
        .oSEG6_from_the_SEG7_Display(HEX6),
        .oSEG7_from_the_SEG7_Display(HEX7),

        // the_DM9000A
        .ENET_CLK_from_the_DM9000A(ENET_CLK),
        .ENET_CMD_from_the_DM9000A(ENET_CMD),
        .ENET_CS_N_from_the_DM9000A(ENET_CS_N),
        .ENET_DATA_to_and_from_the_DM9000A(ENET_DATA),
        .ENET_INT_to_the_DM9000A(ENET_INT),
        .ENET_RD_N_from_the_DM9000A(ENET_RD_N),
        .ENET_RST_N_from_the_DM9000A(ENET_RST_N),
        .ENET_WR_N_from_the_DM9000A(ENET_WR_N),
        .iOSC_50_to_the_DM9000A(CLOCK_50),

        // the_sdram_0
        .zs_addr_from_the_sdram_0(DRAM_ADDR),
        .zs_ba_from_the_sdram_0({DRAM_BA_1,DRAM_BA_0}),
        .zs_cas_n_from_the_sdram_0(DRAM_CAS_N),
        .zs_cke_from_the_sdram_0(DRAM_CKE),
        .zs_cs_n_from_the_sdram_0(DRAM_CS_N),
        .zs_dq_to_and_from_the_sdram_0(DRAM_DQ),
        .zs_dqm_from_the_sdram_0({DRAM_UDQM,DRAM_LDQM}),
        .zs_ras_n_from_the_sdram_0(DRAM_RAS_N),
        .zs_we_n_from_the_sdram_0(DRAM_WE_N)
        );
End module
```

Compiling the generated code along with top level VHDL entity:

Add this file and all the *.vhd files produced by the SOPC Builder to your Quartus II project. Also, add the necessary pin assignments on the DE2 board to your project. Pin assignments are made by importing the assignments given in the file called DE2_pin_assignments.csv in the directory DE2_tutorials\design_files, which is included on the CD-ROM that accompanies the DE2 board and can also be found on Altera's DE2 web pages. Having made the necessary settings compile the code. On compiling this code a SRAM object file(.sof) is generated which is used to program and configure the board.

3.3.4 Programming and Configuration:

Programming and configuring the Cyclone II FPGA in the JTAG programming mode is done as follows:

1. Connect the DE2 board to the host computer by means of a USB cable plugged into the USB-Blaster port. Turn on the power to the DE2 board. Make sure that the RUN/PROG switch is in the RUN position.

2. Select Tools > Programmer

3. If it was not already chosen by default, select JTAG in the Mode box. Also, if the USB-Blaster is not chosen by default, press the Hardware Setup... button and select the USB-Blaster in the window that pops up.

4. The configuration file DE_NET.sof should be listed in the window. If the file was not already listed, then add this file.

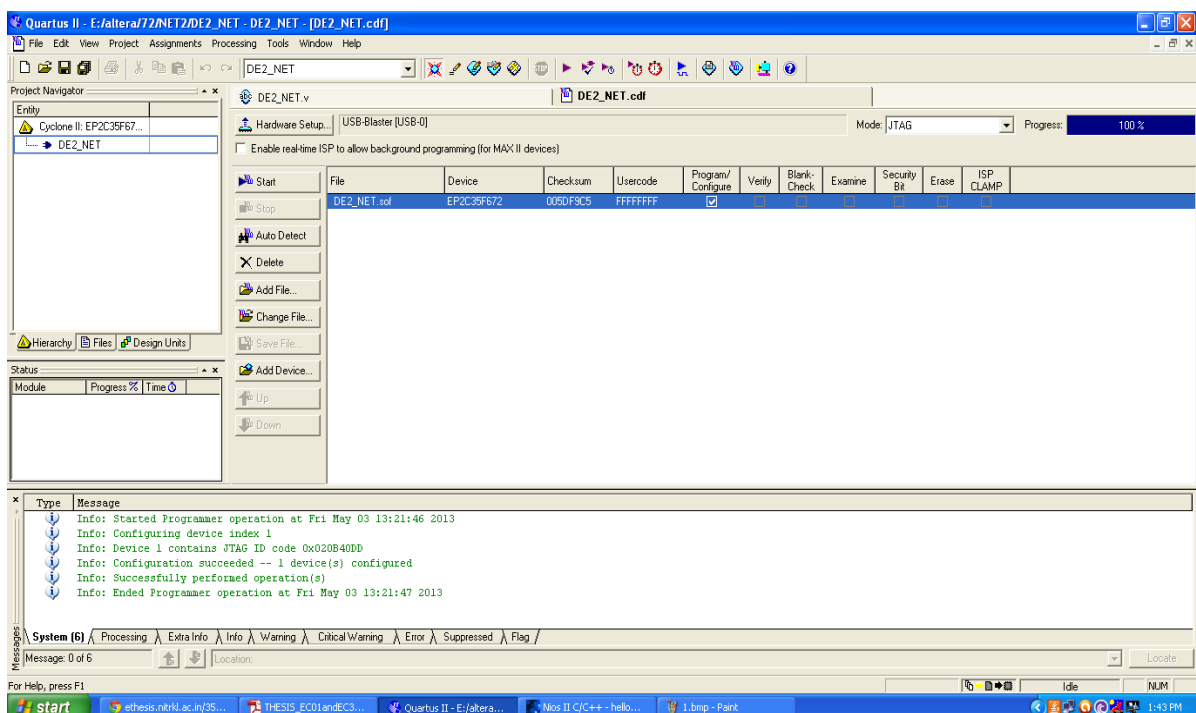5. Click the box under Program/Configure and press start to configure the FPGA.

Figure 11

3.3.5 Running the Application Program using NIOS II IDE:

- Once the Quartus project is compiled, NIOS II IDE is opened and workspace is switched to the one in which Quartus project is made.

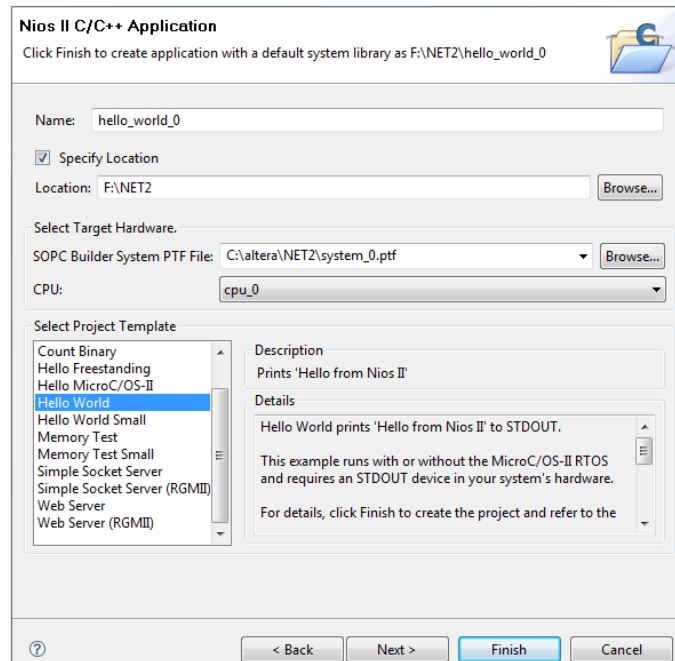- A new C/C++ application project is opened.

Figure 12

- Project location and the SOPC builder system PTF file location are mentioned and finished creating the application.

- Now C program is written to drive DM9000A with the name DM9000A.c.

- Program is written to send Ethernet packets with the name hello_led.c

- Respective header files were defined.

C Code to send Ethernet packets:

```
#include <io.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include "system.h"
#include "DM9000A.C"
unsigned int aaa, i, packet_num, rx_len, counter, times;
unsigned char RXT[80];

unsigned int IPdestination_1,IPdestination_2,IPdestination_3,IPdestination_4;
unsigned int IPsource_1,IPsource_2,IPsource_3,IPsource_4;
unsigned int IPchecksum1,IPchecksum2,IPchecksum3,IPchecksum4,IPchecksum5;
unsigned int Mac_source1, Mac_source2, Mac_source3, Mac_source4, Mac_source5,
Mac_source6;
unsigned int Mac_dest1, Mac_dest2, Mac_dest3, Mac_dest4, Mac_dest5, Mac_dest6;
unsigned int times, lenght_h, lenght_l;
unsigned int flenght, IPlenght_l, IPlenght_h, IPlenght, data_lenght;

void ethernet_interrupts()
{
    packet_num++;
    aaa=ReceivePacket (RXT,&rx_len);
```

```
    if(!aaa)
    {
      printf("\n\nReceive Packet Length = %d",rx_len);
      for(i=0;i<rx_len;i++)
      {
        if(i%8==0)
        printf("\n");
        printf("0x%2X,",RXT[i]);
      }
    }
    outport(SEG7_DISPLAY_BASE,packet_num);
}

int main(void)
{
 IPsource_1 = 0xC0;
 IPsource_2 = 0xA8;
 IPsource_3 = 0x00;
 IPsource_4 = 0x2C;
 IPdestination_1 = 0xC0;
 IPdestination_2 = 0xA8;
 IPdestination_3 = 0x32;
 IPdestination_4 = 0x48;
 Mac_dest1 = 0x00;
 Mac_dest2 = 0x26;
 Mac_dest3 = 0x22;
 Mac_dest4 = 0xB2;
 Mac_dest5 = 0x1D;
 Mac_dest6 = 0x28;
 Mac_source1 = 0x01;
 Mac_source2 = 0x60;
 Mac_source3 = 0x6E;
 Mac_source4 = 0x11;
 Mac_source5 = 0x02;
 Mac_source6 = 0x0F;

 data_lenght = 1468;

 flenght = data_lenght + 0x2E;
 lenght_h = ((data_lenght+8) & 0xFF00)>>8;
 lenght_l = ((data_lenght+8) & 0x00FF);

 IPlenght = data_lenght + 8 + 20;
 IPlenght_h = (IPlenght & 0xFF00)>>8;
 IPlenght_l = (IPlenght & 0x00FF);


 IPchecksum1 = 0x0000C511 + (IPsource_1<<8)+IPsource_2+(IPsource_3<<8)+IPsource_4+

(IPdestination_1<<8)+IPdestination_2+(IPdestination_3<<8)+(IPdestination_4)+
        (IPlenght_h<<8) + IPlenght_l;
 IPchecksum2 = ((IPchecksum1&0x0000FFFF)+(IPchecksum1>>16));
 IPchecksum3 = 0x0000FFFF - IPchecksum2;
 IPchecksum4 = (IPchecksum3 & 0xFF00)>>8;
 IPchecksum5 = (IPchecksum3 & 0x00FF);

 unsigned char SND[flenght], SNDA[flenght];

 unsigned char TXT[] =  { Mac_dest1, Mac_dest2, Mac_dest3, Mac_dest4 ,Mac_dest5,
Mac_dest6,
                          Mac_source1, Mac_source2, Mac_source3, Mac_source4,
Mac_source5, Mac_source6,
                          0x08, 0x00, 0x45, 0x00, IPlenght_h, IPlenght_l,
                          0x00, 0x00, 0x00, 0x00, 0x80, 0x11,
                          IPchecksum4, IPchecksum5, IPsource_1, IPsource_2,
IPsource_3, IPsource_4,
```

```
                           IPdestination_1, IPdestination_2, IPdestination_3,
IPdestination_4, 0x04, 0x00,
                           0x04, 0x00, lenght_h, lenght_l, 0x00, 0x00};

   for (i = 0; i < 42; i++)
    SND[i] = TXT[i];

   for (i = 42; i < flenght-4; i++)
    SND[i] = i-42;

   SND[i++] = 0x35;
   SND[i++] = 0x15;
   SND[i++] = 0xF0;
   SND[i++] = 0x13;

   for (i = 0; i < 42; i++)
    SNDA[i] = TXT[i];

   for (i = 42; i < flenght-4; i++)
    SNDA[i] = i-42;

   SNDA[i++] = 0x55;
   SNDA[i++] = 0x45;
   SNDA[i++] = 0x30;
   SNDA[i++] = 0x12;

 DM9000_init();
 alt_irq_register( DM9000A_IRQ, NULL, (void*)ethernet_interrupts );
 packet_num=0;
 while (1)
 {
       if(RXT[rx_len-1]==0)
   TransmitPacket(SND,flenght);
   else  TransmitPacket(SNDA,flenght);
   times++;
   IOWR(SEG7_DISPLAY_BASE, 0, times);
 }

    return 0;
```

- Once we write the code the project is built and .elf is generated

- Once the built is complete. We can run the program using NIOS II hardware. To do so right click on the project folder-> run as-> NIOS II hardware.

3.3.6 Connecting the Ethernet cable from FPGA to the PC over Homeplug adapters:

- Once the program is run we can find the Ethernet port glowing and the 7SEG display counting the no. of packets transmitted.

- In the PC transmitted packets are captured using Wireshark.

<div align="right">

Chapter 4

RESULTS AND DISCUSSIONS

</div>

RESULTS AND DISCUSSIONS

✓ A NIOS II system is generated using SOPC builder to generate Ethernet packets.

✓ NIOS II system which is generated is integrated into Quartus II project. Pin assignments were done and the project was compiled and the design was downloaded onto the FPGA.

✓ Simultaneously software was developed using NIOS II IDE and .elf file was generated and downloaded onto the board. Once the project was built. It was run.
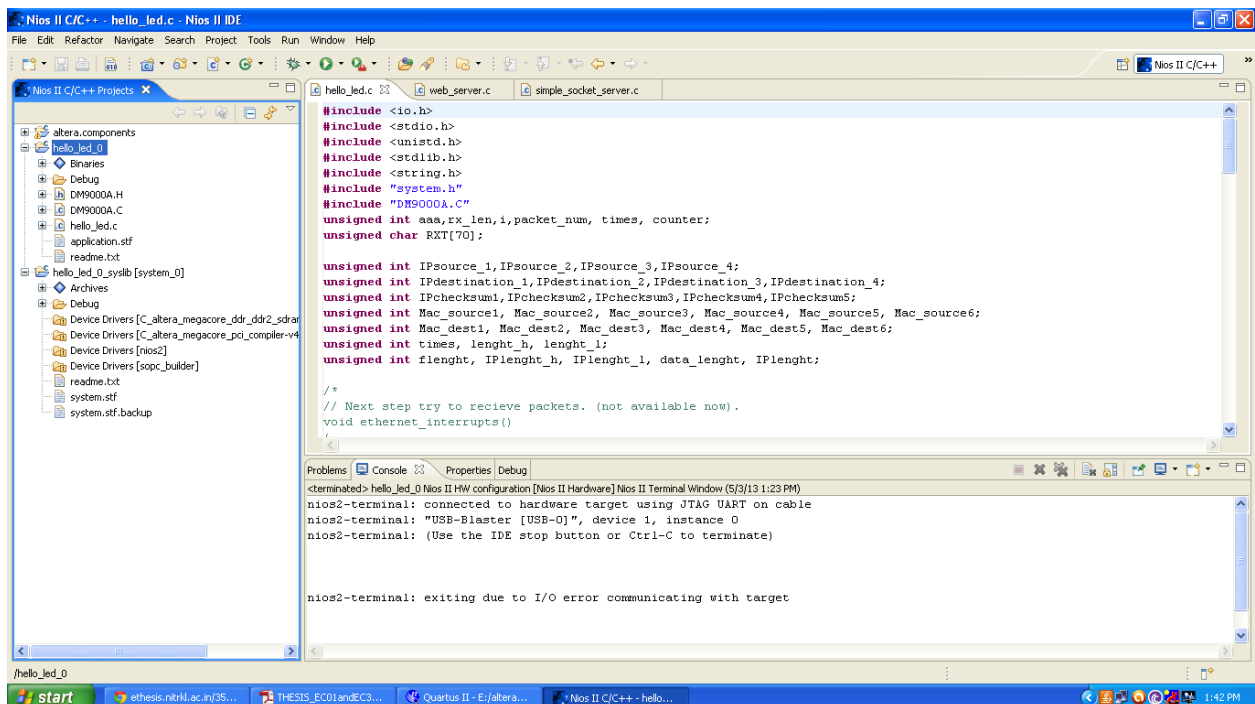

Figure 13

✓ The transmission was done over Ethernet and then over power line and the packets were captured using wireshark. The packets that were captured are analyzed and found to have the same data which was sent.
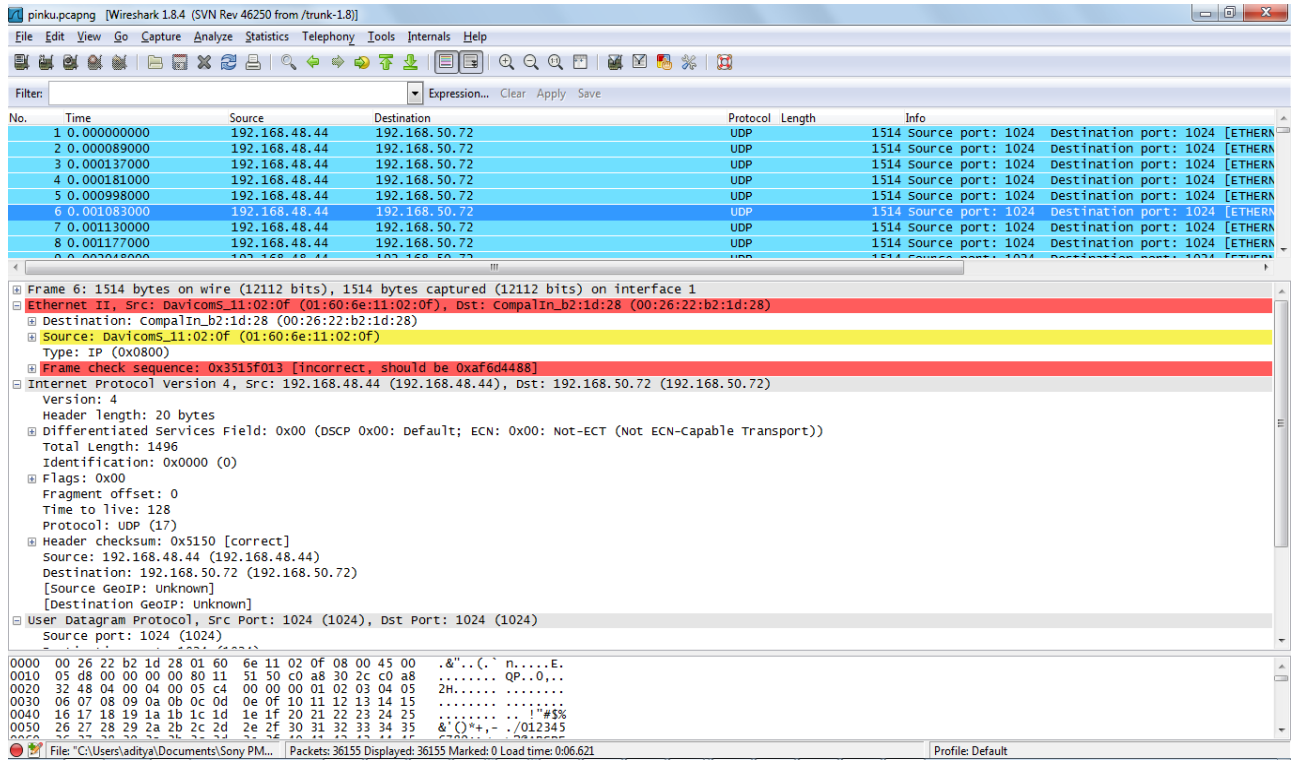
Figure 14

<div align="right">

Chapter 5

CONCLUSION

</div>

5.1 CONCLUSION

The Nios II system was generated and configured in the board successfully. This system can now be developed for any desired application. A system was developed for a transmission of Ethernet packets. The system was developed to be used as web server by editing the top level VHDL entity and running a suitable application program in the system. An application program was run in NIOS II to send Ethernet packets. Packets were captured and analyzed.

5.2 FUTURE SCOPE

- The system generated can be used for a web server for various applications.

- It can be used for transmission of automated messages.

- It can be used in Powerline communication applications to control devices with Ethernet connectivity.

5.3 LIMITATIONS

The major limitations in the project were the following:

1. .SOF file can be generated only if the Quartus II software is licensed.

2. DE 2 Altera kit works preferably in Windows XP SP3 and with NIOS II 7.2 IDE.

3. The code that was used to 'ping' the Altera kit was unsuccessful as integrating it with the one used for transmitting packets could not be done. The reason being the incompatibility of the socket program with Nios II version 7.2.

REFERENCES

[1] Homeplug 1.0 Technology White Paper, www.homeplug.org.

[2] HomePlug 1.0 Specifications

[3] Bingham, J.A.C., "Multicarrier Modulation for Data Transmission: An Idea Whose Time Has Come," IEEE Communications Magazine, pp. 5-14, May, 1990

[4] NIOS II Hardware Development Tutorial, www.altera.com.

[5] NicheStack TCP/IP stack- NIOS II Edition Tutorial.

[6] Ore Baugh, Angela; Ramirez, Gilbert; Beale, Jay (February 14, 2007). Wireshark & Ethereal Network Protocol Analyzer Toolkit. Syngress. p. 448. ISBN 1-59749-073-3

[7] www.wireshark.org->wireshark users guide.

[8] www.altera.com-> My first NIOS II tutorial.

[9] IEEE 802.3 Ethernet standards.