Embedded Network Firewall on FPGA

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the degree of Master of Science

In the Department of Electrical and Computer Engineering

University of Saskatchewan

Saskatoon, Saskatchewan

Canada

By

Raouf Ajami

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Electrical and Computer Engineering
57 Campus Drive
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

The Internet has profoundly changed today's human being life. A variety of information and online services are offered by various companies and organizations via the Internet. Although these services have substantially improved the quality of life, at the same time they have brought new challenges and difficulties. The information security can be easily tampered by many threats from attackers for different purposes. A catastrophe event can happen when a computer or a computer network is exposed to the Internet without any security protection and an attacker can compromise the computer or the network resources for destructive intention.

The security issues can be mitigated by setting up a firewall between the inside network and the outside world. A firewall is a software or hardware network device used to enforce the security policy to the inbound and outbound network traffic, either installed on a single host or a network gateway. A packet filtering firewall controls the header field in each network data packet based on its configuration and permits or denies the data passing thorough the network.

The objective of this thesis is to design a highly customizable hardware packet filtering firewall to be embedded on a network gateway. This firewall has the ability to process the data packets based on: source and destination TCP/UDP port number, source and destination IP address range, source MAC address and combination of source IP address and destination port number. It is capable of accepting configuration changes in real time. An Altera FPGA platform has been used for implementing and evaluating the network firewall.

Acknowledgements

I would like to sincerely appreciate my supervisor Dr. Anh V. Dinh for his advice, support and patience throughout this thesis work.

A special thanks to my friend, Vivek Joshi, for his help, guidance and suggestions during the course of this thesis writing.

Finally, I would like to express gratitude to my lovely wife, Mahasti, for encouragement and help during my studies. Many thanks to my parents for their support in my education.

Contents

List of Tables

List of Figures

List of Abbreviations

ACK	Acknowledge
AMBA	Advanced Micro-controller Bus Architecture
API	Application Programming Interface
ARP	Address Resolution Protocol
ASIC	Application-Specific Integrated Circuit
BSP	Board Support Package
CAM	Content Associative Memory
CSMA/CD	Carrier Sense Multiple Access With Collision Detection
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DARPA	Defense Advanced Research Projects Agency
DoD	Department of Defense
DMZ	Demilitarized Zone
DNS	Domain Name System
EDS	Embedded Development Suites
ENF	Embedded Network Firewall
EPH	Ethernet Protocol Handler
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FTP	File Transfer Protocol
HAL	Hardware Abstraction Layer
HDL	Hardware Description Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol

ISO	International Organization for Standardization
ISR	Interrupt Service Register
LAN	Local Area Network
MAC	Media Access Control
NFM	Network Firewall Module
OS	Operating System
OSI	Open Systems Interconnection
PC	Personal Computer
PIO	Parallel Input/Output
POP3	Post Office Protocol
QoS	Quality of Service
RISC	Reduced Instruction Set Computing
RTOS	Real-Time Operating System
SAN	Storage Area Network
SCTP	Stream Control Transmission Protocol
SMTP	Simple Mail Transfer Protocol
SoC	System on Chip
SOPC	System On a Programmable Chip
TCP	Transmission Control Packet
TCP/IP	Transmission Control Packet / Internet Protocol
ToS	Type of Service
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
WAN	Wide Area Network
WWW	World Wide Web

Chapter 1

Introduction

This chapter provides an introduction to network security in computer networking communication. The first section gives the motivation behind this research and why the data protection is very important and vital in today's world of computing. In the next section, the objective of the research is explained and finally, the thesis outline is identified in the last section.

1.1 Motivation

Internet has indeed made it effortless for commercial world to do business on the web. This rapid E-commerce expansion has often neglected the security risks that threatens the Internet. Significant number of online applications, such as email clients, web services and database systems supplied by companies' I.T departments, provide services to public which attackers desire to access for destructive purposes. Password attacks, identity theft, business and personal data loss, information manipulation and data corruption are only a glance of damages that are in news headlines. A network administrator or security professional's job is to ensure the safety and reliability of a network from malicious and destructive attacks from the network and network-accessible resources. The security staff need to know how these attacks are being carried and how they can be stopped. Table 1.1 shows the estimated cost of recovery for companies and organizations due to malware attacks [1].

From 2004 onward, the loss declined due to increasing and improving network

Table 1.1: Estimation of malware financial impact [1]

Year	**Worldwide impact (US $)**
2006	$13.3 Billion
2005	$14.2 Billion
2004	$17.5 Billion
2003	$13.0 Billion
2002	$11.1 Billion
2001	$13.2 Billion
2000	$17.1 Billion
1999	$13.0 Billion
1998	$6.1 Billion
1997	$3.3 Billion
1996	$1.8 Billion
1995	$0.5 Billion

security infrastructure. The growth in network security is the result of research and costly technology the has been involved for hardening the network [6].

A highly effective solution, to protect the internal network from the outside world, is placing a network firewall where the internal network connects to the Internet Firewalls [7]. Basically a firewall is a part of computer networking equipments designed to restrict unauthorized or unwanted access between sections of a network while permitting authorized communication. It can be configured based upon a set of rules and policies in the organization to permit or deny the data passing through; “nothing can be trusted until it’s tested”. A firewall can greatly reduce attacks to internal resources and overall elevates the site security. Without a firewall, network security can be out of control and practically the damage can be brutal. Although it is possible for an administrator to use available individual software based firewall for each computer in a large size network, but that can be a difficult and time consuming procedure. In reality, it is way more efficient to have a network firewall at gateway of the site rather than protecting each single computer in the network. To a large extent, a network firewall is more powerful and manageable than its smaller counterpart.

1.2 Research objective

The objective of this research is to design an embedded customizable hardware network firewall that can be used on a network gateway (i.e., network router or broadband modem) instead of a dedicated firewall device in order to improve the speed and features accomplished in the other researches in [8], [9].

Cheng in [8] has designed a source Internet Protocol (IP) address based firewall on FPGA that is capable of improving the speed compared to a software based firewall. However, this firewall can only operate in source IP address mode and is not configurable after it is initialized. Laturnas in [9] has improved the FPGA based firewall by speeding up the packet processing time and adding dynamic Content Associative Memory (CAM) expiration, however the firewall still operates in a single mode and can not be configured in real time.

The high speed packet processing firewall, designed in this research, can be configured in different modes and has the ability to accept and apply configuration changes in real-time along with speed improvement compared to [8],[9]. The modes, that this firewall can operate on, are based on header information in Link, Internet and Transport layers of the Transmission Control Protocol/ Internet Protocol (TCP/IP) protocol stack. The following six options are the different modes implemented in the firewall designed in this research:

- TCP/UDP destination port number
- TCP/UDP source port number
- Source IP address
- Destination IP address

- Source Media Access Control (MAC) address
- Combination of source IP address and destination port number

This firewall can be used for various purposes in a network with several applications. The source or destination port filtering is the useful feature for controlling the traffic to a dedicated server. For example, the firewall can be configured and placed in Demilitarized Zone (DMZ) to only permit TCP port 80 and TCP port 443 packets towards a web server which serves up Hyper Text Transfer Protocol (HTTP) and HTTP Secure (HTTPS) requests from users. Source or destination IP address would be beneficial in an organization that would like to permit or deny access for known IP address or a range of IP addresses. There are occasions that a compromised PC inside of a network has to be blocked until further investigation. Source MAC address can be used in this scenario to deny access of that PC to the Internet or other part of the network.

The main block has been built using Verilog Hardware Description Language (HDL) in order to speed up the packet processing. A processor based embedded system with real-time operating system has been designed to achieve highly customized and on the fly configuration change in the firewall. Also CAM has been used to improve performance of packet matching. The whole design has been implemented and evaluated on an Altera Field Programmable Gate Arrays (FPGA) device.

1.3 Thesis outline

The key functionalities of the computer network and the network security are given in Chapter 2. Hardware and software components of this design are described in Chapter 3. Chapter 4 focuses on testing and evaluation of the firewall and finally Chapter 5 provides summary of the research, conclusions and suggestions for future work.

Chapter 2
Background

In this chapter, background information of the computer network, network security and embedded system are presented. The first section discusses fundamentals of networking, specifically TCP/IP networking. The second part talks about network security and different types of risks that are involved, and in the final section, embedded system and FPGA technology are described.

2.1 Network security

Corporate networks are valuable assets. While being exposed to the Internet, the network needs to be protected from security threats. A network security manager needs to know what type of risks the network faces and how to defend it from these risks. In the following points, some of the main risks that threat data (information in the computers) and resources (computers themselves) are discussed.

- **Viruses, Worms, Trojans, and Malware**: Viruses, Worms, Trojans, and Malware are malicious code that are written to spread by themselves, and infect a victim's computer, in order to damage the data or take control of the computer [10]. All of them act in variety of ranges from just annoying the user (pop up messages) to very devastating purpose, such as erasing data and software from the computer. These threats are very difficult to protect through firewall, so scanning software needs to be installed on computers along with other security measures.

- **Intrusion**: This is the most popular type of attack where an attacker wishes to take control of the computers in a network. Attackers have many methods to intrude, ranging from simply guessing the user name and password of the personnels in company to complicated methods of *social engineering* [7], [10]. Social engineering attack means that attacker pretends to be someone else in a company, for example trying to ask for help from network administrator to reset his or her password and then with new password, the attacker gets access to computer and other resources in the network. Firewall can not fully protect a network from such intrusion but there are some ways to limit the number of accounts accessing the network from the outside world.

- **Denial of service (DoS)**: The DoS attack occurs when a computer service crashes and can not be accessed by anyone. Basically, the attacker tries to prevent users from using their own resources on the network. This attack can happen by flooding a server with a large number of service requests which ends up swamping the buffer of the server. The server tries to respond to these *Spam* requests but can not cope and as a result it goes down. This type of threat is also very difficult to avoid if a company provides general services, like Email, to outside users. But a well configured firewall can prevent *DoS* to reach critical servers [7].

- **Information theft**: As obvious from its name, in this type of intrusions, the attacker wants access to the personal information, like peoples' financial information. In this method, the attacker creates an Internet service to capture personal information details and gives that information away, or sells it to unauthorized people. By accessing this information, the attacker can use or sell the data for many illegal purposes. To give an example, compromising an insurance company network could give many confidential information to an attacker. *Packet or network sniffing* is a very popular and successful method among the attackers to find out about what information is passing through the Internet to the destination [11], [12], [13]. A properly configured firewall can help isolate critical servers from unauthorized access.

2.1.1 Security model

There are two main models to shield against attacks discussed in previous pages:

- **Host security**: In this model each single host or computer has to protect itself using a personal firewall, anti-virus, anti-malware and anti-spyware software but it is not a simple task. Today, there are different hosts and operating systems from various vendors which make it difficult for the network administrator to install, update, control and manage all individual hosts on a network. Also, users need some background in information technology and computer security in order to be aware of threats and ways they can protect their resources. This method can be useful in *Small Office Home Office (SOHO)* environment but in a larger network environment these protections are insufficient.

- **Network security**: This model can be very effective in most networks, specifically in large network environment with hundreds or thousands of hosts. This method concentrates on the attack protection over the main *network gateway*, that is connected to the outside world, rather than each single host in the network. This is where firewall enters the battle!

2.1.2 Firewall and traffic control

A firewall is a hardware or software (or combination of both) device on the network that is placed between network segments. It controls the connections that are being made between hosts as well as data flow based on rules and policies defined. Usually the firewall connects the internal network to the Internet and controls the data traffic so that only authorized data would pass through. The firewall inspects the data packet and processes the header information of each packet and determines whether the packet should be passed or dropped according to its policy configuration. This is also called *packet filtering* and it falls under the following categories:

- Source port
- Destination port
- Protocol type
- Source IP address
- Destination IP address
- Source MAC address
- Destination MAC address

For example in Figure 2.1, TCP port 80 is allowed access to the internal network while the TCP port 445 is not allowed.

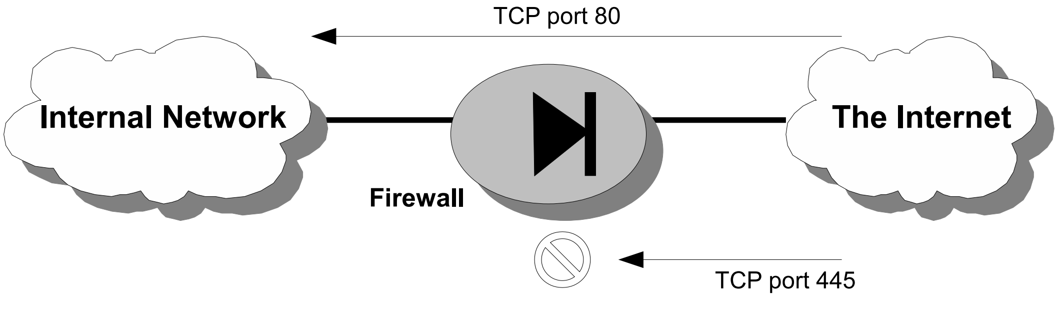


Figure 2.1: A firewall is setup to pass only TCP port 80

In the next section, we are going to investigate computer network technology and find out how firewalls work.

2.2 Computer networking

A network is a collection of computers and devices connected together in order to communicate with each other and share resources [14]. There are several types of commonly used networks by scale :

- Local Area Network (LAN): The most common type of network for physically close computers and devices to each other.

- Wide Area Network (WAN): A network that connects very long distant computers together.

- Internetworking: Connects two or more private networks via separate infrastructure. *The Internet* is short for inter-networking and is the biggest network entity that is aggregation of many inter networks.

When computers share a network and want to communicate, they must follow certain rules to understand their language or *protocol* so to speak. Basically, the protocol defines how the communication occurs [15]. To develop and maintain a network as simple as possible, modular concept is used for exchanging data between networking components [14]. It is much easier to break down a big task to some smaller tasks; each module is capable of a single task or related small functions and communicates with other modules. Each individual module does not know the details of how the rest of the modules do their jobs. This method makes each software or hardware module isolated and independent from each other, which means when modification is required, it can be done with least interference. The modular approach has the following benefits:

- The complex network process is segmented into uncomplicated components.

- Changes in a particular module does not impact the overall process.

- The standardization and integration between modules can be easier among the vendors.

There are two major networking models, The Open Systems Interconnection (OSI) model and the Internet or TCP/IP model. The OSI model, a product of the *International Organization for Standardization (ISO)* [16], sub-divides a networking process into seven smaller sections called layers. Each layer provides services for the direct upper layer and receives services from the immediate lower layer. These layers from the top (closer to the user) to the bottom, are as follows [16]:

- **The application layer**: Provides data communication between user and network.
- **The presentation layer**: Provides data translating and formatting from what user understands to what network understands.
- **The session layer**: Defines functions for starting, controlling and ending communication between hosts in a network.
- **The transport layer**: Provides data transport management and link reliability which ensures the error correction and flow control by segmentation and reassembly.
- **The network layer**: Defines logical addressing, routing and packet delivery.
- **The data link layer**: Responsible for physical addressing and transferring data over physical link.
- **The physical layer**: Defines specification for physical link.

The *Defense Advanced Research Projects Agency (DARPA)*, a scientific agency of the United States *Department of Defense (DoD)* introduced the Internet or TCP/IP model during 1970s [17]. The main reason behind creating this model was a need for a robust network that could survive from any disaster such as nuclear war. This model includes of four layers concept [18]:

- **The application layer**: This layer defines high level protocols to represent, encode and control sessions. To name a few, HTTP, Domain Name System (DNS) and File Transport Protocol (FTP) function are in the application layer.

- **The transport layer**: The transport layer provides host to host connection establishment and handles flow control. TCP and UDP act in this layer and will be discussed in detail later on.

- **The Internet layer**: This layer is responsible for logical addressing and routing data across the networks. The main protocol at this layer is the Internet protocol.

- **The network access layer**: The network access layer handles physical connection to the network medium. Ethernet , Address Resolution Protocol (ARP) and Frame Relay are some protocols that can be found in this layer.

In Figure 2.2, a visual comparison of the OSI and Internet model can be observed. As illustrated in Figure 2.2, the application layer covers the functionality of the application, presentation and session layers of the OSI model. Also, data link and physical layer of the OSI model are correspond to the network access layer of the Internet model.

In both models, each layer adds the appropriate information to the data , called header, and sends it to the lower layer at the transmitting host. This process is called *encapsulation*. The reverse process takes place at the receiving host where the header gets extracted and data gets passed to the upper layer. Encapsulation makes it possible for each layer to logically communicate directly with its correspondent. Therefore, from the point of view of each layer, there is no overhead information attached to the data. Figure 2.3 shows the encapsulation in each layer. It would be

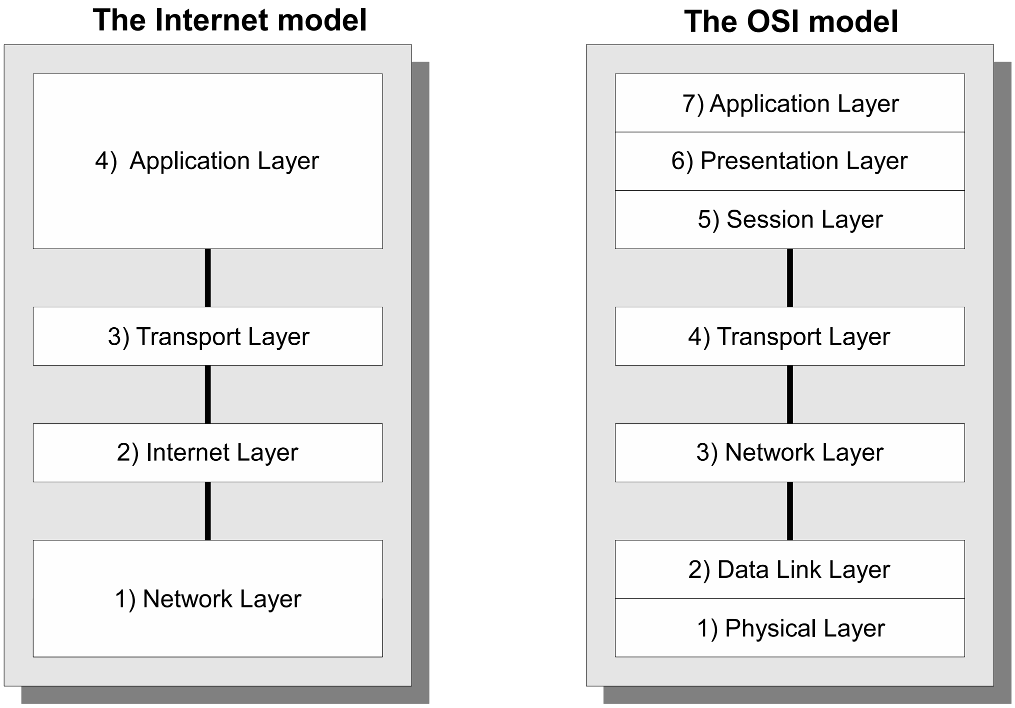


Figure 2.2: OSI and Internet model comparison

necessary to mention that, the data wrapped with transport header in the transport layer is called *segment*; segment in network layer is encapsulated with network layer header which creates a *packet*. Packet is encapsulated in *frame* in data link layer and finally frame is passed to physical layer in a bit format (0s and 1s). Combination of layers is called a *stack*. As [14] explains, the OSI model could not be adopted among the vendors because of timing, technology, implementation, and politics problem. The TCP/IP model became very popular; although, the base concept of networking comes from the OSI model.

2.2.1 Transport layer

Two important protocols for data transfer are found at the transport layer in the Internet model are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

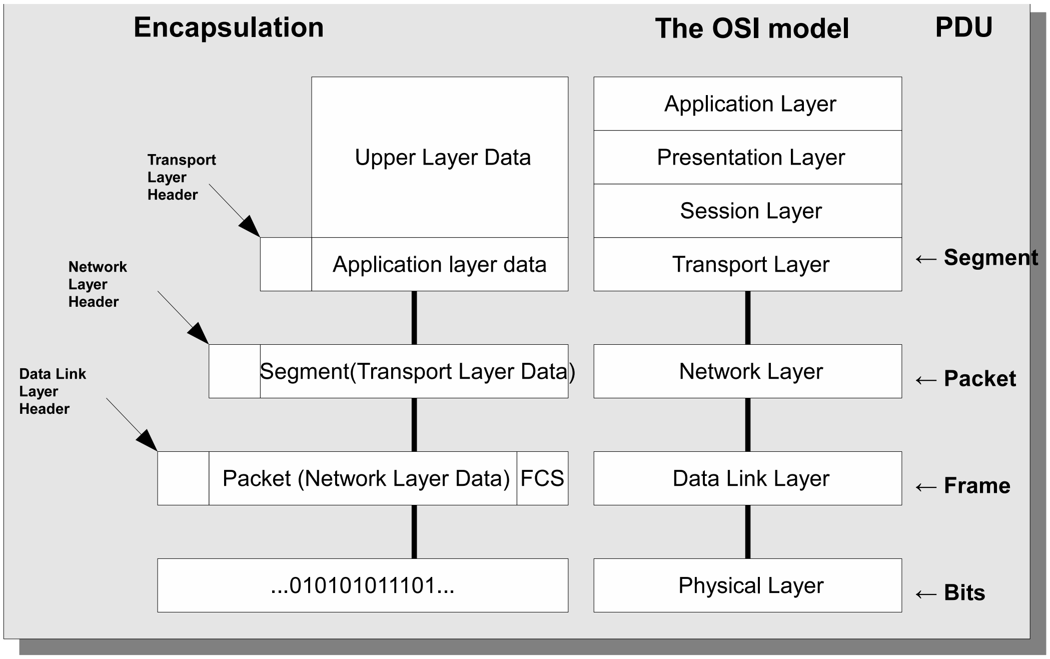


Figure 2.3: Data encapsulation

Transmission Control Protocol (TCP)

TCP is a connection-oriented transport protocol at the transport layer of TCP/IP model which guaranties data delivery between hosts by creating a virtual circuit and three-way handshake mechanism with transport layer on the other side of the connection. Since Internet layer (IP protocol) does not provide any error detection or data recovery, TCP is an efficient way of communication over unreliable medium. Figure 2.4 shows the details of a TCP segment with the following fields:

- **Source port number (16 bit)**: This number represents the application that data is coming from.

- **Destination port number (16 bit)**: This number represents the application that the data is going to.

- **Sequence number (32 bit)**: Necessary information regarding of the order of the data that is transmitting.

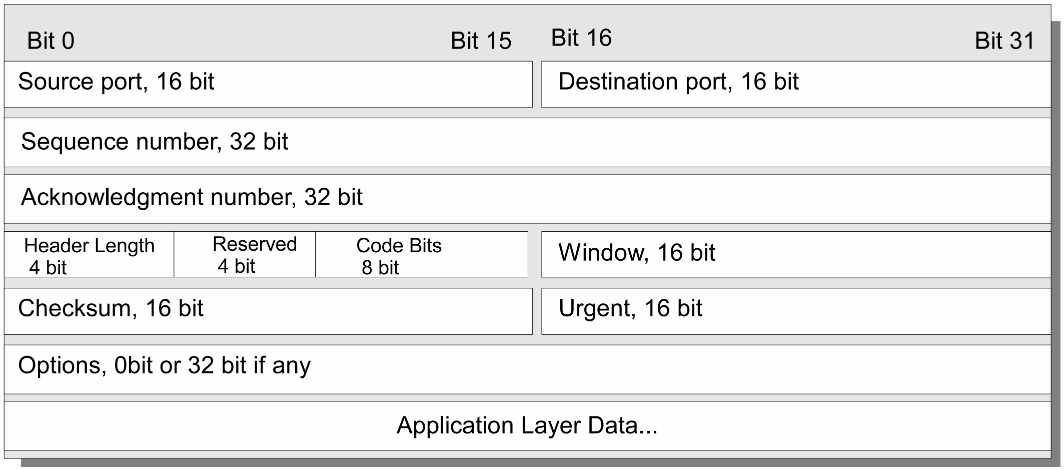


Figure 2.4: Inside of a TCP segment

- **Acknowledgment number (32 bit)**: Shows the next segment that has to be received in the receiver side.

- **Header length (4 bit)**: Total number of bytes in the header field and data.

- **Reserved (4 bit)**: Reserve for future use and set to 0.

- **Code bits (8 bit)**: This field indicates the control bits for session establishment.

- **Window (16 bit)**: Specifies number of bytes to be received before an acknowledgment (ACK) is sent.

- **Checksum (16 bit)**: Shows the calculated checksum of the header field plus data.

- **Urgent (16 bit)**: Urgent bit in the code bits controls this field to show the end of urgent data.

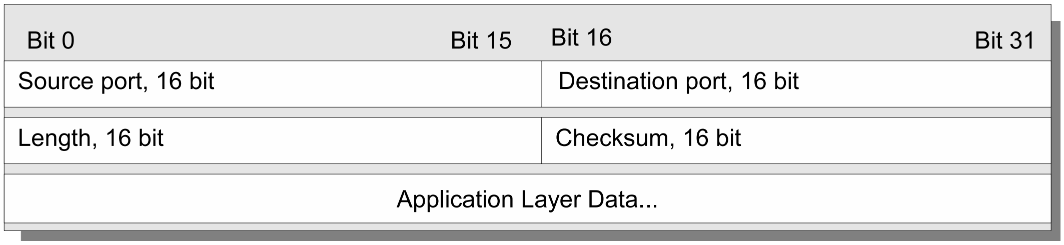
- **Options (variable bit)**: Barely used with some TCP hosts to determine the TCP segment size

- **Data**: Application layer data.

Port number in TCP header indicates which application or service (in application layer), the data is sourced from and which application or service it is destined for. Applications and protocols such as FTP, HTTP and SMTP require high reliability medium between server and client and TCP protocol provides trustworthy data transmission.

User Datagram Protocol (UDP)

UDP is a connectionless and fairly simple protocol compared to TCP at transport layer of the TCP/IP model. UDP does not transfer data as reliable as TCP because in most of today's networking environment, latency and packet loss is very minimal. The UDP protocol was designed to be fast and simple with less overhead. Although UDP communication is prone to errors, there are many applications and services that can tolerate this problem depending on application layer reliability. Figure 2.5 is inside of a UDP segment with the following fields:



Bit 0 Bit 15	Bit 16 Bit 31
Source port, 16 bit	Destination port, 16 bit
Length, 16 bit	Checksum, 16 bit
Application Layer Data...	

Figure 2.5: Inside of a UDP segment

- **Source port number (16 bit)**: This number represents the application that the sending host (source) is using.

- **Destination port number (16 bit)**: This number represents the application that the receiving host (destination) is using.

- **Length (16 bit)**: Total number of bytes in header field and data.
- **Checksum (16 bit)**: Shows the calculated checksum to allow the receiver to know if the segment is corrupted.
- **Data**: Application layer data.

UDP protocol does not need to establish a session before communication begins and does not verify whether the data is received by the recipient (Best effort delivery). This makes it a good choice for the applications and services that verify the data integrity in *application layer*. DNS, Trivial File Transfer Protocol (TFTP) and Simple Network Management Protocol (SNMP) are few examples that utilize this protocol. Like TCP protocol, port numbers are used by *application layer* to know which data is coming from which application and where it is going to.

2.2.2 Internet layer

The Internet protocol (IP) is the main protocol in the Internet layer of the TCP/IP model. IP is a connectionless protocol which provides data delivery over the best path in the network using routing protocols. IP is not a reliable protocol and leaves any sort of packet delivery issue such as packet loss, out of order, delayed and duplicated packet to other layers (upper or lower layer). IP does not establish an end to end connection and each packet is sent individually therefore delay or out of order packet delivery could be the result of packets routed over different path. Although this concept may seem very undetermined, there are other protocols in other layers that are capable of fixing this issue and reliably transmit the data. The current IP version 4 is under transition to IP version 6 with many improvements and new features such as 128 bits IP address field instead of 32 bits (in IPv4) providing vastly larger IP address space. Figure 2.6 illustrates the internals of an IP version 4 (IPv4) packet. An IP packet includes the following fields in its header:

- **Version (4 bit)**: Identifies the format of the IP packet.
- **Internet header length (4 bit)**: Represents the length of header.
- **Differentiated services field (6 bit)**: Or also known as *Type of Service (ToS)* indicates the packet level important to be used in *Quality of Service (QoS)*.
- **Explicit Congestion Notification (ECN, 2 bit)**: Represents the network congestion without dropping packets if both hosts support.
- **Total length (16 bit)**: Represents the total length of the packet including the header and data.
- **Identification (16 bit)**: Same as sequence number in TCP packet.
- **Flags (3 bit)**: Indicates if the packet can be fragmented or not.
- **Fragment offset (13 bit)**: Shows the location of the packet in datagram.
- **Time to live (8 bit)**: Indicates the remaining time of the packet before it is dropped.
- **Protocols (8 bit)**: Represents the type of protocol in higher layer that produced the packet.
- **Header checksum (16 bit)**: Shows the calculated checksum for header only.
- **Source address (32 bit)**: Represents the source IP address (logical address) of the sending host.
- **Destination address (32 bit)**: Represents the destination IP address of the receiving host.
- **Options (variable bit)**: It is normally not used but could contain useful information, for hosts to process the packet.

- **Padding (variable bit)**: Contains some 0s to ensure that IP header is a multiple of 32 bits.

- **Data**: Transport layer data.

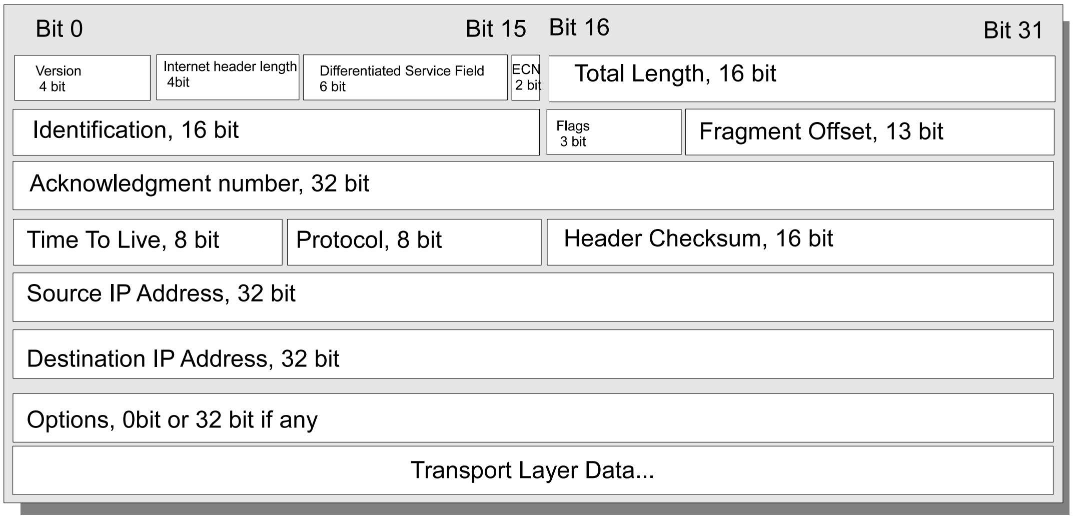


Figure 2.6: Inside of an IPv4 packet

Network addressing

Network communication uses address of hosts for data delivery in the network. There are two types of addresses in a network:

- **Hardware address**: Hardware or physical address was designed in early time of networking to be used in a local network. In such a network the hosts are physically close to each other (LAN). *MAC address* is the most common type of hardware address in TCP/IP model; it contains six bytes of hexadecimal number, uniquely assigned and hard-coded (can not be changed) for any network adapter by vendors. First three bytes identifies the manufacturer and the rest of the three bytes specifies each network adapter. For example, 00:1A:E3:B7:D7:C8 shows that 00:1A:E3 is the manufacturer ID and B7:D7:C8 is the adapter ID. As mentioned, physical address can only be reached within a local network, therefore it is not accessible from outside of the local network or from another network.

- **Logical address**: Logical address or IP address is a globally-geographically assigned address for a host in TCP/IP model. It is unique, but temporary per host among all of the interconnected networks. An IP address is a 32 bit decimal number (along with the *network mask*) where each byte is separated with another by a dot. Dots are very good identifiers for identifying networks and hosts when represented with a network mask. For example, the IP 128.233.192.40 with the network mask 255.255.0.0 indicates that 128.233 is a network ID and 192.40 is a host ID.

2.3 Embedded systems

An embedded system is a computer inside of a device and is used to run a single task or relatively several smaller tasks, most likely in *real-time* mode. The difference between a PC and an embedded system is that the embedded system is usually a single purpose computer that does not have all of the input and output interfaces like the PC does, for example keyboard and monitor. Usually the embedded system is hidden in the device and its functionality is transparent to the user who uses the device. Embedded systems can be found anywhere; to name a few, cell phone, microwave oven and automobile. Since embedded systems are generally designed to work as a single purpose computer, they don't demand too much hardware resources and therefore they are very efficient in terms of cost, reliability, performance and size.

Every embedded system includes hardware and software. Hardware consists of a micro-controller or a microprocessor along with peripherals. Additionally, it may contain other hardware blocks depending on the task. Software or running code on the microprocessor in an embedded system is called firmware and is stored on a flash or read-only memory (ROM). The code has to be compiled and debugged by a compiler, an assembler and a debugger tool in a PC and the final executable code is loaded to the embedded system. For more complex systems that run real-time application, a *Real-Time Operating System (RTOS)* is necessary in order to handle task scheduling and prioritizing different tasks [19].

A general method for implementing an embedded system is to combine all of the components on a single chip or *System on Chip (SoC)*. A typical SoC contains one or more processors containing the software, memory block such as ROM and RAM, peripheral like timer, external interface like USB and Ethernet controller. These blocks are connected through a bus, like *Advanced Micro-controller Bus Architecture (AMBA)* bus, inside of the SoC [20]. SoC can be implemented on ASIC or FPGA.

- **ASIC**: Application Specific Integrated Circuit is an integrated circuit that is fabricated for a specific function. It is intended for making higher volume designs in production which makes it very reasonable due to cost, performance and energy consumption. Once a circuit is fabricated on ASIC, it can not be changed.

- **FPGA**: Field-Programmable Gate Array is an integrated circuit that is fabricated to be programmed by the designer afterwards. FPGA contains logic blocks and configurable interconnections among the blocks that makes it possible to connect these blocks together. Logic blocks include simple logic gates and may contain memory blocks. FPGAs are multi purpose devices and can be programmed many times for various applications and embedded systems. If lower volume of products are desired, SoC on FPGA is not as costly as SoC on ASIC. In fact, FPGA has many advantages over ASIC. These advantages are: reconfiguring the FPGA to fix bugs and updating the design, faster time to market and non-recurring engineering costs.

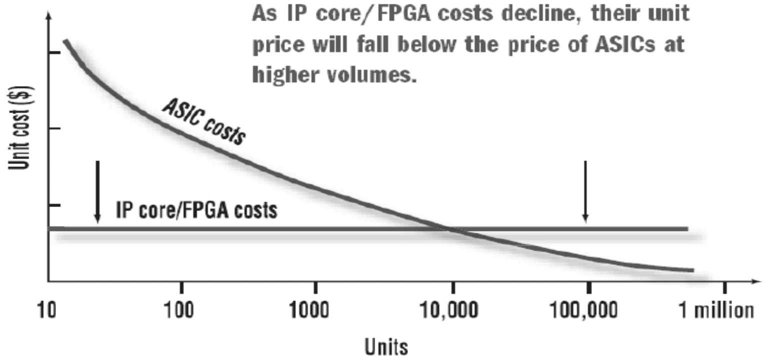


Figure 2.7: Cost comparison between ASIC and FPGA hardware development [2]

Figure 2.7 shows a cost comparison between FPGA and ASIC hardware development. Current leading FPGA manufacturers are Altera and Xilinx with more than 80 percent share of the FPGA market [21]. The most common *Hardware Description Language (HDL)* are *Verilog* and *VHDL* for designing logic circuit. An *Electronic Design Automation (EDA)* proprietary vendor tool is used to compile, simulate, analyze and verify the hardware design and prepare the design for hardware implementation on either FPGA or ASIC.

2.3.1 Altera design solution

Altera Corporation is one of the biggest FPGA manufacturers in the world that specializes in system on chip design platform solution. Altera offers various FPGA technologies, software tool and Intellectual property (IP) cores. Cyclone and Stratix series are the main Altera FPGA device families. Stratix is faster, more powerful and more expensive than the Cyclone series [22].

The main EDA design tools from Altera are *Quartus II* and *SOPC Builder*. SOPC builder software is used to connect various hardware blocks and components from the Altera library to make a complete hardware design. These blocks and components can be memory, CPU, interfaces and user designed blocks.

IP cores are already-designed-blocks from Altera to speed up the logic designing and eliminating the cumbersome design process from scratch. The most important IP core from Altera is the Nios microprocessor family which are loyalty free CPUs (for academic research) and are designed by SOPC builder tool incorporated in the Quartus package.

Nios II microprocessor

Nios II is a soft core 32 bit RISC microprocessor from Nios CPU family, introduced by Altera as a general processor to be used in Altera FPGA devices to make a complete system on chip design. Nios II is more powerful than the original Nios, its older family member, and gets benefit from the 32 bit instruction set, data and address bus [4].

Nios II can be configured in three different classes to offer more flexibility: fast, standard and economy. Fast architecture uses more resources on FPGA and provides faster speed and best performance in the family, whereas the economy type minimizes the resource and therefore results in a slower CPU. Standard type balances the resource and speed. To make a system based on Nios II CPU, two parts are necessary; SOPC builder is used to generate the hardware part, and *Embedded Development Suites (EDS)* (the software package from Altera) is used to create the software.

Altera development board

Altera has provided various development boards for designers as a base hardware platform to build, test and debug embedded systems. For the *embedded network* *firewall* project, the Stratix II development kit [23] has been used. As shown in Figure 2.8, the development board is equipped with the following components and features [3]:

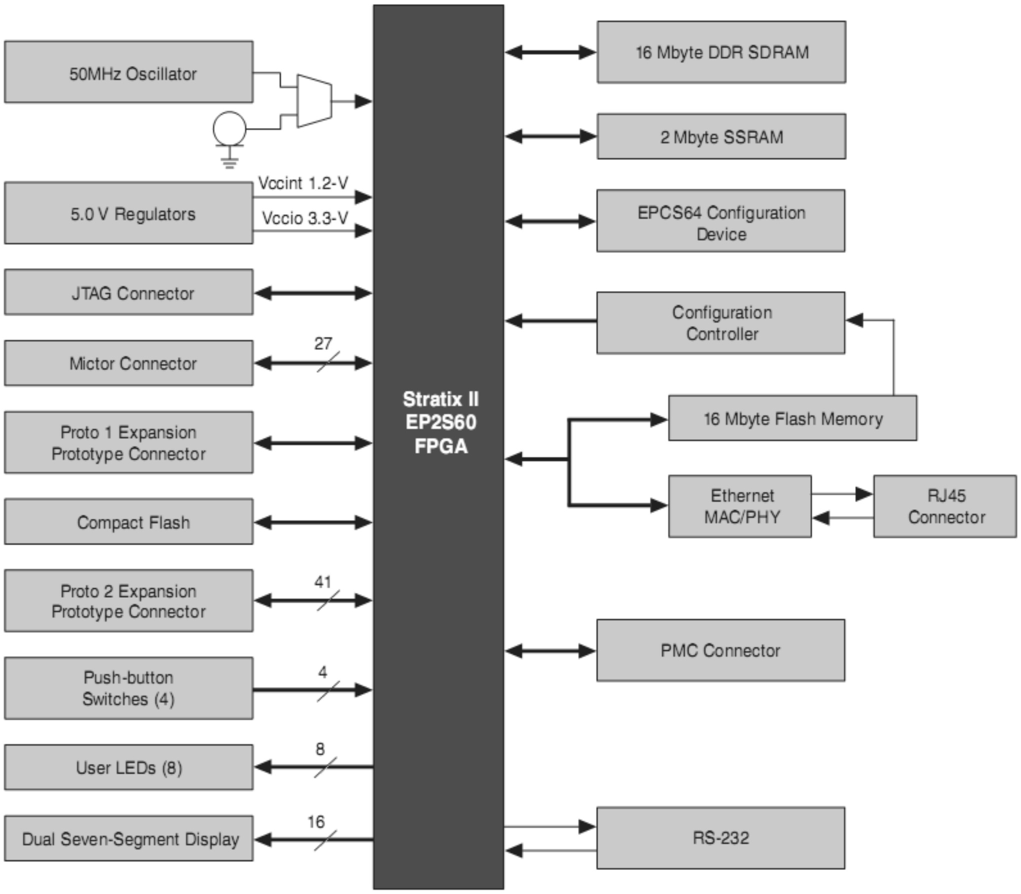


Figure 2.8: Altera development board [3]

- A Stratix II FPGA with more than 13,500 adaptive logic modules (ALM) and 1.3 million bits of on-chip memory
- Sixteen MBytes of flash memory
- Two MBytes of synchronous SRAM
- Thirty two MBytes of double data rate (DDR) SDRAM

- On-board logic for configuring the FPGA from the flash memory
- On-board Ethernet MAC/PHY device and RJ45 connector
- Thirty two bit PMC Connector capable of 33 MHz and 66 MHz operation
- RS-232 DB9 serial port
- Four push-button switches connected to the FPGA user I/O pins
- Eight LEDs connected to the FPGA user I/O pins
- Dual 7-segment LED display
- *Joint Test Action Group (JTAG)* connectors to the Altera devices via Altera download cables
- Fifty MHz oscillator and zero-skew clock distribution circuitry
- Power-on reset circuitry

The board (Figure 2.9) has been pre-programmed with a simple web server to respond to HTTP requests from the network traffic coming from the Ethernet port.

Nios II software development

Altera provides EDS as a software development environment for the Nios II microprocessor. Code can be written and compiled in a PC and then loaded to the Nios II via a JTAG cable. EDS offers proprietary and open source tools for software creation in the Nios II [24]:

- GNU tool chain is based on a standard GNU library and provides gcc compiler, assembler, linker and *make* procedure in a Unix like environment [25].

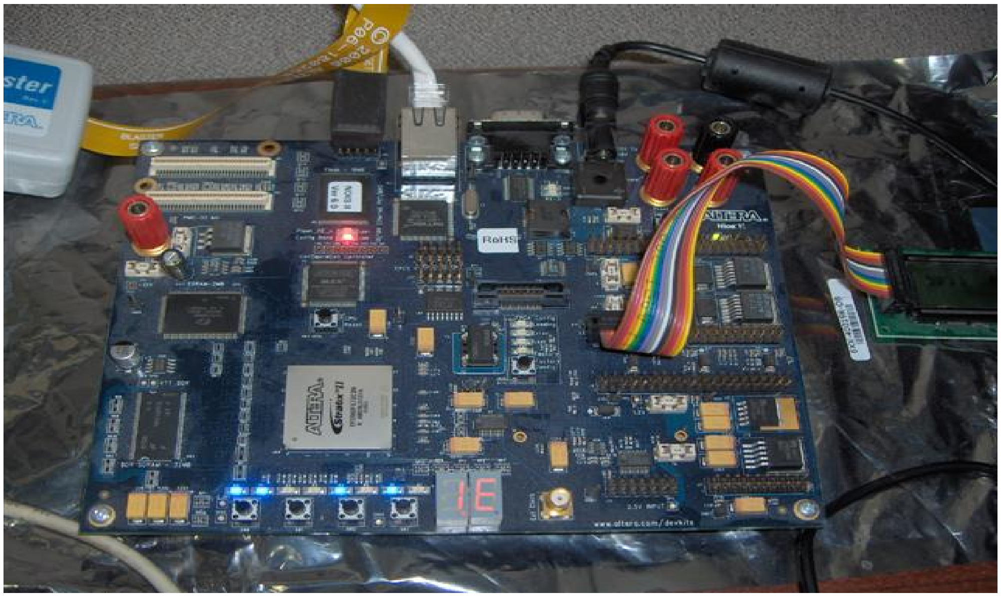


Figure 2.9: Altera development board

- *Board Support Package (BSP)* provides Altera *Hardware Abstraction Layer (HAL)*, *Newlib* C standard library, device drivers and optional RTOS and optional package like the NicheStack TCP/IP stack in a C/C++ environment.

MicroC/OS-II real-time operating system

Altera offers a third party simple and light, yet powerful real-time operating system as part of the BSP which is included in embedded systems that need process scheduling functionality. MircoC/OS-II, from *Micrium*, is a multi-tasking operating system that is mainly used for embedded systems that work in critical safety situation [26]. Altera has recommended to use MircoC/OS-II in any networking related application to make a robust and multi-threaded environment [4].

Ethernet and the NicheStack TCP/IP stack

Altera provides the NicheStack TCP/IP stack as a light version implementation of TCP/IP stack in BSP. *InterNiche Technologies Inc.* has made this stack for use in embedded systems with limited memory and processor resources. This stack offers [24]:

- Internet Protocol (IP)
- Internet Control Message Protocol (ICMP)
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- Dynamic host configuration protocol (DHCP)
- Address resolution protocol (ARP) for Ethernet
- Standard sockets application programming interface (API)

As described in this chapter, there are technology and tools available to design a firewall depending on the cost and performance speed. The goal of this thesis is to design a low operating cost hardware firewall that is small, operates in an efficient manner and has an acceptable speed processing time while keeping it easy to configure in real-time. Next chapter will walk through the design procedure of this *embedded network firewall* step by step.

Chapter 3

Embedded Network Firewall Design

In this chapter, the *embedded network firewall* design is discussed in details. Each individual module and its interaction with other modules in the design is described. In addition to the hardware section, the software part is also presented.

3.1 Hardware modules in the embedded network firewall

Figure 3.1 shows the overall layout of the *embedded network firewall* design. As shown in the block diagram, the main modules in the *embedded network firewall* are Nios II 32 bit microprocessor module, Ethernet module, Content Addressable Memory (CAM) module, Netmask RAM module, Arbiter module and Network Firewall module (NFM). All of these modules tightly work together to achieve a powerful, flexible and easy to configure packet filtering firewall.

At the beginning, when the system is powered up, the firewall module is in inactive mode as the Nios II has not been programmed yet to run any code or OS. Nios II Integrated Development Environment (IDE) development tool is used to upload essential applications including RTOS to run on the Nios II. The software initializes all of the modules in the design according to the selected mode of operation. When the Ethernet module finds any TCP/IP traffic targeted toward the Nios II, it interrupts the NFM. In the next step, the NFM extracts the necessary field (based on

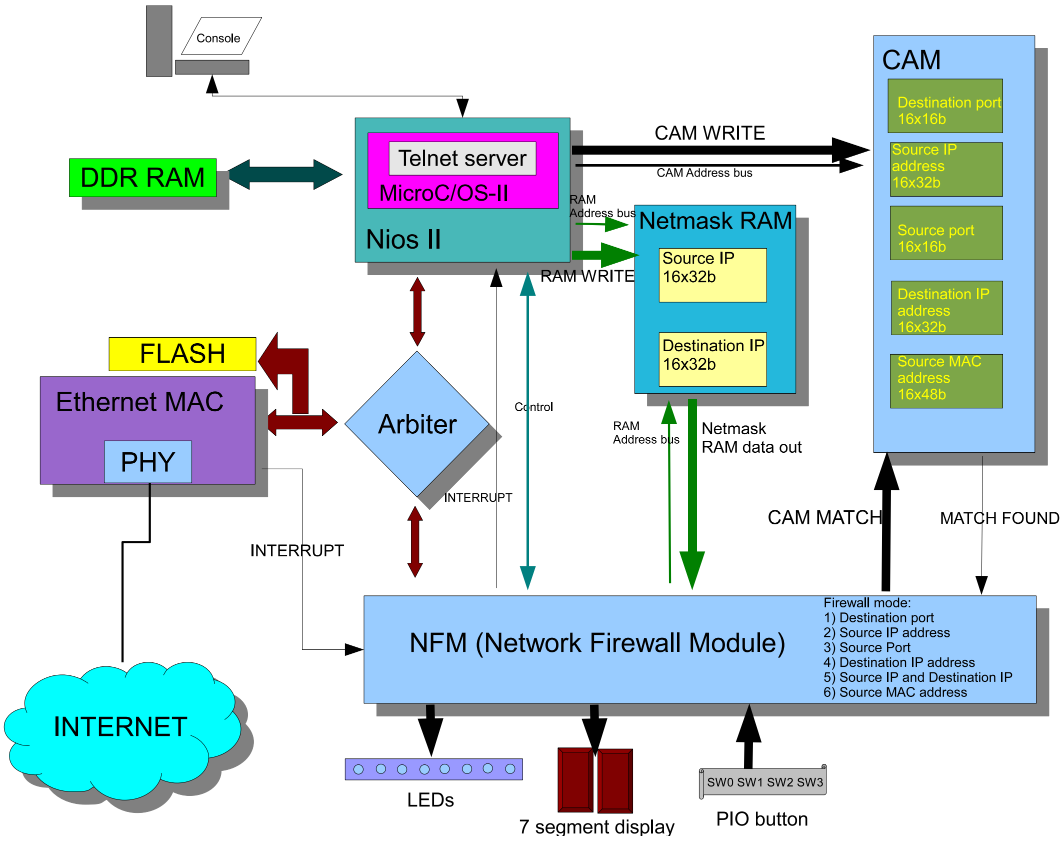


Figure 3.1: Embedded network firewall block diagram

the initial operation mode) off the Ethernet frame and inquires the permission from the CAM module. If the CAM module finds the selected field in its memory block (match found), it gives the pass permission to the NFM and the NFM interrupts the Nios II for a receiving packet. In the case of an invalid packet, the NFM drops the packet and waits for the next interrupt from the Ethernet. After each Ethernet interrupt, the packet status is reported to the Nios II by the NFM for monitoring purposes. The firewall operation mode and configuration can be changed any time on the fly by Telnetting to the Telnet server running on the Nios II. Following sections will explain each module functionality in more details.

3.1.1 Nios II microprocessor module

As briefly described in the background chapter, Nios II is a 32 bit RISC general purpose soft core microprocessor which is widely open to be customized in any Altera based embedded system design. A designer can choose from fast, standard or economy Nios II reference designs which are included in Altera development tool kit. If the cost or performance can not be met using any of these three reference designs, then the CPU can be customized in SOPC builder as required. In the *embedded network firewall* project, Nios II *standard reference design* has been utilized. It only takes less than 5 percent of the FPGA (StratixII 2S60) resources in the development board. Figure 3.2 shows the components in standard reference design. Nios II standard core employs the following properties [3]:

- Thirty two bit instruction set, data and address bus. Instruction set and data bus are separate (Harvard architecture [27])
- Avalon instruction set bus to memory components and Avalon data bus to memory and peripheral components
- Runs at 165 MHz speed
- One hundred twenty seven *Dhrystone Million instructions per second (DMIPS)* considering hardware multiply option
- 5 stage pipelining
- Can access up to 2Gbytes of external memory
- Arithmetic improvement by supporting multiply, divide and shift hardware blocks
- Support JTAG debug module

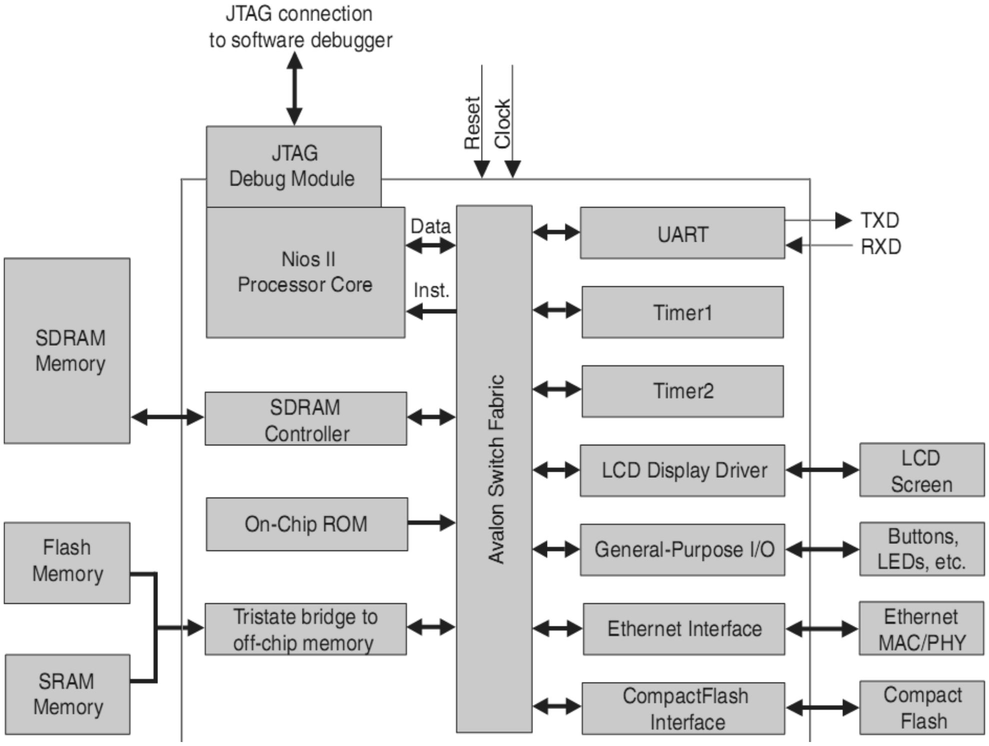


Figure 3.2: Nios II CPU architecture [4]

Like any other micro-controller the standard Nios II reference design comes with several peripherals. Compared to other micro-controllers, Nios II embedded system environment makes it more flexible and easier for peripheral customization. In the Nios II architecture, both memory and peripheral Input/Output are mapped into one integrated address space. Some of the peripherals play highlighted roles in the *embedded network firewall* project, and their features will be described in more detail.

JTAG *Universal Asynchronous Receiver/Transmitter (UART)* peripheral is a character media used to communicate between a PC and JTAG circuitry in Altera FPGA. Also Nios II uses a device driver to manage connection for displaying JTAG data stream on the screen. UART peripheral provides RS-232 character serial communication between Nios II and external device such as terminal software on a host PC.

The *Parallel Input/Output (PIO)* peripheral provides I/O access to external device, for example hardware custom block, controlling LEDs or seven segment display. In the *embedded network firewall* project ten different PIOs have been used as follows:

- Button PIO as control commands for troubleshooting and debugging the design (displaying selected header field on seven segment and passing invalid packet)
- LED PIO to show packet status (i.e., ARP, ICMP, TCP or UDP), status of data availability in CAM or dropped packet
- Seven segment PIO to show selected field in the Ethernet frame and TCP/IP packet
- Two PIOs as control signal between the Nios II and the NFM
- CAM data bus PIO between the NFM and the CAM module
- CAM Address bus PIO between the NFM and the CAM module
- CAM control bus PIO between the NFM and the CAM module
- Status PIO to report the status of the last received packet from the NFM to the Nios II
- Expanded CAM data bus PIO between the NFM and CAM module for storing MAC address

Table 3.1 shows the address mapping for each peripheral in the *embedded network firewall* design. One of the most important peripheral modules in the *embedded network firewall* project is the Ethernet module that will be discussed in detail in the next section.

Table 3.1: Different PIOs used in the design

Block	**Name**	**Physical address**
JTAG core	JTAG_UART	0x021208b0
Ethernet	LAN91C111	0x02110000
LCD display	LCD_DISPLAY	0x02120880
UART	UART1	0x02120840
Button	BUTTON_PIO	0x02120860
LED	LED_PIO	0x02120870
JTAG UART	JTAG_UART	0x021208b0
NFM (from Nios II)	PIO	0x01200020
NFM (to Nios II)	PIO_1	0x01200030
CAM	PIO_DATA_OUT_CAM	0x01200040
CAM	PIO_ADDR_BUS_CAM	0x01200050
CAM	PIO_CTRL_CAM	0x01200060
CAM	PIO_STATUS	0x01200070
CAM	PIO_DATA_OUT_CAM_MAC	0x01200080

3.1.2 Ethernet module

Altera development board comes with networking capability and is equipped with LAN91C111 chip from *Standard Microsystems Corporation (SMSC)* [28]. LAN91C111 control and data pins are hardwired to the FPGA on the development board, and tools are available for Nios II embedded system to communicate with network through the chip. This chip includes MAC and PHY engines and is able to process *Carrier Sense Multiple Access With Collision Detection (CSMA/CD)* protocol in 10/100 Mbps full duplex. The chip has 8 KBytes of internal memory for receiving and transmitting operations; also there are 32 registers in the chip for configuring and controlling the operations. The chip must be initialized by the Nios II for receiving and transmitting operations. The initialization sets some parameters, such as MAC address, memory base address (to be accessed by Nios II) and speed negotiation. Two main operations are packet receiving and packet transmitting:

- **Packet receiving**: When the receive interrupt flag is enabled and the receive queue is empty in on-chip memory, the data is written to assigned memory with a packet number as soon as there is an incoming packet on the Ethernet interface with matching address. After packet reception ends, the status word (for the receiving operation) is written at the beginning of the packet. If there is an error in reception or Cyclic Redundancy Check (CRC) error in the frame, the packet is dropped and memory is released, otherwise the packet number is written at receiving *First In First Out (FIFO)* buffer and an interrupt is made to notify the Nios II for packet receiving. Now the device driver in Nios II processes the data in receiving area and releases the memory.

- **Packet transmitting**: At the beginning of the packet transmission, the device driver attempts to allocate memory in on-chip memory block. When it happens, LAN91C111 interrupts the device driver for a successful memory allocation and a transmit packet number is generated. In the next step, transmit packet number is written into transmit FIFO by the device driver and waits until LAN91C111 accomplishes the transfer. After the transfer completes, the status word is placed at the first word in the memory and the packet number is moved to transmit completion FIFO, and an interrupt is made to notify the Nios II for transmission completion. The device driver reads the interrupt status word to check if the transmission is successful or failure.

The data structure in on-chip memory for both receiving and transmitting locations begin with the status word which is written by MAC engine upon receiving or transmitting operation. The next word shows the number of the data bytes that are written by the MAC after the packet is received, and by the Nios II after the packet is transmitted, and the actual Ethernet frame comes after the byte count. In case of the received frame, the last byte in the data is the CRC which is followed by a control byte (Figure 3.6). In the development board, the Flash memory and LAN91C111 share the data and address bus, and all of the LAN91C111 registers can be accessed from 0x300 address offset. The 16MByte *Advance Micro Device (AMD)* flash memory on the board can be used for holding FPGA configuration and also for general purpose non-volatile memory storage for the Nios II micro processor, for example the Nios II software.

3.1.3 Content Associative Memory module

Content Associative Memory (CAM) is a type of memory system for fast searching application in hardware [29]. It acts differently from the other types of memory like RAM and ROM. Unlike other memory systems, CAM is fed with data instead of address. CAM searches the entire memory for data and if the data is available in memory (data already stored previously), the location of data is returned. Since the whole operation takes only one clock cycle, CAM requires memory cell and associative comparison hardware circuitry for conducting parallel search of each memory cell. This makes CAM very fast but also costly due to consumption of large number of hardware logic blocks. Because of the high speed data search, CAM is widely used in networking devices such as firewalls, network switches and routers, offering fast address lookup and low packet processing latency.

In the *embedded firewall*, CAM is used for saving source and destination TCP/UDP port number, source and destination IP address and MAC address field. Altera does not provide any CAM memory block on the StratixII FPGA device or IP core implementation in the Quartus II package, therefore dual port memory with some extra logic has been used to implement the CAM modules for network firewall design [29]. The CAM template module is a 16 word by 8 bit (CAM 16x8) in total of 128 bits and is made by a 4 Kbits of dual port memory block on FPGA. Figure 3.3 shows the CAM 16x8 block diagram and its internal dual port RAM block wiring.

The 4 Kbit memory block is setup as a 1 bit data and 12 bit address bus input for port-A and 16 bit data and 8 bit address bus for port-B. Port-A is used for data write operation with an asserting “1” on the data input as write enable signal for

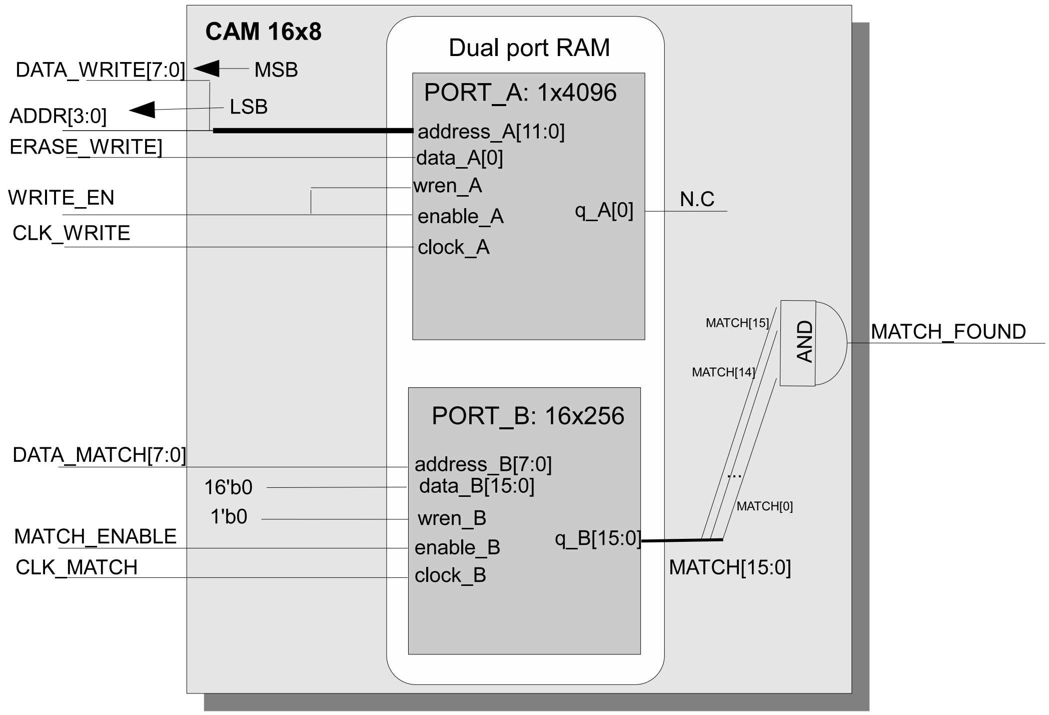


Figure 3.3: CAM 16x8 block diagram (built by dual RAM port)

CAM module, and concatenating 8 bits of data and 4 bits of address in address port of port-A. Port-B is used for data read operation with an asserting “0000 0000 0000 0000” (16’b0) in data input of port-B and providing the desired data at data input of the CAM module. CAM write and read operations are briefly explained in next paragraph by using an example.

- **CAM write** operation happens by providing the data and address to the DATA_WRITE (as MSB) and ADDR (as LSB) input of the CAM module which makes “1” being written at “DATA_WRITE, ADDR” location of memory. For example, one would like to store A5h value at the 6th memory location in the CAM; “1010 0101 , 0110” is provided to “DATA_WRITE , ADDR”, therefore 1 is written at “1010 0101 0110” location (or at the address 2646 in the memory).

- **CAM read** operation is done by providing the value to DATA_MATCH input of the CAM and enabling the CAM read enable input. If the data is already written in the CAM, the location appears in the MATCH output of the CAM module. For example one would like to find out whether A5h value is stored in the CAM or not. A5h is provided to the DATA_MATCH input of the CAM which causes the A5h memory location to be searched and the corresponding data, which is "0000 0000 0100 0000" (6th bit of MATCH goes high), is returned at the MATCH output. The output is the same as address location which the data was previously written at.

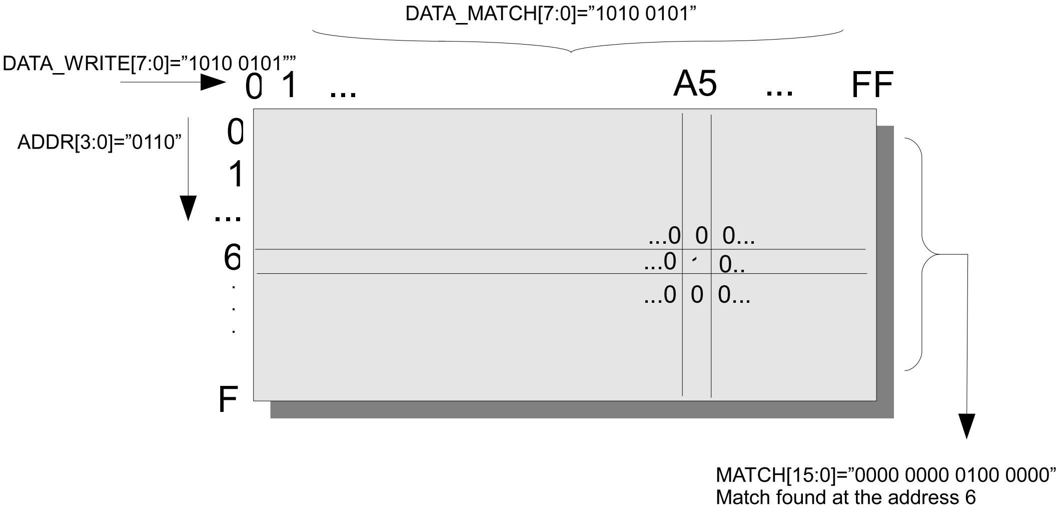

DATA_MATCH[7:0]="1010 0101"
DATA_WRITE[7:0]="1010 0101""
0 1 ... A5 ... FF
ADDR[3:0]="0110"
0
1
...
6
F
...0 0 0...
...0 ´ 0..
...0 0 0...
MATCH[15:0]="0000 0000 0100 0000"
Match found at the address 6

Figure 3.4: An example for CAM read and write operation

Figure 3.4 shows the discussed example. In the *embedded network firewall* design two CAM 16x16 modules are used to store sixteen 16 bit destination and source port numbers, two CAM 16x32 modules are used to store sixteen 32 bit destination and source IP address and one CAM 16x48 module is used to store sixteen 48 bit source MAC address by cascading several CAM 16x8 modules. Total of eighteen 4 Kbits of memory blocks on the FPGA are exhausted for the CAM modules. Table 3.2 summarizes all of the CAM modules in the *embedded network firewall* design.

Table 3.2: CAM modules in the *embedded network firewall* design

Mode	**Capacity**	**Size**	**Component**	**Memory block**
Destination port	16 word	16-bit	2 CAM 16x8	2*4Kb
Source port	16 word	16-bit	2 CAM 16x8	2*4Kb
Destination IP address	16 word	32-bit	4 CAM 16x8	4*4Kb
Source IP address	16 word	32-bit	4 CAM 16x8	4*4Kb
Source MAC address	16 word	48-bit	6 CAM 16x8	6*4Kb

3.1.4 Network Firewall Module (NFM)

The main functionality of the *embedded network firewall* project is accomplished by the network firewall module (NFM). This Verilog module contains a state machine with five states. The machine changes the state from one to another based on the input signals and current conditions. After system initialization and enabling the firewall by the Nios II, the NFM waits for an interrupt from the Ethernet module. Figure 3.5 illustrates the NFM process flowchart.

- **Interrupt process state**: When an interrupt occurs, at first the NFM saves current value of the registers in the Ethernet module and then reads the *interrupt status register (ISR)*. If the status register indicates any error, such as receive *over-run* error or *Ethernet Protocol Handler (EPH)* error, the state machine switches to the *Restore chip state* and all of the registers are restored to their previous values and the Nios II takes over the error handling process. If the status register shows the transmit interrupt, the state machine switches to the *Restore chip state* and the Nios II takes over the interrupt. In case of the receive interrupt, the state machine switches to the *Receive state* for receiving process.

- **Receive state**: In this state, the first word from the beginning of the receive data queue is fetched in order to investigate if there is any issue in the frame reception. The error can be of Alignment, bad CRC, Long length frame or

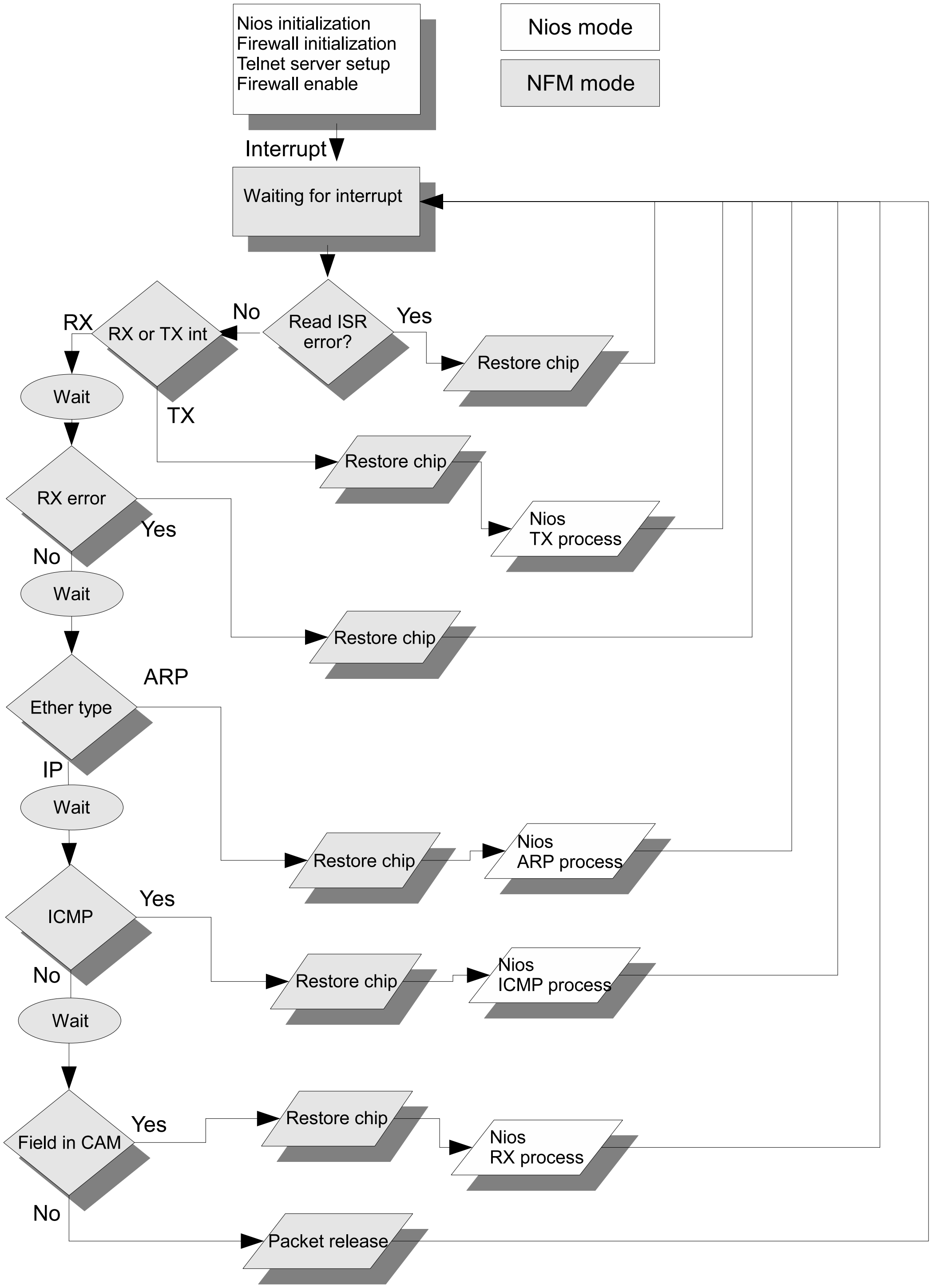


Figure 3.5: NFM process flowchart

short length frame. The state machine switches to the *Restore chip state* for error handling in case of any error otherwise it switches to the *Packet type state* to discover the type of the Ethernet frame. In the next step, according to the firewall condition, specific data field is extracted from the packet (for example destination TCP port number) and is sent to the corresponding CAM module for a match. If the data field is already put in the CAM in the initialization/configuration stage (therefore is available), then the CAM indicates *match found* signal and the state machine switches to the *Restore chip state* and Nios II is notified by an interrupt of a valid receiving packet. If the data is not available in the CAM, the state machine switches to the *Release packet state* to drop the packet. In the source MAC address mode, the firewall acts in opposite way meaning that it drops the frame if the source MAC address is found in the CAM. This mode is useful when a host communication has to be denied in a local network to prevent spamming until further action is taken by the network administrator.

- **Packet type state**: This state discloses the type of the Ethernet frame (Ethertype filed in the Ethernet header). If the receiving frame is an Address Resolution Protocol (ARP) frame, which is used to translate an IP address to a MAC address in a local network, the interrupt handling is done by the Nios II by switching to the *Restore chip state*. If the frame data is not an ARP then the packet is considered to be an Internet Packet (IP) and it is investigated for Internet Control Message Protocol (ICMP) packet. Figure 3.6 illustrates all of the fields inside of the memory structure in LAN91C111; the *packet type state* extracts a few of these fields based on the firewall configuration.

  In this design, for the sake of troubleshooting, all of the ICMP packets are forwarded to the Nios II for ICMP response and therefore the state machine switches to the *Restore chip state*. The firewall ICMP handling can be disabled

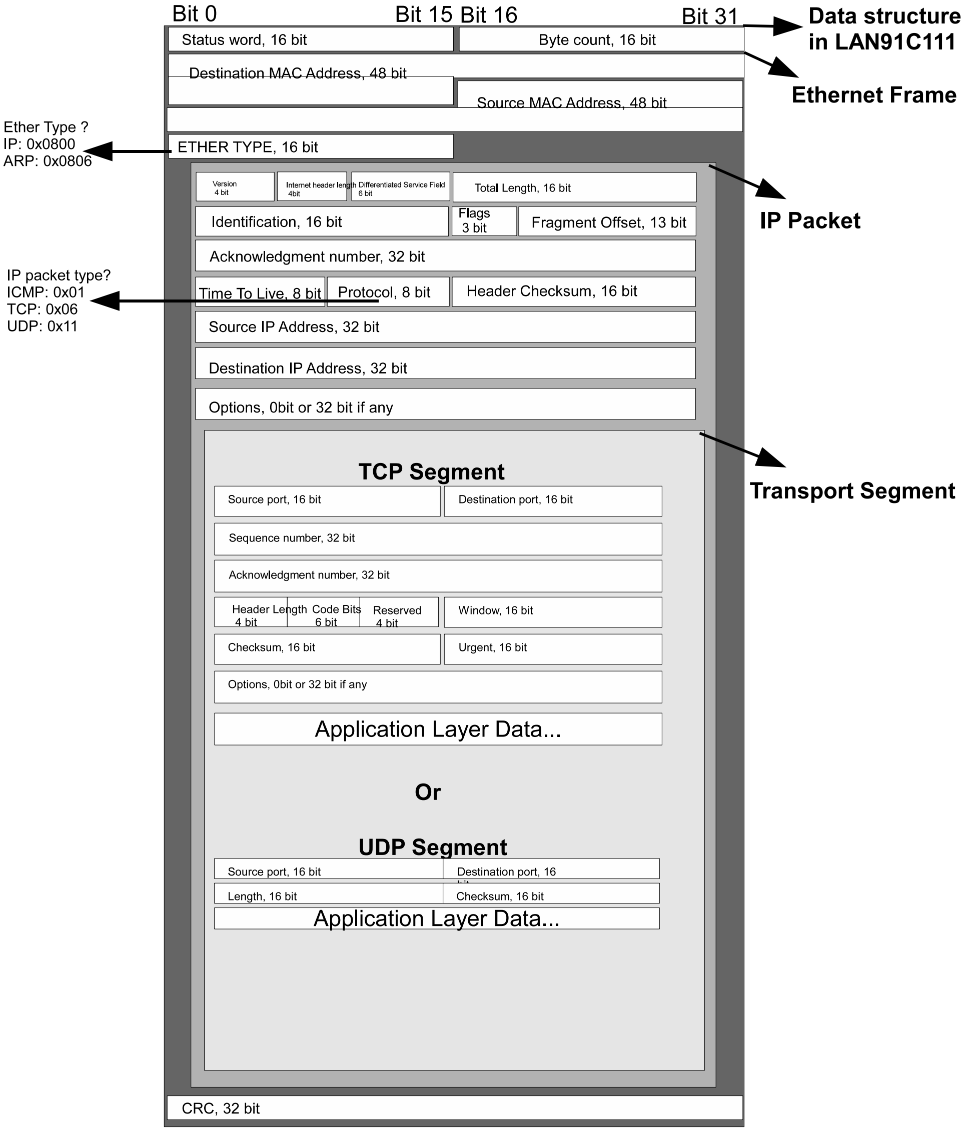


Figure 3.6: Data structure in LAN91C111 on-chip memory

by the user to speed up the packet processing since several clock cycles need for identifying the ICMP packet by the NFM. In the next step based on the firewall condition, specific data field is extracted from the packet and the state machine switches back to the *Receive state.*

- **Release packet state**: When the state machine lands in this state, it means the packet cannot make it to the network beyond the firewall and must be dropped. This action is followed by switching back to the *Restore chip state.*

- **Restore chip state**: At this state the *pointer register* and *bank select register* are restored to their value before the *Interrupt process state.*

- **Wait state**: In the development board design, the ARDY pin of LAN91C111 is not connected to the FPGA and according to the Ethernet manufacturer, any read operation from the receive FIFO should not be done for at least 370 ns after the pointer register is filled with the data register. The board operates at 50 MHz and thus each clock cycle takes 20 ns to be completed, so each read operation has to wait at least 19 clock cycles (or 380 ns) for a safe reading. The *wait state* idles the NFM for 19 clock cycles until the data register output is valid.

3.1.5 Network mask RAM module

In the source and destination IP address condition, the firewall can not only permit or deny a single IP address, but it can also decide what range of IP addresses can access the network beyond the firewall. This can be possible by either entering every single IP address in the source or destination IP address in the CAM module, or by specifying an IP range (network ID and network mask). As it sounds the first option is not efficient because it is a tedious procedure for entering all of the IP addresses into the CAM and needs lots of valuable CAM memory cells. In this project the second option is selected by sacrificing a few clock cycles to search in RAM module for network mask. CAM is not capable of storing *don't care* bit, therefore, the combination of CAM and netmask RAM make it possible for storing the IP address range. Every single location of the netmask RAM module is pulled and tested against the data stored in the CAM IP module and as soon as a match is found, the packet is validated and passed through the firewall. If no match is found in the CAM, the IP address has not been configured in the permitted IP address range and hence the packet must be removed. If there are more data fields in the netmask RAM, this procedure takes more clock cycles to finish.

The *embedded network firewall* project employs two RAM modules, each with sixteen 32 bit words to store network mask for source and destination IP addresses. Figure 3.7 illustrates an example of network mask function. Table 3.3 shows the RAM modules used in the network firewall design.

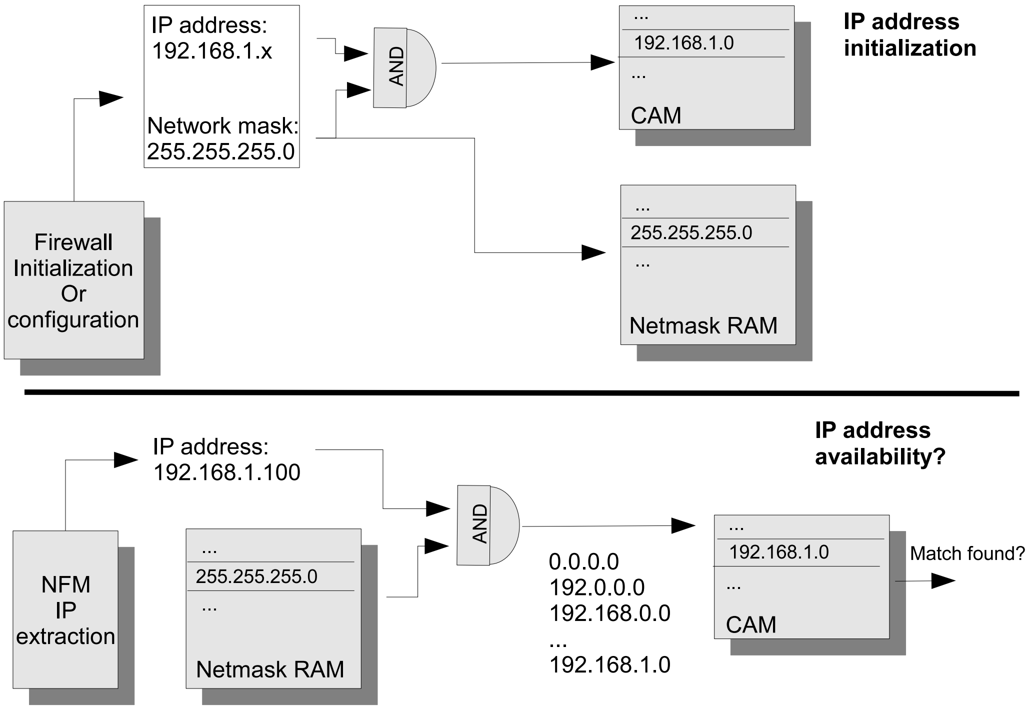


Figure 3.7: Netmask RAM process

Table 3.3: Netmask RAM memory block usage in the project

Mode	**Capacity**	**Size**	**Component**	**Memory block**
Destination IP address	16 word	32-bit	1 RAM 16x32	1*1Kb
Source IP address	16 word	32-bit	1 RAM 16x32	1*1Kb

3.1.6 Arbiter module

So far we have seen the NFM interferes the data communication between the Nios II and the Ethernet module in order to enforce the firewall policy. Obviously, the data transfer can only happen among these modules when there is a shared data and address bus. Arbiter comes as a control module to allocate shared resources (Ethernet module) between the Nios II and the NFM modules and lets only one to one communication at a time. In this design, two modes are used to differentiate when the Nios II is using the Ethernet module and when the NFM is in charge of the Ethernet module. The first mode is called software mode or Nios mode and second mode is called hardware or NFM mode. Software mode is the default mode until the firewall is activated, from then onwards if the Nios II wants to communicate with the Ethernet module, it has to make a request to the NFM. The NFM acknowledges and allows the request when it is in idle mode and no interrupt is being processed. Figure 3.8 shows a block diagram of the arbiter and its related modules. Ethernet module and the Nios II data bus are bi-directional. The arbiter uses tri-state buffer, which is controlled by *mode* signal from the Nios II to allocate the data bus.

3.1.7 Status display module

This module is designed in the *embedded network firewall* project to let the user/administrator observe the visual status of the last processed packet. Four push buttons on board are used to select output PIO from the NFM to be shown on the LEDs and seven segment display. Table 3.4 shows the LED status table and Figure 3.9 while

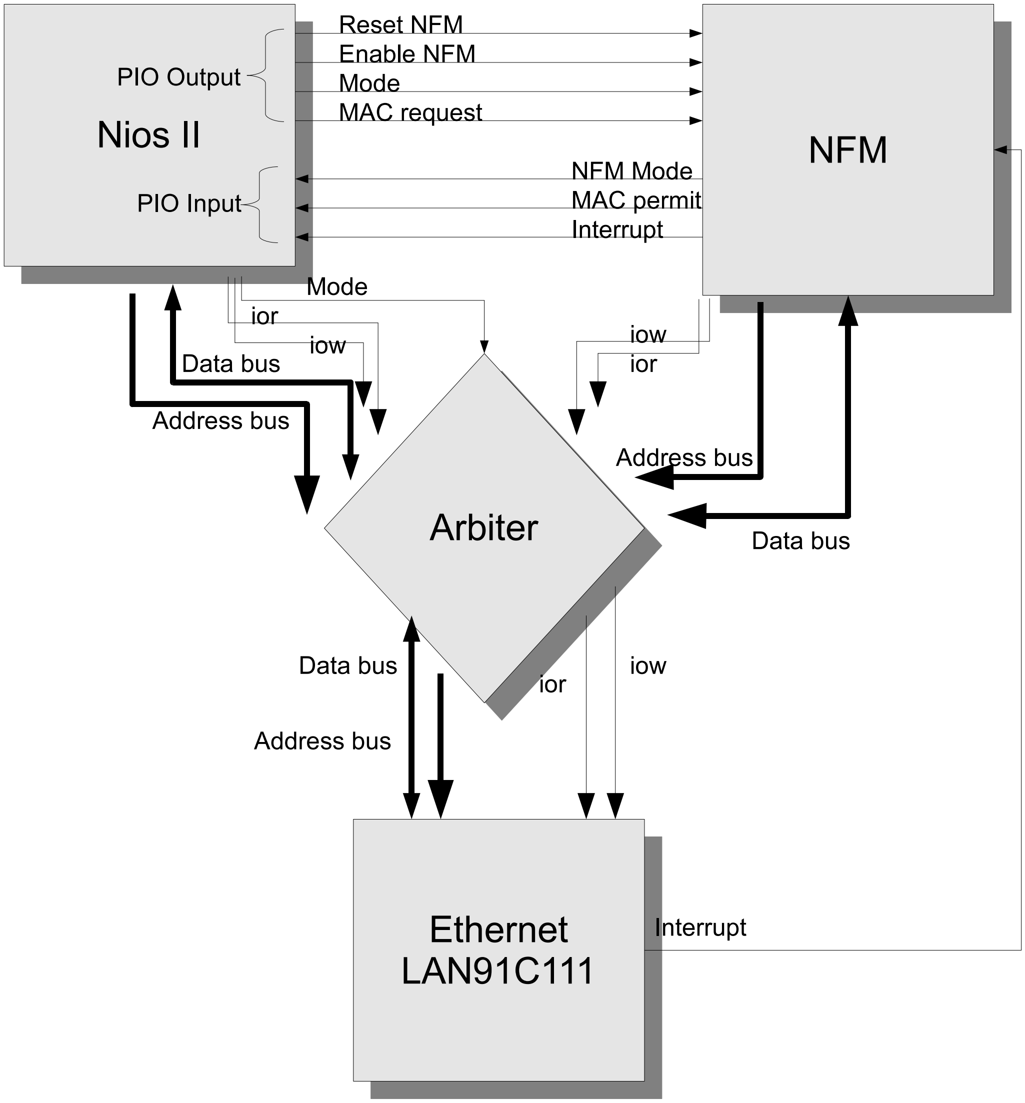


Figure 3.8: Arbiter and connected modules

the display are in action. The *clk_cycle_counter* is the binary counter for showing the number of clock cycles used for the last interrupt processing.

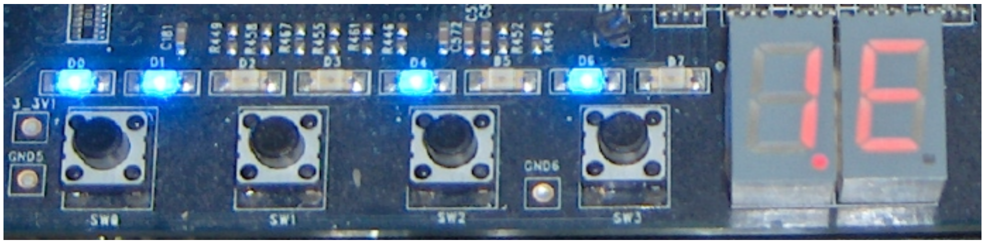


Figure 3.9: LED and seven segment monitoring

Table 3.4: LEDs function table

LED number	**SW 0 pressed**	**SW 0 not pressed**
LED 0	clk_cycle_counter[0]	ISR_register[0]
LED 1	clk_cycle_counter[0]	permitted packet
LED 2	clk_cycle_counter[1]	ARP packet
LED 3	clk_cycle_counter[2]	ICMP packet
LED 4	clk_cycle_counter[3]	TCP packet
LED 5	clk_cycle_counter[4]	UDP packet
LED 6	clk_cycle_counter[5]	MATCH found
LED 7	clk_cycle_counter[6]	Dropped packet

Table 3.5 indicates the seven segment display status. IP address in Table 3.5 is shown in *a.b.c.d* format and MAC address in *aa:bb:cc:dd:ee:ff* format. Only last 4 Bytes of the MAC address can be displayed on the seven segment display.

Table 3.5: Seven segment status display

SW 2	**SW 3**	**src/dst port**	**src/dst IP addr**	**src MAC addr**
0	0	port number	d	ff
0	1	-	c	ee
1	0	-	b	dd
1	1	-	a	cc

3.2 Software modules in the embedded network firewall

The software modules of the *embedded network firewall* design will be explained in the following sections. These modules handle initialization and configuration of the Nios II and the *embedded network firewall* to coordinate and run Telnet server software for changing the firewall configuration and monitoring the firewall. Altera Embedded Design Suites (EDS) proposes Layered Software Model to provide high flexibility, abstraction and simplicity for embedded system application development [5]. Figure 3.10 shows all of the different software layers that are used in the embedded system network application.

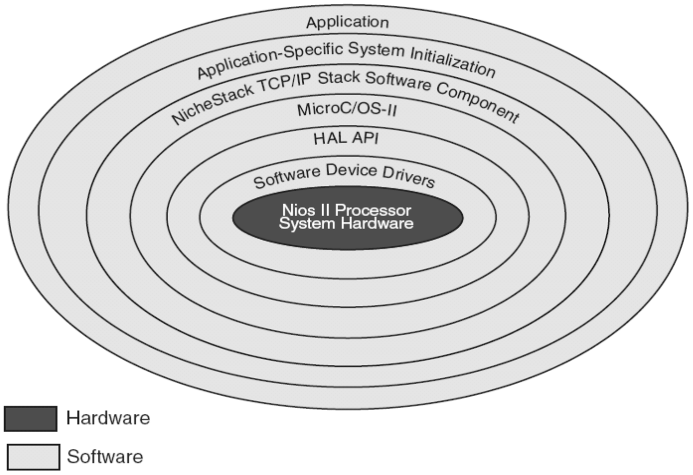

Application
Application-Specific System Initialization
NicheStack TCP/IP Stack Software Component
MicroC/OS-II
HAL API
Software Device Drivers
Nios II Processor System Hardware
Hardware
Software

Figure 3.10: Layered software model for Nios II network application development [5]

- Application layer for Nios II is C/C++ source code for specific purpose. In the *embedded network firewall* design, the Telnet server application is used from the Nios development kit [5].

- Application-Specific system initialization layer contains MicroC/OS-II and NicheStack TCP/IP initialization function that create operating system tasks.

- NicheStack TCP/IP stack software component layer contains networking service function to be used by sockets API in the above layer.

- MicroC/OS-II layer provide real time operating system capable of multi-tasking.

- Hardware Abstraction Layer Application Programing Interface (HAL API) layer offers standard interface to the software device driver.

- Software device driver layer contains software modules for driving Ethernet module.

- Nios II Processor System Hardware layer is the Nios II micro processor implemented on Altera FPGA.

3.2.1 MicroC/OS-II real-time operating system

As it was briefly described in the background chapter, Altera has included Micros/OS-II real time operating system in EDS for the Nios II microprocessor. MicroC/OS-II is a small, portable, ROM-able, scalable, preemptive, real-time, multitasking operating system for embedded systems that provides the following services [30]:

- Task (threads)

- Event flags

- Message passing

- Memory management

- Semaphores

- Time management

In the *embedded network firewall*, the operating system is used to provide an appropriate platform for the NicheStack TCP/IP stack and Telnet server. Many of the operating system services and system calls are not directly used by the Telnet server, however the NicheStack TCP/IP stack gets the full benefit from them.

3.2.2 NicheStack TCP/IP Stack

Nios II IDE is equipped with the NicheStack TCP/IP stack software module to provide TCP/IP networking ability for embedded systems based on the Nios II microprocessor platform. The NicheStack TCP/IP stack brings IP, ICMP, UDP, TCP, DHCP, ARP and API features for network programing on the Nios II. The MicroC/OS-II RTOS is the base multi-threaded environment to be used in the NicheStack TCP/IP stack. The Altera implementation of the NicheStack TCP/IP stack is supplied with a software driver for the Ethernet module. The standard API sockets are the main interfaces to the NicheStack TCP/IP stack. The NicheStack TCP/IP stack uses two threads of resources provided by the operating system for the main task and monitor task. After initialization, the main task waits for a new packet to arrive, and the monitor task checks the time out counter to avoid time out situation.

3.2.3 LAN91C111 device driver

The LAN91C111 comes with a device driver including some low level functions to provide API for software programing. These functions perform chip preparation, initialization, packet sending and receiving and interrupt handling. The device driver is altered to be able to work with the NFM and Nios II in this project. Before the firewall activation in the design, the Nios II uses the Ethernet module without any interference but, as discussed earlier, after the firewall configuration, the Nios II must request for data communication. This means that normal device driver function has to be changed in a way that the Nios II requests and permissions are embedded in the appropriate functions. When an interrupt comes to the Nios II, the Interrupt Service Routine (ISR) is called. *s91_isr()* is the ISR function in *smsc91x.c* (the LAN91C111 device driver) that takes care of the interrupt handling. When this function runs, it checks to see if the firewall is enabled or not by reading the input PIO from the NFM. If the firewall is disabled then it is not necessary to ask for permission, otherwise it requests for the permission by writing the output PIO to the NFM until the permission is granted. After the permission is granted, the firewall goes to the software (Nios) mode and it waits until the Nios II finishes the interrupt handling and gives back the Ethernet control to the NFM by switching back to the hardware (or NFM) mode.

This procedure takes place when any interrupt happens either for packet receiving or packet transmitting. For packet transmission, the Nios II calls *s91_pkt_send()* function to place the packet into the Ethernet on-chip memory. Since this is an Ethernet access, the same procedure must be followed, and permission must be allowed before any access is granted.

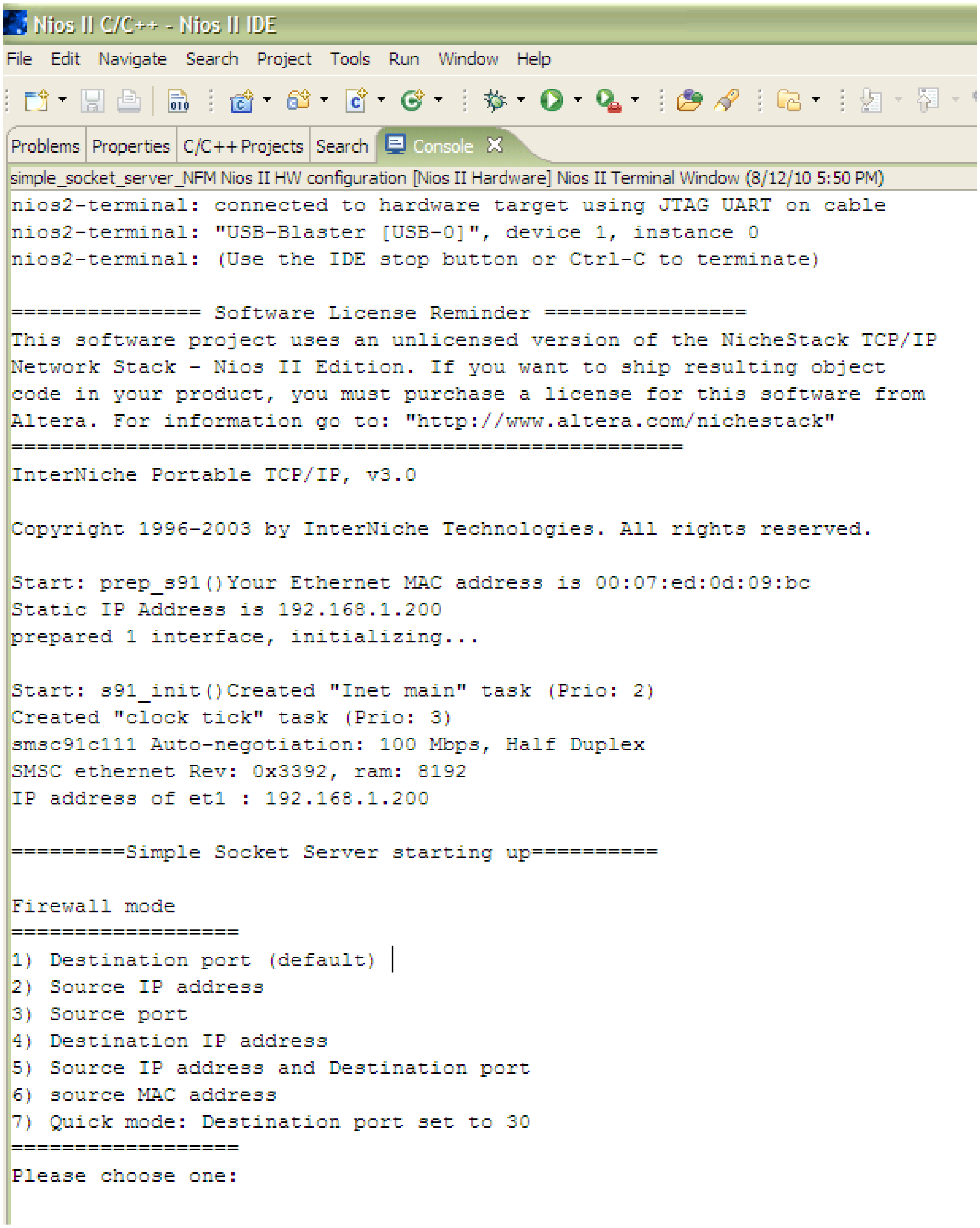
3.2.4 Initialization

The network firewall is initialized after all of the software modules in the system have been setup and the Nios II is ready to start the Telnet server. *iniche_init.c* is the primary C module in Telnet server application that contains *main()* function to start the NicheStack TCP/IP stack and the operating system. Right after Telnet server starts up, firewall initialization function, *nfm_management()*, runs to fill up all of the cells in CAM and netmask RAM modules with 0xFF value. At first the value is put on the CAM data bus PIO and then each CAM or netmask RAM module is selected by writing on corresponding CAM address bus PIO. Finally, each module is enabled by writing the enable command on the CAM control bus PIO. Now all of the memory modules are filled up with 0xFF value and the firewall is ready to be configured by the user. At this stage, firewall mode can be selected from the main setup menu in the console:

- *(1) Destination port (default)*
- *(2) Source IP address*
- *(3) Source port*
- *(4) Destination IP address*
- *(5) Source IP address and Destination port*
- *(6) Source MAC address*
- *(7) Quick mode: Destination port set to 30*

By selecting the first or third option , the port number is selected and the user is asked to enter the number of permitted ports (entries) in the CAM. In the next step, the user must enter each port number in hexadecimal, one at a time, until all of the entries are entered. Source or destination IP address modes are different since each IP address must be followed by a network mask as well.

The sixth option configures the firewall in the source MAC address mode. In this mode, the MAC addresses of the devices in the local network are entered in order to deny them hosts from network communication. Option five can highly restrict the access authorization by combination of the source IP address and the destination port. The final option (seventh option) is built for quick setup to permit access to the Telnet server on the Nios II at destination port 30. After the data entry, the firewall gets enabled by *nfm_control()* function and is ready to perform. Figure 3.11 shows the screen shot of the initialization menu. Next section will discuss more on the Telnet server and real-time configuration.


Nios II C/C++ - Nios II IDE
File Edit Navigate Search Project Tools Run Window Help
Problems Properties C/C++ Projects Search Console
simple_socket_server_NFM Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (8/12/10 5:50 PM)

```
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

=============== Software License Reminder ================
This software project uses an unlicensed version of the NicheStack TCP/IP
Network Stack - Nios II Edition. If you want to ship resulting object
code in your product, you must purchase a license for this software from
Altera. For information go to: "http://www.altera.com/nichestack"
=====================================================
InterNiche Portable TCP/IP, v3.0

Copyright 1996-2003 by InterNiche Technologies. All rights reserved.

Start: prep_s91()Your Ethernet MAC address is 00:07:ed:0d:09:bc
Static IP Address is 192.168.1.200
prepared 1 interface, initializing...

Start: s91_init()Created "Inet main" task (Prio: 2)
Created "clock tick" task (Prio: 3)
smsc91c111 Auto-negotiation: 100 Mbps, Half Duplex
SMSC ethernet Rev: 0x3392, ram: 8192
IP address of et1 : 192.168.1.200

=========Simple Socket Server starting up==========

Firewall mode
==================
1) Destination port (default)
2) Source IP address
3) Source port
4) Destination IP address
5) Source IP address and Destination port
6) source MAC address
7) Quick mode: Destination port set to 30
==================
Please choose one:
```

Figure 3.11: Initialization menu in the Nios IDE console

3.2.5 Telnet Server

Telnet is a client-server based network protocol for transferring text data for interaction communication over TCP protocol on the TCP/IP network. Telnet is usually used to remotely access a server for monitoring or configuration purposes. Telnet application is divided into two parts:

- **Telnet server** is setup at server side of the network for accepting Telnet requests, commonly over TCP port 23 [31].
- **Telnet client** is a terminal software on the client side for establishing a Telnet session with a Telnet server.

Altera EDS has provided a *Simple Socket Server* application for demonstration of TCP/IP communication to Nios II running the NicheStack TCP/IP stack over MicroC/OS-II RTOS [5]. The source code modules for implementing the Telnet server are:

- *iniche_init.c*: *main()* and *nfm_management()* functions for Telnet server and firewall initialization.
- *network_utilities.c* and *network_utilities.h*: functions for setting the MAC and IP addresses.
- *simple_socket_server.c* and *simple_socket_server.h*: all of the tasks and functions to apply the NicheStack TCP/IP Stack sockets interface and to create all of the MicroC/OS-II resources.
- *alt_error_handler.c* and *alt_error_handler.h*: error handler routines for the Simple Socket Server, NicheStack TCP/IP Stack, and MicroC/OS-II.

This simple Telnet server accepts commands over TCP/IP connection from a client terminal on a host PC and sends those commands to LEDs on the board to switch them on and off. In this project, the Simple Socket Server application has been modified to accept commands for the firewall configuration and monitoring the system.

3.2.6 Configuration on the fly

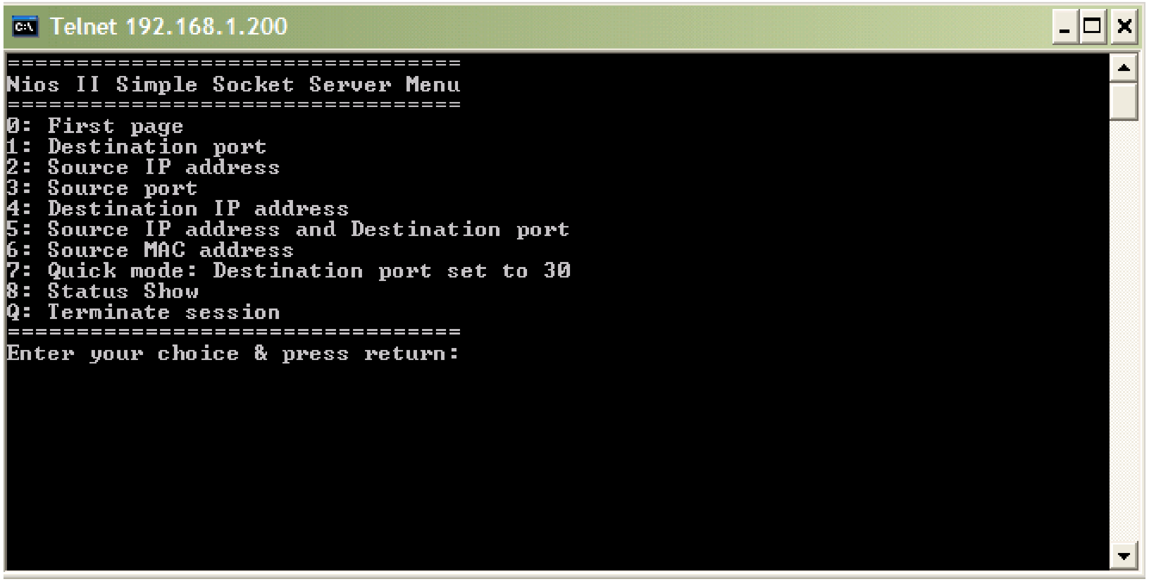
As it was said earlier that real-time configuration aspect of the project is being done by using the Telnet server. The Telnet sever is setup by responding to Telnet request from TCP port 30. Although the common TCP port for Telnet protocol is 23, it would be safer to assign a less-known TCP port for Telnet access to mitigate the chance of finding accustomed open port by attackers.

It would be mandatory to leave the TCP port 30 open in the initialization process either by choosing option seven from the main menu or configuring the destination port 30 along with other desired ports in the first option to be able to access the Telnet server, unless the maximum security measure is in place to block the Telnet server. If the TCP port 30 is blocked either intentionally or unintentionally, it would not be possible to make any configuration changes, or monitor the system status via the Telnet server. In either case, the software or hardware must be reset and initialized again.

In order to Telnet to the *embedded network firewall*, a host PC should have a terminal software ready and then Telnet to the IP address of the server along with port 30. The Telnet server shows the firewall console main menu with the following options:

- *(0) First page*
- *(1) Destination port*
- *(2) Source IP address*
- *(3) Source port*

- *(4) Destination IP address*
- *(5) Source IP address and Destination port*
- *(6) Source MAC address*
- *(7) Quick mode: Destination port set to 30*
- *(8) Status show*
- *(Q) Terminate session*



```
=================================
Nios II Simple Socket Server Menu
=================================
0: First page
1: Destination port
2: Source IP address
3: Source port
4: Destination IP address
5: Source IP address and Destination port
6: Source MAC address
7: Quick mode: Destination port set to 30
8: Status Show
Q: Terminate session
=================================
Enter your choice & press return:
```

Figure 3.12: Firewall Telnet server menu

Figure 3.12 shows the screen shot of the Telnet server. Options "1" to "7" have already been explained in the initialization section. Option "0" takes the user back to first page at any time and option "Q" terminates the session. "Status show" option is used for monitoring purpose and will be explained in the next section.

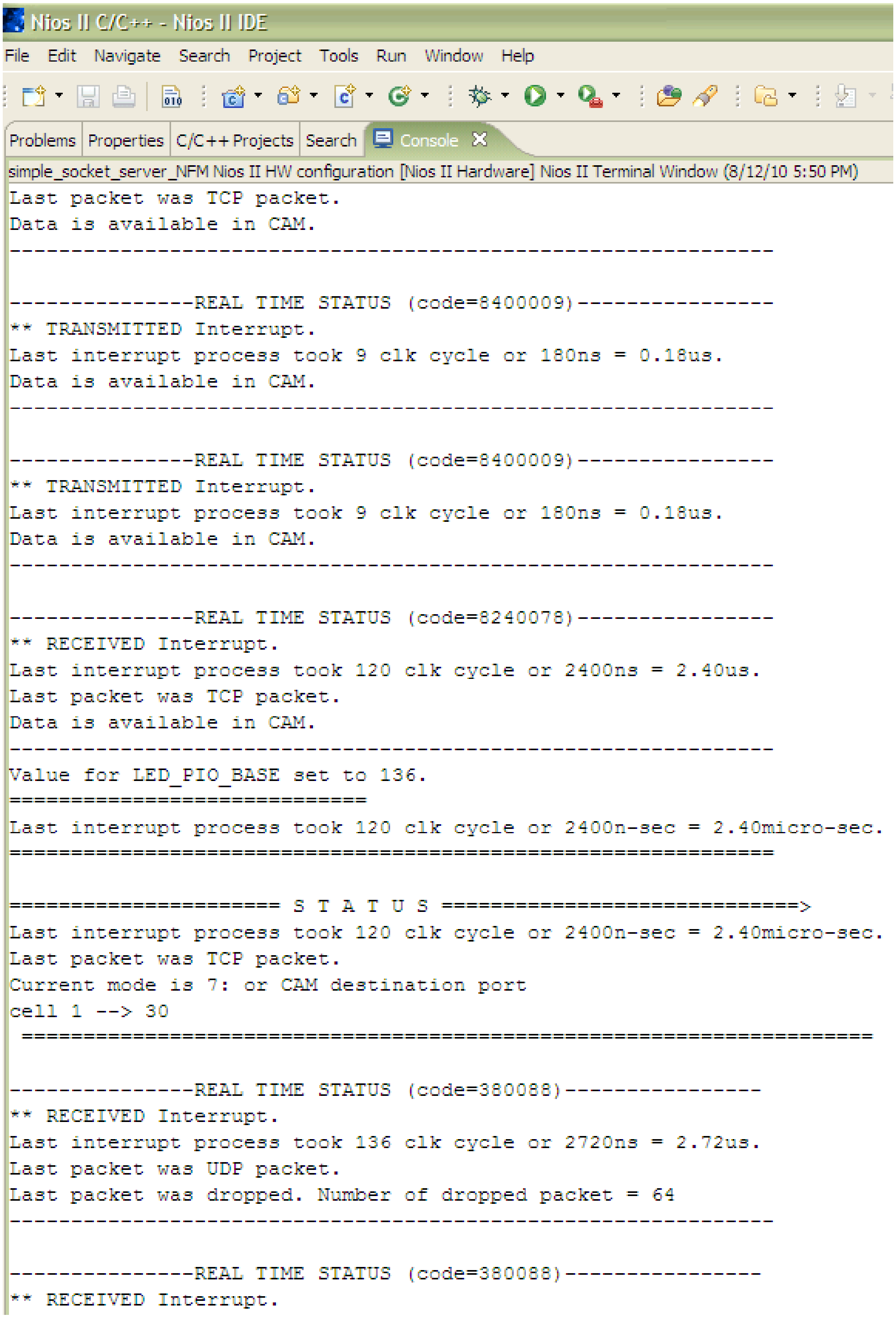
3.2.7 Monitoring

The second function of the Telnet server beside its main purpose is the monitoring system. By choosing the *status show* in the main menu of the firewall console, the following items will be shown:

- Number of hardware clock cycles for the last interrupt process. This item is useful for measuring performance of the firewall .
- Type of the transport layer received packet whether it was TCP or UDP. This option can be disabled to speed up the packet processing.
- Current mode of the firewall.
- Number of entries in the CAM and netmask RAM modules.
- Contents of the CAM and netmask RAM modules; they could be source/destination port number, source/destination IP address or source MAC address.

Another useful monitoring tool has been implemented in the *embedded network firewall* project to show real-time status of the last processed packet. This information is reflected on the Nios II IDE console with the following:

- Real-time status code is the read value of the status PIO in hexadecimal. This is very informative for troubleshooting purpose.
- Type of last interrupt (receive or transmit).
- Number of hardware clock cycles for the last interrupt process. This item is used to measure the firewall performance.
- Type of the last received packet whether it was TCP, UDP, ICMP or ARP. This option can be disabled by the user to speed up the packet processing.
- Number of dropped packets.
- Type of error, if any, in the last received packet including ALIGN, BAD CRC, TOO LONG or TOO SHORT error.


Nios II C/C++ - Nios II IDE
File Edit Navigate Search Project Tools Run Window Help
Problems Properties C/C++ Projects Search Console

simple_socket_server_NFM Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (8/12/10 5:50 PM)

```
Last packet was TCP packet.
Data is available in CAM.
--------------------------------------------------------------


---------------REAL TIME STATUS (code=8400009)----------------
** TRANSMITTED Interrupt.
Last interrupt process took 9 clk cycle or 180ns = 0.18us.
Data is available in CAM.
--------------------------------------------------------------


---------------REAL TIME STATUS (code=8400009)----------------
** TRANSMITTED Interrupt.
Last interrupt process took 9 clk cycle or 180ns = 0.18us.
Data is available in CAM.
--------------------------------------------------------------


---------------REAL TIME STATUS (code=8240078)----------------
** RECEIVED Interrupt.
Last interrupt process took 120 clk cycle or 2400ns = 2.40us.
Last packet was TCP packet.
Data is available in CAM.
--------------------------------------------------------------
Value for LED_PIO_BASE set to 136.
=============================
Last interrupt process took 120 clk cycle or 2400n-sec = 2.40micro-sec.
==============================================================


====================== S T A T U S =============================>
Last interrupt process took 120 clk cycle or 2400n-sec = 2.40micro-sec.
Last packet was TCP packet.
Current mode is 7: or CAM destination port
cell 1 --> 30
 =====================================================================


---------------REAL TIME STATUS (code=380088)----------------
** RECEIVED Interrupt.
Last interrupt process took 136 clk cycle or 2720ns = 2.72us.
Last packet was UDP packet.
Last packet was dropped. Number of dropped packet = 64
--------------------------------------------------------------


---------------REAL TIME STATUS (code=380088)----------------
** RECEIVED Interrupt.
```

Figure 3.13: Status

- CAM data availability.

Figure 3.13 shows the configuration and real-time status. In the coming chapter, the overall performance of the designed firewall will be tested and the test results will be elaborated and compared with two similar firewalls.

Chapter 4

Testing and Results

In this chapter the testing procedure and results of the project are presented. *Wireshark*, *Tcpdump*, *ping*, *Nmap* and Telnet client tools were used to test the embedded network firewall. These tools and their functionalities are explained, test method is described and test results are presented after the test procedure.

4.1 Test tools

There are many hardware and software tools available for testing, debugging and troubleshooting networking devices and protocols. Basically these tools are mostly used for packet capturing and analyzing. In this section, four very useful and popular software tools (used for the network firewall testing) are explained.

4.1.1 Wireshark

Wireshark is a cross platform open source utility, released under the term of *GNU General Public Licence* for free, for network protocol analyzing and troubleshooting [25]. It runs on Unix-like operating systems as well as Microsoft Windows platform. Wireshark offers Graphical User Interface (GUI) packet/frame visualizing with filtering and sorting features. When it runs, it applies the promiscuous mode on running network interface to capture all sorts of live network packets/frames.

4.1.2 Tcpdump

Tcpdump is an another open source and free to use, under term of BSD License, packet analyzer utility runs at Command Line Interface (CLI) on Unix-like operating systems for privilege user like superuser (root account) [32]. Tcpdump offers network packet capturing on a network interface for troubleshooting and debugging various network protocols.

4.1.3 Ping

Ping is a very popular network troubleshooting utility for reachability testing of a host or device connected to TCP/IP based network. Ping sends ICMP echo request to the destination device and waits for an ICMP reply packet, and then calculates the round trip time and packet loss and shows summary statistics [33].

4.1.4 Nmap

Network MAPper (Nmap) is a network security scanner for discovering device and their running services on a TCP/IP network, and it has been one of the main tools in academic researches involving network security. Nmap can do host discovery, port scanning, OS and version detection on a target device [32].

4.2 Testing procedure

A small size network was setup including two host PCs, the Altera board and a network hub. It should be mentioned that in order to capture all of the traffic in a network, the network devices should not be connected to each other through a *network switch* since a switch sends traffic only to a switch port to which the traffic is intended for and all other traffic for other devices are filtered. But a network hub broadcasts all of the traffic to all of the ports in the device and therefore any host in the network can see the traffic. A switch intelligently investigates the Ethernet destination address in all of the incoming traffic to the switch and builds up a MAC address table based on connected devices to each switch port.

At first, the FPGA is uploaded with the hardware design and then the application is compiled in the Nios IDE and loaded to the Nios II. *get_ip_addr()* function in *network_utilities.c* file, in software directory, is edited manually to set the IP address, network mask and network gateway for the firewall instead of DHCP default settings. Figure 4.1 shows the topology of the network setup.

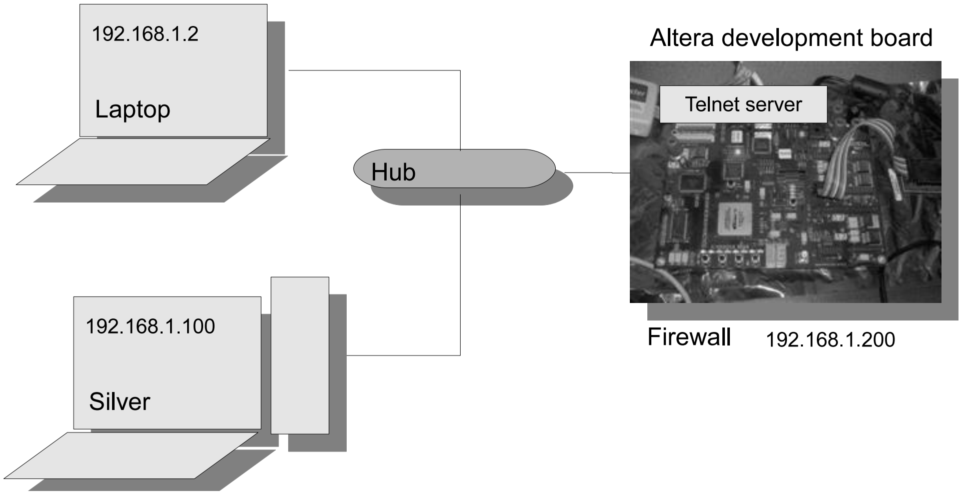


Figure 4.1: Test network setup topology

Table 4.1: IP and MAC address table of the device in the testing scenario

Device name	IP address	Network mask	Gateway	MAC address
Altera board	192.168.1.200	255.255.255.0	192.168.1.1	00:07:ED:0D:09:BC
Laptop	192.168.1.2	255.255.255.0	192.168.1.1	00:13:46:34:81:90
Silver PC	192.168.1.100	255.255.255.0	192.168.1.1	00:04:75:C7:DE:B6

Table 4.1 shows the IP address, network mask and default gateway for all the three devices in our network. The first test after the setup is to *ping* the firewall

to make sure that it is alive in the network and can respond to the ICMP request. Figure 4.2 shows the ICMP request and reply from the firewall.

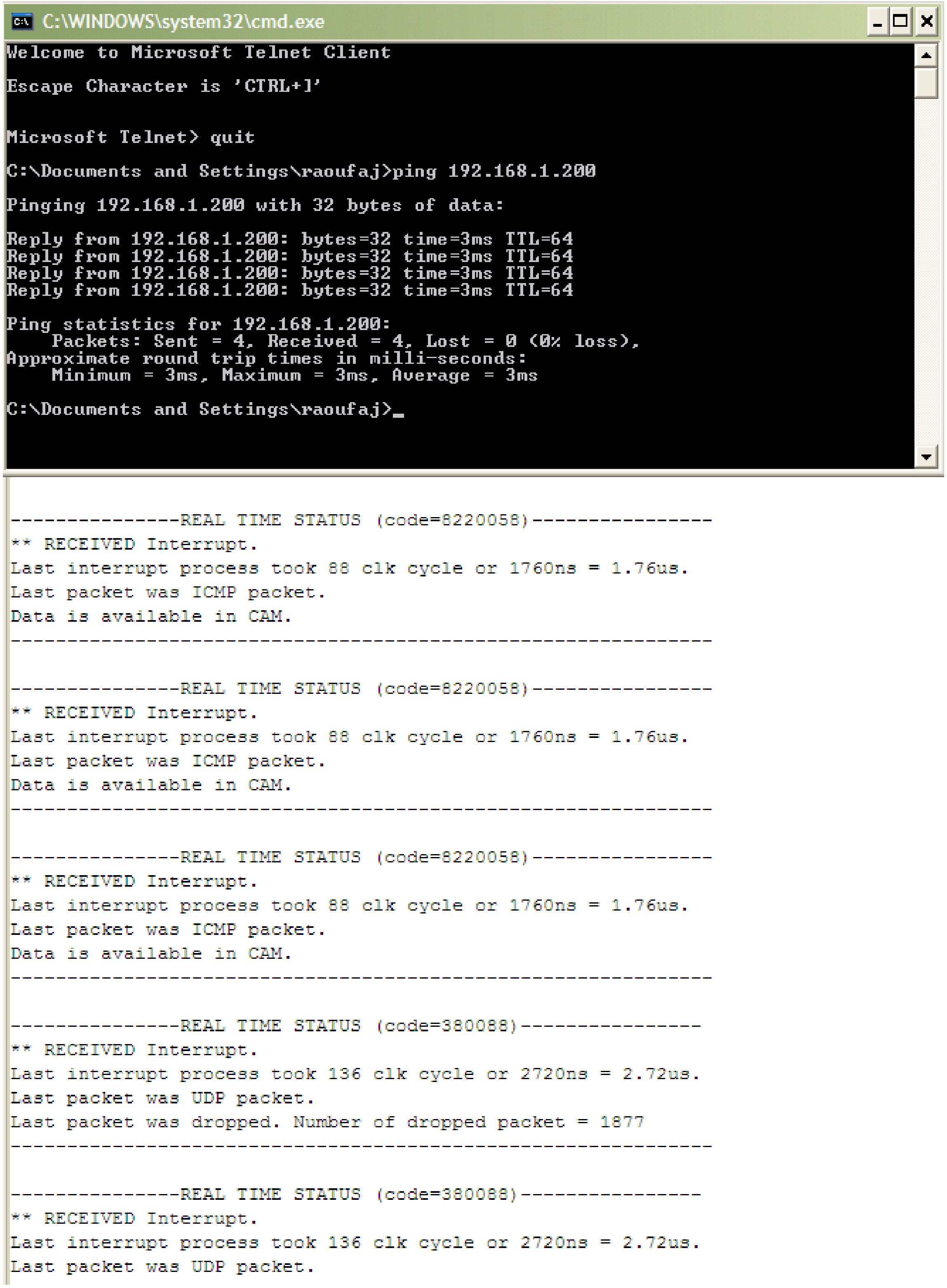
As it can be seen by real time status monitoring in the Nios IDE console, each ICMP packet takes 88 clock cycles (1,760 nano second at 50 MHz) to be processed. Also LED3 illuminates when the ICMP packet arrives.

4.2.1 Destination and source port mode

The simplest way of firewall setup is to choose the 7th option from the initialization menu. This option sets the firewall in the destination port number mode and enters port 30 as permitted port in the destination port CAM. Obviously this mode lets any TCP or UDP packet with destination port number 30 pass through the firewall and reach to the other side of the network. Now the status of the firewall by Telnetting to the firewall IP address can be checked. Figure 4.3 shows the Telnet server menu along with real-time status and configuration status of the firewall.

Real-time status shows the last interrupt was a received TCP packet and it took 118 clock cycles (2,360 nano second at 50 MHz) to process a legitimate packet and extra 6 clock cycles to drop an unauthorized packet. Also the configuration status shows the current mode of the firewall is 7 (quick mode or destination port number 30) and one cell of CAM memory is occupied by value 30. In the next test, monitoring module has been disabled to speed up the packet processing. Figure 4.4 shows that it took 92 clock cycles (1,840 nano second at 50 MHz) to permit a valid packet and 98 clock cycles (1,960 nano second at 50 MHz) to drop an invalid packet.

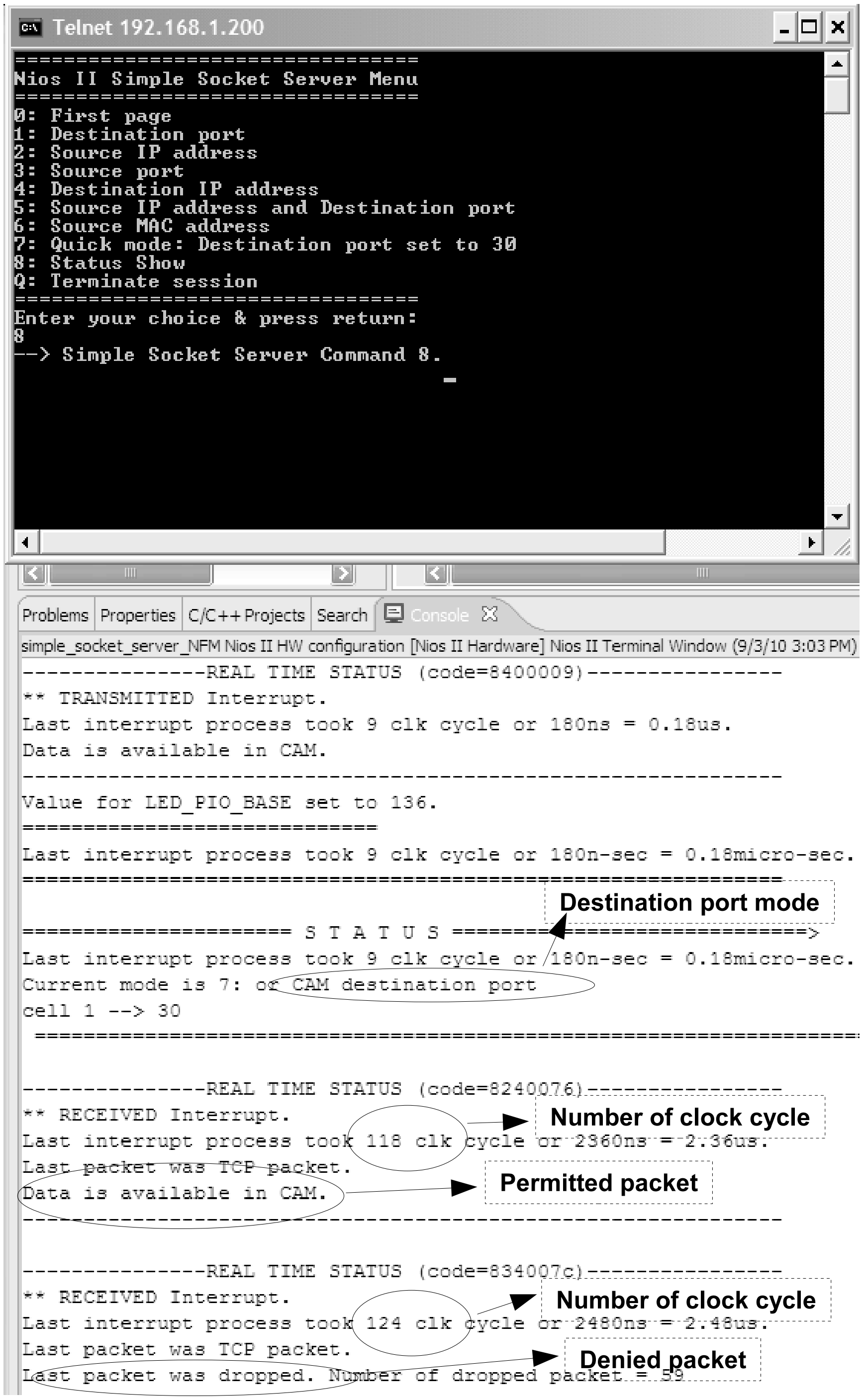
Figure 4.5 shows the result of Nmap scanning of the firewall in the network when port 30 is open and when it is blocked. The first attempt of Nmap scan shows that port 30 is open, and the next Nmap scan after flushing out port 30 from the CAM does not reveal port 30 as an open port. Figure 4.6 shows the screen shot of the Tcpdump during the Telnet activity to the Telnet server from Silver PC. The



```
Welcome to Microsoft Telnet Client

Escape Character is 'CTRL+]'


Microsoft Telnet> quit

C:\Documents and Settings\raoufaj>ping 192.168.1.200

Pinging 192.168.1.200 with 32 bytes of data:

Reply from 192.168.1.200: bytes=32 time=3ms TTL=64
Reply from 192.168.1.200: bytes=32 time=3ms TTL=64
Reply from 192.168.1.200: bytes=32 time=3ms TTL=64
Reply from 192.168.1.200: bytes=32 time=3ms TTL=64

Ping statistics for 192.168.1.200:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 3ms, Maximum = 3ms, Average = 3ms

C:\Documents and Settings\raoufaj>_
```

```
---------------REAL TIME STATUS (code=8220058)----------------
** RECEIVED Interrupt.
Last interrupt process took 88 clk cycle or 1760ns = 1.76us.
Last packet was ICMP packet.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=8220058)----------------
** RECEIVED Interrupt.
Last interrupt process took 88 clk cycle or 1760ns = 1.76us.
Last packet was ICMP packet.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=8220058)----------------
** RECEIVED Interrupt.
Last interrupt process took 88 clk cycle or 1760ns = 1.76us.
Last packet was ICMP packet.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=380088)----------------
** RECEIVED Interrupt.
Last interrupt process took 136 clk cycle or 2720ns = 2.72us.
Last packet was UDP packet.
Last packet was dropped. Number of dropped packet = 1877
--------------------------------------------------------------

---------------REAL TIME STATUS (code=380088)----------------
** RECEIVED Interrupt.
Last interrupt process took 136 clk cycle or 2720ns = 2.72us.
Last packet was UDP packet.
```

Figure 4.2: ICMP request and reply and real time status

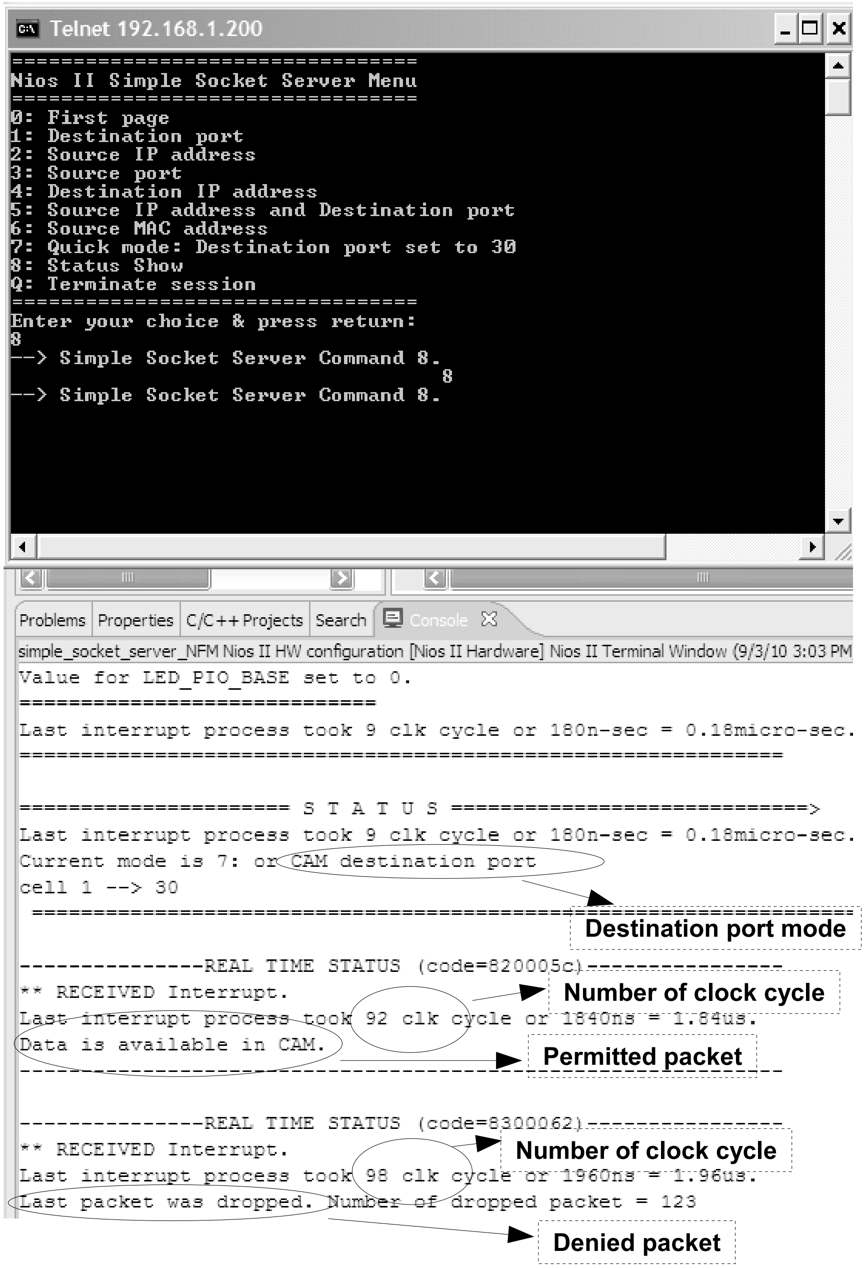


```
=================================
Nios II Simple Socket Server Menu
=================================
0: First page
1: Destination port
2: Source IP address
3: Source port
4: Destination IP address
5: Source IP address and Destination port
6: Source MAC address
7: Quick mode: Destination port set to 30
8: Status Show
Q: Terminate session
=================================
Enter your choice & press return:
8
--> Simple Socket Server Command 8.
```

simple_socket_server_NFM Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (9/3/10 3:03 PM)

```
---------------REAL TIME STATUS (code=8400009)----------------
** TRANSMITTED Interrupt.
Last interrupt process took 9 clk cycle or 180ns = 0.18us.
Data is available in CAM.
--------------------------------------------------------------
Value for LED_PIO_BASE set to 136.
=============================
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
==============================================================

====================== S T A T U S ===========================>
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
Current mode is 7: or CAM destination port
cell 1 --> 30
 ====================================================================

---------------REAL TIME STATUS (code=8240076)----------------
** RECEIVED Interrupt.
Last interrupt process took 118 clk cycle or 2360ns = 2.36us.
Last packet was TCP packet.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=834007c)----------------
** RECEIVED Interrupt.
Last interrupt process took 124 clk cycle or 2480ns = 2.48us.
Last packet was TCP packet.
Last packet was dropped. Number of dropped packet = 59
```

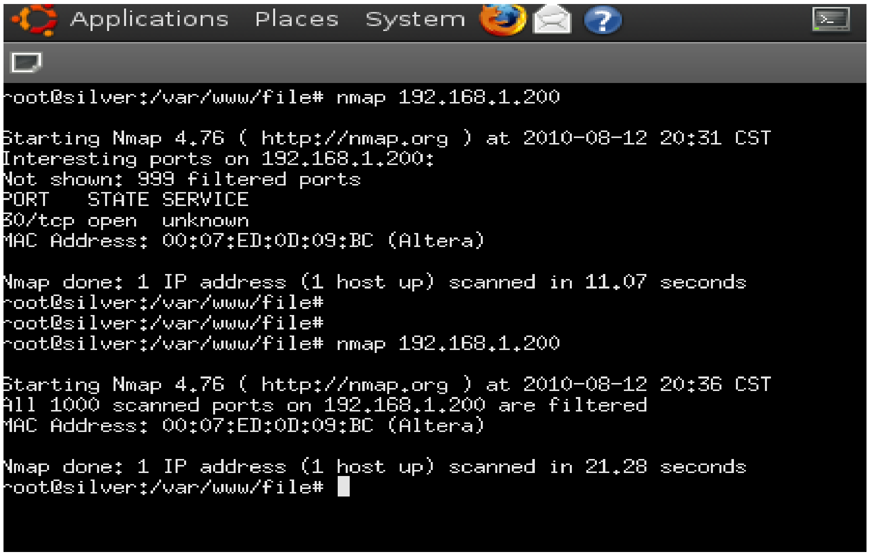
Figure 4.3: Firewall status in the default mode with monitoring



```
=================================
Nios II Simple Socket Server Menu
=================================
0: First page
1: Destination port
2: Source IP address
3: Source port
4: Destination IP address
5: Source IP address and Destination port
6: Source MAC address
7: Quick mode: Destination port set to 30
8: Status Show
Q: Terminate session
=================================
Enter your choice & press return:
8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
```

```
Value for LED_PIO_BASE set to 0.
=============================
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
==============================================================


====================== S T A T U S ============================>
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
Current mode is 7: or CAM destination port
cell 1 --> 30
 ==============================================================

---------------REAL TIME STATUS (code=820005c)----------------
** RECEIVED Interrupt.
Last interrupt process took 92 clk cycle or 1840ns = 1.84us.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=8300062)----------------
** RECEIVED Interrupt.
Last interrupt process took 98 clk cycle or 1960ns = 1.96us.
Last packet was dropped. Number of dropped packet = 123
```

Figure 4.4: Firewall status in the default mode without monitoring



```
root@silver:/var/www/file# nmap 192.168.1.200

Starting Nmap 4.76 ( http://nmap.org ) at 2010-08-12 20:31 CST
Interesting ports on 192.168.1.200:
Not shown: 999 filtered ports
PORT   STATE SERVICE
80/tcp open  unknown
MAC Address: 00:07:ED:0D:09:BC (Altera)

Nmap done: 1 IP address (1 host up) scanned in 11.07 seconds
root@silver:/var/www/file#
root@silver:/var/www/file#
root@silver:/var/www/file# nmap 192.168.1.200

Starting Nmap 4.76 ( http://nmap.org ) at 2010-08-12 20:36 CST
All 1000 scanned ports on 192.168.1.200 are filtered
MAC Address: 00:07:ED:0D:09:BC (Altera)

Nmap done: 1 IP address (1 host up) scanned in 21.28 seconds
root@silver:/var/www/file#
```

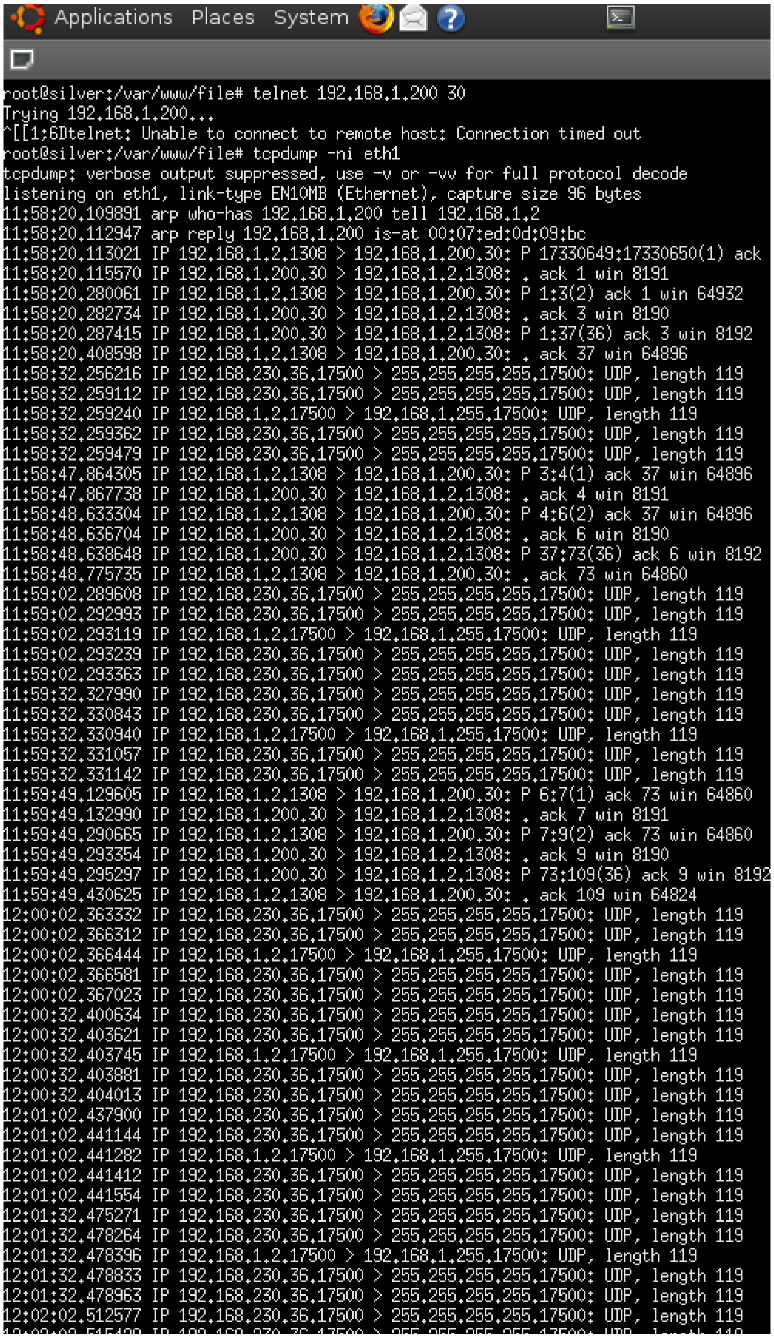
Figure 4.5: Nmap scanning of the firewall in destination port mode

destination port mode results are not shown in here since it is similar to the source port number mode.

4.2.2 Source and destination IP address mode

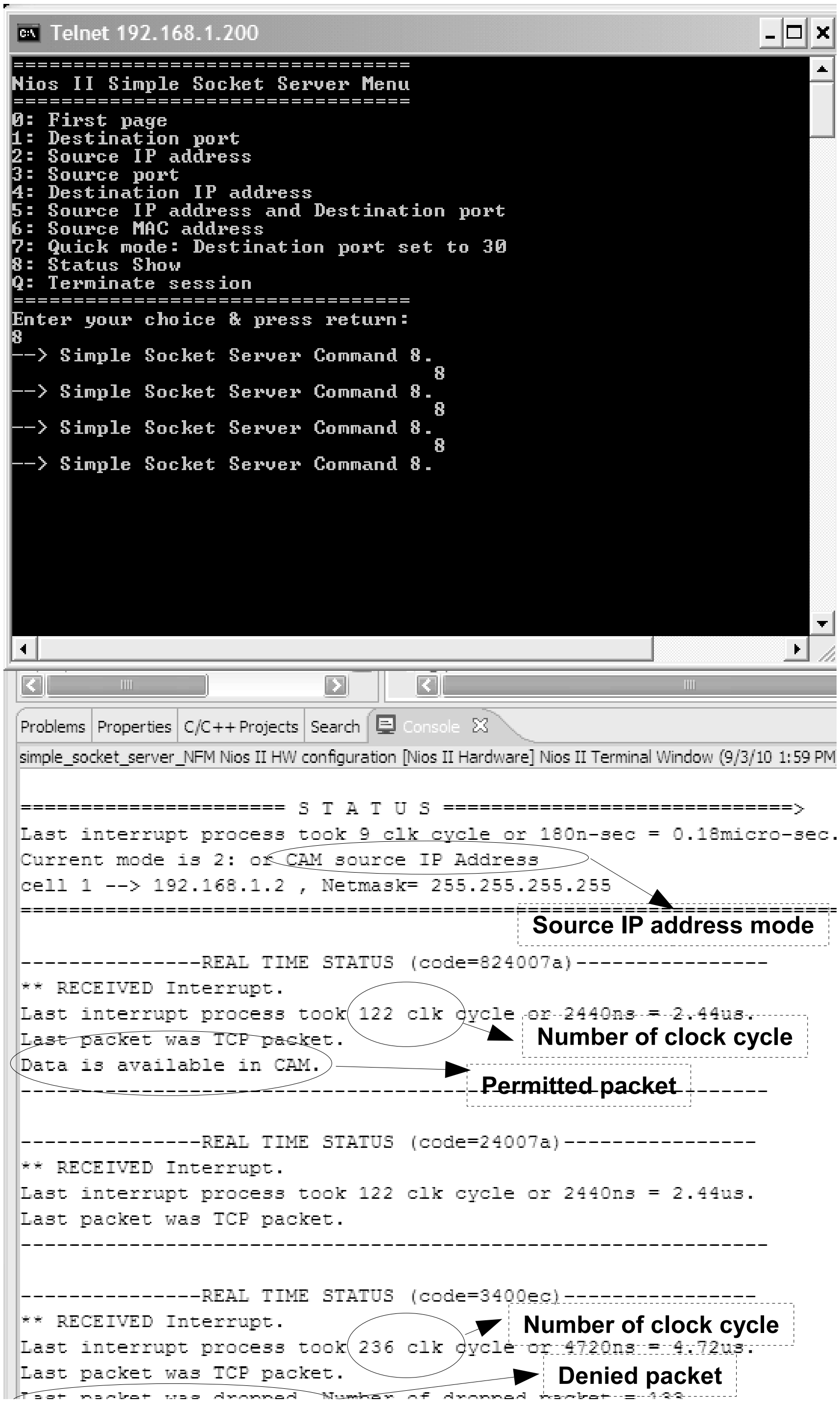
After choosing source IP address mode either at initialization stage or later on in the configuration stage from the Telnet server firewall menu, the laptop IP address is entered as an authorized PC to test the source IP address permitting and blocking mode of the firewall.

Figure 4.7 shows that the laptop was permitted to access the Telnet server menu while the silver PC was denied. This figure also shows the current mode of the firewall and contents of the source IP address CAM. Also one can see that the source IP address processing takes 122 clock cycles (2,440 nano second at 50 MHz). If the packet is not allowed and has to be dropped, it takes 236 clock cycles (4,720 nano second) which is the worst case scenario for processing a packet in this firewall. This takes more time because each memory cell in the IP address netmask RAM must be



```
root@silver:/var/www/file# telnet 192.168.1.200 30
Trying 192.168.1.200...
^[[1;6Dtelnet: Unable to connect to remote host: Connection timed out
root@silver:/var/www/file# tcpdump -ni eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
11:58:20.109891 arp who-has 192.168.1.200 tell 192.168.1.2
11:58:20.112947 arp reply 192.168.1.200 is-at 00:07:ed:0d:09:bc
11:58:20.113021 IP 192.168.1.2.1308 > 192.168.1.200.30: P 17330649:17330650(1) ack
11:58:20.115570 IP 192.168.1.200.30 > 192.168.1.2.1308: . ack 1 win 8191
11:58:20.280061 IP 192.168.1.2.1308 > 192.168.1.200.30: P 1:3(2) ack 1 win 64932
11:58:20.282734 IP 192.168.1.200.30 > 192.168.1.2.1308: . ack 3 win 8190
11:58:20.287415 IP 192.168.1.200.30 > 192.168.1.2.1308: P 1:37(36) ack 3 win 8192
11:58:20.408598 IP 192.168.1.2.1308 > 192.168.1.200.30: . ack 37 win 64896
11:58:32.256216 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:58:32.259112 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:58:32.259240 IP 192.168.1.2.17500 > 192.168.1.255.17500: UDP, length 119
11:58:32.259362 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:58:32.259479 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:58:47.864305 IP 192.168.1.2.1308 > 192.168.1.200.30: P 3:4(1) ack 37 win 64896
11:58:47.867738 IP 192.168.1.200.30 > 192.168.1.2.1308: . ack 4 win 8191
11:58:48.633304 IP 192.168.1.2.1308 > 192.168.1.200.30: P 4:6(2) ack 37 win 64896
11:58:48.636704 IP 192.168.1.200.30 > 192.168.1.2.1308: . ack 6 win 8190
11:58:48.638648 IP 192.168.1.200.30 > 192.168.1.2.1308: P 37:73(36) ack 6 win 8192
11:58:48.775735 IP 192.168.1.2.1308 > 192.168.1.200.30: . ack 73 win 64860
11:59:02.289608 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:59:02.292993 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:59:02.293119 IP 192.168.1.2.17500 > 192.168.1.255.17500: UDP, length 119
11:59:02.293239 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:59:02.293363 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:59:32.327990 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:59:32.330843 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:59:32.330940 IP 192.168.1.2.17500 > 192.168.1.255.17500: UDP, length 119
11:59:32.331057 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:59:32.331142 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
11:59:49.129605 IP 192.168.1.2.1308 > 192.168.1.200.30: P 6:7(1) ack 73 win 64860
11:59:49.132990 IP 192.168.1.200.30 > 192.168.1.2.1308: . ack 7 win 8191
11:59:49.290665 IP 192.168.1.2.1308 > 192.168.1.200.30: P 7:9(2) ack 73 win 64860
11:59:49.293354 IP 192.168.1.200.30 > 192.168.1.2.1308: . ack 9 win 8190
11:59:49.295297 IP 192.168.1.200.30 > 192.168.1.2.1308: P 73:109(36) ack 9 win 8192
11:59:49.430625 IP 192.168.1.2.1308 > 192.168.1.200.30: . ack 109 win 64824
12:00:02.363332 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:00:02.366312 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:00:02.366444 IP 192.168.1.2.17500 > 192.168.1.255.17500: UDP, length 119
12:00:02.366581 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:00:02.367023 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:00:32.400634 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:00:32.403621 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:00:32.403745 IP 192.168.1.2.17500 > 192.168.1.255.17500: UDP, length 119
12:00:32.403881 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:00:32.404013 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:01:02.437900 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:01:02.441144 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:01:02.441282 IP 192.168.1.2.17500 > 192.168.1.255.17500: UDP, length 119
12:01:02.441412 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:01:02.441554 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:01:32.475271 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:01:32.478264 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:01:32.478396 IP 192.168.1.2.17500 > 192.168.1.255.17500: UDP, length 119
12:01:32.478833 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:01:32.478963 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
12:02:02.512577 IP 192.168.230.36.17500 > 255.255.255.255.17500: UDP, length 119
```

Figure 4.6: Tcpdump screenshot



```
=================================
Nios II Simple Socket Server Menu
=================================
0: First page
1: Destination port
2: Source IP address
3: Source port
4: Destination IP address
5: Source IP address and Destination port
6: Source MAC address
7: Quick mode: Destination port set to 30
8: Status Show
Q: Terminate session
=================================
Enter your choice & press return:
8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
```

simple_socket_server_NFM Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (9/3/10 1:59 PM

```
===================== S T A T U S ============================>
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
Current mode is 2: or CAM source IP Address
cell 1 --> 192.168.1.2 , Netmask= 255.255.255.255
======================================================================

---------------REAL TIME STATUS (code=824007a)----------------
** RECEIVED Interrupt.
Last interrupt process took 122 clk cycle or 2440ns = 2.44us.
Last packet was TCP packet.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=24007a)----------------
** RECEIVED Interrupt.
Last interrupt process took 122 clk cycle or 2440ns = 2.44us.
Last packet was TCP packet.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=3400ec)----------------
** RECEIVED Interrupt.
Last interrupt process took 236 clk cycle or 4720ns = 4.72us.
Last packet was TCP packet.
Last packet was dropped. Number of dropped packet = 133
```

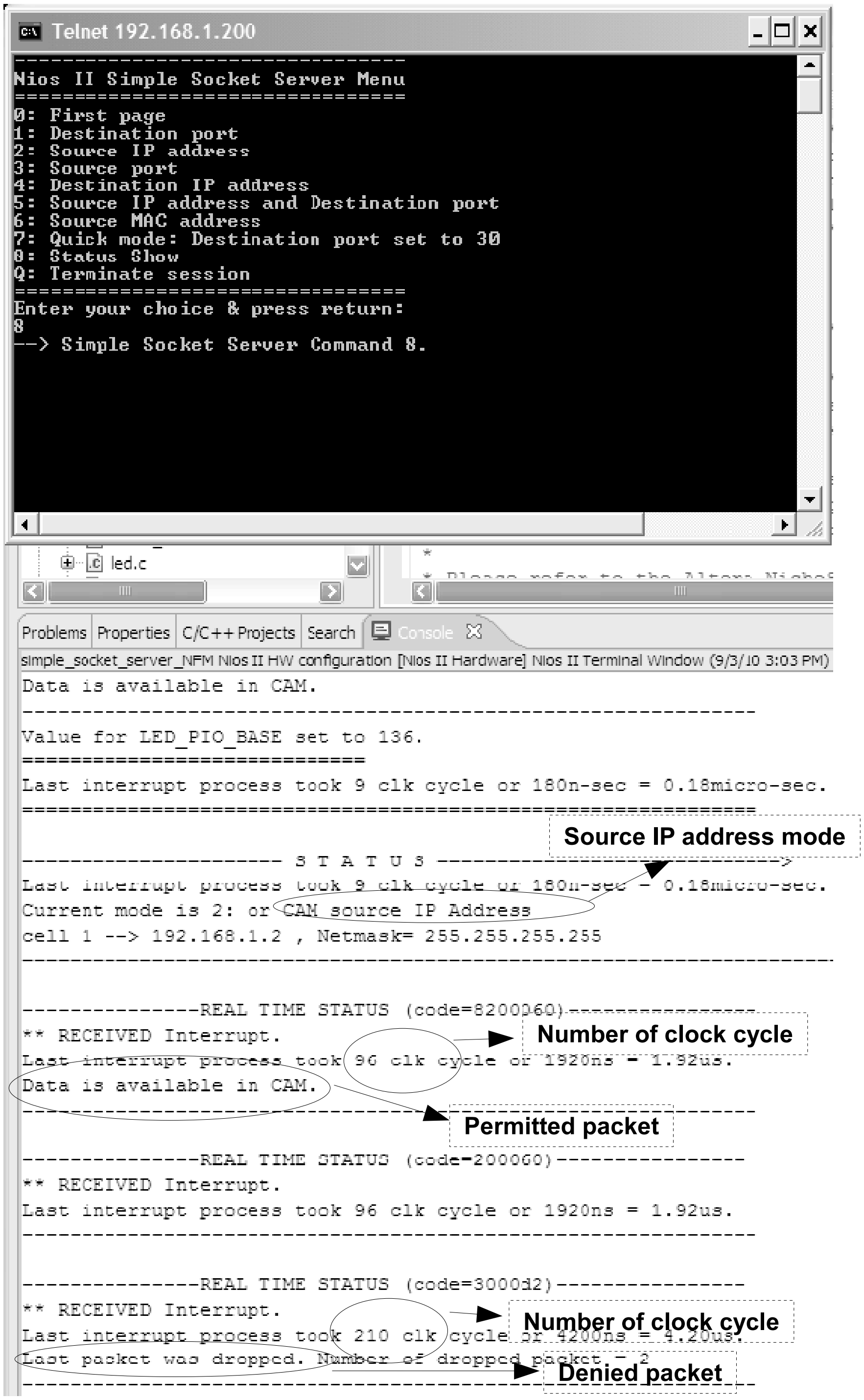
Figure 4.7: Firewall in source IP address mode with monitoring

checked against the IP address CAM for a match. Figure 4.8 shows the test without monitoring module; it takes 96 clock cycles (1,920 nano second) for a valid packet passing and 210 clock cycles (4,200 nano second) for dropping a packet. Destination IP address follows the same procedure and therefore takes the same time for packet processing.

4.2.3 Source IP address and destination port number mode

In this mode, at first each packet is investigated for source IP address matching. If the source IP address of the packet is found in the source IP CAM, the process goes to the next step which is the destination port number matching. If the destination port number of the packet is also found in the destination port number CAM then the packet is permitted to pass through the firewall. To test this, the laptop IP address has been configured in the firewall along with destination port 30. Figure 4.9 shows the laptop can access the Telnet server while the Silver PC is not permitted.

The result shows it takes 153 clock cycles (3,060 nano second at 50 MHz) for the valid source IP address along with the valid destination port number. However, for an invalid destination port along with a valid IP address packet, it takes 169 clock cycles (3,380 nano second at 50 MHz). Figure 4.10 shows 127 and 210 clock cycles for a valid packet and an invalid packet processing respectively when the firewall performs without monitoring functionality. And again, like the source IP address worst case situation, 236 clock cycles are exhausted for dropping an unauthorized packet by the firewall since it has to search all of the netmask RAM cell memories against the source IP address CAM for a MATCH.

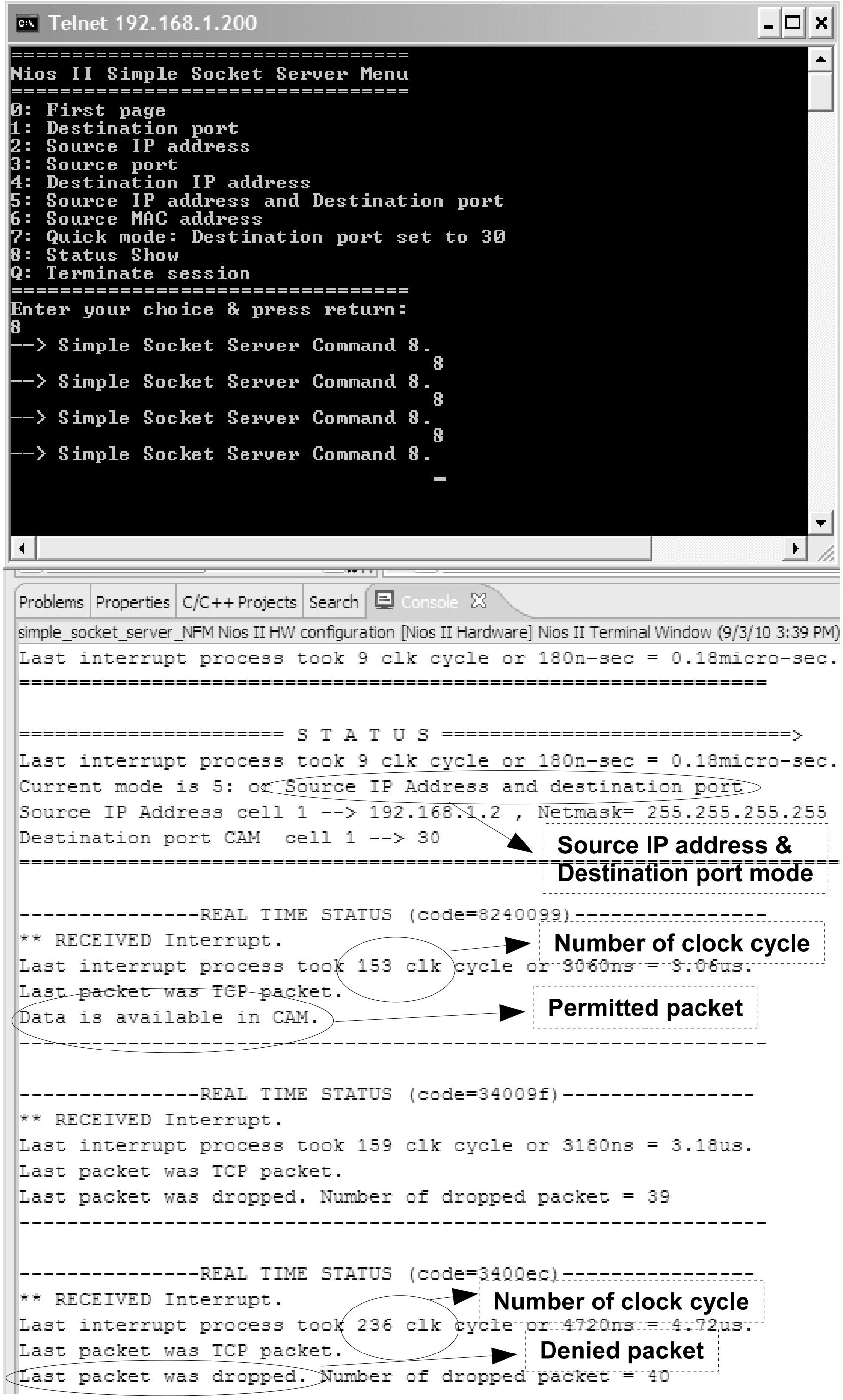


```
---------------------------------
Nios II Simple Socket Server Menu
=================================
0: First page
1: Destination port
2: Source IP address
3: Source port
4: Destination IP address
5: Source IP address and Destination port
6: Source MAC address
7: Quick mode: Destination port set to 30
8: Status Show
Q: Terminate session
=================================
Enter your choice & press return:
8
--> Simple Socket Server Command 8.
```

simple_socket_server_NFM Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (9/3/10 3:03 PM)

```
Data is available in CAM.
--------------------------------------------------------------
Value for LED_PIO_BASE set to 136.
=============================
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
==============================================================

---------------------- S T A T U S ---------------------------->
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
Current mode is 2: or CAM source IP Address
cell 1 --> 192.168.1.2 , Netmask= 255.255.255.255
-------------------------------------------------------------------------


---------------REAL TIME STATUS (code=8200060)----------------
** RECEIVED Interrupt.
Last interrupt process took 96 clk cycle or 1920ns = 1.92us.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=200060)----------------
** RECEIVED Interrupt.
Last interrupt process took 96 clk cycle or 1920ns = 1.92us.
--------------------------------------------------------------


---------------REAL TIME STATUS (code=3000d2)----------------
** RECEIVED Interrupt.
Last interrupt process took 210 clk cycle or 4200ns = 4.20us.
Last packet was dropped. Number of dropped packet = 2
--------------------------------------------------------------
```

Figure 4.8: Firewall in source IP address mode without monitoring

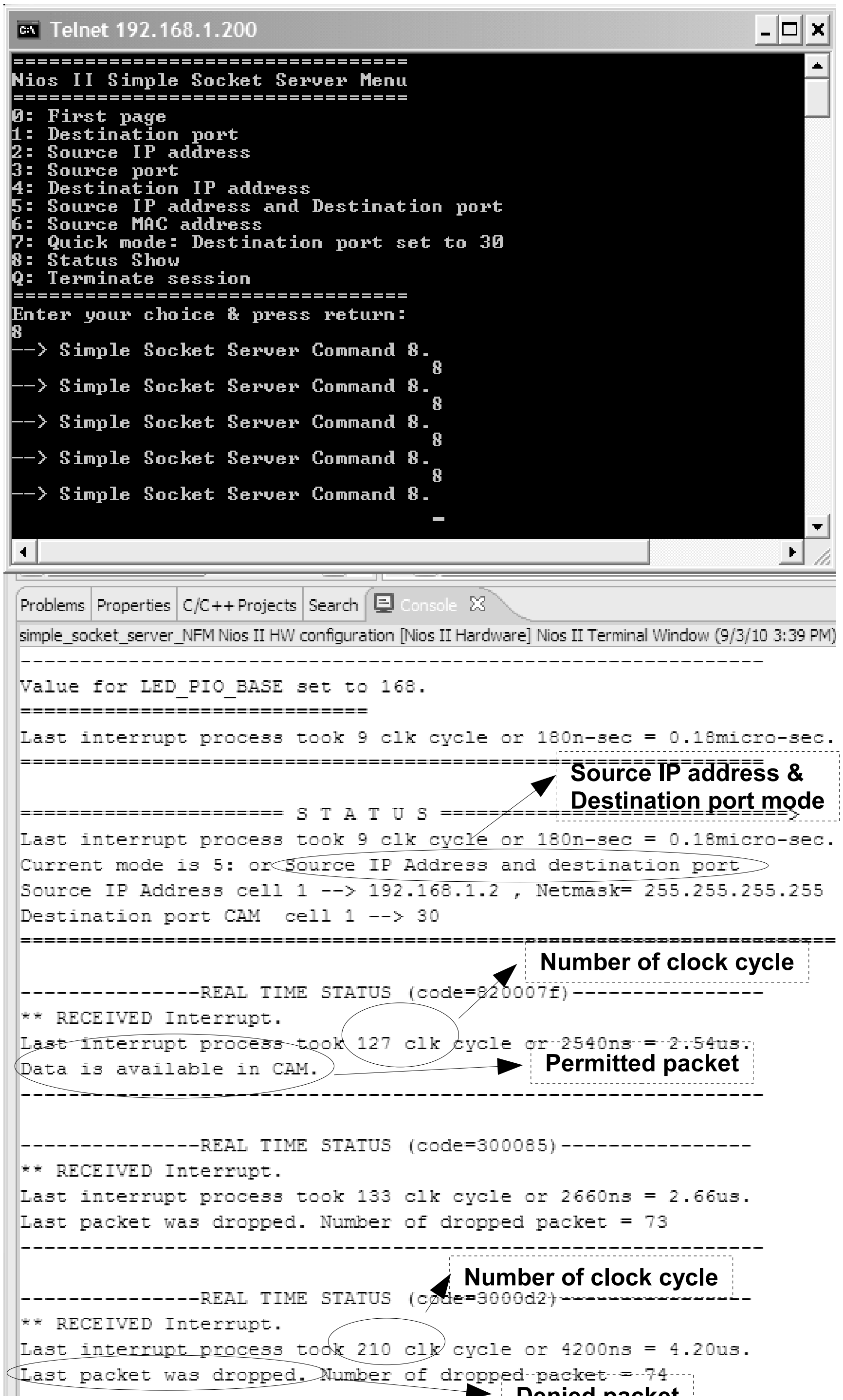


Telnet 192.168.1.200
Problems
Properties
C/C++ Projects
Search
Console
simple_socket_server_NFM Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (9/3/10 3:39 PM)
Source IP address & Destination port mode
Number of clock cycle
Permitted packet
Number of clock cycle
Denied packet

```
=================================
Nios II Simple Socket Server Menu
=================================
0: First page
1: Destination port
2: Source IP address
3: Source port
4: Destination IP address
5: Source IP address and Destination port
6: Source MAC address
7: Quick mode: Destination port set to 30
8: Status Show
Q: Terminate session
=================================
Enter your choice & press return:
8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
```

```
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
==============================================================


====================== S T A T U S ============================>
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
Current mode is 5: or Source IP Address and destination port
Source IP Address cell 1 --> 192.168.1.2 , Netmask= 255.255.255.255
Destination port CAM  cell 1 --> 30
====================================================================

---------------REAL TIME STATUS (code=8240099)----------------
** RECEIVED Interrupt.
Last interrupt process took 153 clk cycle or 3060ns = 3.06us.
Last packet was TCP packet.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=34009f)----------------
** RECEIVED Interrupt.
Last interrupt process took 159 clk cycle or 3180ns = 3.18us.
Last packet was TCP packet.
Last packet was dropped. Number of dropped packet = 39
--------------------------------------------------------------

---------------REAL TIME STATUS (code=3400ec)----------------
** RECEIVED Interrupt.
Last interrupt process took 236 clk cycle or 4720ns = 4.72us.
Last packet was TCP packet.
Last packet was dropped. Number of dropped packet = 40
```

Figure 4.9: Firewall in source IP address and destination port number mode with monitoring



```
================================
Nios II Simple Socket Server Menu
================================
0: First page
1: Destination port
2: Source IP address
3: Source port
4: Destination IP address
5: Source IP address and Destination port
6: Source MAC address
7: Quick mode: Destination port set to 30
8: Status Show
Q: Terminate session
================================
Enter your choice & press return:
8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
```

```
--------------------------------------------------------------
Value for LED_PIO_BASE set to 168.
=============================
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
==============================================================


====================== S T A T U S ==========================>
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
Current mode is 5: or Source IP Address and destination port
Source IP Address cell 1 --> 192.168.1.2 , Netmask= 255.255.255.255
Destination port CAM  cell 1 --> 30
====================================================================


---------------REAL TIME STATUS (code=820007f)----------------
** RECEIVED Interrupt.
Last interrupt process took 127 clk cycle or 2540ns = 2.54us.
Data is available in CAM.
--------------------------------------------------------------


---------------REAL TIME STATUS (code=300085)----------------
** RECEIVED Interrupt.
Last interrupt process took 133 clk cycle or 2660ns = 2.66us.
Last packet was dropped. Number of dropped packet = 73
--------------------------------------------------------------


---------------REAL TIME STATUS (code=3000d2)----------------
** RECEIVED Interrupt.
Last interrupt process took 210 clk cycle or 4200ns = 4.20us.
Last packet was dropped. Number of dropped packet = 74
```

Figure 4.10: Firewall in source IP address and destination port number mode without monitoring

4.2.4 Source MAC address mode

This mode is tested by configuring the Silver PC MAC address as a culprit entry in the MAC address CAM while any other PC could pass the firewall. Figure 4.11 shows that the processing time for a valid request (laptop) takes 122 clock cycles while an invalid access (the Silver PC) takes 128 clock cycle to be dropped. If the monitoring module is disabled, Figure 4.12 illustrates that it takes 96 clock cycles to drop a frame if the source MAC address is in the CAM module and 102 clock cycles to pass a frame when the source MAC address is not available in the CAM module.

Figure 4.13 shows the communication between Silver PC and the firewall while the Laptop is banned.

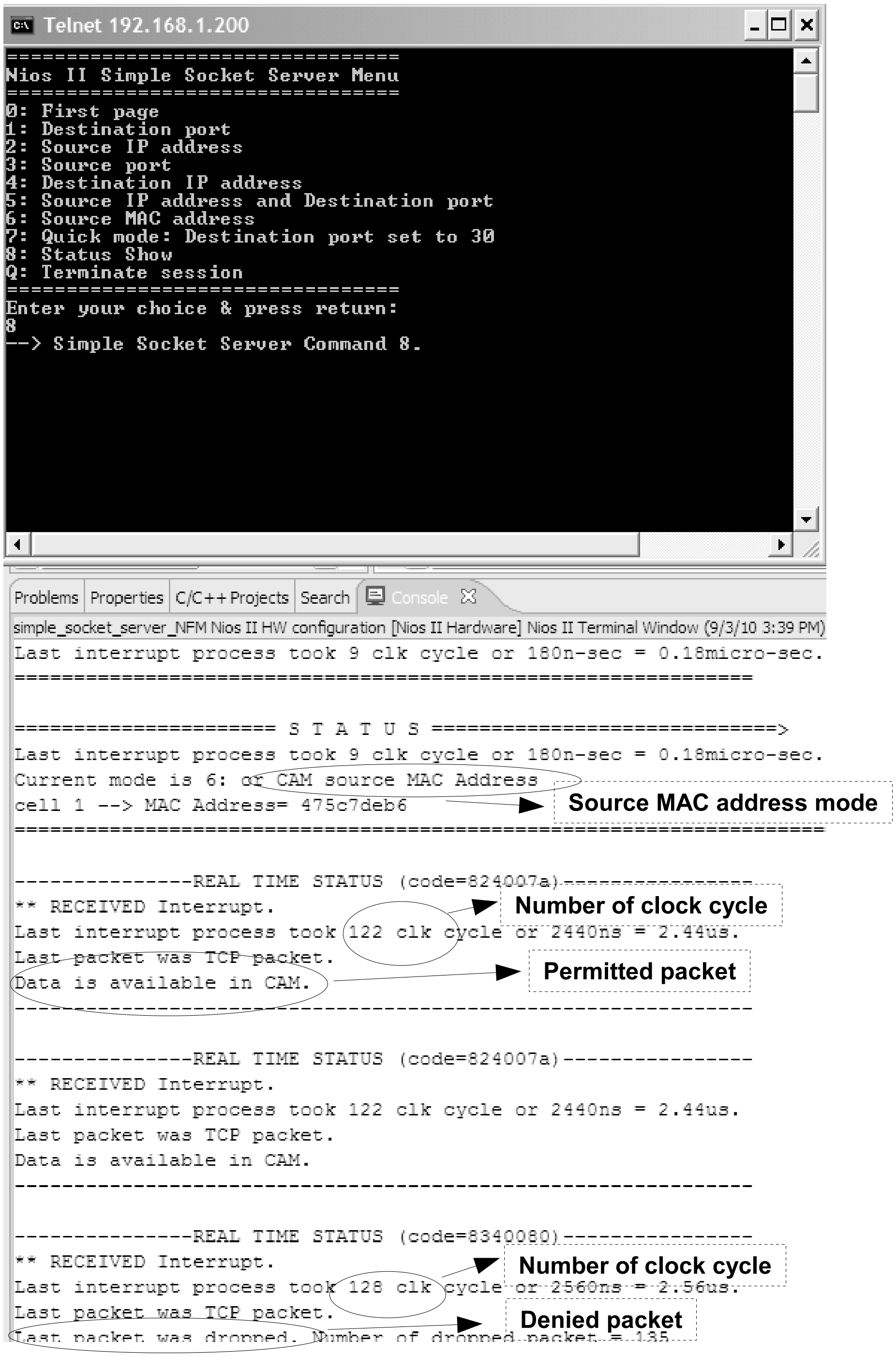
4.3 Results

So far, we have seen all of the packet processing time for different modes in the network firewall design. Table 4.2 has been collected with the timing result data for packet transmission interrupt, packet reception interrupt, ARP packet and ICMP packet processing time.

Table 4.2: Interrupt processing time

Interrupt type	**Clock cycle**	**processing time**
Transmission interrupt	9	180 ns
Receive interrupt	varies	varies
ARP	62	1,240 ns
ICMP	88	1,760 ns

Next table, Table 4.3, specifically shows the timing data result for different modes in the network firewall design. The processing time takes between 1,840 and 4,200 nano seconds. This processing time is not causing any substantial delay for packet processing.

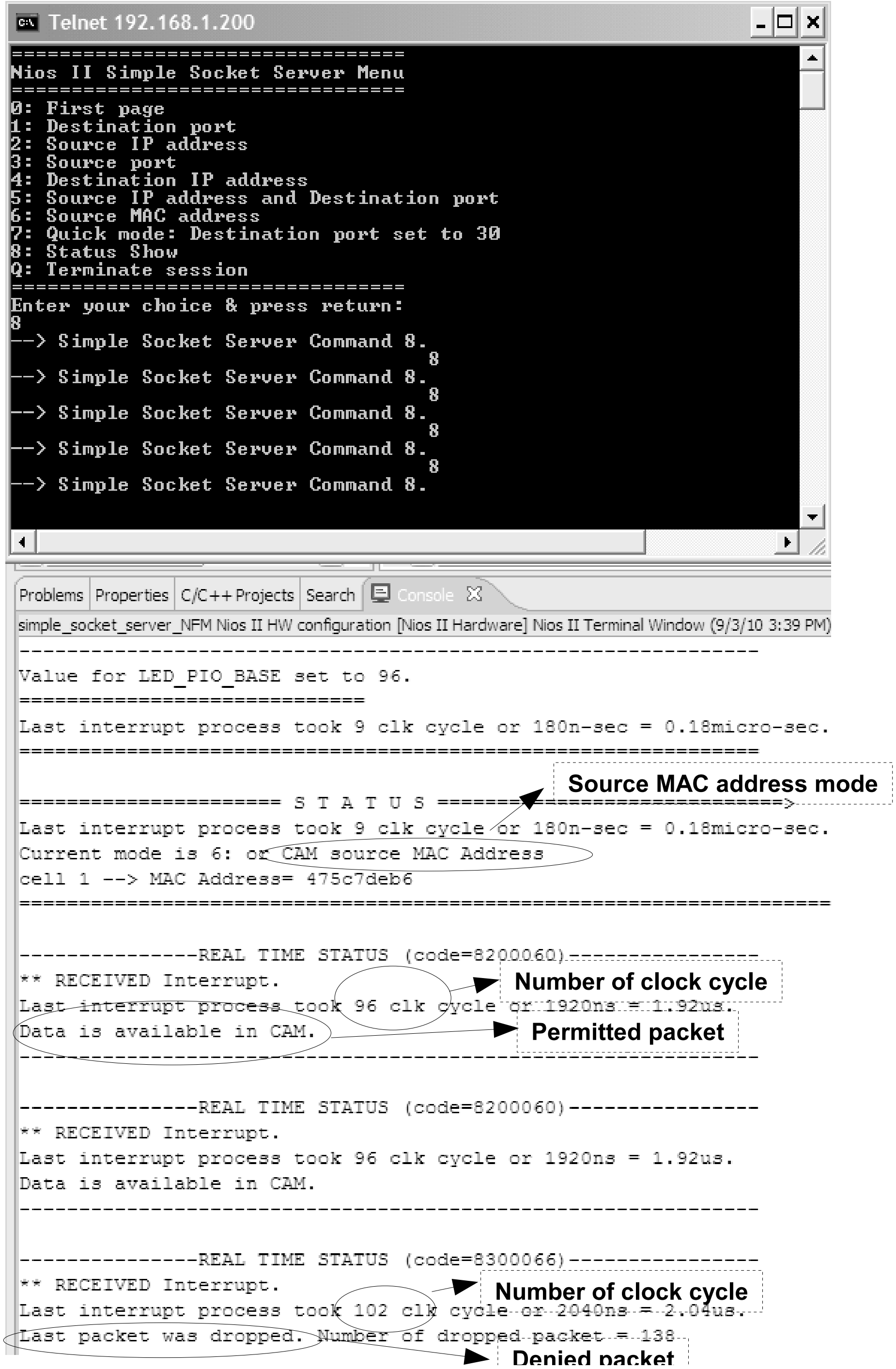


```
================================
Nios II Simple Socket Server Menu
================================
0: First page
1: Destination port
2: Source IP address
3: Source port
4: Destination IP address
5: Source IP address and Destination port
6: Source MAC address
7: Quick mode: Destination port set to 30
8: Status Show
Q: Terminate session
================================
Enter your choice & press return:
8
--> Simple Socket Server Command 8.
```

simple_socket_server_NFM Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (9/3/10 3:39 PM)

```
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
==============================================================

====================== S T A T U S ============================>
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
Current mode is 6: or CAM source MAC Address
cell 1 --> MAC Address= 475c7deb6
====================================================================

---------------REAL TIME STATUS (code=824007a)----------------
** RECEIVED Interrupt.
Last interrupt process took 122 clk cycle or 2440ns = 2.44us.
Last packet was TCP packet.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=824007a)----------------
** RECEIVED Interrupt.
Last interrupt process took 122 clk cycle or 2440ns = 2.44us.
Last packet was TCP packet.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=8340080)----------------
** RECEIVED Interrupt.
Last interrupt process took 128 clk cycle or 2560ns = 2.56us.
Last packet was TCP packet.
Last packet was dropped. Number of dropped packet = 135
```

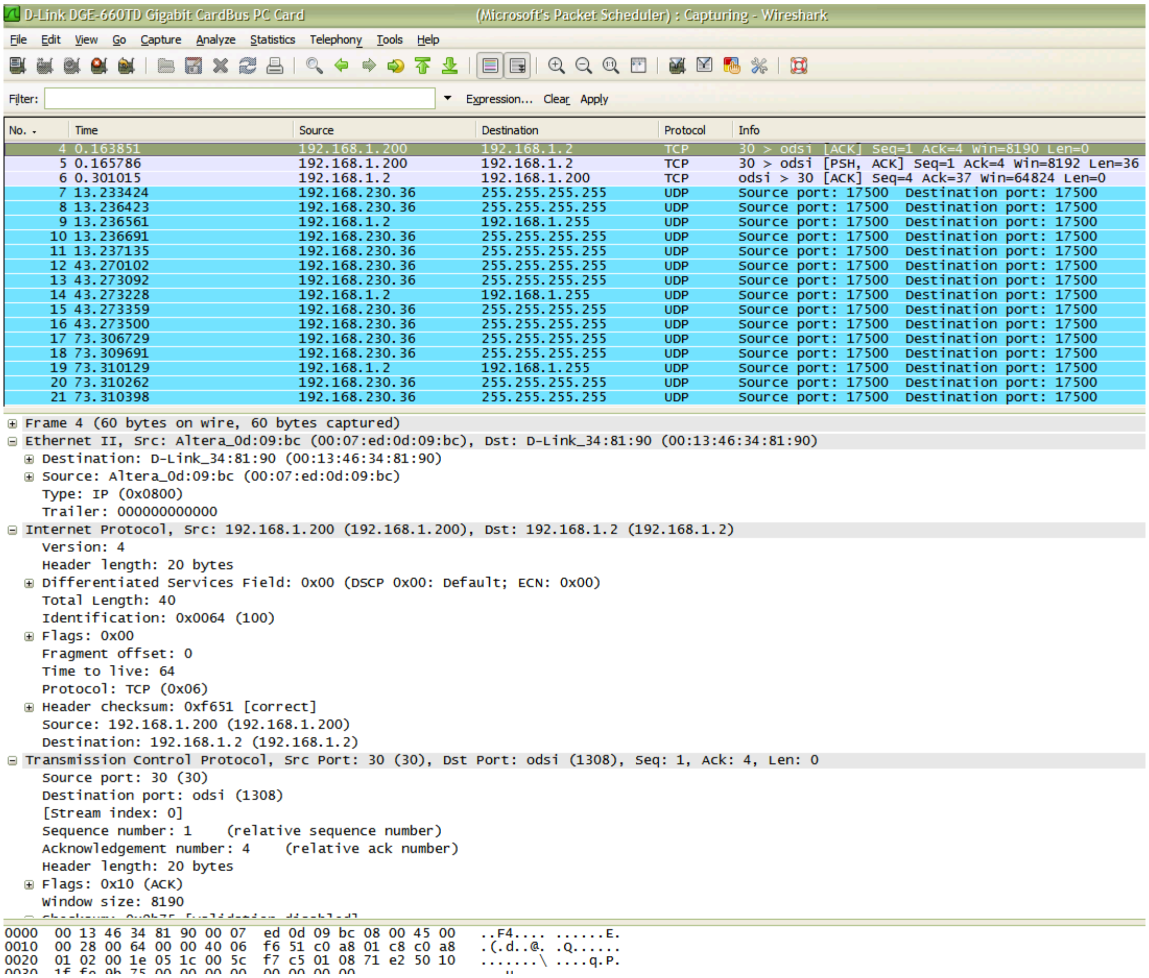
Figure 4.11: Firewall in source MAC address mode with monitoring



```
================================
Nios II Simple Socket Server Menu
================================
0: First page
1: Destination port
2: Source IP address
3: Source port
4: Destination IP address
5: Source IP address and Destination port
6: Source MAC address
7: Quick mode: Destination port set to 30
8: Status Show
Q: Terminate session
================================
Enter your choice & press return:
8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
                                   8
--> Simple Socket Server Command 8.
```

```
--------------------------------------------------------------
Value for LED_PIO_BASE set to 96.
=============================
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
==============================================================

====================== S T A T U S ==============================>
Last interrupt process took 9 clk cycle or 180n-sec = 0.18micro-sec.
Current mode is 6: or CAM source MAC Address
cell 1 --> MAC Address= 475c7deb6
====================================================================

---------------REAL TIME STATUS (code=8200060)----------------
** RECEIVED Interrupt.
Last interrupt process took 96 clk cycle or 1920ns = 1.92us.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=8200060)----------------
** RECEIVED Interrupt.
Last interrupt process took 96 clk cycle or 1920ns = 1.92us.
Data is available in CAM.
--------------------------------------------------------------

---------------REAL TIME STATUS (code=8300066)----------------
** RECEIVED Interrupt.
Last interrupt process took 102 clk cycle or 2040ns = 2.04us.
Last packet was dropped. Number of dropped packet = 138
```

Figure 4.12: Firewall in source MAC address mode without monitoring



D-Link DGE-660TD Gigabit CardBus PC Card (Microsoft's Packet Scheduler) : Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No. -  Time          Source            Destination       Protocol  Info
     4 0.163851      192.168.1.200     192.168.1.2       TCP       30 > odsi [ACK] Seq=1 Ack=4 Win=8190 Len=0
     5 0.165786      192.168.1.200     192.168.1.2       TCP       30 > odsi [PSH, ACK] Seq=1 Ack=4 Win=8192 Len=36
     6 0.301015      192.168.1.2       192.168.1.200     TCP       odsi > 30 [ACK] Seq=4 Ack=37 Win=64824 Len=0
     7 13.233424     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
     8 13.236423     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
     9 13.236561     192.168.1.2       192.168.1.255     UDP       Source port: 17500  Destination port: 17500
    10 13.236691     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
    11 13.237135     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
    12 43.270102     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
    13 43.273092     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
    14 43.273228     192.168.1.2       192.168.1.255     UDP       Source port: 17500  Destination port: 17500
    15 43.273359     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
    16 43.273500     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
    17 73.306729     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
    18 73.309691     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
    19 73.310129     192.168.1.2       192.168.1.255     UDP       Source port: 17500  Destination port: 17500
    20 73.310262     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500
    21 73.310398     192.168.230.36    255.255.255.255   UDP       Source port: 17500  Destination port: 17500

⊞ Frame 4 (60 bytes on wire, 60 bytes captured)
⊟ Ethernet II, Src: Altera_0d:09:bc (00:07:ed:0d:09:bc), Dst: D-Link_34:81:90 (00:13:46:34:81:90)
  ⊞ Destination: D-Link_34:81:90 (00:13:46:34:81:90)
  ⊞ Source: Altera_0d:09:bc (00:07:ed:0d:09:bc)
    Type: IP (0x0800)
    Trailer: 000000000000
⊟ Internet Protocol, Src: 192.168.1.200 (192.168.1.200), Dst: 192.168.1.2 (192.168.1.2)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 40
    Identification: 0x0064 (100)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (0x06)
  ⊞ Header checksum: 0xf651 [correct]
    Source: 192.168.1.200 (192.168.1.200)
    Destination: 192.168.1.2 (192.168.1.2)
⊟ Transmission Control Protocol, Src Port: 30 (30), Dst Port: odsi (1308), Seq: 1, Ack: 4, Len: 0
    Source port: 30 (30)
    Destination port: odsi (1308)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    Acknowledgement number: 4    (relative ack number)
    Header length: 20 bytes
  ⊞ Flags: 0x10 (ACK)
    Window size: 8190
  ⊟ Checksum: 0x9b75 [validation disabled]

0000  00 13 46 34 81 90 00 07  ed 0d 09 bc 08 00 45 00   ..F4.... ......E.
0010  00 28 00 64 00 00 40 06  f6 51 c0 a8 01 c8 c0 a8   .(.d..@. .Q......
0020  01 02 00 1e 05 1c 00 5c  f7 c5 01 08 71 e2 50 10   .......\ ....q.P.
0030  1f fe 9b 75 00 00 00 00  00 00 00 00
```

Figure 4.13: Wireshark screenshot

Table 4.3: Number of clock cycles for valid packet processing

Firewall mode	With monitoring	W/O monitoring
Destination port	118 (2360ns)	92 (1840ns)
Source port	118 (2360ns)	92 (1840ns)
Source IP address	122 (2440ns)	96 (1920ns)
Destination IP address	122 (2440ns)	96 (1920ns)
Source IP and Destination port	153 (3060ns) to 169 (3380ns)	127 (2540ns) to 143 (2860ns)
MAC address	122 (2440ns)	96 (1920ns)

Table 4.4: Number of clock cycles for invalid packet dropping

Firewall mode	**With monitoring**	**W/O monitoring**
Destination port	124 (2480ns)	98 (1960ns)
Source port	124 (2480ns)	98 (1960ns)
Source IP address	236 (4720ns)	210 (4200ns)
Destination IP address	236 (4720ns)	210 (4200ns)
Source IP and Destination port	236 (4720ns)	210 (4200ns)
MAC address	128 (2360ns)	102 (2040ns)

It can be shown that each Ethernet frame, by LAN91C111 or any other 100 Mbps Ethernet module, takes about 123 micro seconds to be processed. Each Ethernet module is 1,538 Bytes (without considering VLAN tagged frame) or 12,304 bits (1,538 * 8 = 12,304). Now if 12,304 bits is divided by 100 Mbps, 123.04 micro second is calculated which is the processing time per packet by the Ethernet module. Hence 1.84 to 4.2 micro seconds (when the monitoring module is disabled) is a fraction of Ethernet module processing time; thus only about 2% to 4% is added to the packet processing time by the network firewall.

The time of 1.84 to 4.2 micro seconds per packet is a very fast packet processing time. This time indicates that the *embedded network firewall* can process from 2.9 up to 6.6 Giga bits of Ethernet data per second. It means that by adding this firewall on a network gateway, no delay can be experienced in the data transfer. The *embedded network firewall* speed can be improved if :

- It is used with a faster hardware clock (currently the circuit board operates at 50MHz).
- It is combined with a faster Ethernet module, e.g. 1000Mbps or Giga bit Ethernet module.
- The waiting cycles are eliminated by having ARDY pin in the LAN91C111 connected to the FPGA for synchronous data transfer.

Table 4.5 is the Quartus compilation report for FPGA resources consumed by the

Table 4.5: Quartus compilation report

Family / Device	Stratix II / EP2S60F672C5
Combinational ALUTs	4,595 / 48,352 (10%)
Dedicated logic registers	3,169 / 48,352 (7%)
Logic utilization	11%
Total block memory bits	164,096 / 2,544,192 (6%)

embedded network firewall design. The report shows that only 11% of logic blocks and only 6% of memory blocks are used on FPGA for the whole design in this particular device.

4.4 Comparisons

In the results section, the processing time for each mode of the firewall was presented. In this section the comparison between the previous hardware firewalls in [8] and [9] and Linux software firewall (*iptables*) with the *embedded network firewall* is shown. Table 4.6 shows the features in these three hardware firewalls. As it was discussed, the *embedded network firewall* benefits from several modes and features that are very useful in a production network environment. This firewall processes packet in the transport layer (application source and destination port number in the TCP/UDP packet), Internet layer (source and destination IP address) and also processes frame in the network access layer (source MAC address). Real-time configuration and monitoring features make this firewall very practical for network administrators in a live network to change the firewall configuration and monitor the system.

Table 4.7 shows the number of clock cycles and processing time for these firewalls. Some data and modes are not available in [8] and [9] and therefore the comparison might not be completed. As it was explained in research objective, the firewall in [8] is a hardware firewall on FPGA and capable of only source IP address filtering, and the hardware firewall in [9] functions as source IP address filtering with CAM

Table 4.6: Firewall features

Features	**ENF**	**Firewall in [8]**	**Firewall in [9]**
Host usage	✓	✓	✓
Network usage	✓	-	-
Source IP address	✓	✓	✓
Destination IP address	✓	-	-
IP network mask	✓	-	-
Destination port number	✓	-	-
Source port number	✓	-	-
Source MAC address	✓	-	-
Source IP and Destination port	✓	-	-
Real-time configuration	✓	-	-
Real-time status report	✓	-	-
Number of policy (CAM size)	16	32	32
Dynamic CAM expiration	-	-	✓
FPGA device	Stratix II	APEX	Stratix
Board speed	50 MHz	33.33 MHz	33.33 MHz
FPGA logic elements	4,595	N/A	3,389
FPGA memory bits	160 KB	N/A	138 KB

expiration feature and both of them improve the speed of packet processing compare to software firewall. The *embedded network firewall* is substantially faster (in IP packet processing) than the firewall in [8] but slower than [9] in packet dropping since the *embedded network firewall* uses network mask along with IP address; however, the network mask expands the IP address range and makes the firewall very practical. If the administrator wants to accept incoming IP packets to the inside network from all of the IP addresses in a specific IP range, then one rule (and therefore only one CAM cell and one netmask RAM cell) is enough to make it possible but the same rule exhausts 254 CAM cells in the other two firewalls. The *embedded network firewall* processes the packet/frame in a maximum of 102 clock cycles (2,040 ns) in the other modes which is faster than the other two firewalls.

Table 4.8 compares the *embedded network firewall* with the software firewall on Nios microprocessor in [8] and *iptables* (Linux firewall). It is obvious that the *embedded network firewall* performs much faster than the software firewall in [8]. The

Table 4.7: Number of clock cycles to perform various tasks in the firewall

Process type	**The ENF**	**Firewall in [8]**	**Firewall in [9]**
Source IP address (pass)	96 (1920ns)	284 (8520ns)	N/A
Source IP address (drop)	210 (4200ns)	N/A	N/A
Packet processing (Min)	62 (1240ns)	284 (8520ns)	N/A
Packet processing (Max)	236 (4720ns)	N/A	95 (2850ns)

embedded network firewall also works faster than the *iptables* in TCP/UDP packet and MAC frame processing, although the *iptables* is a stateful firewall. It is worth to mention that when the packet payload or the number of policy rules increases, the *iptables* processing time also increases but the embedded network firewall is not sensitive to the packet size or the number of policies (except IP address rules) because of, the use of the CAM modules in the design.

Table 4.8: Processing time

Process type	**The ENF**	**Software firewall in [8]**	**iptables [34]**
IP packet (pass)	1,920 ns	280,770 ns	3,180 ns
IP packet (drop)	4,200 ns	287,940 ns	3,560 ns
TCP / UDP (pass)	1,840 / 1,840 ns	N/A	8,590 / 8,970 ns
TCP / UDP (drop)	1,960 / 1,960 ns	N/A	8,970 / 9,350 ns
MAC (pass)	1,920 ns	N/A	9,380 ns
MAC (drop)	2,040 ns	N/A	9,760 ns

Chapter 5

Summary, Conclusions and Future works

In this last chapter, the summary of the entire chapters, conclusion of the work and interesting further improvement around this topic is presented.

5.1 Summary

In the first chapter, the motivation and objective behind this research was explained. The cyber world plays a significant role in today's human life. Although this has made the life easier for accessing information and using a variety of on-line services, it has brought many related issues as well. One very important concern is how to protect the information and network resources against various threats such as network attack. A powerful defence shield is to have a firewall in an autonomous network to enforce the security policy on incoming and outgoing data traffic. A firewall is a hardware or software network device that is placed between the internal network and external network (mostly Internet) to investigate each data packet based on the header information in the packet to permit or deny the information passing through. This type of firewall or packet filter can be configured to be triggered on various criteria based on packet header such as TCP destination port. In this research, a hardware firewall was designed to filter data packets based on any of the following options: source and destination port number in the transport layer, source and destination IP address in the network layer, combination of source IP address and destination port number and MAC address in the data link layer in the TCP/IP network.

In the second chapter, necessary background information regarding this research was presented. It was explained how a firewall could be helpful, if properly configured, against some type of network security threats. The best protection for a network is possible by securing the main gateway to the outside world instead of each individual host in a network environment. The chapter also showed how the TCP/IP networking model (based on OSI model) and each layer operates. Then the Altera FPGA, Nios II microprocessor, MicroC/OS-II real-time operating system and NicheStack TCP/IP stack were presented as a solution for implementing embedded system and system on chip design.

The third chapter disclosed the design procedure and talked about the hardware and software co-design in the network firewall which was implemented on an Altera FPGA development board. The LAN91C111 was the Ethernet hardware module to receive and transmit Ethernet packets in the development board. The main hardware block of the design was the NFM module, written in Verilog HDL, which extracts the requested data field from the header of the packet. It then searches the field in an associated CAM for a match entry and if the data is not available, the packet is dropped otherwise forwarded. Five CAM modules were designed as look-up tables to increase the speed of data search operation. Two RAM modules (netmask RAM) were used for storing the network mask data of the IP address entry in the design. The Nios II standard edition CPU was used for initialization and configuration of the hardware blocks and also for running software modules in the design. The arbiter module was designed to allocate shared resource (the Ethernet module) between the Nios II CPU and the NFM module. The display module was used for monitoring the status of the packet processing for debugging and troubleshooting of the project.

The software modules in the project were utilized for firewall initialization and real-time configuration aspects of the *embedded network firewall*. MicroC/OS-II real-time operating system, NicheStack TCP/IP stack and LAN91C111 device driver were used to build the platform foundation for software implementation of the project. The LAN91C111 device driver was modified in order to be able to work with the firewall reception and transmission processes. The initialization software module was written for firewall primary setup and the Telnet server was developed to remotely access the *embedded network firewall* in order to be able to change the firewall mode, configuration or for monitoring purposes. Chapter four of this thesis explained the testing methodology and provided the results of the *embedded network firewall* operation in a live network.

5.2 Conclusions

The objective of this thesis is accomplished and the *embedded network firewall* successfully operated in six different modes to permit only authorized network communication. The test results showed the packet processing time by the *embedded network firewall* was only a small fraction, 2% to 4% of a 100Mbps Ethernt module processing capabilities. It was shown that it took 62 clock cycles (1.24 micro second) to 236 clock cycles (4.72 micro second) to process a packet, or in the same fashion 2.9 to 6.6 Giga bits per second of packet filtering ability. The comparison between the *embedded network firewall* and previous hardware firewalls on FPGA ([8] and [9]) shows that the *embedded network firewall* outperforms the other firewalls in terms of many practical features such as real-time configuration, real-time status report, transport and physical layer packet/frame processing and IP address range capability. Also comparing the *embedded network firewall* with the *iptables* (the Linux software firewall) revealed that *embedded network firewall* performs faster, and it is not sensitive to the size of packet or the number of rules in the firewall configuration.

The processing speed could be improved if the design is combined with a faster system clock, faster Ethernet module or eliminating waiting cycles (due to Altera board design and connection between the FPGA and Ethernet module).

The whole design consumed a small portion of the Altera StratixII-2S60 FPGA, 11% of the logic blocks and 6% of the memory blocks. This makes an ideal size for the *embedded network firewall* system on a network gateway (broadband modem or router); however the network firewall could be embedded on an ASIC for even smaller size and higher speed.

5.3 Future works

Although the goal of the research in this thesis was achieved, the following suggestion would be interesting to add to the *embedded network firewall*:

- **Hardware improvement**
    - IP version 6 (IPv6) implementation: IPv6 is the next generation of IPv4 with new features and improvements but unfortunately it is very incompatible with IPv4. The transition from IPv4 to IPv6 can be very challenging. To make this firewall to be able to process IPv6 packets, most of the hardware and software modules have to be changed. First of all, MAC device driver and TCP/IP stack have to be capable of IPv6 packet processing and then the rest of the customized hardware modules have to be updated for IPv6 compatibility. The implementation of IPv6 in this firewall is a great project for future work.
    - Stateful TCP implementation: This feature can be useful for keeping track of TCP connection since the TCP is a connection oriented protocol and uses the sequence numbers for data transfer.
    - Stream Control Transmission Protocol (SCTP) compatibility: SCTP is a transport layer protocol which includes features of both TCP and UDP.

SCTP is a reliable data delivery protocol with lower latency than TCP. As a future improvement to the firewall, modifying the NFM module, some software modules, MAC device driver and TCP/IP stack, the firewall can handle SCTP protocol.

  - Network mask RAM optimization: By increasing the number of the IP addresses and network mask entries, the size of the RAM has to be increased, which also increases the time of RAM searching and consequently increases the number of the system clock cycles. This can be an interesting feature to be optimized in the design.

- **Software improvement**

  - Logging : This feature can be very useful for network traffic analysis by logging all sort of the firewall activity to a remote log server. Logging dropped packets can provide adequate information about attempted network attacks.

  - Email notification: This feature can notify the network administrator in case of serious attack or any other emergency case for further action to be taken.

References

[1] Computer Economics. http://www.computereconomics.com/page.cfm?name=MalwareReport, 2007. accessed Nov 2010.

[2] B. Schuh, "IP Cores, FPGAs to Launch 10-Mbps Flight Control Systems," *GE Fanuc Embedded*, vol. 1, no. 1, 2006.

[3] *Nios Development Board Stratix II Edition, Reference Manual.* Altera corporation.

[4] *Nios II Processor Reference Handbook.* Altera corporation.

[5] *Using the NicheStack TCP/IP Stack, Nios II Edition Tutorial.* Altera corporation, 2007.

[6] S. Harris, A. Harper, C. Eagle, and J. Ness, *GRAY HAT HACKING, The Ethical Hacker's Handbook.* McGraw-Hill, second ed., 2008.

[7] E. D. Zwicky, S. Cooper, and D. B. Chapman, *Building Internet Firewalls.* O'Reilly Media, second ed., 2000.

[8] J. Cheng, "Silicon firewall prototype," Master's thesis, 'University of Saskatchewan, 2004.

[9] D. Laturnas, "Dynamic silicon firewall," Master's thesis, University of Saskatchewan, 2006.

[10] W. Noonan and I. Dubrawsky, *Firewall Fundamentals.* Cisco Press, first ed., 2006.

[11] S. McClure, J. Scambray, and G. Kurtz, *Hacking Exposed: Network Security Secrets & Solutions.* McGraw-Hill Osborne Media, fourth ed., 2003.

[12] "Top 100 Network Security Tools." http://netsecurity.about.com/od/hackertools/a/top1002006.htm. accessed Nov 2010.

[13] "Top 15 Security/Hacking Tools Utilities." http://www.darknet.org.uk/2006/04/top-15-securityhacking-tools-utilities/. accessed Nov 2010.

[14] A. S. Tanenbaum, *Computer Networks.* Prentice Hall, fourth ed., 2002.

[15] J. Doherty, N. Anderson, and P. D. Maggiora, *Cisco Networking Simplified.* Cisco Press, second ed., 2008.

[16] “ISO/IEC 7498-1: Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model.” http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip, 1994.

[17] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley Professional, first ed., 1994.

[18] B. Carpenter, “RFC:1958, Architectural Principles of the Internet.”

[19] P. Marwedel, *Embedded System Design*. Springer, first ed., 2005.

[20] J. Andrews, *Co-verification of Hardware and Software for ARM SoC Design*. Newnes, first ed., 2004.

[21] “Altera and Xilinx Report: The Battle Continues.” http://seekingalpha.com/article/85478-altera-and-xilinx-report-the-battle-continues, 2008. accessed Nov 2010.

[22] “Stratix Series High-End FPGAs.” http://www.altera.com/products/devices/stratix-fpgas/about/stx-about.html. accessed Nov 2010.

[23] “Nios II Development Kit, Stratix II Edition.” http://www.altera.com/products/devkits/altera/kit-niosii-2S60.html. accessed Nov 2010.

[24] *Nios II software Developer’s handbook, Reference Manual*. Altera corporation.

[25] “What is GNU?.” http://www.gnu.org.

[26] “Micrium Expands RTOS Family with MicroC/OS-II.” http://embedded-computing.com/micrium-expands-rtos-family-%C2%B5cos-iii, 2009. accessed Nov 2010.

[27] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, third ed., 2002.

[28] *SMSC LAN91C111 10/100 Non-PCI Ethernet Single Chip MAC+PHY, Data Sheet*. Standard Microsystems Corporation.

[29] J.-L. Brelet, “Using Block RAM for High Performance Read/Write CAMs.” http://www.xilinx.com/support/documentation/application_notes/xapp204.pdf, 2000. accessed Nov 2010.

[30] J. J. Labrosse, *MicroC OS II: The Real Time Kernel*. Newnes, second ed., 2002.

[31] J. Postel and J. Reynolds, “RFC:854, Telnet Protocol Specification,” 1983.

[32] E. Nemeth, G. Snyder, and T. R. Hein, *Linux Administration Handbook*. Prentice Hall, second ed., 2006.

[33] J. Postel, “RFC:792, Internet Control Message Protocol,” 1981.

[34] A. J. Melara, “Performance analysis of the linux firewall in a host,” Master’s thesis, California Polytechnic State University, 2002.