UNIVERSITY OF TEXAS AT SAN ANTONIO
San Antonio, Texas

THE DESIGN AND IMPLEMENTATION OF AN EMBEDDED, HOST-BASED INTRUSION DETECTION SYSTEM FOR SYN FLOODING USING FPGA TECHNOLOGY

A Thesis Submitted in Partial Fulfillment of the Requirements for the Bachelor of Science in Electrical Engineering with Tier 2 Honors in the Honors College

Viresh Thusu

Department of Electrical and Computer Engineering
December 2006

PREPARED BY:



Viresh Thusu

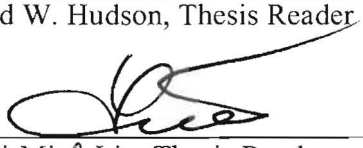
APPROVED BY:



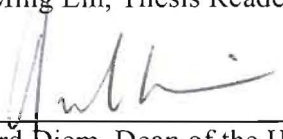
Dr. Parimal Patel, Thesis Advisor



Dr. Fred W. Hudson, Thesis Reader



Dr. Wei-Ming Lin, Thesis Reader



Dr. Richard Diem, Dean of the Honors College

RECEIVED BY THE HONORS COLLEGE
December 2006

ABSTRACT

Technological advancement has had an optimistic influence in the field of communication. Internet is the poster child for success in the field of global communication and has shrunk the world. Everything from e-commerce to education is now-a-days done online. The number of internet users is increasing rapidly. To organize and facilitate efficient data transaction, the internet uses various protocols: set of rules that define an exact format for communication between systems. TCP/IP is one such widely used protocol. Within TCP/IP, communication is established via a Three Way Handshake which is characterized by the transmission of a SYN packet by the initiator. Sometimes when an intruder initiates contact, it floods the server memory buffer with SYN packets; thus, denying the server in rendering service to other network components in a denial of service attack known as SYN Flooding.

The thesis deals with the design and implementation of a host-based embedded Intrusion Detection System for SYN Flooding using FPGA technology. Amirix AP130 development board with a Xilinx Virtex-II Pro (XC2VP30 FF896 -6) chip consisting of an on board IBM Power PC processor, was used. An embedded system, using the Xilinx Platform Studio software tool, with EMAC (@ 100 Mbps), 8 KB OPB BRAM, 64 KB PLB BRAM, OPB timer, and RS -232 cores was created. The software analyzed TCP/IP packets and checked for SYN Flooding attacks. A database was maintained for keeping records of the incoming packets. This database was created using a Queue data structure. The system was successfully tested in a controlled environment.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS 4

Chapter 1: INTRODUCTION 6

1.1 Intrusion Detection System 6

1.2 The TCP/IP Protocol 7

1.3 Establishment of Communication 8

1.4 Thesis Statement 10

Chapter 2: SYSTEM HARDWARE 12

2.1 FPGA 12

2.2 AP130 Board 12

2.3 Virtex-II Pro FPGA 14

2.4 PowerPC405 Processor 14

2.5 Embedded System 16

Chapter 3: DESIGN METHODOLOGY 18

3.1 Hardware 18

3.2 Software 20

3.2.1 Extraction of Fileds 22

3.2.2 Database Management 24

Chapter 4: TESTING AND RESULTS 29

4.1 Testing Methodology 29

4.2 Time Expiration Scenario 30

4.3 Multiple SYN Packet Reception 31

4.4 Independent Acknowledge Reception 33

4.5 Good Connection 34

Chapter 5: CONCLUSION 36

5.1 Conclusion 36

REFERENCES 37

APPENDIX A: C Code 38

APPENDIX B: MHS File 48

APPENDIX C: MSS File 53

APPENDIX D: UCF 55

ACKNOWLEDGEMENTS

My thesis has been a pleasurable educational experience; however, several network components played pivotal roles. Protocol suggests to SYN acknowledge them for believing in my potential.

I would like to thank Dr. Patel, my mentor and thesis advisor, for his continued support and guidance right from the time I joined the Electrical Engineering program at UTSA. Dr. Patel, thank you for consistently prodding me towards success even if it meant spending countless nights at the laboratory eating microwave noodles. The fact that I took my first and last computer engineering course with you completes the circular linked-list of knowledge you have bestowed upon me.

Thank you Dr. Hudson for periodically checking up on me and educating me through the variant courses I have had with you. Our long conversations in the ECE Department corridors have been refreshing and stimulating.

Dr. Lin, I am thankful for your role in strengthening my foundation in the field of digital design without which this thesis would not have been possible.

Dr. Eisenberg, I appreciate your guidance and encouragement throughout my degree. Your to-the-point advice helped to a great extent.

Further, I would like to recognize the contribution made by my seniors in the EE Instrumentation Laboratory: Ashok Tummala, Chirag Parikh, Devang Pandya, and Sunil Kashyap. Thank you for familiarizing me with the Xilinx environment and explaining to me nuances of the various component functionalities.

Last but not the least, I express my deepest gratitude to my family who have always supported and encouraged me in my endeavors. Thank you for being understanding. This thesis could have absolutely not been possible without you.

Chapter 1

INTRODUCTION

1.1 Intrusion Detection System

We live in a technology-savvy society. Even before one completely adapts to a particular technology, there are new discoveries and/or updates that make us incline toward them. Long gone are the days of physical data handling and storage. The computer, a synonym to innovation, processes, stores, and transfers vast amounts of valuable information, but for these purposes it has to communicate with other systems, hence forming a network. Examples of a computer network are internet, LAN (Local Area Network), and WAN (Wide Area Network). Such networking facilitates a smooth but vulnerable electronic environment.

Good and evil co-exist like the two faces of a coin. Where there are intelligent operational methods their nemeses soon follow: intruders. An intruder can be any entity (person or group of people) that is not a network component of the system being intruded. Necessity is the mother of invention; thus, the need for another system that can detect and prevent (if possible) such intrusion. An IDS (Intrusion Detection System) is one such system that acts as a guard dog and exposes inappropriate activity.

There are two kinds of IDS's: host-based and network-based. Host-based detection involves installing the IDS on the target system and tackling attacks from various attackers. On the other hand, a network-based detection system monitors the data transfer on the network and accordingly realizes its functionality. Both software and hardware can be utilized for the creation of either system. The software approach leads to

the occupancy of main memory and reduces the speed of other ongoing processes. However, separate hardware can be employed to do the needful while not affecting the speed and operational capabilities of other applications, of course at an additional cost.

Like any other system, IDS is subjected to the set of rules or protocol it observes. This thesis deals with the study of such systems. Before we look at the design, it is helpful to understand the specific protocol and type of host-based intrusion that was under consideration.

1.2 The TCP/IP Protocol

TCP/IP (Transmission Control Protocol/Internet Protocol) is one of the most widely used protocols whose applications range from e-mail to web browsing. Similar to other protocols, TCP/IP model is divided into four layers. The following figure shows the four layers of this protocol.

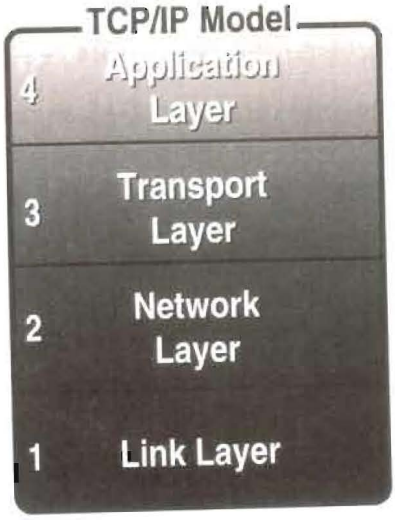


Figure 1.2.1: Layers of TCP/IP model
Source: www.emu.edu.tr

- **Application Layer:** This layer interacts with the operating system and other software applications. Once the interaction is perceived in a meaningful way with regards to whether data needs to be read or written, communication with other entities is established.
- **Transport Layer:** As the name suggests, this layer is involved in maintaining control over the flow and ensures reliable 'transport' of data. Further, it also sends and receives requests for data transfer.
- **Network Layer:** The critical decision of what protocol is to be used is determined at this layer. Therefore, it regulates the way in which data is passed. It deals with the size of the data packet.
- **Link Layer:** The actual data transfer takes place in this layer. This data has various headers attached to it providing information such as the origin of data (IP address), the type of protocol used, and the type of operation to be performed: read or write.

1.3 Establishment of Communication

Due to its popularity, attackers may try to exploit TCP/IP functionalities in various ways. Let us first consider Figure 1.3.1 to gain a better understanding of how two systems interact using TCP/IP protocol.

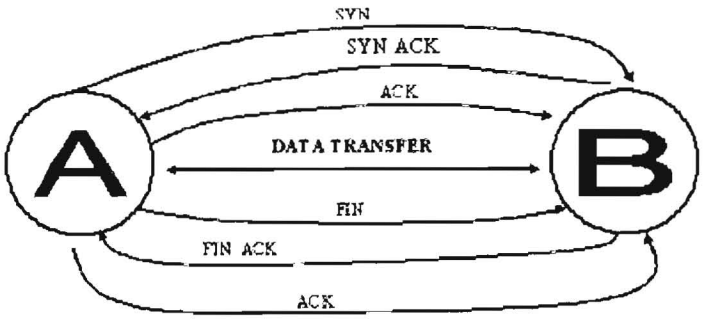


Figure 1.3.1: Communication between two systems using TCP/IP protocol

The above diagram depicts a typical interaction (connection) taking place between system A and system B using TCP/IP. First A sends a request for data transfer by generating a SYN (Synchronize) packet. B then replies by sending a SYN ACK (Synchronize and Acknowledge) packet that tells A that B has "acknowledged" its request and also wants to "synchronize" a connection. To this A generates an ACK (Acknowledge) packet affirming its reception of B's SYN. Such a commencement of communication takes place every time two entities need to exchange information using the TCP/IP protocol. This is known as a Three Way Handshake. Only after the occurrence of a Three Way Handshake does the actual data exchange take place. Since A was responsible for initiating contact, in a normal scenario, it has to account for the termination too, and therefore generates a FIN (Finish) packet at the end of the communication. B confirms it by sending FIN ACK (Finish and Acknowledge) packet to A to which A sends an ACK. ACK is required to be sent by A according to the protocol; however, it is not required to be received by B for termination of communication.

1.4 Thesis Statement

Although the above communication seems to be considerably stable and secure, attackers have found ways to circumvent it. One of the methods they succeed in breaking the communication channel is SYN Flooding. To better understand this particular method of attack please refer to the following figure.

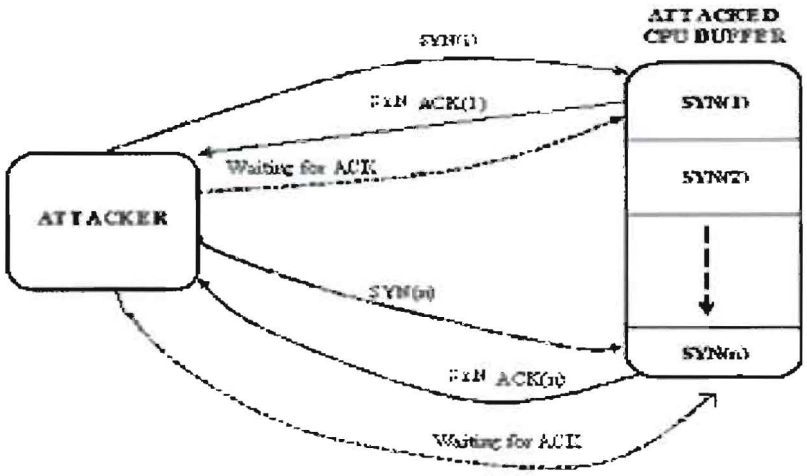


Figure 1.4.1: SYN Flooding

Figure 1.4.1 gives a pictorial description of SYN Flooding. The attacker initiates a Three Way Handshake. The system being attacked – the target – stores the SYN information at time of every initial phase of communication. The target allocates resources such as buffer memory, hardware, time, etc. when it receives SYN packets and sends a SYN ACK packet to the initiator (in this case, the attacker). The target then awaits an ACK packet from the attacker, but instead of responding in the appropriate manner, the attacker sends another SYN packet. The target interprets this as a new request and, once again, responds with an SYN ACK packet after allocating resources.

Thus, resources are being allocated without being really used. This process continues till all the resources are allocated (are fully occupied by false SYN's). The targeted system is now not able to accept any further SYN's from other non malicious sources and as a result is trapped in an infinite 'paralytic' state. Such a method of intrusion is known as SYN Flooding as the buffer is flooded with false SYN's. It is a type of denial of service attack as the target system is denied of rendering its service to other legitimate systems.

Thus, to counter this denial of service attack, a host-based, embedded IDS for SYN Flooding using FPGA technology was designed and implemented on hardware. Such a system should detect and isolate the attacker from normal users. This is possible by keeping track of when the ACK is expected and if it indeed was received or not. If the server does not receive it within the specified time then it should de-allocate the corresponding resources and alert for a possible intrusion. The following chapters shed light on the design process and implementation results.

Chapter 2

SYSTEM HARDWARE

2.1 FPGA

FPGA (Field Programmable Gate Array) is a technology containing programmable logic components and programmable interconnects. It can be programmed and re-programmed for realizing a particular logical functionality; hence it provides flexibility to the designer. The design cycle can be reduced dramatically due to use of FPGA's, when compared to using ASIC (Application Specific Integrated Circuits) design circuits or discrete logic chips. For being a generic structure, FPGA's are inexpensive, make excellent prototypes, and facilitate Embedded and SOC (System on Chip) designs. Such features have made it popular in the industry and are convincing enough to employ FPGA technology for the IDS design.

2.2 AP130 Board

After selecting the appropriate technology, the next step was to select an appropriate development board to implement the design on. Various factors were involved in determining the board. The design requirements needed the presence of an EMAC (Ethernet Memory Access Controller) for receiving data packets over a network, a RS-232 port in order to interact with a user console, a suitable FPGA chip which supported embedded design, and sufficient on-board memory for data and instruction storage. The Amirix Systems' AP130 board, consisting of a Xilinx XC2VP30 FPGA chip, was a worthy candidate as it sufficed all of the above design requirements.

The AP130 features:

- Xilinx Virtex-II Pro FPGA with embedded PowerPC processor (XC2VP30 in the FF896 package with -6 Speed Grade)
- 64 MB DDR SDRAM
- 4 MB Program Flash
- 4 MB Configuration Flash
- Xilinx SystemACE CompactFlash Interface
- Dual 10/100/1000BASE-T Ethernet Ports
- 10/100BASE-T Ethernet Port
- Dual RS-232D Serial Port

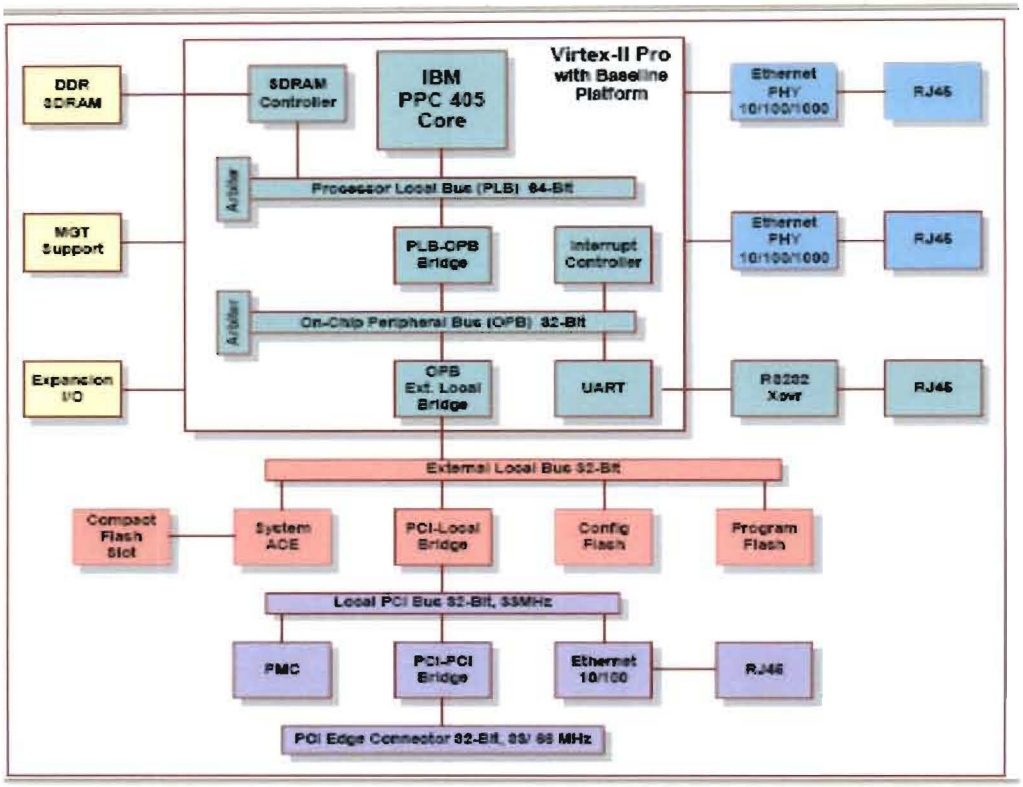


Figure 2.2.1: AP130 Platform Architecture
Source: www.amirix.com

The above figure depicts a block diagram of the AP130 architecture. The abundance of resources makes it an apt choice for various applications.

2.3 Virtex-II Pro FPGA

Xilinx's Virtex-II Pro family is very popular and successful. There are numerous variants available for a designer to work with. In general, their highlights include:

- Up to 20 RocketIO™ or RocketIO X embedded Multi-Gigabit Transceivers (MGT's)
- Up to two IBM PPC 405 (PowerPC™) RISC processor blocks
- Flexible logic resources
- SRAM-based in-system configuration
- Active Interconnect technology
- SelectRAM memory hierarchy
- Dedicated 18-bit x 18-bit multiplier blocks
- High-performance clock management circuitry

The XC2VP30 FF896 -6 FPGA device used has 30,816 Logic Cells, 136 Multiplier blocks (18 x 18 bit), 2,448 Kb of BRAM (Block SelectRAM) blocks, 8 DCM (Digital Clock Management) blocks, 556 user I/O pads, and 8 RocketIO Transceiver blocks.

.

2.4 PowerPC405 Processor

One of the more essential parts of the system is PPC 405 RISC (Reduced Instruction Set Computer) processor. It is a 32 bit processor that can execute instructions at a sustained rate of one instruction per cycle. A list of its features follows

- PowerPC RISC CPU (Central Processing Unit)
  - Implements the PowerPC User Instruction Set Architecture (UISA) and extensions for embedded applications
  - Thirty-two 32-bit general purpose registers (GPR's)
  - Static branch prediction
  - Five-stage pipeline with single-cycle execution of most instructions, including loads/stores
  - Unaligned and aligned load/store support to cache, main memory, and on-chip memory
  - Hardware multiply/divide for faster integer arithmetic (4-cycle multiply, 35-cycle divide)
  - Enhanced string and multiple-word handling

  - Big/little endian operation support
- Cache Storage
  - Separate instruction and data cache units, both two-way set-associative and non-blocking
  - Eight words (32 bytes) per cache line
  - 16 KB array Instruction Cache Unit (ICU), 16 KB array Data Cache Unit (DCU)
  - Operand forwarding during instruction cache line fill
  - Copy-back or write-through DCU strategy
  - Doubleword instruction fetch from cache improves branch latency
- OCM controllers provide dedicated interfaces between Block SelectRAM+ memory and processor block instruction and data paths for high-speed access
- PowerPC timer facilities
  - 64-bit time base
  - Programmable interval timer (PIT)
  - Fixed interval timer (FIT)
  - Watchdog timer (WDT)
- Debug Support
  - Internal debug mode
  - External debug mode
  - Debug Wait mode
  - Real Time Trace debug mode
  - Enhanced debug support with logical operators
- Two hardware interrupt levels support
- Advanced power management support

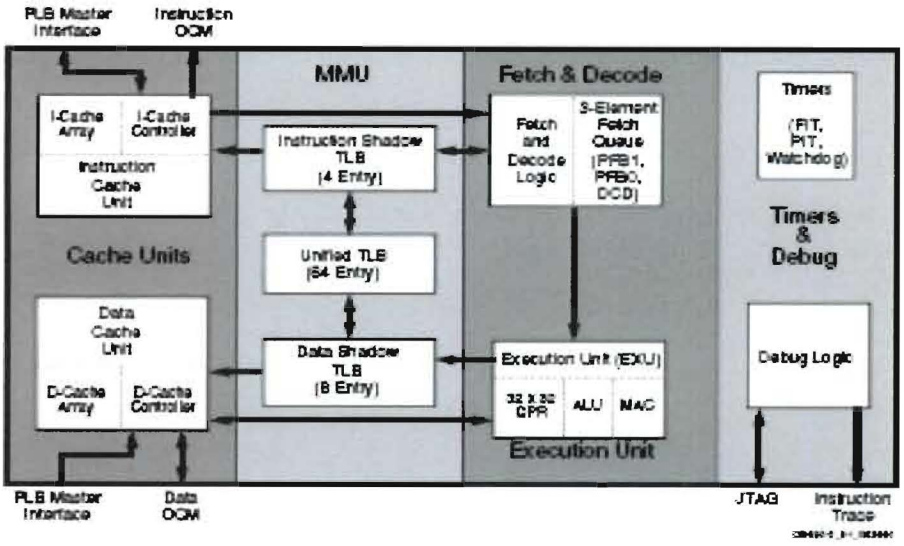


Figure 2.4.1: AP130 Platform Architecture
Source: www.xilinx.com

Figure 2.4.1 shows the internal block diagram of PPC 405. Due to its powerful features, the processor lies at the core of the embedded system implemented.

2.5 Embedded System

An embedded system is a special-purpose system in which the computer is completely dedicated to a device and/or operation. An embedded system performs one or a few pre-defined tasks, usually with very specific requirements. Since the system is devoted to specific tasks, it can be optimized with respect to the size and cost of the final product. An MP3 player is an example of such a system.

The IDS created is a PPC405 based embedded system. To illustrate and better understand what such a system looks like please refer to following figure.

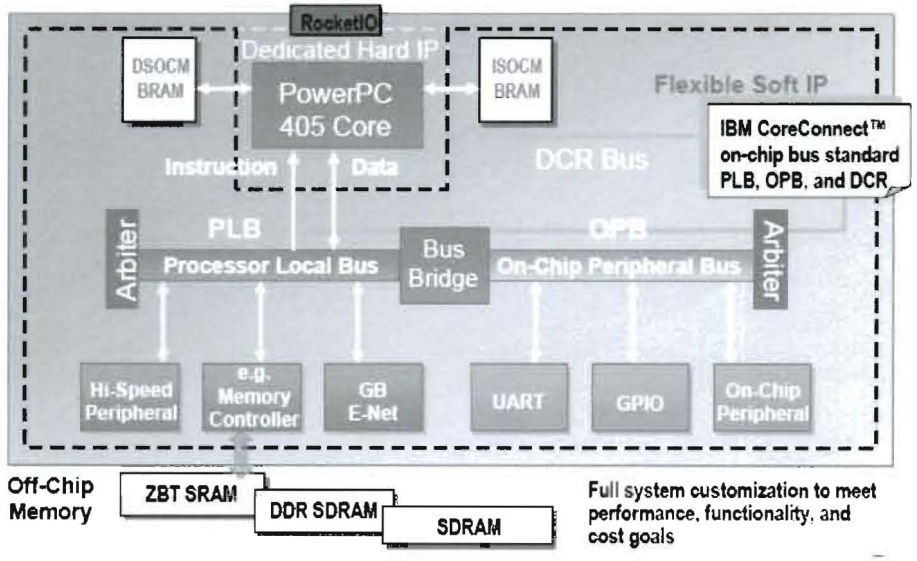


Figure 2.5.1: PPC 405 Based Embedded Design

Figure 2.5.1 gives a general idea of a PPC405 embedded system on a Virtex-II Pro chip. As mentioned earlier, due to the abundant resources and variety of resources available, the designer can create a suitable system tailored to the application needs.

Chapter 3

DESIGN METHODOLOGY

3.1 Hardware

Based on the IDS needs an embedded system, using the Xilinx Platform Studio software tool, with the following features was created:

- PPC 405 CPU
- 100 MHz Clock Frequency
- 100 MHz Bus Cycle
- 8 KB OPB (On chip Peripheral Bus) BRAM
- 64 KB PLB (Processor Local Bus) BRAM
- EMAC @ 100 Mbps (Higher Priority Interrupt)
- OPB Timer (Lower Priority Interrupt)
- RS-232 Core @ 115200 Bauds

Once the system was created, the next step was to formulate the interaction between them. The interaction process started from the moment data was received by the server and continued unto the user interface designed to inform the system administrator of the system status. Figure 3.1.1 depicts the data flow and hardware interaction.

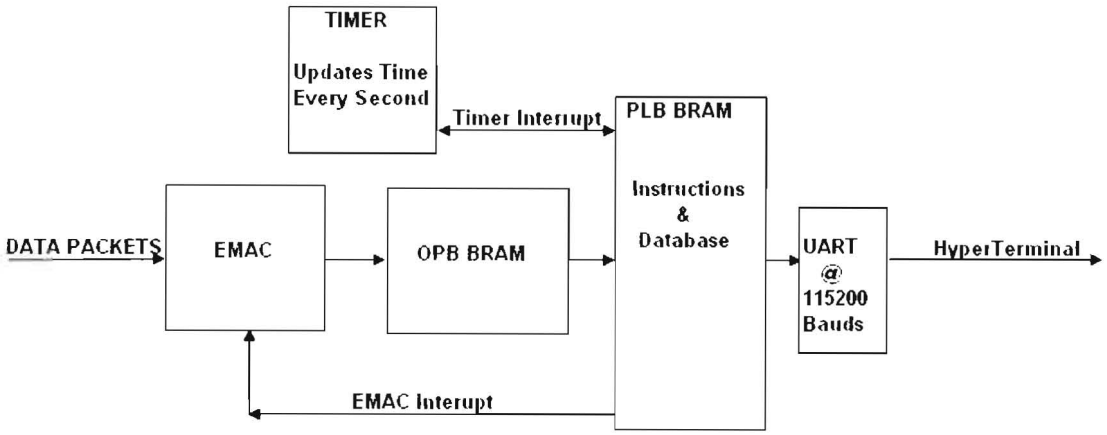


Figure 3.1.1: Block diagram of system dataflow and hardware interaction

The incoming data packets are received by the EMAC and stored into the OPB BRAM. The necessary information is then transferred into the PLB BRAM which houses a database and the actual instructions. The status of the system is displayed onto the user console using the RS-232 serial communication standard at a baud rate of 115,200 bauds. Since the application is time critical, an OPB timer was set up in order to keep track of various time related activities. Both the EMAC and timer use the interrupt mechanism.

In a time critical operation, such as the one in consideration, it is inefficient for the CPU to be needlessly diverted to check status of various system components. The interrupt mechanism helps to optimize CPU time. Its functioning is as follows. Only when a certain component requires attention from the CPU it generates an interrupt request which indicates to the CPU that the component needs to be serviced. The CPU then accepts the request and services the module. This way the CPU's time is well used and it utilizes its time and resources on a need to need basis. If multiple devices generate interrupts, preference is given to the device with the highest interrupt priority. As data reception is of utmost importance, in the IDS design the EMAC is given a higher level interrupt priority than the OPB timer.

The timer updates the time every second; however, the hardware does not have a self-concept of time. The designer needs to derive it from the OPB timer. Figure 3.1.2 shows a snippet of the code dealing with this concept.

```
// set the number of cycles the timer counts before interrupting
XTmrCtr_mSetLoadReg (0x41C00000, 0, 100000000);

void timer_int_handler(void * baseaddr_p)
{

        unsigned int csr ;

        // Read timer0 CSR to see if it raised the interrupt
        csr = XTmrCtr_mGetControlStatusReg(0x41C00000, 0);

        if (csr & XTC_CSR_INT_OCCURED_MASK)
                {
                  interrupt_cnt = interrupt_cnt + 1;
                  timer_flag=1;                    // set flag to check for expired SYN's

                  XTmrCtr_mSetControlStatusReg(0x41C00000, 0, csr);    // Clear the timer interrupt
                }
}
```

Figure 3.1.2: Code for timer

Since the processor clock frequency is 100 MHz, in one second there are 100 million clock cycles; hence to increment the time every second the timer should generate an interrupt after every 100 million clock cycles. This value is loaded into the 'XTmrCtr_mSetLoadReg' register and an interrupt is generated every second, wherein the time is incremented by one second and the interrupt flag in the 'XTmrCtr_mSetControlStatusReg' register is cleared in order to enable an interrupt after one more second. This way the design implements a clock of its own.

3.2 Software:

This section gives a better understanding of the actual code written and the logic behind the code. The database maintained in the PLB BRAM and the error checking for SYN Flooding is all done in software. Before extracting the data from the OPB BRAM, the data packet is examined for protocol. Only if the protocol is TCP/IP, denoted by a 0x06

(hex 6) in the protocol field of the data packet, the necessary information is processed, else the information is discarded and waited to be overwritten by another set of incoming data. Before explaining this process, it is necessary to understand what a data packet looks like and the fields of importance to the design.

Figure 3.2.1 shows a screenshot of a typical TCP/IP data packet. This screenshot is from the TCP Analyzer software utilized to analyze the various data packets both malicious and non-malicious used during the testing phase. The options on the left of the console allow the designer to select a particular field in the packet. The screenshot shows the source IP address selected. Further, the analyzer provides the contents of each field with an appropriate corresponding label making it less tedious for the designer to study the packet.

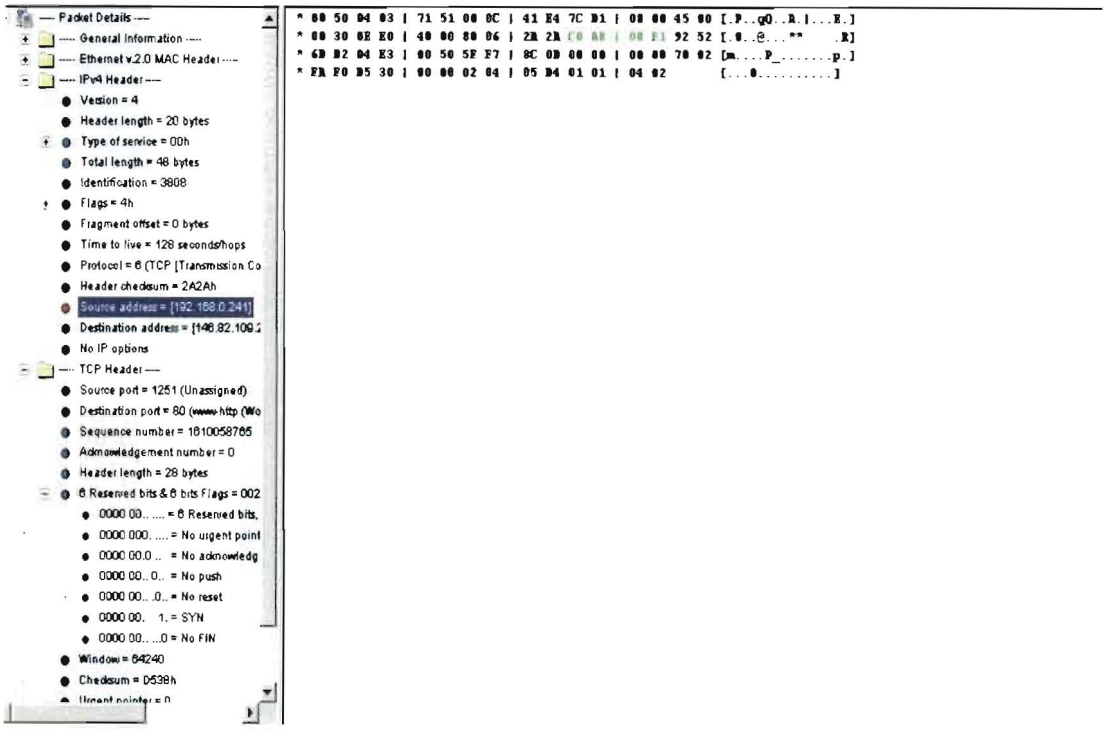


Figure 3.2.1: TCP Analyzer screenshot of a TCP/IP data packet

3.2.1 Extraction of Fields

As the system is operating at the transport layer of TCP/IP, the headers at the layer are of primary importance.

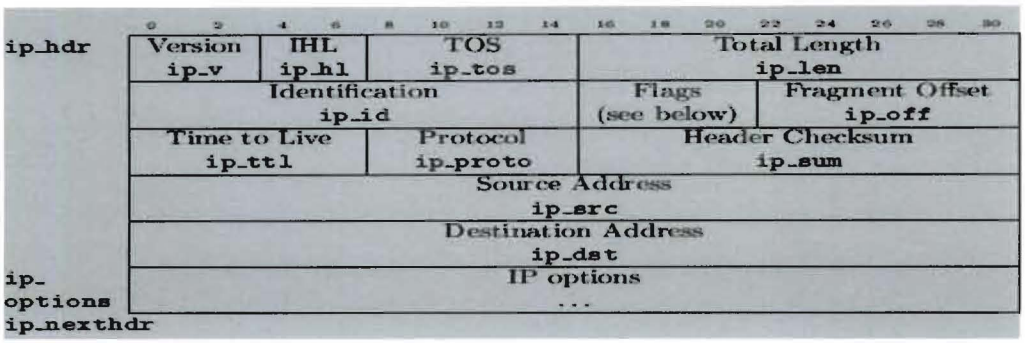
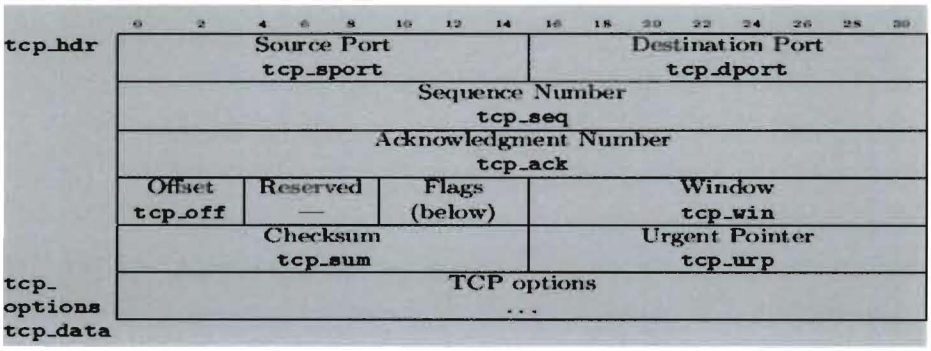


Figure 3.2.1-1: IPv4 Header Format

Figure 3.2.1-1 depicts the IPv4 (version 4) header format. The necessary fields extracted are the protocol, and source IP address fields. The next figure shows the TCP header format along with the bit positions of the control flags.



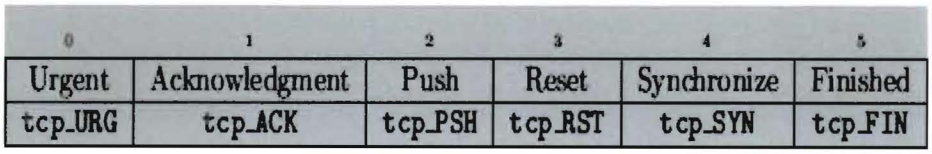


Figure 3.2.1-2: TCP/IP Header Format and Control Flags' bit positions

The TCP/IP header contains the next series of vital information. The essential fields extracted here are the source port and control flags fields.

Figure 3.2.1-3 shows the function written to carry out the extraction of the fields. A 2 byte pointer is utilized to access the OPB BRAM at the 16 bit boundary. First, the pointer is initialized to the starting address of the BRAM and then having already analyzed the TCP/IP packets in the TCP Analyzer, the necessary data is obtain by providing the correct offset to the pointer. Since the source IP address is 4 bytes long, the BRAM is accessed twice to read first its higher 2 bytes and then its 2 lower bytes. After concatenation, the final source IP address is obtained. For the source port, protocol, and control flags only one access is required. As mentioned earlier, the data is valid only if the protocol is TCP/IP.

```
void FifoRecvHandler (void *CallBackRef)
{
  XEmac *EmacPtr = (XEmac *)CallBackRef;
  XStatus Result;


        int i;
        p= 0x00000004;                          // setting the pointer to the starting address of OPB BRAM
        FrameLen = XEM_MAX_FRAME_SIZE;          // obtaining the size of the packet

        Result = XEmac_FifoRecv(EmacPtr, p, &FrameLen);

        ProtocolInfo =*(p+11);
        protocol = ProtocolInfo;                // extracting protocol

        if(protocol == 0x6)                     // checking for TCP IP protocol
        {
                xil_printf("\n\rNEW PACKET \n\r",FrameLen);
                xil_printf("Packet Size = %d bytes \n\r",FrameLen);
                xil_printf("Time of Reception = %d secs \n\r",interrupt_cnt);

                IPhigh = *(p+13);
                IPhigh = IPhigh << 16;
                IPlow = *(p+14);
                SourceIP = (IPhigh | IPlow);    // extracting source IP address
                xil_printf ("Source IP Address = %x\n\r", SourceIP);
```

```
                SourcePort=*(p+17);                    extracting source port
                xil_printf("Sorce Port = %x\n\r",SourcePort);

                FlagInfo =*(p+23);
                flags = FlagInfo;
                flags = (flags & 0x3f);                extracting control flags
                xil_printf("Flags = %x\n\r",flags);

                flag = 1;                              setting flag to indicate packet reception
                counter = counter + 1;                 counter incremented
                        }
}
```

Figure 3.2.1-3: Code for Extraction of Fields

3.2.2 Database Management

Once data is extracted it needs to be stored in a database; however it is important to distinguish between good data from a harmless source, and bad data coming from a intruder. Before storing information or updating the database, various conditions need to be satisfied. Figure 3.2.2-1 illustrates the flow chart of the code. Further, a detailed code can be found in Appendix A.

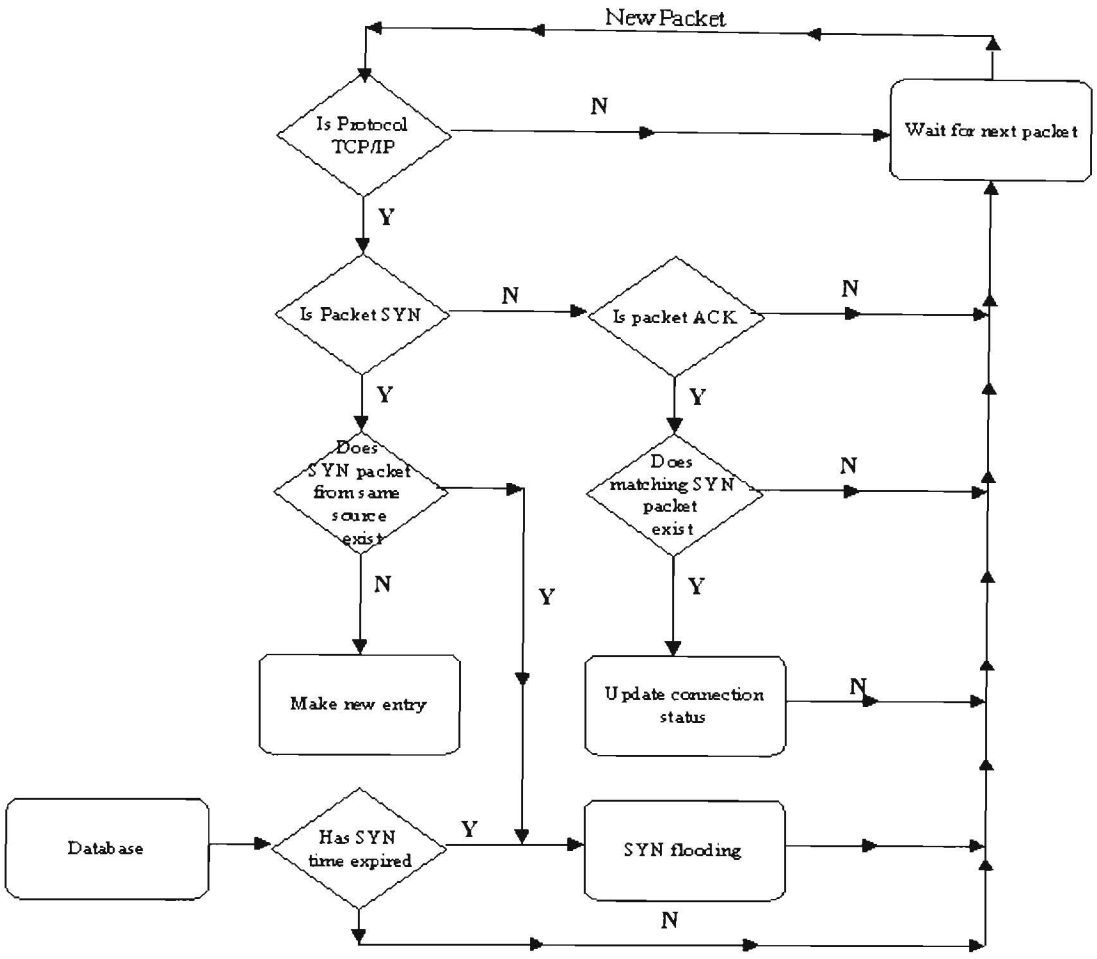


Figure 3.2.2-1: Flow Chart of Software Design

Having satisfied all the conditions shown in the flowchart, data is then added to the database. For each unique connection a 'struct' is created with the necessary fields contained in it. Using the 'struct' data structure is advantageous as it allows efficient storage of information with dissimilar data types, similar to the ones experienced in this case. Figure 3.2.2-2 shows a single structure containing data and a pointer to the next structure.

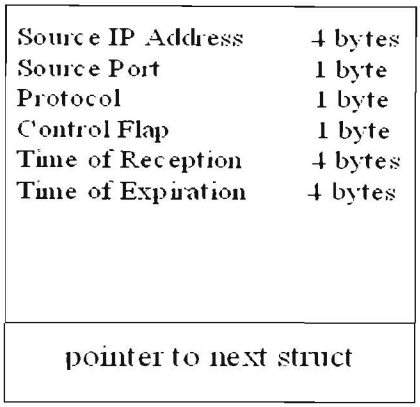


Figure 3.2.2-1: Structure showing its data contents

```
struct node
{
        Xuint32 source_addr;
        Xuint16 source_port;
        Xuint8 Flags;
        int timer_start;
        int timer_end;
        struct node * next;
};
```

Figure 3.2.2-3: Code for structure

Figure 3.2.2-3 shows the C code for the structure. The time of expiration was decided to be four more than the time of reception of packet (4 seconds). The entire database is made up of multiple such structures. In order to locate the individual structure position in memory and dynamically allocate space on a need basis so as to save memory, a linked-list data structure was employed. Each individual structure not only contains packet information, but also a pointer which points to the next incoming packet data

structure. The very last structure contains a 'NULL' in the pointer field marking the end of the linked-list. Figure 3.2.2-4 gives a pictorial depiction of the linked-list.

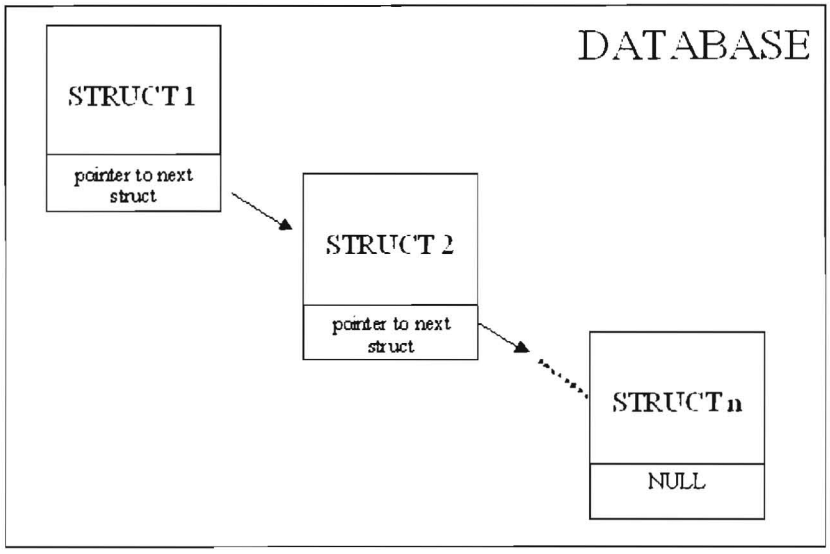


Figure 3.2.2-4: Linket-List Database

The following figure shows the C code used to maintain the data structure.

```
void addq (struct node **f, Xuint32 source_addr, Xuint16 source_port, Xuint8 Flags,int timer_start, int timer_end)

{
        struct node *r, *temp;
        temp=*f;

        if (*f==NULL)
        {
                temp = malloc(sizeof(struct node));
                temp -> source_addr = source_addr;
                temp -> source_port = source_port;
                temp -> Flags = Flags;
                temp -> timer_start = timer_start;
                temp -> timer_end = timer_end;
                temp -> next = NULL;
                *f=temp;
        }

        else
        {
                temp=*f;

                while (temp->next!=NULL)
```

```
                    temp=temp->next;

                r=malloc(sizeof(struct node));
                r -> source_addr = source_addr;
                r -> source_port = source_port;
                r -> Flags = Flags;
                r -> timer_start = timer_start;
                r -> timer_end = timer_end;
                r -> next = NULL;
                temp -> next = r;
        }
}
```

Figure 3.2.2-4: C code for Linked-List

Chapter 4

TESTING AND RESULTS

4.1 Testing Methodology

Before testing the design, it was essential to realize the non-existence of a network.. Therefore, a network was created with the designed system as the server and another system as the client. The general idea was to test all scenarios by sending data packets from the client. In such a controlled environment, the client system emulated both a legitimate network component and an intruder. Figure 4.1.1 shows a screenshot taken from the client indicating packet status at the client end.

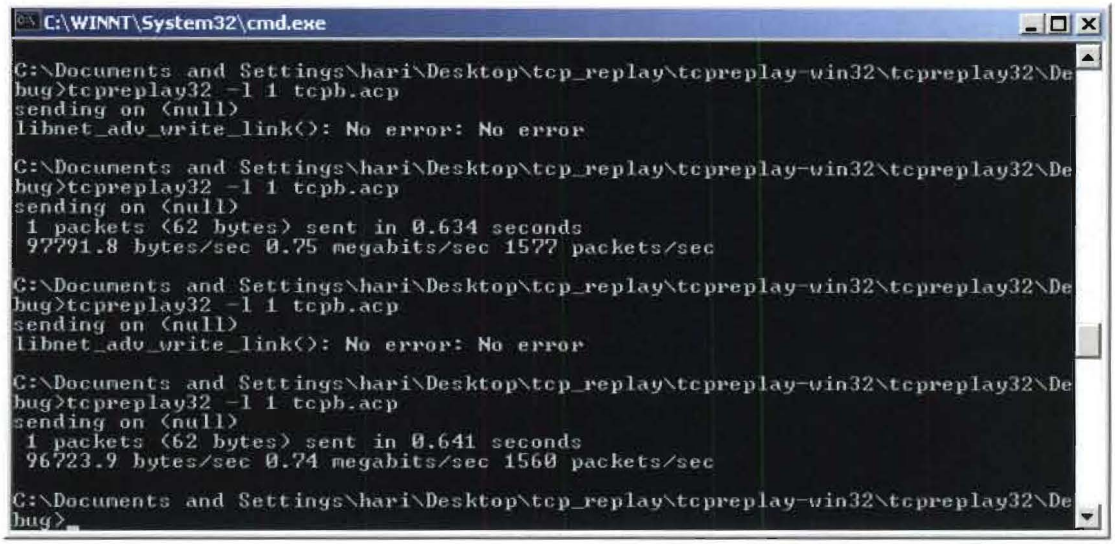


Figure 4.1.1: Packet status screenshot at client end

As witnessed above, the client end gives delivery conformation of the packet. Depending upon the case tested an appropriate response is displayed on the HyperTerminal. Figure 4.1.2 shows a similar response upon reception of a data packet.

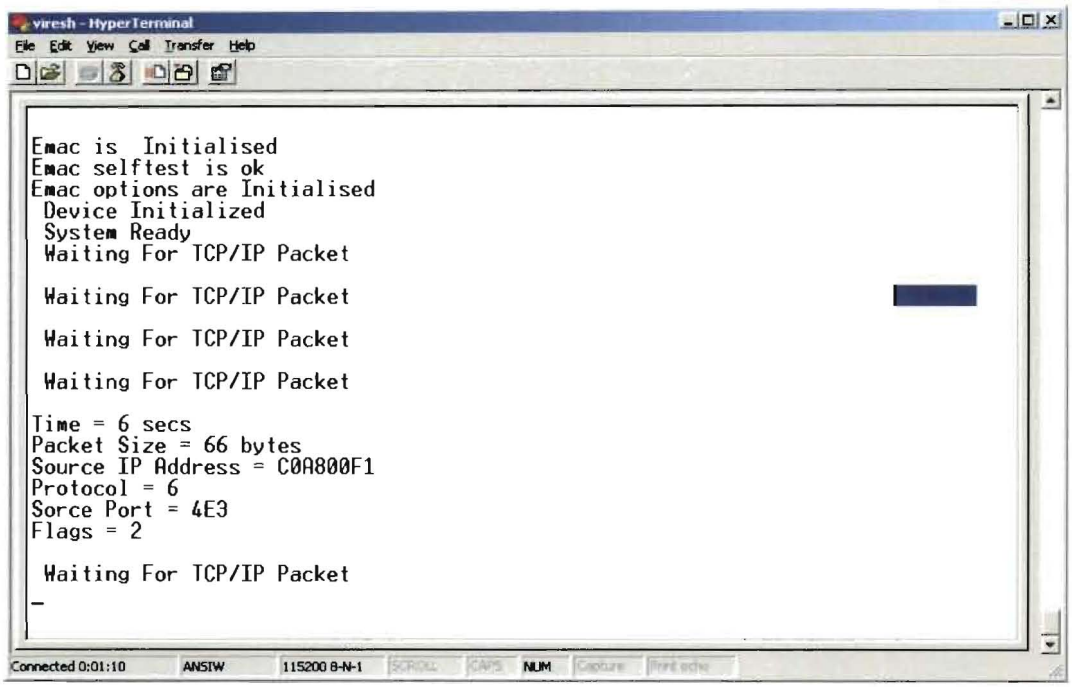


Figure 4.1.2: Server end packet reception

Once the packet is received, the user console informs the system administrator of the delivery. Further, it provides other information like the time of reception, packet size, type of packet, and the source of delivery.

4.2 Time Expiration Scenario

The first case to be tested was the time expiration scenario. A SYN packet was sent from a particular address and no ACK packet was sent to supplement it. The server

acknowledged its reception and created a new entry in the database. After waiting for 4 seconds for the ACK packet, it deemed the source as harmful and declared the attack as part of SYN Flooding. The entry was then updated (a 0xFF was written into the Flags field of the entry) to be overwritten by a subsequent packet. The packet information was displayed on the console. Figure 4.2.1 shows the screenshot for this test.

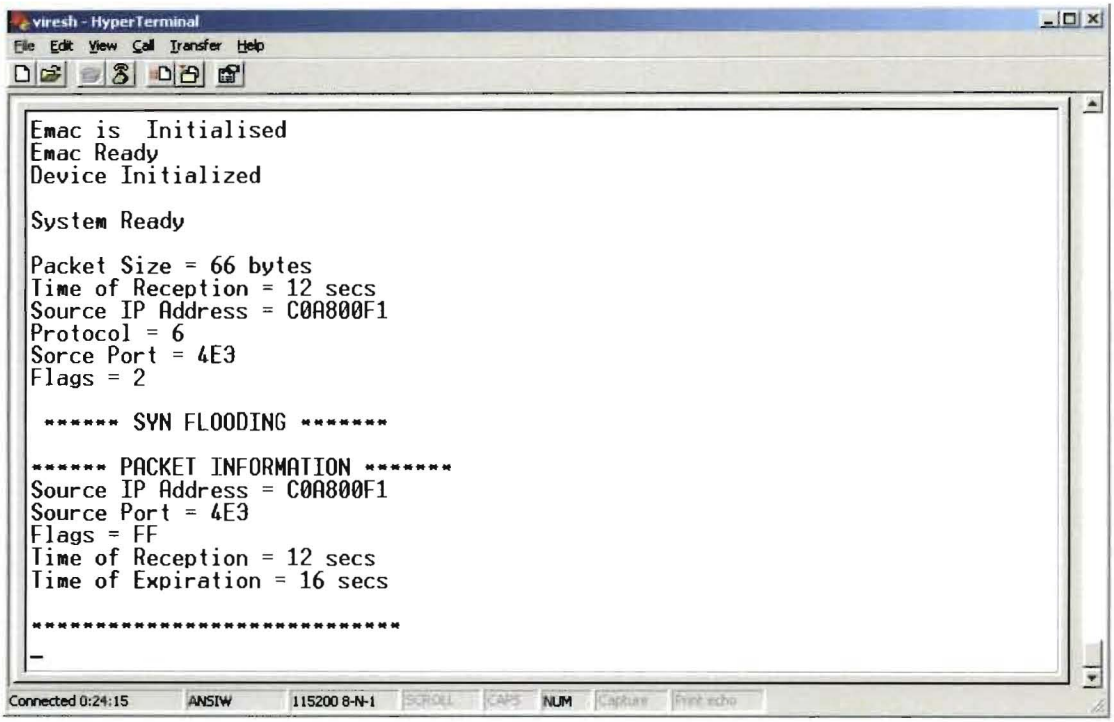


Figure 4.2.1: Time Expiration Test

4.3 Multiple SYN Packet Reception

The next test carried out was for multiple SYN reception. As discussed earlier, intruders often send multiple SYN packets to the target system hoping to fill the buffer until the server is unable to provide service to any other client. Therefore, an environment was created to test this scenario. A SYN packet was sent to the server requesting

communication and then within 4 seconds the same SYN packet was send again. Upon the reception of the second packet the IDS compared it with its existing records and found a match. This was recognized as a intentional denial service attack. Figure 4.3.1 illustrates the IDS response on the interface. As seen in the figure, the packet information is displayed along with the cause of SYN Flooding

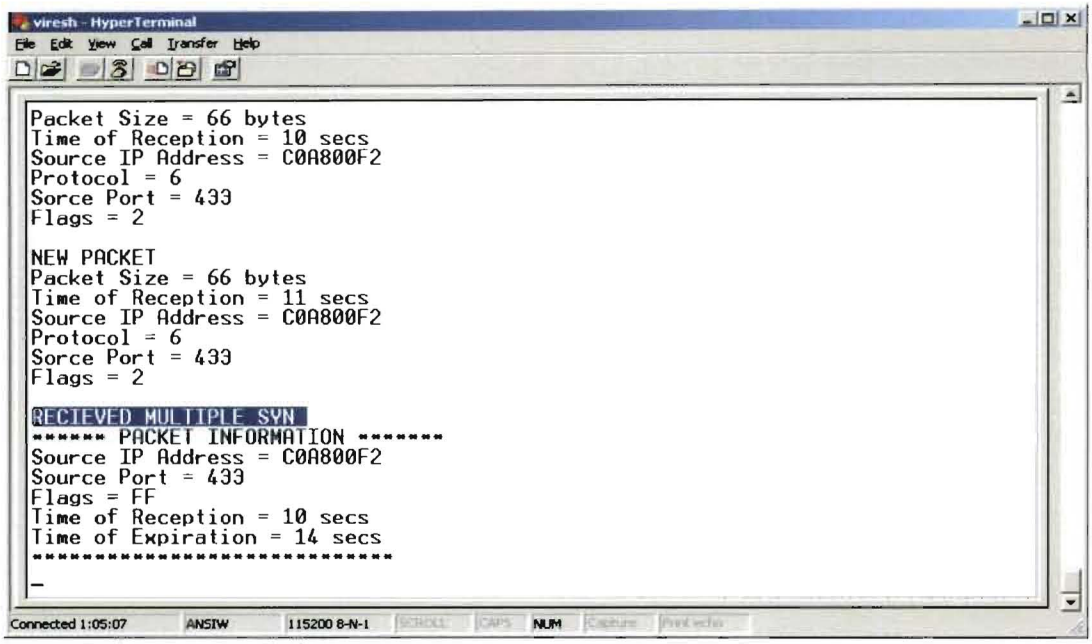


Figure 4.3.1: Multiple SYN Reception From Intruder

After the success of this test it was decided to send in multiple SYN packets again, but this time with different source addresses. The server created separate entries for both the packet after the compare with existing records resulted in a no match. Figure 4.3.2 shows the HyperTerminal results.

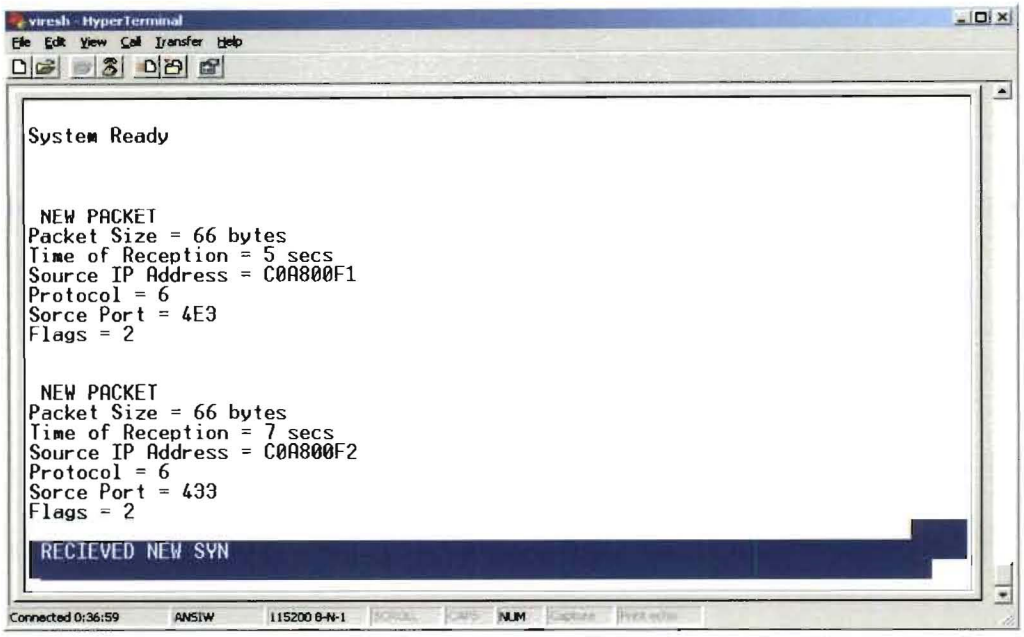


Figure 4.3.2: Multiple SYN Reception From Different Sources

4.4 Independent Acknowledge Reception

Sometimes an ACK packet arrives for which there is no matching SYN packet found. Due to network traffic it might take more than the allotted time for a packet to transfer; however, it is vital to discard its supplementing SYN packet in order to make the server less susceptible to attacks. This scenario too was tested successfully. An ACK packet was sent to which there was no matching SYN packet existing in the database. The IDS compared its information with the entire database and found no matching result. The IDS resulted discarding the packet as shown in Figure 4.4.1

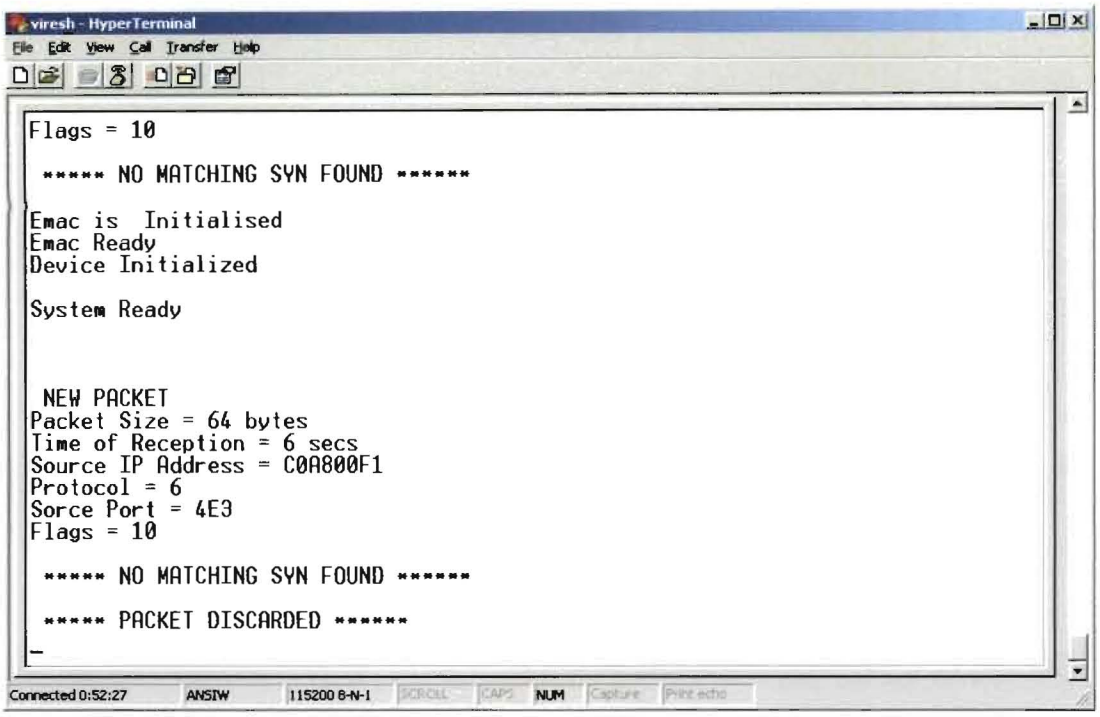


Figure 4.4.1: Independent ACK packet Scenario

4.5 Good Connection

After testing all the possible intruder scenarios, the final test carried out was for a good connection with a harmless source. A SYN packet of a certain source address and source port was sent to the server and within 4 seconds an ACK packet from the same source was sent. The IDS acknowledged the reception of the SYN packet and created a new record. Upon reception of the ACK packet it updated the current entry by changing the control flag status and declared it a good connection. The following figure shows the results of this test.

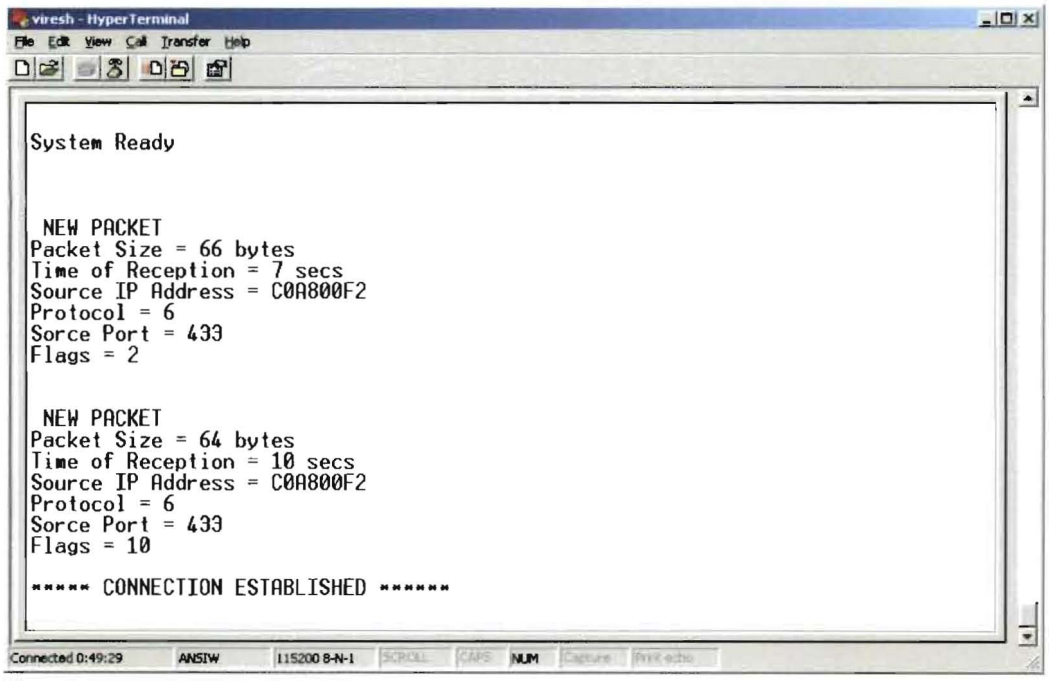


Figure 4.5.1: Good Connection

Chapter 5

CONCLUSION

5.1 Conclusion

A considerable amount of research on the TCP/IP protocol was done in order to completely understand the nuances of the problem statement in hand. Such an extensive research was instrumental during the design process. The designer had to think as both, the server under attack and the intruder attacking the server. Although helpful, viewing the design from two different perspectives resulted in several changes in the code before yielding the finished product.

It is to be noted that the IDS is restricted to a controlled environment. The database management is done in software, and therefore is slower compared to its hardware counterpart. The field of network security is rapidly growing and research in such field was an educational experience worth re-visiting. In conclusion, the design and implementation of a host-based embedded Intrusion Detection System for SYN Flooding detection using FPGA technology was successful.

REFERENCES

1. Kozierok, Charles M. "The TCP/IP Guide." No starch Press, Inc. CA. 2005

2. AMIRIX. <www.amirix.com>

3. Xilinx. <www.xilinx.com>

APPENDIX A

C Code

```
#include "xparameters.h"
#include "xbasic_types.h"
#include "xgpio.h"
#include "xexception_l.h"
#include "xintc_l.h"
#include "xintc.h"
#include "xemac.h"
#include "xemac_l.h"
#include "xuartlite_l.h"
#include "xtime_l.h"
#include "xtmrctr_l.h"
#include "xtmrctr.h"


#define XEM_MAX_FRAME_SIZE_IN_WORDS ((XEM_MAX_FRAME_SIZE /
sizeof(Xuint32)) + 1)
void FifoRecvHandler(void *CallBackRef);
void FifoSendHandler(void *CallBackRef);
void ErrorHandler(void *CallBackRef);
XEmac Emac;
XIntc IntCntr;
Xuint8 *FramePtr;
Xuint32 FrameLen;
Xuint32 RecvBuffer[XEM_MAX_FRAME_SIZE_IN_WORDS];
int result, receive;
Xuint16 *p;
int counter = 0,flag=0,flag1=0,flag2=0,timer_flag=0;
int interrupt_cnt=0,set=0;
int timer_flag;
XTmrCtr timer;

Xuint32 SourceIP, IPhigh, IPlow;
Xuint16 SourcePort, ProtocolInfo, FlagInfo;
Xuint8 protocol, flags;



/*Set Mac */
static Xuint8 LocalAddress[XEM_MAC_ADDR_SIZE] =
{
    0x06, 0x05, 0x04, 0x03, 0x02, 0x01
};

struct node
{
      Xuint32 source_addr;
      Xuint16 source_port;
      Xuint8 Flags;
      int timer_start;
      int timer_end;
```

```
      struct node * next;
};

void addq(struct node **,Xuint32,Xuint16,Xuint8,int,int);
void displayq(struct node *);
void deleteq(struct node **,struct node **,int);
struct node *front;
int temp=0;



//////// INTERRUPT HANDLER FOR TIMER
//////////////////////////##################################
void timer_int_handler(void * baseaddr_p)
{
      unsigned int csr;


      // Read timer0 CSR to see if it raised the interrupt
      csr = XTmrCtr_mGetControlStatusReg(0x41C00000, 0);

      if (csr & XTC_CSR_INT_OCCURED_MASK)
      {
                interrupt_cnt = interrupt_cnt + 1;
                  timer_flag=1;
                  //xil_printf("\n\rTime = %d secs\n\r",interrupt_cnt);


          // Clear the timer interrupt
            XTmrCtr_mSetControlStatusReg(0x41C00000, 0, csr);

    }



}

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

XStatus SetupInterruptSystem(XEmac *InstancePtr)
{

      result = XIntc_Initialize(&IntCntr,XPAR_OPB_INTC_0_DEVICE_ID);
    if (result != XST_SUCCESS)
    {
        xil_printf("\n\rInterrupt handler is not Initialised");
    }
      else
      {
            xil_printf("\n\rInterrupt handler is  Initialised");
      }

      result = XIntc_Connect(&IntCntr,

XPAR_OPB_INTC_0_ETHERNET_1_IP2INTC_IRPT_INTR,
```

```
                                (XInterruptHandler)XEmac_IntrHandlerFifo,
                                &Emac);
    if (result != XST_SUCCESS)
    {
        xil_printf("\n\rInterrupt handler is not connected");
    }
      else
      {
            xil_printf("\n\rInterrupt handler is connected");
      }

XIntc_Start(&IntCntr, XIN_REAL_MODE);

XIntc_Enable(&IntCntr,XPAR_OPB_INTC_0_ETHERNET_1_IP2INTC_IRPT_INTR);

XEmac_IntrHandlerFifo(&Emac);



XExc_Init();

XExc_RegisterHandler(XEXC_ID_NON_CRITICAL_INT,
(XExceptionHandler)XIntc_VoidInterruptHandler, &IntCntr);
XExc_RegisterHandler(XEXC_ID_NON_CRITICAL_INT,
                          (XExceptionHandler)XIntc_InterruptHandler,
                          &IntCntr);
      XExc_RegisterHandler(XEXC_ID_NON_CRITICAL_INT,

      (XExceptionHandler)timer_int_handler, (void
*)XPAR_OPB_TIMER_1_BASEADDR);

      XExc_mEnableExceptions(XEXC_NON_CRITICAL);


}
/*****************************************************************************
***************/


void main()
{
    XEmac *InstancePtr = &Emac;
      Xuint16 DeviceId =  XPAR_ETHERNET_1_DEVICE_ID;
    XStatus Result;
    Xuint32 Options;
      XEmac_Config *ConfigPtr;

      struct node *temp1,*timertemp;

      Xuint16 XPAR_OPB_TIMER_0_DEVICE_ID;

      front = NULL;
      temp1 = NULL;


      XTmrCtr_Initialize(&timer,XPAR_OPB_TIMER_0_DEVICE_ID);
```

```
      XTmrCtr_SetResetValue(&timer,0,0);

      XTmrCtr_Reset(&timer,0);

      /*Configure device for NO DMA setup*/
    ConfigPtr = XEmac_LookupConfig(DeviceId);
    ConfigPtr->IpIfDmaConfig = XEM_CFG_NO_DMA;



    Result = XEmac_Initialize(&Emac, DeviceId);
    if (Result != XST_SUCCESS) xil_printf("\n\rEmac is not
Initialised");
      else xil_printf("\n\rEmac is  Initialised");

    if (XEmac_mIsDma(&Emac)) xil_printf("\n\rDMA present");

    Result = XEmac_SelfTest(&Emac);
    if (Result != XST_SUCCESS) xil_printf("\n\rEmac selftest is not
ok");
      else xil_printf("\n\rEmac Ready");

    Options = ( /*XEM_UNICAST_OPTION*/  XEM_BROADCAST_OPTION |
XEM_PROMISC_OPTION | XEM_FDUPLEX_OPTION | XEM_INSERT_PAD_OPTION);

    Result = XEmac_SetOptions(&Emac, Options);
    if (Result != XST_SUCCESS) xil_printf("\n\rEmac options are not
Initialised");

      SetupInterruptSystem(InstancePtr);
      XEmac_SetFifoSendHandler(&Emac, &Emac, FifoSendHandler);
      XEmac_SetFifoRecvHandler(&Emac, &Emac, FifoRecvHandler);
      XEmac_SetErrorHandler(&Emac, &Emac, ErrorHandler);
      XEmac_mEnable(XPAR_ETHERNET_1_BASEADDR);

      Result = XEmac_Start (&Emac);
      if(Result!= XST_SUCCESS) xil_printf("\n\rThe device has not
started successfully");
      else xil_printf("\n\rDevice Initialized\n\r");

      XEmac_ClearStats(&Emac);


    xil_printf("\n\rSystem Ready\n\r\n\r");



////////////////////////// ENABLE INTERRUPTS ///////////////////////

      XExc_Init();            // Initialising vector table
```

```
      XExc_RegisterHandler((Xuint8)5,
(XExceptionHandler)XIntc_LowLevelInterruptHandler, (void *)0);     //
registering handlers
//
//    // Start the interrupt controller
      XIntc_mMasterEnable(0x41200000);


      // set the number of cycles the timer counts before interrupting
 XTmrCtr_mSetLoadReg(0x41C00000, 0, 100000000);


      // reset the timers, and clear interrupts and enable capture
      XTmrCtr_mSetControlStatusReg(0x41C00000, 0,
XTC_CSR_INT_OCCURED_MASK | XTC_CSR_LOAD_MASK);


      // Enable timer and uart interrupts in the interrupt controller
    XIntc_mEnableIntr(0x41200000,3);

      // start the timers
      XTmrCtr_mSetControlStatusReg(0x41C00000,
0,XTC_CSR_ENABLE_TMR_MASK | XTC_CSR_ENABLE_INT_MASK |
      XTC_CSR_AUTO_RELOAD_MASK | XTC_CSR_DOWN_COUNT_MASK);


 XExc_mEnableExceptions(XEXC_NON_CRITICAL);     // enable exceptions


      while(1)
      {

                  if(timer_flag==1)
                  {

                        timertemp=front;
                        timer_flag=0;
                        while(timertemp!=NULL)
                        {
                              if((timertemp-
>Flags==0x02)&&(interrupt_cnt > timertemp->timer_end))
                              {

                                    timertemp->Flags=0xff;
                                    xil_printf("\n\r ****** SYN
FLOODING *******\n\r");
                                    displayq(timertemp);

                              }
                              timertemp = timertemp->next;
                        }
                  }


            else if (flag ==1)
            {
                  temp1=front;
```

```
                  if ((counter == 1) && (flags == 0x02))
                  {

                    addq(&front, SourceIP, SourcePort, flags,
interrupt_cnt, (interrupt_cnt+4));
                    flag = 0;
                    flag1 = 0;
                  }
                  else if ((counter == 1) && (flags == 0x10))
                  {
                        xil_printf("\n\r ***** NO MATCHING SYN FOUND
****** \n\r");
                        xil_printf("\n\r ***** PACKET DISCARDED ******
\n\r");
                        flag = 0;
                        flag1 = 0;
                  }

                  else
                  {
                        while(temp1!=NULL)
                        {

                              if (flags == 0x02)
                              {

                                    if ((temp1->source_addr ==
SourceIP) && (temp1->Flags!=0xff))
                                          {
                                          xil_printf("\n\rRECIEVED
MULTIPLE SYN \n\r");
                                          temp1->Flags = 0xff;
                                          displayq(temp1);
                                          flag1=0;
                                          flag=0;
                                          break;
                                          }

                                    else if ((temp1->Flags==0xff))
                                          {
                                          xil_printf("\n\r RECIEVED new
SYN with update\n\r");
                                          temp1 -> source_addr =
SourceIP;
                                          temp1 -> source_port =
SourcePort;
                                          temp1-> Flags = 0x02;
                                          temp1 -> timer_start =
interrupt_cnt;
                                          temp1 -> timer_end =
(interrupt_cnt+4);


                                          flag1=0;
                                          flag=0;
                                          break;
```

```
                                        }

                                    else
                                    {
                                        flag1=1;
                                        break;
                                    }
                                }

                                else if (flags == 0x10)
                                    {

                                        if ((temp1->source_addr ==
SourceIP) && (temp1->Flags==0x02))
                                            {
                                                temp1-> Flags =
0x10;

      xil_printf("\n\r***** CONNECTION ESTABLISHED ****** \n\r");

                                                flag=0;
                                                flag1=0;
                                                break;
                                            }

                                        else if ((temp1->source_addr
== SourceIP) &&(temp1->Flags!=0x02))
                                            {
                                                xil_printf("\n\r
***** NO MATCHING SYN FOUND ****** \n\r");
                                                xil_printf("\n\r
***** PACKET DISCARDED ****** \n\r");

                                                flag=0;
                                                flag1=0;

                                                break;
                                            }

                                    }

                                temp1 = temp1->next;
                            }
                        }


                        if(flag1==1)
                        {
                                xil_printf("\n\r RECIEVED NEW SYN \n\r");
                                addq(&temp1, SourceIP, SourcePort, flags,
interrupt_cnt, (interrupt_cnt+4));
                                flag=0;
                        }

                }
```

```
///*********************************************************************
**********/
 void FifoSendHandler(void *CallBackRef)
{
    xil_printf("\n\rSend handler");
}
///*********************************************************************
**********/
 void ErrorHandler(void *CallBackRef)
{
    xil_printf("\n\rError handler");
}
//
///************************ adding element to the datastructure
*********************

void addq(struct node **f, Xuint32 source_addr, Xuint16 source_port,
Xuint8 Flags,int timer_start, int timer_end)

{
      struct node *r,*temp;


      temp=*f;

      if (*f==NULL)
      {
            temp = malloc(sizeof(struct node));
            temp -> source_addr = source_addr;
            temp -> source_port = source_port;
            temp -> Flags = Flags;
            temp -> timer_start = timer_start;
            temp -> timer_end = timer_end;
            temp -> next = NULL;
            *f=temp;
            set++;

      }


      else
      {

            temp=*f

            while (temp->next!=NULL)
                  temp=temp->next;


            r=malloc(sizeof(struct node));

            r -> source_addr = source_addr;
            r -> source_port = source_port;
            r -> Flags = Flags;
            r -> timer_start = timer_start;
            r -> timer_end = timer_end;
            r -> next = NULL;
```

```
                temp -> next = r;
                set++;

        }

}

///* ********************** displaying the datastructure
**************************** */

void displayq(struct node *q)
{

        while(q!=NULL)
        {
                if(q->next == NULL)
                        xil_printf("****** PACKET INFORMATION *******\n\r");
                xil_printf("Source IP Address = %x\n\r",q->source_addr);
                xil_printf("Source Port = %x\n\r",q->source_port);
                xil_printf("Flags = %x\n\r",q->Flags);
                xil_printf("Time of Reception = %d secs\n\r",q-
>timer_start);
                xil_printf("Time of Expiration = %d secs\n\r",q-
>timer_end);
                xil_printf("*****************************\n\r");
                q = q->next;

        }
}
```

APPENDIX B

MHS File

```
##############################################################################
#
# Created by Base System Builder Wizard for Xilinx EDK 8.1.02 Build EDK_I.20.4
# Tue Nov 21 12:17:26 2006
# Target Board:  AMIRIX Systems AP130 PCI Platform FPGA Development Board Rev 1
# Family:        virtex2p
# Device:        XC2VP30
# Package:       FF896
# Speed Grade:  -6
# Processor: PPC 405
# Processor clock frequency: 100.000000 MHz
# Bus clock frequency: 100.000000 MHz
# Debug interface: FPGA JTAG
# On Chip Memory :  72 KB
#
##############################################################################

 PARAMETER VERSION = 2.1.0


 PORT fpga_0_ethernet_1_PHY_tx_clk_pin = fpga_0_ethernet_1_PHY_tx_clk, DIR = I
 PORT fpga_0_ethernet_1_PHY_rx_clk_pin = fpga_0_ethernet_1_PHY_rx_clk, DIR = I
 PORT fpga_0_RS232_1_RX_pin = fpga_0_RS232_1_RX, DIR = I
 PORT fpga_0_RS232_1_TX_pin = fpga_0_RS232_1_TX, DIR = O
 PORT fpga_0_LED_SW_GPIO_IO_pin = fpga_0_LED_SW_GPIO_IO, DIR = IO, VEC = [0:1]
 PORT fpga_0_ethernet_1_PHY_tx_en_pin = fpga_0_ethernet_1_PHY_tx_en, DIR = O
 PORT fpga_0_ethernet_1_PHY_tx_er_pin = fpga_0_ethernet_1_PHY_tx_er, DIR = O
 PORT fpga_0_ethernet_1_PHY_tx_data_pin = fpga_0_ethernet_1_PHY_tx_data, DIR = O, VEC
= [3:0]
 PORT fpga_0_ethernet_1_PHY_dv_pin = fpga_0_ethernet_1_PHY_dv, DIR = I
 PORT fpga_0_ethernet_1_PHY_rx_er_pin = fpga_0_ethernet_1_PHY_rx_er, DIR = I
 PORT fpga_0_ethernet_1_PHY_rx_data_pin = fpga_0_ethernet_1_PHY_rx_data, DIR = I, VEC
= [3:0]
 PORT fpga_0_ethernet_1_PHY_crs_pin = fpga_0_ethernet_1_PHY_crs, DIR = I
 PORT fpga_0_ethernet_1_PHY_col_pin = fpga_0_ethernet_1_PHY_col, DIR = I
 PORT fpga_0_ethernet_1_PHY_Mii_clk_pin = fpga_0_ethernet_1_PHY_Mii_clk, DIR = IO
 PORT fpga_0_ethernet_1_PHY_Mii_data_pin = fpga_0_ethernet_1_PHY_Mii_data, DIR = IO
 PORT sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = DCMCLK
 PORT sys_rst_pin = sys_rst_s, DIR = I


BEGIN ppc405
 PARAMETER INSTANCE = ppc405_0
 PARAMETER HW_VER = 2.00.c
 BUS_INTERFACE JTAGPPC = jtagppc_0_0
```

```
 BUS_INTERFACE IPLB = plb
 BUS_INTERFACE DPLB = plb
 PORT PLBCLK = sys_clk_s
 PORT C405RSTCHIPRESETREQ = C405RSTCHIPRESETREQ
 PORT C405RSTCORERESETREQ = C405RSTCORERESETREQ
 PORT C405RSTSYSRESETREQ = C405RSTSYSRESETREQ
 PORT RSTC405RESETCHIP = RSTC405RESETCHIP
 PORT RSTC405RESETCORE = RSTC405RESETCORE
 PORT RSTC405RESETSYS = RSTC405RESETSYS
 PORT EICC405EXTINPUTIRQ = EICC405EXTINPUTIRQ
 PORT CPMC405CLOCK = sys_clk_s
END

BEGIN ppc405
 PARAMETER INSTANCE = ppc405_1
 PARAMETER HW_VER = 2.00.c
 BUS_INTERFACE JTAGPPC = jtagppc_0_1
END

BEGIN jtagppc_cntlr
 PARAMETER INSTANCE = jtagppc_0
 PARAMETER HW_VER = 2.00.a
 BUS_INTERFACE JTAGPPC0 = jtagppc_0_0
 BUS_INTERFACE JTAGPPC1 = jtagppc_0_1
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = reset_block
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT Ext_Reset_In = sys_rst_s
 PORT Slowest_sync_clk = sys_clk_s
 PORT Chip_Reset_Req = C405RSTCHIPRESETREQ
 PORT Core_Reset_Req = C405RSTCORERESETREQ
 PORT System_Reset_Req = C405RSTSYSRESETREQ
 PORT Rstc405resetchip = RSTC405RESETCHIP
 PORT Rstc405resetcore = RSTC405RESETCORE
 PORT Rstc405resetsys = RSTC405RESETSYS
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Dcm_locked = dcm_0_lock
END

BEGIN plb_v34
 PARAMETER INSTANCE = plb
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT SYS_Rst = sys_bus_reset
 PORT PLB_Clk = sys_clk_s
END

BEGIN opb_v20
 PARAMETER INSTANCE = opb
 PARAMETER HW_VER = 1.10.c
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT SYS_Rst = sys_bus_reset
```

```
 PORT OPB_Clk = sys_clk_s
END

BEGIN plb2opb_bridge
 PARAMETER INSTANCE = plb2opb
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_NUM_ADDR_RNG = 1
 PARAMETER C_RNG0_BASEADDR = 0x00000000
 PARAMETER C_RNG0_HIGHADDR = 0x7fffffff
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE MOPB = opb
 PORT PLB_Clk = sys_clk_s
 PORT OPB_Clk = sys_clk_s
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = RS232_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BAUDRATE = 115200
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_ODD_PARITY = 0
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_CLK_FREQ = 100000000
 PARAMETER C_BASEADDR = 0x40600000
 PARAMETER C_HIGHADDR = 0x4060ffff
 BUS_INTERFACE SOPB = opb
 PORT OPB_Clk = sys_clk_s
 PORT RX = fpga_0_RS232_1_RX
 PORT TX = fpga_0_RS232_1_TX
END

BEGIN opb_gpio
 PARAMETER INSTANCE = LED_SW
 PARAMETER HW_VER = 3.01.b
 PARAMETER C_GPIO_WIDTH = 2
 PARAMETER C_IS_DUAL = 0
 PARAMETER C_IS_BIDIR = 1
 PARAMETER C_ALL_INPUTS = 0
 PARAMETER C_BASEADDR = 0x40000000
 PARAMETER C_HIGHADDR = 0x4000ffff
 BUS_INTERFACE SOPB = opb
 PORT OPB_Clk = sys_clk_s
 PORT GPIO_IO = fpga_0_LED_SW_GPIO_IO
END

BEGIN opb_ethernet
 PARAMETER INSTANCE = ethernet_1
 PARAMETER HW_VER = 1.04.a
 PARAMETER C_DMA_PRESENT = 1
 PARAMETER C_IPIF_RDFIFO_DEPTH = 32768
 PARAMETER C_IPIF_WRFIFO_DEPTH = 32768
 PARAMETER C_OPB_CLK_PERIOD_PS = 10000
 PARAMETER C_BASEADDR = 0x40c00000
 PARAMETER C_HIGHADDR = 0x40c0ffff
 BUS_INTERFACE SOPB = opb
```

```
PORT OPB_Clk = sys_clk_s
PORT IP2INTC_Irpt = ethernet_1_IP2INTC_Irpt
PORT PHY_tx_clk = fpga_0_ethernet_1_PHY_tx_clk
PORT PHY_rx_clk = fpga_0_ethernet_1_PHY_rx_clk
PORT PHY_tx_en = fpga_0_ethernet_1_PHY_tx_en
PORT PHY_tx_er = fpga_0_ethernet_1_PHY_tx_er
PORT PHY_tx_data = fpga_0_ethernet_1_PHY_tx_data
PORT PHY_dv = fpga_0_ethernet_1_PHY_dv
PORT PHY_rx_er = fpga_0_ethernet_1_PHY_rx_er
PORT PHY_rx_data = fpga_0_ethernet_1_PHY_rx_data
PORT PHY_crs = fpga_0_ethernet_1_PHY_crs
PORT PHY_col = fpga_0_ethernet_1_PHY_col
PORT PHY_Mii_clk = fpga_0_ethernet_1_PHY_Mii_clk
PORT PHY_Mii_data = fpga_0_ethernet_1_PHY_Mii_data
END

BEGIN plb_bram_if_cntlr
 PARAMETER INSTANCE = plb_bram_if_cntlr_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER c_plb_clk_period_ps = 10000
 PARAMETER c_baseaddr = 0xffff0000
 PARAMETER c_highaddr = 0xffffffff
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE PORTA = plb_bram_if_cntlr_1_port
 PORT PLB_Clk = sys_clk_s
END

BEGIN bram_block
 PARAMETER INSTANCE = plb_bram_if_cntlr_1_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = plb_bram_if_cntlr_1_port
END

BEGIN opb_bram_if_cntlr
 PARAMETER INSTANCE = opb_bram_if_cntlr_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER c_opb_clk_period_ps = 10000
 PARAMETER c_baseaddr = 0x00000000
 PARAMETER c_highaddr = 0x00001fff
 BUS_INTERFACE SOPB = opb
 BUS_INTERFACE PORTA = opb_bram_if_cntlr_1_port
 PORT OPB_Clk = sys_clk_s
END

BEGIN bram_block
 PARAMETER INSTANCE = opb_bram_if_cntlr_1_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = opb_bram_if_cntlr_1_port
END

BEGIN opb_timer
 PARAMETER INSTANCE = opb_timer_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_COUNT_WIDTH = 32
 PARAMETER C_ONE_TIMER_ONLY = 1
 PARAMETER C_BASEADDR = 0x41c00000
```

```
 PARAMETER C_HIGHADDR = 0x41c0ffff
 BUS_INTERFACE SOPB = opb
 PORT OPB_Clk = sys_clk_s
 PORT Interrupt = opb_timer_1_Interrupt
 PORT CaptureTrig0 = net_gnd
END

BEGIN opb_intc
 PARAMETER INSTANCE = opb_intc_0
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0x41200000
 PARAMETER C_HIGHADDR = 0x4120ffff
 BUS_INTERFACE SOPB = opb
 PORT Irq = EICC405EXTINPUTIRQ
 PORT Intr = ethernet_1_IP2INTC_Irpt & opb_timer_1_Interrupt
END

BEGIN dcm_module
 PARAMETER INSTANCE = dcm_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_CLK0_BUF = TRUE
 PARAMETER C_CLKFX_BUF = TRUE
 PARAMETER C_CLKFX_DIVIDE = 2
 PARAMETER C_CLKFX_MULTIPLY = 5
 PARAMETER C_CLKIN_PERIOD = 25.000000
 PARAMETER C_CLK_FEEDBACK = 1X
 PARAMETER C_DFS_FREQUENCY_MODE = LOW
 PARAMETER C_DLL_FREQUENCY_MODE = LOW
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT CLKIN = dcm_clk_s
 PORT CLKFX = sys_clk_s
 PORT CLK0 = dcm_0_FB
 PORT CLKFB = dcm_0_FB
 PORT RST = net_gnd
 PORT LOCKED = dcm_0_lock
END
```

APPENDIX C

MSS File

```
PARAMETER VERSION = 2.2.0


BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_0
 PARAMETER STDIN = RS232_1
 PARAMETER STDOUT = RS232_1
END

BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_1
END


BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu_ppc405
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc405_0
 PARAMETER COMPILER = powerpc-eabi-gcc
 PARAMETER ARCHIVER = powerpc-eabi-ar
 PARAMETER CORE_CLOCK_FREQ_HZ = 100000000
END

BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu_ppc405
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc405_1
 PARAMETER COMPILER = powerpc-eabi-gcc
 PARAMETER ARCHIVER = powerpc-eabi-ar
END


BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = jtagppc_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = plbarb
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = plb
```

```
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = opbarb
 PARAMETER DRIVER_VER = 1.02.a
 PARAMETER HW_INSTANCE = opb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = plb2opb
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = plb2opb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = RS232_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = gpio
 PARAMETER DRIVER_VER = 2.01.a
 PARAMETER HW_INSTANCE = LED_SW
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = emac
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = ethernet_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = plb_bram_if_cntlr_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = opb_bram_if_cntlr_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = tmrctr
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER HW_INSTANCE = opb_timer_1
 PARAMETER int_handler = timer_int_handler, int_port = Interrupt
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = intc
 PARAMETER DRIVER_VER = 1.00.c
 PARAMETER HW_INSTANCE = opb_intc_0
END
```

APPENDIX D

UCF

```
##############################################################################
## This system.ucf file is generated by Base System Builder based on the
## settings in the selected Xilinx Board Definition file. Please add other
## user constraints to this file based on customer design specifications.
##############################################################################

Net sys_clk_pin LOC=F15;
Net sys_clk_pin IOSTANDARD=LVTTL;
Net sys_rst_pin LOC=R4;
Net sys_rst_pin IOSTANDARD=LVTTL;
## System level constraints
Net sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 25000 ps;
Net sys_rst_pin TIG;
NET "C405RSTCORERESETREQ" TPTHRU = "RST_GRP";
NET "C405RSTCHIPRESETREQ" TPTHRU = "RST_GRP";
NET "C405RSTSYSRESETREQ" TPTHRU = "RST_GRP";
TIMESPEC "TS_RST1" = FROM CPUS THRU RST_GRP TO FFS  TIG;

## IO Devices constraints

#### Module RS232_1 constraints

Net fpga_0_RS232_1_RX_pin LOC=H9;
Net fpga_0_RS232_1_RX_pin IOSTANDARD=LVTTL | DRIVE = 12 | SLOW;
Net fpga_0_RS232_1_TX_pin LOC=C30;
Net fpga_0_RS232_1_TX_pin IOSTANDARD=LVTTL | DRIVE = 12 | SLOW;

#### Module LED_SW constraints

Net fpga_0_LED_SW_GPIO_IO_pin<0> LOC=AD8;
Net fpga_0_LED_SW_GPIO_IO_pin<0> IOSTANDARD=LVTTL | DRIVE = 12 | SLOW | Pullup;
Net fpga_0_LED_SW_GPIO_IO_pin<1> LOC=AE15;
Net fpga_0_LED_SW_GPIO_IO_pin<1> IOSTANDARD=LVTTL | DRIVE = 12 | SLOW | Pullup;

#### Module ethernet_1 constraints

Net fpga_0_ethernet_1_PHY_tx_clk_pin LOC=C16;
Net fpga_0_ethernet_1_PHY_tx_clk_pin IOSTANDARD=LVCMOS33;
Net fpga_0_ethernet_1_PHY_rx_clk_pin LOC=B16;
Net fpga_0_ethernet_1_PHY_rx_clk_pin IOSTANDARD=LVCMOS33;
Net fpga_0_ethernet_1_PHY_tx_en_pin LOC=R28;
Net fpga_0_ethernet_1_PHY_tx_en_pin IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_tx_er_pin LOC=R27;
Net fpga_0_ethernet_1_PHY_tx_er_pin IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_tx_data_pin<0> LOC=R26;
Net fpga_0_ethernet_1_PHY_tx_data_pin<0> IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
```

```
Net fpga_0_ethernet_1_PHY_tx_data_pin<1> LOC=R25;
Net fpga_0_ethernet_1_PHY_tx_data_pin<1> IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_tx_data_pin<2> LOC=T29;
Net fpga_0_ethernet_1_PHY_tx_data_pin<2> IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_tx_data_pin<3> LOC=R29;
Net fpga_0_ethernet_1_PHY_tx_data_pin<3> IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_dv_pin LOC=P29;
Net fpga_0_ethernet_1_PHY_dv_pin IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_rx_er_pin LOC=P28;
Net fpga_0_ethernet_1_PHY_rx_er_pin IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_rx_data_pin<0> LOC=N28;
Net fpga_0_ethernet_1_PHY_rx_data_pin<0> IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_rx_data_pin<1> LOC=N27;
Net fpga_0_ethernet_1_PHY_rx_data_pin<1> IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_rx_data_pin<2> LOC=P24;
Net fpga_0_ethernet_1_PHY_rx_data_pin<2> IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_rx_data_pin<3> LOC=P23;
Net fpga_0_ethernet_1_PHY_rx_data_pin<3> IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_crs_pin LOC=R22;
Net fpga_0_ethernet_1_PHY_crs_pin IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_col_pin LOC=P22;
Net fpga_0_ethernet_1_PHY_col_pin IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_Mii_clk_pin LOC=N29;
Net fpga_0_ethernet_1_PHY_Mii_clk_pin IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
Net fpga_0_ethernet_1_PHY_Mii_data_pin LOC=M29;
Net fpga_0_ethernet_1_PHY_Mii_data_pin IOSTANDARD = LVCMOS33 | DRIVE = 8  | FAST;
```