IP PACKET HANDLING ENGINE FOR SECURE COMMUNICATION

By

NC Ayesha Shafqat

NC Sarawish Altaf

PC Mustafa Raza Naqvi

Project Supervisor

Lt. Col. Dr. Adnan Rashdi

Submitted to the Faculty of Electrical Engineering, Military College of Signals National University of Sciences and Technology, Rawalpindi in Partial Fulfillment for the Requirements of a B.E. Degree in Telecom Engineering

JUNE 2014

CERTIFICATE OF CORRECTNESS AND APPROVAL

Certified that work contained in this thesis titled "IP Packet Handling Engine for Secure Communication", carried out by Ayesha Shafqat, Sarawish Altaf and Mustafa Raza Naqvi under the supervision of Asst. Prof. Dr. Adnan Rashdi for partial fulfillment of Degree of Bachelors of Telecommunication Engineering, is correct and approved.

Approved By

Lt.Col. Dr. Adnan Rashdi

EE Department

Military College of Signals, NUST

Dated: June 2013

ABSTRACT

IP PACKET HANDLING ENGINE FOR SECURE COMMUNICATION

Information security has become a critical concern for Internet users these days. The project “IP Packet Handling Engine” has been designed and implemented to fulfill this dire need by providing a hardware architecture platform for secure IP Communications. The developed handling engine exploits the inherent parallelism feature of FPGAs and implements AES-128 Encryption algorithm in CTR mode to provide a 100% secure hardware solution for IP based communications. The Ethernet Core of FPGAs, along with the encryption codes designed in Verilog (HDL), can be programmed in any FPGA to provide a portable IP Packet Handling Engine. The engine encrypts the outgoing traffic to prevent eavesdroppers from viewing sensitive information. For incoming traffic however, the engine first verifies the source, and then decrypts the packets using the shared secret key. The IP Packet Handling Engine is an easily configurable, cost-effective, and hack immune solution for organizations intending to communicate securely and confidentially over the internet.

DECLARATION

No portion of work presented in this thesis has been submitted in support of another award or qualification either in this institution or anywhere else.

In The Name Of Allah, the Most Benevolent, the Most Merciful.

DEDICATION

This thesis is dedicated to my parents for their love, prayers, endless support and encouragement; To my supervisor, Sir Adnan Rashdi, for putting his trust in me; To my mentor, Sir Farooq Niaz, who made me realize that an idea that is developed and put into action is more important than an idea that exists only as an idea; To my friends, whose words of inspiration & encouragement taught me to never give up with my work, even during the most difficult times; And to the people of Pakistan.

-Ayesha

This thesis is dedicated to my parents and teachers especially Mr. Farooq Niaz and Mr. Intisar Rizwan-ul-Haq who have taught me that the best kind of knowledge to have is that which is learned for its own sake.

-Sarawish

Dedicated to my father for being an immense support throughout this project and my entire life at every step of the way; To my mother for all her prayers and good wishes; To my teachers, Sir Adnan Rashdi and Sir Farooq Niaz, for their guidance; And to my group mates, Ayesha and Sarawish for their unending support and kindness throughout the degree.

-Mustafa

ACKNOWLEDGEMENTS

All praises for Allah Almighty who gave us the strength to accomplish this mighty task despite numerous difficulties and hardships on our way.

We offer our utmost gratitude to our Supervisor, Lt Col Dr. Adnan Rashdi for his constant guidance and encouragement. The project wouldn't have been possible without his expert advice, unfailing patience and invaluable efforts.

We are indebted to Mr. Farooq Niaz and our instructors Mrs. Ayesha Naureen and Mr. Intisar Rizwan for taking out time through their busy schedules and helping us throughout the development of our project.

A special thanks to our parents, families and friends for discussions and exchange of ideas which were a source of encouragement during the entire project.

Lastly, we acknowledge the Lab attendants of SDR Lab who rendered their help during the whole period of our project work.

Table of Contents

LIST OF FIGURES

LIST OF ABBREVIATIONS

AES	Advance Encryption Standard
ASIC	Application Specific Integrated Circuits
BGA	Ball Grid Array
CAT	Category
CLB	Configurable Logic Block
CRC	Cyclic Redundancy Code
CSMA/CD	Carrier Sense Multiple Access/Collision Detect(ion)
CTR	Counter
DCM	Digital Clock Manager
DES	Data Encryption Standard
EMAC	Ethernet Media Access Controller
FCS	Frame Check Sequence
FF	Flip Flop
FPGA	Field Programmable Gate Array
GMII	Gigabit Media Independent Interface
HDL	Hardware Descriptive Language
IC	Integrated Circuits
IP	Internet Protocol
LAN	Local Area Network
LED	Light Emitting Diode

LLC	Logic Link Control
LUT	Look Up Table
MAC	Media Access Controller
MII	Media Independent Interface
NIST	National Institute of Standards and Technology
PC	Personal Computer
PHY	Physical Layer Protocol
RJ	Registered Jack
RTL	Resistor-Transistor Logic
UCF	User Constraint File
USB	Universal Serial Bus
VHDCI	Very High Density Cable Interconnect
VPN	Virtual Private Network
WLAN	Wireless Local Area Network
XOR	Exclusive OR

Chapter 1: Introduction

Chapter 1: Introduction

This chapter provides a brief overview of the project i.e. what exactly is the problem in existing security solutions and how these problems are addressed by the proposed project. It then explains the project objectives, limitations, specifications, deliverables and finally organization of the thesis at the end of this chapter.

1.1. Project Overview

Cyber security is the major need of professional organizations these days. The existing security solutions are not realizing efficiencies in their daily operations that would result from better application of technology. The project "IP Packet Handling Engine" aims at designing and developing hardware architecture to provide secure communication services to professional organizations. The developed handling engine exploits the inherent parallelism feature of FPGAs and implements AES-128 Encryption algorithm in CTR mode to provide a 100% secure hardware solution for IP based communications. The Ethernet Core of FPGAs, along with the encryption codes designed in Verilog (HDL), can be programmed in any FPGA to provide a portable IP Packet Handling Engine. The engine encrypts the outgoing traffic to prevent eavesdroppers from viewing sensitive information. For incoming traffic however, the engine first verifies the source, and then decrypts the packets using the shared secret key. The IP Packet Handling Engine is an easily configurable, cost-effective, and a hack immune solution to the security related problems faced by any organization that intends to securely and confidentially communicate over the internet.

1.2. Problem Statement

In an era of cyber warfare where professional organizations demand efficient communication security over the internet, there are several problems faced by the existing software security solutions:

a. They are slow due to sequential execution of algorithms.
b. They are expensive.
c. They can be hacked easily by an attack on the computer it is running within. To make them fool-proof, a lot of time and investment is needed.
d. They provide low throughput.

Professional Security Solutions on the other hand provide high throughput but some major problems with them are that:

a. They are highly expensive solutions.
b. They are not user friendly, and they cannot be customized according to the customer's and/or organization's needs.
c. The algorithms used are proprietary and thus easily known to the attacker.

The problems identified in current software based security solutions motivated us to design and develop a security device that provides confidentiality to the IP communication over the internet using an IP-less and hack immune platform that is efficient as well as relatively cost effective.

1.3. Approach

The project entitled "IP Packet Handling Engine for Secure Communication" is a customizable hardware solution to provide secure communication services to professional

organizations by securing their IP traffic communication over a network without changing the Ethernet frame format. FPGA boards will be used at the transmitter and receiver ends. Any PC/laptop working over Windows Platform will generate the Ethernet packets. These packets will be sent to the FPGA board using CAT 6 cables after designing and configuring the EMAC of the FPGA. AES-128 in CTR mode will be implemented to encrypt the Ethernet payload. The packets will then be forwarded to the network where only the device connected to user B will be able to recover the data by the help of the shared secret key. Figure 1-1 illustrates the approach adopted by IP Packet Handling Engine to address the problems faced by the professional organizations.

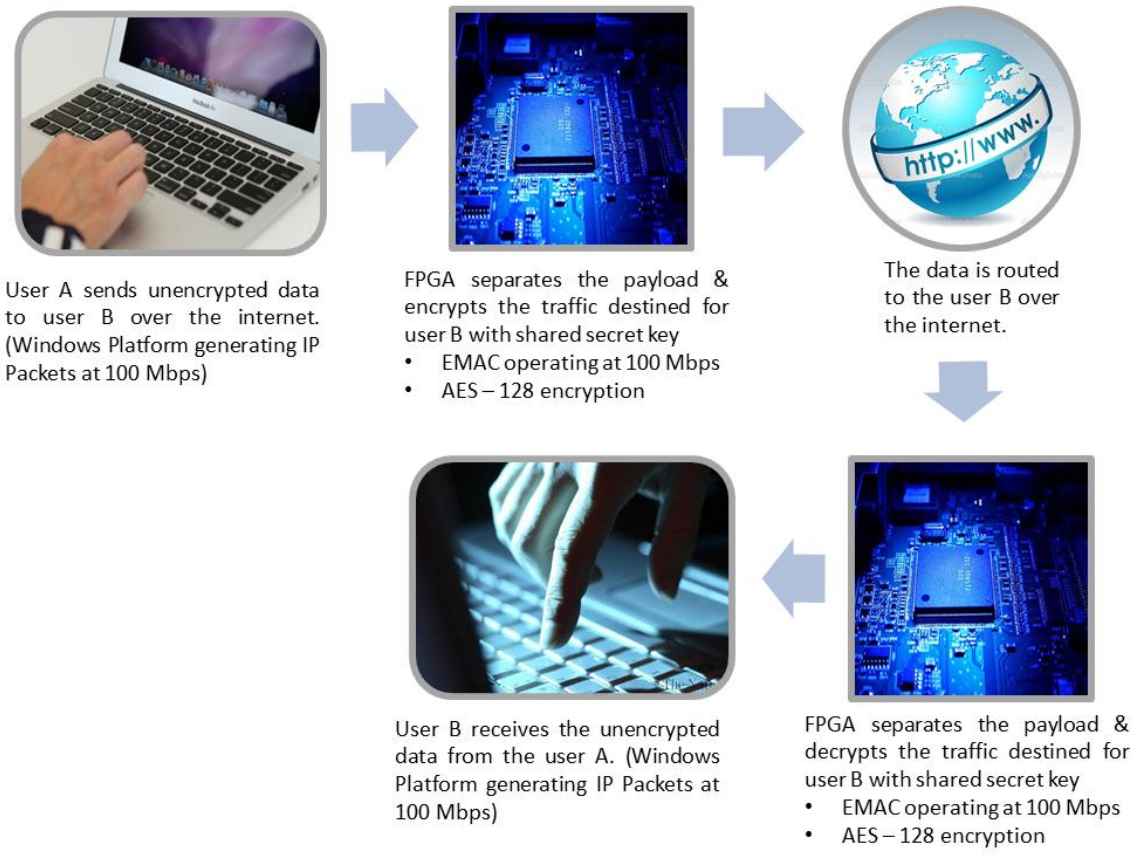


Figure 1-1 System Model

The use of FPGAs exploit inherent parallelism in any algorithm implemented, we will thus achieve better speed. Since the design is purely hardware based so it will relatively be immune to any hacking attacks. Also, the proposed design will be IP-less (works on a MAC layer) and thus invisible in a network. Currently, we aim to provide security solution for a limited number of users. Although the number of users our device can entertain can be increased according to the needs and resources available.

The device will be cost effective and hack immune. Its applications include providing secure communication to military, intelligence, banking systems and/or any organization in the corporate world looking for a network solution to securely and confidentially communicate over the internet. This robust IP Packet handling engine will require minimal to no maintenance after installation.

1.4. Objectives

Nowadays, almost every user in the corporate world requires a platform for secure IP configuration that is hack immune, high speed, cost effective and at the same time, it is easily configurable. The project targets the following user requirements:

a. 100% Hardware Platform
b. Parallel Execution
c. IP-less
d. Hack Immune
e. Custom Solution (Limited Users)
f. Easily configurable.
g. Cost Effective Solution.

1.5. Specifications

Development of the project “IP Packet Handling Engine for Secure Communication” is done using Genesys Virtex-5 FPGA development board (XC5VLX50T, FF1136). The FPGA is programmed using Verilog (HDL) and enabled for a 1 Gbps Ethernet connection with a laptop/PC generating the IP packets. AES 128 is implemented on the payload of all the received packets, which are reassembled and transmitted over the internet via FPGA. The global clock rate of the FPGA is 100 MHz and its throughput performance is near line speed. The device operates at less than 30W – 5V DC.

1.6. Deliverables of the Project

We aim to deliver a working model/ prototype of single pair of IP Encrypter and Decrypter as shown in Figure 1-2.

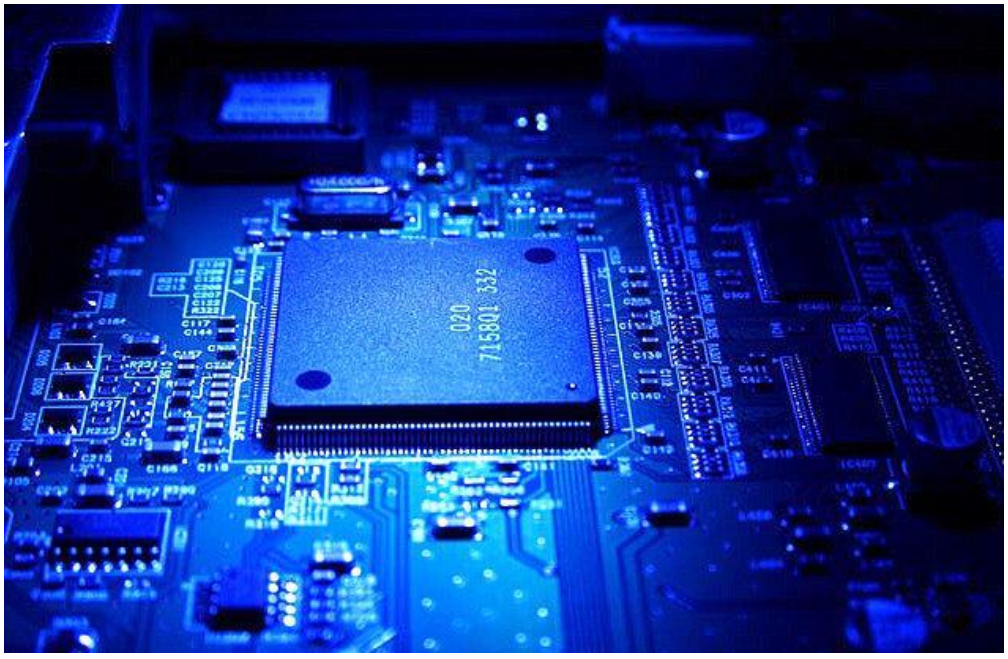


Figure 1-2 Deliverable

This hardware is programmed with all the designed Verilog codes, and is easily configurable with any user's windows platform.

1.7. Organization of Thesis

Chapter 1 It has already given a brief overview of IP Packet Handling Engine, the problems faced in existing security solutions and the solution that IP Packet Handling Engine has come up with.

Chapter 2 Deals with the literature review carried out thoroughly at the beginning of project and lists some of the current projects regarding FPGA based security which are currently being worked upon and implemented in certain parts of the world.

Chapter 3 Gives overview of IP Packet Handling Engine's design and explains in detail the hardware design and software development carried out using HDL coding.

Chapter 4 Analyses and evaluates the performance of IP Packet Handling Engine for Secure Communication on the basis of results of various tests carried out. Based on the performance, IP Packet Handling Engine's benefits are also discussed.

Chapter 5 Discusses some future enhancements of the IP Packet Handling Engine to make it a complete security device.

Chapter 6 Concludes the project research work.

At the end of the document are the appendixes giving additional relevant details about IP Packet Handling Engine with few HDL codes for the cryptographic algorithms implemented. The list of reference material used while preparing this document follows the Appendixes.

Chapter 2: Literature Review

Chapter 2: Literature Review

Technological advancements are taking place all over the world. Over the recent years, the idea of having a security device for a professional organization has gained immense popularity in developed countries.

2.1. Background

This chapter contains a brief description of FPGAs, Ethernet based networks and the encryption algorithm (AES-128). This chapter includes the basic literature overview of the topics related to our project. Each of the topics will be discussed briefly to support and strengthen the purpose of this paper. The aim of this chapter is to give an idea to readers about the hardware based security solutions without going into the details of each protocol used.

2.2. Introduction to FPGAs

FPGAs, also known as Field Programmable Gate Arrays, are prefabricated silicon chips that can be programmed electrically to implement any digital design. Hence, they are an alternative for implementation of digital logic in systems since their invention by Xilinx in 1984. FPGAs were initially designed as simple glue logic chips, but with time, they overpowered the market of application-specific integrated circuits (ASICs) and processors for signal processing and control applications. FPGAs are flexible enough to be reprogrammed to the desired application for exhibiting the required functionality after manufacturing.

FPGAs combine the best design features of ASICs and processor based systems which has driven their adoption across all industries these days. FPGAs provide hardware-timed speed and reliability. Reprogrammable silicon also has the same flexibility of software running on a processor-based system, but it is not limited by the number of processing cores available. FPGAs have a true parallel nature which makes them different from processors. There is no resource crunch so different processing operations do not have to compete for same resources. Every independent task is executed by a dedicated section of the silicon chip and can work autonomously and independently without any influence from other CLBs. As a result, the performance of one part of the application does not affect the rest of the processes.

2.3. FPGA Design and Programming

To define the behavior of the FPGA, the user provides a hardware description language (HDL) or a schematic design. The HDL form is more suited to work with large structures because it's possible to just specify them numerically rather than having to draw every piece by hand. However, schematic entry can allow for easier visualization of a design.

Then, using an electronic design automation tool, a technology-mapped net-list is generated. The net-list can then be fitted to the actual FPGA architecture using a process called place-and-route, usually performed by the FPGA developer's proprietary place-and-route software. The user will validate the map, place and route results via timing analysis, simulation, and other verification methodologies. Once the design and validation process is complete, the binary file generated (also using the FPGA developer's

proprietary software) is used to (re)configure the FPGA. This file is transferred to the FPGA via its programming port.

In a typical design flow, an FPGA application developer will simulate the design at multiple stages throughout the design process. Initially the RTL description in VHDL or Verilog is simulated by creating test benches to simulate the system and observe results. Then, after the synthesis engine has mapped the design to a net-list, the net-list is translated to a gate level description where simulation is repeated to confirm the synthesis proceeded without errors. Finally the design is laid out in the FPGA at which point propagation delays can be added and the simulation run again with these values back-annotated onto the net-list.

2.4. Ethernet Architecture

Ethernet, a physical layer local area network (LAN) technology, is nearly 30 years old. In the last three decades, it has become the most widely used LAN technology because of its speed, low cost, relative ease of installation, high robustness (i.e. high reliability) and adaptability to new technologies. This is combined with wide computer-market acceptance and the ability to support the majority of network protocols.

A key feature of Ethernet is the breaking of data into packets, also referred to as frames, which are then transmitted using the CSMA/CD (carrier sense multiple access/collision detection) protocol until they arrive at the destination without colliding with any other packets. Figure 2-1 shows the IEEE 802.3 Ethernet frame structure.

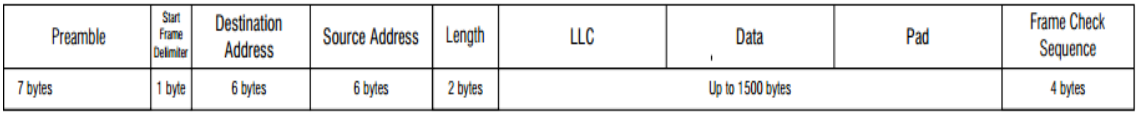


Figure 2-1 IEEE 802.3 Ethernet Frame

In reality, an Ethernet frame is made up of all the necessary parts to fit the requirements and definition of a packet. An Ethernet frame has a header (Preamble - Length), payload (LLC - Pad), and a trailer (Frame Check Sequence) that are bundled together in a specifically organized manner for transmission (see Figure 3).

2.4.1. Header

a. Preamble - Sets bit timing and signals that a frame is being sent (10 Mb/s Ethernet).

b. Start Frame Delimiter - 8-bit sequence (10101011).

c. Destination Address - 48-bit receiving hardware MAC address.

d. Source Address - 48-bit transmitting hardware address.

e. Type - Indicates protocol sending the frame.

f. Length - Indicates the length of data field.

2.4.2. Payload

a. Logical Link Control (LLC) - Governs the assembly of data at the data link (Layer 2) level.

b. Data - Payload contained in a field (between 46 bytes and just over 1500 bytes in length).

c. Pad - 0 bits added to the data field if there are fewer than 46 bytes of data in that field.

2.4.3. Trailer

a. Cyclical Redundancy Check (CRC)—Detects transmission errors.
b. Frame Check Sequence (FCS)—Detects transmission errors and provides quality of service at receiving end.

2.5. AES-128 Algorithm

The Advanced Encryption Standard (AES) is a successor of the 56-bit Data Encryption Standard (DES), which was announced by the National Institute of Standards and Technology (NIST) in November 2001, considering DES unsafe due to its short key with a length.

To determine which algorithm would follow DES, NIST called for different algorithm proposals in a sort of competition. The best of all suggestions would become the new AES. In the final round of this competition the algorithm Rijndael, named after its Belgian inventors Joan Daemen and Vincent Rijmen, won because of its security, ease of implementation, and low memory requirements.

AES has three different versions having a fixed block length of 128 bits, whereas the key length of 128, 192, or 256 bits is allowed only. The project “IP Packet Handling Engine for Secure Communication” includes the design and implementation of 128-bit AES only.

2.5.1. Basic Concept of the Algorithm

The AES algorithm consists of ten rounds of encryption, as can be seen in Figure 2-2. First the 128-bit key is expanded into eleven round keys, each of them 128 bits in size. Each round includes a transformation using the corresponding cipher key to ensure the security of the encryption.

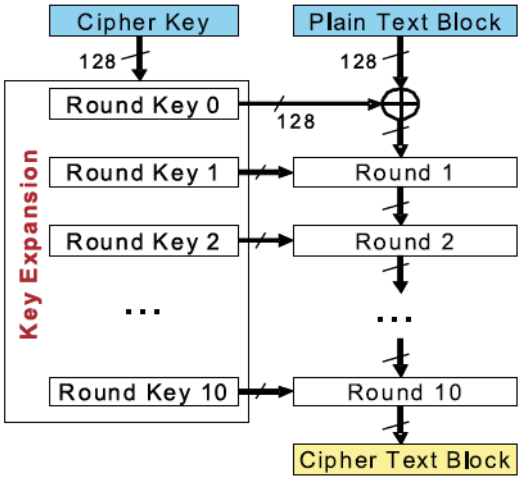


Figure 2-2 AES Algorithm Structure

After an initial round, during which the first round key is XORed to the plain text (Addroundkey operation), nine equally structured rounds follow. Each round consists of the following operations:

a. Substitute bytes

b. Shift rows

c. Mix columns

d. Add round key

The tenth round is similar to rounds one to nine, but the Mix columns step is omitted. In the following sections, these four operations are explained.

2.5.2. Structure of Key and Input Data

Both the key and the input data (also referred to as the state) are structured in a 4x4 matrix of bytes. Figure 2-3 shows how the 128-bit key and input data are distributed into the byte matrices.

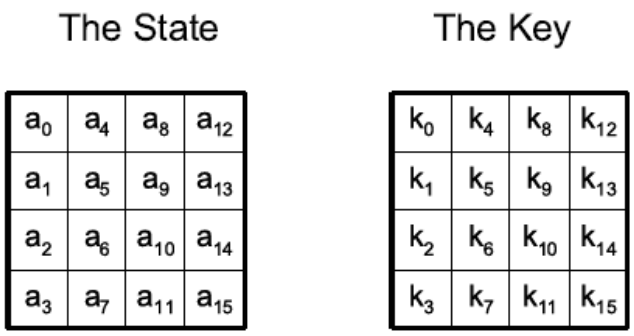


Figure 2-3 Structure of the State and Key Matrix

2.5.3. Substitute Bytes (Subbytes Operation)

The Subbytes operation is a nonlinear substitution. This is a major reason for the security of the AES. There are different ways of interpreting the Subbytes operation. In this application report, it is sufficient to consider the Subbytes step as a lookup in a table. With the help of this lookup table, the 16 bytes of the state (the input data) are substituted by the corresponding values found in the table (see Figure 2-4).

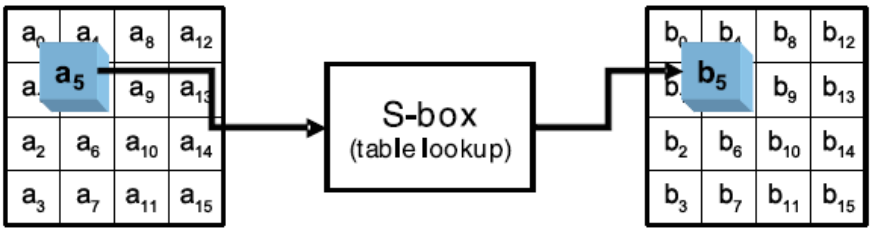


Figure 2-4 Subbytes Operation

2.5.4. Shift Rows (Shiftrows Operation)

As implied by its name, the Shiftrows operation processes different rows. A simple rotate with a different rotate width is performed. The second row of the 4x4 byte input data (the state) is shifted one byte position to the left in the matrix, the third row is shifted two byte positions to the left, and the fourth row is shifted three byte positions to the left. The first row is not changed. Figure 2-5 illustrates the working of Shiftrows.

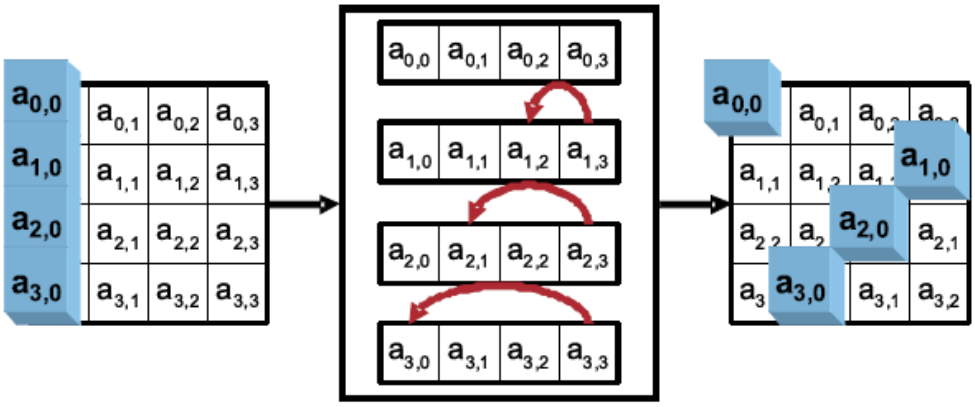


Figure 2-5 Shiftrows Operation

2.5.5. Mix Columns (Mixcolumns Operation)

Probably the most complex operation from a software implementation perspective is the Mixcolumns step. The working method of Mixcolumns can be seen in Figure 2-6.

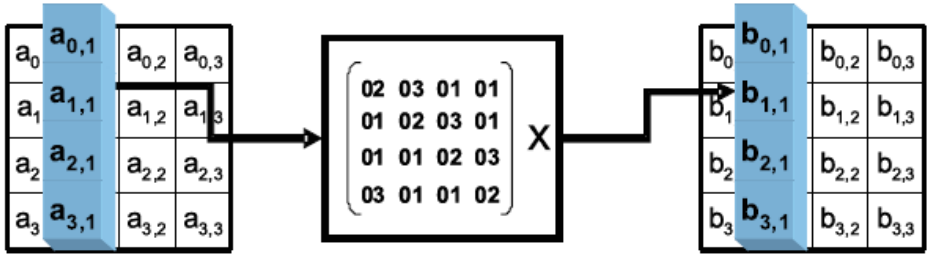


Figure 2-6 Mixcolumns Operation

Opposed to the Shiftrows operation, which works on rows in the 4x4 state matrix, the Mixcolumns operation processes columns. In principle, only a matrix multiplication needs to be executed. To make this operation reversible, the usual addition and multiplication are not used. In AES, Galois field operations are used. This thesis does not go into the mathematical details, it is only important to know that in a Galois field, an addition corresponds to an XOR and a multiplication to a more complex equivalent. The fact that there are many instances of 01 in the multiplication matrix of the Mixcolumns operation makes this step easily computable.

2.5.6. Add Round Key (Addroundkey Operation)

The Addroundkey operation is simple. The corresponding bytes of the input data and the expanded key are XORed (see Figure 2-7).

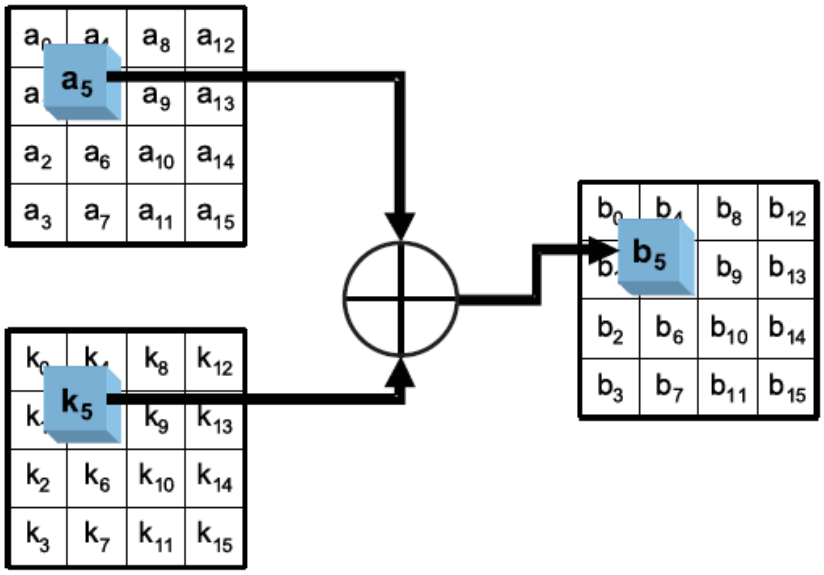


Figure 2-7 Addroundkey Operation

2.5.7. Key Expansion (Keyexpansion Operation)

As previously mentioned, Keyexpansion refers to the process in which the 128 bits of the original key are expanded into eleven 128-bit round keys.

To compute round key (n+1) from round key (n) these steps are performed:

a. Compute the new first column of the next round key as shown in Figure 2-8:

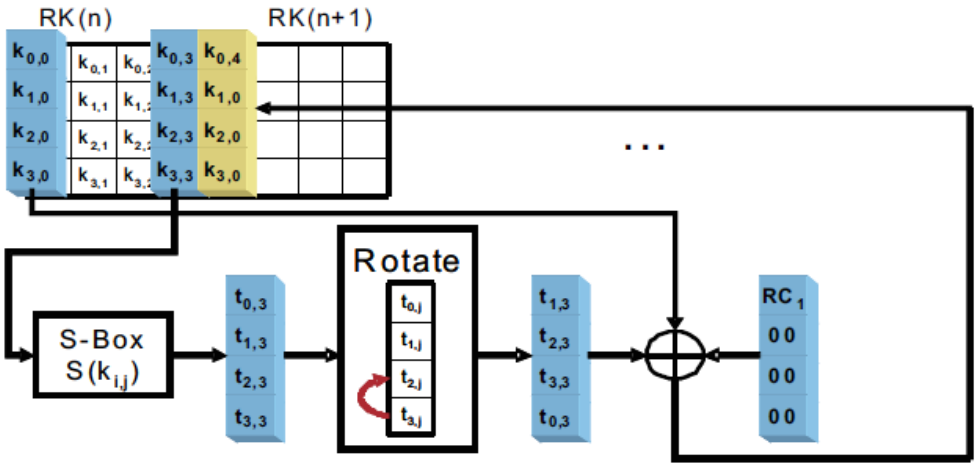


Figure 2-8 Expanding First Column of Next Round Key

First all the bytes of the old fourth column have to be substituted using the Subbytes operation. These four bytes are shifted vertically by one byte position and then XORed to the old first column. The result of these operations is the new first column.

b. Columns 2 to 4 of the new round key are calculated as shown:

   i. [new second column] = [new first column] XOR [old second column]

   ii. [new third column] = [new second column] XOR [old third column]

   iii. [new fourth column] = [new third column] XOR [old fourth column]

Figure 2-9 illustrates the calculation of columns 2-4 of the new round key.

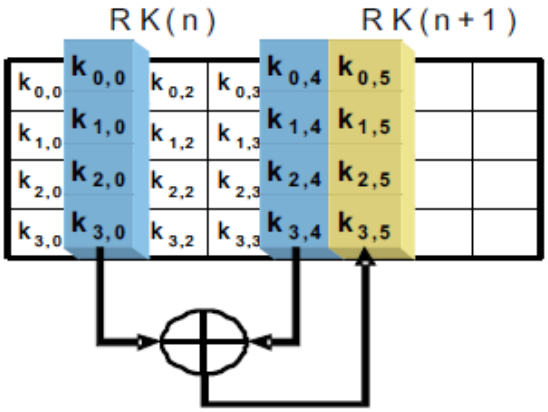


Figure 2-9 Expanding Other Columns of Next Round Key

AES is a popular encryption standard approved by the government and supported by all VPN vendors. AES today is also used in removable media such as USB's and external hard drives. It is effective in both hardware and software and uses less memory than most other symmetric algorithms. Simply put, you can protect your data on your USB memory stick using encryption software running the AES algorithm.

2.6. Existing Literature Related To The Topic:

Currently we can find a lot of literature related to our thesis topic. "FPGA Implementation of AES Encryption and Decryption" by Sounak Samanta of Sardar Vallabhbhai National Institute of Technology, Surat[1] is one such paper. It presents a high speed, fully pipelined FPGA implementation of AES Encryption and Decryption. The focal approach of this design on hardware platform was to attain speed up i.e. high throughput (No. of block processed per second) and at the same time silicon area optimization.

"AES on FPGA from the fastest to the smallest" by University of Sheffield[2] is a paper on two new FPGA designs for the AES. The first is believed to be the fastest, achieving 25 Gbps throughput using a Xilinx Spartan-III (XC3S2000) device. The second is believed to be the smallest and fits into a Xilinx Spartan-II (XC2S15) device, only requiring two block memories and 124 slices to achieve a throughput of 2.2 Mbps. These designs show the extremes of what is possible and have radically different applications from high performance e-commerce IPsec servers to low power mobile and home applications. The research objective of this design was to explore the design space associated with the AES algorithm and in particular its FPGA hardware implementation in terms of speed and area.

"Implementation of the AES-128 on Virtex-5 FPGAs"[3] is another such paper that presents an updated implementation of AES on Xilinx Virtex 5 FPGAs. This design specifically focuses on producing encryption and decryption architectures that perfectly fit with Digital Cinema Initiative Specifications.

"Single-Chip FPGA Implementation of the Advanced Encryption Standard Algorithm" by Maire McLoone and John V. McCanny[4] describes a fully pipelined single-chip AES design which runs at a data rate of 7 Gbps on a Xilinx Virtex-E XCV812E-8-BG-560 FPGA device and proves to be one of the fastest FPGA AES implementations currently available.

"A Design and Implementation of Real Time AES-128 on Real Time Operating System for Multiple FPGA Communication" by University of Calcutta [5] demonstrate the design

and implementation of a 128-bit AES on Xilinx Spartan- 3E (XC3S500E-FG320) device. The system designed is optimized in terms of execution speed and hardware utilization.

Algotronix[6] supplies two families of advanced AES IP cores. These IP cores can be targeted at FPGAs from Xilinx, Altera and Actel as well as ASIC or even CPLD implementations.

2.7. Problem Formulation of the Topic:

From the current work available it can be seen that various examples of AES implementations are already available, making use of different FPGAs. Almost all make use of FPGAs' flexibility and architecture to design systems with increased speed and optimized silicon area usage. But most of the work available does not focus on AES implementation on any particular kind of data traffic and thus does not address the issues that come with that. They simply focus on static inputs and apply AES over them. But there has been some application oriented work that involved AES implementation on images, digital cinema, and voice over IP etc. The project "IP Packet Handling Engine for Secure Communication" takes inspiration from all of this work, and aims to design a hardware architecture that similarly exploits the flexibility of FPGAs to provide a faster AES based security solution which utilizes optimum hardware resources. Along with this, it particularly works to provide a secure networking solution. Since internet is the most common form of communication today, a system is designed that secures traffic over the internet using AES, so this design is customized for networking applications.

Chapter 3: Design & Development

Chapter 3: Design & Development

The project “IP Packet Handing Engine for Secure Communication” uses Genesys™ Virtex-5 FPGA Development Board, manufactured by Digilent, which uses the Xilinx Virtex-5 LX50T in 1136-pin BGA Package, to connect to any PC/Laptop.

The connection is established via the FPGA’s Marvell Alaska Tri-Mode Ethernet PHY (88E1111) which comes paired with a Halo HFJ11-1G01E RJ45 connector, and CAT 6 cable. The Ethernet PHY of the FPGA board is enabled using the Xilinx Core Generator tool. This allows the FPGA board to receive and transmit Ethernet packets from and to any PC/Laptop or network at a speed of 1 Gbps. Once the packets arrive in the FPGA board the HDL logic programmed in the FPGA enables it to separate the header and payload of each Ethernet packet received, implement AES-128 encryption algorithm on the payload, replace the plaintext payload of each Ethernet packet with the cipher-text payload and transmit the secure Ethernet traffic to the network via a CAT 6 cable and a router.

On the receiving end a similar FPGA device receives the secure Ethernet traffic from the network, separates its headers and payloads, decrypts payload, replaces the cipher-text payload with the original plaintext payload, and transmits the original traffic to the intended receiver’s PC/Laptop. Also the FPGA board available had only single Ethernet PHY, which meant that at one time it can be either connected to a PC or the network. Since both the connections were a basic requirement of the project, a solution was devised which involved the use of another FPGA board for its Ethernet PHY (on both the

encrypter and decrypter end), and the two FPGA boards being connected together via the Very High Density Cable Interconnect (VHDCI) cable.

Figure 3-1 below shows the system model of the project:

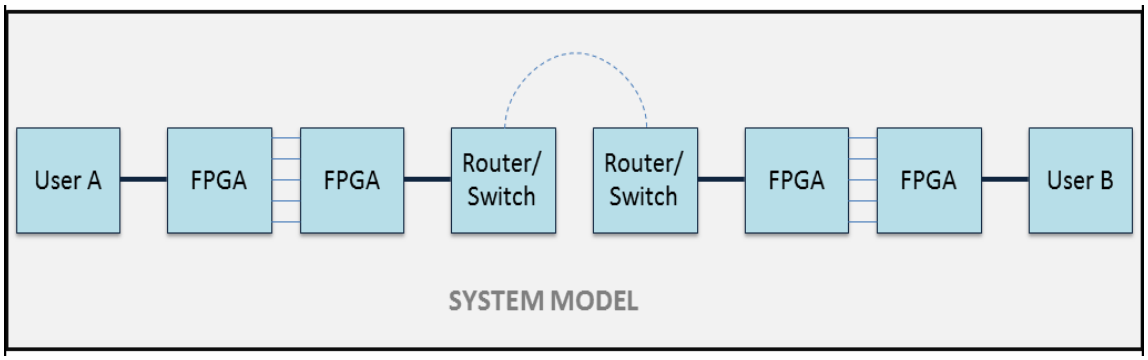


Figure 3-1 System Model

The various hardware, software requirements and modules that constitute the design and development of "IP Packet Handling Engine for Secure Communication" are as shown in Figure 3-2.

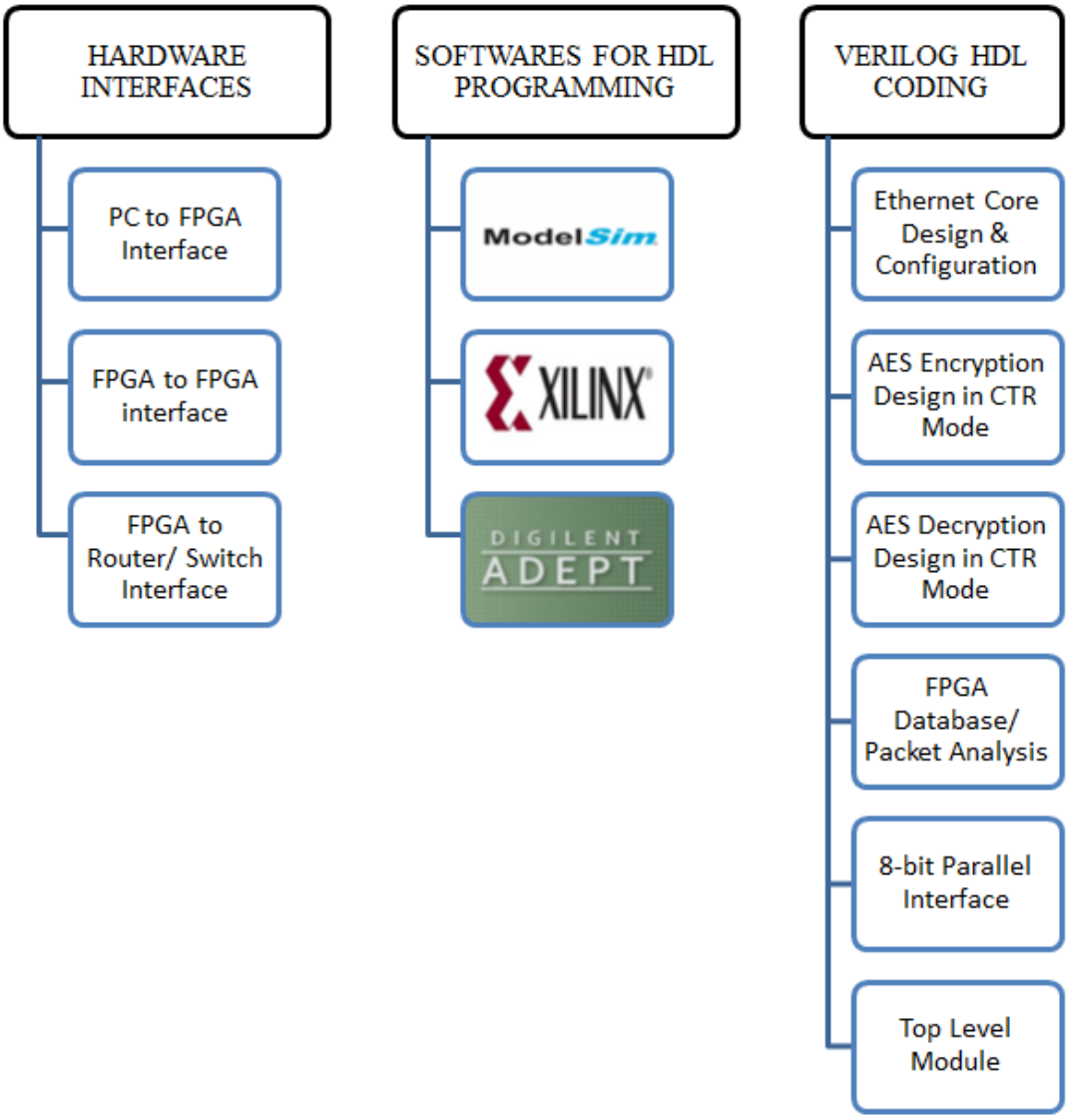


Figure 3-2 Design Requirements

3.1. Hardware Interfacing

The project involves the development of three hardware interfaces that are replicated to both sides of the network:

3.1.1. PC to FPGA Interface

Ethernet PHY of FPGA board is enabled to transmit & receive data to and from the PC/Laptop. The steps involved are:

a. Design of an Embedded Tri-Mode Ethernet MAC Wrapper for Virtex 5 FPGA in Verilog HDL using the Xilinx Core Generator tool.

b. Since the wrapper designed through the software is for standard Virtex-5 FPGA board (ML505), therefore configure it according to the FPGA board available, i.e. Virtex 5 (XC5VLX50T, FF1136), by modifying the User Constraint File (UCF) generated as part of the wrapper.

c. Import the core as a Xilinx project in Xilinx Project Navigator and generate .bit file.

d. Program the .bit file on the FPGA using Digilent Adept software to enable the Ethernet MAC to send and receive the IP traffic at a speed of 1 Gbps.

e. Connect the PC's Ethernet PHY to that of the FPGA board via the CAT 6 cable.

Figure 3-3 below shows the PC to FPGA interface.

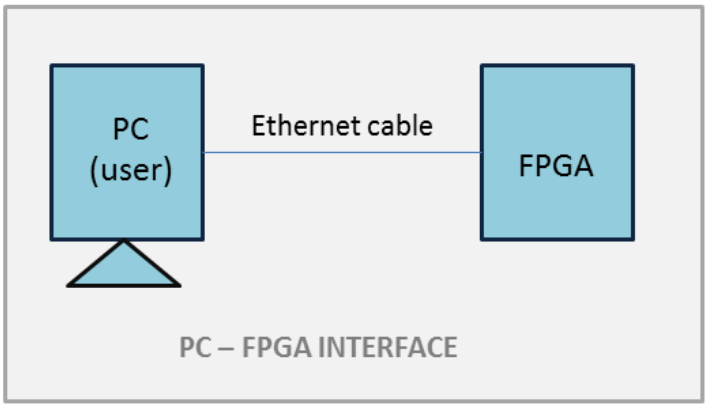


Figure 3-3 PC to FPGA Interface

3.1.2. FPGA to FPGA Interface

The encrypted Ethernet traffic generated by the AES algorithm is communicated between the expansion slots of the FPGA boards so that the data can be taken to the Ethernet port of the second FPGA and then to the network.

Additional hardware required for this FPGA to FPGA interface is the "VHDCI male to male connector cable", as shown by the Figure 3-4 below:

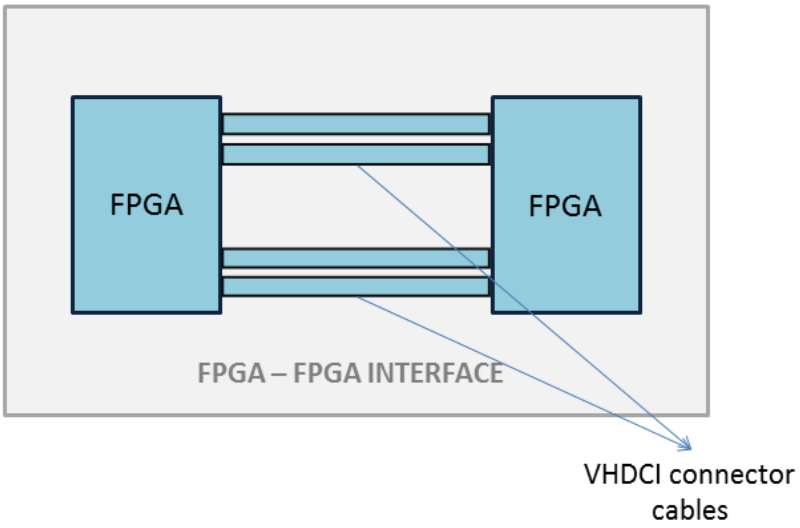


Figure 3-4 FPGA to FPGA Interface

3.1.3. FPGA to Router Interface

A wired Ethernet connection is established through a CAT-6 cable between the RJ-45 connector of the FPGA and the RJ-45 connector of the router or the switch available, as shown in Figure 3-5 below:

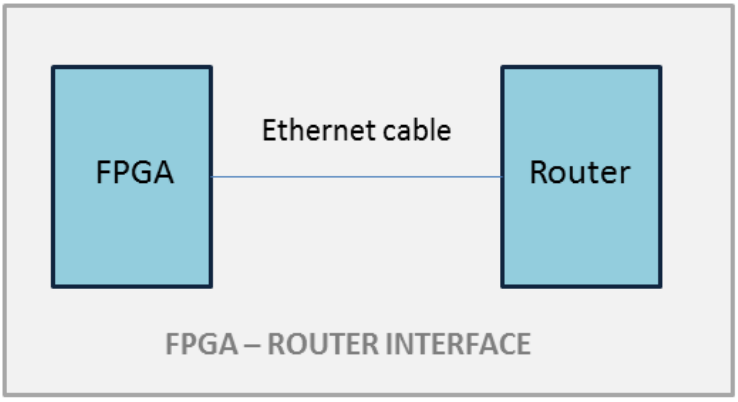


Figure 3-5 FPGA to Router Interface

3.2. Hardware Designing using HDL

3.2.1. Ethernet Core Design and Configuration

To enable the Ethernet ports of the FPGA boards to transmit and receive Ethernet traffic from the network, we used Xilinx's Core Generator tool to design an Embedded Tri-mode Ethernet MAC Wrapper for Virtex-5 FPGA. Since the wrapper designed through the software was for standard Virtex-5 FPGA board (ML505), therefore it had to be configured according to the FPGA board available to us, i.e. Virtex 5 (XC5VLX50T, FF1136) to enable its Ethernet ports. For this purpose, we changed the entire UCF of the project for our development board. Help was taken from xilinx documents UG 194 and UG 340 for generating the EMACs.

Problems were faced in modifying the core generated for ML-505 to work on our custom FPGA board. Changing the UCF of a 50 trillion gate device with 1136 pin package was itself a big project.

3.2.2. AES Encryption in CTR-Mode

The aim of the project "IP Packet Handling Engine for Secure Communication" was to provide one of the important security feature "confidentiality" using AES in CTR (Counter) mode. The algorithm was designed in pipeline form to exploit the device's inherent property of parallelization. Modules of forward S-Box, substitute bytes, shift rows, mix columns, add round key, round key generation and expansion etc. were designed in Verilog (HDL) for AES encryption. The top module of AES encryption was then linked with a counter module that takes a 128-bit initialization vector (96 bit nonce and 32 bit counter) as an input and after applying AES encryption on it, it XORs it with the plain text input. The software simulations of the test bench show the module testing of the generated bit stream. After an initial delay of 162ns, the cipher text stream is generated in the same order as the plain text stream was given to the module. The console shows the counter input generated at each plain text, which goes to the encryption block to generate a temporary cipher text. This temporary cipher text is then XORed with the plain text to produce the final output i-e the cipher text.

All the Verilog coding was done in Xilinx's Project Navigator 13.4 and simulated using ISim.

3.2.3. AES Decryption in CTR Mode:

Since the algorithm designed was in CTR mode, the decryption module consists of the encryption module taking the counter input and XORs its results with the cipher text to obtain the plain text. Both encryption and decryption module use the same cipher key since the algorithm is symmetric. But it is generically designed to take any key as its input, any any nonce value for its initialization vector.

Problems were faced in HDL coding during synchronization of several parallel processes going on inside the FPGA at the same time. Device utilization was too less with efficient bit stream results over encryption and decryption so no bottlenecks were faced as far as the resources are concerned.

3.2.4. FPGA Database/ Packet Analysis:

Databases at both user ends are maintained for data encryption and decryption policy for the other user, along with the shared secret key to be used for the process.

At user A, the FPGA database maintained performs two main functions:

a. If data is for user B, then encrypts with AES using the shared secret key.
b. If data is from user B, then decrypts with AES using the shared secret key.

At user B, the FPGA database maintained performs two main functions:

a. If data is for user A, then encrypts with AES using the shared secret key.
b. If data is from user A, then decrypts with AES using the shared secret key.

Figure 3-6 describes the general decision making on part of the FPGAs at user A and B for this secure communication:

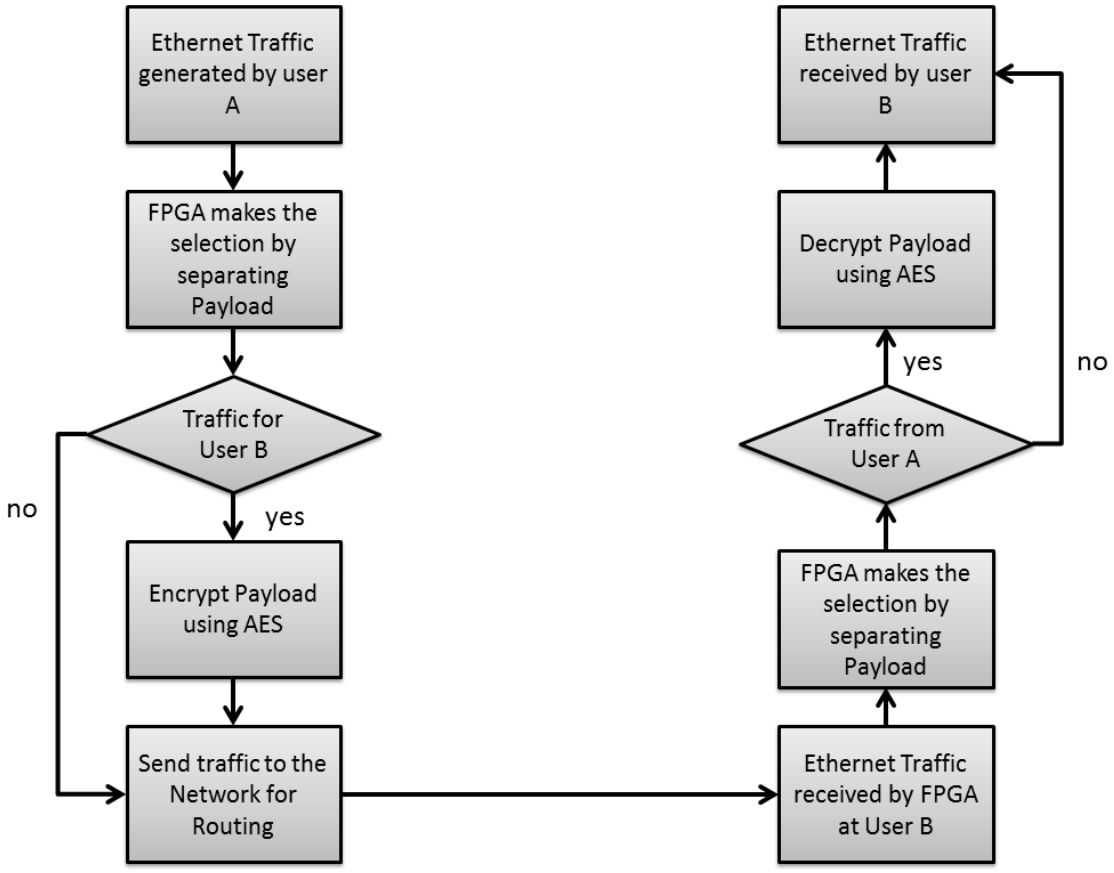


Figure 3-6 Flow Chart for Packet Analysis

3.2.5. 8-bit Parallel Interface

The project "IP Packet Handling Engine for Secure Communication" requires an FPGA device having two Ethernet Ports at user end so that the device can be connected with one port to the user and with the other port to the network. The FPGA development boards available only had 1 Ethernet Port so an 8-bit parallel interface was used to join two FPGA boards. This was done using the expansion connectors available on the Genesys Virtex-5 development boards. A 68-pin VHDCI male to male connector cable was used to join the 68-pin VHDCI female expansion headers available on both boards.

3.2.6. Top Level Module

A top level module was then created which instantiated all the designed modules of the project including the Ethernet Core, AES encryption and decryption module, Ethernet Packet Analysis and the 8-bit interface. The design was synthesized and its UCF was generated to place and route the design elements accordingly. The .bit file generated at the end was programmed on the FPGA which enabled it to perform the functioning of a security device offering confidentiality to its user.

3.3. Hardware Requirements:

Following are the hardware requirements for the project "IP Packet Handling Engine for Secure Communication":

a. Genesys Virtex-5 FPGA Development Board (XC5VLX50T FF1136).
b. CAT 6 cables.
c. VHDCI cables.

3.4. Software Requirements:

Following are the software requirements for the project "IP Packet Handling Engine for Secure Communication":

a. ISE Xilinx Project Navigator.
b. Xilinx Core Generator.
c. ISim.
d. ModelSim.
e. Digilent Adept.
f. Wireshark

3.5. Benefits of our Proposed Project:

The main goal of the project “IP Packet Handling Engine for Secure Communication” is to create a stand-alone device for providing secure communication over the internet. Hardware based solutions vastly support parallelism and require highly deliberate programming which is at odds with the pressing need to perform diversely increasing, sophisticated forms of analysis. The device is cost effective, hack immune and provides secure communication to

a. military,
b. intelligence,
c. banking systems and/or
d. Any organization in the corporate world

Chapter 4: Analysis & Evaluation

6.1 Ethernet Core Design, Configuration & Testing

6.2 AES Encryption over Virtex-5 FPGA

6.3 AES Decryption over Virtex-5 FPGA

6.4 Ethernet Packet Analysis, Payload Encryption & Reassembly

Chapter 4: Analysis and Evaluation

The analysis and evaluation of the five major project tasks mentioned above is done in this section of the report.

4.1. Ethernet Core Design, Configuration & Testing

A Virtex-5 Embedded Tri-mode Ethernet MAC Wrapper was generated using Xilinx's CORE Generator tool. This wrapper file instantiates the full Virtex-5 Ethernet MAC (EMAC) primitive. This wrapper basically enables the MAC functionality of our Virtex 5 FPGA device. The wrapper was generated for PHY interface of GMII and speed of 1 Gbps. After generation of this wrapper, a top module was written to call the address swap module generated as an example design. For 1Gbps speed, a clock of 125MHz was required, so we generated a clock module using DCM from FPGA's internal 100MHz clock. This clock module was instantiated in the top module along with the address swap module.

For software testing, the demonstration test bench in Verilog was used to exercise the example design and the core as shown in Figure 4-1. The four frames that are pushed into the EMAC0 receiver interface at 1 Gbps are minimum length frame, type frame, errored frame and padded frame. The frames received at the transmitter of each Ethernet MAC interface are checked against the stimulus frames to ensure that the data is the same. The monitor process takes into account the source/destination address field and FCS modifications resulting from the address swap module.

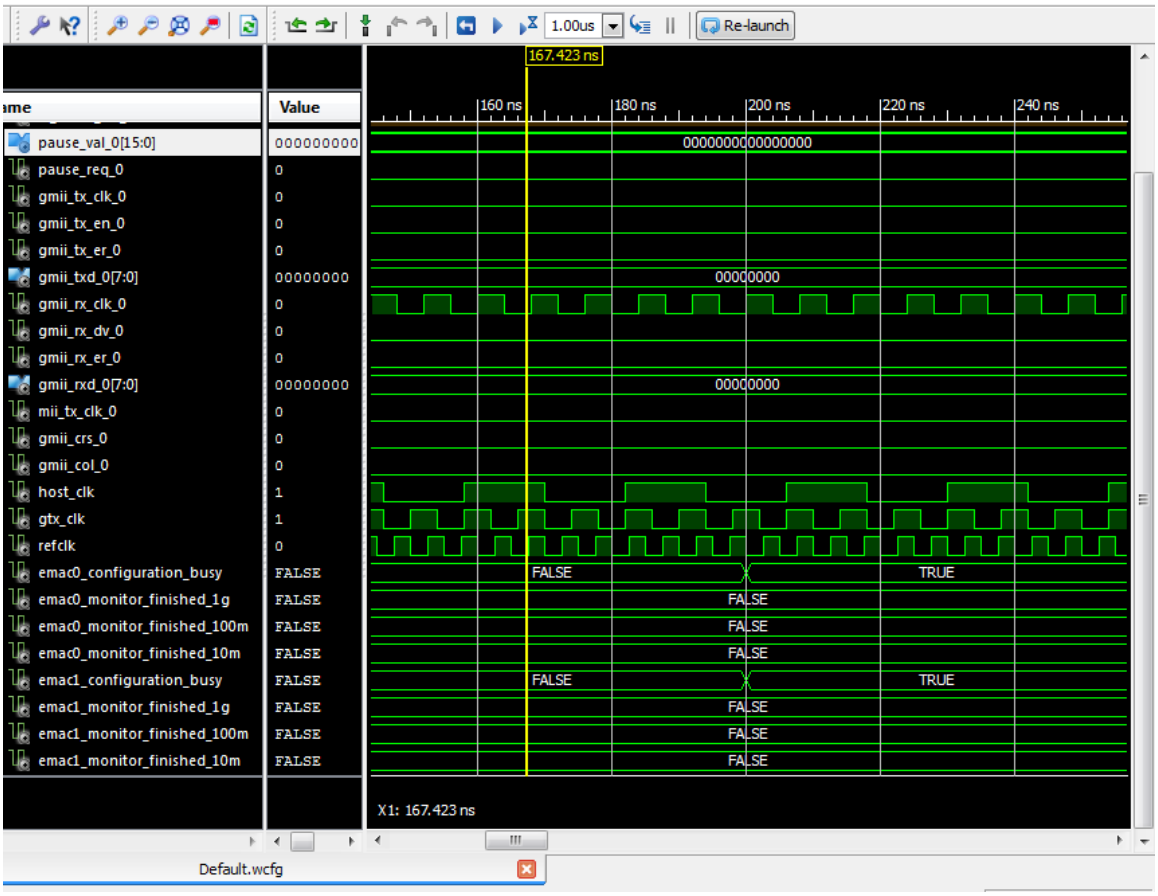
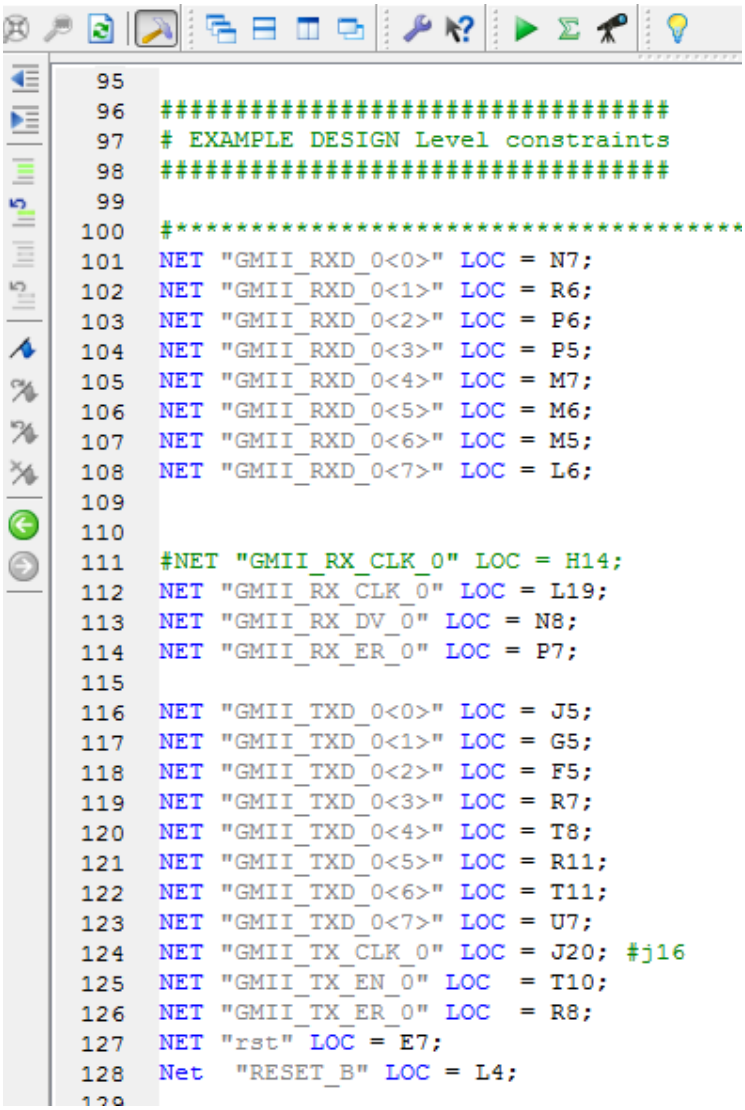


Figure 4-1 Software Testing of Ethernet Core

For hardware testing of the FPGA device's Ethernet core, a built-in example design of address swap module was used. This module makes the Ethernet MAC, of the FPGA device, swap the source and destination MAC addresses of any Ethernet packet it receives, sending it back to the transmitting device as shown in the figure below. For this, the UCF of the design was changed according to the pin configuration given in the Genesys Reference Manual, as shown in the Figure 4-2:



```
95
96  ###################################
97  # EXAMPLE DESIGN Level constraints
98  ###################################
99
100 #*************************************
101 NET "GMII_RXD_0<0>" LOC = N7;
102 NET "GMII_RXD_0<1>" LOC = R6;
103 NET "GMII_RXD_0<2>" LOC = P6;
104 NET "GMII_RXD_0<3>" LOC = P5;
105 NET "GMII_RXD_0<4>" LOC = M7;
106 NET "GMII_RXD_0<5>" LOC = M6;
107 NET "GMII_RXD_0<6>" LOC = M5;
108 NET "GMII_RXD_0<7>" LOC = L6;
109
110
111 #NET "GMII_RX_CLK_0" LOC = H14;
112 NET "GMII_RX_CLK_0" LOC = L19;
113 NET "GMII_RX_DV_0" LOC = N8;
114 NET "GMII_RX_ER_0" LOC = P7;
115
116 NET "GMII_TXD_0<0>" LOC = J5;
117 NET "GMII_TXD_0<1>" LOC = G5;
118 NET "GMII_TXD_0<2>" LOC = F5;
119 NET "GMII_TXD_0<3>" LOC = R7;
120 NET "GMII_TXD_0<4>" LOC = T8;
121 NET "GMII_TXD_0<5>" LOC = R11;
122 NET "GMII_TXD_0<6>" LOC = T11;
123 NET "GMII_TXD_0<7>" LOC = U7;
124 NET "GMII_TX_CLK_0" LOC = J20; #j16
125 NET "GMII_TX_EN_0" LOC  = T10;
126 NET "GMII_TX_ER_0" LOC  = R8;
127 NET "rst" LOC = E7;
128 Net  "RESET_B" LOC = L4;
```

Figure 4-2 UCF for Ethernet Core

The design was then simulated, implemented and its generated .bit file was programmed on the FPGA kit. The kit was connected to the PC via CAT 6 cable. After programming the core, a ping command was given in the command prompt and Ethernet interface capture was started on wireshark. The results obtained in the Figure 4-3 clearly show the MAC address swapping of all the packets returned by the FPGA.

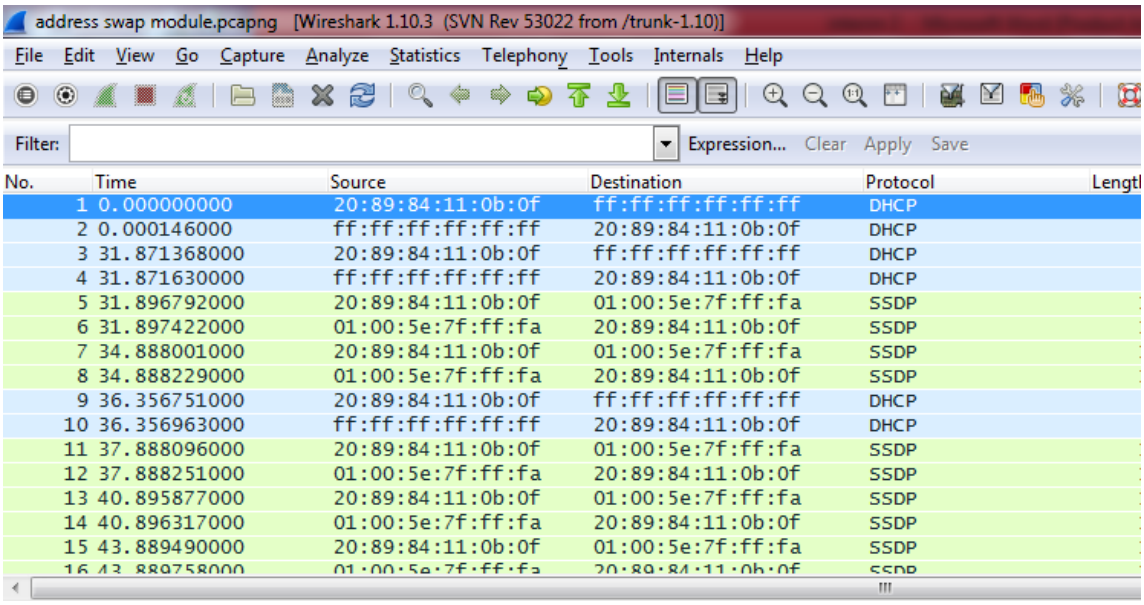


Figure 4-3 EMAC's Address Swap Module monitored in Wireshark

The address swap functionality is clearly being indicated by Wireshark in Figure 4-3 above. This shows that the Ethernet MAC of our FPGA device has been successfully enabled and that the FPGA Ethernet port can now successfully transmit and receive Ethernet packets.

4.2. AES Encryption over Virtex-5 FPGA

Figure 4-4 shows the design of AES encryption algorithm in Xilinx's Project Navigator. The algorithm designed was synthesized, implemented and then its programming file was generated for hardware testing.

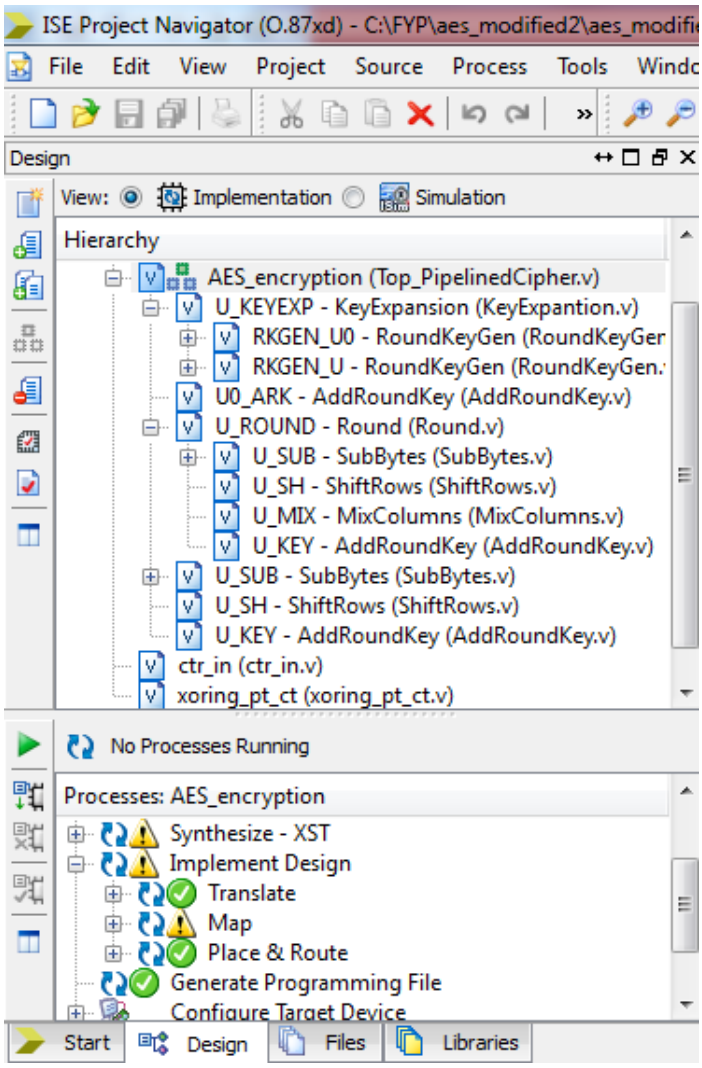


Figure 4-4 AES Encryption Design

We used two design checks. First, the software testing and then the hardware testing. For software testing, we created a test bench with inputs defined in the code. It was then simulated in Xilinx's simulator, ISim. Screen shots of the testing results are shown in Figure 4-5:

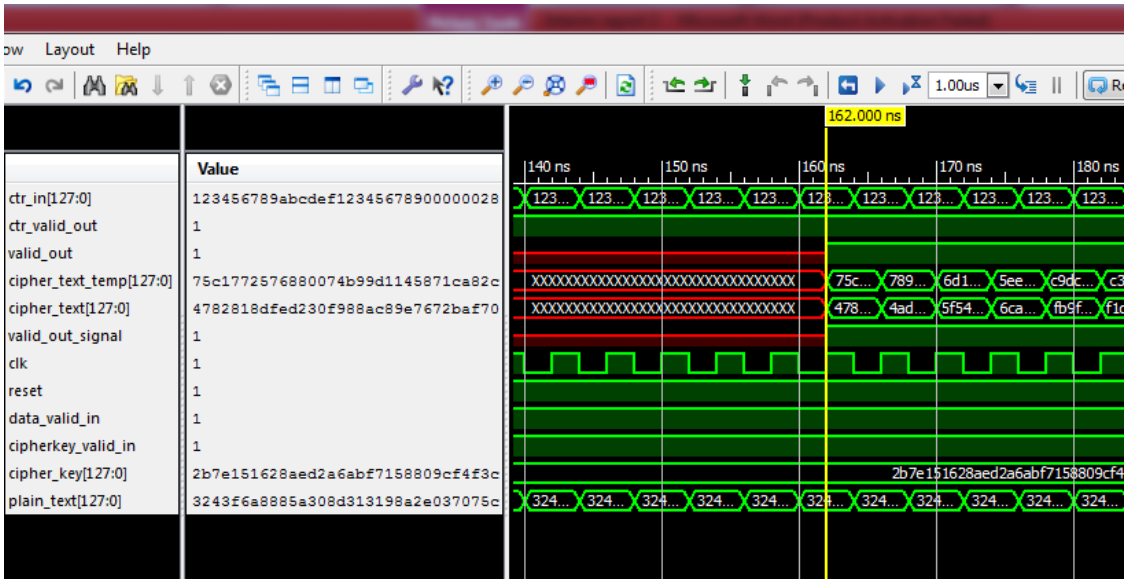


Figure 4-5 AES Encryption Software Simulation starting Delay

The simulator shows the timing diagram which represents a processing delay of 162ns at the start (Figure 4-5) after which the output data stream is generated continuously at the clock rate defined (Figure 4-6).

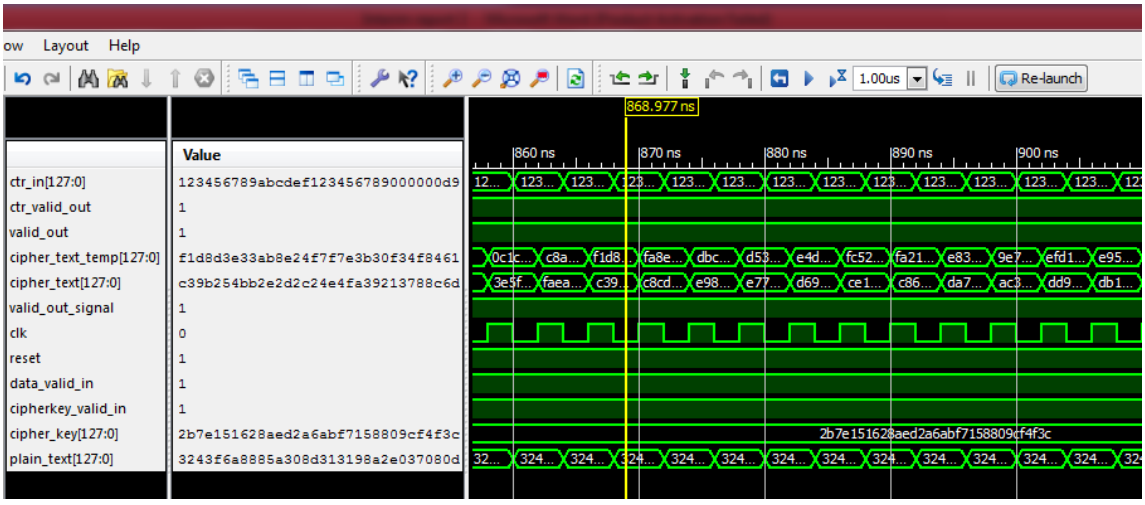


Figure 4-6 AES Data Stream Encryption Simulation

The results of all the inputs (plain text and cipher key) and outputs (counter input and cipher text) monitored are shown on the console below (Figure 4-7):

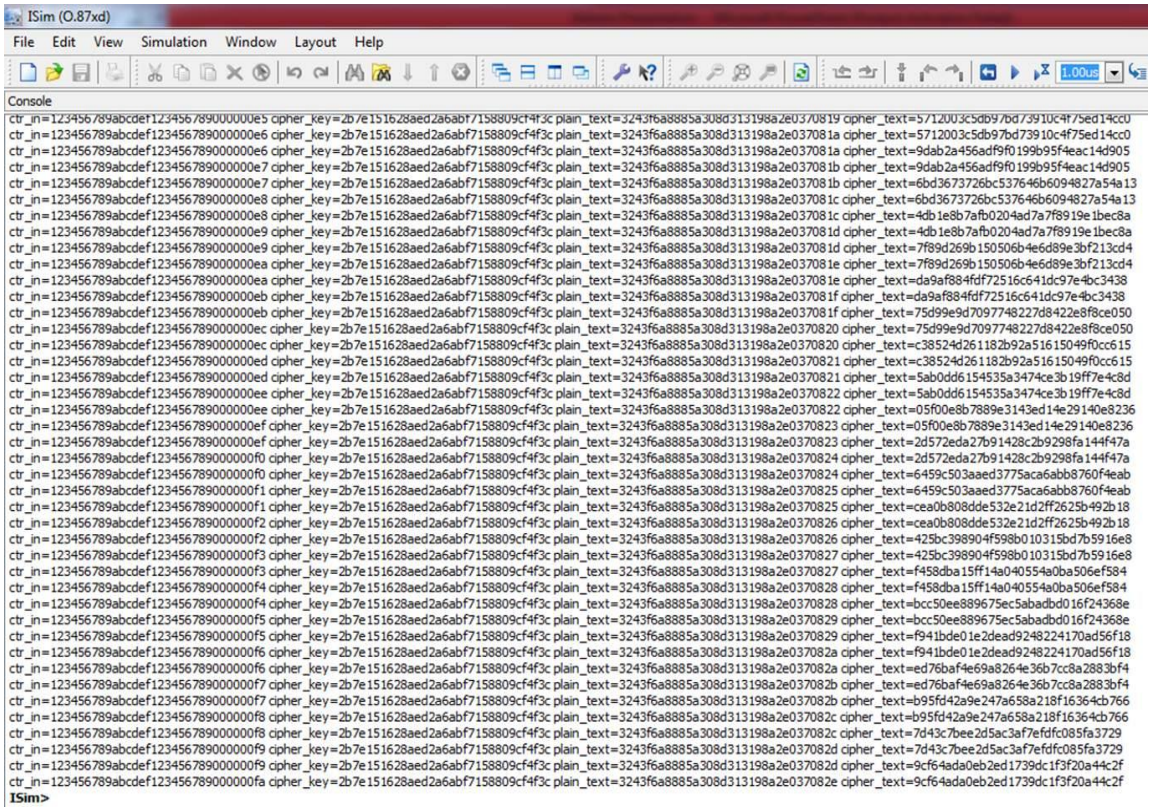


Figure 4-7 Simulation Console Results for Data Stream Encryption

AES CTR Mode Analysis:

1 Clock cycle = 4ns (250 Mbps clock)

Processing delay of 128 bits = 162ns

No. of cycles for encryption = 40.5

Speed = 790 Mbps

For hardware testing, the design (excluding the test bench) was synthesized. The design summary generated showed a device utilization of only 34% (i-e the number of fully used LUT-FF pairs) as shown in Figure 4-8.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	10,769	28,800	37%
Number used as Flip Flops	10,769		
Number of Slice LUTs	10,683	28,800	37%
Number used as logic	10,683	28,800	37%
Number using O6 output only	10,683		
Number of occupied Slices	4,092	7,200	56%
Number of LUT Flip Flop pairs used	16,001		
Number with an unused Flip Flop	5,232	16,001	32%
Number with an unused LUT	5,318	16,001	33%
Number of fully used LUT-FF pairs	5,451	16,001	34%
Number of unique control sets	82		
Number of slice register sites lost to control set restrictions	3	28,800	1%
Number of bonded IOBs	389	480	81%
Number of BUFG/BUFGCTRLs	2	32	6%
Number used as BUFGs	2		
Average Fanout of Non-Clock Nets	7.36		

Center | Design Summary (Programming File Generated) | AES_encryption (RTL1)

Figure 4-8 AES Encryption Design Measurements

The synthesis process also resulted in the formation of RTL schematics of the generated design, part of its top view is shown in Figure 4-9.

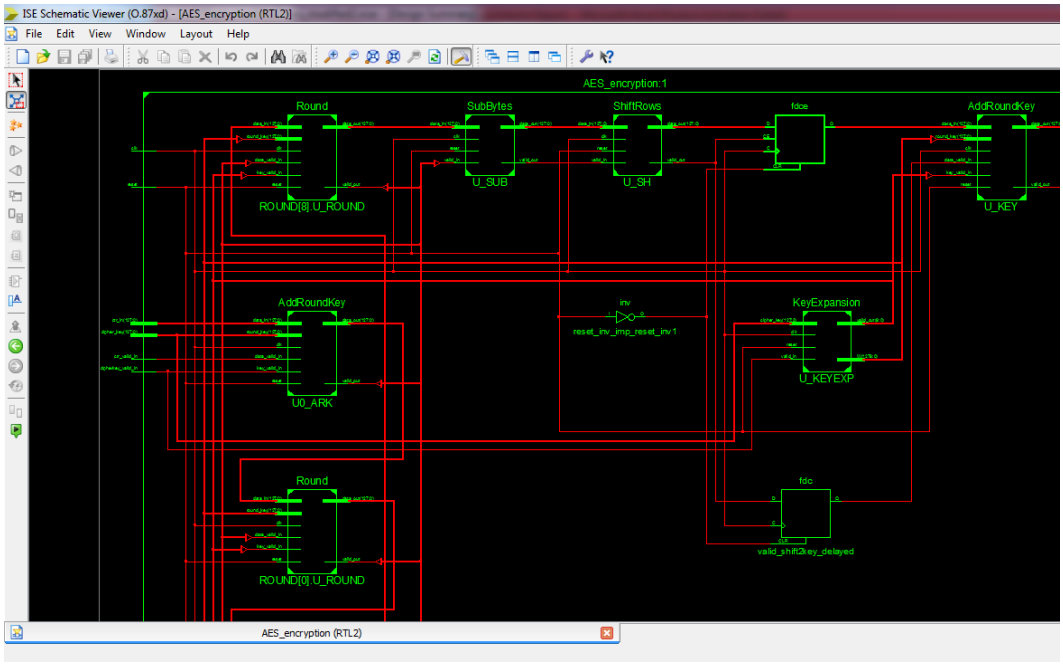


Figure 4-9 RTL Schematics for AES Encryption

After design implementation, a .bit file was generated in the last step to link the code with the hardware. This programming file was ported on the FPGA using the Digilent Adept software via Digilent USB Port available on the FPGA. Results are shown in F igure 4-10.

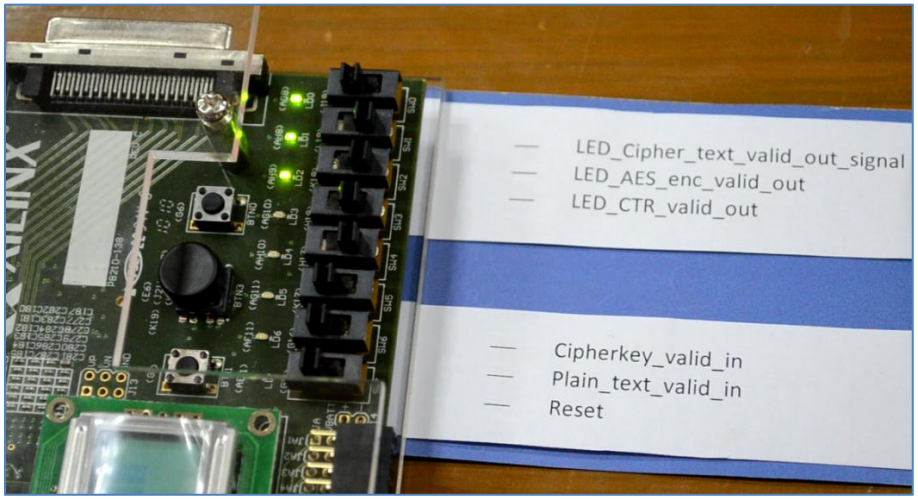


Figure 4-10 AES Encryption Hardware Testing

Since the code was designed with asynchronous active low reset, so no encryption can take place until unless we turn on the reset signal. As soon as we turn on the valid signal for the plain text, we see that the counter module generates the counter input, turning on the counter valid output LED. And as soon we turn on the valid signal for cipher key, AES encryption LED goes high showing that the encryption block was executed and correspondingly the Cipher text valid out LED goes high showing that the final XORing between the temporary cipher text and the plain text has taken place.

4.3. AES Decryption over Virtex-5 FPGA

Figure 4-11 shows the design of AES decryption algorithm in Xilinx's Project Navigator. Both hardware and software testing were performed for AES 128 Decryption design too.

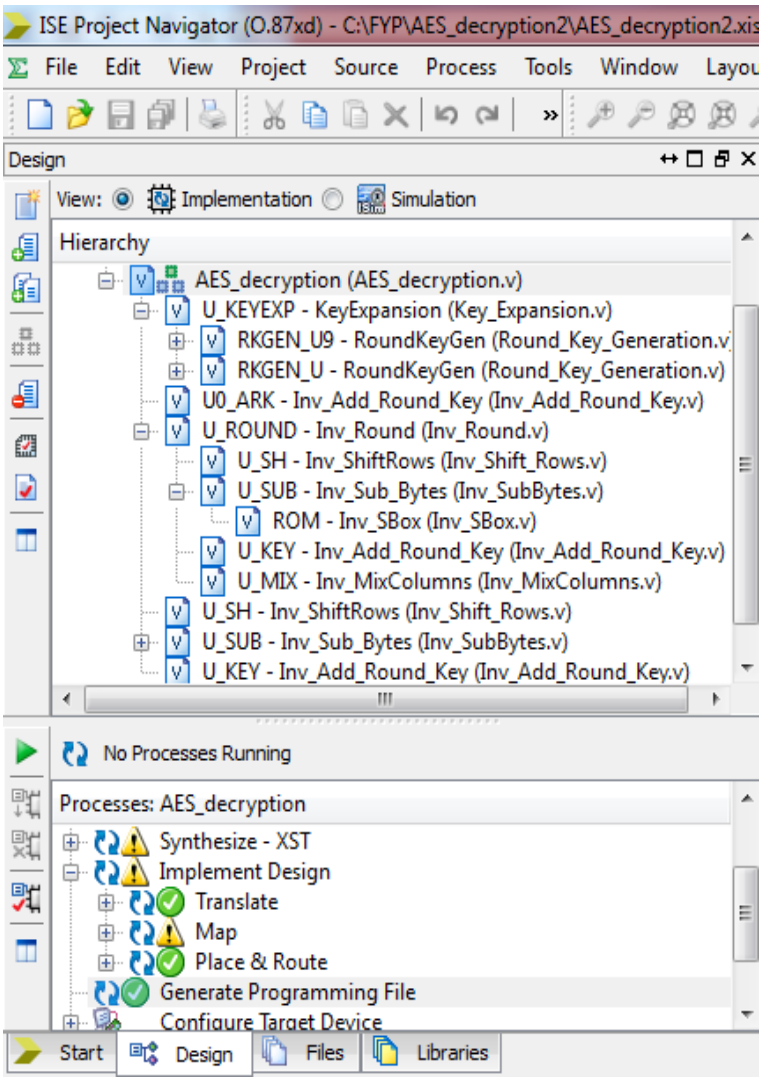


Figure 4-11 AES Decryption Design

The decryption test bench, simulated in Xilinx's simulator (ISim) gives accurate results of the data that was encrypted by the AES encryption code (see Figure 4-12).

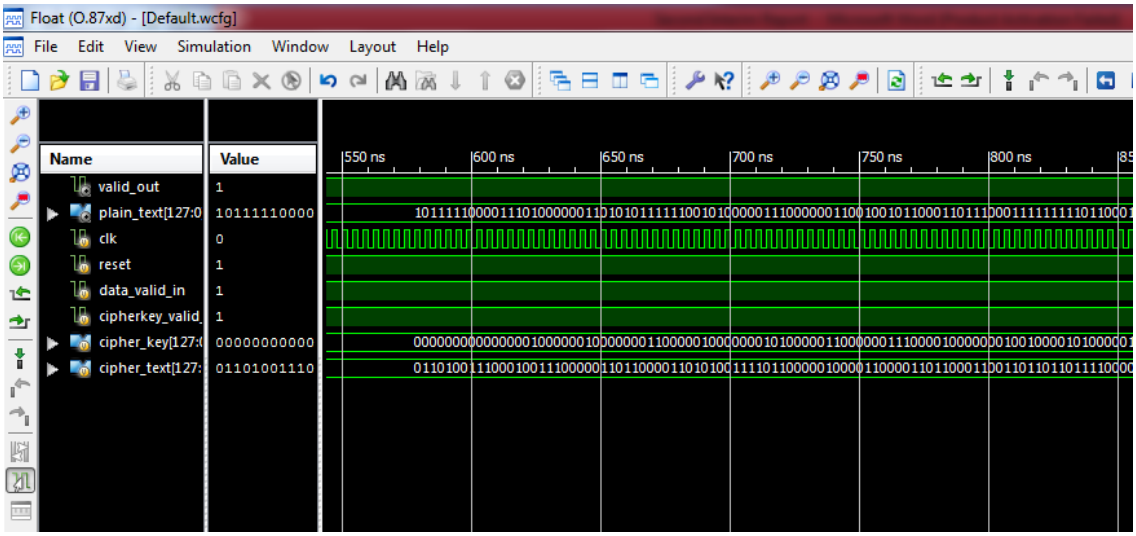


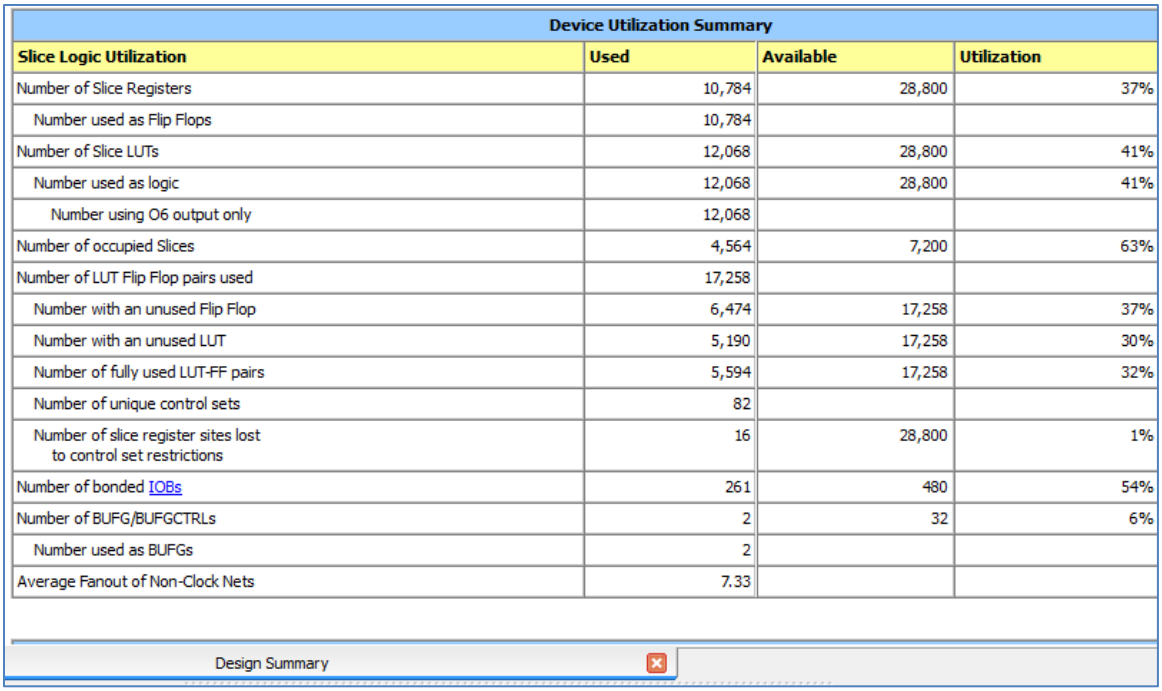
Figure 4-12 AES Decryption Software Simulation

The console shows the cipher text values again by a processing delay of 162ns which is equal to 40.5 clock cycles (see Figure 4-13).

```
cipher_text=69c4e0d86a7b0430d8cdb78070b4c55a cipher_key=000102030405060708090a0b0c0d0e0f plain_text=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
cipher_text=69c4e0d86a7b0430d8cdb78070b4c55a cipher_key=000102030405060708090a0b0c0d0e0f plain_text=be1d0357e50703258dc7fd8c5c2ce1cd
ISim>
```

Figure 4-13 Simulation Console Results for AES Decryption

Figure 4-14 shows the synthesis report obtained showing that the design utilizes only 32% of the device available (i-e the number of fully used LUT-FF pairs).



Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	10,784	28,800	37%
Number used as Flip Flops	10,784		
Number of Slice LUTs	12,068	28,800	41%
Number used as logic	12,068	28,800	41%
Number using O6 output only	12,068		
Number of occupied Slices	4,564	7,200	63%
Number of LUT Flip Flop pairs used	17,258		
Number with an unused Flip Flop	6,474	17,258	37%
Number with an unused LUT	5,190	17,258	30%
Number of fully used LUT-FF pairs	5,594	17,258	32%
Number of unique control sets	82		
Number of slice register sites lost to control set restrictions	16	28,800	1%
Number of bonded IOBs	261	480	54%
Number of BUFG/BUFGCTRLs	2	32	6%
Number used as BUFGs	2		
Average Fanout of Non-Clock Nets	7.33		

Figure 4-14 AES Decryption Design Measurements

Part of the generated RTL schematics for AES decryption algorithm is shown in Figure 4-15.

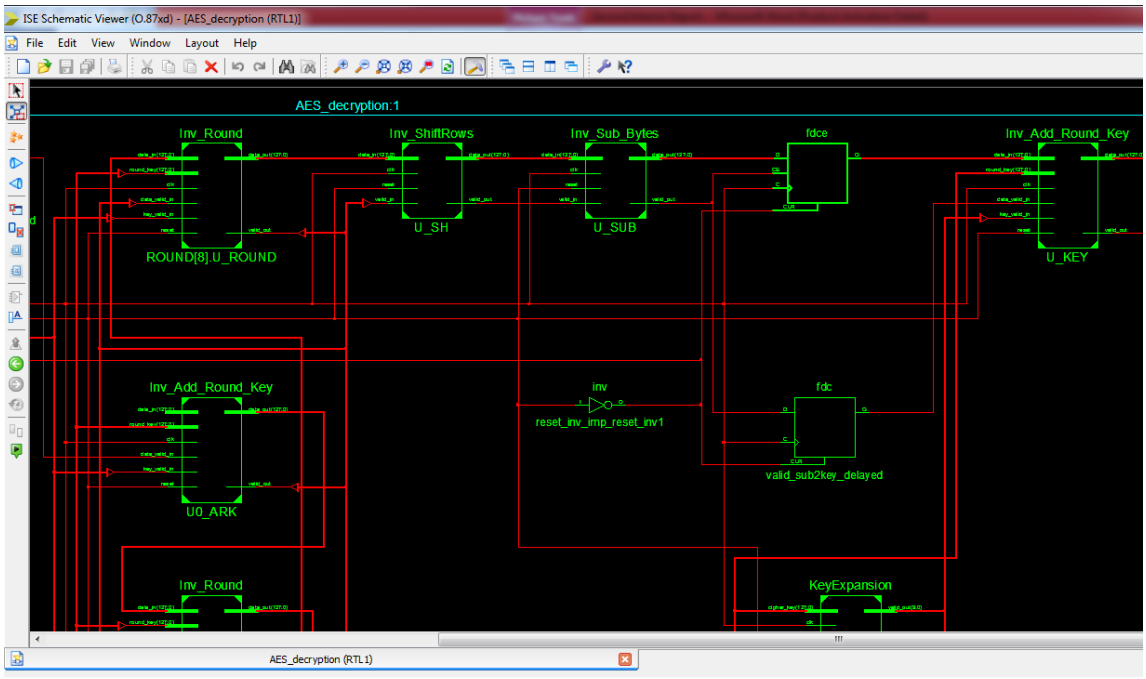


Figure 4-15 RTL Schematics for AES Decryption

After design implementation of AES decryption, a .bit file was generated in the last step to link the code with the hardware. This programming file was ported on the FPGA using the Digilent Adept software via Digilent USB Port available on the FPGA. Figure 4-16 shows the test results.

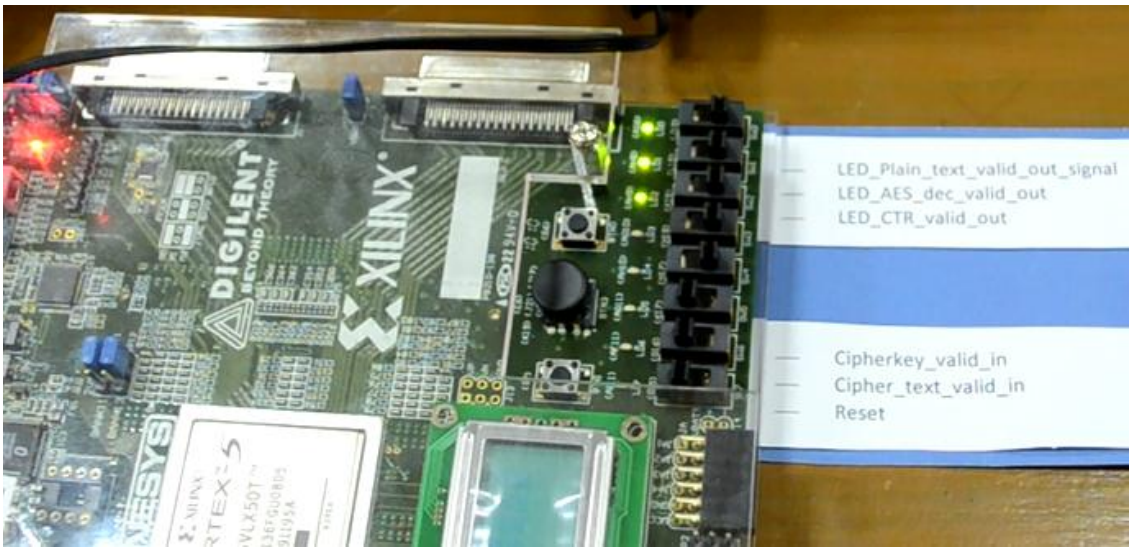


Figure 4-16 AES Decryption Hardware Testing

Again, since the code was designed with asynchronous active low reset, so no decryption can take place until unless we turn on the reset signal. As soon as we turn on the valid signal for the cipher text, we see that the counter module generates the counter input, turning on the counter valid output LED. And as soon we turn on the valid signal for cipher key, AES decryption LED goes high showing that the decryption block was executed and correspondingly the plain text valid out LED goes high showing that the final XORing between the temporary plain text and the cipher text has taken place.

4.3. 8-bit Parallel Interface

The 8-bit interface established between two boards was linked with the 8 on-board LEDs. As soon as the interface is turned on, 8-bit parallel communication starts between the two boards. This is illustrated in figure 29 below. A VHDCI cable from the first FPGA board is connected to the expansion FPGA board at the left and a CAT6 cable is connected via the RJ-45 port at the right side of the figure. No power source is connected to the

expansion board so the FPGA takes power from the VHDCI cable which turns on the power LED (green colour) of the board at the bottom right. When the top level .bit file is programmed on the FPGA, the programming LED (red colour) turns on. As soon as the communication is established between the two boards, the 8 on-board LEDs (top right of figure 4-17) turn on with the corresponding 8-bits of the VHDCI interface.

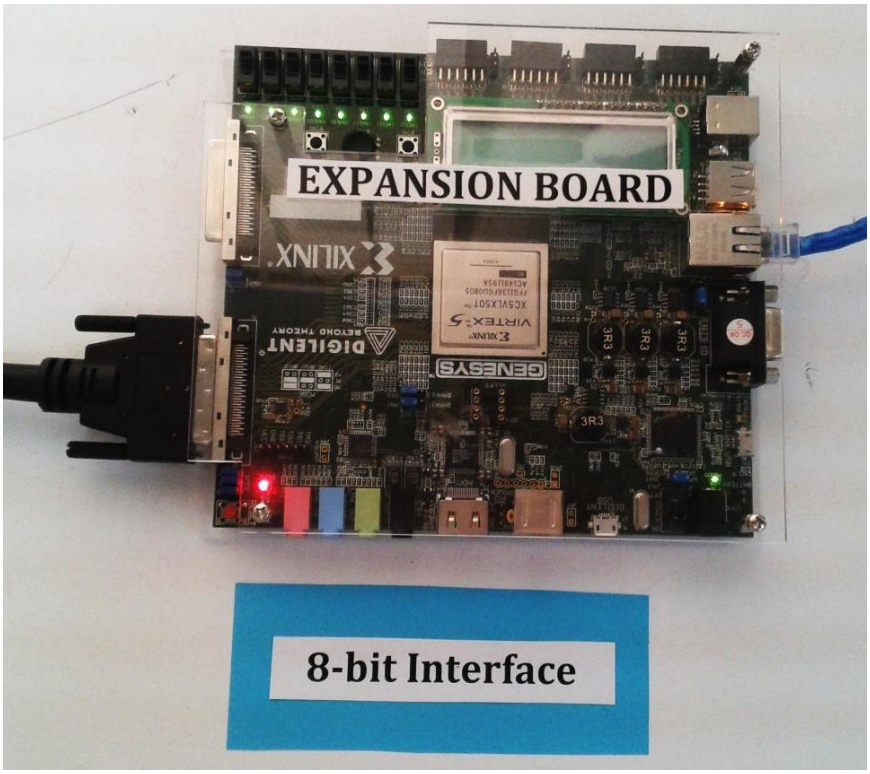


Figure 4-17 Parallel (8-bit) Interface

4.4. Ethernet Packet Analysis, Payload Encryption & Reassembly

A packet analysis code was written in Verilog that performed certain checks on the Ethernet packets received by the FPGA. Checks were applied on the IP version, Source IP, Destination IP, Packet Type, Protocol and Destination Port of the received packet. If the Source IP, Destination IP and Destination Port of the two users match the FPGA

database, and the packets exchanged between them are IP Packets (IP field with value 0x0800) with TCP Protocol (Protocol field containing 0x06) under IPv4 (version field containing 0x4), only then the payload is separated from the headers for encryption. The payload is then encrypted in chunks of 128-bits. In case if the last data chunk is less than 128 bits, then these bits are left unencrypted. During packet reassembly, the headers are concatenated with the encrypted payload and the unencrypted payload (if any). The software simulation of these Verilog codes is shown in Figure 4-18.

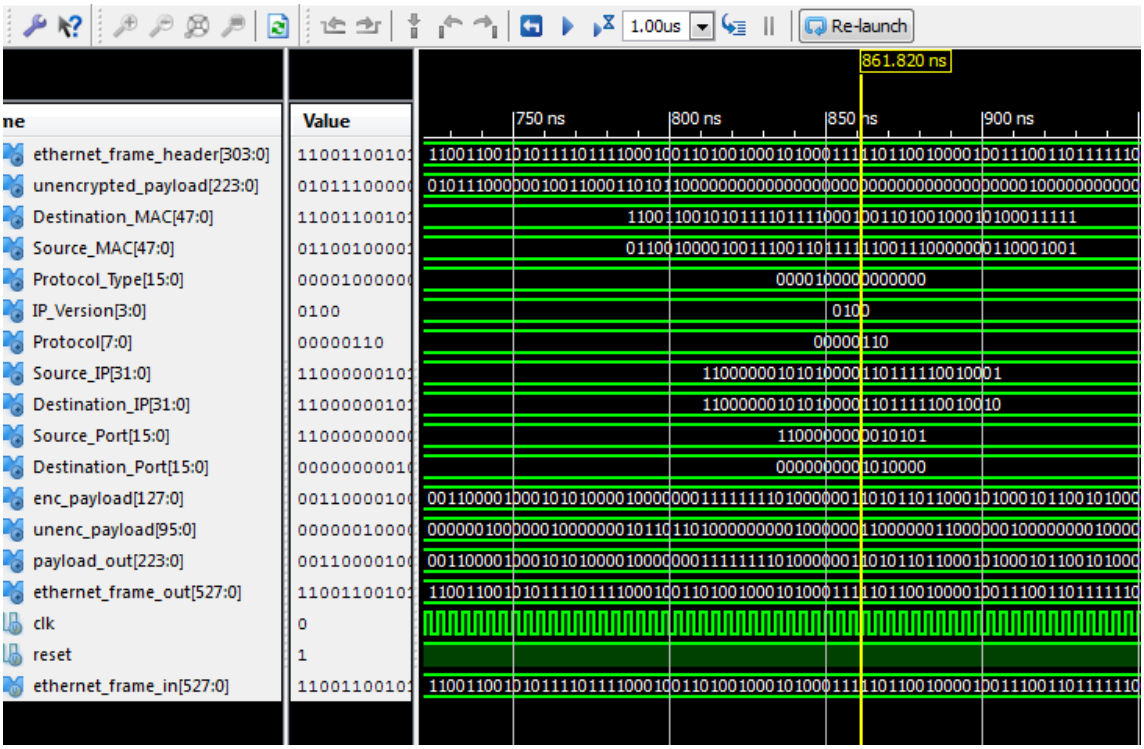


Figure 4-18 Simulation for Packet Analysis and Reassembly

The simulation results are for a 66 byte packet which is represented as Ethernet_frame_in with a hex value ccaf789a451f642737e701890800450000340169400080069 8e6c0a86f91c0a86f92c01500

505c098d60000000008002ffff64d10000020405b40103030201010402. The packet header was separated and analyzed. Since all the checks were passed (as shown in Figure 4-18), the first payload chunk of 128 bits was separated and encrypted, whereas the second payload chunk of 96 bits was left unencrypted. After reassembling the packet, the new encrypted frame produced had a hex value of ccaf789a451f642737e701890800450000340169400080069 8e6c0a86f91c0a86f92c01500 50308a8407fa06b628b2862432212c1d42020405b40103030201010402 (as shown in Figure 4-19).

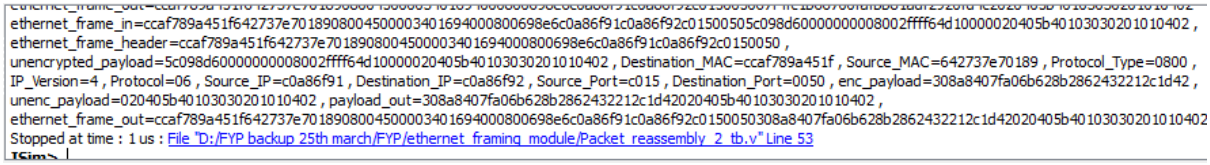


Figure 4-19 Packet Analysis & Reassembly Console Results

Chapter 5: Future Work

7.1 Proposed Project Design

7.2 Future Enhancements

7.3 Project Utility in Current Market

Chapter 5: Future Work

In this chapter we will discuss the future aspects for further developments in the proposed project. Moreover a summarized view of the entire project will also be given which will highlight IP Packet Handling Engine's core processes and conclude the discussion.

7.1. Proposed Project Design

Information security has become a critical concern for internet users these days. IP Packet Handling Engine has been designed and implemented to fulfill this dire need by providing an FPGA based hardware architecture platform for secure IP communications. It encrypts the IP traffic between two users using a pre-shared secret key without changing the Ethernet frame format.

7.2. Future Enhancements

The project entitled "IP Packet Handling Engine for Secure Communication" has been designed keeping in mind the area of FPGA used and the throughput performance upon implementing the cryptographic techniques. The Virtex-5 FPGA used for the project utilizes a total of 42% resources which includes the Ethernet core as well as the AES encryption algorithm. Hence, the device has the capacity to be further enhanced to make a complete security device offering all the security features i.e.

1. Confidentiality
2. Authentication
3. Non-Repudiation
4. Integrity

The bigger the FPGA, the more will be its capacity to handle the cryptographic algorithms in a parallel fashion. Other than this, any algorithm, not just proprietary algorithms can be designed and used for the purpose. This would result in a better inter-organization security.

7.3. Project Utility in the Current Market:

This designed project can be enhanced for implementation in following current markets:

a. Smart Cards.
b. RFID.
c. ATM Networks.
d. Image Encryption etc.

Chapter 6: Conclusion

7.1 Overview

7.2 Objectives Achieved/ Achievements

7.3 Contributions

7.4 Limitations

7.5 Applications

Chapter 6: Conclusion

6.1. Overview

The FPGA based project "IP Packet Handling Engine for Secure Communication" is designed for a fixed number of users. It does not only ensures the security of all the IP packets passing through it, but also overcomes the limitations in the processing power of the software based IP security systems.

6.2. Objectives Achieved/Achievements

The FPGA kit used in the project is not a standard Virtex-5 development board (ML 505); instead it is a custom board. Problems were faced in modifying the core generated for ML-505 to work on the custom FPGA board. Changing the UCF of a 50 trillion gate device with 1136 pin package was itself a big project, and the goal was successfully achieved.

Secondly, the efficiency of the AES algorithm designed in Verilog was very close to the professional designs. Professional FPGA developers have so far achieved AES encryption in 40 cycles, and algorithm designed under this project takes 40.5 cycles, which is a big achievement at student level.

6.3. Contributions

This security solution has been designed to contribute to the world of network security by laying a 100% hardware platform for providing IP security.

Secondly, this FPGA device is an IP Encrypter/ Decrypter but is itself IP-less. Hence, the project "IP Packet Handling Engine for Secure Communication" providing IP security on

MAC layer, rather than the network layer which makes our device invisible in any network. This means that our device can easily fool hackers within a network.

6.4. Limitations

The project “IP Packet Handling Engine for Secure Communication” was limited due to the availability of only 1 Ethernet Port on our FPGA kits. The system is stepped down from 1Gbps to 100Mbps when it goes to the network. Additional task of parallel communication between two FPGA boards was added to make use of the second Ethernet port.

6.5. Applications

The project named “IP Packet Handling Engine for Secure Communication” serves the basis to a simple stand-alone product which provides secure communication over the internet. This product is hack immune and cost effective, providing secure communication to military, intelligence, banking systems and/or any organization in the corporate world that intends to securely and confidentially communicate over the internet.

Appendix

1. ***Appendix A –Top Module of Ethernet Core***
2. ***Appendix B –Top Module of AES Encryption***
3. ***Appendix C –Top Module of Packet Reassembly***
4. ***Appendix D –Top Module of Project***

APPENDIX-A

Top Module of Ethernet Core

```
`timescale 1ns / 1ps
module TOP   (
                    CLKIN_IN,
                    rst,
                    // GMII Interface - EMAC0
                    GMII_TXD_0,
                    GMII_TX_EN_0,
                    GMII_TX_ER_0,
                    GMII_TX_CLK_0,
                    GMII_RXD_0,
                    GMII_RX_DV_0,
                    GMII_RX_ER_0,
                    GMII_RX_CLK_0 ,
                    RESET_B
                    );
// GMII Interface - EMAC0
output   [7:0]  GMII_TXD_0;
output          GMII_TX_EN_0;
output          GMII_TX_ER_0;
output          GMII_TX_CLK_0;
output          RESET_B;
input    [7:0]  GMII_RXD_0;
```

```
input            GMII_RX_DV_0;
input            GMII_RX_ER_0;
input            GMII_RX_CLK_0 ;
input    CLKIN_IN;
input    rst;
wire    [7:0]    GMII_RXD_0;
wire   [7:0]     GMII_TXD_0;
wire RESET_B;
wire rst_i;
wire GTX_CLK_0;
assign RESET_B = rst;                  //Giving active low reset
assign rst_i = ~rst;                   //Giving active high reset

MAC_example_design ONE_MAC
                                       (
                                       // Client Receiver Interface - EMAC0
                                       .EMAC0CLIENTRXDVLD(),
                                       .EMAC0CLIENTRXFRAMEDROP(),
                                       .EMAC0CLIENTRXSTATS(),
                                       .EMAC0CLIENTRXSTATSVLD(),
                                       .EMAC0CLIENTRXSTATSBYTEVLD(),

                                       // Client Transmitter Interface - EMAC0
                                       .CLIENTEMAC0TXIFGDELAY(),
                                       .EMAC0CLIENTTXSTATS(),
```

```
                                .EMAC0CLIENTTXSTATSVLD(),
                                .EMAC0CLIENTTXSTATSBYTEVLD(),

                                // MAC Control Interface - EMAC0
                                .CLIENTEMAC0PAUSEREQ(),
                                .CLIENTEMAC0PAUSEVAL(),

                                // GMII Interface - EMAC0
                                .GMII_TXD_0(GMII_TXD_0),
                                .GMII_TX_EN_0(GMII_TX_EN_0),
                                .GMII_TX_ER_0(GMII_TX_ER_0),
                                .GMII_TX_CLK_0(GMII_TX_CLK_0),
                                .GMII_RXD_0(GMII_RXD_0),
                                .GMII_RX_DV_0(GMII_RX_DV_0),
                                .GMII_RX_ER_0(GMII_RX_ER_0),
                                .GMII_RX_CLK_0(GMII_RX_CLK_0) ,
                                .GTX_CLK_0(GTX_CLK_0),
                                // Reference clock for RGMII IODELAYs
                                .REFCLK(REFCLK),
                                // Asynchronous Reset
                                .RESET(rst_i)
                                );
// Instantiate the module
clock100  clock100_125_200
                        (
```

```
                .CLKIN_IN (CLKIN_IN),
                .CLKFX_OUT (GTX_CLK_0), //125MHz clock for GMII
                .CLKIN_IBUFG_OUT (),
                .CLK0_OUT (),
                .CLK2X_OUT (REFCLK)  //200 MHz reference clock
                );
endmodule
```

APPENDIX-B

Top Module of AES Encryption

```
`timescale 1 ns/1 ps
module AES_encryption(
                        input wire clk,                 //system clock
                        input wire reset,               //asynch reset
                        input wire ctr_valid_in,                //data valid signal
                        input wire cipherkey_valid_in,          //cipher key valid signal
                        input wire [127:0] cipher_key,                  //cipher key
                        input wire [127:0] ctr_in,                              //CTR input
                        output wire valid_out,                  //output valid signal
                        output wire [127:0] cipher_text                 //cipher text
                        );
wire [9:0] valid_round_key;          //all round keys valid signals KeyExpansion output
wire [9:0] valid_round_data;         //all rounds ouput data valid signals
wire [127:0] data_round [0:9];       //all rounds data
wire valid_sub2shift;                //for final round connection
wire valid_shift2key;
wire [127:0]data_sub2shift;
wire [127:0]data_shift2key;
wire [1279:0] W;                        //all round keys
reg[127:0] data_shift2key_delayed;     //for delay register
reg valid_shift2key_delayed;
KeyExpansion keys_start (
                    clk,
```

```
                reset,
                cipherkey_valid_in,
                cipher_key,
                W,
                valid_round_key);
                //instantiate Key Expansion which will feed every round with round key
AddRoundKey ARK_round_0 (
                clk,
                reset,
                ctr_valid_in,
                cipherkey_valid_in,
                ctr_in,cipher_key,
                valid_round_data[0],
                data_round[0]
                );      //due to algorithm,first cipher key will be xored witht plain text

genvar i;               //instantiate all rounds , connect them with key expansion
generate
for(i=0;i<9;i=i+1)
        begin : ROUND
                Round rounds_1_9 (
                        clk,
                        reset,
                        valid_round_data[i],
                        valid_round_key[i],
```

```
                data_round[i],
                W[(10-i)*128-1:(9-i)*128],
                valid_round_data[i+1],
                data_round[i+1]);
        end
endgenerate
//the final round doesn't contain mixcolumns as declared in fips197 standard document
SubBytes SB_round_10 (
                clk,
                reset,
                valid_round_data[9],
                data_round[9],
                valid_sub2shift,
                data_sub2shift
                );
ShiftRows SR_round_10 (
                clk,
                reset,
                valid_sub2shift,
                data_sub2shift,
                valid_shift2key,
                data_shift2key
                );
AddRoundKey ARK_round_10 (
                clk,
```

```
                reset,
                valid_shift2key_delayed,
                valid_round_key[9],
                data_shift2key_delayed,
                W[127:0],
                valid_out,cipher_text);
/*as the final round has only three stages a delay register should be introduced
  to be balanced with key  expansion*/
always @(posedge clk or negedge reset)
begin
        if(!reset)
                begin
                        valid_shift2key_delayed <= 1'b0;
                        data_shift2key_delayed <= 1'b0;
                end
        else
                begin
                if(valid_shift2key)
                        begin
                                data_shift2key_delayed <= data_shift2key;
                        end
                valid_shift2key_delayed <= valid_shift2key;
                end
end
endmodule
```

APPENDIX-C

Top Module of Packet Assembly

```
`timescale 1ns / 1ps
module Packet_reassembly_2(
                input wire clk,
                input wire reset,
                input wire [527:0] ethernet_frame_in,
                output wire [303:0] ethernet_frame_header,  // separated header = 304 bits
                output wire [223:0] unencrypted_payload,    // as input to AES -> 224 bits
                output wire [47:0] Destination_MAC,
                output wire [47:0] Source_MAC,
                output wire [15:0] Protocol_Type,
                output wire [3:0] IP_Version,
                output wire [7:0] Protocol,
                output wire [31:0] Source_IP,
                output wire [31:0] Destination_IP,
                output wire [15:0] Source_Port,
                output wire [15:0] Destination_Port,
                output wire [127:0] enc_payload,
                output wire [95:0] unenc_payload,
                output wire [223:0] payload_out,
                output wire [527:0] ethernet_frame_out
                );
payload_separation hjs(
                clk,      //device clk
```

```
                reset, //async active low reset
                ethernet_frame_in,          // packet size = 66 bytes = 528 bits
                ethernet_frame_header,      // separated header size = 304 bits
                unencrypted_payload, // will be given as an input to AES -> 224 bits
                Destination_MAC,
                Source_MAC,
                Protocol_Type,
                IP_Version,
                Protocol,
                Source_IP,
                Destination_IP,
                Source_Port,
                Destination_Port
                );


payload_enc enccc (
                clk,
                reset,
                unencrypted_payload,
                enc_payload,
                unenc_payload,
                payload_out
                );
assign ethernet_frame_out = {ethernet_frame_header, payload_out};
endmodule
```

APPENDIX-D

Top module of Project

```
`timescale 1ns / 1ps

module TOP_MAIN(

                input wire CLKIN_IN,

                input wire rst,

                // GMII Interface - EMAC0

                output wire [7:0] GMII_TXD_0,

                output wire GMII_TX_EN_0,

                output wire GMII_TX_ER_0,

                output wire GMII_TX_CLK_0,

                input wire [7:0] GMII_RXD_0,

                input wire GMII_RX_DV_0,

                input wire GMII_RX_ER_0,

                input wire GMII_RX_CLK_0 ,

                output wire RESET_B,

                // On board encryption interface signals

                input wire reset,

                input wire data_valid_in,

                input wire cipherkey_valid_in,

                input wire sw1,

                output wire ctr_valid_out,

                output wire valid_out_signal,

                output wire v1,

                output wire v2,
```

```
                        output wire v3,
                        output wire v4,
                        output wire v5,
                        output wire v6,
                        output wire v7,
                        output wire v8
                        );
wire rst_i;
wire GTX_CLK_0;
assign RESET_B = rst;                   //Giving active low reset
assign rst_i = ~rst;                    //Giving active high reset
MAC_example_design ONE_MAC
                        (
                        // Client Receiver Interface - EMAC0
                        .EMAC0CLIENTRXDVLD(),
                        .EMAC0CLIENTRXFRAMEDROP(),
                        .EMAC0CLIENTRXSTATS(),
                        .EMAC0CLIENTRXSTATSVLD(),
                        .EMAC0CLIENTRXSTATSBYTEVLD(),
                        // Client Transmitter Interface - EMAC0
                        .CLIENTEMAC0TXIFGDELAY(),
                        .EMAC0CLIENTTXSTATS(),
                        .EMAC0CLIENTTXSTATSVLD(),
                        .EMAC0CLIENTTXSTATSBYTEVLD(),
                        // MAC Control Interface - EMAC0
```

```
                        .CLIENTEMAC0PAUSEREQ(),
                        .CLIENTEMAC0PAUSEVAL(),
                        // GMII Interface - EMAC0
                        .GMII_TXD_0(GMII_TXD_0),
                        .GMII_TX_EN_0(GMII_TX_EN_0),
                        .GMII_TX_ER_0(GMII_TX_ER_0),
                        .GMII_TX_CLK_0(GMII_TX_CLK_0),
                        .GMII_RXD_0(GMII_RXD_0),
                        .GMII_RX_DV_0(GMII_RX_DV_0),
                        .GMII_RX_ER_0(GMII_RX_ER_0),
                        .GMII_RX_CLK_0(GMII_RX_CLK_0) ,
                        .GTX_CLK_0(GTX_CLK_0),
                        // Reference clock for RGMII IODELAYs
                        .REFCLK(REFCLK),
                        // Asynchronous Reset
                        .RESET(rst_i)
                        );
// Instantiate the module
clock100        clock100_125_200 (
                        .CLKIN_IN (CLKIN_IN),
                        .CLKFX_OUT (GTX_CLK_0), //125MHz clock for GMII
                        .CLKIN_IBUFG_OUT (),
                        .CLK0_OUT (),
                        .CLK2X_OUT (REFCLK)  //200 MHz reference clock
                        );
```

```
board1 b1111 (
                reset,
                data_valid_in,
                cipherkey_valid_in,
                sw1,
                ctr_valid_out,
                valid_out_signal,
                v1,
                v2,
                v3,
                v4,
                v5,
                v6,
                v7,
                v8
                );
endmodule
```

Bibliography

1. References

Bibliography

References

[1] XILINX SUPPORT DOCUMENTATION, "*Virtex-5 Family Overview",* DS100 (v5.0), February 2009.

[2] XILINX SUPPORT DOCUMENTATION, "*Virtex 5 FPGA Configuration User Guide,UG191 (v3.11)",* October 19, 2012.

[3] XILINX SUPPORT DOCUMENTATION, "*Virtex 5 FPGA User Guide,UG190* (v5.4)", March 16, 2012.

[4] XILINX SUPPORT DOCUMENTATION, "*Virtex 5 FPGA Embedded Trimode Ethernet MAC User Guide, UG194* (v1.10)", Feb 14, 2011.

[5] XILINX SUPPORT DOCUMENTATION, "*Virtex 5 FPGA Embedded Trimode Ethernet MAC Wrapper v1.7, UG340",* April 19, 2010.

[6] Behrouz A. Forouzan*, "Data Communications and Networking",* 4th ed, McGraw Hill, 2007.

[7] IEEE TRANSACTIONS, "*IPSec Implementation on Xilinx Virtex-II Pro FPGA and Its Application",* Lu, J. ; Appl. Res. Lab., Washington Univ., USA; Lockwood, J., April 2005.

[8] FIPS Publication 197, "*Announcing the ADVANCED ENCRYPTION STANDARD (AES)*" November 26, 2001

[9] Sliman Arrag, Abdellatif Hamdoun, Abderrahim Tragha and Salah eddine Khamlich, *"Design and Implementation A different Architectures of mixcolumn in FPGA".*

[10] Marvell-Alaska-Ultra-88E1111-GbE Product Brief, Integrated 10/100/1000 Ultra Gigabit Ethernet Transceiver datasheet

[11] DM9000A Ethernet Controller with General Processor Interface datasheet.

[12] Digilent Inc. (Washington, USA,) Internet: http://www.digilentinc.com/Products/Detail.cfm?Prod=GENESYS, 2000 [Sept. 20, 2013].

[13] Deepak Kumar Tala, Internet: http://www.asic-world.com/verilog/veritut.html, 2014 [Apr. 5, 2014]

[14] Jean P. Nicolle, Internet: www.fpga4fun.com, May 18, 2013 [Apr. 5, 2014].

[15] Xilinx Inc., Internet: http://www.xilinx.com/training/free-video-courses.htm, 2014 [May. 4, 2014].

[16] Xilinx Inc., Internet: http://www.xilinx.com/csi/training/how-to-configure-an-fpga.htm, 2014 [May. 4, 2014].

[17] Xilinx Inc., Internet: http://www.xilinx.com/ipcenter/coregen/ip_update_install_instructions.htm, 2014 [May. 4, 2014].