ISTANBUL TECHNICAL UNIVERSITY

FACULTY OF ELECTRICAL AND ELECTRONIC

The Software And Hardware Common Implementatıon Of A Secure Data Communıcatıon Protocol Usıng Aes Algorıthm

GRADUATION PROJECT

AHMET JORGANXHI

040100948

Department:Electronics and Communication Engineering

Program: Electronics Engineering

Supervisor: Assistant Prof. Dr. S. Berna Örs YALÇIN

MAY 2014

ACKNOWLEDGMENT

In a conference Steve Jobs has stated as:" Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to love what you do." As an electronics engineering student I would like to thank my supervisor Assistant Prof. Dr. S. Berna Örs Yalçın for giving me the oppurtinity to do such a great work in the first steps of my life as an engineer and for her guidance and support during this thesis work. Also I would like to thank my friend Arif Gençosmanoğlu for his all helps and to thank to Engineer, Msc. Emre Göncü for all his help and guidance.

May 2014 Ahmet JORGANXHI

INDEX

Page

ABBREVIATIONS

AES	: Advanced Encryption Standard
HDL	: Hardware Description Language
VHDL	: Very high speed integrated circuit Hardware Description
FPGA	: Field Programmable Gate Array
UART	: Universal Asynchronous Receiver/Transmitter
ISE	: Integrated Software Environment
EDK	: Embedded Development Kit
XPS	: Xilinx Platform Studio
SDK	: Software Development Kit
RTL	: Register Transfer Level
ASM	: Algorithmic State Machine

LIST OF FIGURES

THE SOFTWARE AND HARDWARE COMMON IMPLEMENTATION OF A SECURE DATA COMMUNICATION PROTOCOL USING AES ALGORITHM

SUMMARY

Cryptography is the one of the most important study areas at the present time that deals with the study and practice of techniques for secure communication. Generally a cryptographic system first encryps message, and after by using a secured shared key provide secured communication between two systems. Modern cryptograhphic systems intersect important disciplines such as computer science, mathematics and electrical engineering.

The first step of this project was to provide this secured key that in the next steps will be shared between two systems. For this key exchange Diffier-Hellman key exchange method is used. The Diffie-Hellman key exchange method allows two systems that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communication channel. This key exchange method is designed as hardware in Verilog Hardware Description Language as combination of two different modules: one module which performs multiplication by using "Karatsuba" algorithm and another module which performs exponentiation operation by using "Square and Multiply" algorithm and modulo operation. By using these three mathematical operations $a^b mod p$ operation was realized, where a, b and p where 128-bit numbers.

After the designed Diffie-Hellman module together with Advanced Encryption Standard (AES) algorithm is imported to MicroBlaze as peripheral. MicroBlaze is a soft core processor designed for Xilinx Field Programmable Gate Array's (FPGA) from Xilinx. In this project two Spartan-6 FPGA wass used, one for encrypting data and one for decrypting data. So MicroBlaze in this case wass used as a controller between these two designed modules and forms a cryptographic system. These two systems first by using Universal Asynchronous Receiver/Transmitter (UART) protocol communication method

a key exchange was achieved. Then in one system, MicroBlaze by using encryption algorithm of AES, encrypts data with the secured key, that was exchanged between two system, and again by using UART protocol was sent to the other FPGA, where its MicroBlaze receives it. First the key is checked from the MicroBlaze and after the verification of key was achieved the received encrypted data by using the exchanged key and decryption algorithm of the AES is decrypted by MicroBlaze.

GÜVENLİ BİR VERİ HABERLEŞME PROTOKOLÜNÜN AES ALGORİTMASI İLE DONANIM-YAZILIM ORTAK OLARAK GERÇEKLENMESİ

ÖZET

Kriptografi günümüzde güvenli haberleşme tekniklerini araştıran ve bu teknikleri uygulayan bir şifreleme bilimidir. Genel olarak bir kriptografik sistem şifrelenmemiş bir mesajı şifreler ve ortak paylaşılmış olan bir anahtar sayesinde iki sistem arasında güvenli haberleşmeyi sağlar. Modern kriptografik sistemler bilgisayar bilimleri, matematik ve elektrik mühendisliği gibi önemli bilim dallarının kesişmeleri sonucu oluşur.

Bu projenin gerçekleşmesinde ilk adım olarak iki sistem tarafında paylaşılacak olan güvenli ortak anahtarı sağlayan birimin tasarımı gerçeklendi. Bu birimin tasarımında anahtar paylaşım yöntemi olarak Diffie-Hellman yöntemi kullanıldı. Diffie-Hellman anahtar paylaşım yöntemi önceden birbirinden hiçbir bilgisi olmaan iki sistemin güvenli olmayan bir haberleşme kanalında anahtar paylaşımını sağlar. Bu anahtar paylaşım birimi bir donanım olarak Verilog Donanım Tanımlama Dili kullanılarak iki modülünü birleşimi olarak tasarlandı. Bu modüllerden biri "Karatsuba" algoritmasını kullanarak çarpım işlemini gerçekleştirirken diğer modülde "Square and Multiply" yöntemini kullanarak üst alma ve elde edilen sonucun bir sayıya göre mod alma işlemini gerçekleştiriyor. Tasarlanan bu donanım bu üç matematiksel işlemi kullanarak $a^b mod p$, a, b ve p 128- bit sayılar olmak üzere işlemini gerçekleştirecektir.

Bu tasarlan Diffie-Hellman donanımı ve şifreleme ve çözme işlemini gerçekleştiren gelişmiş şifreleme standardı (Advanced Encryption Standar, AES) algoritması donanımı ile birlikte MicroBlaze çevresel birim olarak eklenmiştir. MicroBlaze Xilinx FPGA'lerinde kullanılan bir soft-core işlemcidir. Bu projede biri şifrelemek diğeri de

çözmek amaçlı olmak üzere iki Spartan-6 FPGA kullanılmıştır. Bu projede MicroBlaze tasarlanan bu iki donanımı kontrol ederek bir kriptografik sistem oluşturarak güvensiz bir haberleşme kanalında güvenli bir haberleşme gerçekleşdirdi. İlk olarak iki sistem UART haberleşme protokolünü kullanarak anahtar paylaşım işlemini gerçekleşecektir. Bu adımdan sonra bir sistemin MicroBlaze işlemcisi AES donanımını ve bir adım önce güvenli bir şekilde paylaşılmış olan anahtarı kullanarak bir mesajı şifreleyerek diğer taraftaki sisteme, yani MicroBlaze'e şifrelemiş mesajı Evrensel Eşzamansız Alıcı İletici (Universal Asynchronous Receiver/Transmitter,UART) haberleşme protokolü üzerinden gönderilimiştir. Şifrelenmiş mesajı kabul eden MicroBlaze işlemcisi AES donanımını ve paylaşılmış olan anahtarı kullanarak şifrelenmiş mesajı çözerek güvensiz bir haberleşme kanalında güvenli haberleşme işlemini gerçekleştirmiştir.

1. INTRODUCTION

In this technological age indeed there is no any situation where the security is a not a considerable situtaion because of universal electronic conncectivity, of viruses and hackers. The reasons stated above have made cryptography as a discipline more complete, sophisticated and main discipline for improvement of practical, readily available utilizations to carry out security in different situations [1]. The crucial goal of cryptography is to implement communication of two systems over an insecure channel in such a way that an observer system, cannot find out the data that is communicating in the insecure channel. System A encrypts the data, which can be anything arbytrary, using a prearranged secured key, and sends the encrypted message over the insecured channel. The oberver O, upon seeing the encrypted in the channel cannot resolve what the data was, but the system B that knows the prearranged secure key can decrypt the encrypted message and determine message.

The initial step of a cryptographic system is to share a common key between two system in such a way that the observer system cannot determine it. One of the methods for sharing specific cryptographic key between two systems over an insecure channel is Diffie-Hellman key exchange method. The Diffie-Hellman key exchange method provide a secret key, which is shared in two systems, that can be used for secure communication of information in insecure communication channel. The key is shared between two systems by using two matheamtical operations such as exponentiation and mod operation. In this thesis these two mathematical operations are designed as a hardware by the means of the Verilog Hardware Description Language. In the design of this unit two methods are used: one for multiplication “Karatsuba” method is used and one for exponentiation “Square and Multiply” method is used. The mod operation is achieved by subtraction operation by the means of an adder.

As the security has become more important issue the need for the different cryptographic algorithms also has rised. One of the algorithms that is used in cryptography is Advanced

Encryption Standard (AES) which is a computer security standard issued by the National Institute of Standards and Technology (NIST) and the specifications of this encryption standard are published in the Federal Information Processing Standards (FIPS) Publication 197 [2]. Different versions of AES algortihms are used in cryptography (AES128, AES196 and AES256) varying according to the size of the key which is going to be used for encryption. In this project the hardware model of AES128 is used which is implemented by using Very high speed integrated circuit Hardware Description Language.

Finally both Diffie-Hellman key exchange unit and AES unit are added to soft core processor of the Xilinx Field Programmable Gate Array's (FPGA) named as MicroBlaze. The MicroBlaze is a virtual microprocessor where constructionf of this microprocessor is achieved by linking blocks of code named as core which are found inside a Xilinx FPGA [3]. In this project MicroBlaze with its own peripherals and the two added peripherals forms one cryptographic system. Two such systems will communicate through their UART communication protocol. Then under control of MicroBlaze of each system sequentially a secured key will be shared, a message will be encrypted with the obtained secured key and will be sent from the UART communication protocol and from the other system will be decrypted by using the common shared key.

2.CRYPTOGRAPHY

2.1 Overview

Cryptography is an indispensable tool for protecting information in computer systems[6]. The core of cryptography is of course secure communication that essentially consist of two parts: first is secret key establishment and how it is communicated securely once a key is ensured.Already is stated that secure key establishment essentially makes two system to send messages to each other such that at the end of this protocol there is a shared key that both systems agreed on, and with this shared key both systems would know that they are talking to each other. But an attacker who listen this conservation has no idea what a shared key is. Once both systems have a shared key they want to exchange their messages securely using this key. In other words encryption keys provide both confidentiality and integrity.

Except of these two things stated above, cryptography aslo deals with digital signature. Digital signature basically is the analog signature in the physical world. In the physical world, when you sign a document essentially you write your signature on that document and your signature is always the same. In the digital world this cannot possibly work because if the attacker just obtain one signed document, attacker can copy the signature and use it. And so is simply not possible in the digital world that the signiture to be same for all cases. The way digital signatures work is basically by making the digital signature to be a function of the content being signed[6]. So an attacker who tries to copy the signature from one document to another is not going to succeed because the signature on the new document is not going to be the proper function of the data in new document. As a result the signature will not verify.

Another application of cryptography is anonymous communication. So here imagine user A wants to talk to a chat server B, and perhaps user wants to talk about his/her medical condition and wants to do that anonumously so that the chat server does not actually

know who he/she is. There is a standart method called mix net that allows user A to communicate over the public internet with the server B through the sequence of proxies such that at the end of the communciation server B has no idea who has just talked to[6]. The way mix net work is basically as user A sends messages to serve B through sequence of approxies, these messages getting encrypted in the encrypter appropriately so that server B has no idea has talked to and the proxies themselves do not even know that user A is talking to server B or actually who is talking to who more geneally. One interesting thing about this anonymous communication channel is that communication is bidirectional. In other words even though server B has no idea who has talking to, server B can still response to user A and user A can get this messages.

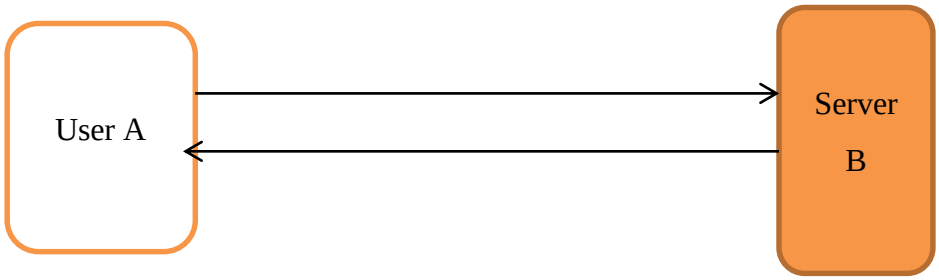

Figure 2.1: Schematic of anonymous communication

Once anonymous communication is achieved, other privacy mechanisms can be built. One example of these mechanisms is anonymous digital cash[12]. In the physical if a person has a physical money, he/she can walk to a book store and buy a book and seller have no idea who the buyer is. In the digital world basically user A may have digital money, and he/she might want to spend that digital money at some online stores. When user A spends his/her coin at store, the store have no idea who user A is. So the same anonimity is provided as in the physical cash. Now the problem is, in the digital world user A can take the coin that he/she had and before he/she spent it, user A can actually make copies of it. Then instead of having just one coin, now he/she has more than one coin and they are all same. There is no anything that prevents user A of having this replicas of digital coins and spend them in other stores. The question is how the digital cash is provided but at the same time how to prevent the user from double spending the digital coin at different stores? In some situations there is a paradox when anonimity is in

confilct with security because if we have anonymous cash there is nothing to prevent user from double spending the coin and because the digital coin is anonymous we have no way of telling it. The way how this tension is resolved is basically by making sure that if user spends the digital coin once then no one knows who he/she is but if user spends the digital coin more than once all of his/her sudden identities completely will be exposed and then user could be subject to all relatively illegal problems.

There are some applications of cryptography that can be classified as purley magical. One such application is Privately outsourcing computation[6]. An example of illustrating the main point of this application a google search example can be considerable. Imagine a user A has search that he/she wants to issue. It turns that they are many special encryption scemes such that user A can send an encryption of her query to google and then because of property of encryptiom sceme google can actuallay compute on the encrypted values wihout knowing what the information are. So google can actually run its massive search algorihm on the encrypted query and recover an encrypted results. Google would send an encrypted result back to user A, user A would decrypt and then he/she would receive the results but the magic here is all google saw is just the encryption of the queries and nothing else. So google as a result has no idea what user has just searched for and nevertheless user actually learn exactly what he/she wants to learn.

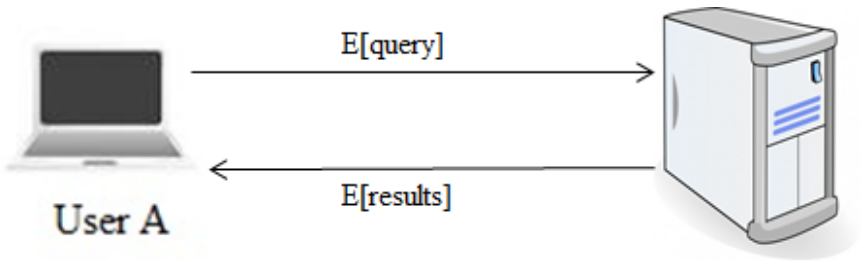

Figure 2.2: Simplified model of privately outsourcing computation

Another application that can be considered as pure magic of cryptography is zero knowledge[6]. Here what happens is that there is a certain number N which an user knows and the way how the number N is constructed is as a product of two large primes. These two primes are p and q, where each one size is thousand digitis. Multiplying two

numbers of size thousand of digits is fairly easy, but if just their product is given, figuring out their factorization into primes is actually quite difficult. So user happens to have this number N and also knows the factorization of N. On the other side a system, which can be an a system or any arbitrary cryptographic system, just has the number N. It does not know actually about the factorization. Now the magical fact about the zero knowledge is that user can proof to system that he/she knows the factorization of N and system can check and become convinced that user actually knows factoriaztion of N. However system learns nothing at all about the factors p and q and this is proofable. System absolytely learns nothing at all about the factors p and q.

Modern cryptography is very rigorous. Every concept in modern cryptography is following three rigorous steps. First step is a new primitive like a digital signature is precisely specifying a threat model, that is what can an attacker do to attack a digital signature and there is an exact definition on signature for being unforgeable. Then the next step is proposing a construction and in final step a proof is going to be obtained. This proof is about any attacker that can attack the construction under this threat model, and that attacker also can be used for solving some underlied hard problem. And as a result if the problem is really hard that actually prooves no attacker can break the construction under the threat model[6].

2.2 History of Cryptography

A growing discipline in this technological age is cryptology. Despite the fact that it has been used for thousand years to protect secret messages, the systematic study of cryptography as a science quite new. Before the modern era, encryption was attempted to provide confidency in communication, such as those of agents, military leaders and diplomats.

The first admitted sign about the beginning point of cryptography is in or around 2000 BC in Egypt where hieroglyphics were used to adorn the tombs of decreased emperors[4]. Scriber has used some uncommon hieroglyphics in place of more typical ones. The aim of the scriber was not to mask the message but perhaps to adjust its form in

a way which would make it to come out more dignified. Even though the epigraph was not a form of secret writing, but combinated some sort of conversion of the original text, and is the oldest text to do so.

Fast forwarding to around 100BC, Italian emperor Julius Ceasar used a system of cryptograpghy (i.e. the "Ceasar Cipher") which moved each letter two places farther over the alphaber (A cipher is an algorithm used for an ecnryption or decryption)[4]. So the cipher developed by Ceasar was just shifting each character by 3, so the character 'A' was substituted by 'D', character 'B' was substituted by 'E' and so on.

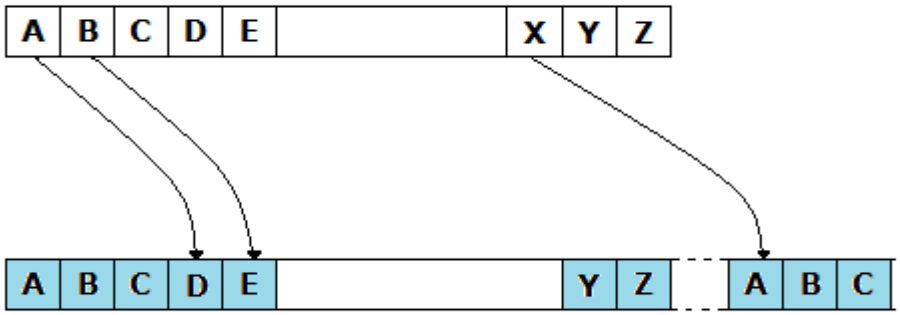

Figure 2.3: Ceasar Cipher

Easily can be seen these kind of system such as Ceasar Cipher depends on the confidentiality of the system and not on the encryption key. So once the mysteri of the system is solved, the encrypted messages can be decrypted easily. The substitution ciphers can be broken easily by the frequency of the letters in the language.

During the 16th century, the most famous cryptographer Blaise de Vigenere (1523-1596), wrote "Tracte des Chiffres", where a Trithemius table was used, but the way the system key was working was changed[4]. Vigenere has designed a cipher that was using the encryption key. In one of the designed chiphers, the encryption key was repeated multiple times traversing the whole message, and then the encrypted text was by adding the message character with the key character module 26. As Ceasar cipher also Vigeneres cipher can be broken easily, however the Vigeneres cipher is considered as the first cipher that has introduced the use of the encryption key.

The importance and the development of the ciphers has continued with development on elecetricity and electronics during the periods following World Wars and after. During World War II , the nutral country Sweden had one of the most cryptanalysis departmen in the world which it was formed in 1936 and 22 peoples were employed. In World War II, U.S. army has shown a great success at breaking Japanese code, while the same success could not be shown from Japanese, even though Japanese were thinking that their codes were unbreakable[4]. Today cryptography is one of the most study area that each day improves itself as a science for providing high level security to every one.

2.3 Symmetric-key Cryptography

Symmetric-key cryptography is one type of cryptography where the encryption key is common for both encryption and decryption algorithms[5]. A symmetric key cryptosystem is constructed from five main components. These components are: plaintext, encryption algorithm, secret key, ciphertext and decryption algorithm.

1- **Plaintext:** is the original clear data or information that is provided as input to the encryption algorithm.
2- **Encryption algorithm:** is the algorithm that executes different mathematical operations such as permutation and substitutions on the plaintext.
3- **Secret Key:** is another input that is applied to the encryption algorithm and also to the decryption algorithm. The output of the encryption algorithm differs according to the secret key, because is the factor which determines the definite substitutions and permutations.
4- **Ciphertext:** is the scrambled message obtained as output from encryption algorithm. It varies according to the plaintext and secret key.
5- **Decryption Algorithm:** necessarily is the inversion process of the encryption algorithm. Ciphertext and secured key are applied as input and as output the original plaintext is produced.

Easily can be seen that in symmetric-key cryptosystems the point is not to keep secret the algorithm but it is required to keep secret the secret key. In the symmetric-key cryptosystems senders and receivers must have shared this secret key securely because if an attacker can discover he can decrypt the ciphertext and obtain the plaintext.

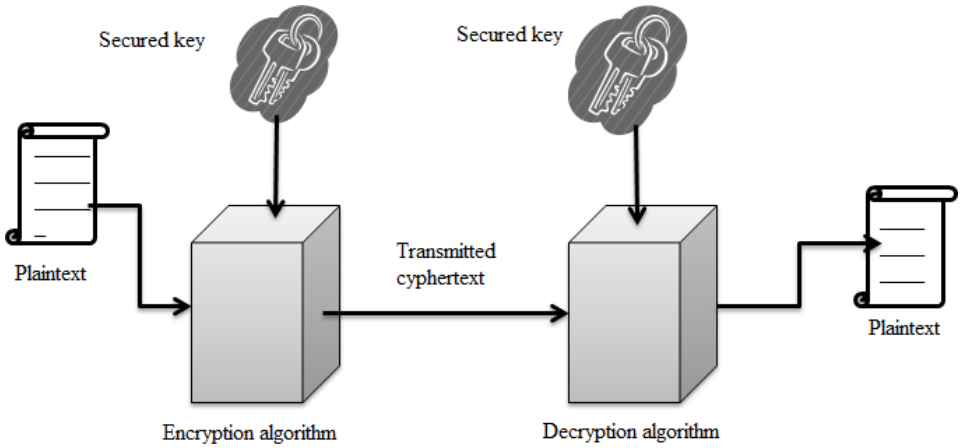

Figure 2.4: Model of symmetric encryption

3. DIFFIE-HELLMAN KEY EXCHANGE

3.1 The Diffie-Hellman Key Exchange Mechanism

The Diffie-Hellman key exchange method allows two systems that have no prior knwoledge of each other to jointly establish a shared secret key over an insecure communication channel. This key exchange method was matured in 1976 and announced in "New Directions in Cryptography"[6]. The Diffie-Hellman key exchange fixes the following problem. System A and system B are both a symmetric-key cryptosystems. System A and system B wants to share their secret key in order to communicate securely with each other. Both systems A and B must have a shared secret key but the attacker must not have any idea about this secret key. One mechanism of key exchange is Merkle Puzzles, where with block ciphers, a secret key can be shared with one and another. On the other hand in the case of Merkle puzzles the attacker has a quadratic gap compared to the the participant, in other words if the participant spent time proportional to "n" , the attacker can break the protocol in time proportional to n^2 . As a result, that protocol is too insecured to be considered practical. In contrast to the Markle puzzle key exchnage mechanism, in Diffie-Hellman key exchange mechanism an exponential gap is achieved between the attackers work and the participants work.

The mechanism of Diffie-Hellman key exchange works as follow. First select a large prime "p", actually "p" is usually used to denote primes, and a fix an integer "α", that happens to live in the range of {1,...,p}. Both "p" and "α" are parameters of Diffie-Hellman protocol that are choosen once and are fixed forever. After both systems A and B, choose a random number "a" and "b" respectively, in the range of {1,...,p-1}. Then system A is going to compute RA= α^a modp. So system A computes the exponentation and reduces the result to the modulo of p, and the result is sent to the system B. On the other hand also system B computes RB= α^b modp and sends to the system A. So system A sends RA to system B and system sends RB to system A, and now a shared secret key

can be generated. So what is the shared key? Well it is named K_{ab} and it is defined as α^{ab} modp. The amazing observation that Diffie-Hellman had back in 1976 is that in fact both parties can compute the value of K= α^{ab}. System A can compute this value since, it can take the value RB, that it had received from system B, and raise to the power of "a" . Also the system B compute the value in same manner as system A, and as a result both systems A and B, have computed the K and end up with the same secret key. This protocol has introduced a new age in cryptography, where now is not just about designing block ciphers but it is actually about designing algebraic protocols that have properties as in this case.

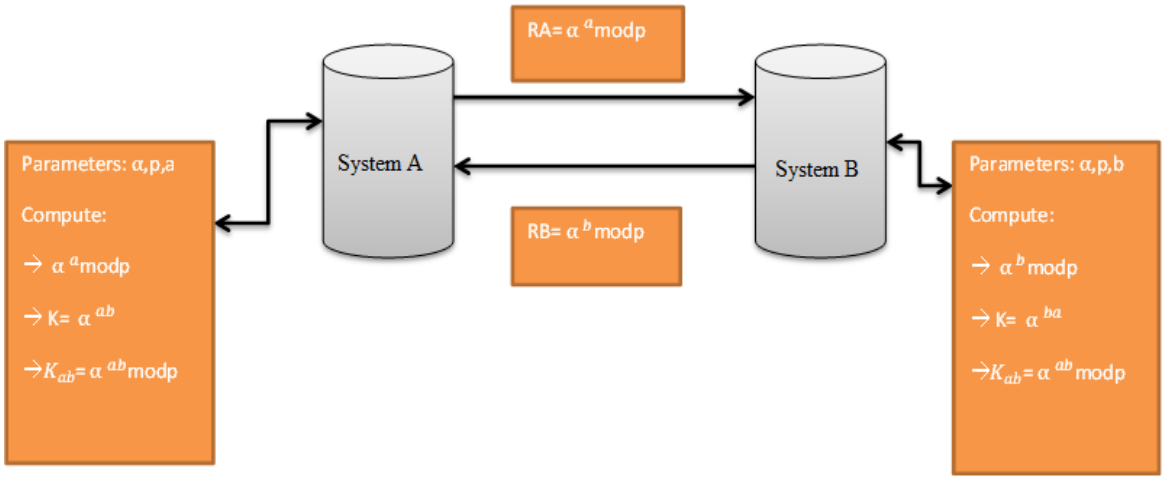

Figure 3.1: Schematic of Diffie-Hellman key exchange method

3.2 Security

It is very easy to verify that both systems A and B end up with the same secret key. The bigger questions is why this protocol is secure? In other words, why an attacker can not figure out the same shared key that both systems A and B have computed themselves ? The attacker or eavesdropper sees α and p which are fixed forever and everybody knows what they are. He also sees the value that system A sends to system B namely as RA and he sees the value that system B sends to system A namely as RB. The question is can attacker compute α^{ab} just given these four values. The solution of this problem is that the attacker has to solve the Discrete Logarithm Problem (DLP) or the Diffie-Hellman

Problem (DHP), and then he can find random generated numbers “a” and “b” after which is very easy for attacker to calculate α^{ab}[7].

The DLP is the problem of solving “x” if $y \equiv \alpha^{x}(mod\ p)$, wih y, α and p given. Two solutions are known but there is no any algorithm that can succeed this in an acceptable amount of time. First solution is known as Brute Force, which is not an efficient solution, since it needs “p” steps to compute the solution and in every trial needs a fair amount of work. The other solution is Shank’s algorithm, which is more efficient than Brute Force, but is not practical for high values of “p”, since it needs $\sqrt{p}$*log(p) steps for calculation. If it is supposes that “p” is selected as 170141183460469231731687303715884105727 than the number of steps would be approximately $1.14824x10^{21}$. Even if Google’s computers were used, which are calculated to perform 300 trillion calculations per second, it would last nearly 5 years to solve[7].

3.3 Design and implementation

In this thesis the Diffie-Hellman key exchange algorithm is designed with Verilog Hardware Description Language in Xilinx ISE 14.1 and is both implemented and tested in a Spartan-6 FPGA. The diffie-hellman key exchange is designed for 128-bit key. In its design for multiplication, a multiplier module with “Karatsuba” algorithm, and a modulo module, where a Ripple-Carry adder is used as a sub-module, are designed first. Then these two designed modules are used as sub-modules in a module where the exponentation operation is performed by using “Square and Multiply” algorithm, and as a result a unit which performs exponentation and modulo operation is obtained.

3.3.1 Verilog hardware description language

Verilog Hardware Description Language (Verilog HDL) is a formal notation designed by Xilinx for use in all stages of th establishment of an electronic system. Since it is both machine and human readable, it backs the development, verification, synthesis and testing of hardware[7].Verilog HDL serves as a tool with great power in digital system design since it offers to a designer the oppurtinity of simulation of the design, where the designer obtains the case to make critical decisions about his/her design. Designer also

can make analysis how his/her design will behave if it woulde be designed synchronously or assynchronously by just adding using command. As it is mentioned above the readability of the Verilog HDL makes it very attractive to engineers and useful. All these operations such as synthesis, simulations, verifications etc. are compiled in a tool designed from Xilinx known as Xilinx ISE.

3.3.2 Karatsuba multiplication

Multiplying two n digit numbers requires n^2 multiplications, this is actually how peoples multiply two n digit numbers. Let's consider multiplying two 2-digit numbers 14 and 16. It is clear that the result is 224, but it is also clear that it is more difficult than multiplication of 10 and 16. Since when a number is multiplied by 10, just a zero is added to the multiplicand number, so in this case the result becomes 160. It is known that the multiplication of 14 and 16 is equal to multiplication of 10 and 16 and multiplication of 4 and 16 where the result will be 160+24=224. As it can be seen for multiplication of two 2-digit numbers four multiplication and some addition is required. On the other hand if it is required multiplication of two 10-digit number, it is very difficult to perform multiplication with classical way. In 1960, in a seminar one of the most famous russian mathematican Andrey Kolmogorov has stated that the multiplcation of two n-digit number cannot be performed with less than n^2 multiplication. On the other hand just one week later a young student with name Anatolii Alexeevitch Karatsuba has demonstrated that two n-digit numbers can be multiplied with n^{log3} multiplications [8]. The proposition of this young student was as follow. Two numbers x and y can be represented in terms of x1, x2 and y1, y2 as x=x1 * B^m+x2 and y=y1*B^m+y2 where here B is the base of the numbers x and y and m is the half of the digit numbers. Certainly the multiplication of x and y becomes as the following product

$$x*y=(x1 * B^m+x2) * (y1*B^m+y2) = x1*y1*B^{2m} + (x1*y2 + x2*y1)*B^m + x2y2.$$

So as it can be seen with four multiplication x*y is got. Karatsubla also came up with a great idea which has reduced the number of multiplication to three. The proposition was to calculate (x1*y2 + x2*y1) as follow.

$$(x1*y2 + x2*y1)=(x1+x2)*(y1+y2)-x1y1-x2y2$$

As a result wiht three multiplications the result of multiplication of x and y could be obtained.

As it can be seen, Karatsuba multiplication algorithm is much more faster and efficient than the standart multiplication algorithm. Karatsuba algorithm is not only used in for integer multiplication, it is used also for polynomial mutliplication. Karatsuba progresses multiplication process by substituing inital complexity of O(n^2) with O(n^{log3}) (O describes the limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions), where the case is described also in the diagram below[8].

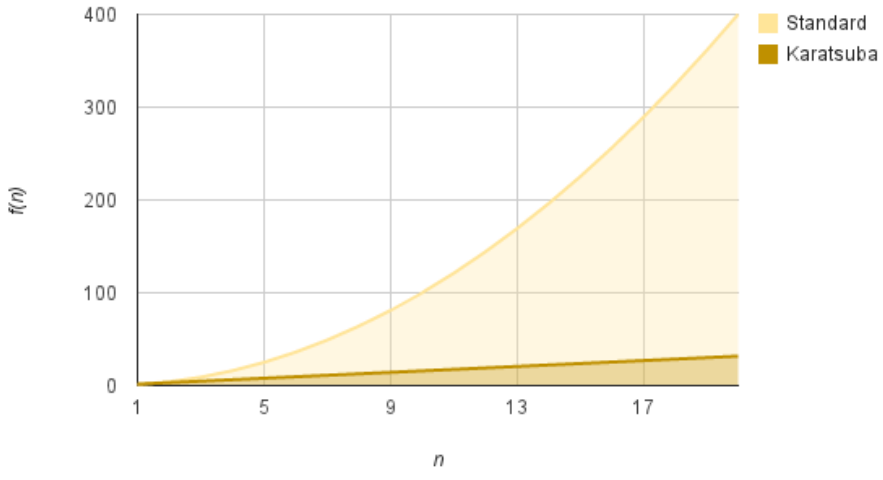

Figure 3.2: Complexity comparison of Standard and Karatsuba multiplication[8]

3.3.3 Design and implementation of karatsuba multiplication

Karatsuba Multiplication is designed as combination of both combinational and sequental logics.Combinational logic is a type of digital logic where output is a pure function of the present only. On the other hand sequential logic is a type of digital logic where output is not just function of present inputs but also it depends also on the past inputs so the past outputs. The module is designed for multiplication of two 128-bit numbers, so it will have two 128-bit inputs and one 256-bit output.

The combinational part of the design is the two multiplier module and the ripple-carry adder module. Two bit multiplier module by using standard multplication algorithm, multiplies two 2-bit numbers and produces 4-bit result. The ripple-carry adder is a module designed for addition two n-bit number and produce n+1-bit number as output. The ripple-carry adder is composed of full-adders and it is named as ripple-carry since each carry bit is rippled to the next full-adder.

The other parts of the design are sequentially designed. Two separate Finite State Machine (FSM) are designed for the design. Finite State Machine (FSM) or simply state machine is a mathematical model of computation used to design sequential logic. These two state machines are described by using Algorithmic State Machine (ASM) diagrams. ASM method is a method for designing finite state machines. One state machine is designed for multiplication of two 4-bit numbers. In this module two-bit multiplier and n-bit ripple-carry adder are used as sub-module.Since the module is designed in sequentially manner, in each clock cycle, operations that are in line will be performed.

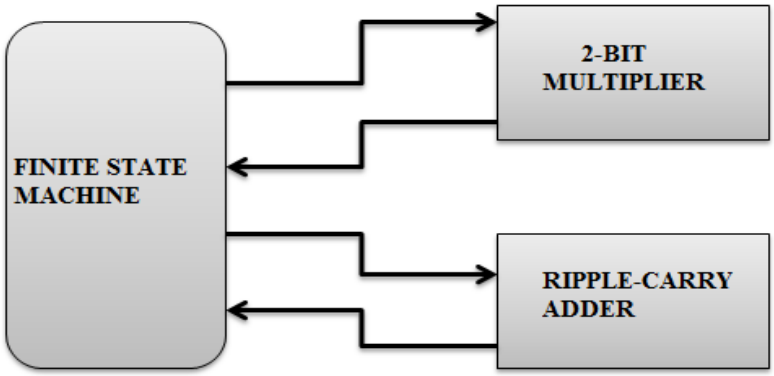

Figure 3.3: Simplified model of 4-bit Karatsuba multiplier

The 4-bit multiplier is designed to perform multiplication as Karatsuba algorithm. As it is described in previous section, in Karatsuba multiplication, both multiplier and multiplicand are divided into higher and lower parts, and then after some multiplication and addition operations the product is obtained. In this case, since the multiplier and multiplicand are 4-bit, their higher and lower parts will be 2-bit, and for this reason the designed 2-bit multiplier will be used for multiplication of these parts. For addition the ripple-carry adder module is used. The order of these operations is controlled by the state machine, so in a way it will serve as controller. In each state, the operation which is in the seqeunce is performed by sending the values that are required to be calculated to these modules, and the results are returned and stored in different registers. In final state the final result is assigned to the product output and “done” output will be assigned to 1 since it will notify that the calculation is over. The 4-bit mutliplier module has also two extra inputs, namely “start” and “reset”, where start input is for enabling multiplication operation and reset is for clearing the outputs and to start the operation from the beginning.

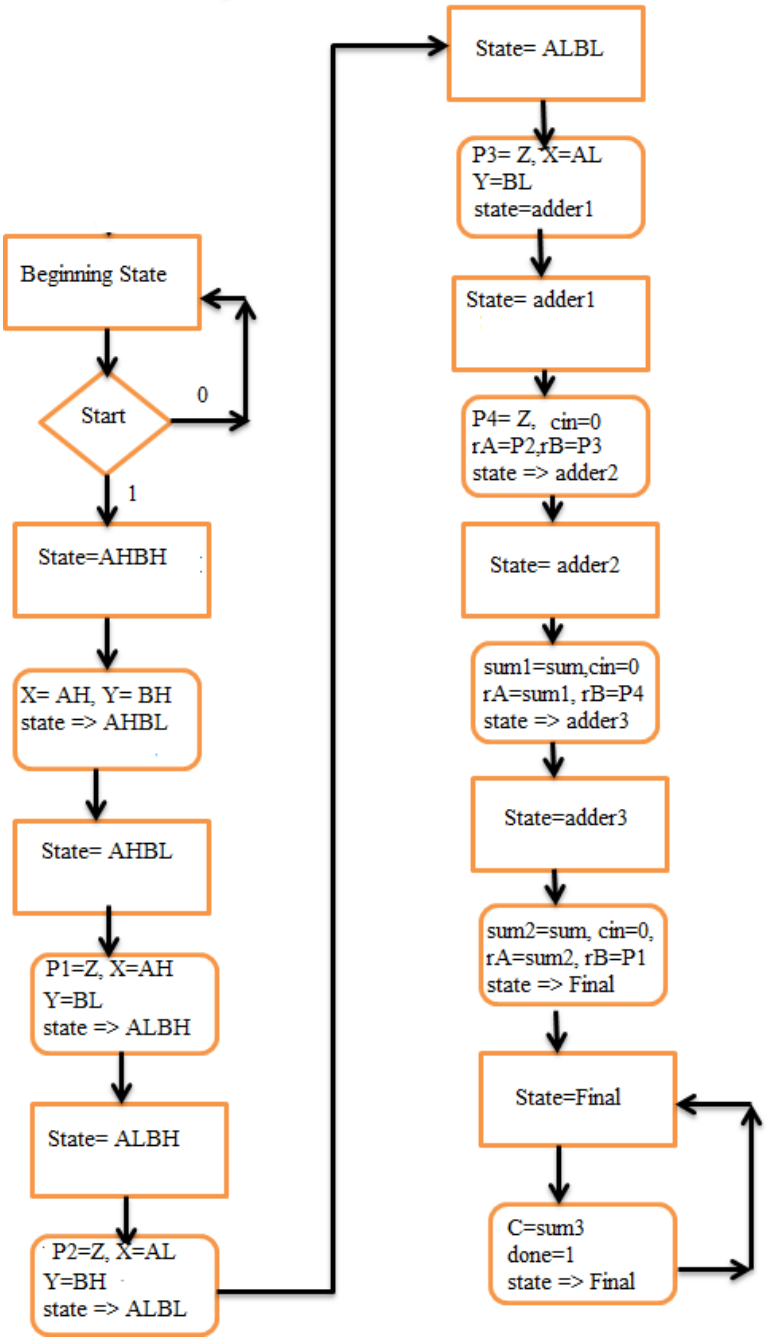

Figure 3.4: ASM diagram of 4-bit Karatsuba multiplier

The described designs above can be considered as substructure of main Karatsuba multiplication module. In this thesis, Karatsuba multiplier is designed for 128-bit multiplication, so for multiplication a 64-bit Karatsuba multiplier is required. This is achieved by first by designing 8-bit, then 16-bit, 32-bit multiplier and finally 64-bit Karatsuba multiplier. So each multiplier is constructed from other lower multipliers, for example 8-bit multiplier will be constructed from 4-bit Karatsuba multiplier and a ripple-carry adder. Also 16-bit Karatsuba multiplier is constructed from 8-bit Karatsuba multiplier and a ripple-carry adder and this continues to higher bit multipliers. The state machine of the 8-bit Karatsuba multiplier is same as the 128-bit Karatsuba multiplier. The difference from the state machine used in 4-bit Karatsuba multiplier is that in this case the multiplication is going to be calculated from a sequential module, so the product requires a period of time for calculation. So for getting the calculated result, extra states are added for getting the result from the 4-bit Karatsuba multiplier. During transition to these states first the "done" output is controlled if the calculation is ready or not. If the "done" output is 1 then it gets the product and performs the operations that must be performed in that state and passes to the next state otherwise it waits for calculation. Also after every calculation the reset input of sub-modules becomes to 1, in order to make system ready for new calculation.

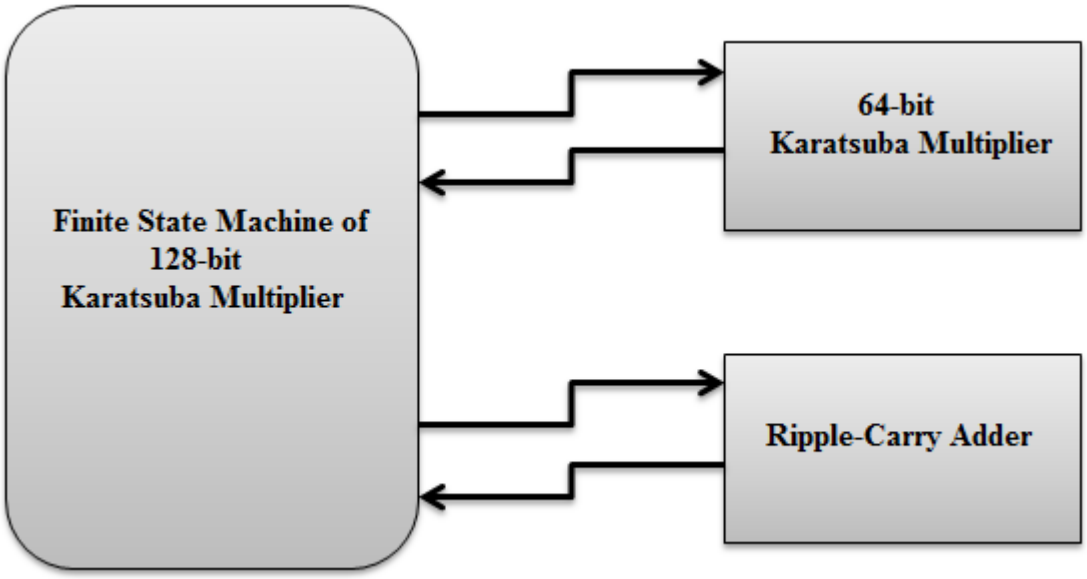

Figure 3.5: Simplified model of 128-bit Karatsuba multiplier

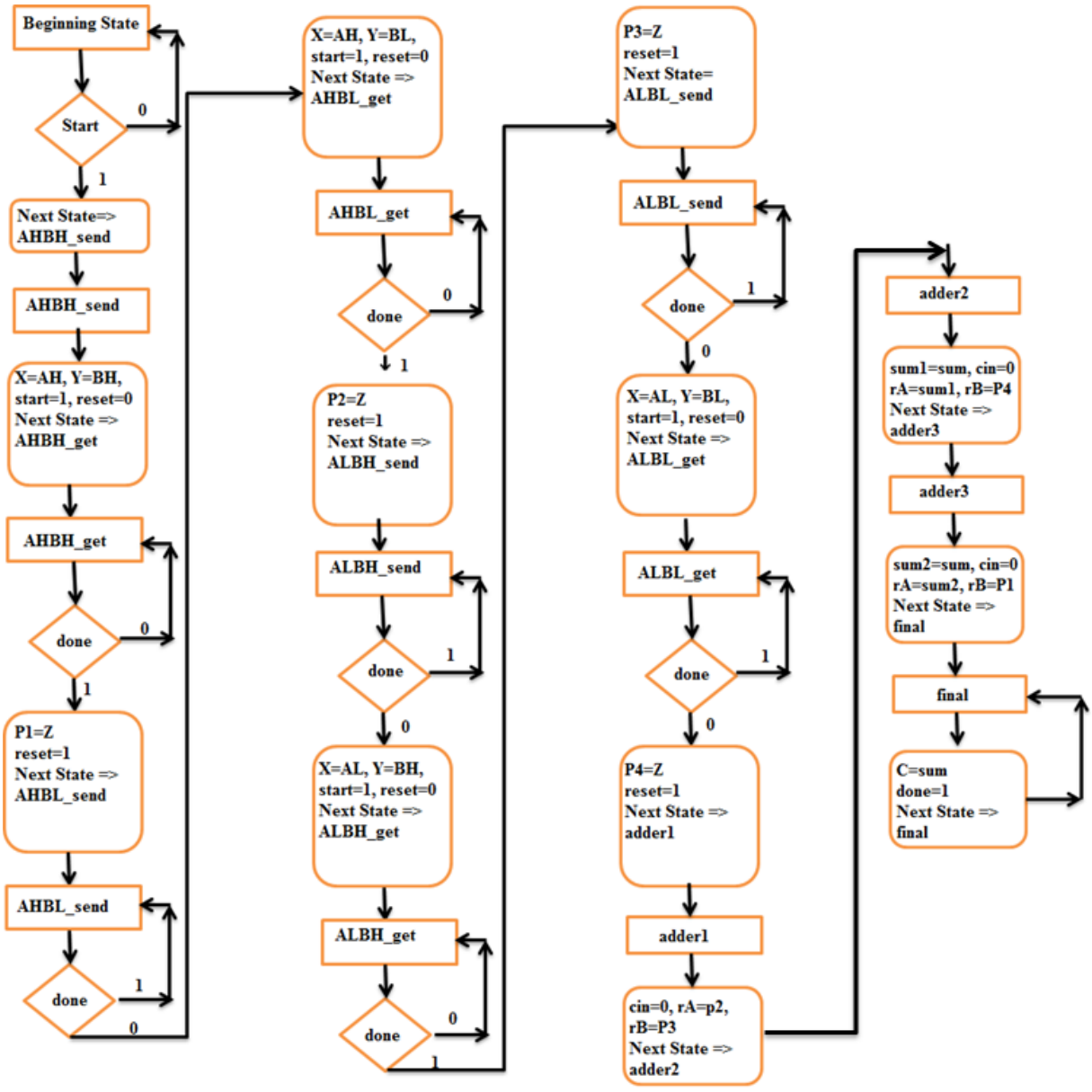

Figure 3.6: ASM diagram of 128-bit Karatsuba Multiplier

The RTL Schematic and the Design Summary of the 128-bit Karatsuba Multiplier obtained from Xilinx ISE 14.1 are as shown in the images below.

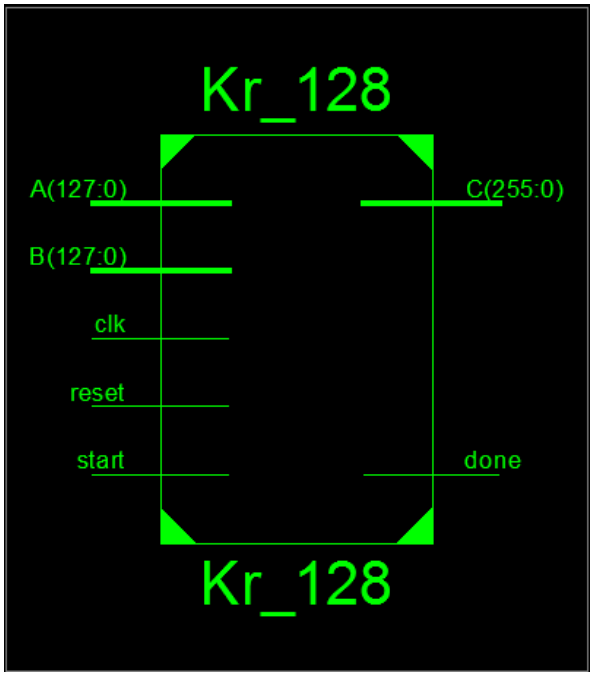

Figure 3.7: RTL schematic of 128-bit Karatsuba Multiplier

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	4010	54576	7%
Number of Slice LUTs	3003	27288	11%
Number of fully used LUT-FF pairs	1795	5218	34%
Number of bonded IOBs	516	218	236%
Number of BUFG/BUFGCTRLs	1	16	6%

Figure 3.8: Design Summary of 128-bit Karatsuba Multiplier

3.3.4 Square and multiply algorithm

The last step of Diffie-Hellman key exchange protocol design was designing the module which was going to perform exponentiation and module operation The exponentiation operation is performed by using Square and Multiply algorithm but the algorithm is modified in such a way it performs also modulo operation. So as a result $a^b modp$ is obtained. Square and Multiply algorithm or referred as exponentiation by squaring is a common technique used for fast computation large integer powers of a number [9]. The modification on the algorithm since in Diffie-Hellman key exchange protocol the secret key is shared between systems over α^{ab}modp computation.

In square and multiply are many different methods such as basic method, 2^k-ary method, sliding window method, Montogomery's ladder technique and Fixed base exponent. In this thesis 2^k-ary method is used. 2^k-ary method is a method proposed by Brauer in1939 that calculates x^n after expanding the exponent in base of 2^k. The pseudo code of the modified version of the algorithm is shown below.

C= $A^B mod\ N$

B= ($b_k\ b_{k-1}$ …… $b_1\ b_0$)

C=1

for i=k-1 to 0

C=$C^2 modN$

If b_i=1

C= C.A modN

Figure 3.9: The pseudo code of the Square and multiply algorithm

3.3.5 Design and implementation of square and multiply algorithm

The square and multiply algorithm is also designed and implemented in Verilog Hardware Description Language in a Spartan-6 with two 128-bit inputs and one 128-bit output. By implementing the modified version of in a way the Diffie-Hellman key exchange protocol is also implemented. Since a module which performs $a^b modp$ operation will be obtained.

As it can be seen from the algorithm above to design this algorithm, modules that can perform multiplication and modulo operations are required. The multiplication process is performed by using 128-bit Karatsuba multiplier since inputs are 128-bits and the modulo operation is performed by using the ripple-carry adder, since the modulo operation is simply a subtraction process which continues until the minimum one is obtained. Modulo module is a sequential logic design where the finite state machine controls if the result has arrived to the minimum or not.

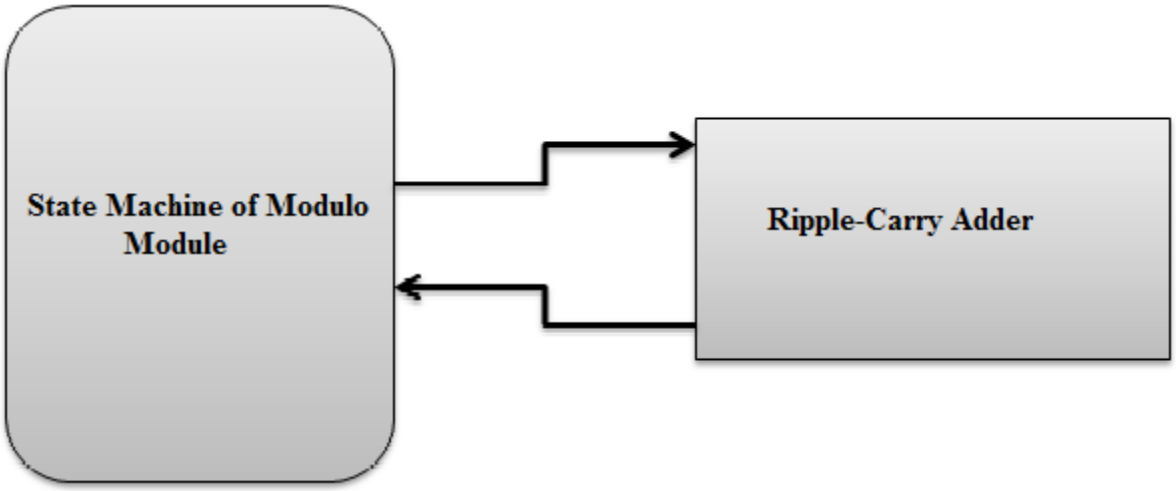

Figure 3.10: Simplified model of Square and Multiply module

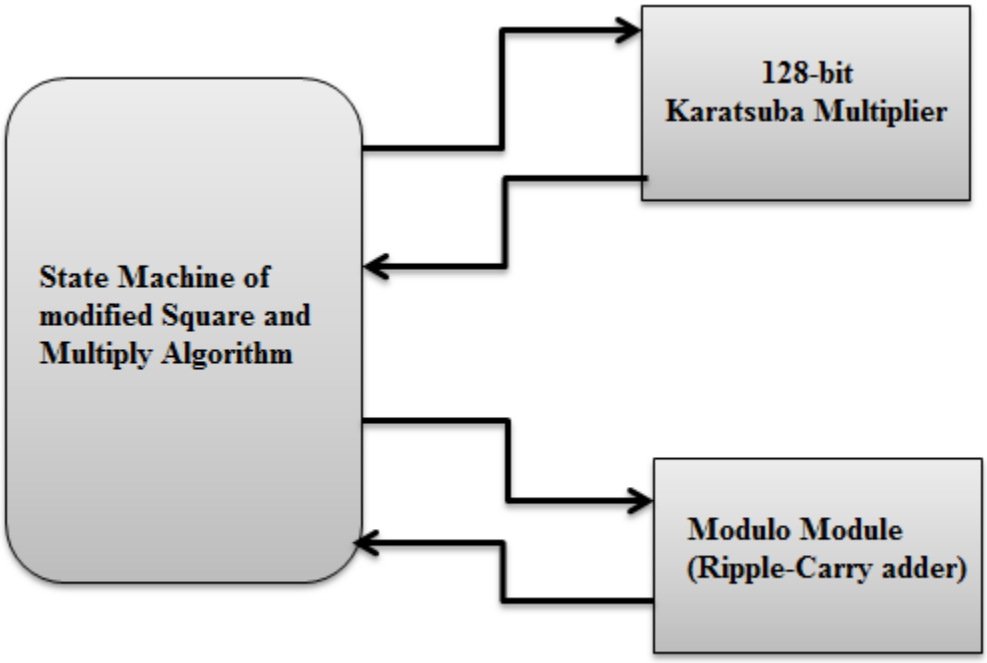

Figure 3.11: Simplified model of modified Square and Multiply module

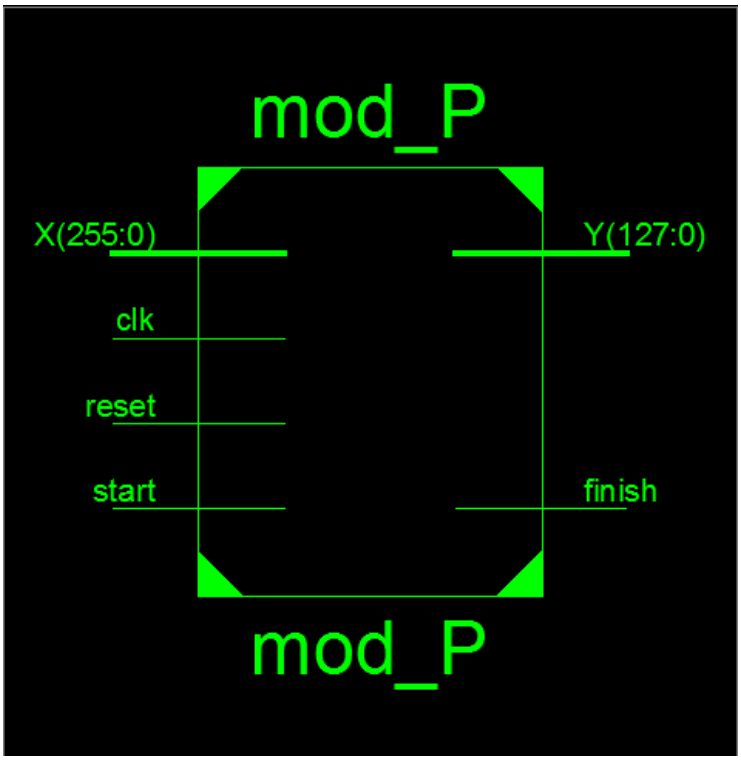

Figure 3.11 : RTL schematic of Modulo module [10]

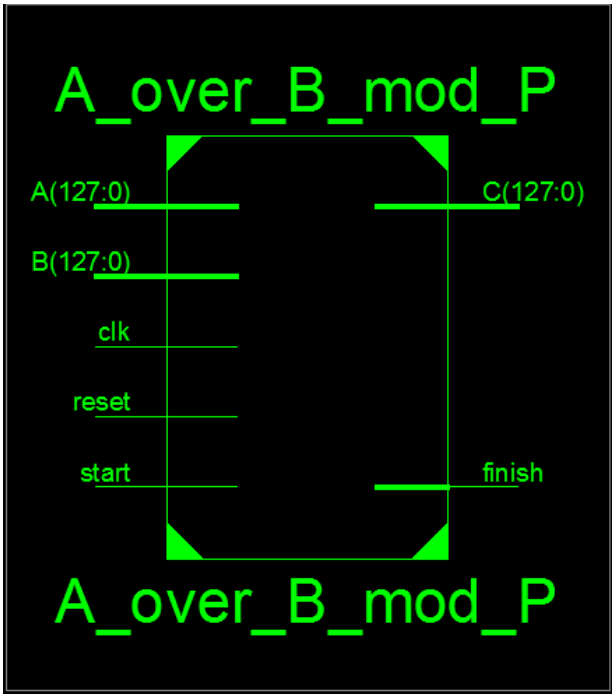

Figure 3.12 : RTL schematic of modified Square and Multiply module[10]

4. ADVANCED ENCRYPTION ALGORITHM (AES)

4.1 Introduction

In 1997, National Institute of Standards and Technology (NIST) requested proposals for new block ciphers and it received 15 submissions a year later, in 1998. And finally in 2000 it adopted the cipher called Rijndael as Advanced Encryption Standard (AES). AES algorithm is a symmetric block cipher that can perform both encryption and decryption process. AES was designed in Belgium and its input, output, round key and state length is 128 bits so can encrypt or decrypt 128 bits data and can use three possible key sizes, a128 bits, 192 bits and 256 bits[2]. The assumption is that a larger key size is, the more secure the block cipher is as a suit random permutation. On the other hand since many round operations are inclusive in the operation of this block cipher, the largest key means the slowest block cipher. So AES with 128 bit key size is the fastest block cipher than the others.

The AES is constructed according Substitution-Permutation Network mechanism, which is a mechanism where all bits are changed in every round[2]. The operation mechanism of the Substitution-Permutation Network is as in each round the current state is exored with round key, here round key states for the values derived from the cipher key using the key expansion routine and state states for two dimensional arrays of bytes, then according to the substitution tables blocks of states are replaced with other blocks and then bits are permitted and shuffled around. This process goes until it reaches the final round and then the output is obtained[6].

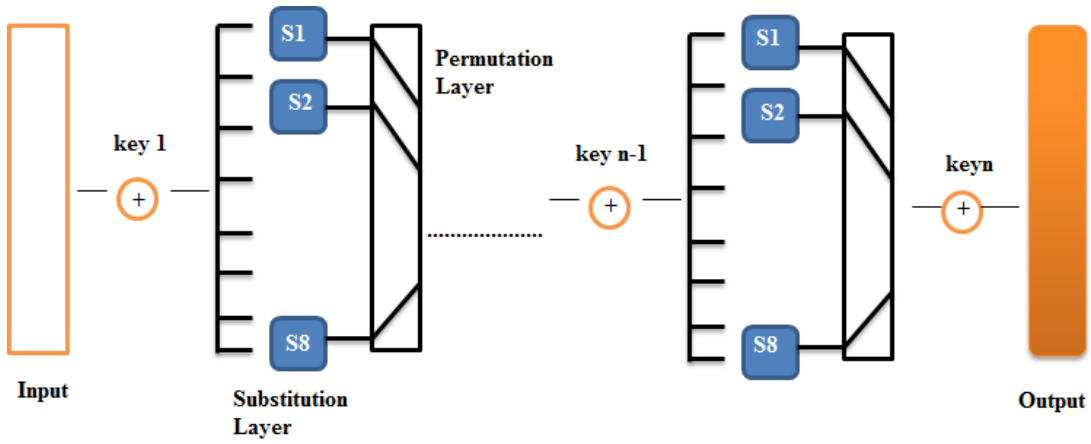

Figure 4.1: Simplified model of Substitution-Permutation Network

The main point in this construction is that every stage is reversible so for decryption the output would be taken and is applied to each step of the network as described above in the reverse order. So inversion in the Substitution-Permutation Network is simply done by just applying all stages in the reverse order. This is the main difference between the AES and Data Encryption Standard (DES), since in DES the substitution tables are nor reversible, so the encrypted data cannot be decrypted as in the AES case just by applying operations in the reverse order[12].

4.2 Specifications of AES 128 encryption

In this thesis AES with 128-bit key size is used for encryption and decryption, since the shared secret key by using Diffie-Hellman key exchange protocol is 128-bit.AES operates over 128-bits block, which means the length of input, output, round key and state is 128-bit. Those 128-bits are written as 4x4 matrix, where each cell of the matrix contain 8-bits or in the other words 1-byte and then operation starts with first round. As described in the previous section first with the round key is exored and then some functions that involves substitution and permutations and other operations are applied on the state. The applied functions are named as SubByte, ShiftRow and MixColumn. These functions that are applied are reversible, otherwise the decryption could not be occurred as described in the previous section. These operations are repeated ten times but intestinally in the last round function named as MixColumn is not applied and then

finally it is exored with last round key and the output are obtained. The output is in the form of 4x4 matrix since in all phases this form is kept.

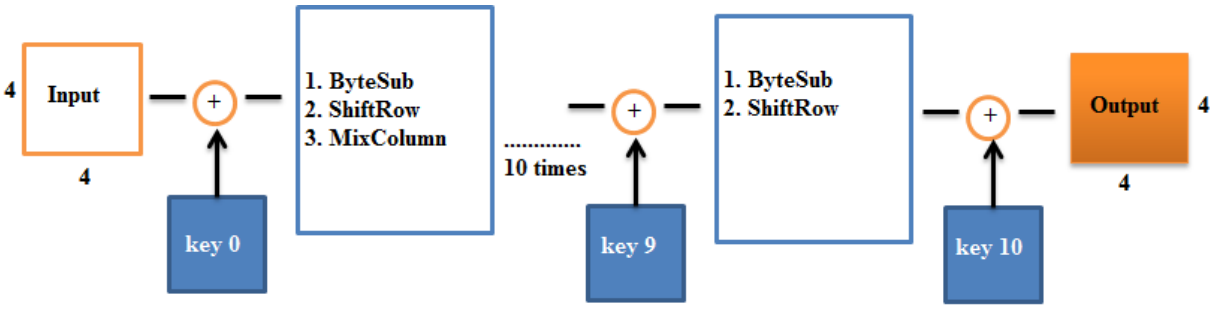

Figure 4.2: Schematic of AES encryption process

4.2.1 SubByte transformation

SubByte function is a non-linear byte replacement which operates on every byte of State according to a substitution table, which is invertible, namely as S-box. This S-box is composed by using two transformations.

→ For mapping element {0,0} to itself perform multiplicative inverse operation in the finite field GF(2^8).

→ The affine transformation shown below is applied over GF(2):

$$b_i^{'} = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

where i changes from 0 to 8, b_i is the i^{th} bit of the byte and c_i is the i^{th} bit of byte c with the value {63}. b_i' shows the new value of the i^{th} bit of the byte[2].

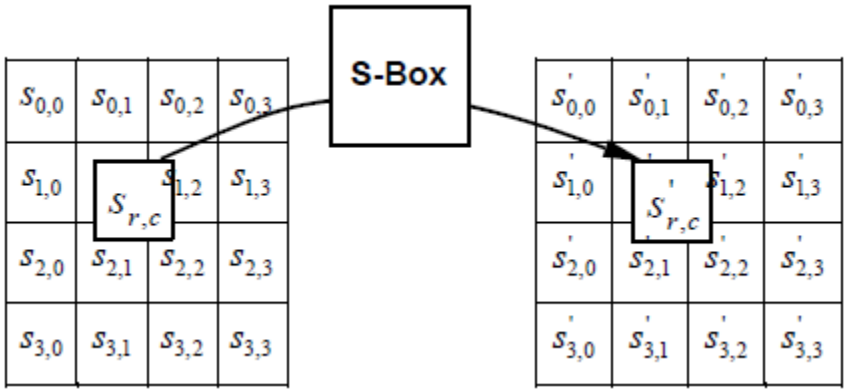

Figure 4.3: Operation of SubBytes function [2]

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 4.4: Hexadecimal form of S-box[2]

4.2.2 ShiftRows transformation

This transformation operates on the rows of the state, where the bytes in the last three rows are shifted cyclically by a certain offset[11]. As it can be understood thar only the first row is not shifted. ShiftRow functions shifts the bytes in the row according to the relation below:

$$S'_{r,c} = S_{r,(c+shift(r,Nb))modNb}$$

The shift(r,Nb) function is for swapping the bytes in the highest position with the lowest one.

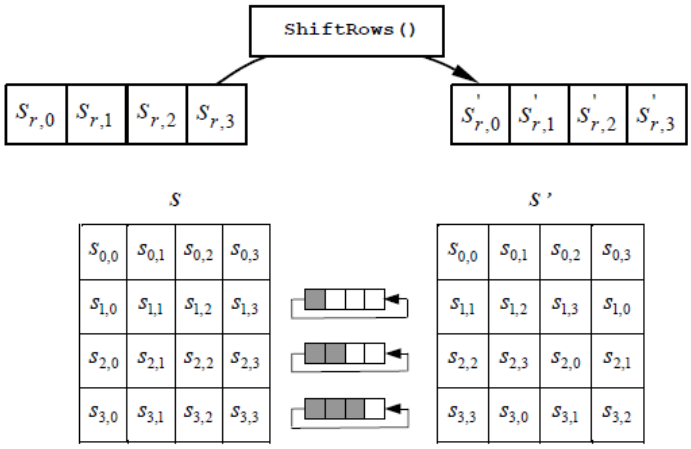

Figure 4.5: The operation of ShiftRow function[2]

4.2.3 MixColumns transformation

In this transformation each column is treated as a four term polynomial and investigated as polynomial over GF(2^8) and is multiplied with a fixed polynomial a(x) which is given as $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. So the new column values can be computed according to the relations below[2].

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

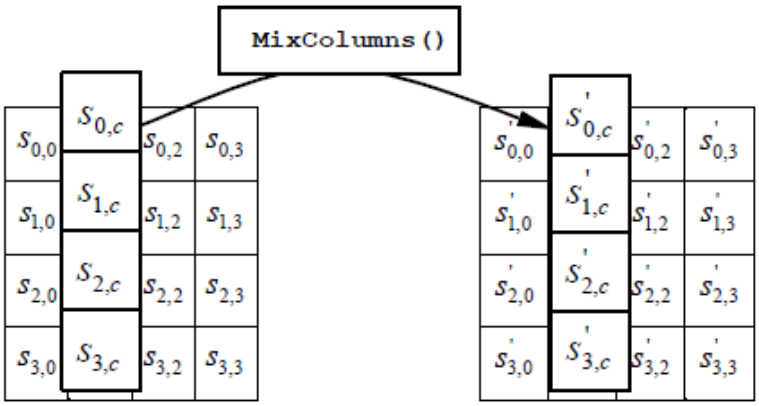

Figure 4.6: The operation of MixColumns function[2]

4.2.4 AddRoundKey() transformation

In this transformation the round key is added to the state by the means of the XOR operation. Every round key includes Nb words, that are each added into the columns of the state. The relation of the transformation is as follow[2].

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*Nb+c}] \quad \text{for } 0 \le c < \boldsymbol{Nb}$$

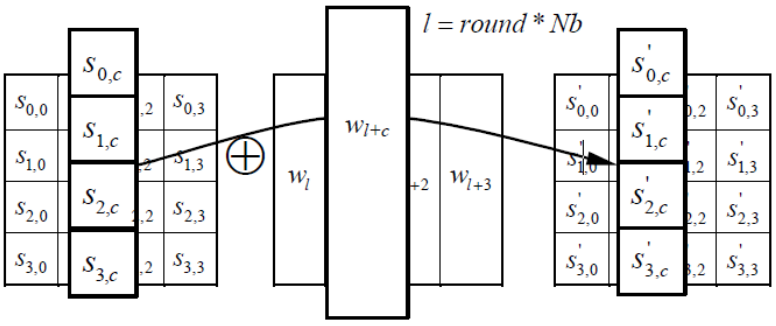

Figure 4.7: The operation of AddRoundKey() function[2]

4.3 Key expansion

Key expansion is the process in the AES algorithm, in which the key schedule is derived by using the cipher key. Totally Nb(Nr+1) words are generated in the key expansion process . In the key expansion algorithm initially a set of Nb words is required and in every Nr rounds Nb words of key data are required [2].

The pseudo code of the key expansion algorithm is as follow.

```
KeyExpansion(byte CipherKey[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
word temp

i = 0
while (i < Nk)
w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
i = i+1
end while

i = Nk
while (i < Nb * (Nr+1)]
temp = w[i-1]

if (i mod Nk = 0)
temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
else if (Nk > 6 and i mod Nk = 4)
temp = SubWord(temp)
end if

w[i] = w[i-Nk] xor temp
i = i + 1
end while
end
```

Figure 4.8: The pseudo code of the Key Expansion algorithm [2]

In the pseudo code, the array w[i] represents the round keys that are derived from the Key Expansion process and N_k represent the length of the cipher key. The SubWord() function implements the same S-box substitution to each 4-bytes in the word. In the RotWord() function a word $[a_0, a_1, a_2, a_3]$ is taken as input and it is returned as

$[a_1, a_2, a_3, a_0]$ in the output. The round constant word array, Rcon[i], is consist of 32 bit value given by [x^{i-1}, {00},{00},{00}] [2].

4.4 Specifications of AES128 decryption

In the decryption process of the AES algorithm the same steps as in the encryption prcess will be followed, but in reverse order. Since the transformations used in the encryption process are reversible, this property brings out a great ease for decryption process. The transformations that are used in the decryption process are named as InvShiftRows(), InvSubBytes (), InvMixColumns() and AddRoundKey(). In the decryption case, the output generated from the encryption process is considered as input, and it undergoes first the AddRoundKey(), where the current state goes through a XOR operation with the round key. After ten times the state undergoes a transformation process, where the applied transformations are InvShiftRows(), InvSubBytes(), AddRoundKey() and InvColumns() respectively. Then in the final round the state undergoes a transformation process of InvShiftRows(), InvSubBytes() and AddRoundKey().

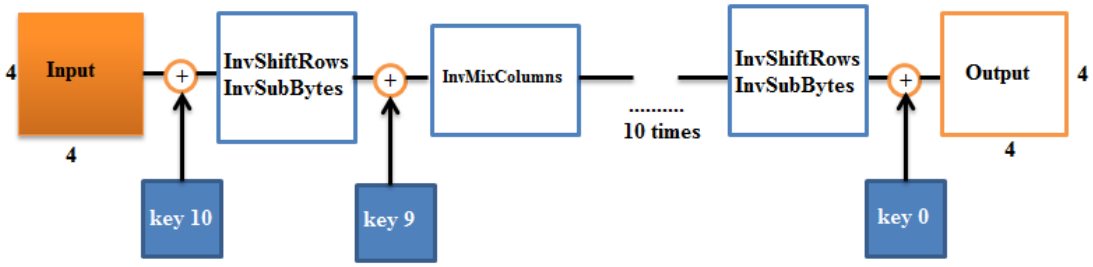

Figure 4.9: Schematic of AES decryption process

4.4.1 InvShiftRows() transformation

Simply is the inverse of the ShiftRows() transformation. The first row r=0 is not shifted, just the last three rows are cyclically shifted by Nb-shift(r,Nb) where the shift(r,Nb) is a

value which depends on the row number[11]. This transformation progresses as shown in the relation below:

$$s'_{r,(c+shift(r,Nb))modNb} = s_{r,c} \quad \text{for } 0<r<4 \text{ and } 0 \leq c \leq Nb$$

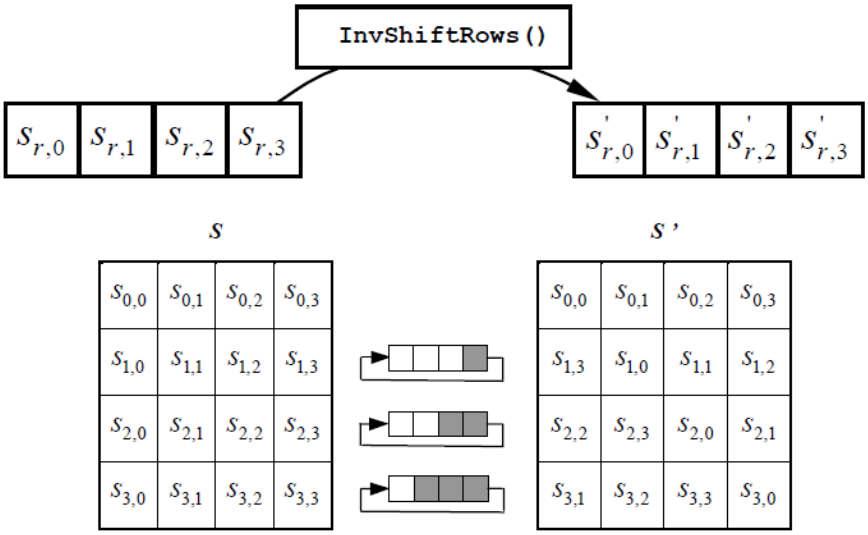

Figure 4.10: The operation of InvShiftRows() Transformation [2]

4.4.2 InvSubBytes() transformation

Simply as in the InvShiftRows() transformation case, also in this case this transformation is the inverse of the SubBytes() transformation. It applies the inverse S-box to each byte of the State. The inverse S-box is derived by applying the inverse affine transformation followed by taking the multiplicative inverse in GF(2^8) [2].

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure 4.11: The inverse S-box used in InvSubBytes() Transformation[2]

4.4.3 InvMixColumns() transformation

InvMixColumns() transformation is the inverse of the MixColumns() transformation. It acts on the State column-by-column and every column is considered as four term polynomial over GF(2^8) and multiplied with a fixed $a^{-1}(x)$ polynomial given as[2]:

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}\text{[11]}$$

As outcome of this transformation, the four bytes in the column are substituted by the following:

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c}) \quad \text{[2]}$$

4.5 Design and implementation of AES128 algorithm

The AES 128 algorithm used in this thesis is designed and implemented in Very high speed integrated circuit Hardware Description Language (VHDL). The designed AES 128 encryption and decryption modules are taken from the library of Embedded System Laboratory of Electrics-Electronics faculty of ITU. The encryption part was named as Cipher_serial_Table and decryption part was named as inv_cipher_serial_table. The

Register Transfer Level (RTL) schematic and the design summary of the designs are shown in the images below.

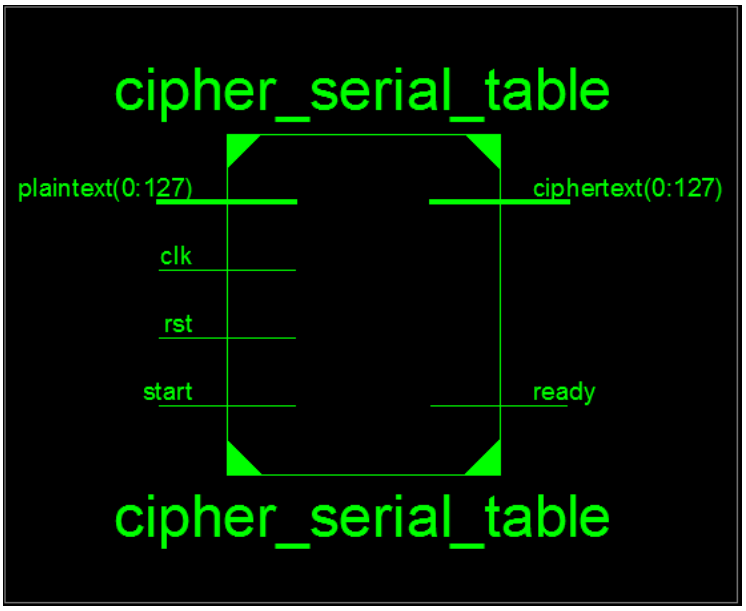

Figure 4.12: RTL schematic of cipher_serial_table module

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	540	54576	0%
Number of Slice LUTs	1466	27288	5%
Number of fully used LUT-FF pairs	455	1551	29%
Number of bonded IOBs	260	218	119%
Number of BUFG/BUFGCTRLs	1	16	6%

Figure 4.13: Design summary of cipher_serial_table module

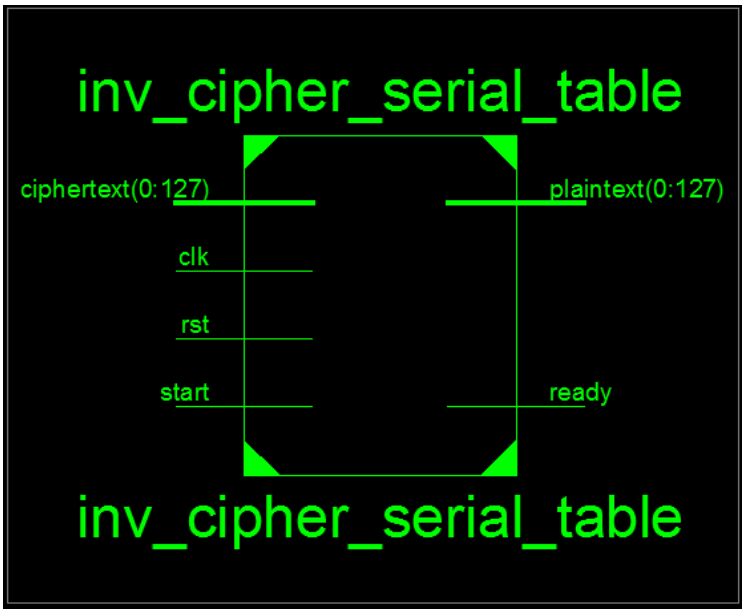

Figure 4.14: RTL schematic of inv_cipher_serial_table module

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	444	54576	0%
Number of Slice LUTs	1651	27288	6%
Number of fully used LUT-FF pairs	421	1674	25%
Number of bonded IOBs	260	218	119%
Number of BUFG/BUFGCTRLs	1	16	6%

Figure 4.15: Design summary of inv_cipher_serial_table module

5. MICROBLAZE

5.1 Overview

The designed modules are used as peripheral of a microprocessor, and the used microprocessor will undertake the role of controller by controlling these peripherals and implement a secure data communication protocol. The used microprocessor in this thesis is, the usable soft-core microprocessor of Xilinx Field Programmable Gate Arrays (FPGAs) with Xilinx Embedded Development Kit (EDK) software tool known as MicroBlaze[3].

The MicroBlaze is a virtual microprocessor, which is constructed by integration of blocks of code named as core located in the internal of a Xilinx FPGA [3]. The main point of the MicroBlaze is that, according to the specific needs of the design can be concluded and fit with special peripherals such as UART, Flash, General Purpose Input / Output and etc. According to its instruction-set architecture, the MicroBlaze process is optimized model of 32-bit Harward Reduce Instruction Set Computer (RISC) architecture for implementation in Xilinx FPGAs. Its apart 32-bit data and instruction buses operates at high speed to accomplish programs and access data from both chip and external memory at the same time [3].

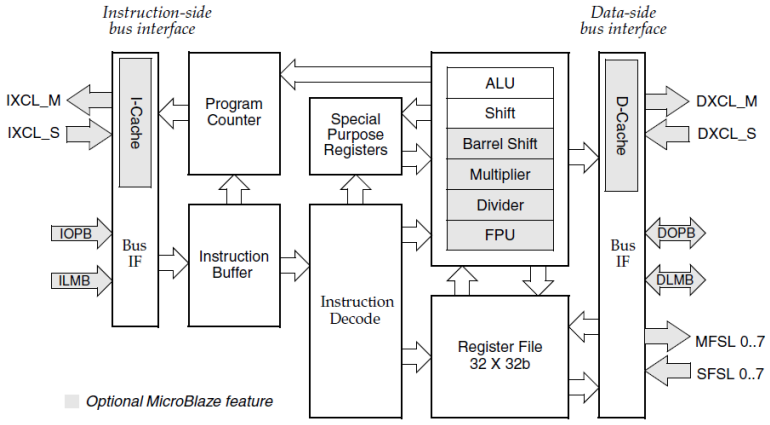

Figure 5.1: MicroBlaze core block diagram [12]

The pipeline mechanism of the MicroBlaze is parallel and is separated into three steps: Fetch, Decode and Execute. For each step is required one clock cycle to be supplemented, so for one instruction three clock cycles are required to be completed. Addition to the specific above, MicroBlaze contains 32-bit general-purpose registers, an Arithmetic Logic Unit (ALU), a shift unit and two levels of interrupt [3]. The MicroBlaze provides to the designer a great ease with being configurable according to required specific needs.

5.2 Xilinx Embedded Development Kit (EDK)

The Xilinx Embedded Development Kit (EDK) provides to the designer an environment where the designer can construct an embedded processor system with its all specifications which can be implemented in a Xilinx FPGA device. EDK is a fragment a tool, which is developed by Xilinx, which is required for implementation of a design into Xilinx programmable logic devices known as Integrated Software Environment (ISE®) Design Suite Embedded and System Edition [11]. Components of EDK are:

1- The Xilinx Platform Studio
2- The Software Development Kit (SDK)
3- Intellectual Property (IP) cores

The flow of design in EDK, in the other words the steps that designer has to follow are to start with an ISE project, then to add an embedded processor source to the ISE project. As the ISE components preparation is completed, then EDK start synthesizing the microprocessor hardware design, mapping it into a FPGA and as a final step of flow it generates and downloads the bitstream.

An embedded hardware platform is consisted of one or more processors, peripherals and memory blocks and interconnection via processor busses [13]. EDK as tool provides a great ease to a designer, since as a tool it deals with connection of peripherals and FPGA hardware designs, address mapping of the designed system, establishing communication protocol and with other interconnection, so to the designer is just left to focus on its

hardware and software designs. In the image below is shown the design flow diagram of a system which is designed by using the microcontroller located inside the FPGA. The stages shown in the image are created by EDK and are offer to the designer with a user interface and make the complex designs to be implemented more easily and in a shorter time.

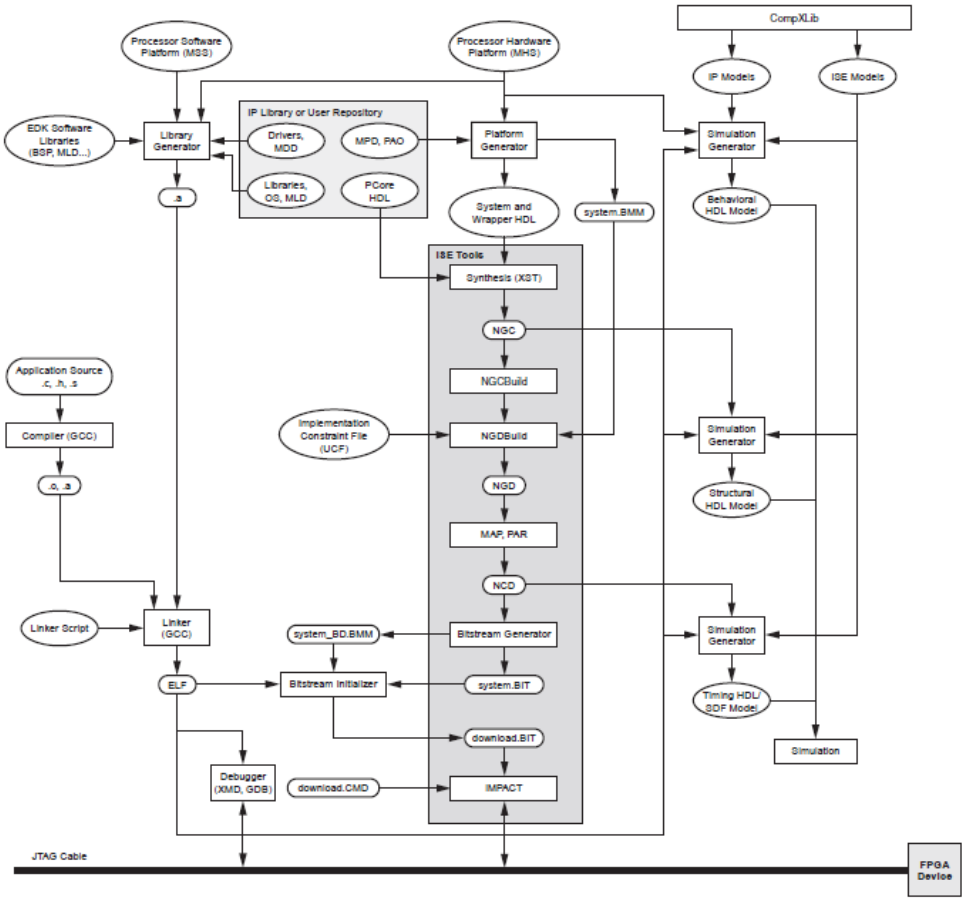

Figure 5.2: Embedded Development Kit (EDK) tools architecture [11]

In the previous section it is stated that MicroBlaze can be configured according to the needs. This case is implemented in EDK tool. EDK provides to the designer to create its own system base with Base System Builder (BSB). BSB provides to the designer to add its designed hardware on the MicroBaze as peripheral by using Intellectual Property (IP) cores, the connections of these systems is implemented from EDK.

5.2.1 Xilinx Platform Studio (XPS)

One of the components of EDK is Xilinx Platform Studio (XPS), which provides environment for constructing embedded processor systems based on MicroBlaze and PowerPC processors [11]. In the XPS the address mapping stage of the peripherals, which are connected to the MicroBlaze, is completed. Later the synthesizing and implementation stages of the design project are accomplished.

5.2.2 Xilinx Software Development Kit (SDK)

Another component of EDK is Xilinx Software Development Kit (SDK), which provides to the designer a development environment based on the Eclipse open-source standard for software application projects [13]. As a tool includes some feature that it includes[11]:

1- Feature-rich C/C++ code editor and compilation environment
2- Project management
3- Supports environment for group working
4- 4- Imports the hardware platform definitions generated by XPS

6. IMPLEMENTATION

The final stage in this thesis was collecting the designed unit, combining them in a embedded hardware platform, where in this thesis was MicroBlaze, and implement a secure data communication between these two systems. Initial step in the implementation the designed Diffie-Hellman Key exchange module and the AES 128 module, including both encryption and decryption parts, in the ISE platfrom are created as peripheral of the MicroBlaze in the Xilinx Platform Studio. Next, for both designs the definitions are defined and imported back. In the EDK platform, to the each hardware designs an IP is defined so that it can be added with other available IP's to a microprocessor based system. Then the defined system in the EDK platform is exported and launched in the SDK platform and automatically from the system the libraries of each IP is formed. Then in the SDK for controlling these hardware's a C/C++ base programming code is written and by the means of the SDK, the files with extensions "bit"and "elf" are combined and sent to FPGA.

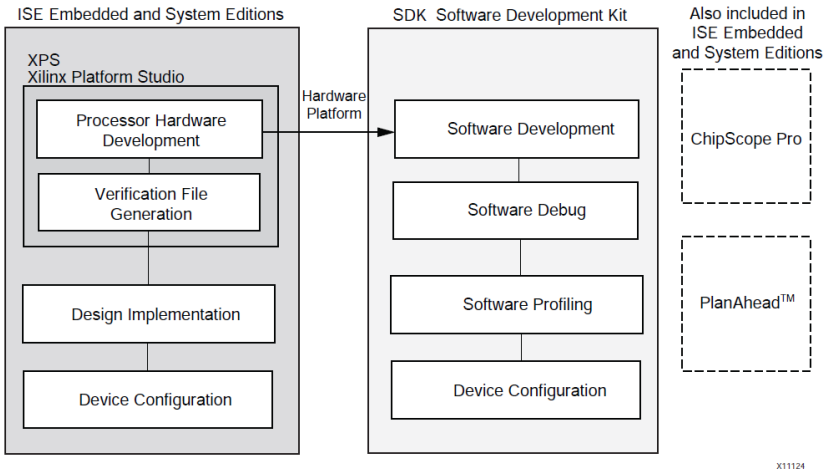

Figure 6.1: System Design Flow [11]

6.1 Optimization

During the design and the implementation of this thesis the Diffie-Hellman key exchange module was designed twice. The reason of designing twice the key exchange design was that the first design was too large to be fit in the Spartan-6 FPGA. So it has been faced with an area problem since it was being used ineffectively. The area was being used ineffectively because of the Karatsuba multiplier was designed as combinational digital system

Designing Karatsuba multiplier completely combinational was providing a great efficiency in the aspect of the operation speed but it was covering a large area. If the relation 3.1 of the Karatsuba Multiplication is taken in view, it can be easily seen for the multiplication of two 128-bit numbers it is required to perform four multiplication operation of two 64-bit number, so the multiplication operation will be implemented over the numbers that the size is half of the main multiplier or multiplicand number's size. This fact continues for 64-bit multiplication and for multiplication of other numbers with lower sizes. In the first design, as in the final case, the design has started with the design of a 2-bit combinational multiplier. This designed multiplier was performing the multiplication operation using standard multiplication method. Then by using a Ripple-Carry adder and the 2-bits multiplier a 4-bit Karatsuba multiplier was designed. The idea of the design was the same as in the optimized version of the 4-bit Karatsuba multiplier. But the difference is that in the first designed 4-bit Karatsuba multiplier four separate 2-bit multipliers and three separate Ripple-Carry adders are used as sub-module. The schematic of the design is illustrated in the Figure 6.2 below. On the other hand in the final designed 4-bit Karatsuba multiplier only one 2-bit multiplier and Ripple-Carry adder as sub-module. Later 8-bit Karatsuba multiplier is designed with the same idea by using separate four 4-bit Karatsuba multipliers and three Ripple-Carry adders as sub-module. The design has continued for the other higher order bit sizes until 128-bit Karatsuba multiplier with the idea described above. The schematic below describes also the structure of other designed Karatsuba multipliers of higher order bits. As a result with redesigning the Karatsuba multiplier as sequential digital system the required optimization in the covering was obtained.

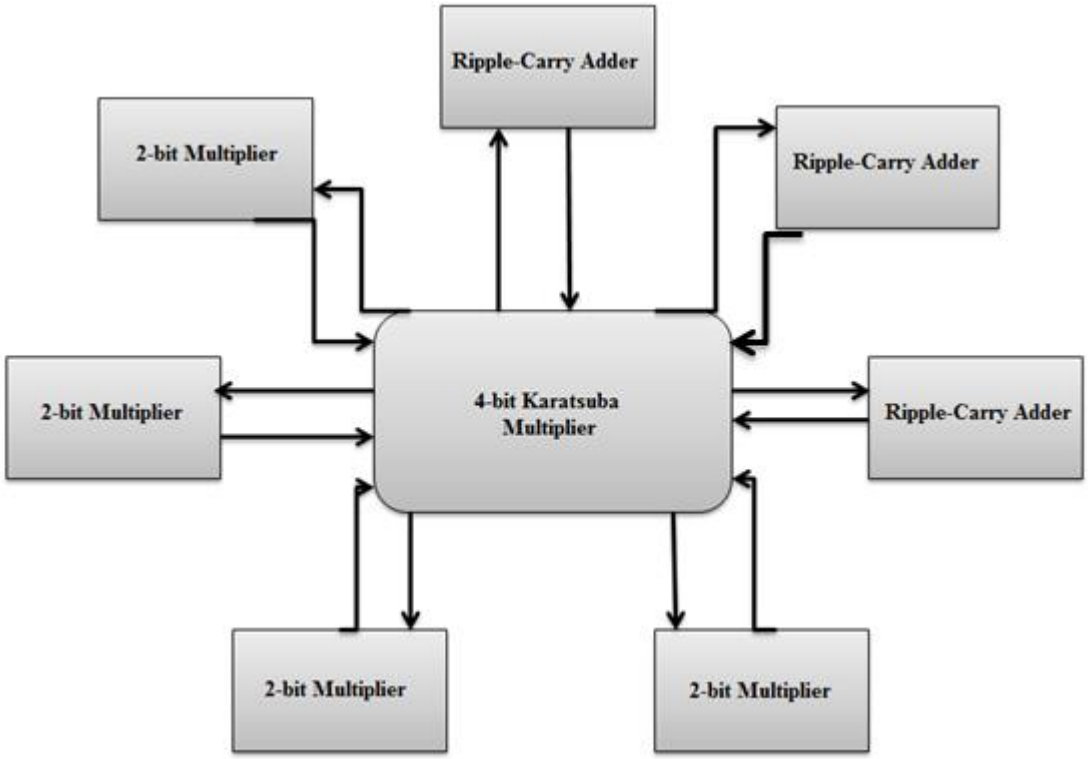

Figure 6.2: Schematic of the 4-bit Karatsuba Multiplier

An optimization is done also to the module which was computing the $A^B modp$ operation in the aspect of speed. In the first case the operation was being calculated as repeating (B-1) time $A \cdot A \cdot modp$ operation and this was causing to cover a large area in the device and was taking time for computation. The percentage of covering large area of the device was not changed after the optimization. The module was optimized by redesigning it according to the Square and Multiple algorithm and after optimization every calculation was calculated approximately for 128 clock cycle [10].

In the following the design summaries of 128-bit Karatsuba multiplier designed as combinational digital system, of first design and optimized version exponentiation and modulo operation calculation module.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	57195	27288	209%
Number of fully used LUT-FF pairs	0	57195	0%
Number of bonded IOBs	512	218	234%

Figure 6.3: Design summary of 128-bit Karatsuba multiplier designed as combinational digital system

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	520	54576	0%
Number of Slice LUTs	1939	27288	7%
Number of fully used LUT-FF pairs	507	1952	25%
Number of bonded IOBs	388	218	177%
Number of BUFG/BUFGCTRLs	1	16	6%

Figure 6.4: Design summary of first design of exponentiation and modulo operation module

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	3839	54576	7%
Number of Slice LUTs	2943	27288	10%
Number of fully used LUT-FF pairs	1822	4960	36%
Number of bonded IOBs	388	218	177%
Number of BUFG/BUFGCTRLs	1	16	6%

Figure 6.5: Design summary of Square and Multiply module

6.2 Simulations and results

After the optimization of the components of the Diffie-Hellman key exchange, it was the turn to simulate the optimized Diffie-Hellman key exchange module and the AES 128 encryption and decryption modules with the ISim tool of ISE. Below the simulation images of these three modules are shown. The result in the simulation of the Diffie-Hellman key exchange module is computed for the input values A=670, B=5 and p=53. In the AES128 encryption module the result displayed in the simulation image is the encrypted value of the input plaintext with the hexadecimal value 3243f6a8885a308d313198a2e03034. Lastly the result desplayed in the simulation image of the AES128 decryption module is decrypted value of the input cipher with the hexadecimal value 3925841d02dc09fbdc118597196a0b32.

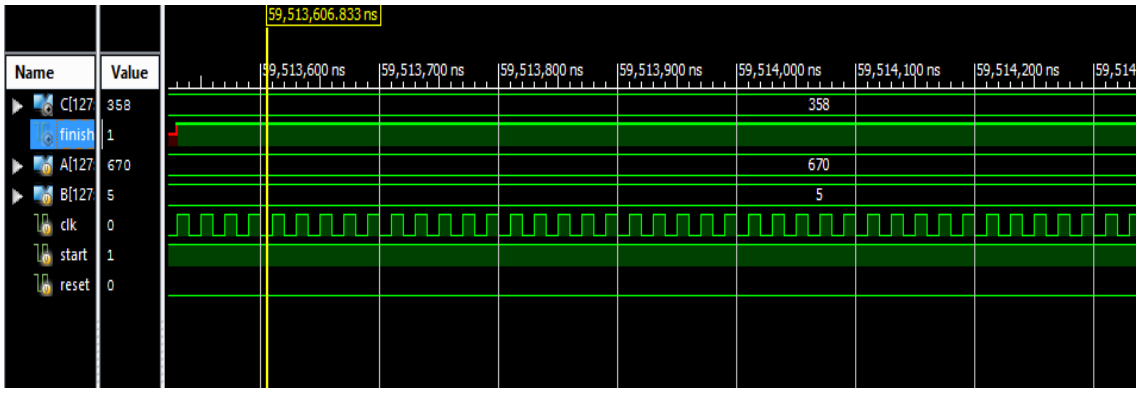

Figure 6.6: Simulation image of Diffie-Hellman key exchange module

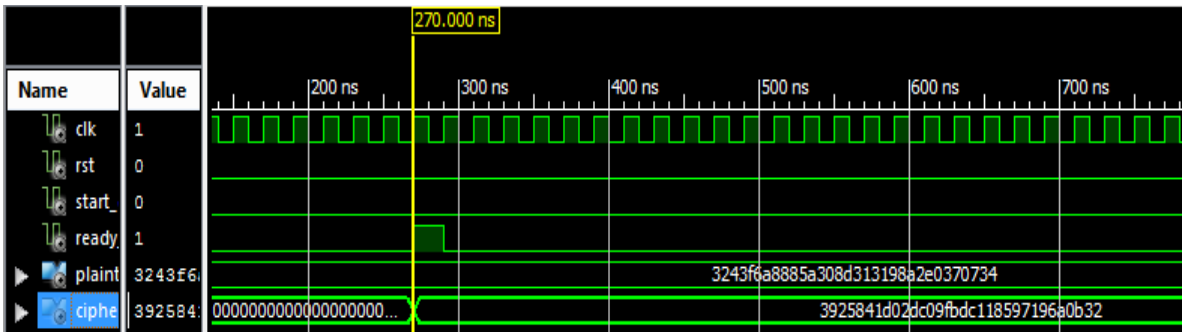

Figure 6.7: Simulation image of AES128 encryption module

The simulation results were the facts that ensured the designed hardware's work properly, so there was no any reason in the respect of right results which prevents to use the modules as peripheral of MicroBlaze. But considering the fact that the registers of the MicroBlaze were 32-bit, it was required to make also some changes on designed modules since the input size and output size of the modules were 128-bit. For this reason a module which in each clock cycle with rising edge of the enable input takes 32-bit input data and collects in a 128-bit register and then supplies as an input to the main module was designed. Also a module which takes 128-bit output data as input and in each clock cycle with the rising edge of the enable input gives the 32-bit portions of the data as output was designed. As a result, by connecting these buffer modules, the designed modules has become ready for using them as peripheral of MicroBlaze. The new schematic of the Diffie-Hellman key exchange module is illustrated in the image below.

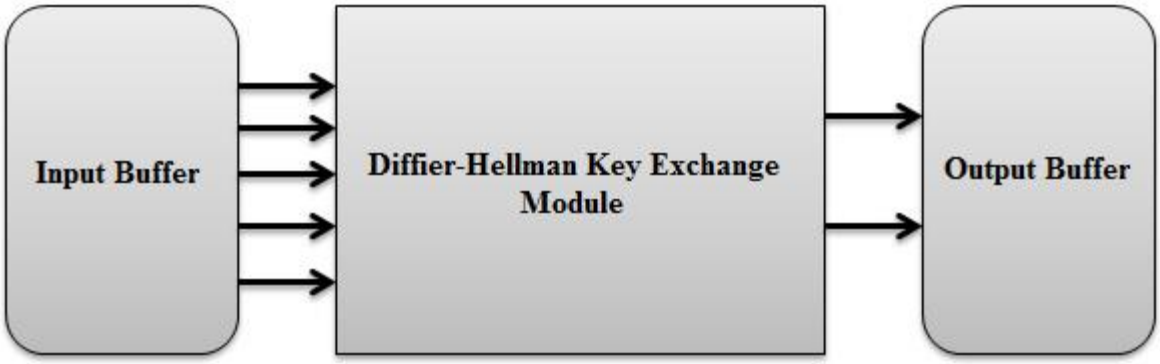

Figure 6.8: Simplified model of Diffie-Hellman key exchange module

After adding the designed modules as peripheral to MicroBlaze, the project was exported to the SDK for developing software for controlling the activation sequence of the peripherals, operations between the peripherals such as sending data to the other system and receiving data from the other system.

In the images below, the results obtained during implementation of the project are shown. Initially the same modules are added as peripheral into two separate MicroBlaze. Then for both MicroBlaze software is developed for performing a secure data communication between each other. First, for each MicroBlaze a secret key is obtained. Then by using RS232 UART communication protocol both systems have shared their secret key. After this communication process, each system has executed the last step in order to complete

the common shared key process as described in the Diffie-Hellman key exchange mechanism. Since a common shared key was obtained, it was the turn of encrypting a message by using AES128 encryption peripheral. As the encryption process was over, the encrypted message was sent to the other MicroBlaze by the means of the RS232 UART communication protocol. Finally, the received encrypted message by using the AES128 decryption peripheral is decrypted and a secure data communication in a insecure communication channel is implemented.

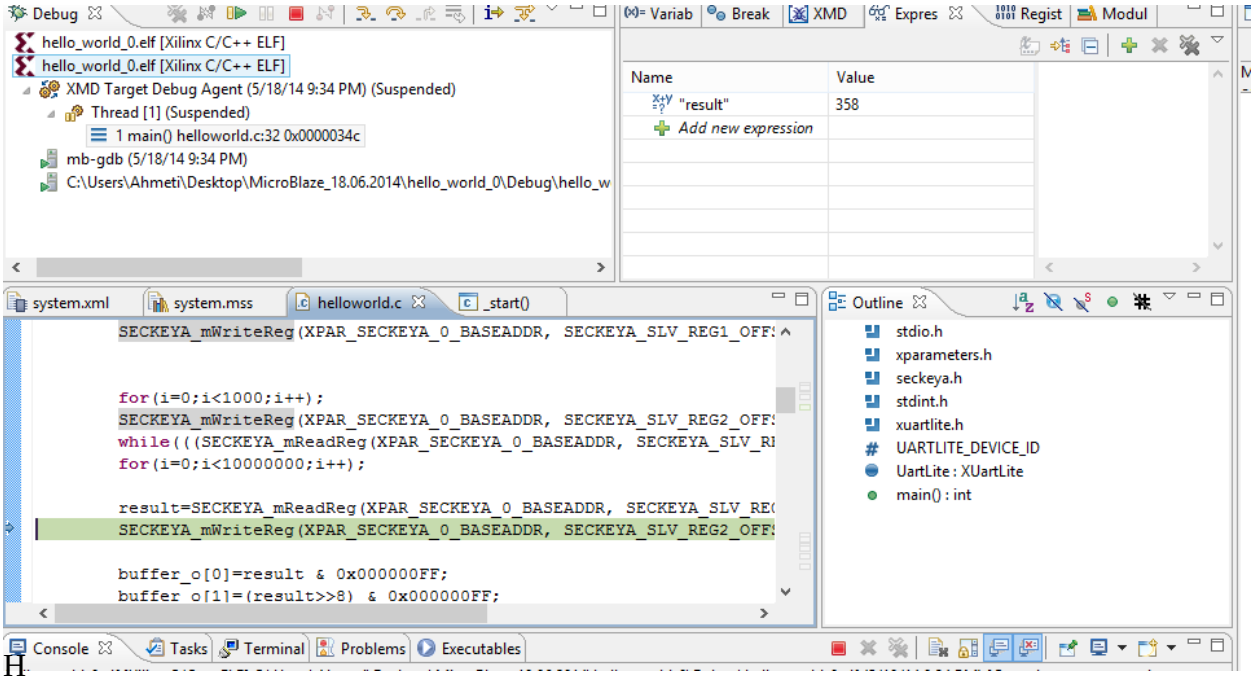

Figure 6.9: Process of producing secret key of first MicroBlaze

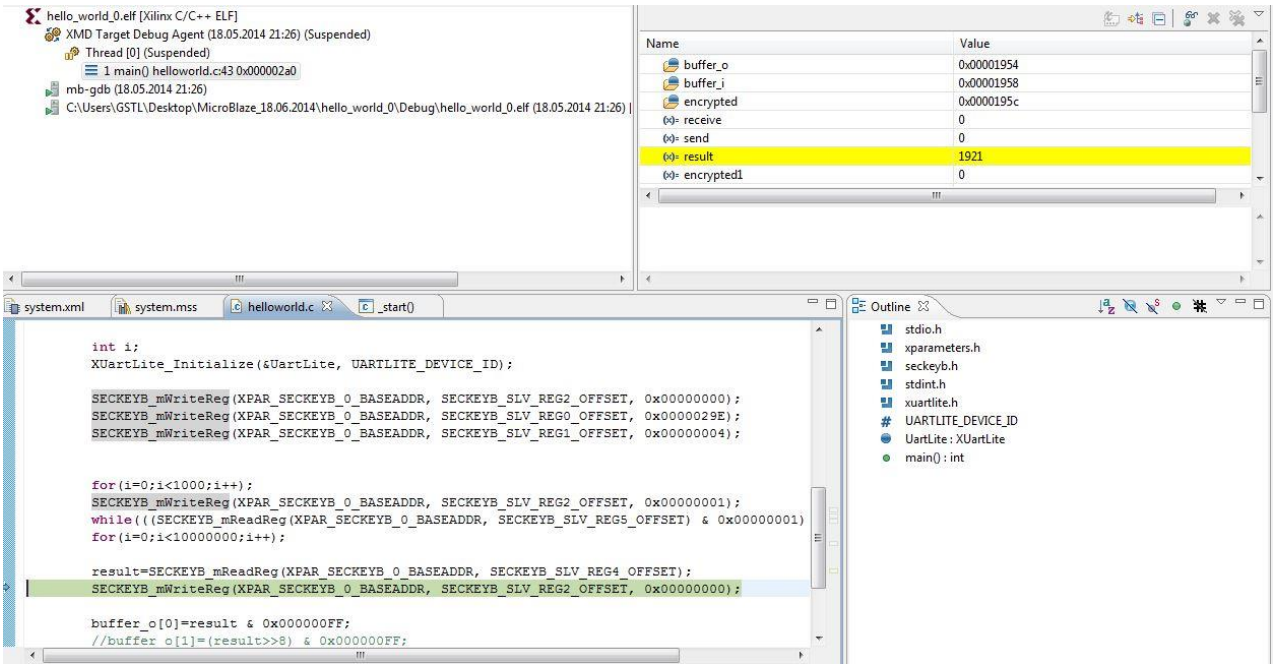

Figure 6.10: Process of producing secret key of second MicroBlaze

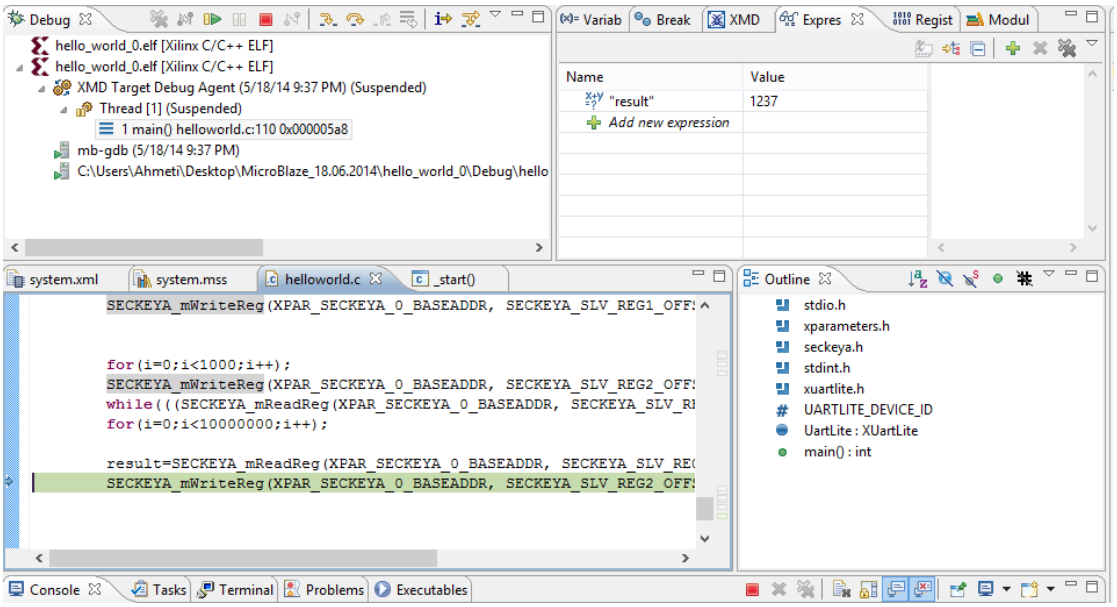

Figure 6.11: Computing the shared secret key in first MicroBlaze

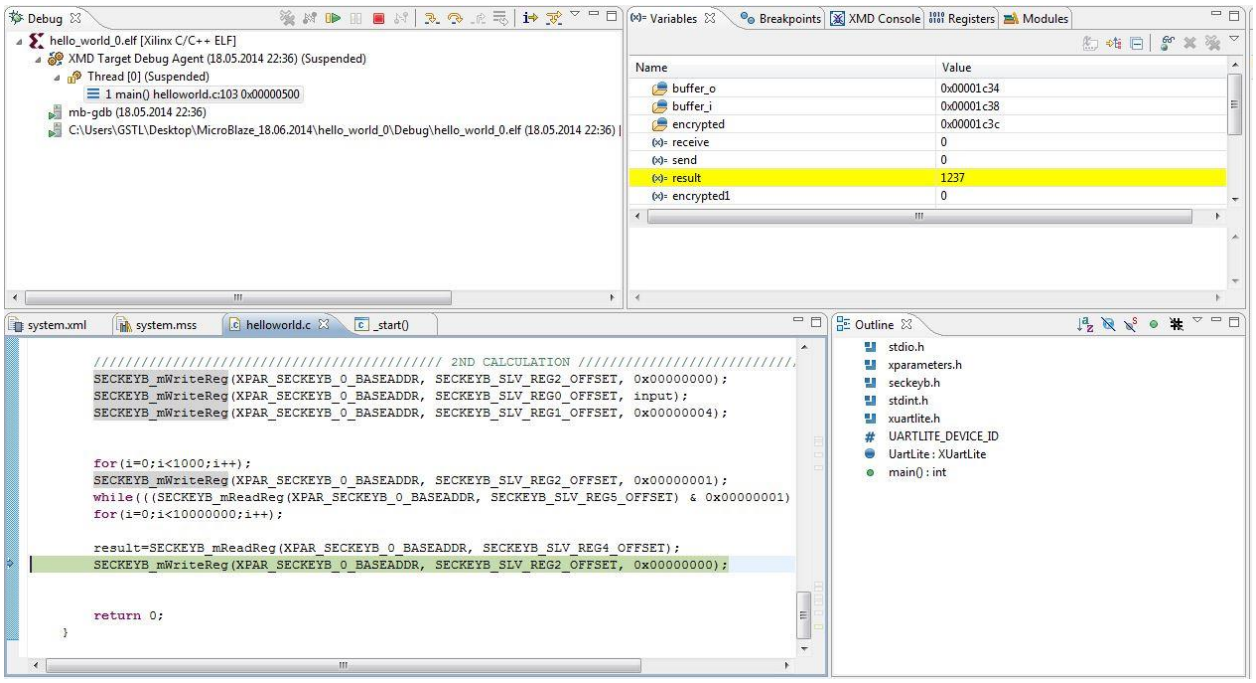

Figure 6.12: Computing the shared secret key in the second MicroBlaze

7. CONCLUSION

In this project a protocol which implements secure data communication in an insecure communication channel was implemented. The system consists of two parts, including software and hardware. The basic components of the secure data communication protocol were designed, modeled and verified using the Verilog HDL and VHDL. Initially for sharing secret key mechanisms and AES algorithm a research was carried on. As a result of these researches it was observed that the most secured and practical key exchange mechanism was Diffie-Hellman key exchange mechanism.

The project has been followed with design and implementation of the Diffie-Hellman key exchange mechanism using Verilog hardware description language. The Diffie-Hellman is a mechanism where the secret key is shared between two systems through the algebraic operation $\alpha^{\beta} mod p$. The designed hardware is composed of two components, including a multiplier hardware, which performs multiplication through Karatsuba algorithm, and an exponentiation and modulo hardware, which operates through Square and Multiply algorithm. In operation the multiplier hardware has created problem on covering the area of FPGA ineffectively. The problem has been fixed with optimizing the designed hardware with redesigning it as sequential digital system.

Later the designed hardware together with the AES 128 hardware is added into two separate MicroBlaze as peripheral. By exporting the project into the SDK, software for the control between these peripherals and MicroBlaze is generated and as a result a secure data communication protocol is implemented.

There are specific aspects of this project that may be investigated in the future. One example is redesigning the multiplier hardware in a way that it takes less time for calculation. Another aspect is increasing the capability of the Diffie-Hellman hardware so it can perform operation for higher order bits such as 192 bits or 256 bits, so it can be implemented with AES192 and AES256 hardware's and implement a more secured communication.

REFERENCES

[1] **Stallings,W.** (2011). *Cryptography and network security principles and practice* (5th ed.). United States of America: Pearson Education Inc., pp. xv

[2] **FIPS 197.** (2001). *Advanced Encryption Standard*. Nationa Institute of Standard and Technology, pp. 1

[3] **Jesman, R. Vallina, Fernando M. Saniie, J.** (nd). *MicroBlaze Tutorial Creating a Simple Embedded System and Adding Custom Peripherals Using Xilinx EDK Software Tools*. Illinois Institute of Technology, pp. **4-5**

[4] **Cohen,F.** (1995). *A Short History of Cryptography*, from http://web.itu.edu.tr/~orssi/dersler/cryptography/kaynaklar.html, pp 1-5

[5] **Ayushi.** (2010). *A Symmetric Key Cryptographic Algorithm*, International Journal of Computer Applications (0975 - 8887), pp. 2

[6] **Boneh, D.** (2014). *Cryptography I*. Retrieved from Stanford University, https://class.coursera.org/crypto-009

[7] **Srouji, J. & Fitzpatrick, T. & Korpusik, N. & Sutherlan, S.** (2005). *IEEE Standard for Verilog Hardware Description Language*. United States of America, pp: 1

[8] **URL-1** <http://www.stoimen.com/blog/2012/05/15/computer-algorithms-karatsuba-fast-multiplication/>

[9]**URL-2**< http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Exponentiation_by_squaring.html>

[10] **Gencosmanoglu, B. A.** (2014). *The software/hardware implementation of a secure data communication protocol with TEA algorithm*, pp: 20-26

[11] **Xilinx.** (2010). *Embedded System Tools Reference Manual*, pp: 19-24

[12] **Xilinx.** (2008). *MicroBlaze Processor Reference Guide*, pp: 10

RESUME

Name Surname: Ahmet Jorganxhi

Place and Date of Birth: Prizren, Kosova

High School: Mehmet Akif College, Kosova (2006-2010)

Major: Science

B.Sc.: Istanbul Technical University, Turkey (2010-Today)

Major: Electronics and Communication Engineering

Orientation: Electronics Engineering