ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS ENGINEERING FACULTY

Implementatıon Of Dıffie-Hellman Key Exchange Protocol Usıng Mıcroblaze On Fpga

BSc Thesis by
Onur ŞAHİN
(040110098)

Department: Electronics and Communication Engineering
Programme: Electronics and Communication Engineering

Supervisor: Assoc. Prof. Dr. Sıddıka Berna Örs Yalçın

MAY 2017

İSTANBUL TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

FPGA ÜZERİNDE MICROBLAZE KULLANILARAK DIFFIE-HELLMAN ANAHTAR DEĞİŞİMİ PROTOKOLÜNÜN GERÇEKLENMESİ

Bitirme Ödevi

Onur ŞAHİN

(040110098)

Bölümü: Elektronik ve Haberleşme Mühendisliği

Programı: Elektronik ve Haberleşme Mühendisliği

Danışmanı: Doç. Dr. Sıddıka Berna Örs Yalçın

MAY 2017

To my beloved family,

PREFACE

First of all, I would like to show my greatest appreciation to my supervisor, Assoc. Prof. Dr. Sıddıka Berna Örs Yalçın for her brilliant assitance and guidance during this thesis preparation time. Also I would like to thank my friend Arda Tanrıkulu for his all helps and to thank to Research Asst. Latif Akçay for his help and guidance. Above all, I would like to thank my family fo their endless support.

May 2017

Onur Şahin

CONTENTS

Page

PREFACE.....vii

INDEX.....ix

ABBREVIATIONS.....xi

LIST OF FIGURES.....xii

ÖZET.....xiii

SUMMARY.....xiv

1. INTRODUCTION.....1

2. CRYPTOGRAPHY.....3

2.1 Overview.....3

2.2 Symmetric-key Cryptography.....3

3. DIFFIE-HELLMAN KEY EXCHANGE.....4

3.1 The Diffie-Hellman Key Exchange Protocol.....4

3.2 Security.....6

3.3 Diffie-Hellman Key Exchange Implementation.....7

4. ADDER-SUBTRACTOR MODULE DESIGN.....9

4.1 Verilog Hardware Description Language.....9

4.2 Xilinx ISE.....9

4.3 Full Adder.....9

4.4 Ripple Carry Adder.....11

4.5 17-bit Adder-Subtractor.....12

5. MICROBLAZE.....15

5.1 Overview.....15

5.2 Xilinx Embedded Development Kit (EDK).....16

5.2.1 Xilinx Platform Studio.....17

5.2.1.1 Overview.....17

5.2.1.2 Custom IP Implementation.....18

5.2.2 Xilinx Software Development Kit.....21
5.2.2.1 Overview.....21
5.2.2.2 C Project.....21

6. HARDWARE DESIGN.....22
6.1 Communication Between FPGA and Computer.....22
6.1.1 UART Protocol.....22
6.1.2 SDK Terminal Configurations.....23

7. RESULTS 23

REFERENCES.....??

RESUME.....??

ABBREVIATIONS

DH : Diffie Hellman

EDK : Embedded Development Kit

FPGA : Field Programmable Gate Array

ISE : Integrated Software Environment

RCA : Ripple Carry Adder

RTL : Register Transfer Level

SDK : Software Development Kit

UART : Universal Asynchronous Receiver/Transmitter

USB : Universal Serial Bus

XPS : Xilinx Platform Studio

IP: Intellectual Property

BFM: Bus Functional Model

FIGURE LIST

Figure 2.1: Model of shared secret key encryption
Figure 3.1: Schematic of Diffie-Hellman key exchange method
Figure 3.2: Register function codes in SDK
Figure 3.3: Fast exponentiation algorithm codes in C language
Figure 4.1: Gate-level diagram for full adder
Figure 4.2: Verilog code for full adder with parameters
Figure 4.3: Ripple carry adder diagram for 4 full adder
Figure 4.4: Verilog code for RCA with option parameter
Figure 4.5: Example operations for two's complement
Figure 4.6: Verilog code of adder-subtractor top module
Figure 5.1: MicroBlaze Core Block Diagram
Figure 5.2: Embedded Development Kit (EDK) tools architecture
Figure 5.3: PLB interconnect scheme
Figure 5.4: BFM simulation system architecture diagram
Figure 5.5: Verilog code of AND gate
Figure 5.6: User Logic implementation
Figure 6.1: Simulation results just before the information come to the AND gate
Figure 6.2: Simulation results when the information is carried to the AND gate
Figure 6.3: Base and High addresses values
Figure 6.4: Design summary of AND gate module
Figure 6.5: Protocol results screen from the SDK
Figure 6.6: Base and High addresses and sizes of instances
Figure 6.7: Design summary of adder-subtractor module

FPGA ÜZERİNDE MICROBLAZE KULLANILARAK DIFFIE-HELLMAN ANAHTAR DEĞİŞİMİ PROTOKOLÜNÜN GERÇEKLENMESİ

ÖZET

Kriptografi günümüzde gönderilen bir bilginin istenmeyen şahışlar tarafından ulaşılmasını engelleyerek güvenli haberleşme tekniklerini uygulayan ve araştıran bir şifreleme bilimidir. Bu bilimi kullanan verici bir sistem şifrelenmemiş bir mesajı şifreyelerek alıcı sisteme yollar ve ortak paylaşılmış bir anahtar sayesinde güvenlik sağlanır ve haberleşme gerçeklenir. 1949 yılında Claude Shannon, "Bell Systems Techincal" dergisinde haberleşme teorisi ve gizlilik sistemleri üzerine bir makale yayınlamıştır. Bu makale kriptografi araştırmalarına katalitik etkide bulunmuş ve modern kriptografinin gelişiminde önemli rol oynamıştır. Diffie-Hellman anahtar değişimi protokolü ise dijital dünyada temel bir kriptografi sistemi olarak görülür ve güvenli haberleşmenin temel taşlarından biridir. Protokolün gerçeklenebilmesi için bir adet alanda programlanabilir kapı dizileri (FPGA) kartı kullanılmıştır.

Bu projenin ilk aşamasında bir yazılımsal çekirdek işlemci olan MicroBlaze, Xilinx Platform Studio programı kullanılarak FPGA üzerinde devreye alındı. İlk olarak Verilog Donanım Tanımlama Dili ile yazılmış VE kapısı modülü özel IP oluşturulmasında kullanıldı. Bu oluşturulan özel IP'nin MicroBlaze ile bağlantısı kurularak SDK ortamına geçildi. Simulasyonda sağlıklı bir gözlem yapabilmek için ve kapısının girişlerine gerçekleşebilecek olası değerler gönderildi ve oluşturulan elf dosyası sonucunda simulasyon çalışıtırıldı. Ve kapısının MicroBlaze ile sağlıklı bir şekilde haberleşebildiği gözlendi. Diffie-Hellman algoritmasının gerçeklenebilmesi için ise bir toplayıcı-çıkarıcı modülüne ihtiyaç duyuldu. Verilog dilinde bu toplama çıkarma işlemlerini yapabilen modül tanımlandı. 1 işaret biti ve 16 büyüklük biti olmak üzere toplam 17 bitten oluşan bu modül yeniden bir özel IP yaratılarak MicroBlaze ile Haberleşmesi sağlandı. SDK ortamına geçildi ve Diffie-Hellman anahtar değişim protokolü gerçeklendi. Sonuçlar SDK ortamında FPGA üzerinde bulunan UART çıkışından terminalde okundu ve program çıktısında anahtar değişimi protokolünün düzgün bir şekilde çalıştığı gözlendi.

IMPLEMENTATION OF DIFFIE-HELLMAN KEY EXCHANGE PROTOCOL ON FPGA USING MICROBLAZE

SUMMARY

Cryptography is a science of ciphering, which today implements and investigates secure communication techniques by preventing the transmission of information by unwanted persons. A transmitting system that uses this knowledge encrypts an unencrypted message to the receiving system, and a common shared key provides security and communication. In 1949, Claude Shannon published an article on communication theory and privacy systems in "Bell Systems Techincal" magazine. This article has catalytic effect on cryptography research and plays an important role in the development of modern cryptography. The Diffie-Hellman key exchange protocol is seen as a basic cryptography system in the digital world and is one of the cornerstones of secure communications. One field programmable gate array (FPGA) card is used to implement the protocol.

In the first phase of this project, MicroBlaze, a software core processor, was deployed on the FPGA using the Xilinx Platform Studio program. First it was written in Verilog Hardware Description Language AND gate module was used to create a custom IP. This created custom IP was connected to MicroBlaze and switched to SDK environment. In order to make a healthy observation in the simulation, the possible values that can be realized at the entrance of the gate were sent and the simulation was run as the result of the generated elf file. And it was observed that the gate could communicate with MicroBlaze in a healthy way. To implement the Diffie-Hellman algorithm, a adder-subtractor module was needed. In Verilog language, this module is able to perform subtraction operations. This module consists of a total of 17 bits including 1 sign bit and 16 magnitude bits, and a custom IP is created to communicate with MicroBlaze again. SDK environment, and the Diffie-Hellman key exchange protocol was implemented. The results were read in the SDK environment from the UART output on the FPGA and it was observed that the key exchange protocol worked properly on the program output.

1. INTRODUCTION

This section includes a detailed description of the use of MicroBlaze, why it is needed, what programs are used to implement and simulate it.

In this digital epoch truly there is no any condition where the security is not crucial because of universal electronic connectivity, of viruses and hackers, of electronic fraud [1]. At this point software and hardware security measures are at the forefront. One of these security measures is to create a cryptographic system. The first step of a cryptographic system is sharing a common key between two system which cannot be detected by 3rd party system. Common key do not take place in the communication channel in this way the security is provided. One of the particular methods for exchanging cryptographic keys between two system over a public channel is the Diffie-Hellman key exchange method. The Diffie-Hellman algorithm depends on the complexity of calculating discrete logarithms [1].

The difficulty of resolving a password increases with its length. Considering the key size used in this project, although today's computer processors have large processing capacities, there is also a need for hardware for the purpose. The FPGA hardware description language is a hardware block ready to program, and it satisfies this need. The necessary hardware used in this project for the algorithm to implement the Diffie-Hellman protocol is the adder-subtractor module which is the basis of the algorithm. Processors that can be used on FPGAs, such as MicroBlaze, allow us to transfer and manipulate the module that performs the operations that require rapid implementation of the key exchange protocol to the software environment. First, this module will be communicated with MicroBlaze to an AND gate without being programmed onto the FPGA. The aim here is to check whether the MicroBlaze and the AND gate communicate in a healthy manner via simulation. Once this communication has been observed, the collector-extractor module can now be designed.

The adder-subtractor module was written with Verilog, one of the hardware description languages. Using Xilinx ISE it was tested that these modules can be added and subtracted properly by sending appropriate values to their inputs. At this point, the communication of the MicroBlaze and adder-subtractor modules had to be carried out. A custom IP was created using the Xilinx Platform Studio and the design was transferred to the Xilinx Software Development Kit program. The purpose here is to provide the ability to write the algorithm of the Diffie-Hellman protocol in C language. In the C project, we have realized the algorithm by means of the adder-extractor module and it is observed that the protocol is run properly from the terminal on the SDK by FPGA programming.

2. CRYPTOGRAPHY

2.1 Overview

Cryptography is where security engineering meets mathematics[3]. So the existence purpose of cryptography is secure communication that fundamentally originates in two parts. The first one is secured key establishment. It is obvious that secured key establishment essentially amounts to Alice and Bob sending messages to one another such that at the end of this protocol, there's a shared key that they both decided, shared key X, and beyond just a shared key, simply Alice would know that she's talking to Bob and Bob would know that he's talking to Alice. But the attacker who listens the communication channel has no idea what the X is. And also an attacker cannot even rake this traffic up without being detected. After they have a shared key, Bob and Alice want to exchange their messages trustfully using this key.

The significant point about the cryptography for this project is length of the key. In this project 16-bit key is used. Longer the key means more the safety. Therefore FPGA's are the one of the solution for seeing the results quickly. There is no doubt any circuit design can be implemented into FPGA. Togetherness of this embedded system and security algorithms like Diffie-Hellman key exchange allows to digital world progress rapidly in terms of security measure.

2.2 Symmetric-key Cryptography

Symmetric-key cryptography is one type of cryptography where the same cryptographic keys for both encryption of plaintext and decryption of ciphertext are used[4]. There is five main things in a cryptosystem. These things are: plaint text, encryption algorithm, secret key, ciphertext and the decryption algorithm. Plain text is the pure data which is input of encryption algorithm. Encryption algorithm is the machine that performs complex mathematical operations onto plaintext. Secret key is the main part of this

cryptosystem which is applied both encryption and decryption algorithm. If the key changes the output of encryption algorithm also differs because of the particular mathematical operation. Ciphertext is the encryption algorithm's output information that sent in communication channel. The last one is decryption algorithm. It refers the reversal process of the encryption algorithm. As said before the main point of this symmetric-key cryptosystem is secret key. Attacker shouldn't discover the key so it never sent in transmission channel. The following figure shows the model of symmetric-key cryptography system.

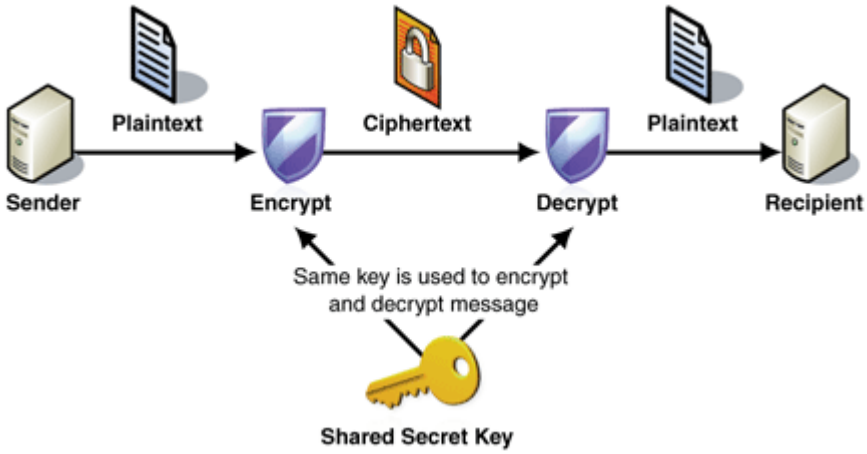


Figure 2.1: Model of shared secret key encryption[1]

3. DIFFIE-HELLMAN KEY EXCHANGE

3.1 The Diffie-Hellman Key Exchange Algorithm

The Diffie-Hellman key exchange method makes way for two systems that have no initial knowledge of each other to specify commonly a shared secret key over an unsecured transmission channel[3]. This key exchange method was evolved in 1976 and released in "New Directions in Cryptography". The Diffie-Hellman key exchange leads to solution of following problem. System X and system Y are both symmetric-

key cryptographic systems. System X and system Y are intentioned to share their secret key for the secure communication with each other. Both systems X and Y must have a common shared secret key so the hacker must not have any idea what the secret key is.

The mechanism of Diffie-Hellman key exchange will be explained basically. First pick a large prime "N", in fact "N" is usually used to represent primes, and an integer "α", that take place in the range of $[1, \ldots, N]$. Both "N" and "α" are parameters of Diffie-Hellman protocol that are chosen once and are constant forever. System X and Y select a random number "a" and "b" respectively, in the range of $[1, \ldots, N-1]$. Then system X is going to compute Result $X = \alpha^a \ mod \ p$. So system X computes the exponentiation and reduces the result to the modulo of p, and the result is sent to the system Y. System Y also calculates the Result $Y = \alpha^b \ mod \ p$ and sends to the system X. So system X sends Result X to system Y and system Y sends Result Y to system X, and now a shared secret key can be generated. So there is a question properly which one is the shared key? It is named as S_{ab} and it is defined as $\alpha^{ab} \ mod \ p$. This spectacular investigation that Diffie-Hellman had 41 years ago is that indeed both sides can calculate the value of $S = \alpha^{ab}$. System X can compute this value since, it can take the value Result Y, that it had received from system Y, and raise to the power of "a" . Also the system Y compute the value in same style as system X, and as a result both systems X and Y, have computed the S and resulted in the same secret key. This protocol opened a road to new era in cryptography.

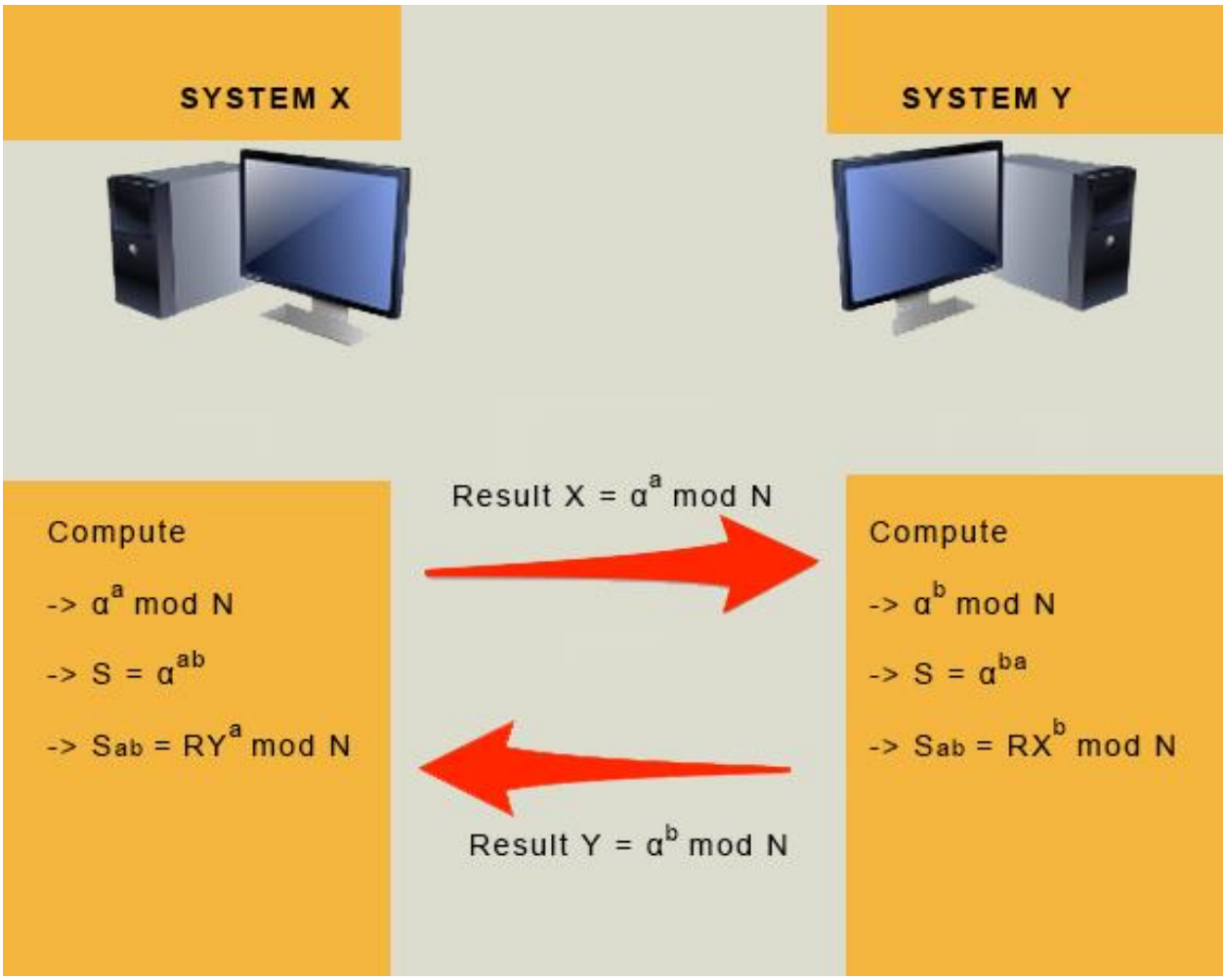


Figure 3.1: Schematic of Diffie-Hellman key exchange method[1]

3.2 Security

It is obvious that both systems X and Y end up with the same shared key. The attractive point of this protocol is the reason of trustworthy[1]. In other saying, what is the reason behind the attacker can not solve the same shared key that both systems X and Y have calculated themselves? The hacker or earwitness figures out α and N which are constant and known by everybody what they are. He or she also gets the value that system X sends to system Y named as Result X and he sees the value that system Y sends to system X named as Result Y. The point is whether attacker can

compute α^{ab} just given these four values. The explanation of this problem is that the hacker has to solve the Discrete Logarithm Problem (DLP) and then he can figure out random generated numbers “a” and “b” after which is very simple for hacker to calculate α^{ab} [10].

The DLP is the problem of solving “d” if $c \equiv \alpha^{d} (mod\ N)$, with c, α and p are known. Two solutions are exist but there is no any algorithm that can succeed this in a decent time. First solution is Brute Force, which is not an impressive solution, since it needs “N” steps to calculate the solution and in every try needs a really long time of work.

3.3 Diffie-Hellman Key Exchange Implementation

In this thesis the Diffie-Hellman key exchange algorithm is designed with C Language in Xilinx SDK and is both implemented and tested in a Spartan-6 FPGA. The diffie-hellman key exchange is designed for 16-bit key. Fundamentally, in SDK the exported adder-subtractor design is used because of the mathematical operations like multiplication and exponentiation. There is no way to accomplish any mathematical expression without perfoming addition or subtraction process.

Firstly the modulo algorithm should be designed. If the subtraction repeats while the subtrahend is less than or equal to minuend this process results in successful modulo operation. The following figure shows how it is written in C language.

```
int mod(int A, int N){
int kalan;
while(A>=N){

    ADDERSUB_mWriteReg(XPAR_ADDERSUB_0_BASEADDR, ADDERSUB_SLV_REG0_OFFSET, A);
    ADDERSUB_mWriteReg(XPAR_ADDERSUB_0_BASEADDR, ADDERSUB_SLV_REG1_OFFSET, N);
    ADDERSUB_mWriteReg(XPAR_ADDERSUB_0_BASEADDR, ADDERSUB_SLV_REG2_OFFSET, 0x00000001);

    A = ADDERSUB_mReadReg(XPAR_ADDERSUB_0_BASEADDR, ADDERSUB_SLV_REG3_OFFSET);

}

kalan = A;
return kalan;
```

Figure 3.2: Register function codes in SDK

Addition operation has the same structure with AND gate implementation which is explained later in this thesis. Secondly the algorithm should calculate A+B (mod N) so the addition function is called before the modulo function in addition-modulo function. The remaining functions are the multiplication-modulo function and the exponentiation-modulo function. These are different from the addition-modulo function and requires some conditional operations. For the multiplication-modulo function the addition-modulo is called for the 16 times which is length of the key and if the i-th bit of B is 1 the addition-modulo is called again. Multiplication-modulo function returns C variable which is defined firstly 0. This operation is repeated for the exponentiation-modulo function and the differences are pre-defined C is 1 for this time and multiplication function is called instead of addition-modulo function. This algorithm allows the calculation of fast modular exponentiation. The following code figure shows how it is completed.

```
int mult(int A,int B,int N){

    int i;
    int C=0;

    for(i=15;i>=0;i=i-1){

        C=modadd(C,C,N);

        if(B&(1<<i))

        C=modadd(C,A,N);
    }


    return C;
}
```

```
int expo(int A,int B, int N){

    int i;
    int C=1;

    for(i=15;i>=0;i=i-1){

        C=mult(C,C,N);

        if(B&(1<<i))

        C=mult(C,A,N);
    }


    return C;
}
```

Figure 3.3: Fast exponentiation algorithm codes in C language

In the main the Diffie-Hellman function is called for the calculation of Result X and Result Y. After that Diffie-Hellman function is called again for this time parameter A is changed with Result X and Result Y for the calculation of keys respectively System

Y and System X. From the terminal of FPGA's serial port it is easy to see this algorithm works very well. In simulation part it will be mentioned.

4. ADDER-SUBTRACTOR MODULE DESIGN

4.1 Verilog Hardware Description Language

Verilog Hardware Description Language (Verilog HDL) is a formal demonstration designed by Xilinx for use in all phases of an electronic system setup. As both machine and human readable, it supports the development, validation, synthesis and testing of equipment [10]. Verilog HDL serves as a tool that is a major force in digital system design because a designer offers the designer the advantage of the design simulation he or she receives to make critical decisions about the designer's design. The designer can also analyze how the design will behave if it is designed to be synchronous or asynchronous with the addition of its own command. As mentioned above, the readability of Verilog HDL makes it very attractive and useful for engineers. All of these synthesis, simulation, verification, etc. operations were compiled in a tool designed by Xilinx and known as Xilinx ISE.

4.2 Xilinx Integrated Software Environment

Xilinx ISE is a software tool produced by Xilinx for synthesizing and analyzing HDL designs, allowing developers to synthesize their designs, perform timed analysis, display RTL diagrams, simulate an adaptation as a different stimulus response, and configure the target device with the programmer. Simulation for system level testing can be done with ISIM logic simulator. Test programs should also be written in HDL languages. In this project, ISIM is used for logical verification to ensure that the 17-bit add-subtract module produces the expected result.

4.3 Full Adder

In digital systems adder is an inevitable logic unit for the implementation of any mathematical operation[10]. Each operation is related to sum operation logically. There is two type of adder, the first one is half adder and the second one is full adder. Half adder, adds two 1-bit operands X and Y, producing a 2-bit sum. The sum can range from 0 to 2, which requires two bits to express. The low-order bit of the sum may be named HS (half sum), and the high-order bit may be named Cout (carry out). It can be seen from the following equations for HS and Cout:

$$HS = X.Y' + X'.Y \quad \textbf{(4.1)}$$

$$Cout = X.Y \quad \textbf{(4.2)}$$

For this project full adders are used instead of half adders because the chain structure should be implemented to compound 17-bit together. Full adder has Cin(carry in) input except the addend-bit inputs X and Y. The sum of the three inputs can range from 0 to 3 that means two output bit still enough. Following equations show how it is different from the half adder equations and the following figure shows gate-level circuit diagram for full adder.

$$S(sum) = X.Y'.Cin' + X'.Y.Cin' + X'.Y'.Cin + X.Y.Cin \quad \textbf{(4.3)}$$

$$Cout = X.Y + X.Cin + Y.Cin \quad \textbf{(4.4)}$$

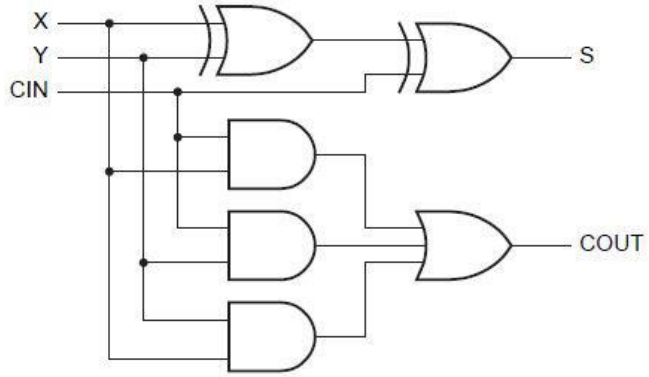


Figure 4.1: Gate-level diagram for full adder[10]

In verilog language it is really simple to create this structure. Following figure indicates the full adder with parameters x,y,cin,sum and cout. This module is the keystone of Diffie-Hellman key exchange algorithm, in other words this is the undermost module in this project.

```
module full_adder(x,y,cin,sum,cout);

input  x;
input  y;
input  cin;
output sum, cout;
assign sum = x^(y^cin);
assign cout = (x&&y) || (cin&&(x^y));

endmodule
```

Figure 4.2: Verilog code for full adder with parameters

4.4 Ripple Carry Adder

Creation of a logical circuit using full adders to add N-bit numbers is possible[10]. Each full adder inputs a Cin, which is the Cout of the previous adder. This kind of chain adder is called a ripple-carry adder, since each carry bit "ripples" to the next full adder. In this project 17-bit adder-subtractor module is designed. Therefore 17 full adder are connected to each other. The following figure is an example for 4-bit ripple carry adder.

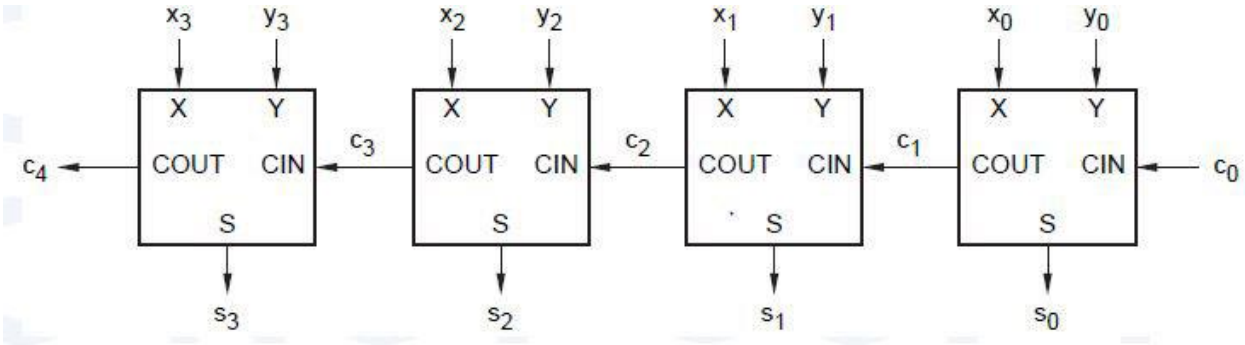


Figure 4.3: Ripple carry adder diagram for 4 full adder[10]

Calling 17 times the full adder module with the carry wire parameter in Verilog allows to us unite them. The important point is the first Cin(C0) should be the option bit for the making decision about which operation is going to be executed addition or subtraction? In this project it is named as “option” input. The following verilog code shows the ripple carry adder connections.

```
module Ripple_Carry_Adder(a,b,option,s);

input option;

input [16:0] a,b;

output [16:0] s;

wire [17:1] carry;


full_adder fa0(a[0],b[0],option,s[0],carry[1]);
full_adder fa1(a[1],b[1],carry[1],s[1],carry[2]);
full_adder fa2(a[2],b[2],carry[2],s[2],carry[3]);
full_adder fa3(a[3],b[3],carry[3],s[3],carry[4]);
full_adder fa4(a[4],b[4],carry[4],s[4],carry[5]);
full_adder fa5(a[5],b[5],carry[5],s[5],carry[6]);
full_adder fa6(a[6],b[6],carry[6],s[6],carry[7]);
full_adder fa7(a[7],b[7],carry[7],s[7],carry[8]);
full_adder fa8(a[8],b[8],carry[8],s[8],carry[9]);
full_adder fa9(a[9],b[9],carry[9],s[9],carry[10]);
full_adder fa10(a[10],b[10],carry[10],s[10],carry[11]);
full_adder fa11(a[11],b[11],carry[11],s[11],carry[12]);
full_adder fa12(a[12],b[12],carry[12],s[12],carry[13]);
full_adder fa13(a[13],b[13],carry[13],s[13],carry[14]);
full_adder fa14(a[14],b[14],carry[14],s[14],carry[15]);
full_adder fa15(a[15],b[15],carry[15],s[15],carry[16]);
full_adder fa16(a[16],b[16],carry[16],s[16],carry[17]);

endmodule
```

Figure 4.4: Verilog code for RCA with option parameter

4.5 17-bit Adder-Subtractor

Here in this project Adder-Subtractor module is going to be 17-bit because of there should be 1 sign bit and 16 magnitude bit for realization of subtraction operation[10]. Combining 17 full adder is not enough to make subtraction operation. Two's complement should be applied to complete our design. Two's complement number representation is used for signed numbers. While the signed-magnitude system negates a number by changing only its sign, a complement number system negates a number

by taking its complement as defined by the system[desktop]. The following figure exactly shows how two's complement work.

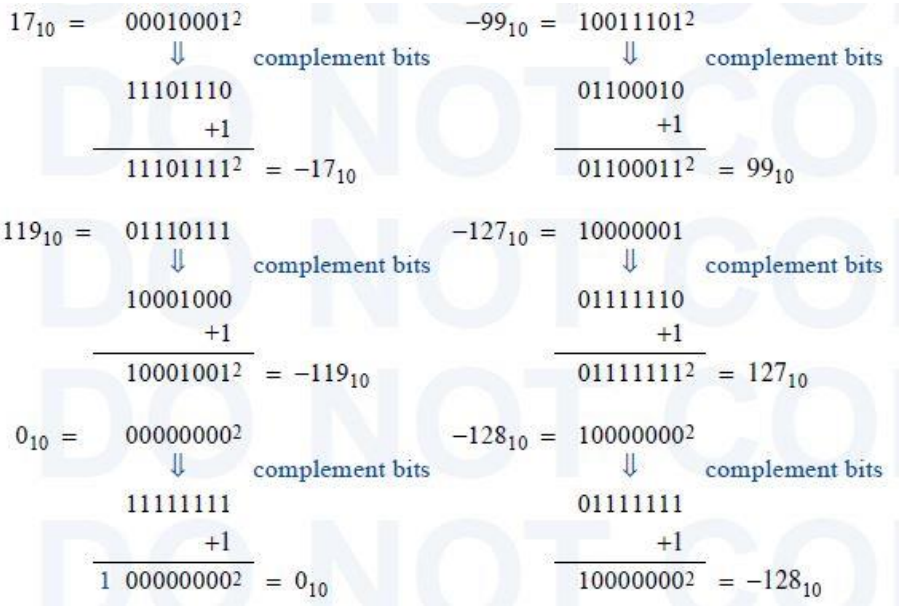


Figure 4.5: Example operations for two's complement

For subtraction operation the subtrahend's two's complement form is calculated and then the addition can be applied. If the minuend, subtrahend's two's complement form and 1 are summed and the subtraction operation has successfully done. As mentioned before all mathematical operations are related to addition operation. At this point it is obvious that if the option 0 is zero the circuit is going to produce addition operation. If it is 1 the circuit is going to produce subtraction operation. The reason for this the option bit and i-th bit of the subtrahend are the inputs of XOR gate. If we XOR the bit A with 1 the result is always be the inverse of the A. If we XOR the bit A with 0 the result is always be the same of the A. The following code figure simply shows us how the adder-subtractor module is implemented.

```
module Subtractorx(m,n,opt,o);

localparam L = 16;

input [16:0] m;
input [16:0] n;
input opt; // if opt is 0 adder,if 1 subtractor

wire [16:0] p;
output [16:0] o;

genvar i;
generate

for(i=0; i<L+1 ; i=i+1) begin:x

  assign p[i] = opt^(n[i]);

end

endgenerate

Ripple_Carry_Adder RCA0(m,p,opt,o);

endmodule
```

Figure 4.6: Verilog code of adder-subtractor top module

5. MICROBLAZE

5.1 Overview

Designed modules are used as a peripheral part of a microprocessor and the microprocessor takes over the controller role by controlling these peripherals and applying a secure data communication protocol. The soft-core microprocessor that can be used with Xilinx Field Programmable Gate Arrays (FPGAs) through the Xilinx Embedded Development Kit (EDK) software known as MicroBlaze [2].

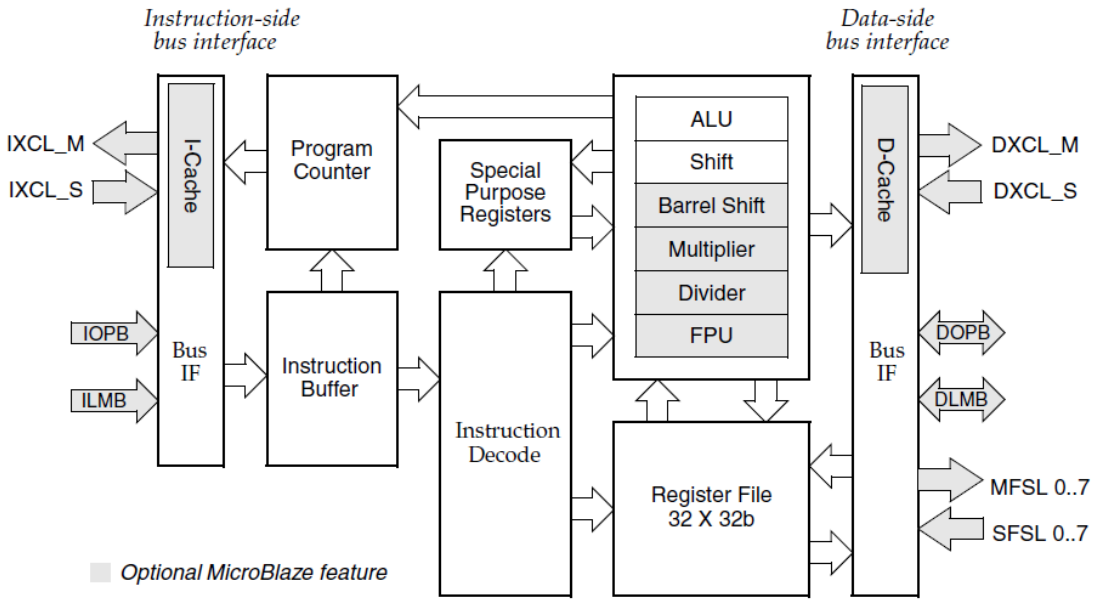


Figure 5.1: MicroBlaze Core Block Diagram[6]

The MicroBlaze, a virtual microprocessor, which is constructed by integration of blocks of code named as core placed in the internal of a Xilinx FPGA[2].The main point of MicroBlaze is the architecture (RISC) for Universal Asynchronous Receiver/Transmitter, Flash, General Purpose Input / Output and similar Xerox® Xilinx FPGAs for the Harvard Reduction Instruction Set Computer. Separate 32-bit data and command buses run at high speed to perform programs and provide data at the same time to provide both chipset and external memory[2].

5.2 Xilinx Embedded Development Kit (EDK)

The Xilinx Development Kit (EDK) provides an environment in which the designer can create a processor system embedded in all the features that can be implemented in a Xilinx FPGA device. EDK is an integral part of the Xilinx programmable logic components, as defined by the Integrated Software Environment (ISE®) Design Suite Embedded System Edition [5], developed by Xilinx. The EDK components are:

1- The Xilinx Platform Studio

2- The Software Development Kit (SDK)

3- Intellectual Property (IP) cores

The design flow in the EDK, in other words the features of the designer to start with an ISE project, then to add an embedded processor source in the ISE project. When the ISE components are complete, EDK starts synthesizing the design in the microprocessor hardware, matches the FPGA and generates the bitstream as the final step.

An embedded hardware platform consists of one or more processors, peripherals and inter- connections via a memory block and processor path [10]. As a tool, EDK has been left to focus on the designer as it provides a great convenience to the designer as a link between designers, peripherals, and FPGA hardware designs, address mapping of the system being designed, communication protocols and other interconnection tools. Hardware and software designs. The following figure shows the design flow diagram of a system designed using microcontroller in the FPGA. The steps shown in

the picture are created by EDK and presented to the designer through a user interface, making it easier and faster to implement complex designs.

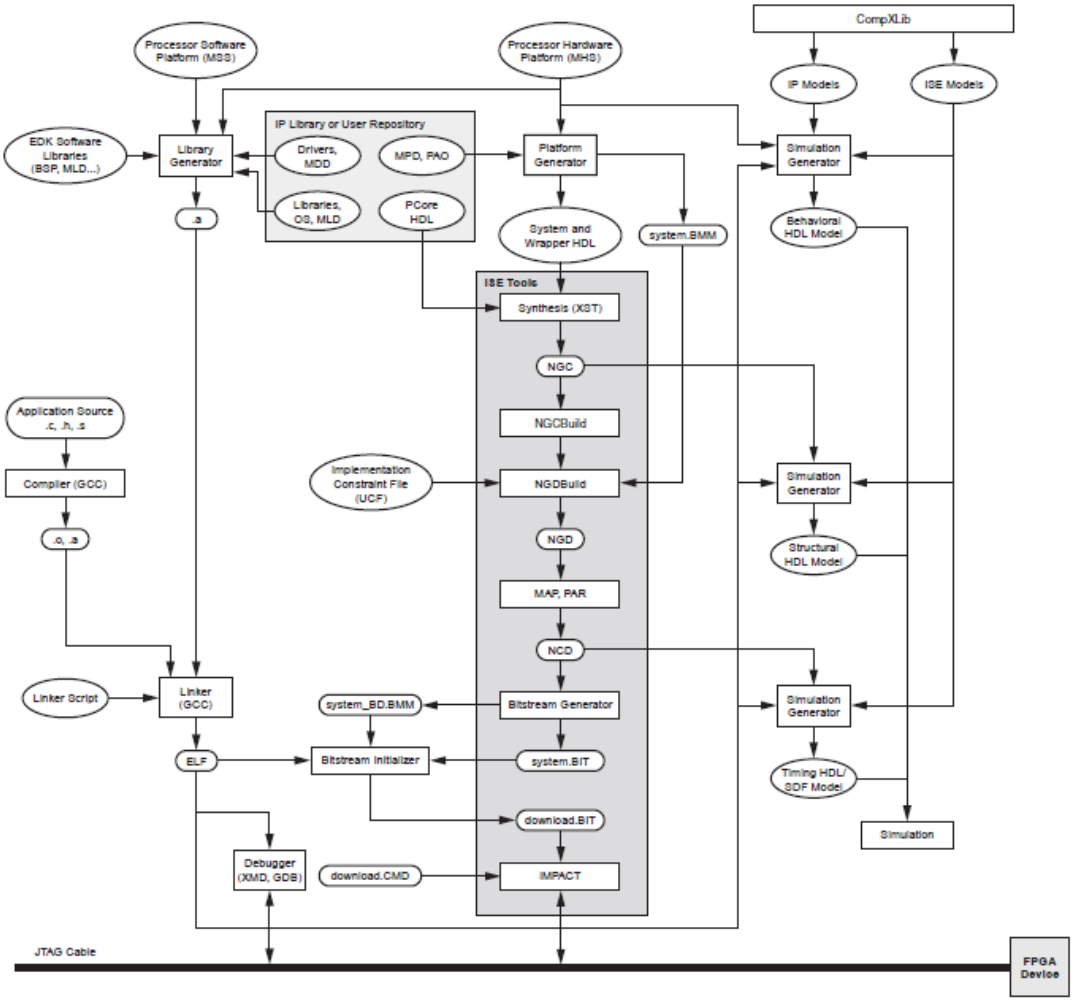


Figure 5.2: Embedded Development Kit (EDK) tools architecture[5]

It was stated that MicroBlaze could be configured in the previous section. This case is applied in the EDK tool. The EDK base system comes with the Base System Generator (BSB). BSB, hardware based on micro-blaze Intellectual Property (IP) cores, the connections of these systems are made from EDK.

5.2.1 Xilinx Platform Studio

5.2.1.1 Overview

One of the components of the EDK is Xilinx Platform Studio (XPS), which provides an environment for building embedded processor systems based on MicroBlaze and PowerPC processors [5]. In XPS, the address mapping phase of peripherals connected to MicroBlaze is completed. Later, synthesis and application phases of the design project were realized.

5.2.1.2 Custom IP Implementation

For the simple implementation in this project a two-input AND gate is selected. Firstly, base system builder is used for the creating new project. PLB interconnect system is selected. In the peripheral configuration section only the "dlmb_cntlr" and "ilmb_cntlr" are used as MicroBlaze peripherals because there is no need external hardwares for this implementation. Two necessary operation creating and importing peripheral are applied in Xilinx Platform Studio. There are some important points when the creating new peripheral, the peripheral name should be the same with top verilog module which is implemented later. In the slave service and configuration section only the user logic software register is selected.

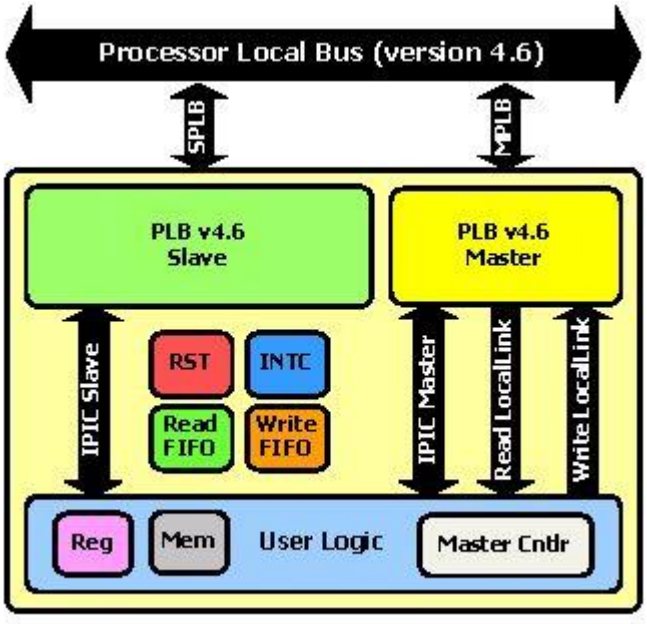


Figure 5.3: PLB interconnect scheme[7]

3 registers are enough for AND gate implementation because there is two input and one output. BFM simulation platform must be generated for the observation main communication signals between the MicroBlaze and the peripheral which is created by user.

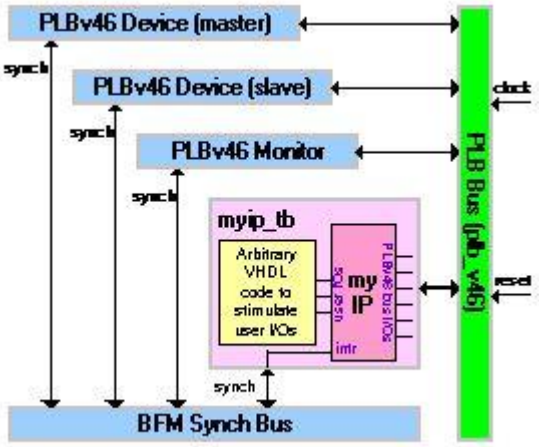


Figure 5.4: BFM simulation system architecture diagram[8]

There is another remarkable language option about the “user_logic” module which allows to user making new connections and some arrangements between soft core processer and the peripheral. Although the periheral template file is always written in VHDL language, the stub user_logic template file can be in Verilog for a mixed language design. All modules in this project are written in Verilog language so the “user_logic” template should be generated in Verilog language for the easiness.

```
1  `timescale 1ns / 1ps
2
3  module andKapi( input a,
4  input b,
5  output o);
6  assign o = a & b;
7  endmodule
```

Figure 5.5: Verilog code of AND gate

User_logic module should be edited for the making required connections accurately. Firstly the output should be defined as a wire. In this project o_wire element is driven by the AND gate's output o. The last bit of registers is used because 1-bit operation is enough for the observation bus signals. The third register(slv_reg2) is the combination of 31-bit zeros and the least significant bit is o_wire.

```
// Nets for user logic slave model s/w accessible register example
reg        [0 : C_SLV_DWIDTH-1]           slv_reg0;
reg        [0 : C_SLV_DWIDTH-1]           slv_reg1;
reg        [0 : C_SLV_DWIDTH-1]           slv_reg2;
wire       [0 : 2]                        slv_reg_write_sel;
wire       [0 : 2]                        slv_reg_read_sel;
reg        [0 : C_SLV_DWIDTH-1]           slv_ip2bus_data;
wire                                      slv_read_ack;
wire                                      slv_write_ack;
wire                                      o_wire;
integer                                   byte_index, bit_index;

// --USER logic implementation added here

  andKapi isim ( .a(slv_reg0[31]),
      .b(slv_reg1[31]),
      .o(o_wire));

 // implement slave model register read mux
 always @( slv_reg_read_sel or slv_reg0 or slv_reg1 or slv_reg2 )
   begin: SLAVE_REG_READ_PROC

     case ( slv_reg_read_sel )
       3'b100 : slv_ip2bus_data <= slv_reg0;
       3'b010 : slv_ip2bus_data <= slv_reg1;
       3'b001 : slv_ip2bus_data <= {31'd0,o_wire};
       default : slv_ip2bus_data <= 0;
     endcase

   end // SLAVE_REG_READ_PROC
```

Figure 5.6: User Logic implementation

After the all connections are made, the peripheral is imported by the implementation of following steps. Firstly the peripheral analysis order file is selected and the order must be respectively AND gate verilog module, user_logic verilog module and AND gate VHDL module. In this project there is no need to indicate the attribute of the interrupt signal. After these steps are accomplished the local peripheral core carried

into bus interfaces section because of making connection between custom IP and the MicroBlaze. Custom IP and the soft core processor are connected over PLB bus by the selection of bus mb_plb. For the addition of new peripheral address into MicroBlaze's address map the generation of address option is used. This design is exported to SDK after all the phases are done.

5.2.2 Xilinx Software Development Kit

5.2.2.1 Overview

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 [6]. As a tool includes some feature that it includes[5]:

1- Feature-rich C/C++ code editor and compilation environment

2- Project management

3- Supports environment for group working

4- Imports the hardware platform definitions generated by XPS

5.2.2.2 C Project

SDK allows to user creating a project in C language and sending information to the registers by using the header files of xparameters, platform and the custom IP which is automatically produced by XPS after the SDK exportation. This software's most significant feature is making developer's job easier by providing C language to control hardware system. For AND gate implementation we need two write register to define which two data will be written to input and one read register to save of output result. In the following code figure these registers can be seen easily.

```
ANDGATE_mWriteReg(XPAR_ANDGATE_0_BASEADDR, ANDGATE_SLV_REG0_OFFSET, 0x00000001);
ANDGATE_mWriteReg(XPAR_ANDGATE_0_BASEADDR, ANDGATE_SLV_REG1_OFFSET, 0x00000000);

cikis = ANDGATE_mReadReg(XPAR_ANDGATE_0_BASEADDR, ANDGATE_SLV_REG2_OFFSET);
```

For programming FPGA over SDK three type files are needed. These are Bitstream, BMM file and the ELF file. ELF file is used for the initialization Block RAM and it is a common standard file format for executable files, object code and shared libraries.

After coding part is done firstly linker script is generated to provide protection and organization of all related files before the elf file update. In this project elf file is going to be used for simulation in XPS and programming FPGA.

6. HARDWARE DESIGN

6.1 Communication between FPGA and the computer

One FPGA board is planned to communicate with the computer. The communication method is selected as Universal Asynchronous Reciever/Transmitter (UART) protocol. Usually, RS232 cable is used for the UART communication. This cable enables the transmission of data by serial communication.

Spartan-6 FPGA which is used in this project does not have the RS232 socket so the information sharing is made by using Rx and Tx ports to send and receive data.

6.1.1. UART protocol

The UART performs serial-to-parallel conversions on data received from a peripheral device and parallel-to-serial conversion on data received from the CPU. The CPU can read the UART status at any time. The UART includes control capability and a

processor interrupt system that can be tailored to minimize software management of the communications link[9].

Sending serial data on a single line accurately needs some control to be applied. Therefore UART protocol, except the data bits, has a parity bit which is optional, a stop bit and a start bit. When there is no data to send the line is in idle case. Idle case terminates when the start bit is seen and sends data until the stop bit is raised. Parity bit is optional that comes before the end bit. Tx line is used to transmit data and Rx is to receive data. Every single bit of data is sent according to the Baudrate which is the rate of data sent per second. Baudrate is generally set to 9600.

6.1.2. SDK Terminal Configurations

After the FPGA programmed the results should have seen from the terminal. Firstly the vcp driver kit must be installed for the UART communication over usb. When the board is connected to computer, driver allows the computer to see UART output as a virtual com port. From the terminal settings connection type selected as serial, Baudrate set to 9600 and port selected as COM3 with encoding ISO-8859-1. When all the changes are done the result is seen from the SDK's terminal window.

6. RESULTS

6.1 AND Gate Implementation Simulation Results

After the Bus2IP (master-to-slave) becomes logically 1 the values which are sent from SDK is carried to the AND gate's input. Here a, b and o represents AND gate's inputs and the output. The following figures show exactly what is happening between IP and MicroBlaze in terms of communication.

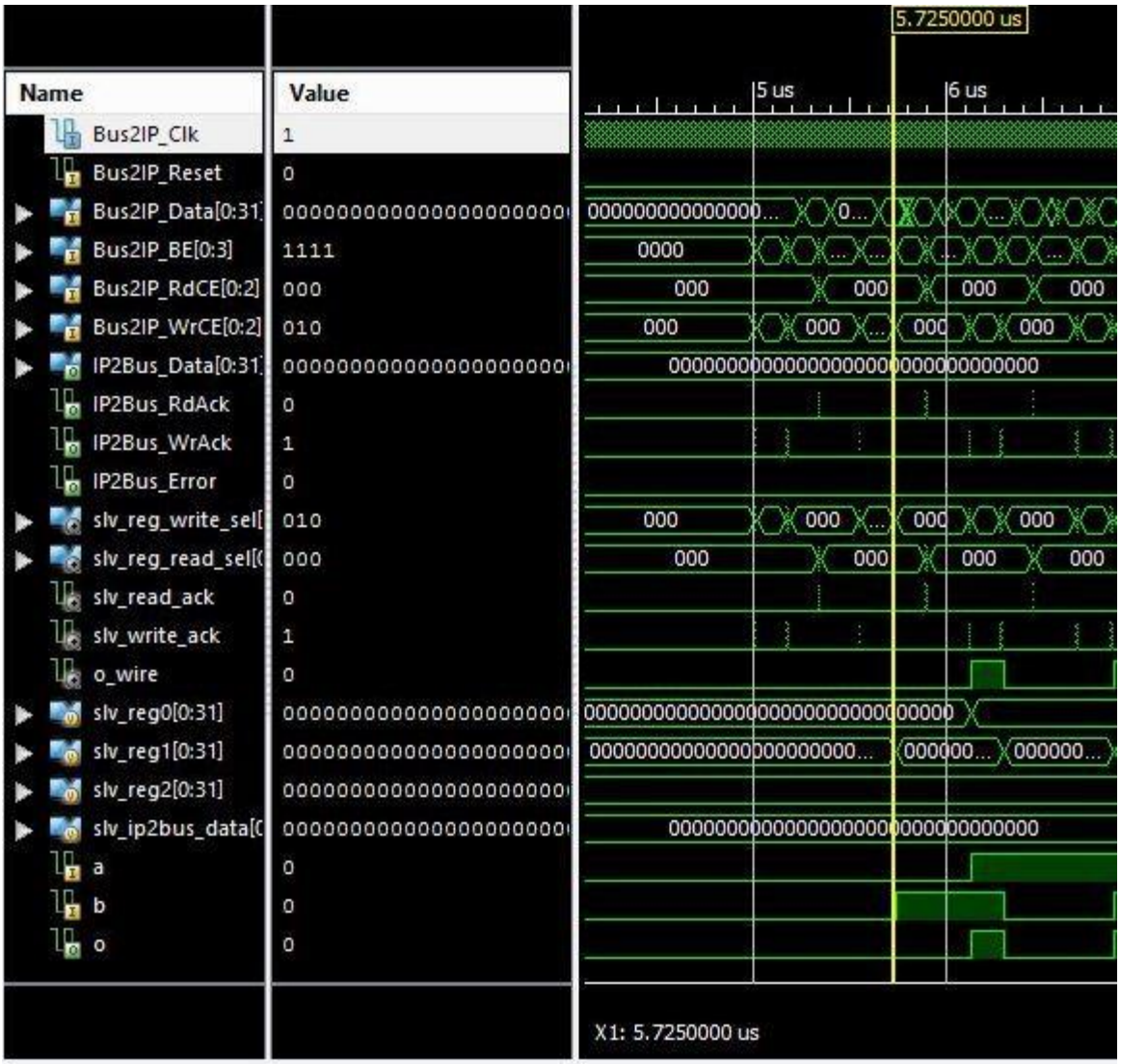


Figure 6.1: Simulation results just before the information come to the AND gate

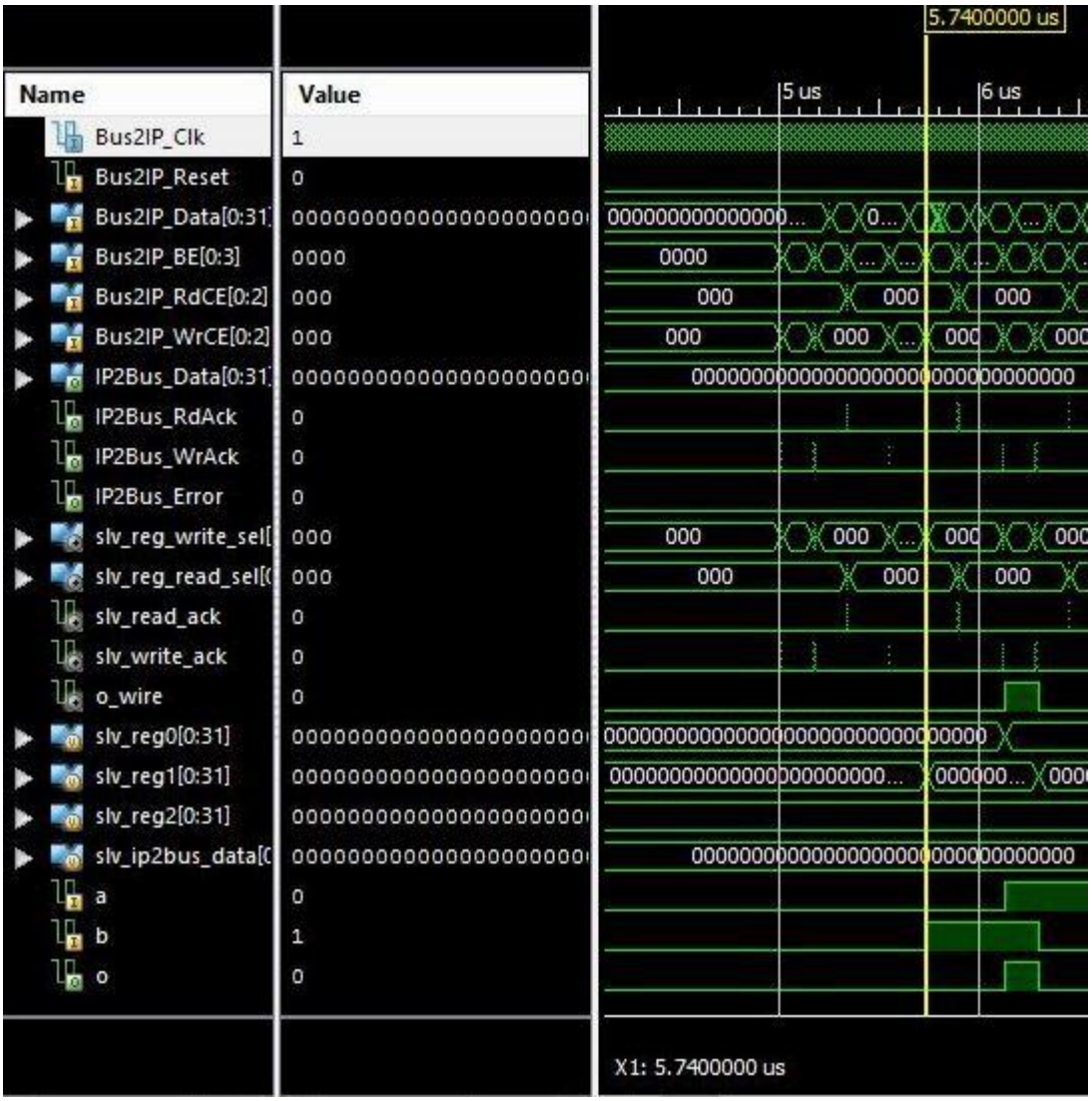
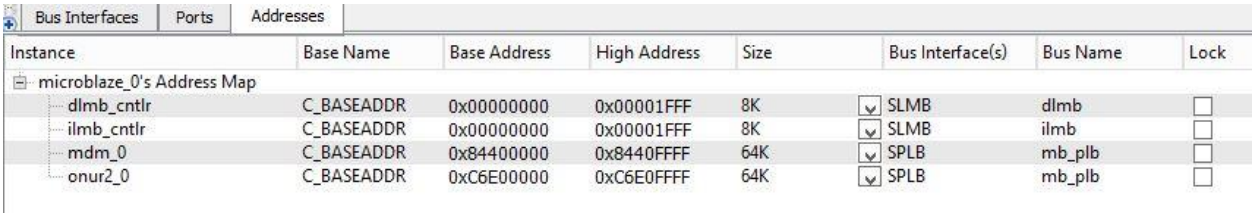


Figure 6.2: Simulation results when the information is carried to the AND gate

In EDK the base and high addresses are generated for the custom IP. The base address must be lower than the high address it can be seen from following figure.



Bus Interfaces | Ports | Addresses

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
dlmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	dlmb	
ilmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	ilmb	
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb	
onur2_0	C_BASEADDR	0xC6E00000	0xC6E0FFFF	64K	SPLB	mb_plb	

Figure 6.3: Base and High addresses values

Device Utilization Summary (actual values)			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	1,434	18,224	7%
Number used as Flip Flops	1,427		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	7		
Number of Slice LUTs	1,783	9,112	19%
Number used as logic	1,616	9,112	17%
Number using O6 output only	1,318		
Number using O5 output only	42		
Number using O5 and O6	256		
Number used as ROM	0		
Number used as Memory	130	2,176	5%

Figure 6.4: Design summary of AND gate module

6.2 Diffie-Hellman Key Exchange Protocol Implementation Results

The alice function calculates the $A\hat{}B \ mod \ N$ and $A\hat{}X \ mod \ N$ and they are assigned to $sonuc\_a$ and $sonuc\_b$ respectively. These parameters are called again instead of integer A in alice function as the protocol required and they are assigned to $result\_alice$ and $result\_bob$. Xil_printf is used because it takes really small space in the memory by comparison with printf. By the following it can be seen final results are equal to each other and the key exchange protocol successfully implemented.

```
int A=64234;
int B=63788;
int X=62356;
int N=497;
int sonuc_a;
int sonuc_b;
int result_alice;
int result_bob;


int main()
{

    init_platform();

    sonuc_a=alice(A,B,N);
    sonuc_b=alice(A,X,N);

    result_alice = alice(sonuc_b,B,N);
    result_bob = alice(sonuc_a,X,N);

    xil_printf("result_alice = %d \r\n",result_alice);
    xil_printf("result_bob = %d \r\n",result_bob);

    cleanup_platform();

    return 0;
}
```

Problems | Tasks | Console | Properties | Terminal 1

```
Serial: (COM3, 9600, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
result_alice = 25
result_bob = 25
```

Figure 6.5: Protocol results screen from the SDK

For the memory optimization dlmb_cntlr and ilmb_cntlr sizes are increased to 16 kilobyte from 8 kilobyte. Basically longer code means bigger memory size need. Uart addresses are also generated for this time it can be seen from the figure below.

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
dlmb_cntlr	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	dlmb	
ilmb_cntlr	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	ilmb	
RS232_Uart_1	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	mb_plb	
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb	
addersub_0	C_BASEADDR	0xCE200000	0xCE20FFFF	64K	SPLB	mb_plb	

Figure 6.6: Base and High addresses and sizes of instances

Device Utilization Summary (actual values)			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	1,634	18,224	8%
Number used as Flip Flops	1,627		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	7		
Number of Slice LUTs	2,025	9,112	22%
Number used as logic	1,856	9,112	20%
Number using O6 output only	1,528		
Number using O5 output only	42		
Number using O5 and O6	286		
Number used as ROM	0		
Number used as Memory	141	2,176	6%

Figure 6.7: Design summary of adder-subtractor module

According to the results, the numbers A, B and X were chosen as appropriate for 16 bits. However, when this length is increased, it will become very difficult for a third person to find the key. In the future, as more security solutions require longer ciphers, this 16-bit limited design is likely to be an example for future larger designs.

REFERENCES

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

[2] **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

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

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

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

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

[7] **Xilinx Custom IP** (n.d.). Retrieved May 22, 2017, from < https://www.xilinx.com/support/documentation/ip_documentation/plb_v34.pdf>

[8] **Xilinx BFM Simulation** (n.d.). Retrieved May 21, 2017, from < https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/bfm_simulation.pdf>

[9] **Wakerly, John F.** (1999). Digital Design Principles and Practices(4th ed.). United States of America: Pearson Education Inc., pp. 33-34, 391-393

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