In the name of ALLAH,
the most Beneficent, the most Merciful

IMPLEMENTATION OF RSA ALGORITHM SECURE AGAINST TIMING ATTACKS

Submitted by:
Muhammad Aqeel Aslam

Supervisor:
Dr. Athar Mahboob

Thesis
Submitted to
Department of Electronic and Power Engineering,
College of Marine Engineering (PNEC), Karachi
National University of Sciences and Technology, Islamabad

In Partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
With Specialization in Communications

March 2012

Abstract

Data security is an important aspect of information transmission and storage in an electronic form. Cryptographic systems are used to encrypt such information to guarantee its security. To retrieve such information, the encrypted form must be first decrypted. One of the most popular cryptographic systems is the RSA public key crypto system. The larger the RSA public modulus size, the stronger will be the RSA cryptosystem. Unfortunately, the RSA is extremely vulnerable to timing attacks which can deduce the private RSA exponent due to regularity of operations in the straight forward implementation of exponentiation using the square and multiply method or its variants. Timing attacks constitute a major threat to the all systems using RSA and hence, implementations must be protected. The work reported here proposes Secure Implementation of RSA algorithm against timing attacks. This implementation is done using Verilog HDL and targeting Xilinx FPGA devices.

The “Secure Implementation of RSA Algorithm against Timing Attacks” is done by using the output of a Random Number Generator, and blinding the exponentiation operation. The Exponentiation is done by using Montgomery Reduction scheme which is the fastest known technique of exponentiation. It does not require division, as the division in hardware is quite complex. The random number generator is implemented using a Barrel shifter and to nullify the effect of random number we implemented the Extended Euclidean Algorithm which provides modulo inverse of the generated random number. To hide the operation time we multiply the random number with the private key. After exponentiation the result is then further multiplied with the inverse of the generated random number. By doing so the timing of the algorithm is changed and hence, the desired result is achieved.

Acknowledgment

With the grace of Almighty ALLAH, the most merciful and the kindest, who blessed me with wisdom, knowledge and health to complete the master thesis which He always bestowed on me. I am very thankful to ALLAH who gave me courage and strength to complete this thesis after hard work of the course. I pray that, ALLAH may shower His countless blessings and peace upon the last Holy Prophet Peace be Upon Him (P.B.U.H).

I am very thankful to my thesis supervisor Dr. Athar Mahboob, and I would like to express my sincere gratitude to him for his kind and continuous guidance with great courage and patience. I am very grateful to all GCE members and especially Dr. Arshad Aziz for his timely guidance and advice that really helped me through difficult situations.

I am very thankful to my mother, brothers and sister who encouraged me during the MS program. Last but not the least, I am also very obliged to my all class mates and seniors especially, Kashif Latif who helped me a lot during the thesis phase.

Contents

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

- Half Adder (HA)
- Full Adder (FA)
- Ripple Carry Adder (RCA)
- Carry Save Adder (CSA)
- Carry Look Ahead Adder (CLA)
- Partial Product (PP)
- Partial Product Array (PPA)
- Partial Product Reduction (PPR)
- Simple Power Analysis (SPA)
- Differential Power Analysis (DPA)
- Multiplexer (MUX)
- Exclusive-OR (XOR)
- Greatest Common Divisor (GCD)
- Field Programmable Gate Array (FPGA)
- Direct Sequence Spread Spectrum (DSSS)

CHAPTER 1
INTRODUCTION

1.1 Background

Encryption is a well recognized technique for the protection of data and information. It is used effectively to protect sensitive data. The transfer of data from one form to another form, which is unreadable without the secret key, is called encryption. Several techniques have been used for many years. Cryptographic systems are classified into two main categories secret key cryptosystem and public key cryptosystem. Only one key is involved in Secret key cryptosystem. This key is used for both encryption and decryption. However, public key cryptosystem uses two different keys one for encryption and other one for the decryption.

Day by day the significance of FPGA (Field Programmable Gate Array) is increasing. FPGAs are playing very important role in commercial area and research area as well. The technology of FPGA is more affordable as compared to ASICs. FPGAs are reconfigurable platforms. The choice of reconfigurable platforms for cryptographic algorithm appears to be practical and it provides high speed in applications.

In 1978 RSA algorithm was first developed by Rivest, Shamir and Aldeman. RSA is an example of public key algorithm. If the modulus size is large, the algorithm is secure. This algorithm is computationally intensive and it operates on very large integers. RSA algorithm can be used for encryption / decryption and digital signatures. RSA is the most popular method for the public key cryptosystems. The key size determines the security of RSA algorithm. The larger is the key size, more is the security.

1.2 Thesis Motivation

The motive of this thesis is to develop a secure implementation of RSA algorithm against timing attacks. Kocher [2] in his paper at RSA Data Security and Crypto first discussed the timing attacks in 1996. Timing attacks exploit the information about the time variation during the cryptographic operations. Cryptographic algorithm operations take different amount of time for every logical operation [2]. The idea behind this thesis is to secure RSA cryptographic algorithm against timing attacks.

1.3 Thesis Organization

The thesis presents the Implementation of RSA Algorithm Secure against Timing Attacks.

Chapter 2 presents the basic concept of RSA algorithm.

Chapter 3 discusses the efficient method to implement arithmetic operations which includes addition and multiplication on reconfigurable platform.

Chapter 4 chapter provides a vast knowledge on modular exponentiation. It also includes different techniques which are being used.

Chapter 5 gives the fundamental concepts of side channel attacks and how we can protect cryptographic modules against such attacks.

Chapter 6 discusses the hardware implementation of proposed architecture of RSA algorithm against timing attacks.

Chapter 7 presents the basic knowledge of FPGA and it also summarizes the advantages of FPGA over ASICs.

Chapter 8 shows the synthesis and simulation results of our implementation.

Chapter 9 gives the conclusion and possible future advancements.

CHAPTER 2
THE RSA ALGORITHM

This chapter discusses the details of the RSA Algorithm. The basic concept of the RSA Algorithm is explained in the following sections. RSA Encryption / Decryption, RSA Security and RSA example is stated in the following sections in order to develop better understanding of the proposed work.

2.1 Introduction

RSA is an algorithm, which is used for encryption and authentication. The security of RSA algorithm relies on a problem which is known as Integration Factoring Problem (IFP). RSA is a public key crypto system, it can encrypt and decrypt with the same function. RSA involves two keys, namely public key and private key. Public key is made publically available while the private key is kept secret [1][3]. For encrypting the information public key is used whereas for decryption process private key is used.
RSA algorithm involves computation complexity of factoring very large prime numbers. In symmetric key cryptosystem two problems can occur. Fist the key exchange between sender and receiver may be unsafe. Second problem is that no digital signature is available in secret key cryptosystems. Asymmetric cryptosystem overcomes these two problems.

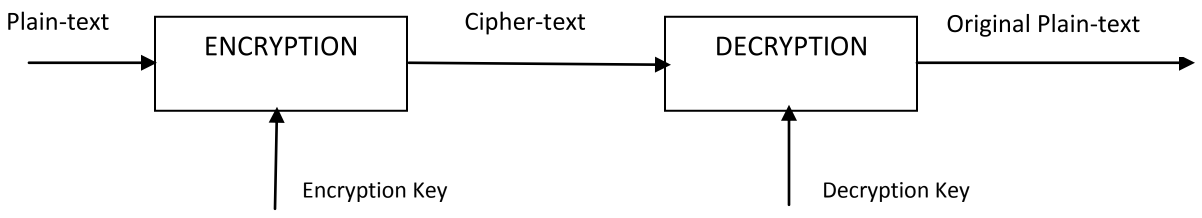

Figure 2.1 Public Key algorithm Encryption and Decryption with different keys

2.2 RSA Key Generation

The following steps are involved in the generation of RSA key.

1. Two prime numbers p and q are generated first.
2. Calculate *n*

$$n = p * q$$

3. Calculate ø *(n)*

$$ø(n) = (p-1)(q-1)$$

4. Now, select a number such that 1 < *e* <*ø(n)*

$$gcd\ (e, ø(n)) = 1$$

and compute *d* with

$$d = e^{-1}\ mod\ ø\ (n)$$

5. Public Key = {*e*,*n*}and Private Key = {*d*,*n*}

2.3 RSA Encryption

The following formula is used for the calculation of encryption.

$$C = P^e\ mod\ n$$

Where *C* is the Ciphertext, and *P* is the Plaintext.

2.4 RSA Decryption

The decryption is calculated by the following formula.

$$P = C^d mod\ n$$

2.5 RSA Example

For example pick p =11, q = 13 and compute.

$$n = p * q = 11 * 13 = 143$$

$$ø(n) = (p-1)(q-1) = 10 * 12 = 120$$

For example *e* = 17. The private component *d* is computed by:

$$d = e^{-1}(mod\ ø(n))$$

$$d = 17^{-1}mod\ (120)$$

$$d = 113$$

2.6 RSA Security

RSA gets is difficulty from the factorization. The basis of RSA security is factoring large prime numbers. RSA Problem is also the fundamental part of the security of the RSA. The condition of RSA problem makes sure that there is exactly only one unique *n* in the field. It is difficult to determine *ø(n)*. Without the knowledge of *ø(n)*, it would be hard to derive *d* based on the knowledge of *e*. This algorithm is used in security protocols. The following table shows some of the applications where RSA is used to provide security [1,4].

Table 2.1 Applications of RSA Algorithm

ISPSEC / IKE	IP Data Security
TLS / SSL	Transport Data Security
PGP	Email Security
SSH	Terminal Connection Security
SILC	Conferencing Service Security

CHAPTER 3
ARITHMETIC ON HARDWARE

This chapter discusses the hardware implementation of arithmetic units. In this thesis we used unsigned addition and multiplication. Most of the cryptographic algorithms require some of the operations to be executed repeatedly. It is very important that these units should be implemented very efficiently.

3.1 Addition

Addition plays an important role in the cryptographic algorithm. RSA speed depends upon the speed of the addition unit. When we add two bits sum and carry bits are generated, the carry bit is used in addition with next two input bits, in that way this carry bit is propagated from LSB to MSB. Therefore, the design of adder should be such that it will minimize the delay. To minimize this delay we use fast adders.

3.1.1 Types of Adder

To add single bit we have the following two types of adder.

3.1.1.1 Half Adder

Two inputs namely A and B are used in half adder, with two outputs namely sum and carry out. Sum is obtained by implementing the logical operation of XOR to the two inputs A and B, whereas the carry output is attained by performing the logical operation of AND to the inputs A and B.

$$S = A \oplus B$$

$$C = A \bullet B$$

We cannot use this adder because when multi bits are used, the half adder does not include the carry output.

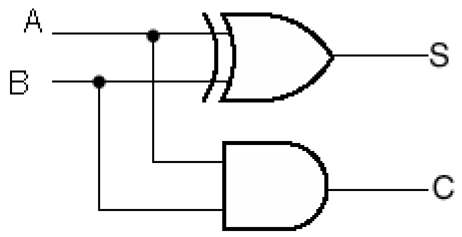

Figure 3.1 Half bit adder

3.1.1.2 Full Adder

Full adder consists of three inputs A, B, Carry-in and it produces two outputs Sum and Carry-out. Full adder can be used for the multi bit addition. Full adder can connected to the other full adder in order to achieve the multi bit addition. The sum and carry out for full adder is obtained by the following formula.

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = (A \bullet B) + (C_{in} . \bullet (A \oplus B))$$

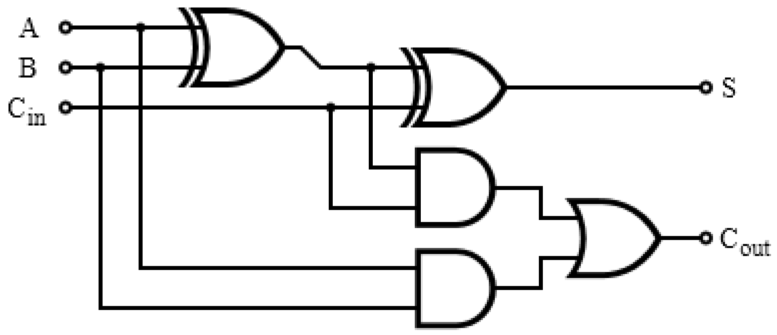

Figure 3.2 Full bit adder

3.1.1.3 Ripple Carry Adder

Multiple full bit adders used a logical circuit to add *N*-bit numbers. Each full adder uses C_{in} as the input, which was the output of the last adder. The response time of ripple carry adder is slow, as it has to wait for the carry bit to be calculated from the previous full adder. Each full adder requires three levels of logic. In a 32 bit ripple carry adder, the critical path delay is equal to 65 gate delays.

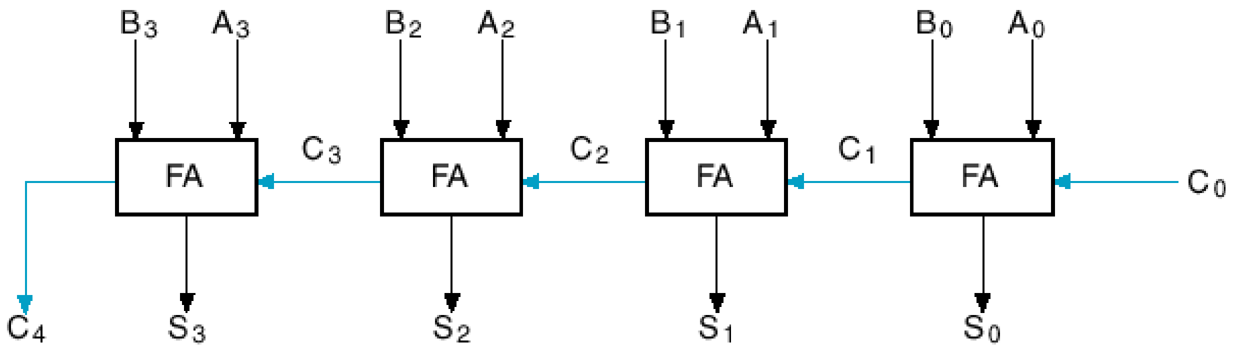

Figure 3.3 Ripple carry adder

3.1.1.4 Carry Look Ahead Adder

Carry look-ahead adder reduces the computation time. A cell will generate a carry if both of cell's data bits are 1. A cell will propagate a carry if either of the data bits and carry-in of the cell are 1. The generate g_i and propagate p_i are defined as:

$$g_i = a_i \bullet b_i$$

$$p_i = a_i \oplus b_i$$

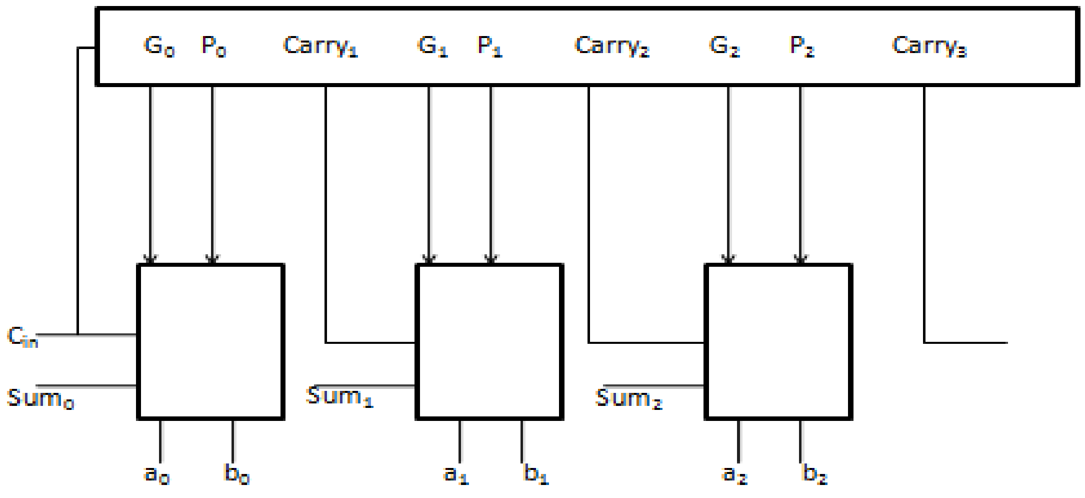

Figure 3.4 Carry Look Ahead Adder

3.1.1.5 Carry save Adder

Currently Carry Save Adder (CSA) is the most widely used adder to implement the fast arithmetic on the hardware platform. It save the carry generated at each stage rather than propagate to the next adder. Each stage of adder produces two outputs sum(s) and carry(c).

3.1.1.6 Comparison between Ripple Carry Adder and Carry Look Ahead Adder

The following table shows the comparison between Ripple Carry Adder and Carry Look Ahead Adder.

Table 3.1 Comparison between Adders

S. No	Ripple Carry Adder	Carry Look Ahead Adder
1.	It is a straight forward way of adding two or more than two bits.	Carry Look Ahead Adder calculates the carry bits before the sum.
2.	Wait time is much more because carry is calculated alongside sum.	It reduces the wait time.
3.	In 16 bit Ripple Carry Adder, we find CPU : 2.89 / 3.06 s | Elapsed : 3.00 / 3.00 s	In 16 bit Carry Look Ahead Adder we get CPU : 2.90 / 3.07 s | Elapsed : 3.00 / 3.00 s
4.	In Ripple Carry Adder every carry out has to be carry in of the next stage.	Carry Look Ahead Adder uses the logic of generating and propagating carries.

3.2 Multiplication

Multiplication plays an important role in arithmetic units. Multiplication can be constructed by sequential circuit or by combinational circuits. Combinational circuit is faster than the sequential circuit but sequential multiplier require more clock cycles for the required operation. Combinational circuit requires more area or silicon hardware.

Multiplication of two digital numbers is performed by addition and shifting of bits. Considering the multiplication of two numbers, the multiplicand is *a* with there presentation of [a_1,a_2 ,a_3 , ……………., a_n] and multiplier *b* with the representation [b_1 ,b_2 , b_3 ,…………….., b_n] both are *n* bit numbers. The resultant of the product is a and *b* is *2n* bit wider. The entire steps of the multiplication are the following.

(i) Partial Product Array Generation
(ii) Partial Product Array Reduction
(iii) Final Addition

It can be represented by the following figure.

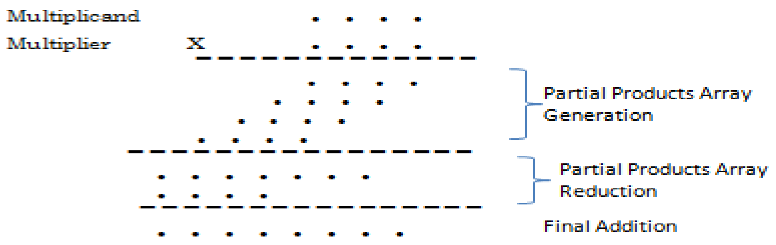
Multiplicand
Multiplier
X
Partial Products Array Generation
Partial Products Array Reduction
Final Addition

Fig 3.5 Digital Multiplication Flow

3.2.1 Partial Product Array Generation

The first step in the digital multiplication we have to generate n shift copies of the multiplicand. This implementation can be implemented by using a logical and operation which performs AND *(a_i , b_j)* where *i* and *j* = 0 to *n*-1. This result is known as partial products. If we arrange these products by their corresponding bit positions we may get the following figure. These partial product bits are arranged in column to attain the final value. This trapezoidal structure is known as Partial Product Array.

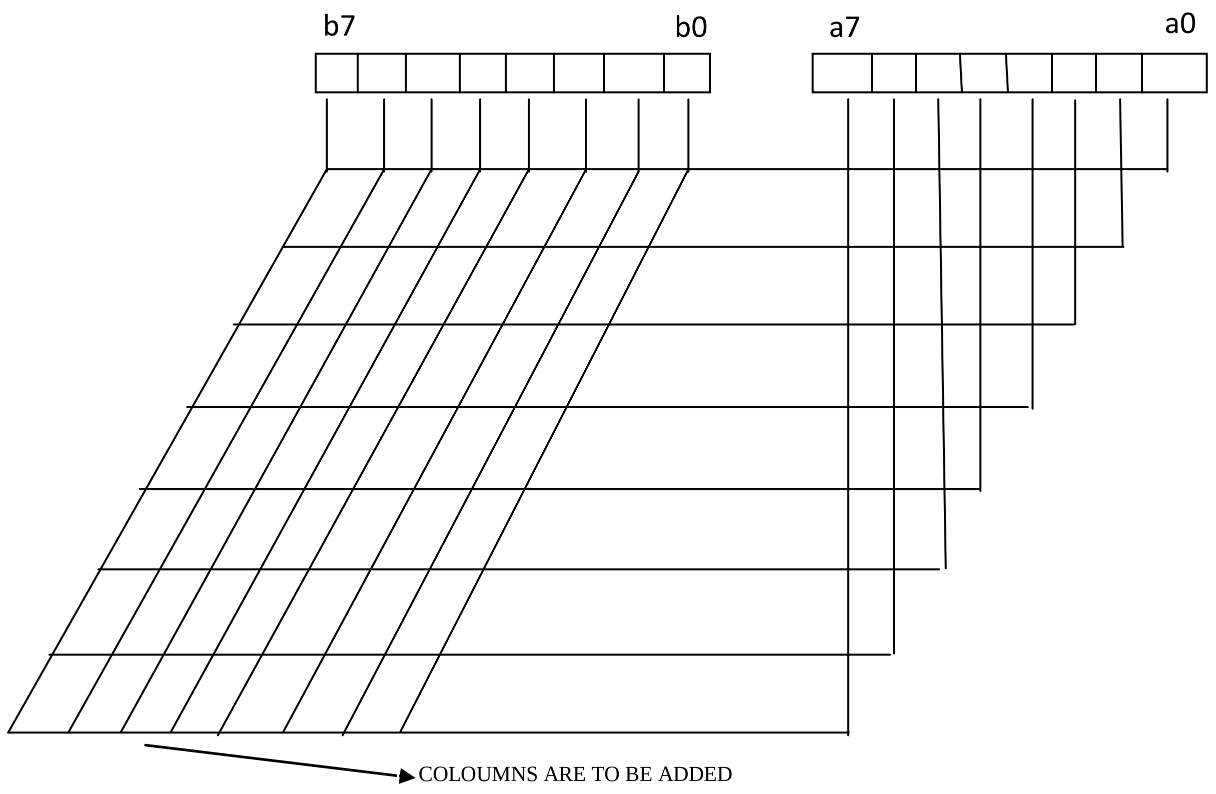

Figure 3.6 Partial Product Generation of 8*8 bit Multiplication

3.2.2 Partial Product Array Reduction

The efficiency of multiplier depends upon the manner in which PPA bits are added. One method is to add two partial products at the same time and their result is added to the next partial product. In this way all the partial products are added. This is a very insufficient way in hardware implementation. Each addition process contributes a delay in the final addition. This makes the multiplier very slow. To reduce the critical path reduction techniques are used. Fast adders can be used to reduce n number of iterations in to 2. The following techniques are used for the reduction purposes along with the fast adders.

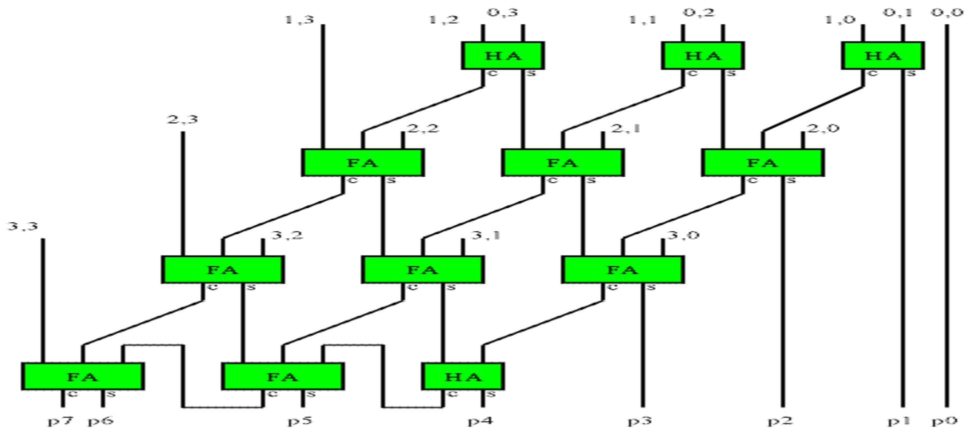

Figure 3.7 Partial Products of 4*4 bits

3.2.2.3 Wallace Data Tree Reduction Scheme

The reduction is performed in parallel in groups of 3s. As the number of partial products increases the size of the multiplier also increases. Each adder level incurs one Full Adder (FA) delay in the path. The larger the number of adder levels, the bigger will be the critical path of the combination cloud of the Partial Product Reduction (PPR) unit. Following figure shows the 4 x 4 example of Wallace Tree Reduction Scheme.

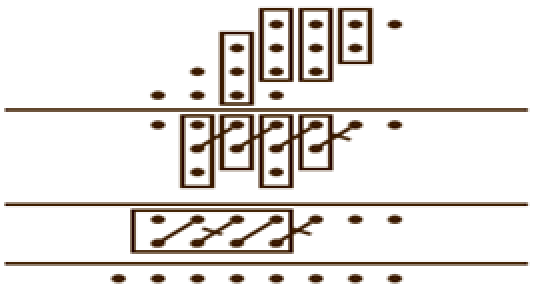

Fig 3.8 Wallace Tree Reduction 4*4 Example

3.2.2.4 Dadda Tree Reduction Scheme

The compression rate of dada tree reduction scheme is similar to the Wallace tree reduction scheme. Dadda tree reduction unit results in an optimal number of hardware blocks. The architecture of dadda tree reduction scheme is simple to implement but it exhibits the larger critical path than the Wallace tree reduction scheme.

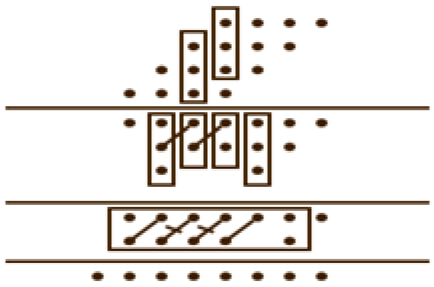

Fig 3.9 Dadda Tree Reduction 4*4

The following table 3.2 shows the number of full adders according to the number of partial products.

Number of Partial Products	**Number of full adder levels**
3	1
4	2
$5 \leq n \leq 6$	3
$7 \leq n \leq 9$	4
$10 \leq n \leq 13$	5
$14 \leq n \leq 19$	6
$20 \leq n \leq 28$	7
$29 \leq n \leq 42$	8
$43 \leq n \leq 63$	9

3.3 Final Addition

The final addition has to be done by fast adder like carry save adder, carry look-ahead adder or any other carry propagate adder is used for purpose. The addition of 512 bits or more require a very time consuming operation. A special attention should be paid on the multiplier architecture because if the adder is a slower one than it may contribute a significant delay in the critical path. CLA / CSA tree itself can be used for this purpose.

CHAPTER 4
SIDE CHANNEL ATTACKS ON CRYPTOSYSTEM

4.1 Introduction

Side channel attacks can destroy any cryptosystem. Cryptosystem takes slightly different amount of time to process different inputs. In side channel attack the information can be retrieved from the encryption device that is neither the plaintext to be encrypted nor the cipher-text resulting from the encryption process. In a side channel attack an attacker attempts to compromise a crypto system by analyzing the time required to execute each operation. Each logical operation requires some time for execution [3]. An attacker can work backward to find the input by the precise measurement of time. It is generally agreed that RSA is secure from direct attack. Performance characteristic of the cryptosystem depends upon the encryption key and data input.

4.2 Side Channel Attacks

Side Channel attacks comprises of following classes.

4.2.1 Timing Attacks

Timing attacks are a form of side channel attacks. Timing attacks are based on measuring time it takes for a unit to perform operation [2]. Generally, encryption system requires different processing time for different inputs. Some attacks can exploit the timing measurements to find the entire secret key. Timing attacks can be used potentially against any cryptosystem including symmetric functions.

Calculating variances is fairly simple. It provides a improved way to make out correct bits. The number of samples will allow recovering the information by the properties of signal and noise. If noise is too much, than, more samples will be needed to find the actual information and the secret key.

4.2.2 Power Consumption Attacks

Those attacks which are based upon the analyzing of power consumption of the unit when it is performing encryption operation. An analyst can take out secret information such as undisclosed keys. Simple Power Analysis (SPA) involves visual examination of different graphs of the current used by a device over a period of time. As a device performs different operations variation in power consumption occurs [5]. The implementation of squaring and multiplication in RSA can be distinguished. An attacker can learn about the processes that are occurring inside the unit and can get the required information.

Differential Power Analysis (DPA) involves statistically analyzing power consumption measurements from a cryptosystem. Differential Power Analysis uses error correction and signal processing.

4.2.3 Electromagnetic Attacks

These are the attacks which are based upon the leakage of the electromagnetic radiation. It can provide directly plain-text and other information as well. Such measurements can be used to infer cryptographic keys using techniques equivalent to those in power analysis, or it can be used in non-cryptographic attack.

4.2.4 Acoustic Cryptanalysis

These are the attacks which exploit the sound produced during the computation rather than the power analysis.

4.2.5 Cache Attacks

This kind of attacks reveals the information between the processes of computer.

4.2.6 Differential Fault Analysis

This type of attacks often reveals the information by introducing faults in the computation. All types of attacks involve the physical effects. Useful secrets and information about the system can be attained by the operation performed by the cryptosystem. This information may include cryptographic key, partial state information, full or partial plaintext and so on. The term secret degradation is sometimes used to express the degradation of secret key material resulting from side channel leakage.

4.3 Preventing Side Channel Attacks

The ability to resist side channel attacks the designer of cryptographic modules should use any of the following techniques in order to make the cryptosystem more reliable. Every operation performed by the module should be data independent in their time consumption. Whenever, different sub operations are performed according to the inputs, they should require same number of clock cycles. Time required to perform operation should be fixed for every piece of data, this will exclude the all possibilities of the timing attacks.

4.3.1 Constant Exponentiation Delay

This technique is also used for the prevention of side channel attacks. In this technique we make sure that all the exponentiations takes the same amount of time before providing the result [2]. This is the simplest way of preventing data from side channel attacks but it degrades the overall performance of the algorithm as each operation requires different amount of time to execute but if we implement this technique each and every operation would require same amount of time.

4.3.2 Random Delay

Random delay technique provides better performance of the algorithm as compared to the constant exponentiation delay. In this technique we add random delay to the exponentiation in order to confuse the timing attacks. Kocher [2] points out in his paper that if we do not add enough noise then the observer can still succeed by collecting additional measurements which were made for the compensation of the random delay.

4.3.3 Blinding

Blinding is far better technique than constant exponentiation delay and random delay. It is the most widely accepted method in the defense of RSA. Blinding prevents the side channel attacks on encryption system [34].

RSA blinding introduces the randomness. Blinding makes the timing information unusable. To recover the input information of the cryptographic algorithm, a cryptanalysis gathers the algorithm's result. Blinding serves to alter the algorithm's input into some unpredictable state. Blinding can prevent some or all leakage of useful information which can be used for finding the actual inputs of the cryptographic algorithm. Blinding can be employed against simple timing attacks. Ideally, a random value $r \in Z_N$ is chosen for each signing operation. We compute $(mr)^d$ mod n, instead of computing m^d mod n. Then, multiply the result with r^{-1} mod. By doing so, the cryptanalyst can no longer choose the messages being input to Montgomery Multiplication algorithm.

CHAPTER 5
HARDWARE IMPLEMENTATION

5.1 Introduction

This chapter includes the hardware implementation of the RSA Algorithm against Timing attacks. The RSA algorithm involves addition, modular multiplication and modular exponentiation. This chapter also includes the algorithms involved in the implementation of RSA.

5.2 Addition

Addition is the basic part of the Arithmetic Logic Unit (ALU), which is used in all other operations. The overall performance depends upon the speed of addition. We started our implementation by using Ripple Carry Adder. After some further literature review it came into notice that Ripple Carry Adder has some drawbacks in terms of delay. Therefore, we switched on to Carry Look Ahead Adder, which has less delay while propagating carry. As already mentioned, that addition operation is the most important and fundamental unit and it plays a major role in all operations. The delay effect is minimized by CLA. First, we made a Carry Look Ahead adder of 4 bits, we instantiate it to make a 16 bit adder. We instantiate these 16 bit adder to form the Carry Look Ahead Adder of 64 bits and then these 64 bit adders cascaded to make the required adder of 512 bits.

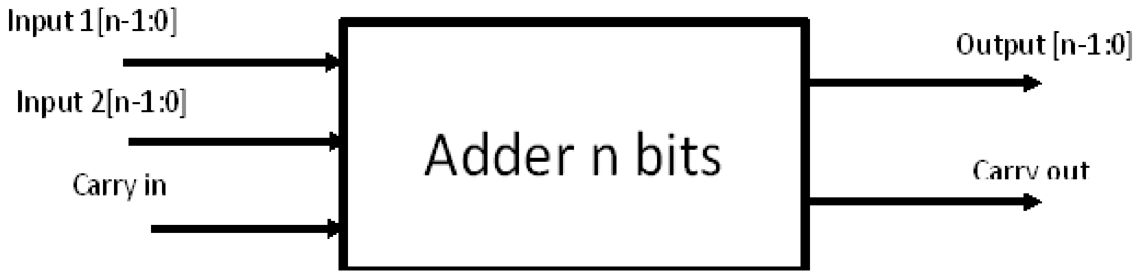

Fig 5.1 Addition of n bits

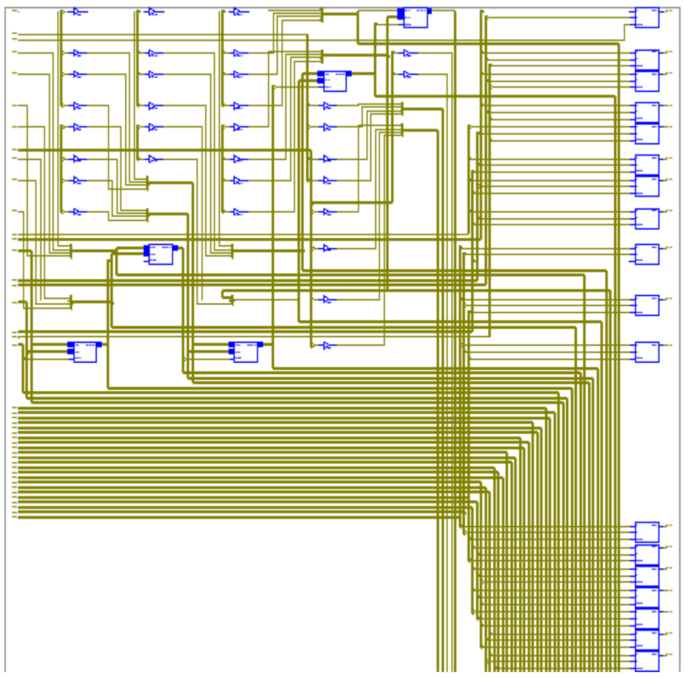

Figure 5.2: RTL schematic of 64 bits Addition

The results of the Ripple Carry Adder and Carry Look ahead Adder has been shown in the following figure. A comparison between Ripple Carry Adder and Carry Look Ahead Adder has already shown in the section 3.1.1.6. Figure 5.2 shows the schematic diagram of 64 bit addition using CLA.

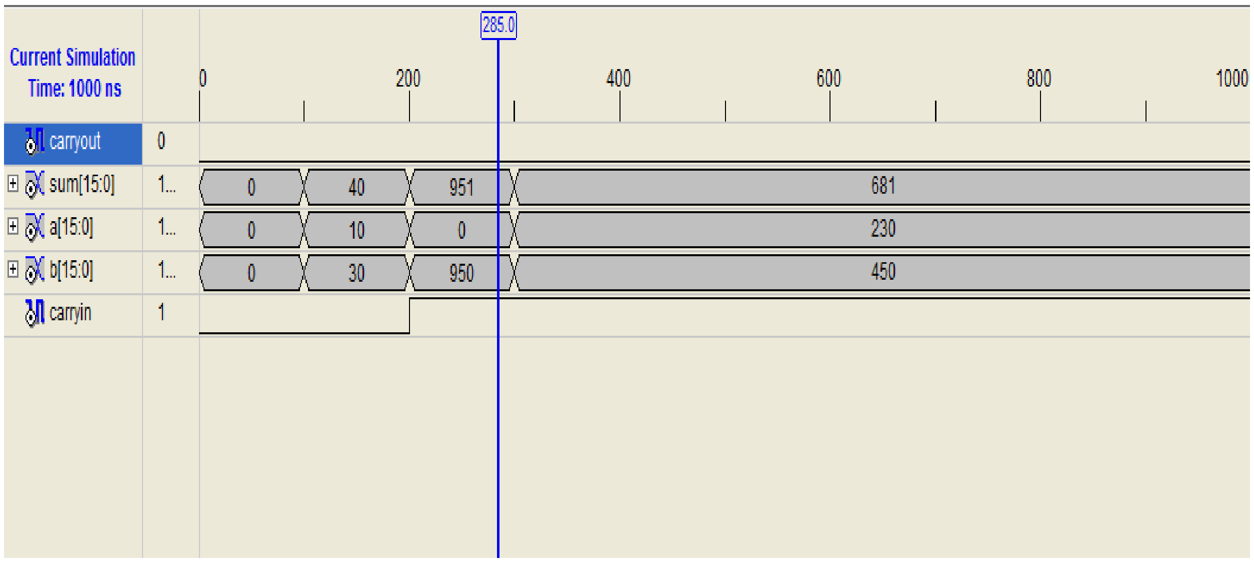

Figure 5.3: Simulation Result of 16 bit Carry Look Ahead Adder

Figure 5.3 shows the result of 16 bit CLA. There are two inputs namely A and B, whereas carry in is the bit which is of one bit and this bit is generated by the sum of the two inputs. Carry in is used as the third input, sum and carry output are the results of the addition. In figure 5.3, we have taken different sets of inputs and getting the results in terms of sum and carry output. Carry in will be high whenever the sum of the inputs A and B are of two bits or of more than two bits. Each and every operation requires 100 ns.

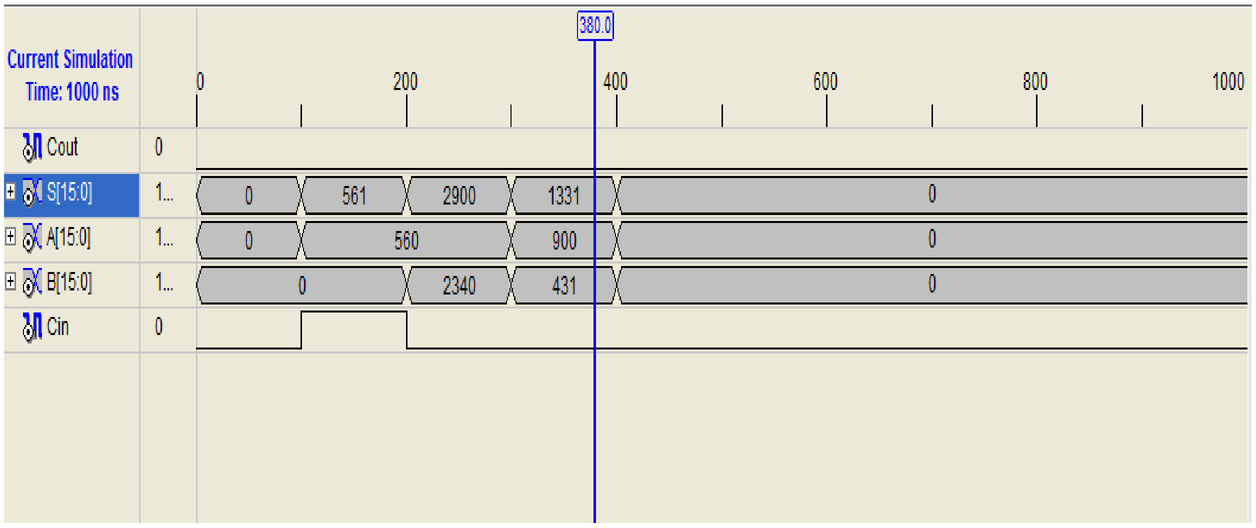

Figure 5.4: Simulation Result of 16 bit Ripple Carry Adder

Figure 5.4, shows the simulation result of 16 bit Ripple Carry Adder. CLA and RCA both are fast adders. But RCA has some delay as compared to CLA. A and B are the two inputs, whereas Cin is the carry in. Carry out and Sum are the results. Carry out is of 1 bit whereas the sum is of 16 bit. Each operation requires 100ns.

5.3 Modular Multiplication

Multiplication was done by suing Shift and Add method. In Modular Multiplication this requires an additional module of division. Division is the most complex part of the hardware. We switch on to Booth multiplier and 16 bit multiplication results were shown in the following figure. Booth multiplication also has some limitations. Since we are working on the security of the system, so we have already loss some of the speed. There are two methods which are quite useful in Modular Arithmetic. They are Wallace Tree Reduction and Dadda Tree Reduction.

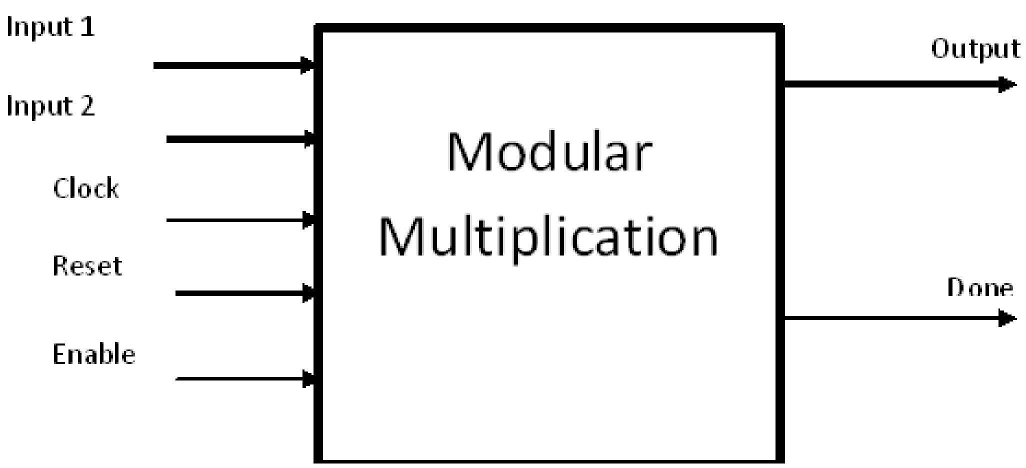

Fig 5.5 Block Diagram of Modular Multiplication

Therefore, we proffered Wallace Tree Reduction Method in order to gain some speed in the RSA Algorithm. There are two inputs namely, input 1 and input 2, whereas the clock, reset and enables are the controlling signals. Output is denoted by output and done indicates that the output has been achieved.

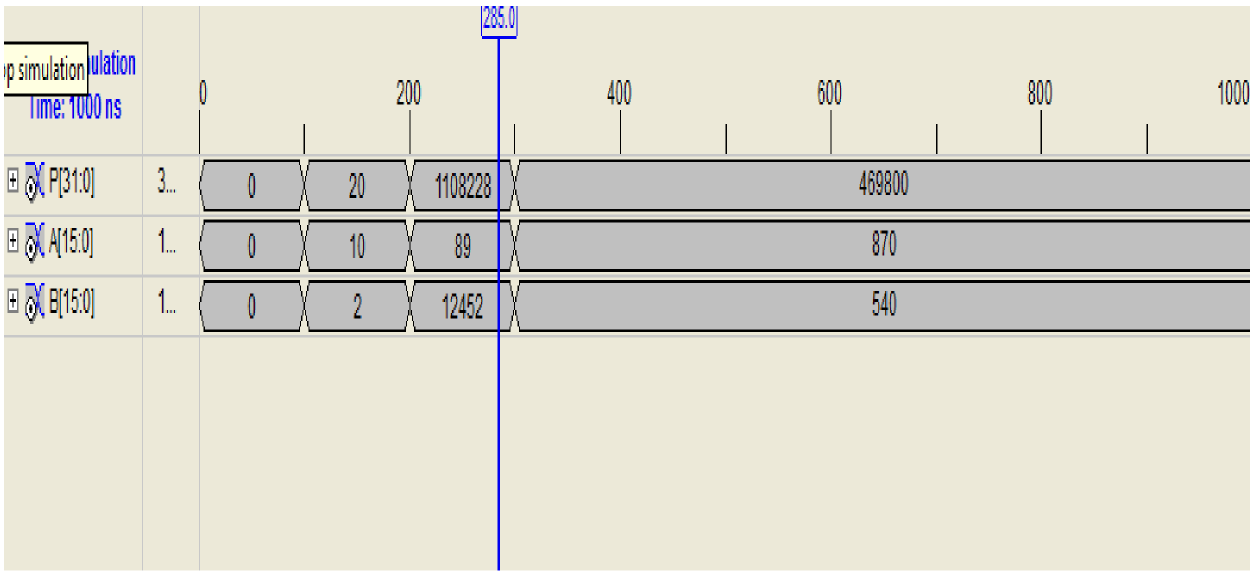

Fig 5.6 Shift and Add Multiplication

The figure 5.6 shows the simulation result of shift and adds multiplication. There are two inputs namely, A and B. Output is denoted by P. A and B are of 16 bits, whereas P is of 32 bits. Each operation requires 10 ns to complete the result. From the figure it is obvious that every logical operation is requiring the same amount of time that is of 10ns.

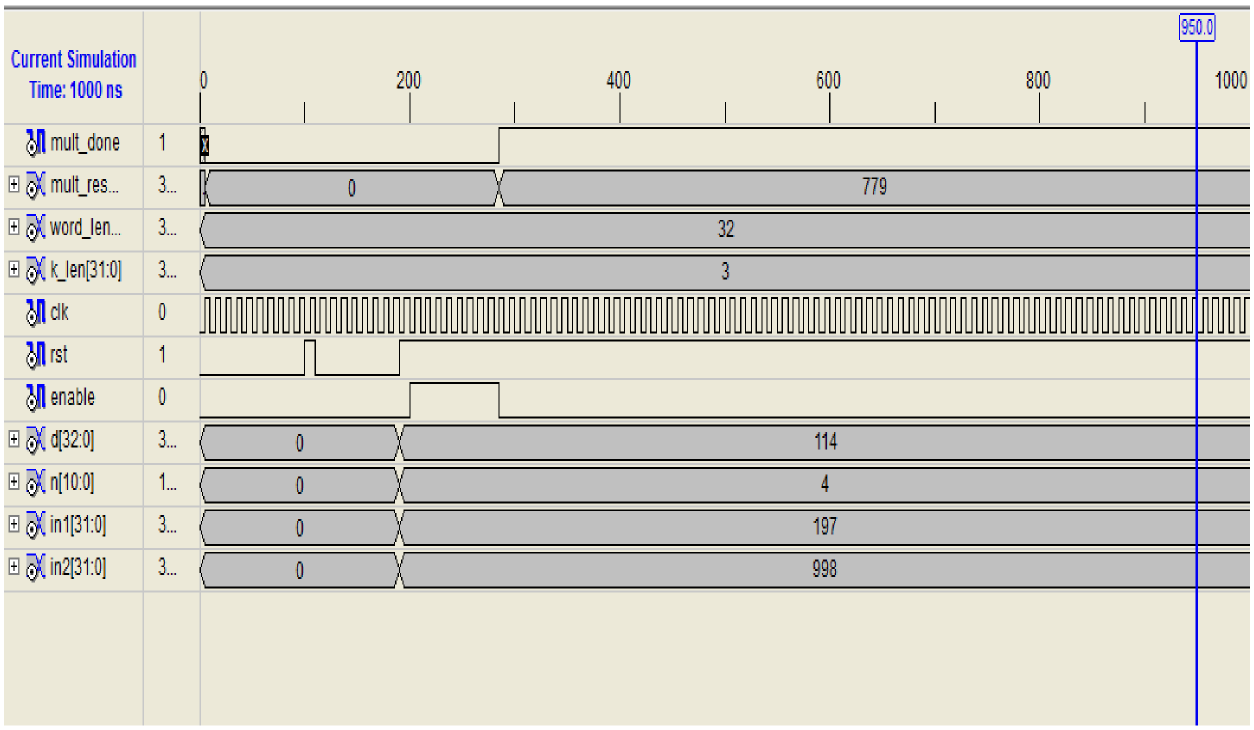

Fig 5.7 Simulation Result of Montgomery Multiplication using Wallace Tree Reduction Method

The figure 5.7 shows the simulation results of Montgomery Multiplication using Wallace Tree Reduction Method. It is formed using sequential logic. The output is high unless the reset bit goes low.

5.4 Modular Exponentiation

Square and multiply algorithm is used effectively to calculate Modular Exponentiation. In most of the cryptographic protocols this algorithm is used.

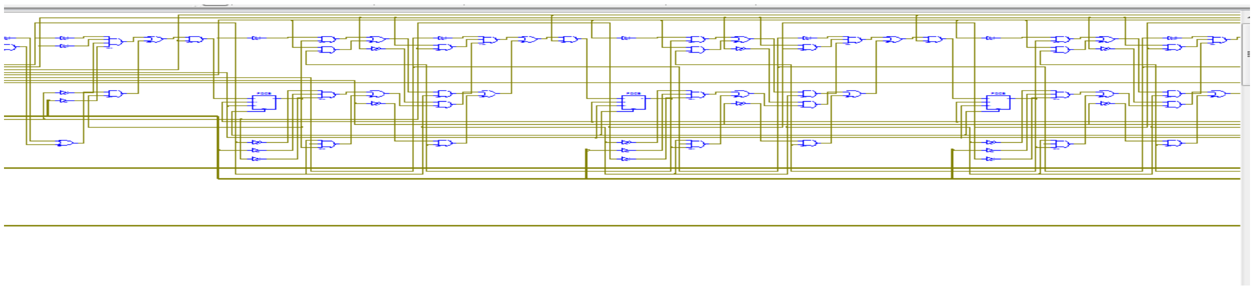

Figure 5.8 View RTL schematic of Modular Exponentiation

The above figure represents the schematic diagram of Modular exponentiation. Square and multiply algorithm is much faster than simple multiplication algorithms. Therefore, we used square and multiply algorithm for exponentiation.

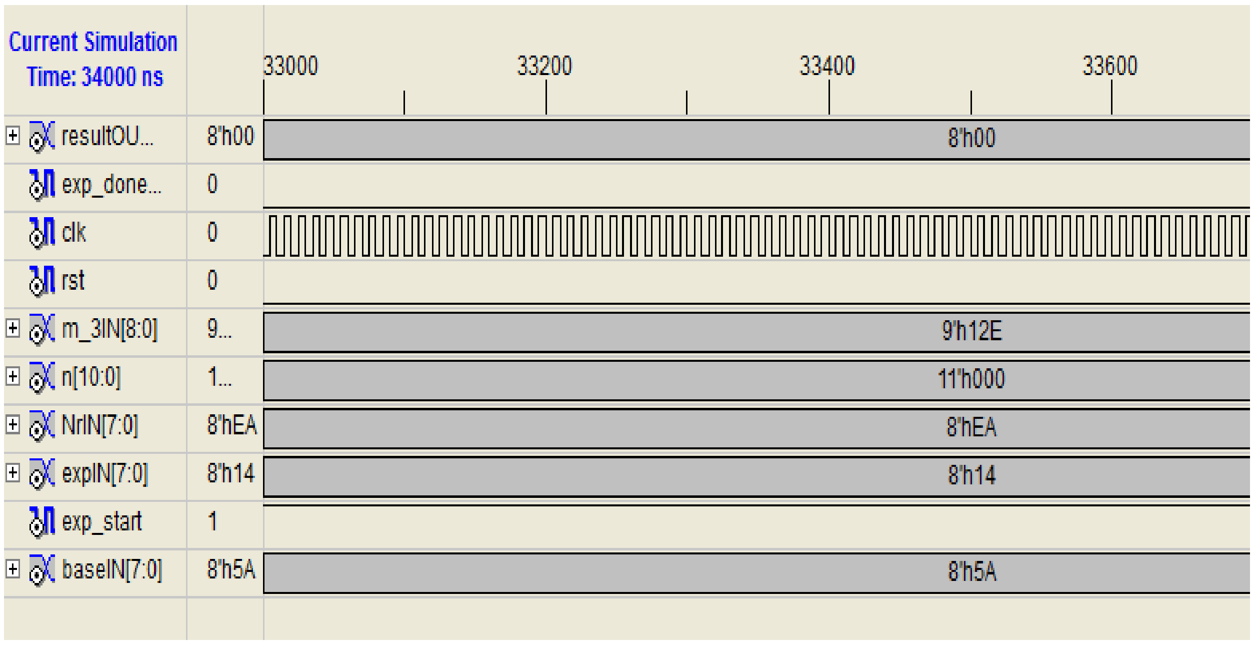

Figure 5.9 Simulation Result of Modular Exponentiation

Figure 5.9 shows the simulation result of Modular exponentiation. We get the result, when the exp done bit is high. As long as exp done bit is low, the output is zero.

5.4 Random Number Generator

Random number plays a vital role in the encryption / decryption. RNG changes the exact time of the operation. It changes the time of operation such sufficiently that the attacker cannot find the exact operation time. Pseudo Random Number Generators are implementing in hardware. For practical use we established 32 bit random number. It provides 2^{32} combinations which are sufficient and have a wide range. We use Barrel shifter for generation of random number. Barrel Shifter has multiple usages in digital design systems. These registers can be used in Encryption / Decryption, Digital Signal Processing, Wireless Communications, Data Integrity Checksum, Data Compression, Random Numbers Generation, Direct Sequence Spread Spectrum (DSSS), Scrambler / Descrambler and Optimized Counters. The following figure shows the block diagram of the Random Number Generator.

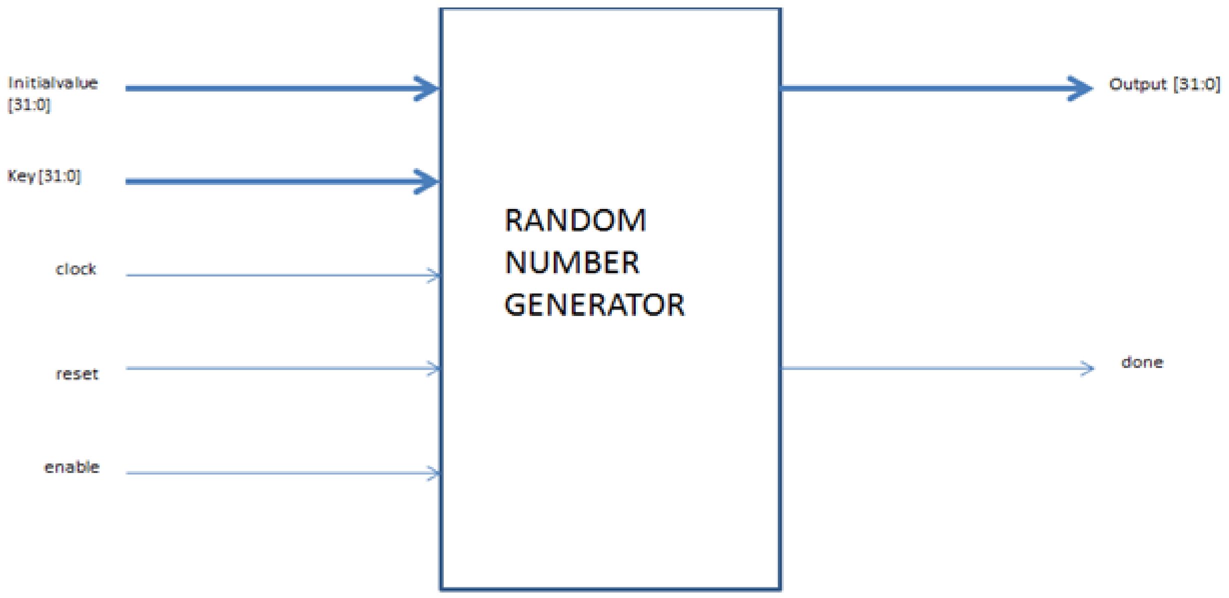

Fig 5.10 Block Diagram of Random Number Generator

Figure 5.10 shows the block diagram of Random Number Generator. Output is attained when the done bit is high. Clock, reset and enable are of 1 bit, whereas initial value and key is of 32 bit. Output is also of 32 bit.

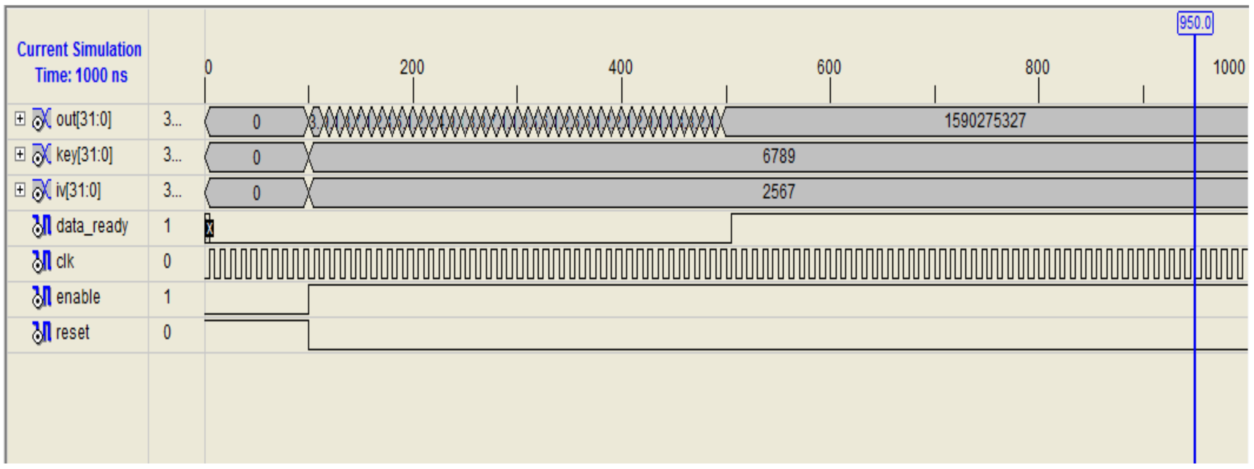

Fig 5.11 Random Number Generator 32 bit

Figure 5.11 indicates the simulation result of 32 bit random number generator. Output is obtained when the reset is low and enable bit is high. The clock cycles of random number generator output is 40 ns.

5.5 Proposed Architecture

Our proposed structure of Secure Implementation of RSA Algorithm against Timing Attacks consists of the following steps. First we implemented random number generator using barrel shifter. The random number generated using barrel shifter comprises of 32 bits. A barrel shifter shifts the data in a digital circuit. It shifts a particular number of bits in one cycle.

Extended Euclidean Algorithm implemented to find out the modulus inverse of the generated random number. The effect of the random number has to be vanished from the original data. In order to remove the effect of the generated random number, we calculated modulo inverse of random number. The Extended Euclidean Algorithm is derived from Euclidean Algorithm. Highest Common Factor is finding through Extended Euclidean Algorithm. The greatest common divisor and inverse modulo number is obtained by using Extended Euclidean Algorithm.

Modular Exponentiation was developed by using Montgomery Multiplication along with Wallace Tree Reduction scheme. Montgomery multiplication was chosen because it does not involve division. Additional module of division is omitted in Montgomery Multiplication and Wallace tree reduction is used in order to shorten the addition steps. We introduce random time to make the timing information unusable.

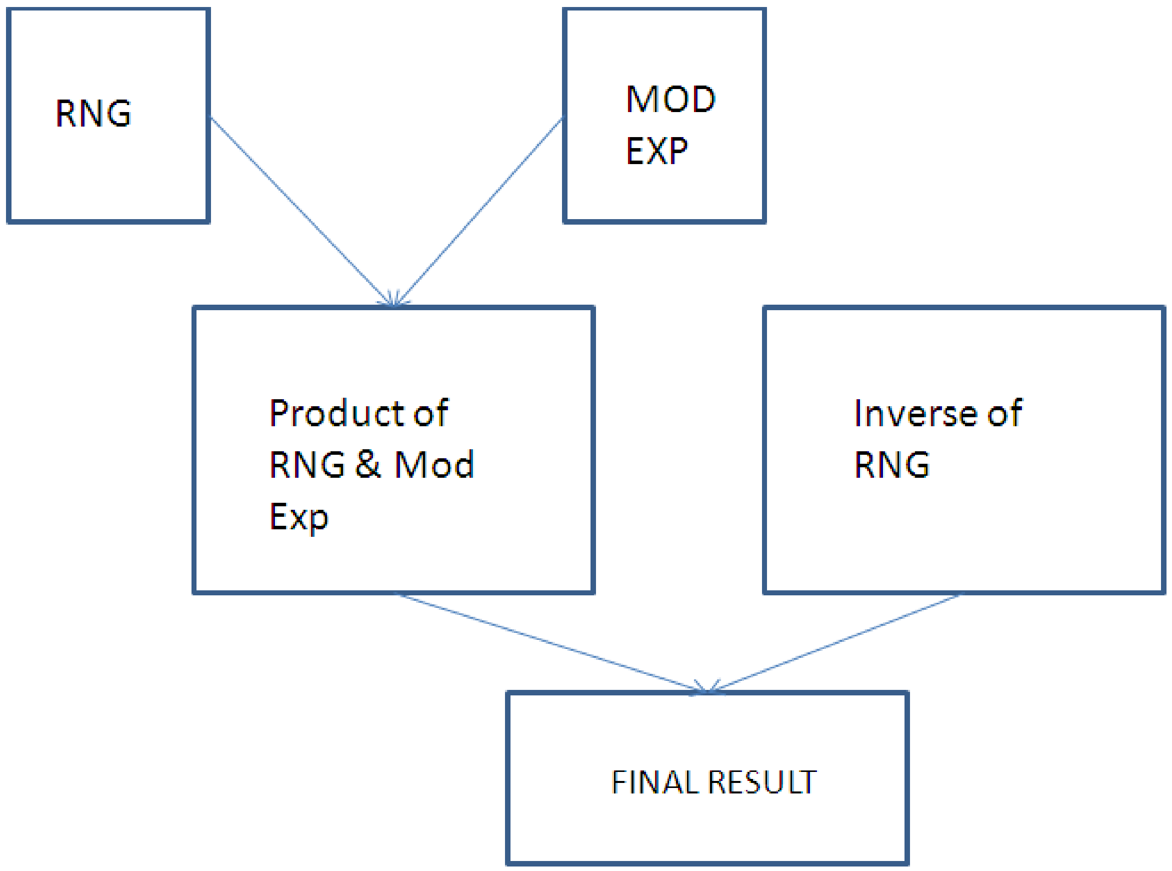

Fig 5.12 Block Diagram of Architecture

The following diagram shows the simulation result of the proposed architecture of RSA Algorithm

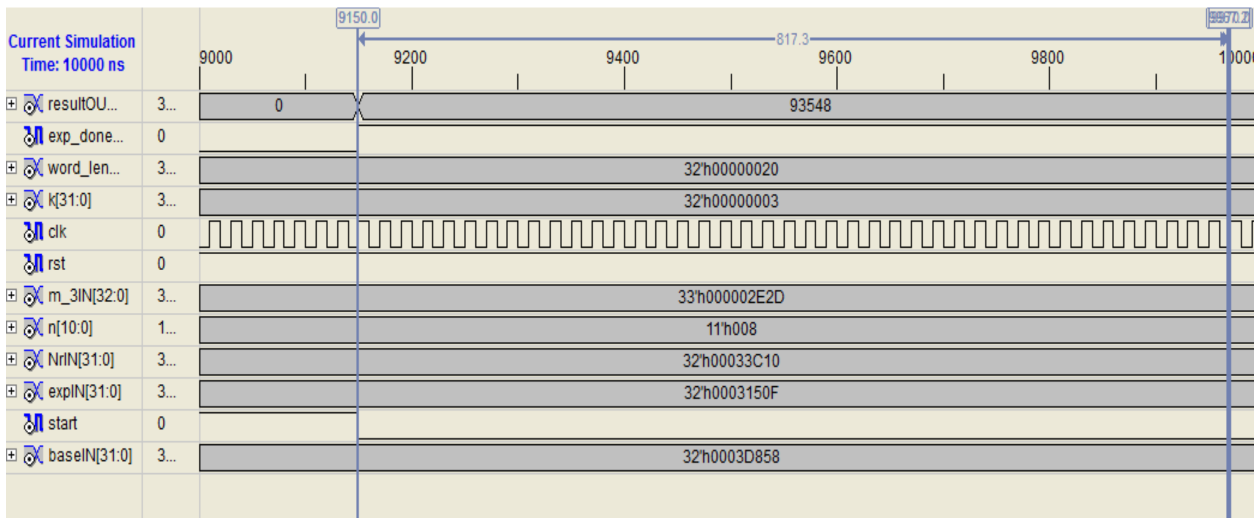
Current Simulation Time: 10000 ns
9150.0
817.3
9000
9200
9400
9600
9800
resultOU... 3... 0 93548
exp_done... 0
word_len... 3... 32'h00000020
k[31:0] 3... 32'h00000003
clk 0
rst 0
m_3IN[32:0] 3... 33'h000002E2D
n[10:0] 1... 11'h008
NrIN[31:0] 3... 32'h00033C10
expIN[31:0] 3... 32'h0003150F
start 0
baseIN[31:0] 3... 32'h0003D858

Fig 5.13 Simulation Result of proposed architecture of RSA

The above figure 5.13 shows the simulation result of the proposed architecture of RSA.

5.6 FPGA Design Flow

FPGA Design flow is shown in the following figure.

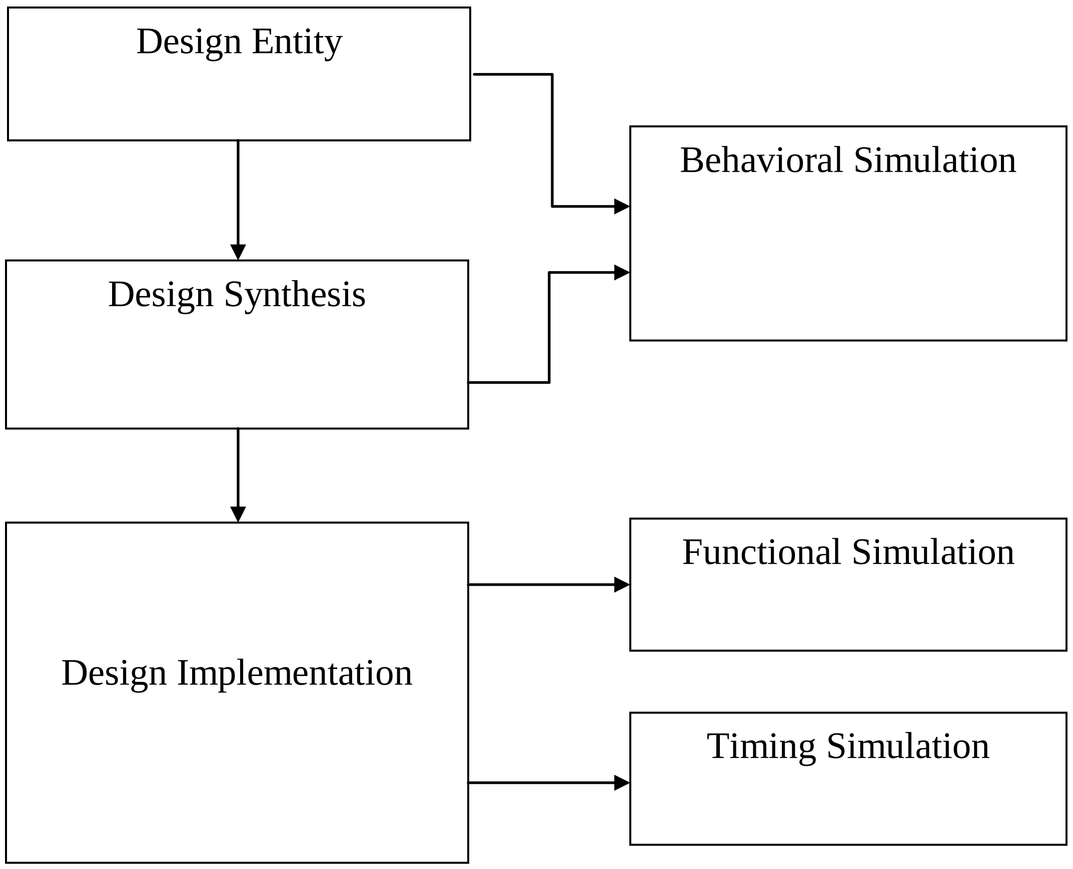

Fig 5.14 FPGA Design Flow

5.6.1 Design Entity

The 1st step is to enter our design. This is done by creating source files. These source files can be created in the forms of schematic or Hardware Descriptive Language.

5.6.2 Behavioral Simulation

The logical functionality of the design is tested by using user defined test benches.

5.6.3 Design Synthesis

The synthesis creates the technology dependent files from the different source files.

5.6.4 Design Verification

This step is done at different stages. This is the important step of the design. The simulator is used to verify the functionality of the circuit.

5.6.5 Design Implementation

After generating the synthesis step the implementation is converted into a logical design.

CHAPTER 6
FIELD PROGRAMMABLE GATE ARRAY

6.1 Introduction

Field Programmable Gate Array (FPGA) belongs to a reconfigurable hardware family. FPGAs are silicon chips which are programmable. Reconfigurable silicon chips provide same flexibility of software running on a processor-based system. FPGAs are absolutely reconfigurable. In recent years FPGAs are using for commercial as well as in military applications. Using these chips, we did not require pick the breadboard and soldering iron. Each dedicated section of the assigned a particular task and each block performs its function without the influence of any other logic blocks.

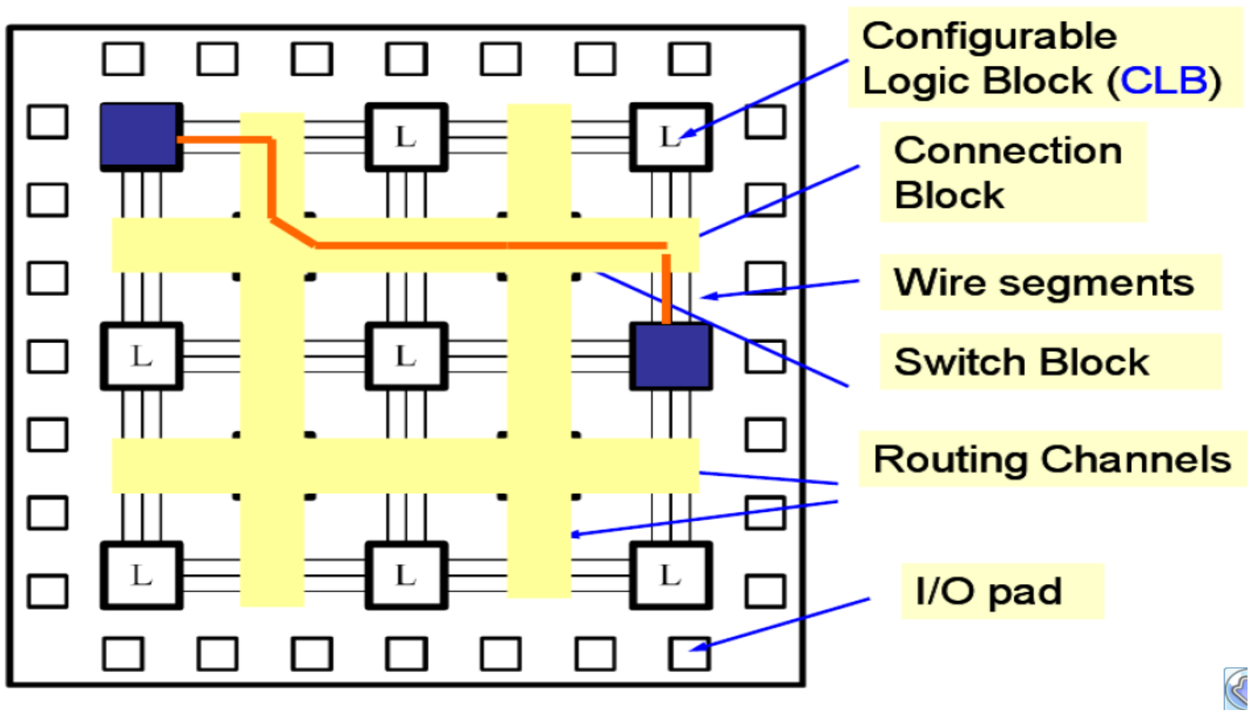

Fig 6.1 Xilinc FPGA Internal Architecture

Due to this, the performance of the one part of the process is not affected when additional task is added. FPGA provides reliability and hardware high speed [1]. FPGAs functions according to the user’s program rather than the manufacturer of the device. FPGA contains (64 to 10,000) identical cells.

6.2 FPGA Scope

FPGA technology is gaining the momentum day by day and worldwide value of FPGA was first invented by Xilinx in 1984, FPGAs have now replaced

6.3 Advantages of FPGA

FPGAs have many advantages over ongoing other resources; some of these advantages are summarized as follows.

Performance: The performance of the FPGA is much better than other resources. It provides fast hardware response time.

Time to Market: FPGA provides flexibility. We can check the idea and any concept using FPGAs before performing the fabrication process of custom ASIC.

Cost: As requirements of the system changes over time very rapidly, the cost of adding additional components in FPGA is very negligible when this is compared to the large expense of the re-spinning an ASIC.

Reliability: For any processor core, only one instruction can execute at one time but FPGA provides the flexibility for executing more than one instruction at the same time.

Long-term Maintenance: FPGA provides long term availability. A device manufactured in one generation and migrates to the next generation the code will remain unchanged. FPGA chips are upgradeable.

6.4 Xilinx FPGA

Five fundamental programmable functional elements are involved in the Spartan-3 family.

Configurable Logic Blocks (CLBs) A CLB has four slices; each slice has two logic cells. Each logic cell has two Look Up Tables (LUTs) plus a storage element such as flip flops. Slices can be configured as LUT, Distributed RAM and Shift register.

Input / Output Blocks (IOBs) The purpose of the IOBs is to control the flow of data from input pins to the output pins and the internal logic of the device. Each IOB supports bidirectional flow of data but IOB can be used as unidirectional flow of data. IOB provides interface between the package pins and CLBs. Outputs can be forced to high impedance. For high performance Inputs and Outputs can be registered.

Block RAM (BRAM) provides data storage in the form of 18 k-bits. BRAM is the most efficient memory implementation. It contains 4 to 104 memory blocks. BRAM can be used for the larger memories. It builds both single and dual port RAMs. BRAM is different from Distributed RAM as it is synchronous to write and read.

Multiplier Blocks provides fast arithmetic functions. It takes two 18 bit binary numbers and calculates the product. It also can be used for 2’s complement signed operation. Multiplier block is asynchronous.

Digital Clock Manager Blocks (DCM) provides fully digital solutions distributing, delaying, multiplying, dividing and phase shifting clock signals.

DCM provides daughter clocks. One important feature of DCM is jitter removal. Clock edges a bit early or a bit late. The FPGA clock manager is used for detect and correct this jitter and provides a clean daughter clock signal for use inside the device. The other main feature of DCM is frequency synthesis.

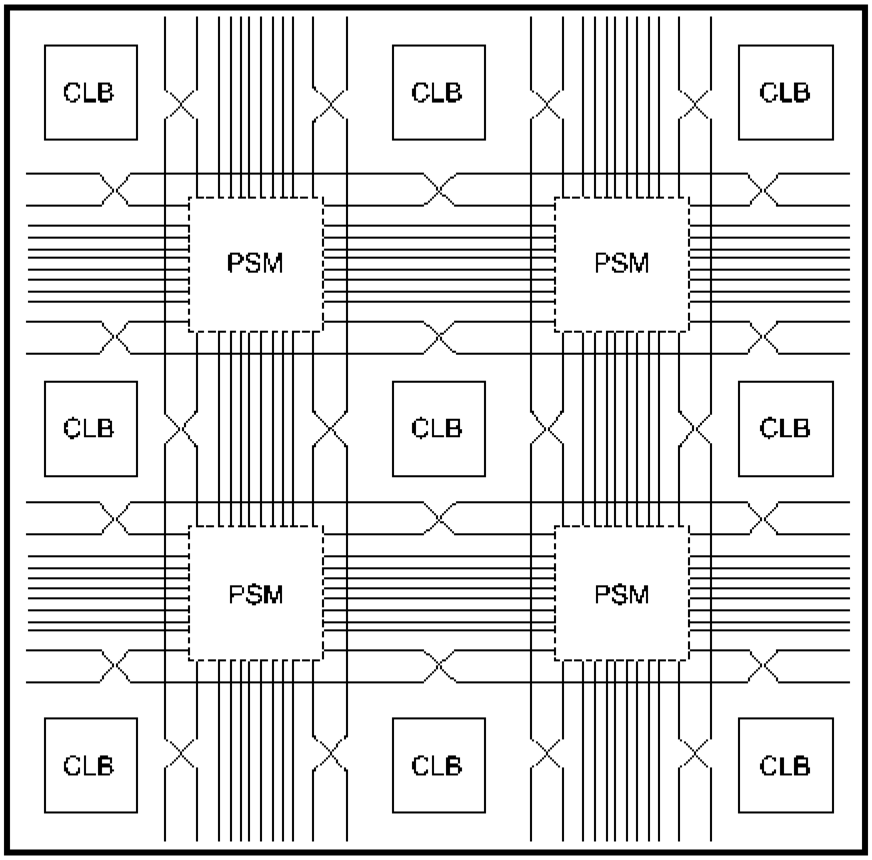

Fig 6.2 Generic FPGA Architecture

The frequency of the clock signal provided to the FPGA outside may not be the exactly what a designer wants. The DCM generates daughter clocks with frequencies that are derived by multiplying or dividing the original signal. DCM may also be used for phase shifting. Designer may want the clocks which are delayed with respect to each other. DCM provides the feature of phase shifts of common values such as 120^O and 240^o. The DCMs are positioned in the center with two at the top and two at the bottom of the device.

CHAPTER 7
RESULTS

7.1 Introduction

The result of the research is summarized in this chapter. The simulation results and synthesis results are presented in this chapter. The layout of the proposed architecture was shown in Chapter 6.

7.2 Simulation Results

This section shows the Simulation Result of the proposed architecture of “Secure Implementation of RSA Algorithm against Timing Attacks”.

7.3 Synthesis Results

This section shows the individual synthesis results of each of the module which are used in the Implementation of the proposed title.

The synthesis result for the implementation of Random Number Generator is expressed in the following table:

Table 7.1 Synthesis Results of Random Number Generator

Logic Components	**Used**
Adder / Sub tractors	3-bit Adder (1) &32-bit adder (2)
Registers / Flip Flops	101
Xors	32-bit Xor

The following table shows the clock cycles of each operation. A random number of 32 bits was generated.

Table 7.2 Clock Cycles of Random Number Generator

Initial Value	**Key**	**Generated Random Number**	**Hamming weight**	**Clock Cycles**
32’h EA	32’h511	32’hF8EDBAAD	21	40
32’h 2654	32’hB3A	32’hC1F6A7CA	18	40
32’h1987	32’h800	32’h3E6A5A38	16	40
32’h6DD61	32’h302F	32’h7F81C7E8	18	40
32’h48FFEA	32’h23BACA	32’h25E38CCE	16	40
32’hBC6146	32’h96B43F	32’h6B5F7A09	18	40
32’h3C413	32’hCF3658	32’h7315FAE	17	40
32’h9C4800	32’h64000A	32’h3BE75E9C	20	40
32’h8180209	32’hECAABC0	32’h6E0F29B3	17	40
32’h48FA533C	32’h5B38E80B	32’h6BA931A4D	18	40
32’h499602D2	32’h499602D2	32’h4668CD10	12	40

The above table indicates that when we change the initial value or key, anew random number is generated but the number of cycles for each operation remains the same. In every execution a new random number is attained. The every generated number is different from the previous number, hence introducing the random number.

The following table shows the number of clock cycles in each operation of multiplication.

Table 7.3 Number of clock cycles for Multiplication

Multiplicand	Multiplier	Multiplication Result	Clock Cycles
135790	864203	11220	100
2345678	456789	12868	98
2468761	98791	8604	101
147952	290541	737233	101
246789	123678	385537	99

The above table shows that for different multiplicand and multiplier, we are obtaining different results, but the number of clock cycles for each operation is the same. This indicates that for different number of bits, the clock cycles are same.

The following table indicates the number of clock cycles when RSA simulated.

Table 7.4 Number of Clock Cycles for RSA

Base	Exponent	Result	Clock Cycles
161984	161998	43310	1023ns
236789	456721	23334	1021ns
194023	196507	51104	1020ns
251992	201999	93548	1020ns

The above table shows the result of the RSA. In the last column clock cycles are showing that for each operation we have same number of clock cycles irrespective of the base and exponent. The result of the exponentiation is varying but the time required to execute the operations is the same.

The following table shows the clock cycles for each operation when blinding is performed on the RSA.

Table 7.5 Number of Clock Cycles for Blinded RSA

RNG	**Base**	**Exponent**	**Clock Cycles**
YES	251992	201999	1020ns
YES	161984	161998	1200ns
YES	236789	456721	1197ns
YES	975310	864209	1200ns
YES	194023	196507	12020ns

This table shows the result of the proposed architecture of RSA algorithm. The results are taken when Random Number is used; due to the addition of randomness the clock cycles are changed for each operation.

This table indicates the comparison between the above two tables.

Table 7.6 Difference between Clock Cycles

Base	**Exponent**	**Clock Cycles**	**Base**	**Exponent**	**Clock Cycles**	**Difference**
251992	201999	1120ns	251992	201999	1020ns	100ns
161984	161998	1200ns	161984	161998	1023ns	177ns

The above table shows that the difference between clock cycles when Modular Exponentiation is performed without blinding, and when the Modular Exponentiation is performed along with the blinding. Taking same base and exponent values, we are getting different amount of time for same operation. This indicates that when we are performing blinding the execution time for operation changes slightly as randomness is included in the algorithm..

CHAPTER 8
CONCLUSION AND FUTURE WORK

8.1 Conclusion

In this research we have developed a hardware model of RSA cryptographic system which provides more security to our cryptographic system. The thesis begins with the introductory description of the cryptographic systems. RSA technique is the most popular and widely method, which is used for the encryption. The goal set in this thesis assignment was fulfilled. The thesis presented the enhancement of the security. The implemented RSA algorithm is much more secure. The above thesis was under taken in order to develop a secure Implementation of RSA algorithm against Timing Attacks.

The methodology implemented for the security of RSA was blinding. Blinding was used in order to avoid the timing attacks and improve the security of the general RSA algorithm. Montgomery multiplier was used as it is fastest multiplication technique till now.

8.2 Future Work

The goal of the thesis was achieved by establishing a random number. The random number was 32 bit long. It produces 2^{32} combinations. Every value was repeated after at least 2^{32} iterations. In future as the progress is going on, in the field of cryptography, the number of combinations should be much more as it is now. The modulus inverse of the generated random number was calculated using Extended Euclidean Algorithm. Instead of using Extended Euclidean Algorithm, we can use other methods to calculate modulo inverse of the generated random number. The other method for calculating multiplicative inverse would be Laszlo Hars modification due to less number of slice consumption. It will allow us to build a much larger RSA system with high throughput.

We use Montgomery exponentiation which uses Montgomery multiplication technique along with Wallace tree reduction method for the exponentiation, Dadda tree reduction can also be used instead of Wallace tree reduction. Study the use of faster adder and serial multiplier architectures. For example tree adder implementation may have O (log n) delay, while typical architectures have linear delay.

Even with the use of blinding the sharing will expose the average time per operation. This can be used to infer the hamming weight of the exponent. Before performing exponentiation a random multiple can be added to the exponent. Addition process should not have timing characteristics in itself, which may expose the random multiple. This method may be helpful in preventing attacks that gain information leaked during the modular exponentiation operation due to electromagnetic radiation, system performance fluctuations, changes in power consumption, since the exponent bits change with each operation.

REFENRENCES:

[1] Implementation of Efficient Modular Exponentiation on Reconfigurable Platforms, August 2008, Kashif Latif, Department of Electronic and Power Engineering, College of Marine Engineering (PNEC), Karachi, National University of Sciences and Technology, Rawalpindi

[2] Timing Attacks on Implementation of RSA, Diffie-Hellman, DSS and Other systems, Paul C. Kocher, Cryptography Research, Inc. 607 Market Street, 5th Floor San Francisco, CA 94105, USA

[3] Attacks on the RSA Cryptosystem, VaibhavVaish, Department of Computer Science & Engineering, Indian Institute of Technology, New Delhi, India, 1999

[4] Implementing a 1024 bit RSA on FPGA, 2003, Jing Lu and Wan Qian, Reconfigurable Network Group, Applied Research Lab, Department of Computer Science and Engineering, Washington University in St. Louis

[5] Timing Attacks on software Implementation of RSA, Project Report, Harshman Singh 903-40-5260, June 07, 2004

[6] Efficient Hardware Design and Implementation of AES Cryptosystem, Pravin B. Ghewari et al. International Journal of Engineering Science and Technology, Vol. 2(3), 2010, 213-219

[7] FPGA Implementation of RSA, Public-Key Cryptographic Coprocessor, MiCE Department, Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 8 13 10 UTM Skudai, Johor, Malaysia.

[8] Implementation of RSA Cryptosystem Using Verilog, Chiranth E, Chakravarthy H.V.A, Nagamohanareddy P, Umesh T.H, Chethan Kumar M, International Journal of Scientific & Engineering Research Volume 2, Issue 5, May-2011 1, ISSN 2229-5518, IJSER © 2011

[9] The Protection of Modular Exponentiation Operands from Their Reconstruction by Simple Power Analysis, Akram A. Moustafa and Saleh Alomar, Proceedings of the World Congress on Engineering and Computer Science 2009 Vol I, WCECS 2009, October 20-22, 2009, San Francisco, USA

[10] The Implementation of the 1024-bit RSA Encryption/Decryption Algorithms Based on FPGA, Yingjie Qu, and Qing Zhao Qingdao University of Science & Technology, Qingdao, China, ISBN 978-952-5726-04-6 (Print), 978-952-5726-05-3 (CD-ROM) Proceedings of the International Symposium on Intelligent Information Systems and Applications (IISA'09) Qingdao, P. R. China, Oct. 28-30, 2009, pp. 420-423

[11] Assorted Attacks on the RSA Cryptographic Algorithm Edmond J. Murphy, Boston College Computer Science Department Professor Howard Straubing May 9, 2005

[12] An Efficient VLSI Architecture for Rivest-Shamir-Adleman Public-key Cryptosystem, Department of Electrical Engineering Tamkang University, Tamsui, Taiwan 251, R.O.C., , Tamkang Journal of Science and Engineering, Vol. 7, No 4, pp. 241_250 (2004)

[13] Protecting RSA Against Fault Attack: The Embedding Method, Marc Joye, Thomson R&D, Security Competence Center, Published in L. Breveglieri et al., Eds, Fault Diagnosis and Tolerance in Cryptography (FDTC 2009), IEEE Computer Society, pp. 41–45, 2009

[14] VLSI Implementation of RSA Encryption System Using Ancient Indian Vedic Mathematics, Himanshu Thapliyal and M.B Srinivas, International Institute of Information Technology, Hyderabad-500019, India

[15] Carry-Save Montgomery Modular Exponentiation on Reconfigurable Hardware, A. Cilardo, A. Mazzeo, L. Romano, G. P. Saggese, Universit`a degli Studi di Napoli Federico II, Italy, Volume 3, IEEE Computer Society Washington, DC, USA @ 2004

[16] An Overview of Side Channel Attacks and Its, Countermeasures using Elliptic Curve Cryptography M.Prabu R.Shanmugalakshmi, M.Prabhu et. al. / (IJCSE) International Journal on Computer Science and Engineering, Vol. 02, No. 04, 2010, 1492-1495

[17] A Review of Modular Multiplication Methods and Respective Hardware Implementations, Nadia Nedjah, Department of Electronics Engineering and Telecommunications, Engineering Faculty, Informatica **30** (2006) 111–129 **111**

[18] FPGA Implementation of RSA Encryption Engine with Flexible Key Size, Muhammad I Ibrahimy, Mamun B. I Reaz, International Journal of Communications, Issue 3, Volume 1, 2007

[19] Hardware Implementation of RC4A Stream Cipher, Abdullah Al Noman, Roslina Mohd. Sidek, Abdul Rahman Ramli, International Journal of Cryptology Research 1(2): 225-233 (2009)

[20] RSA & Public Key Cryptography in FPGAs, John Fry, Altera Corporation – Europe, Martin Langhammer, Altera Corporation

[21] Timing Attacks on Secure Systems, Dan Halperin, April 13, 2006

[22] Differential Power analysis resistant hardware implementation of the RSA cryptosystem, Turk J Elec Eng & Comp Sci, Vol.18, No.1, 2010, c T¨UB˙ITAK doi:10.3906/elk-0904-4

[23] A Hardware Model of an Expandable RSA Cryptographic System, Adnan Abdul-Aziz M.S. Gutub, Faculty of The College of Graduate Studies King Fahad University of Petroleum & Minerals Dhahran, Saudi Arabia, December 1998

[24] Implementation of a Generic Modular Cryptosystem for the RSA on Reconfigurable Hardware, Deepak Krishnankutty, Department of Computer Science & Engineering, National Institute of Technology, Rourkela, India, 2009

[25] Hardware Attacks on Cryptographic Devices, Implementation Attacks on Embedded Systems and Other Portable Devices, Jem Berkes, University of Waterloo 2006

[26] Design and Implementation of PRBS Generator Using VHDL, Sandeep Mukharjee & Ruchir Pandey, Department of Computer Science & Engineering, National Institute of Technology, Rourkela, India, 2009

[27] Selected RNS Bases for Modular Multiplication, J.C. Bajard, LIRMM, CNRS-Univ. Montpellier2, M. Kaihara, EPFL Lausanne & T. Plantard, Univ. Wollongong, Australia , 2009 19th IEEE International Symposium on Computer Arithmetic

[28] Montgomery Algorithm for Modular Multiplication, Professor Dr. D. J. Guan, August 25, 2003

[29] Differential Power Analysis Resistant Hardware Implementation of the RSA Cryptosystem, Keklik ALPTEK_N BAYAM, Institute of Science & Technology, Istanbul Technical University, May 2007

[30] Modular Multiplication using Montgomery method, Haoxin (Larry) Song, ECE 543 Final Project Report

[31] Automatic Verification of Arithmetic Circuits in RTL using Term Rewriting Systems, Shobha Vasudevan, The University of Texas at Austin, December 2003

[32] http://en.wikipedia.org/wiki/Adder_(electronics)

[33] Handbook of Applied Cryptography, Alfred J. Menezes, Paul C. Van, Oorschot Scott A. Vanstone, CRC Press ISBN: 0-8493-8523-7, August 2001

[34] Cryptography and Network Security, 3rd Edition, William Stalling