University of Mosul
College of Electronic Engineering

FPGA Based of RSA Algorithm Implementation

Zaid Abdulsattar Abdulrazzaq

M.Sc. Dissertation in
Computer and Information Engineering

Supervised by

Assistant Prof Dr. Sa'ad Daoud Suleiman

2018 A.D. 1439 A.H.

University of Mosul

College of Electronic Engineering

FPGA Based of RSA Algorithm Implementation

A Thesis Submitted by

Zaid Abdulsattar Abdulrazzaq

To

The Council of College of Electronic Engineering

University of Mosul

In Partial Fulfillment of the Requirements

For the Degree of Master of Sciences

In

Computer and Information Engineering

Supervised by

Assistant Prof Dr. Sa'ad Daoud Suleiman

2018 A.D. 1439 A.H.

﴿وعنده مفاتيح الغيب لا يعلمها الا هو ويعلم ما في البر والبحر وما تسقط من ورقة الا يعلمها ولا حبة في ظلمات الارض ولا رطب ولا يابس الا في كتاب مبين ﴾ الانعام : 59

Supervisor's Certification

I certify that the dissertation entitled (**FPGA Based of RSA Algorithm Implementation**) was prepared by **Zaid Abdulsattar** under my supervision at the Department of Computer and Information Engineering, University of Mosul, as a partial requirement for the Master of Science Degree in Computer and Information Engineering.

Signature:

Name: Assistant Prof Dr. Sa'ad Daoud Suleiman

Department of Computer and Information Engineering

Date: / /2018

Report of Linguistic Reviewer

I certify that the linguistic review of this dissertation was carried out by me and it is accepted linguistically and in expression.

Signature:

Name:

Date: / /2018

Report of the Head of Department

I certify that this dissertation was carried out in the Department of Computer and Information Engineering. I nominate it to be forwarded to discussion.

Signature:

Name: Assistant Prof Dr. Abdul Baree R. Suleiman

Date: / /2018

Report of the Head of Postgraduate Studies Committee

According to the recommendations presented by the supervisor of this dissertation and the linguistic reviewer, I nominate this dissertation to be forwarded to discussion.

Signature:

Name: Assistant Prof Dr. Abdul Baree R. Suleiman

Date: / /2018

Committee Certification

We the examining committee, certify that we have read this dissertation entitled (**FPGA based RSA algorithm**) and have examined the postgraduate student (**Zaid abdulsatar**) in its contents and that in our opinion; it meets the standards of a dissertation for the degree of Master of Science in Computer and Information Engineering.

Signature:
Name:
Head of committee
Date: / /2018

Signature:
Name:
Member
Date: / /2018

Signature:
Name:
Member
Date: / /2018

Signature:
Name: Assistant Prof Dr. Saad Daoud
Member and Supervisor
Date: / /2018

The college council, in its …… meeting on / /2018, has decided to award the degree of Master of Science in Computer and Information Engineering to the candidate.

Signature:
Name:
Dean of the College
Date: / /2018

Signature:
Name:
Council registrar
Date: / /2018

Acknowledgements

Thanks to Almighty Allah for giving me the strength and confidence to realize and complete this work.

I am deeply indebted to my supervisor Prof. Asst. Dr. Saad Daoud Suleiman for the effective guidance and helpful discussions during the progress of this thesis.

Special thanks to My colleagues M. Mohammed Hazim, M. Sarmad Fakhrulddin, M. Maʾmoon Abduljabbar, M. Manar Husam providing the facilities required for achieving this work. Thanks also go to the Department staff and to my colleagues for their help.

Last but not least, I would like to express my thanks and gratitude to my aunt Dr. Maha Hasso and all my family for their support.

Abstract

RSA is highly secure cryptosystem for data transmission but it is relatively slow algorithm to be used in real time.

In this thesis an analysis, design and implementation for main algorithms have been used in RSA focusing on execution time using higher language (JAVA) and FPGA with results comparison.

The execution time obtained for modular multiplications for FPGA using interleaved was 142.416µs , Montgomery was 109.995µs , faster Montgomery was 85.505µs and modified interleaved with 110.819µs which are the core algorithms used in RSA implementation.

The execution time using java implementation was 29.259 ms, using modified modular algorithm, faster Montgomery multiplication algorithm and Chinese remainder theorem were 26.057 ms, 14.098 ms and 6.522 ms respectively. Noticeable speed up was obtained using FPGA as comparison with JAVA

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Name
CRT	Chinese Remainder Theorem
FPGA	Field Programmable Gate Array
IOBs	Input Output Buffers
ISE	Xilinx Integrated Software Environment
LR	Left to Right Modular Exponentiation algorithm
LUT	LookUp Tables
MIT	Massachusetts Institute of Technology
MMPS	Modular Multiplications per Second
MOD	Modular Mathematical Operation
Mon Pro	Montgomery production
PR	Private Key
PU	Pubic Key
RL	Right to Left Modular Exponentiation algorithm
RSA	Ron Rivest, Adi Shamir, and Len Adleman algorithm
RSACRT	RSA using Chinese Remainder Theorem
RSAF	RSA using Faster Montgomery Multiplication
RSAM	RSA using Modified Interleaved Multiplication
SoC	System on Chip
SP601	Spartan 601
VHDL	Very High Description Language
XST	Xilinx Synthesis Technology

Chapter One

Introduction and Literature Review

1.1 Background

With the increase of data communication and expansion of internet multiple services like electronic commercial transmissions, the most important thing is security over the networks, one of the most widely used method for that reason is Rivest Shamir Adleman RSA[1].

RSA is one of the most popular algorithms which was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and published first in 1978. The RSA encryption is a block cipher algorithm in which the plaintext and ciphertext are between 0 and 2^n numbers (where n is the number of bits), often the value of n is 1024 bits, so the message and encrypted number is less than 2^{1024}. RSA uses modular exponentiation , first , Plaintext is encrypted in blocks, with each block having a binary value less than some number n. That is, the block size must be less than or equal to log2(n) + 1, in practical , the block size is n bits, where $2^n \leq n \leq 2^{n+1}$[1].

1.2 RSA Key Generation[1]

The method is to select two large prime numbers p and q with condition $p \neq q$

$N = p \times q$ ……..(1.1)

$\emptyset(n) = (p-1) \times (q-1)$…….....(1.2)

Select e so that 1<e< Ø(n) and e co prime to Ø(n)

$d = e^{-1} \text{ MOD } \emptyset(n)$ …….. ……..(1.3)

public key = (e , N)

private key = (d, $\emptyset(n)$)

1.3 RSA Encryption[1]

$c = M^e MOD\ N$.....(1.4)

Where M : message (plaintext) , e : public key , N : public key , c : encrypted message

1.4 RSA Decryption[1]

$M = C^d\ MOD\ N$.....(1.5)

Where c : encrypted message, d : private key, N : public key, M : message.

1.5 RSA Encryption Decryption Example[1]

For a numeric example :-

1. Select two prime numbers, p = 17 and q = 11.
2. Find N Refer to equation (1.1)

   $N = p \times q$

   $N = 17 \times 11 = 187$.
3. Find Ø (n) Refer to equation (1.2)

   $\emptyset(n) = (p - 1) \times (q - 1)$

   Ø (n)= $16 \times 10 = 160$.
4. Select e such that e is relatively prime to Ø (n) = 160 , less than Ø (n), let e = 7.
5. Refer to equation (1.3)

   $d = e^{-1}$ MOD $\emptyset(n)$

   $d = 7^{-1}\ mod\ 160$

   and d must be less than 160.

   The value of d is 23 because $23 \times 7 = 161 = (1 \times 160) + 1$.

   (d can be computed using extended Euclid's algorithm)

The keys are public key = (7, 187) and private key = (23, 160) And a message input M = 88 ?

the encryption performed using equation (1.4)

$c = M^e \ mod \ N$

C = 88^7 MOD 187.

Based on mathematical properties of modular arithmetic the calculation which can be done like the following

88^7 MOD 187 = [(88^4 MOD 187) × (88^2 MOD 187) × (88^1 MOD 187)] MOD 187

88^1 MOD 187 = 88

88^2 MOD 187 = 7744 MOD 187 = 77

88^4 MOD 187 = 59,969,536 MOD 187 = 132

88^7 MOD 187 = (88 × 77 × 132) MOD 187 = 894,432 MOD 187 = 11.

Second: the decryption performed using equation (1.5)

$M = C^d \ MOD \ N$

M = 11^{23} MOD 187

Based on The mathematical properties of MOD operation :

11^{23} MOD 187 = [(11^1 MOD 187) × (11^2 MOD 187) × (11^4 MOD 187) × (11^8 MOD 187) × (11^8 MOD 187)] MOD 187

11^1 MOD 187 = 11

11^2 MOD 187 = 121

11^4 MOD 187 = 14,641 MOD 187 = 55

11^8 MOD 187 = 214,358,881 MOD 187 = 33

11^{23} MOD 187 = (11 × 121 × 55 × 33 × 33) MOD 187 = 79,720,245 MOD 187 = 88.

1.6 RSA Security

There are four possible approaches to attack the RSA algorithm. They are as follows:

1. Brute force attack: in this approach the attacker tries all possible private keys.
2. Mathematical attacks: in this approach the attacker is using multiple methods to factor the product N in two primes p and q, this enables computation of Ø(n) using equation(1.2) (Ø(n) = (p-1) ×(q-1), which enables calculation of d using equation(1.3) ($d = e^{-1}$ MOD Ø(n)).
3. Timing attack: This depends on the execution time of the decryption algorithm bit by bit and on the modular exponentiation algorithm design.
4. Chosen cipher text attack: this type of attack is done according to the properties of the RSA algorithm[1].

1.7 Literature Survey and Related Works

In 2017, Implementation of RSA Algorithm with Chinese Remainder Theorem for Modulus N 1024 Bit was proposed by Desi Wulansari, Much Aziz Muslim and Endang Sugiharti [2]. They presented results of the testing algorithm RSA-CRT 1024 bits , the deign achieved approximately 3 times faster in performing the decryption speed (that means RSA time implementation using CRT equal to 3 times RSA time without using CRT).

In 2014, FPGA Implementation of RSA Encryption Algorithm for E-Passport Application is proposed by Khaled Shehata, Hanady Hussien and Sara Yehia [3]. In this paper, the design presented an implementation method for 1024-bit RSA encryption/decryption algorithm using modular exponentiation. This method used square and multiply algorithm. The paper

used add and shift algorithm for design modular multiplier. All the designs implemented using using Xilinx ISE 12.3 software tool, VHDL language code targeting device Virtex-5 XC5VTX240T-2FF175 FPGA and achieved speed was 36.3 MHz

In 2012, Implementation of RSA Algorithm on FPGA was proposed by Ankit Anand and Pushkar Praveen [4]. They presented implementation of modular exponentiation and Montgomery multiplier, fully described using VHDL and Xilinx ISE software, Target device 3s1600efg320-4 and would be possible to implement RSA with key sizes such as 1024 bits, 1536 bits, and 2048 bits with the clock frequency 69.09MHz and consumed 13,779 units of logic elements.

In 2011, A FPGA implementation of the RSA encryption algorithm was proposed by P. Gabriel Vasile Iana1 , Petre Anghelescu1 and Gheorghe Serban1[5]. This paper presented implementation of RSA as a prototype Xilinx Spartan3 using Montgomery multiplier. The key space was 1024 bit and the execution time was 212.99ms with a 50 MHz RSA clock system to achieve 208Kbps bit rate.

In 2011, hardware algorithm using 2048-bit RSA encryption/decryption was proposed by Song Bo, Kensuke Kawakami, Koji Nakano, Yasuaki Ito[6] where implementation was designed using one DSP48E1 using one BRAM and few logic blocks (slices) in the Xilinx Virtex-6 family FPGA. The execution time results obtained RSA module for 2048-bit RSA encryption/decryption runs in execution time 277.26ms.

In 2009 Efficient Hardware Implementation of RSA Cryptography was proposed by Mostafizur Rahman, Iqbalur Rahman Rokon and Miftahur Rahman [7] , presented hardware implementation of modular exponentiation using interleaved multiplication. The design was modeled using Verilog HDL

software tool and targeted to Virtex FPGA hardware device. Key size was 8 bits, 1.2ms execution time system and 100MHZ clock speed.

In 2009 RSA Encryption and Decryption using Redundant Number System on the FPGA was proposed by Koji Nakano, Kensuke Kawakami, and Koji Shigemoto[8]. The idea of design was to accumulate the modulo exponentiation using Montgomery multiplication algorithm by embedded multipliers and embedded 18k-bit block RAMs in . The hardware algorithms the system has been implemented on Xilinx VirtexII Pro family FPGA XC2VP30-6, key size 1024-bit can operate in less than 2.521ms or 1.892ms execution time .

In 2008, Parametric, Secure and Compact Implementation of RSA on FPGA was proposed by Ersin Öksüzoğlu, Erkay Savaş[9]. The design utilized block multipliers as the main mathematical unit to build Montgomery multiplier and Block-RAM as storage unit using Xilinx Spartan-3E using a pipelining method. The execution time was 7.62 μs and 27.0 μs for 1020-bit and 2040-bit key sizes modular multiplications respectively. The execution time for 1024 bit key size modular exponentiation RSA was 7.81 ms.

In 2004, fast Architectures For FPGA-Based Implementation of RSA Encryption Algorithm is proposed by Omar Nihouche, Mokhtar Nibouche, Ahmed Bouridane, and Ammar Belatreche[10]. They presented multiple structures f RSA modules using Montgomery modular multiplier, was implemented in Xilinx ISE 6.2 Software tool and XC4015OXV-8 FPGA hardware device The execution time of RSA was 27.88 ms.

In 2003, Efficient Architectures for implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic was proposed by Alan Daly and William Marnane[11]. They presented a pipelined technique using the maximum carry chain length of the FPGA

that implemented the modular exponentiation operation required for RSA using Montgomery multiplication. The operation speed of the system was 49.64 MHZ with 45.8 kb / s data rate.

1.8 Aim of the Thesis

The methodology used in this thesis involved :-

1. Reviewing the theoretical foundation of RSA based on modular exponentiation algorithms.
2. Implementing different types of modular multiplication emphasizing on execution speed.
3. Choosing an optimum Modular multiplication algorithm based on execution time.
4. Designing a modular exponentiation operation that represents RSA encryption and decryption equations.
5. Discussing the final hardware RSA modules speed and throughput.

1.9 Thesis Layout

Expect the introduction showed in this chapter, the remaining chapters are organized as following: Chapter Two handles RSA Theoretical Background and presents hardware architecture algorithms used in RSA implementation . Chapter Three handles RSA software implementation using Java . Chapter four deals with FPGA implementation of RSA algorithm. Finally chapter five presents the conclusion and some suggestions future works.

Chapter Two

RSA Algorithm Architectures

2.1 Introduction

This chapter provides introduction about hardware algorithms used for RSA implementation.

2.2 Modular Exponentiation Operation[12]

RSA encryption and decryption is a modular exponentiation operation can be represented by equation (1.4) and equation (1.5) representing following equation :-

$$c = p^e \ MOD \ M \quad \text{(2.1)}$$

Where

p : plain text(in encryption) or Cipher text (in decryption)

e : public key(in encryption) or private key(in decryption)

M : modulus (which represented N of RSA algorithm see key generation process)

C : Cipher text(in encryption) or Plaintext(in decryption)

The equation above is called modular exponentiation. There are multiple techniques for hardware implementation of modular exponentiation, The most widely used are:

- Right-to-left (RL)
- Left-to-Right (LR)

2.2.1 RL Binary Method

Inputs : p , e, N

Output: $C := p^e \ mod \ N$

K : number of bits in e;

1) C := 1 ;
2) For i=0 to k-1 do (where k : number of bits in e)
3) if (e_i = 1) then (where e_i : i^{th} bit of e)
4) $C := C \times p \ mod \ N$ (where ×: multiply)
5) End if;
6) p:= $p \times p \ mod \ N$; (square)
7) End for;
8) return C;

The bits of e are scanned from least significant to most significant, if the bit index i of e is equal 1 it performed two modular multiplications(multiply and square) else performed only one (square). For (e = 55), RL algorithm will work as shown below in table 2.1.

Table 2.1 RL Binary Method [12].

When e = 55 and k= 6 bit			
i	**e_i**	**Step 3(C)**	**Step 6(P)**
0	1	$1 * P = P$	$(P)^2 = P^2$
1	1	$P * P^2 = P^3$	$((P)^2)^2 = P^4$
2	1	$P^3 * P^4 = P^7$	$(P^4)^2 = P^8$
3	0	P^7	$(P^8)^2 = P^{16}$
4	1	$P^7 * P^{16} = P^{23}$	$(P^{16})^2 = P^{32}$
$e_5 = 1$, thus C:= $P^{23} * P^{32} = P^{55}$			

2.2.2 LR Binary Method

Inputs : p, e, N

Output : $C := p^e \ mod \ N$

K : number of bits in e;

1) $C := 1$;
2) For i= k-1 down to 0 do
3) $C := C \times C \ mod \ N$ (square)
4) if(e_i =1) then $C := C \times p \ mod \ N$;(multiply)
5) End if;
6) End for;
7) return C;

The bits of e are scanned from most significant bit to least significant bit, if the bit index i of e is equal 1 it performed two modular multiplications(multiply and square) else performed only one (square).

For (e = 55), LR algorithm will work as shown in table 2.2 .

Table 2.2 LR Binary Method [12]

When e = 55 and k= 6 bit			
i	**e_i**	**Step 3(C)**	**Step 6(P)**
5	1	1 * 1 = 1	1*P = P
4	1	P* P = P^2	P^2*P = P^3
3	0	P^3 * P^3 = P^6	P^6
2	1	P^6*P^6 =P^{12}	P^{12}*P = P^{13}
1	1	P^{13}* P^{13}= P^{26}	P^{26}*P = P^{27}
0	1	P^{27}* P^{27}= P^{54}	P^{54}*P = P^{55}

The main points for the two algorithms are

- Both methods require *k* squarings and an average of *½ (k)* multiplications where *k is the number of bits in e* .
- Both methods require two registers p and C.
- The multiplication and squaring computations in the RL method are independent of each other so the execution could be done in parallel[12].

The implementation in this thesis based on RL algorithm because of parallelism computing possibility in performing in results.

2.3 Modular Multiplication

The efficiency of the modular exponentiation depends basically on implementing an optimum modular multiplication as seen in RL and LR binary methods algorithm[13,14,15,16].

Modular multiplication is an essential computation of ($Z = X \times Y \ mod \ M$ where X , Y, M and Z are input integer numbers).

The four known methods for computing modular multiplication are

- Montgomery multiplication,
- Standard Interleaved Multiplication,
- Faster Montgomery multiplication
- Modified interleaved multiplication.

2.3.1 Montgomery Multiplication

Peter L. Montgomery algorithm invented in 1985 for computing $z = \frac{X \times Y}{2^n} \ mod \ M$ (X, Y, M are numbers, n is the number of bits in X). This algorithm can perform first conversion of numbers to Montgomery domain and then the result is re-converted into Montgomery domain, This

transformation exchanges division by several shift operations, that are accomplished according to the following equations [14]:

$Xm = (X \times 2^n) mod\ M$ ……..... (2.2)

$Ym = (Y \times 2^n) mod\ M$ …… (2.3)

$Z = (X \times Y)\ mod\ M$ …… (2.4)

$Zm = Mon\ pro\ (Xm, Ym, M)$ …….. …. (2.5)

$Zm = X \times Y \times 2^n mod\ M$ …… (2.6)

$Zm = Z \times 2^n mod\ M$ ……..… (2.7)

The key concepts of the Montgomery algorithm are the following :

A. Adding a multiple of M to the intermediate results doesn't affect the value of the final result, because the result is computed modulo M and M is an odd number.

B. After each addition in the internal loop the least significant bit (LSB) of the intermediate result is tested. If it equals 1, the intermediate result is odd then add M to make it even. This even number can be divided by 2 with zero remainder. The division by 2 reduces the intermediate result to n+1 bits again.

C. After n steps of these divisions one division by 2^n can be performed. This algorithm is very easy to implement since it operates on least significant bit first and does not require any comparisons. The hardware implementation is presented in algorithm 1 and described in Figure 2.1 [13] and [14].

Algorithm 1: Montgomery multiplication[14,13]

Inputs : X , Y , M with X > 0,Y < M

Output : Z = $\frac{X \times Y}{2^n}\ mod\ M$

x_i : i^{th} bit of X;

n: number of bits in X

z_0: LSB of Z

1) Z:=0;
2) for(i=0 ; i<n ; i++){
3) Z:= Z + x_i × Y;
4) Z:= Z + z_0 × M;
5) Z:= Z div 2;}
6) if (Z ≥ M) then Z := Z – M

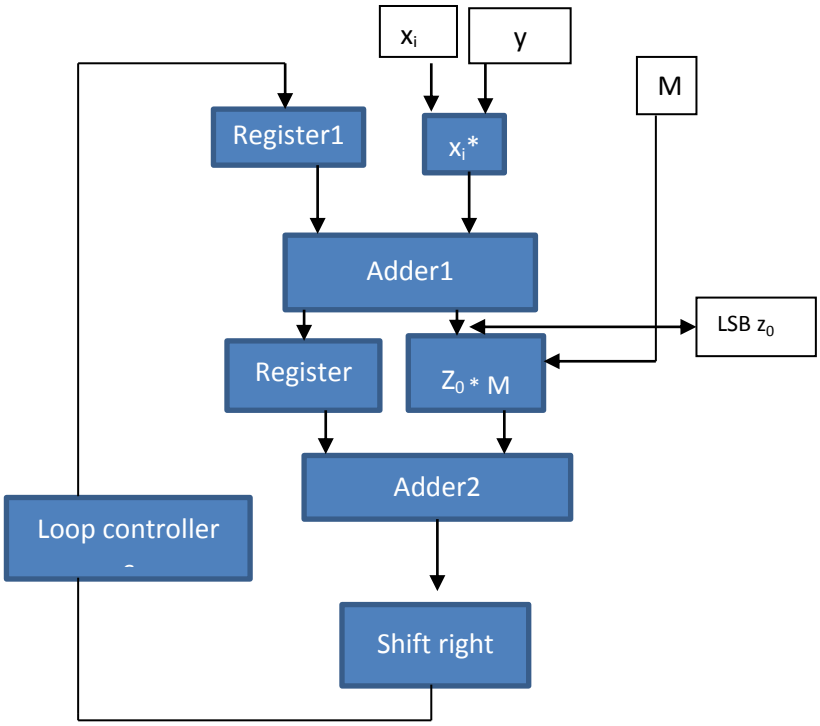


Figure 2.1 The Inner Loop Of Montgomery Algorithm [13].

Table 2.3 shows numeric example of Montgomery algorithm.

Table 2.3 Montgomery Example

$X = 11(1011)_b$, $Y = 7(0111)_b$, $M = 13(1101)_b$, $Z = 0$				
I	x_i	Z	Z	Z
1	1	7	20	10
1	1	17	30	15
0	0	15	28	14
1	1	21	34	17
17 -13 = 4				

2.3.2 Standard Interleaved Multiplication Algorithm

This algorithm was invented to keep the intermediate results as short as possible. For n steps the algorithm performs the following operations:

1. Shift left : $2 \times Z$
2. Partial product calculation: $x_i \times Y$
3. Add step (1) to step (2) results in : $2 \times Z + x_i \times Y$
4. Two subtractions modulus from the result in third step

   If ($Z \geq M$) then $Z := Z - M$;

   If ($Z \geq M$) then $Z := Z - M$;

The hardware implementation presented in algorithm 2 and described in Figure 2.2 [13,14].

Algorithm2 : Standard interleaved modulo multiplication[13,14]

Inputs: X, Y, with $0 \leq X , Y \leq M$

output: $Z = X \times Y \, mod \, M$

n : number of bits in X;

x_i: i^{th} bit in X ;

1) Z:= 0 ;
2) For (i = n-1 ; i≥0 ; i--) {
3) Z := 2 *Z ;
4) I := x_i *Y;
5) Z := Z + I;
6) If (Z ≥M) then Z := Z – M ;
7) If (Z ≥M) then Z := Z – M ; }

The main advantages of this algorithm are the following :

- The whole algorithm requires one loop only.
- The intermediate registers are not longer than (n+2) bits.

yet there are some disadvantages as well :

- The algorithm requires one adder and two subtractors in steps (5) , (6) and (7) .
- The latency to perform steps (4) and (5) because the addition in step (5) has to wait for step (4) end.
- The comparisons in steps (6) and (7) are of full bit length of Z and cannot be pipelined without delay because the result in step (7) depends on the result of step (6)[13,14].

Later In this thesis the latency problem was solved by modifying the interleaved algorithm which is a newly added feature by the author.

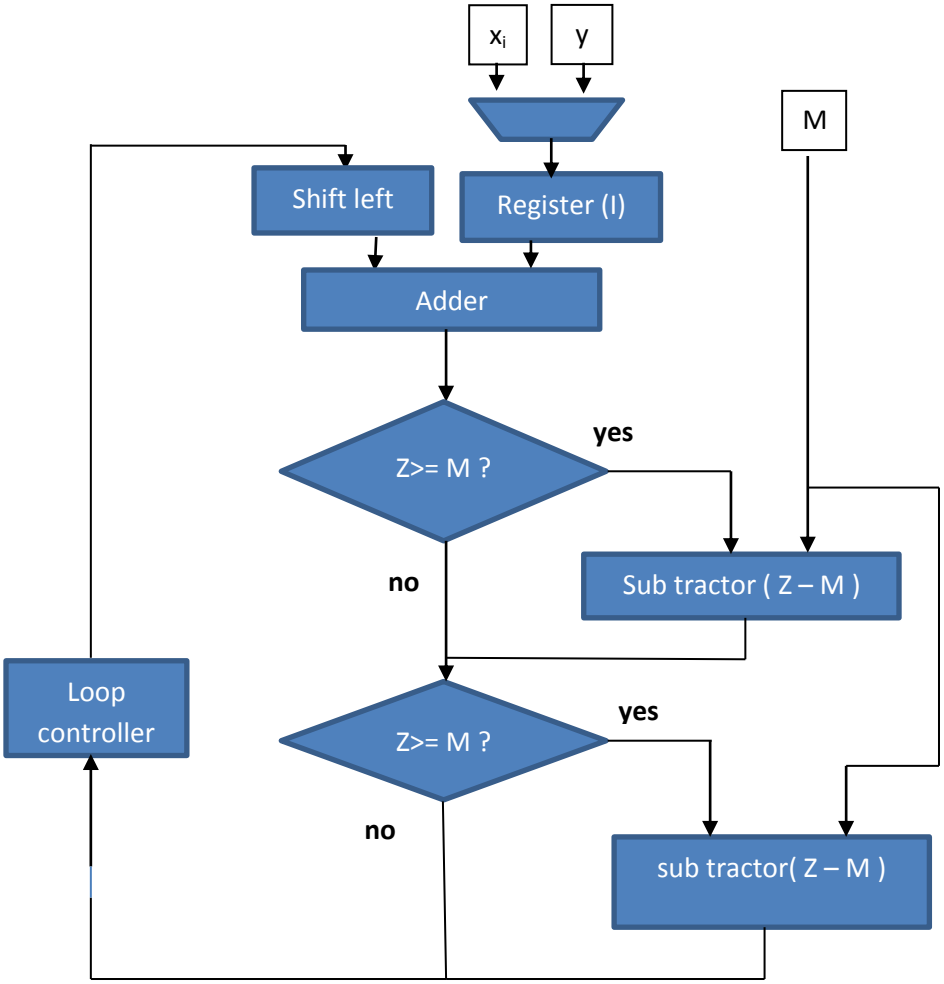


Figure 2.2 Standard Interleaved Multiplication Method[3,4]

Table 2.4 shows numeric example of interleaved algorithm.

Table 2.4 Standard Interleaved Example

$X = 11(1011)_b$, $Y = 7(0111)_b$, $M = 13(1101)_b$, $Z = 0$,					
I	Z	x(i)	Z	Z	Z
3	0	1	7	7	7
2	14	0	14	1	1
1	2	1	9	9	9
0	18	1	25	12	12

2.3.3 Faster Montgomery Algorithm

Method to reduce the chip area for practical hardware implementation of Montgomery Algorithm by computing the four Intermediate results to be added in the loop of the algorithm that means reduction in the number of additions from 2 to 1 inside the loop in Montgomery.

There are four possible scenarios for this method[13]:

A. if the old value of the result is an even number, and if the bit x_i of X is 0, then add none before will be performing the reduction of result by division by 2 (right shift).
B. if the old value an odd number and if the bit x_i of X is 0, then add M will be to make the intermediate result even and then divide the result by 2 (right shift).
C. if the old value of the intermediate result in the loop is an even number, and if the bit x_i of X is 1, with the incrimination of $x_i \times Y$ is even, too, so there is no need to add M to make the intermediate result even. In the loop add Y before performing the division by 2.
D. The same scenario is necessary if the old value of result is even, and the bit x_i of X is 1 , and Y is odd, in this case, Z +Y+M will be an even number, too. The computation of Y+M can be done prior to the loop. This saves one of the two additions. The hardware implementation is presented in algorithm 3 and described in Figure 2.3 [14].

Algorithm 3 : Faster Montgomery multiplication[14]

— Inputs : X,Y,M with X>0,Y<M

— Output : Z= $\frac{X \times Y}{2^n} \ mod \ M$

— x_i : i^{th} bit of X;

— n: number of bits in X;

— z_0: LSB of Z;

— y_0: LSB of y;

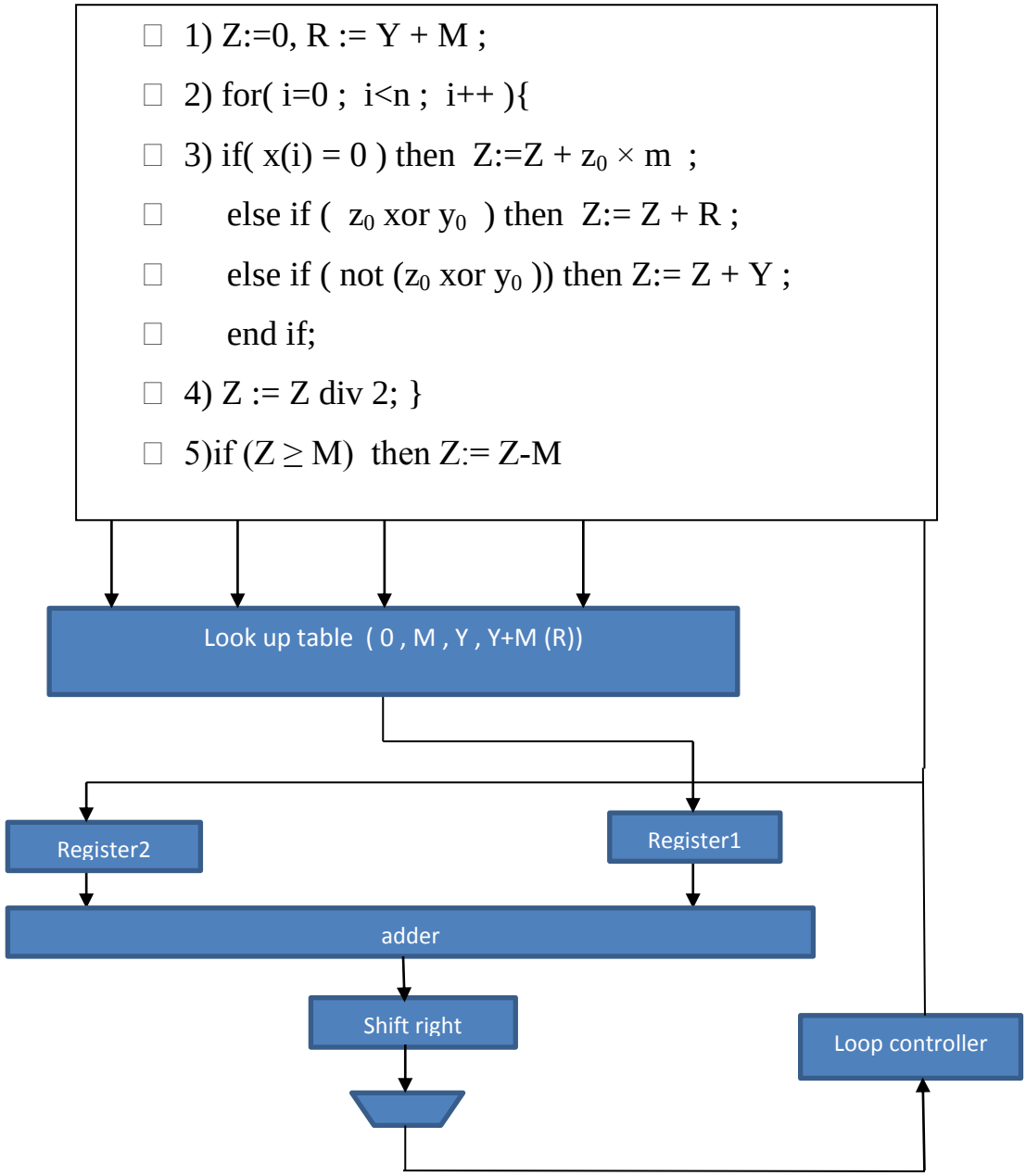


Figure 2.3 The Inner Loop Of Faster Montgomery Method [13]

Table 2.5 shows numeric example for faster Montgomery algorithm.

Table 2.5 Faster Montgomery Example

$x = 11(1011)_b$, $y = 7(0111)_b$, $M = 13(1101)_b$, R= 20 ,Z = 0, y(0) = 1			
I	x(i)	Z	Z
0	1	20	10
1	1	30	15
2	0	28	14
3	1	34	17
Step 5 : 17 -13 = 4			

2.3.4 Modified (Contributed) Interleaved Multiplication

An idea contributed in this thesis that modifies interleaved multiplication algorithm to decrease clock latency from 4 to 3 inside the loop by computing value (R) which equals (2 × M) prior the loop and using the following scenarios in step 6 of interleaved algorithm :

1) If the previous value of Z is larger than R then subtract (R) from Z
2) Else if the previous value of Z is larger than M then subtract (M) from Z.

The hardware implementation is presented in algorithm 4 and described in Figure 2.4 .

Algorithm4 : Modified interleaved modulo multiplication[16]

Inputs: X, Y, with $0 \leq X , Y \leq M$

output: $Z = X \times Y \, mod \, M$

n : number of bits in X;

x_i: i^{th} bit in X ;

1) Z:= 0 , R := 2 × M;

2) For (i = n-1 ; i ≥ 0 ; i--) {

3) Z := 2 × Z ;

4) I := x_i ×Y;

5) Z := Z + I;

6) If (Z ≥ R) then Z := Z – R ;

else if (Z ≥ M) then Z := Z – M ; }

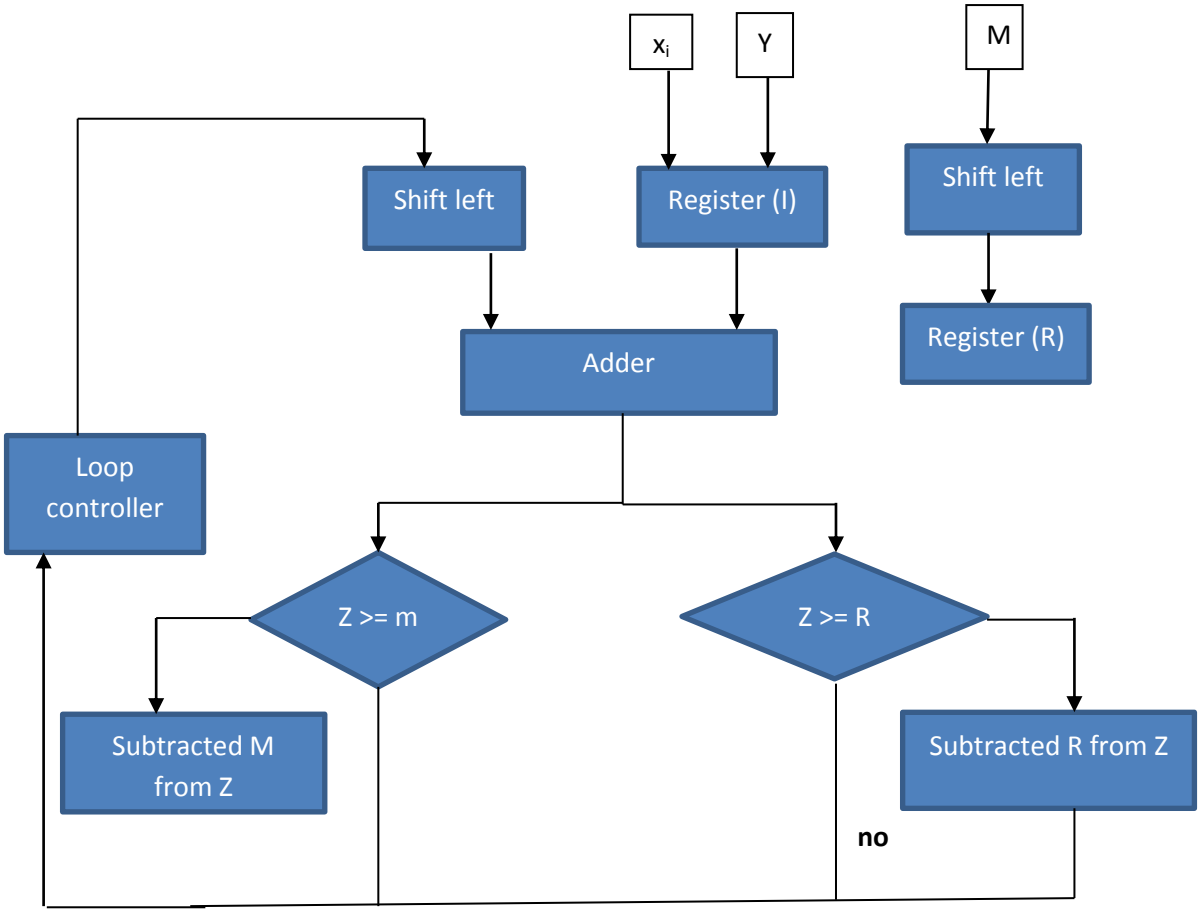


Figure 2.4 Modified Interleaved Multiplication Method.

Table 2.6 shows numeric example for algorithm .

Table 2.6 Modified Interleaved Example

$X = 11(1011)_b$, $Y = 7(0111)_b$, $M = 13(1101)_b$, Z = 0, R = 26				
I	Z	x(i)	Z	Z
3	0	1	7	7
2	14	0	14	1
1	2	1	9	9
0	18	1	25	12
Final result equal to 12				

2.4 Chinese Remainder Theorem(CRT)[18]

New method is used to decrease time of implementation of RSA by using strategies to dividing the width of the numbers by 2 then perform implementation. These strategies are described in algorithm 5 and figure 2.5

Algorithm 5 : RSA_CRT Algorithm[18]

Input : m , e , RSA *private keys* = (p , q)

m : plaintext or message

e : public key

p , q : two prime numbers are randomly chosen

Output : $C = m^e \ mod \ N$ where

N : public key calculated from key generation process (p × q)

C : cipher text

1) Calculate

$C1 = m\ MOD\ p$ ……(2.2)

$C2 = m\ MOD\ q$ ……(2.3)

2) Calculate

$dp = e\ MOD\ (p-1)$ ……(2.4)

$dq = e\ MOD (q-1)$……(2.5)

3) Calculate

$x1 = C1^{dp}\ MOD\ p$ ……(2.6)

$x2 = C2^{dq}\ MOD\ q$ ……(2.7)

4) Find

$Cp = q^{-1}\ MOD\ p$ ……(2.8)

$Cq = p^{-1}\ MOD\ q$ ……(2.9)

$C = (q \times Cp \times x1 + p \times Cq \times x2)\ MOD\ N$ ……(2.10)

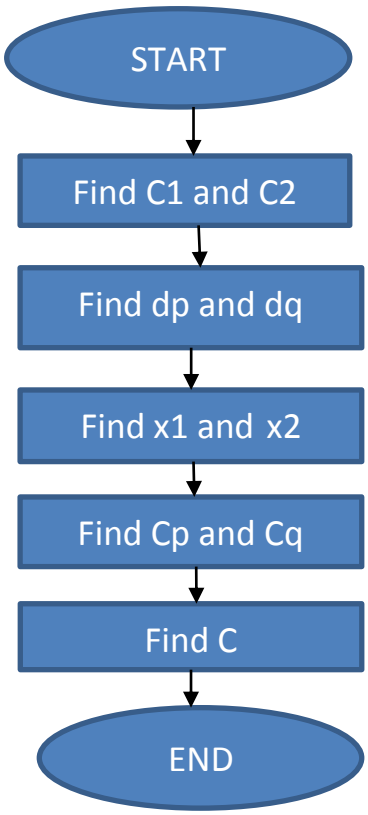


Figure 2.5 CRT Flow Chart

To illustrate the algorithm a numerical example applied as shown :

Suppose values of p = 37 , q = 89 , N = p × q = 3293 , m = 2494, e = 2987, how to compute 2494^{2987} mod 3293 ?

Input : m= 2494, e= 2987 , p= 37, q=89 .

1. $C1 = 2494 \; mod \; 37 = 15$, $C2 = 2494 \; mod \; 89 = 2$.
2. $dp = 2987 \; mod \; 36 = 35$, $dq = 2987 \; mod \; 88 = 83$.
3. $x1 = 15^{35} \; mod \; 37 = 5$, $x2 = 2^{83} \; mod \; 89 = 64$.
4. $Cp = 89^{-1} \; mod \; 37 = 5$, $Cq = 37^{-1} \; mod \; 89$.
5. $C = (89 \times 5 \times 5 + 37 \times 77 \times 64) \; mod \; 3293 = 153$.

Chapter Three

Software Implementation

3.1 Introduction

This chapter explains RSA implementation using Java high level language .The implementation is tested using 1024 bits key size.

3.2 Java libraries used

- java.math.BigInteger

  Java.math.BigIntegers library has a collection of instructions for executing long number widths arithmetic .
- java.util.Random

  java.util.Random library used for random number generation.
- java.io.
- java.util.concurrent.TimeUnit

  java.util.concurrent.TimeUnit library is used to find execution time of encryption and decryption algorithms.

3.3 RSA in Java

Figure 3.1 shows a flow chart of RSA algorithm :

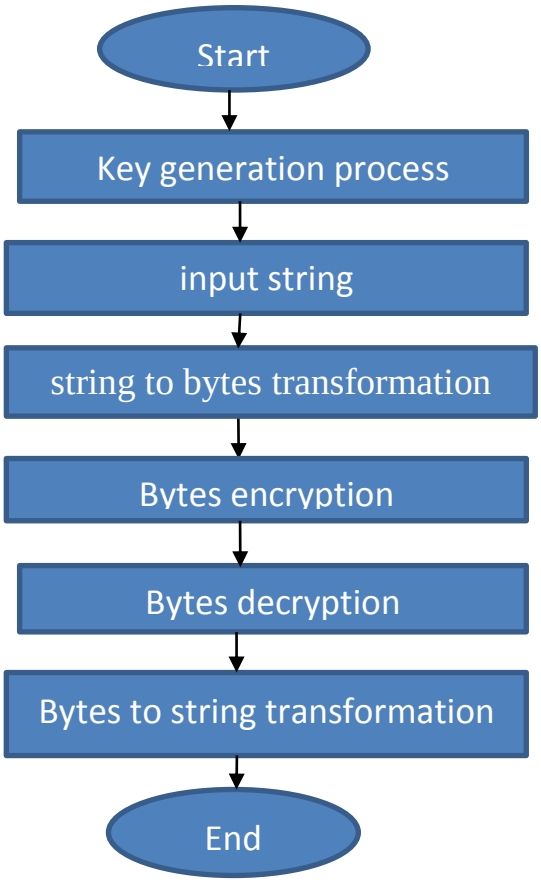


Figure 3.1 RSA Flowchart In Java

Figure 3.2 shows a flow chart for key generation process :

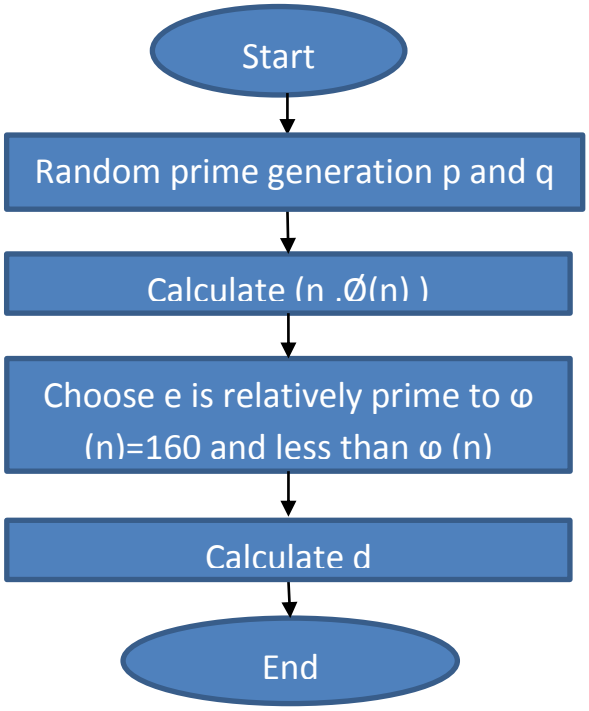


Figure 3.2 RSA Key Generation In Java

The key generation steps are

- Select p, q — p and q both prime, $p \neq q$
- Calculate n=p*q
- Calculate $\varphi(n)=(p-1)(q-1)$
- Select integer e — $\gcd(\varphi(n),e)=1;\ 1<e<\varphi(n)$
- Calculate d — $d = e^{-1} \pmod{\varphi(n)}$
- Public key — PU={e,n}
- Private key — PR={d,n}

The string was entered using DataInputStream then converted to bytes using GetBytes() , before encryption process the start time was stored using System.nanoTime(), the encryption equation($c = p^e \bmod m$) was done using encrypted number .modpow(e ,n) then end time stored using System.nanoTime().

The encryption time was calculated by subtracting start time from end time.

The decryption process started also done by decrypted number .modpow(d ,n) , and then the bytes converted to string bytes to string().

3.4 Encryption and Decryption Results

The plaintext was (zaid abdulsatar) , and bit length equal to 1024 bits the results of execution (see Appendix a). The encryption and decryption time in ms are shown in Table 3.1.

Key generation process :

p:
12236041962475294070741124887750793539251606387026253488874935448075197804469962232199079975235402434916611294921117695090934302550586941987540609690795959

q:
81114362954140813456371349639798854400715000022835940352452697944988089899891021878459937871943653847338063965781679104659902275115809447077771449343861 97

N is :
99251874886631845087302077416730661008979019834161360381838697226835377860031020981923904604485679026487496107358145920757283792241746547267892084832531911202390010132875264427965334733775926636229344046974313766063786043538049336651613855307122740798365318940644325377373549771146969173274619805278032977923

Public key is :
12394190533084738165116593404652019314904423022780262898705836481985422095342212585838477056006857142189272312279568160143901695852984457750571111138235723

private key is :
50382142745426563838835358495739244966263816725329355160246880576908897216682766277206061807734531251903881602936073710005759705737563384528892904990476144431937706695141392377988982904048313493699445730535997145894395221242536142522046059080046311890565919257781477372923392835299254334517360575633817154043

Plaintext(p) : zaid abdulsatar

String in Bytes(p) :
122971051003297981001171081159711697114

Encrypted String in Bytes (C) : 67-13198-63-391262-83-3011-798424102120-1213114-84-31-8014-18114189955-861154166-881051187218-12112649-7-14-41-1217711910438-4-11622-1-25-3820-6-9412982-1256455-116-422182-6028-47113-5335-116119-78-6192-56-6927-121-8382-17-5781-103-2432-3261-109126-84779-1149373-11294-37-6634912451-103125-67117-86372-8461127-10796-109-2436-20-106-22-105

Elapsed time of RSA enryption in nanoseconds : 29259186

Decrypted String in Bytes:
122971051003297981001171081159711697114

Elapsed time of RSA decryption in nanoseconds : 19264425

Decrypted String: zaid abdulsatar

Table 3.1 Encryption and Decryption Time in Java

Encryption time(ms)	Decryption time(ms)
29.259186	19.264425

Chapter Four

FPGA Implementation

4.1 Introduction

This chapter presents an implementation of RSA algorithms using Spartan-6 SP601 evaluation board XC6SLX16 device CSG324 package -2 speed XST based on VHDL language.

4.2 Montgomery Modular Multiplier

As shown in figure 4.1 Montgomery multiplier was designed with seven states :-

S0: assign (start = '0') , initialize the system with inputs (x , y , m) , flag counter =1 , p =0 , then make (start= '1'), go to S1 .

S1: if (x(i) = '1') then load adder1 with values of p , y then go to S2.

S2: if (p(0) = '1') : load adder2 with values of values of p , m then go to S3.

S3: shift value of p then go to S4.

S4: check the counter if finished make flag counter =0 , and go to S5 else go to S1.

S5: if (p >= m) decrement p by m, go to S6 .

S6: go to S0.

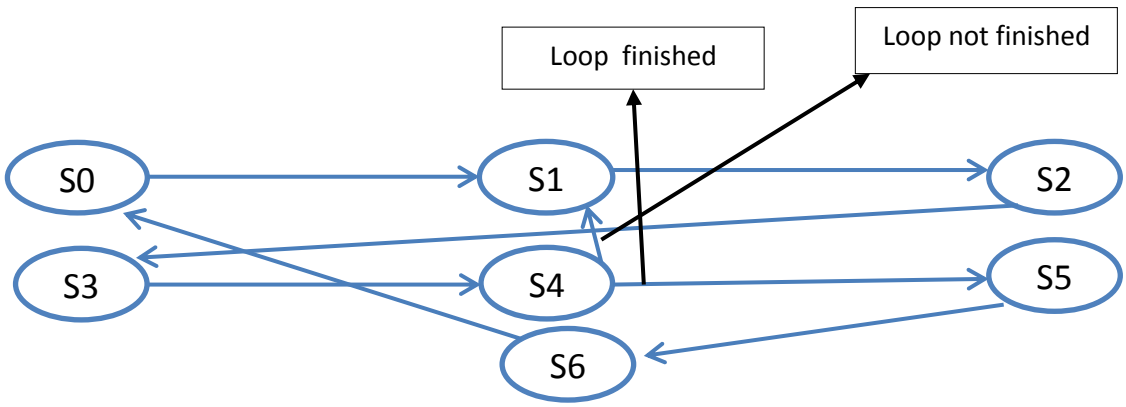


Figure 4.1 States Of Montgomery

The simulation result of Montgomery multiplier is presented in figure 4.2 The size of each number is 4 bit (where x = 11 , y= 7 , m = 13) , the start input signal controls the state of the multiplier, at first (start signal= "0"), which means the multiplier in loading state (storing the value of inputs in registers) , then start input signal changed to "1", the multiplier becomes in running state (processing the data) until done signal has become 1 (done signal = "1") that means the multiplier has finished running. It is found that the multiplier needs 18 clock cycles for finishing the calculation.

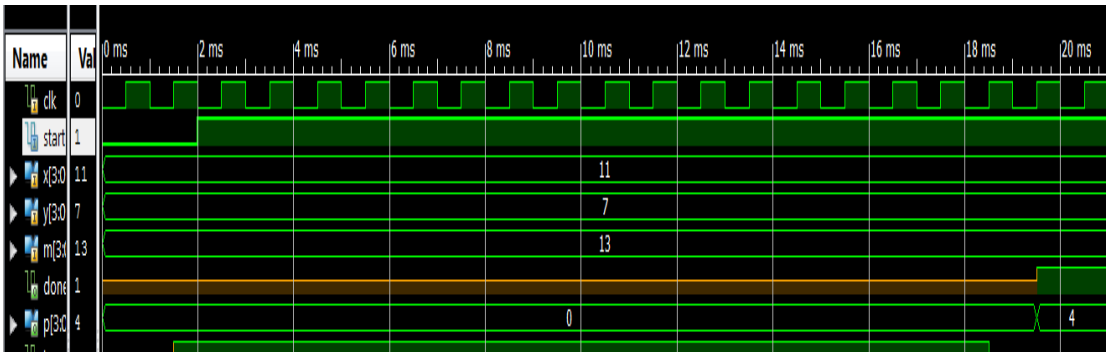


Figure 4.2 Montgomery Multiplier Timing(4 bits)

As in simulation result of Montgomery multiplier presented in figure 4.3 the size of each number is 8 bit where x = 88 , y= 7 , m = 187, it is found that the multiplier needed 34 clock cycle to finish its calculation.

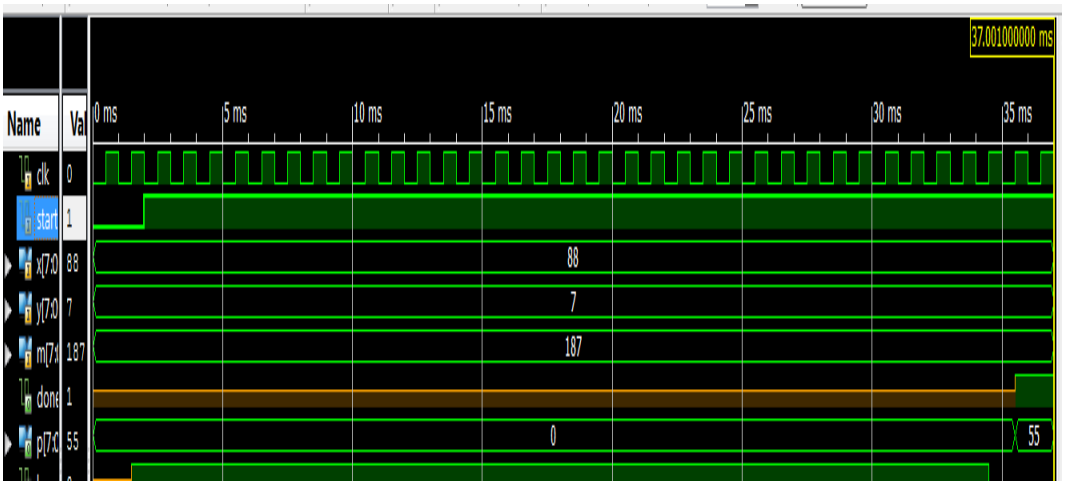


Figure 4.3 Montgomery Multiplier Timing (8 bits)

Table 4.1 shows Device utilization of Montgomery multiplier, the size of each number is 8 bit .

Table 4.1 Device Utilization Montgomery Multiplier (8 bits)

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	98	18224	0%
Number of Slice LUTs	153	9112	1%
Number of Fully LUT-FF Pairs	74	177	41%
Number of Bounded IOBs	35	232	15%
Number of BUFG / BUFGCTRL/BUFHCEs	1	16	6%

The simulation result of Montgomery multiplier presented in figure 4.4, the size of each number is 32 bit where x = 88 , y= 7 , m = 187, it is found that the multiplier needed 130 clock cycle to finish its calculation.

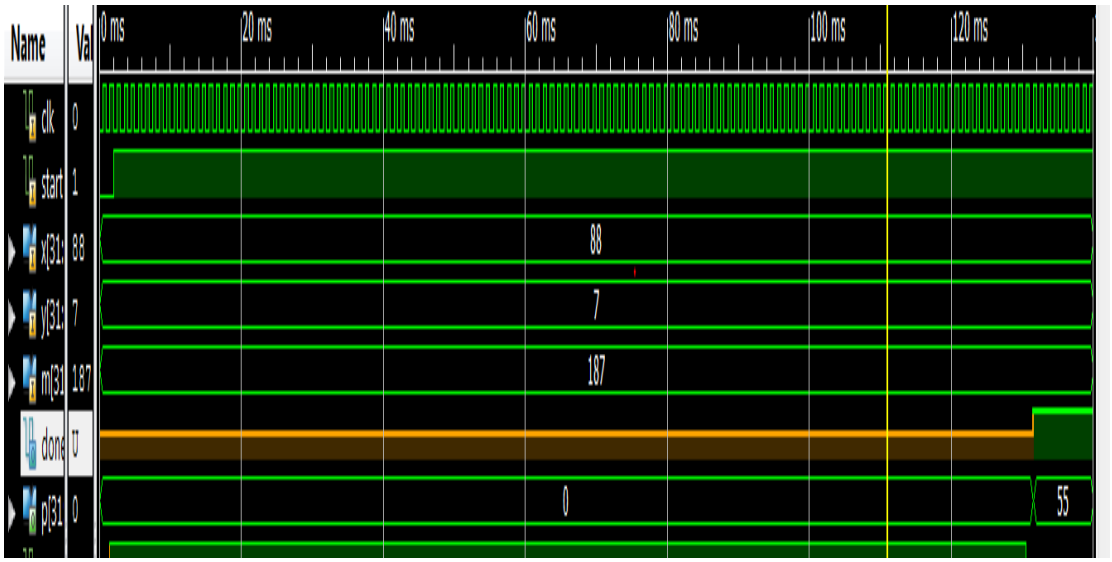


Figure 4.4 Montgomery Multiplier Timing (32 bits)

If the size of numbers in Montgomery was n, the number of clocks will be $(n \times 4 + 2)$. Table 4.2 shows number of clocks, maximum frequency and clock period for different size of bits.

Table 4.2 Montgomery Clocks And Operating Frequency

Number of bits(n)	Number of clocks required(n×4 +2)	Maximum frequency in MHZ	Minimum period in ns
4	18	200.521	4.987
128	514	129.076	7.747
256	1026	111.290	8.986
512	2050	66.946	14.937
1024	4098	37.256	26.842
2048	8194	19.743	50.650

4.3 Interleaved Modular Multiplier

As shown in figure 4.5 interleaved multiplier was designed with seven states :-

S0 : assign (start = '0') , initialize the system with inputs (x , y , m) , make flag counter =1 , p =0 , then make the (start= '1') declaring that data is ready, go to S1 .

S1 : shift the value of (P) , if (x(i) = '1') then load I register with Y else clear I register , go to S2 .

S2 : load adder with values of p , I registers then go to S3.

S3 : if the value of P larger than M decrement M from this value, go to S4.

S4 : if the value of P again larger than M decrement M from this value, go to S5.

S5 : check the index of bits if finished make the flag counter = 0 and then finish else go to state S1 .

S6 : go to S0.

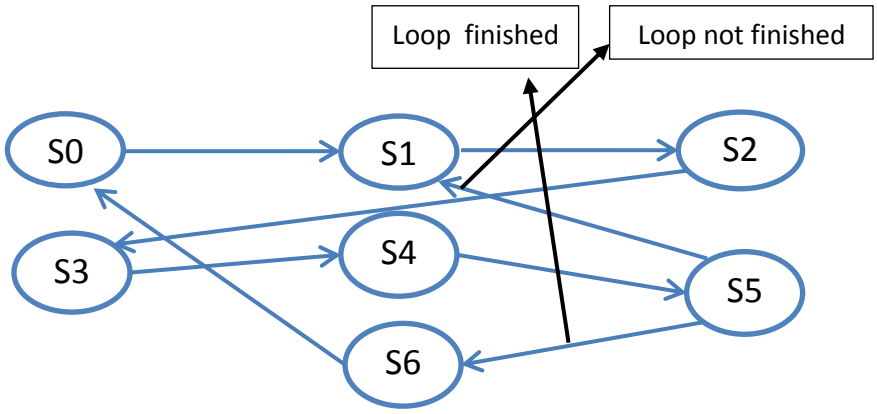


Figure 4.5 States Of Interleaved

simulation result of interleaved multiplier presented in figure 4.6 The size of each number is 4 bit (where x = 11 , y= 7 , m = 13) , the start input signal controls the state of the multiplier, at first (start signal= "0"), which means the multiplier in loading state (storing the value of inputs in registers) , then start input signal changed to "1", the multiplier becomes in running state (processing the data) until done signal has become 1 (done signal = "1") that means the multiplier is finished running. It is found the multiplier need 20 clock cycles for finished the calculation.

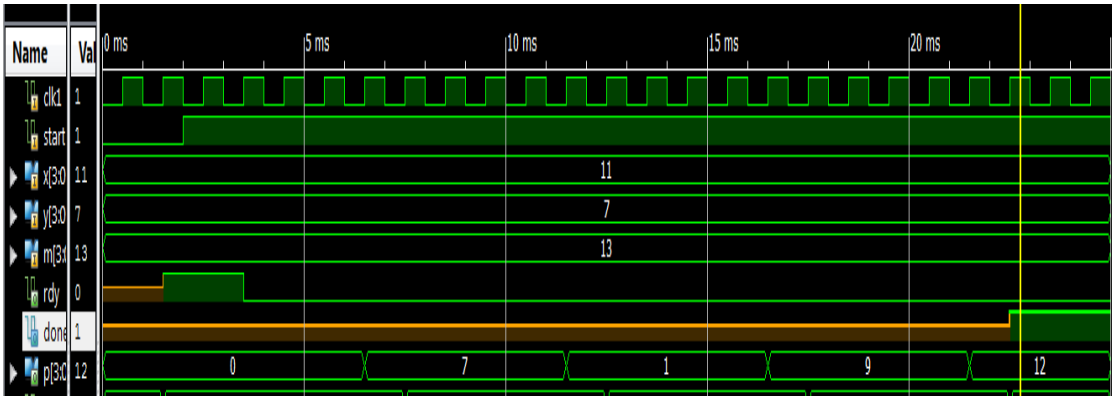


Figure 4.6 Interleaved Multiplier Timing (4 bits)

The simulation result of interleaved multiplier presented in figure 4.7 in which the size of each number is 8 bit where x = 88 , y= 7 , m = 187. It is found that the multiplier needed 40 clock cycle to finish its calculation.

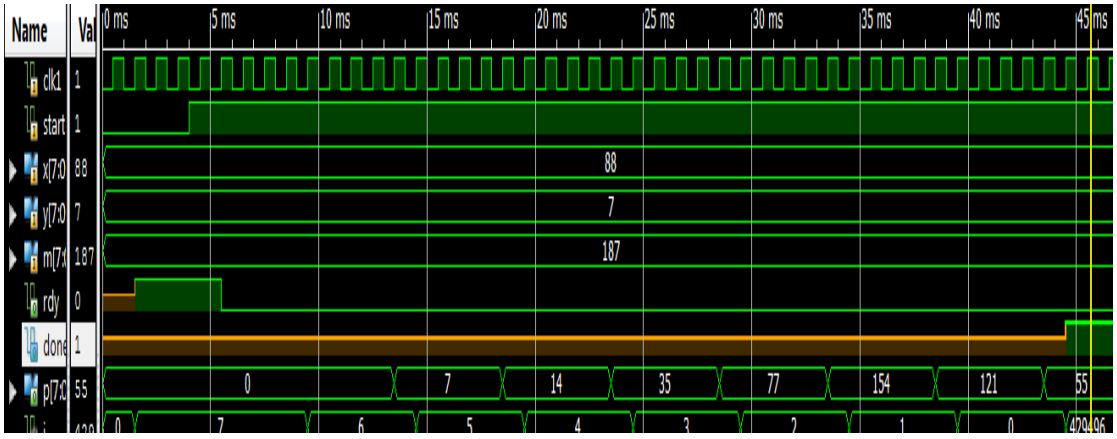


Figure 4.7 Interleaved Multiplier Timing (8 bits)

Device utilization of interleaved multiplier presented in Table 4.3, the size of each number is 8 bit .

Table 4.3 Device Utilization Interleaved Multiplier (8 bits)

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	78	18224	0%
Number of Slice LUTs	142	9112	1%
Number of Fully LUT-FF Pairs	78	142	54%
Number of Bounded IOBs	36	232	15%
Number of BUFG / BUFGCTRL/BUFHCEs	1	16	6%

The simulation result of interleaved multiplier presented in figure 4.8 where the size of each number is 32 bit and where x = 88 , y= 7 , m = 187, it is found that the multiplier needed 160 clock cycle to finish its calculation.

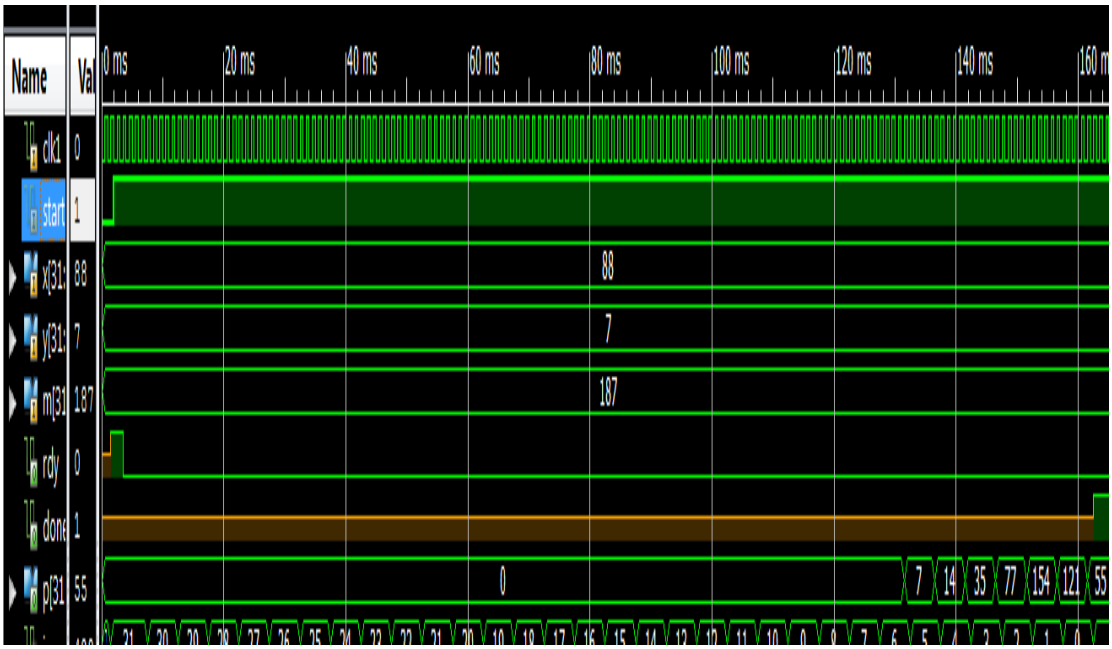


Figure 4.8 Interleaved Multiplier Timing (32 bits)

If the size of numbers in interleaved was n, the number of clocks will be $(n \times 5)$. Table 4.4 shows number of clocks, maximum frequency and clock period for different size of bits.

Table 4.4 Interleaved Clocks And Operating Frequency

No. of bits (n)	No. clocks required(n×5)	Maximum frequency in MHZ	Minimum period in ns
4	20	240.790	4.153
128	640	135.612	7.374
256	1280	110.253	9.070
512	2560	66.569	15.022
1024	5120	37.139	26.926
2048	10240	19.710	50.735

4.4 Faster Montgomery Modular Multiplier

As shown in figure 4.9 Faster multiplier which was designed with six states :-

S0 : assign (start = '0') , initialize the system with inputs (x , y , m) , make flag counter =1 , p =0 , then make the (start= '1'), go to S1 ..

S1 : load adder with values of p (getting it from look up table) and m then go to S2 .

S2 : shift right value of p, go to S3.

S3 : check the counter if finished make flag counter =0 , and go to S4 else go to S1

S4 : if (p >= m) decrement p by m, then go to S6 .

S5 : go to S0.

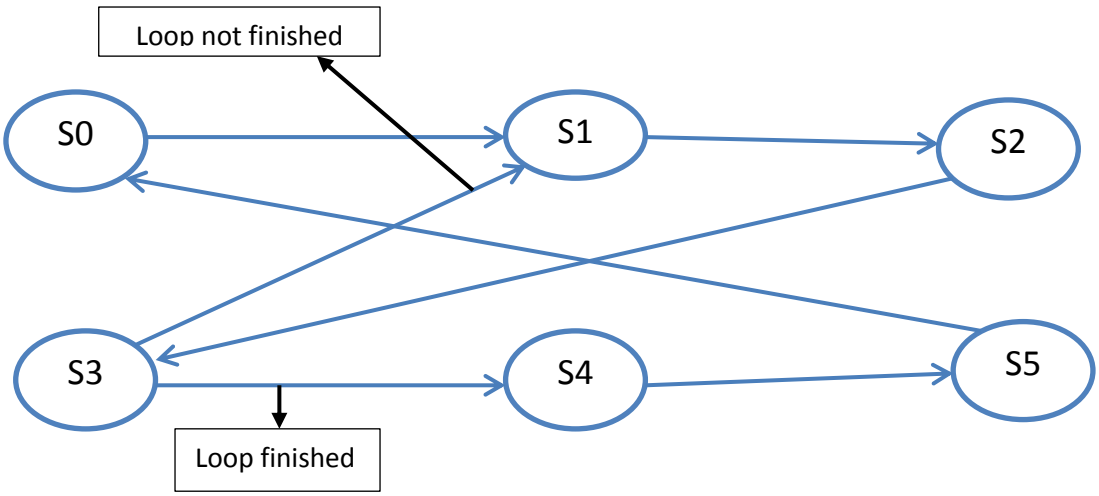


Figure 4.9 States Of Faster

The simulation result of Faster multiplier presented in figure 4.10 where the size of each number is 4 bit (where x = 11 , y= 7 , m = 13) , the

start input signal controls the state of the multiplier, at first (start signal= “0”), which means the multiplier in loading state (storing the value of inputs in registers) , then start input signal changed to “1”, the multiplier becomes in running state (processing the data) until done signal has become 1 (done signal = “1”) that means the multiplier is finished running. It is found that the multiplier needs 14 clock cycles for finishing the calculation.

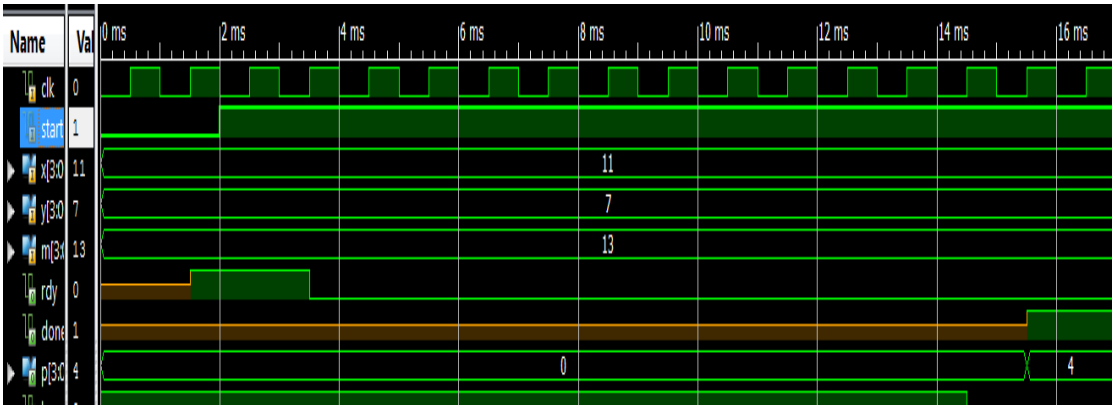


Figure 4.10 Faster Multiplier Timing (4 bits)

In the simulation result of Faster multiplier presented in figure 4.11 where the size of each number is 8 bit where x = 88 , y= 7 , m = 187, it is found that the multiplier needed 26 clock cycle to finish its calculation.

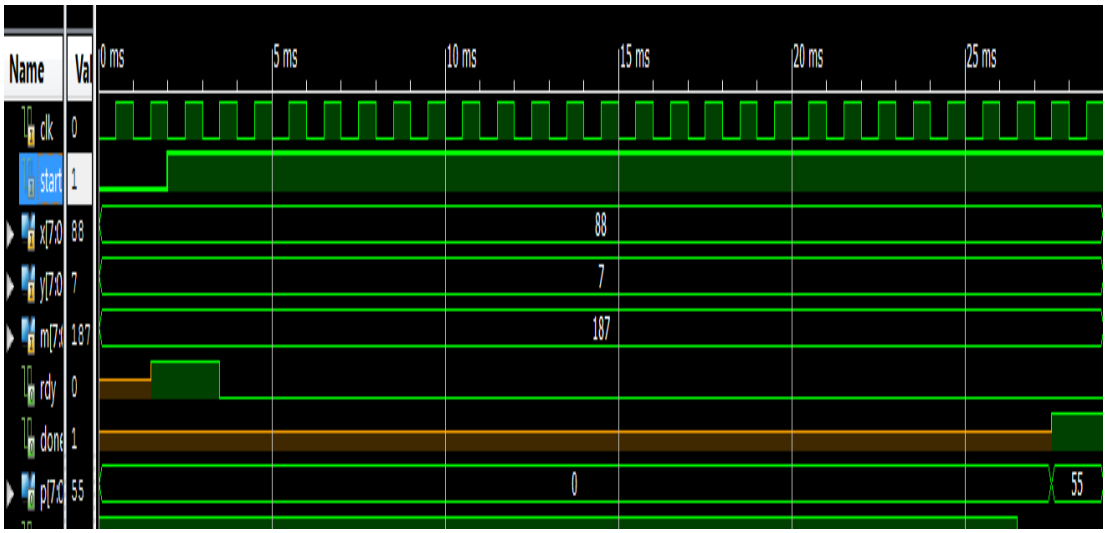


Figure 4.11 Faster Multiplier Timing (8 bits)

Device utilization of Faster multiplier presented in Table 4.5, the size of each number is 8 bit .

Table 4.5 Device Utilization Faster Multiplier (8 bit)

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	69	18224	0%
Number of Slice LUTs	62	9112	0%
Number of Fully LUT-FF Pairs	32	99	32%
Number of Bounded IOBs	36	232	15%
Number of BUFG / BUFGCTRL/BUFHCEs	1	16	6%

In the simulation result of Faster multiplier presented in figure 4.12 where the size of each number is 32 bit and where x = 88 , y= 7 , m = 187. it is found that the multiplier needed 98 clock cycle to finish its calculation.

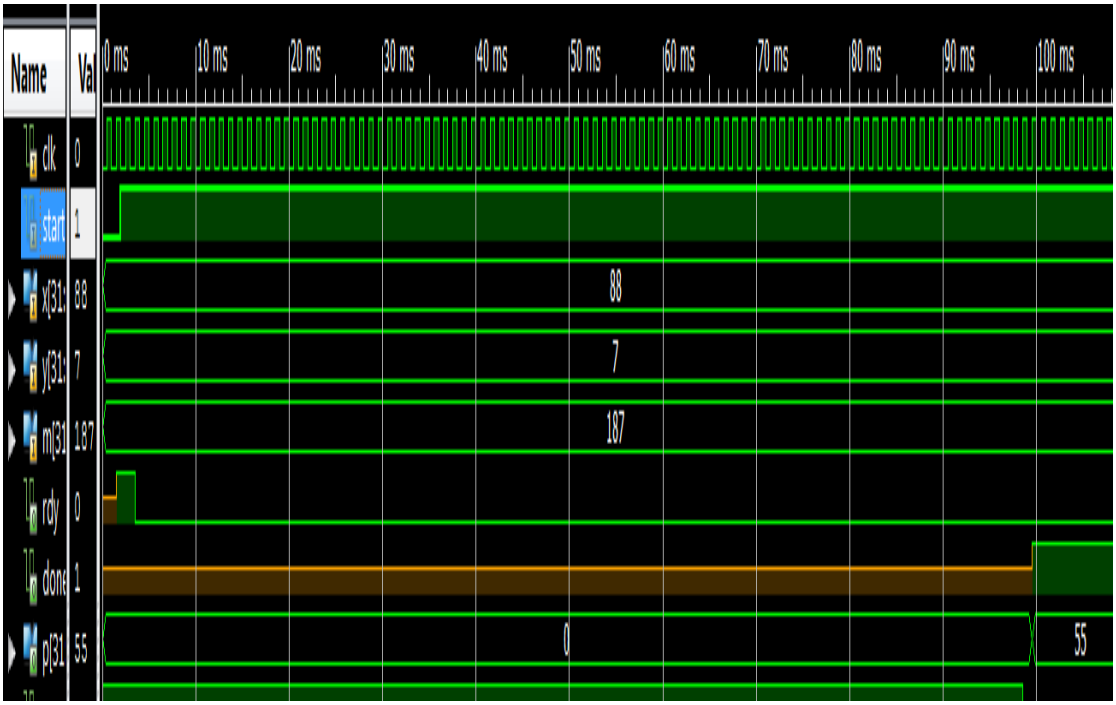


Figure 4.12 Faster Multiplier Timing (32 bits)

If the size of numbers in Faster is n, the number of clocks will be ($n \times 3 + 2$). Table 4.6 shows number of clocks, maximum frequency and clock period for different size of bits.

Table 4.6 Faster Multiplier Clocks And Operating Frequency

No. of bits (n)	No. of clocks required (n*3+2)	Maximum frequency in MHZ	Minimum period in ns
4	14	188.893	5.294
128	386	136.091	7.348
256	770	100.406	9.960
512	1538	62.848	15.911
1024	3074	35.951	27.816
2048	6146	19.371	51.624

4.5 Modified(Contributed) Interleaved Modular Multiplier

As shown in figure 4.13 Modified interleaved multiplier was designed with six states :-

S0 : assign (start = '0') , initialize the system with inputs (x , y , m) , make flag counter =1 , p =0 , then make the (start= '1') declaring that data is ready, find R (shift left m), go to S1 .

S1 : shift the value of (P) , if (x(i) = '1') then load I register with Y else clear I register , go to S2 .

S2 : load adder with values of p , I registers then go to S3.

S3 : if the value of P is larger than M decrement M from this value, else if the value of P again is larger than R decrement R from this value, go to S4.

S4 : check the index of bits if finished make the flag counter = 0 and then go to S5 else go to state S1 .

S5 : go to S0.

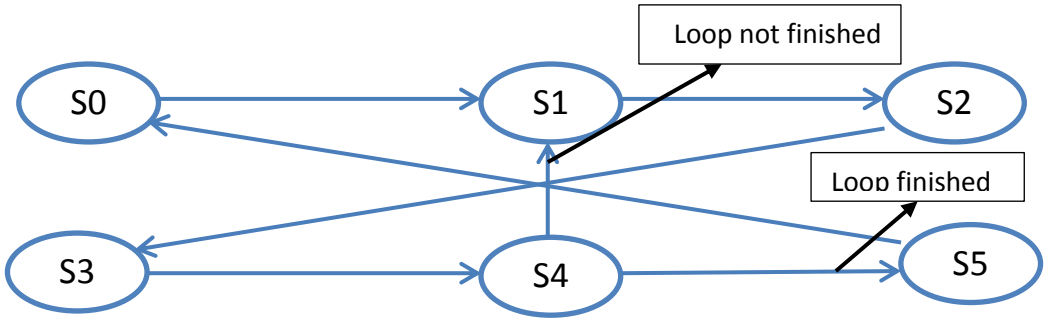


Figure 4.13 States Of Modified

The simulation result of Modified interleaved multiplier presented in figure 4.14 The size of each number is 4 bit (where x = 11 , y= 7 , m = 13), the start input signal controls the state of the multiplier, at first (start signal= “0”), which means the multiplier in loading state (storing the value of inputs in registers) , then start input signal changed to “1” and the multiplier becomes in running state (processing the data) until done signal becomes 1 (done signal = “1”) that means the multiplier has finished running. It is found that the multiplier needs 16 clock cycles for finishing the calculation.

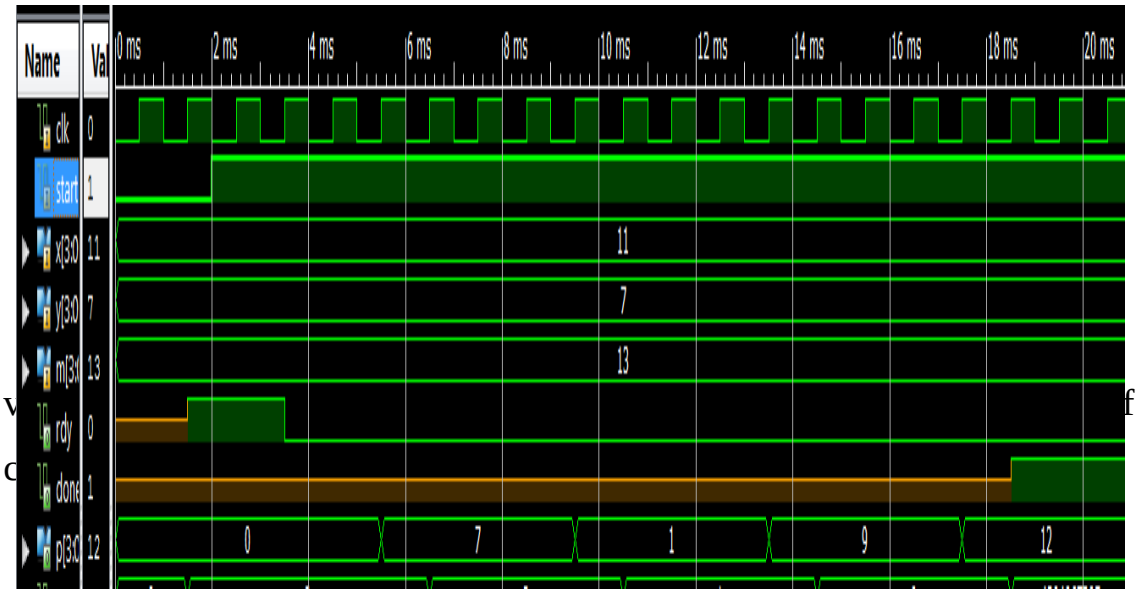


Figure 4.14 Modified Interleaved Multiplier Timing (4 bits)

In the simulation result of Modified multiplier presented in figure 4.15 where the size of each number is 8 bit where x = 88 , y= 7 , m = 187, it is found that the multiplier needed 32 clock cycle to finish its calculation.

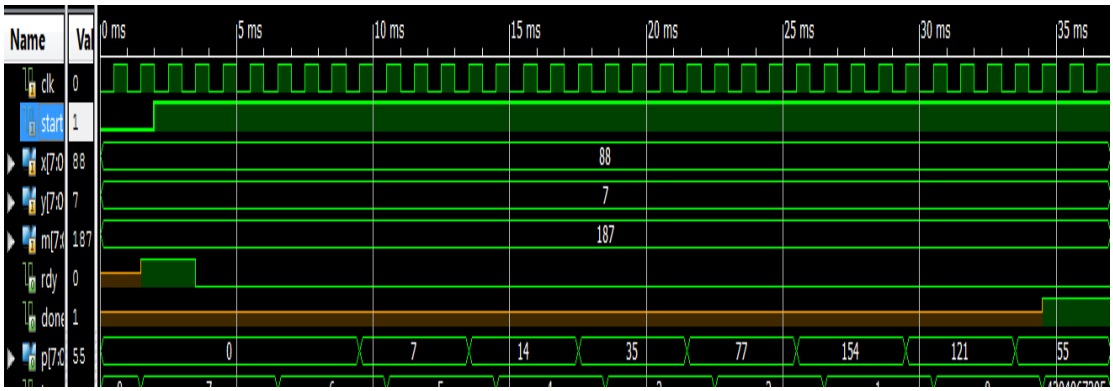


Figure 4.15 Modified Interleaved Multiplier Timing (8 bits)

Device utilization of modified multiplier presented in Table 4.7, the size of each number is 8 bit.

Table 4.7 Device Utilization Modified Multiplier (8bits)

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	34	18224	0%
Number of Slice LUTs	45	9112	0%
Number of Fully LUT-FF Pairs	24	60	40%
Number of Bounded IOBs	20	232	8%
Number of BUFG / BUFGCTRL/BUFHCEs	1	16	6%

In the simulation result of Modified multiplier presented in figure 4.16 where the size of each number is 32 bit where x = 88 , y= 7 , m = 187, it is found that the multiplier needed 128 clock cycle to finish its calculation.

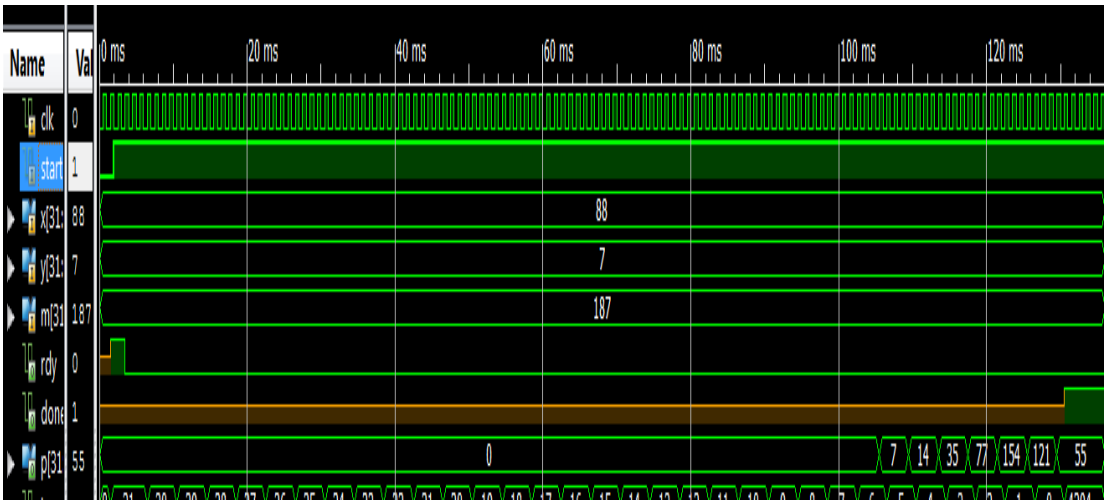


Figure 4.16 Modified Interleaved Multiplier Timing(32 bits)

If the size of numbers in Modified interleaved multiplier was n, the number of clocks will be ($n \times 4$).

Table 4.8 shows the number of clocks, the maximum frequency and the clock period for different size of bits.

Table 4.8 Modified Multiplier Clocks And Operating Frequency

No. of bits (n)	Number of clocks required (n×4)	Maximum frequency in MHZ	Minimum period in ns
4	16	231.054	4.328
128	512	133.726	7.478
256	1024	108.701	9.200
512	2048	66.000	15.151
1024	4096	36.961	27.056
2048	8192	19.660	50.864

4.6 Multipliers Analysis

Table 4.9 and figure 4.17 shows operation frequency differences among multipliers knowing that :

$$Mp = \frac{1}{Mf} \quad \text{.....(4.1)}$$

Where (Mp : minimum period , Mf : maximum frequency).

It can be found that Montgomery multiplier is the fastest clock frequency.

Table 4.9 Frequencies Of Different Sizes Multipliers

No. of bits	Montgomery	Faster Montgomery	Interleaved	Modified interleaved
256	111.290	100.406	110.253	108.701
512	66.946	62.848	66.569	66.000
1024	37.256	35.951	35.951	36.961
2048	19.743	19.371	19.710	19.660

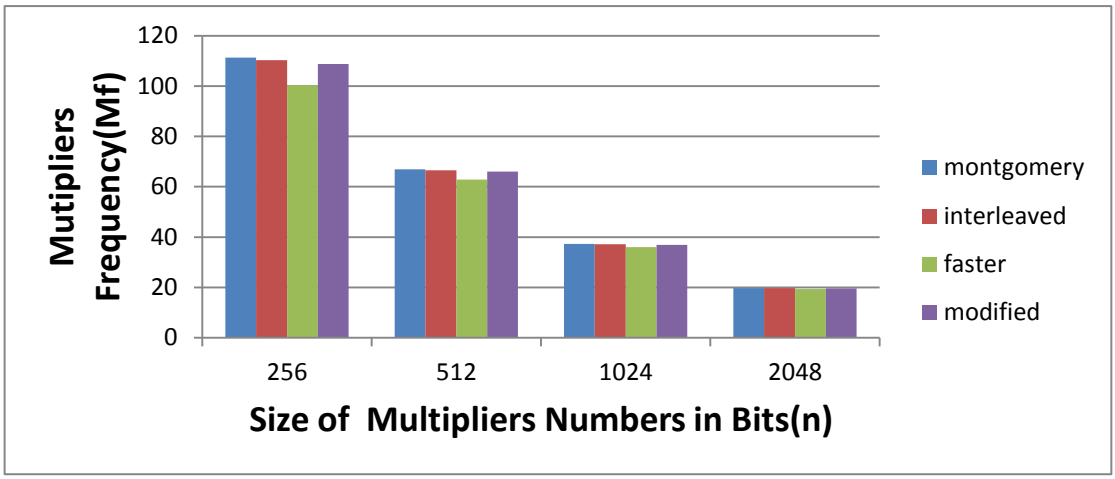


Figure 4.17 Differences Among Multipliers frequencies

The table 4.10 and figure 4.18 showed that the "Faster method" is the fastest multiplier because it needed a less number of clocks.

Table 4.10 No. of Clocks For Each Multiplier

No. of bits	Montgomery	Faster Montgomery	Interleaved	Modified Interleaved
256	1026	770	1280	1024
512	2050	1538	2560	2048
1024	4098	3074	5120	4096
2048	8194	6146	10240	8192
N	$n \times 4 + 2$	$n \times 3 + 2$	$n \times 5$	$n \times 4$

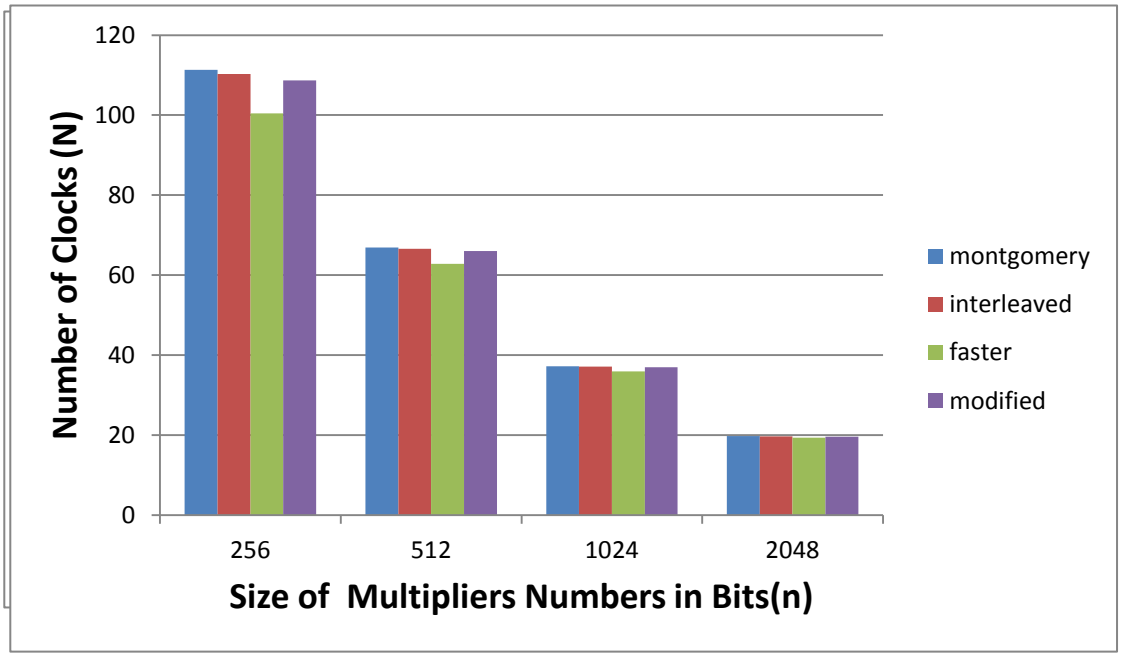


Figure 4.18 Number Of Clocks Needed By Each Multiplier

In FPGA hardware execution time can be calculating using the following equations:

$$Ht = N \times \frac{1}{Mf} \quad \text{.....(4.2)}$$

Or

$$Ht = N \times Mp \quad \text{.....(4.3)}$$

Where (Ht : Hardware time , N : number of clocks , Mf : maximum frequency , Mp : minimum period) .

By using above equation it can be found that the hardware execution time for each multiplier as in table 4.11 and figure 4.19. The results showed faster multiplier has the fastest speed among the others.

Table 4.11 Hardware Execution Time For Each Multiplier(µs)

Number of bits	Montgomery	Faster Montgomery	Interleaved	Modified Interleaved
256	9.219	7.668	11.609	9.42
512	30.621	24.471	38.456	31.03
1024	109.995	85.505	142.416	110.819
2048	415.033	317.278	519.533	416.683

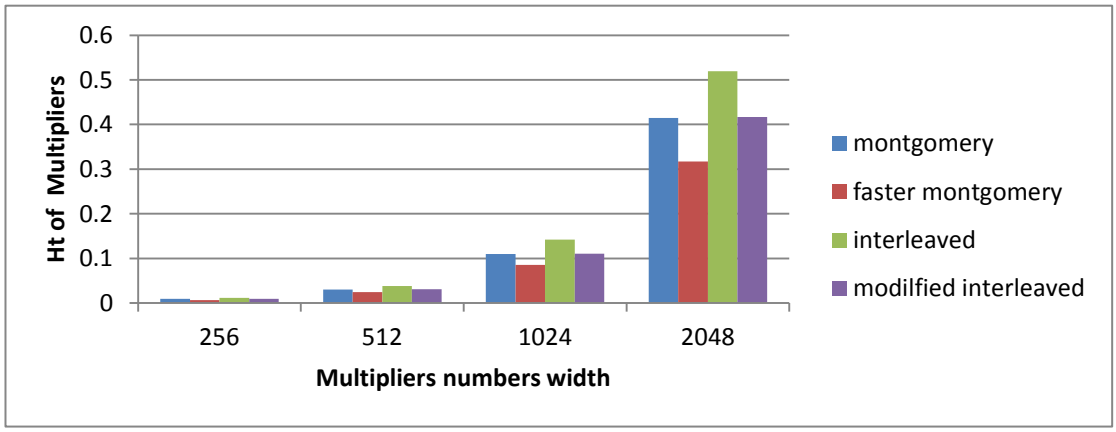


Figure 4.19 Hardware Execution Time For Each Multiplier(μs)

4.7 RSA Implementation

Optimum speed Modular multiplication is the core to design less time implementation of modular exponentiation.

The thesis selected right to left modular exponentiation algorithm(RL) and modified , faster modular multiplication methods with a key size (1024 bit) for design with three methods RSAM, RSAF, RSACRT.

4.7.1 RSA using Modified Interleaved Multiplication(RSAM)

As shown in figure 4.20 this model was designed with five states :

S0 : Start initialize the input data and make the counter of bits equal 0.

S1 : Initialization Modified multipliers.

S2 : If Modified multipliers finished initialization go to S3

else waiting.

S3 : Modified multipliers starts running .

S4 : check counter if finished declare done else go to S1.

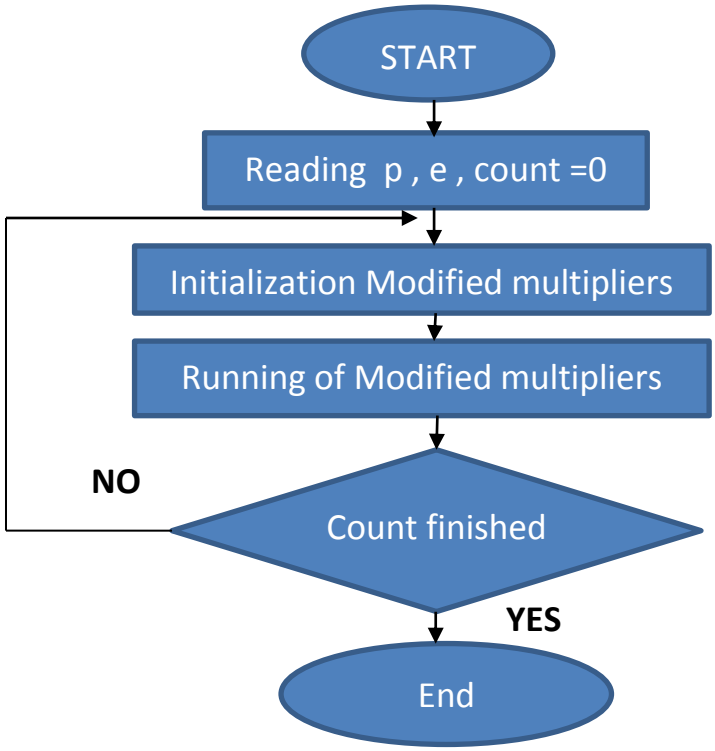


Figure 4.20 Flow Chart Of RSAM

Table 4.12 shows device utilization of RSAM encryption. The simulation result of RSAM encryption presented in Figure 4.21 where the values of keys were found (return to Appendix A "results of Java"), and the result was written to a text file(see Appendix B1 "Results of FPGA Implementation of RSAM Encryption-Decryption").

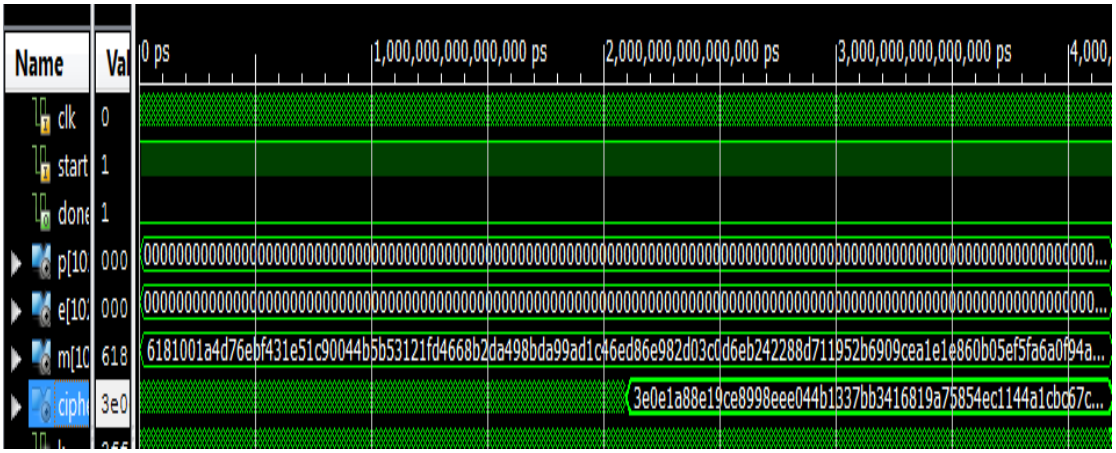


Figure 4.21 RSAM Encryption Timing

Table 4.12 Device Utilization Of RSAM Encryption

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	54	18224	0%
Number of Slice LUTs	81	9112	0%
Number of Fully LUT-FF Pairs	54	81	66%
Number of Bounded IOBs	3	232	1%
Number of BUFG / BUFGCTRL/BUFHCEs	1	16	6%

The simulation result of RSAM decryption presented in Figure 4.22. The message after decryption was the same original message (plaintext) .

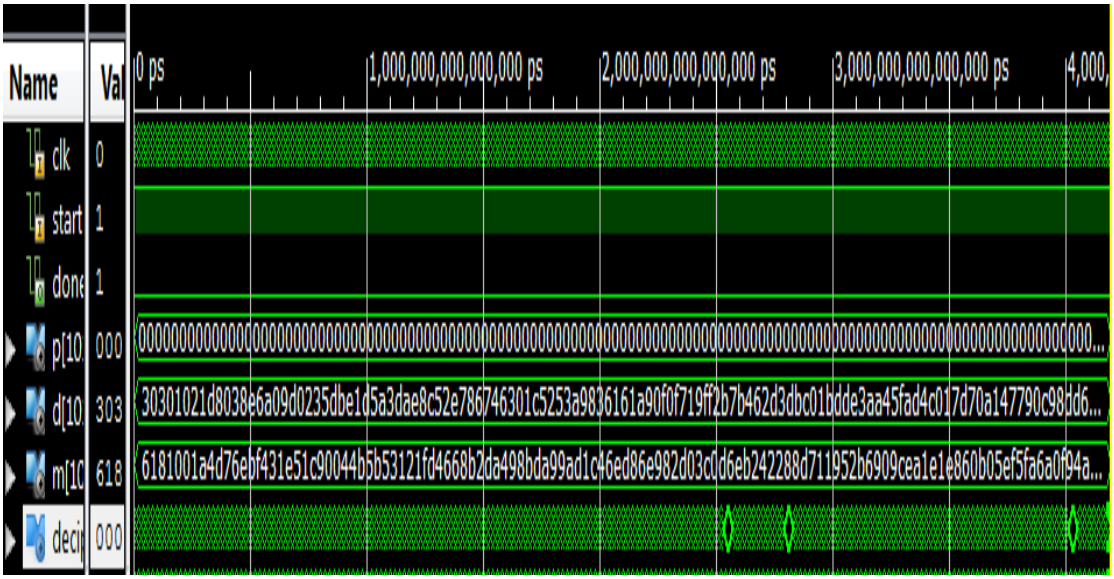


Figure 4.22 RSAM Decryption Timing

Table 4.13 shows encryption-decryption model utilization.

Figure 4.23 shows encryption-decryption timing, done4 signal equal 1 when the plaintext equal to plaintext after encryption and decryption process .

Table 4.13 Device Utilization Of RSAM Encryption-Decryption Timing

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	38053	18224	208%
Number of Slice LUTs	37078	9112	406%
Number of Fully LUT-FF Pairs	13616	81	22%
Number of Bounded IOBs	5	232	2%
Number of BUFG / BUFGCTRL/BUFHCEs	1	16	6%

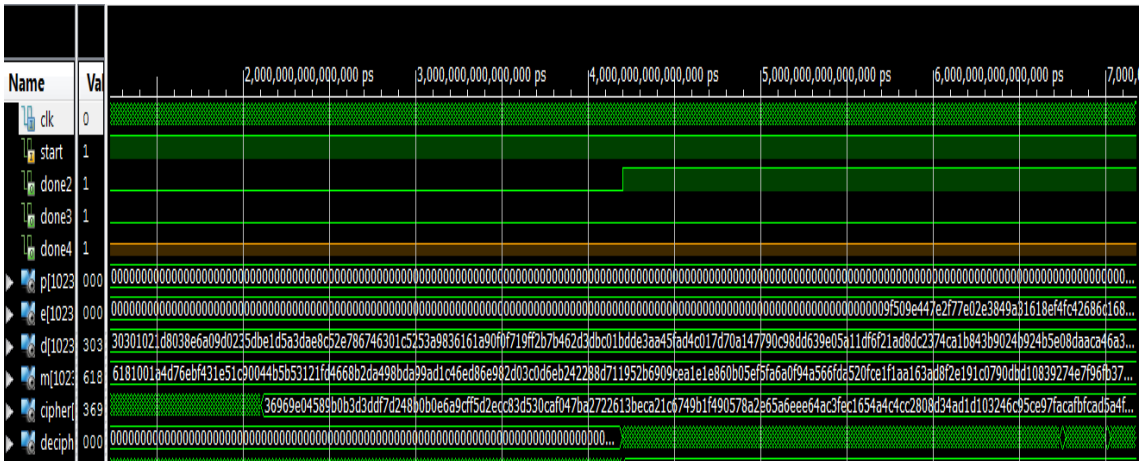


Figure 4.23 RSAM Encryption Decryption Timing

RSA Keys :

N modulus(m) is :
992518748866318450873020774167306610089790198341613603818386
972268353778600310209819239046044856790264874961073581459207
572837922417465472678920848325319112023900101328752644279653
347337759266362293440469743137660637860435380493366516138553
071227407983653189406443253773735497711469691732746198052780
32977923

Public key e is :
123941905330847381651165934046520193149044230227802628987058
364819854220953422125858384770560068571421892723122795681601
439016958529844577505711111138235723

private key d is :
503821427454265638388353584957392449662638167253293551602468
805769088972166827662772060618077345312519038816029360737100
057597057375633845288929049904761444319377066951413923779889
829040483134936994457305359971458943952212425361425220460590
800463118905659192577814773729233928352992543345173605756338
17154043

Plaintext (P) : 1229710510032979810011710811597116971l4

Cipher text after encryption(cipher)(C) :
383332420341710316819306994032099011529867085552572700
238616481649774498888340733900140561831159794196868527650999088406470105624701443041115653183294003731849516840942608761148303881948208845921361144023185972403009432046723448041010772210633926830233297781110461479829255091678219953256633632151032822603 09.

Plaintext after Decryption(decipher)(P) :
1229710510032979810011710811597116971l4

The speed up of RSAM can be calculated using equation :

$S = \frac{St}{Ht}$ ..... (4.4)

Where (S : Speed up, St : Software execution Time(Java time) , Ht : Hardware execution Time(FPGA time))

St = 29259186 ns = 29.259 ms (see Table 3.1)

By using equation (4.3) to find hardware execution time

$Ht = No \times Mp$ = 4199424 × 6.205 ns = 26.057 ms

$S = \frac{29.259\ \text{ms}}{26.057\ \text{ms}} = 1.122$

The throughput (number of modular multiplications achieved by RSAM per second) was calculated using following equation:

$Th = \frac{NMM}{\text{Ht}}$ ..... (4.4)

Where (Th : Throughput, NMM : number of modualr multiplications)

$Th = \frac{2048}{26.057 \times 10^{-3}}$ = 78596 MMPS(Modular Multiplication Per Second).

Table 4.14 shows Plaintext, Cipher text and decrypted Plaintext numeric values of RSAM

Table 4.14 RSAM Plaintext , cipher text and decrypted Plaintext

Plaintext P	cipher text C	Decrypted P
12297105100329798100117 1081159711697114	38333242034171031681930699403209901152986708555257270023861648164977449888834073390014056183115979419686852765099908840647010562470144304111565318329400373184951684094260876114830388194820884592136114402318597240300943204672344804101077221063392683023329778111046147982925509167821995325663363215103282260309	12297105100979810011 7108115971169714

4.7.2 RSA using Faster Montgomery Multiplication RSAF

Before Faster Montgomery multiplications RSAF began with mapping to Montgomery domain.

As shown in figure 4.24 this model was designed with five states :

S0 : Start initialize the input data and make the counter of bits equal 0.

S1 : Initialization Faster multipliers.

S2 : If Faster multipliers finished initialization go to S3
else waiting.

S3 : Faster multipliers starts running .

S4 : check counter if finished declare done else go to S1.

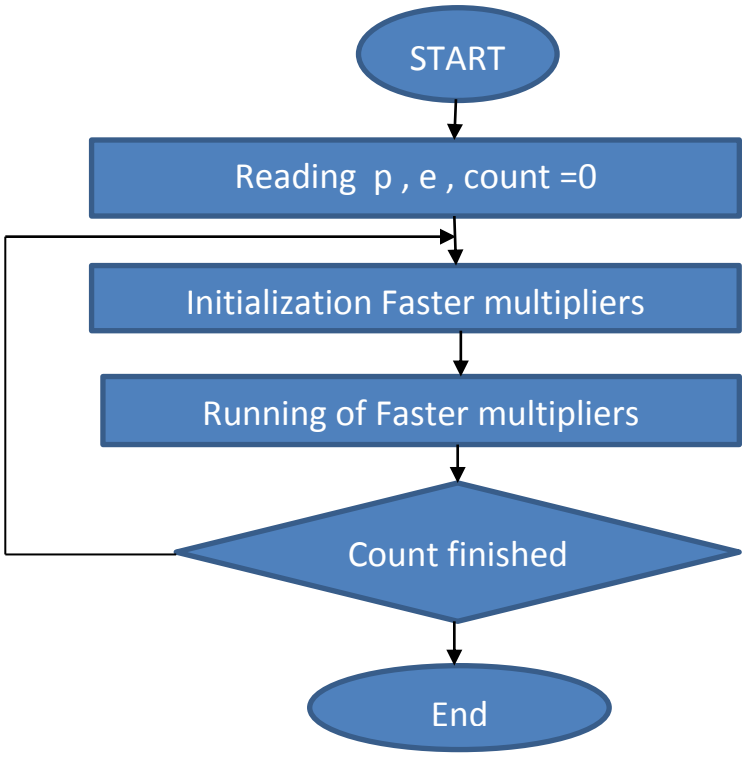


Figure 4.24 Flow Chart Of RSAF

Table 4.15 shows device utilization of RSAF encryption through which the simulation result of RSAF encryption presented in Figure 4.25 where the values of keys were found in Chapter Three (see Appendix A

“Results of Java “) and the result was written to a text file(see Appendix B2 “Results of FPGA Implementation of RSAF”), the results were written to a text file.

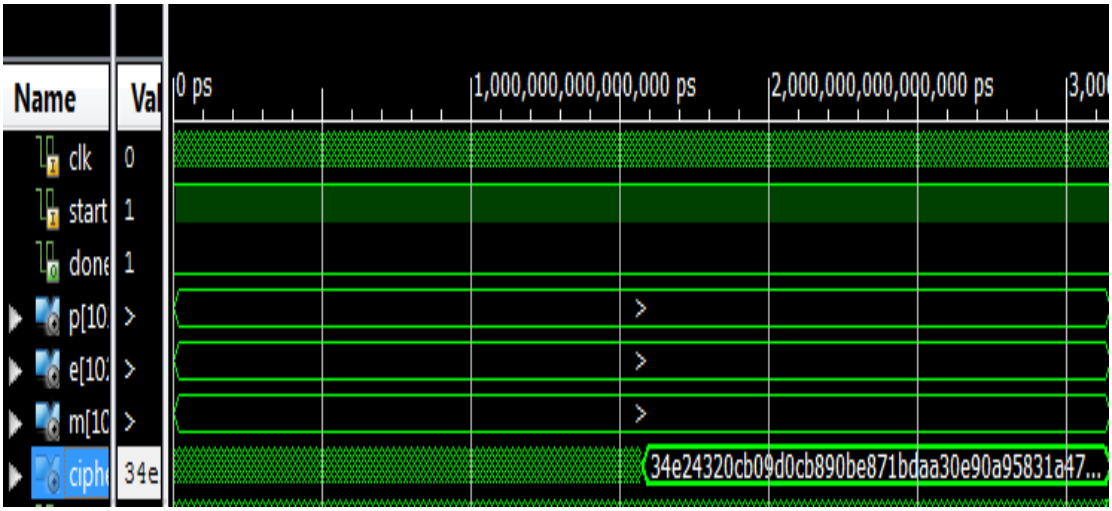


Figure 4.25 RSAF Timing

Table 4.15 Device Utilization Of RSAF

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	33	18224	0%
Number of Slice LUTs	68	9112	0%
Number of Fully LUT-FF Pairs	32	69	46%
Number of Bounded IOBs	3	232	1%
Number of BUFG / BUFGCTRL/BUFHCEs	1	16	6%

RSA Keys :

N modulus(m in model) is :
992518748866318450873020774167306610089790198341613603818386
972268353778600310209819239046044856790264874961073581459207
572837922417465472678920848325319112023900101328752644279653
347337759266362293440469743137660637860435380493366516138553
071227407983653189406443253773735497711469691732746198052780
32977923

Public key e is :
123941905330847381651165934046520193149044230227802628987058
364819854220953422125858384770560068571421892723122795681601
439016958529844577505711111138235723

private key d is :
503821427454265638388353584957392449662638167253293551602468
805769088972166827662772060618077345312519038816029360737100
057597057375633845288929049904761444319377066951413923779889
829040483134936994457305359971458943952212425361425220460590
800463118905659192577814773729233928352992543345173605756338

Plaintext (P) : 1229710510032979810011710811597116971l4

Plaintext after mapping(P) :
360922253456671810511623246362769139218348871649597l
624017930292337770686286895220534903405880387259242 8
480752504203069904193800176916066655328624873918913 9
638094036036329956076744279703121035286471515528691 1
046113889973560896829352695053280858824567909121347 0
665980231699590456437652158031077747478502908533

Cipher text mapping(C) :
428301684032396833510127067753053356987870546668871 8
904132456217918143707489759877873822768210884851469 2
659926209268916128149516981195394779958455138271621 8
040475270584698680046867899655725104395647896370770 8
438993008750439527907122318030890886525134767685781 2
454624108093891531065233923953604647762971520818

Cipher text After performing k Faster Montgomery Multiplications (R) :

371362931050136667468744410230983654064728825356458780263791787419580474704524800369386885833458064779830305390847914605444585504976634193358538836869634624501180593684552765706344228089513132814370491255728404826598635827278068246658792672174073521320516676 20

Cipher text after remapping from Montgomery domain (C) :

383332420341710316819306994032099011529867085552572700238616481649774498888340733900140561831159794196868527650999088406470105624701443041115653183294003731849516840942608761148303881948208845921361144023185972403009432046723448041010772210633926830233297781110461479829255091678219953256633632151032822603 09

The cipher text has the same result of Table 4.13 RSAM cipher text.

The speed up of RSAF calculated using equation (4.3) :

$S = \frac{St}{Ht}$

St = 29259186 ns = 29.259 ms (see Table 3.1)

find hardware execution time found using equation (4.3)

$Ht = N \times Mp$ = 3151872 × 4.473 ns = 14.098 ms

$S = \frac{29.259\ \text{ms}}{14.098\ \text{ms}}$ = 2.075

The throughput calculated using equation (4.4):

$Th = \frac{NMM}{\text{Ht}} = \frac{2048}{14.098 \times 10^{-3}}$ = 145268 MMPS.

4.7.3 RSA using Chinese Remainder Theorem (RSACRT)

This model is designed by RSAM modular exponentiation involved finding x1 and x2 of size 512 bits equations (2.1) and implemented RSA with 1024 bit. This model was designed As shown in figure 4.26. Table 4.16 shows device utilization of RSACRT encryption. The simulation result of RSACRT encryption presented in Figure 4.27 where the result was written to a text file(see Appendix B3 "Results of FPGA Implementation of RSACRT"), the result was written to a text file.

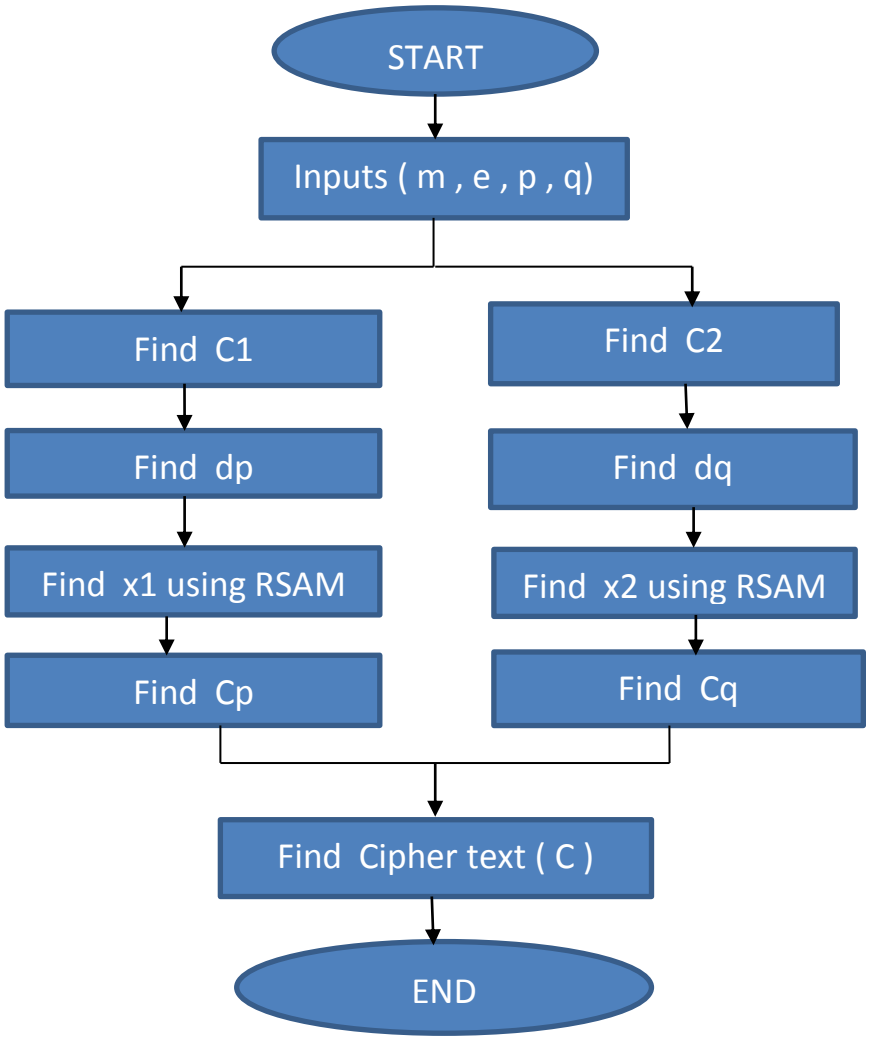


Figure 4.26 Flow Chart Of RSACRT

Table 4.16 Device Utilization Of RSACRT

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	1016	18224	0%
Number of Slice LUTs	149	9112	1%
Number of Fully LUT-FF Pairs	106	149	71%
Number of Bounded IOBs	4	232	1%
Number of BUFG / BUFGCTRL/BUFHCEs	1	16	6%

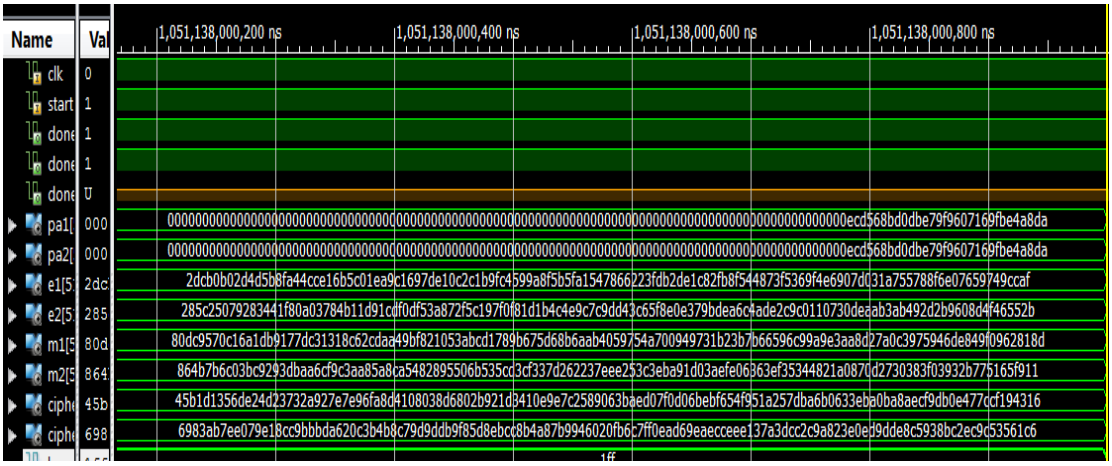


Figure 4.27 RSACRT Timing

RSA Keys :

p:
67490325138009718775720697886504877139214027635745264109796816698934947477571765284539470452754194534503940702407822732650626312478179008831563240845152133

q:
70335921166166025943271156350494076435672103374274886111525846518686016061523539548799111896539340034048006864159495167930736279861055864984428486206405293

public key (e) :
91474136945066696154315349674613524805129172186909808337140362098971674802651257254445686473821978581732818121074325431629107708160452698268666876581842513

Plaintext Message (m):
122971051003297981001171081159711697114

c1 = m mod p :
122971051003297981001171081159711697114
C2 = m mod q :
122971051003297981001171081159711697114

dp :
2 398 381 180 705 697 737 859 465 178 810 864 766 591 514 455 116 454 422 734 354 540 003 672 732 507 949 196 990 621 602 106 778 404 722 887 741 866 650 269 897 848 139 568 227 368 943 710 363 573 669 039
dq : 2 113 821 577 890 067 021 104 419 332 411 944 836 945 706 881 263 492 222 561 451 558 028 565 874 112 771 770 564 657 457 728 263 854 768 481 125 691 483 026 369 837 142 829 939 683 328 423 839 037 543 723

X1 calculated using 512 bit RSAM :
365020230754426262787362864924506074890907294805515556776999358085625704901342772576109391510097271266381278357839640895831788576808887917574063966344066 2
X1 calculated using 512 bit RSAM :
552623416060985965205991983127815573615677094741773243254512847868302954206539597852938083355770144841186571398031301719171399147009572206008082060210221 4

Cp :
361961978914384986252259054120187472796294560611843613434379014188779188685098064413872018879922849325123862260794612389402923808933906695599334815144858 7
Cq:
326135822057133420653292880807541982986744924119098984073527572336324279203204383069996150832583570530848205023115502899749491012750662477811150947359793 5

Cipher text (C) $= (q \times Cp \times x1 + p \times Cq \times x2\ ) MOD\ (p \times q)$ **:**

27 700 721 744 276 189 079 503 123 767 937 526 151 253 235
821 022 147 733 863 498 363 850 464 122 808 798 235 456 299
931 110 070 807 541 406 704 411 111 462 997 153 453 153 148
850 041 808 762 911 196 666 402 621 530 349 573 453 377 013
817 261 882 726 440 526 018 242 828 602 432 904 258 522 059
585 721 787 700 967 777 261 219 423 012 270 319 505 122 970

The speed up of RSACRT can be calculated using equation (4.3) :

S = $\frac{\text{St}}{Ht}$

St = 29259186 ns = 29.259 ms (see Table 3.1)

Hardware execution calculated using equation (4.3)

$Ht = N \times Mp$ = 1051136 × 6.205ns = 6.522 ms

S = $\frac{29.259\ \text{ms}}{6.522\ \text{ms}}$ = 4.486

Throughput was calculated using equation (4.4):

$Th = \frac{NMM}{\text{Ht}} = \frac{2048}{6.522 \times 10^{-3}}$ = 314014 MMPS(Modular Multiplication Per Second) .

4.7.4 RSA architects analysis

Return to Appendix B4“Implementation Pictures” to show pictures implementation design for all algorithms.

Table 4.17 presents execution time , throughput , speed up between RSA architectures. The results showed that RSACRT was the fastest method for implementation of RSA algorithm.

Table 4.17 Execution time , Throughput and Speed up For each RSA Architecture

RSA architecture	Ht(ms)	Th(MMPS)	S
RSAM	26.057	78596	1.12
RSAF	14.09	145268	2.075
RSACRT	6.52	314014	4.48

Chapter Five

Conclusions and Future Work

5.1 Conclusions

A. The implementations were done without division operation since the division in hardware uses big area and consume more time in execution .

B. When RSAM is used for RSA there is no need for mapping and remapping to and from to Montgomery domain.

C. Speed up of Modified interleaved multiplication compared to Interleaved multiplication is near (1.2) regardless of the number of bits.

D. The fastest implementation is RSACRT with 153 HZ operation clock and 1024 bit word.

5.2 Recommendations future works

1. Implementation of RSACRT system using RSAF instead of RSAM.
2. Implementation of RSA system using SoC chips such as ZYNQ family instead of Spartan6 can increase hardware speed up and decrease area of implementations of all algorithms.
3. Increase in key space from 1024 bit to 2048 bit or 4096 bit to increase the security of RSA system.
4. This work can be used as core for modular exponentiation which is the basic core of other cryptographic systems such as DH or ECC algorithms.

Refrences:

[1] William Stallings , "*cryptography and network security*" fifth edition principles and practice ,from M.I.T in computer science and a b.s. from Notre Dame in electrical engineering.

[2] Desi Wulansari, Much Aziz Muslim & Endang Sugiharti , "*Implementation of RSA Algorithm with Chinese Remainder Theorem for Modulus N 1024 Bit and 4096 Bit*", Department of Computer Science Faculty of Mathematics and Natural Science Semarang State University Semarang, 50229, Indonesia, 2017.

[3] Khaled Shehata, Hanady Hussien, Sara Yehia, "*FPGA Implementation of RSA Encryption Algorithm for E-Passport Application*" , World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering Vol:8, No:1, 2014.

[4] Ankit Anand , Pushkar Praveen , " *Implementation of RSA Algorithm on FPGA*", Centre for Development of Advanced Computing, (CDAC) Noida, India, Vol. 1 Issue 5, July - 2012.

[5] Gabriel Vasile Iana1, Petre Anghelescu1, Gheorghe Serban1,"*RSA encryption algorithm implemented on FPGA*", University of Pitesti , Department of Electronics and Computers, Romania, Arges, Pitesti, Str. Targul din Vale, No. 1, Code: 110040, 2011.

[6] Song Bo, Kensuke Kawakami, Koji Nakano, Yasuaki Ito, " *An RSA Encryption Hardware Algorithm using a Single DSP Block and a Single Block RAM on the FPGA*", Department of

Information Engineering School of Engineering, Hiroshima University 1-4-1 Kagamiyama, Higashi-Hiroshima, Hiroshima, 739-8527, JAPAN, 2011.

[7] Mostafizur Rahman, Iqbalur Rahman Rokon and Miftahur Rahman , "*Efficient Hardware Implementation of RSA Cryptography* " , Department of Electrical Engineering and Computer Science North South University Dhaka , Bangladesh ,2009.

[8] Koji Nakano, Kensuke Kawakami, and Koji Shigemoto, "*RSA Encryption and Decryption using the Redundant Number System on the FPGA*", Department of Information Engineering, Hiroshima University Kagamiyama 1-4-1, Higashi-Hiroshima, JAPAN, 2009.

[9] Ersin Öksüzoğlu, Erkay Savaş, "*Parametric, Secure and Compact Implementation of RSA on FPGA*", Sabanci University, Istanbul, TURKEY, 2008.

[10] Omar Nihouche', Mokhtar Nibouche, Ahmed Bouridane, and Ammar Belatreche, "*Fast Architectures For FPGA-Based Implementation of RSA Encryption Algorithm*", Faculty of Engineering, University of Ulster at Magee, Northland Rd, Deny, BT48 7JL, UK. Faculty of Computing, Engineering and Mathematical Sciences. University of the West of England Bristol BS16 IQY, UK. School of computer Science, Queen's University of Belfast, 18 Malone Rd, Belfast BT7 INN, UK, 2004.

[11] Alan Daly, William Marnane, "*Efficient Architectures for*

implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic", Dept. of Electrical and Electronic Engineering University College Cork Ireland, 2003.

[12] Cetin Kaya .C «*RSA Hardware Implementation*».

[13] V. Bunimov, M. Schimmler, "*Area-Time Optimal Modular Multiplication*", Embedded Cryptographic Hardware: Methodologie and Architectures,2004.

[14] P. L. Montgomery, "*Modular multiplication without trial division*",Math. Computation, vol. 44, pp.519 - 521, 1985.

[15] G. R. Blakley, "*A computer algorithm for the product AB modulo M*", IEEE Transactions on Computers, 32(5): pp 497 - 500, May 1983.

[16] K. R. Sloan, Jr. Comments on "*A computer algorithm for the product AB modulo M*", IEEE Transactions on Computers, 34(3): pp 290 - 292, March 1985.

[17] Lu, Y., Zhang, R., & Lin, D. "*New Partial Key Exposure Attacks on CRT-RSA with Large Public Exponents*." In International Conference on Applied Cryptography and Network Security (pp. 151-162), Springer International Publishing. June, 2014.

[18] Quisquater, J. J., & Couvreur, C. "*Fast Decipherment Algorithm for RSA Public-Key Cryptosystem*". Electronics Letters, 18(21), 905-907. 1982.

Appendix A

Results of Java

p:
12236041962475294070741124887750793539251606387026253488874935448075197804469962232199079975235402434916611294921117695090934302550586941987540609690795959

q:
81114362954140813456371349639798854400715000022835940352452697944988089899891021878459937871943653847338063965781679104659902275115809447077771449343861 97

N is :
99251874886631845087302077416730661008979019834161360381838697226835377860031020981923904604485679026487496107358145920757283792241746547267892084832531911202390010132875264427965334733775926636229344046974313766063786043538049336651613855307122740798365318940644325377373549771146969173274619805278032977923

Public key is :
12394190533084738165116593404652019314904423022780262898705836481985422095342212585838477056006857142189272312279568160143901695852984457750571111138235723

private key is :
50382142745426563838835358495739244966263816725329355160246880576908897216682766277206061807734531251903881602936073710005759705737563384528892904990476144431937706695141392377988982904048313493699445730535997145894395221242536142522046059080046311890565919257781477372923392835299254334517360575633817154043

Enter the plain text:

zaid abdulsatar

Encrypting String: zaid abdulsatar

String in Bytes: 122971051003297981001171081159711697114

Elapsed time of RSA encryption in nanoseconds: 29259186

Encrypted String in Bytes: 67-13198-63-391262-83-3011-798424102120-1213114-84-31-8014-18114189955-861154166-881051187218-12112649-7-14-41-1217711910438-4-11622-1-25-3820-6-9412982-1256455-116-422182-6028-47113-5335-116119-78-6192-56-6927-121-8382-17-5781-103-2432-3261-109126-84779-1149373-11294-37-6634912451-103125-67117-86372-8461127-10796-109-2436-20-106-22-105

Elapsed time of RSA decryption in nanoseconds: 19264425

Decrypted String in Bytes: 122971051003297981001171081159711697114

Decrypted String: zaid abdulsatar

BUILD SUCCESSFUL (total time: 13 seconds).

Appendix B1

Results of FPGA Implementation of RSAM Encryption_Decryption

p:
12236041962475294070741124887750793539251606387026253488874935448075197804469962232199079975235402434916611294921117695090934302550586941987540609690795959

q:
81114362954140813456371349639798854400715000022835940352452697944988089899891021878459937871943653847338063965781679104659902275115809447077771449343861 97

N is :
99251874886631845087302077416730661008979019834161360381838697226835377860031020981923904604485679026487496107358145920757283792241746547267892084832531911202390010132875264427965334733775926636229344046974313766063786043538049336651613855307122740798365318940644325377373549771146969173274619805278032977923

public key is :
12394190533084738165116593404652019314904423022780262898705836481985422095342212585838477056006857142189272312279568160143901695852984457750571111138235723

plaintext message = 122971051003297981001171081159711697114

Cipher text =

383332420341710316819306994032099011529867085552572700238616481649774498888340733900140561831159794196868527650999088406470105624701443041115653183294003731849516840942608761148303881948208845921361144023185972403009432046723448041010772210633 9

2683023329778111046147982925509167821995325663363215103282260309.

private key is :
503821427454265638388353584957392449662638167253293551602468805769088972166827662772060618077345312519038816029360737100057597057375633845288929049904761444319377066951413923779889829040483134936994457305359971458943952212425361425220460590800463118905659192577814773729233928352992543345173605756338171540 43

plaintext = 1229710510097981001171081159711697114.

Appendix B2

Results of FPGA Implementation of RSAF

p:
12236041962475294070741124887750793539251606387026253488874935448075197804469962232199079975235402434916611294921117695090934302550586941987540609690795959

q:
81114362954140813456371349639798854400715000022835940352452697944988089899891021878459937871943653847338063965781679104659902275115809447077771449343861 97

N is :
99251874886631845087302077416730661008979019834161360381838697226835377860031020981923904604485679026487496107358145920757283792241746547267892084832531911202390010132875264427965334733775926636229344046974313766063786043538049336651613855307122740798365318940644325377373549771146969173274619805278032977923

public key is :
12394190533084738165116593404652019314904423022780262898705836481985422095342212585838477056006857142189272312279568160143901695852984457750571111138235723

private key is :
50382142745426563838835358495739244966263816725329355160246880576908897216682766277206061807734531251903881602936073710005759705737563384528892904990476144431937706695141392377988982904048313493699445730535997145894395221242536142522046059080046311890565919257781477372923392835299254334517360575633817154043

let message(plaintext) = 1229710510032979810011710811597116971 14

plaintext after mapping process

P =
360922534566718105116232463627691392183488716495971624017930
292337770686286895220534903405880387259242848075250420306990419380017691606665532862487391891396380940360363299560767442797031210352864715155286911046113889973560896829352695053280858824567909121347066598023169959045643765215803107774747850290853 3 .

R =
428301684032396833510127067753053356987870546668871890413245621791814370748975987787382276821088485146926599262092689161281495169811953947799584551382716218040475270584698680046867899655725104395647896370770843899300875043952790712231803089088652513476768578124546241080938915310652339239536046477629715208 18.

The result after 1024 iterations of Faster Montgomery

R =
371362931050136667468744410230983654064728825356458780263791787419580474704524800369386885833458064779830305390847914605444585504976634193358538836869634624501180593684552765706344228089513132814370491255728404826598635827278068246658792672174073521320516676201883814073827804100951286362576201924236452030 98.

Remapping the result from Montgomery process

R =
383332420341710316819306994032099011529867085552572700238616481649774498888340733900140561831159794196868527650999088406470105624701443041115653183294003731849516840942608761148303881948208845921361144023185972403009432046723448041010772210633926830233297781110461479829255091678219953256633632151032822603 09.

Which is the same result in RSAM

Cipher text =
383332420341710316819306994032099011529867085552572700238616481649774498888340733900140561831159794196868527650999088406470105624701443041115653183294003731849516840942608761148303881948208845921361144023185972403009432046723448041010772210633926830233297781110461479829255091678219953256633632151032822603 09.

Appendix B3

Results of FPGA Implementation of RSACRT

p:
6749032513800971877572069788650487713921402763574526410979681
6698934947477571765284539470452754194534503940702407822732650
6263124781790088315632408451521 3

q:
7033592116616602594327115635049407643567210337427488611152584
6518686016061523539548799111896539340034048006864159495167930
7362798610558649844284862064052 9

N is :
4746994188385964794843636140340695029003788627352844935688070
4302296434427651770363195373416829002527394318460531164232763
8619691597216985105889529142678075928823110982620861799746398
7732310129645349307399634715053197713367600765937802373204986
3716420775604246379733942803978914124146968281636555586504867
677

public key is :
9147413694506669615431534967461352480512917218690980833714036
2098971674802651257254445686473821978581732818121074325431629
1077081604526982686668765818425 1

private key is :
3635168986198912336274330150182383896696641720891133579005992
8243077945551569210308155382544218382329784542779640722698986
1130597360356321141588976685679048691577351487859479218999158
2796291191335175116230864109763569994589662561977078536856931
8987431972017904462748111236835606559971076977515814384961362
43.

Plaintext Message (m) = 1229710510032979810011710811597116971 14.

c1 = m mod p

1229710510032979810011710811597116971 14 %
1223604196247529407074112488775079353925160638702625348887493
5448075197804469962232199079975235402434916611294921117695090
9343025505869419875406096907959 59 =
1229710510032979810011710811597116971 14.

c2 = m mod q =

1229710510032979810011710811597116971 14 %
8111436295414081345637134963979885440071500002283594035245269
7944988089899891021878459937871943653847338063965781679104659
9022751158094470777714493438619 7 =
1229710510032979810011710811597116971 14.

p-1 =
6749032513800971877572069788650487713921402763574526410979681
6698934947477571765284539470452754194534503940702407822732650
6263124781790088315632408451521 2.

q-1 =
7033592116616602594327115635049407643567210337427488611152584
6518686016061523539548799111896539340034048006864159495167930
7362798610558649844284862064052 8.

dp = e mod p -1 = 2 398 381 180 705 697 737 859 465 178 810 864 766
591 514 455 116 454 422 734 354 540 003 672 732 507 949 196 990 621
602 106 778 404 722 887 741 866 650 269 897 848 139 568 227 368 943
710 363 573 669 039.

dq = e mod (q-1) = 2 113 821 577 890 067 021 104 419 332 411 944 836
945 706 881 263 492 222 561 451 558 028 565 874 112 771 770 564 657
457 728 263 854 768 481 125 691 483 026 369 837 142 829 939 683 328
423 839 037 543 723.

X1 = $C1^{dp}$ mod p =

365020230754426262787362864924506074890907294805515556776999358085625704901342772576109391510097271266381278357839640895831788576808887917574063966344066 2.

X2 = C2dq mod q =
552623416060985965205991983127815573615677094741773243254512847868302954206539597852938083355770144841186571398031301719171399147009572206008082060210221 4

Cp = q -1 mod p =
361961978914384986252259054120187472796294560611843613434379014188779188685098064413872018879922849325123862260794612389402923808933906695599334815144858 7

Cq = p-1 mod q =
326135822057133420653292880807541982986744924119098984073527572336324279203204383069996150832583570530848205023115502899749491012750662477811150947359 7.

C = (q × Cp × x1 + p × Cq × x2) mod N =

((703359211661660259432711563504940764356721033742748861115258465186860160615235395487991118965393400340480068641594951679307362798610558649844284862064052 9 ×
361961978914384986252259054120187472796294560611843613434379014188779188685098064413872018879922849325123862260794612389402923808933906695599334815144858 7 ×
365020230754426262787362864924506074890907294805515556776999358085625704901342772576109391510097271266381278357839640895831788576808887917574063966344066 2) +
(674903251380097187757206978865048771392140276357452641097968166989349474775717652845394704527541945345039407024078227326506263124781790088315632408451521 3 ×
326135822057133420653292880807541982986744924119098984073527572336324279203204383069996150832583570530848205023115502899749491012750662477811150947359 7 ×
552623416060985965205991983127815573615677094741773243254512847

4786830295420653959785293808335577014484118657139803130171917
139914700957220600808206021022 14)) %
4746994188385964794843636140340695029003788627352844935688070
430229643442765177036319537341682900252739431846053116423276 3
861969159721698510588952914267807592882311098262086179974639 8
773231012964534930739963471505319771336760076593780237320498 6
371642077560424637973394280397891412414696828163655558650486 7
677

= 27 700 721 744 276 189 079 503 123 767 937 526 151 253 235 821 022 147 733 863 498 363 850 464 122 808 798 235 456 299 931 110 070 807 541 406 704 411 111 462 997 153 453 153 148 850 041 808 762 911 196 666 402 621 530 349 573 453 377 013 817 261 882 726 440 526 018 242 828 602 432 904 258 522 059 585 721 787 700 967 777 261 219 423 012 270 319 505 122 970 888 282 098 220 806 545 322 205 052 607 455 478 924.

The result tested by mathematical operations using calculator with large numbers

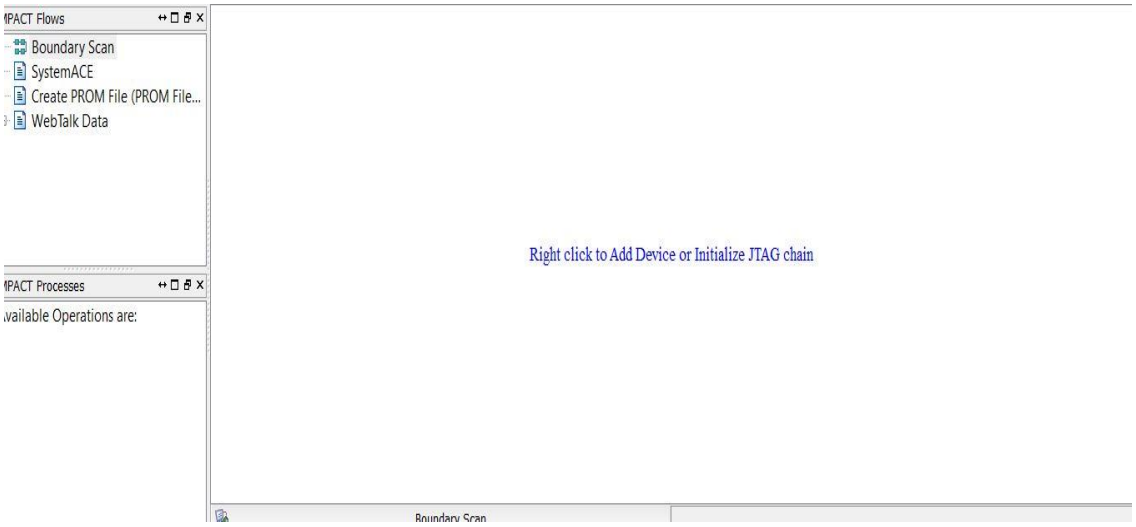
$N = p \times q$

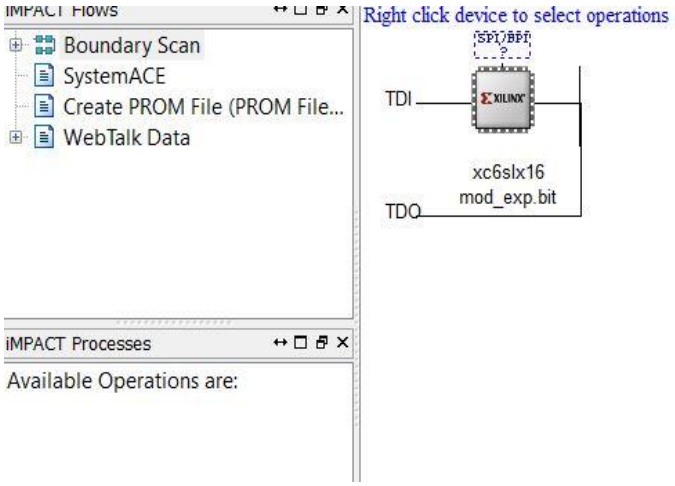
$C = message \text{ ^ } e \; mod \; N$

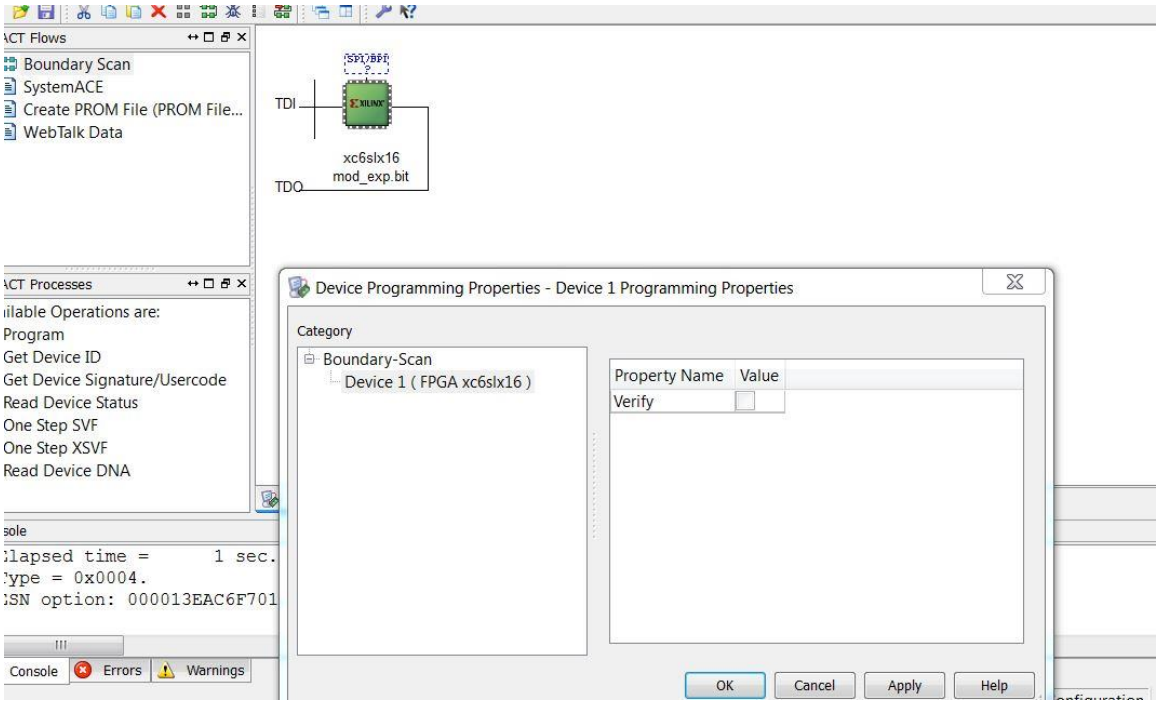
2770072174427618907950312376793752615125323582102214773386349 8363850464122808798235456299931110070807541406704411111462997 1534531531488500418087629111966664026215303495734533770138172 6188272644052601824282860243290425852205958572178770096777726 1219423012270319505122970888282098220806545322205052607455478 924.

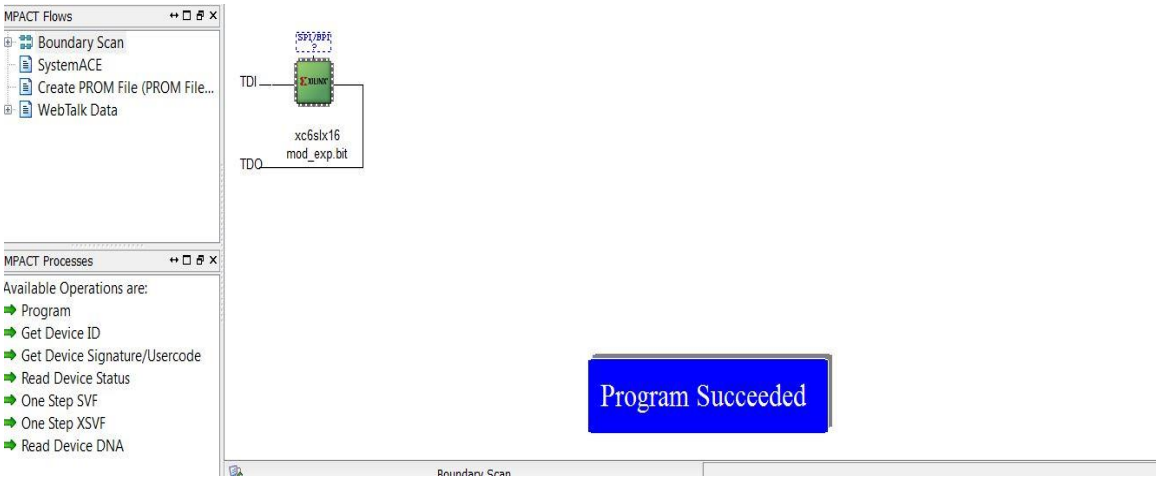
Appendix B4

Implementation Pictures all RSA algorithms






ACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File...
WebTalk Data
TDI
xc6slx16
mod_exp.bit
TDO
ACT Processes
ilable Operations are:
Program
Get Device ID
Get Device Signature/Usercode
Read Device Status
One Step SVF
One Step XSVF
Read Device DNA
Device Programming Properties - Device 1 Programming Properties
Category
Boundary-Scan
Device 1 (FPGA xc6slx16)
Property Name
Value
Verify
OK
Cancel
Apply
Help
sole
lapsed time = 1 sec.
ype = 0x0004.
SN option: 000013EAC6F701
Console
Errors
Warnings


MPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File...
WebTalk Data
TDI
xc6slx16
mod_exp.bit
TDO
MPACT Processes
Available Operations are:
Program
Get Device ID
Get Device Signature/Usercode
Read Device Status
One Step SVF
One Step XSVF
Read Device DNA
Program Succeeded

الخلاصة

RSAهو نظام التشفير الآمن للغاية لنقل البيانات ولكنه خوارزمية بطيئة نسبيًا اذا استخدمت في الوقت الحقيقي .

في هذه الأطروحة يتم تحليل وتصميم وتنفيذ الخوارزميات الرئيسية المستخدمة في RSA مع التركيز على وقت التنفيذ باستخدام لغة أعلى (JAVA) و FPGA مع مقارنة النتائج .

وقد كان وقت التنفيذ الذي تم الحصول عليه من أجل مضاعفات معيارية للـ FPGA التي تم تنفيذها في خوارزمية التعديل Interleaved 142.416 مايكرو ثانية وخوارزمية مونتغومري109.995 مايكرو ثانية وخوارزمية مونتغومري المسرعة 85.505 مايكرو ثانية والتعديل المسرعة 110.819مايكرو ثانية وهي الخوارزميات الأساسية المستخدمة في تنفيذ RSA .

فيما كان وقت التنفيذ باستخدام لغة جافا 29.259 مللي ثانية ، في FPGA باستخدام خوارزمية التعديل المسرعة ، مونتغمري المسرعة ونظرية الباقي الصينية كانت 26.057 مللي ثانية ، 14.098 مللي ثانية و 6.522 مللي ثانية على التوالي. تم الحصول على سرعة ملحوظة باستخدام FPGA بالمقارنة معJAVA

إقرار لجنة المناقشة

نشهد بأننا أعضاء لجنة التقويم والمناقشة قد اطلعنا على هذه الرسالة الموسومة **(تصميم خوارزمية ال RSA باستخدام منظومة البوابات القابلة للبرمجة)** وناقشنا الطالب **(زيد عبد الستار عبد الرزاق)** في محتوياتها وفيما له علاقة بها بتاريخ / /2018 وقد وجدناه جديراً بنيل شهادة الماجستير–علوم في اختصاص هندسة الحاسوب والمعلوماتية.

التوقيع:
رئيس اللجنة:
التاريخ: / /2018

التوقيع:
عضو اللجنة(المشرف): د.سعد داؤود الشماع
التاريخ: / /2018

التوقيع:
عضو اللجنة:
التاريخ: / /2018

التوقيع:
عضو اللجنة (المشرف):
التاريخ: / /2018

قرار مجلس الكلية

اجتمع مجلس كلية هندسة الالكترونيات بجلسته المنعقدة بتاريخ : / /2018

وقرر المجلس منح الطالب شهادة الماجستير علوم في اختصاص هندسة الحاسوب والمعلوماتية.

مقرر المجلس: **د.**
التاريخ: / /2018

رئيس مجلس الكلية:
التاريخ: / /2018

جامعة الموصل

كلية هندسة الالكترونيات

تصميم خوارزمية ال RSA باستخدام منظومة البوابات القابلة للبرمجة

رسالة تقدم بها

زيد عبد الستار عبد الرزاق

إلى

مجلس كلية هندسة الالكترونيات

جامعة الموصل

كجزء من متطلبات نيل شهادة الماجستير

في

هندسة الحاسوب والمعلوماتية

بإشراف

د. سعد داؤود الشماع

١٤٣٩ هـ

٢٠١٨م

جامعة الموصل
كلية هندسة الالكترونيات

تصميم خوارزمية ال RSA باستخدام منظومة البوابات القابلة للبرمجة

زيد عبد الستار عبد الرزاق

رسالة ماجستير
هندسة الحاسوب والمعلوماتية

بإشراف
د. سعد داؤود الشماع

١٤٣٩ هـ ٢٠١٨م