A HIGH-SPEED ASIC IMPLEMENTATION OF THE RSA CRYPTOSYSTEM

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

SONER YEŞİL

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2003

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. Mübeccel Demirekler
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality, as a thesis for the degree of Master of Science.

_____

Prof. Dr. Murat Aşkar
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Melek Yücel _____

Prof. Dr. Murat Aşkar _____

Prof. Dr. Rüyal Ergül _____

Assoc. Prof. Dr. Tayfun Akın _____

A. Neslin İsmailoğlu (M.S. in EE) _____

ABSTRACT

A HIGH-SPEED ASIC IMPLEMENTATION OF THE RSA CRYPTOSYSTEM

Yeşil, Soner

M.S., Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Murat Aşkar

September 2003, 96 pages

This thesis presents the ASIC implementation of the RSA algorithm, which is one of the most widely used Public Key Cryptosystems (PKC) in the world. In RSA Cryptosystem, modular exponentiation of large integers is used for both encryption and decryption processes. The security of the RSA increases as the number of the bits increase. However, as the numbers become larger (1024-bit or higher) the challenge is to provide architectures, which can be implemented in

hardware, operate at high clock speeds, use a minimum of resources and can be used in real-time applications.

In this thesis, a semi-custom VLSI implementation of the RSA Cryptosystem is performed for both 512-bit and 1024-bit processes using 0.35μm AMI Semiconductor Standard Cell Libraries. By suiting the design into a systolic and regular architecture, the broadcasting signals and routing delays are minimized in the implementation. With this regular architecture, the results of 3ns clock period (627Kbps) using 87K gates (8.7mm^2 with I/O pads) for the 512-bit implementation, and 4ns clock period (237Kps) using 132K gates (10.4mm^2 with I/O pads) for the 1024-bit implementation have been achieved. These results are obtained for the worst-case conditions and they include the post-layout routing delays. The design is also verified in real time using the Xilinx V2000E FPGA on the Celoxica RC1000 Hardware. The 1024-bit VLSI implementation has been sent to IMEC for fabrication as a prototype chip through Europractice Multi-Project Wafer (MPW) runs.

Keywords: PKC, RSA, Systolic Architecture, Montgomery Modular Multiplication, The Binary Method.

ÖZ

RSA KRİPTO SİSTEMİNİN YÜKSEK HIZLI TUMDEVRE UYGULAMASI

Yeşil, Soner

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Murat Aşkar

September 2003, 96 sayfa

Tez çalışmamızda, dünyada yaygın olarak kullanılan bir Açık Anahtar Kripto Sistemi olan RSA Algoritması'nın Uygulamaya Özel Tümdevre Gerçekleştirmesi sunulmaktadır. RSA Kripto Sistemi'nde, şifreleme ve deşifreleme işlemleri için, çok büyük tamsayıların kullanıldığı (1024-bit veya daha fazla) modüler üs alma matematiksel işlemi kullanılmaktadır. Kullanılan tamsayıların bit uzunluğu arttıkça, RSA Kripto Sisteminin güvenliği de artmaktadır. Öte yandan, sayıların büyümesiyle birlikte, donanıma uygun, hızlı çalışabilen, mümkün olan en

az seviyede özkaynak içeren ve gerçek zamanlı uygulamalarda kullanılabilecek mimariler tasarlamak önem kazanmaktadır.

Bu tez içersinde, 0.35μm AMI Semiconductors Standart Hücre Kütüphanesi kullanılarak gerçekleştirilen 512-bit ve 1024-bit RSA işlemlerinin yarı özel tasarımları yer almaktadır. Birbirine özdeş ve çok sayıda yapının birbiri ardına sıralanmasıyla (sistolik yapı) olusturulan bir mimarinin tasarımda kullanılmasıyla, tümdevrenin bütününe yayılan sinyallerin sayı ve uzunlukları en az sayıya indirgenmiştir. Bu düzenli yapı sonucunda, 512-bit uygulamada 3ns saat hızı (627 Kbps) ve 87 bin kapı değerinde bir alana ($8.7mm^2$ giriş/çıkış bağlantılarıyla birlikte), 1024-bit uygulamada ise 4ns saat hızı (237 Kbps) ve 132 bin kapı değerinde bir alana ($10.4mm^2$ giriş/çıkış bağlantılarıyla birlikte) ulaşılmıştır. Bu sonuçlar, en kötü koşullar öne sürülerek ve tümdevre içersindeki yol atama gecikmeleri dikkate alınarak elde edilen sonuçlardır. Tümdevre gerçekleştirmenin yanısıra, Celoxica RC1000 Donanımı ve bu donanım üzerinde yer alan Xilinx V2000E FPGA kullanarak, söz konusu tasarımın gerçek zamanlı doğrulanması da yapılmıştır. 1024-bit RSA tümdevre tasarımı, Europractice MPW (Çoklu Tümdevre Üretim Programı) dahilinde, bir prototip tümdevre olarak üretilmek amacıyla IMEC firmasına gönderilmiştir.

Anahtar Kelimeler: RSA, Açık Anahtar Kripto Sistemi, Montgomery Moduler Çarpma, Sistolik Yapılar.

ACKNOWLEDGMENTS

I am very grateful to Assist. Prof. Dr. Y. Çağatay Tekmen for his endless support and encouragement at all stages of my thesis. I would also like to express my appreciation to him because of his valuable suggestions, guidance and experience in solving problems at the critical stages of the thesis, where I gave way to despair.

I would like to express my acknowledgments to my supervisor Prof. Dr. Murat Aşkar for his inspiration of my thesis subject by initiating the RSA Project at TÜBİTAK-ODTÜ-BİLTEN.

I would also like to thank to Assoc. Prof. Dr. Melek Yücel for her contributions on the RSA Project, and her valuable suggestions and interest in the development of this thesis.

Special thanks to TÜBİTAK-ODTÜ-BİLTEN for facilities provided for the completion of this thesis. I would like to thank to my colleagues here, especially to my coordinator Neslin İsmailoğlu for her comprehension in sharing her deep experience in VLSI design and Oğuz Benderli for his endless interest and support throughout my thesis. I am also very grateful to my colleague and sincere friend Refik Sever for his valuable contributions to my study.

Finally, I would like to express my deep gratitude to my dear wife Sezen, for her patience and continuous support, and my dear family for their love and encouragement throughout this thesis.

To my wife Sezen…

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

LIST OF ABBREVIATIONS

PKC	Public Key Cryptosystem
RSA	Rivest Shamir Adleman
ASIC	Application Specific Integrated Circuit
BİLTEN	Bilgi Teknolojileri ve Elektronik Araştırma Enstitüsü
S&M	Squaring and Multiplication
FPGA	Field Programmable Gate Array
VLSI	Very Large Scale Integrated Circuit
R-L	Right-to-Left
L-R	Left-to-Right
CLB	Configurable Logic Block
FF	Flip-Flop
IC	Integrated Circuit
I/O	Input/Output
LSB	Least Significant Bit
LUT	Look-up Table
MSB	Most Significan Bit

PE	Processing Element
RAM	Random Access Memory
TÜBİTAK	Türkiye Bilimsel ve Teknik Araştırma Kurumu
IMEC	Interuniversity MicroElectronics Center
AMIS	Automotive Medical Industrial Semiconductor
MPW	Multi-Project Wafer
CMOS	Complementary Metal-Oxide Semiconductor
FA	Full Adder
HDL	Hardware Description Language
DRC	Design Rule Check
ERC	Electrical Rule Check
LVS	Layout Versus Schematic

CHAPTER 1

INTRODUCTION

In the last quarter of the 20th century, especially in the 90's, the field of cryptography has faced a new problem beyond privacy, which had been the main goal until that time. With the widespread popularity of electronic communication all-over the world, difficulties in the key distribution, key management and authentication began to rise and researchers focused on these problems without any concession of the traditional objective, security.

"We stand today on the brink of a revolution in cryptography", said Diffie and Hellman in 1976 [1] as the beginning sentence of their paper in which the concept of "Public-Key Cryptosystem" (PKC) was born. After 2 years in 1978, an elegant implementation of the public-key cryptosystem came from Rivest, Shamir and Adleman, named as the RSA Public-Key Cryptosystem [2]. Today, because of the high-security provided, RSA is still known as the most widely used public-key cryptosystem in the world.

Although providing high security, currently available RSA hardware needs to be improved on the speed and area issues. The security of the RSA increases as the number of bits in the algorithm increase. However, high number of bits end up

with slower architectures and increased area. The challenge is to provide fast architectures and efficiently used resources as the number of bits increase.

This chapter presents an introduction to the cryptography and Public-Key Cryptosystems. The first section gives some basics of cryptography and continues with the needs of today's cryptography. Section 1.2 describes the concept of PKC and some facilities provided by this cryptosystem such as authentication, integrity, and non-repudiation. The last section of this chapter gives a brief discussion of the scope of the research in this thesis. Also the thesis organization and chapter summaries are given in this section.

1.1. Basics of Cryptography

Cryptography is basically the art and science of enabling two people to communicate over an insecure channel in such a way that an unintended recipient cannot understand what is being said.

A message is a plaintext usually denoted by *M* and represented by a binary data in digital applications. Encryption is the processing of the message into a form that is virtually impossible to understand without the key. An encrypted message is cipher text, denoted as *C***.** The process of recovering the original message from the encrypted data is called decryption. It is the inverse function of the encryption.

The following formal definition of the concept of cryptography is taken from the text-book named "Cryptography: Theory and Practice," which is written by D. Stinson [5].

A cryptosystem is a five-tuple (P, C, K, E, D), where the following conditions are satisfied:

1. P *is a finite set of possible plaintexts*
2. C is a finite set of possible cipher texts
3. K, the key space, is a finite set of possible keys

4. For each $k \in K$, there is an encryption rule $e_k \in E$ and a corresponding decryption rule $d_k \in D$. Each $e_k : P \to C$ and $d_k : C \to P$ are functions such that $d_k(e_k(x)) = x$ for every plaintext $x \in P$.

Although the traditional private-key cryptosystems satisfy the privacy, which has been the fundamental goal of cryptography, they are inadequate to overcome the recently arising secure communication problems such as:

i) Key distribution problem through a secure channel

ii) Key management problem

iii) And the following security problems:

— **Authentication**: The sender of the message should be able to sign it in such a way that an intruder cannot forge the signature.

— **Integrity**: The intended recipient of the encrypted message should make sure that an intruder has not modified the message.

— **Non-repudiation**: The owner of a signed message should not be able to gainsay his/her signature.

A new concept of *Public-Key Cryptosystem* was invented by Diffie and Hellman in 1976 [1] to overcome these problems. The next section gives detailed information on how these problems can be solved with this new cryptosystem.

1.2. Public-Key Cryptosystems (PKC)

In a Public-key Cryptosystem, the encryption procedure of each user (*E*) is publicly revealed but the decryption procedure (*D*) is only private to that user. The enciphering and deciphering can be shown as follows:

Enciphering: $C = E(M)$ (Public procedure)

Deciphering: $M = D(C)$ (Private procedure)

Where M is the message and C is the cipher text.

If the system has the properties,

a) $M = D(E(M))$

b) $E(\cdot)$ is publicly revealed and easy to compute for everyone but $D(\cdot)$ is impractical to be computed except for its owner who has the key

then $E(\cdot)$ is called trap-door one-way function because it is easy to implement in one way but very difficult in the other way. However, if one obtains the necessary key, $D(\cdot)$ is as easy as $E(\cdot)$. This is the reason that it is called "trap-door". In addition to the above 2 properties, a third property of

c) $M = D(E(M)) = E(D(M))$

gives a new name to the system as "trap-door one-way permutation". The result is that every message is the cipher text for some other message and every cipher text can be used as a message.

The fundamental idea behind a Public-Key Cryptosystem is that it is computationally impractical to determine $D(\cdot)$ given $E(\cdot)$, so that the encryption rule $E(\cdot)$ can be made public. Using the advantage of PKC, one can send an encrypted message to another (without the prior communication of a secret key) by using the public encryption rule $E(\cdot)$. Only the intended recipient can decrypt the cipher text, using his/her secret decryption rule $D(\cdot)$.

Another consequence of Public-Key Cryptosystem is the *signature* facility obtained by processing the message by private procedure (*D(M)*) prior to the public one. Anyone can verify this signature by using the public procedure of the sender, *E(D(M)).* Because the signature is a private procedure, it cannot be forged and also the sender cannot deny the signature.

1.1. The Scope of the Research and Thesis Outline

The RSA PKC uses modular exponentiation operation both for encryption and decryption. The security of the RSA increases as the numbers in the modular exponentiation are increased. However, this increase corresponds to larger and slower architectures. Since the demand for higher levels of security is increasing day by day, it becomes important to find the ways of implementing the RSA PKC in more efficient and faster architectures. Within this scope of the research, this thesis describes a hardware implementation of the RSA PKC, which uses a linear systolic architecture operating at high clock frequencies with a minimum of resources.

A mathematical background including the main algorithms used in the RSA algorithm is given in Chapter 2. In addition to the mathematics, this chapter also gives a brief discussion on the implementation properties of these algorithms. Chapter 3 presents a literature review of the recent RSA implementations. In this chapter, the methods and improvements on the RSA algorithm are categorized and described first, and then the implementation results are presented. Chapter 4 begins with the theoretical study followed at the early stages of the thesis, then gives the methods and algorithms chosen, and finally presents the implementation details. A comparison of the implementation results to the previous implementations is also given at the end of this chapter. The last chapter is a conclusion part, in which the underlying reasons of the achieved results are evaluated and the ways of improving this study are discussed. The appendix part includes the structure of the CLBs and LUTs in the XILINX Virtex FPGAs, general features of the Celoxica RC1000 Hardware, and general characteristics of the AMI Semiconductor Standard Cell Libraries.

CHAPTER 2

A MATHEMATICAL BACKGROUND OF THE RSA PKC

RSA is a cryptographic algorithm, which provides high security. Modular exponentiation of long integers is the main mathematical operation of the RSA used in both encryption and decryption. The security of this algorithm is based on the factorisation problem of long integers.

This chapter describes the underlying mathematics of the RSA PKC. The first section explains how the modular exponentiation can used for enciphering and deciphering purposes. An example of a secure data transmission using RSA algorithm is also presented in the first section. The other sections include some algorithms commonly used in the implementation of this cryptosystem. Finally the chapter ends by giving some information on the Chinese Remainder Theorem and its application to RSA.

2.1. The RSA Algorithm

The *RSA Cryptosystem* uses computations in Z_N where N is the product of two distinct very large primes p and q. The message *M* is represented as a number between 0 and $N-1$, and relatively prime to N. The encryption $E(\cdot)$ and decryption $D(\cdot)$ procedures are defined as

$$C = E(M) = M^e \pmod{N}.$$
$$M = D(C) = C^d \pmod{N}.$$

where M is the message and C is the cipher text, e is the public key and d is the private key. Modular exponentiation operation is used for both encryption and decryption processes. The relation between the keys is as follows:

$$N = p \cdot q$$
$$e \cdot d \equiv 1 \pmod{\phi(N)}$$

where $\phi(N)$ is the Euler totient function, which equals to the number of positive integers less than N, which are relatively prime to N. Since N is the product of two primes p and q,

$$\phi(N) = \phi(p) \cdot \phi(q)$$

and

$$\phi(p) = (p-1),$$
$$\phi(q) = (q-1).$$

then $\phi(N)$ becomes

$$\phi(N) = (p-1) \cdot (q-1)$$

As seen from the above equations, in order to obtain the private key, d, from the public key, e, one should factorize the public modulus N into its prime factors p and q. Because of this reason the security of the RSA cryptosystem lies in the factorization of large integers (e.g., > 1024-bit numbers for the modulus N), which is computationally infeasible with today's technology.

One can verify that the encryption and decryption procedures are inverse operations as follows:

$$e \cdot d \equiv 1 \pmod{\phi(N)}.$$
$$e \cdot d = t \cdot \phi(N) + 1., \quad \text{where } t \text{ is any positive integer}$$
$$(M^e)^d \equiv M^{(t\cdot\phi(N)+1)} \pmod{N}$$
$$\equiv (M^{\phi(N)})^t \cdot M \pmod{N}$$
$$\equiv 1^t \cdot M \pmod{N}$$
$$\equiv M \pmod{N}$$

where Euler's theorem is used for the equation:

$$M^{\phi(N)} \equiv 1 \pmod{N}$$

To perform the RSA,

1) Compute N as the product of two large random primes p and q.

2) Pick a public key e in the range $1 < e < \phi(N)$ and relatively prime to $\phi(N)$. Usually the public key is selected as a small number such as $2^{16}+1$ in order to make the computation of the encryption fast.

3) Compute $d = e^{-1} \pmod{\phi(N)}$ using the Extended Euclidian Algorithm.

4) The public key is the pair of positive integers (e, N) and private key is d with encryption and decryption procedures as follows:

$$C \equiv M^e \pmod{N}$$
$$M \equiv C^d \pmod{N}$$

5) Each user in the system will have different public key pairs and private keys:

$$Alice \rightarrow \quad (N_a, e_a, d_a)$$
$$Bob \rightarrow (N_b, e_b, d_b)$$
$$Oscar \rightarrow (N_o, e_o, d_o)$$
$$\vdots$$

6) In order to send a signed message to Bob, Alice performs the following operation:

i) $S = M^{d_a} \pmod{N_a}$ (Alice signs the message with her private key)

ii) $C = S^{e_b} \pmod{N_b}$ (She then encrypts the signed message with Bob's public key)

7) In order to decrypt and authenticate the encrypted message sent by Alice, Bob does the following operation:

i) $S = C^{d_b} \pmod{N_b}$ (Bob decrypts the signed message by his private key)

ii) $M = S^{e_a} \pmod{N_a}$ (Then he recovers the message by using Alice's public key)

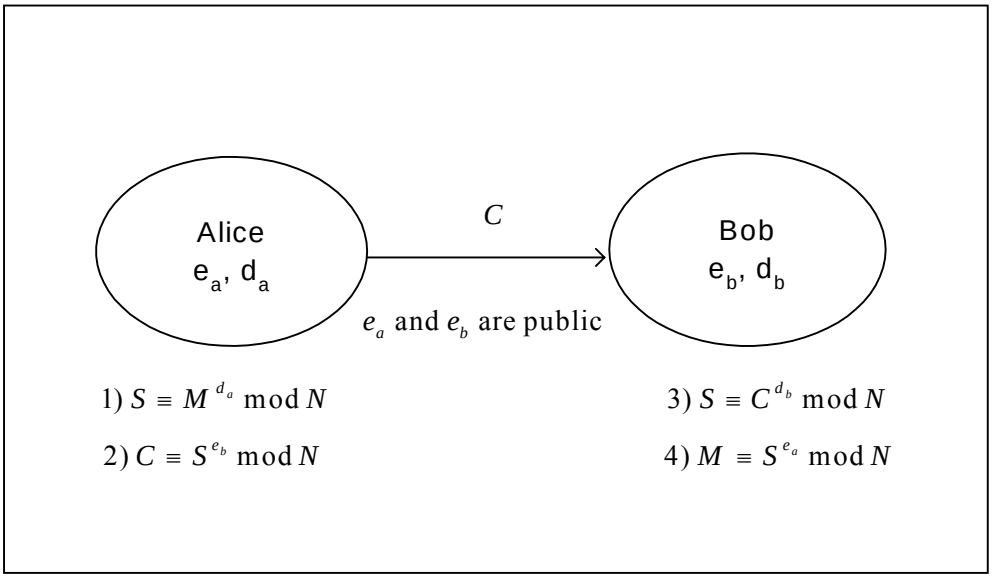


Figure 2.1: An example of sending a signed message in RSA PKC. Alice sends only the cipher text. There is no need to transfer the key.

The trap-door one-way permutation for the *RSA Public-Key Cryptosystem* is the *modular exponentiation* operation, which can be performed as a series of *modular multiplication* operations. The modular exponentiation is also very commonly used for the other algorithms different from RSA cryptosystem such as Diffie and Hellman key exchange scheme [1], ElGamal Signature scheme [6], and Digital Signature Standard (DSS) [7]. Although it is a simple and widely used mathematical operation, modular exponentiation has an important drawback of being time-consuming especially when the integers are large (>1024-bits). Enormous number of research, both in theoretical and electronics, is still going on to improve the performance of the modular exponentiation operation while decreasing the resource usage.

The following sections give brief discussions on various algorithms that are used in the modular exponentiation and multiplication operations. The *Binary Method* for the exponentiation and *Montgomery's Algorithm* for the modular multiplication will be mainly discussed, since these are the most widely used and most efficient algorithms. The reader can find very clear explanations and informing examples about some other algorithms in the technical report of RSA Laboratories written by Çetin Kaya Koç [4].

2.2. Modular Exponentiation

The modular exponentiation operation is performed as a series of modular multiplications. Hence, the performance of the modular exponentiation depends mainly on the following two criteria:

1) The number of the modular multiplications in the modular exponentiation algorithm.

2) The stand-alone performance and physical area of the modular multiplication module. (The multiplication unit consumes most of the silicon area in an RSA implementation.)

Therefore, one should choose proper modular exponentiation and multiplication algorithms suiting to each other so that the $time \times area$ product can settle-down to an optimum value. That is, the design implemented on a system should be fast enough to satisfy the time needs, on the other hand it should be as small as possible in order not to cause placement and cost problems.

2.2.1. The Binary Method (Square and Multiply Method)

This method examines the exponent in bit wise fashion either from left-to-right (L-R Binary Method) or right-to-left (R-L Binary Method). These two algorithms are as follows:

Let k be the number of bits :

$$e = (e_{k-1}e_{k-2}\cdots e_1e_0) = \sum_{i=0}^{k-1} e_i 2^i, \qquad \text{where } e_i \in \{0,1\} \text{ and } e_{k-1} = 1.$$

a) The L-R Binary Method

Inputs: M, e, N

Output: $C = M^e \pmod N$.

1. **if** $(e_{k-1} = 1)$ **then** $C = M$ **else** $C = 1$.
2. **for** $i = k-2$ **downto** 0
   - **2.a.** $C = C \cdot C \pmod N$
   - **2.b. if** $e_i = 1$ **then** $C = C \cdot M \pmod N$
3. **return** C

Figure 2.2:The L-R Binary Method

The squaring (step-2.a.) operation is performed at each step but the multiplication operation (step-2.b.) is performed if the corresponding bit of the exponent is equal to 1. Therefore the total number of multiplications needed for the *LR Binary Method* is $(k-1)+H(e)-1$, where $H(e)$ is the Hamming Weight of the

exponent, which equals to the number of 1's in the exponent. Assuming the equal probability of 1's and 0's in the exponent, the average number of multiplications for this algorithm is $\frac{3}{2}(k-1)$.

b) The R-L Binary Method

Inputs: M, e, N

Output: $C = M^e \pmod N$.

1. $X_0 = M$; $C_0 = 1$;
2. **for** $i = 0$ **to** $k - 1$
   - 2.a. $X_{i+1} = X_i \cdot X_i \pmod N$
   - 2.b. **if** $(e_i = 1)$ **then** $C_{i+1} = C_i \cdot X_i \pmod N$
   - 2.c. **else** $C_{i+1} = C_i$.
3. **return** C

Figure 2.3: The R-L Binary Method

Although it has the same principle as *The L-R Binary Method*, this algorithm has several advantages for fast exponentiation implementations. The main advantage is that the steps of 2.a. and 2.b. in the above algorithm are independent from each other and can be performed in parallel. This reduces the number of iterations directly down to the number of squaring operations, which is fixed whatever the exponentiation algorithm is used. The parallelism can be achieved either by using extra hardware or within the same hardware by using the idle clocks of the system as in the proposed implementation in this thesis. One drawback of this algorithm compared to the L-R method is that the location of the most significant 1 in the exponent should be detected in order to prevent unnecessary squaring operations. However, detection of the most significant 1 is a simple procedure, which can be done, for example, with a 10-bit counter for the 1024-bit exponent. An example of R-L Binary Method, in which the squaring and multiplication operations are performed in parallel, is demonstrated in Figure 2.8.

2.2.2. The *m*-ary Method

The *m*-ary method is a generalization of the binary method. Differently, this method involves treating the *m*-bits of the exponent instead of 1-bit. Usually, *m* is chosen to be a power of 2. The main advantage of this method is, the number of multiplications needed for the modular exponentiation decreases as *m* increases. However, more complex structures and preprocessing are needed since the powers of the message up to *m* $(M, M^2, \cdots, M^{m-1})$ are to be calculated and stored beforehand. Table 2.1, which is taken from the technical report written by Çetin Kaya Koç [4], summarizes the comparison of the *m*-ary and the binary methods according to the required number of multiplications.

Table 2.1: Comparison of m-ary Method to Binary Method.

k	*binary*	*m-ary*	*Opt.r* ($m = 2^r$)	*Savings %*
8	11	10	2	9.1
16	23	21	2	8.6
32	47	43	2, 3	8.5
64	95	85	3	10.5
128	191	167	3, 4	12.6
256	383	325	4	15.1
512	767	635	5	17.2
1024	1535	1246	5	18.8
2048	3071	2439	6	20.6

There are also many other algorithms for modular exponentiation, which are based on reducing the number of modular multiplications and thus improving the exponentiation performance [4]. Since the number of squaring operations cannot be reduced in either of the algorithms, to perform the multiplying and squaring operations in parallel gives the best performance for the exponentiation operation. Therefore in this thesis, *The R-L Binary Method* is chosen to be the exponentiation algorithm, in which the parallelism can be exploited without any need of extra hardware and clocks.

2.3. Modular Multiplication

As mentioned in the previous sections, modular multiplication is the kernel operation of RSA cryptosystem. Any improvement in the modular multiplication operation directly affects the performance of the RSA.

P.L. Montgomery found an ingenious way of computing modular multiplication operation in 1985 [8]. Different from previous methods, the Montgomery Multiplication Algorithm uses residue representations of the numbers that are to be multiplied. This representation provides division by r, where r is a power of 2. Therefore, it is quite suitable for implementing on digital signal processors, general-purpose microprocessors and digital VLSI structures. The next section gives the underlying mathematics of Montgomery's approach on the modular multiplication.

2.3.1. Montgomery's Method

This method is very commonly used to speed up the modular multiplication and squaring operations required during the exponentiation process. In a modular multiplication, with modulo N being k-bit number, each number is represented by a radix r, which is usually a power of 2. The Montgomery Multiplication Algorithm requires that r and N are relatively prime to each other, i.e. $\gcd(r, N) = 1$. Since N is odd (multiplication of two primes) and r is a power of 2, this condition is inherently satisfied. Given two integers $A < N$ and $B < N$, this method computes $A \cdot B \cdot r^{-m} \bmod N$, where m is the number of the digits of N in radix-m. With a representation of any number X of the form

$$X = \sum_{i=0}^{m-1} x_i r^i, \quad \text{where } x_i \in \{0,1,\cdots,r-1\},$$

Montgomery Modular Multiplication Algorithm is as follows:

Inputs: A,B,N.

Output: $P = A \cdot B \cdot r^{-m} \bmod N$

$P_0 = 0$.

For $i = 0$ **to** $m-1$

a. $q_i = (p_0 + a_i \cdot b_0) \cdot (r - n_0)^{-1} \bmod r$;

b. $P_{i+1} = (P_i + a_i \cdot B + q_i \cdot N)/r$;

End

Post Condition: $r^m \cdot P = Q \cdot N + A \cdot B$

Figure 2.4: Montgomery's Algorithm.

In the above algorithm, computation of q_i in step-a guarantees that the value of P in step-b is divisible by r. Since r is a power of 2, division by r is just shifting out the least significant zero digit in radix-r. This property of the algorithm makes it very easy to implement in digital design. However, one should remove the r^{-m} factor at the output in order to get the desired result, $A \cdot B \bmod N$. This process is performed as follows:

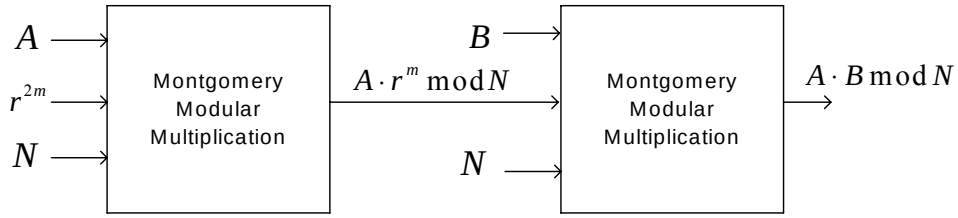


Figure 2.5: Removing the r^{-m} factor.

Table 2.2 gives a summary of the computations in the algorithm. The extraction of the post-condition in the algorithm can be obtained by rearranging the terms of the final value P_m. This process is shown in Figure 2.6.

Table 2.2: Operations in the Montgomery's Algorithm.

	Step-2a of Algorithm in Figure 1.4: $P_{i+1} = (P_i + a_i \cdot B + q_i \cdot N)/r$
$i = 0$	$P_1 = (a_0 \cdot B + q_0 \cdot N)r^{-1}$ $= (a_0 \cdot r^{-1}) \cdot B + (q_0 \cdot r^{-1}) \cdot N.$
$i = 1$	$P_2 = (\underbrace{(a_0 \cdot B + q_0 \cdot N)r^{-1}}_{P_1} + a_1 \cdot B + q_1 \cdot N)r^{-1}.$ $= (a_0 \cdot r^{-2} + a_1 \cdot r^{-1}) \cdot B + (q_0 \cdot r^{-2} + q_1 \cdot r^{-1}) \cdot N$
$i = 2$	$P_3 = (\underbrace{(\underbrace{(a_0 \cdot B + q_0 \cdot N)r^{-1}}_{P_1} + a_1 \cdot B + q_1 \cdot N)r^{-1} + a_2 \cdot B + q_2 \cdot N}_{P_2})r^{-1}$ $= (a_0 \cdot r^{-3} + a_1 \cdot r^{-2} + a_2 \cdot r^{-1}) \cdot B + (q_0 \cdot r^{-3} + q_1 \cdot r^{-2} + q_2 \cdot r^{-1}) \cdot N$
$\vdots$	$\vdots$
$i = m-1$	$P_m = (\cdots((a_0 \cdot B + q_0 \cdot N)r^{-1} + \cdots) + a_{m-1} \cdot B + q_{m-1} \cdot N)r^{-1}.$ $= (a_0 \cdot r^{-m} + \cdots + a_{m-1} \cdot r^{-1}) \cdot B + (q_0 \cdot r^{-m} + \cdots + q_{m-1} \cdot r^{-1}) \cdot N$

$$
\begin{aligned}
&P_m = (a_0 \cdot B \cdot r^{-m} + a_1 \cdot B \cdot r^{-m+1} + \cdots + a_{m-1} \cdot B \cdot r^{-1}) + \\
&\qquad (q_0 \cdot N \cdot r^{-m} + q_1 \cdot N \cdot r^{-m+1} + \cdots + q_{m-1} \cdot N \cdot r^{-1}). \Rightarrow \\
&r^m \cdot P_m = (\sum_{i=0}^{m-1} a_i \cdot r^i) \cdot B + (\sum_{i=0}^{m-1} q_i \cdot r^i) \cdot N \Rightarrow \\
&r^m \cdot P_m = A \cdot B + Q \cdot N \Rightarrow \\
&P_m = A \cdot B \cdot r^{-m} \bmod N.
\end{aligned}
$$

Figure 2.6: Extraction of the post-condition of the algorithm.

2.4. Modular Exponentiation Using Montgomery's Multiplication Method

Although Montgomery's algorithm is very popular in fast exponentiation operations, it is not very practical when only one modular multiplication is to be used. This is because of the removing process of the r^{-m} factor at the output. However, in the case of modular exponentiation, there is no need to remove this factor at each modular multiplication. Instead, the inputs of the modular exponentiation algorithm are pre-multiplied by r^{2m}, and their N-residues are obtained. (Given an integer $A < N$, its N-residue is defined as $\overline{A} = A \cdot r^m \bmod N$.) Then, these residues are used throughout the whole exponentiation. One final multiplication by 1 gives the desired result of the modular exponentiation. The following example (Figure 2.8) gives an exponentiation process with exponent equals to decimal 21. Montgomery Modular Multiplication is used with the R-L Binary Method for the modular exponentiation:

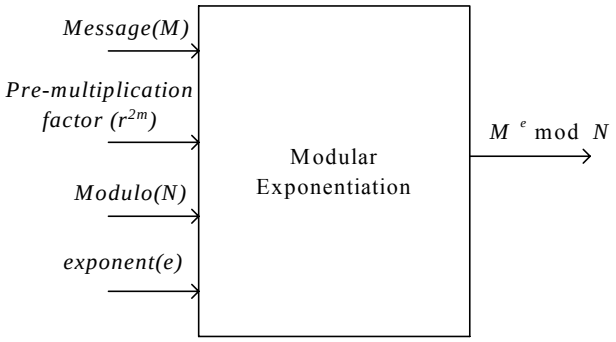


Figure 2.7: Modular Exponentiation Block.

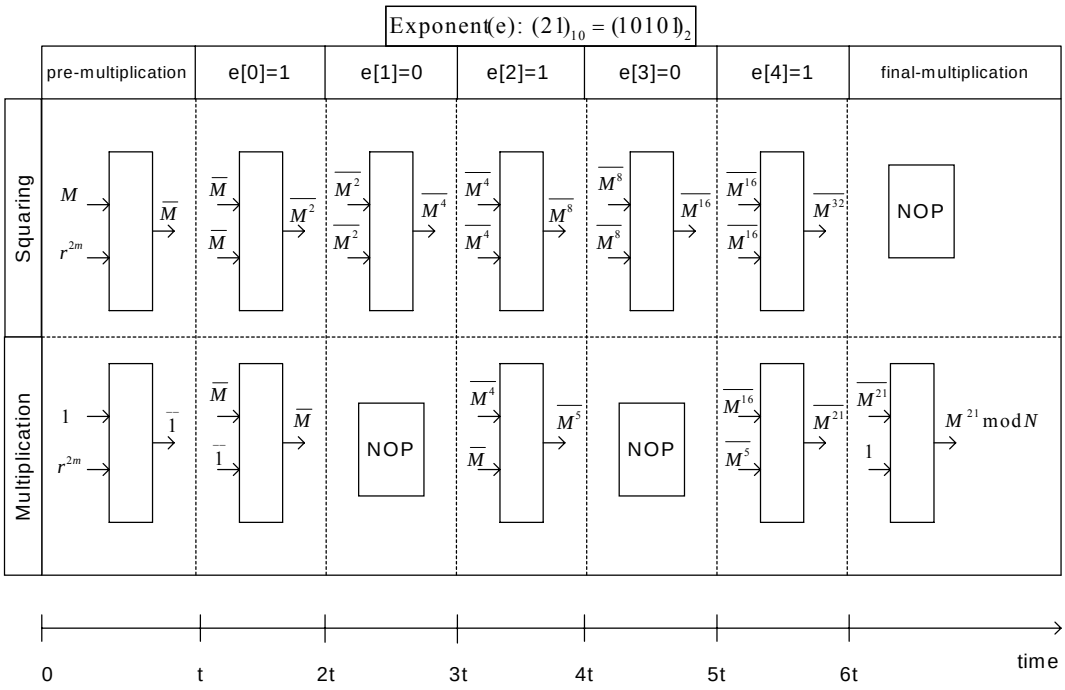


Figure 2.8: The R-L Binary Method with pre-and-final multiplications.

As seen from the above figure, the squaring operations (the upper row) and the multiplication operations (the bottom row) are performed in parallel. The time needed for the pre-and-final multiplications becomes negligible, as the length of the exponent gets larger (>1024-bits usually).

2.5. Chinese Remainder Theorem (CRT)

Chinese Remainder Theorem, which uses the main property of RSA, $N = p \cdot q$, where p and q are very large random primes, is used to speed-up the RSA cryptosystem. Since encryption is fast enough because of a small exponent, the CRT method is preferred for the decryption. According to this theorem, computation of $M = C^d \bmod N$ can be performed in two separate parts:

$$M_1 = C^d \bmod p \qquad (1)$$
$$M_2 = C^d \bmod q \qquad (2)$$

However, according to Fermat's theorem:

Consider a prime p that defines a set of integers $Z_p = \{1,2,\cdots,p-1\}$, then each element $\alpha \in Z_p$ satisfies

$$\alpha^{p-1} = 1 \mod p .$$

Using this theorem, writing the private exponent d as

$$d = A \cdot (p-1) + d_1,$$

where A is any positive integer, or,

$$d_1 \equiv d \mod (p-1),$$

then equation (1) and (2) can be rewritten as

$$M_1 \equiv C^{(A\cdot(p-1)+d_1)} \bmod p \equiv C^{d_1} \bmod p, \text{ (since } C^{(p-1)} \equiv 1 \mod p).$$

$$M_2 \equiv C^{(A\cdot(q-1)+d_2)} \bmod q \equiv C^{d_2} \bmod q, \text{ (since } C^{(q-1)} \equiv 1 \mod q).$$

With the newly produced numbers, M_1 and M_2, the computation of $M = C^d \bmod N$ with CRT is:

$$M = M_1 + [(M_2 - M_1) \cdot (p^{-1} \bmod q) \bmod q] \cdot p$$

Since p and q are about one-half of N, d_1 and d_2 are about also one-half of d. Therefore M_1 and/or M_2 can be computed in ¼ of the time needed for computation of M. This results in about 4 times speed-up in the RSA decryption procedure can be obtained if M_1 and M_2 are produced in parallel, and 2 times speed-up if they are produced in serial. Since the owner of the private key knows p and q, $d_1, d_2,$ and $(p^{-1} \bmod q)$ can be pre-computed and stored before the decryption process.

CHAPTER 3

A LITERATURE REVIEW OF THE HARDWARE IMPLEMENTATIONS OF RSA

In the previous chapter, it is mentioned that ever-increasing number of research is taking place on the performance and the area improvements of the RSA implementations. In this chapter, the reader will be informed about the development process both in theory and practice, and about the latest implementation results of the RSA algorithm.

The number of research, in parallel with the highly increasing need for the PKC algorithms, had a great jump with the beginning of the 90s. As a result of this jump, outstanding improvements on the $time \times area$ product of the RSA implementations came out in the last decade. But the question is: why are we trying to minimize the $time \times area$ product?

While examining the mathematics of the RSA, it was emphasized that the security of the RSA lies in the factorization problem of the large integers. In [37], Colin D. Walter gives a good example demonstrating this fact: The effort for factorization doubles for every 15-bits when the modulus is about 1024-bits. However, these 15 extra bits require only 5% computation time. Therefore, just speeding-up the operation 5% results in a two times difficult problem of breaking the system. Because of this reason, security is the main reason for the enormous research on speeding-up the RSA algorithm. On the other hand, the speed is also

highly needed if RSA is to be used in the real-time applications such as enciphering the real-time audio-video data.

One other way of achieving more security is to use higher number of bits in the RSA operation. Today, although 1024-bit operation is still widely used, the demand for 2048-bit RSA chips is arising. However, the silicon area needed for the RSA implementation limits the number of bits to be used. Large chip size brings placement and cost problems. Because of this reason, reducing the silicon area is an other research goal for improving the RSA.

This chapter presents a survey of the studies from the $time \times area$ product perspective. In the first section, the reader can find the theoretical basics of the various improvements. The remaining sections give information about the implementations up to date. Section 3.2 presents the VLSI implementations and Section.3.3 presents the FPGA implementations of the RSA Cryptosystem.

3.1. Theoretical Studies

Since the main mathematical operation of RSA is the modular exponentiation, which is performed as a series of modular multiplications, the studies on improving the RSA performance can be categorized as in Figure 3.1.

In 1994 Çetin Kaya Koç prepared a technical report for the RSA Data Security, Inc. [4], including a wide research on some various modular exponentiation and multiplication algorithms. Among these algorithms, the Binary Method and Montgomery's Modular Multiplication Method are the most widely preferred algorithms for exponentiation and modular multiplication, respectively.

All the implementations presented in this chapter use the Binary Method (L-R and R-L Methods are used in almost 50% among these implementations) except for the implementation of Chiang *et. al.* [23], in which the exponent is treated in 4-bit fashion. In this implementation the values $M^3, M^5,$ and M^7 are pre-computed and stored into a RAM. With this method, their gain is: a worst case of $4k/3$

multiplications in the exponentiation instead of $2k$ multiplications. But the drawback is area occupation and extra time due to storage and pre-computation processes, respectively.

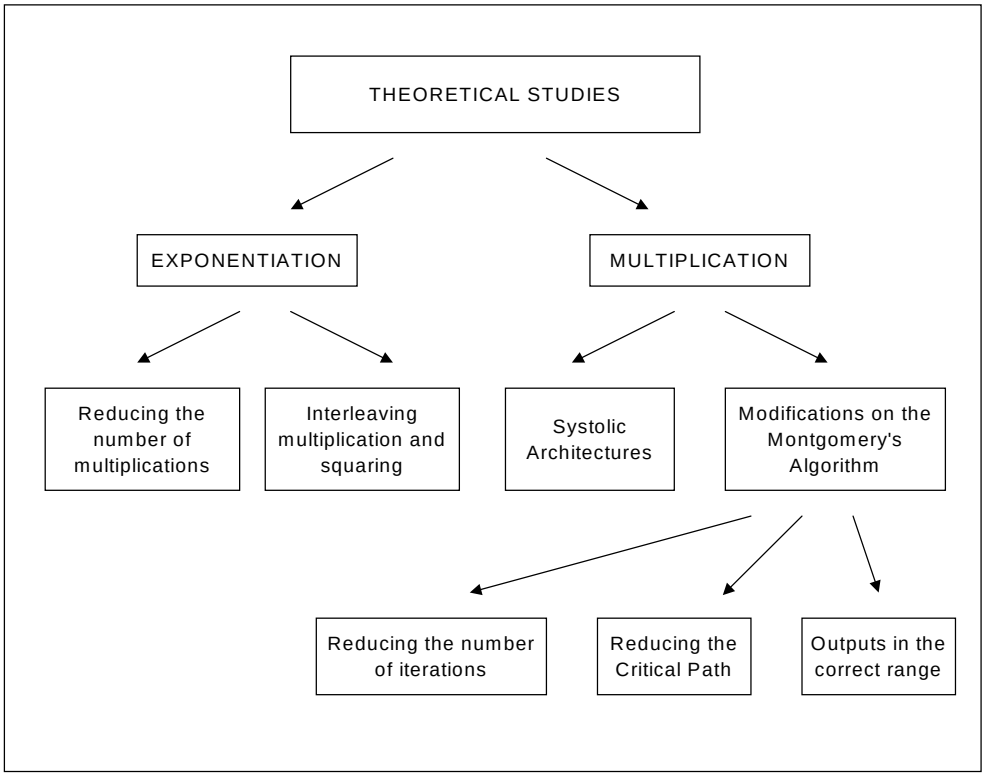


Figure 3.1: Theoretical studies on improving modular exponentiation.

As for the modular multiplication, almost all the implementations use Montgomery's method. Many modifications, those of which are categorized in Subsection.3.1.2.2, are applied to Montgomery's algorithm in RSA implementations. The details of these implementations will be examined in the proceeding sections of this chapter. A comparison of the results of these implementations will also be given together with results of this thesis as a table in chapter 4.

3.1.1. Exponentiation

The idea behind the studies on the exponentiation algorithm is to reduce the number of multiplications. Some of these algorithms are: The m-ary method, The

Adaptive m-ary Method, The Power Tree Method, and The Booth Recoding Method [4]. Generally they differ from the Binary Method by examining the exponent by two or more bits at each time. Savings up to 20% (64-ary method for n=2048) in the number of multiplications compared to the Binary Method can be achieved with these methods (Table 2.1). However, the drawback is that some pre-computation and storage requirements are introduced. The designer should carefully weigh the advantages and disadvantages of making a choice among The Binary Method and other "reduced number of multiplications" methods. Moreover, if parallelism can be achieved by using R-L Binary Method, the number of multiplications directly reduces to its lower bound, which is equal to the number of squaring operations in the exponentiation, and there is no need for pre-computing and storage effort.

The implementations to date support the observation above so that The Binary Method can be considered as the standard for modular exponentiation due to being so widely used. This method, in which the exponent is examined in bit-wise fashion, can be performed both from MSB (The L-R Method-Figure 2.1.) or LSB (The R-L Method-Figure 2.2.). The latter of these methods provide parallelism in the multiplication and squaring operations. There are two ways for parallelism:

- Placing two parallel multipliers in the hardware.

- Interleaving the two operations at successive clocks.

The first one suffers from area occupation of the extra multiplier, but can be accepted in situations where speed is more critical. On the other hand, the second one has great advantages especially for systolic architectures, which will be explained in section 3.1.2.1. To give an idea, Figure 3.2 and Figure 3.3 depict how the interleaving can be performed by using idle clocks with 100% efficiency.

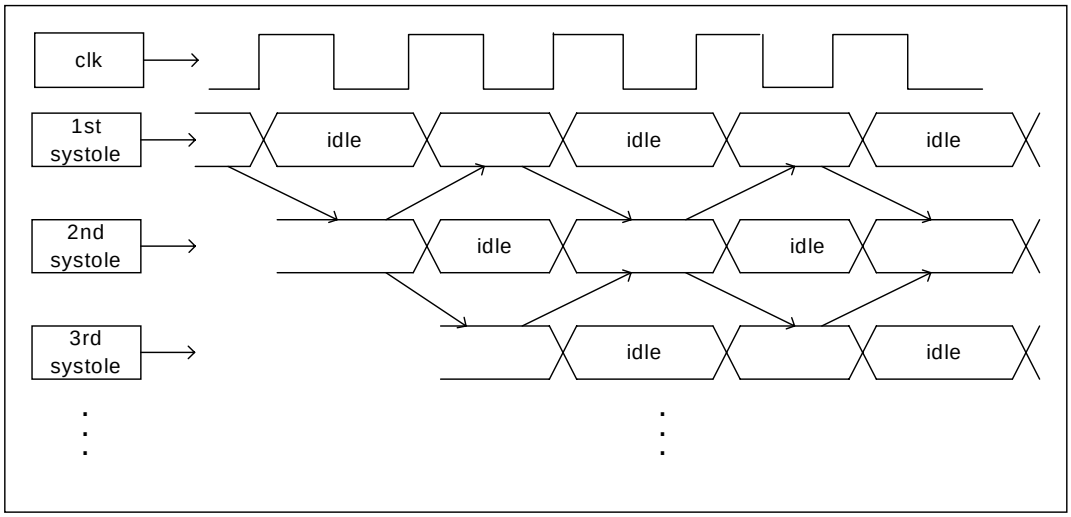


Figure 3.2: Only one operation (multiplication or squaring), without interleaving.

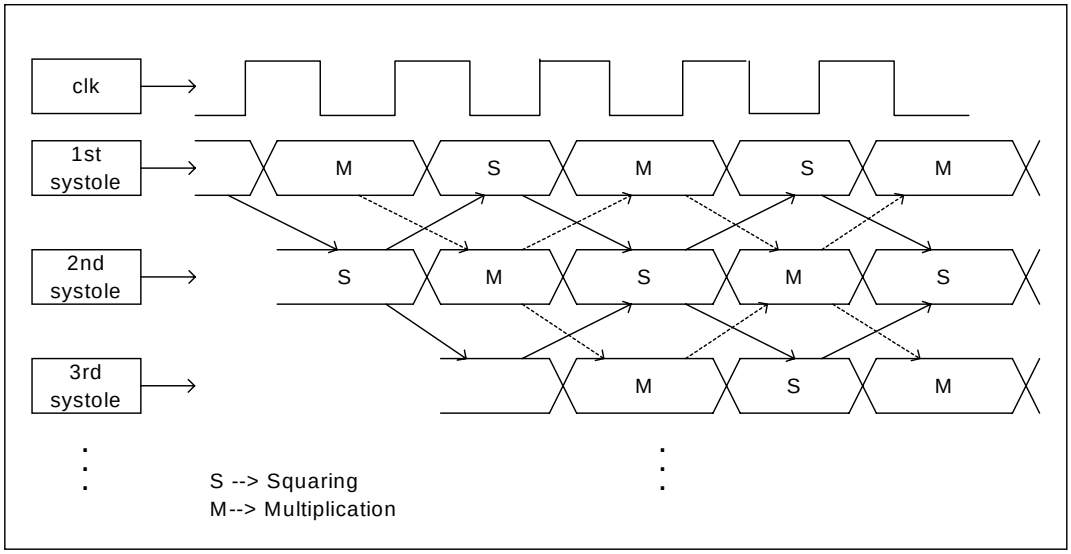


Figure 3.3: Interleaved multiplication and squaring operations by using idle clocks.

3.1.2. Multiplication

The usual way of multiplication is by scanning one of the multipliers from left-to-write and adding a multiple of multiplicand by a right shift of the partial product. This method gives the maximal carry ripple length of the parallel additions corresponding to the length of the multiplicand. In modular multiplication, instead of first multiplying and then performing modular reduction at the end, researchers

are trying to find the ways of interleaving the modular reduction into the partial steps of multiplication. With this aspect of view, many different methods were proposed in order to perform fast modular multiplication of large integers. This section will give some of these algorithms, one of which will be examined in more detail: The Montgomery Modular Multiplication Algorithm.

In 1991, Orup and Kornerup [9] proposed a modular multiplication scheme in radix-4 as follows:

Inputs : A, B, N
Output : $S = A \cdot B \bmod N$
$S = 0$; i = m - 1;
WHILE $i \geq 0$ DO
    $q = \text{Estimate}(S \text{ div } N)$;
    $S = 2^k S + a_i B - 2^k qN$;
    $i = i - 1$;
END
Correction of S

Figure 3.4: Modulo Multiplication with quotient estimation.

This method has drawbacks in modulo reduction and final correction, in which subtraction of N is performed until S belongs to the correct interval. Also the quotient estimation in the above algorithm is not a simple procedure, which is achieved by searching q values as follows:

$$S - q2^r N, \qquad q \in \{0,1,\cdots,q_{\max}\},$$

where at each step the sign of the result is checked.

Another method for modular multiplication is the L-algorithm [23], in which the modulo-reduction is performed by checking the overflows of the operation. The algorithm is given in Figure 3.5. As seen from the figure, intermediate steps (P_i's) should be reduced into correct range at each step. In [23], the number of clocks per

multiplication was given as $2n$ for an n-bit operation. However, they had to pre-compute and store some constants in order to perform modulo reduction.

Inputs : A, B, N.
Output : $P = AB \bmod N$
$P_0 = 0; M_0 = A;$
for $(i = 1; i <= n; i++)\{$
if $(b_i == 1)$
$P_i = P_{i-1} + M_{i-1} \bmod N;$
else
$P_i = P_{i-1};$
$M_i = M_{i-1} << 1 \bmod N;\}$

Figure 3.5: The L-algorithm (LSB first).

Sign-estimation technique used in the design of K-S Cho *et. al.* [32] is another method used for modular reduction in the intermediate steps of modular multiplication. In this type of multiplication algorithm, the numbers are represented in 2's complements and addition is performed instead of subtraction. In [32], they checked the 5 MSB of the partial products by using a 5-bit carry-look adder for the modular reduction. They achieved about $n/2$ iterations for n-bit multiplication. However, because of intermediate modular reduction steps, the operating speed was only 40 MHz, which is rather slow compared to other implementations.

Blackley's Method used in K. Sakiyama's FPGA implementation [34] for modular multiplication (Figure 3.6) also suffers from intermediate modular reduction problem. Step 4 in this algorithm may require two successive subtractions, $R = R - N$, to make the intermediate result in the range $[0, N-1]$, which slows down the operation.

input : A, B, N
output : $R = AB \bmod N$
1. $R = 0$
2. for $i = 0$ to $n - 1$
3. $\quad R = 2R + a_{k-1-i} \cdot B$
4. $\quad R = R \bmod N$
5. return R

Figure 3.6: Blackley's Method

While these algorithms suffer from modulo reduction, P.L. Montgomery found a very clever solution to this problem: His algorithm reverses the treating of the digits of the multiplicand, operating from the LSB of the partial products. For the intermediate modular reduction process, a multiple of modulo, N, is added instead of subtracting. With this method, the partial product is guaranteed to be divisible by the radix-r, which is usually a power of 2. Therefore modular reduction becomes a simple right shifting and removing process of the right-most digit, which is zero. Because of the above advantages, it is used in almost all the implementations to date.

The Montgomery Algorithm, as presented in C.D. Walter's systolic implementation [10], is as follows:

Inputs : A, B, N
Output : $P = ABr^{-m} \bmod N$
$P_0 = 0;$
for $i = 0$ to $m - 1$
$\quad$ a. $q_i = (p_0 + a_i b_0) \cdot (r - n_0)^{-1} \bmod r;$
$\quad$ b. $P_{i+1} = (P_i + a_i B + q_i N) / r.$
end
$r^m P_m = AB + QN.$

Figure 3.7: Montgomery's Multiplication Algorithm in radix-r.

3.1.2.1. Montgomery's Algorithm in Systolic Architectures:

A systolic architecture is a regular array of processing elements, denoted as PE_i, where:

- All the PE's are similar to each other and operate synchronously. Hence the design can be expanded by repeating the systoles.
- All the communicate locally. Therefore routing problems are minimized.
- Each PE is as simple as possible so that the architecture can operate at high clock frequencies.

The following are two examples for systolic arrays:

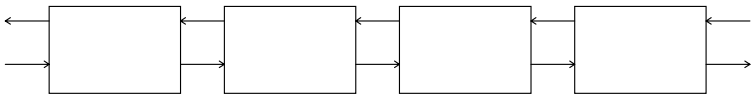


Figure 3.8: Linear systolic array.

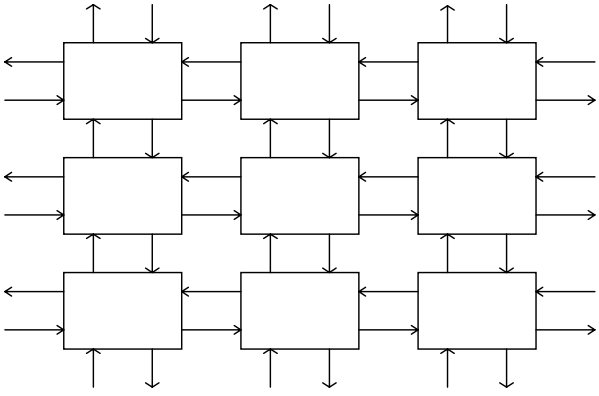


Figure 3.9: Rectangular systolic array

Because of these three properties, the systolic architectures are very suitable for the Montgomery's Modular Multiplication Algorithm. In 1992, Colin D. Walter proposed a method for systolic modular multiplication, in which the operations are performed in radix-2 [10]. In this architecture, one of the multipliers is fed to the array in bit wise fashion. Each bit of the other multiplier is hardwired to each systole. At a given clock cycle, each processing element performs the combinational operation and directly passes the incoming data to the next systole at the next clock. The carry propagation is pipelined through the systoles. The intermediate result is right-shifted through the systoles digit-by-digit at each clock. This process is illustrated in Figure 3.10.

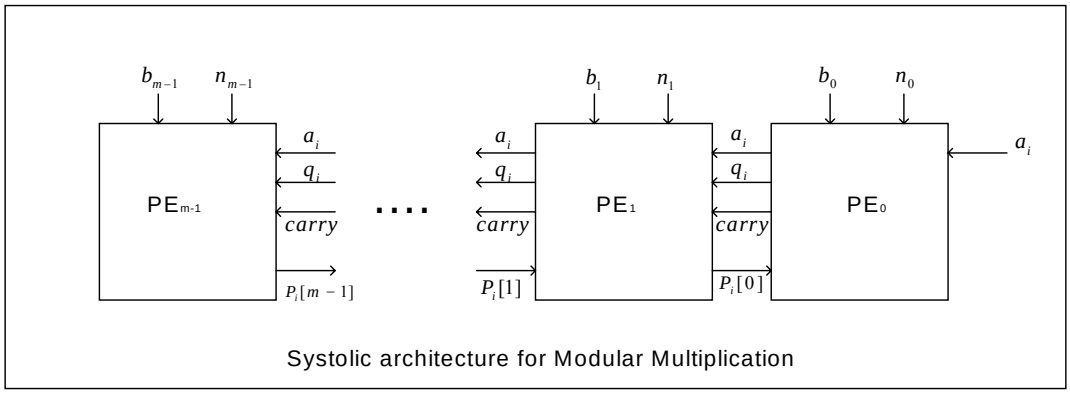


Figure 3.10: A linear array of PEs for Montgomery's Algorithm.

Figure 3.11, which is taken from [10], shows the interior circuitry of the proposed PE in the modular multiplication design. As shown in the figure, each PE is composed of 5 XOR, 2 OR, and 7 AND gates. In addition to the combinational circuitry, the systole involves 5 1-bit registers (2-for carry-bits, 1-for each $A[i], Q[i],$ and $P[i]$). For a k-bit modular multiplication, if the linear array structure is to be used, the architecture requires k systoles, therefore at least 5k XOR, 2k OR, 7k AND gates and 5k FFs.

Among the implementations using Montgomery's algorithm, many of them suited modular multiplication into systolic architecture.

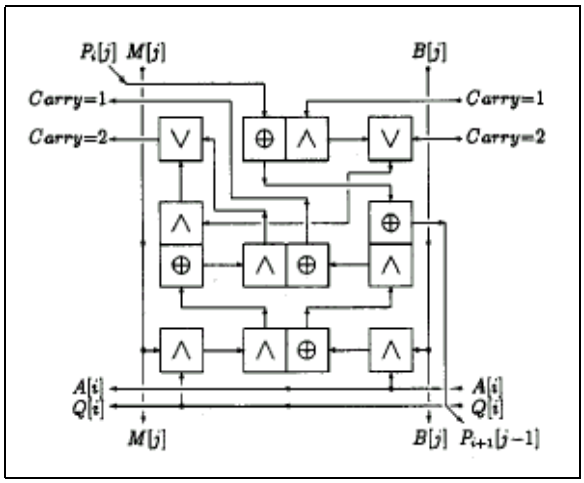


Figure 3.11: Typical cell for radix-2.

3.1.2.2. Modifications on the Montgomery's Algorithm:

Although, Montgomery's Algorithm is ideally suitable for digital applications, lots of small but effective modifications have been made in the implementations up to date. In this section, these modifications are categorized so that the reader will have a better understanding of the improvements while examining the implementations in sections 3.2, and 3.3.

a) Reducing the Number of Iterations:

This method is very commonly used in both ASIC and software implementations. Two ways for reducing the number of iterations are widely used:

- Using a high-radix representation of the numbers,
- Using booth-like multiplication,

Since the modular multiplication is the kernel operation, reducing the number of iterations in a multiplication directly affects the total number of clocks. However, as complexity of each iteration increases, the time needed for each

iteration increases. This results in a low clock frequency, which slows down the operation.

In Figure 3.7, multiplication in radix-r is shown. The number of iterations $(m = n/(\log_2 r))$ decreases as r increases. However, the operations in step-a and step-b requires more complex circuitry as the radix increases. For example, when radix is 2, the calculation of q_i in step-a needs only one XOR and one AND gate (Since N is always odd, $(r - n_0)^{-1} \bmod r = 1$, when $r = 2$). On the other hand, if radix-8 is used, all the digits are in 3-bit length. Therefore the needed amount of sources for the same calculation turns to be (two 3×3 multipliers + 6-bit adder + 3-bit LUT) (Figure 3.12). This increases the critical path. However, the total number of clock cycles required to complete a modular multiplication is reduced to 1/3rd of radix-2.

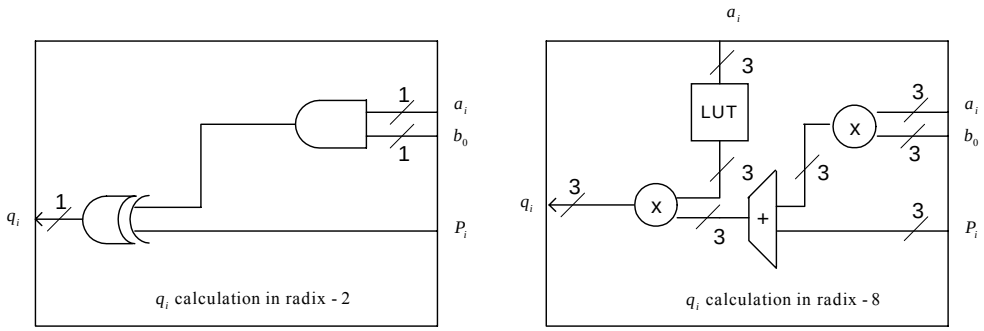


Figure 3.12: Q-Calculation circuitries for two different radix values.

b) Reducing the Critical Path:

One way of removing the disadvantage of using high radix is to modify the algorithm in such a way that the critical path is reduced. As depicted in Figure 3.7, step-a has the potential of increasing the critical path. In order to reduce the complexity of q_i - calculation , one of the inputs, say B, is shifted up by r. That is, the least significant digit of B becomes zero. This operation directly eliminates one of the multiplications and the addition operation in the above figure. Recovering of the modification is achieved simply by increasing the number of iterations by one as

in the implementations of A. Royo *et. al.*[15] and M. K. Hani *et. al.* [38]. This process is shown in Figure 3.13:

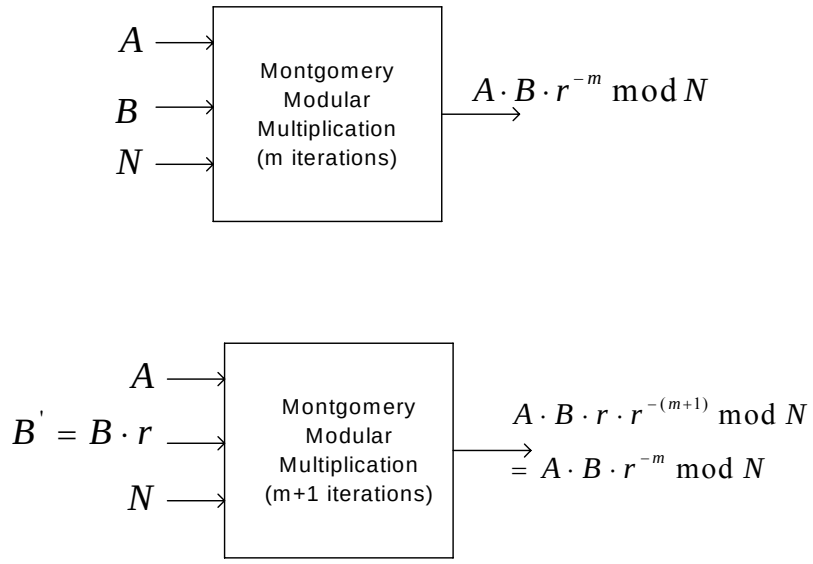


Figure 3.13: Recovery of the modification in the q_i-calculation.

c) Obtaining the Outputs in the Correct Range:

In the exponentiation, the outputs of the modular multiplication unit are reused as inputs for a new modular multiplication. Therefore, the bounds on the I/O of the Montgomery algorithm become important. However, a careful inspection on the algorithm in Figure 3.7 shows that the intermediate results are bounded to $B+N$ as shown in the table below:

Table 3.1: All the a_i's and q_i's are assumed to be $(r-1)$ in the worst case.

$i=0$	$P_1 = [0 + B(r-1) + N(r-1)]/r < B+N$
$i=1$	$P_2 < [B+N+B(r-1)+N(r-1)]/r = B+N$
$\vdots$	$\vdots$
$i=m-1$	$P_m < [B+N+B(r-1)+N(r-1)]/r = B+N$

Therefore the output of the first modular multiplication is bounded in the range $[0, B+N)$ or $[0,2N)$. The ongoing multiplications will result in the bound of $[0,3N)$. In some implementations [15][33], a final comparison by N and subtraction is performed to reduce this bound again to $[0,2N)$. On the other hand, in some other implementations [13][14][25][31], the bounding of the output is performed via applying one or more iterations in the algorithm. In other words, in the extra iterations, since a_i's are equal to zero anymore, one term of the addition in step-b will be vanished and division by r will reduce the result to satisfy the needed bound.

Table 3.2: Bound is satisfied via one more iteration.

⋮	⋮
$i = m-1$	$P_m < 3N$
$i = m$	$P_{m+1} < [3N + 0 + N(r-1)]/r = N + 2N/r < 2N,\ (r >= 2)$

However, final multiplication by 1 at the end of the exponentiation should guarantee that the result of the exponentiation must be in the range $[0, N)$. In the papers of C.D. Walter [19], and T. Blum and C. Paar [22], clear explanations demonstrate that the final multiplication by 1 will result in a bounded output in the range $[0, N)$. The origin of their idea comes from the requirements of the RSA mathematics. Actually, it is seen by inspection that, multiplication by 1 $(A = (000\cdots01)_2)$ will introduce all the a_i 's equal to zero, except for the first one. A worst-case examination of the algorithm, in which all the digits of quotient, q_i's,

having their highest value $(r-1)$, the limit of the output will approach to the value N.

Table 3.3: Final multiplication by 1. All the a_i's are zero, except for the first one.

$i=0$	$P_1=[0+1\cdot B+0]/r=Br^{-1}$
$i=1$	$P_2=[Br^{-1}+0+N(r-1)]/r=N+Br^{-2}-Nr^{-1}$
$\vdots$	$\vdots$
$i=m-1$	$P_m=[N+Br^{-(m-1)}-Nr^{-(m-2)}+N(r-1)]/r=N+Br^{-m}-Nr^{-(m-1)}$

Since $B<2N$ at the beginning of the final multiplication, the following inequality can be obtained:

$$Br^{-m}-Nr^{-(m-1)}<(2N-rN)r^{-m}\le 0, \qquad \text{for r} \ge 2 .$$

Therefore, the result at the last iteration falls into the range $[0,N)$.

3.2. VLSI Implementations

This section presents the recent VLSI implementations of RSA algorithm. However, a survey of the RSA implementations, prepared by Ernest F. Brickell in 1990 [3] is given in Table 3.4 to give the reader a reference point to compare the improvements after 90s.

Table 3.4: A Survey of RSA Hardware Implementations up to 1989 (Brickell [3]).

	Year	Tech.	# of bits per chip	Clock	Baud (# of bits)	# of clocks per 512-bit op
Sandia	1981	$3\mu m$	168	4MHz	1.2K (336)	4.0×10^6
Bus. Sim.	1985	GateArray	32	5MHz	3.8K (512)	0.67×10^6
AT&T	1987	$1.5\mu m$	298	12MHz	7.7K (1024)	0.4×10^6
Cylink	1987	$1.5\mu m$	1024	16MHz	3.4K (1024)	1.2×10^6
Cryptech	1988	GateArray	120	14MHz	17K (512)	0.4×10^6
CNET	1988	$1\mu m$	1024	25MHz	5.3K (512)	2.3×10^6
Brit.Telecom	1988	$2.5\mu m$	256	10MHz	10.2K (256)	1×10^6
Plessy	1989	-	512	-	10.2K (512)	
Sandia	1989	$2\mu m$	272	8MHz	10K (512)	0.4×10^6
Philips	1989	$1.2\mu m$	512	16MHz	2K (512)	4.1×10^6

In 1996, two similar VLSI implementations, based on the reducing critical path approach, came from two different universities of Taiwan with P.-S. Chen *et. al.* [13] and C.-C. Yang *et. al.* [14]. In [13] they used a method with $2n$ operations per multiplication with a clock frequency of $50MHz$. Their implementation results were as follows:

- 512-bit RSA operation
- A Prototype CMOS VLSI design using $0.8\mu m$ technology.
- $\cong 78K$ gate count at a chip area of $76mm^2$
- 1.05×10^6 clock cycles per 512-bit operation
- A baud of $24.3Kb/s@50MHz$ clock frequency

On the other hand, in [14] they proposed a similar algorithm to [13], but they used only about $n+2$ iterations per multiplication. The drawback of their method was introducing one more addition operation in each iteration. The results of this implementation were:

- 512-bit RSA operation
- A single chip using COMPASS $0.6\mu m$ SPDM library
- $\cong 74K$ gate count at a chip area of $56mm^2$
- The critical path of $6.06ns$ in the simulations.
- 0.54×10^6 worst-case clock cycles per operation.
- Expected clock frequency of $125MHz$ with estimated routing delay about $2ns$.
- Expected baud of $118Kb/s@125MHz$ clock frequency (worst-case)

In [15], A. Royo *et. al.* used a somewhat different technique for the Montgomery Algorithm. First of all, they used a radix-4 Modified Montgomery Algorithm by reducing the critical path as mentioned in the previous section. Differently, they shifted up B by r^2 and then applied two more iterations in the multiplication algorithm. In addition to this, they used a Carry Save Representation (CSR) for the intermediate results, by which they believe that the operation speed is increased. However, this technique had a drawback: conversion from CSR to non-redundant binary representation at every modular multiplication. Their results were as follows:

- 768-bit Operation
- ES2 $0.7\mu m$ CMOS technology.
- Area of $77mm^2$
- 400 clocks per multiplication and 0.5×10^6 clocks per exponentiation. (Average case)
- $72.5Kb/s@50MHz$.

The design of J.-H. Guo *et. al.* [16] is based on the bit-serial systolic architecture, which does not include any broadcasting signals. Although, a similar algorithm to the one in [14] was used, they reached to a higher performance because of a more compact hardware design. In addition to this, they used R-L Binary method, which utilizes the parallelism of the multiplication and squaring. However, the parallelism was performed by using two multipliers, therefore the area of this design is almost twice as much as the previous designs in [13] and [14]. Their results can be summarized as follows:

- 512-bit operation
- COMPASS $0.6\mu m$ CMOS standard cell library
- $132K$ gate count at a chip area of $68mm^2$.

- Estimated clock frequency of $143MHz$

- Estimated baud of $278Kb/s@143MHz$.

A different approach on the Montgomery's Algorithm came from Jye-Jong Leu and An-Yeu Wu [20] in 1999. In this implementation, as shown in the algorithm taken from [20] (Figure 3.14), the Booth function calculates the coefficients (c_i's) of the input B of the multiplier. Another multiplexing operation is performed to determine the coefficient of N in the algorithm. Different from the previous examples, left-shift is performed instead of right-shift for modular reduction. With these modifications, the proposed algorithm takes $n/2$ operations per multiplication. In the implementation, they used a systole-like architecture with $n/2+1$ unit cells, each corresponding to 2 bits of the operation. However, they noticed that at any clock cycle, half of these cells stayed idle. Therefore they reduced the number of cells by half to $n/4+1$, and thus they reached a total of about $64K$ gate count. One other solution for 100% utilization of the idle cells could be by exploiting parallelism in the exponentiation with R-L Binary Method as illustrated in Figure 2.8. For the exponentiation, they used L-R Binary method, in which $3n/2$ multiplications are needed on the average. The critical path obtained in the design (without routing delays) was $6.7ns$. For routing, they made an estimation by adding 30% to critical path delay. A summary of this implementation is as follows:

- 512-bit Operation

- $0.6\mu m$ (Technology name was not given)

- $64K$ gate count (The chip area was not stated)

- $0.53M$ clock cycles.

- Estimated clock frequency of $115MHz$.

- Estimated baud of $111Kb/s@115MHz$.

```
Booth( A )                    /* Booth encoder */
{       q_0 = 0; a_n = 0; a_{n+1} = 0;
        for (i = 0; i <= n; i = i + 2)
        {       switch({ a_{i+1} a_i q_{i/1} })
                {   case 0 :  c_{i/2} = 0;    break;
                    case 1 :  c_{i/2} = 1;    break;
                    case 2 :  c_{i/2} = 1;    break;
                    case 3 :  c_{i/2} = 2;    break;
                    case 4 :  c_{i/2} = -2;   break;
                    case 5 :  c_{i/2} = -1;   break;
                    case 6 :  c_{i/2} = -1;   break;
                    case 7 :  c_{i/2} = 0;    break;
                }
                q_{i/2+1} = a_{i+1};
          }
          return  C = {c_{n/2}, c_{n/2-1}, c_{n/2-2}, ... , c_{n/2}, c_{n/2}};
}
Sel( q_i, n_I )
{       r_0 = q_i[0];
        if (n_I == 0)
              r_1 = q_i[1] ⊕ q_i[0];
        else
              r_1 = q_i[1];
        return   R = { r_1, r_0 };
}
M3( A, B, N )
{       C = Booth( A ) = {c_{n/2}, c_{n/2-1}, c_{n/2-2}, ... , c_{n/2}, c_{n/2}};
        P[0] = 0;          /* c_i = {2,1,0,-1,-2}; */
        For (i = 0; i <= n/2; i = i + +)
        {       q_I = (P[i] + c_i · B) mod 4;
                R = Sel( q_i, n_I );
                P[i + 1] = (P[i] + c_i · B + R · N) << 2;
        }
        return   P[n/2 + 1]
}
```

Figure 3.14. The proposed Booth-encoded Montgomery Algorithm in [20].

In 1999, the authors of [13] further improved their study on Montgomery Algorithm and proposed a new method in [21]. In this new design they revised the algorithm such that one addition is required in each iteration of the modular multiplication. They used 2's complement multiplier and modular-shift adder, both of which are composed of linear cellular arrays. However, this was a theoretical study rather than a hardware implementation so that the results were presented in terms of full adder (FA) parameters used in the design. The time spent per operation was given as $2n^2\tau$, and the area was given as $2n\alpha$, where τ and α are roughly the delay and the area of a FA.

The implementation of J.-S. Chiang *et. al.* [23] differs from the others with respect to the exponentiation and multiplication algorithms used. The exponentiation is performed by scanning 4-bits of the exponent at a time. Even though, the worst-case number of multiplications is reduced to $4n/3$ with this method, the R-L Binary Method with interleaved squaring and multiplication still gives better result (n-multiplications per exponentiation). On the other hand, this method needs pre-computation and storage of $M^3, M^5,$ and M^7 in the RAMs. As for the multiplication, they used modified L-algorithm with $2n$ cycles per multiplication. The results of the implementation were:

- 512-bit Operation
- COMPASS standard cell library (TSMC $0.6\mu m$ process).
- Critical path of $6ns$ estimated by COMPASS Input Slope Model
- $0.7M$ clock cycles per operation
- $122Kb/s@166MHz$ estimated clock.

Recently, 1024-bit implementations have begun to take place of the previous 512-bit implementations. The design proposed by Y.S. Kim *et. al.* [25] is one of them. They used Montgomery's Algorithm and L-R Binary Method for the modular exponentiation. Instead of a systolic architecture, they used 32-bit Carry Save

Adders (CSA) and Carry Propagation Adders (CPA) in the design. The results were as follows:

- 1024-bit Operation
- $0.65\mu m$ SOG Technology
- $112K$ gate counts
- Clock cycles of $2.2M$ (worst - case) and $1.6M$ (average case)
- Clock frequency of $50MHz$
- Worst-case operating time of $43ms$ ($32.5ms$ in the average case)

Two implementations for a 1024-bit RSA operation were proposed by T.-W. Kwon *et. al.* [31]. One of them was an area-critical design, which uses the L-R Binary Method, and thus needs $2n$ multiplications for an n - bit process at worst-case. The other was a speed-critical design using the R-L Binary Method, in which the parallelism was performed via two separate multipliers. Montgomery's Method was used in both designs for the modular multiplication. Some of the authors of this paper had also worked in the implementation of the 1024-bit RSA Processor in [25]. Newly, they improved their previous design by adding R-L Binary Method with parallel multiplication and squaring. Although they reached a 2 times better speed performance, the second multiplier had increased the area of the design. The comparison of the two proposed implementations is as follows:

Table 3.5: 1024-bit RSA implementations using two types of Binary Method (Kwon *et. al.* [31]).

	L-R Method	R-L Method
Technology	$0.5\mu m$ SOG	$0.5\mu m$ SOG
Gate counts	92K	156K
Clock Frequency	50MHz	50MHz
Clock Cycles	1.6M(average) 2.2M(worst)	1.1M
Operating Time	32.5ms(average) 43ms(worst)	22ms

The implementation of K.-S. Cho *et. al.* [32] uses a new radix-4 modular multiplication algorithm based on sign-estimation technique, of which a brief discussion was given in section 3.1.2. With this technique, they reached to a value of $n/2+3$ clock cycles per n-bit multiplication. In addition, they used R-L Binary Method by performing n multiplications per n-bit exponentiation. Since, the parallelism was achieved via a second multiplier, the drawback of this implementation became the high number of gate counts: $230K$. Although the design had resulted in a slow operating clock of $40MHz$, they obtained a considerably high performance for a 1024-bit RSA process. The result was $13ms$ operating time and a baud of $78.8Kb/s@40Mhz$. Obviously, there are two main reasons underlying such a performance with a clock frequency as low as 40MHz:

- Reduced number of iterations in the multiplication
- R-L Binary Method with parallel multiplication and squaring operations.

3.3. FPGA Implementations

In 1998 J. Poldre, K. Tammenae, and M. Mandre have analyzed the FPGA implementations of the RSA by using three families of FPGAs: XC4000, XC6200, and FLEX10K [39]. They used the R-L Binary Method with parallel S&M, and radix-2 Montgomery algorithm in their implementation. They proposed two different architectures, systolic and classical, for the modular exponentiation. However the timing results were some estimation based on the number of clocks and delay values of the FPGA resources. Besides, only public key processes requiring operations with a small exponent (not private key processes) were taken into account. These results are shown in Table 3.6 and Table 3.7, which are taken from the corresponding paper [39]. The *k*-values in the tables represent the bit-lengths of the unit operations in the design.

Table 3.6: Estimated CLB count and number of cycles for two types of architectures in 3 FPGA families (J. Poldre *et. al.*[39]).

bits	k	XC4000 Systole	XC4000 Classic	XC6200 Systole	XC6200 Classic	FLEX10K Systole	FLEX10K Classic	cycles Systole	cycles Classic
512	2	4096	6144	16384	24578	8192	12288	256	128
512	4	7168	10752	26624	39936	14336	21504	128	64
512	8	12288	18432	40960	61440	24576	36864	64	32
512	2	8192	12288	32768	49152	16384	24578	512	256
512	4	14336	21504	53248	79872	28672	43008	256	128
512	8	24576	36864	81920	122880	49152	73728	128	64

Table 3.7: Estimated time (in msec) of RSA Public Exponentiation Process. Only exponentiation times for small exponent are presented (J. Poldre *et. al.*[39]).

Bits	*k*	XC4000 Systol.	XC4000 Classic.	XC6200 Systol.	XC6200 Classic.	FLEX10K Systol.	FLEX10K Classic.	Cycles Systol.	Cycles Classic.
512	2	3,9	2,0	5,2	2,6	5,2	2,6	256	128
512	4	2,3	1,1	3,4	1,7	3,3	1,6	128	64
512	8	1,3	0,7	2,1	1,0	2,0	1,0	64	32
1024	2	15,7	7,9	21,0	10,5	21,0	10,5	512	256
1024	4	9,2	4,6	13,6	6,8	13,1	6,6	256	128
1024	8	5,2	2,6	8,4	4,2	7,9	3,9	128	64

In 1999, T. Blum and C. Paar proposed an FPGA implementation of the RSA Cryptosystem, which uses the modified Montgomery's Algorithm in a systolic architecture [22]. In this implementation, they performed operations in radix-2, using the R-L Binary Method with interleaved S&M. Also, they simplified the q_i-calculation by shifting up one of the modular multiplication inputs as explained in 3.1.2.2.b. Differing from the previous implementations, they applied three more iterations in their modular multiplication algorithm than the original case. Although, parallel S&M introduced great advantages in reducing total number of clocks, they obtained 2(n+2)(n+4) clock cycles for an n-bit operation, which is rather high because of the operations handled in radix-2. Typical PE in this architecture was different from the convenient ones proposed by C.D. Walter [10] in such a way that each PE performed calculations more than 1-bit although radix-2 was used. They implemented the design for three types of PE's (4, 8, and 16 bits) in order to find the solution which best suits to the Xilinx XC4000 series FPGA's, which they used in the implementation. The following table taken from [22] summarizes their results

for these three types of PE's. The *u* values represent the number of bits computed in each PE.

Table 3.8: CLB usage and execution time for a full modular exponentiation (Blum *et. al.* [22]).

	512 bit		768 bit		1024 bit	
u	C CLBs	T [ms]	C CLBs	T [ms]	C CLBs	T [ms]
4	2555	9.38	3745	22.71	4865	40.50
8	2094	10.13	3123	23.06	4224	49.36
16	2001	11.56	2946	25.68	3786	49.78

After two years, the same authors implemented a similar systolic architecture in Xilinx XC40250XV and XC40150XV FPGA's with operations performed in radix-16 [36]. Different from the previous one, the modular multiplication algorithm used in this new implementation (Figure 3.15) was an optimized version of the Montgomery's Algorithm to be used for high radix computations as in proposed by H. Orup [40]. However, in the paper [36], they didn't give sufficient details of how they implemented the initialization part of the new algorithm. Using a higher radix in the Montgomery computations, they reduced the number of clocks about a quarter of the previous implementation, while preserving the clock frequency. Therefore they have achieved to a four times faster RSA operation than the previous implementation. Table 3.9 compares the two FPGA implementations [22] and [36] of T.Blum and C.Paar.

MONT (A, B): Montgomery Modular Multiplication for computing

$A \cdot B \bmod M$, where $M = \sum_{i=0}^{m-1} (2^k)^i m_i$, $m_i \in \{0,1,\ldots,2^k - 1\}$;

$\widetilde{M} = (M' \bmod 2^k) M$, $\widetilde{M} = \sum_{i=0}^{m} (2^k)^i \widetilde{m}_i, \widetilde{m}_i \in \{0,1,\ldots,2^k - 1\}$;

$B = \sum_{i=0}^{m+1} (2^k)^i b_i, b_i \in \{0,1,\ldots,2^k - 1\}$;

$A = \sum_{i=0}^{m+2} (2^k)^i a_i, a_i \in \{0,1,\ldots,2^k - 1\}, a_{m+2} = 0$;

$A, B < 2\widetilde{M}$; $4\widetilde{M} < 2^{km}$; $M' = -M^{-1}$;

1. $S_0 = 0$
2. **For** $i = 0$ **to** $m + 2$ **do**
3. $q_i = (S_i) \bmod 2^k$
4. $S_{i+1} = (S_i + q_i \widetilde{M}) / 2^k + a_i B$
5. **End for**

Figure 3.15: The Modular multiplication Algorithm proposed in [36].

Table 3.9: Comparison of the two FPGA implementations [22] and [36].

Radix	512-bit		768-bit		1024-bit	
	CLB	Time (ms)	CLB	Time (ms)	CLB	Time (ms)
2 [22]	2555	9.38	3745	22.71	4805	40.05
16 [36]	3413	2.93	5071	6.25	6633	11.95

In [30], A. Daly and W. Marnane proposed three types of Montgomery Algorithms and compared their FPGA performances. Among the proposed algorithms, they had chosen the one in Figure 3.16, in which the computations are handled in radix-2 and the q-calculation is made easier by shifting-up one of the inputs as in [22]. As for the exponentiation, they used the R-L Binary Method, however they performed parallel S&M with two multipliers instead of interleaving

them in one multiplier. The target device in this implementation was the Xilinx V1000FG680-6. The results of this FPGA implementation are shown in Table 3.10.

MonPro3 (*A*, *B*, *M*)
{
$S_{-1} = 0$;
$A = 2 \times A$;
for $i = 0$ **to** n **do**
$q_i = (S_{i-1}) \bmod 2$;
$S_i = (S_{i-1} + q_i M + b_i A)/2$;
end for
Return S_n;
}

Figure 3.16: The modular multiplication algorithm used in [30].

Table 3.10: Implementation results in Xilinx V1000FG680 FPGA (Daly *et.al.* [30]).

Bit Length(n)	No. of Slices (% of chip)	Clk. Freq. (MHz)	Data Rate
120	1146 (9%)	83.51	673.2 kb/s
240	2301 (18%)	58.15	238.3 kb/s
480	4610 (37%)	55.92	115.5 kb/s
720	6917 (56%)	50.66	70.0 kb/s
1080 (1024)	10369 (84%)	49.63	45.8 kb/s

CHAPTER 4

HARDWARE IMPLEMENTATION

This chapter presents the details of the hardware implementation of the RSA PKC performed within this thesis. The design and implementation results presented in this chapter are summarized in a paper, which was presented and published as a work in progress at Euromicro Symposium on Digital System Design, DSD2003, in September 2003 [42].

Section 4.1 examines the methods and modifications that are used in the design. Also a top-to-bottom hierarchic description of the design architecture is given in this section. VLSI and FPGA implementation details are handled separately in Section 4.2 and 4.3, respectively. Finally the last section gives a comparison of the results presented here to the ones in the literature, which were presented in Chapter 3. The reader may refer to the previous section for detailed explanations of the used algorithms and methods in this chapter.

4.1. Design Aspects

4.1.1. Design Methodology

For the exponentiation operation, The R-L Binary Method is used (Figure 2.2). This Method treats the exponent from the LSB in bit-wise fashion. The multiplication and squaring operations are interleaved into successive clock cycles

using a single multiplier, hence a 100% utilization of the hardware and time has been reached in the design. In other words, with interleaved multiplication and squaring, there are no idle clocks and idle resources while the system is running. Also there is no need to use exponentiation algorithms such as m-ary Method, Power Tree Method or Booth Recoding Method in order to reduce the number of multiplications.

For modular multiplication, Montgomery's Modular Multiplication Algorithm (Figure 2.3) is used. Among the modifications on this algorithm, which were categorized in Chapter 3, the following are utilized in this implementation:

- The number of iterations are reduced by performing the algorithm in radix-4. A higher radix would result in a less number of iterations but since the computations become more complex as the radix increases, the critical path and needed amount of resources also increase. Radix-16 computations were also tried in the implementation. However, while the total operation time remained almost as the same as for radix-4, the area increased about 32%. The following table explains the effect of increased radix:

Table 4.1: Comparison of radix-4 and radix-16 designs.

	Gate Count	Clock Period	Worst-case # of clocks
Radix-4	132K	4.06ns	$\cong 1024 \times 2 \times (1024/2)$
Radix-16	175K	8.3ns	$\cong 1024 \times 2 \times (1024/4)$

- Another modification was performed to reduce the critical path by simplifying the q_i-calculation (see section 3.12.2.b). In the modular multiplication, one of the inputs, B, was shifted-up by 4, making the least significant digit zero. The new q_i-calculation became:

$$q_i = (p_0 + q_i \cdot \overset{0}{\cancel{b_0}}) \cdot (r - n_0)^{-1} \bmod r \quad \text{and} \quad b_0 = 0 \quad \Rightarrow$$
$$q_i = p_0 \cdot (r - n_0)^{-1} \bmod r$$

As explained in section 3.1.2.2.b, one more iteration was applied in the algorithm to recover the same result after this simplification.

- In order to guarantee that the outputs of the modular multiplication are in the range $[0, 2N)$, another extra iteration was added in the multiplication algorithm. The mathematical details of this modification were given in section 3.1.2.2.c.

With the above modifications, the algorithm that is used in the design has turned to be:

Inputs: A, B, N;
Output: $P = A \cdot B \cdot 4^{-(m+1)} \bmod N$;
$B' = B \cdot 4$; */* Reducing the critical path */*
$P_0 = 0$;
For $i = 0$ **to** $m + 1$ */* Two more iterations */*
a. $q_i = p_0 \cdot (4 - n_0)^{-1} \bmod 4$; */* Simpler q_i-calculation */*
b. $P_{i+1} = (P_i + a_i \cdot B' + q_i \cdot N) / 4$;
End
Post Condition: $4^{m+1} \cdot P_{m+1} = A \cdot B + Q \cdot N$

Figure 4.1: The Algorithm used in the design.

As seen in the above algorithm, there are two more iterations with respect to the original algorithm (Figure 3.7). One of these extra iterations is for the recovery of the shift-up operation of one of the inputs. The other iteration is for obtaining the output in the range $[0, 2N)$.

4.1.2. Design Architecture

There are four levels of hierarchy in the 1024-bit RSA Cryptosystem design as seen in Figure 4.2. In this section, the design architecture will be described from top to bottom.

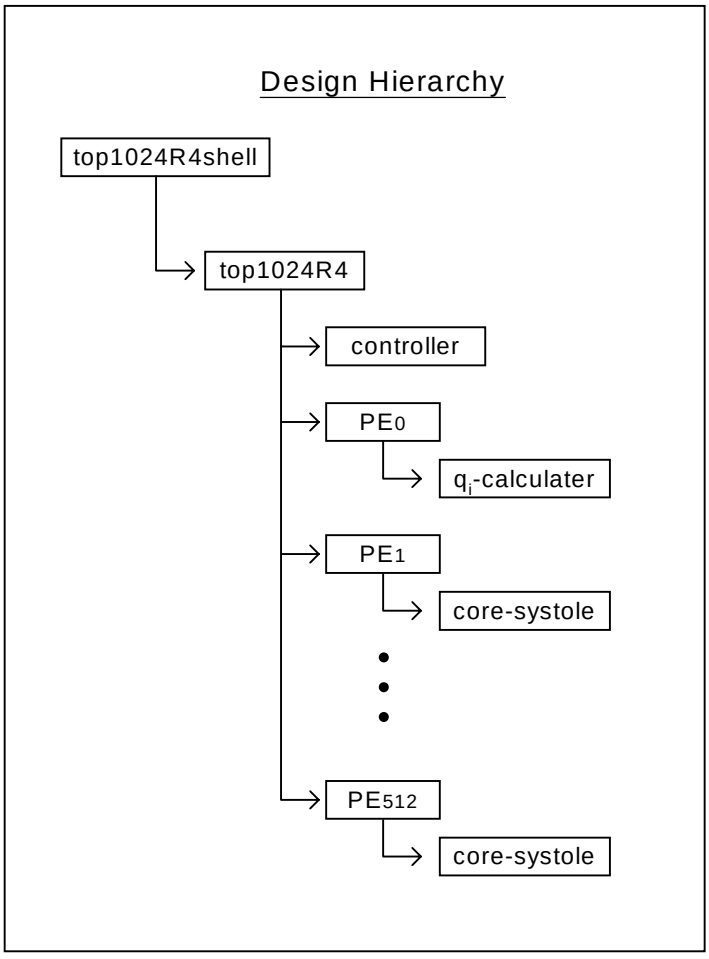


Figure 4.2: Hierarchy of the design.

The first level is used for the chip interface. The I/O interface of the top-module is shown in Figure 4.3. The Modulus (N), Pre-computed Factor (R), and the Message (M) are taken in by 8-bit interface; and the Key is taken in by a 4-bit interface. Data-valid (*dvin*) signal controls the acceptance of these inputs to the system. When *dvin* is at logic level LOW, all the inputs are read at the rising edge of the *clk* input. The 8-bit inputs of the top-module, N, R, and M are registered and transmitted into the second level of the hierarchy in 2-bits thru shift registers. The remaining 4-bits input, e, is registered and transmitted into the second level, using a 1-bit shift register. This operation is shown in Figure 4.4.

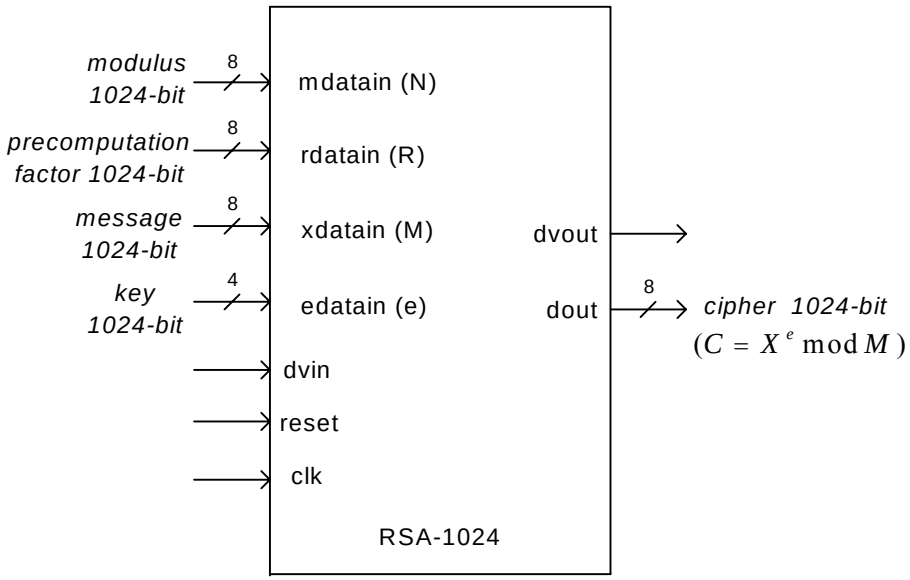


Figure 4.3: I/O interface of the top-module.

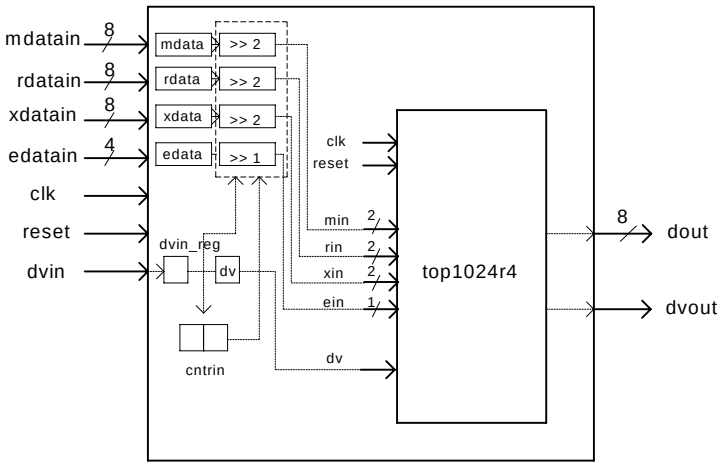


Figure 4.4: Data interface in the 1st level hierarchy.

The output of the RSA Cryptosystem, C, is also a 1024-bit number and exported from the module by an 8-bit interface via an active-LOW data-valid signal named as dvout in the design.

In the second level of the hierarchy, the module "top1024R4" includes a systolic architecture and a controller unit. There exist $n/2+2$ processing elements (PE) for an n-bit operation. The two extra PE's are needed in order not to loose data when overflow occurs in the intermediate steps of the modular multiplication. The right-most processing element, PE_0, in Figure 4.5 represents the least significant digit of the operation and includes the q_i-calculation circuitry, which is the step-a of the used multiplication algorithm in Figure 4.1.

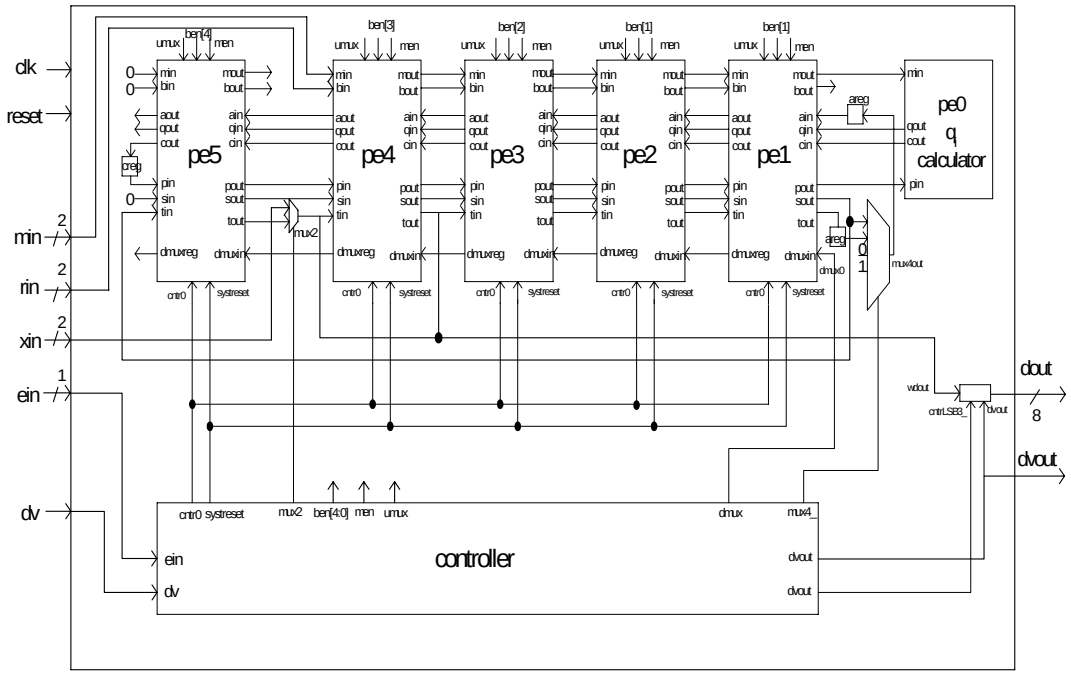


Figure 4.5: A sample systolic architecture for 8-bit RSA.

In the third level, the architecture of the PE's and the function of the controller module are presented. Each PE represents the 2-bits of the RSA operation since radix-4 is used in the design. A PE consists of a core-systole and some logic structures surrounding the core-systole (Figure 4.6). These structures are needed for the input acquisition, storage of some intermediate steps and preparation of the PE's for the next Squaring and Multiplication (S&M).

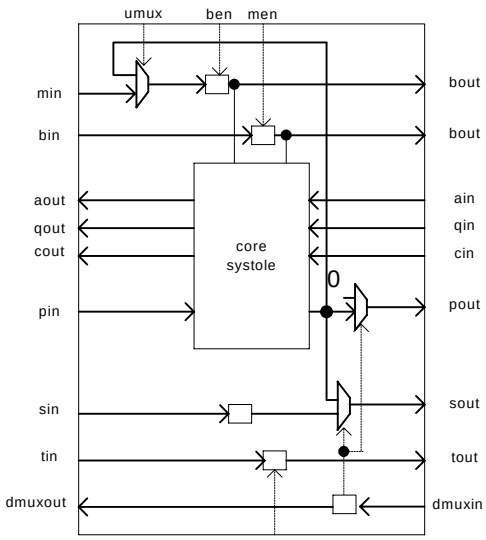


Figure 4.6: The structure of a PE.

The 4th and the last level of the hierarchy is the interior structure of the core-systoles. The main mathematical operation of the Montgomery's Algorithm takes place in this level. All core-systoles have the same architecture except for the one in the right-most PE. This one is different because the q_i-calculation circuitry is included. The two types of the core-systoles are presented in Figure 4.7.

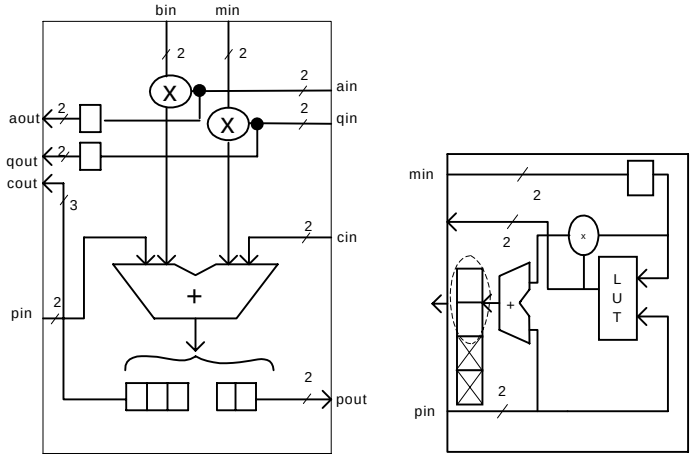


Figure 4.7: The two types of the core-systoles. The LUT module performs the operation $q_i = p_0 \cdot (r - n_0)^{-1} \bmod r$.

4.1.3. The Operation

In each iteration of the multiplication algorithm, once the q_i is computed (step-a) in the right-most PE, all the remaining systoles do the corresponding computation of the step-b by using the same a_i and q_i values, which are shifted through the systoles from right-to-left at each clock cycle. This computation is performed in a pipelined manner, therefore takes two clock cycles for each operation. In other words, at the i^{th} - clock, the j^{th} processing element, PE_j, performs the combinational operation in step-b and gives the outputs to the PE_{j+1} sequentially at the $(i+1)^{th}$ - clock cycle. However, in order to perform the next operation, the PE_j should wait one clock so that the PE_{j+1} performs the step-b operation and gives the outputs back to PE_j at the next clock cycle. This means that each PE states idle for one clock cycle while waiting for the outputs of the nearing PE in the left (Figure 4.8). In the implementation, this inefficiency was avoided by using the idle clocks for another multiplication, which have been mentioned as "parallelism" or "interleaving" in the previous chapters. Since in the R-L Binary Method (step-2a and step-2b in Figure 2.2), there are two modular multiplications (called as squaring and multiplication), which are independent from each other, these two multiplications are interleaved by using the idle clocks, and thus 100% efficiency in the utilization of time and area has been reached. The operation of the PE's is summarized in Figure 4.9.

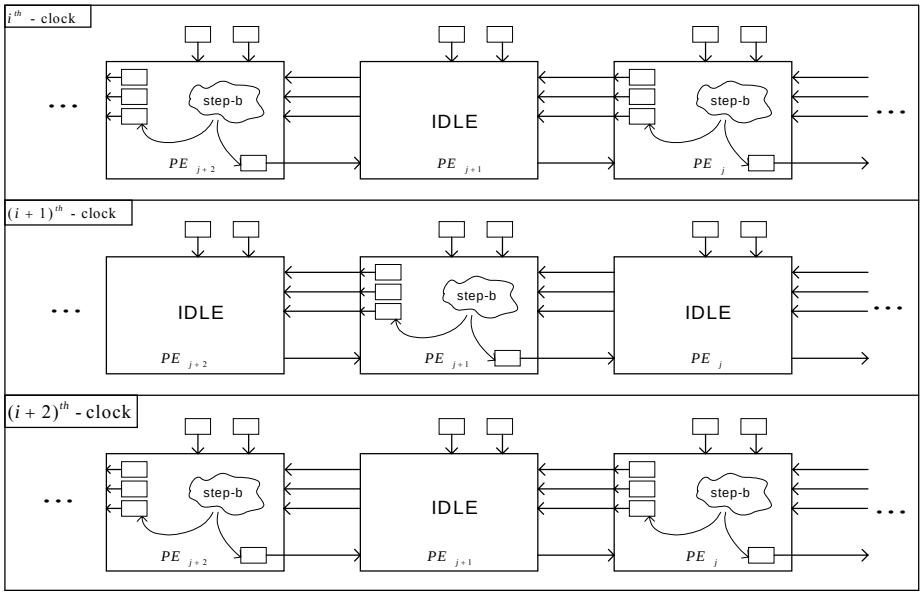


Figure 4.8: Single multiplication without interleaving. Each PE states idle for one clock cycle.

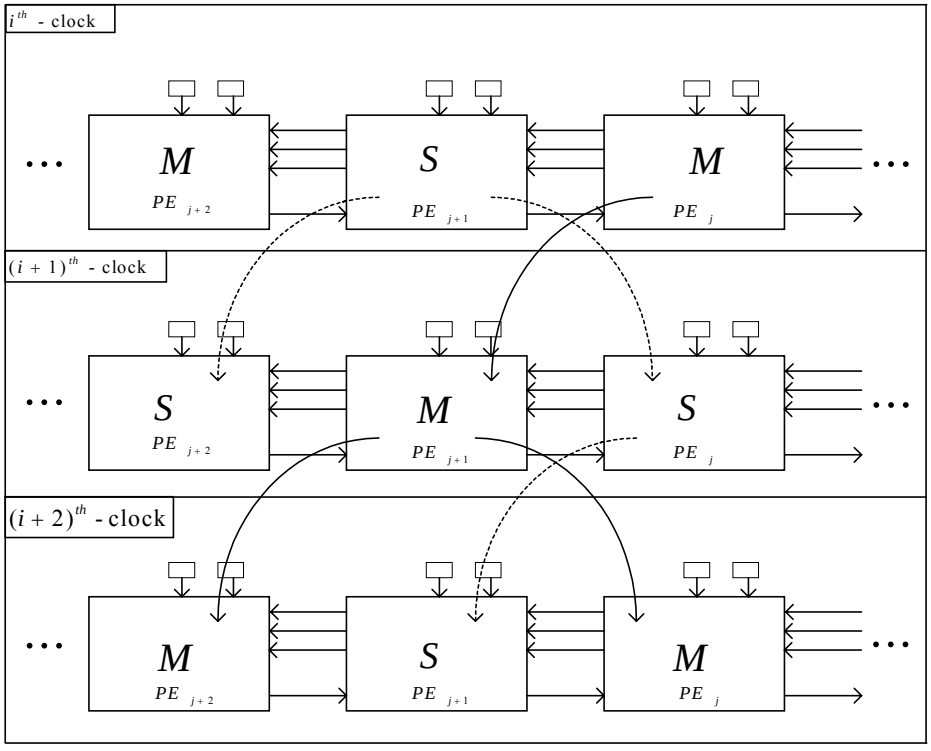


Figure 4.9: Interleaved Squaring and Multiplication operations. 100% utilization of the PE's and no extra clocks for Multiplications. (S: Squaring; M: Multiplication).

Recalling back to the R-L Binary Method (Figure 2.2), it can be seen that the independent operations, step-2a and step-2b, have a common multiplicand. This property of the algorithm provides an efficient utilization of the hardware when interleaving these two independent Squaring and Multiplication (S&M) operations. In this implementation the common multiplicand is placed into the registers at the top of the PE's, and the non-common multipliers are fed digit-by-digit from the right-most PE at successive clocks (Figure 4.10). These digits are actually nothing but the a_i values in the multiplication algorithm in Figure 3.7.

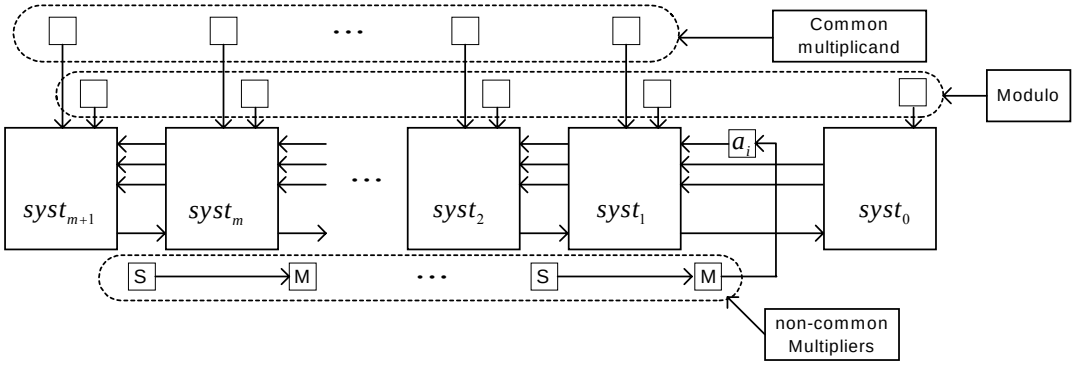


Figure 4.10: Implementation of the R-L Binary Method in the systolic architecture.

Since the outputs of the multiplication are reused as the inputs in the same architecture, a reorganization process of the registers takes place at the end of each S&M. The common multiplicand is the result of the Squaring operation. Therefore, at the end of an S&M, the result of the S, which appears digit-by-digit at the systole outputs from right-to-left, is moved to the upper registers at successive clocks. These upper registers correspond to the B input of the Modular multiplication Algorithm. (The right-most systole doesn't have a B input because of the modification, in which the B input is shifted-up by 4 resulting the least significant digit zero.) Since the modular multiplication algorithm starts from the least significant digit, the new S&M can start just at the next clock at the end of the previous S&M. In other words, the next iteration of the R-L Binary Method starts without waiting the completion of the previous iteration.

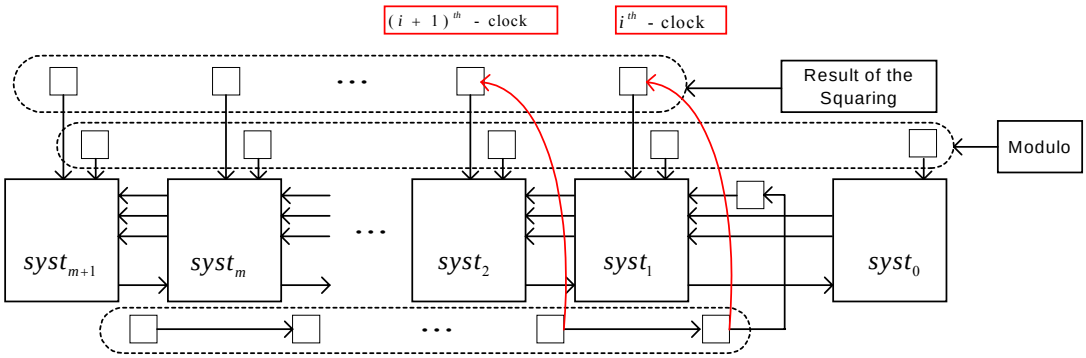


Figure 4.11: The result of the squaring is stored digit-by-digit to be reused in the next Squaring (S) and Multiplication (M). Syst1 starts the new S&M at the $(i+1)^{th}$-clock.

In case of $e_i = 0$ in the R-L Binary Method, only Squaring is performed. The result of the previous multiplication should be stored when this happens. In this architecture, extra registers are used in each PE to provide this storage. Figure 4.12 shows the storage of the Multiplication result digit-by-digit into the newly added registers. Then the stored data is fed back to the systoles as the a_i inputs when a 1 comes from the exponent input.

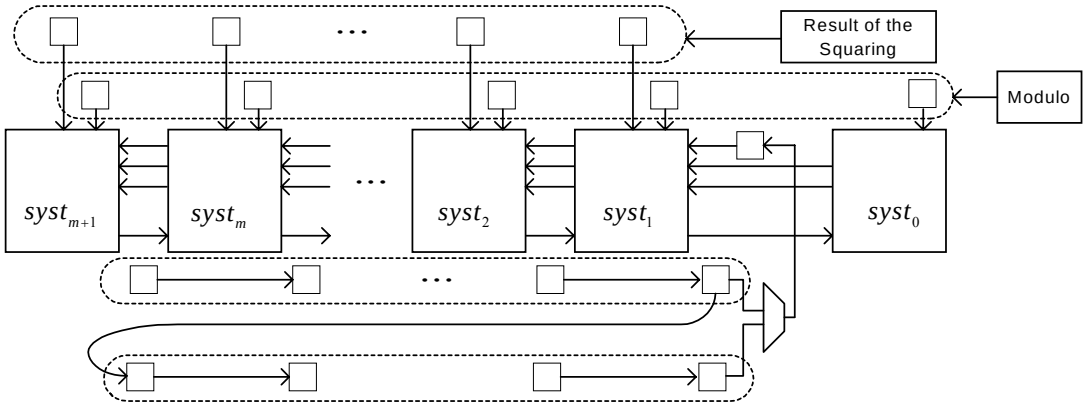


Figure 4.12: Storage of the multiplication result digit-by-digit when the exponent bit is zero.

A state machine in the controller unit operates according to the incoming bits of the exponent and decides whether to store the multiplication result or not. There are 8 states of operation, in which a counter and the incoming bit of the exponent controls the state transition. The counter value tells the end of the present state; and the exponent bit tells the next state to go. The state diagram and the function of the states are given in Figure 4.13 and Table 4.2, respectively. Figure 4.14 illustrates an example for the modular exponentiation process using the state diagram in Figure 4.13.

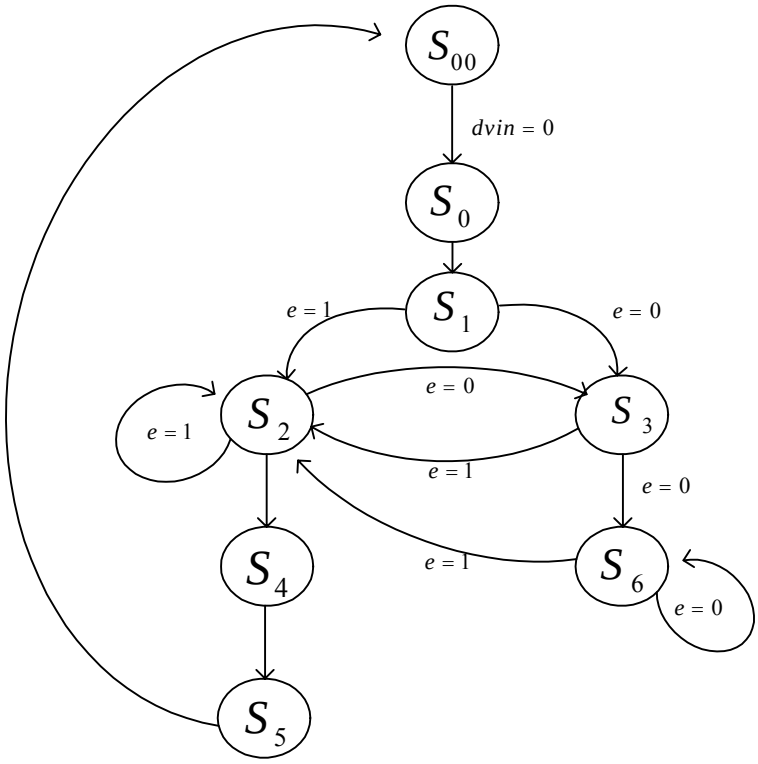


Figure 4.13: The State Diagram.

Table 4.2: Description of the states.

State Names	Description
S_{00}	Idle State. Waits for the dvin LOW.
S_0	Getting the inputs R, M, N, e, and placing them into the shift-registers.
S_1	Pre-multiplication of the inputs by R to obtain the residues.
S_2	Squaring and Multiplication in parallel.
S_3	Squaring continues and the result of the previous Multiplication is being stored.
S_4	Final multiplication by 1 to remove the $r^{-(m+1)}$ factor.
S_5	Outputting the result of the exponentiation
S_6	Only Squaring takes place. The stored value in S_3 remains constant.

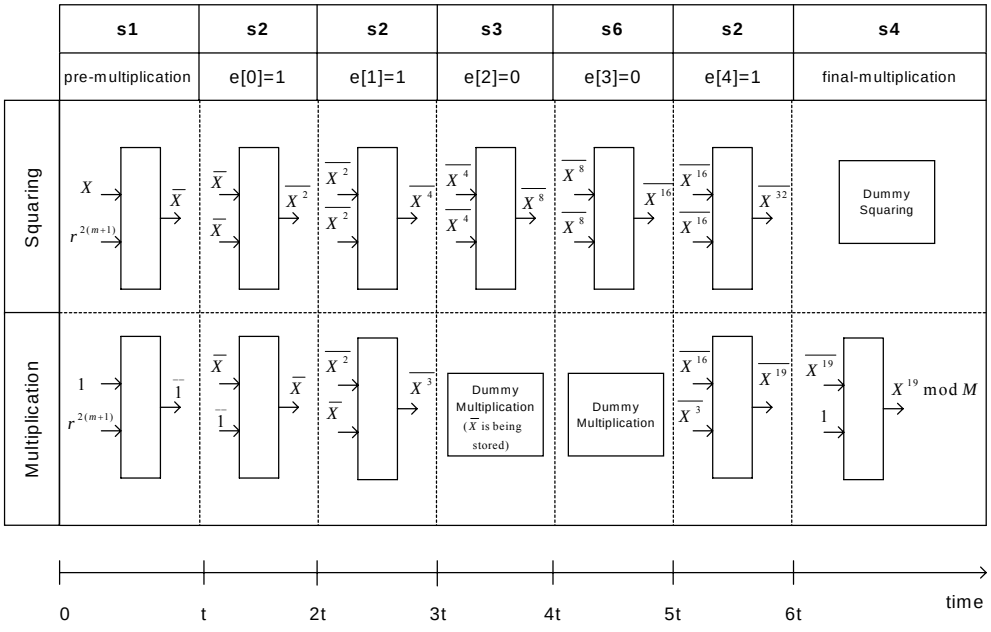


Figure 4.14: Modular Exponentiation example with the corresponding states.

4.2. VLSI Implementation

The design presented here is a Semi-Custom Design, in which the AMI Semiconductor 0.35µm CMOS Standard Cell Libraries are used. This technology accommodates 5 metal layers for routing purposes resulting in chip densities as high as 15K gates per mm^2. The following design tools running on Sun Sparc Work Stations, which were supported by TÜBİTAK-ODTÜ-BİLTEN, were used during the VLSI implementation:

- Synthesis and Timing Analyzing → Synopsys Design Analyzer 1999.10
- Layout (Placement & Routing) → Cadence Silicon Ensemble 5.2
- Simulation → Cadence Affirma Verilog-XL 2.8
- Post-Layout Corrections → Cadence Design Framework II

The following procedure was applied in the implementation:

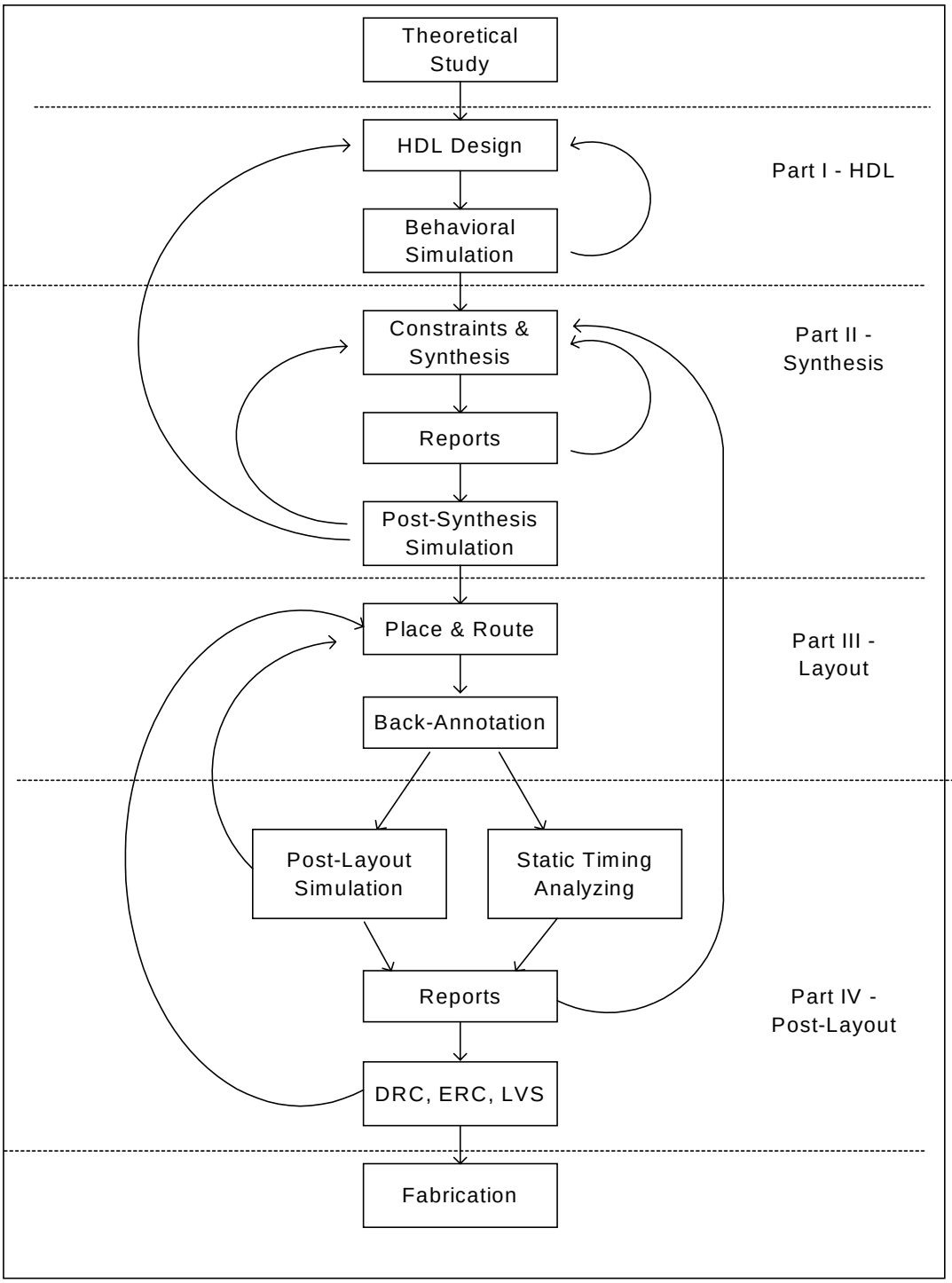


Figure 4.15: VLSI Implementation Procedure.

4.2.1. Design with HDL

As the Hardware Description Language (HDL), VERILOG HDL was used in the design. The need for the VERILOG-HDL was especially appeared in designing the controller unit. Designing the complex structures via HDL provides fast design cycles and gives better results than gate-level designs.

The HDL design was first simulated functionally by using the Cadence Affirma Verilog-XL 2.8 simulation tool. This step of the implementation procedure provided the early detection of any errors, which might cause many problems in the later steps. After being completely sure about the functionality, the next step was the structural design: Synthesis.

4.2.2. Synthesis

For the synthesis of the design, the Synopsys Design Analyzer tool was used. The synthesis process was performed in three steps from bottom to top of the hierarchy.

At first step, an individual PE was synthesized with the following constraints:

- Clock Frequency: 200 MHz
- Operating Conditions: Worst-Case Industrial (MTC45000 WCIND)
- Wire Load: 100-250.
- Timing critical synthesis.

Another synthesis constraint was selection of the Flip-Flops to be used in the design. Some of the Flip-Flops in the MTC45000 library were designed for low power consumption. They had very high set-up and hold times reaching up to 7ns. Therefore, this type of Flip-Flops was eliminated during the synthesis. The results of the above high-effort timing critical synthesis are:

Table 4.3: Synthesis results of an individual PE.

Combinational Area	128.3 gate counts
Non-combinational Area	102 gate counts
Total Area	230 gate counts
Max. Delay	4.98ns

After the synthesis of one PE, the structure was marked as “don’t touch” and thus preserved in the synthesis of the upper level modules. With this method, the tool considers the synthesized PE as a black box and does not overrule the hierarchical structure. Since the whole design consists of many of these PE’s, the tool just copied the synthesized PE’s while synthesizing the upper level modules. This reduced the total synthesis time and preserved the locality of the systoles. Hence, in addition to the regularity and high operating clock frequencies, the systolic architecture provided also fast design cycles.

The second step was the synthesis of the controller unit. The same constraints were applied as in the synthesis of the PE, except for the wire-load model. The controller is a rather larger design than a PE and includes some broadcasting signals to control the PE’s. Therefore a higher wire-load model was needed in the synthesis of the controller. Choosing the appropriate wire-load model is an important issue in such a way that if one chooses a small model below the needs of the architecture, the layout results will probably fail to meet timing constraints because of the large fan-outs and routing problems in the layout. On the other hand, if a high-degree model is to be chosen, then the layout tool may put unnecessary buffers, which increase the chip area and timing. The negative effect of inappropriate wire-load model appears after the layout process. Therefore, about 20 Synthesis-Layout iterations were experienced until reaching to the optimum results

with the wire-load model of 36000-42000 for the controller unit. The synthesis results were as follows:

Table 4.4: Synthesis results of the control unit.

Combinational Area	4713 gate counts
Non-combinational Area	9785 gate counts
Total Area	14498 gate counts
Max. Delay	5.11ns

The third and the final step was the synthesis of the top module. The synthesizer didn't touch to the previously synthesized modules, that is the PE's and the controller unit. The remaining to be synthesized were the I/O interface circuitry and some multiplexers and registers, which existed in the same level with the PE's and the controller. The results of the synthesis of the total design are in Table 4.5.

Table 4.5: Synthesis results of the whole design with I/O pads.

Combinational Area	84232 gate counts
Non-combinational Area	68644 gate counts
Total Area	152876 gate counts (with I/O pads)
Max. Delay	6.69ns

4.2.3. Layout

Reaching the desired results in the synthesis, the synthesized design is exported as VERILOG NETLIST to the Cadence Silicon Ensemble tool for the placement and routing. This process includes the following steps:

1) Initialize floorplan: Floorplan dimensions, and some floorplan options are selected in this step before the placement starts. Also a decision on the row utilization is made. A 95% row utilization was preferred in the design in order to provide some area for the clock distribution process.

2) Placing I/O's: The designed chip includes total of 68 pins:

- 31 input pins (8 pins for each message, pre-computation number, and modulo; 4 pins for exponent; 1 pin for each clock, reset data valid signals),

- 9 output pins (8 pins for output data; 1 pin for output data valid signal),

- 28 power pins (14-VDD, 14-VSS pins).

3) Place cells: The cells identified in the synthesis are placed automatically into the floorplan in this step. The tool performs the placement by taking account the timing constraints, which are imported at the beginning of the layout process, as a constraint file.

4) Generate clock tree: As in the timing driven placement of the cells, the clock distribution process is also performed according to the imported constraint file. In this step, the user specifies the maximum skew value and maximum delay on the clock signal. The tool generates a clock tree by inserting some buffers where needed from the imported design library.

5) Place filler cells: After the newly inserted buffers in the previous step, the empty spaces must be filled. Filler cells are dummy structures used to

provide the continuity of design layers underneath the routing layers. They have various dimensions and are used both in filling the spaces between the I/O pads and the core cells.

6) Add power rings: Two rings of metals are placed surrounding the core, one for VDD and the other for VSS.

7) Connect rings: Power pins of the cells are shorted to the two rings created in the previous step.

8) Global route: The tool auto-routs all the nets in the design in this step.

9) Verify: In this step, a final verification of the connectivity, geometry, and antenna effect after the global routing of the whole chip is done.

10) Back-annotation: As the last step of the layout, the Standard Design Format (SDF) file is produced from the layout. This file includes triplet routing delay values (max, min, and typical delays) calculated separately for maximum and minimum type path delays using the layout data. This file carries the routing information and is used in the post-layout simulations and post-layout timing analysis of the design.

4.2.4. Post-layout Work

In addition to the back-annotation (generation of the SDF file), a VERILOG netlist of the layout is also extracted to be used in the post-layout simulation and static timing analyzing. The design is simulated in Cadence Affirma Verilog-XL 2.8 simulation tool by annotating the SDF file into the VERILOG netlist. By introducing the SDF delay information, the Static Timing Analyzing was performed using the Design Analyzer. The results of the timing analyzing are given below:

Table 4.6: Static timing analysing results (with I/O pads and routing delays) after layout.

Combinational Area	84515 gate counts
Non-combinational Area	68644 gate counts
Total Area	153159 gate counts (with I/O pads)
Max. Delay	3.97ns

For the power analysis of the designed chip, an estimation can be made based on the formula given below:

$$P_{estimated} = P_{ND2} \times N \times f \times S,$$

where:

$P_{ND2} =$ Power dissipation in a 2 input NAND gate in μWatts/MHz.

$N =$ Total number of gates in the design.

$f =$ Operating frequency in MHz.

$S =$ Percentage of switching gates at a given time.

With $P_{ND2} = 0.0848$ μW/MHz, and taking $S = 30\%$, and $f = 250MHz$, the worst-case power can be estimated as:

$$\begin{aligned} P_{estimated} &= 0.0848\ \mu W / MHz \times 153159 \times 250\ MHz \times 30/100 \\ &\cong 974mW \end{aligned}$$

This is a high power value for a single chip, even though it is a worst-case estimate. However, low power design is not in the scope of the research presented in this thesis. Moreover, it is possible to lower the power dissipation by easily moving the design to technologies with smaller feature sizes and lower supply voltages, when commercial products are to be designed.

After verification of the operation in the post-layout simulation and obtaining the desired performance in the timing analyzer, the GDSII stream file was produced from the SILICON ENSEMBLE. This file is required for design submission and for exporting the design into Cadence for post-layout checks (Design Rule Check (DRC), Electrical Rule Check (ERC), and Layout Versus Schematic (LVS)) of the design.

The design has been sent for fabrication to EUROPRACTICE IC SERVICE as a single-chip product. The fabrication will be performed by using the technology AMI Semiconductor 0.35 µm CMOS C035M-D (5M/1P).

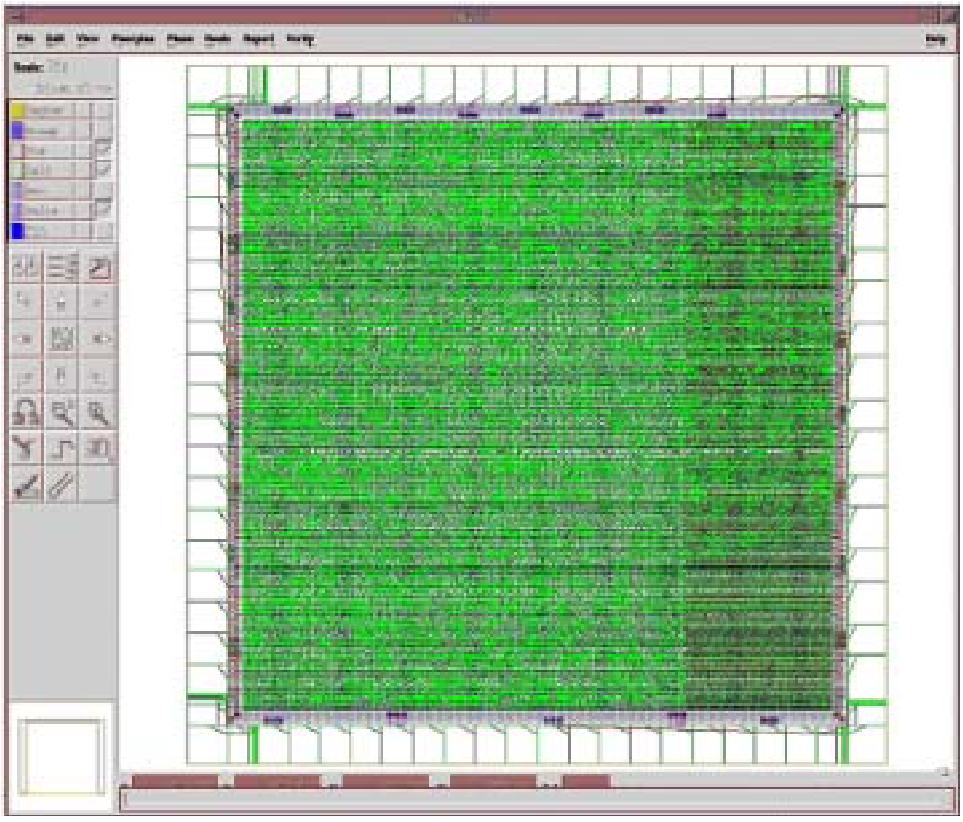


Figure 4.16: The layout produced via Cadence Silicon Ensemble.

Figure 4.17: Zoomed view on the left-bottom corner of the layout of Figure 4.16.

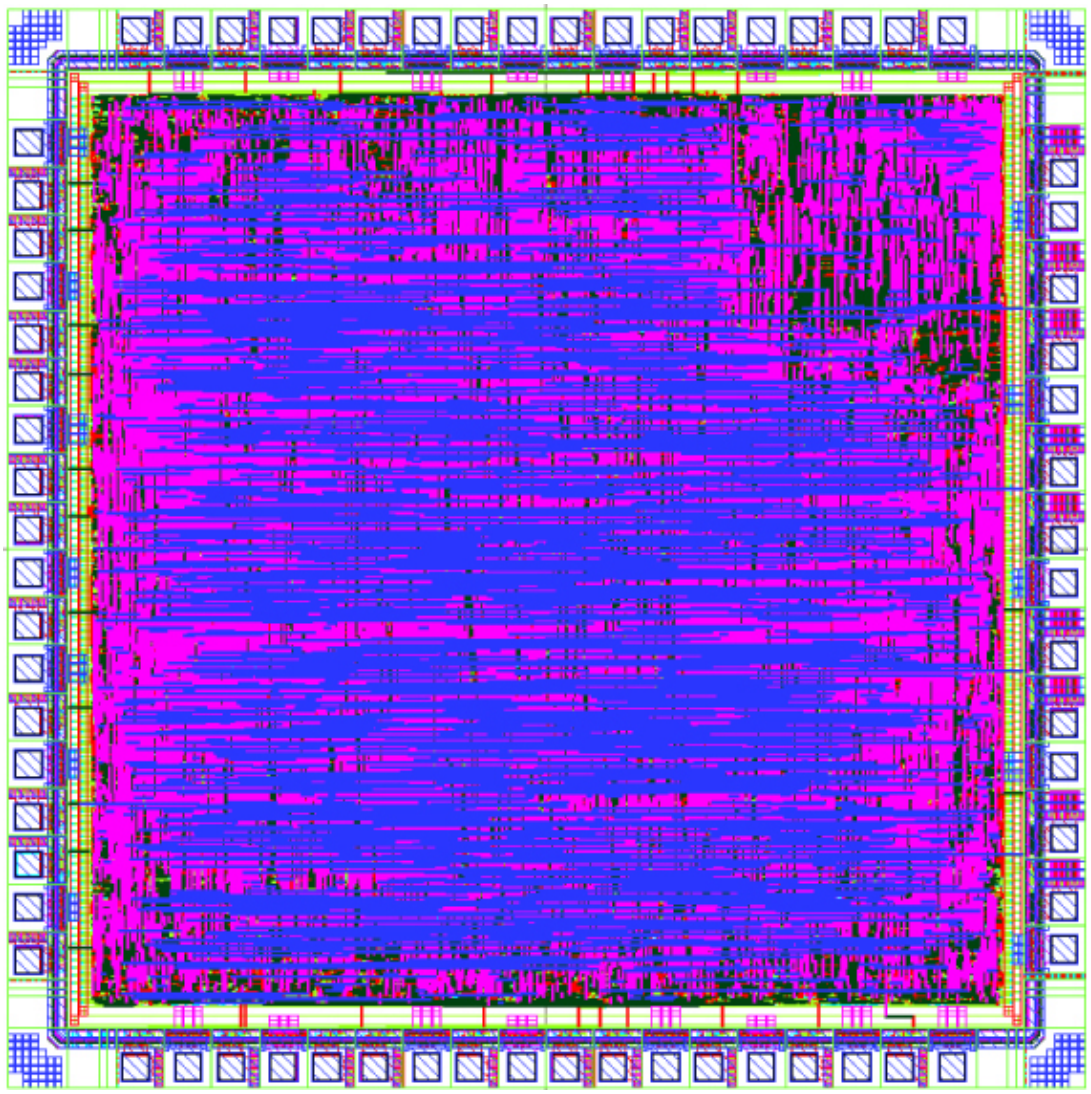


Figure 4.18: Layout view in Cadence Design Framework II.

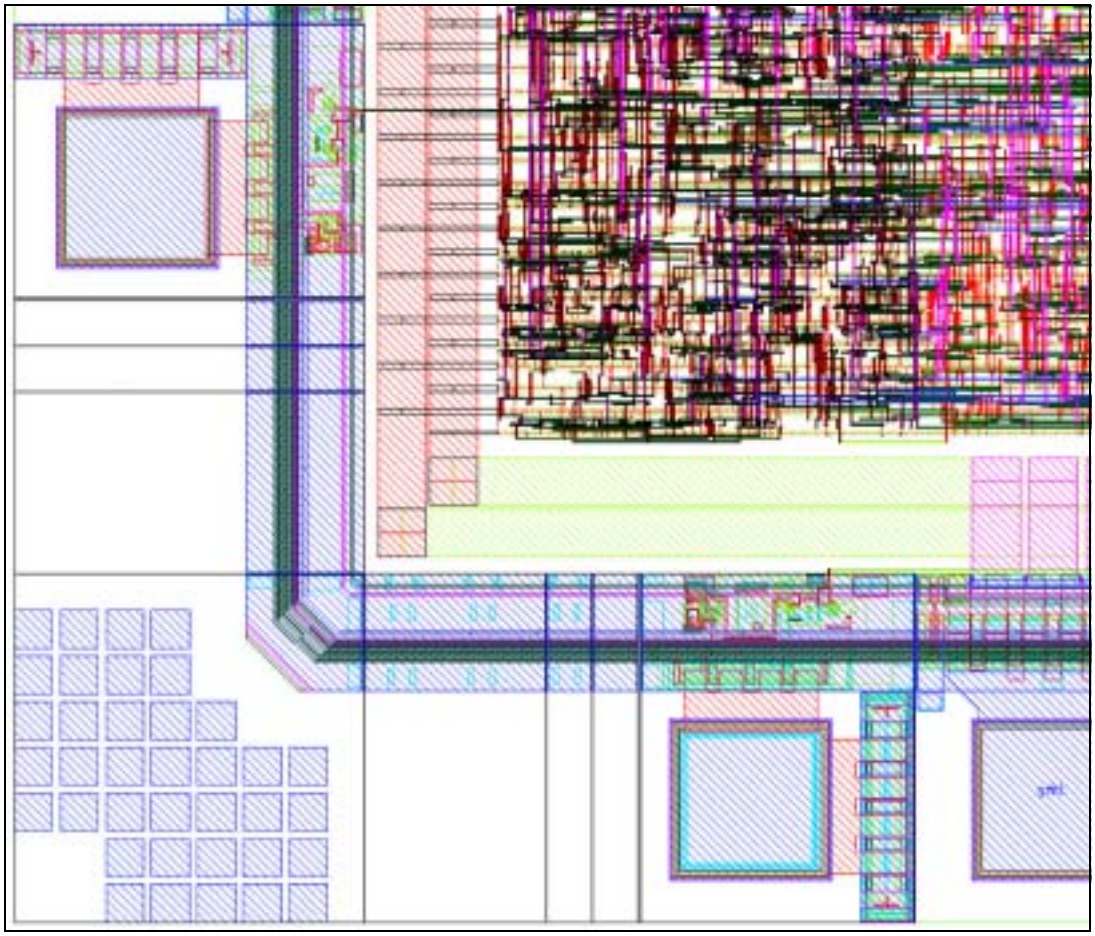


Figure 4.19: Zoomed view on the left-bottom corner of the layout of Figure 4.18.

4.3. FPGA Implementation

The Field Programmable Gate Array (FPGA) is an integrated circuit that contains many identical logic cells that can be viewed as standard components. The individual cells are interconnected by a matrix of wires and programmable switches. A design is implemented on an FPGA by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix.

Design with the FPGA's provides many facilities such as faster design cycles, simpler and less expensive realizations of the designs compared to VLSI implementations. On the other hand, designing with an FPGA has also some drawbacks: They are area inefficient and slower when compared to VLSI implementations. Therefore, VLSI is still prefferred for large and speed critical designs. However, FPGA technology is developing with an increasing speed and reducing the above disadvantages.

The design presented in the previous section was implemented on the FPGA for the real-time verification. Therefore, a high degree of optimization was not performed while implementing on the FPGA. The following tools supported by TÜBİTAK-ODTÜ-BİLTEN were used for this implementation:

- Synthesis & Implementation → Xilinx Project Navigator 5.1i.
- Real-time Test → Celoxica RC1000 Hardware on a PC.

4.3.1. Synthesis and Layout

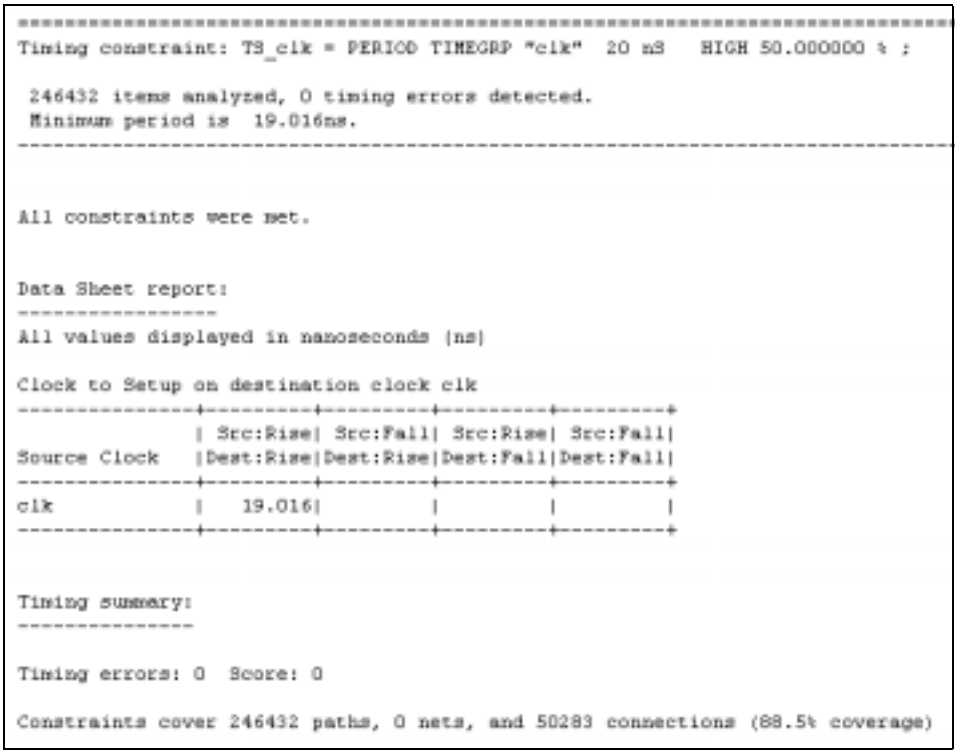
Since RC1000 hardware includes Xilinx Virtex2000E FPGA, the design was synthesized and implemented for this FPGA with a timing constraint of 20ns clock period (50MHz). Using the Xilinx Project Navigator 5.1i tool for this process, the following results were obtained:

```
=========================================================
HDL Synthesis Report

Macro Statistics
# FSMs                                  : 1
# Registers                             : 5672
  1-bit register                        : 1549
  2-bit register                        : 3597
  8-bit register                        : 7
  4-bit register                        : 2
  11-bit register                       : 2
  512-bit register                      : 1
  3-bit register                        : 514
# Counters                              : 2
  11-bit up counter                     : 1
  2-bit up counter                      : 1
# Multiplexers                          : 1031
  2-to-1 multiplexer                    : 1031
# Adders/Subtractors                    : 1541
  11-bit adder                          : 1
  4-bit adder carry out                 : 514
  5-bit adder                           : 513
  3-bit adder carry out                 : 513
# Multipliers                           : 1027
  2x2-bit multiplier                    : 1027
# Comparators                           : 2
  11-bit comparator equal               : 1
  11-bit comparator not equal           : 1

=========================================================
```

Figure 4.20: A captured view form FPGA synthesis report.

```
Constraints file: top1024R4shell.pcf

Loading device database for application par from file "top1024R4shell_map.ncd".
   "top1024R4shell" is an NCD, version 2.37, device xcv2000e, package bg560,
speed -6
Loading device for application par from file 'v2000e.nph' in environment
C:/Xilinx.
Device speed data version:  PRODUCTION 1.68 2002-06-19.


Device utilization summary:

   Number of External GCLKIOBs          1 out of 4      25%
   Number of External IOBs             39 out of 404     9%
   Number of LOCed External IOBs        0 out of 39      0%
   Number of SLICEs                 10634 out of 19200   55%
   Number of GCLKs                      1 out of 4      25%



Overall effort level (-ol):   2 (set by user)
Placer effort level (-pl):    2 (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl):    2 (set by user)

Starting initial Timing Analysis.  REAL time: 8 secs
Finished initial Timing Analysis.  REAL time: 21 secs
```

Figure 4.21: A view from FPGA map report.



```
Timing constraint: TS_clk = PERIOD TIMEGRP "clk"  20 nS   HIGH 50.000000 % ;

 246432 items analyzed, 0 timing errors detected.
 Minimum period is  19.016ns.
--------------------------------------------------------------------------------


All constraints were met.


Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Clock to Setup on destination clock clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clk            |   19.016|         |         |         |
---------------+---------+---------+---------+---------+


Timing summary:
---------------

Timing errors: 0  Score: 0

Constraints cover 246432 paths, 0 nets, and 50283 connections (88.5% coverage)
```

Figure 4.22: A view from FPGA post-layout timing report.

4.3.2. Real-time Test on FPGA

The design was tested in real-time using the Celoxica RC1000 Hardware accomodating the Xilinx V2000E FPGA. RC1000 is a PCI bus plug-in card (Figure 4.24) for PC's, consisting of one large XILINX FPGA (BG560 package), four banks of memory for data storage, and two PMC sites for I/O with the outside world [42]. Memory banks can be accessed by both FPGA and PCI bus. The card is controlled by PC through PCI bus by running executables written in Handle-C Programming Language.

In the test of the FPGA implementation, the steps given below were followed (Figure 4.23):

1) A RAM interface design was included in the FPGA with the RSA module. This interface enables the module to access the RAM's on the RC1000 hardware.

2) A C executable was used to access the RAMs and the FPGA. This exe does the following:

- Places the test data into the RAMs,
- Sets the FPGA clock,
- Configures the FPGA with the configuration file obtained from Xilinx Project Navigator,
- Releases the control of the RAM banks and resets the FPGA.
- After finishing the RSA operation, FPGA writes the output into the RAM. Then PC again takes over the RAM control, reads the output of the FPGA and writes it into a file.

3) The RSA operation was verified by observing the output in hexadecimal format using simply a text editor. A sample tested encryption and decryption is shown in Figure 4.26 and Figure 4.27.

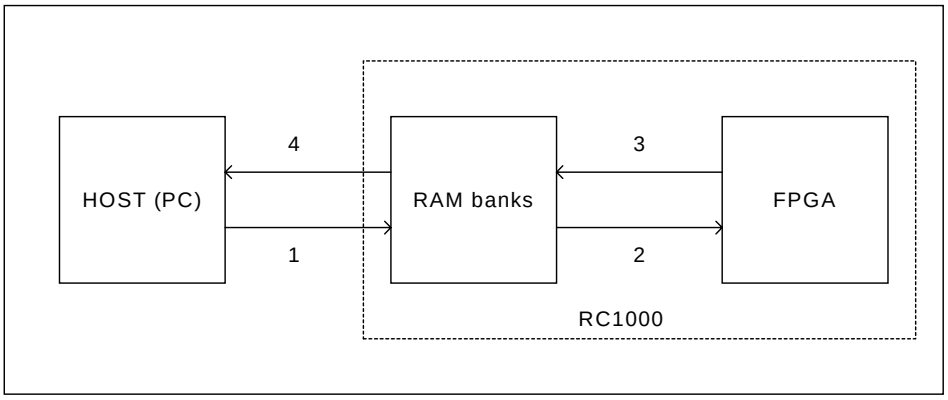


Figure 4.23: Data flow in the test of the FPGA with RC1000.

1) PC writes the test data into the RAM. 2) FPGA reads the inputs from the RAM. 3) FPGA writes the RSA output into the RAM. 4) PC reads from RAM and writes the FPGA outputs into a file.

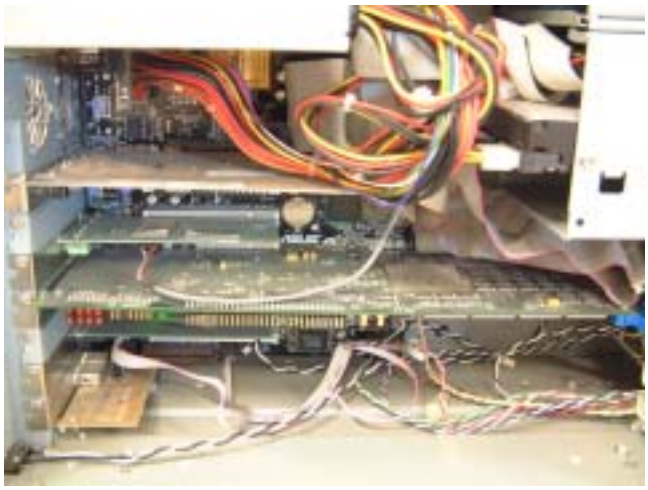


Figure 4.24: Test Setup. (RC1000 Board mounted to the main-board of a PC).

Figure 4.25: Zoomed view on the Xilinx V2000E FPGA on the RC1000 board.

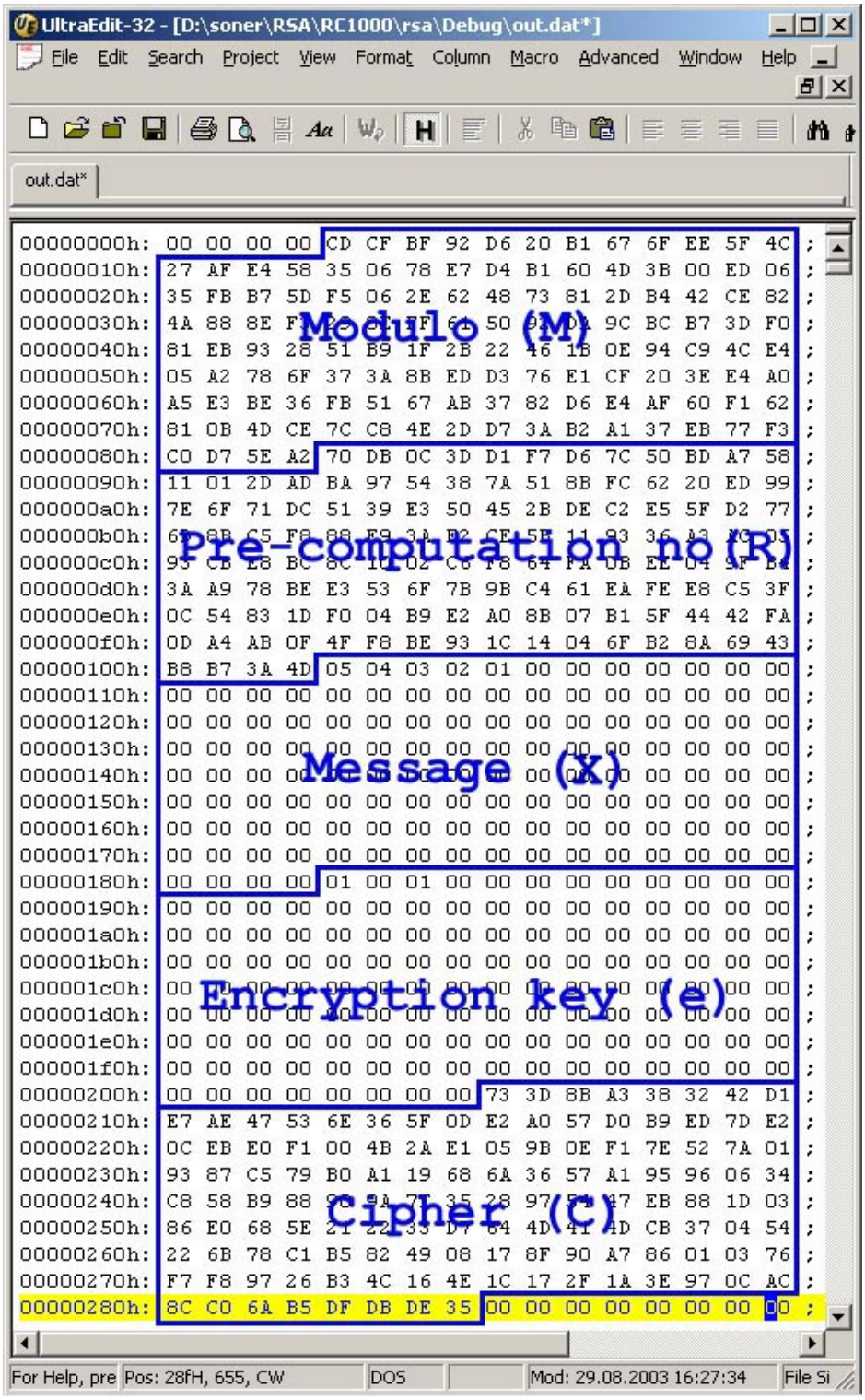


Figure 4.26: Encryption example in the FPGA test:

Inputs: M, R, X, e; Output: $C = X^e \bmod M$;

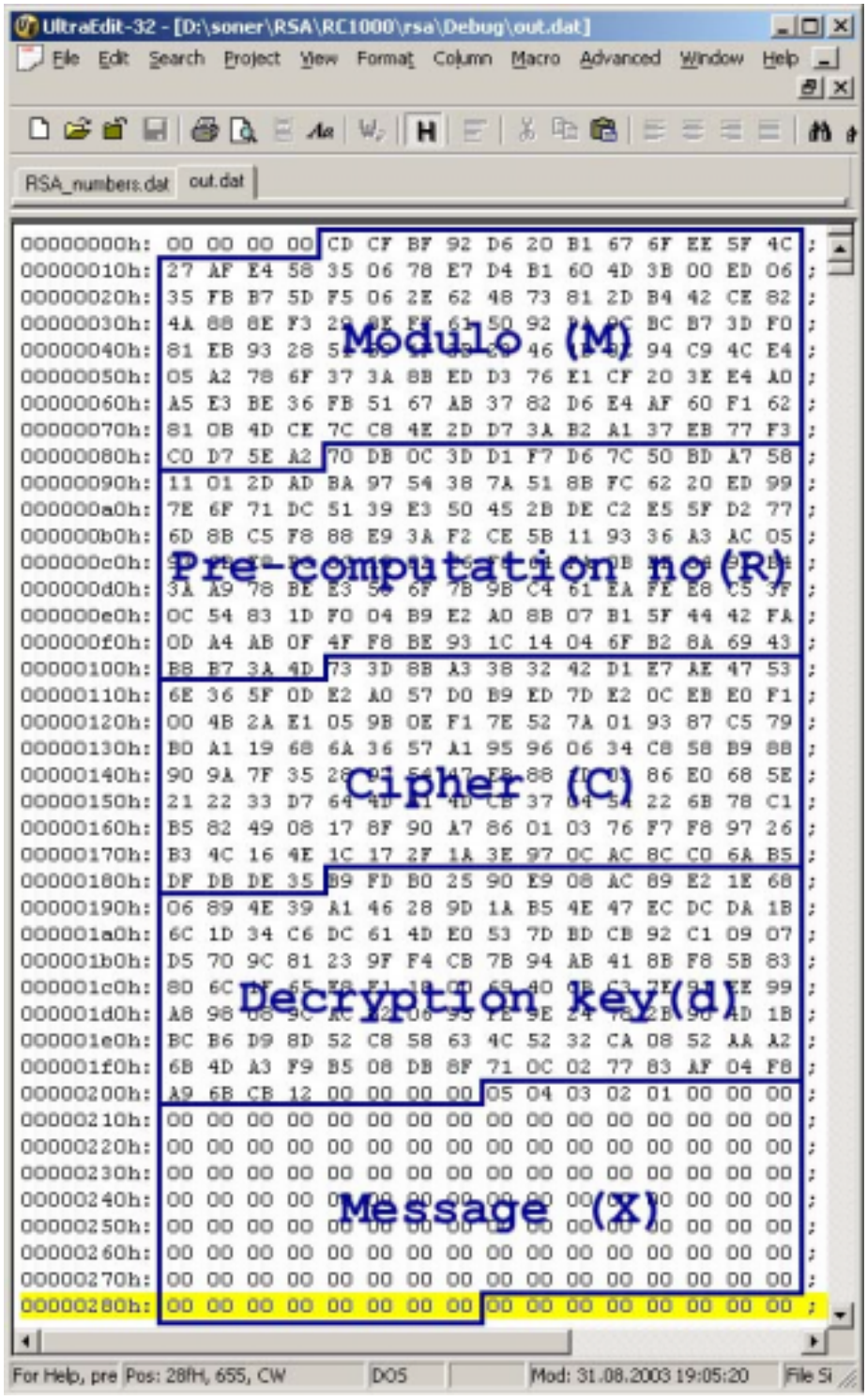


Figure 4.27: Decryption of the Cipher Text in Figure 4.23. 1024-bit RSA is verified in the FPGA by recovering the Message. Inputs: *M, R, C, d*; Output: $X = C^d \bmod M$;

4.4. Comparison of the results

The purpose of this section is to provide a quick reference for the reader to examine the implementation results presented in this thesis and compare them to the previous ones in the literature. The results of the RSA implementations, which were given in Chapter 3, are summarized in Table 4.7.

Table 4.7: Comparison of the VLSI implementations.

Paper &Year	No of bits (n)	Tech	Gate count	Chip area	No. of clocks	Clock Freq. (MHz)	Op. time (msec)	Baud (Kb/s)
Chen[13] 1996	512	$0.8\mu m$	78K	$76mm^2$	1.05M $(\sim 4n^2)$	50	20.6	24.3
Yang[14] 1996	512	$0.6\mu m$ Compass SPDM	74K	$56mm^2$	0.54M $(\sim 2n^2)$	125	4.24	118
Royo[15] 1997	768	$0.7\mu m$ ES2	-	$77mm^2$	0.5M $(\sim n^2)$	50	10.4	72.5
Guo[16] 1999	512	$0.6\mu m$ Compass	132K	$68mm^2$	0.258M $(\sim n^2)$	143	1.8	278
Leu[20] 1999	512	$0.6\mu m$	64K	-	0.53M $(\sim 2n^2)$	115	4.5	111
Chiang[23] 1999	512	Compass $0.6\mu m$ TSMC	-	-	0.7M $(\sim (4n/3)(2n))$	166	4.09	122
Kim[25] 2000	1024	$0.65\mu m$ SOG	112K	-	2.2M $(\sim 2n^2)$	50	43	23.25
Kwon[31] 2001	1024	$0.5\mu m$ SOG	92K	-	2.2M $(\sim 2n^2)$	50	43	23.25
Kwon[31] 2001	1024	$0.5\mu m$ SOG	156K	-	1.1M $(\sim n^2)$	50	22	45.45
Cho[32] 2001	1024	-	230K	-	0.527M $(\sim n(n/2+3))$	40	13	78.8
Our design 2003	512	$0.35\mu m$ AMIS	87K (Without I/O pads)	$8.7mm^2$	0.265M $(\sim n(n+4))$	333	0.8	627
Our design 2003	1024	$0.35\mu m$ AMIS	132K (Without I/O pads)	$10.4mm^2$	1.05M $(\sim n(n+4))$	250	4.22	237

CHAPTER 5

CONCLUSION

This thesis presented a high-speed ASIC implementation of the RSA Public-Key Cryptosystem. The implementation basically performs the modular exponentiation of large integers, which is the main operation used for both encryption and decryption in RSA. The R-L Binary Method and Montgomery's Multiplication Algorithm in radix-4 were combined in linear systolic architecture with a state machine for the modular exponentiation operation, which is main mathematical operation of the RSA. A semi-custom VLSI implementation was performed for both 512-bit and 1024-bit processes by using the AMI Semiconductor $0.35\mu m$ Standard Cell Libraries.

The results obtained in this implementation were: 87K gate count and 627Kb/s baud at 3ns worst-case clock for the 512-bit operation; 132K gate count and 237Kb/s baud at 4ns worst-case clock for the 1024-bit operation. In addition to the VLSI implementations, a real-time test of the hardware was performed at a clock speed of 80MHz by using the Celoxica RC1000 Hardware with Xilinx V2000EBG560 FPGA on it. With these results, the fastest RSA processor and the lowest $area \times time$ product within our knowledge in the literature was obtained in the literature within this thesis. There are three main reasons underlying the effective results of the proposed implementation: Properly chosen algorithms and optimizations on these algorithms, minimized routing delays with the linear systolic

architecture, and finally the AMIS 0.35μm CMOS technology used in the implementation.

By interleaving the Squaring and Multiplication steps of the R-L Binary Method at consecutive clock cycles, a 100% utilization of the time and resources have been achieved in the design. This method also reduced the total number of clocks without any need of extra hardware. Using radix-4 calculations in the Montgomery's algorithm was another factor reducing the number of clocks. In addition to this, some other simplifications to reduce the critical path were applied in the algorithm.

The design was fitted into a linear systolic architecture, in which a series of identical structures are brought together and communicate locally at high clock frequencies. The result of this structure was the minimized number of broadcasting signals in the architecture, thus very high clock speeds. Also, a controller unit in the architecture managed the resources in the systoles in such a way that the systolic architecture performed a continuous operation throughout the exponentiation, without any need of extra storage elements for the intermediate results of the exponentiation process.

The CMOS 0.35μm technology was another important factor in achieving the above results in the implementation. Although the systolic architecture played the main role in obtaining high clock speeds, the technology used in the design had also great contributions to the timing of the design by providing very high-speed logic cells. The high density of the technology also contributed to the mapping of 150K gates into an area of about $10mm^2$.

In conclusion, with the above results, the goals at the beginning of the thesis have been achieved and the 1024-bit VLSI implementation has been sent to IMEC for fabrication as a prototype chip. However, this study can be moved further. Newer technologies will provide implementing high radix operations at faster and smaller architectures. This will result in a less number of clocks, and thus a faster operation. As another future work, a scalable architecture, in which it will be

possible to perform 2048 and 4096 bit RSA operations with a multiple of the 1024-bit chips in serial, will be implemented. This will bring flexibility to the user in choosing the level of security in the application.

REFERENCES

[1] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. on Information Theory,* vol. IT-22, pp. 644-654, November 1976.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM,* vol. 21, pp. 120-126, February 1978.

[3] E. F. Brickell, "A survey of Hardware Implementations of RSA," In G. Brassard, editor, *Advances in Cryptology, Crypto 89, Proceedings, Lecture Notes in Computer Science,* No. 435, pp. 368-370, New York, NY, Springer-Verlag, 1989.

[4] Ç. K. Koç, "High-Speed RSA Implementation," *Technical Report,* RSA Laboratories, RSA Data Security, Inc., pp. 46-49, 1994.

[5] D. Stinson, "Cryptography: Theory and Practice," *CRC Press LLC,* March 1995.

[6] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, 31(4), pp.469-472, July 1985.

[7] National Institute for Standards and Technology. Digital signature standard (DSS). *Federal Register*, pp. 56-169, August 1991.

[8] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computations,* vol. 44, pp. 519-521, 1985.

[9] H. Orup and P. Kornerup, "A High-radix Hardware Algorithm for Calculating the Exponential M^E Modulo N," *10-th IEEE symposium on COMPUTER ARITHMETIC*, pp. 51-57, 1991.

[10] C. Walter, "Systolic Modular Multiplication," *IEEE Transactions on Computers,* vol. *42(3)*, pp. 376-378, March 1993.

[11] C. Walter, "Still Faster Modular Multiplication," *Electronic Letters,* vol. 31, pp. 263-264, February 1995.

[12] T. Acar, B. S. Kaliski Jr., and Ç.K. Koç, "Analyzing and Comparing Montgomery Modular Multiplication Algorithms," *IEEE Micro*, vol. 16(3), pp. 26-33, June 1996.

[13] P. S. Chen, S. A. Hwang, and C. W. Wu, "A Systolic RSA Public Key Cryptosystem," *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, (Atlanta), pp. 408-411, May 1996.

[14] C. C. Yang, C. W. Jen, and T. S. Chang, "The IC Design of A High Speed RSA Processor," *Proceeding of IEEE Asia Pacific Conference on Circuits and Systems,* Seoul, Korea, pp. 18-21, November 1996.

[15] A. Royo, J. Moran, and J. C. Lopez, "Design and implementation of a coprocessor for cryptography applications," *Proceeding European Design and Test Conference*, pp. 213-217, Paris 1997.

[16] J. H. Guo, C. L. Wang, and H. C. Hu, “Design and Implementation of an RSA Public-Key Cryptoystem,” *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems*, Orlando, FL, pp. 504-507, May 30 - June 2, 1999.

[17] C. C. Yang, T. S. Chang, and C. W. Jen, “A New RSA Cryptosystem Hardware Design Based on Montgomery's Algorithm,” *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing,* vol. 45, no. 7, July 1998.

[18] J. H. Guo and C. L. Wang, “A novel digit-serial systolic array for modular multiplication,” *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, Monterey, CA, pp. 177-180, May 31 - June 3, 1998.

[19] C. D. Walter, “Montgomery exponentiation needs no final subtraction,” Electronics Letters, vol.35, no.21, pp. 1831-1832, October 1999.

[20] J. J. Leu and A. Y. Wu, “A Scalable Low-Complexity Digit-Serial VLSI Architecture for RSA Cryptosystem,” *Proc. IEEE Workshop on Signal Processing Systems (SiPS-99)*, Taipei, pp. 586-595, October 1999.

[21] C. Y. Su, S. A. Hwang, P. S. Chen, and C. W. Wu, “An Improved Montgomery's Algorithm for High-Speed RSA Public-Key Cryptosystem,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol.7, no.2, pp. 280-284, June 1999.

[22] T. Blum, and C. Paar, “Montgomery Modular Exponentiation on Reconfigurable Hardware,” *Proceedings of the 14th IEEE Symposium on Computer Arithmetic, Adelaide,* , pp. 70-77, April 1999.

[23] J. S. Chiang and J. K. Chen, "An Efficient VLSI Architecture for RSA Public-Key Cryptosystem,” *ISCAS99,* vol. 1, pp. 496-499, 1999.

[23] M. D. Shieh, C. H. Wu, M. H. Sheu, and C. H. Wu, "A VLSI Architecture of Fast High-Radix Modular Multiplication for RSA Cryptosystem," *IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 500-503, May 1999.

[24] Y. S. Kim, W. S. Kang, and J. R. Choi, "Implementation of 1024-bit modular processor for RSA cryptosystem," *Proceedings of Asia-Pasific Conference on ASIC (AP-ASIC), Cheju Island, Korea*, August 2000.

[25] C. D. Walter, "Improved Linear Systolic Array for Fast Modular Exponentiation," *IEEE Computers and Digital Techniques*, vol. 147, no. 5, pp. 323-328, September 2000.

[26] W. C. Tsai, C. B. Shung, and S. J. Wang, "Two Systolic Architectures for Modular Multiplication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol.8, no.1, pp. 103-107, February 2000.

[27] N. Tomabechi and T. Ito, "Design of a High-Speed RSA Encryption Processor with Built-in Table for Residue Calculation of Redundant Binary Numbers," *ISCAS 2000-IEEE International Symposium on Circuits and Systems, Geneva, Switzerland,* vol. 5, pp. 697-700, May 2000.

[28] J. H. Hong and C. W. Wu, "Radix-4 Modular Multiplication and Exponentiation Algorithms for the RSA Public-Key Cryptosystem," *Proceedings on the 2000 Conference on Asia and South Pacific Design Automation*, *Yokohama, Japan,* pp. 565-570, January 2000.

[29] A. Daly and W. Marnane, "Efficient Architectures for Implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic," *FPGA'02, Monterey California, USA,* pp. 40-49, February 2002.

[30] T. W. Kwon, C. S. You, W. S. Heo, Y. K. Kang, and J. R. Choi. "Two implementation methods of a 1024-bit RSA cryptoprocessor based on modified montgomery algorithm," *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, pp. 650-653, May 2001.

[31] K. S. Cho, J. H. Ryu and J. D. Cho, "High-Speed Modular Multiplication Algorithm for RSA Cryptosystem," *IECON*, pp. 479-483, 2001.

[32] V. Bunimov, M. Schimmler, and B. Tolg, "A Complexity-Effective Version of Montgomery's Algorithm," Workshop on Complexity Effective Designs (WCED02), May 2002.

[33] K. Sakiyama and S. Kim, "An FPGA Implementation and Performance Evaluation of Modular Multiplication Operation for RSA Cryptography Algorithm," http://www.cs.ucla.edu/~milos/PROJ02/KSreport.pdf

[34] H. Handschuh and P. Paillier, " Smart Card Crypto-Coprocessors for Public-Key Cryptography," J.-J. Quisqarter and B.Schneier, Eds., *Smart Card Research and Applications,* vol. 1820 of *Lecture Notes in Computer Science,* pp. 386-394, Springer-Verlag, 2000.

[35] T. Blum and C. Paar, "High Radix Montgomery Modular Exponentiation on Reconfigurable Hardware," *IEEE Transactions on Computers*, vol. 50, no.7, pp. 759-764, July 2001.

[37] C. D. Walter, "Montgomery's Multiplication Technique: How to Make it Smaller and Faster," *Proc.CHES 99, LNCS*, vol. 1717, pp.80-93, Springer, 1999.

[36] M. K. Hani, T. S. Lin, and N. Saikh-Husin, "FPGA Implementation of RSA Public-Key Cryptographic Coprocessor," *TENCON 2000. Proceedings*, vol. 3, pp. 6-11, 24-27 Sept. 2000.

[37] J. Põldre, K. Tammemäe, and M. Mandre, “Modular Exponent Realization on FPGAs,” *FPL'98, Tallinn,* pp. 336-347, 1998.

[38] H. Orup, “Simplifying quotient determination in high-radix modular multiplication,” *Computer Arithmetic, 1995, Proceedings of the 12th Symposium*, pp.193 –199, July 1995.

[39] Xilinx Inc., “The Programmable Logic Data Book 2000,” http://www.xilinx.com.

[40] Celoxica Ltd, “RC1000 Hardware Reference Manual,” http://www.celoxica.com.

[41] Europractice IC Service, “AMI Semiconductor 0.35um CMOS,” http://www.europractice.imec.be

[42] S. Yeşil, A. N. İsmailoğlu, and Y. Çağatay Tekmen, “A High-Speed ASIC Implementation of the RSA Cryptosystem,” *Proceedings of the Work in Progress, Euromicro Symposium on Digital System Design, DSD2003*, Belek, Turkey, September 2003.

APPENDIX A

VIRTEX-E CLB AND LUT

A.1. Configurable Logic Blocks (CLBs) and Slices

The basic building block of the Virtex-E CLB is the *logic cell*. A logic cell includes a 4-input function generator, carry logic, and a storage element. The output from the function generator in each logic cell drives both the CLB output and the D input of the flip-flop. Each Virtex-E CLB contains four logic cells as shown in Figure A.1. Each CLB is divided into two *slices*.

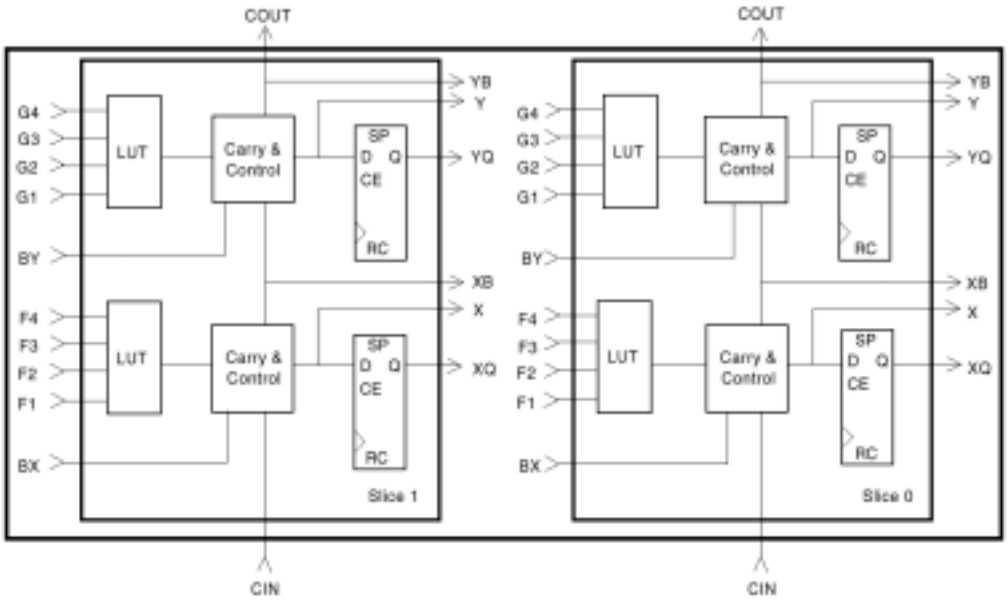


Figure A.1: Virtex-E CLB. Each Virtex-E CLB contains four logic cells and CLB is divided into two slices.

A.2. Look-up Tables (FGs)

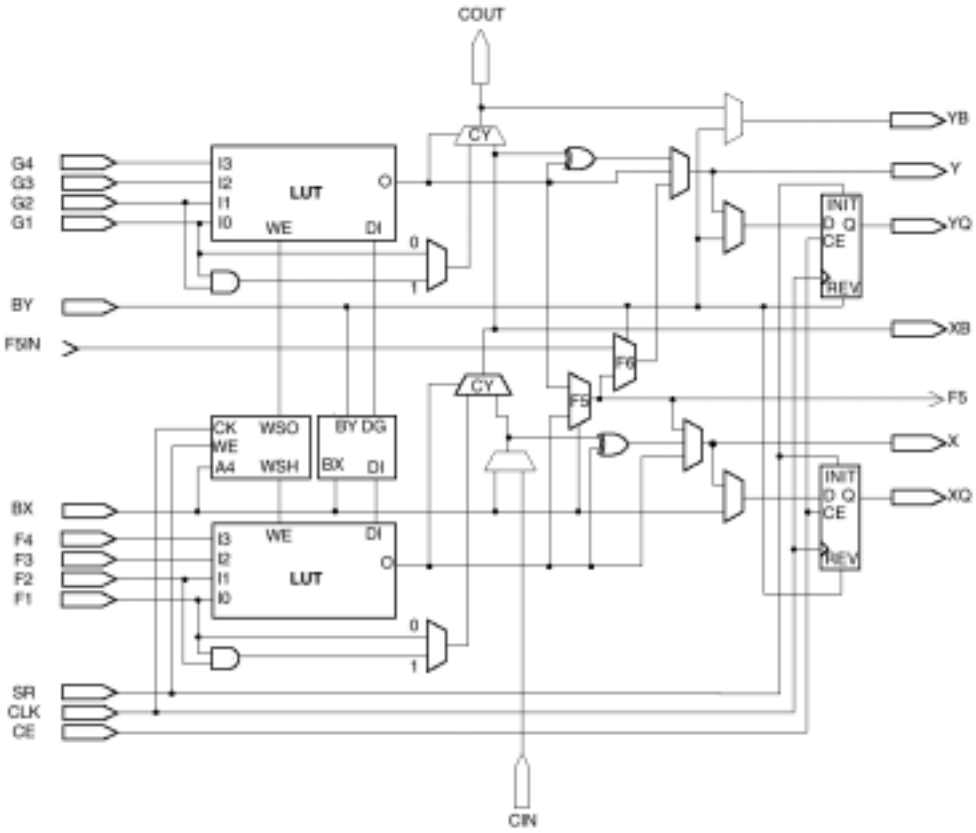


Figure A.2: The detailed schematic of a slice. A slice contains two LUTs, two DFFs, and one CY.

Virtex-E function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function generator, each LUT can provide a 16 × 1-bit RAM, and a 16-bit shift register. Figure B.3 shows the detailed schematic of a slice having two LUTs.

APPENDIX B

CELOXICA RC1000 HARDWARE

B.1. Overview

The RC1000-PP hardware platform is a standard PCI bus card equipped with a XILINX® Virtex TM family BG560 part with up to 1,000,000 system gates . It has 8Mb of SRAM directly connected to the FPGA in four 32 bit wide memory banks. The memory is also visible to the host CPU across the PCI bus as if it were normal memory. Each of the 4 banks may be granted to either the host CPU or the FPGA at any one time. Data can therefore be shared between the FPGA and host CPU by placing it in the SRAM on the board. It is then accessible to the FPGA directly and to the host CPU either by DMA transfers across the PCI bus or simply as a virtual address. The board is equipped with two industry standard PMC connectors for directly connecting other processors and I/O devices to the FPGA; a PCI-PCI bridge chip also connects these interfaces to the host PCI bus, thereby protecting the available bandwidth from the PMC to the FPGA from host PCI bus traffic. A 50 pin unassigned header is provided for either inter-board communication, allowing multiple RC1000-PPs to be connected in parallel or for connecting custom interfaces. The support software provides Linux(Intel), Windows®98 and NT®4.0+ drivers for the board, together with application examples written in Handel-C, or the board may be programmed using the XILINX® Alliance Series and Foundation.

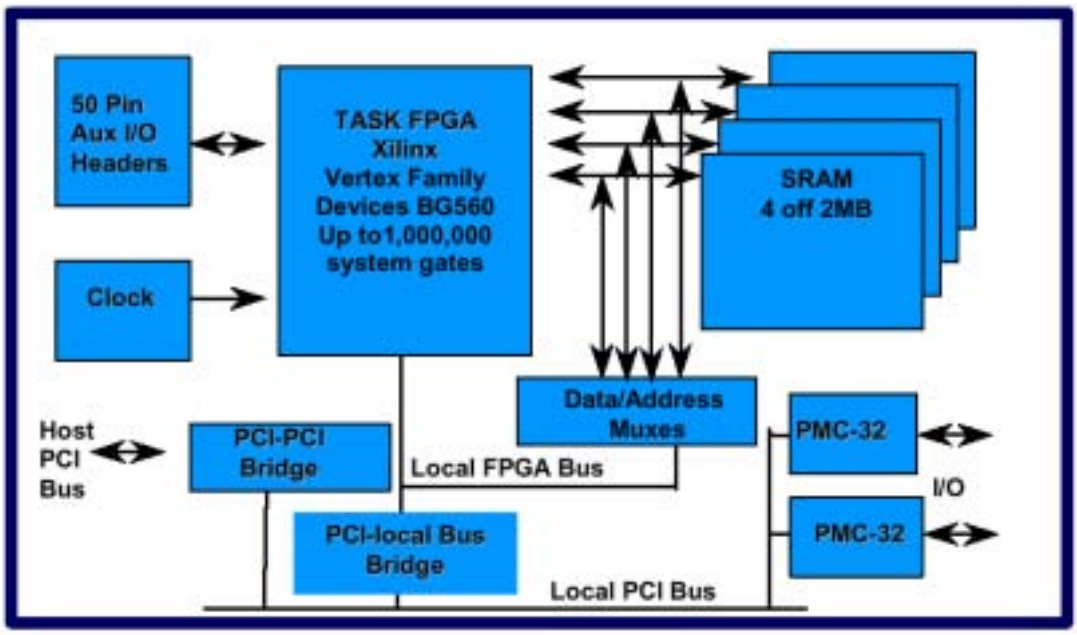


Figure B.1: Block Diagram of RC1000 Hardware

APPENDIX C

AMI SEMICONDUCTOR 0.35μm TECHNOLOGY

C.1. Mixed A/D Technology

The 0.35 μm CMOS technology is a mixed Analog/Digital process. It is derived from the fully digital 0.35μ CMOS process and extended with analog capabilities

C.2. General Characteristics

- 0.35 μm, up to 5 metal layers
- Self-aligned twin tub N- and P Poly gates
- W-plug filling of stackable contacts and vias
- Nitride based passivation
- 2.0V to 3.6V Supply
- Protection :
    - Latchup resistance > +/- 200mA
    - ESD > +/- 2000V
- 6 Inch epi wafers

C.3. Layout Rules

- Drawn minimum gate length : 0.35μm for both PMOS and NMOS
- Polysilicon pitch : 0.9μm

- Metal 1 pitch : 1.1µm
- Metal 2 pitch : 1.4µm
- Metal 3 pitch : 1.4µm
- Metal 4 pitch : 1.4µm
- Metal 5 pitch : 2.8µm

C.4. Standard Cell Libraries

Following libraries are available for the AMI Semiconductor 0.35 µm CMOS technology :

AMI Semiconductor libraries supporting the ADS Asic Design Framework

- High Speed and Low Power Library (MTC 45000)
    - 393 core cells (gates, latches, flipflops,..)
    - 101 I/O cells (with slew rate controlled outputs and spike suppression)
    - ROM Density up to 240 Kbits/mm2
    - RAM Density (Static, single port): 25 Kbits/mm2
    - Gate density: 15000 NAND equiv. gates/mm2
    - Temp. range : -55 ... + 125deg.C
    - Typical gate delay(3,3V)
      - Unloaded invertor delay of 50ps
      - 2-input NAND delay of 610ps (typ) with fanout=2
    - Power : 0.5 µW/gate/MHz at 3 V
    - Additional analog modules
      - High ohmic polysilicon resistors (1kOhm/sq)
      - High value double poly capacitors (1.1 nF/mm2)
- Versatile I/O Library : PAD limited I/O cells (MTC45100)
- ROM and RAM compilation