A Hardware Model of an Expandable RSA Cryptographic System

by

Adnan Abdul-Aziz M.S. Gutub

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

December, 1998

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

A Hardware Model of an Expandable RSA Cryptographic System

BY

Adnan Abdul-Aziz M.S. Gutub

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

Computer Engineering

December 1998

بسم الله الرحمن الرحيم

هذا من فضل ربّي

لِيَبلوَنِي أأشكرُ أم أكفُر ومَن شَكرَ فإنَّمَا
يَشكرُ لِنَفسهِ ومَن كفَرَ فإنّ ربّي غَنيٌّ كريمٌ

سليمان عليه السلام
القرآن الكريم : سورة النمل

This is of the bounty of my Lord, to test me as to weather I am grateful or ungrateful. He who is grateful is but grateful for his own good; and he who is ungrateful, verily, my Lord is self-sufficient, most generous.

Prophet Solomon

Quran 27:40

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

The Thesis, written by

Adnan Abdul-Aziz M. S. Gutub

under the direction of his Thesis advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER ENGINEERING

Thesis Committee:

27.12.98
Dr. Alaaeldin Amin (Chairman)

27.12.98
Dr. Khalid Elleithy (Member)

28/12/98
Dr. Khalid Al-Tawil (Member)

Abdullah Almejel
Department Chairman

Dean, College of Graduate Studies

30-12-98
Date

Dedicated

To

My Parents

Acknowledgments

First and foremost, all praise to the Almighty ALLAH who gave me the courage and patience to carry out this work.

Acknowledgment is due to King Fahd University of Petroleum and Minerals for providing support to do this work.

My deep appreciation goes to Dr. Alaaeldin Amin, who served as my Thesis advisor. I also wish to thank the other members of my Thesis Committee Dr. Khalid Elleithy and Dr. Khalid Al-Tawil.

Thanks are due to my Professors: Dr. Sadiq Sait, Dr. Mostafa Abd-El-Barr and Dr. Atif Al-Najjar, for their encouragement and valuable advises.

I am very grateful to my wonderful wife (Manal M. Fattani) and children (Muna and Alaa), without their patience and sacrifices such a major undertaking and its completion would have not been possible.

Finally, my profound gratitude and appreciation go to the rosy part of my life, to my mother (Amnah S. Sait) and father (Dr. Abdul-Aziz M.S. Kutub), for their continuos prayers, encouragement and moral support.

Contents

List of Figures

List of Tables

Abstract

Name: *Adnan Abdul-Aziz M. S. Gutub*

Title: *A Hardware Model of an Expandable RSA Cryptographic System*

Major Field: *Computer Engineering*

Date of Degree: *December 1998*

Data security is an important aspect of information transmission and storage in an electronic form. Cryptographic systems are used to encrypt such information to guarantee its security. To retrieve such information, the encrypted form must be first decrypted. One of the most popular cryptographic systems is the RSA system. The security of the RSA-encrypted information largely depends on the size of the used encryption key. The larger the key size is the longer the encryption/decryption time will be. To cope with the continuous demand for larger key sizes, faster hardware implementations of the RSA algorithm has become an active area of research. One disadvantage of hardware implementations is their fixed key sizes. If the key size is to be increased, the hardware design should be fully replaced.

The work reported here proposes an RSA hardware implementation that can be expanded as the key size gets larger. This implementation is modeled using VHDL in a parametrizable manner. Two other parameterized RSA hardware designs have also been VHDL modeled for comparison. The three models are compared for a 1024-bit key size and the results are analyzed. The complexity of the designs are compared and conclusions regarding optimal delay and area parameters are made.

Master of Science Degree

King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
December 1998

بسم الله الرحمن الرحيم

خلاصة الرسالة

الاســـــم : عدنان بن عبدالعزيز بن محمد صديق قطب

عنوان الرسالة : تصميم دائرة إلكترونية دقيقة قابلة للتطوير حسب الحاجة لعمليـــة التشفير بنظام RSA

التخصـــص : هندسة الحاسب الآلي

تاريخ الشهادة : شعبان ١٤١٩هـ

التشفير(Cryptography) هو الأسلوب الأمثل لحماية المعلومات والحفاظ على سريتها. وأحد أنجح طـــرق التشفير المستخدمة يعرف بـــ(RSA) ، وهي الطريقة التي تعتمد على حجم مفتاح الشفرة لتعقيد استنباط الرسالة الأساسية في وقت قصير ، وتعاني طريقة (RSA) من سلبية في تصميمها بالدوائر الإلكترونية الدقيقة ، وهـــي أن الدوائر مصممة للتشفير بمفتاح ذو حجم معين ثابت ، لو تغير لأحتاج تغيير التصميم الإلكتروني بالكلية.

وقد تم في هذا البحث تصميم طريقة جديدة مبنية على فكرة تطوير الدائرة الإلكترونية الدقيقـــة حســـب الحاجة، بحيث تم تمثيل هذا التصميم باستعمال نموذج محاكاة التصميم الإلكتروني باســـتخدام لغـــة (VHDL) ، وقورن هذا النموذج بالتفصيل مع تصميمين آخرين أيضاً باستخدام (VHDL) ، وأثبت التصميم المقترح تفوقـــاً في السرعة بالرغم من أنه يعتبر الأكبر مساحة.

وقد تم إجراء مقارنة بين التصاميم الثلاثة من حيث كفاءة الأداء والسرعة و التكلفة (التكلفة = المساحة × الســـرعة^{2})، وأظهر التصميم المقترح نتائج مقاربة لأفضل تصميم. واعتبرت زيادة التكلفة نوع من الثمن مقابل للمرونة المتوفرة في هذا التصميم والتي تجعله قابل للتطوير حسب حاجة المستخدم.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران – المملكة العربية السعودية

شعبان ١٤١٩هـ

Chapter 1

Introduction

Encryption is a well recognized technique for information protection. It is used effectively to protect sensitive data such as passwords that are stored in a computer as well as information transmitted through different communication media.

Encryption is the transformation of data into a form which is unreadable without a secret decryption key. Over the years, several encryption (or cryptographic) techniques have been used, however, most of them were not practical [10, 13].

Depending on the encryption/decryption key, cryptographic systems can be classified into two main categories: secret key cryptosystems and public key cryptosystems. The secret key cryptosystems uses one key for both encryption and decryption. Public key cryptosystems, however, use two different keys, one for encryption and the other for decryption.

The most popular public key method is the RSA [1, 6, 10, 11, 12]. The security of this method depends on the size of the key. The larger the key size is, the more secure the system will be [2]. More security, in this context, means that more computation time would be needed to hack the system. This largely depends on the implementation technology which is growing at a very fast rate.

RSA cryptographic systems can either be software or hardware implemented. Software implementations are typically very slow compared to hardware ones. Whereas software implementations are flexible allowing changes in the key size, hardware im-

plementations lack such flexibility. For example, if the RSA key size is to be increased to improve security, the hardware must be redesigned.

1.1 Thesis Objective

The objective of this Thesis is to design an expandable implementation of an RSA processor. The processor is to be designed in a bit-sliced manner, such that larger key sizes can be easily accommodated by merely adding more slices.

For proper evaluation of the proposed expandable implementation, a VHDL structural model is to be developed. This model will be used to verify and simulate the operation of the proposed design. Furthermore, other available RSA implementations are modeled using VHDL as well to be compared to the proposed expandable design in terms of area and speed.

1.2 Thesis Outline

In the next chapter, a general review of cryptographic concepts is given. Some examples of traditional secret key cryptosystems are discussed. The public key RSA cryptosystem has been covered in more depth because of its simplicity and popularity.

In chapter 3, reported RSA hardware implementations are described. The emphasis, however, is on the design of a custom integrated circuit, to achieve the highest performance. Some general techniques for performing modular operations are briefly discussed.

In chapter 4, the design and operation of the proposed expandable RSA hardware is detailed.

Chapter 5, introduces two other hardware implementations. Both of which use the repeated squaring algorithm for modulo-exponentiation. However, the techniques used to compute modular multiplication are different. One implementation is based on Montgomery's method for modulo multiplication, while the other method performs

the modulo multiplication through integrated addition and subtraction operations.

In chapter 6, the proposed expandable hardware is compared with the two other implementations on the basis of time, area and AT^2 cost. The three implementations are analyzed for 1024-bit key size, and the results are compared.

Chapter 7, summarizes the results of this thesis work and provides some proposed future work in this area.

Chapter 2

Cryptographic Systems

2.1 Introduction

Cryptography is the science concerned with the process of encryption and decryption. The word, cryptography, is taken from the Greek word *kryptos* meaning *hidden*, and *graphia* meaning *writing* [6]. In data communication, for example, the original message is called the plain text 'P'. If this P is to be sent over an insecure channel it must be encrypted (Figure 2.1). The resulting text is referred to as the cipher text 'C'; sending C over an insecure channel protects the original message P against possible attacks of eavesdroppers. Encryption must be a reversible process, i.e. an inverse operation (decryption or deciphering) must exist to transform the encrypted information back to the original plain text [1].

The encryption algorithm is the procedure followed to encipher the plain text into

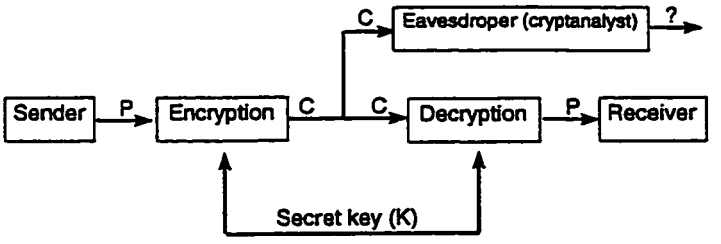


Figure 2.1: The information flow in a classical cryptographic system

its cipher form using a special number called the encryption/decryption key. This key is like a password to the encrypted message. Without knowledge of this key and the used algorithm, the cipher text can not be transformed back to the plain text. For this reason, the key must be transmitted to the receiver through a very secure channel (Figure 2.1).

Attempting to decrypt encrypted data without knowledge of the key is called cryptanalysis. A cryptographic system is secure [6], if the plain text "P" can not be obtained from the cipher text "C" through cryptanalysis.

Based on the type of cryptographic key used, cryptographic systems fall into two general categories; namely: secret key cryptography and public key cryptography. While the secret key category depends on using the same key for both encryption and decryption, public key cryptographic systems adopt a two-key system: one for encryption and another for decryption.

In the following, a number of traditional secret cryptographic systems will be briefly described. More detailed information can be found in the literature [4, 5, 8, 19].

2.1.1 Substitution

This type of encryption replaces the actual bits, characters or blocks of characters with substitutes. For example, one letter replaces another, based on a certain rule. The first clearly documented substitution, known as Caesar's cipher [6], simply replaces a given letter from the plain text by the letter three places beyond it. Another substitution approach [1] replaces the letters (BWEKQFMVYALUCONPHSIDXTRGZJ) by the normal alphabet. Using this order of alphabet; say: "A" in the plain text is replaced by "B", "B" in the plain text is replaced by "W", "C" by "E", etc. Thus, if the plain text message is:

THE QUEEN HAS GIVEN BIRTH TO A HEALTHY SIX POUND BOY

Then the cipher text [1] will be:

DVQ HXQQO VBI MYTQO WYSDV DN B VQBUDVZ IYG PNXOK WNZ

This type of cryptosystem can be cryptanalyzed by building frequency tables of letters, letter pairs and letter triples in the transmitted encrypted message and then comparing that to the well known English letter frequencies.

2.1.2 · Transposition

In this cryptosystem, the letters are permuted according to a certain rule. For example [1], the message:
THE QUEEN HAS GIVEN BIRTH TO A HEALTHY SIX POUND BOY
is broken into five character groups (including spaces) and the letters in each group are rearranged according to the permutation :

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 1 & 4 & 3 \end{pmatrix}$$

so the third letter in the plain text is written first, the first letter is written second and the fifth letter is written third, and so on. The cipher text will be:
ETQ HEU NESHG AEI NVRBHTIO A TE LAHYTS H IOPXDUB NXOXXY

While substitution cryptosystems preserve the location of the letter in the message, but changes the letters themselves, transposition cryptosystems preserve the letters but their locations within the message are changed. Transposition can be broken by seeking permutations and rejoining them until the plain text is reconstructed.

2.2 Public Key Cryptosystems

Based on the type of cryptographic key used, cryptographic systems fall into two general categories; namely: secret key cryptography and public key cryptography. While the secret key category depends on using the same key for both encryption and decryption, public key cryptographic systems adopt a 2-key system: one for encryption and another for decryption. This method solves the key management and key distribution problems discussed before. Public key systems depend on some basic fundamental operators, two of which are discussed in the following section.

2.2.1 Fundamental Operators

The security of public key cryptographic systems depends basically on the computational complexity of the encryption/decryption operation. With the exception of some assumptions made about the computational difficulty of the used operators, there is no concrete mathematical proof for the security of such systems [1]. These operators are the one-way and the trap-door functions. The existence of these operators is still considered an open research area [8, 9, 11].

One Way Functions

One way functions are functions that have no inverse function. For example, if M is a message, and $F(M)$ is a function that generates the cipher text, C. $F(M)$, is called a one-way function if M can not be regenerated from C by the use of the function F while $F(M)$ itself, can be easily computed for any message M.

Trap-Door Functions

A trap-door function is a one-way function but with a unique inverse function F^{-1}, the trap-door. This unique inverse function F^{-1} is the only way to regenerate the function input M, given its output C. For example, the message M is the input to the function $F(M)$, which will generate C. The only way to regenerate M back from C is through the trap-door function F^{-1}. Thus,

- If M and F are known, then C can be generated $\Longrightarrow F(M) \equiv (C)$
- If C and F^{-1} are known, then M can be regenerated $\Longrightarrow F^{-1}(C) \equiv (M)$

Some restrictions on trap-door functions must be taken into consideration [11]. These restrictions are:

- Both F and F^{-1} must be easy to compute for all needed values.
- Computing C from M without F is computationally infeasible.

- Computing F^{-1} from F, or F from F^{-1}, is computationally infeasible.
- Computing M from C without F^{-1} is computationally infeasible.

These two operators are the basic building blocks for public key cryptosystems.

2.2.2 Historical Background

Diffie and Hellman in 1976 [12], suggested two solutions for the key distribution problem. These solutions laid the foundation for what is now known as the public key distribution system and the concept of public-key cryptosystems.

The first solution depends on the use of one-way functions. In this case, the sender and receiver exchange some one way function codes through which a secret cipher key is generated. These exchanged codes should prevent eavesdroppers from knowing or computing the secret ciphering key. By this method the key is distributed without a need for secure channels. This method is covered in more detailed fashion in the literature [1, 10, 11, 12].

The second proposed solution, was a general conceptual model for general public-key cryptosystems. This model was given [1], without any practical example or implementation as shown in Figure 2.2. The main contribution was the notion that keys could come in pairs, one is a private key and the other is a public one. These keys should not be computable from one another following the trapdoor function. Thus, if any message is encrypted by one of the two keys, it can only be decrypted by the other key.

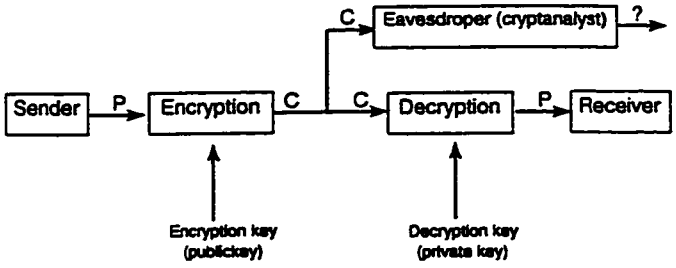


Figure 2.2: Public key cryptographic system (general concept)

Numerous public-key ciphering algorithms were, thereafter, proposed. However, many have been proven to be either insecure or impractical due to either the large size of the keys or the large size of the generated cipher text [10]. Only a few of these public-key algorithms have so far, been secure and practical. Some are suitable for encryption or key distribution, while others are more suitable for authentication and digital signatures. Three algorithms have been used both for encryption and digital signatures. These are the RSA algorithm, ElGamal algorithm and Rabin algorithm [10].

From a practical point of view, the major drawback of these three algorithms is their slow speed as compared to single key systems. A compromise solution for the speed problem is presented by merging the two methods, the public key cryptosystem for key distribution and the single key cryptosystems (e.g. DES), for message encryption. This hybrid approach has greatly improved system security as well as performance [11]. Hybrid cryptosystems, however, are not useful for all cryptographic applications, e.g. digital signature. Thus, speeding up the public key system is still a vital research area [1, 6, 12].

2.3 The RSA System

In 1978, R. Rivest, A. Shamir and L. Adleman, came up with an algorithm which can be used for both encryption and digital signature. This algorithm was named *RSA Cryptographic System* [2], and is widely considered as the simplest public key algorithm [10]. This is due to its high security, relative simplicity and good performance. None of the other public key algorithms is as widely used as RSA [1, 10, 17].

RSA's security is based on the difficulty of the integer factoring problem, which is known to be a very hard problem to solve [8, 9]. The popularity of the method is due to the use of the same basic calculations of modular exponentiation for both encryption and decryption [11, 13].

2.3.1 RSA Encryption

This system requires each user to have a public encryption key (e, n), and a private decryption key (d, n). To encrypt a message, it should be represented as a sequence of integers: $m_1, m_2, m_3, ...$, with each m_i being an integer in the range $[0, n-1]$. Each message block m_i is encrypted into a cipher text C_i, using the public key: (e, n) as follows:

$$C_i = E(m_i) = m_i^e \ mod \ n$$

In order to retrieve the original message m_i, the private decryption key: (d, n), is used as follows:

$$m_i = D(C_i) = C_i^d \ mod \ n$$

2.3.2 Generation of the RSA Keys

Choose p and q as two random large primes [2]. Then, compute n as the product of p and q: $n = p \times q$. Note that n is made public, but its factors p and q are secret. Hiding p and q, hides the relation between d and e. Next, compute Euler's function: $\phi(n) = (p-1)(q-1)$. Then, choose e (the encryption key), such that e and $\phi(n)$ are relatively prime, i.e. $gcd(e, \phi(n)) = 1$. Finally, d (the decryption key), can be calculated by using the secret number $\phi(n)$, such that:

$$e \times d = 1 \ mod \ \phi(n) = 1 \ mod \ (p-1)(q-1)$$

It can be shown that d and n are also relatively prime integers [11].

From these calculations, we can conclude that the public encryption key is: e, n; and the private decryption key is: d, n.

2.3.3 Example on Encryption Using RSA System

In the following example, sender A, wants to send the message: "HELLO " to receiver B. The sender needs to use the receiver's public key:(e, n), for encryption. To decrypt, receiver B, is required to use his private key: (d, n).

Receiver B, needs first to make the initial calculations for his public key: (e, n), and his private key: (d, n), as follows:

Let $p = 83$ and $q = 37 \implies n = p \times q = 83 \times 37 = 3071$

$\phi(n) = (p-1)(q-1) = 82 \times 36 = 2952 \qquad d = 709$ & $e = 229$

such that: $e \times d = 229 \times 709 = 162361(mod\ 2952) = 1(mod\ 2952)$

Accordingly, B's public key is: (229,3071), and his private key is: (709,3071).

Using the convention "A"=00, "B"=01, ... , "Z"=25, and "(blank space)"=26. Then, the message "HELLO " can be represented as: "07, 04, 11, 11, 14, 26". Taking the message blocks as: $m_1 = 0704$, $m_2 = 1111$, $m_3 = 1426$.

After that, the encryption key (229,3071) is used to compute:

$C_1 = E(m_1) = {m_1}^e\ mod\ n = (0704)^{229}\ mod\ 3071 = 2443.$

$C_2 = E(m_2) = {m_2}^e\ mod\ n = (1111)^{229}\ mod\ 3071 = 1629.$

$C_3 = E(m_3) = {m_3}^e\ mod\ n = (1426)^{229}\ mod\ 3071 = 1556.$

Thus, the cipher text to be sent by the sender A, will consist of the following sequence: "24, 43, 16, 29, 15, 56".

The receiver's private key: (709,3071), is the only way to retrieve the message from the cipher text. So to regenerate the plain text the receiver B needs to compute:

$m_1 = D(C_1) = {C_1}^d\ mod\ n = (2443)^{709}\ mod\ 3071 = 704.$

$m_2 = D(C_2) = {C_2}^d\ mod\ n = (1629)^{709}\ mod\ 3071 = 1111.$

$m_3 = D(C_3) = {C_3}^d\ mod\ n = (1556)^{709}\ mod\ 3071 = 1426.$

As observed, the plain text "07, 04, 11, 11, 14, 26", is regenerated again representing the message "HELLO ".

2.3.4 The RSA Digital Signature Scheme

The RSA algorithm can be used as a digital signature scheme as well. For example, if user A needs to send a signed message to another user B, he (user A) will use his own private key to compute S as follows:

$$S = D(m) = m^d mod\ n$$

User A will then send S, for B to verify A's signature. To do this he needs to use A's public key; as the following calculation:

$$E(S) = E(D(m)) = (D(m))^e \ mod\ n = (m^d \ mod\ n)^e \ mod\ n = m^{de} \ mod\ n = m$$

This will generate the message m, only if user A has signed it and the message had not been tampered with [2, 10, 11, 12].

2.3.5 Security of the RSA Cryptosystem

Decryption of RSA-encrypted messages by cryptanalsis is as difficult as factoring a large number into its prime factors. An advice by Rivest [2], was to use at least a 100-digit integer for both p and q. This will let n have 200-digits or even more. Large size keys, however, result in slower encryption and decryption.

2.3.6 RSA Speed

Some techniques are proposed to improve the speed of software RSA systems [12]. However, these techniques are still slower than hardware implementations [21]. Compared to private key systems, e.g. DES, the RSA is much slower. The fastest reported hardware has a throughput which is 1000 times slower than what has been reported for the DES system [10]. Hardware speed, however, can be further improved by improvement in the technology used for hardware designs. More information regarding RSA hardware implementations is given in the next chapter.

2.4 Summary

In this chapter, a general review of cryptographic concepts is given. Some examples of traditional secret key cryptosystems are discussed. The public key RSA cryptosystem has been covered in more depth because of its simplicity and popularity.

Chapter 3

Review of RSA Hardware Implementations

3.1 Introduction

Efforts to Implement public key systems in hardware began in the late 1970's. These early designs, however, were not practical because of their high cost, low speed and inflexibility. In the past decade, however, implementing public key systems in hardware has changed due to several factors [50, 51]. The most important factor is the tremendous progress in technology. Millions of transistors can now be fit onto a single chip. Another important factor is the tremendous increase in the number of computer communication networks, e.g. the internet. Security of transactions conducted across a network has become a vital issue to the reliability and success of these networks. Public key cryptography plays an important role in this regard.

The RSA encryption algorithm is considered the most practical public key system in use due to its high security and acceptable complexity compared to other public-key techniques [1, 2, 10, 17]. Several hardware approaches for implementing the RSA algorithm have been proposed [21, 23, 25, 29, 31, 42, 43, 44, 46, 47]. One approach uses available digital signal processors (DSP) [25, 26], for this purpose. Another approach

uses several active memory chips programmed to perform the RSA algorithm [22].

An important implementation strategy is the design of custom integrated circuits that are capable of performing the required modular operations. Since this offers the best performance, our work will concentrate on this particular approach.

In 1989, E. Brickell [21], published a survey of hardware implementations of the RSA algorithm. Ten different implementations were covered in this survey and their encryption speeds were compared. However, such comparison is somewhat misleading due to the differences in the fabrication technology of the different implementations.

Since the key RSA operation is the modular multiplication or more accurately the modular exponentiation, different methods were developed to implement these modulo computations. In the following section, some such methods are discussed.

3.2 General Techniques for Modular Operations

3.2.1 The Repeated Squaring Algorithm

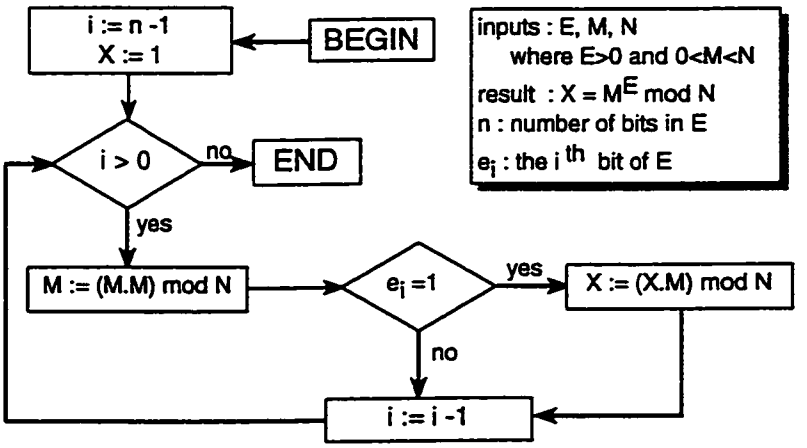


Figure 3.1: The repeated squaring algorithm

Modular exponentiation can be computed through a set of iterative modular multiplications or through a set of repeated squarings [21, 32, 33, 42]. The repeated squaring algorithm is mainly performed by repeating modular multiplication a number of times as the number of bits of the exponent. The bits of the exponent should be scanned

starting with the MSB (most significant bit). Whenever it is '1' a further modular multiplication is to be performed, as illustrated in Figure 3.1.

The bits of the exponent if scanned with the LSB (least significant bit) first, allows for about 30% improvement in the average speed through the use of parallelism of the modular multiplication processes [43, 46, 47]. This improved approach is shown in Figure 3.2.

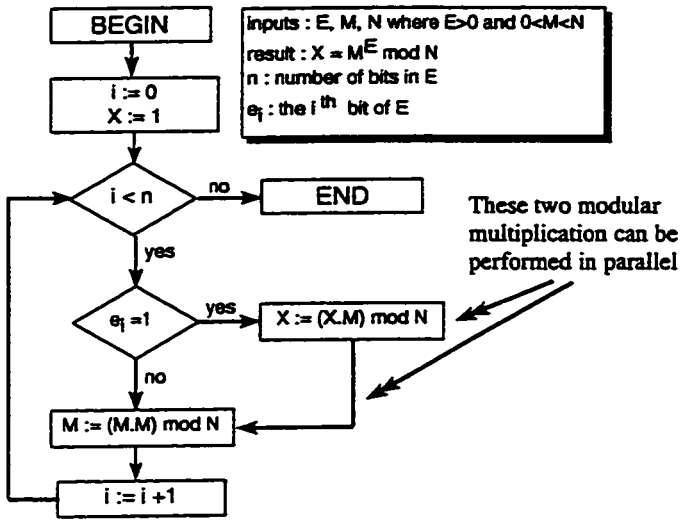


Figure 3.2: The improved repeated squaring algorithm

3.2.2 General Modular Multiplication Techniques

Modular multiplication can be implemented in different ways. Residue number systems, for example, have been recently receiving more attention [34, 41]. Reported implementations, however, have ignored important hardware implementation issues particularly the need for conversion between binary and residue numbers. This has discouraged researchers from using such an approach [35].

The straight-forward approach to compute modular multiplication is by performing the multiplication and then subtracting the modulus several times until the result is less than the modulus. This approach is inefficient and suffers from low speed. This can, however, be improved by merging the modulo subtraction with the multiplication add operations [43, 46, 47], as outlined in Figure 3.3.

Another approach is based on look-up tables, i.e. ROM-based methods [37]. This method, however, is quite expensive since the required memory space grows exponentially with the word size [35].

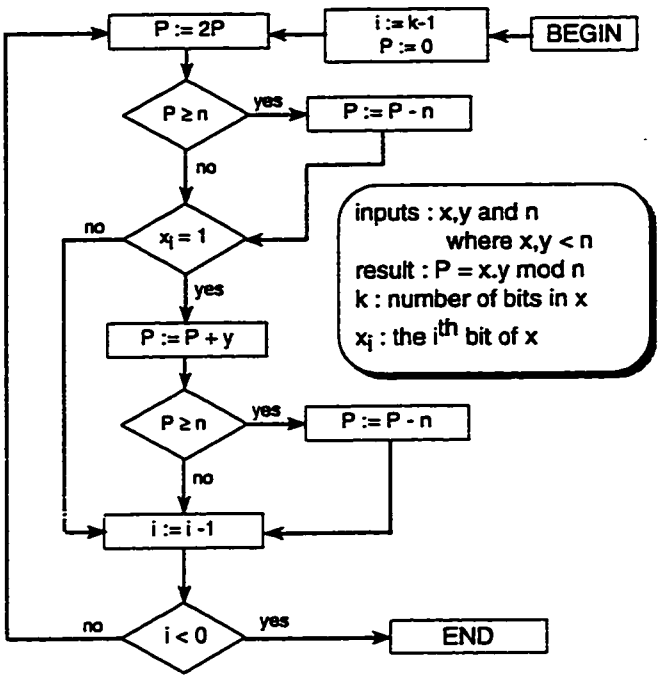


Figure 3.3: The multiplication with reduction modified algorithm

In 1991, G. Alia and E. Martinelli [36], came up with a new method to compute modulo multiplication. They found that their method is optimal to implement using few fast VLSI binary-multipliers, compared with ROM-based methods.

An improved look-up table method was proposed by C. Wu and Y. Chon in 1994 [35]. This method depends on the use of look-up tables by specially designed block multipliers, which are designed as a bit-parallel multiplier and a bit-serial one. It is partly parallel and partly serial to get the advantages of both techniques. The parallel multiplier results in high throughput rate and the serial help reduce power consumption. The computing speed and the area of this modular multiplier is dominated by the used adders [35].

In 1994, C. D. Walter [39], proposed a logarithmic speed modular multiplication al-

gorithm. This algorithm and its implementation will be covered briefly in the following section.

In 1985, Peter Montgomery [24], proposed a new method for modular reduction without regular division. His method was found to be fast [23, 24, 28], and suitable for hardware designs [27, 29, 30, 31]. Montgomery's multiplication method has been widely adopted for use in exponential RSA implementation. Elaboration of Montgomery's algorithm will be given later this chapter.

3.3 Logarithmic Speed Implementation

A logarithmic speed modular multiplication hardware is proposed by C. D. Walter [39]. It performs each modular multiplication for $n - digit$ inputs in $O(log\ n)$ time.

3.3.1 The Algorithm

The algorithm used for this implementation divides the operation of $(A \times B)\ mod\ N$ into six computational steps as follows:

1. compute: $A \times B$;
2. compute: $1/N$;
3. compute: $(A \times B) \times (1/N)$
4. compute: $fract((A \times B) \times (1/N))$; ($fract$ discards the integer part of a real number and returns the non-negative fractional part.)
5. compute: $N \times (fract((A \times B) \times (1/N)))$;
6. compute: $rnd[N \times (fract[(A \times B) \times (1/N)])]$; ($rnd$ rounds the real number to the nearest integer.)

3.3.2 The Implementation

The three multiplication processes are computed sequentially on the same hardware, with one clock cycle per multiplication process. Each multiplication process is performed as repeated additions using the Wallace tree structure [48]. The reported critical path for a 512-bit multiplication process, contains around 40 XOR-gates. With three required multiplications (steps: 1, 3 and 5 of the algorithm), a total of 120 XOR-gates delay is required for the three clock cycles needed. Note that step-2 of the algorithm, computing $(1/N)$, is assumed to be precomputed by software.

The reported area needed to construct such logarithmic time implementation for 512-bit numbers, is equivalent to 5×10^6 XOR gates [39]. In fact, this design stretches current technology to the limit, but provides a great improvement in speed.

3.4 Implementations of Montgomery's Algorithm

3.4.1 Montgomery's Algorithm For Exponentiation

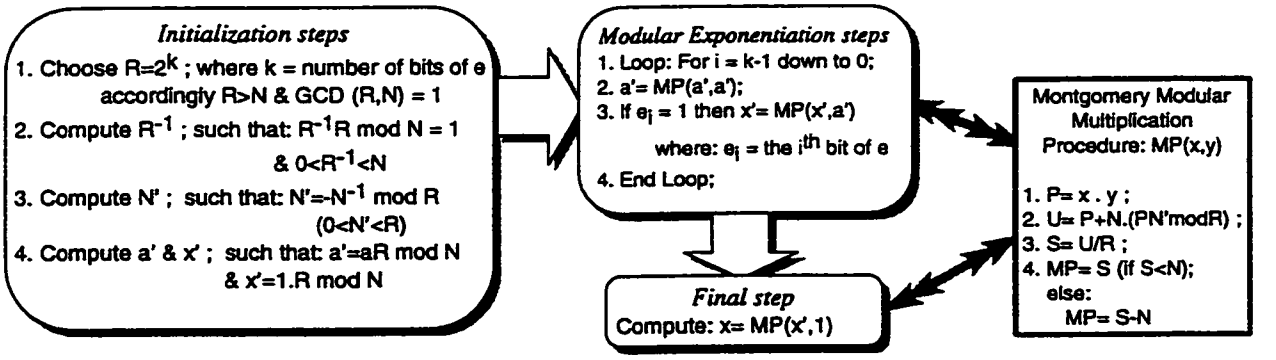


Figure 3.4: Montgomery's algorithm for modular exponentiation

For the RSA method, the problem to be solved is: $x = a^e \ mod \ N$. Montgomery's technique will compute x without trial division. Figure 3.4, shows the steps to be followed to perform this operation. Note that the *mod* R and division by R operations, in the Montgomery modular multiplication procedure, can be inexpensively computed since $R = 2^k$.

3.4.2 Montgomery's Algorithm Hardware Designs

S. E. Eldridge and C. D. Walter [29], studied Montgomery's modular multiplication algorithm and proposed a hardware implementation for it. They compared Montgomery's method with earlier reported techniques and concluded that Montgomery's algorithm is better suited for implementation. Based on this, they designed a faster hardware which doubled the overall speed for the same number of clock cycles. Their design, however, is practical only for moduli up to 1000-bits in length which is considered as a design limitation. Their implementation is constructed using a tree structure of repeated units of adders that perform the multiplication process as repeated addition. This hardware model has a bottleneck of broadcasting the quotient digits throughout the design [28].

C. D. Walter [28], further improved the previous design. The limitation on the modulus was completely removed and the critical path was simplified to solve the problem of broadcasting data all-over the hardware. This design [28], performs each modular multiplication process for n-digits in $O(n + log\ n)$ time, which is slower than the logarithmic speed implementation [39], proposed by the same author, as briefly described in the previous section.

A systolic array for modular multiplication using Montgomery's algorithm was described by C. D. Walter in 1993 [27]. For n-digit operands, this implementation requires $2n+2$ clock cycles for the first output digit to appear after the first input digit is fed to the systolic array. The implementation is constructed as a two-dimensional array of n^2 cells. All cells are identical except for the cells on the rightmost column which do not generate output digits. Each such cell performs two multiplications and three additions, which required five XOR-gates, seven AND-gates, and two OR-gates, assuming binary numbers representation.

For RSA applications, typical inputs have at least 500-bits. The number of cells required is at least 500^2, or about 4×10^6 gates. This large area stretches today's technology to the limit.

3.5 Full RSA Implementations

Many RSA implementations have been proposed in the literature [21, 23, 24, 31, 42, 43, 44, 46, 47]. This is mainly due to the difficulty of obtaining high speeds at reasonably reduced hardware cost. Furthermore, most of the reported implementations do not provide enough information on the design details, e.g. [21, 27, 28, 35, 39, 44, 45].

H. Orup, E. Svendsen and E. Andreasen [47], presented an efficient RSA hardware implementation called VICTOR, where they improved some earlier algorithms. The modular exponentiation technique used is similar to the modified repeated squaring algorithm shown in Figure 3.2.

The hardware processor proposed by F. al-Twaijry and S. Barton [46], describes some techniques to speed-up the central computational process in the RSA algorithm. Assuming that k is the maximum number of bits available for the RSA operations, the time-consuming modular multiplication operation can be performed by a complete integer multiplication of two k-bits numbers followed by modulo reduction of the resulting $2k$-bit product. In this design [46], however, modulo reduction is performed at each step of the multiplication process as shown in Figure 3.3, such that the result never grows beyond $k + 1$ bits. This design modification resulted in a reported 56% increase in speed. The area of this high speed design [46], is the major drawback of such an approach.

Another RSA implementation, proposed by Jorg Sauerbrey [31], uses a systolic hardware for modular exponentiation using Montgomery's algorithm. Since our work targeted the design of an expandable RSA processor, the work of Sauerbrey seemed to be a logical starting model for such a design. Modeling the implementation reported by Sauerbrey has revealed that his design had a major flaw which makes it incorrect. Details of that are clarified in the following section.

3.6 Systolic Arrays for Modular Exponentiation

In 1992, Jorg Sauerbrey [31], reported a systolic array hardware design that performs modular exponentiation. This design uses two identical systolic arrays to build the basic multiplier which can handle multi-bits operands.

3.6.1 Systolic Array for Multiplication

The systolic array consists of a set of identical cells that can process numbers in base 2^b, where b is the number of bits per word. The message M, is divided into l-words, each representing a digit of b-bits. The minimum number of cells required to process the l-digits is $= \lceil l/2 \rceil + 1$.

The l-digits of the message M requires $2l$-clock cycles to complete the multiplication process. During the first l-clock cycles, digits of the two operands are fed serially to the systolic array with the least significant digit being fed first.

The $2l$-digits of the product will be produced serially one digit per clock cycle starting with the least significant digit. Accordingly, each multiplication process requires $2l$-clock cycles. Whereas the l-digits of the two input operands are fed during the first l clock cycles, zeros are fed in the second l-clock cycles.

Figure 3.5 shows the interconnection and the input of the array when performing the operation: $p = x.y + q$. The control input z signals the start of the multiplication process. The behavior of each cell is described by the algorithm shown in Figure 3.6.

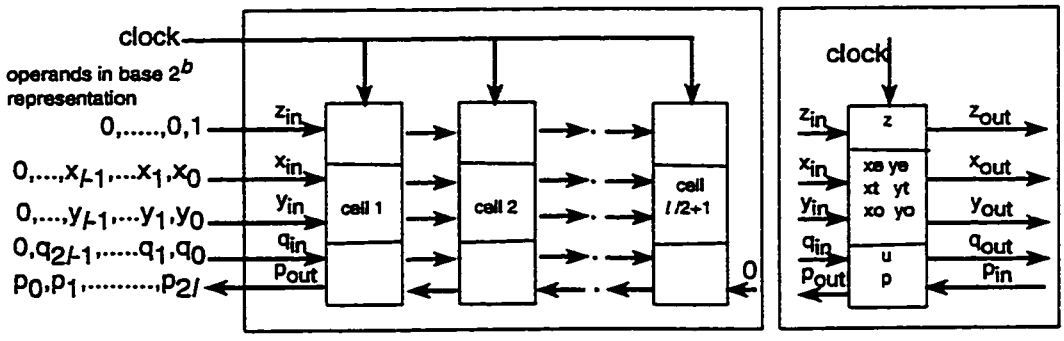


Figure 3.5: The systolic array

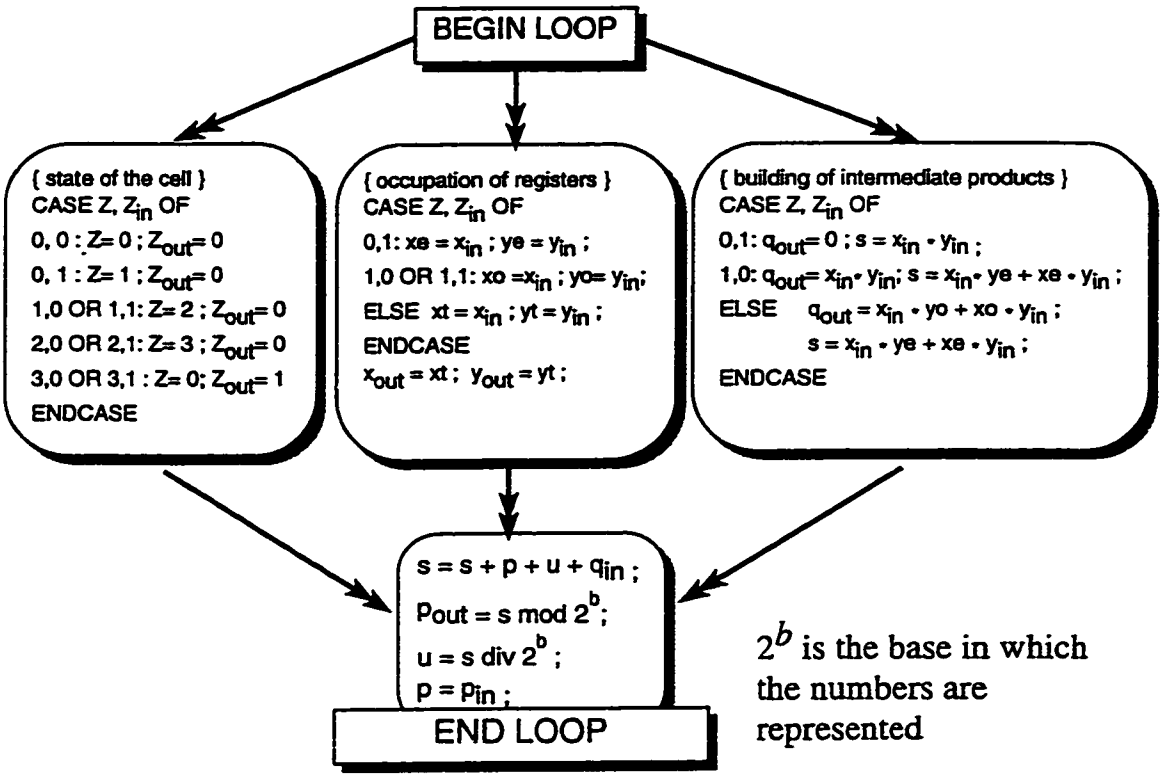


Figure 3.6: The algorithm for a cell behavior

3.6.2 Montgomery Reduction by the Systolic Multiplier

Montgomery's reduction algorithm for the modular multiplication is:

$$MP(x, y, n, r) = x.y.r^{-1} \ mod \ n$$

The algorithm can be rewritten to allow digit-wise processing of large operands as follows:

(Note that $x, y < n < r$, $r = 2^{bl}$ and $gcd(r, n) = 1$)

1. $n'_0 \leftarrow -n^{-1} \ mod \ 2^b$
2. $p \leftarrow x.y$
3. $for \ i = 0 \ to \ l-1 \ do$
4. $begin$
5. $v_i \leftarrow p_i.n'_0 \ mod \ 2^b$ (p_i is the i^{th} digit of the resulting product p)

6. $p \leftarrow p + v_i.n.2^{bi}$

7. *end*

8. *return* p/r; (if $p/r < n$) else *return* $p/r - n$

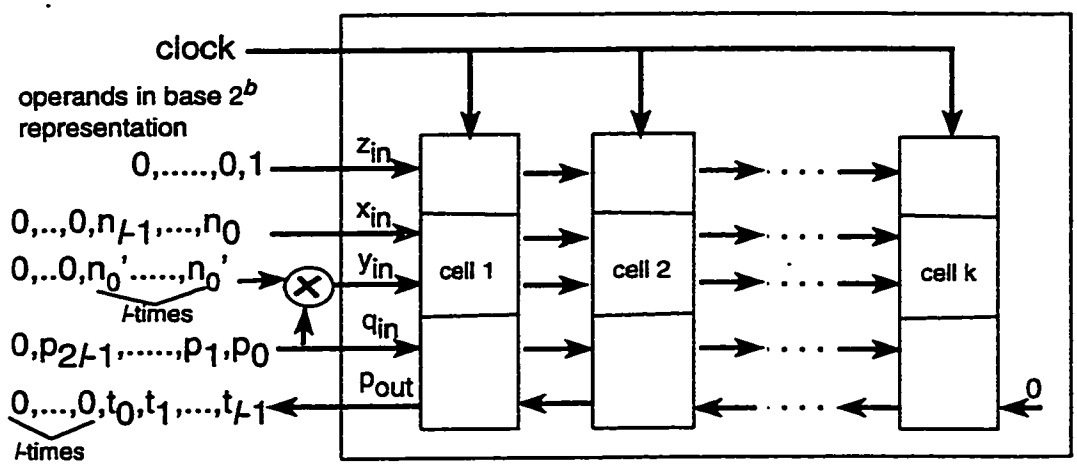


Figure 3.7: The systolic Montgomery reduction

The systolic multiplier performs steps 3 to 7, of the above Montgomery-reduction algorithm. The rest of the algorithm is computed only once, and can thus be computed by a software program. It is assumed that the product digits p_i are available serially, and that the factor n_0' has already been calculated. In this case, v_i is computed by a single multiplier as shown in Figure 3.7. The systolic multiplier receives this product (v_i) and computes step-6, of the Montgomery-reduction algorithm.

According to the algorithm (step-6), p is modified l-times throughout the $for - Loop$ (steps 3 to 7). The implementation proposed by Sauerbrey [31], which is shown in Figure 3.7, does not allow p to be updated as required by the algorithm, and accordingly does not produce correct results. This has been confirmed by VHDL model simulation.

In conclusion, the output of the systolic hardware as shown in Figure 3.7, does not match the MP-algorithm stated above. Sauerbrey [31], has proposed a complete Montgomery modular multiplier based on the hardware described in Figure 3.7. This design is built from two systolic multipliers as shown in Figure 3.8. The first systolic multiplier computes the product $x.y$ (step 2 of the Montgomery-reduction algorithm). The computed product is delayed by one clock cycle and is used as p in steps 5 and 6 in the Montgomery algorithm.

To get the correct result, the p_i used in step 5 must be the one computed by step-6 of the previous loop iteration.

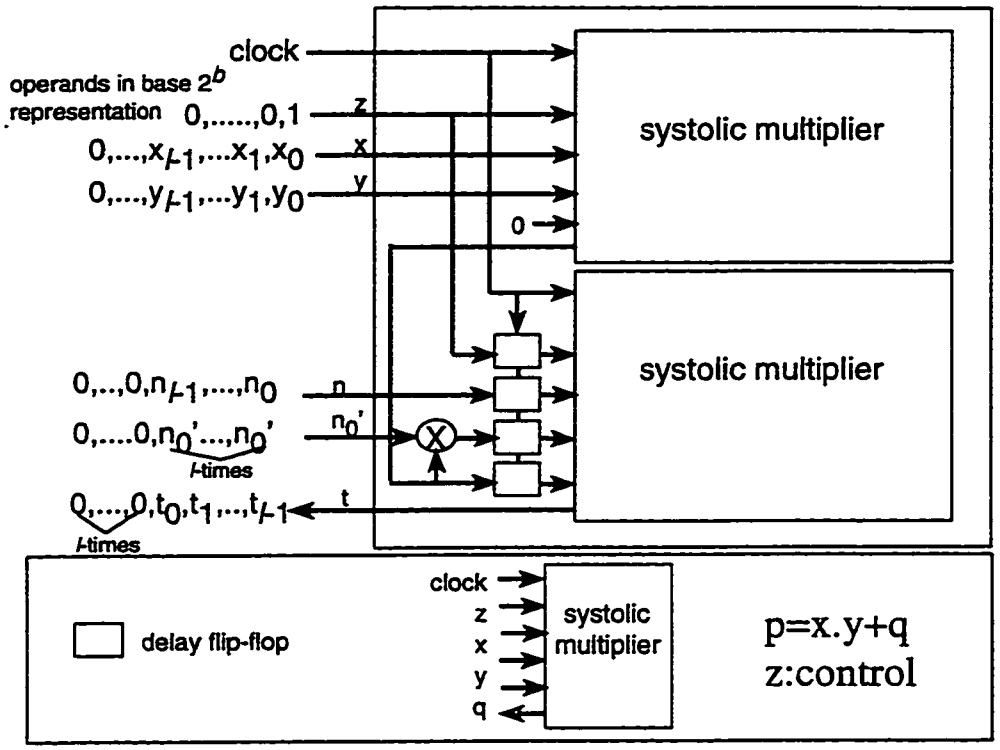


Figure 3.8: Sauerbrey's implementation of Montgomery modular multiplication

3.7 Summary

Several reported implementations of RSA-related hardware have been reviewed. Some are for modular exponentiation, e.g.[31, 46, 47], while others are for modular multiplication, e.g.[27, 39], which is the most time consuming process in the RSA algorithm.

The major operation in the RSA method is the modulo exponentiation which is computed by the repeated squaring algorithm. This repeated squaring algorithm can be improved [46, 47], to perform the required two multiplications in parallel.

Different techniques have been used to calculate modular multiplication. One such technique uses standard integer multiplication followed by a modulo reduction, which is a time consuming process [43]. Merging the modulo reduction with the multiplication can speed-up the process, e.g.[43, 46, 47].

Another modular multiplication method has a logarithmic speed [39], but it uses a very costly hardware. The algorithm used for the computation of $x \times y \ (mod\ n)$, is: $rnd\ (fract((x \times y) \times (1/n)) \times n)$. Assuming that $(1/n)$ is precomputed, this implementation must perform three multiplications in sequence; first, $(x \times y)$, then $(x \times y) \times (1/n)$, and finally $(fract(x \times y) \times (1/n)) \times n$. Note that the function: $fract$ discards the integer part of a real number and returns the non-negative fractional part; and the function: rnd rounds the real number to the nearest integer.

A third modular multiplication technique is developed by P. Montgomery [24]. This technique has been widely used for RSA hardware [23, 27, 28, 29, 30, 31], because of its practical speed and relative simplicity for VLSI implementation [24]. The hardware using Montgomery's modular multiplication [27, 31], is required to perform two multiplications and one addition.

Several reported full RSA designs have been introduced. Detailed investigation of the systolic hardware implementation proposed by J. Sauerbrey has found it to be incorrect. Our thesis work is concentrated on finding a way to make an expandable hardware. This seemed to be feasible using systolic arrays, such as the systolic hardware proposed by J. Sauerbrey [31] (using Montgomery's algorithm for modular multiplication). However, Sauerbrey's systolic implementation needs to be corrected and modified for expandability.

Chapter 4

A Hardware Model of an Expandable RSA Cryptographic System

4.1 Introduction

The security of the RSA cryptographic system depends on the encryption and decryption key size. As the key size increases the security of the system is improved [2]. All RSA hardware implementations, as reported in the previous chapter, are designed to accommodate fixed key sizes. If larger key sizes are needed the hardware must be redesigned.

One of the goals of this work, is to develop an expandable RSA implementation where duplication of well defined bit-sliced hardware will adapt the system to larger key sizes. To incorporate such flexibility, hardware and performance overheads are expected. The proposed expandable RSA system depends mainly on the systolic multiplier used by Sauerbrey [31]. This model has been redesigned and modified for expandability.

In the following, the basic systolic multiplier is described and the Montgomery product algorithm with its implementation as well as modifications for expandability are detailed. Next, modular exponentiation hardware is described and the architecture of the expandable chip is given.

4.2 The Systolic Multiplier

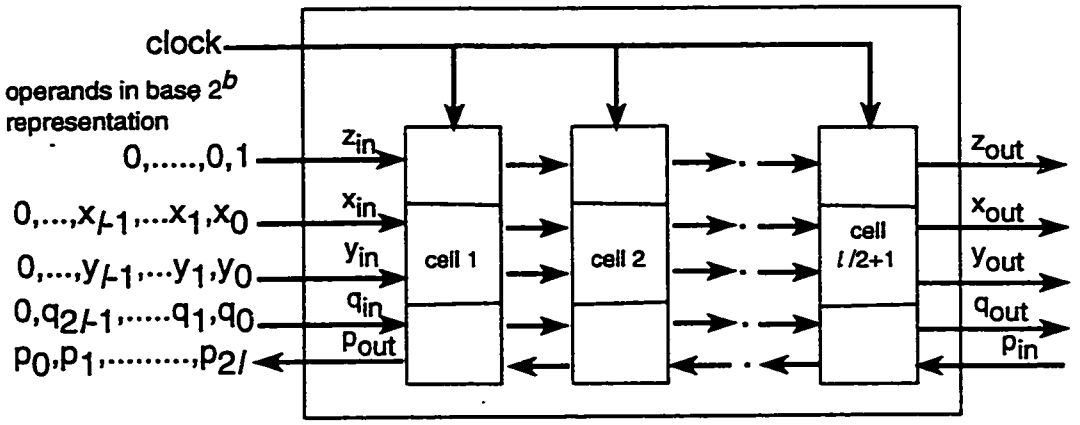


Figure 4.1: The word-serial multiplier (systolic array)

The systolic multiplier is made of a set of cascaded identical cells connected as shown in Figure 4.1. This systolic multiplier can perform the operation: $p = x.y + q$, in a word-serial manner. If x, y and q have l-words, where each word represents a digit in base 2^b, the time required for a complete operation is $2l$ clock cycles [31]. Note that the base can be any power of 2 number, and b is the number of bits in each word. The input z of the systolic multiplier is a control signal which indicates the beginning of the operation. To multiply two operands of l-words, the number of cells required by the systolic multiplier is $\lceil l/2 \rceil + 1$ [31].

This systolic multiplier is chosen because of its expandable capability. Figure 4.1, shows a multiplier for l-digit numbers. If the numbers to be multiplied are increased in size to $2l$-digits, the only required modification on the design is to add another identical systolic multiplier in cascade, as shown in Figure 4.2. Clarification and modeling of each cell in the

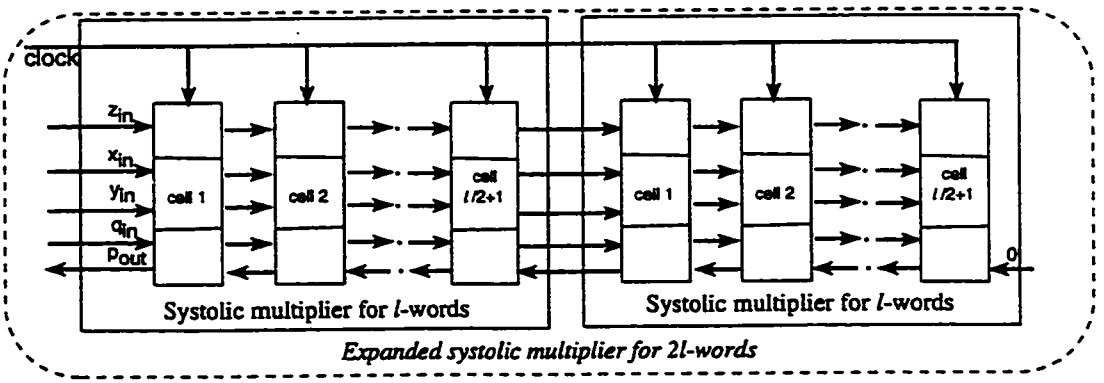


Figure 4.2: Expandability of the systolic multiplier

systolic multiplier is given in the following subsection.

4.2.1 The Basic Cell of The Systolic Multiplier

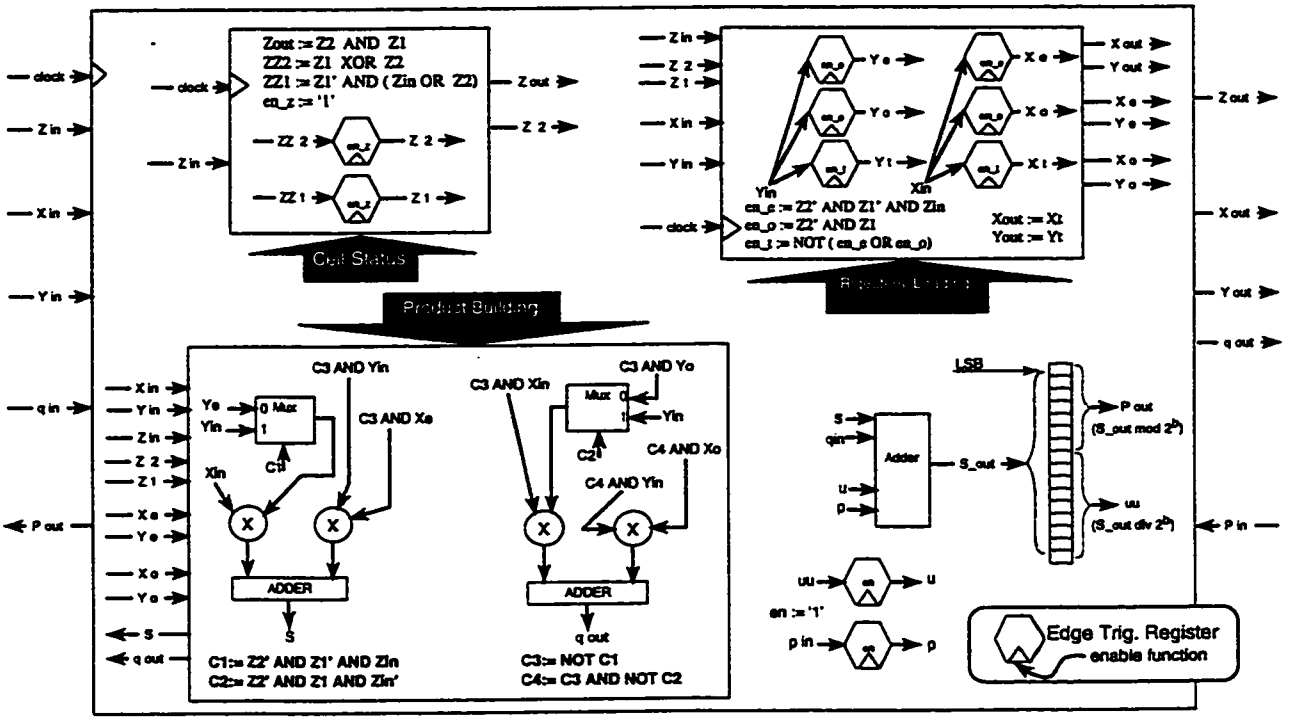


Figure 4.3: Hardware design of the cell

The basic cell of the systolic multiplier is designed to perform the algorithm shown in Figure 3.6. This design is mainly built up from the following:

- Four b-bit parallel multipliers.
- Three adders.
- Ten different registers.
- Two multiplexers.
- Some simple gates: ten AND gates, two OR gates and an XOR gate.

The b-bit parallel multipliers are needed to compute the intermediate products required at each cell. More information about the b-bit parallel multiplier is given in the following subsection.

The registers used in the cell are enable-high positive edge triggered, i.e. if the enable function (en) is high, the register will load data at the rising edge of the clock. The registers in the cell (Figure 4.3) are classified into three types according to their size. The first type is single bit registers, e.g. Z2 and Z1, which control the state of the cell. The second type is b-bit registers, where b is the number of bits in each word. There are seven registers of this type: xe, ye, xo, yo, xt, yt and p. The last type of registers has $(b+2)$-bits, e.g. u. The extra 2-bits are added to account for the carry after addition, e.g. S-out:=S+qin+u+p.

4.2.2 The b-bit Parallel Multiplier

Each cell of the systolic multiplier has four parallel multipliers that perform the required multiplications for that specific cell. Each parallel multiplier is designed to multiply two words, of b-bits each, in parallel. Figure 4.4, shows an example of such parallel multiplier for 4-bit words.

The complexity of the parallel multiplier varies depending on the size of the word (b). In general, the required multiplier hardware can be stated as function of the word size (b), as follows:

1. b^2 2-input AND gates.
2. b Half-adders.
   each half-adder is constructed using an XOR gate and an AND gate.
3. $b^2 - 2b$ Full-adders.
   each full-adder is constructed using two XOR gates, two AND gates and an OR gate.

4.3 Montgomery Product Design

Peter Montgomery [24] in 1985, came up with a smart method to compute modular multiplication without trial division. His method to compute:$z = x.y \ mod \ N$, can be summarized in the following:

1. Choose $R > N$ such that $R = 2^k$, where k is the number of bits in N,
   accordingly R is relatively prime to N.

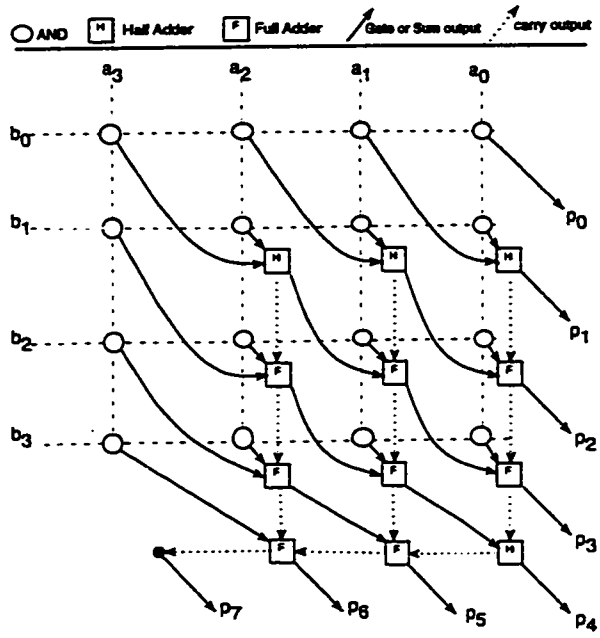


Figure 4.4: Hardware design of 4-bits parallel multiplier

2. Transform x and y to x' and y' (Montgomery's-representation),
   where $x' = xR \ mod \ N$ and $y' = yR \ mod \ N$.

3. Calculate Montgomery-product : $z' = MP(x', y') = x'y'R^{-1} \ mod \ N$.

4. Transform z' to the normal representaticn z, i.e. $z = z'R^{-1} \ mod \ N$.

Steps 1, 2 and 4 can be calculated through a software program since they need to be computed only once. However, step 3, i.e. MP(x',y') is repeated to perform exponentiation and it can be calculated as follows:

1. $P = x' \times y'$;

2. $U = P + N \times \ (P \times N' \ mod \ R)$;

3. $S = U/R$;

4. $MP = S \ (if \ S < N)$ or $MP = S - N \ (if \ S \geq N)$;

By choosing R to be a power of 2 number ($R = 2^k$), $mod \ R$ and division by R operations can be computed inexpensively. This method to compute MP(x',y') can be organized in a serial manner, as shown in Figure 4.5. This algorithm is well-suited for a systolic multiplier implementation similar to the one described in the previous section.

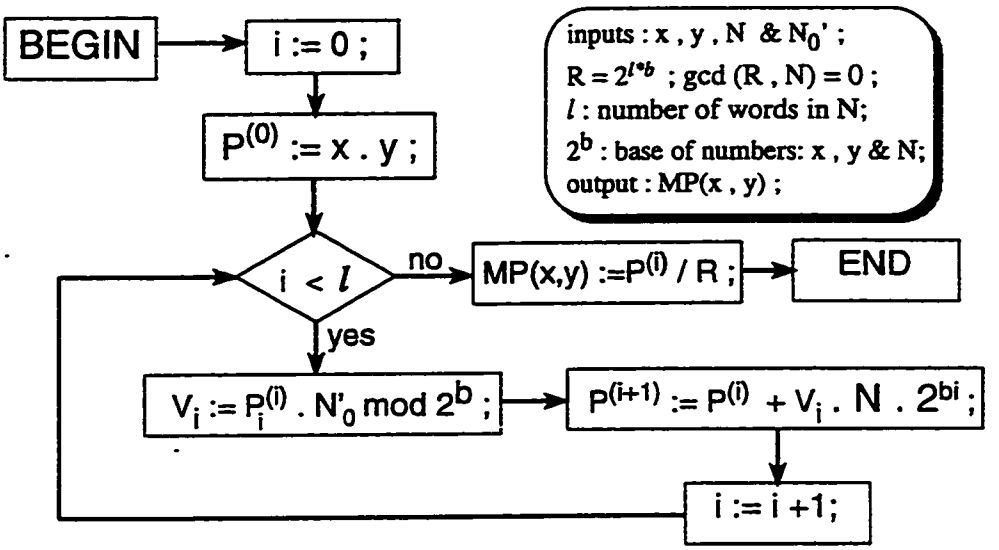


Figure 4.5: The MP-algorithm (Montgomery Product)

4.3.1 Montgomery Product Implementation

To implement the Montgomery product algorithm (Figure 4.5) using the systolic multiplier shown in Figure 4.1, a signal flow graph is developed to describe the flow of data. The signal flow graph for a 4-word number is shown in Figure 4.6.

Note that two types of processors are required, one is a parallel multiplier, and the other is a systolic multiplier. The output starts coming out after $2l$ clock cycles. However, the first l-digits of the output will be discarded to account for the division by R, and accordingly the MP result will be serially computed after $3l$ clock cycles.

The hardware implementation derived from the signal flow graph (for $l = 4$ words) is shown in Figure 4.7. The full MP result will be available after $4l$ clock cycles. Two types

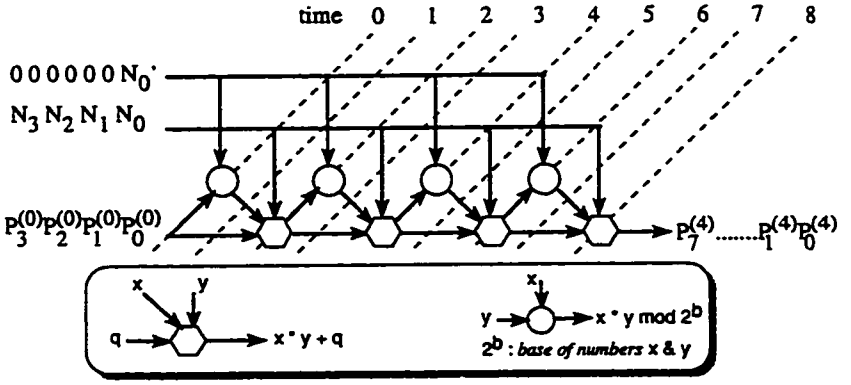


Figure 4.6: The signal flow graph

of registers are used for proper data synchronization: T and 2T, which delay data by one and two clock cycles respectively. The overall number of registers required for an l-words design is $(6l - 3)$. The number of parallel multipliers used is l, while the number of systolic multipliers used is $l + 1$. The extra systolic multiplier is not shown in Figure 4.7, but it is necessary to compute $p^{(0)}$.

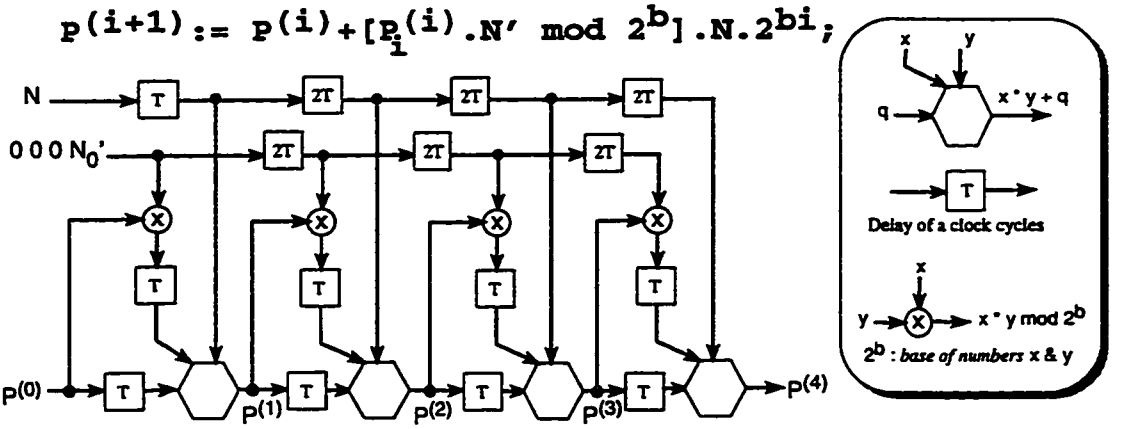


Figure 4.7: The signal flow graph MP implementation (parallel hardware)

4.3.2 Expandability of the parallel MP Implementation

To expand the design shown in Figure 4.7, not only should the number of systolic multipliers be increased, but also the size of each systolic multiplier must increase. This is due to the fact that the number of cascaded stages of the systolic multipliers depends on the number of words (l). Thus, such design does not allow regular linear expandability.

For example, if the design needs to be expanded to handle $2l$-words, the expansion is performed in both horizontal and vertical directions. The horizontal expansion is to let the number of systolic multipliers increase to $2l$, while the vertical expansion is to add hardware to each systolic multiplier to accommodate $2l$-words instead of l. Thus, expandability for such architecture is not linear (Figure 4.8). This makes developing a standard expandable chip, using this approach, unachievable.

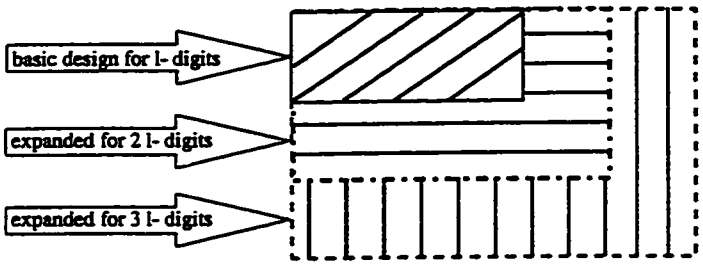


Figure 4.8: Expandability of the parallel implementation

4.3.3 The Expandable MP Design

For linear regular expandability of the MP implementation, the design must be reorganized for serial instead of parallel processing. This can be achieved by projecting all systolic multipliers into one and projecting all parallel multipliers into one, i.e. using only one parallel multiplier and one systolic multiplier for the whole process as shown in Figure 4.9.

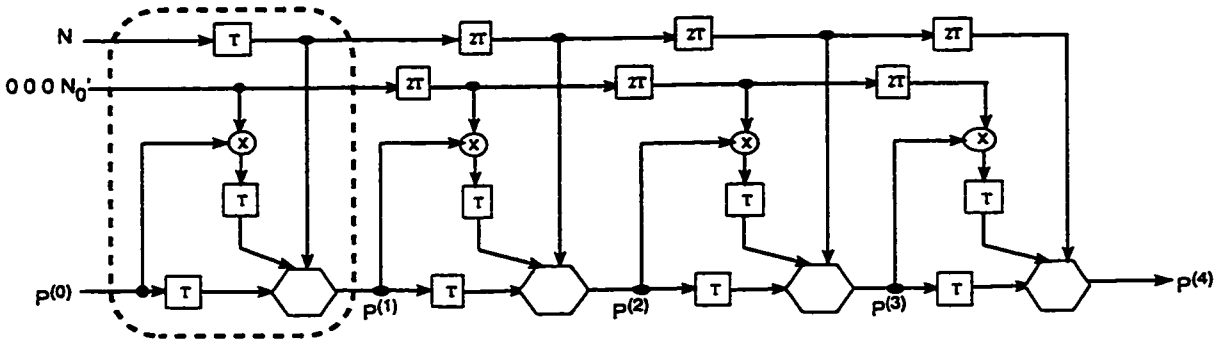


Figure 4.9: Projecting all parallel and systolic multipliers into one

The following example clarifies this idea. Assuming that l = 4-words, and $p^{(0)}$ is pre-computed, let the words of $p^{(0)}$ be fed in a word-serial manner to the projected processor and the outputs be saved in an output register, as shown in Figure 4.10. After $2l$ clock cycles all words of $p^{(1)}$ will be available in the output register. Then, $p^{(1)}$ is serially fed to the processor with the other inputs N and N_0', properly synchronized. After additional $2l$ clock cycles, all digits of $p^{(2)}$ will be available in the output register. Likewise, $p^{(3)}$ and $p^{(4)}$ are computed using the same procedure allowing $2l$ clock cycles for each.

The time required for each $p^{(i+1)}$ to be available at the output register is $2l+1$ clock cycles, including an extra cycle for proper synchronization. The digits of N must be synchronized to those of $p^{(i)}$ and delayed by two more clock cycles. The single word V_i (Figure 4.5) is calculated using a parallel multiplier to multiply N_0' with $p_i^{(i)}$, and discarding the most

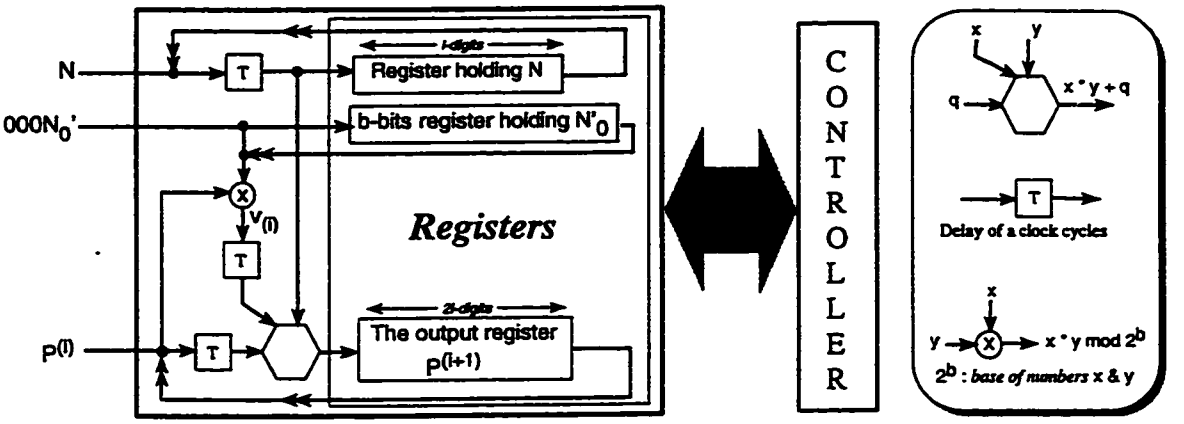


Figure 4.10: The expandable serial MP implementation

significant word.

Figure 4.10, assumes that $p^{(0)}$ is already computed. However, to compute $p^{(0)}$ with the same hardware, additional multiplexors should be added in the controller and more time is required to complete the process.

To allow for expandability, the controller is organized using shift registers and multiplexors, to make the registers expandable, as shown in Figure 4.11. The systolic multiplier is made expandable by passing its cascading signals as external interface signals as shown in Figure 4.2.

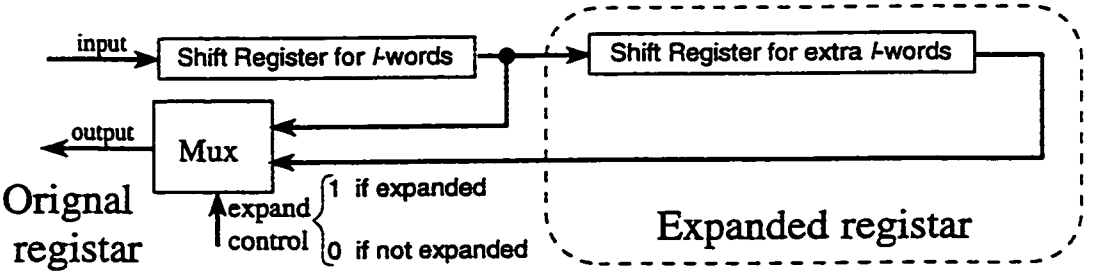


Figure 4.11: Expandable shift registers design

Note that the basic processor chip differs from its expansion. In the expansion chip, the b-bit parallel multiplier and some registers are not needed. Thus, the complete design can be organized as the basic MP-processor and any number of expanded processors depending on the required key size, as shown in Figure 4.12. The basic MP-processor for a key size of l-digits, consists of the following:

- A systolic multiplier

- Ten multiplexers in the data processor
- Three registers of b-bits in the data processor
- A parallel multiplier
- Seven multiplexers in the controller
- Six registers of b-bits in the controller

The expanded MP-hardware for an added l-digits consists of the following:

- A systolic multiplier
- nine multiplexers
- nine registers of b-bits

For a key size of l-digits, the overall time required for completing the MP-process and start getting MP output is: $(2l+1)(l+1) = 2l^2 + 3l + 1$ clock cycles.

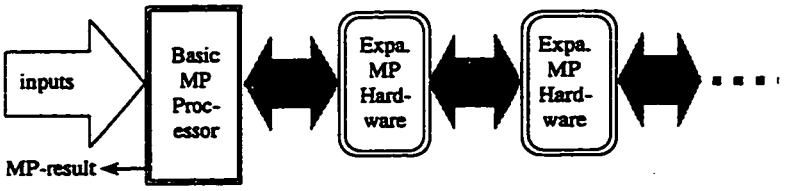


Figure 4.12: The expandable MP system

4.4 The Modular Exponentiation System

The modular exponentiation system uses the improved repeated squaring algorithm shown in Figure 3.2. This algorithm can perform the two required modular multiplications in parallel. It is also applicable for the Montgomery modular multiplication with some additional pre-computations. These pre-computations transform the numbers from its ordinary representation to Montgomery's representation. The algorithm used for Montgomery modular exponentiation is shown in Figure 4.13. Note that the initial computation of X is to be computed in software. The rest of the algorithm is more efficiently performed in hardware because of the excessive number of required repetitions.

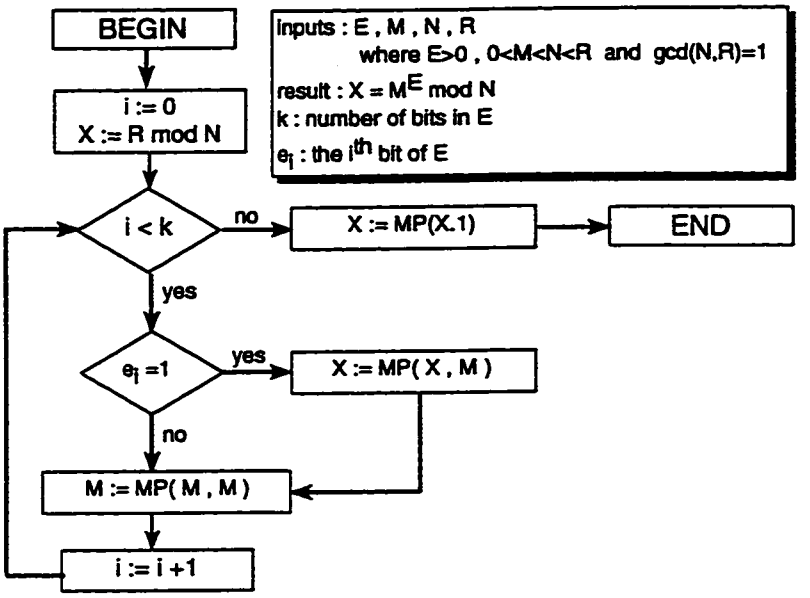


Figure 4.13: The Montgomery modular exponentiation algorithm

4.4.1 The Basic Exponentiation Processor

For highest speed, the algorithm shown in Figure 4.13, is implemented using two MP-processors to compute: X and M in parallel. A controller is built up of eleven multiplexers and a register to hold the bits of the exponent E. One multiplexer is used to control the MP-processor updating the X-value, depending on the values of the exponent bits. Nine other multiplexors are used for the reason of loading and processing data. One more multiplexer is needed for allowing expansion of the register holding the exponent E.

The basic blocks of the exponentiation processor are shown in Figure 4.14. If the basic processor is to be expanded, an expansion unit is designed to be connected to it. Note that the controller, of the basic processor, is expanded by only expanding the exponent register.

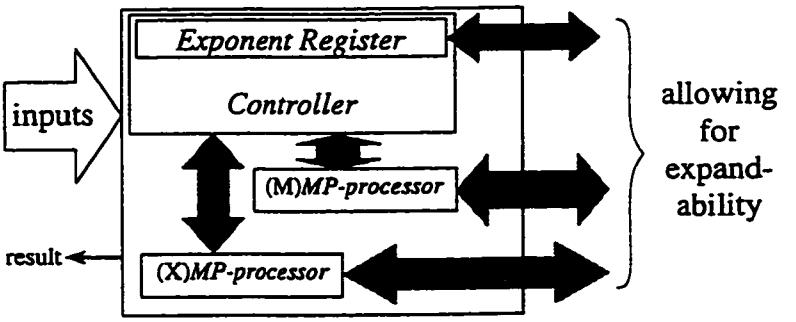


Figure 4.14: The basic exponentiation processor

4.4.2 The Expansion hardware

Figure 4.15, shows the proposed expansion chip when the number of digits is to be doubled. For example, if the basic chip can handle l-words, adding this expansion hardware will allow processing of $2l$-words; i.e. each added expanded chip will let the design accommodate an additional l-words.

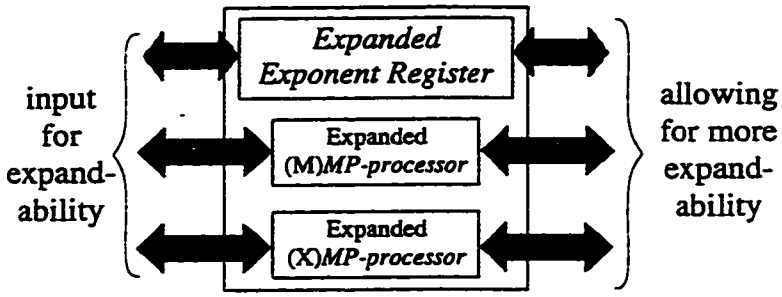


Figure 4.15: The expansion hardware

Each expansion hardware incorporates two expandable MP processors and a register to accommodate the expandability of the exponent, as shown in Figure 4.15. The two expandable MP modules are completely identical and will be clarified in the following subsection.

4.4.3 The Expandable MP Module

The idea of expandability is mainly that of expanding the systolic multiplier and expanding the registers. Expanding the systolic multiplier is performed by adding another systolic multiplier with $\lceil l/2 \rceil$ cells to accommodate the multiplication of numbers with $2l$-words, as shown in Figure 4.2. Expanding the registers is performed by adding more shift registers as shown in Figure 4.11.

For the expandable MP module, four additional registers are required for the data path and five are needed for the controller. These nine registers require nine multiplexors to allow for further expandability, as shown in Figure 4.16.

4.5 Summary

In this chapter, a new hardware model of an expandable RSA cryptographic system is proposed. The new hardware corrects the problem in the design reported by Sauerbrey [31],

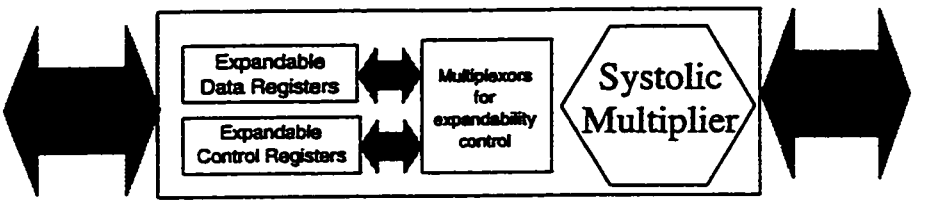


Figure 4.16: The expanded MP module

in addition to allowing for its expandability. Targeting the maximum possible speed, the used exponentiation algorithm allows parallel computation of two Montgomery product at the expense of added hardware.

Chapter 5

Other Implementations

5.1 Introduction

The hardware model of the expandable RSA cryptographic system, as proposed in this Thesis, has been described in the previous chapter. For comparison purposes, two other RSA designs have been modeled. They differ mainly in the techniques used for implementing modular multiplication. In both cases, the modular exponentiation algorithm is the same. These two implementations are the merged exponentiation hardware and the add/subtract exponentiation design [24, 43].

5.2 The Merged Exponentiation Hardware

The merged exponentiation hardware depends on Montgomery's technique for modular multiplication. However, Montgomery's multiplication algorithm is reorganized to merge the multiplication with reduction for each digit. This algorithm was found to be the best among several other merged algorithms compared for both speed and hardware complexity, as outlined by C. K. Koc in 1996 [24]. It deals with large numbers by dividing them into l-words (digits) with each word having b-bits.

5.2.1 The Merged Montgomery Product Algorithm

The merged Montgomery product (MP) algorithm is shown in Figure 5.1 [24]. The algorithm computes: $\text{MP}(x, y) = xyr^{-1} \ mod \ n$, where: $r > n > x, y$ and $gcd(r, n) = 1$.

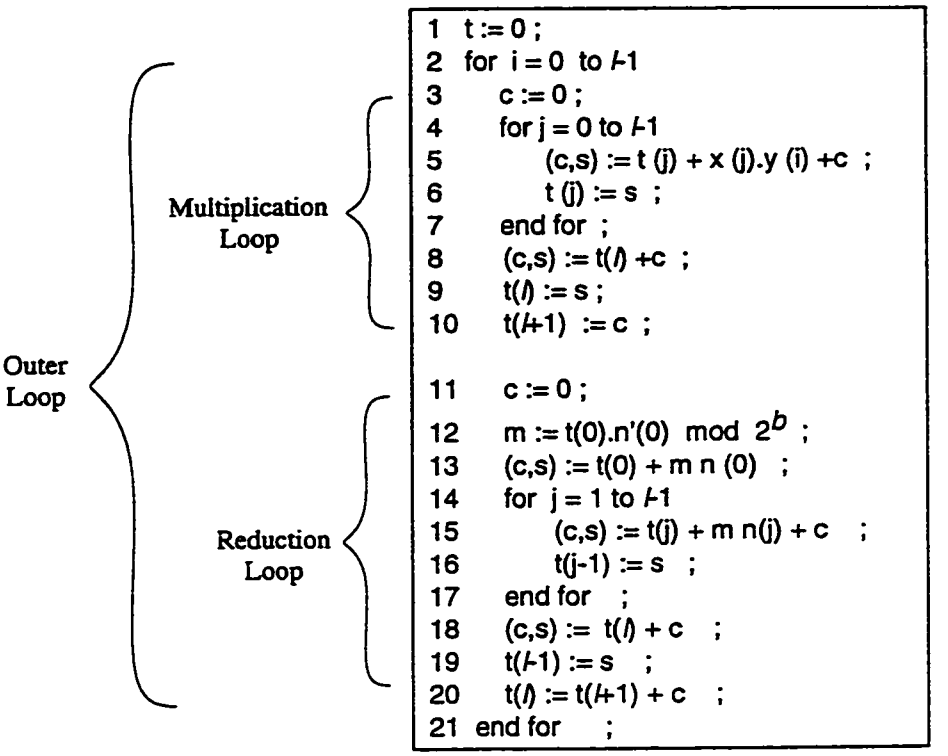


Figure 5.1: The MP merged algorithm

It can be observed (Figure 5.1) that the MP algorithm consists of an outer loop and two inner loops. The outer loop controls the loop-index 'i' which fixes the multiplication operand 'y(i)' for the inner first loop. The two inner loops represent the multiplication and the reduction operations. The multiplication loop performs a digit-wise multiplication of 'y(i)' by 'x'. The MP-algorithm of Figure 5.1 assumes that all digits of 't' will be first computed by the multiplication loop, then these digits are processed by the reduction loop.

Careful study of the above algorithm shows that the reduction process does not need to wait until all digits of the multiplication process are computed. Accordingly, the two inner loops can be merged into one inner loop in such a way that each digit of t computed by the multiplication process is passed to the reduction process in a digit-wise manner. It can also be observed that the multiplication loop runs l times (0 to l-1), while the reduction loop runs l-1 times (1 to l-1). To accommodate this, several conditionals are used within the body of

the merged loop whose index is made to vary from 0 to l. The reorganized MP-algorithm is shown in Figure 5.2.

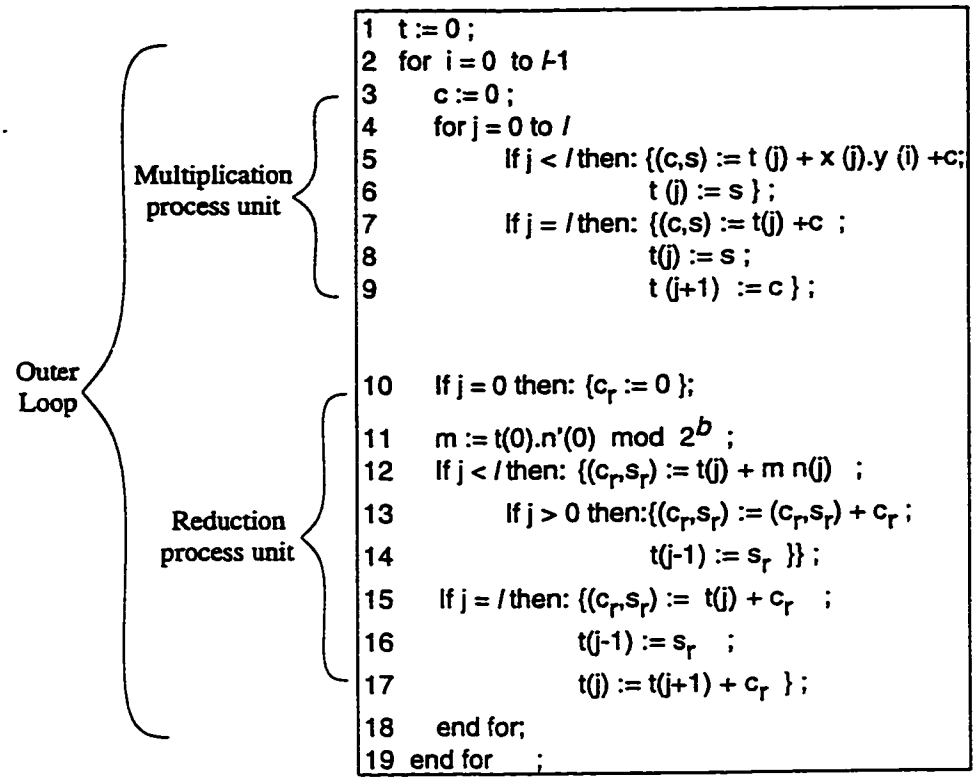


Figure 5.2: The reorganized MP merged algorithm

In the MP-algorithm shown in Figure 5.1, the same sum and carry (c,s) variables are used for both the multiplication and the reduction loops since both loops are disjoint. However, in the reorganized algorithm (Figure 5.2), different sum and carry variables are used for the multiplication and the reduction processes.

It is to be noted that, the reorganized MP-algorithm shown in Figure 5.2 is more efficient in terms of hardware cost. It does not require registers to hold the intermediate values of t, computed by the multiplication process. A general implementation model of the reorganized MP-algorithm (Figure 5.2) is shown in Figure 5.3.

5.2.2 The Merged MP Implementation

Considering Figure 5.3, the implementations of the multiplication and the reduction units are described in the following subsections. The controller and the data path are shown in

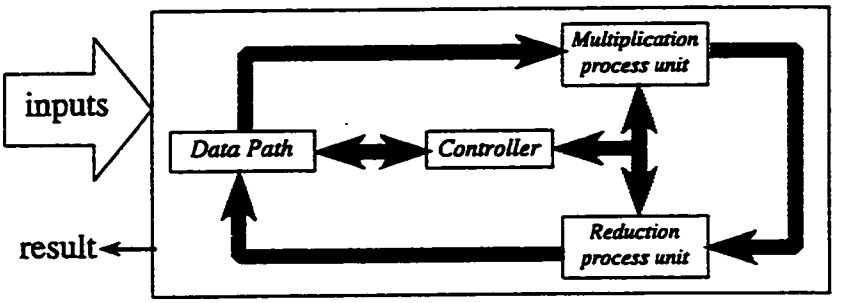


Figure 5.3: The MP merged algorithm implementation model

Figure 5.4 with their connections to the multiplication and reduction units.

The controller consists of a shift register and two OR gates. The data path is mainly made of data registers and multiplexers. Mux-1 and Mux-2 are for loading new values of inputs. Mux-3 resets the register to zero at the beginning of the process. Mux-4 is to perform step 1 of the MP-algorithm (Figure 5.2). The size of the shift register following Mux-4, is one stage less than the shift registers following Mux-1 and Mux-2. This is done to accommodate the loading required in steps 14, 16, and 17, of the MP-algorithm (Figure 5.2).

The signal 'start' triggers the complete MP-process. Loading the values of x, y, is controlled by signal 'load'. Another data loading signal is 'n_load', which is responsible of loading the modulus n. Although the number of clock cycles required to load the input data x, y and n is the same, two different loading signals are used, namely 'load' and 'n_load'. This is due to the fact that data values of x and y change depending on the message being encrypted/decrypted, while the value of the modulus n is fixed depending only on the encryption/decryption key.

The multiplication process requires multiplying each digit of y by all the digits of x, as shown in Figure 5.2. Fixing a digit of y ($y(i)$) for each multiplication process is performed with the use of an enable-shift-register, i.e. shifting the digits of y is enabled only once for each outer loop iteration. The digits of x are fed and rotated through the use of a shift register and Mux-1. The signal 'z_1' indicates that the output of the multiplication process (t_{out1}) is computed for this y-digit. The bus t_{out1} is passed to the reduction unit as 't_{in2}'. The reduction unit will process the data values of t_{in2} as well as the fixed digit of n'_0. The signal 'z_2' indicates that the result of the reduction process using this digit of y is available on 't_{out2}'. This data value (t_{out2}) is then fed back to the multiplication process as the bus 't_{in1}' to update the value of 't', as required by the MP-algorithm shown in Figure 5.2. The

signal 'z_{out}' indicates that the final result value is found at 't_out' and that will end the MP process.

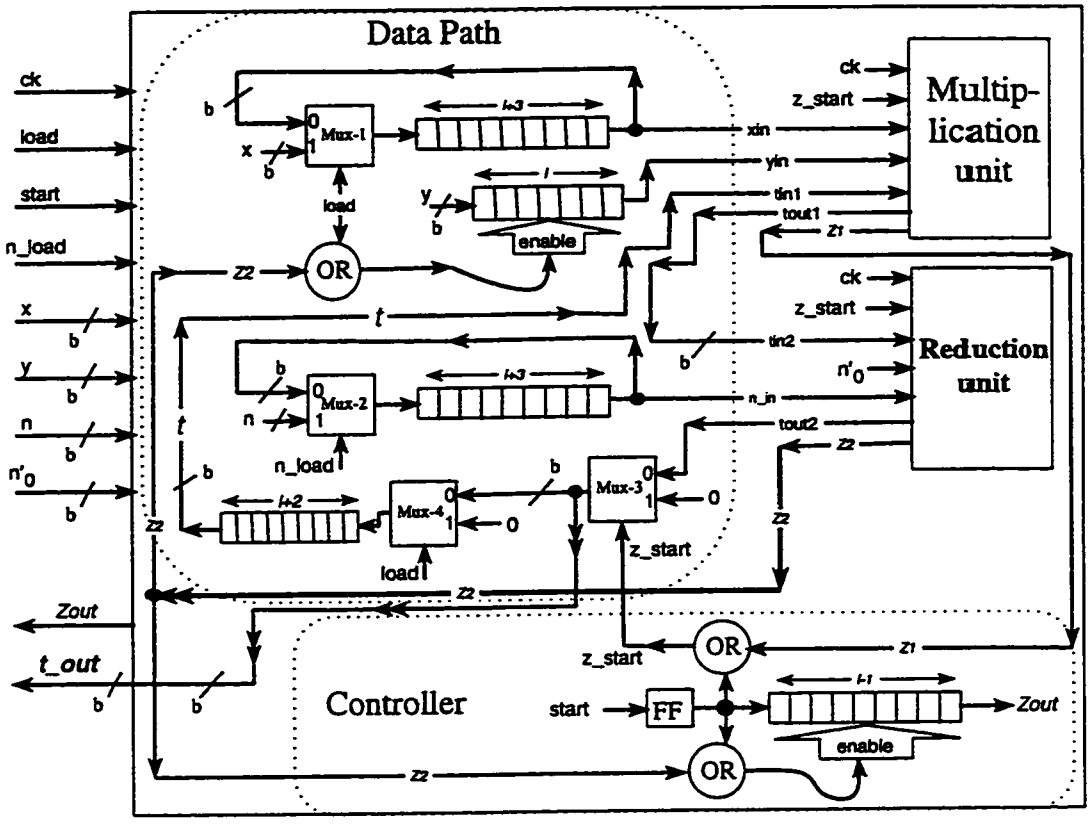


Figure 5.4: The merged MP multiplier implementation

5.2.3 The Multiplication Loop Implementation

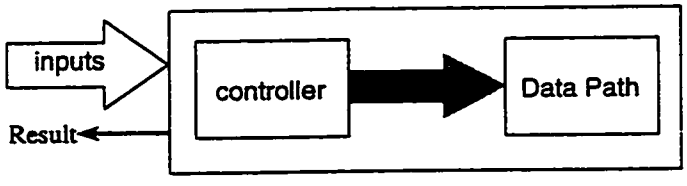


Figure 5.5: The multiplication process implementation outline

Steps 3 to 9 of the MP-algorithm shown in Figure 5.2, form the multiplication process. The function of this process is to cumulatively compute the product digits of xy into t. The implementation for this multiplication process is constructed using a controller and a data path as outlined in Figure 5.5.

Figure 5.6 shows the components and connections of the controller and the data path in a detailed manner. The multiplication process computes the product values of t, by multiplying the y-digit by all the digits of x.

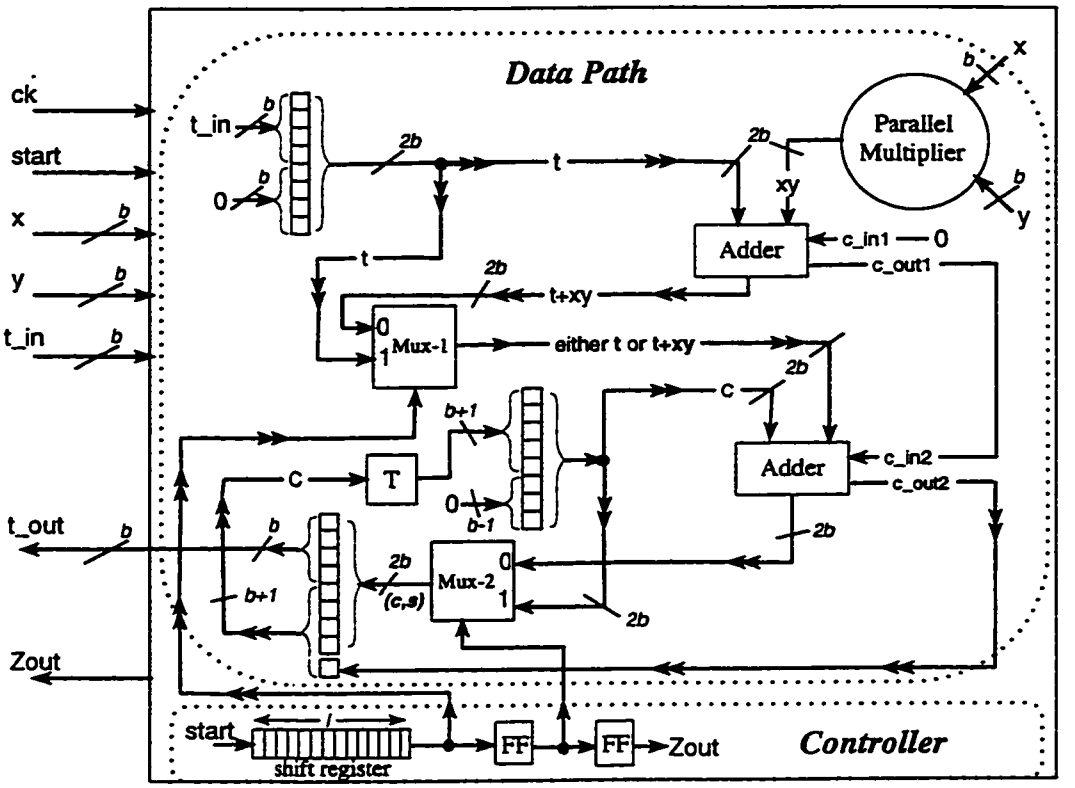


Figure 5.6: The multiplication process implementation

The controller is mainly a shift register that has a control signal 'start' to trigger the process. The output of this shift register controls the hardware to perform step-7 and step-8 of the MP-algorithm (Figure 5.2). The flip-flop following the shift register controls step-9 of the MP-algorithm. The required number of iterations is $(l + 2)$, where l is the number of words (digits). The first l-iterations allow the hardware to perform steps 3 through 6, followed by an iteration to perform steps 7 and 8. The iteration number $(l + 2)$ performs step-9. The signal 'z_{out}' is an indicator that the result is ready.

5.2.4 The Reduction Loop Implementation

The reduction process hardware performs steps 10 to 17, of the MP-algorithm shown in Figure 5.2. The hardware is made of two main parts, a controller and a data path. The

controller consists of a shift register and a simple OR gate. The data path is constructed of two parallel multipliers, three multiplexers and two adders. The connections between the controller and the data path are detailed in Figure 5.7.

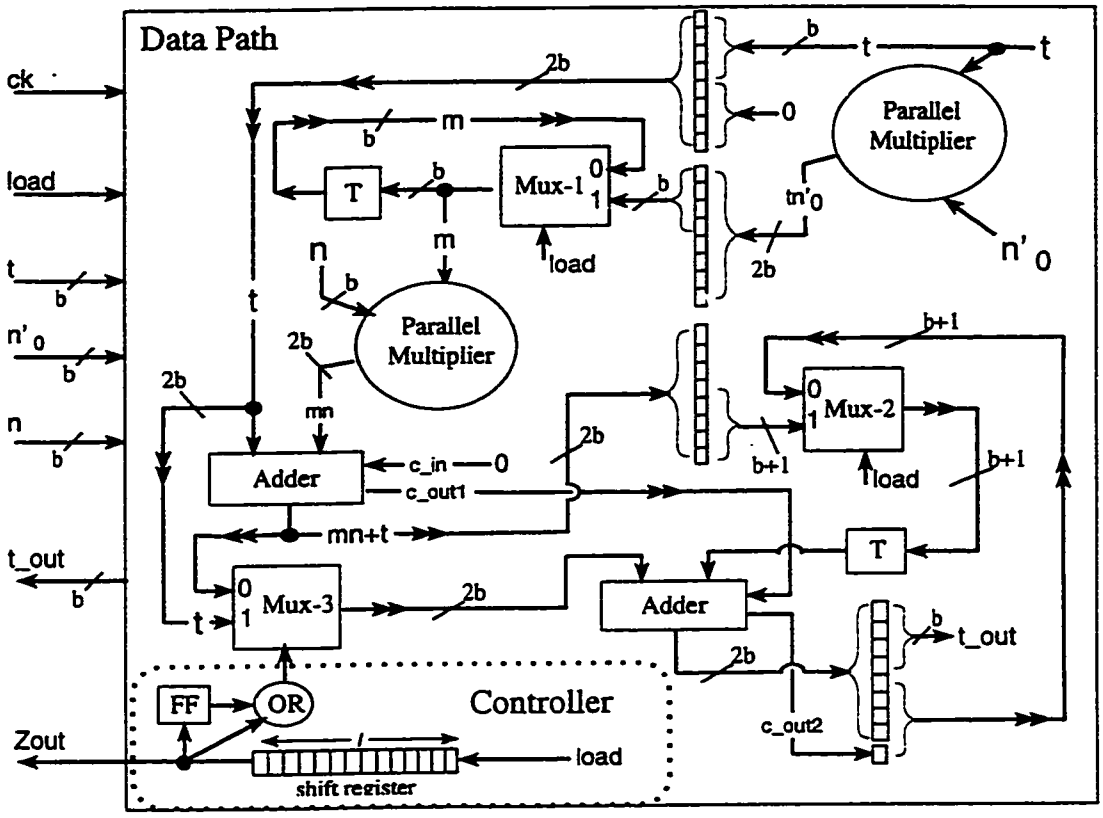


Figure 5.7: The reduction process implementation

The two parallel multipliers compute tn'_0 (step 11 in the algorithm) and mn (step 12). The value of 'm' is the first b-bits of (tn'_0) because of the mod 2^b operation (step-11). Fixing 'm' during the rest of the loop, is achieved by Mux-1 which is controlled with signal 'load'. The signal 'load' is '1' only at the first iteration indicating the start of the reduction loop process.

The output of the shift register is '0' for the first $(l-1)$ iterations, allowing the design to perform steps 10 to 14, of the MP-algorithm (Figure 5.2). Then, when the output of the shift register is '1', step-18 is performed, followed by step-17. The signal 'z_{out}' indicates that the result is ready.

5.2.5 The Merged Exponentiation Implementation

The merged Montgomery exponentiation algorithm is similar to the repeated squaring algorithm shown in Figure 4.13. The modular exponentiation hardware is constructed of two MP-multipliers, a controller and a shift register that stores the exponent, as shown in Figure 5.8. The components required for the controller are seven multiplexers, seven shift registers and four simple gates.

The time required to load the data is $l + 2$ clock cycles. The time required for the first result digit to be computed is l^3+4l^2+l-2 clock cycles. The complete result needs additional l clock cycles to be performed.

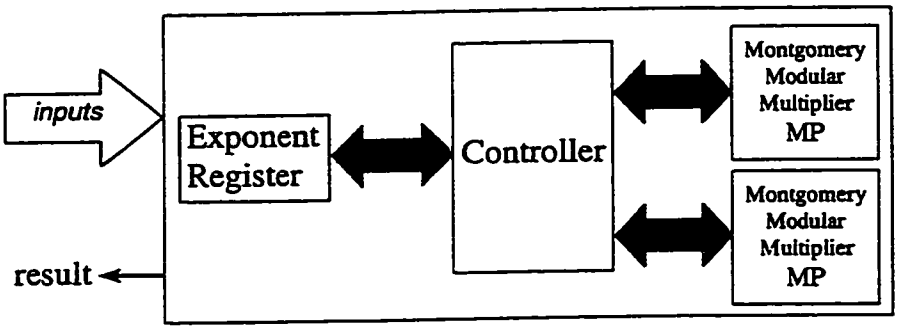


Figure 5.8: The merged modular exponentiation hardware

5.3 The Add/Subtract Exponentiation Design

The add/subtract design deals with the input data as a single number rather than a multi-precision one. Accordingly, the hardware is designed to process all the bits, without dividing them into smaller size words (digits). The multiplication and reduction are performed as addition and subtraction, as shown Figure 5.10. This algorithm has been compared with other add/subtract algorithms, in terms of the required number of clock cycles and it has been reported as the best one [43].

The algorithm is used mainly to perform: $x.y \ mod \ n$, which is modeled in hardware, as shown in Figure 5.9. In the following section, the modular add/subtract reduction unit is described.

5.3.1 The Add/Subtract Reduction Unit Implementation

The add/subtract reduction unit implements the algorithm shown in Figure 5.10. This algorithm has three comparisons which are modeled in hardware using the multiplexers: Mux-1, Mux-2 and Mux-3.

The multiplication of 'p' by 2 is performed by simply left-shifting the bits of 'p', and adding a zero-bit as the LSB (least significant bit), as clarified in Figure 5.11. This shifting process causes the result '2p' to be $(b+1)$-bits, which requires $(b+1)$-bits subtractor to perform the $(p-n)$ operation. An extra zero-bit is appended to 'n' as the MSB (most significant bit). The result is the b-bits coming out of Mux-3, ignoring the MSB, as shown in Figure 5.11.

5.3.2 The Add/Subtract Multiplication Implementation

The modular add/subtract multiplication implementation is outlined in Figure 5.9. The reduction unit is modeled as explained in the previous subsection. The register shown in Figure 5.9, is to hold the input 'x' and process its bits serially starting with the MSB. The controller is built of a multiplexer, a data register and a shift register. Figure 5.12 shows the reduction unit as a block and clarifies the controller and the register with their connections to the reduction unit. The signal 'load' indicates the loading state for the input data values.

For example, the number 'x' is loaded at one clock cycle to the parallel load shift register. Then, it is processed bit by bit with the MSB first, as required by the reduction unit. The values of 'y' and 'n' are fixed during the loop. The signal 'z_{out}' if '1', indicates that the output of the modular multiplier is completely computed.

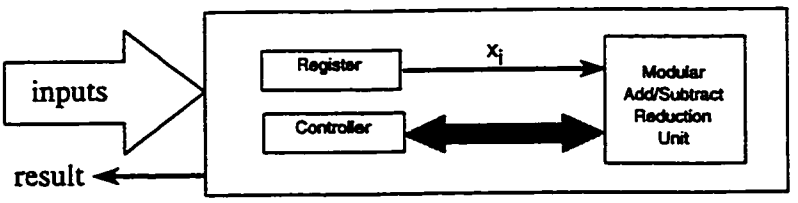


Figure 5.9: The add/subtract modular multiplication hardware

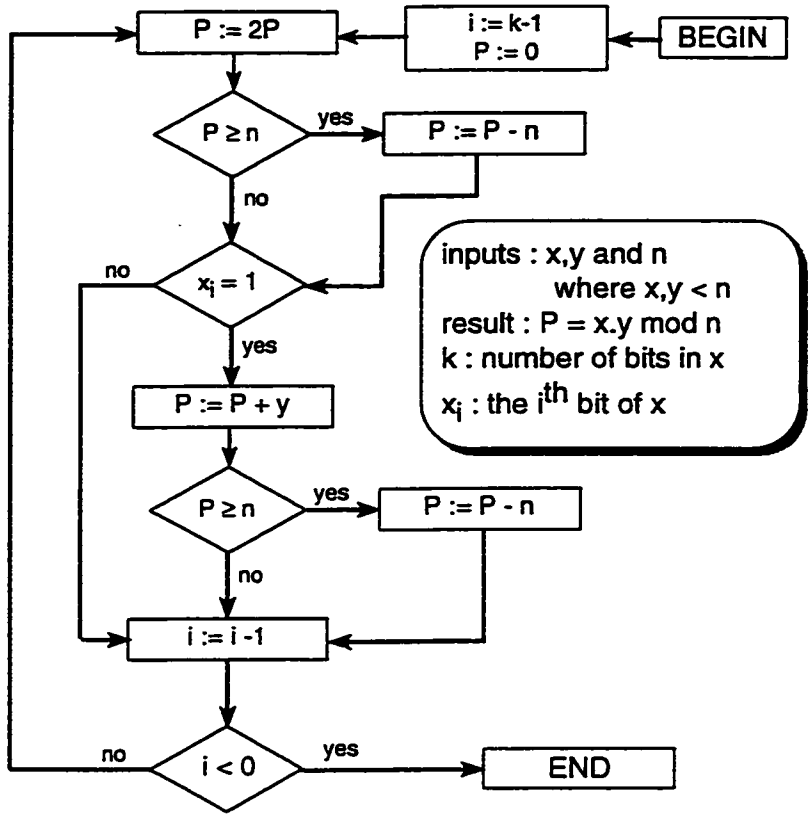


Figure 5.10: The add/subtract multiplication algorithm

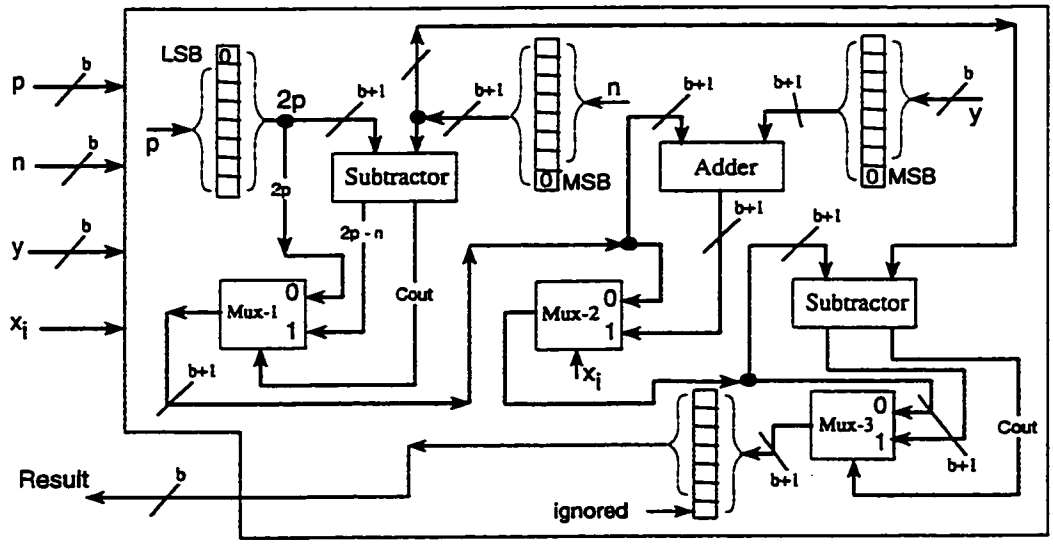


Figure 5.11: The modular add/subtract reduction implementation

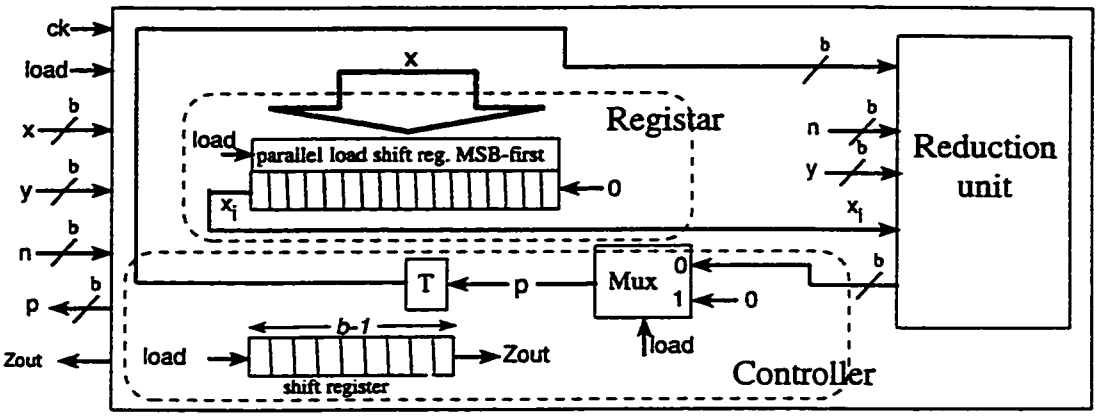


Figure 5.12: The modular add/subtract multiplication implementation

5.3.3 The Modular Add/Subtract Exponentiation Implementation

The modular exponentiation: $X = M^E \ mod \ n$, is performed by the modified repeated squaring algorithm shown in Figure 3.2. To implement this algorithm, the requirements are: two modular multipliers, a shift register (to hold the exponent), and a controller, as outlined in Figure 5.13.

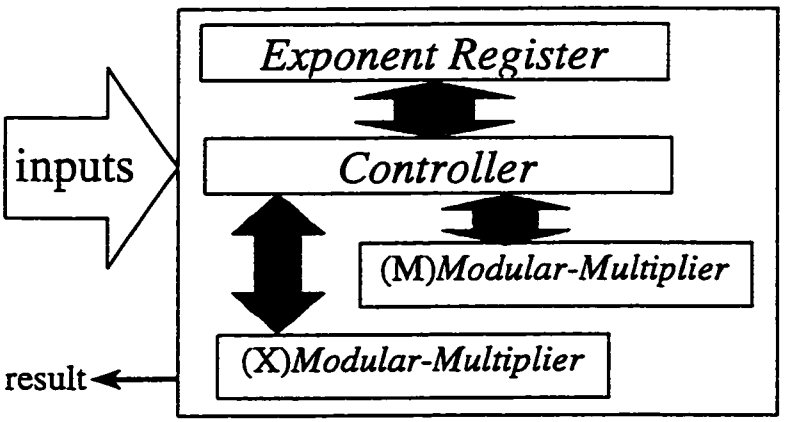


Figure 5.13: The modular exponentiation outline

The exponent parallel load shift register is to load the exponent 'E' at the first clock cycle. Then, the exponent is processed bit by bit, with the LSB first.

The controller is structured from: two delay registers, four simple gates, and four multiplexers used for the loading purpose. Figure 5.14, shows the connections of the controller and the two modular multipliers.

The signal 'loading' is to enable the shift register. It indicates that the complete modular multiplication process is completed, and a new e_i value is required.

The flip-flop enabled with the signal 'load-x' is to store the result of 'x_{out}', if the exponent bit e_i is '1'. While the other flip-flop enabled with the signal 'loading', is to store the new M when ever it is ready.

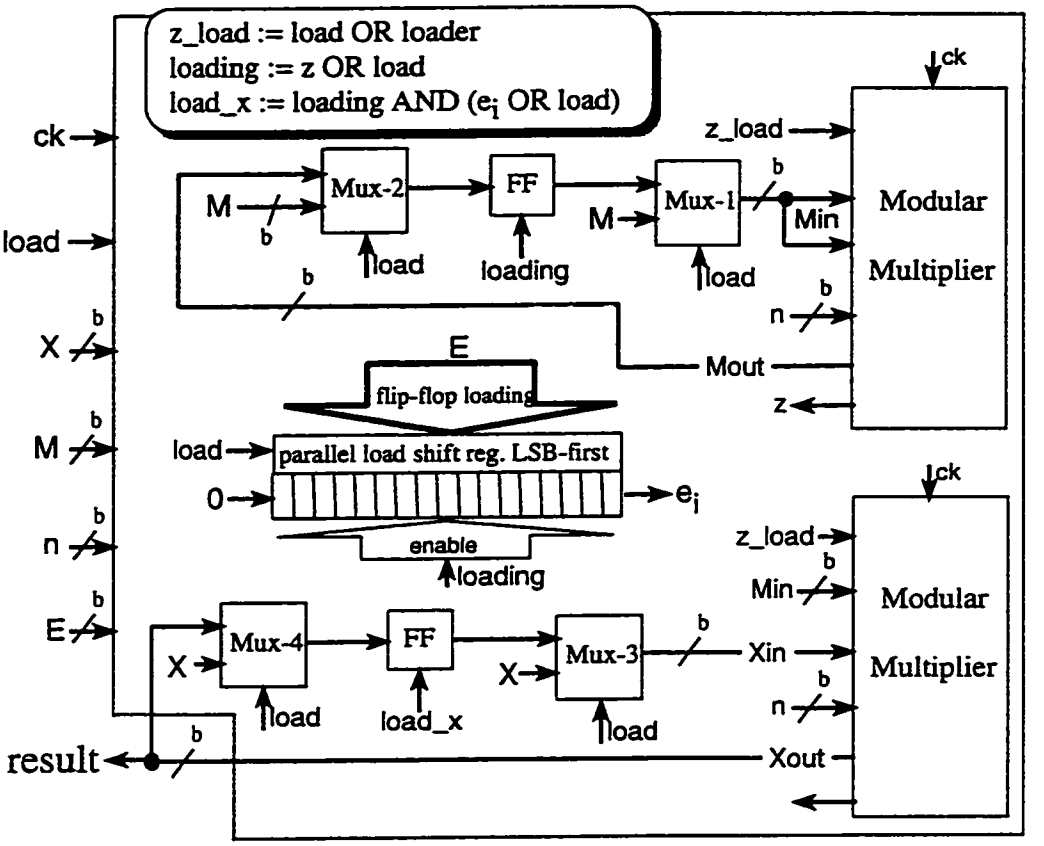


Figure 5.14: The modular add/subtract exponentiation implementation

5.4 Summary

Two modular exponentiation implementations are described in this chapter. Both of them use the repeated squaring algorithm for modulo-exponentiation. However, the techniques used to compute modular multiplication are different. One implementation is based on Montgomery's method for modulo multiplication but by merging the multiplication and reduction processes together. The other modulo-multiplication method is basically transforming the multiplication into addition and the reduction into subtraction.

Chapter 6

Modeling and Analysis

6.1 Introduction

The expandable RSA hardware is modeled using VHDL (Very high speed integrated circuits Hardware Description Language), to be verified and simulated. The developed VHDL model is parameterizable in terms of the number of words (digits) and the size of each word in the encryption/decryption key. In addition to the expandable design, two other implementations are modeled using VHDL. Both models are also parameterizable.

The algorithm used for modular exponentiation in all designs is the modified repeated squaring technique, shown in Figure 3.2. The methods used to compute modular multiplications are, however, different. One modular multiplication approach uses Montgomery's method, as in the expandable hardware, but with merged multiplication and reduction steps. The other design merges multiplication and division through a repeated addition and subtraction process, performing the algorithm shown in Figure 5.10.

In the following sections, estimates of the area, speed and total cost of each of the three designs will be developed and compared.

6.2 Implementation Area

The exact area of any design depends on the minimum feature of the used technology. However, for technology independence, the number of transistors is chosen as an area measure

[50].

Gate Type	Number of Transistors
NOT (Inverter)	2
NAND	4
NOR	4
AND	6
OR	6
XOR	8

Table 6.1: The number of transistors building the basic gates

Basic Components	The Building Logic	Number of Transistors
Half Adder (HA)	XOR + AND	14
Full Adder (FA)	2 XOR + 2 AND + OR	34
D Flip-Flop (DFF)	6 NAND	24
ADDER	HA + $(b-1)$ FA	34 b - 20
ADDER (cin)	b FA	34 b
Subtracter	b NOT + b FA	36 b
1-bit Multiplexer (Bit Mux)	2 AND + OR + NOT	20
b-bit Multiplexer (Bus Mux)	2 b AND + b OR + 3 NOT	18 b +6
1-bit Register (Bit Reg.)	DFF	24
b-bit Register (Bus Reg.)	b DFF	24 b
b-bit Parallel Multiplier	b^2 AND + b HA + $(b^2 - 2\ b)$ FA	40 b^2 - 54 b
b-bit Parallel Load Shift Reg.	b DFF + b Bit Mux	44 b

Table 6.2: The number of transistors building the basic components used

In using the number of transistors as an area measure, a CMOS technology is assumed, where the PMOS and NMOS transistors are used equivalently [50, 51]. In Table 6.1, the number of transistors required for several logic gates is given [55, 56]. Other logic modules are constructed from these basic gates. Some such modules are listed

in Table 6.2. The construction of these logic modules is performed in a standardized manner as found in the literature [48, 52, 53].

All components are parametrized in terms of the number of bits used (b). For example, the 'ADDER' in Table 6.2, is assumed to add two b-bits number with no carry-in; i.e. a HA is used instead of a FA in the first stage. The 'ADDER' with 'cin' is a similar adder but with a carry-in input. The 'subtracter' is considered as an 'ADDER' with one input complemented [52, 53]. The b-bit 'Bus Mux' is a multiplexor that selects a one of two input b-bit buses. The b-bit 'parallel multiplier' is the fast parallel multiplier shown in Figure 4.4. The 'Parallel Load Shift Reg.' is a shift register that can load the data in parallel [51, 57].

Components	The Building Logic	Number of Transistors
The basic cell building the systolic multiplier (basic cell)	4-Parallel Multiplier + 2-Bit Reg. + 8-Bus Reg.+ 3-ADDER + 2-Bus Mux + (2 b + 10)-AND+ (b + 2)-OR + 2 NOT + XOR	160 b^2 + 166 b + 66
The Systolic Multiplier	(l/2 + 1)-basic cell	80 b^2 l + 83 b l + 33 l + 160 b^2 + 166 b + 66
The Montgomery Product (MP)	Systolic Multiplier + 7-Bit Mux + 10-Bus Mux + (9 b l + 14 l -1)-DFF + (5 b + 6 l^2 + 2 l^2 b)-DFF + Parallel Multiplier	80 b^2 l + 299 b l + 369 l + 200 b^2 + 412 b + 242 + 144 l^2 + 48 b l^2
The Complete Implementation	2-MP + 6-Bit Mux + 5-Bus Mux + (l+3)-DFF	486 + 502 b + 393 l + 80 b^2 l + 299 b l + 200 b^2 + 144 l^2 + 48 b l^2

Table 6.3: Area (number of transistors) of the expandable RSA implementation

6.2.1 Area of The RSA Implementations

The number of transistors of the expandable RSA implementation is listed in Table 6.3. In this design, the input message to be encrypted or decrypted is divided into l-words, with each word having b-bits. Accordingly, the number of transistors is given in terms of the number of bits 'b' and the number of words 'l'. Likewise, the number of transistors used in the merged exponentiation hardware is listed in Table 6.4. The number of transistors of the add/subtract exponentiation design is listed in Table 6.5. In this design, the input message is not divided into words, but is completely processed in (b-bits).

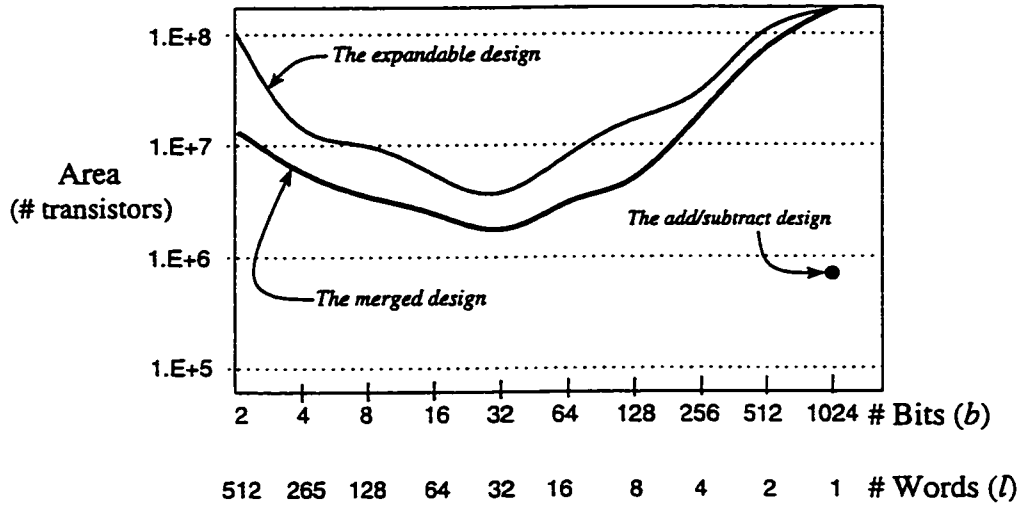


Figure 6.1: The area of the designs for key size of 1024-bits

Figure 6.1 shows an example of area estimation of the three designs for a key size of 1024-bits. It is observed that for the expandable and merged designs, the area is larger than the add/subtract one. Note that the expandable and the merged designs have different areas for the same key size.

It can be observed from the example (Figure 6.1) that the area decreases for both designs until b equals l. After which, the area increases back again. The reason behind this decreasing and increasing in the sizes is that the area of the models are dominated by the sizes of the multipliers and registers. The size of a multiplier depends on b, while it depends on l for the registers. Because of that, the area of the registers dominates

when b is smaller than l, while the area of the multipliers is dominating when l is smaller than b. Therefore, the best area is found when b equals l.

6.3 Speed and Cost

The speed of each implementation is represented by the number of clock cycles required to complete the exponentiation process. The clock period is estimated as the longest path delay. Thus, the overall time is computed with the multiplication of the longest path delay by the number of required clock cycles.

Since area and delay are two conflicting cost measures, we use the $AT^2(Area * Time^2)$ measure [51], as an overall cost measure.

Components	The Building Logic	Number of Transistors
Multiplication loop design	(l+2)-Bit Reg. + Parallel Multiplier + 2-[2-ADDER (cin)] + 2-[2-Bus Mux] +2-Bus Reg.	40 b^2 + 210 b 72 + 24 l
Reduction loop design	2-Parallel Multiplier + 5-Bus Mux 2-[2-ADDER (cin)] + Bit Reg. + l-DFF + 3 Bus Reg. + OR-gate	80 b^2 + 198 b 30 + 24 l
Modular multiplier design (MP*)	Multiplication loop + 4-Bus Mux Reduction loop + (3l + 8) b-DFF (l-1)-Bit Reg. + DFF + 3-OR	120 b^2 + 672 b 192 + 24 l 72 b l
The complete modular exponentiation hardware	2-MP* + 6-Bus Mux + (l^2 + 5l + 8)-Bus Reg. + 2-Bit Mux + (4l +5)-Bit Reg. + 3-OR + AND-gate	240 b^2 + 1644 b 604 + 144 l + 264 l b 24 l^2 b

Table 6.4: Area (number of transistors) of the merged exponentiation hardware

6.3.1 The Expandable Hardware Cost

The time required to complete the modular multiplication process is $2\ l^2 + 4\ l$ cycles.

The time required to complete the exponentiation process is $2\ l^3 + 4\ l^2$ cycles.

The longest path estimated in gates is $13\ b - 4$ gates.

Assume that each gate delay is t ns.

The overall time is $(26\ b\ l^3 + 52\ b\ l^2 - 8\ l^3 - 16\ l^2)\ t$ nsec.

The AT^2 cost of the design $(Area * Time^2)$ is $(486 + 502\ b + 393\ l + 80\ b^2\ l + 299\ b\ l + 200\ b^2 + 144\ l^2 + 48\ b\ l^2) * (26\ b\ l^3 + 52\ b\ l^2 - 8\ l^3 - 16\ l^2)^2 * t^2)$

Components	The Building Logic	Number of Transistors
Reduction Unit	2-Subtracter + 3-Bus Mux ADDER (cin)	$160\ b + 18$
Modular Multiplier	Parallel Load Shift Reg. MSB-first (b-1)-DFF + Reduction Unit + Bus Mux + Bus Reg.	$270\ b$
Modular Exponentiation Implementation	Parallel Load Shift Reg. LSB-first + 2-Modular Multiplier + 4-Bus Mux + 2-Bus Reg. + 3-OR + AND-gate	$704\ b + 48$

Table 6.5: Area (number of transistors) of the add/subtract exponentiation design

6.3.2 The Merged Exponentiation Design Cost

The time required to complete the modular multiplication process is $l^2 + 4\ l + 3$ cycles.

The time needed to complete the modular exponentiation process is $l^3 + 4\ l^2 + 3\ l$ cycles.

The longest path estimated is $31\ b - 13$ gates.

The overall time is $(31\ b\ l^3 + 124\ b\ l^2 + 93\ l\ b - 13\ l^3 - 52\ l^2 - 39\ l)\ t$ nsec.

The AT^2 cost is $(240\ b^2 + 1644\ b + 604 + 144\ l + 264\ l\ b + 24\ l^2\ b) * (31\ b\ l^3 + 124\ b\ l^2 + 93\ l\ b - 13\ l^3 - 52\ l^2 - 39\ l))^2 * t^2$

6.3.3 The Add/Subtract Exponentiation Design Cost

The time required to complete the modular multiplication is $b + 1$ cycles.

The time needed to complete the exponentiation process is $b^2 + b$ cycles.

The longest path delay in gates is $6\ b + 5$ gates.

The overall time is $6b^3 + 11b^2 + 5b\ t$ nsec.

The AT^2 cost is $(704\ b + 48) * (6b^3 + 11b^2 + 5b)^2 * t^2$

6.4 VHDL Modeling

VHDL is an IEEE standard hardware description language [54]. It can describe hardware models at various levels of abstraction such as, behavioral, structural and data flow levels. The structural level can be described as register transfer level or as logic gate level or a combination of both [49].

The three modular exponentiation implementations are VHDL modeled at the structural level. The design entities are modeled to be parameterizable in terms of the number of words (l), and the word size (b). Note that the expandable design and the merged one divide the input data into l-words, each is b-bits. The add/subtract hardware, however, does not. It deals with the data as complete numbers represented in b-bits. All designs have been simulated and verified to be functionally correct.

Figure 6.2 shows an example for the VHDL code. This code models a bn-bit parallel multiplier, where bn is a parameter which determines the size of the multiplier. This model has been used throughout all designs to define multiplication of different sizes, as shown for 4-bits in Figure 4.4.

```
ENTITY bit_multiplier IS
        GENERIC (bn : INTEGER); -- ********** undefind number of bits **********
        PORT (a , b : IN BIT_VECTOR (bn-1 DOWNTO 0);
                pout : OUT BIT_VECTOR (2*bn-1 DOWNTO 0));
END bit_multiplier;
ARCHITECTURE normal OF bit_multiplier IS
        COMPONENT and2 PORT (I1, I2 : IN BIT ; o1 : OUT BIT); END COMPONENT;
        COMPONENT fa PORT (I1, I2, I3 : IN BIT ; sum, carry : OUT BIT); END COMPONENT;
        COMPONENT ha PORT (I1, I2 : IN BIT ; sum, carry : OUT BIT); END COMPONENT;
        FOR ALL : and2 USE ENTITY WORK.and2 (normal);
        FOR ALL : fa USE ENTITY WORK.full_adder (normal);
        FOR ALL : ha USE ENTITY WORK.half_adder (normal);
        SIGNAL outand : BIT_VECTOR (0 to bn*bn-1); -- all the signals coming out of and2
        SIGNAL outsum, outcarry : BIT_VECTOR (0 to bn*(bn-1)-1); -- all adders outputs
BEGIN
and_all : FOR r in 0 TO bn-1 GENERATE
                c1 : FOR c IN 0 TO bn-1 GENERATE
                c2 : and2 PORT MAP (a(c) , b(r) , outand(bn*r + c));END GENERATE;
        END GENERATE;
half_adders : FOR c IN 0 TO bn-2 GENERATE
        c3 : ha PORT MAP (outand(bn+c) , outand(c+1) , outsum(c) , outcarry(c)); END GENERATE;
adders : FOR r IN 1 TO bn-1 GENERATE
        c4 : FOR c IN 0 TO bn-2 GENERATE
                c5 : IF c = 0 AND r = bn-1 GENERATE
                        c6 : ha PORT MAP (outcarry((bn-1)*(bn-1)-bn+1), outsum((bn-1)*(bn-1)-bn+2),
                        outsum((bn-1)**2 ), outcarry((bn-1)**2 ));END GENERATE ;
                c7 : IF c = (bn-2) AND r < bn-1 GENERATE
                        c8 : fa PORT MAP (outcarry((bn-1)*(r-1)+c), outand(bn*(r+1)+c), outand(bn*r+c+1),
                                outsum((bn-1)*r+c), outcarry((bn-1)*r+c));END GENERATE ;
                c9 : IF c = (bn-2) AND r = bn-1 AND c > 0 GENERATE
                        c10 : fa PORT MAP (outcarry((bn-1)*(r-1)+c), outcarry((bn-1)*r+c-1),
                        outand(bn*r+c+1), outsum((bn-1)*r+c), outcarry((bn-1)*r+c));END GENERATE ;
                c11 : IF c < (bn-2) AND r = bn-1 AND c > 0 GENERATE
                        c12 : fa PORT MAP (outcarry((bn-1)*(r-1)+c), outcarry((bn-1)*r+c-1),
                        outsum((bn-1)*(r-1)+c+1), outsum((bn-1)*r+c), outcarry((bn-1)*r+c));END GENERATE ;
                c13 : IF c < (bn-2) AND r < bn-1 AND r > 0 GENERATE
                        c14 : fa PORT MAP (outcarry((bn-1)*(r-1)+c), outand(bn*(r+1)+c),
                        outsum((bn-1)*(r-1)+c+1), outsum((bn-1)*r+c), outcarry((bn-1)*r+c));END GENERATE ;
                END GENERATE;
END GENERATE;
  all_products : FOR r IN 0 TO bn-1 GENERATE
                c15 : IF r = 0 GENERATE
                        c16 : pout(r) <= outand(r) ;END GENERATE ;
                c17 : pout(r+1) <= outsum(r*(bn-1));END GENERATE ;
        c18 : FOR c IN 1 TO bn-2 GENERATE
                        c19 : pout(bn+c) <= outsum(c+(bn-1)*(bn-1));END GENERATE;
        c20 : pout(2*bn-1) <= outcarry((bn-1)*(bn-1)+bn-2);
END normal;
```

Figure 6.2: The VHDL code of the parallel multiplier model

6.5 Analysis

6.5.1 Area and Delay

The three implementations are analyzed assuming a 1024-bit key. The expandable design and the merged one have a freedom in choosing l (the number of words), and b (the number of bits per word), while the add/subtract hardware does not. The add/subtract design is analyzed for a key size of 1024-bits processed as one word, as listed in Table 6.6. The merged hardware analysis is listed in Table 6.7, while the expandable design analysis is listed in Table 6.8.

# bits b (bits)	# digits l words	# clock cycles	longest path delay (t *nsec.*)	overall time (t *msec.*)	Area # transistors	Cost AT^2
1024	1	1049600	6149	6453.99	720944	3 E+13

Table 6.6: Analysis of the add/subtract modular exponentiation design

Comparing the speeds of all designs, it can be observed that the speed of the merged design is fairly close to that of the expandable one, with the expandable speed slightly better. The add/subtract hardware, however, suffers from a very low speed as shown in Figure 6.3.

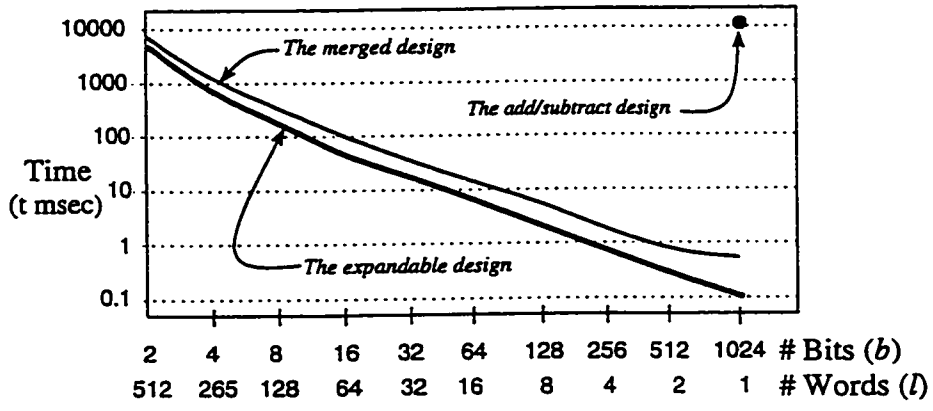


Figure 6.3: The time vs. number of bits analysis

For the same speed, the expandable hardware is larger in area than the merged one (Figure 6.1), which is expected due to the added expandability feature. On the other hand, the add/subtract design has a feasible area, but its overall speed is very low.

# bits b (bits)	# digits l words	# clock cycles	longest path delay (t *nsec.*)	overall time (t *msec.*)	Area # transistors	Cost AT^2
2	512	135,267,840	49	6628.124	12,931,828	6 E+14
4	256	17,040,128	111	1891.45	6,609,676	2 E+13
8	128	2,163,072	235	508.322	3,463,612	9 E+11
16	64	278,720	483	134.622	1,940,764	3 E+10
32	32	36,960	979	36.184	1,360,348	2 E+9
64	16	5,168	1,971	10.1862	1,754,716	2 E+8
128	8	792	3,955	3.1324	4,611,292	4 E+7
256	4	140	7,923	1.10922	16,519,324	1 E+7
512	2	30	15,859	0.47577	64,076,668	6 E+6
1024	1	8	31,731	0.2539	253,637,356	4 E+6

Table 6.7: Analysis of the merged Montgomery modular exponentiation design

6.5.2 The Implementations Cost

The overall AT^2 cost for each design, with key size assumed to be 1024-bits, is listed as the last column in the Tables: 6.6, 6.7, and 6.8.

The cost graph is shown in Figure 6.4, for the purpose of comparison. It can be seen that the cost of the expandable hardware is higher than the merged one, except when the number of bits chosen is very high, such as 256-bits or more. The reason behind this is that the cost function is dominated by the time and the difference in time between the two designs is not constant. As shown in Figure 6.3, the time of the

# bits b (bits)	# digits l words	# clock cycles	longest path delay (t *nsec.*)	overall time (t *msec.*)	Area # transistors	Cost AT^2
2	512	269,484,032	22	5,928.649	63,588,082	2 E+15
4	256	33,816,576	48	1,623.196	22,760,254	6 E+13
8	128	4,259,840	100	425.984	9,679,894	2 E+12
16	64	540,672	204	110.297	5,437,318	7 E+10
32	32	69,632	412	28.688	4,881,862	4 E+9
64	16	9,216	828	7.6308	87,230,454	4 E+8
128	8	1,280	1,660	2.1248	14,539,054	7 E+7
256	4	192	3,324	0.6382	34,714,378	1 E+7
512	2	32	6,652	0.2129	95,035,192	4 E+6
1024	1	6	13,308	0.0799	294,471,679	2 E+6

Table 6.8: Analysis of the expandable modular exponentiation design

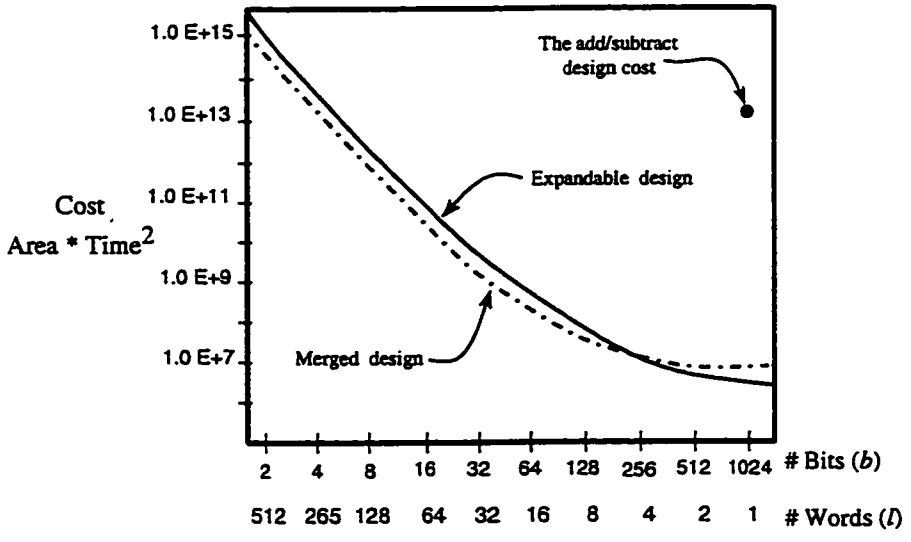


Figure 6.4: The Cost ($Area * Time^2$) analysis for key size of 1024-bits

expandable hardware at large number of bits is much better than it for small number of bits. However, implementing the expandable design with large number of bits is unpractical due to the requirement of a very large area, as shown in Figure 6.1.

Figure 6.5 shows a comparison between the merged design and the expandable one for three different key sizes: 512-bits, 1024-bits and 2048 bits. It can be seen that the cost of the expandable hardware is better only when the number of bits used is very large. The cost of the merged design is mostly lower than the expandable hardware, which is expected due to the overhead required for expandability.

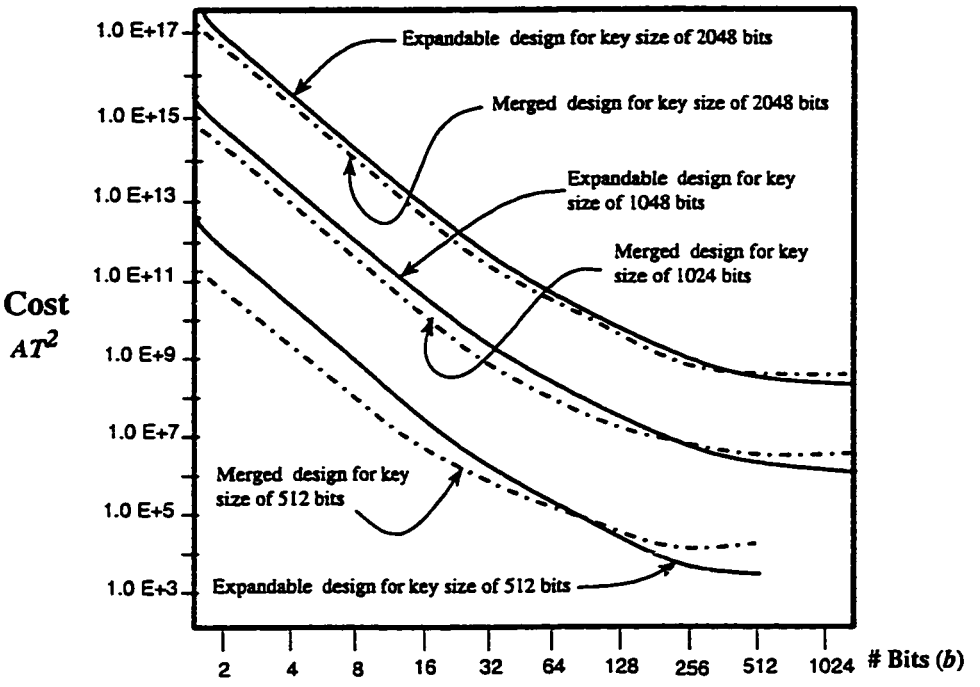


Figure 6.5: The expandable and merged designs costs for different key sizes

6.6 Summary

In this chapter, The expandable hardware is compared with two other designs on the basis of time, area and AT^2 cost. The three designs are then analyzed for 1024-bit numbers. The expandable design had the best speed while its area is the worst. The add/subtract hardware had the smallest area but the time required to complete the

process is the largest, therefore its speed is very low. The size of the merged design is better than the expandable one but not as good as the add/subtract hardware. The merged design had a slightly lower speed than the expandable one. The cost of the merged design is found to be better than the expandable one, except when the number of bits chosen is very high. While the cost of the add/subtract design is found to be the worst.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we have developed a hardware model for an expandable RSA cryptographic system. The thesis begins by an introductory description of cryptographic systems theory covering both secret key cryptosystems as well as public key cryptosystems. Public key systems are covered with some depth particularly the RSA technique which is the most popular public key method.

Several hardware approaches for implementing the RSA method have been surveyed and compared. One implementation was chosen to be modified for expandability [31]. However, a design flaw was discovered in that system. This implementation has been correctly redesigned and then modified for expandability.

The developed expandable design is based on Montgomery's algorithm for modular multiplication. The expandability is achieved using an expandable systolic multiplier.

Two other hardware designs: the merged Montgomery hardware and the add/subtract hardware, have been implemented for comparison purposes. All three designs have been modeled structurally using VHDL. The developed models are designed to allow changing major hardware size parameters. These designs have been simulated and verified to be functionally correct.

Analytical expressions for delay, area and AT^2 cost were derived in terms of l and b,

which are the number of words in the encryption key and the number of bits per word respectively. It has been found that the add/subtract design is the best in terms of area, while its delay and cost are unacceptably large. The other two implementations strike a compromise between time and area at various values of b and l. The cost of both designs decreases with the decrease of delay which is inversely proportional to the number of bits per word, b.

It has also been observed that the cost of the expandable hardware is slightly higher than that of the merged hardware for the same b and l, except when the number of bits chosen is very high. The reason behind this is that, the cost function is dominated by the time, and the difference in time between the two designs is not constant. Implementing the designs that have a very high number of bits requires a very large area, which is not feasible using todays technology.

7.2 Future Work

- Investigate asynchronous design methodology to improve the speed and average cost. This has the potential of improving the overall performance since asynchronous designs follow the average case speed, unlike synchronous systems which follow the worst case speed.

- Study the use of faster adder and serial multiplier architectures. For example, tree adder implementations may have an O(log n) delay, while typical adder architectures have linear delay.

- Investigate other approaches for modular multiplications such as, residue number systems (RNS). However, RNS needs conversion between binary and residue numbers which may be time consuming [34, 41].

- Investigate expandability of the add/subtract hardware. The add/subtract hardware is mainly structured of adders. If the adders are made expandable, the complete add/subtract hardware can be made expandable.

Bibliography

[1] Diffie W. and Hellman, "Privacy and Authentication: An Introduction to Cryptography," *Proceedings of the IEEE*, vol. 67, no. 3, pp. 397-427, March 1979.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of ACM*, vol. 21, no. 2, pp. 120-126, February 1978.

[3] C. E. Shannon, "Communication theory of secrecy systems," *Bell Syst. Tech. J.*, vol. 28, pp. 656-715, Oct. 1949.

[4] H. F. Gaines, *Cryptanalysis: A Study of Ciphers and their Solution.* New York: Dover Publications, 1956.

[5] W. F. Friedman, *Military Cryptanalysis.* Washington DC: US Government Printing Office, 1944.

[6] Lance J. Hoffman, *Building in Big Brother: The Cryptographic Policy Debate*, Springer-Verlag New York, Inc. 1995.

[7] Andrew S. and Tanenbaum, *Computer Networks*, Prentice-Hall International, Inc. Second edition, 1989.

[8] Marshall C. Yovits, *Advances in Computers*, vol. 22, Academic Press, Inc. 1983.

[9] J. van Leeuwen, *Handbook of Theoretical Computer Science*, Elsevier Science Publishers B.V., 1990.

[10] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*, John Wiley & Sons, New York, 2nd edition, 1996.

[11] Marijke De Soete, "Public Key Cryptography," *Computer Security and Industrial Cryptography: State of the Art and Evolution*, Springer-Verlag Inc., Leuven, Belgium, May 21-23, 1991.

[12] W. Diffie, "The First Ten Years of Public-Key Cryptography," *Proceedings of the IEEE*, vol. 76, no. 5, pp. 560-577, May 1988.

[13] D. Coppersmith, "Cryptography," *IBM J. RES. Develop.*, vol. 31, no. 2, March 1987.

[14] Gerd E. Keiser, *Local Area Network*, McGraw-Hill, New York, 1989.

[15] Burt Kaliski, "A Survay of Encryption Standards," *IEEE Micro.*,pp 74-81, December 1993.

[16] Ivan Niven, Herbert S. Zuckerman, and Hugh Montgomery, *An Introduction to the Theory of Numbers*, John Wiley & Sons, New York, 1991.

[17] Lein Harn and Shoubao Yang, "Public key Cryptosystems Based on the Discrete Logarithm Problem," *Advances in Cryptology-AUSCRYPT'92*, Australia, December 1992, pp. 469-476.

[18] D. Kahn, *The Codebreakers, The Story of Secret Writing.*, New York:Macmillan, 1967.

[19] A. Sinkov, *Elementary Cryptanalysis, A Mathematical Approach.* New York: Random House, New Mathematical Library, no. 22, 1968.

[20] Tuckerman, "A study of the Vigenere-Vernam single and multiple loop enciphering systems," *IBM T. J. Waston Research Center*, Yorktown Heights, NY, RC 2879, May 14, 1970.

[21] Ernest F. Brickell, "A Survey of Hardware Implementations of RSA," *Advances in Cryptology-CRYPTO'89 Proceedings*, NewYork, Springer Verlog, 1990, pp. 368-370

[22] P. Bertin, D. Roncin and J. Vuillemin, *Introduction to programmable active memories.* Internal Report, Digital Equipment Coporation, 1989.

[23] Po-Song Chen, Shih-Arn Hwang, and Cheng-Wen Wu, "A Systolic RSA Public Key Cryptosystem," *IEEE International Symposium on Circuits and Systems, ISCAS'96* , NewYork, 1996, pp. 408-411.

[24] Cetin Kaya Koc, Tolga Acar, and Burton S. Kaliski, Jr. "Analyzing and Comparing Montgomery Multiplication Algorithms," *IEEE Micro*, June 1996, pp. 26-33.

[25] M H Er, D J Wong, A. Sethu, and K S Ngeow, "Design and Implementaion of RSA Cryptosystem Using Multiple DSP Chips," *IEEE International Symposium on Circuits and Systems*, NewYork, 1991, pp. 49-52.

[26] Dana Taipale, "Implementing the Rivest, Shamir, Adleman Cryptographic Algorithm on the Motorola 56300 Family of Digital Signal Processors," *Proceedings of the 1996 Southcon Conference*, Orlando, 1996, pp. 10-17.

[27] C. D. Walter, "Systolic Modular Multiplication," *IEEE Transactions on Computers*, vol. 42, no. 3, March 1993, pp. 376-378.

[28] C. D. Walter, "Still Faster Modular Multiplication," *Electronics Letters*, vol. 31, no. 4, February 1995, pp. 263-264.

[29] S. E. Eldridge and C. D. Walter, "Hardware Implementation of Montgomery's Modular Multiplication Algorithm," *IEEE Transactions on Computers*, vol. 42, no. 6, June 1993, pp. 693-699.

[30] B. S. Kaliski Jr. "The Montgomery Inverse and Its Applications," *IEEE Transactions on Computers*, vol. 44, no. 8, pp. 1064-1065, August 1995.

[31] Jorg Sauerbrey, "A Modular Exponentiation Unit Based on Systolic Arrays," *Advances in Cryptology, AUSCRYPT'92*, Gold Coast, Queensland, Australia, December 1992, pp. 505-516.

[32] S. Yen and C. Laih, "The Fast Cascade Exponentiation Algorithm and its Applications on Cryptography," *Advances in Cryptology, AUSCRYPT'92*, Gold Coast, Queensland, Australia, December 1992, pp. 447-456.

[33] K. Lam, K. Sung and L. Hui, "A Cardinalised Binary Representation for Exponentiation," *Computers & Mathematics with Applications*, vol. 30, no. 8, October 1995, pp.33-39.

[34] E. F. Brickell, "A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography," *Advances in Cryptology: Proceeding of CRYPTO'82*, Plenum Press, 1983, pp. 51-60.

[35] C. Wu and Y. Chou, "General Modular Multiplication by Block Multiplication and Table Lookup," *IEEE International Symposium on Circuits and Systems, ISCAS'94* , London, UK: IEEE, 1994, pp. 295-298.

[36] G. Alia and E. Martinelli, "A VLSI Modulo Multiplier," *IEEE Transactions on Computers*, vol. 40, no. 7, July 1991, pp. 873-878.

[37] E. LU, L. Harn, J. Lee and W. Hwang, "A Programmable VLSI Architecture for Computing Multiplication and Polynomial Evaluation Modulo a Positive Integer," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 1, February 1988, pp. 204-207.

[38] B. Arazi, "Double-Precision Modular Multiplication Based on a Single-Precision Modular Multiplier and a Standard CPU," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 5, June 1993, pp. 761-769.

[39] C. D. Walter, "Logarithmic Speed Modular Multiplication," *Electronics Letters*, 18th August 1994, vol.30, no.17, pp.1397-1398.

[40] C. D. Walter, "Space/Time Trade-Offs for Higher Radix Modular Multiplication Using Repeated Addition," *IEEE Transactions on Computers*, Vol. 46, no. 2, February 1997, pp. 139-141.

[41] N. Takagi and S. Yajima, "Modular Multiplication Hardware Algorithms with a Redundant Representation and Their Application to RSA Cryptosystem," *IEEE Transactions on Computers*, vol. 41, no. 7, July 1992, pp. 887-891.

[42] C. N. Zhang, "An Improved Binary Algorithm for RSA," *Computers & Mathematics with Applications*, March 1993, pp. 15-24.

[43] G. Orton, M. Roy, P. Scott, L. Peppard and S. Tavares, "VLSI Implementation of Public-Key Encryption Algorithms," *in Advances in Cryptology: CRYPTO'86 Proceedings*, A. M. Odlyzko, Ed. 1987, pp. 277-301.

[44] Holger Sedlak, "The RSA Cryptography Processor," *Advances in Cryptology: EUROCRYPT'87 Proceedings*, C. Pomerance, Ed. NewYork 1988, pp. 95-105.

[45] F. Hoornaert, M. Decroos, J. Vandewalle and R. Govaerts, "Fast RSA-Hardware: Dream or Reality ?," *Advances in Cryptology : EUROCRYPT'88 Proceedings*, 1988, pp. 257-264.

[46] F. Al-Tuwaijry & S. Barton, "A High Speed RSA Processor," *Sixth International Conference on Digital Processing of Signals in Communications*, September 2-6 1991, Longhborough, UK, IEE 1991, pp. 210-214.

[47] H. Orup, E. Svendsen and E. Andreasen, "VICTOR An Efficient RSA Hardware Implementation," *Advances in Cryptology-EUROCRYPT'90 : Workshop on The Theory and Application of Cryptographic Techniques*, May 21-24 1990, pp. 245-252.

[48] N. R. Scott, *Computer Number Systems & Arithmetic*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

[49] D.R.Coelho, *The VHDL handbook*, Kluwer Academic Publishers, 1989.

[50] Sait S. and Youssef H., *VLSI Physical Design Automation : Theory and Practice*, McGraw Hill, UK, 1995.

[51] Mukherjee A., *Introduction to NMOS and CMOS VLSI Systems Design*, Prentice Hall,1986.

[52] Mano M., *Digital Design*, Prentice Hall, 2nd edition, 1984.

[53] Mano M., *Computer System Architechure*, Prentice Hall, 2nd edition, 1982.

[54] Ashenden P. J., *The VHDL Cookbook*, 1st edition, July 1990.

[55] Sedra A. and Smith K., *Microelectronic Circuits*, Saunders College, 3rd edition, 1991.

[56] Millman J. and Grabel A., Microelectronics, McGraw Hill, 2nd edition, 1987.

[57] Hill F. and Peterson G. to Swiching Theory and Logic Design, John Willy & Sons, 3rd edition, 1987.

Vita

❋ ***Adnan Abdul-Aziz M. S. Gutub***

❋ Received Bachelor of Science in Electrical Engineering from King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia, in January 1995.

❋ Working as a Graduate Assistant in Computer Engineering Department at KFUPM since May 1995.

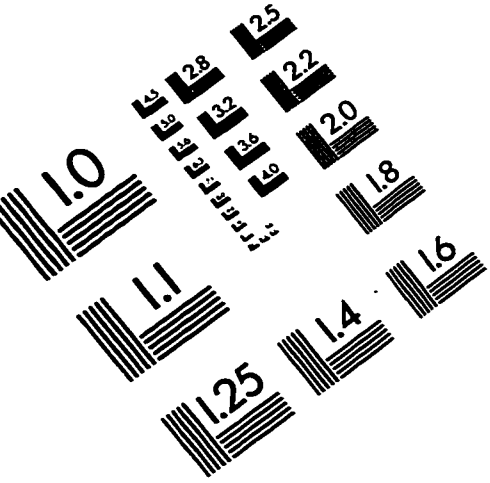
❋ Started Computer Engineering graduate program in January 1996.

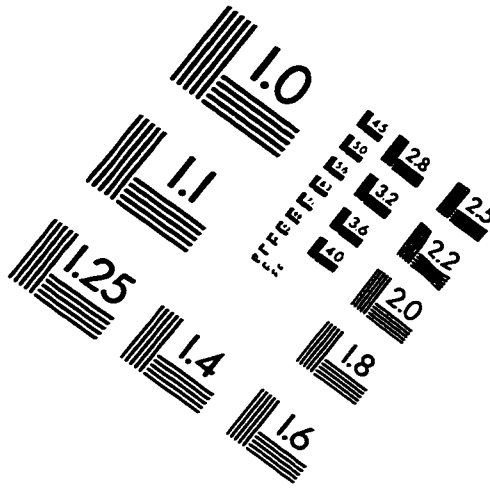
❋ Completed the Master of Science in Computer Engineering from KFUPM in December 1998.

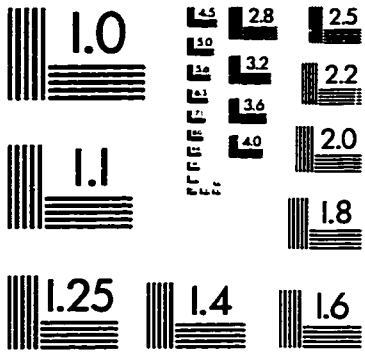
نبذة أكاديمية عن الباحث

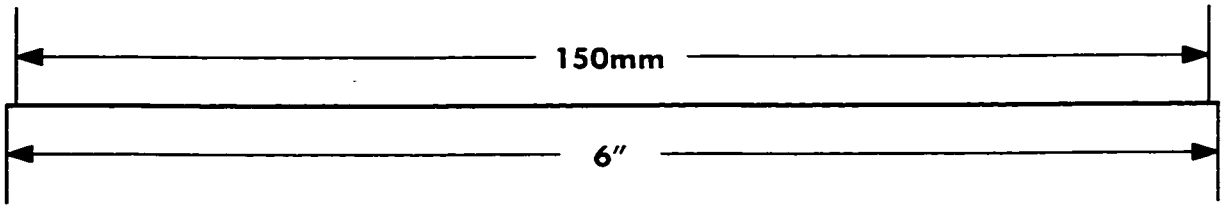
* عدنان بن عبدالعزيز بن محمد صديق قطب
* حصل على درجة بكالوريوس علوم في الهندسة الكهربائية من جامعة الملك فهد للبترول والمعادن، الظهران، المملكة العربية السعودية في شعبان 1415 هـ
* يعمل كمعيد بقسم هندسة الحاسب الآلي في جامعة الملك فهد للبترول والمعادن منذ غرة شهر ذوالحجة 1415هـ
* أكمل متطلبات درجة الماجستير في علوم هندسة الحاسب الآلي في رجب 1419هـ

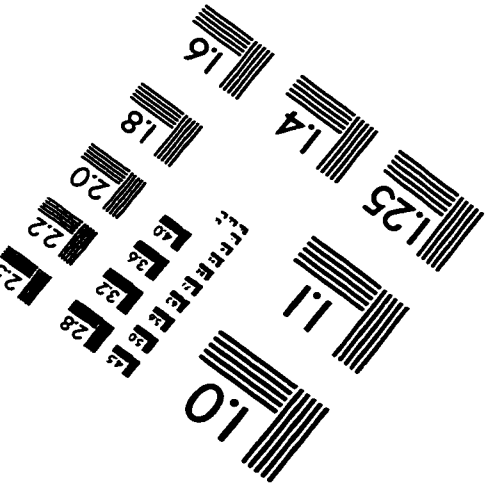
IMAGE EVALUATION
TEST TARGET (QA-3)

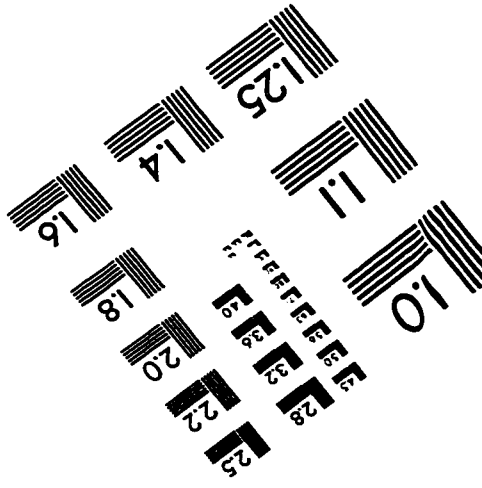












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989