VLSI Architectures for Public Key Cryptology

by

Allan Tomlinson

A thesis submitted to the Faculty of Science,
University of Edinburgh, for the degree of
Doctor of Philosophy

Department of Electrical Engineering

1991

Abstract

This thesis addresses the issue of the efficient implementation of public key cryptosystems. Unlike conventional systems, public key cryptosystems allow secure exchange of information between two parties without prior exchange of secret keys. In addition, many public key cryptosystems may be used to provide digital signatures for authentication of documents. The underlying mathematics of most of these systems however, is more complex than that found in conventional systems, resulting in relatively poor performance of public key cryptosystems in terms of encryption rates.

To improve the bandwidth of the encryption algorithms, processors specifically designed to implement public key cryptosystems are needed. The research presented in this thesis has identified modular multiplication of large integers to be a bottleneck in virtually all public key algorithms and proposes a novel approach to this operation suitable for hardware implementation.

A modular multiplier architecture based on this technique has been proposed and forms the basis of a cascadable modular arithmetic processor capable of dealing with user defined word lengths. The device has been fabricated and results of tests on the finished chip suggest that the RSA encryption algorithm with a 512 bit modulus will achieve a throughput of 30 Kbits/s.

Declaration of Originality

The material presented in this thesis has been researched and composed entirely by myself at the Department of Electrical Engineering at the University of Edinburgh between October 1987 and May 1991except where indicated in the text.

Acknowledgements

Firstly I would like to thank Prof. Peter Denyer for his support and guidance as my supervisor. I would also like to thank my second supervisor, Dr. David Renshaw for his encouragement, and Dr. Stewart Smith for the proof on page 74. Thanks also to Kevin McDermott, Alan Beverage, and Paul Bates from ES2 who helped get the chip through ES2's design release procedures, and to the computing support staff and library staff at the University of Edinburgh.

I am also grateful to my colleagues, who have provided a friendly and intellectually stimulating atmosphere within the department. These include - David Mallon, Gerrard Allan, Subindrao Johal, Colin Carruthers, Martin Ryder, Paul Neil, Iain Findlay, Douglas Grant, Jonathon Puddicome, Hamish Fallside, Douglas Chisholm, Ross Kennedy, Stewart Anderson and Henry Bruce.

Finally I would like to thank my wife Angela for her support and patience throughout this work.

Table of Contents

List of Figures

Chapter 1 Introduction

"Gentlemen do not read each other's mail."

-- USA. Secretary of State H. L. Stimson
on closing the Black Chamber 1929

1.1 The Need for Data Security

It is an unfortunate fact that people will read each other's mail, especially if they stand to profit by doing so. Indeed Stimson himself, as Secretary of War in 1940, came to realize this, actively encouraging such activities through the United States war department's Military Intelligence Division. This is where cryptology is traditionally thought to belong and that, broadly speaking, was the case until the last decade when the falling price of powerful personal computers made them more accessible to the general public. Now, as we enter the era of Information Technology, Wide Area Networks are a reality and vast amounts of sensitive information stored on computer databases are routinely exchanged via public communication links. These networks and the falling cost and increasing performance of personal computers make it possible gain access to information stored on computers anywhere in the world. The ease with which this can be accomplished is demonstrated in a recent book by Clifford Stoll [145], where it is reported how, for more than a year, an intruder rifled through some three dozen computer systems in the USA from his home in West Germany.

Such 'hackers' regularly make the headlines and have been doing so for a number of years. Initially computer crime was regarded by the public as something that could not affect them and was generally dismissed, but with the growing awareness of Information Technology, especially in the financial sector, data security and authenticity is now a real concern. An indication of the extent of the problem can be found in the May 1988 proceedings of the IEEE where G.Simmons lists *sixteen* reasons [138] for cheating in information based systems.With security conscious business managers increasingly looking to cryptology to protect their interests, data security is set to become one of the growth areas of the 90's.

1.2 The Impact of Public Key Cryptography

Data encryption, the process of scrambling a message under the control of a secret key, has historically been used to protect sensitive information in military and diplomatic situations, and more recently for secure financial transactions. Since the security of such systems depends on the secrecy of the key, the growth of communications networks has led to problems of secure key distribution. How do two users agree in advance on a key that will be known only to themselves?. This was one of the driving forces that led to the discovery of public key cryptography by Diffie & Hellman in 1975 [45]. This technique, discussed in detail in chapter 2, uses *separate* keys for encryption and decryption and, with no distribution of secret keys, allows many people to encrypt messages in such a way that only one person can read them: the key distribution problem is solved by the user making his encryption key known to the public. Another benefit of the public key cryptosystem is that the encryption and decryption processes may be mutual inverses. In this case, an encryption followed by a decryption will have the same effect as a decryption followed by an encryption. This means that a user can 'encrypt' a message with his private decryption key so that many users may 'decrypt' it with his public encryption key. In effect, the user has *signed* his message with a signature that cannot be forged, thus providing a means of guaranteeing the authenticity of a message.

This radical twist to an old idea prompted David Kahn to describe public key cryptography as being "the most revolutionary new concept in the field since polyalphabetic substitution emerged in the Renaissance." [14] [p440]

1.3 The Case for Cryptography ASICs

It would appear then that the solution to society's data security and secure communications problems lies with public key cryptography. If this is the case then it is appropriate to question why public key cryptosystems are not in widespread use. Firstly, despite increasing awareness, adoption of computer security practices is still slow due to the cost involved in installing a security system that has previously not been needed. Secondly, secret-key encryption has been endorsed as a Data Encryption Standard [5] by the U.S. government. Finally and perhaps most importantly, the underlying mathematics behind most public key

cryptosystems tends to be more computationally complex than their secret key counterparts resulting in slower encryption rates. Figure 3.2 in Chapter 3 summarises the performance of the most popular public key algorithm published by Rivest Shamir and Adleman, RSA [120], and demonstrates that in software, encryption rates of only around 500 bits/s can be achieved with this algorithm. The Data Encryption Standard on the other hand has recently been reported [143] as achieving encryption rates ranging from 20 Kbits/s on a PC to 100 Kbits/s on a VAX 780.

So. before public key cryptography can compete with secret key cryptography, either faster algorithms will have to be found, or novel hardware architectures will have to be designed to implement existing algorithms. This thesis proposes to take the latter course and to use ASIC technology to investigate new hardware architectures specifically designed for cryptographic applications.

1.4 Aims of This Research

Most of the important public key cryptosystems proposed to date rely on modular arithmetic for their operation. This type of arithmetic is complex, and inevitably involves division by the modulus which, for the 512 bit integers often required for security, leads to poor performance when compared to conventional cryptosystems.

The research presented in this thesis addresses the design of special architectures capable of dealing efficiently with large integer modular arithmetic for public key cryptography. It is hoped that the resulting architectures will improve on the 12 Kbits/s of the fastest RSA processors commercially available [68], and be able to compete favourably with the 20 Kbits/s to 100 Kbits/s of software implementations of the Data Encryption Standard.

By proposing new architectures to improve the performance of public key cryptosystems it is hoped that the benefits offered by public key cryptography will become a more attractive option to the increasing number of security-conscious computer users.

Chapter 2 Introduction to Cryptology

The aim of this chapter is to provide some of the fundamental ideas and theory of cryptology. Further, general, information may be found in [18], [41], [71], [83], [89], or the more mathematical text from van Tilborg [148]. This chapter also serves to define how the work presented in chapters four and five relates to the whole of cryptologic research.

2.1 Cryptology Basics

2.1.1 Terminology

Cryptology is the general term used to describe the study of both *cryptography* and *cryptanalysis*. In the former, the aim is to design codes or ciphers to protect the secrecy and authenticity of information, whereas the latter discipline concerns itself with the breaking of codes to gain access to private information, or to forge coded signals so that they are accepted as genuine. The original message is often referred to as the *plaintext* or *cleartext* and the process of transforming this plaintext into *ciphertext* is known as *encryption*. The reverse process is known as *decryption*. In classical cryptography both the encryption and the decryption transformations depend on the same secret key so that knowledge of the key makes the transformations easy. Without the key the transformations should be virtually impossible. For example, one of the earliest known ciphers, attributed to the Roman emperor Julius Caesar, cyclically shifts each letter in the plaintext by three so that 'caesar' becomes 'fdhvdu'. The key here is the number of places by which the alphabet is shifted. The basic model of a classical cryptosystem, Figure 2.1 shows the distinguishing feature of the secure channel by which the secret key is communicated. This has given rise to the term *secret key cryptosystems*, used to refer to classical systems, as opposed to the more recent *public key systems*.

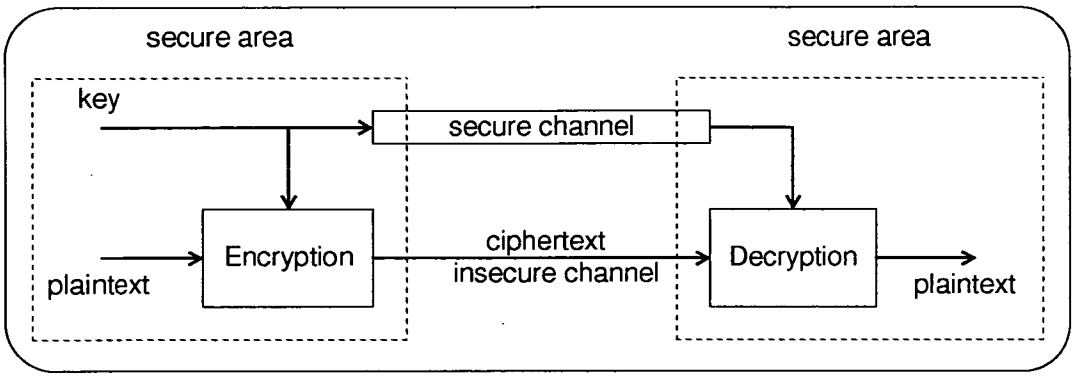

Figure 2.1 Classical Cryptosystem

2.1.2 Methods of Attack

Auguste Kerckchoff (1835 - 1903) suggested that in designing a cryptosystem it should be assumed that the enemy cryptanalyst knows all the details of the encryption and decryption transformations except for the value of the secret key. In other words, the security of the system lies entirely with the secrecy of the key, as is indeed the case for all public domain ciphers such as DES [5], FEAL [133], and RSA [120]. This leaves the classical cryptosystem open to three broad areas of attack.

1. Ciphertext-Only attack — The cryptanalyst only has access to the ciphertext.
2. Known-Plaintext attack — The cryptanalyst knows some plaintext/ciphertext pairs for the current secret key.
3. Chosen-Plaintext attack — The cryptanalyst can obtain the plaintext/ciphertext pairs for his choice of plaintext

Another successful method of attack not suitable for the following analysis but nevertheless one that should not be ignored in practical situations is theft of the secret key.

From the above, and from Kerckchoff's assumption, it is clear that knowledge of some plaintext/ciphertext pairs alone is not enough to break a cipher. Only when the secret key has been deduced is the cipher said to be completely broken. A cipher is *partially* broken if the plaintext can be deduced sufficiently often without knowledge of the secret key.

2.2 Information Theory

2.2.1 Practical Security

With so many methods of attack, it is appropriate to question whether a cipher can ever be totally secure. To answer this question some means of quantifying and measuring the security of a cryptosystem is first required. These are some of the issues addressed by C. E. Shannon in his 1949 paper "Communication Theory of Secrecy Systems" [132].

Shannon examined the classical system of Figure 2.1 subject to ciphertext only attack. Under these conditions, he defined *perfect secrecy* to be the intuitive situation whereby intercepting the ciphertext gives the cryptanalyst no information. Casting the problem in terms of information theory [131], led Shannon to the theorem that a necessary and sufficient condition for perfect secrecy is that all messages and cryptograms are *statistically independent*. This means that the probability of receiving a particular cryptogram **Y** given that the message **X** was sent encrypted by key **Z**, is the same as the probability of receiving the same **Y** from any other message **X'** encrypted under a different key **Z'**.

Figure 2.2 adapted from Shannon [132] illustrates a perfect system with three equally likely messages, and three equally likely keys. The cryptanalyst, on intercepting **Y** has no way of guessing which key was used and therefore, which message was sent.

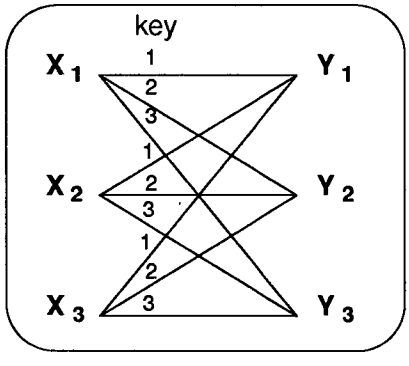

Figure 2.2 Perfect Secrecy System

Perfect secrecy is possible then, if statistical independence is achieved through use of completely random keys which are at least as long as the message they encipher. The only cipher to satisfy these conditions is the Vernam Cipher [150] with the key length greater than

or equal to the message length. This is better known as the *one time pad* from its use during World War II by spies who were issued with a pad of paper containing the randomly chosen secret key and told it could be used for only one encryption.

The key length required for the one time pad makes it impractical for the majority of cryptosystems. That is not to say that all other systems are insecure. Among these theoretically soluble systems there exist wide variations in the amount of effort needed to effect a solution, and in the amount of ciphertext that must be acquired to make this solution unique.

Shannon proposed the idea of two types of security *theoretical security* and *practical security*. The one time pad is a perfect system which is theoretically or unconditionally secure. This means that it is impossible to break under a ciphertext only attack even if the cryptanalyst has unlimited computational resources. Practical, or computational security implies that a system is secure against an attack from a cryptanalyst who has access to finite computational resources.

To quantify the secrecy of a system, Shannon used the key and message *equivocation functions*. These functions measure the conditional entropy of the key, or message, given the received ciphertext, and are applied to the system of Figure 2.1 where

The plaintext	$\mathbf{X} = \{ X_1, X_2, \ldots X_M \}$	is M symbols chosen from L_x possibilities
The ciphertext	$\mathbf{Y} = \{ Y_1, Y_2, \ldots Y_N \}$	is N symbols chosen from L_y possibilities
The key	$\mathbf{Z} = \{ Z_1, Z_2, \ldots Z_K \}$	is K symbols chosen from L_z possibilities

The key equivocation function $H_y(\mathbf{Z})$ is the conditional entropy of the key and measures the uncertainty about the key given that $Y_1, Y_2, \ldots Y_N$ has been received.

$$H_y(\mathbf{Z}) = H(\mathbf{Z}|\, Y_1, Y_2 \ldots Y_N)$$

$$H_y(\mathbf{Z}) = \sum_{y,z} p(y,z) \log\left(\frac{1}{p_y(z)}\right) \qquad \text{(EQ 2.1)}$$

Where $p_y(z)$ is the conditional probability $p(z)$ given *y*.

and $p(y,z)$ is the joint probability $p(y)$ and $p(z)$

Shannon showed this quantity to be related to the entropy of the key, message, and cryptogram, as follows

$$H_y(\mathbf{Z}) = H(\mathbf{X}) + H(\mathbf{Z}) - H(\mathbf{Y}) \qquad \text{(EQ 2.2)}$$

In particular, if H(**X**) = H(**Y**) then the equivocation of the key is equal to the *a priori* uncertainty of the key H(**Z**). This is the case in the perfect system previously described.

For most ciphers the probabilities are too complex to work out to determine the equivocation function exactly. However, Shannon has shown that the function $H_y(\mathbf{Z})$ will have the form described by Figure 2.3.

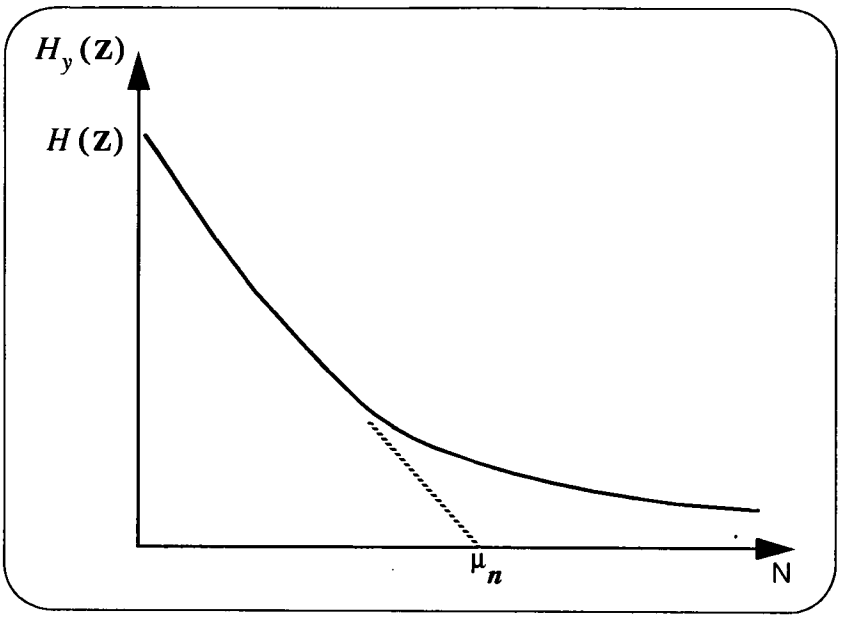

Figure 2.3 Unicity Distance

The function starts off at $H_y(\mathbf{Z}) = H(\mathbf{Z})$ when $N = 0$, decreases linearly with a slope of $-D$ and then follows a decaying exponential with half life *1/D*. The linear region may be extrapolated to the intersection of the N-axis where $N = \frac{H(\mathbf{Z})}{D}$. At this point $H_y(\mathbf{Z}) = 0$, which means only one key can have produced the ciphertext **Y** and the system can, in theory, be broken under a ciphertext only attack given enough computational resources. Exhaustive cryptanalysis, or trying every possible key, at this point will yield a unique solution. Shannon referred to this point as the *unicity distance* μ_n.

From Figure 2.3 it can be seen that

$$\mu_n = \frac{H(\mathbf{Z})}{D} \qquad \text{(EQ 2.3)}$$

Where *D* is the *redundancy per letter of the ciphertext*

If, as is often the case, there is no expansion of the plaintext, then N = M , and $L_x = L_y$, then provided the key is chosen completely at random, the redundancy of the ciphertext will be equal to the redundancy of the plaintext itself and *D* can be approximated by

$$D = R - r \qquad \text{(EQ 2.4)}$$

Where R = ***absolute*** rate of the plaintext = $\log_2(L_x)$

and r = ***actual*** rate of the plaintext

The actual rate of a language for messages of length M characters long is

$$r = \frac{H(\mathbf{X})}{M} \qquad \text{(EQ 2.5)}$$

For English, with

M = 1 (1-grams)	r ≈ 4.15 bits per letter
M = 2 (2-grams)	r ≈ 3.62 bits per letter
M = 3 (3-grams)	r ≈ 3.22 bits per letter

For large M estimates of r for English range from 1.5 to 1.0 bits per letter.

For English then,

$$D = R - r \approx \log 26 - 1.0 = 3.7 \quad \text{bits per letter}$$

Using *D* in (EQ 2.3) gives the unicity distance in bits per letter. It is often more instructive to use the ***percentage redundancy*** per letter of the plaintext as Massey does in [83]. This gives the unicity distance in bits which is more appropriate to modern ciphers that operate on a binary language, and allows comparisons to be made between ciphers whose symbols come from differing alphabets.

The percentage redundancy is:

$$\rho = \frac{D}{R} \qquad \text{(EQ 2.6)}$$

Which is approximately 0.8 for English.

For example, the Caesar cipher on page 4 has 26 possible keys, since

$$H(\mathbf{z}) = \sum_{z} p(z) \log\left(\frac{1}{p(z)}\right) \qquad \text{(EQ 2.7)}$$

and all keys are equally likely so that $p(z) = 1/K$ then

$$H(z) = K\left(\frac{1}{K}\log K\right) = \log K$$

so for the Caesar cipher,

$$\mu_n = \frac{H(Z)}{\rho} = \frac{\log 26}{0.8} \qquad \text{(EQ 2.8)}$$

Which is approximately 6 bits, or just over one character.

For a ***random substitution*** there are 26! possible keys, and equation (EQ 2.8) shows the unicity distance is increased to approximately 110 bits, or 22 characters.

The DES cipher discussed later, has a 56 bit key, and unicity distance of 70 bits or, if 5 binary digits are used to code the 26 letters of the alphabet's 14 characters. If 7 bits are used to code each character, allowing one parity bit, then DES will have a unicity distance of 10 characters.

An RSA cipher with 512 bit key will have unicity distance of 640 bits.

Although, strictly speaking this analysis applies only to Shannon's "random ciphers", experienced cryptographers believe the formulae to be valid in virtually all secrecy systems, with the exception of those probabilistic systems described in section 2.2.3. Shannon's model is routinely used to measure the unicity distance of many ordinary ciphers.

2.2.2 Work Characteristic

The unicity distance measures the secrecy of a system in that it determines how much ciphertext has to be intercepted before a unique key has to have been used. It also gives the cryptographer an indication of how often the key should be changed. Having intercepted enough ciphertext however, does not mean that the cipher is broken, there still remains some work to be done, and this can vary widely from one cipher to another and from day to day.

Shannon defined the concept of the work characteristic of a system W(N), an indication of the average amount of work measured say in computing time on a CRAY, to determine the key for a cryptogram of N letters. The function W(N) is a measure of the amount of "practical secrecy" afforded by the system and Shannon postulated the behaviour of this function to be essentially

as shown in Figure 2.4 for any type of system where the equivocation function approaches zero.

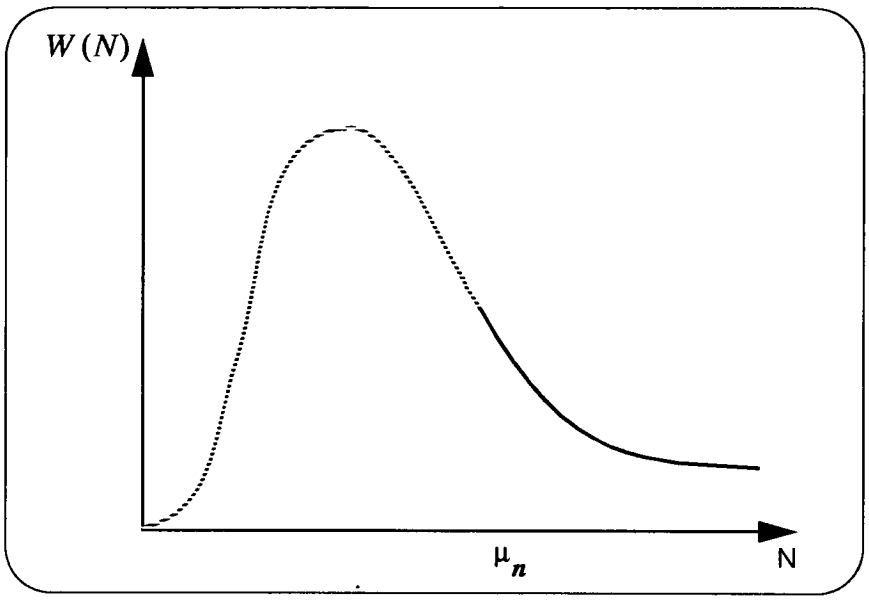

Figure 2.4 Work Characteristic

In the dotted region there are numerous possible solutions and all must be determined. At the unicity distance only one solution exists but a great deal of work is needed to isolate it thereafter, as more material is acquired, the work reduces to some asymptotic value where additional data does not help

The difficulty in solving a particular cipher may change gradually as faster processors become available, or overnight if a new algorithm is discovered. Thus, in practice, there are two work functions. The historical work function, W_h based on ***known*** cryptanalytic techniques, and the intrinsic work function W_i which will form a lower bound on W_h. It is generally the former that is referred to when people talk of a cipher taking millions of years to break. There are no practical ciphers today whose intrinsic work function is known.

However, there are many ways of maximising W(N). It is the aim of good cipher design to maximize W(N) so that the time needed to break the cipher makes it impractical, or at least so long that the information obtained ***no longer has any value.*** This emphasizes the point that in

cryptology it is more often the *value* of the information we are concerned with than the information itself.

2.2.3 Improving Secrecy

The obvious way to improve secrecy is to increase the unicity distance μ_n.

$$\mu_n = \frac{H(\mathbf{Z})}{\rho} \qquad \text{(EQ 2.9)}$$

To do this we can increase H(**Z**) by using more keys and/or making sure all keys are equally likely (this is the mathematics behind choosing a non-obvious password!). Alternatively we can look at ways of decreasing ρ. Two well known techniques for doing this are data compression, and non-deterministic, or probabilistic encipherment.

In the system of Figure 2.1 with no plaintext expansion, ρ is equal to the redundancy of the message. If **X** is ideally compressed so there is no redundancy, then all possible messages are equally likely. Thus deciphering **Y** with *any* key yields a possible solution. In other words, as **X** approaches an ideally compressed source $\mu_n \rightarrow \infty$. For example, if a message is someone's telephone number composed of a group of 5 decimal digits, then deciphering to 93864 is just as valid as 84028. In this case even the Caesar cipher will provide perfect secrecy under a ciphertext only attack.

Data compression is in general a useful cryptographic tool and should be applied if at all possible. This was well known to cryptographers in the pre-computer age when, according to Massey [83], many letters and blanks were deleted from messages before encrypting. THSISAFRMOFDTACMPRESON.

Another approach used by old-time cryptographers was to insert extra symbols in the message to hide the statistics of the message. THXISAXNEXAMPXLE. This second trick is an example of probabilistic cryptography usually referred to plaintext padding. The distinguishing feature of probabilistic ciphers is that the key and the plaintext do not uniquely determine the ciphertext. Some randomising function, known *only at the encryption site*, is applied to the message before encryption as illustrated in Figure 2.5.

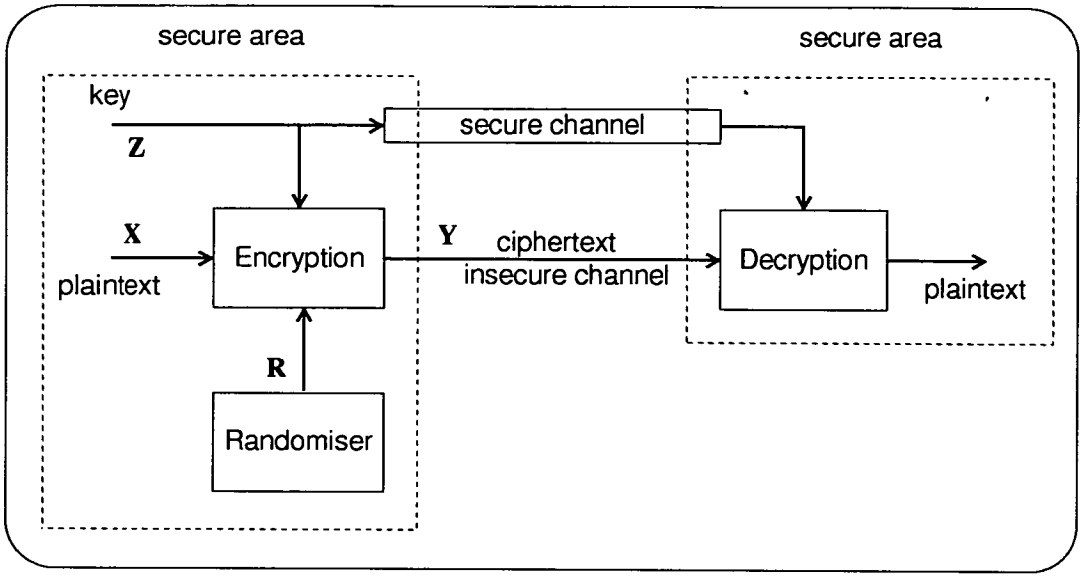

Figure 2.5 Probabilistic Secret Key System

At first sight plaintext padding would appear to be adding to the redundancy, but the extra symbols can be selected from a large set in a very random fashion so that the redundancy of the *ciphertext* is in fact reduced. Figure 2.6 shows J random symbols $\mathbf{R} = \{ R_1, R_2, \ldots R_J \}$ added to the M message symbols **X**.

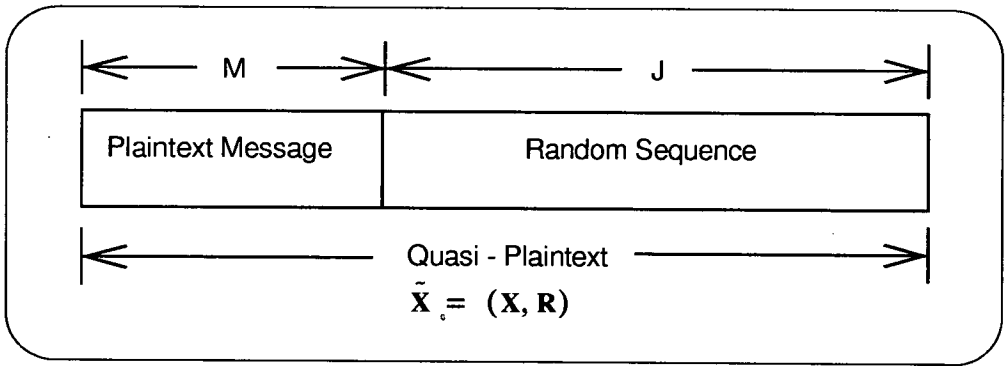

Figure 2.6 Plaintext Padding

If these random characters are taken from the same language as **X** so that $L_R = L_X$, then

$$H(\tilde{\mathbf{X}}) = H(\mathbf{X}) + H(\mathbf{R}) \qquad \text{(EQ 2.10)}$$

but since **R** is J symbols taken from L_X then there can be L_X^J possible **R**'s, thus

$$H(\tilde{\mathbf{X}}) = H(\mathbf{X}) + J\log L_X$$

So, using (EQ 2.4) the redundancy of the message is

$$\tilde{D} = \log L_X - \frac{H(\mathbf{X}) + H(\mathbf{R})}{M+J} \qquad \text{(EQ 2.11)}$$

$$\tilde{D} = \log L_X - \frac{H(\mathbf{X}) + J\log L_X}{M+J}$$

$$\tilde{D} = \frac{M\log L_X - H(\mathbf{X})}{M+J}$$

$$\tilde{D} = \frac{M}{M+J}\left[\log L_X - \frac{H(\mathbf{X})}{M}\right]$$

$$\tilde{D} = \frac{M}{M+J}D$$

$$\tilde{\mu}_n = \frac{M+J}{M}\mu_n \qquad \text{(EQ 2.12)}$$

So if for example the DES cipher is used in single bit cipher feedback mode, as described in the Guidelines For Implementing and Using the NBS Data Encryption Standard[6], where one bit of plaintext is enciphered with 63 random bits, then the unicity distance will be increased by a factor of 64. From the calculation on page 11 this means we have an improvement from 70 to 4480 bits.

A second non-deterministic technique that can be used to reduce the redundancy of the ciphertext is that of *homophonic substitution.*

The cryptography systems mentioned so far all have a one to one correspondence between plaintext letters and ciphertext letters. Homophonic ciphers do not have such a one to one mapping. Each plaintext symbol maps on to a ciphertext symbol chosen at random from a set

of homophones. The implication is that the size of the ciphertext alphabet L_Y is greater than that of the plaintext alphabet L_X. The situation is illustrated in the following, where the plaintext comes from the English alphabet, and the ciphertext from the set of ASCII symbols.

plaintext	ciphertext
A	F $ %
B	6 +
C	X & 0
D	? P R
E	@ ! K T | # ~

Figure 2.7 Homophonic Substitution

From this short example the ciphertext "F&@" and "$&#" both decipher to the word "ACE".

Homophonic substitution can be extremely effective if the statistics of the message are known beforehand. The homophones can then be chosen so that the more commonly occurring symbols, such as the letter E in English, map on to a larger set of homophones than the less frequent ones, thus producing ciphertext that approaches statistical independence from the plaintext.

For example, if a binary message source produces '0' with a probability of 0.25 and '1' with a probability of 0.75, then the following transformation will provide a perfect homophonic substitution.

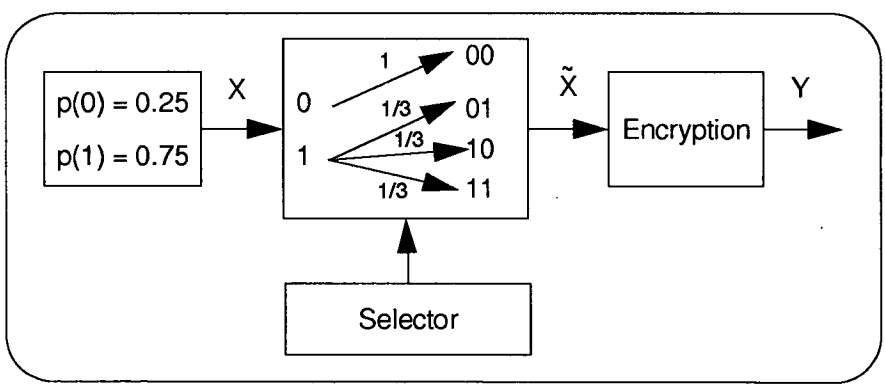

Figure 2.8 Homophonic Encryption System

In the above the selector randomly chooses the mapping for '1', thus the output sequence $\tilde{\mathbf{x}}$ will be a sequence of random bits. The redundancy of the ciphertext will be zero and the unicity distance $\mu_n \rightarrow \infty$.

These probabilistic techniques help to ensure the same ciphertext is never produced twice from the same message, thus improving security against a chosen plaintext attack. The price paid is, of course, the expansion of the plaintext although this is likely to be acceptable where security is important. The benefits afforded by probabilistic encryption have prompted Massey to suggest in [82] that research in this area offers the best chance of leading to practical provable computationally-secure ciphers with small keys.

The importance of all three of the above is that they can be applied to *any* encryption system and will greatly enhance the secrecy.

To conclude this section on improving security, Shannon's two principles of *diffusion* and *confusion* are be discussed. Unlike the previous methods, these techniques apply to cipher design and cannot be used to "pre-process" the message.

In many non-ideal ciphers the statistics of the plaintext are reflected in the ciphertext. By analysing the frequency distributions of letters in the cryptogram, the cryptanalyst can often break these ciphers. Shannon suggested two methods to frustrate such statistical analysis.

In the method of *diffusion* the aim is to dissipate the statistical structure of the message into the long range statistics of the ciphertext, the effect of this is to break up the digrams and trigrams of the plaintext. To achieve this in practice means aiming to ensure that each symbol in the message and the key affects as much of the ciphertext as possible. The "chaining" modes of DES described in [6] are a good example of this technique.

The method of *confusion* recognises that some statistics "leak" through the encrypting transformation, but aims to make the relationship too complicated to be of any use to the cryptanalyst.

One of the strongest theoretical arguments against the use of additive stream ciphers, described in section 2.4.1, is that they can never achieve good diffusion of the key. Each key symbol can only influence one ciphertext symbol.

Product ciphers provide a way to design good confusion and diffusion into a cipher without making the algorithms themselves too complex. These ciphers are composed of many component ciphers, each one contributing a small amount to either the diffusion or confusion of the complete cipher. As explained in section 2.3.2, DES is a classic example of how strong ciphers may be built up by this method.

2.2.4 Authenticity

The discussion so far has concerned itself mainly with cryptography as a means of providing secrecy. Another increasingly important use of cryptography is to provide authenticity. When the recipient of a cryptogram deciphers the text to form a message that makes sense, he may still be uncertain that the message was sent by a valid party. For example, unless precautions are taken, anyone could interrupt messages between the two users and replay them, or even replace them with new ciphertext.

It is only a recent realization that although a system may be highly secure, that does not have any bearing on its authenticity. In other words, secrecy and authenticity are *independent* attributes of a security system. The theory of authenticity owes much to the work of Gus Simmons of the Sandia National Laboratories in the U.S.A. Simmons was interested in authenticity with respect to nuclear test-ban monitoring by remote seismic observatories [136]. The idea was that the U.S.A. and the Soviet Union put seismometers on each other's territory to ensure the limits imposed on nuclear testing were observed. The difficulty lay in convincing the monitoring nation that the host nation was not tampering with the data transmitted from the observatory. Conventional cryptography was not applicable because the host nation had to

know what information the monitoring nation was transmitting back home. What was needed was authenticity without secrecy.

In addressing this problem Simmons suggested that Shannon's model of a cryptosystem, Figure 2.1, be modified to allow the cryptanalyst more freedom. In Simmons' model, Figure 2.9 the cryptanalyst is not only eavesdropping, he is now actively tampering with the transmitted messages.

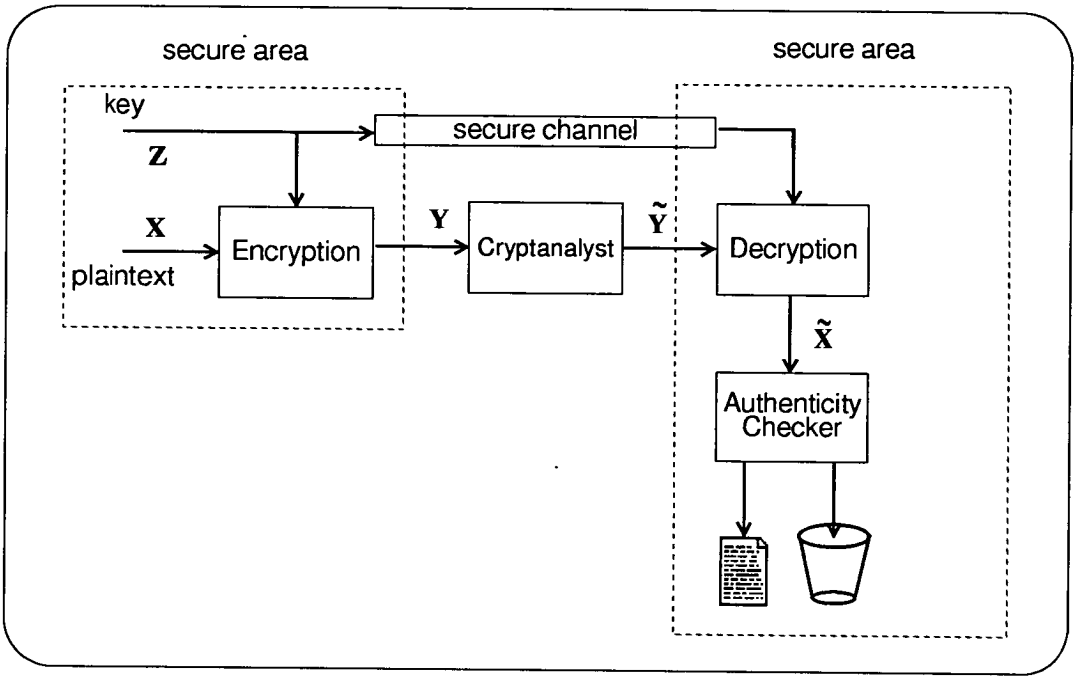

Figure 2.9 Simmons' Cryptosystem

Simmons defined two kinds of authenticity attack. An ***impersonation attack*** where the cryptanalyst sends a fraudulent cryptogram without waiting to see the genuine cryptogram; and a ***substitution attack*** where the cryptanalyst waits for a genuine cryptogram to be sent, examines it, and then forwards a fraudulent cryptogram. Success under an impersonation attack means that the fraudulent cryptogram is accepted as valid under key **Z**. Success under a substitution attack has the additional constraint that the message decrypted be different from the message sent.

Let P_I be the probability of success under an impersonation attack, and P_S the probability of success under a substitution attack. If ...

the total number of possible cryptograms is N_Y,

the total number of possible messages is N_X,

and the total number of possible keys is N_Z,

then for each key there must be *at least* N_X possible cryptograms.

So, if the cryptanalyst selects one cryptogram at random from N_Y, then the probability of success under an impersonation attack will be

$$P_I \geq \frac{N_X}{N_Y} \qquad \text{(EQ 2.13)}$$

The implications of this are:

1. Complete protection, $P_I = 0$, is impossible.
2. For good security N_Y should be much greater than N_X.
3. Equality exists only when there are exactly N_X valid cryptograms for each key. Thus probabilistic encryption adversely affects security against impersonation attacks.

In the example on page 13, $N_X = N_Y$ and $P_I = 1$, demonstrating how perfect secrecy may be achieved with *no authenticity.*

In [137], Simmons shows how

$$\log P_I \geq -I(Y;Z) \qquad \text{(EQ 2.14)}$$

Where I(Y;Z) is the information Z gives about Y

$$I(Y;Z) = H(Y) - H(Y|Z) \qquad \text{(EQ 2.15)}$$

It can be shown that this information is always mutual information,

$$I(Y;Z) = I(Z;Y) \qquad \text{(EQ 2.16)}$$

Simmons defined the *probability of deception* as

$$P_d = max(P_I, P_S) \qquad \text{(EQ 2.17)}$$

and showed this too, to be bounded by

$$\log P_d \geq -I(Y;Z) \qquad \text{(EQ 2.18)}$$

Simmons then defined *perfect authenticity* to be equality in (EQ 2.18).

The conclusion from the above is that if the probability of deception is to be minimised, then *the cryptogram has to provide a lot of information about the key.* In other words part of the secret key has to be dedicated to providing authenticity, rather than secrecy. The following example serves to illustrate the point.

The key is defined to be an even number of symbols

$Z = \{ Z_1, Z_2, \ldots Z_K \}$

and the message X, is one bit, either a '1' or a '0'. The key is used only once for each message. To transmit another message (another *bit* in this example) a new key is used.

If the following encryption transformation is used:

If x = 0 then $y = \{ 0, Z_1, Z_2, \ldots Z_{K/2} \}$

If x = 1 then $y = \{ 1, Z_{K/2+1}, \ldots Z_K \}$

then *perfect authenticity* is achieved with *no secrecy.* This is what Simmons required to solve the problem mentioned in the introduction.

The preceding examples have illustrated that secrecy and authenticity are separate attributes of a cryptographic system and that it should never be assumed that possession of one automatically provides the other. That is not to say a system can not have both. If the above example were modified slightly so that instead of transmitting X, $X + Z_{K+1}$ was sent, where Z_{K+1} is an additional key symbol, then perfect secrecy and perfect authenticity is achieved. Transmitting $X + Z_{K+1}$ is equivalent to the one time pad.

2.3 Block Ciphers and the Data Encryption Standard

This section uses the Data Encryption Standard to illustrate the design principles behind block cipher construction. These principles have their roots in Shannon's work [132], and formed the basis of IBM's research into nonlinear block ciphers [53] which produced the LUCIFER cipher [52], and ultimately resulted in the Data Encryption Standard or DES [5].

Although DES has several different modes of operation, when used in "Electronic Code Book" mode, or ECB, it is an excellent example of a *block cipher*. The advanced modes of DES operation such as Cipher Block Feedback illustrate how block ciphers may be adapted to take on certain desirable properties of *stream ciphers*. These modifications to the basic cipher are discussed in more detail in the section on stream ciphers, section 2.4.

The distinction between block and stream ciphers lies in the transformation that is applied to successive plaintext blocks, and to a lesser extent the length of the plaintext blocks. With a block cipher successive blocks always encounter the same transforming function, and the transformation is usually over a large blocklength. Stream ciphers on the other hand have some internal memory and, in general, transform successive blocks with a different function, the transformation being governed by the internal state of the system. Thus, if the same message is encrypted twice, a block cipher would produce two identical blocks of ciphertext whereas the stream cipher would produce two different cryptograms.

2.3.1 The DES Controversy

The Data Encryption Standard is the most widely used cipher. It is also the most controversial.

Before the introduction of DES, cryptography algorithms could be classed as belonging to one of the following three categories:

1. Outdated ciphers, up to about Word War II
2. Commercial ciphers with proprietary algorithms known only to the vendors
3. Classified government ciphers.

This meant that apart from the government and commercial organisations who designed the ciphers, users could not have any confidence that the algorithms available offered enough security. The United States National Bureau of Standards (NBS) therefore undertook to

develop a high quality cipher for public use. They invited the public to submit algorithms for consideration as the new cipher and asked the National Security Agency (NSA) to evaluate the responses or provide an algorithm if none were received. The algorithm chosen was a modification of a cipher that used a 128 bit key, IBM's LUCIFER [52]. The company's original submission used a 768 bit key, but this was to be reduced to the 56 bits used at present. It is reported in [83] that the NSA were "instrumental in reducing the DES secret key to 56 bits". The reduction of the key was immediately met with scepticism and prompted Diffie and Hellman [46] to publish the conceptual design of a machine capable of trying every possible key which they estimated would break DES in about 12 hours. Hellman later proposed a modification [66] to this design which he estimated could break 100 cryptograms in parallel each day. These estimates were regarded by many to extremely optimistic, but the controversy did raise questions about how secure the cipher should be to be considered practically secure, or as Smid and Branstad [143] put it "how good is good enough". In their discussion of the DES key length Smid and Branstad make the point that the key had to be small enough to keep costs down and to maintain user friendliness.

The second criticism of the cipher lay with the design of the S-boxes described in section 2.3.3. The S-boxes were designed by the NSA who refused to publish the principles on which they based their decisions. It was argued that this was because the S-boxes concealed a "trap-door" which would make it easy for the NSA to break them.

To answer their critics, the NBS held two workshops, one to discuss the mathematics of the algorithm [7], the other to discuss the key length [8]. No "trap doors" were identified and the key was considered to be adequate for the users needs for the next 10 to 15 years. The standard was therefore accepted and published in January 1977 with the recommendation that it be reviewed every five years. The last review was in 1988.

The latest development in the analysis of the DES cipher was presented at the CRYPTO '90 conference by Eli Biham and Adi Shamir [23] who described a chosen plaintext attack on DES. The cryptanalysis algorithm described by Biham and Shamir is capable of breaking the DES cipher in less time than an exhaustive search of the key space provided the number of iterations of the encryption algorithm is 15 or less. The DES cipher uses 16 iterations. Attempts

by the authors to strengthen the cipher by changing the key schedule or the S-box design did not work. These results highlight the importance of good S-box design and suggest that the NSA were probably well ahead of the rest of the cryptographic community when DES was designed.

2.3.2 Cipher Elements

The DES cipher is a *product* cipher consisting of sixteen "rounds", or iterations, of successive transformations. The transformation carried out at each round is constructed from a *substitution* and a *transposition* cipher. In a substitution cipher, each symbol in the plaintext alphabet is mapped on to a fixed substitute in the ciphertext alphabet. Homophonic substitution is an exception. A weakness of substitution ciphers, such as the Caesar cipher, is that the relative frequencies of letters and groups of letters leak through the transformation and unless precautions are taken these ciphers may be broken by frequency analysis of the ciphertext. If the methods of section 2.2.3 are not appropriate, then the key should be changed often enough to ensure no plaintext symbol occurs more than once in the key's lifetime. Another approach would be to use a large alphabet.

Substitutions were the earliest ciphers to be used. Next were transposition ciphers. The key in a transposition cipher is a fixed permutation of the plaintext block. Although the frequency of single symbols still leak through, digrams, trigrams, etc. are broken up thus altering the statistics of the ciphertext.

Although substitution and transposition are weak ciphers when used alone, combining them, and repeatedly applying the transformations as a *product* cipher, as is done in the DES cipher, can result in extremely strong algorithms. The distinguishing feature of product ciphers is that a *single key*, or some part or permutation of it, is used to control each individual transformation as illustrated in Figure 2.10.

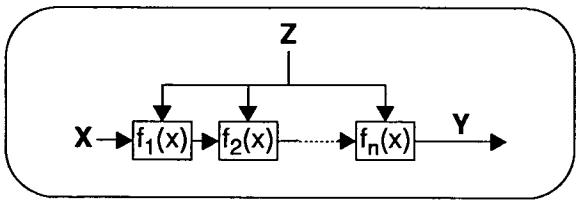

Figure 2.10 Product Cipher

This is the difference between a product cipher and a ***cascade*** cipher where each transformation is controlled by a separate key as in Figure 2.11.

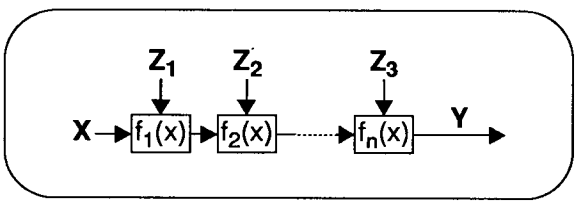

Figure 2.11 Cascade Cipher

The functions used in each round of DES are *involutions.* An involution is a function that is its own inverse such as f(x) = -x, or a transposition that swaps two halves of a block. If an involution is used to encrypt plaintext then the ***same function*** can be used for decryption. So in DES, if the algorithm is run backwards then each transform undoes the previous one as shown in Figure 2.12, thus both encryption and decryption use the same algorithm and key. The only difference is that the sub keys used in each round are applied in the reverse order.

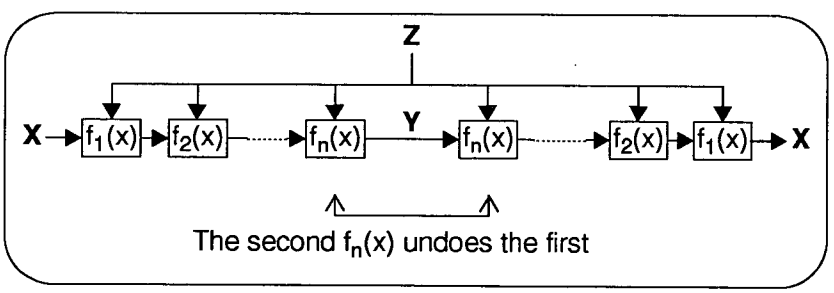

Figure 2.12 Effect of Involutions

2.3.3 DES Structure

The DES algorithm operates on 64 bit blocks of plaintext and produces 64 bit blocks of ciphertext using a 56 bit key. The complete definition may be found in [5] where the following figures have been adapted from, and a software version of the cipher written in C may be found in [125].

Each iteration of the DES algorithm depends on one of 16 intermediate keys derived from the input key using the key scheduling algorithm of Figure 2.13. The Permuted Choice 1 in the key scheduling algorithm performs a permutation on the 64 bits of the input key and discards bits 8, 16, 24, etc. to create a 56 bit active key. The active key is divided into two halves C_i and D_i and each half is then cyclically shifted left either once or twice each iteration. After shifting, the two halves are re-combined and undergo another permutation where eight more bits are discarded resulting in the 48 bit intermediate key Z_i.

The encryption algorithm shown in Figure 2.14 transposes the 64 input bits under the Initial Permutation, then splits the data into two 32 bit words which are transformed each round under the control of key Z_i.

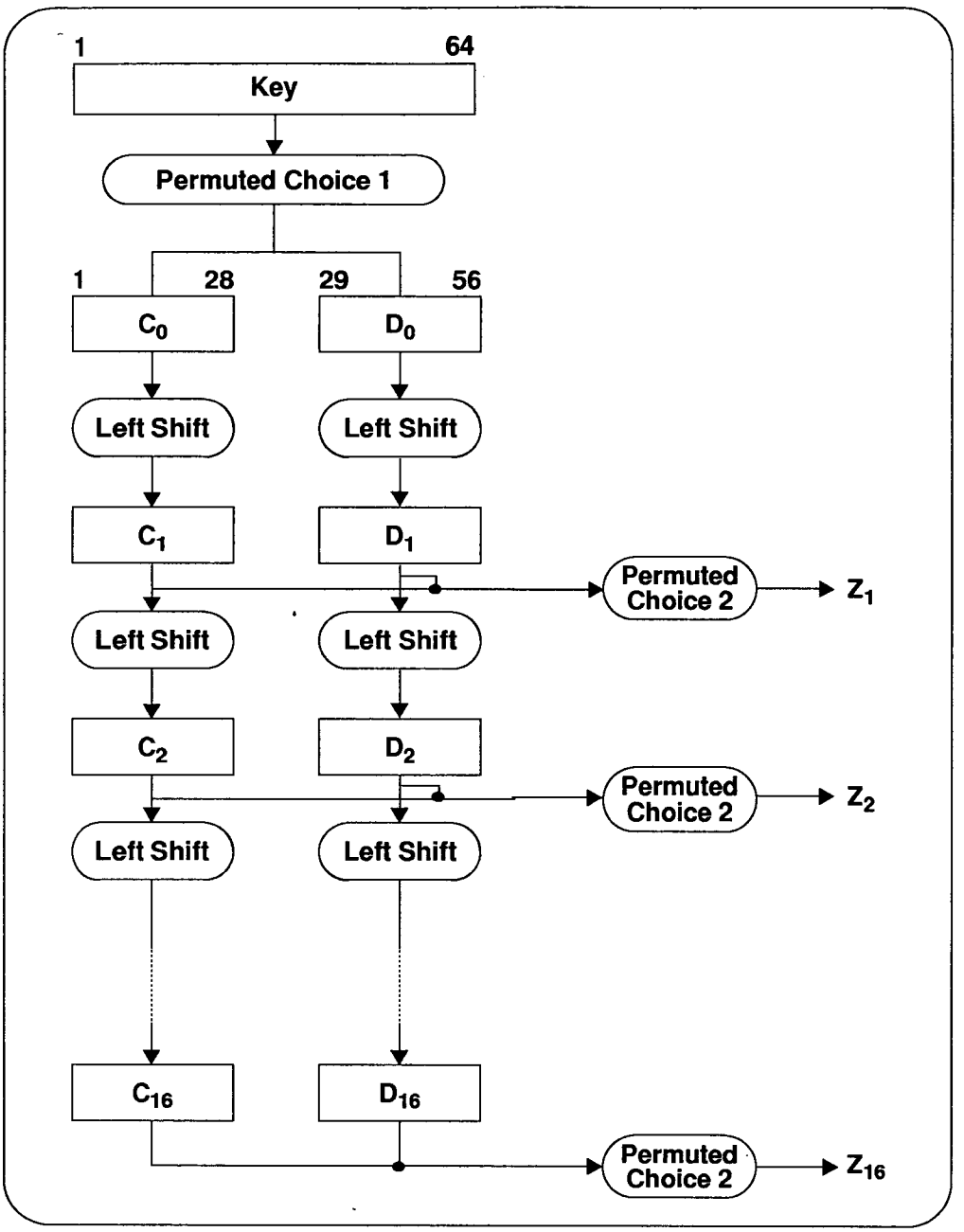

Figure 2.13 Des Key Scheduling Algorithm

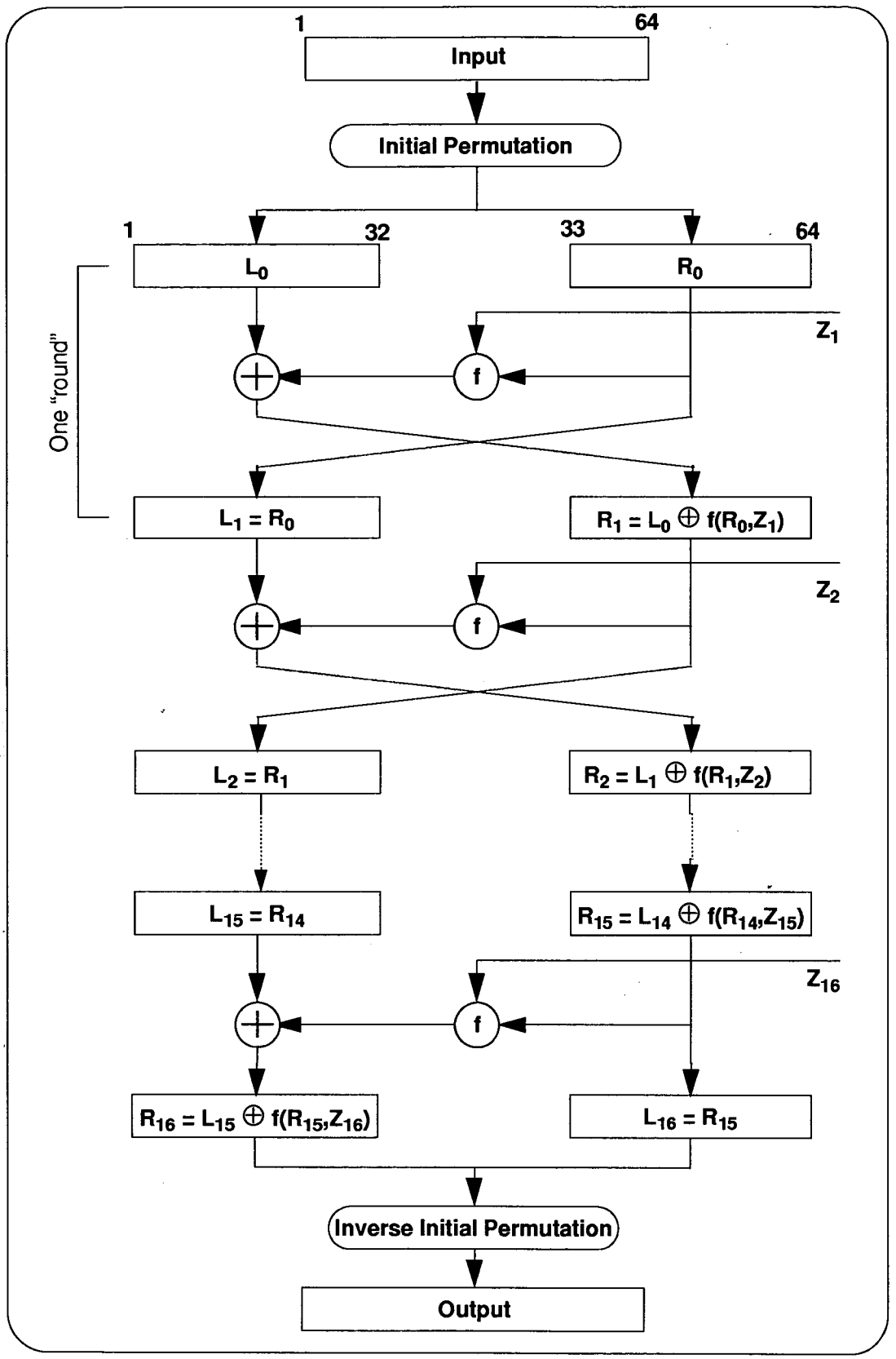

Figure 2.14 DES Encryption Algorithm

Examination of Figure 2.14 reveals that each round is composed of two ciphers as illustrated in Figure 2.15. A substitution applied to the left half of the word followed by a transposition of the left and right halves. The transposition does not depend on the key, but is included to provide Shannon's diffusion. The substitution provides the confusion.

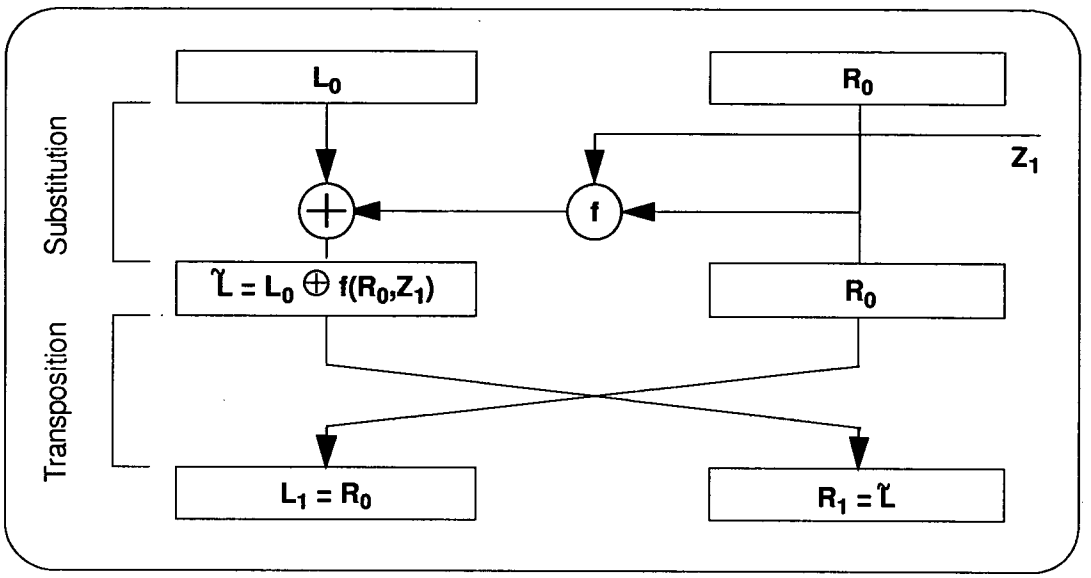

Figure 2.15 Composition of DES Round

It is clear from Figure 2.15 that since the transposition swaps left and right halves, it is an involution. The substitution is the exclusive-or of the left half of the word with some function of the right half of the word and the key. If this substitution is applied twice in succession to a block of data, using the same key, the nature of the exclusive-or function ensures that this too is an involution. Thus in the DES cipher the decryption algorithm is identical to the encryption algorithm except that the intermediate keys have to be applied in the reverse order. The "Initial Permutation" has no cryptographic significance - it "undoes" itself at the end of the algorithm. The reason for the Initial Permutation is not published in the standard but it may be that its eight-bit orientation eases hardware implementations as illustrated by Verbauwhede et al. in [149] who use an array of eight-bit shift registers to perform the permutation.

Figure 2.16 shows the DES Cipher Function represented by "f( )" in the previous diagrams. In this function the right half of the data word is expanded from 32 to 48 bits by duplicating some

of the bits during the permutation process. The intermediate key on the other hand is reduced in the key scheduling algorithm, from 56 to 48 bits. In IBM's original submission the user could *choose* these 48 bits in all 16 rounds giving the 768 bit key noted on page 23. The results of these expansions and contractions are exor'd before entering the eight S-boxes which reduce the data back down from 48 to 32 bits.

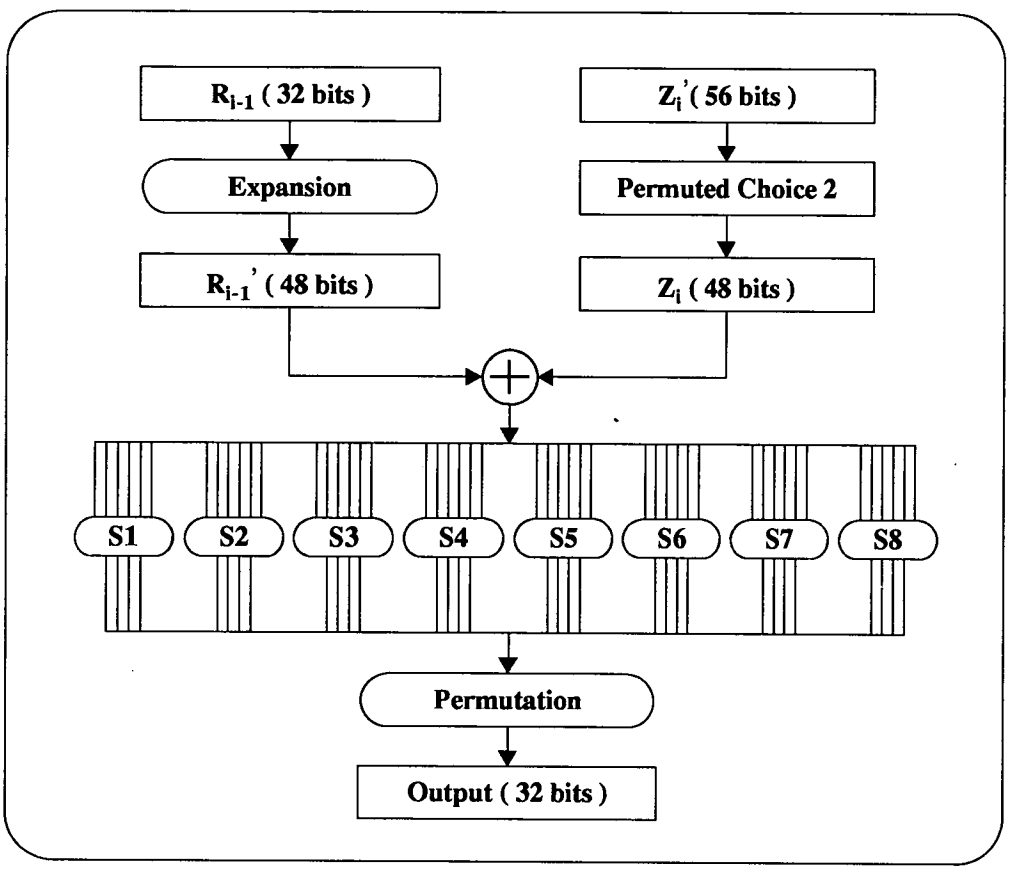

Figure 2.16 The DES Cipher Function

The S-boxes perform a four bit substitution on the inner four input bits controlled by the outer two input bits. They have been designed so that changing one input bit changes at least two output bits which provides an avalanche effect as the cipher proceeds.

2.3.4 Weak Keys

The nature of the key scheduling algorithm gives rise to some obviously bad choices of keys that cause the decryption process to be ***exactly*** the same as the encryption process. For example, any key resulting in C_0 and D_0 both equal to zero will produce identical Z_is. There are

four such weak keys. Semi-weak keys are similar but occur in pairs, encryption by one of these keys is equivalent to decryption by its dual. A full list of weak keys is given in the NBS Guidelines document [6].

The only danger is that weak keys might be used during multiple encryption, using DES to form a cascade cipher. The existence of these keys however, has fuelled speculation that DES might be a group. This would be catastrophic for multiple encryption since the closure property of groups means that successive encryptions would be equivalent to a single DES encryption. Fortunately there is strong evidence that DES is not a group [73].

2.3.5 The Future of DES

DES is probably the most closely analysed cryptography algorithm, yet despite intensive scrutiny no one has identified a weakness that could be exploited better than exhaustive cryptanalysis. The general consensus is that DES appears to be an excellent cipher given its small key length, and should find wide use for many years yet.

However, as mentioned on page 23 the standard was recommended for 10 to 15 years use with reviews every five years and is due its third review before January 1992. The NBS will then have to decide whether to reaffirm, revise, or withdraw the standard. The NSA meanwhile have been working on a program (CCEP) intended to design cryptography algorithms to replace DES [15]. According to Smid and Branstadt [143], the NSA have stated in a letter that the CCEP intends to provide Government endorsed cryptographic equipment. The algorithms will be designed by the NSA and not published, but made available through tamper-proof chips. Whether the CCEP program succeeds in finding a suitable replacement for DES remains to be seen.

2.4 Stream Ciphers

The difference between block and stream ciphers was explained on page 22. This section discusses stream ciphers in more detail and distinguishes between *synchronous* and *self-synchronous* stream ciphers, concluding with several examples of how block ciphers may be modified to take on certain desirable characteristics of stream ciphers.

The theory of stream ciphers is the subject of much research and may be pursued in more detail in [18] and [122]

2.4.1 Synchronous Stream Ciphers

It was stated in section 2.3 that stream ciphers transform input symbols in a manner that depends on the internal state of the system. In a synchronous stream cipher, the next state depends only on the present state, and is unaffected by the input symbols. When comparing block and stream ciphers, the absence of inter-symbol dependence may be advantageous in a noisy channel since one corrupted symbol will have no effect on any others. On the other hand, having no diffusion of the plaintext allows a cryptanalyst to alter one symbol in a message without having to worry about how this will affect the remaining symbols. Diffie and Hellman [47] discuss how error detecting codes may be applied to cryptosystems, and observed that a keyed or non-linear error detecting code may be applied to a synchronous stream cipher to provide automatic authentication.

The basic model of a synchronous stream cipher system is shown in Figure 2.17 where the exclusive-or function is used to combine the message stream with the *running key*. With the proliferation of digital information, it is easy to see why such binary additive stream ciphers are the most popular stream ciphers in use today. The exclusive-or function allows the encryption and decryption to be performed by identical devices since addition and subtraction are equivalent operations modulo two.

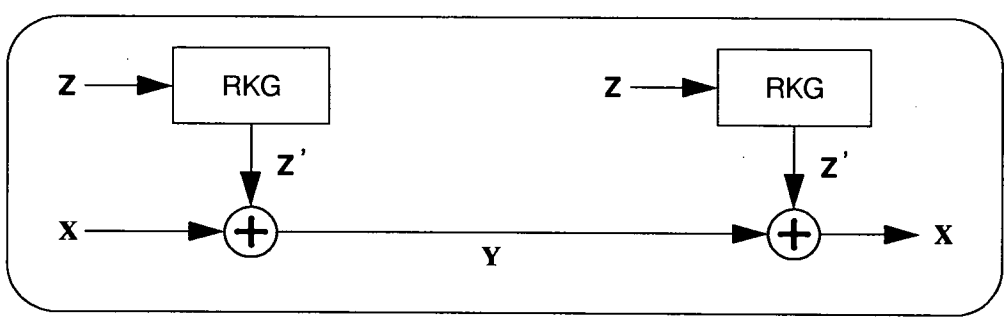

Figure 2.17 Synchronous Stream Cipher

The running key generator (RKG) in the above uses the key **Z** as a seed to generate the running key **Z'**. If this running key is truly random then the ciphertext will be statistically independent from the plaintext and the system will have perfect secrecy as described in

section 2.2.1. Chaitin [33], however, has shown in that no truly random sequences can be generated using a finite algorithm. Any finite state machine not subject to external influences will always cycle repeatedly through a fixed number of states producing a periodic output. Since a known-plaintext attack exposes the running key, the best a stream cipher designer can do, is to build the RKG in such a way that it is difficult for a resource limited cryptanalyst, upon observation of $Z'_1, Z'_2 \ldots Z'_n$, to reliably predict Z'_{n+1} without knowledge of the secret key **Z**. To achieve this, the running key **Z'** should have the following properties.

1. A long period
2. Good short term randomness
3. Large linear complexity.

The linear complexity of a sequence is defined as the length *L*, of the shortest linear-feedback shift-register (LFSR) that could have produced the sequence.

The reason for insisting on a large linear complexity *L*, is that the Berlekamp-Massey algorithm [19] [81] describes an efficient method of finding the shortest LFSR that could have generated the sequence after examining only *2L* bits of the running key. This effectively precludes the use of LFSRs alone as running key generators. The poor security of such a system under a known plaintext attack is demonstrated in [47] [41] and [18]. Good RKGs can, however, be designed using LFSRs as building blocks of a larger system as shown in Figure 2.18.

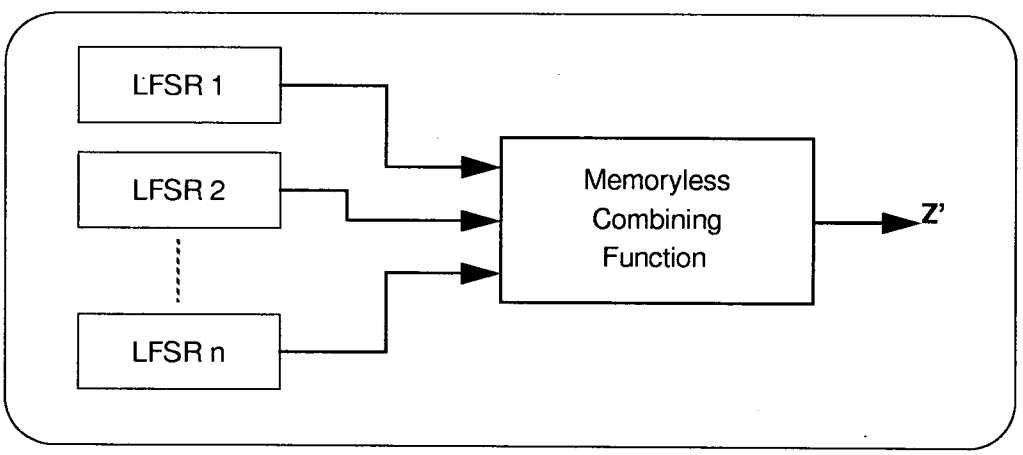

Figure 2.18 Running Key Generator

For example, Geffe [59] suggested the arrangement of Figure 2.19 as the memoryless combining function, where LFSR 2 selects the output from either LFSR 1 or LFSR 3.

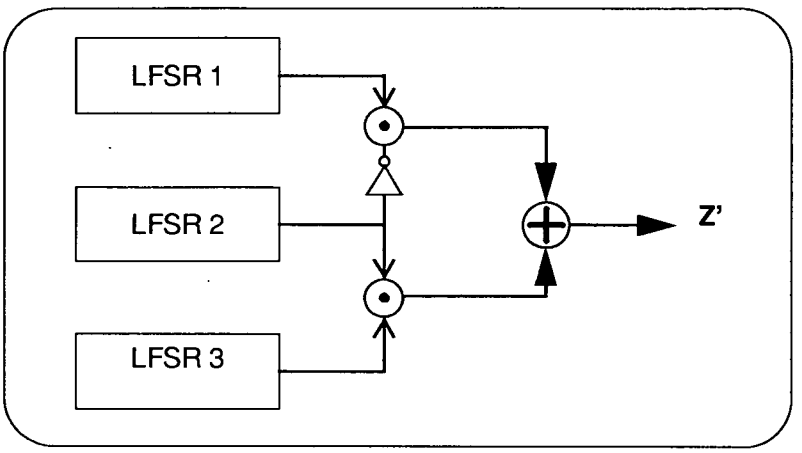

Figure 2.19 Geffe's Combiner

Siegenthaler [135] has shown that stream ciphers constructed from this, and several other combining functions, can be broken when subject to a ciphertext-only correlation attack.This is possible because leakage of the LFSR statistics through the combining function makes the cipher subject to a "divide and conquer" attack. In [134] Siegenthaler defined correlation-immunity for nonlinear combining functions and showed how to design combining functions that avoid leakage. He also proved that high correlation-immunity required the combining function to have a low nonlinear order. Rueppel and Staffelbach's work on linear complexity [123], however, showed that a high nonlinear order was needed if large linear complexity is desired. Thus in the design of memoryless combining functions, there is a trade-off between correlation-immunity and linear complexity.

To overcome this trade-off, combining functions with memory can be used although these functions are in general much more difficult to analyse. The memory can be provided by the LFSRs themselves as is done in [84]. Figure 2.20, adapted from [84], illustrates Massey and Rueppel's running key generator where two LFSRs are clocked at different rates to construct a combining function with memory that has a high linear complexity. In this example, the longer LFSR is clocked at d times the rate of the shorter one.

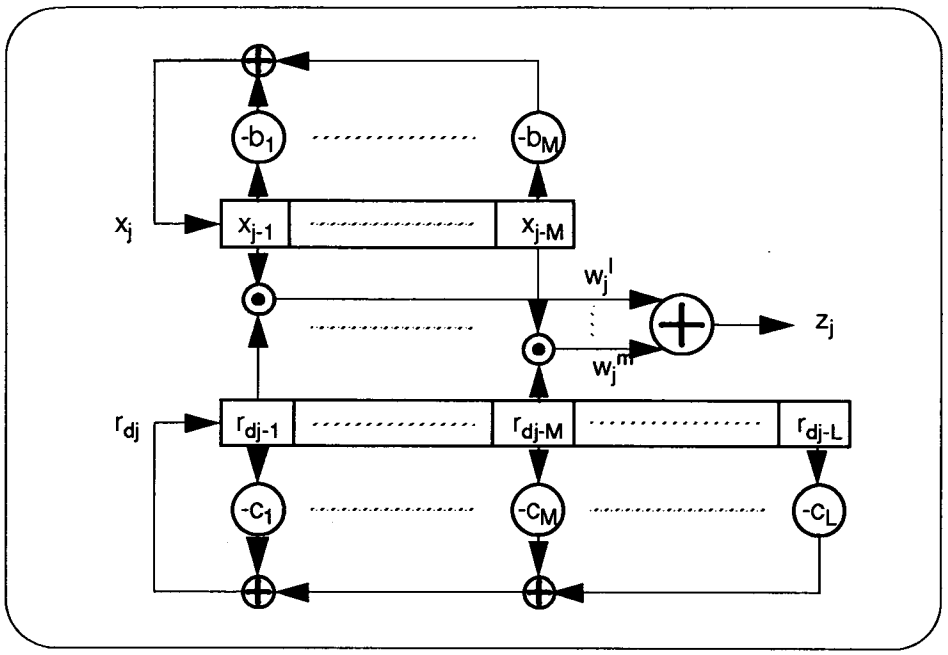

Figure 2.20 Massey - Rueppel Generator

The addition of memory to the combining function introduces diffusion, and so goes some way towards answering the criticisms of additive stream ciphers made on page 18.

Shift register sequences are not the only way to generate running keys. Figure 2.21 illustrates how a block cipher, such as DES, may be used in output feedback (OFB) mode to produce a synchronous key stream. The first encryption uses an initialization vector (IV) as input to the exclusive-or function.

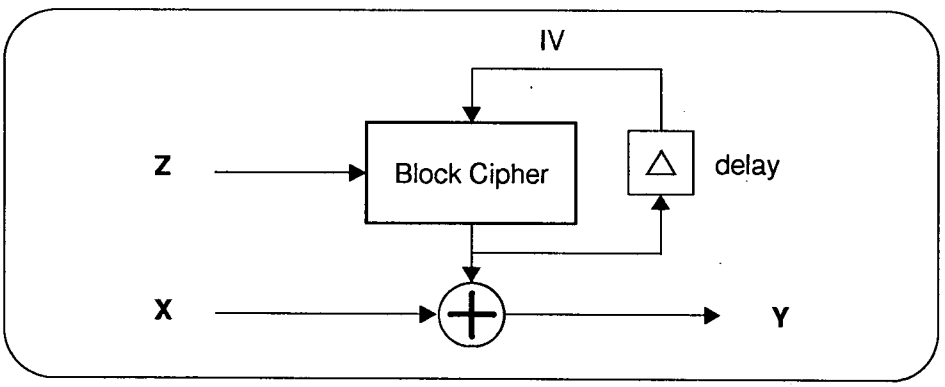

Figure 2.21 Output Feedback

Diffie and Hellman [47] demonstrate how the OFB cipher may be modified by using a counter as shown in Figure 2.22. This scheme eases random access to files since individual symbols can be deciphered by setting the counter to the appropriate value; with OFB the preceding ciphertext block must be known beforehand.

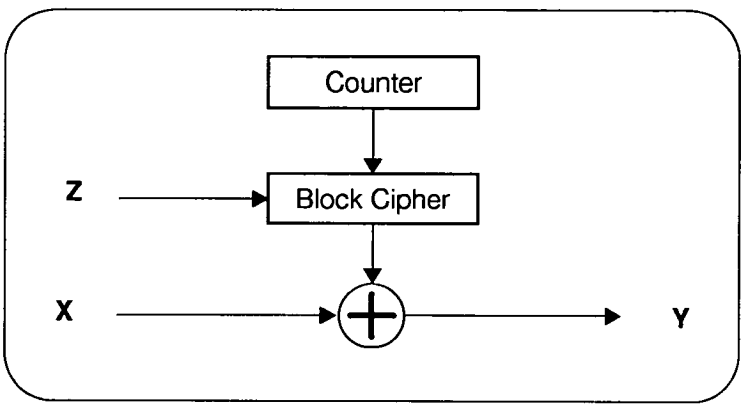

Figure 2.22 The Counter System

Becker and Piper [18] describe a similar system, replacing the counter with an LFSR.

2.4.2 Self Synchronous Stream Ciphers

The RKGs at the transmitter and receiver in the additive stream ciphers discussed above must always run in perfect synchronism. This may not always be possible in a practical situation. Self synchronous stream ciphers, on the other hand, derive each key symbol from a fixed number of the preceding input symbols and so, by definition, must have a limited amount of memory. Since these systems have limited memory, any errors in the input stream will produce a fixed number of errors in the output, after which correct operation is resumed.

The idea of self synchronous stream ciphers can be traced back to the autokey ciphers invented by Vigenère in the 16th Century. Vigenère's autokey ciphers were based on a substitution as used in the Ceaser cipher, but instead of a fixed amount, each letter was shifted by an amount determined by either the message or the ciphertext. Figure 2.23 illustrates the scheme. In (a) the message symbols form the key whereas in (b) it is the ciphertext symbols that are used. In both schemes the letter 'A' has been used as a seed.

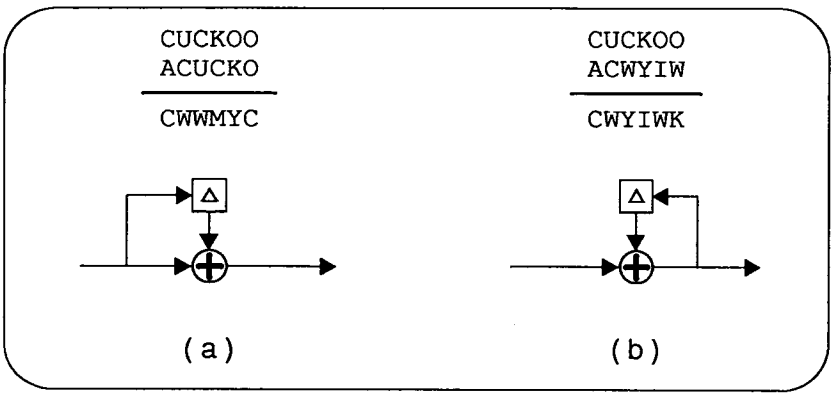

Figure 2.23 Vigenère's Autokey Ciphers

The feedback loop of Figure 2.23 (b) makes each output symbol dependent on the entire preceding message. Diffusing the message statistics over all the ciphertext makes cryptanalysis of this scheme much harder than that of Figure 2.23 (a) where the diffusion is only over message symbol pairs. Although the above procedure for generating the key stream exposes the key in the ciphertext, this is easily overcome by using a non-linear function, such as a block cipher, for this purpose. This is precisely what is done when DES is used in cipher block chain (CBC) mode [6].

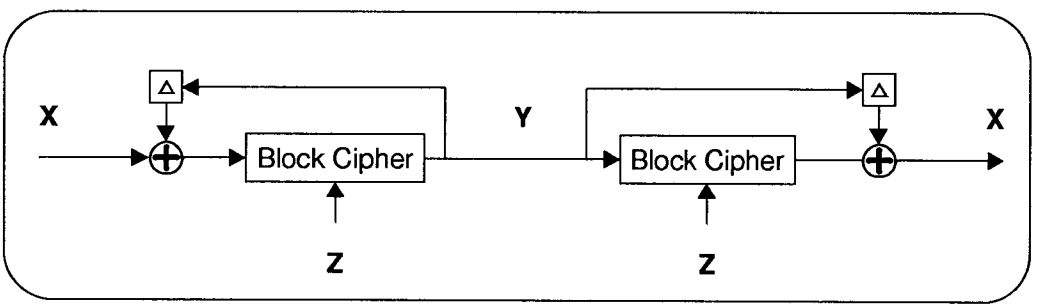

Figure 2.24 Cipher Block Chaining

Again, as with the OFB mode, CBC requires an initialization vector.

The cipher feedback (CFB) system shown in Figure 2.25 is similar to CBC in that it too diffuses the message throughout the ciphertext and, as all stream ciphers do, makes data tampering difficult for the cryptanalyst. Where the two approaches differ is in the data format: CFB may be adapted to the user's data format, and is not restricted to the size of the block cipher. The cost of this is the inefficiency in producing ciphertext that is not used.

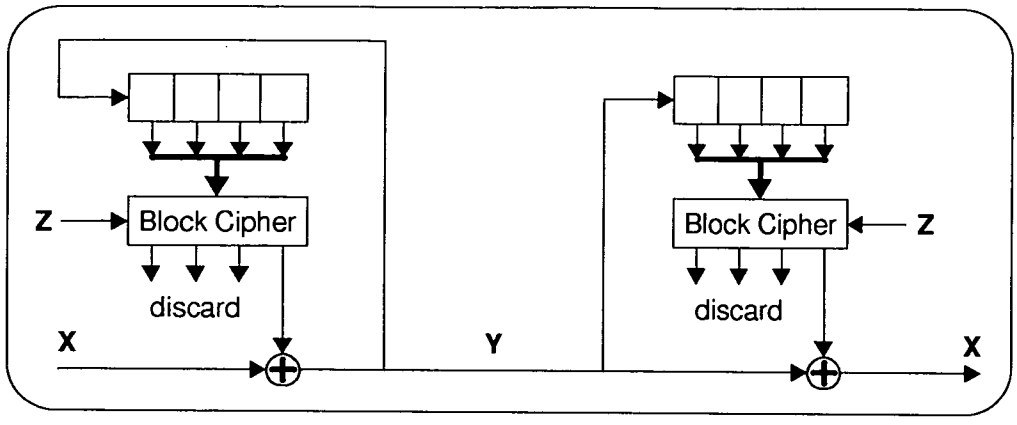

Figure 2.25 Cipher Feedback

2.5 Public Key Systems

A major problem with conventional cryptosystems described previously is the difficulty in distributing secret keys. In the classical system the encryption function and the decryption function are inseparable, both the sender and receiver must have the same key. How then can two users, who have never met before, agree in advance on a key that will be known to themselves and to no one else? This problem, and several secret key solutions are discussed by Diffie and Hellman in [47] section V-A, and by Ralph Merkle in [87].

Another limitation of conventional systems is the problem of digital signatures. Written signatures are used to verify that documents came from, or were witnessed by, a particular person. To be effective the signature has to be difficult to copy, so how can digital messages which can be copied perfectly, bear a signature? The authenticity systems of section 2.2.4 can prevent third party forgery but cannot ***settle disputes*** between sender and receiver.

These were the problems being addressed by Diffie and Hellman in 1976 [45] when they revealed that practically-secure systems can be built that require ***no secure transfer of any secret key whatsoever.*** Furthermore, by separating the encryption and decryption functions, public key cryptography provides an elegant solution to the authenticity problem.

2.5.1 Elements of Modern Cryptography

The important contribution that Diffie and Hellman made was the proposal of the "trap door one way function". *One way functions* were known at that time for their use in computer login protocols and access control [151], and may be defined as a class of functions that are easy to compute but difficult to invert. In a login protocol, the user's password is transformed by a one way function and stored, together with his name, in a read only password file. Each time the user logs in the password is transformed and checked against the contents of the file. Since a one way transform has been used, knowledge of this file is of no help in retrieving original passwords, and even a legitimate user will find it practically impossible to decipher his own password.

Trap door one way functions were defined by Diffie and Hellman to be one way functions for which a simply computed inverse ***does*** exist if certain "trap door information" is known. To be more specific, a trap door function is defined as a family of invertible functions $f_z(x)$ which may be used to define algorithms $E_z(x)$ and $D_z(x)$ that allow easy computation of $y = f_z(x)$ and $x = f_z^{-1}(y)$. However, for virtually all *z* and all *y* in the range of $f_z(x)$, it is practically impossible, without knowledge of *z* (the trap door information), to compute $x = f_z^{-1}(y)$, ***even if*** $E_z(x)$ ***is known.***

It is this ability to make $E_z(x)$ known to the public that gave rise to the term public key cryptography. With such a cryptosystem the encryption function and decryption function are separated and key distribution problem may be solved by simply having a publicly available directory of subscribers and their public keys. The digital signature problem is solved by a subscriber encrypting his message with his private key, the message being verified by decrypting it with the his public key. Figure 2.26 illustrates the public key cryptosystem and may be compared to the classical system on page 5.

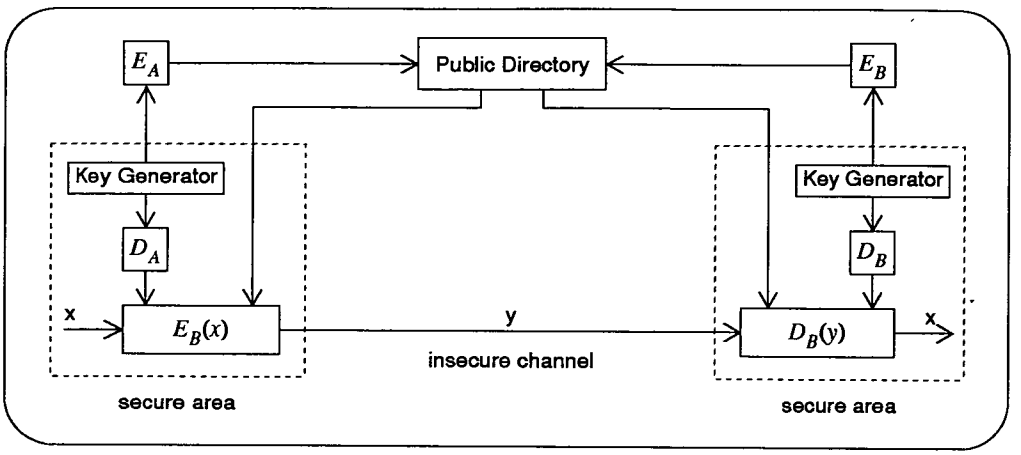

Figure 2.26 Public Key Cryptosystem

In Figure 2.26 authenticity is assumed to be guaranteed in all communications with the public directory. Diffie and Hellman [45] defined five properties such a public key cryptosystem should have:

1. The ciphertext space must be the same as the plaintext space
2. For all z E_z has an inverse D_z
3. For all z E_z and D_z are easy to compute for all messages and ciphertext
4. Without knowledge of z, it is infeasible to derive D_z from E_z
5. For all z it is feasible to compute inverse pairs E_z and D_z

2.5.2 Discrete Exponentials

In their 1976 paper, Diffie and Hellman conjectured the discrete exponential to be a good candidate for a one way function and suggested how this might be used as the basis of public key exchange protocol. This conjecture is based on the fact that if α is a primitive element in the Galois Field GF(q), and q is a large prime number, then the function

$$f(x) = \alpha^X \text{ modulo } q \qquad \text{(EQ 2.19)}$$

is easy to compute, taking at most $2\log_2(q)$ multiplications using Knuth's square and multiply procedure [75]. Calculating discrete logarithms on the other hand is not so straightforward. Pohlig and Hellman [107] have shown that when q is chosen so that $q - 1$ has a large prime

factor, then calculation of the discrete logarithm will take the order of $\sqrt{q}$ multiplications modulo q. Ideally q - 1 should be twice a prime number.

The operation of the Diffie - Hellman public key distribution scheme is illustrated here by the introduction of Rivest Shamir and Adleman's hypothetical users Alice and Bob [120].

Alice and Bob choose random messages x_A and x_B respectively, and transform them using (EQ 2.19) to obtain y_A and y_B. α and q are assumed to be public knowledge. Alice and Bob may now exchange y_A and y_B and calculate the common key $K_{AB} = \alpha^{x_A x_B}$ modulo q as shown below.

$$\text{Alice calculates } K_{AB} = y_B^{x_A} \text{ modulo } q \qquad \text{(EQ 2.20)}$$

$$\text{Bob calculates } K_{AB} = y_A^{x_B} \text{ modulo } q \qquad \text{(EQ 2.21)}$$

Unless an intruder can calculate K_{AB} from y_A and y_B without first obtaining either x_A or x_B, then he has to compute $K_{AB} = y_B^{\log_\alpha y_A}$ modulo q. This technique forms the basis of CYLINK'S Secure Electronic Exchange of Keys, or SEEK system [98].

In 1985 Taher El Gamal [48] showed how this scheme could be developed into a public key cryptosystem for key distribution and digital signatures. El Gamal however, produced twice as much ciphertext as the original message. More recently, Kevin McCurley [85] demonstrated how, by using carefully chosen composite numbers for the key, security could be *proved* mathematically. McCurley then showed how El Gamal's scheme could be modified so that a cryptanalyst had to first factor the modulus before breaking the original cipher.

2.5.3 Knapsack Cryptosystems

Another noteworthy one way function is the knapsack function. This function is derived from the notion of packing items into a knapsack with the intention of filling it completely with no space left over, hence the name. Mathematically the problem may be stated as:

Given a set of n positive integers $a_1, a_2, \ldots, a_n$ and a positive integer s,

does there exist $x_1, x_2, \ldots, x_n$, with $x_i \in \{0, 1\}$ for $i = 1, 2, \ldots, n$

such that $s = x_1 a_1 + x_2 a_2 + \ldots + x_n a_n$.

This problem is well known in the field of complexity theory and belongs to a class of problems known as NP - complete which, *in the general case*, are considered to be computationally complex.

An n-bit message vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is encrypted by forming the dot product with the cargo vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)$. Although the general case of the knapsack problem is difficult to solve there are particular cases, superincreasing knapsacks, that are easy to solve. A superincreasing knapsack is one in which each integer in the series, is greater than the sum of all those preceding it. In other words if in the above set of integers:

$$a_i > \sum_{j=1}^{i-1} a_j$$

then $s < a_n \Leftrightarrow x_n = 0$. Indeed, if $\mathbf{a} = (1, 2, 4, \ldots, 2^{n-1})$ then the solution is trivial. Shortly after Diffie and Hellman's 1976 publication Ralph Merkle began working with Hellman to use this fact to build a trap door into the knapsack cipher.

The Merkle-Hellman knapsack cipher [88] begins with a superincreasing knapsack, then multiplies each element in the cargo vector **a'** by a scalar constant *w* modulo *m*. The modulus is chosen to be greater than the sum of all elements in the original cargo vector, and the scalar *w* is chosen so that it has an inverse w^{-1} modulo *m*. The resulting vector, **a** = *w***a'** mod *m* is then transposed, the idea being that without knowledge of *w*, *m* and the transposition, the simple knapsack problem has become a difficult knapsack.

Now if Alice and Bob wish to communicate, Alice can make **a** public, allowing Bob to encipher his message **x** by calculating $s = \mathbf{a} \cdot \mathbf{x}$. Alice recovers the message by calculating:

$$\begin{aligned} s' &= w^{-1} s \bmod m \\ &= w^{-1} (\sum a_i x_i) \bmod m \\ &= w^{-1} (\sum w a_i' x_i) \bmod m \\ &= (\sum a_i' x_i) \bmod m \end{aligned}$$

And since $m > \sum a'_i \Rightarrow s' = \sum a_i' x_i$ in integer arithmetic as well as mod *m*. So, by using her secret key information (w^{-1}, *m*, and the permutation), Alice can transform the difficult knapsack *s* to a simple knapsack *s'* and then extract the message **x**.

Merkle and Hellman proposed improving security further by repeating the procedure to obtain **a** from **a'** several of times, forming a more obscure public key with each iteration.

The trap door in the Merkle - Hellman knapsack cipher is the use of modulo multiplication to disguise an easy problem, the superincreasing knapsack, as something difficult. In 1984 Adi Shamir [129] broke this cipher, not by solving the problem, but by stripping off this disguise. The following year Brickell [29] demonstrated how the iterated knapsack could be broken. The events that led to the breaking of the knapsack cipher are described in the 1988 paper by Brickell and Odlyzko [31].

2.5.4 The McEliece System

McEliece [86] based his encryption system on the error correcting codes known as *Goppa codes* which belong to the same class of error correcting codes as the Reed - Solomon codes and can be decoded by the same well known and efficient methods. McEliece applied a similar technique to these codes as Merkle and Hellman applied to the superincreasing knapsack, that is, he disguised the Goppa code as a more general linear code for which decoding without the key, like the general knapsack problem, is considered to be NP complete [44].

The error correcting scheme multiplies the message vector **x** by a matrix **G** to produce a code-word vector **y** for transmission. The received vector **y'** may contain errors that are removed with knowledge of **G** to recover the original message **x**. McEliece modified this scheme by disguising the matrix **G** by pre and post multiplying it by two other matrices to form the public key **G'** where

$$\mathbf{G'} = \mathbf{SGP} \qquad \text{(EQ 2.22)}$$

Using **G'** instead of **G** generates a linear code with the same rate and minimum distance as the original Goppa code. To encrypt the message vector **x**, it is multiplied on to the public matrix **G'** and the result added to a locally generated error vector **e**

$$\mathbf{y} = \mathbf{xG'} + \mathbf{e} \qquad \text{(EQ 2.23)}$$

To decode the ciphertext, it is first multiplied by $\mathbf{P}^{-1}$ to obtain a Goppa code word that can be decoded with knowledge of **G**. The result is multiplied by $\mathbf{S}^{-1}$ to recover **x**.

The McEliece system has never achieved wide acceptance. Several reasons have been suggested for this: the data expansion may be undesirable, or the need for large public keys (of the order of 10^6 bits). The similarity to knapsack ciphers may also be a reason, although Adams and Meijer [10] have recently demonstrated that a knapsack like attack is extremely unlikely to succeed. Adams and Meijer also show that with well chosen parameters cryptanalysis of the McEliece system is significantly more difficult than cryptanalysis of DES and compares favourably with RSA.

Perhaps the biggest weakness of McEliece's system, as far as public key encryption is concerned is its unsuitability for authentication. The nature of the error correcting code precludes a one to one mapping between ciphertext and plaintext. For authenticity, the plaintext message must be signed with a private key in such a way that application of the encryption transformation using the public key will only produce a meaningful message if the correct key is used. For the McEliece system this is impossible.

2.5.5 Other Public Key Schemes

In addition to the three methods discussed in this section and the RSA cryptosystem described in section 2.6, a number of other public key cryptosystems have been proposed.

In 1978 Rabin [112] produced a variant of RSA for which he could *prove* that cryptanalysis was equivalent to factoring the modulus. This scheme was slightly more complicated than RSA but was simplified in 1979 by Williams [152]. However, as pointed out by Rivest and cited in Williams' paper, both these schemes are vulnerable to a chosen plaintext attack which in a public key environment is a distinct possibility.

A fast signature scheme based on polynomial congruential equations was proposed by Ong, Schnorr and Shamir [102] in 1984, the OSS scheme, but was broken in 1987 by Pollard and Schnorr [108]. Goldwasser Micali and Rivest [61] proposed a method which they proved to be secure against a chosen plaintext attack and which was subsequently modified by Goldreich [60] in 1986.

Another signature scheme has been suggested by Fiat and Shamir [54] and subsequently modified [51] [90] to be provably secure against chosen plaintext attacks. However this

scheme results in data expansion which although improvements have been made [100] remains slower than the original scheme. More recently, in 1990, a signature scheme has been proposed by Okamoto [101] which although not provably secure, claims to be up to twenty times faster than RSA.

2.6 RSA

By far the best known public key encryption scheme is due to Rivest Shamir and Adleman, RSA [120]. This scheme, like the Diffie Pohlig Hellman scheme of section 2.5.2, employs discrete exponentiation, however, the security of the RSA scheme lies in the fact that finding large prime numbers is computationally easy whereas factoring a product of two primes appears to be computationally infeasible.

2.6.1 The RSA Cryptosystem

In the RSA cryptosystem both the plaintext space and the ciphertext space are the ring of integers $\mathbf{Z}_m$, where m is a modulus formed by the product of two large random prime numbers p and q.

The encrypting transformation is controlled by a public key comprising a pair of numbers e and m, where m is the modulus and e is an element of $\mathbf{Z}_m$. The transformation is defined as

$$y = E(x) = x^e \text{ modulo m} \qquad \text{(EQ 2.24)}$$

The decrypting transformation is similarly defined by the same modulus, and a private key d.

$$x = D(y) = y^d \text{ modulo m} \qquad \text{(EQ 2.25)}$$

The private key d is chosen to be the multiplicative inverse of e in the ring of integers $\mathbf{Z}_{\Phi(m)}$ where $\Phi(m)$ is Euler's totient function, explained in section A.4. The extended Euclidean algorithm described in section A.1 may be used to calculate d given e and $\Phi(m)$. Examination of this algorithm will show that inverses may be computed in polynomial time.

Both these transformations are easily computed provided the keys are known. To recover the plaintext from the ciphertext without knowledge of the private key requires either calculating discrete logarithms or alternatively, obtaining d from knowledge of e and m, this however is known to be at least as difficult as factoring the modulus.

2.6.2 The Underlying Mathematics

To demonstrate that the RSA cryptosystem decrypts correctly it is necessary to show that

$$\begin{aligned} x &= D(E(x)) \\ &= E(x)^d \\ &= (x^e)^d \text{ modulo m} \\ &= (x^e)^d \text{ modulo m} \\ &= x^{e \cdot d} \text{ modulo m} \end{aligned} \qquad \text{(EQ 2.26)}$$

So to prove correct decryption it is necessary to prove that:

$$x^{e \cdot d} = x \text{ modulo m} \qquad \text{(EQ 2.27)}$$

Case 1: x and m relatively prime:

Since *e* and *d* are chosen such that

$$e \cdot d = 1 \text{ modulo } \Phi(\text{m}), \qquad \text{(EQ 2.28)}$$

then

$$e \cdot d = Q\Phi(m) + 1$$

so

$$x^{e \cdot d} = x^{Q\Phi(m)+1}$$

$$x^{e \cdot d} = x \cdot x^{Q\Phi(m)} \qquad \text{(EQ 2.29)}$$

Now, Euler's theorem (see appendix A.4) states that if *x* and *m* are relatively prime then

$$x^{\Phi(m)} = 1 \text{ modulo m}, \qquad \text{(EQ 2.30)}$$

thus

$$x^{Q\Phi(m)} = 1^Q = 1$$

and (EQ 2.29) simplifies to

$$x^{e \cdot d} = x \text{ modulo m} \qquad \text{(EQ 2.31)}$$

Thus for *x* and *m* relatively prime the RSA cryptosystem decrypts correctly, and since there are only $p + q - 1$ integers in $\mathbf{Z}_m$ divisible by *p* or *q* this covers virtually all cases when *p* and *q* are large.

Case 2: x and m not relatively prime:

If $x = 0$ then the claim that $x^{e \cdot d} = x$ modulo *m* is trivial, so assuming $x \neq 0$ and that *p* divides *x* (either *p* or *q* can divide *x* but not both) then we have:

$$x^{e \cdot d} = x \quad \text{modulo p} \qquad \text{(EQ 2.32)}$$

and

$$x^{e \cdot d} = x^{Q\Phi(m)+1} \quad \text{modulo q}$$

$$x^{e \cdot d} = x^{Q(p-1)(q-1)+1} \quad \text{modulo q}$$

$$x^{e \cdot d} = x \cdot x^{Q(p-1)(q-1)} \quad \text{modulo q}$$

$$x^{e \cdot d} = x \cdot (x^{q-1} \text{ modulo q})^{Q(p-1)} \quad \text{modulo q} \qquad \text{(EQ 2.33)}$$

but Fermat's theorem (section A.5 on page 129) states that if *q* is prime then

$$x^{q-1} = 1 \quad \text{modulo } q, \qquad \text{(EQ 2.34)}$$

so (EQ 2.33) simplifies to

$$x^{e \cdot d} = x \quad \text{modulo q} \qquad \text{(EQ 2.35)}$$

Now, since $m = pq$, $x \in \mathbf{Z}_m$ and $x^{e \cdot d} \in \mathbf{Z}_m$ it is straightforward to deduce from (EQ 2.32) and (EQ 2.35) that:

$$x^{e \cdot d} = x \quad \text{modulo m} \qquad \text{(EQ 2.36)}$$

Equations (EQ 2.31) and (EQ 2.36) show that RSA decrypts correctly in all cases. Furthermore, because

$$D(E(x)) = E(D(x)) = x^{e \cdot d} \qquad \text{(EQ 2.37)}$$

the encryption and decryption functions are mutual inverses, and a decryption followed by an encryption has the same effect as an encryption followed by a decryption. Thus RSA may be used for authenticity as well as secrecy.

2.6.3 Operation of RSA Cryptosystem

The operation of the RSA public key cryptosystem is illustrated here referring again to Rivest Shamir and Adleman's hypothetical users Alice and Bob [120].

1. Alice chooses two large random primes p and q, and from them calculates m and $\Phi(m)$. These primes may be chosen by probabilistic methods which are faster than searching for true primes.
2. To ensure her private key has an inverse, Alice selects d such that $\gcd(d, \Phi(m)) = 1$, and then computes e, the inverse of d in $\mathbf{Z}_{\Phi(m)}$.
3. Alice publishes e and m, but keeps d, p, and q secret.
4. Bob may now send messages to Alice using Alice's public key, which only Alice can decipher with her private key.

This provides secrecy but offers *no authenticity whatsoever.* Anyone with access to Alice's public key can send messages to her claiming to be Bob and Alice has no means of verifying the authenticity of the message. To provide authenticity,

1. Bob transforms the message he wishes to send using *his own* private key and sends this result to Alice.
2. Since $D(x)$ and $E(x)$ are mutual inverses (EQ 2.37), Alice is able to recover the message using Bob's public key.

This time we have *authenticity without secrecy,* anyone with access to Bob's public key may decipher his message. One use of authenticity without secrecy is suggested in section 2.2.4 on page 18.

These two examples have shown how secrecy and authenticity are completely independent attributes of the RSA cryptosystem. To combine both attributes, the message must be transformed twice:

1. Bob first transforms his message using his private key $D_B(x)$, and then transforms the result using Alice's public key and sends $E_A(D_B(x))$.
2. Now secrecy is ensured, since only Alice is able to invert the second transform. Having done this, Alice can now use Bob's public key to recover the message and verify its authenticity.

If, in the latter example, Bob's key used a modulus that was smaller than the modulus used by Alice, the message would have to be re-blocked between transformations. Rivest Shamir and Adleman recognised this and suggested a *threshold scheme* to avoid re-blocking. The idea is

to choose a threshold value, *h*, for the cryptosystem. Each user then maintains two sets of keys, one set with modulus less than *h*, for authenticity, and a second set, with modulus greater than *h*, for secrecy. Alternative solutions have been suggested by Konfelder [78], and Davies and Price [38] [39].

2.6.4 Security of RSA

The security of the RSA cryptosystem relies on the difficulty of factoring the modulus *m* to obtain $\Phi(m)$ and thus the private key *d*. In order to defeat known factoring algorithms, the primes *p* and *q* that form the modulus must be chosen very carefully. Rivest Shamir and Adleman suggest the following restrictions on the choice of primes:

1. The primes p and q should differ in length by only a few digits.
2. Both $p - 1$ and $q - 1$ should have large prime factors.
3. The greatest common divisor of $p - 1$ and $q - 1$ should be small.

Rivest Shamir and Adleman also suggested restricting the private key *d* to the range $\max(p, q) < d < m$ and restricting *e* such that $e > \log_2(m)$.

Simmons and Norris [141] showed how the RSA cipher could be broken when subject to an iteration attack. They demonstrated that for certain keys, repeated application of the enciphering transformation would eventually yield the original message. In response to this, Rivest [115] showed that if *p* and *q* were chosen such that $p - 1$ and $q - 1$ both had a large prime factors *r* and *s*, and that $r - 1$ and $s - 1$ *also* had large prime factors, then the probability of success in an iteration attack was so small as to be inconsequential.

In another attack on RSA, Blakley and Borosh [25] demonstrated that for certain keys, there are at least nine messages not concealed by RSA. That is to say, for certain keys $x^e = x$ modulo *m*. Blakley and Borosh suggest that to avoid this feature, *p* and *q* should be ***safe*** primes, where $p = 2r + 1$, and $q = 2s + 1$ and *r* and *s* are odd primes. Pohlig and Hellman [107] also suggest the use of safe primes in their scheme.

Although care has to be taken in the selection of primes for RSA, both Williams and Schmid [153], and Gordon [62] [63] have shown that finding appropriate values for *p* and *q* is not difficult.

So, assuming the primes have been chosen correctly, the cryptanalyst is faced with the problem of either factoring the modulus, or calculating discrete logarithms. Algorithms for both these problems are described by Lenstra and Lenstra [80], and Pomerance [109].

More recently, attention has been focused on the announcements in the press in June, 1990 by Lenstra and Manasse, that they had succeeded in factoring the ninth Fermat number, *F9*, and the implications this has for the security of RSA. In an article posted on the Internet bulletin board, Ronald L. Rivest presented an analysis of this, pointing out that the *number field sieve* or *NFS* algorithm used by Lenstra and Manasse is specifically designed to factor numbers that, like *F9*, have a very simple structure of the form $a^b + c$ where c is a relatively small, $F9 = 2^{512} + 1$. Numbers with such a special structure are extremely rare and unlikely to arise in practical cryptography. Rivest, however, identified three important points raised by this achievement:

1. The status of factoring is still subject to further developments, and conservative choice of key length should be made.
2. The *NFS* algorithm may yet be developed to cope with more general numbers, and the potential impact of this should be considered.
3. Despite best efforts, factoring remains a very hard problem, the best algorithm [80] taking $O(e^{\sqrt{(\log n)\log\log n}})$ time.

If the *NFS* algorithm were extended to cope with more general numbers, Rivest estimated that the time required to factor a 512 bit number would be of the order of 2×10^7 MIP-year, (where one MIP-year is the work done by a one MIP computer running for one year), which is clearly a substantial degree of security.

Chapter 3 Implementing Public Key Cryptosystems

Chapter 2 presented a broad view of contemporary cryptographic techniques. This chapter concentrates on public key cryptography and looks at the work that has been done to implement these systems both in hardware and in software. Since public key techniques tend to involve more computationally intensive algorithms than their secret key counterparts, they present a technical challenge to achieve comparable data throughput. As will be evident in conclusion to the following section, it is the poor performance of public key systems in this area that has impeded their use in practical applications and motivated research to design more efficient software or hardware solutions. As the results of such research have been emerging, confidence in public key systems has been growing to the extent where they are now being considered by such bodies as the ISO [110] and CCITT [9]

The public key algorithm most frequently referred to in the literature is RSA. The popularity of this algorithm is probably due to its versatility, and in particular its suitability for digital signatures. Other well known algorithms such as exponential key exchange, discussed in section 2.5.2, and zero knowledge proofs [54], like RSA, rely on modular exponentiation and it is this problem that the majority of research publications have addressed.

3.1 Software

The prevalent opinion at the time the RSA cipher was published was that it could not be implemented effectively in software, in 1980 Ronald Rivest even suggested that "a typical microprocessor based implementation might achieve an encryption rate of ten bits per second" [117]. This attitude did not, however, discourage the development of RSA software. Many situations exist, such as key management, where high encryption rates are not crucial to overall system performance.

3.1.1 Initial Work

Within a year of Rivest Shamir and Adleman's publication Michelman [91] announced a general purpose computer based implementation of the algorithm, and in 1985 the National Physical Laboratory published a PASCAL version [12] that would run on a BBC

microcomputer. Performance figures for RSA encryption, in software, with a 512 bit modulus were presented at the 1986 cryptology conference, *CRYPTO '86*, by Gordon Rankine [113]. He suggested typical encryption times of 4 minutes using a Motorola 6809, 70 seconds on an Intel 8086, and 30 seconds on a Motorola 68000. At the same conference Paul Barrett [16] presented the work he had done to execute the RSA algorithm on the new digital signal processing chip from Texas Instruments, the TMS32010. Making use of the special hardware on this chip and a long multiplication algorithm that reduced propagation of carries, Barrett achieved typical encryption times of 2.5 seconds for 512 bit RSA.

The following year Jung [70] described techniques to minimize both the space and time requirements for the RSA algorithm when implemented on a general purpose computer. Jung proposed a *common encryption exponent* to be shared by every user, thus defining the modulus as the public key. The common encryption exponent suggested by Jung was the fifth Fermat prime $2^{16}+1 = 65537$. In doing this, the storage requirement for public keys is immediately halved and, by choosing a relatively short exponent, the encryption time is drastically reduced. The decryption time however, will depend on the size of the decryption exponent and hence the value of the modulus chosen by the user. Storage requirements are reduced further by restricting moduli to be "close" to a power of two as is the case with the OSS scheme [102]. To speed up decryption Jung performed the decryption calculations using the shorter moduli p and q and used the Chinese Remainder Theorem, as suggested by Quisquater and Couvreur [111], to obtain the result modulo $m = pq$.

Applying these techniques Jung implemented the RSA cryptosystem on three general purpose computers. On a 0.8 MIPS Siemens mainframe Jung could generate a 512 bit signature in 1.5 seconds, and check the authenticity of a signature in 0.3 seconds. On a z80 based hand-held computer, the same tasks took 45 seconds and 3 seconds respectively for a 256 bit modulus. An 80186 based PC, programmed in 'C' and assembler could sign a 256 bit message in 1.5 seconds, and verify it in 0.2 seconds.

In 1989 Beth and Gollmann [22] reviewed several algorithms for public key cryptography. Although these algorithms were originally intended for dedicated hardware, many of the ideas can be directly applied to software solutions. In their paper, Beth and Gollmann suggest that

computers based on the 68000 family of processors should be capable of encrypting a 512 bit modulus block in less than one second.

3.1.2 Bong and Ruland

Dieter Bong and Christoph Ruland [26] published their work on optimized software techniques for modular exponentiation in 1989. Like Jung, they too advocate the use of a small fixed modulus for encryption and the Chinese Remainder Theorem for decryption. Bong and Ruland present several algorithms for modular arithmetic and show that a careful choice of algorithm has to be made to suit both the key length and the characteristics of the processor to be used. They tested their algorithms on two general purpose microprocessors, and a dedicated encryption chip, the latter is discussed in section 3.2.4. For the 68000 microprocessor running at eight MHz, encryption times of 110 ms. were achieved for 512 bit modular exponentiation, with decryption times of 6.2 seconds. The second processor, an eight MHz 80286, achieved encryption times of 63 ms. and decryption times of 3.7 seconds under the same conditions.

3.1.3 Laurichesse

in 1990 Denis Laurichesse [79] modified the standard "square and multiply" routine [75] for exponentiation to a more general "raise to the power and multiply" algorithm not restricted to arithmetic in base 2. Laurichesse found the optimal base for this algorithm to be 16, for which a 14% reduction was achieved in the number of operations involved in 256 bit exponentiation. Using this algorithm together with the Chinese Remainder Theorem and Montgomery's "N residue" arithmetic [94], which eliminates division in modular calculations, Laurichesse developed a multiple precision arithmetic algorithm for RSA encryption suitable for software implementation.

The algorithm was coded in 'C' and assembler for several general purpose microprocessors and for the T800 transputer. The decryption rates for 512 bit RSA are shown in Figure 3.1, where the performance benefits of 32 bit processors can be clearly seen.

Laurichesse proposed a hardware design making use of an AMD 29C323 for multiplication which is claimed to achieve decryption rates of 50 Kbits/s. The author also suggests that rates of 300 Kbits/s could be achieved using techniques developed by Shand et al. [130].

Host	Processor	Freq.(MHz)	word length (bits)	rate(bits/s)	time(s)
PC AT	80286	8	16	205	2.50
PC AT	80386	16	32	900	0.57
PCAT	80386	20	32	1100	0.47
SUN 3/60	68020	20	32	1150	0.45
BULL DPX 2000	68030	25	32	1500	0.34
PC AT	T800	20	32	1670	0.31

Figure 3.1 Decryption Rates Achieved by Laurichesse

3.1.4 Shand

Also in 1990, Mark Shand at DEC's Paris Research Laboratory completed his work on hardware/software trade-off involved in long integer arithmetic [130]. This work was based on the DEC BigNum [128] software package for high performance long integer arithmetic, which Shand and his colleagues modified to optimise the inner loops of the algorithms. Again the Chinese Remainder Theorem was used to speed up decryption, and Montgomery's technique was applied to eliminate division from the modular arithmetic calculations. Shand et al. report decryption times for 512 bit RSA of 3.53 seconds using the standard BigNum package on a 68020 based machine, without using the Chinese Remainder Theorem. Applying the Chinese Remainder Theorem under the same circumstances resulted in a decryption time of 0.51 seconds which clearly illustrates the advantage of this technique. Running this modified software on a MIPS R2000 machine resulted in decryption times of 49.7 ms. for 512 bit RSA, or a decryption rate of 10300 bits per second.

Further increases in throughput were achieved by using hardware accelerators, as discussed in section 3.2.

3.1.5 Summary

Figure 3.2 summarizes the software performance in implementing the RSA cryptosystem. The times given in the Encrypt and Decrypt columns are for encryption and decryption with a 512 bit modulus unless otherwise stated, and the times for decryption, unless otherwise stated, are all achieved by taking advantage of the Chinese Remainder Theorem.

	Year	Processor	Encrypt	Decrypt	Comments
Rankine	1986	6809	4 min.		
		8086	70 sec.		
		68000	30 sec.		
Barrett	1986	TMS32010	2.5 sec.		
Jung	1987	z80	45 sec.		256 bits, short 'e' key
		80186	1.5 sec.		256 bits, short 'e' key
		mainframe	1.5 sec.		512 bits. short 'e' key
Bong &	1989	68000	110 ms.	6.2 sec.	short 'e' key
Ruland		80286	63ms.	3.7 sec.	
Laurichesse	1990	80286		2.50 sec.	
		80383		0.47 sec.	
		68020		0.45 sec.	
		68030		0.34 sec.	
		T800		0.31 sec.	
Shand	1990	68020		3.53 sec.	without CRT
		68020		0.51 sec.	with CRT
		R2000		50 ms.	with CRT

Figure 3.2 Summary of RSA Software Performance

To use the Chinese Remainder Theorem, the factors p and q of the modulus m must be known, thus this technique can only be applied with knowledge of the private key. Transformations involving public keys can not use this short-cut.

So given this restriction, public key transformations in software will take the order of seconds for a 512 bit block, giving encryption rates of around 500 bits/s at best. This does not compare favourably with secret key cryptosystems, the DES cipher has recently been reported [143] as achieving encryption rates ranging from 20 Kbits/s on a PC to 100 Kbits/s on a VAX 780.

To compete, in terms of data throughput, with conventional cryptosystems either new algorithms for computing the modular exponential will have to be developed, or the RSA system will have to implemented efficiently in hardware. From a security point of view the hardware solution is obviously most attractive, ideally implementing the whole encryption unit in VLSI inside a tamper proof package.

3.2 Hardware

With such comparatively low encryption rates, it is hardly surprising that much research over the past decade has concentrated on improving the performance of public key algorithms by designing dedicated hardware systems.

3.2.1 Initial Work

Rivest Shamir and Adleman's paper was published in 1978. The following year the U.S. Sandia National Laboratories announced a board level implementation of the RSA system [44] capable of between 100 and 400 bits/s, followed by chip designs by Reiden et al. [114] and by Brickell [28]. Reiden used two identical chips to execute the square and multiply exponentiation algorithm [75], one chip for squaring, the other for multiplying. Each chip was a dedicated 336 bit modular multiplier designed in 3μ CMOS technology that adopted Blakley's [24] approach of reducing the partial products as they were formed to restrict word growth. Reiden et al. expected 20 MHz. operation, and encryption times of 0.78 seconds for 336 bit blocks, or 420 bits/s. In the second Sandia design, Norris and Simmons [99] examined the modular multiplication algorithm and realised that eliminating carry propagation during the formation of partial products could reduce the time required to form a product from $O(n^2)$ to $O(n)$ for n-bit data. They modified the well known carry save adder, used to eliminate carry propagation in ordinary multiplication, to deal with what they called "delayed carry" integers. Brickell adopted

this technique and in 1982, proposed a design that used four delayed carry multiplier chips running at 20MHz. This design was expected to achieve encryption rates of up to 20 Kbits/s for 512 bit modular exponentiation.

According to Diffie [44], Ronald Rivest also produced a board at approximately the same time as the Sandia designers, capable of encryption with a 100 bit modulus in one twentieth of a second. This was intended as proof of concept. Rivest Shamir, and Adleman then proceeded to design an NMOS chip [117] with a 512 bit ALU intended to be used as a general purpose big number processor. Running at 4 MHz. this chip was expected to achieve a throughput in excess of 1.2 Kbits/s. The chip was fabricated but due to a design error was too unreliable to complete a full encryption [118].

The Japanese company NTT had also been working on the design of an RSA chip, and in 1982 Miyaguchi [92] published the design of cascadable chip for calculating the modular exponential. The algorithm Miyaguchi used performed multiplication and division by the modulus in one operation and, as Brickell had done, used an approximation method to compute the residues. This design was claimed to be capable of 50 Kbits/s for 512 bit encryption.

In the following year, Simmons and Tavares [142] from Queen's University in Canada announced the design of modular multiplication chip designed in 6μ NMOS technology. Their chip computed the product first, and then performed modular reduction by subtracting the modulus from the result until the sign bit changed. Three intermediate designs had been completed by the time their paper was published, one of which had returned from fabrication. Although no performance figures were available, Simmons and Tavares hoped to obtain encryption times of 200μs. for 128 bit data. In subsequent work [103] [104] published in 1986, this group adopted Blakley's method [24] of reducing the partial products as they are formed. Reduction was again performed by monitoring the sign and magnitude of intermediate results and subtracting the modulus accordingly. By performing several additions in parallel, and using an asynchronous clocking scheme with self-timed adders to detect carry propagation, the group hoped to achieve encryption rates of up to 40 Kbits/s. for 512 bit encryption. At the CRYPTO '86 conference Orton et al. [105] reported the design of a 32 bit synchronous chip

fabricated in 3μ CMOS technology that ran at 200 KHz. with a throughput of 4 Kbits/s. A 22 bit version of the asynchronous chip was also reported to have been fabricated in 3μ CMOS, and test results from this design were extrapolated to estimate performance of 40 Kbits/s for 512 bit encryption for a 2μ CMOS chip running at 30 MHz.

In 1985 Martin Kochanski of Business Simulations Ltd., in England, announced the availability of a chip set based on a 32 bit bit-sliced processor, capable of performing 512 bit RSA at typical rates of around 5 Kbits/s at 5 MHz. [76] [77]. The chip was essentially a modular multiplier implemented on a 2.3 μ CMOS gate array and was cascadable, without additional circuity to 1023 bits. Although the design details were not published, it was a bit serial design similar to Brickell's performing *n* bit modular multiplication in *n* cycles plus some overhead. Kochanski claimed his design was less complex than Brickell's, requiring less circuitry to implement, and being less sensitive to details of implementation.

Another RSA system announced in 1985 was the security processor C.R.I.P.T. [106], designed by Jean Claude Pailles and Marc Girault from the French PTT research centre. The system was designed as a *removable unit* for use on more than one host and was implemented on two separate chips, a microcontroller, and an RSA chip that performed the basic modular multiplication. The RSA chip design is reported in more detail by Gallay and Depret [58]. This was 2μ CMOS design resulting in a 10.5mm x 8.4mm chip capable of performing 512 bit modular exponentiation in 320ms which corresponds to 1.6 Kbits/s. The architecture was based on a 256 bit machine coping with 512 and 1024 bit data in two and four clock cycles respectively. Modular multiplication was carried out MSB first according to Blakley's algorithm and reduction of the partial products done by examining the overflow resulting from the addition of the multiplicand to the running total. If overflow was detected, the modulus was subtracted from the running total. To reduce carry propagation times, the 256 bit adders were constructed from 16 x 16-bit adders that saved the carries allowing them to be dealt with on the next clock cycle.

In 1986 British Telecom and RAANND Systems announced the design of a single chip RSA processor capable of 512 bit exponentiation in 750 ms., or 680 bits/s. [113]. As was the case with Kochanski's design, the details were not published, although a general description was

given. The chip was a standard cell design measuring 16 mm per side, based on a 64 bit ALU architecture running at 20MHz. According to Rankine [113], there was a cubic relation for encryption times, which would suggest that carries were being allowed to propagate through the adders.

Another chip announced in 1986 was the CY1024 from CYLINK [1]. This was a 28 pin CMOS device that performed modular arithmetic on data up to 1028 bits, cascadable to 16384 bits, or 16 chips. The chip ran at clock rates up to 20MHz. and could perform 512 bit modular exponentiation in 80 ms. giving a throughput of 6.4 Kbits/s. Again, no details of this design were published.

3.2.2 Sedlak

A novel approach to modular multiplication was proposed by Holgar Sedlak in 1987 [126]. Sedlak suggested partitioning the multiplicand into runs of 1's and 0's and using look ahead algorithms to decrease the number of operations involved in the multiplication and the reduction of the partial products. A look ahead technique is also applied to the two addition operations required in Sedlak's design. By using 20 bit carry look ahead adders, Sedlak expects to be able to achieve 30 MHz. operation and a throughput of 200 Kbits/s for exponentiation with a 780 bit modulus. With the fourth Fermat number as the encryption key, rates of up to 3 Mbits/s are theoretically possible. These figures however, assume two 440 bit processors operating in parallel making use of the Chinese Remainder Theorem. From Sedlak's data, the encryption rate for a single 440 bit machine would be around 100 Kbits/s, extrapolating this to a 512 bit machine would imply an encryption rate of 94 Kbits/s.

3.2.3 Kawamura and Hirano

The popularity of digital signal processing (DSP) applications which employ residue arithmetic has motivated the development of table look up implementations of these algorithms [144] [146]. Cryptology algorithms however, tend to use much larger moduli than their DSP counterparts, and regularly require the modulus to be changed, thus making look up tables unattractive.

Realising that complete look up tables for modular arithmetic are not practical when applied to cryptology systems, Kawamura and Hirano [74] from Toshiba Corporation, suggested a method for the construction of a reduced look up table for modular reduction. In the modular multiplication scheme suggested in 1988 by Kawamura and Hirano, the full product is formed first. The result is then split into blocks *b* bits long, and the residues corresponding to the most significant block stored in a look up table. Reduction is performed by an iterative process where the most significant block is replaced by its residue until the result is less than the modulus. Kawamura and Hirano conclude by discussing the trade off then between block size and the number of iterations needed to complete reduction.

3.2.4 Cryptech

At the 1988 conference Eurocrypt '88, Frank Hoornaert et. al. from Cryptech and the University of Leuven presented the design of an RSA chip capable of 512 bit encryption at an impressive 17 Kbits/s. [68]. Modular multiplication is carried out by the usual method of reducing the partial products as they are formed. The reduction procedure is based on the idea of calculating the quotient by division, multiplying this by the modulus, and subtracting the result from the original number to obtain the modulus. Thus if R is the number to be reduced, then $q \leftarrow \lfloor R/n \rfloor$, and $R \leftarrow R - qn$. However, instead of performing a division, a sub-estimation of the quotient is carried out based on only a few bits of the number and the residue. Furthermore, by restricting the number of possible quotients, the multiplication qn may be implemented by a look up table. The estimate of q has to be accurate enough to avoid divergence.

This algorithm, illustrated in Figure 3.3, requires two additions per iteration, one for the multiply and one for the reduction. These are implemented out using carry save adders to prevent the addition being delayed by the length of the numbers.

Hoornaert et. al. reported on an ASIC design of a 120 bit datapath chip in 1.5μ CMOS. The chip could run at 14 MHz. and was cascadable to cope with larger moduli. A board designed with six of these chips and capable of handling moduli up to 712 bits was reported to have achieved a throughput of 17 Kbits/s. for 512 bit RSA. Rates of 512 Kbits/s. were achieved by using short exponents such as *F4*.

Cryptech subsequently announced several commercially available boards for personal computers capable of encryption rates of at least 12.6 Kbits/s. for 512 bit RSA [3]. Bong and Ruland [26] used one of these boards to achieve encryption times of 49 ms. or 10.5 Kbits/s.

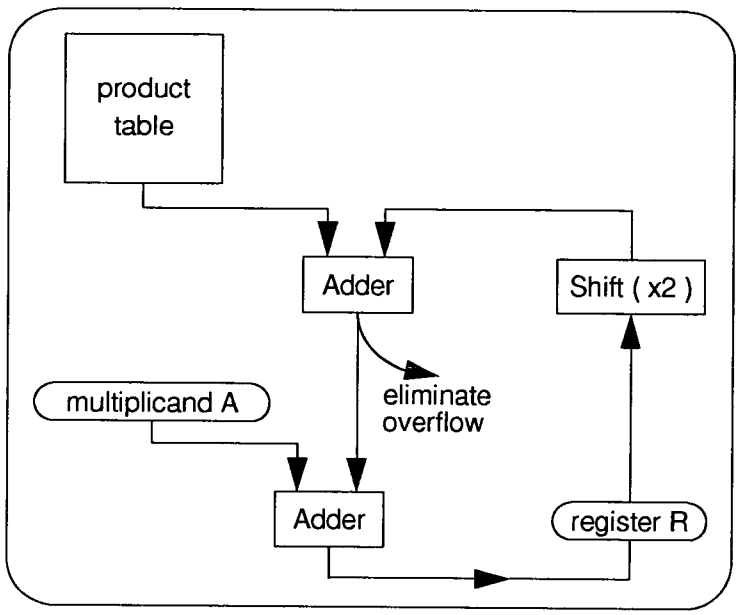

Figure 3.3 Cryptech's Multiplication Algorithm

3.2.5 Lu

Also in 1988, Lu Erl-Huei et al. [49] published the design of VLSI modular multiplier in 1988 that performed *n*-bit multiplication in *n* steps. This design reduced intermediate products by subtracting both the modulus, *N*, and 2*N*, from the intermediate result and using the overflow bits from these operations to determine which result should be selected. By performing these operations concurrently the algorithm could execute in *n* steps. The design of a four bit cascadable multiplier was presented that used carry look-ahead to speed up the additions. This design was estimated to run at 6 MHz. and consume a chip area of approximately 6mm x 6mm if implemented in 2.5μ CMOS.

3.2.6 British Telecom

The following year, Peter Ivey et. al. [69] from British Telecom published the design of a single chip RSA device implemented in 2μ CMOS which used a self timed methodology to speed up multiplication. The standard square and multiply algorithm was used for exponentiation and

modular reduction during multiplication was performed by comparing the intermediate result with the modulus and subtracting the modulus accordingly. The 256 bit data path used carry propagate adders with carry completion circuits placed at every fourth bit slice. The method of addition used allowed addition times of the order $n\log n$ for n bits, resulting in a performance of 5 Kbits/s for 256 bit RSA encryption. Later versions of the chip are expected to achieve 15 to 20 Kbits/s performance.

3.2.7 Hatfield

A novel approach to modular reduction was presented at the Crypto '89 conference by Paul Findlay and Brian Johnson [55] from Hatfield Polytechnic. Multiplication and modular reduction were treated as two separate tasks in this design with the output of the multiplier feeding the input of the reduction unit. Findlay and Johnson realized that the double precision multiplier output can be represented by the sum of residues of powers of 2 which can be easily stored in a look up table. For n-bit multiplication, a table of $2n$ residues, each n-bits wide would be required. Since such a table would not be practical for implementation in silicon Findlay and Johnson suggested a simple algorithm that can be used to calculate the residues. A bit-serial unit to carry out the reduction calculation was described which when used with a bit-serial multiplier combined to form a modular multiplication unit requiring $2n$ cycles to perform an n-bit modular multiplication.The advantage of this approach is that no bit testing or conditional branching is required and the hardware may be designed to have no broadcast bits, thus allowing faster clock rates.

3.2.8 Morita

Also at the Crypto '89 conference, Hikaru Morita from NTT presented the design of a higher radix modular multiplication algorithm [96]. Morita's algorithm is capable of performing n bit modular multiplication in $n/\log_2(r)$ clock cycles when the radix r is greater than or equal to four. The basic multiplication operation in this algorithm reduces partial products as they are formed by an estimation technique. Using the latest CMOS technology Morita expects to be able to design a 512 bit modular multiplier chip capable of performing RSA encryption at 80 Kbits/s with a 30 MHz. clock.

3.2.9 Shand

Mark Shand's work in 1990 [130] at DEC PRL on software implementations of RSA was discussed in section 3.1.4. To achieve higher rates Shand investigated the use of hardware accelerator boards, DEC's Programmable Active Memory or PAM arrays [20]. These accelerator boards are based on a 5 x 5 array of Xilinx 3020 programmable gate array chips with 512 K of RAM and a VME interface and allowed several different multiplier architectures to be tested.

With a single hardware accelerator unit rates of 1.8 Kbits/s were reported for 512 bit RSA encryption. Employing the Chinese Remainder Theorem and fine tuning the assembly code gave rates of 3.9 Kbits/s for a single unit. By adopting a modular multiplication technique that avoided comparisons with the modulus and using three hardware multiplier units encryption rates were increased further. Two of the multiplier units used Montgomery's recoding technique and worked modulo p and q, the factors of the RSA modulus. The third multiplier employed Booth recoding to combine the results from the first two to retrieve the desired product. With this arrangement the team at DEC PRL achieved the outstanding encryption rate of 200 Kbits/s for 512 bit RSA.

3.2.10 Recent Developments

Çetin K. Koç and Ching Yu Hung [32] have recently completed work on the design of systolic arrays for modular multiplication. This design performs modular multiplication by using Carry Save Adders and estimating the sign of the partial products to flag subtraction of the modulus. The proposed design takes $3n$ clock cycles to perform n bit modular multiplication but the systolic nature allows multiplications to pipelined thus producing a new result every clock cycle.

3.2.11 Summary

Figure 3.4 summarizes the hardware performance of RSA cryptosystems and may be compared with Figure 3.2 on page 56. The omissions from this table are due to encryption rates not being published in the original material.

	Year	Technology	Rate (bits/s)	Comments
Sandia	1979 1982 1982	Discrete NMOS	100 - 400 420 20 K	 336 bit modulus Estimated performance
Rivest	1979 1980	Discrete NMOS	2K 1.2 K	100 bit modulus, "proof of concept" Failed to Function
NTT	1982		50K	Estimated performance
Queens University	1983 1986 1986	6 µ NMOS 2 µ CMOS 3 µ CMOS	640 K 40 K 4 K	Estimated performance for 128 bit modulus Estimated performance Actual performance for 32 bit modulus
Kochanski	1985	2.3 µ CMOS	5 K	Gate array design Commercially available
C.R.I.P.T	1985	2 µ CMOS	1.6 K	
RAANND	1986		680	Standard Cell ASIC
Cylink	1986	1.5 µ CMOS	6.4 K	Commercially available
Sedlak	1987		94 K	Estimated performance
Cryptech	1988	1.5 µ CMOS	12 K - 17 K	ASIC design, Commercially available
British Telecom	1989	2 µ CMOS	5 K	256 bit modulus design for key management
Morita	1989		80 K	Radix 4 multiplier
DEC PRL	1990	PAM	1.8 K 200 K	One accelerator board Three accelerator boards and CRT

Figure 3.4 Summary of RSA Hardware Performance

Brief reviews of RSA hardware have been presented at the cryptology conferences EUROCRYPT '84 and CRYPTO '89 by Ronald Rivest [119] and Ernest Brickell [30].

3.3 Discussion

Since the conception of public key cryptography, the implementation of practical systems with data throughput comparable with secret key systems has proved both a technological and commercial challenge. Much research has been carried out into the design of algorithms and hardware accelerators to speed up modular exponentiation, and in particular, the core operation of this process, modular multiplication. While much has been accomplished in software, data throughput still remains at least an order of magnitude slower than for secret key systems. The need for a hardware solution to the problem of fast public key systems is reflected in the number of publications suggesting this approach.

With the exception of Laurichesse [79], little work has been done to investigate the exponentiation algorithm, the consensus of opinion being that the core operation of modular multiplication is where optimization will yield best results. In addressing this problem two different approaches have been taken. The earliest designs [28] [114] adopted the approach of reducing partial products as they were formed during multiplication as described by Blakley [24] in a later publication. The alternative is to allow the data to grow and reduce the double precision product on completion of the multiplication [55], [74], [142]. This allows standard hardware multipliers to be used and if a pipelined reduction unit can be designed such as was done at Hatfield [55], may produce acceptable results.

The most popular method of modular multiplication however, is to reduce the partial products as they are formed. This has the obvious advantage of restricting word growth and reducing hardware requirements.

Multiplication is generally carried out by examining the multiplier one bit at a time and conditionally adding the multiplicand to the partial product. Some [130] have used recoding techniques to examine more than one bit of the multiplier at a time but this has yet to be implemented in hardware for a large integer modular multiplier. If, as is the case in the majority of publications, reduction of the partial products is desired then the multiplier must be examined most significant bit (MSB) first as explained by Blakley [24]. Addition of the multiplicand to the partial product is often carried out using parallel adders that allow carries to propagate along the word length [49], [69], [113], [114], [117], [126]. Since the carry

propagation time is proportional to the number of bits in the modulus, this approach results in a quadratic relation for multiplication times, and a cubic relation for encryption times as reported by Rankine [113]. This relation may be improved upon by adopting the look-ahead techniques proposed by Sedlak [126] and Erl-Huei [49] although the area requirements of these techniques may inhibit extension to arbitrary length. The self timed adders used by Orton [105] and Ivey [69] also offer some improvement at the expense of increased complexity.

The use of carry save adders [28], [32], [55], [68], [76] and [106] on the other hand eliminates carry propagation completely and, provided care is taken in the reduction process, can produce quadratic relations for encryption times. The hardware overhead in this type of design is more than compensated for as security demands the use of encryption moduli up to 1024 bits and beyond.

Where most researchers differ is in the means of modular reduction. Many ([49], [69], [105], [106], [126]) use some form of magnitude comparison with the modulus followed by a subtraction. Although improvements may be made by performing concurrent subtractions of multiples of the modulus for post selection [49], [105], any magnitude comparisons inevitably involve carry propagation which when carried out at each step of the multiplication algorithm result in the undesirable cubic relationship between encryption times and modulus size. To avoid magnitude comparisons, estimation techniques may be applied [28], [92] which when coupled with a short look up table [68] give excellent performance.

If progress is to made in practical public key cryptography, then the way forward lies in designing efficient hardware to speed up the modular multiplication operation so crucial to many public key algorithms. As pointed out by Hoornaert et. al. [68], "hardware knowledge has to be integrated into the algorithmic study to obtain the optimal calculation scheme". One specific area where this integration has to occur is in the modular reduction operation, where algorithms and hardware are required to perform modular reduction without incurring any carry propagation. The means by which this is accomplished is the subject of the following chapter.

Chapter 4 Algorithm Design

Chapter 3 identified modular multiplication as a key area for optimization to enhance the performance of public key cryptosystems. This chapter describes a novel modular multiplication algorithm suitable for a hardware design which avoids any carry propagation, thus allowing n bit modular exponentiation to be performed in $O(n^2)$ time.

As discussed in the previous chapter, although algorithms may be designed to perform modular exponentiation in $O(n^2)$ *clock cycles*, if hardware limitations are ignored, these same algorithms will require $O(n^3)$ *time* for execution. This distinction is crucial for the large moduli needed in public key cryptosystems

To achieve performance in $O(n^2)$ time, carry propagation along the word length must be avoided completely. This effectively precludes any magnitude comparisons, so the problem is then, how to perform modular reduction without involving magnitude comparisons. One solution is to estimate the residues as was successfully demonstrated in the Cryptech chip [68]. The alternative proposed in this thesis is to perform modular multiplication without any modulo reduction steps.

4.1 Modular Multiplication with Partial Reduction

Multiplication is carried out MSB first according to the add - shift - reduce procedure described by Blakley [24], but with the following modifications.

1. The partial product is allowed to grow by two bits each iteration.
2. At the end of each iteration, these upper bits are reset to zero.
3. The residue corresponding to the two reset bits is added to the partial product on the next iteration.

That the reduction of the partial products may be incomplete, in that after resetting the upper bits the remaining number may be greater than the modulus, is of little practical consequence. What is important is that the result is constrained to the length of the multiplier array.

On completion of this procedure reduction may be carried out, if desired, by subtracting the modulus, but since the word length has been constrained to two bits of growth only a few subtractions of the modulus will be needed to accomplish this. Furthermore, as the result of

the multiplication is only one step in the much longer procedure of exponentiation, this reduction step need typically only be performed once in every 512 multiplications.

Figure 4.1 illustrates the algorithm in the style of Hoornaert et. al. [68], and may be compared with Figure 3.3 on page 62. Figure 4.2 illustrates the algorithm using pseudo code.

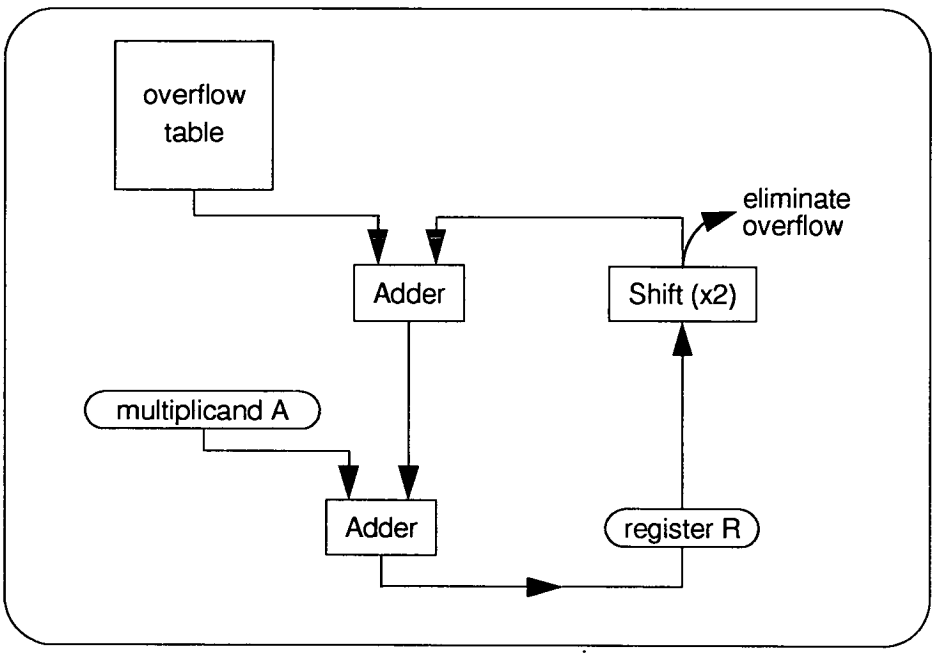

Figure 4.1 Illustration of Multiplication Algorithm

Although the size of the product table in [68] is not stated, it is unlikely to require less storage than the three residues the overflow table in the above needs to represent the two upper bits that have been reset.

Figure 4.3 demonstrates the operation of the algorithm by showing how 14 * 15 mod 17 (= 6) is calculated. First of all the residues have to be calculated for the modulus 10001, since the word is allowed to grow by two bits the residues of 01 00000, 10 00000, and 11 00000 are required. However, as the residues are added on the following cycle, they are effectively left-shifted in significance thus it is the residues of 010 00000, 100 00000, 110 00000 that are needed. These are 01101, 01001, and 00101 respectively.

```
program modmult;

const wordlength = 5,    (* number of bits in the machine *)
     modulus = 17, res1 = 13, res1 = 9, res1 = 5;

type vector = array[0..wordlength-1],
     long = array[0..wordlength+2],
     bit = (0,1);

var a, b: integer,       (* input multiplier & multiplicand *)
     res : integer,      (* residue corresponding to overflow bits *)
     acc : integer,      (* accumulator *)
     i : integer,        (* counter *)
     v0, v1: bit,        (* overflow bits *)
     bvec: vector,       (* binary equivalent of input b *)
     accvec: long;       (* binary equivalent of accumulator *)

BEGIN (* modmult *)
     i := wordlength;
     res := 0;
     read( a, b );
     bvec := Convert_to_Vector( b );
REPEAT
     IF bvec[i-1] = 1 THEN acc := acc + a;
     acc := acc + res;
     IF i < 1 THEN acc := acc * 2;
     accvec := Convert_to_Long( acc );
     ( v0, v1 ) := Most_Significant_Two_Bits_of( accvec );(*v0=MSB*)
     res := CASE ( v0, v1 ) OF
                ( 0, 0 ): 0;
                ( 0, 1 ): res1;
                ( 1, 0 ): res2;
                ( 1, 1 ): res3;
     END; (* Case ( v0, v1 ) *)
     Reset_Most_Significant_Two_Bits_of( accvec );
     acc := Convert_to_Int( accvec );
     i := i - 1;
UNTIL i = 0;
WHILE acc < modulus DO acc := acc - modulus;
writeln( a, ' * ', b, ' mod ', c, ' = ', acc )
END. (* modmult *)
```

Figure 4.2 Pseudo Code for Multiplication Algorithm

Example: 14 * 15 mod 17 = 6

Residues are: 01 = 13
10 = 9
11 = 5

Iteration (i)	b[i]	Residue	PP_2	PP_{10}	Comment
5	0	0	00 00000	00	
4	1	0	00 01110	14	add 14
			000 1110	28	shift
			0 1110	28	reset
3	1	0	01 01010	42	add 14
			010 1010	84	shift
			0 1010	20	reset
2	1	13	01 00010	34	add 14
			01 01111	47	add residue
			010 1111	94	shift
			0 1111	30	reset
1	1	13	01 01100	44	add 14
			01 11001	57	add residue
					no shift
				40	-17
				23	-17
				06	

Figure 4.3 Example of Algorithm

In the above, b[i] is the i^{th} bit of the multiplier (15, 01111) and determines whether the multiplicand (14) is added or not. PP is the partial product. As explained in [24], no shift is carried out on the last step of the multiplication procedure.

Figure 4.4 shows how this algorithm may be implemented in hardware with "A" representing the multiplicand, and "R" representing the residue to be added. Both these numbers are input

in parallel, A_4 and R_4 being the MSBs. "A" will of course be masked with each bit b[i] of the multiplier, and "R" selected by the two overflow bits.

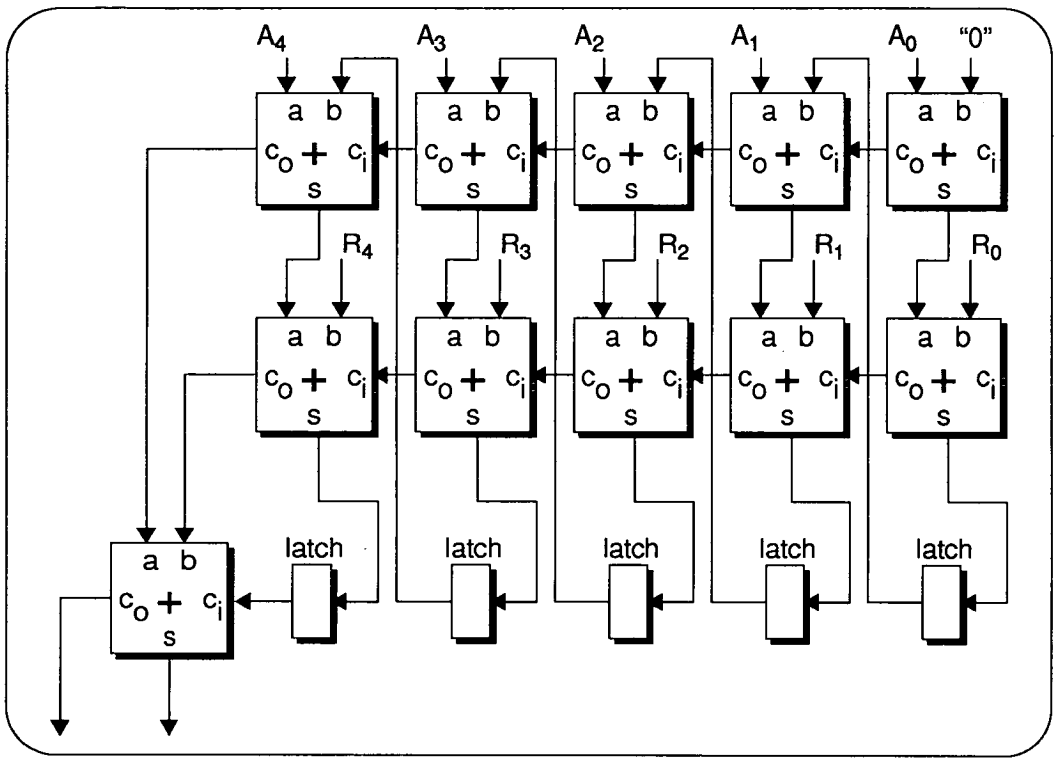

Figure 4.4 Hardware Implementation of Algorithm

Having demonstrated that ***complete*** modular reduction is not necessary to restrict word growth, the ***partial reduction*** technique may now be applied to bit serial designs to eliminate carry propagation in the above.

4.2 Bit Serial Design

Figure 4.5 shows how the basic cell of Figure 4.4 is modified for bit serial implementation. In Figure 4.6 each bit serial cell is represented by a block to illustrate how the five bit multiplier array is constructed.

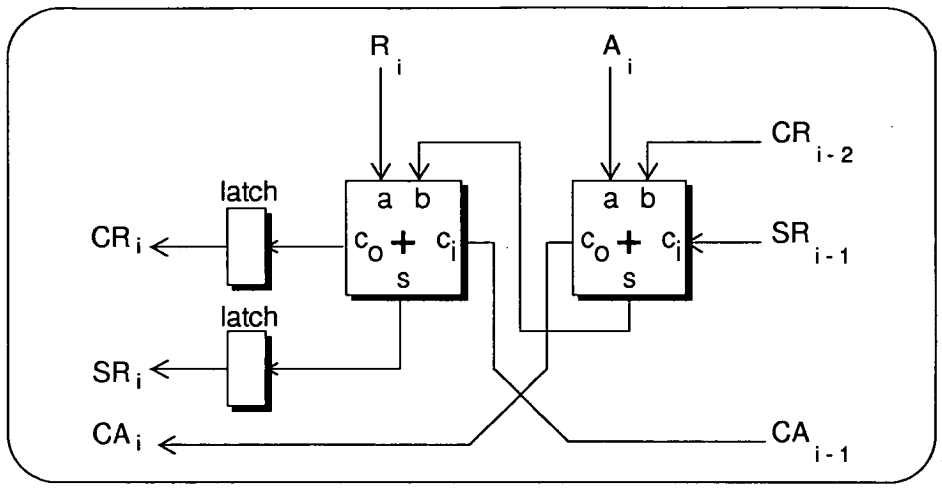

Figure 4.5 Bit Serial Modular Multiplier Cell

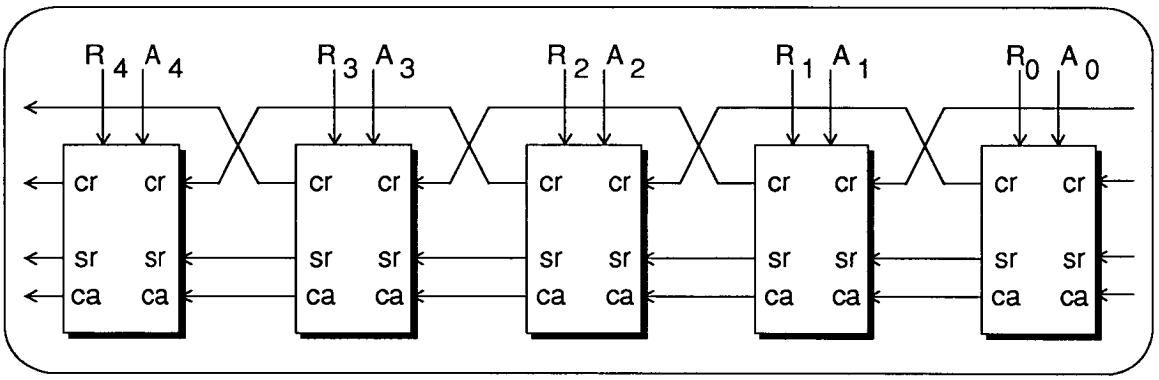

Figure 4.6 Five Bit Multiplier Array

The four inputs on the right of the array are all logical zeroes, the four outputs at the left extend the array to larger bit lengths. The result of the multiplication is held in sum and carry form in the two registers formed by the SR and CR latches.

To complete the circuit, the overflow bits representing the data lost off the left hand side of the array must be resolved by adding all bits with the same significance, as shown in Figure 4.7.

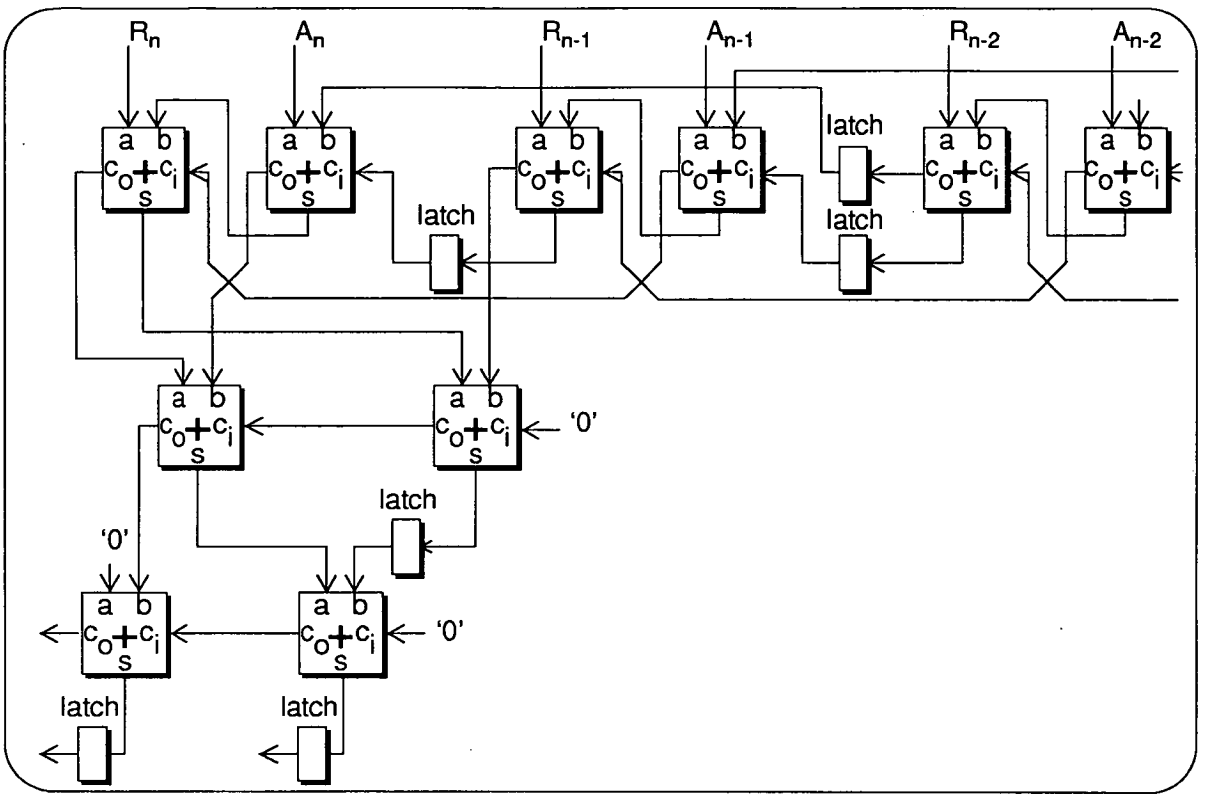

Figure 4.7 Resolution of Overflow Bits

In Figure 4.7 the maximum possible overflow appears to be '100', not '11' as suggested by the carry propagate hardware. However, for *n* bit unsigned operands, the largest possible number resulting from the addition of the four *n* bit numbers A, R, SR, and CR, is

$$4\,(2^n - 1) \qquad \text{(EQ 4.1)}$$

But since the accumulator is only *n* bits long, it can only hold

$$4\,(2^n - 1) \bmod 2^n \qquad \text{(EQ 4.2)}$$

But

$$4\,(2^n - 1) \bmod 2^n \;=\; (3 \cdot 2^n + 2^n - 4) \bmod 2^n \qquad \text{(EQ 4.3)}$$

And since $(2^n - 4) \bmod 2^n$ can be held in the *n* bit accumulator, the worst case overflow is:

$$(3 \cdot 2^n) \bmod 2^n \;=\; 011\ldots \qquad \text{(EQ 4.4)}$$

Thus the carry output from the last adder is never set.

The worked example of Figure 4.3 is repeated in Figure 4.8 to demonstrate the operation of the bit serial algorithm.The data shows the inputs and outputs for each adder in the array and the result of the overflow calculation. This is done for five cycles, then the sums and carries are added to produce the result.

Example: 14 * 15 mod 17 = 6

Residues are: 001 = 13 011 = 5
010 = 9 100 = 1

iteration (j)	b[i]	First Adder		Second Adder		Overflow
5	0	A	00000	R	00000	
		SR_{i-1}	00000	SA_i	00000	
		CR_{i-2}	00000	CA_{i-1}	00000	
		CA	00000	CR	00000	
		SA	00000	SR	00000	000
4	1	A	01110	R	00000	
		SR_{i-1}	00000	SA_i	01110	
		CR_{i-2}	00000	CA_{i-1}	00000	
		CA	00000	CR	00000	
		SA	01110	SR	01110	000
3	1	A	01110	R	00000	
		SR_{i-1}	11100	SA_i	10010	
		CR_{i-2}	00000	CA_{i-1}	11000	
		CA	01100	CR	10000	
		SA	10010	SR	01010	010
2	1	A	01110	R	01101	
		SR_{i-1}	10100	SA_i	11010	
		CR_{i-2}	00000	CA_{i-1}	01000	
		CA	00100	CR	01000	
		SA	11010	SR	11111	010
1	1	A	01110	R	01101	
		SR_{i-1}	11110	SA_i	10000	
		CR_{i-2}	00000	CA_{i-1}	11100	
		CA	01110	CR	11100	
		SA	10000	SR	00001	

CR + SR = 56 + 1 = 57 = 6

Figure 4.8 Example of Bit Serial Algorithm

4.3 Testing the Algorithm

A program was written in PASCAL to test the algorithm by comparing its results with those obtained by standard computer arithmetic and exiting if there was a difference. This program exhaustively tested all combinations from 0 * 0 mod 1, to 277 * 277 mod 278, a total of over seven million multiplications, without error.

The test program itself was verified by introducing a deliberate bug into the algorithm which led to an erroneous result, halting the program execution.

4.4 Performance Estimates

The throughput for RSA encryption using this multiplication algorithm may now be estimated.

It is assumed that the square and multiply algorithm for exponentiation is to be used without modification, and that two hardware multipliers are used, one for squaring the other for multiplying. An n bit exponentiation will then take n multiplications. The effect of using only one hardware multiplier is data dependent, exponentiation taking between n and $2n$ multiplications.

It is further assumed that the results of each multiplication remain in sum and carry form until the exponentiation has ended, thus the reduction step need only be carried out once in every n multiplies and may effectively be ignored. Each multiplication may then be carried out in n clock cycles, and exponentiation in n^2 cycles. In hardware, the final reduction step may be carried out on a separate carry propagate adder allowing the subsequent exponentiations to proceed immediately.

Examination of Figure 4.7 shows the longest path that signals have to propagate through to be via six adders. If δ_a is the maximum delay through a one bit adder, then maximum frequency the array can operate at is

$$f = \frac{1}{6\delta_a} \qquad \text{(EQ 4.5)}$$

And since an n bit exponentiation takes n^2 cycles the throughput is

$$\frac{1}{n \cdot 6\delta_a} \qquad \text{(EQ 4.6)}$$

Figure 4.9 tabulates the data throughput in Kbits/s based on (EQ 4.6) and may be compared with Figure 3.4 on page 65. Even assuming the worst case operating frequency, these figures compare favourably with the best designs summarised in Figure 3.4.

δ_a / ns. / n	3	5	10	15
256	217	130	65	43
512	109	65	33	22
1024	54	32	16	11
Clock / MHz.	55.6	33.3	16.7	11.1

Figure 4.9 Performance Estimates

Chapter 5 Chip Design

This chapter describes the design of a chip capable of performing most of the modular arithmetic operations commonly used in public key cryptography. The core element of the design is the modular multiplier described in Chapter 4.

5.1 Design Criteria

The intention of this research, as stated in section 1.4, is to design fast hardware for public key cryptography. To achieve this goal it was decided at the outset of the project to aim for a design that did not propagate any carries when performing modular multiplication. This decision precluded the use of any magnitude comparisons during modular reduction and led to the partial reduction technique described in Chapter 4.

A second constraint on the hardware design is that it should not be restricted to a specific key length. One advantage that the RSA scheme has over block ciphers, like DES or FEAL, is that the security of the system may be increased at any time, simply by using longer keys. If the hardware is restricted to a fixed key length, then this advantage is lost and the chip will soon become obsolete. To provide a facility for variable key length the processor must either multiplex in time, or be cascadable. As the first option detracts from the prime aim of the research, a cascadable design was chosen, thus providing the option to increase security with no loss of performance.

A third design consideration was testability. Here the basic requirements of good VLSI design and good cryptosystem design diverge. In the former it is desirable to have easy read and write access to as many internal nodes in the circuit as possible, whereas in the latter such an attribute would seriously compromise security. Since this was envisaged as a prototype design only, the former approach was taken. All registers on the chip being accessible either directly from the data ports or via scan paths. The problem of ensuring testability in commercial designs without compromising security is not addressed in this thesis. One possible solution may be to use fusable links, as is done in PROM design, to disable memory read functions after final production tests.

Another practical consideration was to keep the pin count low to avoid packaging the finished device in a pin grid array which would complicate any future board design.

5.2 Chip Architecture

Having established a rough specification for the chip and a basic multiplier design, the support circuitry for the multiplier, and the interface circuits were defined.

Output from the multiplier array is a number in sum and carry form that may be used in subsequent operations on chip with complete reduction being carried out externally if required. However, this may not always be convenient and so a means of completely reducing the number internally is desirable. This is accomplished by including a ripple adder, an accumulator, and a register for the modulus on the chip as indicated in Figure 5.1. Inclusion of these elements allows reduction to be completed while the next product is being formed in the array, and provides facilities for simple modular addition and subtraction. The clock for this part of the circuit obviously has to run at a slower rate than the clock for the array.

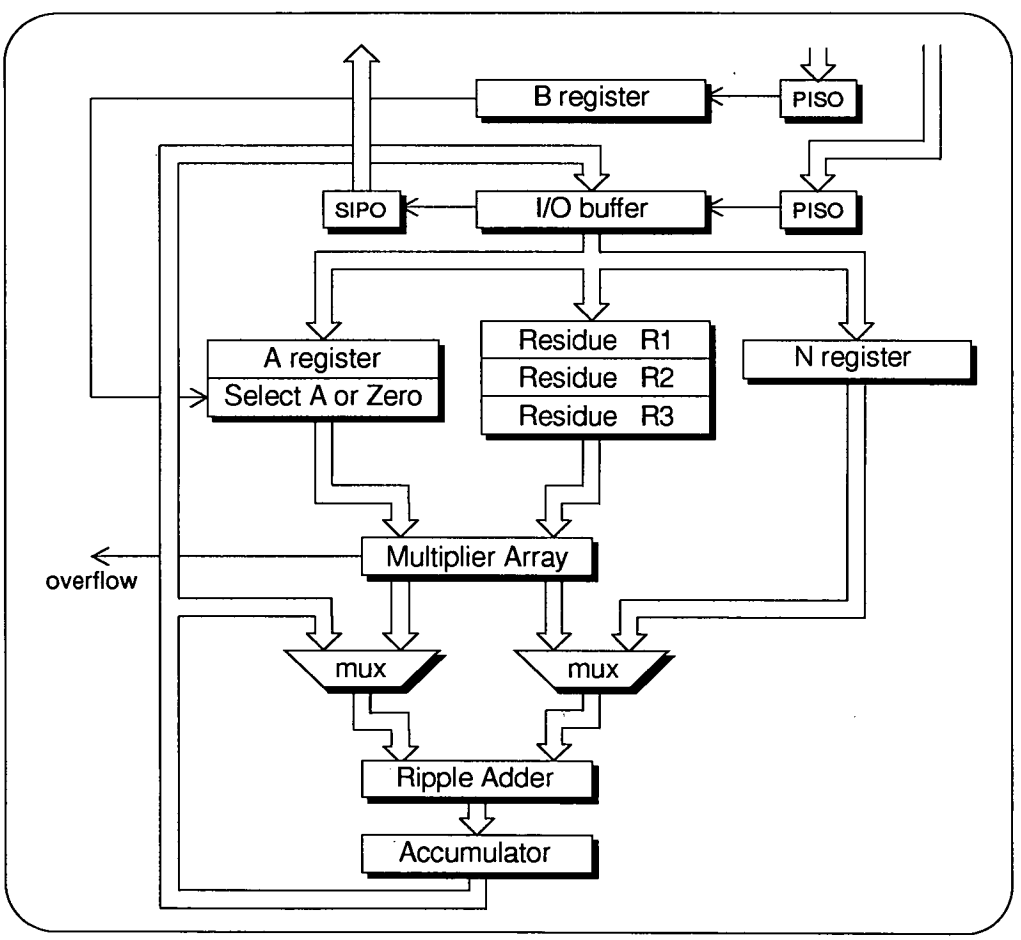

Figure 5.1 Modular Arithmetic Chip Architecture

The Input/Output (I/O) buffer may be loaded and read either in parallel or serially, permitting simultaneous reading and writing of data while multiplication is being carried out on the array and reduction on the adder and accumulator. Eight-bit parallel to serial (PISO) and serial to parallel (SIPO) converters at either end of this buffer permit byte oriented input and output with slower strobes than those required for the serial links. Two data input ports allow the multiplicand "A", and the multiplier "B", to be loaded simultaneously. The modulus and residues will not change as frequently as "A" and "B" and therefore share the "A" input port.

Figure 5.2 indicates how this architecture allows modular multiplication to be pipelined for maximum throughput.

Operation	Data Block			
Load Data	i	i+1	i+2	i+3
Multiply	i-1	i	i+1	i+2
Reduce	i-2	i-1	i	i+1
Read Data	i-3	i-2	i-1	i

Figure 5.2 Pipelined Modular Multiplication.

5.3 Logic Design

At this stage in the design the behaviour of the chip was modelled using the hardware description language ELLA to facilitate testing and debugging of the proposed architecture without describing the low level details of the hardware.

5.3.1 Description of Single Chip

After defining basic data types and cells such as single bit adders and latches, the multiplier cell of Figure 4.5 was described as shown in Figure 5.3

```
FN MULTCELL = (bool: ai ri caim1 crim2 srim1 clock rbar enable )
                                                    -> [3]bool:
  BEGIN
    MAKE    BITADD : add1 add2,
            FULATCH { 1 } : latch1 latch2.
    JOIN    ( ai, srim1, crim2 ) -> add1,
            ( ri, add1[1], caim1 ) -> add2,
            ( REF1( add2[1] ), rbar, enable, clock ) -> latch1,
            ( REF1( add2[2] ), rbar, enable, clock ) -> latch2.
    OUTPUT REF3( REF1( add1[2] ), latch2, latch1 )
            # CA(i), CR(i), SR(i) #
  END.
```

Figure 5.3 ELLA Description of Multiplier Cell

This cell was then replicated to define the parameterisable multiplier array of Figure 5.4.

```
MAC ARRAY { INT n } = ( [n]bool: ain rin, [4]bool: sigin, bool: clock
            rbar enable ) -> ( [4]bool, [3]bool, [n+3]bool, [n]bool):
BEGIN
  MAKE [n-2]MULTCELL : mcell,
       NMI1CELL : ncell,
       ENDCELL : ecell.
  LET
    ca = sigin[1] CONC ( [INT k=1..n-2] mcell[k][1]
           CONC ( REF2A ( ncell[1][1], ecell[1][1] ) ) ),
    cr = sigin[4] CONC ( sigin[2] CONC ( [INT k=1..n-2] mcell[k][2]
           CONC ( REF2A ( ncell[1][2], ecell[1][2] ) ) ) ),
    sr = sigin[3] CONC ( [INT k=1..n-2] mcell[k][3]
           CONC ( REF2A ( ncell[1][3], ecell[1][3] ) ) ),
    sigout = ecell[1] CONC ncell[1][2],
    addout = ecell[2][2..4],
    sums = ( sr[2..n] CONC ecell[2] ),
    cars = ( cr[2..n] CONC f ).
  FOR INT k=1..n-2
    JOIN ( ain[k], rin[k], ca[k], cr[k], sr[k],clock, rbar,
                                          enable)-> mcell[k].
  JOIN ( ain[n-1], rin[n-1], ca[n-1], cr[n-1], sr[n-1], clock,
                                    rbar, enable ) -> ncell,
    ( ain[ n ], rin[ n ], ca[ n ], cr[ n ], sr[ n ], ncell[2],
                                    clock, rbar, enable ) -> ecell.
  OUTPUT ( sigout, addout, sums, cars )
END.
```

Figure 5.4 ELLA Description of n-bit Multiplier Array

Once the multiplier array and modular reduction circuits had been defined, an interface was added which included serial links not shown on Figure 5.1, to allow cascading of several chips. Finally, a mnemonic instruction decoder was described and the design tested and debugged.

5.3.2 Description of Cascaded Chips

This single chip model was then used to describe and simulate a complete cascaded system of three eight bit chips.

```
FN SYST = ( instruc: mem mul, [2]hex: data, [4]bool: ext_addr,
            bool: s_buff s_b ldb_b ldb_ps ldb_cl ldb_c ldb_cm
            sel_addr a_clk r_clk ) -> ( [2]hex, bool ):
BEGIN
   MAKE     CHIP { 8 } : lschip midchip mschip,
            MUX2 { [4]bool } : addr.
   LET      int_addr = ( BITREV { 3 } mschip[2] ) CONC f,
            b_ctl = mschip[4],
            sig1 = lschip[1], sig2 = midchip[1],
            s_bf1 = lschip[3], s_bf2 = midchip[3],
            s_b1 = lschip[4], s_b2 = midchip[4].
   JOIN     ( int_addr, ext_addr, sel_addr ) -> addr,
            ( mem, mul, data, addr, s_buff, s_b,
             b_ctl, ldb_ps, ldb_b, ldb_cl, a_clk, r_clk ) -> lschip,
            ( mem, mul, sig1, addr, s_bf1, s_b1,
             b_ctl, t , t , ldb_c , a_clk, r_clk ) -> midchip,
            ( mem, mul, sig2, addr, s_bf2, s_b2,
             b_ctl, ldb_ps, t , ldb_cm, a_clk, r_clk ) -> mschip.
   OUTPUT ( mschip[1], mschip[5] )
END.
```

Figure 5.5 ELLA Description of Cascaded System

In Figure 5.5 three chips are instanced, each one having an eight bit wide modular multiplier, and are combined to form a 24 bit multiplier. Each chip has a 'mem' and 'mul' input, these are mnemonic instruction inputs to control memory and multiplier operations. The least significant chip has a hexadecimal 'data' input port that connects to both the B register port and the I/O Buffer of Figure 5.1. The 'addr' input accepts the address of the residue and may come from either the overflow bits of the most significant chip during normal operation, or from an external source to allow residues to be accessed directly. The remaining inputs control the loading and reading of the PISOs and SIPO, selection of serial/parallel input and output, and masking of

the 'A' register during multiplication. A second external clock provides a means of running the ripple adder at a slower rate than the multiplier array. As this was a prototype design, this was thought to be safer than generating the slow clock internally.

This cascaded system was tested and debugged until the multiplication results agreed with those calculated by the PASCAL program of section 4.3, and all other features of the chip behaved as required. Once satisfied with the logical design, the physical design could proceed with the knowledge that functionality is correct. This allowed attention to be concentrated on implementation issues such as timing, buffering and clock distribution.

5.4 Physical Design

It was known in advance that the chip was to be fabricated using ES2's ASIC design tools, SOLO 1200, so the final ELLA description was constructed in a manner that would easily map on to the basic gates available in the SOLO library. A rough draft of the design was produced using the SOLO schematic capture tools to describe a parameterisable design that could generate a multiplier chip of user defined word length. From this schematic description, circuit layout is synthesised very quickly and it was soon clear from the resulting silicon area, that the target design should be a 32-bit processor. A full custom approach may have produced more efficient layout resulting in a 64 or 128 bit processor but the time and effort required to do so was considered too much for a prototype design. In adopting this semi-custom approach, control over floorplanning and routing is forfeited and so the most efficient layout is not realised, consequently each component has to be heavily buffered leading to even greater chip area. Furthermore, dynamic logic was unavailable and all storage had to be constructed from fully static d-type latches.

Since routing is done automatically, following the design hierarchy, it was decided to adopt a bit slice approach to this design in an attempt to minimise the wire lengths needed for local routing and avoid the difficulties involved in routing 32 bit busses.

5.4.1 Basic Cell Designs

The basic latch used throughout this design was the edge triggered d-type flip flop arranged to form the enable type latch of Figure 5.6. Here, the enable input 'en' determines whether to gate

new data 'd' into the latch or to hold previous data. This type of latch was chosen to avoid having to gate the clock, thus reducing the possibility of race conditions that may arise due to the lack of control over floorplanning and circuit delays resulting from the automatic design layout. A synchronous reset 'r' is also included in this design.

Output from the latch is via a parameterisable buffer to simplify the matching of drive capability to circuit loads in the complete design.

The latch in Figure 5.6 is a scan path latch used in preference to ordinary d-types to improve testability of the design in areas where register access was not straightforward.

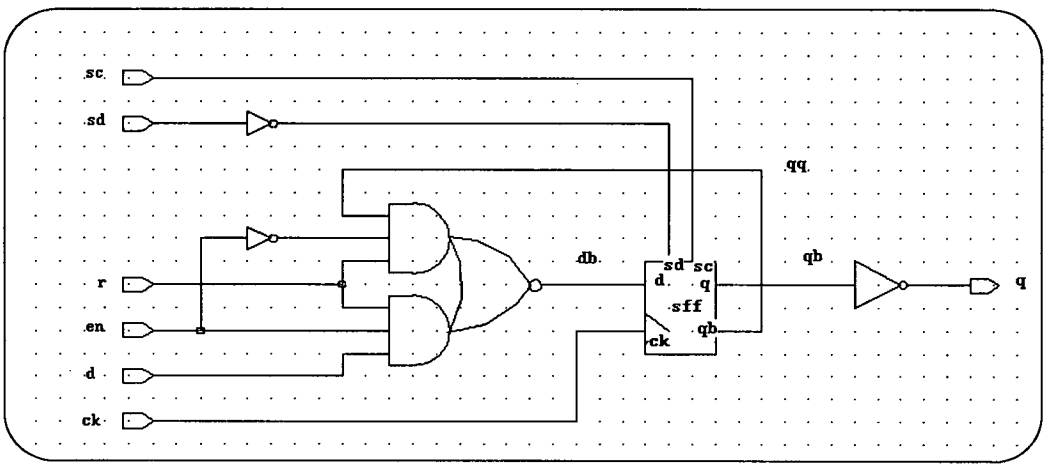

Figure 5.6 Enable-type Latch

The circuit diagrams shown in this chapter are screen dumps taken from the SOLO schematic capture tool DRAFT, used to create the design.

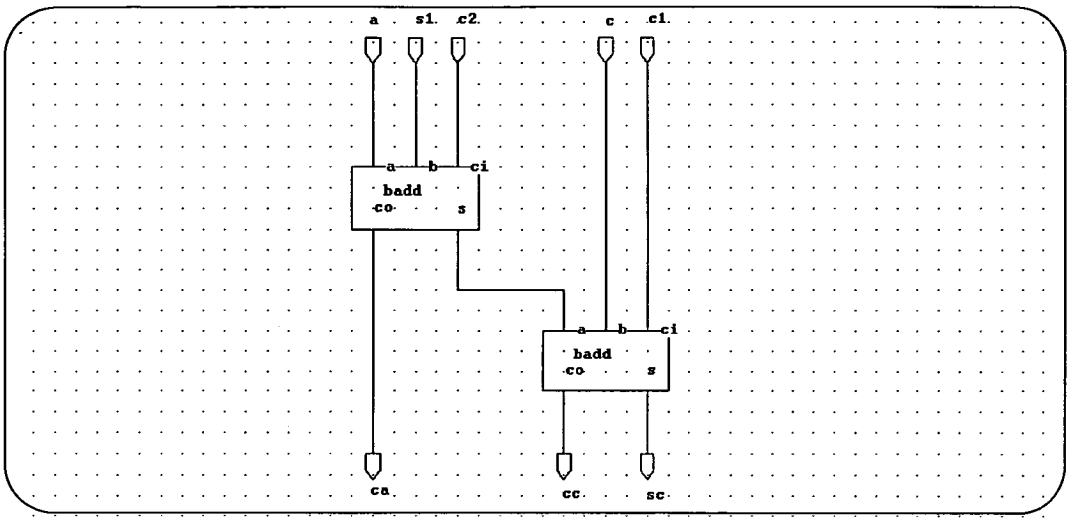

Figure 5.7 Adder for Multiplier Array: 'madd'

The 'badd' cells in Figure 5.7 are based on the standard single bit binary add cell available in the SOLO library, to which parameterisable buffers have been added at the output. These two adders form the multiplier add cell 'madd' used to build the multiplier cell 'mcell' Figure 5.8.

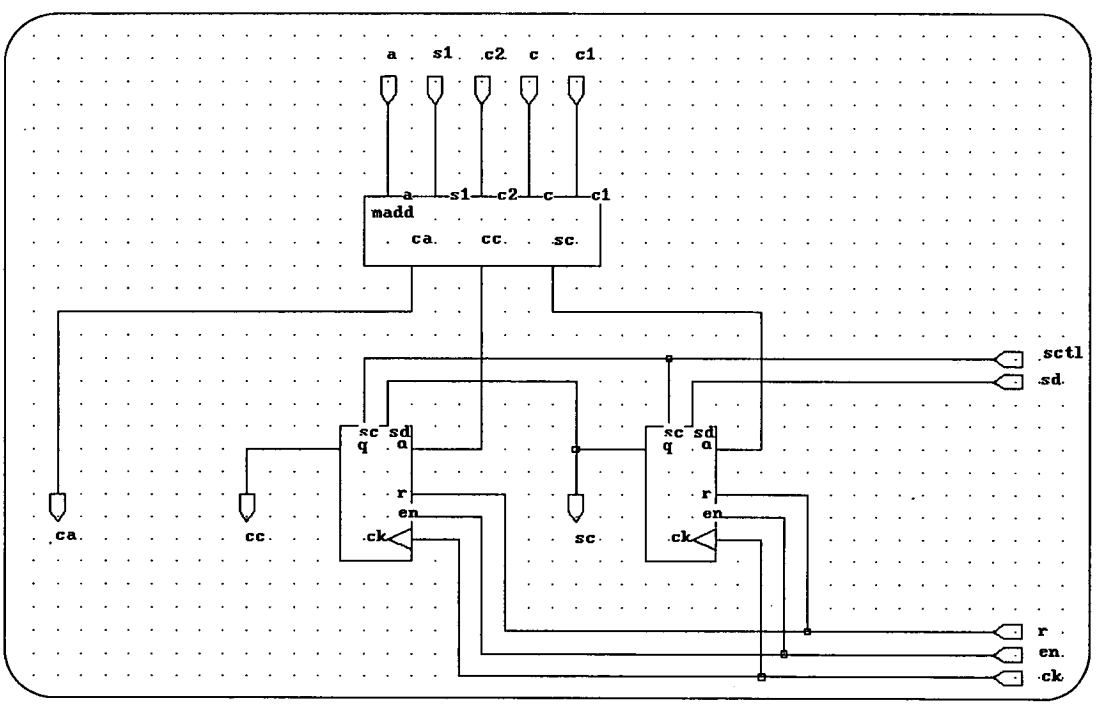

Figure 5.8 Multiplier Array Cell: 'mcell'

Figure 5.8 is the final version of the cell originally proposed on page 73.

Memory for the residues was constructed from enable type latches, a decoder with write enable, and a multiplexer as shown in Figure 5.9. No user defined RAM was available.

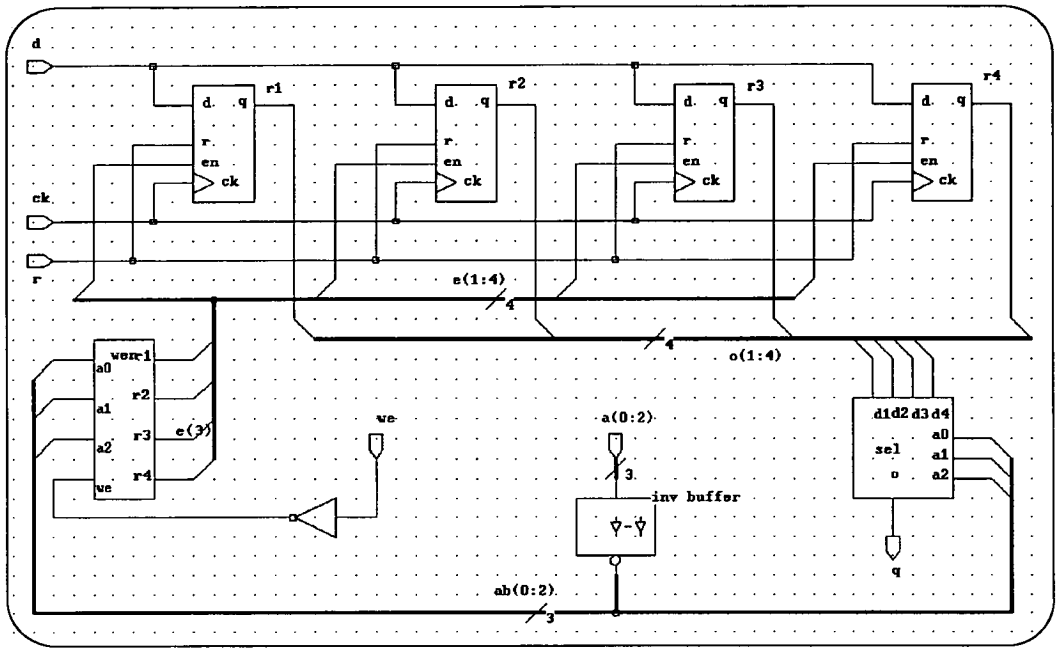

Figure 5.9 Storage for Residues: 'cram'

The residue, or congruence RAM, 'cram', is combined with the multiplier cell in Figure 5.10 together with storage for the modulus 'nreg' and the multiplicand 'areg'. The output from 'areg' is masked with each bit of the multiplier 'bb' before being input to the multiplier cell. Separate reset lines allow either the memory, or the array to be reset at the user's discretion.

This combined cell 'ccell' is connected to the ripple adder and accumulator 'addr' in Figure 5.11 to form the basic bit slice cell. Like 'mcell' and 'cram' in 'ccell' (Figure 5.10) 'addr' also has separate reset and enable lines, but a separate external clock is required for this part of the chip as explained in section 5.2. Figure 5.11 is a bit slice of the chip architecture shown in Figure 5.1 with the slight modification of the multiplexer used to route either the residues or the modulus (n) to the ripple adder. This allows the partial reduction technique to be applied to the final reduction operation as well as multiplication, and reduces the number of times the modulus has to be subtracted to achieve complete reduction. The other multiplexer controlled by 'ser' in Figure 5.11 is constructed from the two multiplexers of Figure 5.1 determining whether the ripple adder adds sums and carries from the array, or is used for reduction.

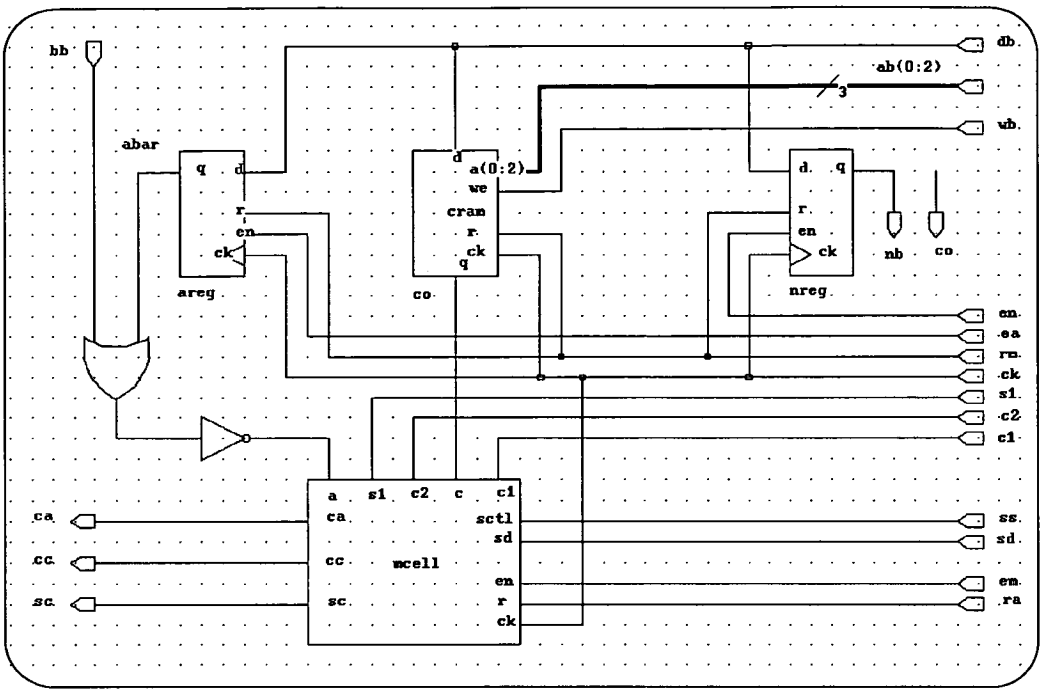

Figure 5.10 Combining Storage and Multiplier Cells: 'ccell'

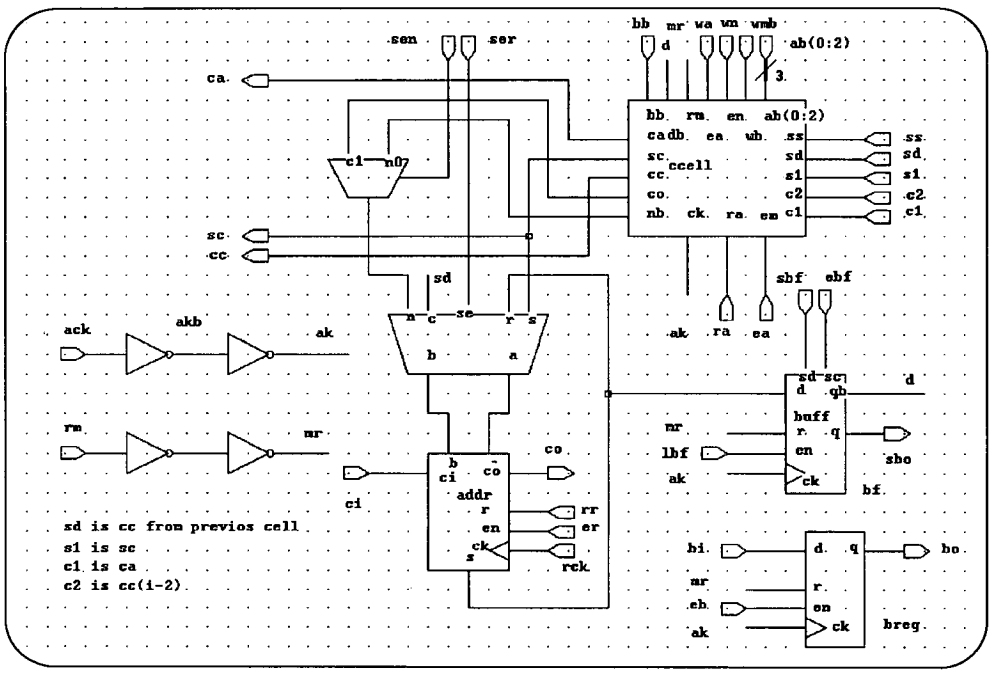

Figure 5.11 Complete Bit Slice Cell: 'cell'

The multiplicand register 'breg' is serially linked to higher and lower order bits and reset by the same signal as 'ccell'. The I/O buffer is constructed from scan path type latches to allow both serial and parallel data input and output.

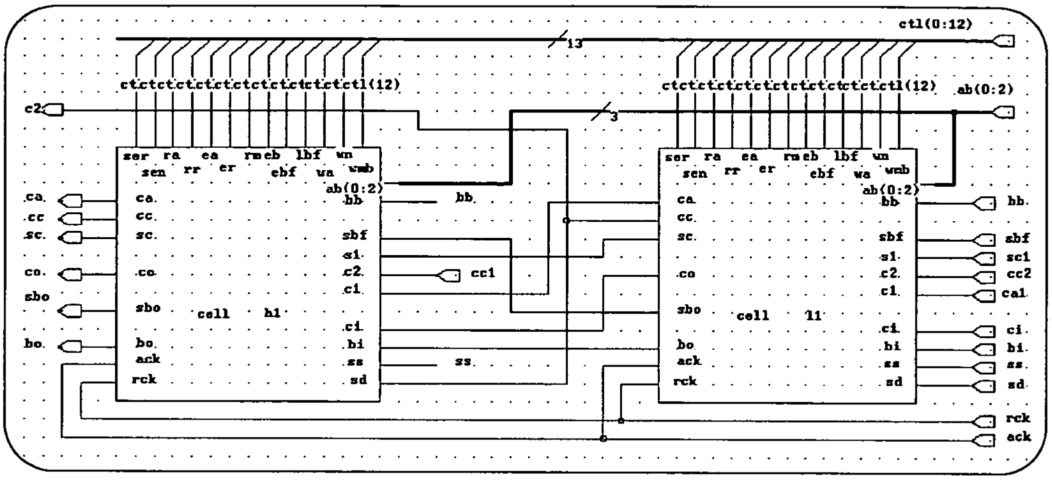

Figure 5.12 Combination of Two Single Cells: 'cell2'

Figure 5.12 shows how two single cells are combined, and Figure 5.13 shows how two 'cell2s' are combined to form a 'cell4'. The complete chip is built up in this hierarchical manner to ease buffering and clock distribution.

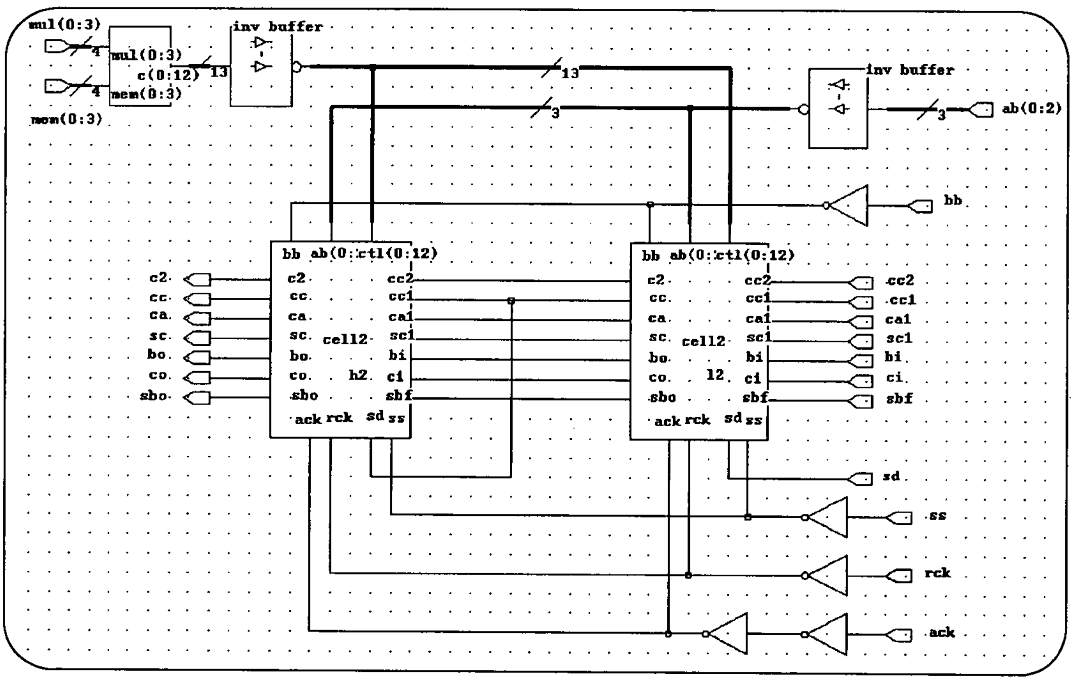

Figure 5.13 Combination of Two cell2's: 'cell4'

For flexibility, it was decided to allow separate commands for memory and multiplier operations and, from the ELLA simulations, it was known that a minimum of nine commands were needed for both operations. The command decoder therefore required two four bit inputs, and a thirteen bit output as shown in Figure 5.13. Decoding was carried out at this level to minimise the routing of the thirteen control signals without allocating too much silicon to the decoders.

5.4.2 End Cell Designs

The cells at the most significant end of the array differ slightly from the rest of the array as described in the following.

In Figure 4.7 on page 74, both carry outputs and the sum output from the cell at the end of the array are connected to the input of the two bit ripple adder used to calculate the overflow. To allow this, the multiplier cell of Figure 5.8 is modified by adding the unlatched carry output signal 'ccd' and the unlatched sum output signal 'scd'. The 'ca' output from this cell is connected directly to an output pin, hence the heavy buffering.

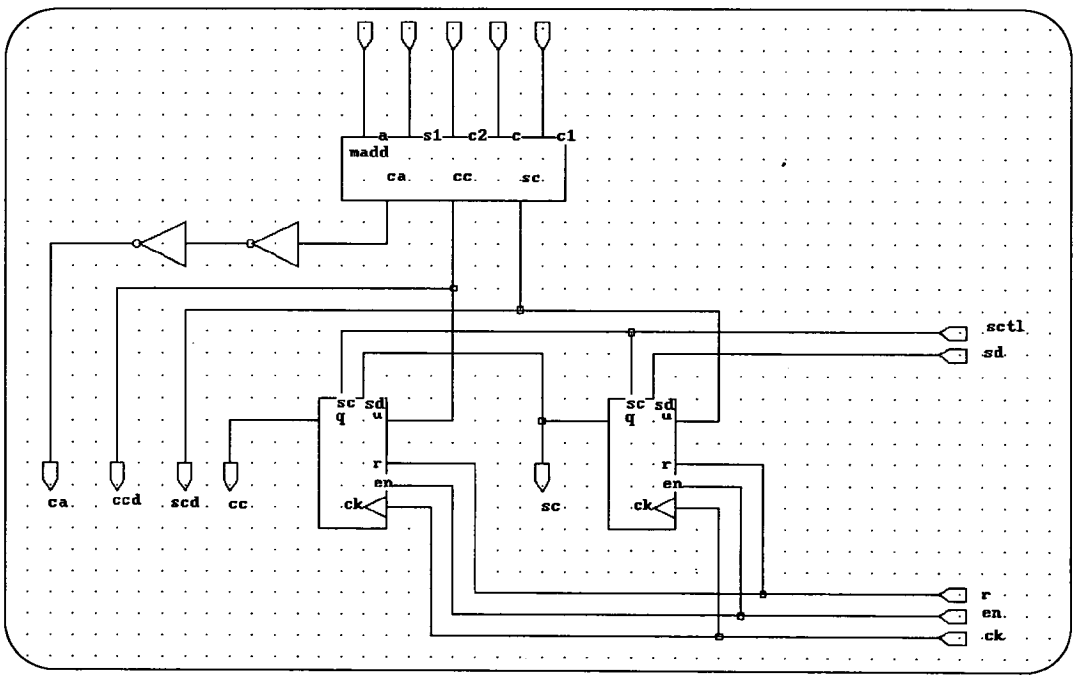

Figure 5.14 End Cell for Multiplier Array: 'mcelle'

The (n - 1) cell is similarly modified to provide access to the unlatched carry from the residue addition.

Figure 5.15 and Figure 5.16 highlight these additional signals to show how they propagate up through the design hierarchy, and may be compared to the basic bit slice cell of Figure 5.11

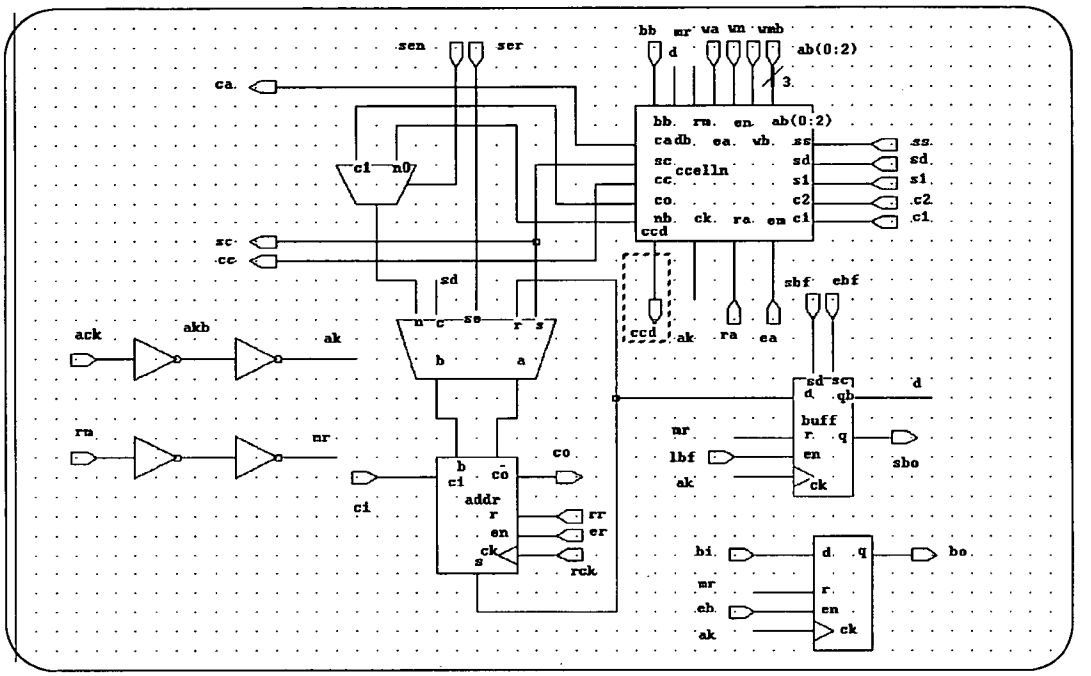

Figure 5.15 Bit Slice Cell (n - 1): 'celln'

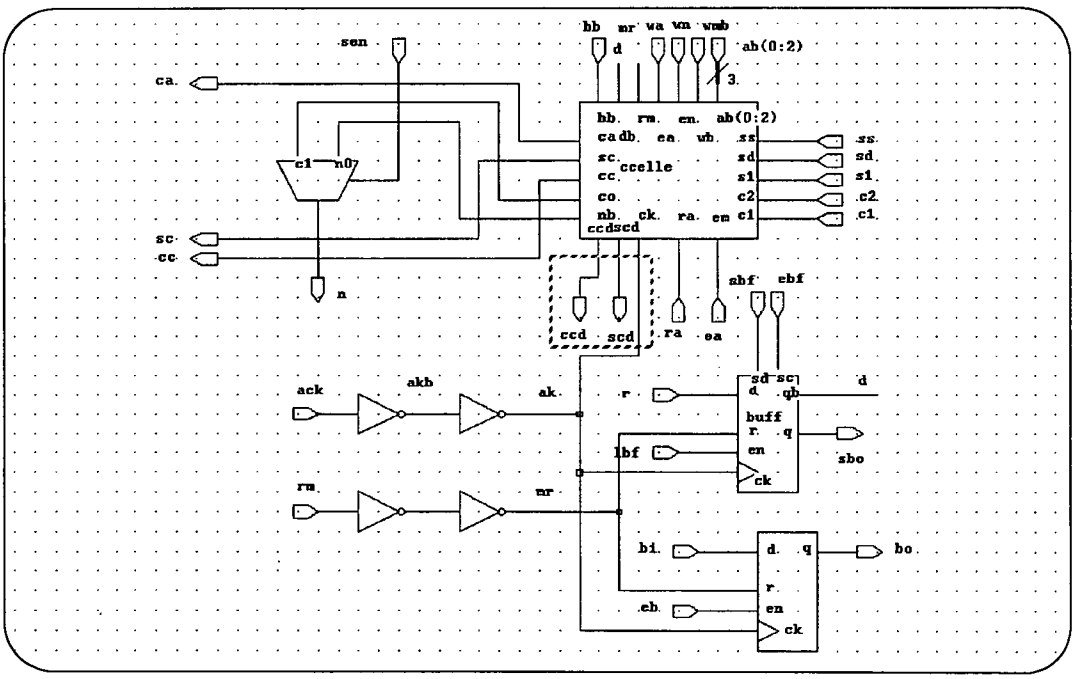

Figure 5.16 End Cell for Bit Slice: 'celle'

When adding the sums and carries from the array to form the normal binary representation of the data, the carry from the end cell has already been dealt with in the calculation of the overflow bits, thus the 'addr' cell is missing from Figure 5.16.

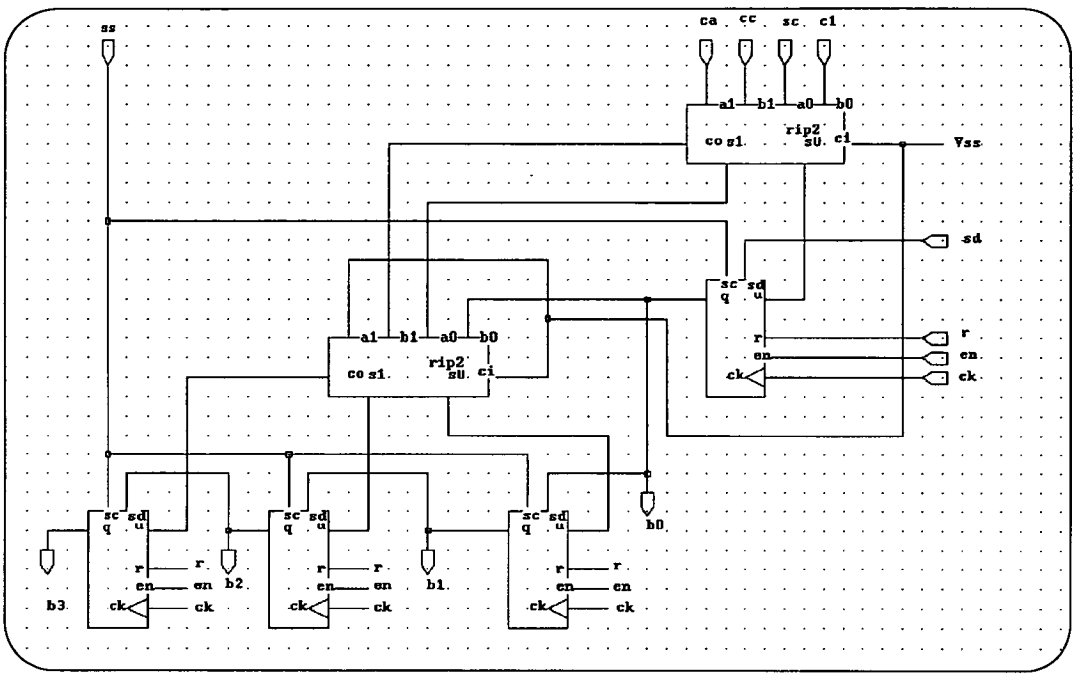

Figure 5.17 Decoder for Overflow Bits: 'mydecode'

Figure 5.17 shows where the sum and carry from the end cell are combined. The result is the signal b0 above. The four outputs from this circuit are connected to both the address output port for selecting residues, and the upper bits of the ripple add reduction circuit, Figure 5.18.

Although b0 and b3 in the above are not needed externally, they are routed out to improve testability of the circuit.

The basic bit slice cell includes one bit of a ripple adder and a latch to facilitate reduction by subtraction of the modulus. The contents of the array however, once sums and carries have been resolved, may form an (n + 3) bit number. Thus the ripple adder formed by combining the bit slices has to be extended by three bits which is the purpose of the circuit of Figure 5.18.

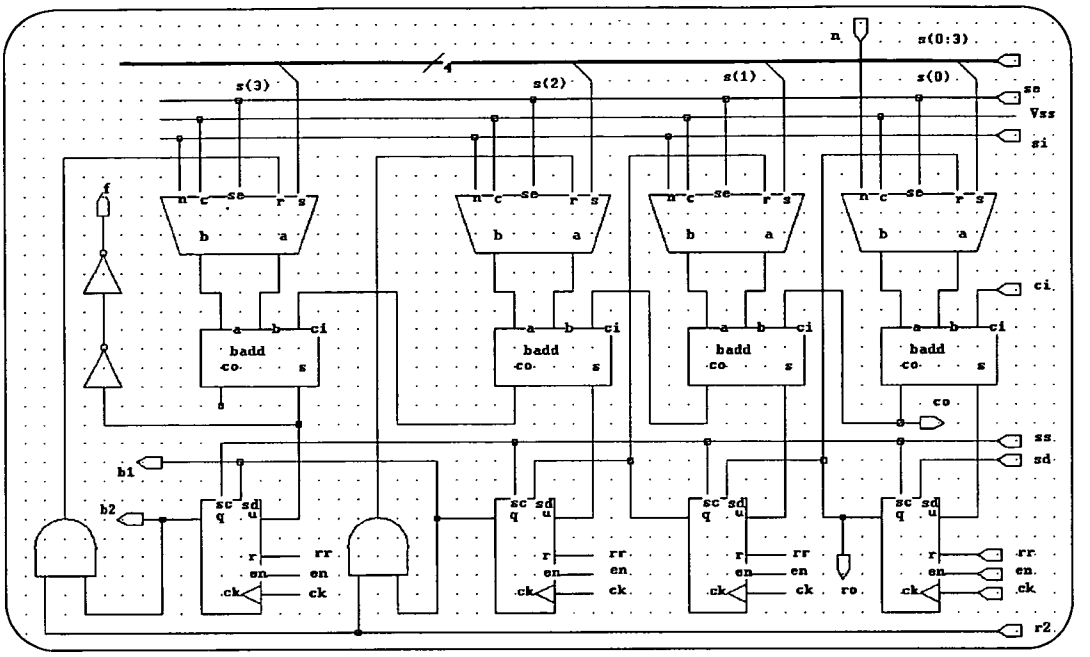

Figure 5.18 Upper Bits of Ripple Adder: 'rip4'

In Figure 5.18, the s(0:3) inputs are the overflow bits already calculated by the decoder of Figure 5.17. There is no need to combine sums and carries as the 'addr' cell does for lower order bits, and so the carry input 'c' to the multiplexer is connected to logical zero. The synthesis software optimises the layout of cells with inputs connected to VDD or VSS so no area is wasted in the above by connecting the inputs to VSS.

The other inputs to the adder, selected by control signal 'se', are from the accumulator register 'r' and modulus or residue 'n'. As discussed on page 86, the adder may perform partial reduction by adding residues, followed by complete reduction by subtracting the modulus. The 'n' input to the ripple adder must therefore be set to logical zero when adding the positive residues, and logical one when adding the two's compliment of the modulus. This sign is controlled by the input signal 'si'.

The 'ci' input is the carry from lower order bits. 'ss', and 'sd' are scan path select and scan path data. 'rr', 'en', and 'ck' are reset, enable, and clock signals for the latches.

The 'r2' input is a reset signal used to clear the two most significant bits during partial reduction.

The output signal 'co' is the ripple carry output that connects to the next device when chips are cascaded. The register output 'ro' transfers data to the I/O buffer in 'celle' Figure 5.16.

The output 'f' is a 'finish' signal that indicates reduction has been completed by outputting a logical one as soon as subtraction of the modulus yields a negative result. At this point the contents of the accumulator have been completely reduced and may be transferred to the I/O buffer.

The remaining output signals in Figure 5.18, b1 and b2, form the address of the residue to be added during partial reduction when they will be reset by 'r2'.

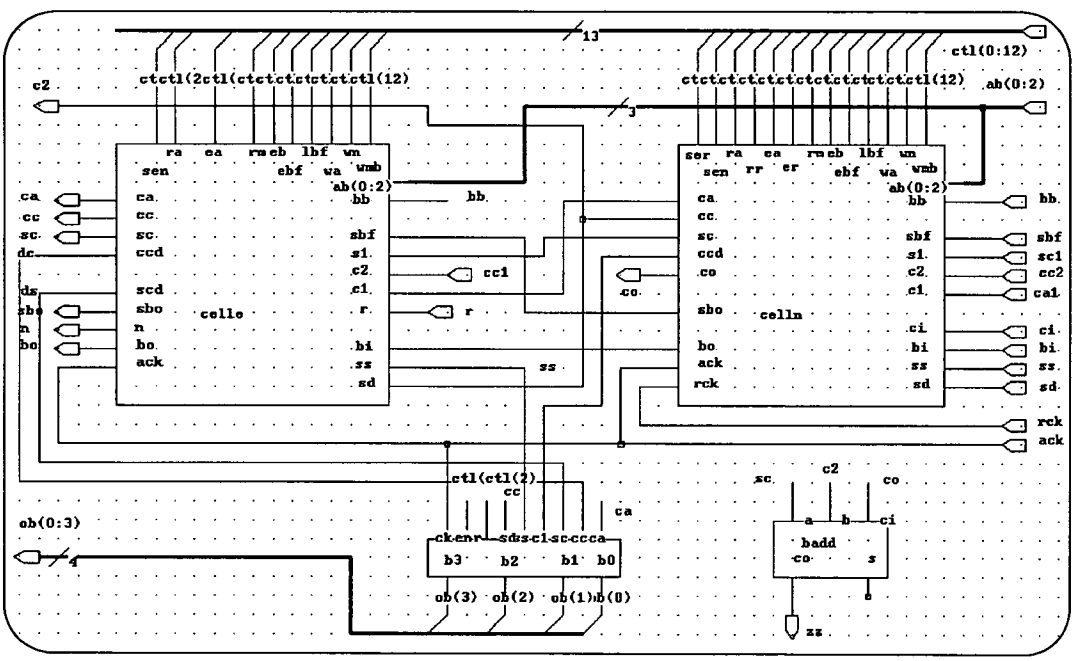

Figure 5.19 Combination of Two Most Significant Cells: 'cell2e'

Figure 5.19 shows the combination of the two most significant cells and connection to the overflow decoder to calculate the address of the next residue.

When the sums and carries from the multiplier array are resolved in 'rip4' the carry from bit b0 in Figure 5.17 has already propagated and does not appear anywhere in Figure 5.18. The carry out signal from 'rip4' will therefore be incorrect during this particular operation. To correct this, the adder in 'cell2e' computes the carry out directly from the last cell. All other arithmetic operations carried out on the ripple adder and accumulator will produce the correct value on

signal 'co' of Figure 5.18. The correct carry signal is selected by the multiplexer in the lower right hand side of Figure 5.20 before leaving the chip. The other multiplexer in this circuit determines whether the residue address is taken from the overflow decoder or the upper bits of the ripple adder.

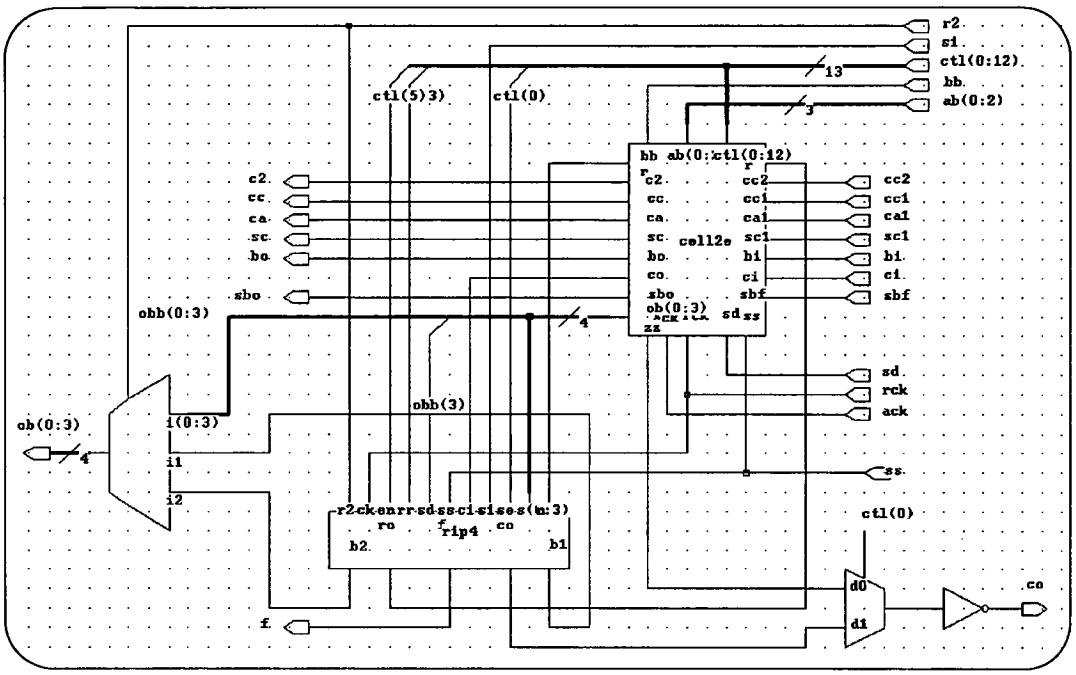

Figure 5.20 Selection of Outputs From End Cells: 'c2end'

5.4.3 Peripheral Cell Designs

The two most significant cells in the array, 'c2end' above, are combined with two standard cells, 'cell2' of Figure 5.12, to form a 'cell4e' and the hierarchy is built up until 'cell32' is created. This 'cell32' is connected to the peripheral circuits to form the core design of Figure 5.21.

In Figure 5.21either the serial output from the PISOs or an external serial input may be selected as input to the B register or I/O buffer. The serial input to the SIPO is also available at an output pin. The enable input to these registers allows data to shifted in/out serially, and the load signal latches parallel data.

Finally, multiplexers are added to the core design that select internal or external addresses and multiplier inputs b_i. The unconnected signals on Figure 5.22 are all routed to the external pins of the chip.

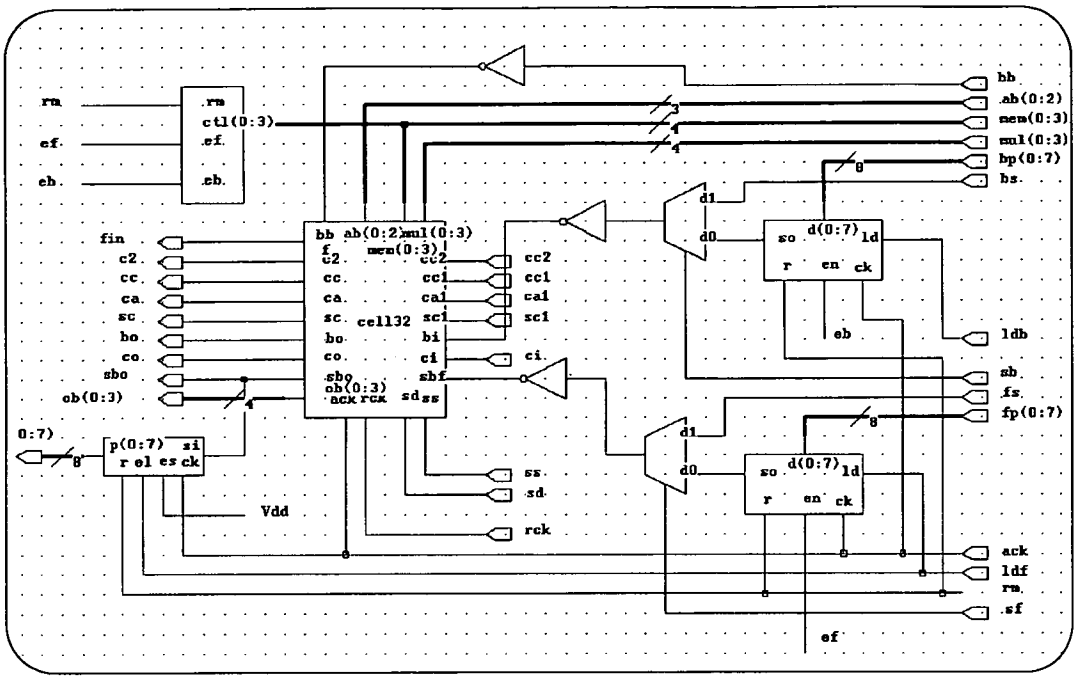

Figure 5.21 Core Design: 'core32'

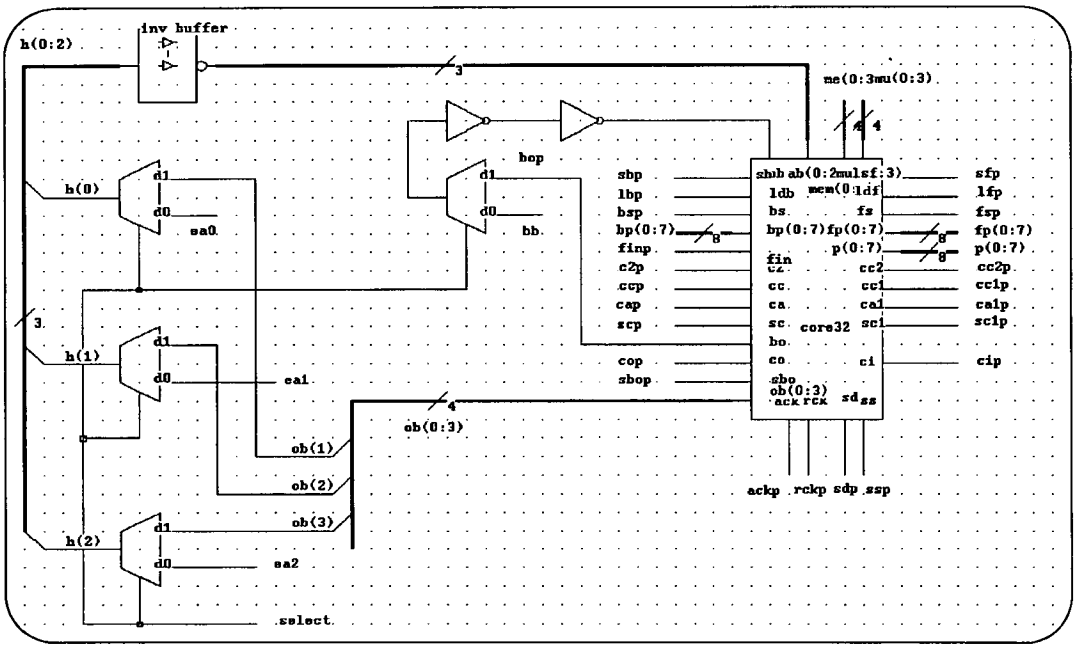

Figure 5.22 Top Level of Hierarchy

5.4.4 Buffering Strategy

ES2 measure drive capability by comparison with standard inverters, thus two inverters connected in parallel have a drive capability of two and so on. A similar scheme applies to input loads, and the two measures are combined to give an indication of relative fanout.

The standard ES2 latches used throughout this design have a drive capability of 0.3 units. Used as it stands, this latch would not have sufficient capability to drive the circuit of Figure 5.6 without unacceptably high relative fanout. Two inverters were therefore added to the outputs of the basic latch to form the 'sff' latch which is capable of driving the output buffer of the enable type latch. The output buffer has a drive capability of four units which is the default value for these parts. Most of the basic cells described in the preceding were buffered in this manner with a default drive of four units on their outputs. In most cases this was sufficient to cope with the local routing and loads encountered by these cells, however, once circuit loads were extracted from the layout, a more detailed estimate of fanout could be made and the drive capability of the buffers adjusted accordingly.

Global signals on the other hand were buffered in a tree like structure that was an automatic consequence of the hierarchical construction. The tree structure attempted to balance the delay of all global signals to each cell thereby reducing the possibility of skewing.

5.4.5 Bus Strategy

It is well known that automatic routing is a non trivial problem and that as much silicon area in a design may be allocated to routing as to active devices. One benefit of the bit slice approach and hierarchical description taken with this design is that routing is kept local, eliminating the need to route 32 bit wide busses throughout the chip.

As mentioned in section 5.4.1, the limited influence the designer has on floorplanning makes circuit delays difficult to control. As a consequence exact control over the timing of tristate buffers is difficult to achieve, which may lead to tristate driven busses being left in a high impedence state for an unacceptable amount of time. Since this could result in busses drifting to the inverter threshold voltage and drawing current, it was decided to use multiplexed busses throughout the design to avoid this possibility.

5.4.6 Control

It was originally intended to include some form of programmable logic array control circuit on the chip. Such a circuit would however, have to be constructed from separate logic gates resulting in the consumption of a large amount of silicon, reducing the data path from thirty two to sixteen bits. Opting for external control not only allows a wider data path but, together with separate memory and arithmetic control ports, gives more flexibility to the device operation.

The ELLA simulations defined a minimum set of instructions needed to perform modular arithmetic, to which several more diagnostic type commands were added. The instruction set is tabulated in Figure 5.23 where the hexadecimal value of the control inputs to the chip is given.

Arithmetic Instructions			Memory Instructions		
Mnemonic	Description	Hex Value	Mnemonic	Description	Hex Value
RST	Reset All	0	RST	Reset All	0
RBA	Reset Array	8	TBA	Xfer I/O Buffer -> A	A
RBR	Reset Register	4	TBN	Xfer I/O Buffer -> N	6
SPC	Add Sums + Carries	9	TBM	Xfer I/O Buff -> Mem	2
PLR	Add Residue	5	TRB	Xfer Register -> Buff	E
HAA	Halt Array	A	HAB	Halt B register	9
HAR	Halt Register	6	HAF	Halt I/O Buffer	5
HA2	Halt Both	3	HA2	Halt Both	8
RUN	Run	F	RUN	Run	F

Figure 5.23 Instruction Set

Arithmetic Instructions:

RBA: Resets the multiplier array and overflow decoder. Activates control bit ctl(2) which is connected to the 'ra' signal in the bit slice cells Figure 5.11, Figure 5.15, and Figure 5.16.

RBR: Resets the ripple adder register (accumulator). Activates control bit ctl(2) which is connected to the 'rr' signal in Figure 5.11, Figure 5.15, and Figure 5.16. The 'rr' signal also resets the latches in 'rip4'.

RST: Resets both the array and the accumulator.

SPC: Activates control signal ctl(0) which is connected to 'ser' in the bit slice cells and 'se' in 'rip4'. This signal normally selects the modulus and accumulator inputs to the adder but, when activated, ctl(0) connects the sum and carry outputs from the multiplier array to the adder. Control signal ctl(4), is also activated during this operation to disable the latches in the multiplier array.

PLR: Activates control signal ctl(1) which is connected to 'sen' in the bit slice cells. This signal is used in the bit slice cell to select between output from the modulus register 'n', and the residue memory, to be routed to the ripple adder. During normal operation data from the modulus register (-N) is added to (subtracted from) the number in the accumulator. Control signal ctl(1), when active, selects data from the residue memory according to the address inputs and allows partial reduction to be carried out in the ripple adder.

This command also activates the 'r2' input to 'rip4', resetting the two upper bits before the residue is added on the next cycle.

HAA: This disables signal ctl(4), disabling the latches in 'mcell', and halting the multiplier array.

HAR: Disables signal ctl(5) which is connected to the 'er' signal in the bit slice cells, and the 'en' signal in 'rip4'. This effectively halts the accumulator register in the reduction circuit.

HA2: Halts both the multiplier array, and the ripple adder accumulator.

RUN: This command will enable both the multiplier array and the ripple adder. The ripple adder will be configured to add the contents of 'nreg', the two's compliment of the modulus, to the accumulator. Since a two's compliment number is being added, this command also sets the 'si' input in 'rip4' to a logical one.

Memory Instructions:

RST: Activates ctl(6), connected to 'rm' in the bit slice cells, and resets the I/O register, A, B, and N registers, and the residue memory. Also resets the three interface registers in Figure 5.21.

TBM: Activates ctl(12), connected to 'wb' in the bit slice cells, enabling the latches in the residue memory to accept data from the bus connected to the I/O buffer.

TBN: Activates ctl(11), connected to 'wn' in the bit slice cells, enabling the N register to accept data from the bus connected to the I/O buffer.

TBA: Activates ctl(10), connected to 'wa' in the bit slice cells, enabling A register to accept data from the bus connected to the I/O buffer.

HAF: In response to this command, ctl(7), 'eb' in the bit slice cells, remains active while ctl(8) 'ef' and ctl(9) 'lf' are disabled. This effectively halts the I/O buffer. The 'eb' and 'ef' signals in the core circuit Figure 5.21 are affected in the same manner.

HAB: This command disables the B register by deactivating ctl(7). Control signals 8 and 9 however remain active. Ctl(8) is connected to the scan select input of the I/O buffer and configures this register for serial I/O. Ctl(9) is the enable signal for the I/O buffer.

HA2: Both the I/O buffer and the B register are halted

TRB: This sets control signal ctl(8) to allow parallel input to the I/O buffer, and ctl(9) to load data from the accumulator.

RUN: Control signal ctl(8) is set to configure the I/O buffer for serial I/O and ctl(9) enables this register. Ctl(7) enables the B register which allows the multiplicand to be shifted out most significant bit first. Control signals 10, 11, and 12, the memory write signals are all disabled.

Figure 5.24 shows how the commands are decoded and which signals in the basic bit slice cells are affected.

Arithmetic Instructions							Memory Instructions							
Ctl signal	0	1	2	3	4	5	Ctl signal	6	7	8	9	10	11	12
'cell' i.d.	sr	sn	ra	rr	ea	er	'cell' i.d.	rm	eb	ef	lf	wa	wn	wm
RST	0	0	1	1	0	0	RST	1	1	0	1	1	1	0
RBA	0	0	1	0	0	0	TBA	0	1	0	1	0	1	0
RBR	0	0	0	1	0	0	TBN	0	1	0	1	1	0	0
SPC	1	0	0	0	1	0	TBM	0	1	0	1	1	1	1
PLR	0	1	0	0	0	0	TRB	0	1	0	0	1	1	0
HAA	0	0	0	0	1	0	HAB	0	1	1	0	1	1	0
HAR	0	0	0	0	0	1	HAF	0	0	0	1	1	1	0
HA2	0	0	0	0	1	1	HA2	0	1	0	1	1	1	0
RUN	0	0	0	0	0	0	RUN	0	0	1	0	1	1	0

Figure 5.24 Control Signals

5.4.7 Verification

The schematic descriptions in the preceding sections are automatically translated into ES2's hardware description language, MODEL. The same model description is subsequently used for both simulation and synthesis.

Two levels of simulation are available: a switch level simulator capable of modelling circuit loads extracted from layout, and a logic level simulator that is less accurate but runs faster. The logic simulator was used at this stage in the design to verify correct logical operation.

Simulations are driven by setting up signals then specifying a time in nanoseconds for the simulation to run before changing the inputs again. Output signals and internal nodes may be marked for tracing and displayed either as a timing diagram, or as a truth table.

Low level drive files were defined first, to execute the basic commands of the previous section. Giving these files the same names as the mnemonics allowed higher level drive files to be described in a pseudo assembly language by calling the lower level files.

Before testing the full thirty two bit chip a smaller, eight bit design was used to debug the basic cells. By monitoring internal signals, the correct loading of data was established, and the multiplier array operation could be compared with both the PASCAL and ELLA simulations. Once the eight bit device had been debugged three of these chips were cascaded, modelling the ELLA of Figure 5.5. This cascaded system was thoroughly tested to verify correct operation of all serial links between chips before the thirty two bit chip was simulated.

The thirty two bit chip was tested to make sure all registers could be loaded and read back through external ports. As stated in the introduction to this chapter, this feature is not good cryptographic practice and would have to be disabled if the chip were to be used in a practical system. Full pipelined operation of the chip was then tested by running several multiplications according to the procedure described by Figure 5.2 on page 80.

Each time a modification to the design or test vectors was required both the thirty two bit device and the three chip system were re-simulated to make ensure the change had no undesirable side affects.

5.5 Silicon Production

To complete the physical design, the silicon layout of the device must be generated, circuit loads extracted, and simulations repeated before sending the design for fabrication. Once circuit loads are known, buffers can be adjusted to minimise relative fanouts. Placement may also be controlled to some extent, allowing wire lengths to be optimised.

5.5.1 Optimization of MODEL Code

During synthesis, the basic ES2 gates are placed in an array of rows and columns in an attempt to produce a die with a 1:1 aspect ratio. As a result, some cells may be split over two rows or columns introducing unpredictable variations in the performance of identical cells. Gaps may also be left within cells to assist routing between rows.

Fortunately the MODEL language has some features that allow limited control over cell placement.

The "uninterrupted" qualifier below forces the components that form the 'sff' latch of Figure 5.6 to be placed on the same row without leaving gaps for routing.

```
Part sff [ck, d, sc, sd] -> q, qb

  Signal s1b2x1, s1c2x1

  Uninterrupted

  bsdffn [Vdd,d,sd,sc,ck] -> s1c2x1,s1b2x1
  not [s1c2x1] -> qb
  not [s1b2x1] -> q
End
```

Figure 5.25 Uninterrupted Qualifier

The "serial" qualifier in the definition of the enable type latch instructs the software to maintain the textual order of calls within 'sffn' and any parts called from this part. Use of this qualifier and careful arrangement of the code can help local routing.

```
Part sffn [ck, d, en, r, sd, sc] -> q

  Signal qb, qq, s1c1x3, db, s1c1x1

  Serial

  not [en] -> s1c1x1
  not [sd] -> s1c1x3
  andnor(3,3) [qq,s1c1x1,r,r,en,d] -> db
  sff [ck,db,sc,s1c1x3] -> qb,qq
  buffer(4,4) [qb] -> q
End
```

Figure 5.26 Serial Qualifier

The penalty incurred by these qualifiers is that more silicon area may be required for the design, particularly where the "uninterrupted" qualifier is concerned. Only the circuit of Figure 5.6, and its non scan path equivalent, made use of these features to provide latches with consistent timing parameters.

5.5.2 Fanout Checks

The silicon layout was synthesised from the MODEL code and circuit loads extracted by the design software to produce a list of absolute and relative fanout for every node in the circuit.

The initial layout had seven nodes with relative fanout of around 40, and two at 20. Buffers driving these nodes were adjusted to reduce these figures to below 14 as shown below. The maximum relative fanout suggested by ES2 is 16, although for some nodes even this is not acceptable. Clock signals throughout this design were buffered to keep relative fanouts below eight.

Rel. Fanout	Number of Nodes
10 - 11	25
11 - 12	4
12 - 13	4
13 - 14	3
> 14	0

Figure 5.27 Relative Fanouts

Several nodes with large absolute fanouts were also modified, either by adding intermediate stages or by forming tree buffers, to produce more acceptable values. The suggested limits for absolute fanout were 35 for fast nodes and 60 for all others. Figure 5.28 shows the final distribution of absolute fanouts.

Abs. Fanout	Number of Nodes
30 - 40	114
40 - 50	16
50 - 60	2
60 - 70	2
> 70	0

Figure 5.28 Absolute Fanouts

Adjusting fanouts by modifying buffers is obviously an iterative process: each time a buffer is modified the placement is changed, altering circuit loads. The data presented in Figure 5.27 and Figure 5.28 are the final values obtained before the design was fabricated.

5.5.3 Wire Length Checks

In addition to providing fanout information, the design software produced a record of the wire lengths for each node in the circuit and issued a warning for each wire over 10,000 µm long.

The route of these long wires could be identified by instructing the synthesis software to plot the layout of single nets only. Several wires were routing signals from one side of the chip to the other, a distance of approximately 8mm. The source and destination of these wires could be traced back to the placement data file, which could be modified to physically move parts and reduce wire lengths. As with the fanout adjustments, this too was an iterative process: shortening one wire inevitably lengthened another. For some exceptionally long tracks, extra buffers had to be inserted to split the wire at a convenient point.

Many of the longer wires were routing signals from the core design to external pads so, where possible, the offending pads were moved closer to the signals source or destination. Unfortunately this was not always possible since some degree of order was required of the external pins. Most of the longer wires in Figure 5.29 are connections between the core design and the bonding pads. Clock signals were all well below the 10,000 µm warning limit.

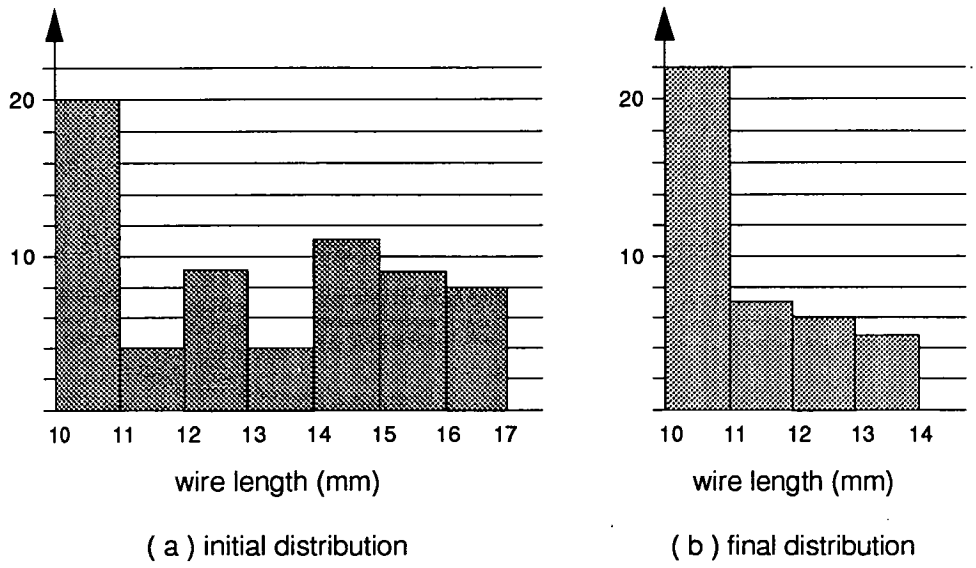

Figure 5.29 Wire Lengths Over 10 mm.

5.5.4 Layout

The final physical design task undertaken was the manipulation of the array parameters to minimise unused silicon area, and produce a die that would fit into one of ES2's standard packages. The array parameters that could be adjusted were the number of columns; the number of rows per column; and the number of cells per row. The situation is illustrated below.

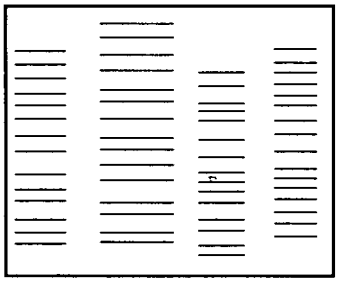

Figure 5.30 Arrangement of Cells

From initial attempts at generating artwork it was known that the device would be too big to fit any of the Dual In Line packages supplied by ES2. The most suitable package available from ES2 was the ceramic 68 pin Leadless Chip Carrier, the next largest size was an 84 pin

package. Sixty eight pins are more than enough for this device so the extra pins were used to improve testability by bringing out difficult to observe signals and allowing direct control over several registers and multiplexers.

Once the rows and columns had been adjusted to minimise area, the width of the power and ground rails were increased as far as possible within the constraints of the package.

The effects of altering the array parameters, moving cells to shorten wires, and buffering to improve fanouts are all inter-related, so each time a modification was made all three had to be checked again to make sure there were no adverse side affects. The final layout is the result of many iterations of the previous three processes and is presented in Figure 5.31.

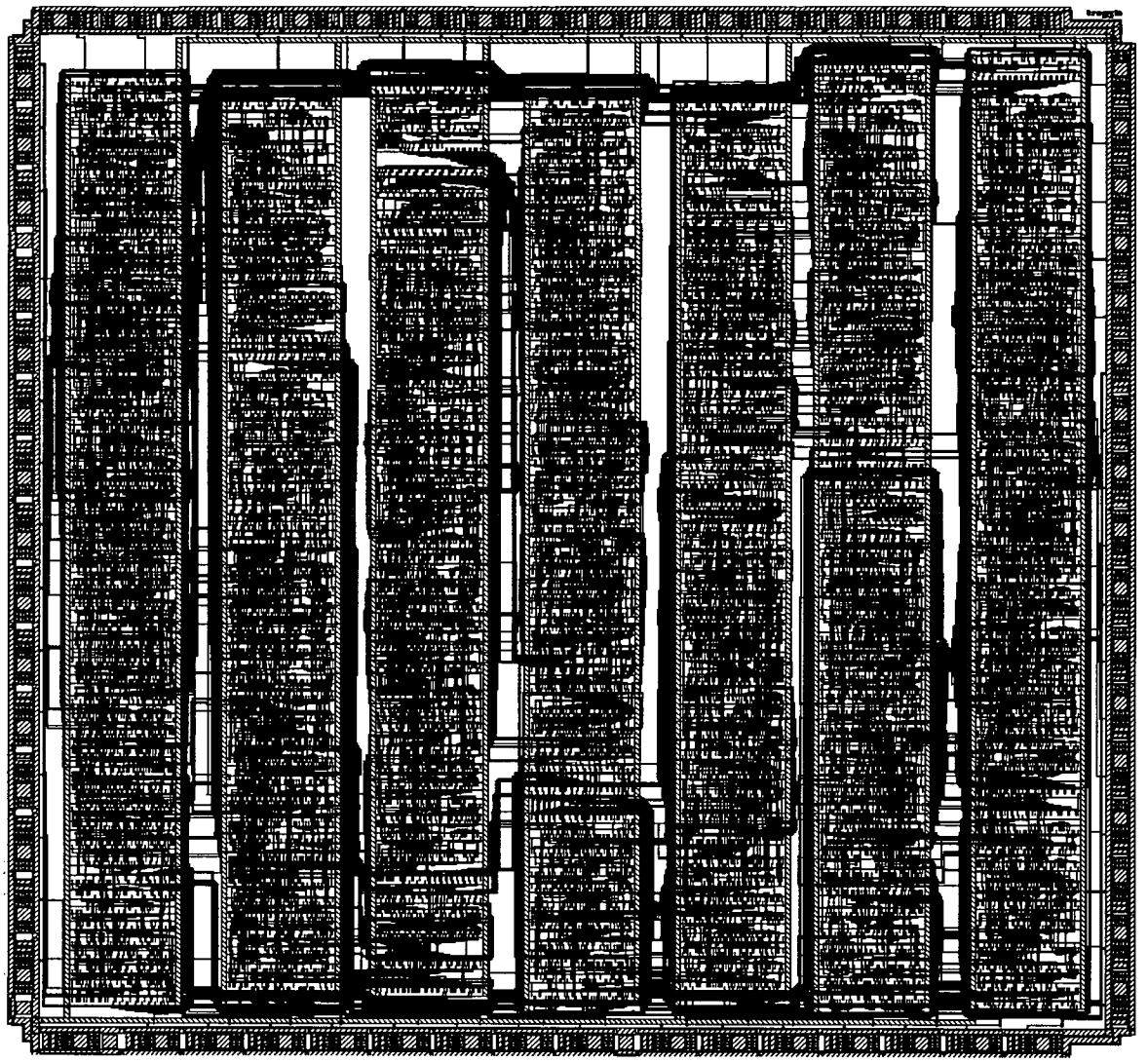

Figure 5.31 Silicon Layout

Layout is in 2μ CMOS technology using two layers of metalisation. Approximately 64,000 transistors were used and the design occupied an area of 8.77mm x 8.38mm, or 73.49 mm^2. The array itself measured 8.27mm x 7.86mm = 65 mm^2.

Figure 5.32 identifies the pins on the chip and gives a brief description. Figure 5.33 shows where the pins are located on the chip looking from above, and Figure 5.33 shows where the pins appear when the chip is inserted in the carrier.

PIN NAME	PIN No.	DESCRIPTION
mu3 - mu0	68 - 3	Multiply control inputs
me3 - me0	4 - 7	Memory control inputs
fs	8	I/O buffer serial input
bs	9	B register serial input
lf	10	Load I/O buffer PISO Read I/O buffer SIPO - active low
lb	11	Load B register SIPO - active low
sf	12	I/O buffer input select serial/parallel
sb	13	B register input select serial/parallel
fp7 - fp0	14 - 21	I/O buffer parallel data input
bp7 - bp0	22 - 29	B register parallel data input
VSS	30	
ob0 - ob3	31 - 34	Overflow bits
fin	35	Finish flag - active high
c2	36	CC(n - 2) output
cc	37	CC(n) output
ca	38	CA(n) output
sc	39	SC(n) output
bo	40	B register serial output
co	41	Ripple carry output
sbo	42	I/O buffer serial output
VDD	43	
VSSR	44	
p0 - p7	45 - 52	I/O buffer parallel output
VDD	53	
a0 - a2	54 - 56	Address input
ack	57	Array clock input
rck	58	Ripple adder clock input
ss	59	Scan path control
sd	60	Scan path data input
b	61	B(i) input
se	62	Select int / ext address & b inputs
cc2	63	CC(n - 2) input
cc	64	CC(n) input
ca	65	CA(n) input
sc1	66	SC(n) input
ci	67	Ripple carry input

Figure 5.32 Pin Description

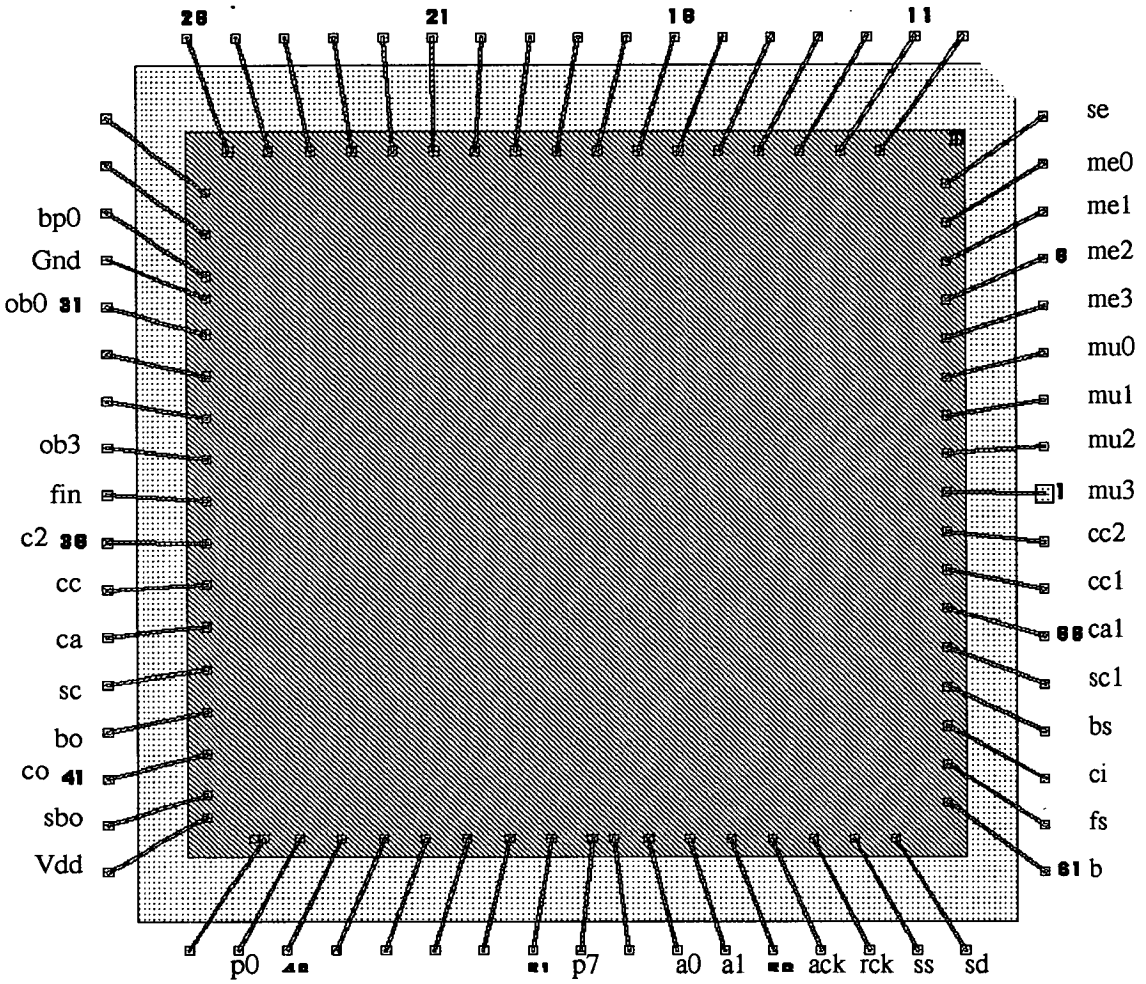

Figure 5.33 Pin Locations: Chip

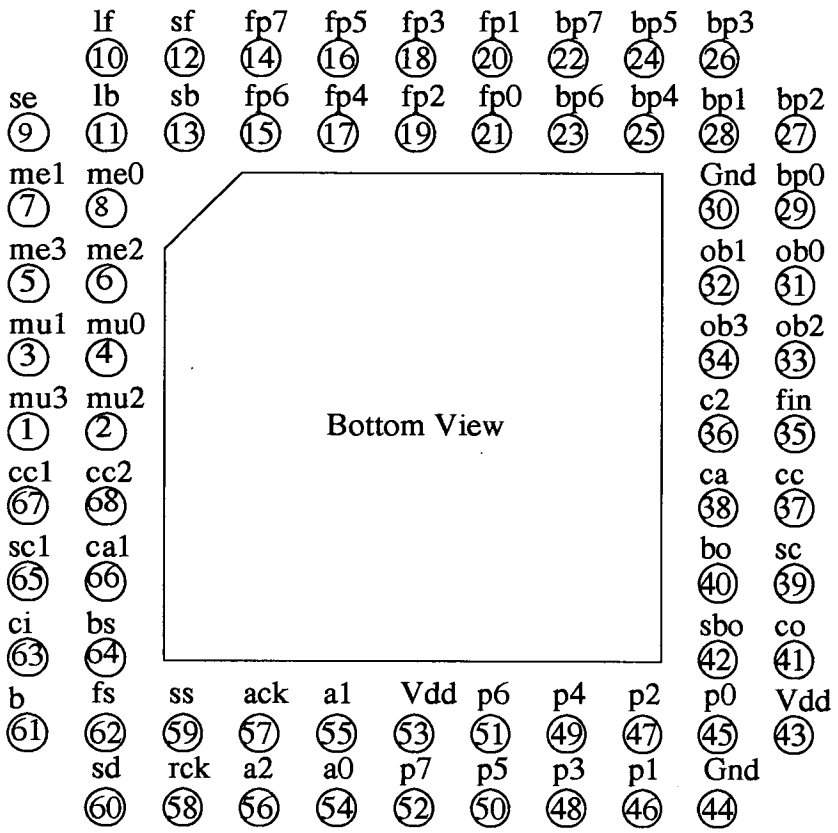

Figure 5.34 Pin Locations: Carrier

5.6 Test Vectors

The tests described in section 5.4.7 to establish functionality used logic models of the circuits and took no account of circuit loads. When the silicon layout was finalised, this data could be taken into consideration in the simulations and more detailed switch level circuit models could be used.

In addition to verification of device operation, the test vectors described in this section are submitted to the silicon foundry with the design files and used by the manufacturer to test device operation after fabrication. The test vectors therefore, must do more than checking that the chip functions as intended, but must also identify any fabrication defects likely to affect device operation. Two sets of test vectors are described here. The first tests the chip as it is intended to be used when configured as a thirty two bit multiplier. The second set tests all

registers and latches in the device to ensure both a logical one and zero can be loaded and read back. The ripple adder is then tested, and finally a series of random test vectors are applied to the inputs in an attempt to toggle as many nodes as possible in the remaining circuits.

5.6.1 Functional Test

This test resets the device then loads the modulus and corresponding residues. A series of five multiplications are then carried out fully pipelined. Once the fifth result is read back, a new modulus and set of residues are loaded and one more multiplication is done. Results are compared with those obtained from software models of the chip.

Several more examples were simulated to check functionality before submitting the design, but the above was thought adequate for post fabrication test. Figure 5.35 is a timing diagram showing the multiplication of the hexadecimal data :

(50 96 81 C2) X (AE DA 8F 6C) modulo (C3 71 F9 D9) = (24 ED 57 7C)

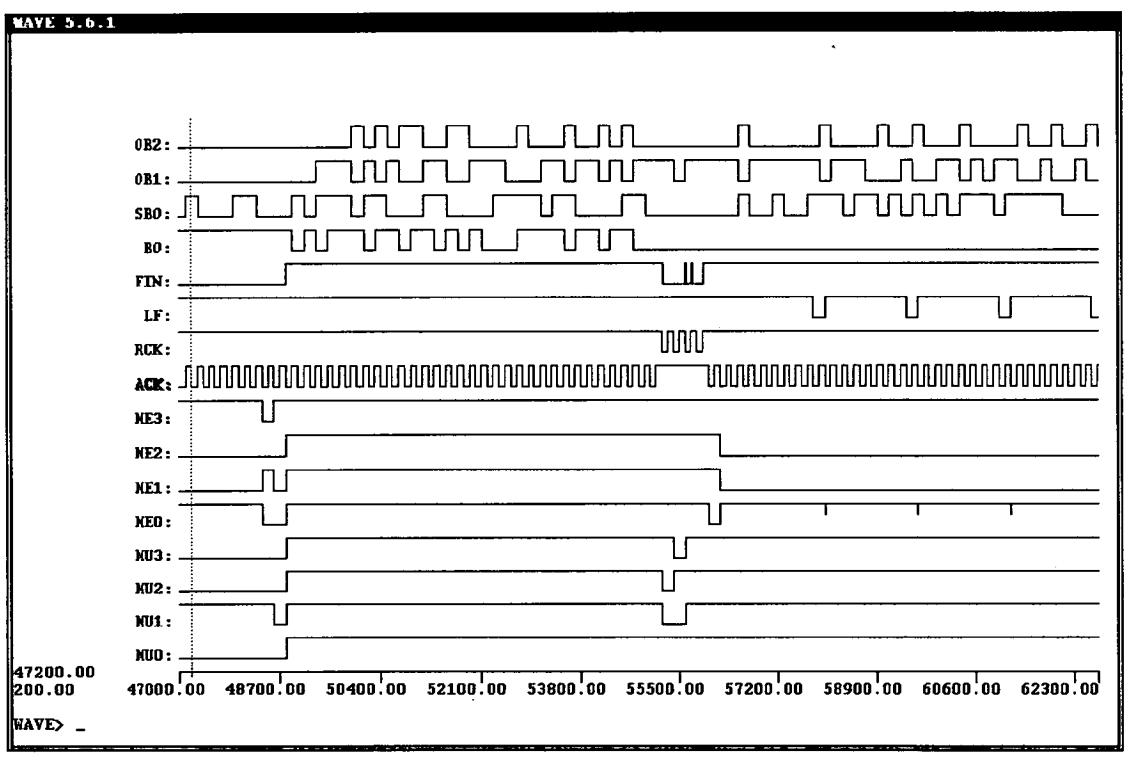

Figure 5.35 Timing Diagram of Simulation Results

The RUN command is issued just after 487000 ns. in Figure 5.35 at which point the address bits begin to appear on the overflow output pins OB1 and OB2, OB0 and OB1 are not displayed. The multiplier data, shifted out from the B register is displayed as signal BO. ACK and RCK are the clocks applied to the Array and the Ripple adder, FIN is the finish signal. Since the finish signal is the overflow from the ripple add, some glitches are inevitable, a suitable delay will be required before sampling this signal. When this flag is eventually set, the result is transferred to the I/O buffer and read out either from the parallel port using the LF strobes, or from the serial port SBO. The simulation results shown in Figure 5.35 were for both clocks running at 5 MHz. as is indicated by the 200 ns. figure in the lower right hand corner that measures the interval between the start of the display and the dotted bar. The data for this example was used to test the chip after fabrication, and a similar timing diagram is presented in Chapter 6.

5.6.2 Fault Coverage

The chip is reset and the hexadecimal value AA AA AA AA is loaded into the I/O buffer and 55 55 55 55 loaded into the B register via the parallel ports. These data are then read back from the I/O buffer's parallel port, and the B register's serial port. This sequence is then repeated with inverted data, thus ensuring none of the latches or nodes in the in the I/O circuits have 'stuck-at' faults.

The test continues by switching the chip into scan path mode and serially loading a pattern of alternating ones and zeroes through the scan path in the multiplier array, the decoder circuit, Figure 5.17, and the upper bits of the ripple adder Figure 5.18, to emerge at the overflow output pins.

The multiplier sum and carry latches are then loaded with data that results in the hexadecimal FF FF FF FF in the accumulator after addition. The result is read back and the test repeated with data that yields a result of zero, with a carry propagating all the way through the adder.

A pattern of alternating ones and zeroes is then loaded alternately into the A register, the N register, and the three residue registers via the parallel input port. After reading this data back,

via the multiplier array and accumulator, the test is repeated with opposite patterns of ones and zeroes. At this point any faults in the internal registers will be identified

The chip is then configured for serial input to the I/O buffer and B register and a pattern of alternating ones and zeroes shifted through both registers. The data is observed at the serial outputs of both these registers.

The carry inputs are then set to one, and a final multiplication carried out. Finally, all combinations of control vectors are applied to the inputs including those that do not relate to the specific commands described in section 5.4.6.

5.6.3 Random Vectors

Although the previous tests toggle all latches, some nodes still exist that remain untested. In an attempt to cover the remaining nodes, a short program was written in 'C' to generate a drive file for simulation that applied a random sequence of test vectors to the device. Applying 3,000 random test vectors to the device at this stage achieved a toggle rate of 87.6%. Extending this to 10,000 random vectors increased this figure to 88.0%. Many of the untoggled nodes were traced back to the 'setbar' and 'clearbar' inputs of the basic latches which in this design were tied to VDD. Taking this into consideration, the toggle rate for 3,000 random vectors was estimated to be at least 95%. The improvement gained in running more random vectors was too small to justify the extra simulation and test time required.

5.6.4 Performance Simulations

Further simulations were run to provide an indication of device performance in terms of encryption throughput. These tests were carried out using the following delay models for the transistors and circuit loads extracted from the layout.

	Temperature / °C	VSS / volts
Maximum delays:	70	4.5
Minimum delays:	0	5.5

Figure 5.36 Conditions for Performance Simulations

Array Speed

The multiplier array was tested by running repeated simulations at increasing clock speeds and observing the data as it left the most significant bits of the array. That is, the four carry out signals and the overflow bits calculated by the decoder of Figure 5.17 were monitored.

With minimum delay models the simulations produced the expected results until the clock rate reached 25 MHz. Maximum delay simulations ran correctly up to 15 MHz.

These data imply a throughput for 512 bit RSA encryption of between 30 and 50 kbits/s.

This estimate however takes no account of delays incurred by signal propagation between chips, and makes no allowance for global clock distribution between chips.

Adder Speed

The speed of the ripple adder was estimated by timing the carry propagation delay from the carry input pin 'cin' to the finish flag output pin 'fin' connected to the end of the ripple adder in Figure 5.18. With minimum delay models the carry propagation time was 88.8 ns. giving a clock rate of 11.3 MHz. Maximum delay models resulted in a carry propagation time of 153.5 ns. which is equivalent to 6.5 MHz.

5.7 Design Release Procedures

Before the silicon foundry would accept the design for fabrication, a series of checks were carried out by the design software.

First of all the design was simulated twice with the test vectors of section 5.6, once with maximum estimates of circuit delays, and once with the minimum estimates. The results of both simulations were compared by the software, and any discrepancies flagged as errors. The number of nodes toggled, the fanouts, and wire lengths were all checked to ensure they were above acceptable limits. Finally, the design rules for packaging and bonding were checked.

Once the software had successfully completed its checks, the design files were loaded on to tape and sent to the foundry for fabrication.

Chapter 6 Post Fabrication Tests

6.1 Static Tests

The test vectors described in section 5.6 were used by the silicon foundry to test the chips after fabrication. These signals were applied to the device at 1000 ns. intervals and 990 ns. allowed for the outputs to settle before being compared with the results predicted by simulation. Forty chips were fabricated and successfully tested before being delivered.

6.2 Dynamic Tests

6.2.1 Test Equipment

Before the dynamic tests could proceed a small board was built to supply power to the device and to connect the pins of the Leadless Chip Carrier (LCC) to vero pins where signals could be applied and observed.

A Textronix DAS 9100 logic analyser was used to apply the input vectors and monitor the response of the Device Under Test (DUT). The DAS could supply up to thirty-two input signals and monitor sixteen outputs from the device. Clock signals of up to 5 MHz. were also available and were used to drive the both array clock input 'ack' and the ripple adder clock 'rck'.

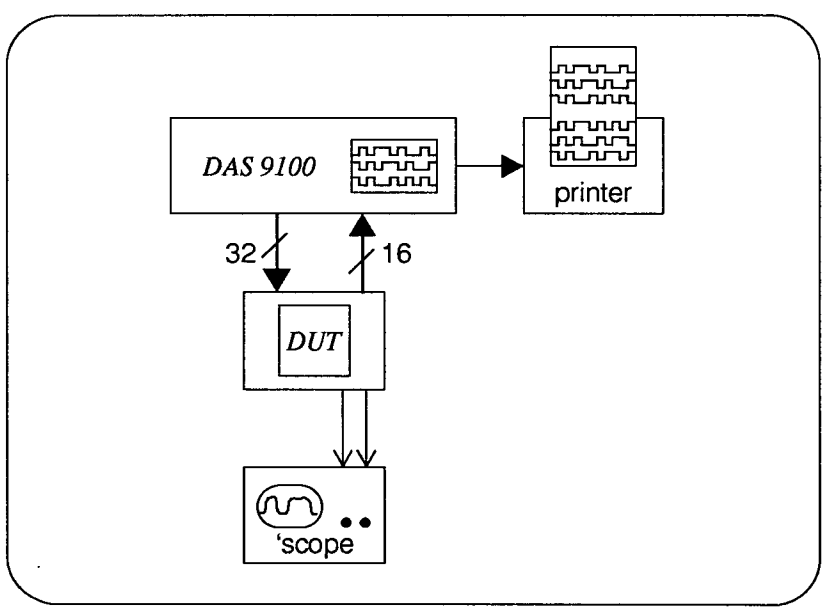

Figure 6.1 Test Equipment

The synchronisation of the two external clocks proved crucial to the device operation at high speeds and was optimised empirically by trying several of the clock waveform settings available from the DAS. Figure 6.2 illustrates the timing of the two clocks and the data input signals used during the 5 MHz. tests. The phasing of the two external clocks may prove to be a limitation on device operation at higher frequencies. Any projections therefore assume that the clock phasing is due to the test equipment, and can only be based on simulation results.

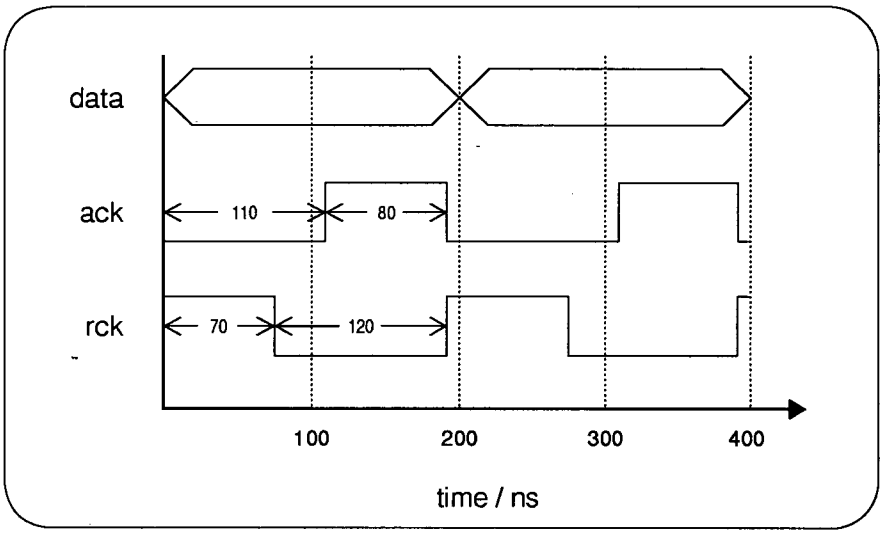

Figure 6.2 Timing for Input Signals

6.2.2 Test Procedure

As with the simulations and static tests, the operation of the two input buffers were verified to ensure data could be loaded and read back correctly first at 1 MHz., then at 5 MHz. Secondly, the scan path was used to verify the operation of the latches in the multiplier array by clocking a pattern of logical ones and zeroes through the device at 5 MHz. and observing the output at the 'fin' pin. The scan path was then used to load data to test the adder by observing carry outputs and reading results back from the accumulator.

Once these circuits were confirmed to be operating correctly the remaining registers were tested at 5 MHz. by loading them with alternating ones and zeroes and reading back the contents, where necessary through the multiplier array and ripple adder. Having established confidence in the memory transfer operations, the modular multiplication operation was tested.

6.2.3 Results

Figure 6.3 is the timing diagram acquired by the logic analyser during the modular multiplication operation for the example described in section 5.6.1 on page 111. The overflow bits that provide the address of the residue to be added may be compared to the results predicted by simulation, Figure 5.35, to verify correct operation.

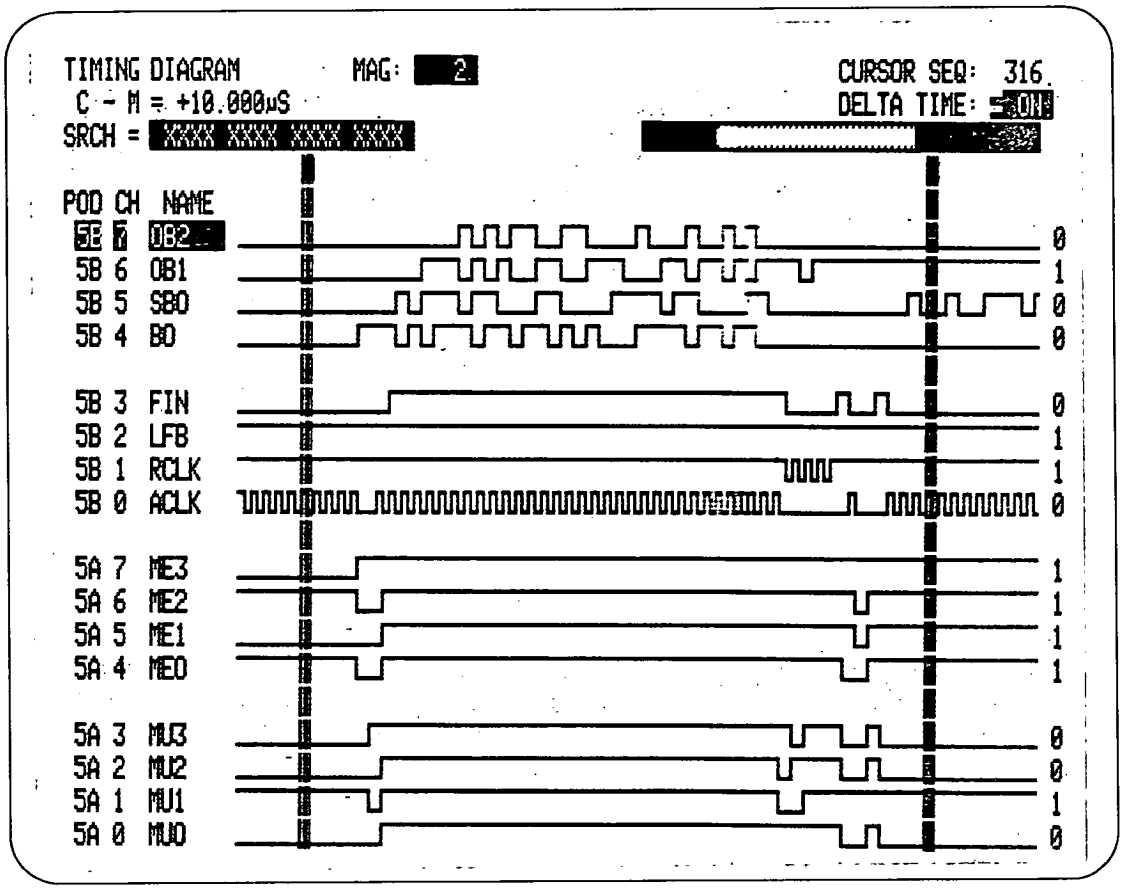

Figure 6.3 Overflow Bits From Logic Analyser

Figure 6.4 is the continuation of this example and shows the correct result being read out from the serial output port, SBO.

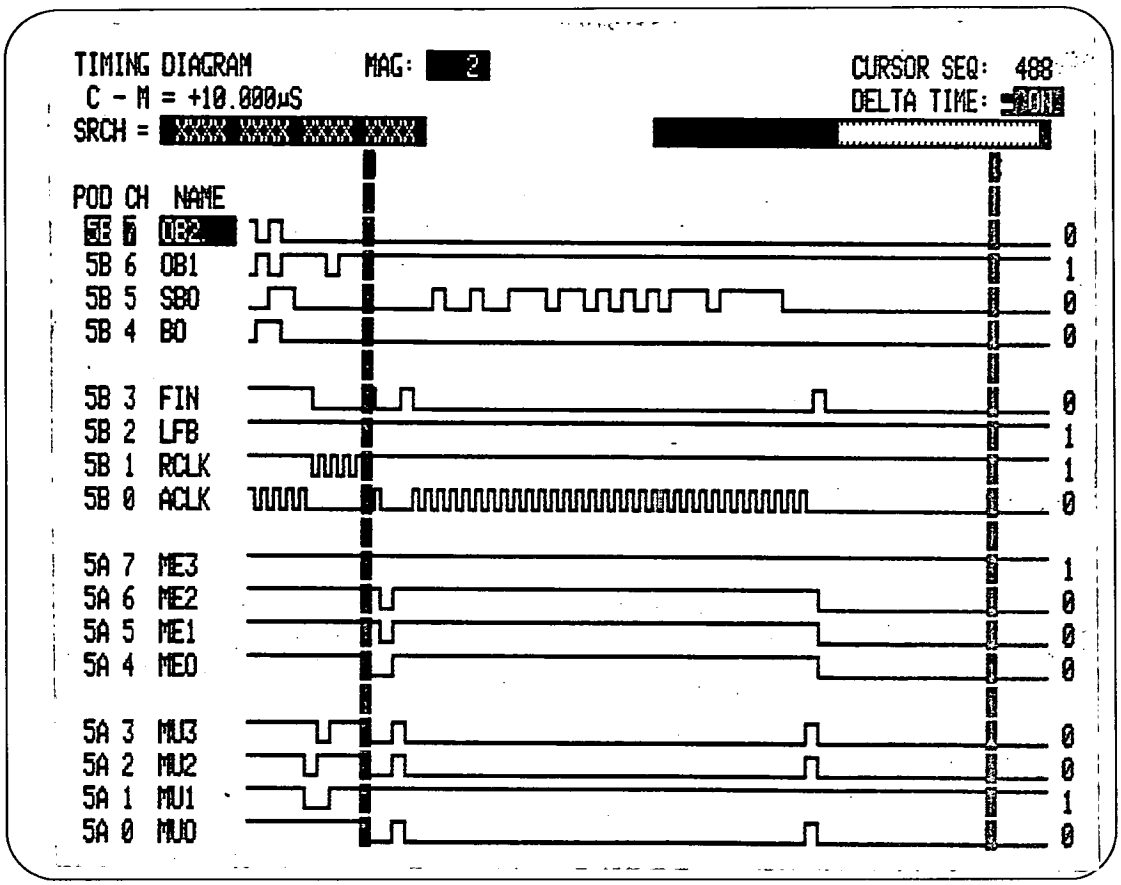

Figure 6.4 Multiplication Results From Logic Analyser

The oscilloscope photographs of Figure 6.5 show characteristic pulses from the 'OB2' output pin and were taken during the execution of this example at 5MHz.

The simulation data allowed the signals appearing on all other pins to be verified, including the serial links, thus predicting correct operation of a cascaded system. Several more examples were tested in this manner, all of which were successful at 5 MHZ.

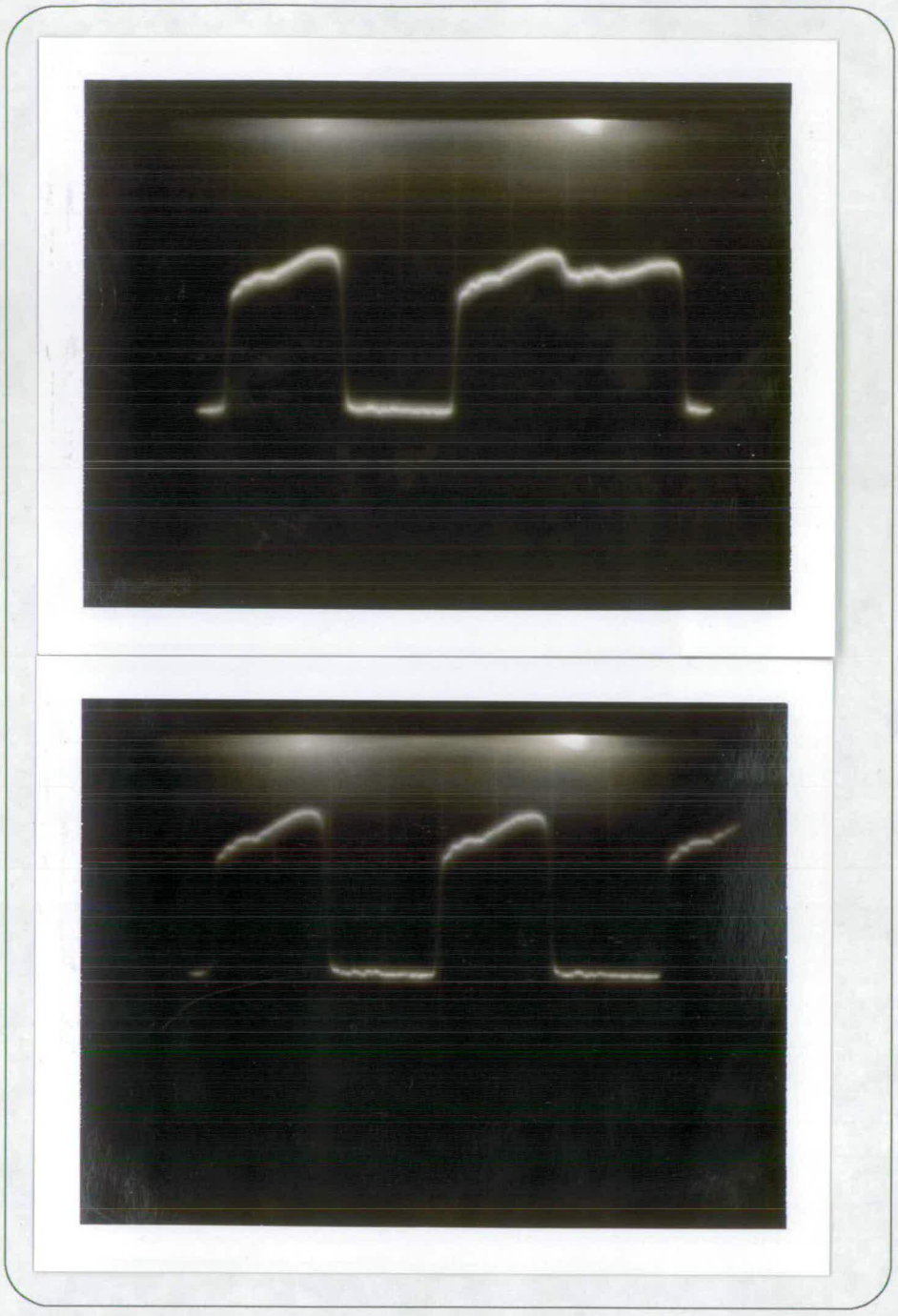

Figure 6.5 Characteristic Pulses From Overflow Bits

6.2.4 Power Consumption

The current drawn by the device was measured with VDD at 5.0 volts. When idle, the device drew 1mA., when running at 5 MHz., a current of 7mA. was measured, giving a power consumption of 35 mW.

Device operation was also verified with the positive supply voltage reduced to 4.0 volts, and again at 6.0 volts. In both cases the chip performed as expected.

6.3 Discussion

The tests described in this chapter have demonstrated correct device operation up to the 5 MHz. limit of the test equipment used. According to the analysis of section 4.4, this clock rate will result in a throughput for 512 bit RSA of 10 Kbits/s.

However, since the 32 bit ripple adder functioned correctly at this speed too, the delay through a single add stage may be calculated by dividing the clock period of 200 ns. by 32 to give 6.25 ns. The maximum clock rate for this device may then be calculated using equation (EQ 4.5) to be at least 27 MHz., which would imply a projected throughput for 512 bit RSA of 52 Kbits/s.

In practice, communication delays between chips and synchronisation of global signals across several chips may restrict the maximum operating frequency before the above limit is reached. Allowing a 50% derating factor for board level considerations gives an upper bound on the maximum operating frequency of around 15 MHz. At this frequency an encryption rate of 30 Kbits/s. can be expected for 512 bit RSA.

These figures are in good agreement with the results predicted by the simulations in section 5.6.4 which suggest worst and best case operating frequencies of 7 MHz. and 11 MHz. for the ripple adder, and 15 MHz. and 25 MHz. for the multiplier array.

Chapter 7 Concluding Remarks

A survey of modern cryptography has identified public key cryptography as an area where revolutionary theoretical developments are experiencing difficulty achieving commercial acceptance due to the poor performance of practical systems. Both hardware and software implementations of public key systems were reviewed and it emerged that the complexity of modular multiplication was by far the biggest restriction to high performance public key cryptosystems.

The modular multiplication operation was examined in detail from both theoretical and practical viewpoints and a novel algorithm presented to perform this operation. This algorithm uses modular arithmetic to restrict word growth as opposed to completely reducing data during intermediate calculations. Comparisons with the modulus are thereby eliminated, thus completely eliminating carry propagation from the multiplication procedure. This technique of partial reduction is proposed as practical means of improving the throughput of public key cryptosystems, allowing a fast clock rate which is independent of the long word lengths required to provide adequate security. In theory, the architecture proposed in this thesis could run up to 50 MHz. resulting in encryption rates for 512 bit RSA of 100 Kbits/s.

To demonstrate the proposed architecture and explore the practical aspects of implementing cryptology algorithms in VLSI, a modular arithmetic ASIC has been designed and fabricated. The device is a 32 bit wide data path cascadable to user defined word length which, when tested, functioned correctly up to the 5 MHz. limit of the test system. These tests allowed a figure of 27 MHz. to be estimated as a lower bound on the maximum clock rate for the device. Allowing a 50% reduction of this estimate for board level design considerations, it is feasible to expect an array of 16 of these chips to perform 512 bit RSA encryption at a rate of at least 30 Kbits/s.

7.1 Comparison with Similar Work

To allow comparison of this architecture with other published work, a figure of merit has been estimated for each design. The first stage in calculating this figure is to estimate cost, in terms of production yield. by estimating the hardware requirements of each architecture. This has been accomplished by referring to circuit diagrams and descriptions presented in the published

material to estimate the number of logic gates, adders, and latches needed for each bit of the modulus. The area consumed by each component is then allocated a number proportional to the silicon area it is expected to consume. To ensure these numbers are process independent, all data are based on figures given by ES2 for their logic cell library and are presented in Figure 7.1

Device	Units of area
Inverter	1
n Input And / Or	*n* + 2
Xor / Xnor	9
Adder	19
Latch	24

Figure 7.1 Silicon Area Occupied by Logic Gates

The encryption rate for each device is divided by the area per bit to yield the final figure of merit (FOM) in Figure 7.2. The column headed A/D indicates whether the published rate is for a proposed architecture (A) or for a physical device (D).

Design	Latches	Adders	Xors	4 I/P	2 I/P	Inv.	Area / bit	Rate	A/D	FOM
Tomlinson	6	2			1		186	100	A	537
Sedlak	6	2			59		418	94	A	224
Morita	9		6	15	15		450	80	A	177
Tomlinson	6	2			1		186	30	D	161
Q.U. CMOS	8	3		1	8	2	291	40	A	137
Q.U. NMOS	11	1		1	4	2	309	40	A	129
Brickell	6		5		13		241	20	A	82
Cryptech	10	2			1		282	17	D	60
B.T.	5	1	0.25		0.5	1.25	144	5	D	34
Kochanski					75		300	5	D	16
C.R.I.P.T	5	2			2		166	1.6	D	9
Rivest	11	2					302	1.2	A	3

Figure 7.2 Comparison of Published Architectures

The principal advantage of the approach proposed in this thesis is the benefit achieved by applying systolic techniques to large integer arithmetic. That is, the maximum clock rate and hence device performance is independent of word length.Also of importance is the fact that the method of partial reduction does not result in over complicated algorithms or circuit designs, and greatly reduces the storage requirements when compared with other table lookup approaches to modular arithmetic.

Although the storage requirements are low when compared with architectures based on lookup tables, the overall hardware requirements of the proposed architecture are necessarily greater than for the slower architectures based on carry propagate adders, some of which have no need to store any residues at all. Nor is the architecture completely systolic. The global clock and 'b[i]' signals will ultimately prove to limit device performance as word length increases. Synchronising the global clock will always be critical in any implementation of this architecture.

As mentioned above, one reason for designs in Figure 7.2 having a low figure of merit is their high hardware requirements. The Area/bit column in Figure 7.2 shows that only two designs require less hardware than the architecture presented in this thesis. This is a direct consequence of the partial reduction technique proposed in chapter four. It should be pointed out, however, that some of the architectures with low figures of merit, such as CRIPT or the British Telecom design are commercial designs with specific applications in mind which may not require high encryption rates. Other designs, such as Rivest's and Kochanski's use a technology that is now out of date and would probably achieve higher encryption rates if modern fabrication processes were used. Another point about Rivest's design is that it was one of the first RSA chips to be built. It was an early attempt at designing a general purpose 512 bit ALU to demonstrate that the RSA cipher could achieve *reasonable* encryption rates and not, therefore, finely tuned for large integer modular arithmetic.

Of the architectures that have higher figures of merit, both Sedlak and Morita have proposed modifications to the multiplication or exponentiation algorithms that imply some degree of parallelism. These two designs have by far the highest figures for Area/bit but make up for the excess hardware by increased encryption rates. The best figure of merit for a device that has been fabricated and is commercially available is due to the design by Cryptech. It is interesting

to note that the Cryptech design is similar to the architecture presented in this thesis as is illustrated by Figure 3.3 and Figure 4.1. The main difference being in the way that the partial products are reduced.

7.2 Future Directions

7.2.1 The Device

Work is under way to design a 512 bit modular arithmetic accelerator based on an array of 16 of the chips that were fabricated. As far as the device itself is concerned, the obvious improvement is to design some look ahead logic to reduce the critical delay in calculating the overflow bits. For a small hardware overhead the $6\delta_a$ delay of section 4.4 could be reduced to $2\delta_a$, the delay through a single stage of the multiplier array. This would effectively treble the frequencies and encryption rates of Figure 4.9, which for a δ_a of 5 ns. would imply an throughput of 200 Kbits/s for 512 bit RSA, and an upper limit on the operating frequency of 100 MHz.

The pin count could be reduced by using a common B-register and I/O buffer port, using an internal configuration register instead of external pins, and perhaps removing some of the test pins. These measures could allow the device to be mounted in a DIL package. Further improvements may be achieved by taking advantage of fabrication geometry reductions which may allow a 64 bit wide ASIC to be designed in a short time scale. Alternatively, a custom design may even allow a 128 or 256 bit device to be fabricated.

7.2.2 The Architecture

One of the major criticisms of the type of architecture presented in this thesis is the need to maintain a synchronous global clock. In response to this, it will be worth investigating the possible use of self timed circuits, or the systolic arrays for modular multiplication proposed by Çetin and Ching [32]. The method of partial reduction may also be adapted to take advantage of the fast multiplication algorithms proposed by Sedlak[126] and Morita [96].

7.3 Conclusion

The aim of the research presented in this thesis has been to improve the performance of public key cryptosystems. In pursuit of this goal the interdependence of algorithm and hardware design has become increasingly apparent. Investigating the requirements of both these areas has resulted in the proposal of a novel algorithm and demonstration of a device architecture that can be used to improve the throughput of modular arithmetic processors.

The techniques of proposed in this thesis could, in theory, be applied to design an RSA processor capable of 512 bit encryption at a rate of 200 Kbits/s with a 100 MHz. clock. In practice however, other design constraints such as delays through pad drivers will restrict the maximum operating frequency to around the 50 MHz. achievable with today's MOS technology.

This demonstrates that the methods proposed in this thesis have moved the throughput bottleneck from the basic multiplier architecture, to the limits imposed by the choice of implementation technology. Although this suggests an upper bound of 100 Kbits/s. for 512 bit RSA encryption, higher rates may be achieved with parallel processing architectures which is essentially what Sedlak[126], Morita [96], and Çetin and Ching [32] have suggested.

In conclusion, it may be stated that modular arithmetic is by far the most important algebraic system for cryptology, and is virtually the exclusive means by which public key cryptosystems are designed. Although conventional systems will always have an important role to play, the potential market for key management and authentication schemes will provide increasing motivation to research efficient public key cryptosystems. While key management schemes, by their nature, do not require high speed ciphering rates, digital signatures and identification schemes do. It is these applications that stand to benefit most from high performance modular arithmetic chips, and it is hoped that the work presented in this thesis may be developed further by researchers wishing to bring the benefits of public key cryptology to public use.

Appendix A. Background Maths for RSA

A.1 Greatest Common Divisor Theorem for the Integers

Given n_1 and n_2 not both zero in the ring of integers $\mathbf{Z}_m$, then there exists a and b in $\mathbf{Z}_m$ such that:

$$\gcd(n_1, n_2) = an_1 + bn_2 \qquad \text{(EQ A.1)}$$

Expressing the gcd in this way allows inverses to be computed by the extended Euclidean gcd algorithm below.

```
program extended_Euclidean_gcd;
(* calculates g = gcd( n1, n2 ) = a*n1 + b*n2 *)

var g, n1, n2, a, b,              (* variables identified above *)
    q, r,                         (* quotient, remainder, and   *)
    a1, b1, a2, b2, t: integer;   (* temporary storage.         *)

BEGIN
a1 := 1; b1 := 0; a2 := 0; b1 := 1; (* initialisation *)
writeln('Enter n1 and n2 ');
readln( n1, n2 );

REPEAT
   q := n1 div n2; r := n1 mod n2;
   IF  r = 0
   THEN BEGIN g := n2; a := a2; b := b2; END
   ELSE BEGIN
          n1 := n2; n2 := r;
          t := a2; a2 := a1 - q*a2; a1 := t;
          t := b2; b2 := b1 - q*b2; b1 := t;
        END;
UNTIL r = 0;

writeln('gcd(',n1,','n2,') = ',g, ' = ('a,'*',n1,' + ',b,'*',n2,'))
END.
```

A.2 Inverses in the Ring of Integers Z_m

In the ring of integers $\mathbf{Z}_m$, if $\gcd(u, m) = 1$ then u is *relatively prime* and said to be a *unit* in $\mathbf{Z}_m$. If u is a unit in $\mathbf{Z}_m$ then u will have a unique multiplicative inverse since from (EQ A.1)

$$\gcd(u, m) = am + bu = 1 \qquad \text{(EQ A.2)}$$

and since $a \cdot m = 0$ modulo m, $bu = 1$ thus:

$$b = u^{-1} \qquad \text{(EQ A.3)}$$

A.3 Closure of the Set of Units Z_m

Let the set of units in $\mathbf{Z}_m$ be denoted $\mathbf{Z}_m^*$. If a and b are members of the set Z_m^* then there exist unique multiplicative inverses a^{-1} and b^{-1}. So

$$(a \times b) \times (a^{-1} \times b^{-1}) = a \times (b \times b^{-1}) \times a^{-1} = a \times a^{-1} = 1 \quad \text{modulo } m \qquad \text{(EQ A.4)}$$

Thus the product, modulo m, of two members of the set Z_m^* also has a unique inverse.

So if every element in $\mathbf{Z}_m^*$ is multiplied by the same unit, modulo m, then the same set of units Z_m^* is generated only in a different order.

A.4 Euler's Theorem

The number of integers in $\mathbf{Z}_m$ which are relatively prime to m is measured by Euler's Totient Function and usually given the symbol $\Phi(m)$. Thus for the set of units $\mathbf{Z}_m^*$, $\Phi(m)$ is simply the total number of elements in the set.

From section A.3 it can be seen that multiplying all r_i's in $\mathbf{Z}_m^*$ together will give the same result as multiplying all ar_i's together. More formally,

$$\prod_{i=1}^{\Phi(m)} ar_i = \prod_{i=1}^{\Phi(m)} r_i \qquad \text{modulo m} \qquad \text{(EQ A.5)}$$

So, taking a out of the product in (EQ A.5)

$$a^{\Phi(m)} \prod_{i=1}^{\Phi(m)} r_i = \prod_{i=1}^{\Phi(m)} r_i \qquad \text{modulo m} \qquad \text{(EQ A.6)}$$

thus:

$$a^{\Phi(m)} = 1 \qquad \text{modulo m} \qquad \text{(EQ A.7)}$$

Euler's Theorem may be stated as follows:

> If a is a unit in $\mathbf{Z}_m$, then $a^{\Phi(m)} = 1$ modulo m

A.5 Fermat's Theorem

Fermat's theorem is a special case of Euler's theorem for prime modulii. If m is a prime modulus then every nonzero element in $\mathbf{Z}_m$ will be relatively prime to m, and the set of units $\mathbf{Z}_m^* \setminus \{0\}$ will be equal to $\mathbf{Z}_m$. Thus $\Phi(m) = m - 1$ and

$$a^{m-1} = 1 \qquad \text{modulo m} \qquad \text{(EQ A.8)}$$

Fermat's Theorem may be stated as follows:

> If a is an element in $\mathbf{Z}_m$, and m is prime then $a^{m-1} = 1$ modulo m

A.6 Totient Function for Composite Numbers

Finding the totient function for prime numbers is trivial, composite numbers however, are more difficult to deal with directly. Let m be a composite number that can be factored into two primes p and q.

To find $\Phi(m)$, all the elements of $\mathbf{Z}_m$ that are relatively prime to m must be found. To do this, it is easier to find the elements of $\mathbf{Z}_m$ that are *not* relatively prime, these will be all multiples of p, and all multiples of q, less than $m = pq$.

That is: $\{p, 2p, 3p, \ldots, (q-1)p\}$, $(q-1)$ elements

and: $\{q, 2q, 3q, \ldots, (p-1)q\}$, $(p-1)$ elements

So $\Phi(m)$ = Total no. of elements in $\mathbf{Z}_m$ - no. of elements that are not relatively prime

$$\Phi(m) = m - 1 - (p-1) - (q-1) \qquad \text{(EQ A.9)}$$

$$\Phi(m) = pq - 1 - (p-1) - (q-1)$$
$$\Phi(m) = pq - p - q + 1$$
$$\Phi(m) = (p-1)(q-1)$$

$$\Phi(m) = \Phi(p)\Phi(q) \qquad \text{(EQ A.10)}$$

So if m can be factored into p and q then:

$$\boxed{\Phi(m) = \Phi(p)\Phi(q) = (p-1)(q-1)}$$

Appendix B. Bibliography

[1] CY512 and CY1024 Key Management Processors. CYLINK, 110 South Wolfe Road, Sunnyvale, CA94086, USA

[2] RSA Cryptochip. RSA Security Inc., 1717 Karameos Drive, Sunnyvale, CA94087, USA

[3] RSA Processor and PC-RSA Processor. CRYPTECH NEDERLAND, Gronigeweg 6, 2803 PU Gouda, The Netherlands

[4] Data Ciphering Processors Am9518, Am9568, AmZ8068. Advanced Micro Devices, 901 Thompson Place, PO Box 3453, Sunnyvale CA94088 USA

[5] Data Encryption Standard FIPS PUB 46, National Tech. Info. Service, Springfield, VA, Jan. 1977.

[6] Guidelines For Implementing and Using the NBS Data Encryption Standard FIPS PUB 74, National Tech. Info. Service, Springfield, VA, Apr. 1981.

[7] NBS Report on Workshop, Fall 1976

[8] "Report of the workshop on estimation of significant advances in computer technology," NBSIR 76-1189, National Bureau of Standards, Dec. 1976.

[9] "The Directory-Authentication Framework", CCITT, Draft Recommendation X.509, COM VII-204-E, Jan. 1988

[10] Adams, C. M., and Meijer H., "Security related comments regarding McEliece's public key cryptosystem", in *Advances in Cryptology - Proceedings of CRYPTO '87*, ed. C. Pomerance, Lecture Notes in Computer Science, No. 293, pp. 224 - 228, Springer-Verlag, 1988.

[11] Alia, G., and Martinelli, E., "A VLSI Structure for X (mod m) Operation", *Journal of VLSI Signal Processing*, Vol. 1, No. 4, April 1990, pp. 257 - 264,

[12] Aruliah, A. A., "A Pascal Implementation of the RSA Algorithm", NPL Report DITC 66/85, Sept. 1985

[13] Baker, P.W., "Fast computation of A*B modulo N," *Electronics Letters*, vol. 23, pp. 794 - 5, May 1987.

[14] Bamford, J., *The Puzzle Palace,* Penguin, 1982

[15] Barker C., "An industry perspective of the CCEP," presented at the 2nd Annual AIAA Computer Security Conference Proceedings, Dec. 1986

[16] Barrett, P., "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor," in *Advances in Cryptology - Proceedings of CRYPTO '86*, ed. A.M. Odlyzko, Lecture Notes in Computer Science, No.263, pp. 311-323, Springer-Verlag, 1987.

[17] Beker, H. J., Friend, J. M. K., and Halliden, P. W., "Simplifying Key Management in Electronic Fund Transfer Point Of Sale Systems", *Electronics Letters*, Vol. 19, No. 12, June 1983, pp. 442 - 444.

[18] Beker, H., and Piper, F., *Cipher Systems: The Protection of Communications*, Northwood Books, London, 1982.

[19] Berlekamp, E. R., *Algebraic Coding Theory*, McGraw-Hill, 1968.

[20] Bertin, D. R., and Vuillemin, J., "Introduction to Programmable Active Memories", in *Systolic Array Processors*, ed. J.McCanny, J. McWhirter, and E. Swartzlander, Prentice - Hall, pp. 301 -309, 1989.

[21] Beth, T., B.M. Cook, and D. Gollman, "Architectures for exponentiation in GF(2**n)," in *Advances in Cryptology - Proceedings of CRYPTO '86*, ed. A.M. Odlyzko, Lecture Notes in Computer Science, No.263, pp. 302 - 310, Springer-Verlag, 1987.

[22] Beth, T. E., and Gollmann, D., "Algorithm Engineering for Public Key Algorithms", *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 4, May 1989, pp 485 - 466.

[23] Biham, E. and Shamir, A. "Differential Cryptanalysis of DES-like Cryptosystems". presented at CRYPTO '90

[24] Blakley, G.R., "A Computer Algorithm for Calculating the Product AB Modulo M," *IEEE Trans. On Computers*, vol. C-32, pp. 497-500, May 1983.

[25] Blakley, G.R., and Borosh, I., "Rivest-Shamir-Adleman Public Key Cryptosystems do not always conceal messages", *Comp. and Math. with Applic.* vol. 5, pp 169 - 178, 1979.

[26] Bong, D., and Ruland, C., "Optimised Software Implementations of the Modular Exponentiation on General Purpose Microprocessors", *Computers and Security*, Vol. 8, 1989, pp 621 - 630.

[27] Branstad, D. "Hellman's data does not support his conclusion", *IEEE Spectrum*, July 1979, pg. 41.

[28] Brickell, E. F., "A fast modular multiplication algorithm with application to two key cryptography," in *Advances in Cryptology - Proceedings of CRYPTO '82*, ed. A.T. Sherman, pp. 51 - 60, Plenum Press, 1983.

[29] Brickell, E. F., "Breaking Iterated Knapsacks", in *Advances in Cryptology: Proceedings of CRYPTO 84*, ed. G.R. Blakley and D. Chaum, Lecture Notes in Computer Science, No. 196, Springer-Verlag, 1985.

[30] Brickell, E. F., "A Survey of Hardware Implementations of RSA", in *Advances in Cryptology: Proceedings of CRYPTO 89*

[31] Brickell, E. F., and Odlyzko A. M., "Cryptanalysis: A Survey of Recent Results", *Proc. IEEE*, vol. 76, pp. 578-593, May 1988.

[32] Çetin K. Koç and Ching Yu Hung, "Bit-level systolic arrays for modular multiplication", *The Journal of VLSI Signal Processing*. (to be published).

[33] Chaitin, G. J., "Information-Theoretic Limitations of Formal Systems", J. *ACM*, Vol. 24(2), pp 403 -424, July 1974.

[34] Clayden, D.O., "Some methods for computing the RSA modular exponential," *NPL Technical Memo* TTCC 20/85, Aug. 1985.

[35] Davida, G. I., "Hellman's scheme breaks DES in its basic form", *IEEE Spectrum*, July 1979, pg. 39.

[36] Davida, G. I., and F. B. Dancs, "A Crypto-Engine," in *Advances in Cryptology - Proceedings of CRYPTO '87*, ed. C. Pomerance, Lecture Notes in Computer Science, No. 293, pp. 257-268, Springer-Verlag, 1988.

[37] Davio, M., Y. Desmedt, J. Goubert, F. Hoornaert, and J. -J. Quisquater, "Efficient Hardware and Software Implementations for the DES," in *Advances in Cryptology: Proceedings of CRYPTO 84*, ed. G.R. Blakley and D. Chaum, Lecture Notes in Computer Science, No. 196, pp. 144-146, Springer-Verlag, 1985.

[38] Davis, D. W., "Applying the RSA Digital Signature to Electronic Mail", *Computer*, February 1983, pp. 55 - 62.

[39] Davis, D. W., and Price, W. L., "The application of digital signatures based on Public Key Cryptosystems", NPL report DNACS 39/80, National Physical Labs., Teddington, Middlesex, England, Dec. 1980

[40] De Milo, R., and Merritt, M., "Protocols for Data Security", *Computer*, February 1983, pp. 39 - 51.

[41] Denning, D. E. R., *Cryptography and Data Security*, Addison-Wesley, Reading, MA, 1982.

[42] Denning, D. E. R., "Protecting Public Keys and Signature Keys", *Computer*, February 1983, pp. 27 - 35.

[43] Desmedt, Y., Vandewalle, J., and Govaerts, R., "How Iterative Transformations can help to crack the Merkle - Hellman cryptographic scheme", *Electronics Letters*, Vol. 18, No. 21, October 1982, pp. 910 - 911.

[44] Diffie, W., "The First Ten Years of Public-Key Cryptography," *Proc. IEEE*, vol. 76, pp. 560-577, May 1988.

[45] Diffie, W., and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. Info. Theory*, vol. IT-22, pp. 644-654, Nov. 1976.

[46] Diffie, W., and M. E. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard", *Computer*, vol. 10, pp 78-84, June 1977.

[47] Diffie, W., and M. E. Hellman, "Privacy and Authentication: An Introduction to Cryptography", *Proceedings of the IEEE*, Vol. 67, No. 3, March 1979.

[48] El Gamal, T., "A Public Key cryptosystem and a signature scheme based on discrete Logarithms", *IEEE Trans. on Information Theory*, Vol. IT-31, No. 4, July 1985, pp. 469 - 472.

[49] Erl-Huei, Lu Lein Harn, Jau-Yein Lee, and Wen-Yih Hwang, "A programmable VLSI architecture for computing multiplication and polynomial evaluation modulo a positive integer," *IEEE J. Solid-State Circuits*, vol. 23, pp. 204 - 207, Feb. 1988.

[50] Fairfield, R. C., A. Matusevich, and J. Plany, "An LSI Digital Encryption Processor (DEP)," in *Advances in Cryptology: Proceedings of CRYPTO 84*, ed. G.R. Blakley and D. Chaum, Lecture Notes in Computer Science, No. 196, pp. 115-143, Springer-Verlag, 1985.

[51] Feige, U., Fiat, A., and Shamir, A., "Zero Knowledge proofs of identity", *Journal of Cryptology*, Vol. 1, No. 2, pp. 77 - 94, 1988.

[52] Feistel, H., "Cryptography and Computer Privacy", *Scientific American*, Vol.228(5), pp.15-23,May 1973

[53] Feistel, H., Notz, W., and Smith, J.L., "Some cryptographic techniques for machine-to machine data communications", *Proceedings of the IEEE*, Vol. 63, NO. 11, November 1975, pp. 1545 - 1554.

[54] Fiat, A., and Shamir, A., "How to prove yourself", in *Advances in Cryptology - Proceedings of CRYPTO '86*, ed. A.M. Odlyzko, Lecture Notes in Computer Science, No.263, pp. 186 - 194, Springer-Verlag, 1987.

[55] Findlay, P. A., and Johnson, B. A., "Modular exponentiation using recursive sums of residues", in *Advances in Cryptology - Proceedings of CRYPTO '89*

[56] Fitzgerald, K., "Data Security", *IEEE Spectrum,* pp. 22-26, Aug. 1989.

[57] Fumy, W., "On the F-function of FEAL", in *Advances in Cryptology - Proceedings of EUROCRYPT '87*, ed. D. Chaum and W.L. Price, Lecture Notes in Computer Science, No. 304, Springer-Verlag, 1988.

[58] Gallay, P., and E. Depret, "A Cryptography Processor," *IEEE International Solid-State Circuits Conference*, pp. 148-149, Feb. 1988.

[59] Geffe, P.R., "How to protect data with ciphers that are really hard to break", *Electronics*, Jan. 1973, pp. 99 - 101.

[60] Goldreich, O., "Two remarks concerning the GMR signature scheme", in *Advances in Cryptology - Proceedings of CRYPTO '86*, ed. A.M. Odlyzko, Lecture Notes in Computer Science, No.263, pp. 278-301, Springer-Verlag, 1987.

[61] Goldwasser, S., Micali, S., and Rivest, R. L., "A digital signature scheme secure against adaptive chosen message attacks", *SIAM J. Computing*, vol. 17, pp. 281 - 308., April 1988.

[62] Gordon, J. A., "Strong RSA keys", *Electronics Letters*, Vol. 20, No. 12, pp. 514 -516, June 1984.

[63] Gordon, J. A., "Strong Primes are easy to find", in *Advances in Cryptology Proceedings of EUROCRYPT '84*, ed. T. Beth, N. Cot, and I. Ingemarsson, Lecture Notes in Computer Science, Springer-Verlag, 1985, pp. 216 - 223

[64] Hellman, M. E., "An extension of the Shannon theory approach to cryptography" IEEE Trans. on Info. Theory, vol. IT-23, pp. 289-294, May 1977

[65] Hellman, M. E., "DES will be insecure within ten years", *IEEE Spectrum*, July 1979, pp. 32 - 39.

[66] Hellman, M. E., "A Cryptanalytic Time-Memory Trade off", *IEEE Trans. on Info. Theory*, vol. IT-26, pp401-406, July 1980.

[67] Herlestam, T., "Critical remarks on some public-key cryptosystems", *BIT*, Vol. 18, 1978, pp. 493 - 496.

[68] Hoornaert, F., M. Decroos, J. Vandewalle, and R. Govaerts, Fast RSA-hardware : Dream or reality. Presented at *EUROCRYPT '88*

[69] Ivey, P. A., A. L. Cox, J. R. Harbridge, and J. K. Oldfield, "A single-chip public key encryption subsystem," *IEEE J. Of Solid-State Circuits*, vol. 24, pp. 1071-5, Aug. 1989.

[70] Jung, A., "Implementing the RSA Cryptosystem", *Computers and Security*, Vol. 6, 1987, pp 342 - 350.

[71] Kahn, D., *The Codebreakers, The Story of Secret Writing*, abridged ed., Signet, New York, NY, 1973.

[72] Kak, S. C., "Data security in computer networks", *Computer*, February 1983, pp. 8 - 10.

[73] Kaliski B. S., Rivest R. L., and Sherman A. T., "Is the Data Encryption Standard a group?", *Journal of Cryptology*, Vol. 1, No. 1, pp. 3 - 36, 1988

[74] Kawamura, S., and K. Hirano, "A Fast Modular Arithmetic Algorithm Using a Residue Table.", Presented at *EUROCRYPT '88*

[75] Knuth, D.E., *The Art of Computer Programming, Vol. 2, Semi-numerical algorithms*, 2nd ed., pp. 441-442, Addison-Wesley, Reading, MA, 1981.

[76] Kochanski, M., "Developing an RSA Chip," in *Advances in Cryptology - Proceedings of CRYPTO '85*, ed. H.C. Williams, Lecture notes in Computer Science, pp. 350-368, Springer-Verlag, 1986.

[77] Kochanski, M., "Split Key," *Systems International*, p. 103, Oct. 1986.

[78] Konfelder, L. M., "On The Signature Reblocking Problem in Public-Key Cryptosystems", *Comm. ACM*, Vol. 21(2) p.179, Feb. 1978.

[79] Laurichesse, D., "Mise en Oeuvre Optimisee du Chiffre RSA", *Rapport LAAS No. 90052*, Mars 1990

[80] Lenstra, A. K., and Lenstra Jr., H. W., "Algorithms in Number Theory", in *Handbook of Theoretical Computer Science.*

[81] Massey, J. L., "Feedback shift register synthesis and BCH decoding", *IEEE Trans. on Information Theory*, Vol. IT-15, pp. 122 - 127, Jan. 1969.

[82] Massey, J. L., "Probabilistic Encipherment", *E & M Journal of Österreichische Verband für Elektotechnik*, 1987

[83] Massey, J. L., "An Introduction to Contemporary Cryptology," *Proc. IEEE*, vol. 76, pp. 533-549, May 1988.

[84] Massey, J. L., and Reuppel, R. A., "Linear ciphers and random sequence generators with multiple clocks", in *Advances in Cryptology Proceedings of EUROCRYPT '84*, ed. T. Beth, N. Cot, and I. Ingemarsson, Lecture Notes in Computer Science, Springer-Verlag, 1985.

[85] McCurley, K. S., "A Key Distribution System Equivalent To Factoring", *Journal of Cryptology*, Vol. 1., No. 2., pp95 -105, 1988.

[86] McEliece, R. J., "A public key cryptosystem based on algebraic coding theory", JPL DSN Progress Report 42 - 44, pp. 114 - 116, Jet Propulsion Labs, Pasedena Ca. Jan. - Feb. 1978

[87] Merkle, R. C., "Secure Communications Over Insecure Channels", *Comm. ACM*, Vol. 21, No. 4, pp. 294-299, April. 1978

[88] Merkle, R. C, "Hiding information and signatures in trap door knapsacks", *IEEE Transactions on Information Theory*, Vol. IT-24, No. 5, September 1978, pp. 525 - 530.

[89] Meyer, C.H., and S.M. Matyas, *Cryptography : A new dimension on computer data security,* Wiley, New York, 1982.

[90] Micali, S., and Shamir, A., "An improvement of the Fiat-Shamir identification and signature scheme", presented at *CRYPTO '88*

[91] Michelman, E. H., "The Design and Operation of Public Key Cryptosystems", *NCC*, ,Vol. 48, pp. 305 - 311, 1979.

[92] Miyaguchi, S., "Fast Encryption Algorithm for the RSA Crypto- graphic System," *Proc. COMPCON '82*, pp. 672 - 8, 1982.

[93] Mohan, S.B., and B.S. Adiga, "Fast algorithms for implementing RSA public key cryptosystem," *Electronics Letters*, vol. 21, p. 761, Aug. 1985.

[94] Montgomery, P.L., "Modular multiplication without trial division", *Mathematics of Computation*, Vol. 44, No. 170, April 1985, pp 519 - 521.

[95] Moore, J., H., "Protocol Failures in Cryptosystems", *Proc. IEEE*, vol. 76, pp. 594-602, May 1988.

[96] Morita, H., "A fast Modular multiplication algorithm based on a Higher Radix", in *Advances in Cryptology: Proceedings of CRYPTO 89.*

[97] Müller-Schloer, C., "A Microprocessor-based Cryptoprocessor", *IEEE MICRO*, Oct. 1983, pp 5 - 15.

[98] Newman, D. B. Jr., Omura, J., and Pickholtz, R. L., "Public Key Management for Network Security", *IEEE Network Magazine*, Vol. 1, No. 2, April 1987

[99] Norris, M. J., and Simmons, G. J., "Algorithms for high speed modular arithmetic", *Congressus Numeratium*, Vol. 31, pp. 153 - 163, 1981

[100] Ohta, K., and Okamoto, T., "A modification of the Fiat-Shamir scheme", presented at *CRYPTO '88*

[101] Okamoto, T., "A fast signature scheme based on congruential polynomial operators", *IEEE Transactions on Information Theory*, Vol. IT-36, No. 1, January 1990, pp. 47 - 53.

[102] Ong, H., Schnorr, C. P., and Shamir, A., "An efficient signature scheme based on quadratic equations", *Proc. sixteenth STOC*, 1984, pp. 208 - 216.

[103] Orton, G.A., L.E. Peppard, and S.E. Tavares, "A Fast Asynchronous RSA Chip," *IEEE Custom Integrated Circuits Conference*, pp. 439-443, Rochester, NY, May 1986.

[104] Orton, G.A., M.P. Roy, P.A. Scott, L.E. Peppard, and S.E. Tavares, "New results in mapping data encryption algorithms into VLSI," Presented *4th Int. Workshop on VLSI in Communications*, Ottowa, June 1986.

[105] Orton, G.A., M.P. Roy, P.A. Scott, L.E. Peppard, and S.E. Tavares, "VLSI Implementation of public-key encryption algorithms," in *Advances in Cryptology - Proceedings of CRYPTO '86*, ed. A.M. Odlyzko, Lecture Notes in Computer Science, No.263, pp. 278-301, Springer-Verlag, 1987.

[106] Pailles, J.C., and M. Girault, "The security processor C.R.I.P.T.," *Fourth IFIP security on information systems security*, Dec. 1986.

[107] Pohlig, S.C. , and Hellman, "An Improved algorithm for computing logarithms in GF(p) and its cryptographic significance.", *IEEE Trans., on Information Theory*, Vol. IT-24, Jan. 1978, pp. 106 - 111

[108] Pollard, J. M., "An efficient solution of the congruence $x^2 + y^2 = m \pmod n$", *IEEE Trans. on Information Theory*, Vol. IT-33, pp 702 - 709, Sept. 1987

[109] Pomerance, C., "Fast, rigorous factorisation and discrete logarithm algorithms", in *Discrete Algorithms and Complexity*, ed. D. S. Johnson et. al., New York, NY: Academic Press, 1987, pp 119 - 143.

[110] Price, W.L., "Standards for data security - a change of direction," in *Advances in Cryptology - Proceedings of CRYPTO '87*, ed. C. Pomerance, Lecture Notes in Computer Science, No. 293, pp. 3-8, Springer-Verlag, 1988.

[111] Quisquater, J. -J., and Couvreur, C., "Fast decipherment algorithm for RSA Public Key Cryptosystem", *Electronics Letters*, Vol. 18, No. 21, October 1982, pp. 905 - 907.

[112] Rabin, M. O., "Digitized signatures and public key functions as intractable as factorization", MIT Laboratory for Computer Science, MIT/LCS/TR-212, Jan. 1979

[113] Rankine, G., "THOMAS - A complete Single Chip RSA Device," in *Advances in Cryptology - Proceedings of CRYPTO '86*, ed. A.M. Odlyzko, Lecture Notes in Computer Science, No.263, pp. 480-487, Springer-Verlag, 1987.

[114] Rieden, R.F., J.B. Snyder, R.W. Widman, and W.J. Barnard, "A two chip implementation of the RSA public-key encryption algorithm," *1982 Government Microcircuit Applications Conference*, pp. 24 - 7, Orlando, FL, Nov. 1982.

[115] Rivest, R.L., "Remarks on a Proposed Cryptanalytic Attack of the M.I.T. Public Key Cryptosystem", *Cryptologia*, Vol. 2(1), pp. 62 - 65, January 1978.

[116] Rivest, R.L., "Critical remarks on 'Critical remarks on some public-key cryptosystems' by T. Herlestam", BIT, Vol. 19, 1979, pp. 274 - 275.

[117] Rivest, R.L., "A Description of a Single-Chip Implementation of the RSA Cipher," *Lambda*, vol. 1, pp. 14-18, Fourth Quarter 1980.

[118] Rivest, R.L., "A short report on the RSA chip," in *Advances in Cryptology - Proceedings of CRYPTO '82*, ed. A.T. Sherman, p. 327, Plenum Press, 1983.

[119] Rivest, R.L., "RSA Chips (Past/Present/Future)," in *Advances in Cryptology Proceedings of EUROCRYPT '84*, ed. T. Beth, N. Cot, and I. Ingemarsson, Lecture Notes in Computer Science, pp. 159-165, Springer-Verlag, 1985.

[120] Rivest, R.L., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Comm. ACM*, vol. 21, pp. 120-126, Feb. 1978.

[121] Roy, M.P., L.E. Peppard, and S.E. Tavares, "A CMOS Bit-Slice Implementation of the RSA Public Key Encryption Algorithm," *1985 Canadian Conference on VLSI*, pp. 52-56, Toronto, Nov. 1985.

[122] Rueppel, R. A., *Analysis and Design of Stream Ciphers*, Springer-Verlag, Hiedelberg and New York, 1986.

[123] Rueppel, R. A., and Staffelbach, "Products of linear recurring sequences with maximum complexity", *IEEE Trans., on Information Theory*, Vol. IT-33, Jan. 1987, pp. 124 - 131.

[124] Scott, P.A., S.E. Tavares, and L.E. Peppard, "A Fast VLSI Multiplier for GF(2**n)," *IEEE J. On selected areas in comm.*, vol. SAC-4, pp. 62-66, Jan. 1986.

[125] Seberry, J., and Pieprzyk, J., "Cryptography: An Introduction to Computer Security", Prentice Hall, 1989

[126] Sedlak, H., "The RSA Cryptography Processor," in *Advances in Cryptology - Proceedings of EUROCRYPT '87*, ed. D. Chaum and W.L. Price, Lecture Notes in Computer Science, No. 304, pp. 95-105, Springer-Verlag, 1988.

[127] Selim, G. A., "Digital Signatures: A tutorial survey", *Computer*, February 1983, pp. 15 - 24.

[128] Serpette, B., Vuillemin, J., and Hervé, J-C., "BigNum: A Portable and Efficient Package for Arbitrary-Precision Arithmetic", DEC Paris Research Laboratory Report, May 1989.

[129] Shamir, A., "A polynomial time algorithm for breaking the Merkle-Hellman cryptosystem", *IEEE Transactions on Information Theory*, Vol. IT-30, Sept. 1984, pp. 699 - 704.

[130] Shand, M., Bertin, P., Vuillemin, J., "Resource tradeoffs in fast long integer multiplication (Extended Abstract)", DEC Paris Research Laboratory Report, Jan. 1990.

[131] Shannon, C.E., "A Mathematical Theory of Communication", *Bell Syst. Tech. J.*, vol. 27, pp. 379-423, and pp. 623-656, July and Oct., 1949.

[132] Shannon, C.E., "Communication Theory of Secrecy Systems," *Bell Syst. Tech. J.*, vol. 28, pp. 656-715, Oct. 1949.

[133] Shimizu, A., and Miyaguchi, S., "Fast Data Encipherment Algorithm FEAL", in *Advances in Cryptology - Proceedings of EUROCRYPT '87*, ed. D. Chaum and W.L. Price, Lecture Notes in Computer Science, No. 304, pp. 267-278, Springer-Verlag, 1988.

[134] Siegenthaler, T., "Correlation immunity of nonlinear combining functions for cryptographic applications", *IEEE Transactions on Information Theory*, Vol. IT-30, Oct. 1984, pp. 776 - 780.

[135] Siegenthaler, T., "Decrypting a class of stream ciphers using ciphertext only", *IEEE Transactions on Computers*, Vol. C-34, Jan. 1985, pp. 81 - 85.

[136] Simmons, G.J., "Authentication without secrecy : A secure communications problem uniquely solvable by asymmetric encryption techniques," *IEEE EASCON '79*, pp. 661 - 2, Washington D.C., Oct. 1979.

[137] Simmons, G.J., "Authentication Theory/Coding theory", in *Advances in Cryptology: Proceedings of CRYPTO 84*, ed. G.R. Blakley and D. Chaum, Lecture Notes in Computer Science, No. 196, pp. 411 - 431, Springer-Verlag, 1985.

[138] Simmons, G.J., Editorial Comment, *Proc. IEEE*, vol. 76, pp. 515-518, May 1988.

[139] Simmons, G.J., "A Survey of Information Authentication", *Proc. IEEE*, vol. 76, pp. 603 - 620, May 1988.

[140] Simmons, G.J., "How to Insure that Data Acquired to Verify Treaty Compliance are Trustworthy", *Proc. IEEE*, vol. 76, pp. 621 -627, May 1988.

[141] Simmons, G.J., and Norris, J. N., "Preliminary comments on the M.I.T. Public Key Cryptosystem", *Cryptologia*, Vol. 1(4), pp. 406 - 414, October 1977.

[142] Simmons, D.E., and Tavares, S. E., "An NMOS Implementation of a Large Number Multiplier for Data Encryption Systems," *Proc. 1983 Custom Integrated Circuits Conf.*, pp. 262-266, Rochester, NY, May 1983.

[143] Smid, E. M., and Branstad, "The Data Encryption Standard: Past and Future", *Proc. IEEE*, vol. 76, pp. 550-559, May 1988.

[144] Soderstrand, M. A., Jenkins, W. K., and Jullien, G. A., editors, *Residue Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, 1986.

[145] Stoll, C., "The Cuckoo's Egg", The Bodley Head, 1989

[146] Taylor, F. J., "Large VLSI modulii multipliers," *Proc. IEEE Symp. Circuits and Systems,* vol. Pt. 1, pp. 379 - 383, April 1980.

[147] Tuchman, W. "Hellman presents no short-cut solutions to the DES", *IEEE Spectrum*, July 1979, pp. 40 - 41.

[148] van Tilborg, H. C. A., *An Introduction to Cryptology*, Kluwer Academic Publishers, 1988

[149] Verbauwhede, I., F. Hoornaert, J. Vandewalle, and H. De Man, "Security Considerations in the Design and Implementation of a New DES Chip," in *Advances in Cryptology*

- *Proceedings of EUROCRYPT '87*, ed. D. Chaum and W.L. Price, Lecture Notes in Computer Science, No. 304, pp. 287-300, Springer-Verlag, 1988.

[150] Vernam, G. S., "Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications," *Journal of American Institute of Electrical Engineers*, Vol. XLV, pp109-115, 1926.

[151] Wilkes, M.V., *Time-Sharing Computer Systems*, American Elsevier, 1972

[152] Williams, H. C., "A Modification of the RSA Public Key Encryption Procedure", *IEEE Trans. on Information Theory*, Vol. IT-26, No. 6, November 1980, pp. 726 - 729.

[153] Williams, H. C., and Schmid, B.,"Some remarks concerning the M.I.T public key cryptosystem", *BIT*, vol.19, pp. 525 - 538, 1979

Appendix C. Author's Publications

[1] Tomlinson, A., "A Bit-Serial Modulo Multiplier", *Electronics Letters*, Vol. 25, No. 24, pg. 1664, November 1989.

[2] Tomlinson, A., "Modulo Multiplier to enhance encryption rates", *Electronic Engineering*, Vol. 62, No. 760, April 1990, pg. 25

These articles have been photocopied and included in this thesis with permission from the IEE and Electronic Engineering.

BIT-SERIAL MODULAR MULTIPLIER

Indexing terms: Information theory, Signal processing, Mathematical techniques, Modular arithmetic, Data encryption

A bit-serial modular multiplier is presented which uses a table look-up method to perform modular reduction. Since the clock frequency is independent of word length, this design is most useful when dealing with large integers, and is required by many modern cryptographic systems.

Introduction: One of the most interesting developments in the field of cryptology is that of public-key encryption.[1] In this scheme the cipher has two separate keys, one for encryption and a second for decryption. The encryption key can be stored in a public directory, allowing anyone to encrypt messages, which can then only be deciphered by the intended recipient who holds the decryption key. Although several public-key algorithms have been proposed, the predominant encryption technique in this area is the RSA[2] system. This algorithm is based on the modular exponentiation of very large integers, typically 512 bits long or more.

When this system is implemented on general-purpose machines, the resulting data rates are disappointing in comparison with those obtained by conventional secret-key techniques. For example, a 512-bit modular exponentiation may take up to 30 s to complete on a 68000,[3] or 2·5 s on a TMS32010.[4] To overcome this problem dedicated hardware is needed to carry out modular arithmetic on large integers.

Reviews of existing hardware[5,6] reveal that many designs perform modulator multiplication using ripple adders for multiplication and reduction. The carry propagation time in such designs becomes a limiting factor as the word length increases. One notable exception is the bit-serial design proposed by Brickell[7] in 1982, which has been reported as performing 512-bit modular exponentiation at a rate of 25 kbit/s. The design proposed here is also bit-serial, but differs from Brickell's in the way modulo reduction is performed.

Multiplication procedure: Modulator multiplication is performed most significant bit first according to the add-shift-reduce procedure described by Blakley[8] in 1983, but with the following modifications:

(1) The intermediate product is allowed to grow by two bits each cycle.

(2) At the end of the cycle, these upper bits are reset to zero.

(3) The residue corresponding to the two reset bits is added to the intermediate product on the next cycle.

The benefit of this approach is that it eliminates the need to compare the intermediate product with the modulus to perform modulo reduction. The operation simply involves the decoding of two bits to select the appropriate residue from a look-up table. That the intermediate reduction may be incomplete, in that after resetting the upper bits the remaining number may be greater than the modulus, is of little practical consequence. Once the multiplication has ended, reduction is completed by subtracting the modulus, but because the word length has been constrained to two bits of growth, no more than seven subtractions of the modulus will be needed to do this.

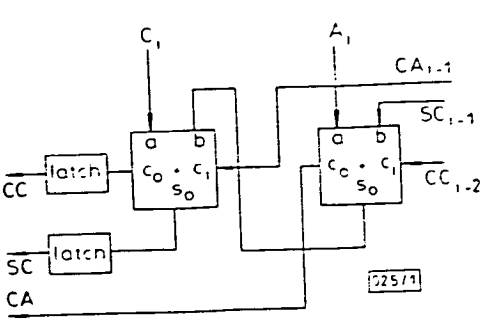

Fig. 1 *Multiplier cell*

Hardware design: The basic multiplier cell to compute $A * B$ modulo N can be seen in Fig. 1. The multiplier B is examined most significant bit first and the first adder adds the multiplicand A to the array if the bit is set. If the bit is not set then zero is added. The second adder then adds the residue C_i selected from the look-up table, and outputs the sum and carry to two latches. Fig. 2 shows how five basic cells are cascaded to form a 5-bit modular multiplier. Three registers are needed to store the residues, and an adder and accumulator to add the sums and carries at the end of the multiplication and subtract the modulus N to complete the reduction. Once the sums and carries stored in the array have been added, the next multiplication can proceed in parallel with the subtractions.

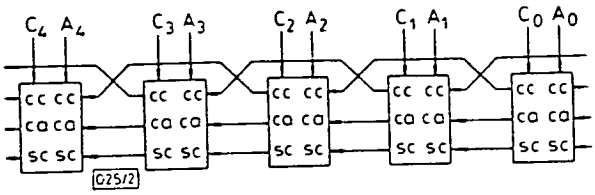

Fig. 2 *Five-bit multiplier array*

Performance estimation: Since the final subtractions can be carried out in parallel with the next multiplication, the time taken to complete an N-bit modulator multiply is simply N clock cycles. The bit-serial nature of this design means that the clock frequency will be independent of the word length and limited only by the delay through a single cell. Thus the time for an N-bit exponentiation using the square and multiply algorithm,[9] with concurrent squaring and multiplying, will be $N^2 * \delta$, where δ is the delay through one cell. Assuming a delay of roughly 40 ns through each cell, the time for a 512-bit exponentiation will be 10 ms and the data rate 50 kbit/s.

A modular arithmetic ASIC is currently being designed using this technique, and prototypes are expected to be tested within the next few months.

Summary: An architecture for bit-serial modulator multiplication has been presented which uses a look-up table to perform modulo reduction. It is estimated that this structure can achieve data rates of up to 50 kbit/s for 512-bit modular exponentiation, and a semicustom IC is currently being fabricated to test the design.

A. TOMLINSON

9th October 1989

Department of Electrical Engineering
University of Edinburgh
King's Buildings, Edinburgh EH9 3JZ, United Kingdom

References

1 DIFFIE, W., and HELLMAN, M. E.: 'New directions in cryptography', *IEEE Trans.*, 1976, **IT-22**, pp. 644–654

2 RIVEST, R. L., SHAMIR, A., and ADLEMAN, L.: 'A method for obtaining digital signatures and public key cryptosystems', *Commun. ACM*, 1978, **21**, pp. 120–126

3 RANKINE, G.: 'THOMAS—a complete single chip RSA device', *in* ODLYZKO, A. M. (Ed.): 'Advances in cryptology—Proc. of Crypto '86' (Lecture Notes in Comput. Science, Springer-Verlag, 1987, **263**), pp. 480–487

4 BARRETT, P.: 'Implementing the Rivest, Shamir and Adleman public key encryption algorithm on a standard digital signal processor'. Ibid., 1987, pp. 311–323

5 RIVEST, R. L.: 'RSA chips (past/present/future)', *in* BETH, T., COT, N., and INGEMARSSON, I. (Eds.): 'Advances in cryptology—Proc. Eurocrypt 84' (Lecture Notes in Comput. Sci., Springer-Verlag, 1985), pp. 159–165

6 DIFFIE, W.: 'The first ten years of public-key cryptography', *Proc. IEEE*, 1988, **76**, pp. 560–577

7 BRICKELL, E. F.: 'A fast modular multiplication algorithm with application to two key crytography', *in* SHERMAN, A. T. (Ed.): 'Advances in cryptology—Proc. Crypto '82' (Plenum Press, 1983), pp. 51–60

8 BLAKLEY, G. R.: 'A computer algorithm for calculating the product AB modulo M', *IEEE Trans.*, 1983, **C-32**, pp. 497–500

9 KNUTH, D. E.: 'Semi-numerical algorithms', *in* 'The art of computer programming, vol. 2, 2nd edn.' (Addison-Wesley, Reading, MA, 1981), pp. 441–442

Modulo multiplier to enhance encryption rates

he use of public key encryption to encode digital communications has advantages in erms of providing data security. The RSA technique [1] is equently used but has the isadvantage that it relies on odular exponentiation, and ence multiplication, of large tegers of the order of 512 bits. Although dedicated RSA ardware is available, the encryption rates from these implementations are of the order of 5-20kbits/s. Alan Tomlinson at he University of Edinburgh has eveloped an improved algorithm for bit-serial modulo multiplication, using a look-up table o perform partial modulo eduction, which it is claimed as the potential to improve on hese rates.

Blakley [2] has explained hat for modulo multiplication it s possible to reduce the partial products as they are formed. his has the advantage of voiding word growth and the inal time-consuming division process to find the residue. It does however requre large ook-up tables or the use of division to determine the residues at the partial product stage.

Tomlinson's refinement is to ntain word growth at the partial product stage to two bits at most. Instead of completely reducing the partial product, the residues implied by the upper two overflow bits are added at the next cycle of bit-serial multiplication.

Only three possible residues are implied by the upper bits and this allows them to be loaded into a small look-up table at the time of modulo selection.

The fact that the partial products are not completely reduced is not of practical significance. What is of significance is that the result is limited to the length of the multiply array and an n-bit modulo multiply will return an n-bit result in n-cycles.

The basic cell to compute the i-th bit of A*B modulo N is shown in figure 1. The first adder conditionally adds its position bit of the multiplicand A. The second adder then adds the position bit of the residue C selected by the overflow bits from the previous cycle. The outputs are sent to sum and carry latches.

These results are used at the i+1 and i+2 bit multiplier cells respectively in the next cycle (the next MSB of the multiplier).

Such cells can be cascaded to form an n-bit multiplier array where all cells operate in parallel, left shifting partial results at each cycle.

Figure 2 shows an IC architecture which has the multiplier array at its core.

The hardware description language ELLA was used to model both a single system and a cascade of four ICs with this architecture. Subsequently a size was selected for the register and multiplier array length (32bit) and a semi-custom device designed and fabricated using SOLO 1200, the ASIC design tool from ES2. The result was a 64,000 transistor IC implemented in 2µm CMOS technology and measuring 8.77mm × 8.38mm.

To date the IC has been tested at up to 5MHz which should, where devices are cascaded, yield 5Mbit/s throughput for multiplication or 5/n Mbit/s for n-bit exponentiation. This translates to 10kbits/s for 512bit exponentiation. If the frequency of operation can be increased to 25MHz then 50kbits/s transmission rates could be obtained. Also a full custom approach to IC design is expected to produce a more efficient layout making 64-or 128bit register and array lengths viable.

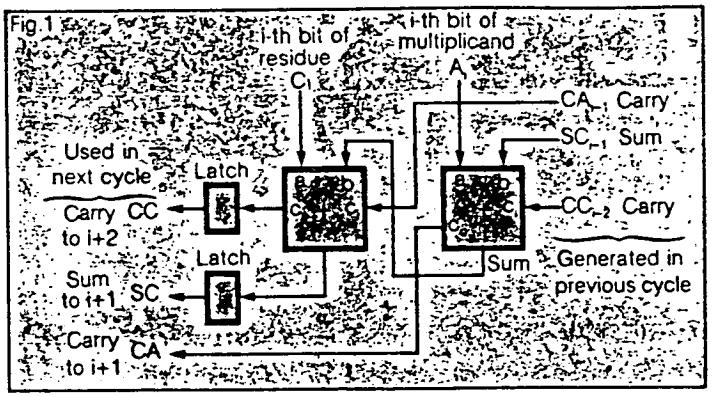

Figure 1: Single bit multiplier cell

For further information contact Mr A Tomlinson, Department of Electrical Engineering, University of Edinburgh, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JL Tel 031 668 1550 ext. 219

References

[1] Rivest, Shamir, and Adleman 'A method for obtaining digital signatures and public key cryptosystems', Commun ACM, 1978, 21 pp120-126

[2] Blakley 'A computer algorithm for calculating the product AB modulo M', IEEE Trans., 1983, C-32, pp497-500

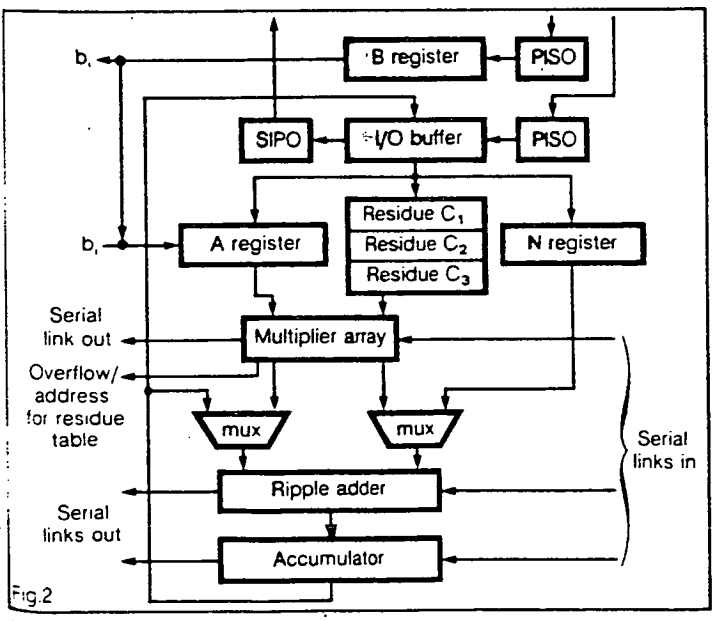

Figure 2: Modulo multiplier IC architecture

Appendix D. Index