P-3059

FPGA IMPLEMENTATION OF ENCRYPTION/DECRYPTION USING RC6 ALGORITHM

A PROJECT REPORT

Submitted by

MADHIVANAN.N	71206106027
SURESH BALAJI.J	71206106052
VIMAL INIYAN.T.P	71206106059
ESWARAMOORTHI.P	71206106301

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2010

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"FPGA IMPLEMENTATION OF ENCRYPTION/DEDRYPTION USING RC6 ALGORITHM"** is the bonafide work of **"MADHIVANAN.N, SURESH BALAJI.J, VIMALINIYAN.T.P, ESWARAMOORTHI.P"** who carried out the project work under my supervision.

[illegible]	[illegible]
SIGNATURE	**SIGNATURE**
Dr. RAJESWARI MARIAPPAN, Ph.D.	Ms.M.ALAGUMEENAAKSHI, M.E
HEAD OF THE DEPARTMENT	**SUPERVISOR** **SENIOR LECTURER**
Electronics and Communication Engineering Kumaraguru College of Technology Coimbatore – 641006.	Electronics and Communication Engineering Kumaraguru College of Technology Coimbatore – 641006.

The candidates with university register numbers **71206106027, 71206106052, 71206106059, 71206106301** were examined by us in the project viva-voce examination held on 15.04.2010

[illegible]
15.04.2010
Internal Examiner

[illegible]
External Examiner

ACKNOWLEDGEMENT

We acknowledge our thanks to **Dr.J.SHANMUGAM Ph.d., DIRECTOR,** Kumaraguru College of Technology, Coimbatore, for providing an opportunity to take up this project.

We acknowledge our thanks to **Dr. RAMACHANDRAN, Ph.D., PRINCIPAL**, Kumaraguru College of Technology, Coimbatore, for providing an opportunity to take up this project.

We take this opportunity to thank, **Dr. RAJESWARI MARIAPPAN, M.E., Ph.D., HEAD OF THE DEPARTMENT**, Electronics and Communication Engineering, Kumaraguru College of Technology, Coimbatore, for her extended support in completion of the project.

We wish to express our sincere thanks and deep sense of gratitude to **Prof. A.VASUKI, M.E., PROJECT COORDINATOR**, Department of Electronics and Communication Engineering, Kumaraguru College of Technology, Coimbatore, for her wise counsel with timely and inspiring directions during this project work.

We take this opportunity to express our profound thanks and gratitude to **Ms. M.ALAGUMEENAAKSHI, M.E., PROJECT GUIDE,** Department of Electronics and Communication Engineering, Kumaraguru College of Technology, Coimbatore, for her valuable guidance and support for the successful completion of the project work.

We wish to express our sincere thanks to **Ms. AMIRTHA GOWRI, M.E., CLASS ADVISOR,** Department of Electronics and Communication Engineering, Kumaraguru College of Technology, for her extended support in completion of the project.

Finally, we express our thanks to our parents and friends for their encouragement in completion of this project work successfully.

ABSTRACT

Due to the growing need for accessibility to huge amount of data, integrity of data has become essential. RC6 algorithm is one of the most simple and secured way to protect data when compared to AES, DES, Triple DES and other cryptographic algorithm. This project aims at implementing RC6 algorithm on Field Programmable Gate Array. The project comprises of encryption, decryption, Diffusion and key expansion. Encryption using RC6 is an iterative process involving multiple passes, by the end of which the input data evolves into a cipher text. As RC6 is a symmetric block cipher, decryption involves the same step as in encryption albeit in the reverse order. Each pass of encryption and decryption involves the use of unique key. To facilitate this, the input key is expanded into a larger one in the key expansion module. To increase data dependency Diffusion multiplier function is used.

The work includes coding in VHDL, followed by simulation using modelsim and synthesis using Xilinx ISE. Implementation in hardware level can be done in both Virtex and Spartan series of Xilinx family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

DES	Data Encryption Standard
AES	Advanced Encryption Standard
NIST	National Institute of Standards and Technology
RTL	Register Transfer Level
FPGA	Field Programmable Gate Array
VHDL	VHSIC Hardware Description Language

CHAPTER 1

INTRODUCTION

1.1 CRYPTOGRAHY

In today's electronic age, the importance of cryptography in securing electronic data transaction is unquestionable. Everyday, users electronically generate and communicate a large volume of information with others. This information includes medical, financial and legal files, internet banking, phone conversations and e-commerce transactions. By using various cryptography techniques, people and organizations can secure electronic information to protect economic interest, prevent fraud and guarantee the privacy of individuals.

Cryptography is a set of protocols, algorithms and techniques providing security for digital information by encoding and decoding the information using a specific key. The cryptography key controls both the encryption and decryption processes. The two basic objective of cryptography are:

- Privacy to prevent the unauthorized disclosure of data
- Authenticity to prevent the unauthorized modification of data.

Cryptography is the science and study of creating and using systems for communicating in secrecy via communication channels that are not secure. The cipher process has been illustrated in Fig1.1. A sender maintains secrecy by transmitting data, in a plaintext into unintelligible form, known as cipher text, in a process known as encryption or encipherment. The receiver recovers the original plaintext using the inverse process of converting cipher text into plaintext and this process is known as decryption or decipherment.

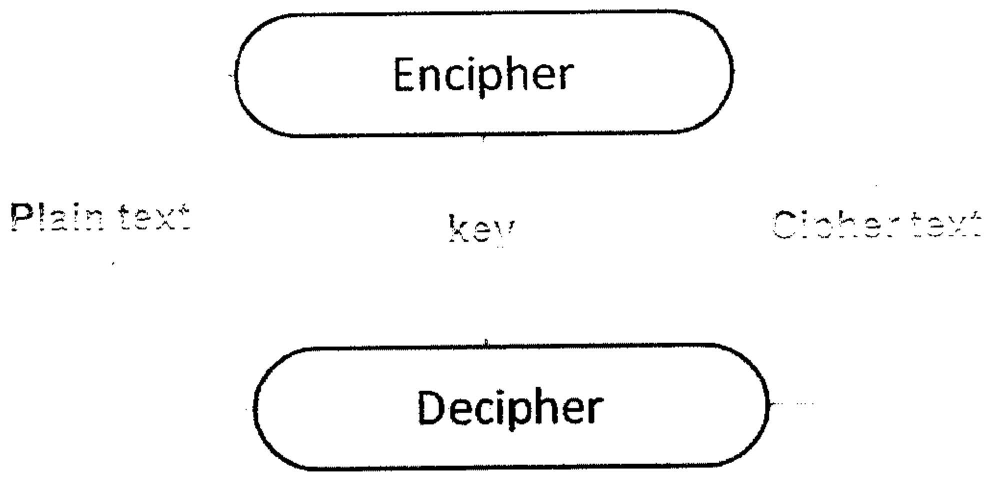


Fig 1.1 Cipher process

Cryptography is required to protect information in wireless network, Digital communication, internet etc. from being intercepted and stolen by an unwanted third party with the use of keys. This project seeks to implement the RC6 cryptographic algorithm in VHDL and provide a simple, robust implementation of RC6 algorithm in hardware. A hardware implementation of RC6 algorithm would be a powerful tool for any mobile device or smart cards or any technology requiring strong encryption. The overall design is an incredibly fast, efficient RC6 algorithm implementation suitable for an abundant of applications.

1.2 CRYPTOLOGY

Cryptology is the science which incorporates both cryptography and cryptanalysis. Fig 1.2 illustrates the cryptology procedure below.

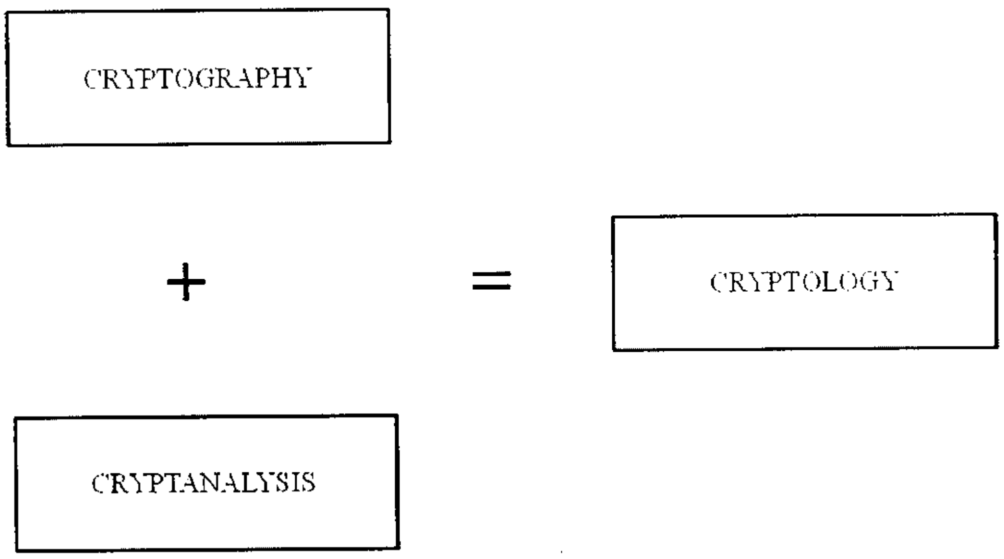


Fig1.2 Cryptology process

1.2.1 CRYPTANALYSIS

Cryptanalysis is the study of taking encrypted data, and trying to decrypt it without use of the key. The other side of cryptography, cryptanalysis is used to break codes by finding weaknesses within it. In addition to being used by hackers with bad intentions, cryptanalysis is also often used by the military. Cryptanalysis is also appropriately used by designers of encryption systems to find, and subsequently correct, any weaknesses that may exist in the system under design.

Cryptanalysis refers to the study of ciphers, cipher text, or cryptosystems (that is, to secret code systems) with a view to finding weaknesses in them that will permit retrieval of the plaintext from the cipher text, without necessarily knowing the key or the algorithm. This is known as *breaking* the cipher, cipher text, or cryptosystem.

Successful cryptanalysis is a combination of mathematics, inquisitiveness, intuition, persistence, powerful computing resources.

1.3 HACKERS ATTACK AND CRYPTANALYSIS

Cryptanalysis is the science and study of breaking the ciphers. A cipher can be broken if it is possible to determine the plain text or key from the cipher text, or if someone can determine the key from plain text –cipher text pairs. The attempt to find the key of encryption algorithm by brute force requires a very long time. Longer the key length more immune is the key against brute force attack. A Cryptanalysis attack is an attack by an intruder trying to discover the contents of a straightforward random attack.

There are numerous techniques for performing cryptanalysis, depending on what access the cryptanalyst has to the plaintext, cipher text, or other aspects of the cryptosystem. Below are some of the most common types of attacks:

- Known-plaintext analysis
- Chosen-plaintext analysis (also known as differential cryptanalysis)
- Cipher text-only analysis
- Man-in-the-middle attack
- Timing/differential power analysis

Triple DES with 168-bit key length has become more popular. The Triple DES algorithm provides high security and has been proven to be immune to hacker attacks. However, it seems to have the following drawbacks:

- The algorithm is sluggish
- It uses 64-bit block size. For efficiency and security, a larger block size is desirable.

Comparatively, RC6 offers a high level of security, more efficiency and less computational complexity. Therefore it is fast replacing its encryption predecessors. Due to low memory and power consumption, simplicity and reliability RC6 algorithm is chosen from symmetric encryption algorithm for implementation. Also the use of diffusion multiplier function increase data dependency to increase its security.

1.4 DESIGN OBJECTIVE OF BASE ALGORITHM RC5

RC5 was designed with the following objectives in mind

RC5 is a symmetric block cipher. The same secret cryptographic key is used for encryption and for decryption. The plaintext and cipher text are fixed-length bit sequences (blocks).

RC5 is suitable for hardware and software implementations. This means that RC5 use only computational primitive operations commonly found on typical microprocessor

RC5 is a fast cryptographic algorithm. This more-or-less implies that RC5 is word oriented. The basic computational operations are based on the operators that work on full words of data at a time.

RC5 is adaptable to processors of different word -lengths. For example as 64-bit processors become available, it should be possible for RC5 to exploit their longer word length. Therefore, the number w of bits in a word is a parameter of RC5, different choices of this parameter result in different RC5 algorithms.

RC5 is iterative in structure; with a variable number of rounds .The user can explicitly manipulate the trade-off between higher speed and higher security. The number of rounds r is a second parameter of RC5.

RC5 can have a variable length cryptographic key. The user can choose the level of security appropriate for the desired application, or as required by external considerations such as export restrictions .The key length b (in bytes) is thus a third parameter of RC5.

RC5 is simple. It is easy to implement. More importantly a simpler structure is perhaps more interesting to analyze and evaluate, so that the cryptographic strength of RC5 can be more rapidly determined.

RC5 have a low memory requirement, so that it may be easily implemented on smart cards or other devices with restricted memory.

RC5 provides high security when suitable parameter values are chosen.

1.4.1 DERIVING RC6 FROM RC5

The RC6 algorithm is based on RC5 algorithm.RC6 algorithm have all the advantages and design objective same as RC5 algorithm. With the base of RC5 algorithm in consideration, RC6 algorithm was designed with more security, speed and others aspects of AES evaluation criteria. RC6 algorithm can be obtained from RC5 algorithm by simple steps.

1. Parallel execution of two RC5 algorithms.
2. Mixing up data between the copies.
3. Adding Pre- and Post-Whitening keys.

The above steps are done and final RC6 algorithm is obtained. The two parallel computation of RC5 algorithm results in RC6 algorithm. The Pre and Post whitening is added to increase data dependency making more complicated for the hackers to hack the algorithm. Very high security better than all other cryptographic algorithm can be obtained by choosing number of rounds as r as 20.

CHAPTER 2

FUNCTIONS OF ENCRYPTION AND DECRYPTION

2.1 ENCRYPTION AND DECRYPTION

The process of converting plain text messages into cipher text message is called as Encryption. Fig 2.1 illustrates the idea.

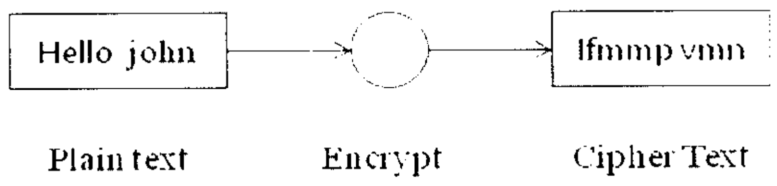


Fig 2.1: Encryption

The reverse process of transforming cipher text messages back to plain text messages is called as Decryption. Fig 2.2 illustrates the idea.

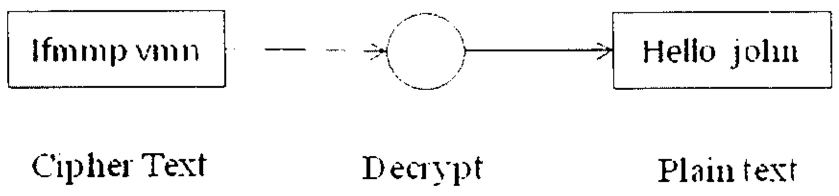


Fig 2.2 Decryption

2.2 ENCRYPTION AND DECRYPTION IN PRACTICAL

In computer-to-computer communications, the computer at the sender's end usually transforms a plain text messages into cipher text by performing encryption. The encrypted cipher text message is then sent to the receiver over a network (Such as internet). The receiver's computer then takes the encrypted message, and performs the reverse operation of encryption, i.e. it performs the decryption process to obtain the original plain text message. This is shown in Fig 2.3.

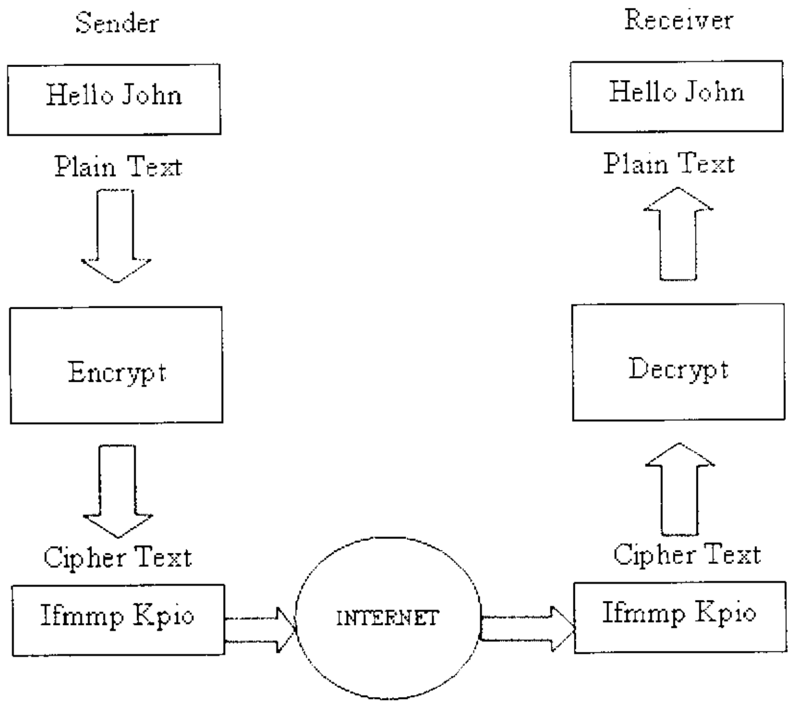


Fig 2.3: Encryption and Decryption in the real world

Every encryption and decryption process has two aspects: the algorithm and the key used for encryption and decryption. This is shown in Fig 2.4.

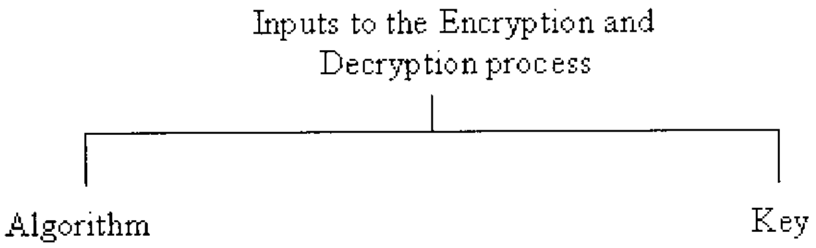


Fig 2.4: Aspects of encryption and decryption

To encrypt a plain text message, the sender performs encryption, i.e. applies the encryption algorithm. To decrypt a received encrypted message, the recipient performs decryption, i.e. applies the decryption algorithm. Clearly, the decryption algorithm must be the same procedure as the encryption algorithm in reverse order. Otherwise, decryption would not be able to retrieve the original message. Thus, the sender and the receiver must agree on a common algorithm for any meaningful communication to take place. The algorithm basically takes one text as input and produces another as the output.

The second aspect to be concentrated while performing encryption and decryption of messages is the key. As long as the sender and the receiver know the key, no one except the sender and receiver can do anything with the message. In general, the algorithm used for encryption and decryption processes is usually known to everybody. However, it is the key used for encryption and decryption that makes the process of cryptography secure.

Broadly, there are two cryptographic mechanisms, depending on what keys are used. If the same key is used for encryption and decryption, we call the mechanism as Symmetric key cryptography. However, if two different keys are used in a cryptographic mechanism, wherein one key is used for encryption, and another, different key is used for decryption; we call the mechanism as Asymmetric Key Cryptography. This is shown in Fig 2.5.

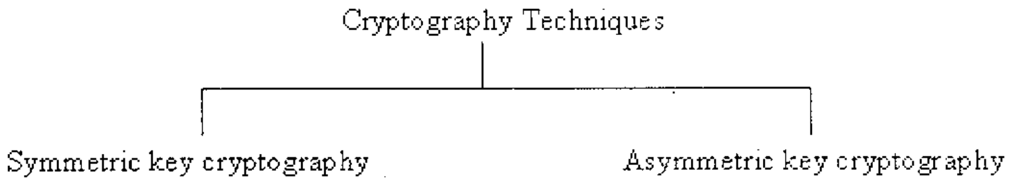


Fig 2.5: Cryptographic techniques

2.3 ENCRYPTON TECHNIQUES

Encryption techniques use complicated algorithms to transform digital information from plaintext to cipher text. Every time the encryption key is changed, the cipher text will be different, although the algorithms stay the same. The relationship between the encryption and decryption keys classifies the encryption methods into the distinct categories: symmetric and asymmetric encryption.

2.3.1 SYMMETRIC ENCRYPTION:

In symmetric encryption, the encryption and decryption keys are the same. This method is faster and easier to implement than asymmetric encryption, since the sender and the receiver use the same key to transmit and receive information. In addition, the key sizes are smaller in symmetric encryption compared to asymmetric algorithms. However, for private exchanging of the key between the sender and receiver, both parties have to agree and trust on a communication medium to exchange the key.

Examples of some common private key encryption algorithms are:

- International data encryption algorithm
- Fast data encipherment algorithm
- Data encryption standard
- Triple DES
- AES
- Serpent
- RC6
- Blowfish

Symmetric encryption is also divided into two groups: Block and stream ciphers. Block ciphers work on blocks of data and are commonly used to encrypt the documents, while stream ciphers operate on bits of data.

2.3.2 ASYMMETRIC ENCRYPTION:

Asymmetric encryption is a method where the encryption and decryption keys are different. These systems are also called public key encryption systems, since the encryption key does not have to be secret. The sender can publish the encryption key and any can encrypt messages going to specific users. However, only the receiver can decrypt the message. Since the decryption key cannot be generated with the knowledge of encryption key. This method is slower and requires more computation power than symmetric encryption.

Examples of some common public key encryption algorithm are:

- Rivest-Shamir-Adelman
- Diffie-Hellman
- Various Elliptic Curve techniques
- Paillier cryptosystem
- RSA encryption algorithm
- Cramer-shoup encryption

CHAPTER 3

AES AN OVERVIEW

The AES was published by national institute of standards and technology in 2001. AES is a symmetric block cipher that is indented to replace DES as the approved standard for a wide range of applications.

3.1 ORIGIN OF AES

In 1999, NIST issued a new version of its DES standard that indicated that DES should only be used for legacy systems and that Triple DES be used. Because of its drawbacks, 3DES was not found to be a reasonable candidate for long term use. This led to NIST's call for proposals for a new AES.

The new algorithm is expected to have security strength equal to or better than Triple DES and significantly improved efficiency. In addition to these general requirements, NIST specified that AES must be a symmetric block cipher with block length of 128 bits and support for key lengths of 128,192 and 256 bits.

In a first round of evaluation, 15 proposed algorithms were accepted. A second round narrowed the field to 5 algorithms. Rivest-Cipher 6 (RC6) algorithm is also one of the selected candidates in the finalist.

3.2 AES EVALUATION CRITERIA:

NIST used three categories of criteria to evaluate potential candidates for the new algorithm.

3.2.1 SECURITY:

This refers to the effort required to crypt analyze an algorithm and the emphasis in the evaluation was on the practicality of the attack.

- **Actual security:** High security compared to other algorithms.
- **Randomness:** The extent to which the algorithm output is indistinguishable from a permutation on the input block.
- **Soundness:** The mathematical basis for the algorithm's security.

3.2.2 COST:

NIST intends AES to be practical in a wide range of applications. Accordingly, AES must have high computational efficiency, so as to be usable in Smart cards, high-speed applications such as broadband links, etc….

- **Licensing requirements:** NIST intends that when the AES is issued, the algorithms specified in the AES shall be an available on a worldwide, non exclusive, royalty-free basis.
- **Computational efficiency:** The evaluation of computational efficiency will be applicable to both hardware and software implementations. Round 1 analysis is on software implementations and specifically on the key-block

size combination. More attention will be paid to hardware implementation and other supported key-block size combination during Round 2 analysis. Computational efficiency essentially refers to the speed of the algorithm. Public comments on each algorithm's efficiency will also be taken into consideration by NIST.

- **Memory requirements:** The memory required for both hardware and software implementations of the algorithm will also be considered during the evaluation process. Round 1 analysis by NIST will focus primarily on software implementations; more attention will be paid to hardware implementations during round 2. Memory requirements will include such factors as gate counts for hardware implementations and code size and RAM requirements for software implementations.

3.2.3 ALGORITHM AND IMPLEMENTATION CHARACTERISTICS

This category includes a variety of considerations, including flexibility; suitability for a variety of hardware and software implementation and simplicity, which will make an analysis of security more straightforward.

- **Flexibility:** Candidate algorithm with greater flexibility will meet the needs of more users than less flexible ones. However, some extremes of functionality are of little practical application (e.g., extremely short key lengths); for those cases, preferences will not be given. Some examples of flexibility may include the following:
    - i. The algorithm can accommodate additional key and block sizes (e.g., 64-bit block size, key sizes other than those specified in the Minimum Acceptability Requirements sections).

ii. The algorithm can be implemented securely and efficiently in a wide variety of platform and applications (e.g., 8-bit processors, ATM networks, voice and satellite communication etc).

iii. The algorithm can be implemented as a stream cipher, message authentication code generator, pseudorandom number generator, hashing algorithm etc.

- **Hardware and software suitability:** A candidate algorithm shall not be restrictive in the sense that it can only be implemented in hardware. If one can also implement the algorithm efficiently in firmware, then this will be an advantage in the area of flexibility.
- **Simplicity:** A candidate algorithm shall be judged according to relative simplicity of design.

3.3 NIST EVALUATION OF RC6 ALGORITHM

- **General Security:** RC6 algorithm has no known security attacks so far. The algorithm has adequate security margin with less number of key sizes and high security than other algorithms when larger key size and data is used.

- **Software implementation:** RC6 algorithm performs encryption and decryption very well across a variety of platforms, including 8 bit, 32 bit and 64 bit platforms and DSPs. There is a decrease in performance in platforms which do not support addition and multiplications.

- **Restricted-Space Environments:** In general RC6 is very well suitable for restricted space environments where either encryption or decryption or both is implemented. It has very low RAM and ROM requirements. It has low memory requirements due to its simplicity with high security.

- **Attacks on Implementation:** The operations used by RC6 are easiest to defend against power and timing attacks. This security is due to data dependent rotation of bits and not due to predetermined rotations. RC6 appears to gain major speed advantage over its competitors when protection is considered. The use of multiplier function increases the data dependent rotations. There are no known attacks on RC6 algorithm so far.

- **Encryption Vs Decryption:** The encryption and decryption function in RC6 algorithm are similar. The reverse operation of encryption process results in decryption. On FPGA implementation of both encryption and decryption is possible with less space. This does not degrade the performance. Further speed can be increased by using the pipelined methods.

- **Other Versatility and Flexibility:** RC6 supports block sizes and key sizes of 32bits, 64 bits, 128 bits, 192 bits, 256 bits, in any combinations. This makes RC6 a flexible algorithm of implementation.

Due to the various advantages in both software and hardware level implementation considerations reveal that RC6 algorithm as the best suited cryptographic algorithm.

CHAPTER 4

FUNCTION OF RC5 ALGORITHM

4.1 RC5 SYMMETRIC KEY CRYPTOGRAPHIC ALGORITHM

RC5 is a symmetric key block encryption algorithm developed by Ron Rivest. The main features of RC5 are that it is quite fast as it uses only the primitive computer operations (such as addition, XOR, shift, etc.). It allows for a variable number of rounds and a variable bit-size key to add to the flexibility. Different applications that demand varying security needs can set these values accordingly. Another important aspect is that RC5 requires less memory for execution, and is therefore, suitable not only for desktop computers, but also for smart cards and other devices that have a small memory capacity.

4.2. HOW RC5 WORKS?

In RC5, the word size (i.e. input plain text block size), number of rounds and number of 8-bit bytes (octets) of the key, all can be of variable length. These values can consist of the sizes as shown in Fig 4.1. Of course, once decided, these values remain the same for a particular execution of the cryptographic algorithm. These are variable in the sense that before the execution of a particular instance of RC5, these values can be chosen from those allowed. This is unlike DES, for instance, where the block size must be of 64 bits and the key size must always be of 56 bits; or unlike IDEA, which uses 64-bit blocks and 128-bit keys.

Parameter	Allowed values
Word size in bits (RC5 encrypts 2-word blocks at a time)	16, 32, 64
Number of rounds	0-255
Number of 8-bit bytes (octets) in the key	0-255

Fig 4.1: RC5 block, round and key details

The following conclusions emerge from the table:

- The plain text block size can be of 32, 64 or 128 bits (since 2-word blocks are used).
- The key length can be 0 to 2040 bits (8-bit keys are used).

The output resulting from RC5 is the cipher text, which has the same size as the Input plain text. Since RC5 allows for variable values in the three parameters, as specified, a particular instance of RC5 algorithm is denoted as RC5-w/r/b, where w = word size in bits, r = number of rounds, b = number of 8-bit bytes in the key. Thus, if we have RC5 -32/16/16, it means that we are using RC5 with a block size of 64-bits (remember that RC5 uses 2-word blocks), 16 rounds of encryption, and 16 bytes (i.e. 128 bits) in the key. Rivest has suggested RC5-32/12/16 as the minimum safety version.

4.3 PRINCIPLES OF OPERATION

As shown in Figure, there is one initial operation consisting of two steps, then a number of rounds. The number of rounds (r) can vary from 0 to 255.

For simplicity, we shall assume that we are working on an input plain text block with size 64 bits. The same principles of operation will apply to other block sizes, in general. The working of RC5 is illustrated using Figure 4.2.

In the first two steps of the one-time initial operation, the input plain text is divided into two 32-bit blocks A and B. The first two sub-keys S[0] and S[1] are added to A and B, respectively. This produces C and D respectively, and marks the end of one-time operation.

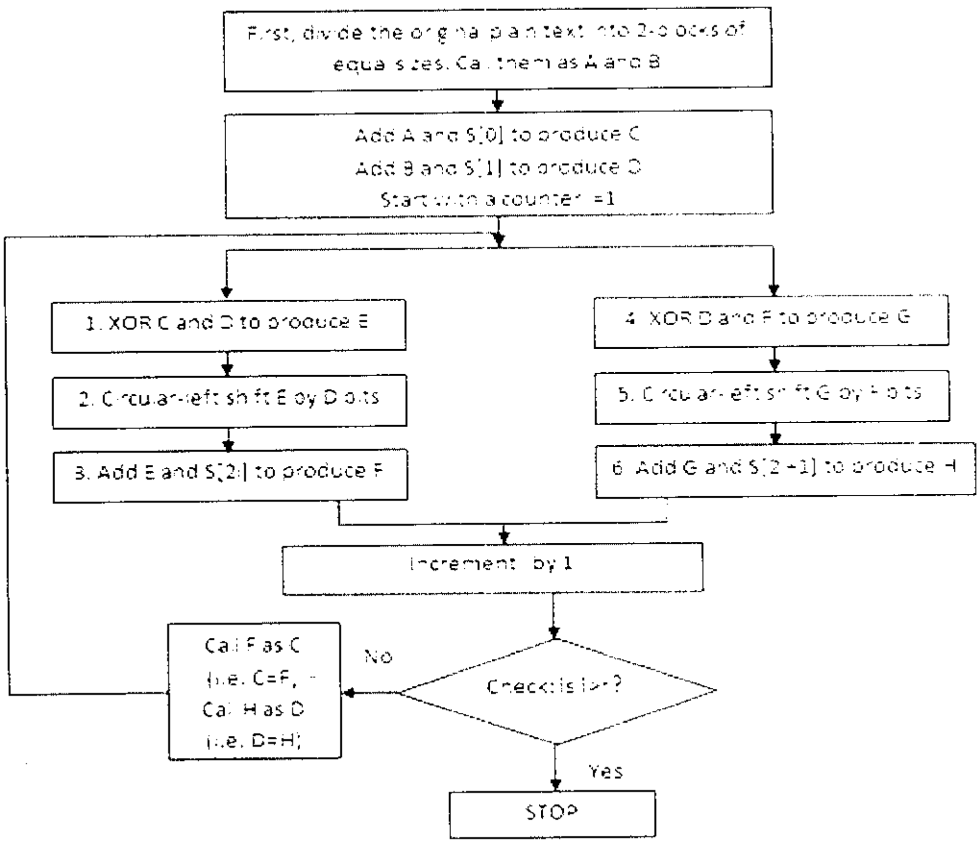

First, divide the original plain text into 2 blocks of equal sizes. Call them as A and B
Add A and S[0] to produce C
Add B and S[1] to produce D
Start with a counter i=1
1. XOR C and D to produce E
2. Circular-left shift E by D bits
3. Add E and S[2i] to produce F
4. XOR D and F to produce G
5. Circular-left shift G by F bits
6. Add G and S[2i+1] to produce H
Increment i by 1
Check: is i>r?
No
Call F as C (i.e. C=F), Call H as D (i.e. D=H)
Yes
STOP

Fig 4.2 Flow chart for Encryption using RC5

Then, the rounds begin. In each round, there are following operations:

- Bitwise XOR
- Left circular-shift
- Addition with the next sub-key, for both C and D. This is the addition operation first, and then the result of the addition mod 2^w (since w = 32 here, we have 2^{32}) is performed.

If you observe the operations shown in Fig4.2 carefully, you will note that the output of one block is fed back as the input to another block, making the whole logic quite complicated to decipher.

4.3.1 ONE-TIME INITIAL OPERATION:

The one time initial operation done in RC5 algorithm is shown in Fig 4.3 . This consists of two simple steps: first, the input plain text is divided into two equal-sized blocks, A and B. Then the first sub-key, i.e. S[0] is added to A, and the second sub-key i.e. S[1] is added to B. These operations are mod 2^{32}, and produce C and D, respectively.

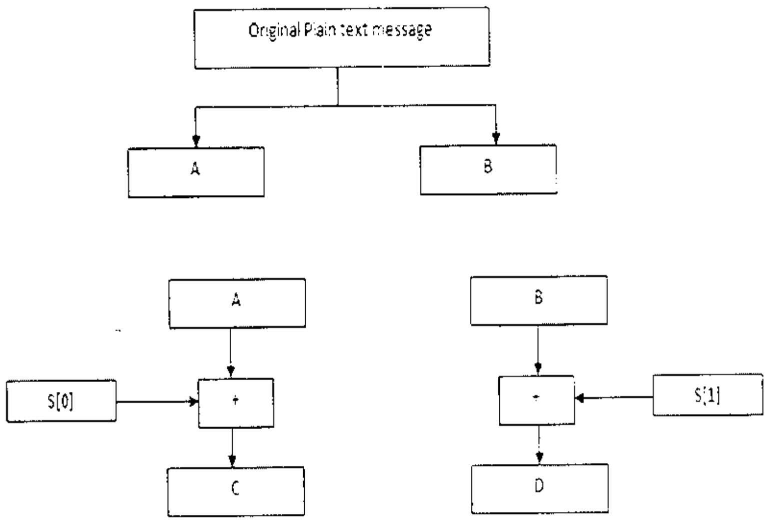


Fig 4.3: One-time initial operation in RC5

Step1: XOR C and D

In the first step of each round, C and D are XORed together to form E, as shown in Fig 4.4.

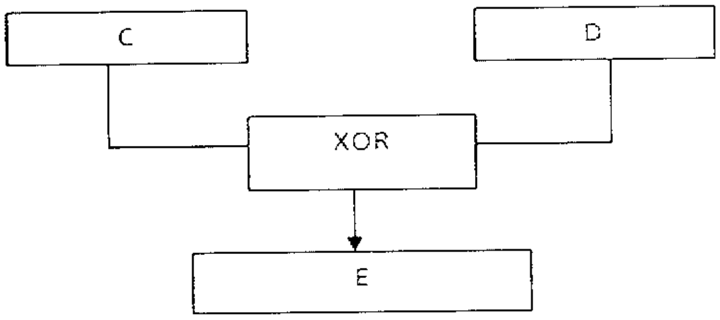


Fig 4.4 Step1 in a round

Step2: Circular-left shift E

Now, E is circular left shifted by D positions, as shown in Fig 4.5. The shifting of D positions is the Shifting of positions by $\log_2(w)$ bits of D. Where w is the bit size of D.

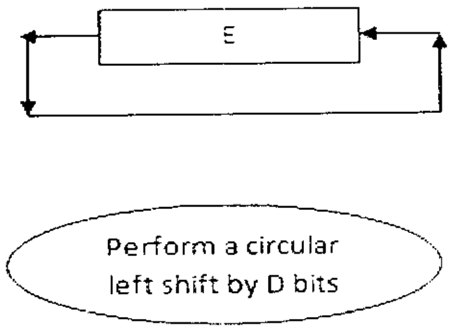


Fig 4.5 Step2 in around

Step3: Add E and next sub-key

In this step, E is added to the next sub-key (which is S[2] for the first round, and S[2i] in general, for any round, where i start with 1. the output of this process is F. This is shown in Fig 4.6.

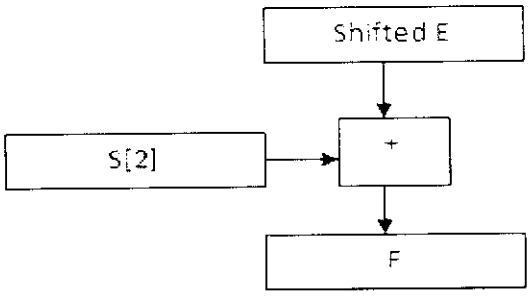


Fig 4.6 Step3 in a round

Now note that the operations (XOR, circular-left shift and add) in steps 4 to 6 that follow are the same as the operations in steps 1 to 3. The only difference, of course, is that the inputs to the steps 4-6 are different from that of steps 1-3.

Step4: XOR D and F

This step is similar to step1, Here; D and F are XORed to produce G. This is shown in Fig 4.7. All the shifting is $\log_2(w)$ bits of the word length taken for the encryption or decryption process.

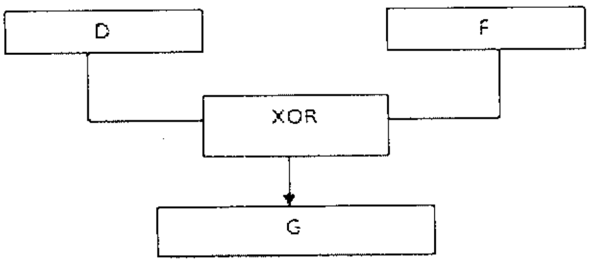


Fig 4.7 Step4 in a round

Step5: Circular-left shift G

This step is step2, Here; G is circular-left shifted by F positions, as shown in Figure 4.8. The shifting of F positions is the Shifting of positions by $\log_2(w)$ bits of F. Where w is the bit size of word F in bits.

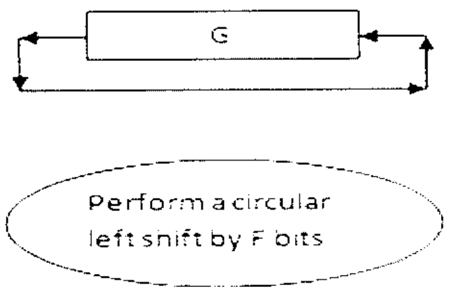


Fig 4.8 Step5 in a round

Step6: Add G and next sub-key

In this step (which is identical to step3), G is added to the next sub-key (which is S[3] for the first round, and S[2i + 1] in general, for any round, i starts with 1. The output of this process is H. This is shown in Fig 4.9 below.

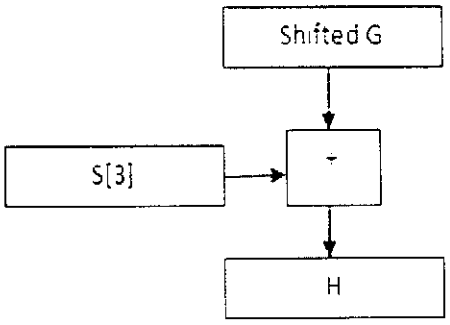


Figure 4.9 Step6 in a round

Step7: Miscellaneous Tasks

In this step, to see if all the rounds are over or not are checked. For this, perform the following steps:

- Increment i by 1
- Check to see if i < r

Assuming that i still less than r, we rename F as C and H as D, and return back to step 1. This is shown in Fig 4.10.

```
i = i + 1
if I < r
        Call F and C again
        Call H and D again
        Go back to step1
Else
        Stop
End if
```

Fig 4.10 Step7 in a round

The final encrypted data is obtained as the output of the final step. The decryption of the data is the same procedure followed by the reverse operation done in encryption steps.

4.4 MATHEMATICAL REPRESENTATION

Interestingly, all these operations (one-time initial operation and all the rounds) in RC5 algorithm can be signified mathematically in a very cryptic fashion, as shown in below. Note that <<< means circular-left shift.

```
A= A + S[0]
B= B + S[1]
For i=1 to r
    A = ((B XOR A) <<<A) + S[2i]
    B = ((B XOR A) <<<A) + S[2i + 1]
Next i
```

The mathematical representation of RC5 algorithm decryption process is shown in below. Note that >>> means circular-right shift.

```
For i=1 to r to 1 step-1 (i.e. decrement i each time by 1)
    A = ((B-S[2i + 1]>>> A) XOR A
    B = ((A-S[2i]>>> B) XOR B
Next i
B = B - S[1]
A = A - S[0]
```

4.5 SUB-KEY CREATION

Sub-key creation process in RC5 is a two step process.

1. In the first step, the sub-keys (denoted by S[0], S[1],...) are generated
2. The original key is called as L. In the second step, the sub-keys (S[0]. S[1] ...) are mixed with the corresponding sub-portions of the original key (i.e. L[0], L[1] ...). This is shown in Fig 4.11.

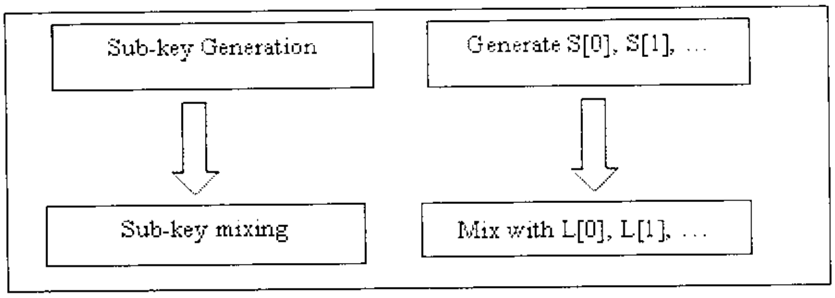


Fig 4.11 Sub-key generation process

Step 1: Sub-key generation

In this step, two constants P and Q are used. The array of sub-keys to be generated is called as S. The first sub-key S[0] is initialized with the value of P.

Each next sub-key (i.e. S[1], S[2], ...) is calculated on the basis of the previous sub-key and the constant value Q, using the addition mod 2^{32} operations. The process is done 2(r + 1) - 1 times, where r is the number of rounds, as before. Thus, if we have 12 rounds, this process will be done 2(12 + 1) – 1 times, i.e. 2(13) – 1 times, i.e. 25 times. Thus, we will generate sub-keys S[0], S[1], ...S[25]. This process is illustrated in Fig 4.12.

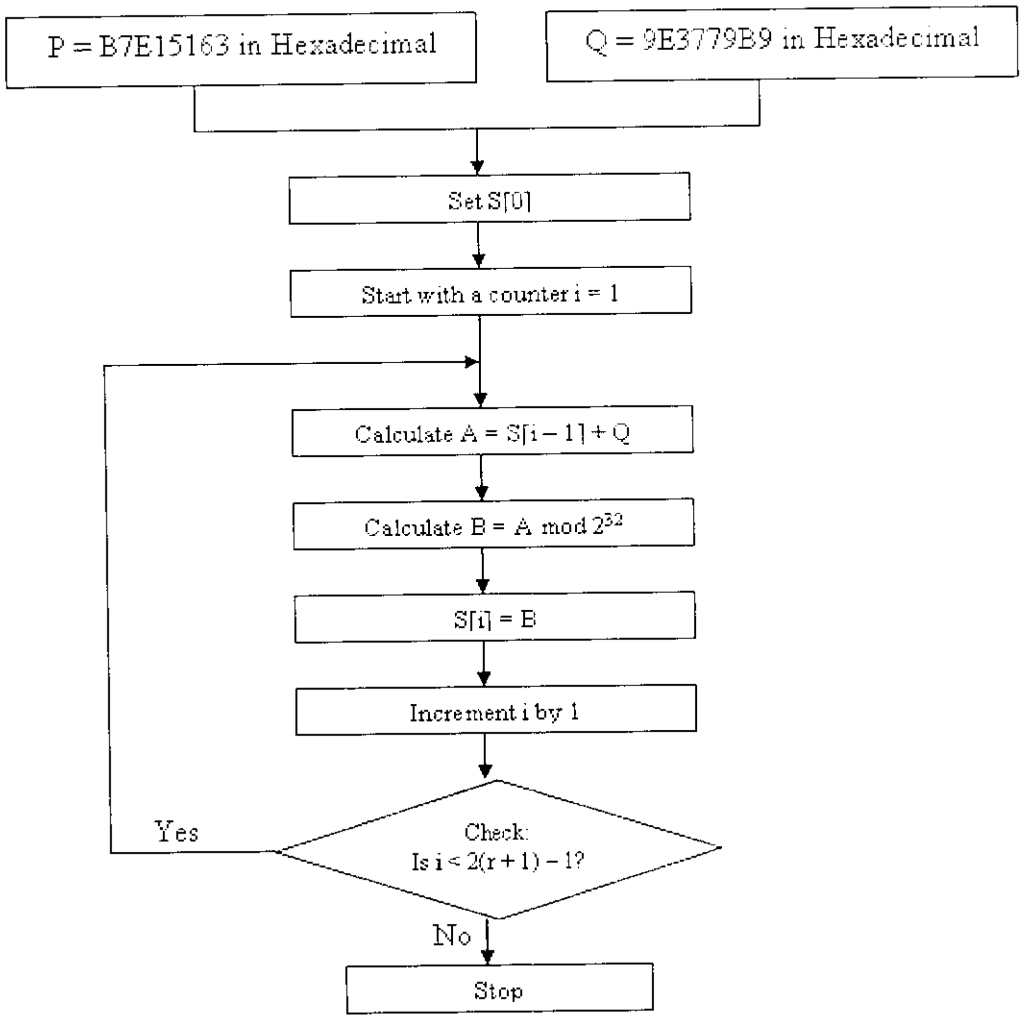

P = B7E15163 in Hexadecimal
Q = 9E3779B9 in Hexadecimal
Set S[0]
Start with a counter i = 1
Calculate A = S[i – 1] + Q
Calculate B = A mod 2^32
S[i] = B
Increment i by 1
Check:
Is i < 2(r + 1) – 1?
Yes
No
Stop

Fig 4.12 Sub-key generation

The mathematical form of sub-key generation is shown below.

```
S[0] = P
For i = 1 to 2 (r + 1) – 1
        S[i] = (S[i – 1] + Q) MOD 2^32
Next i
```

Step 2: Sub-key mixing

In the sub-key mixing stage, the sub-keys S[0], S[1], ...are mixed with the sub-portions of the original key, i.e. L[0], L[1], ...L[c]. Note that c is the last sub-key position in the original key. This process is shown mathematically below.

```
$i=j=0$
$A=B=0$
Do 3n times (where n is the maximum of 2 (r + 1) and c)
    $A = S[i] = (S[i] + A + B) <<< 3$
    $B = L[i] = (L[i] + A + B) <<< (A + B)$
    $i = (i + 1) \bmod 2(r + 1)$
    $j = (j + 1) \bmod c$
End-Do
```

CHAPTER 5

RC6 ALGORITHM –ENCRYPTION AND DECRYPTION

5.1 INTRODUCTION

The RC6 algorithm is a block cipher that was one of the finalists in the Advanced Encryption Standard (AES) competition; The AES competition sponsored by the National Institute of Standards and Technology (NIST), began in 1997. The RC6 algorithm evolved from its predecessor RC5, a simple and parameterized family of encryption algorithms. Since RC6 is an evolution of RC5, evolutionary differences will be noted accordingly. Also, for consistency in all AES-related documents and RC6 research, whenever RC6 is mentioned without any trailing parameters, the assumed parameters are the AES required parameters, which will be defined later.

The evolution leading to RC6 has provided a simple cipher yielding numerous evaluations and adequate security in a small package. After describing the structure of the algorithm, the prominent goal that stands out is simplicity. Through this simplicity, multiple evaluations have been performed, including AES-related evaluations. The fact that such a small, simple algorithm contended for AES with such high security requirements is noteworthy.

RC6 is an iterated symmetric block cipher which means that:

- RC6 works by repeating the same defined steps multiple times.
- RC6 is a secret key encryption algorithm.
- RC6 operates on a fixed number of bytes.

RC6 as well as most encryption algorithm are reversible. This means that almost the same steps are performed to complete both encryption and decryption in reverse order. The algorithm operates on bits as well as bytes, which makes it simpler to implement. The symmetric key is used for both encryption and decryption process.

Block diagram of the process flow is shown in the Fig 5.1. The input data is encrypted using the key. This encrypted data is then decrypted to recover the original input data.

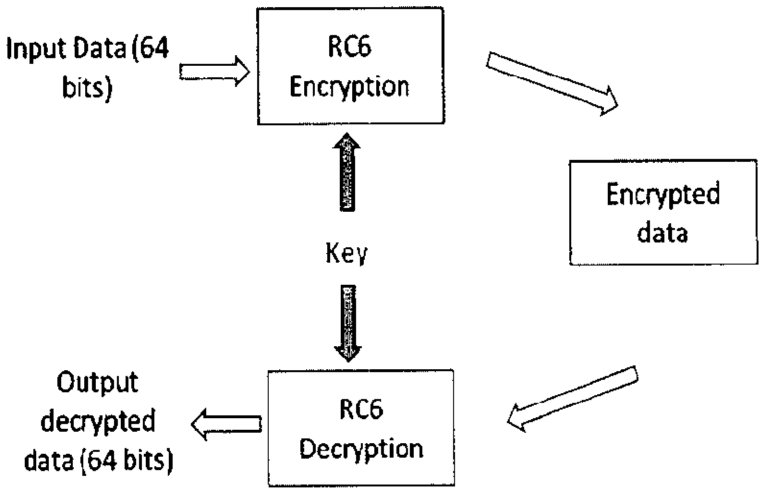


Fig 5.1 Process flow

5.2 RC6 ALGORITHM DESIGN OBJECTIVES

The design objective of the RC6 algorithm has the best captured three goals of high security, exceptional simplicity, and good performance.

The most significant working of RC6 algorithm made was the following:

1. Security would be the over-riding requisite for a credible AES candidate.

2. Performance on older 8- and 16-bit environments would probably be less important over the lifetime of the AES than current 32-bit and 64-bit environments. Today most 8-bit processors are to be found in cheaper smart cards. Many of the more versatile smart cards use more advanced processors. It is inevitable that the fraction of smart cards with 8, 16, -bit processors would rapidly decline with the inevitable drop in the cost of more advanced cards. Furthermore it is important to recognize the range of applications that might require encryption on a smart card. Such applications would typically be in critical performance. (Certainly a smart card would be used for the bulk of encryption of data!) So, in the interests of compromise, we aimed the performance of RC6 at 32- and 64-bit environments will be more important in the most of the applications.

3. Software implementations and hardware implementations would probably be more important. For most applications, we believed that an implementation of RC6 in software as well as hardware would likely be the best choice for security at low memory space requirements. We felt that RC6's primitive operations were typically well-supported on modern microprocessors and that this would increasingly become the normal cryptographic algorithm in use. The exceptional effort has gone into the design of such processors.

5.3 COMPLETE STRUCTURE OF THE RC6 ALGORITHM

The Conventional structure of the RC6 Cipher process is shown in Fig 5.2 below. The input: plaintext is stored in four w-bit input registers A, B, C & D . The parameter r is the number of rounds ,w-bit round keys S[0, ... , 2r + 3] . The output: Ciphertext is stored in A, B, C, D. The cipher process of encryption and decryption are same. The reverse operation of encryption is carried for decryption.

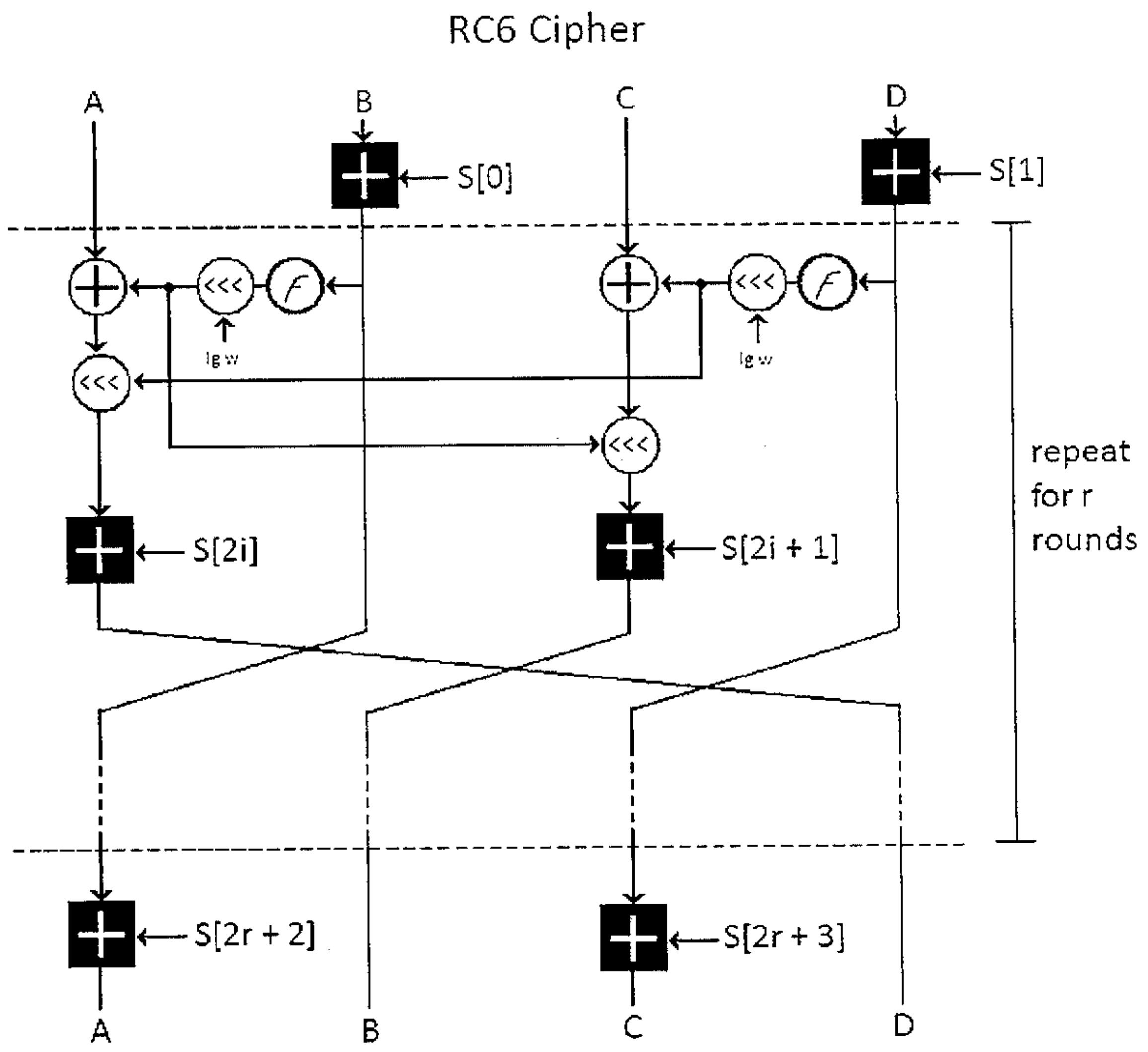


Fig 5.2 Structure of RC6 cipher

5.4 DESCRIPTION

In 1995, RC5 came about from Ronald Rivest, one of the creators of the RSA algorithm (Rivest et al., 1998a). RC5 is a symmetric block cipher that relies heavily on data-dependent rotations. RC5 has been the subject of many studies that have expanded the knowledge of how RC5's structure contributes to its security. With certain architectural constraints by the AES competition, RC5 did not appear to be the best fit. However, in 1998, RC5's successor is born: RC6. Improvements over RC5 include using four w-bit word registers, integer multiplication as an additional primitive operation, and introducing a quadratic equation into the transformation (Rivest et al., 1998a).

5.4.1 ALGORITHM DETAILS

RC6, like RC5, consists of three components: a key expansion algorithm, an encryption algorithm, and a decryption algorithm. The parameterization is shown in the following specification: RC6-*w*/*r*/*b*, where *w* is the word size, *r* is the non-negative number of rounds, and *b* is the byte size of the encryption key (Rivest et al., 1998a). RC6 makes use of data-dependent rotations, similar to DES rounds (Rivest et al., 1998a). RC6 is based on seven primitive operations as shown in Table 1. Normally, there are only six primitive operations; however, the parallel assignment is primitive and an essential operation to RC6. The addition, subtraction, and multiplication operations use two's complement representations. Integer multiplication is used to increase diffusion per round and increase the speed of the cipher (Rivest et al., 1998a).

Operation	**Description**
a + b	Integer addition modulo 2^w
a – b	Integer subtraction modulo 2^w
a [illegible] b	Bitwise exclusive-or (XOR) of w-bit words
a x b	Integer multiplication modulo 2^w
a <<< b	Rotate the *w*-bit word a to the left by the amount given by the least significant ($\log_2 w$) bits of *b*
a >>> b	Rotate the *w*-bit word a to the right by the amount given by the least significant ($\log_2 w$) bits of *b*
Enc: (A,B,C,D) = (B,C,D,A) **Dec:** (A,B,C,D) = (D,A,B,C)	Parallel assignment of values on the right to registers on the left.

Table 1: RC6 Operations

5.5 DIFFUSION

Diffusion involves propagating bit changes from one block to other blocks. An avalanche effect is where one small change in the plaintext triggers major changes in the cipher text. To speed up the avalanche of change between rounds, a quadratic equation is introduced (Rivest et al., 1998a). By increasing the rate of diffusion, the rotation amounts spoiling sooner is more likely, due to the changes from simple differentials (Rivest et al., 1998a). To achieve the security goals for transformation, the following quadratic equation is used twice within each round:

$$f(x) = x\,(2x + 1)(mod\ 2^{w})$$

The high-order bits of this equation, which depend on all of the bits of x, are used to determine the rotation amount used . In conjunction with the quadratic equation, the ($\log_2$ w) bit shift complicates advanced cryptanalytic attacks (Rivest et al., 1998a). Integer multiplication also contributes by making sure that all of the bits of the rotation amounts are dependent on the bits of another register. The Diffusion or Diffusion multiplier function increase the data dependency making algorithm complicated for hacking.

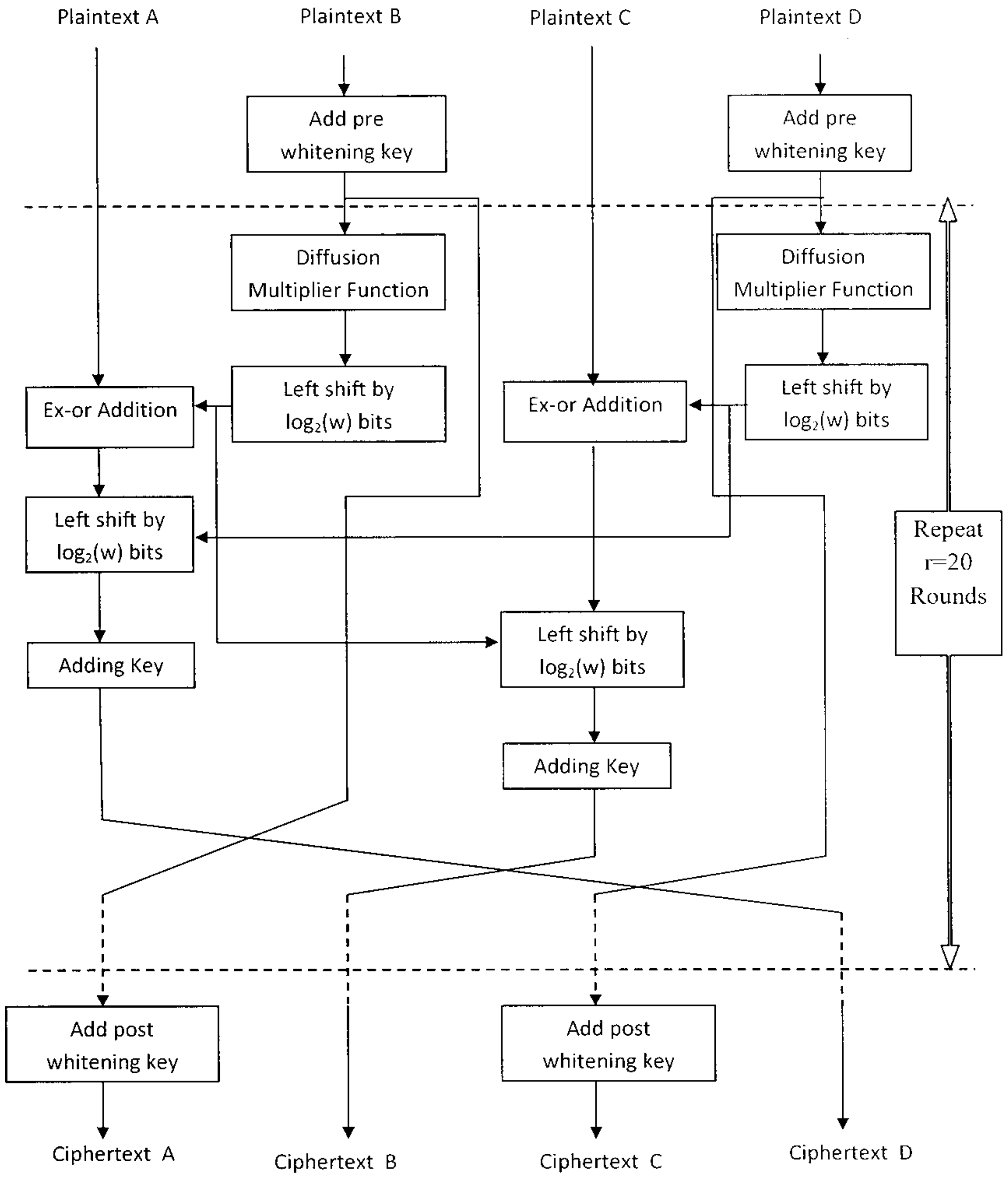
5.6 ENCRYTPION PROCEDURE

The processes of encryption composed of three stages: pre-whitening, an inner loop of rounds, and post-whitening. Pre-whitening and post-whitening remove the possibility of the plaintext revealing part of the input to the first round of encryption and the cipher text revealing part of the input to the last round of encryption

The encryption process step is illustrated in Fig 5.3 and uses the operations in Table 1. First, the registers B and D undergo pre-whitening. Next, there are r rounds, which are designated by the Looping operation. The registers B and D are put through the quadratic equation and rotated ($\log_2$ w) bits to the left, respectively.

The resulting value of B has an exclusive-or (XOR) operation with A, and D with C respectively. This value *t* is then left-rotated *u* bits and added to round key S[2i]; the resulting value of D and C is left-rotated *t* bits and added to round key S[2i + 1]. The t and u are variables used as shown in the mathematical form of the RC6 algorithm shown in the chapter 5.8.1.

In the final stage of the round, the register values are permuted, using parallel assignment, to mix the AB computation with the CD computation, increasing cryptanalytic complexity (Rivest et al., 1998a). Finally, registers A and C undergo post-whitening. The value of r can be any number of times with a tradeoff increased security and less time consumption . For higher security the r value chosen is 20 , where it takes abundant years for cracking the algorithm. The addition, subtraction operations performed in the algorithm are modulo addition and modulo subtraction operations.

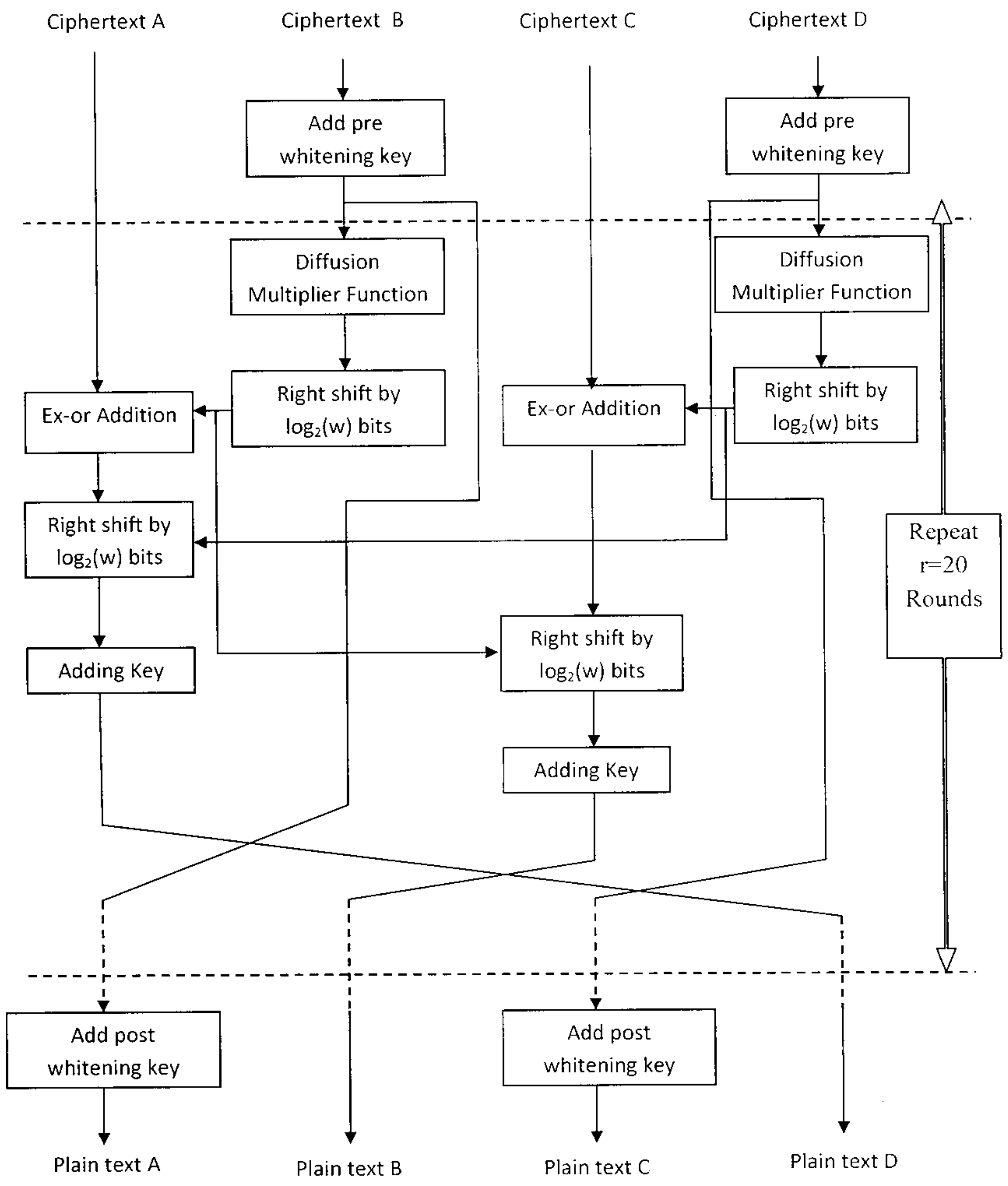


5.3 Encryption steps

5.7 DECRYTPION PROCEDURE

The processes of Decryption also composed of three stages: pre-whitening, an inner loop of rounds, and post-whitening. Pre-whitening and post-whitening remove the possibility of the input data revealing part of the input to the first round of decryption and the plain text revealing part of the input to the last round of decryption.

The decryption process step is illustrated in Fig 5.4 and uses the operations in Table 1. The procedure begins with a pre-whitening step for C and A. The loop runs in reverse for the number of *r* rounds. Within the loop, the first task is parallel assignment. From there, the aforementioned quadratic equation is used on D and B respectively. The resulting value for *u*, and *t* respectively, is left-rotated ($\log_2$ w) bits. The round key S[2i + 1] is subtracted from register C value, the result of which is right-rotated *t* bits; round key S[2i] is subtracted from register A value, the result of which is right-rotated *u* bits. This resulting value involving register C has an exclusive-or operation with *u*, A with *t* respectively. After completing the loop, D and B undergo a post-whitening step. For higher security the r value chosen is 20 , where it takes abundant years for cracking . The variables t, u used are the temporary variables used as shown in the chapter 5.8.2. It's evident that the decryption process is done as an inverse of the encryption process which also reduces the hardware requirement to great extent.



5.4 Decryption steps

5.8 KEY SETUP

The key expansion algorithm is used to expand the user-supplied key to fill an expanded array S, so S resembles an array of *t* random binary words. The key schedule algorithm for RC6 differs from the RC5 version where more words are derived from the user-supplied key. The user must supply a key of *b* bytes, where 0 = *b* = 255, and from which (2*r*+4) words are derived and stored in a round key array S. Zero bytes are appended to give the key length equal to a "non-zero integral number" (Rivest et al., 1998a). The key bytes are then loaded in little-endian order into an array L of size *c*; when b = 0, c = 1 and L[0] = 0 . The (2*r*+4) derived words are stored in array S for later decryption or encryption. Figure 1 illustrates the key schedule used in both RC5 and RC6. P_w and Q_w are "magic constants", or word-sized binary constants where Odd(x) is the least odd integer greater than or equal to |x| . In Figure 1, the base of natural logarithms and the golden ratio are respectively defined as e = 2.718281828459... and ø = 1.618033988749... . The mathematical form of the key setup is shown below.

5.8.1 MATHEMATICAL FORM OF KEY SCHEDULE

The mathematical representation of the key schedule with rc6-w/r/b is shown below .The parameter 'w' is the word size in bits, 'r' is the number of rounds. 'b' is the length of the key used.

Input:

User-supplied b byte key preloaded into the c-word

Array L[0,..., c - 1]

Number r of rounds

$P_w = Odd((e - 2)2^w)$

$Q_w = Odd((\text{ø} - 1)2^w)$

Output:

w-bit round keys S[0,…, 2r + 3]

Procedure:

```
$S[0] = P_w$
**for** $i = 1$ **to** $(2r + 3)$ **do**
$S[i] = S[i \_ 1] + Q_w$
$A = B = i = j = 0$
$v = 3 \times \max\{c, 2r + 4\}$
**for** $s = 1$ **to** $v$ **do**
    {
        $A = S[i] = (S[i] + A + B) \lll 3$
        $B = L[j] = (L[j] + A + B) \lll (A + B)$
        $i = (i + 1) \bmod (2r + 4)$
        $j = (j + 1) \bmod c$
    }
```

5.9 MATHEMATICAL FORM OF ENCRYPTION AND DECRYPTION

The mathematical form of the RC6 algorithm is shown below. The parameter 'w' is the word size in bits, 'r' is the number of rounds , and 'b' is the length of the key used.

5.9.1 RC6 Encryption with RC6-w/r/b

The mathematical form of the encryption using RC6 algorithm is shown below. The optimized value of r chosen is 20.

Input:

Plaintext stored in four w-bit input registers A,B,C,D

Number r of rounds

w-bit round keys S[0,...,2r + 3]

Output:

Cipher text stored in A,B,C,and D

Procedure:

```
B = B + S[0]
D = D + S[1]
for i = 1 to r do
    {
        t = (B x (2B + 1)) <<< $\log_2 w$
        u = (D x (2D + 1)) <<< $\log_2 w$
        A = ((A $\oplus$ t) <<< u) + S[2i]
        C = ((C $\oplus$ u) <<< t) + S[2i+ 1]
        (A,B,C,D) = (B,C,D,A)
    }
A = A + S[2r + 2]
C = C + S[2r + 3]
```

5.9.2 RC6 Decryption with RC6-w/r/b

The mathematical form of the encryption using RC6 algorithm is shown below. The optimized value of r chosen is 20.

Input

Cipher text stored in four w-bit input registers A,B,C,D

Number r of rounds

w-bit round keys S[0,…,2r + 3]

Output:

Plaintext stored in A,B,C,D

Procedure:

```
$C = C - S[2r + 3]$
$A = A - S[2r + 2]$
for $i = r$ downto $1$ do
  {
     $(A,B,C,D) = (D,A,B,C)$
     $u = (D \times (2D + 1)) \lll \log_2 w$
     $t = (B \times (2B + 1)) \lll \log_2 w$
     $C = ((C - S[2i + 1]) \ggg t) \oplus u$
     $A = ((A - S[2i]) \ggg u) \oplus t$
  }
$D = D - S[1]$
$B = B - S[0]$
```

CHAPTER 6

RC6 ALGORITHM DEVELOPING ENVIRONMENT

6.1 HARDWARE ENVIRONMENT

The hardware environment used for implementing the RC6 algorithm is "XILINX SPARTAN 3 FPGA XC3S400PQ208". The technical specification of the hardware is shown below.

- Software: Model Sim6.2 /Xilinx Foundation Series 9.2i
- Hardware:- Spartan-3 Kit
  - Device : XC3S400
  - System Gates: 400k
  - Equivalent Logic Cells: 8,064
  - Distributed RAM Bits (K=1024): 56K
  - Block RAM Bits (K=1024): 288K
  - Dedicated Multipliers: 16
  - DCMs: 4
  - Maximum User I/O: 264

6.2 DESIGN TECHNIQUES

The development of new types of sophisticated field-programmable devices (FPDs), the process of designing digital hardware has changed dramatically over the past few years. Unlike previous generations of technology, in which board-level designs included large numbers of Small Scale Integrated (SSI) chips containing basic gates, virtually every digital design produced today consists mostly of high-density devices. The most compelling advantages of FPDs are instant manufacturing turnaround, low start-up costs, low financial risk and (since programming is done by the end user) ease of design changes. Some of the most important terminologies used in the digital hard ware design are discussed below.

- **PLA -Programmable Logic Array (PLA)** is a relatively small FPD that contains two levels of logic, an AND-plane and an OR-plane, where both levels are programmable. Although PLA structures are sometimes embedded into full-custom chips, only PLAs that are provided as separate integrated circuits and are user-programmable are considered.
- **PAL-Programmable Array Logic (PAL)** is a relatively small FPD that has a programmable AND-plane followed by a fixed OR-plane.
- **Field-Programmable Device (FPD)** —A general term that refers to any type of integrated circuit used for implementing digital hardware, where the chip can be configured by the end user to realize different designs. Programming of such a device often involves placing the chip into a special programming unit, but some chips can also be configured "in-system". Another name for FPDs is programmable logic devices (PLDs); although PLDs encompass the same types of chips as FPDs, we prefer the term FPD because historically the word PLD has referred to relatively simple types of devices.
- **CPLD - Complex Programmable Logic Devices** that consists of an arrangement of multiple SPLD (simple PLD either PLA or PAL) like blocks on a single chip. Alternative names sometimes adopted for this style of chip are Enhanced PLD (EPLD), Super PAL, Mega PAL, and others.
- **FPGA - A Field-Programmable Gate Array** is an FPD featuring a general structure that allows very high logic capacity. Whereas CPLDs feature logic resources with a wide number of inputs (AND planes), FPGAs offer more

narrow logic resources. FPGAs also offer a higher ratio of flip-flops to logic resources than do CPLDs.

- **HCPLDs - High-capacity PLDs**: A single acronym that refers to both CPLDs and FPGAs. This term has been coined in trade literature for providing an easy way to refer to both types of devices.
- **Logic Capacity** - The amount of digital logic that can be mapped into a single FPD. This is usually measured in units of "equivalent number of gates in a traditional gate array". In other words, the capacity of an FPD is measured by the size of gate array that it is comparable to. In simpler terms, logic capacity can be thought of as "number of 2-input NAND gates".
- **Logic Density-** The amount of logic per unit area in an FPD.

6.2.1 TYPES OF FPGA

The two basic categories of Field-Programmable Gate Array used are shown below.

1. SRAM-based FPGAs
2. Anti fuse- based FPGAs.

In the first category, Xilinx and Altera are the leading manufacturers in terms of number of users, with the major competitor being AT&T. For antifuse-based products, Actel, Quick logic and Cypress, and Xilinx are major manufacturer. The SRAM based FPGA Xilinx and Altera is shown in Fig 6.1and Fig 6.2 below.

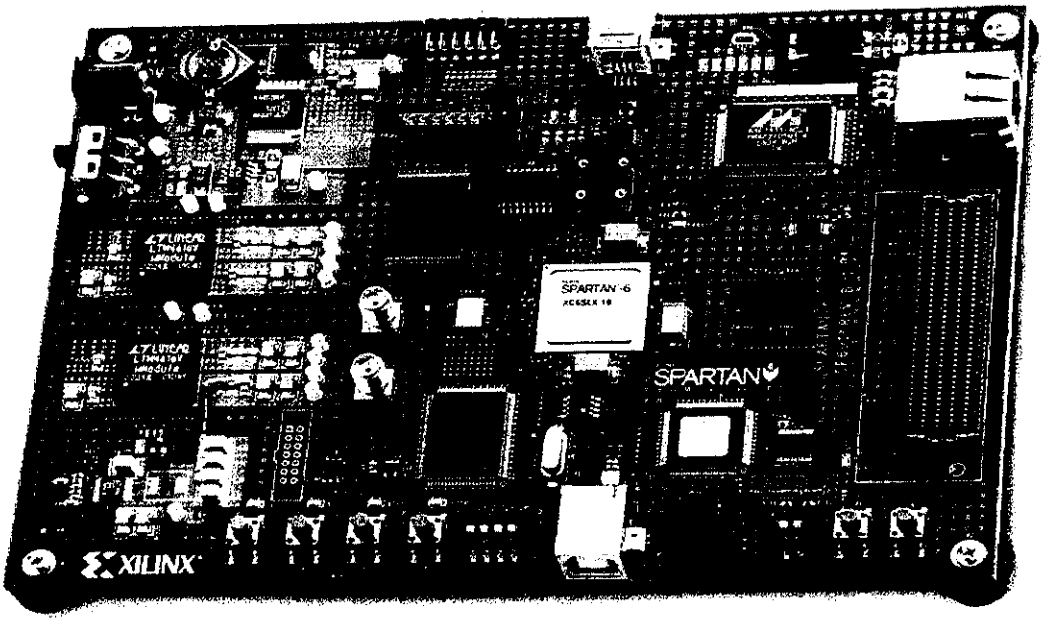


Fig 6.1 Xilinx FPGA Kit

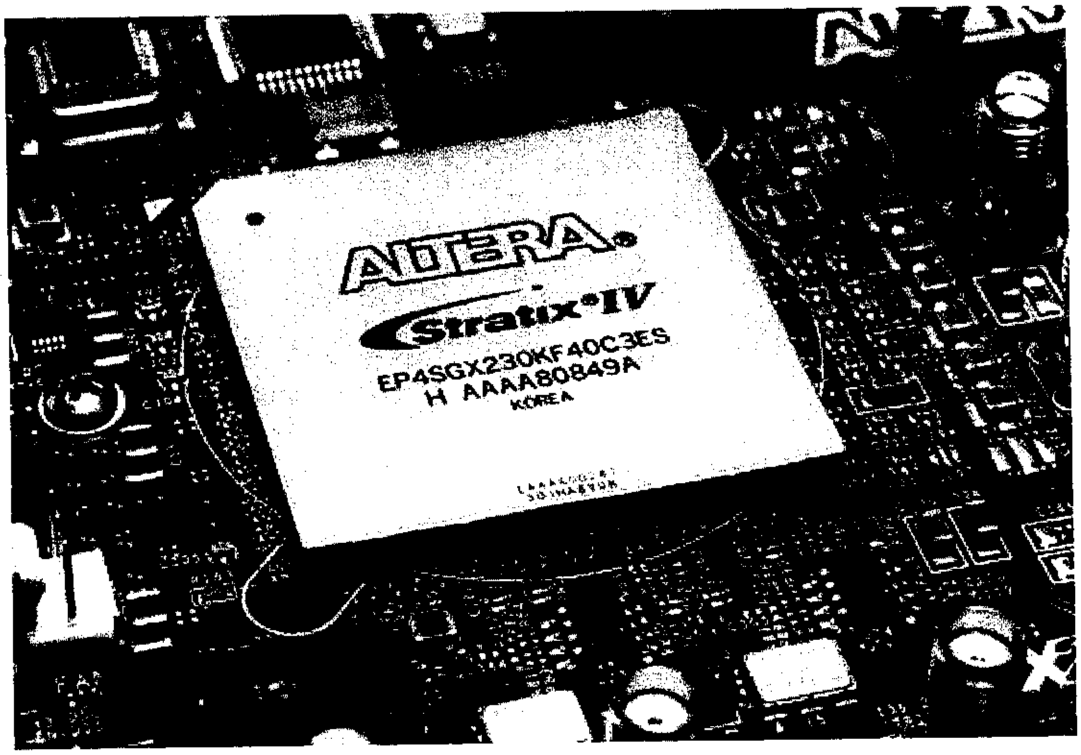


Fig 6.2 Altera FPGA kit

6.2.2 NEED FOR FPGAs

Field-Programmable Gate Array offer high speeds and high range of capacities. FPGAs are useful for a very wide assortment of applications, from implementing random glue logic to prototyping small gate arrays. One of the most common uses in industry at this time, and a strong reason for the large growth of the FPGA market, is the conversion of designs that consist of multiple SPLDs into a smaller number of FPGAs. FPGAs can realize reasonably complex designs, such as graphics controller, LAN controllers, UARTs, cache control, and many others. As a general rule-of-thumb, circuits that can exploit wide AND/OR gates, and do not need a very large number of flip-flops are good candidates for implementation in FPGAs. A significant advantage of FPGAs is that they provide simple design changes through re-programming (all commercial FPGA are re-programmable). Within system programmable FPGAs it is even possible to re-configure hardware (example –To change a protocol for a communications circuit) without power-down.

Designs often partition naturally into the SPLD-like blocks in a FPGA. The result is more predictable speed-performance than would be the case if a design were split into many small pieces and then those pieces were mapped into different areas of the chip. Predictability of circuit implementation is one of the strongest advantages of FPGA architectures. Hardware description languages are used for programming the FPGAs.

6.3.1 IMPLEMENTATION LANGUAGE

VHDL and Verilog are the two commonly used Hardware Description Languages (HDL) in the VLSI industry. In this project, VHDL is used for implementing the RC6 Encryption and Decryption Core. VHDL stands for VHSIC Hardware Description Language. VHSIC abbreviation for Very High Speed Integrated Circuits, an initiative funded by the United States Department of Defense in the 1980's that led to the creation of VHDL. VHDL was the original and first hardware description language to be standardized by the IEEE, through the IEEE 1076 standard in 1988. An additional standard, the IEEE 1164, was later added to introduce a multi-valued logic system in 1993. The IEEE 1076 was further revised in 2000 and the current revision was updated in 2002. A hardware design can be implemented using VHDL in a descriptive structure and the code can be used for hardware simulation and synthesis. Benefits of using VHDL are:

- VHDL is a standard
- It is a technology or vendor independent language
- It is easily portable and reusable
- Modular level design and system integration are easy

It is supported by both FPGA vendors and ASIC foundries for fabrication.

The RC6 algorithm hardware design is implemented using VHDL in a descriptive structure. The code can be used for hardware simulation and synthesis. The benefits of the designed algorithm using VHDL provides all the advantage of VHDL. For simplicity and implementation purpose the program is coded for encryption of four 16-bit data (64-bits). The key size used for encryption and decryption of RC6 algorithm is16-bit. The manual verification of the encrypted data cannot be seen visually due the high frequency of FPGA device. For the purpose of verifying the output, hyper terminal can be used for giving input and verifying the output in the hyper terminal itself. This can be done by making fewer modifications in the program and connecting the FPGA in serial data transfer modes for the above purpose. The flexibility of the implemented RC6 algorithm allows the above process due to coding in the VHDL. The simulation results and synthesized results reveal that the memory used for implementation is low. The total memory usage for encryption algorithm in AES algorithm uses 574880 kilobytes and RC6 uses 170080 kilobytes. The total memory usage for decryption procedure in AES algorithm uses 1321952 kilobytes and RC6 uses 171104 kilobytes. The simulation results and synthesis report for the RC6 algorithm are discussed in the following chapters.

CHAPTER 7

SIMULATION RESULTS

7.1 SIMULATION

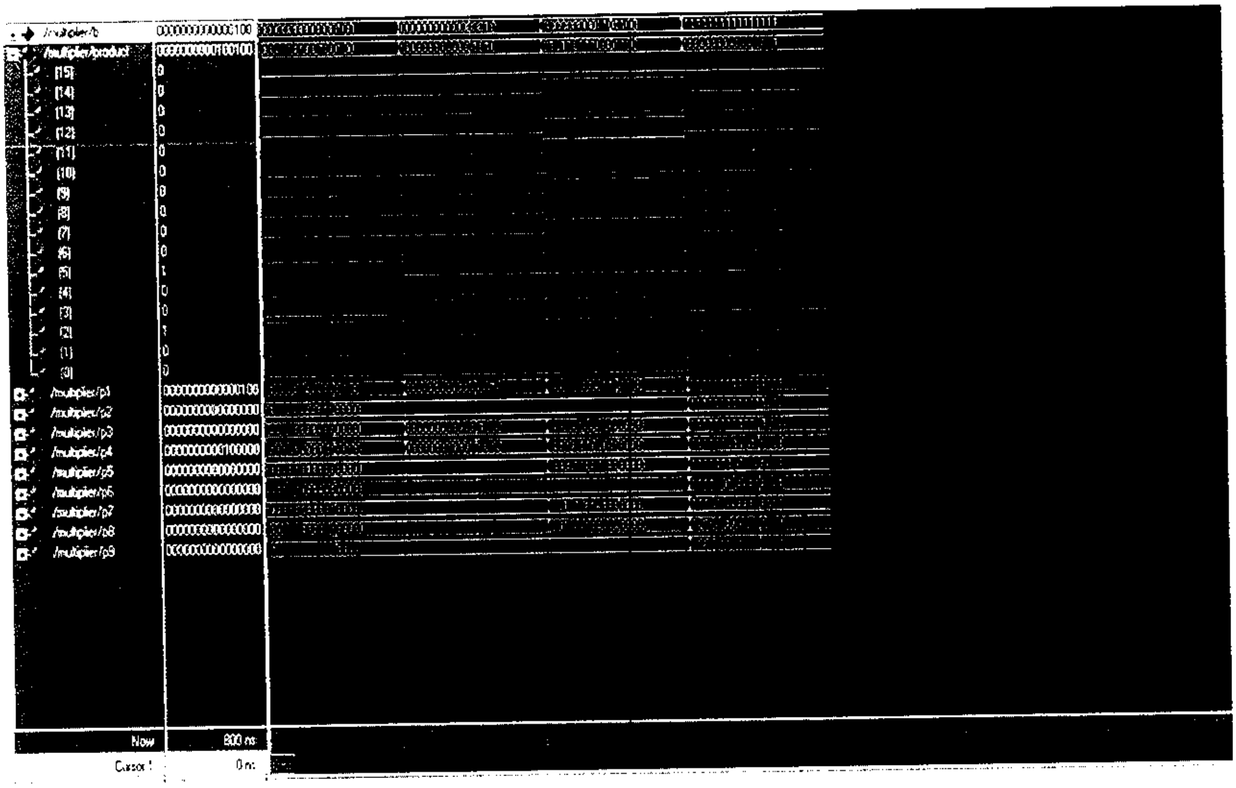
Simulation is the process of testing the design with user generated simulators. Simulation can be done at various stages of the design cycle.

- Functional (Behavioral code)
- Gate level (Netlist)
- Timing (Placed & Routed Netlist)

The simulation results for the encryption, decryption module and Diffusion multiplier are presented in this chapter. Code for the various modules has been written in VHDL and simulated using the MODELSIM simulator. The synthesis part has been carried out using Xilinx and implementation has been done at hardware level on device Xilinx - Spartan family.

7.1.1 SIMULATION RESULTS FOR THE DIFFUSION MULTIPLIER

The simulation result of Diffusion multiplier function is shown in Fig 7.1 below. This diffusion function is used for increasing the data dependency to resist against hacking.



```
/multiplier/b         0000000000000100
/multiplier/product   0000000000100100
  (15)                0
  (14)                0
  (13)                0
  (12)                0
  (11)                0
  (10)                0
  (9)                 0
  (8)                 0
  (7)                 0
  (6)                 0
  (5)                 1
  (4)                 0
  (3)                 0
  (2)                 1
  (1)                 0
  (0)                 0
/multiplier/p1        0000000000000100
/multiplier/p2        0000000000000000
/multiplier/p3        0000000000000000
/multiplier/p4        0000000000100000
/multiplier/p5        0000000000000000
/multiplier/p6        0000000000000000
/multiplier/p7        0000000000000000
/multiplier/p8        0000000000000000
/multiplier/p9        0000000000000000
Now                   800 ns
Cursor 1              0 ns
```

Fig 7.1 Simulation results for the diffusion multiplier

7.1.2 SIMULATION RESULTS FOR ENCRYPTION

The simulation result for encryption is shown in Fig 7.2 below. The given four 16 bit plain text data is encrypted into a four 16 bit cipher text. The encryption module is designed in such a way it works when the control input is 1 (high) and reset is 0 (zero) on execution.

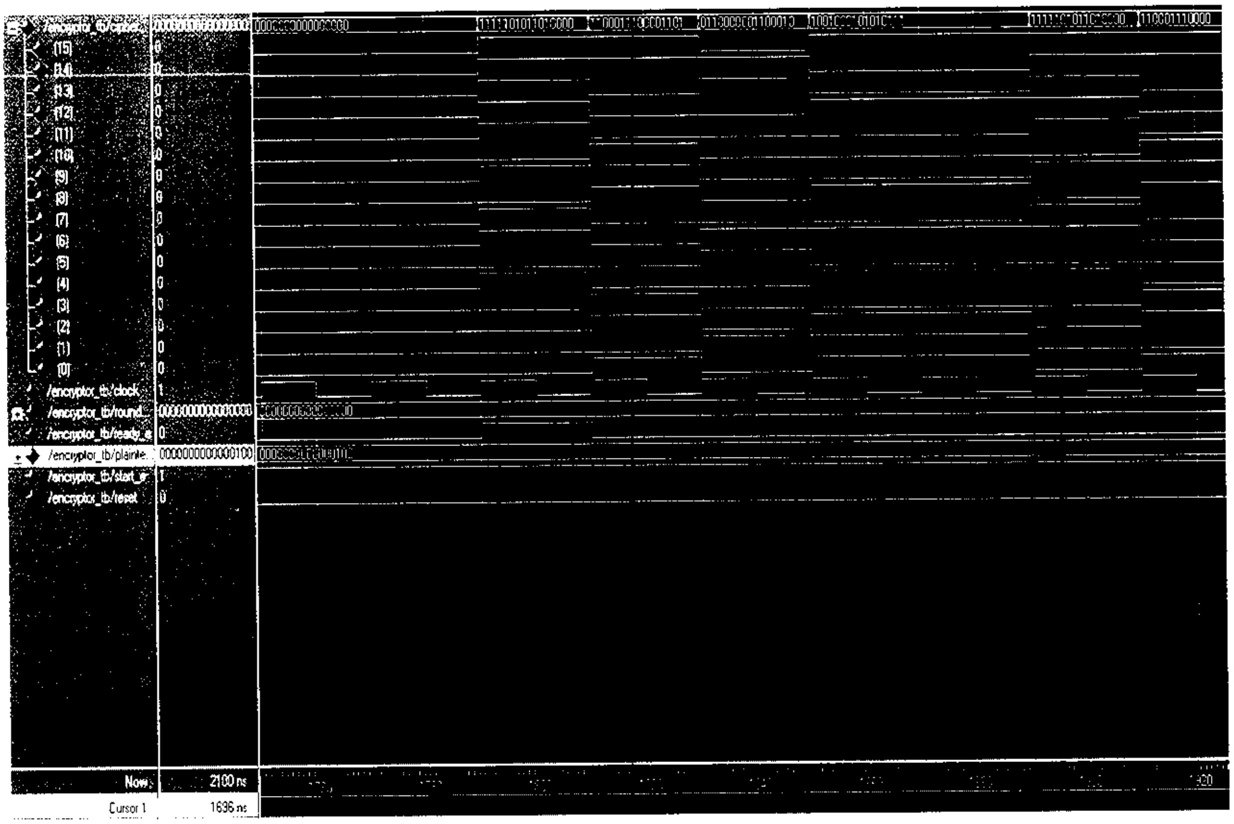


Fig 7.2 Simulation result for encryption

7.1.3 SIMULATION RESULTS FOR DECRYPTION

The simulation result for decryption is shown in Fig 7.3 below. The given four 16 bit cipher text data is decrypted into a four 16 bit plain text. The decryption module is designed in such a way it works when the control input is 1 (high) and reset is 0 (zero) on execution.

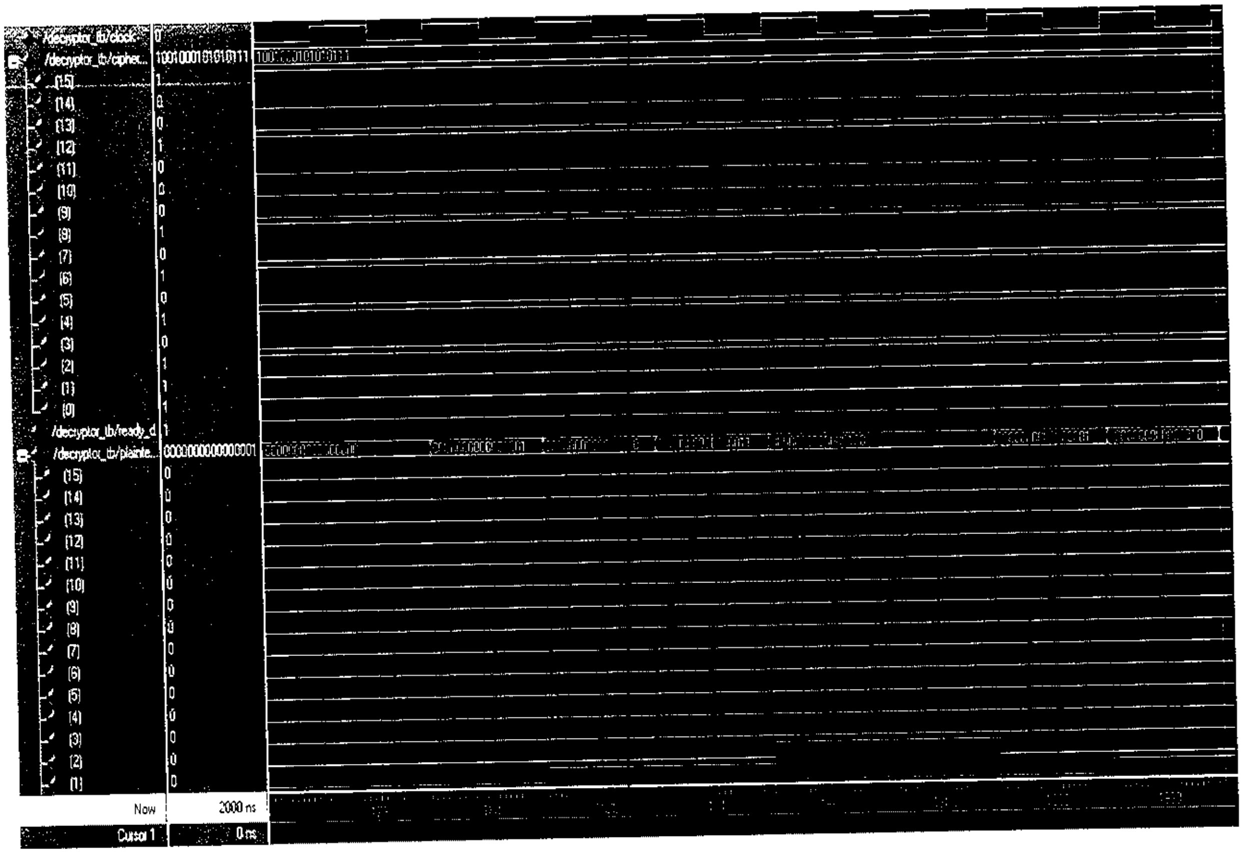


Fig 7.3 Simulation result for decryption

CHAPTER 8

HARDWARE IMPLEMENTATION

8.1 SYNTHESIS REPORT

Synthesis is the process of converting the VHDL code into technology specific netlist. And also certain user constraints during the synthesis, like placement, timing etc. The synthesis reports of the module are follows.

8.1.1 SYNTHESIS REPORT OF ENCRYPTION

1) Synthesis Options Summary
2) HDL Compilation
3) HDL Analysis
4) HDL Synthesis
   4.1) HDL Synthesis Report
5) Advanced HDL Synthesis
   5.1) Advanced HDL Synthesis Report
6) Low Level Synthesis
7) Final Report
   7.1) Device utilization summary
   7.2) TIMING REPORT

Synthesis Options Summary

Source Parameters

Input File Name	: "encryptor.prj"
Input Format	: mixed

Ignore Synthesis Constraint File	: NO

Target Parameters

Output File Name	: "encryptor"
Output Format	: NGC
Target Device	: xc3s500e-5-ft256

Source Options

Top Module Name	: encryptor
Automatic FSM Extraction	: YES
FSM Encoding Algorithm	: Auto
FSM Style	: lut
RAM Extraction	: Yes
RAM Style	: Auto
ROM Extraction	: Yes
Mux Style	: Auto
Decoder Extraction	: YES
Priority Encoder Extraction	: YES
Shift Register Extraction	: YES
Logical Shifter Extraction	: YES
XOR Collapsing	: YES
ROM Style	: Auto
Mux Extraction	: YES
Resource Sharing	: YES
Multiplier Style	: auto

Automatic Register Balancing	: No

Target Options

Add IO Buffers	: YES
Global Maximum Fanout	: 500
Add Generic Clock Buffer (BUFG)	: 8
Register Duplication	: YES
Slice Packing	: YES
Pack IO Registers into IOBs	: auto
Equivalent register Removal	:YES

General Options

Optimization Goal	: Speed
Optimization Effort	: 1
Keep Hierarchy	: NO
RTL Output	: Yes
Global Optimization	: AllClockNets
Write Timing Constraints	: NO
Hierarchy Separator	: /
Bus Delimiter	: <>
Case Specifier	: maintain
Slice Utilization Ratio	: 100
Slice Utilization Ratio Delta	: 5

Other Options

lso	: encryptor.lso

```
Read Cores                          : YES
cross_clock_analysis                : NO
verilog2001                         : YES
safe_implementation                 : No
Optimize Instantiated Primitives    : NO
use_clock_enable                    : Yes
use_sync_set                        : Yes
use_sync_reset                      : Yes
```

HDL Synthesis Report

```
Macro Statistics
# Adders/Subtractors                : 130
 16-bit adder                       : 130
# Counters                          : 1
 7-bit up counter                   : 1
# Registers                         : 14
 1-bit register                     : 2
 16-bit register                    : 12
# Latches                           : 1
 16-bit latch                       : 1
# Comparators                       : 1
 8-bit comparator less              : 1
# Multiplexers                      : 2
 16-bit 16-to-1 multiplexer         : 2
# Xors                              : 78
```

```
 1-bit xor2                          : 62
 16-bit xor2                         : 2
 16-bit xor3                         : 14
```

Advanced HDL Synthesis

```
Macro Statistics
# FSMs                               : 2
# Adders/Subtractors                 : 130
 16-bit adder                        : 130
# Counters                           : 1
 7-bit up counter                    : 1
# Registers                          : 177
 Flip-Flops                          : 177
# Latches                            : 1
 16-bit latch                        : 1
# Comparators                        : 1
 8-bit comparator less               : 1
# Multiplexers                       : 2
 16-bit 16-to-1 multiplexer          : 2
# Xors                               : 78
 1-bit xor2                          : 62
 16-bit xor2                         : 2
 16-bit xor3                         : 14
```

Final Report

```
Final Results
RTL Top Level Output File Name          : encryptor.ngr
Top Level Output File Name              : encryptor
Output Format                           : NGC
Optimization Goal                       : Speed
Keep Hierarchy                          : NO
```

Design Statistics

```
# IOs                                   : 52
```

Cell Usage:

```
# BELS                                  : 6796
#      GND                              : 1
#      INV                              : 406
#      LUT1                             : 274
#      LUT2                             : 1368
#      LUT2_D                           : 26
#      LUT2_L                           : 21
#      LUT3                             : 117
#      LUT3_D                           : 29
#      LUT3_L                           : 34
#      LUT4                             : 366
#      LUT4_D                           : 49
#      LUT4_L                           : 168
#      MULT_AND                         : 2
#     MUXCY                        : 1946
#     MUXF5                        : 44
#     VCC                          : 1
#     XORCY                        : 1944
#     FlipFlops/Latches            : 226
#     FDC                          : 100
#     FDCE                         : 5
#     FDE                          : 105
#     LD_1                         : 16
# Clock Buffers                    : 1
#     BUFGP                        : 1
# IO Buffers                       : 51
#     IBUF                         : 34
#     OBUF                         : 17
```

Device utilization summary:

```
Selected Device                    : 3s500eft256-5
Number of Slices                   : 1479 out of   4656    31%
 Number of Slice Flip Flops        : 210 out of   9312     2%
 Number of 4 input LUTs            : 2452 out of   9312    26%
 Number of bonded IOBs             : 52 out of    190    27%
 IOB Flip Flops                    : 16
 Number of GCLKs                   : 1 out of     24     4%
```

Timing Summary:

```
Minimum period: 12.348ns (Maximum Frequency: 80.982MHz)
```

```
   Maximum combinational path delay: No path found


Total memory usage is 170080 kilobytes
```

8.1.2 SYNTHESIS REPORT OF DECRYPTION

Synthesis Options Summary

Source Parameters

Input File Name	: "decryptor.prj"
Input Format	: mixed
Ignore Synthesis Constraint File	: NO

Target Parameters

Output File Name	: "decryptor"
Output Format	: NGC
Target Device	: xc3s500e-5-ft256

S Options

```
Top Module Name                      : decryptor
Automatic FSM Extraction             : YES
FSM Encoding Algorithm               : Auto
FSM Style                            : lut
RAM Extraction                       : Yes
RAM Style                            : Auto
ROM Extraction                       : Yes
Mux Style                            : Auto
Decoder Extraction                   : YES
Priority Encoder Extraction          : YES
Shift Register Extraction            : YES
Logical Shifter Extraction           : YES
XOR Collapsing                       : YES
ROM Style                            : Auto
Mux Extraction                       : YES
Resource Sharing                     : YES
Multiplier Style                     : auto
Automatic Register Balancing         : No
```

Target Options

```
Add IO Buffers                       : YES
Global Maximum Fanout                : 500
Add Generic Clock Buffer (BUFG)      : 8
Register Duplication                 : YES
Slice Packing                        : YES
```

Pack IO Registers into IOBs	: auto
Equivalent register Removal	: YES

General Options

Optimization Goal	: Speed
Optimization Effort	: 1
Keep Hierarchy	: NO
RTL Output	: Yes
Global Optimization	: AllClockNets
Write Timing Constraints	: NO
Hierarchy Separator	: /
Bus Delimiter	: <>
Case Specifier	: maintain
Slice Utilization Ratio	: 100
Slice Utilization Ratio Delta	: 5

Other Options

lso	: decryptor.lso
Read Cores	: YES
cross_clock_analysis	: NO
verilog2001	: YES
safe_implementation	: No
Optimize Instantiated Primitives	: NO
use_clock_enable	: Yes
	: Yes

```
use_sync_reset                    : Yes
```

HDL Synthesis Report

```
Macro Statistics
# Adders/Subtractors              : 130
 16-bit adder                     : 124
 16-bit subtractor                : 6
# Counters                        : 1
 7-bit up counter                 : 1
# Registers                       : 12
 1-bit register                   : 1
 16-bit register                  : 11
# Comparators                     : 1
 8-bit comparator less            : 1
# Logic shifters                  : 2
 16-bit shifter logical right     : 2
# Xors                            : 78
 1-bit xor2                       : 62
 16-bit xor2                      : 2
 16-bit xor3                      : 14
```

Advanced HDL Synthesis Report

```
Macro Statistics
# FSMs                            : 2
# Adders/Subtractors              : 130
```

16-bit adder	: 124
16-bit subtractor	: 6
# Counters	: 1
7-bit up counter	: 1
# Registers	: 184
Flip-Flops	: 184
# Comparators	: 1
8-bit comparator less	: 1
# Logic shifters	: 2
16-bit shifter logical right	: 2
# Xors	: 78
1-bit xor2	: 62
16-bit xor2	: 2
16-bit xor3	: 14

Final Report

Final Results

RTL Top Level Output File Name	: decryptor.ngr
Top Level Output File Name	: decryptor
Output Format	: NGC
Optimization Goal	: Speed
Keep Hierarchy	: NO

Design Statistics

# IOs	: 52

Cell Usage:

```
# BELS                    : 6788
#    GND                  : 1
#    INV                  : 405
#    LUT1                 : 276
#    LUT2                 : 1397
#    LUT2_D               : 20
#    LUT2_L               : 43
#    LUT3                 : 75
#    LUT3_D               : 18
#    LUT3_L               : 75
#    LUT4                 : 319
#    LUT4_D               : 48
#    LUT4_L               : 149
#    MUXCY                : 1946
#    MUXF5                : 65
#    VCC                  : 1
#    XORCY                : 1950
# FlipFlops/Latches       : 221
#    FDC                  : 120
#    FDCE                 : 5
#    FDE                  : 96
# Clock Buffers           : 1
#    BUFGP                : 1
# IO Buffers              : 51
#    IBUF                                  : 34
#    OBUF                                  : 17

                          Device utilization summary:

Selected Device                            : 3s500eft256-5


Number of Slices                           : 1458 out of   4656    31%
Number of Slice Flip Flops                 : 221 out of   9312     2%
Number of 4 input LUTs                     : 2420 out of   9312    25%
Number of bonded IOBs                      : 52 out of    190    27%
Number of GCLKs                            : 1 out of     24     4%


                               Timing Summary:

Minimum period: 12.589ns (Maximum Frequency: 79.433MHz)
Maximum combinational path delay: No path found


Total memory usage is 171104 kilobytes
```

8.2 DESIGN SUMMARY OF ENCRYPTION

The design summary of encryption is shown in Fig 8.1 below. This indicates the number of IOB latches, input LUTs and the percentage the device resources used.

ENCRYPTION Project Status

Project File:	encryption.ise	**Current State:**	Programming File Generated
Module Name:	Encryptor	• **Errors:**	No Errors
Target Device:	xc3s500e-5ft256	• **Warnings:**	37 Warnings
Product Version:	ISE, 8.1.03i	• **Updated:**	Wed Mar 24 21:29:08 2010

ENCRYPTION Partition Summary

No Partition information was Found

Current Errors

No Errors Found

Device Utilization Summary

Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	210	9,312	2%	
Number of 4 input LUTs	2,584	9,312	27%	
Logic Distribution				
Number of occupied Slices	1,514	4,656	32%	
Number of Slices containing only related logic	1,514	1,514	100%	
Number of Slices containing unrelated logic	0	1,514	0%	
Total Number 4 input LUTs	2,894	9,312	31%	
Number used as logic	2,584			

Number used as a route-thru	310		
Number of bonded IOBs	52	190	27%
IOB Latches	16		
Number of GCLKs	1	24	4%
Total equivalent gate count for design	29,040		
Additional JTAG gate count for IOBs	2,496		

Performance Summary

Final Timing Score:	0	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Clocked Report

Clock Net	**Resource**	**Locked**	**Fanout**	**Net skew(ns)**	**Max Delay(ns)**
Clk_BUFGP	BUFGMUX_X2Y0	NO	176	.059	0.16

Detailed Reports

Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Wed Mar 24 21:27:19 2010	0	35Warnings	1 Infos
Translation Report	Current	Wed Mar 24 21:27:48 2010	0	0	0
Map Report	Current	Wed Mar 24 21:28:02 2010	0	1 Warning	1 Infos
Place and Route Report	Current	Wed Mar 24 21:28:39 2010	0	0	1 Infos
Static Timing Report	Current	Wed Mar 24 21:28:48 2010	0	0	1 Infos
Bitgen Report	Current	Wed Mar 24 21:29:08 2010	0	1 Warning	0

Secondary Report

Report Name	**Status**	**Generated**
Xplorer Report		

Fig 8.1 Design Summary of Encryption

8.3 DESIGN SUMMARY OF DECRYPTION

The design summary of decryption is shown in Fig 8.2 below. This indicates the number of IOB latches, input LUTs and the percentage the device resources used.

DECRYPTIONCODES Project Status

Project File:	decryptioncodes.ise	**Current State:**	Programming File Generated
Module Name:	decryptor	• **Errors:**	No Errors
Target Device:	xc3s500e-5ft256	• **Warnings:**	29 Warnings
Product Version:	ISE, 8.1.03i	• **Updated:**	Wed Mar 24 23:38:18 2010

DECRYPTIONCODES Partition Summary

No Partition information was Found

Current Errors

No Errors Found

Device Utilization Summary

Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	221	9,312	2%	
Number of 4 input LUTs	2,548	9,312	27%	
Logic Distribution				
Number of occupied Slices	1,507	4,656	32%	
Number of Slices containing only related logic	1,507	1,507	100%	
Number of Slices containing unrelated logic	0	1,507	0%	
Total Number 4 input LUTs	2,862	9,312	30%	
Number used as logic	2,548			
Number used as a route-thru	314			

Number of bonded IOBs	52	190	27%
Number of GCLKs	1	24	4%
Total equivalent gate count for design	28,915		
Additional JTAG gate count for IOBs	2,496		

Performance Summary

Final Timing Score:	0	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Failing Constraints

All Constraints Were Met

Clock Report

Clock Net	Resource	Locked	Fanout	Net Skew(ns)	Max Delay(ns)
clock_BUFGP	BUFGMUX_X2Y0	No	180	0.049	0.153

Detailed Reports

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Wed Mar 24 23:34:30 2010	0	29 Warnings	3 Infos
Translation Report	Current	Wed Mar 24 23:34:44 2010	0	0	0
Map Report	Current	Wed Mar 24 23:34:53 2010	0	0	2 Infos
Place and Route Report	Current	Wed Mar 24 23:35:29 2010	0	0	2 Infos
Static Timing Report	Current	Wed Mar 24 23:35:36 2010	0	0	2 Infos
Bitgen Report	Current	Wed Mar 24 23:35:53 2010	0	0	0

Secondary Report

Report Name	Status	Generated
Xplorer Report		

Fig 8.2 DESIGN SUMMARY OF DECRYPTION

8.4 RTL SCHEMATIC OF THE DIFFUSION MULTIPLIER FUCTION

The RTL schematic of the Diffusion multiplier function used in the module is shown in the Fig 8.3 shown below. The module contains Input data of 16 bits and the output multiplier product of 16 bits.

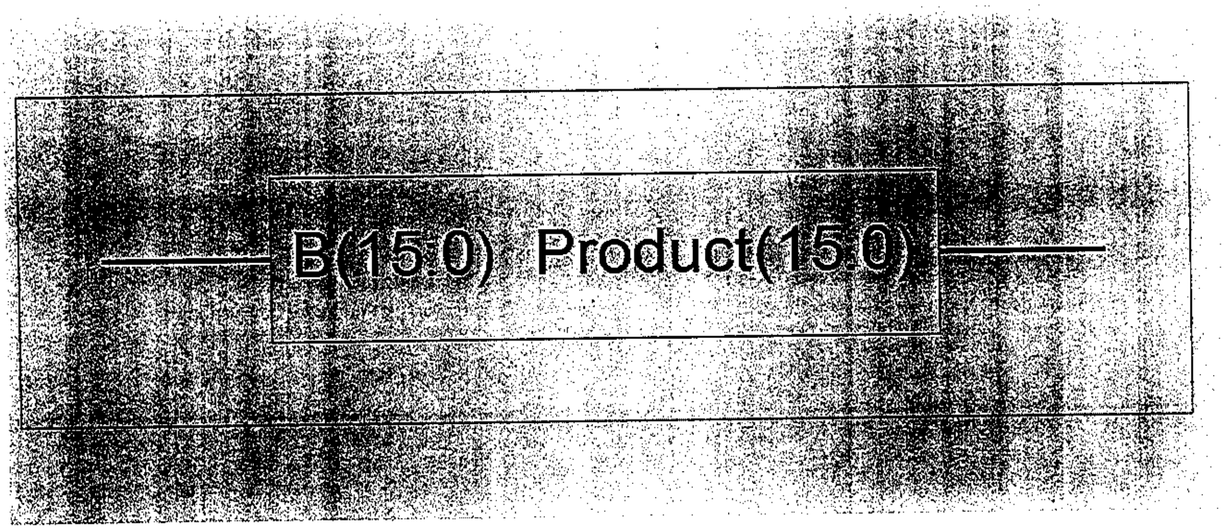


Fig 8.3 RTL Schematic of the Diffusion multiplier function

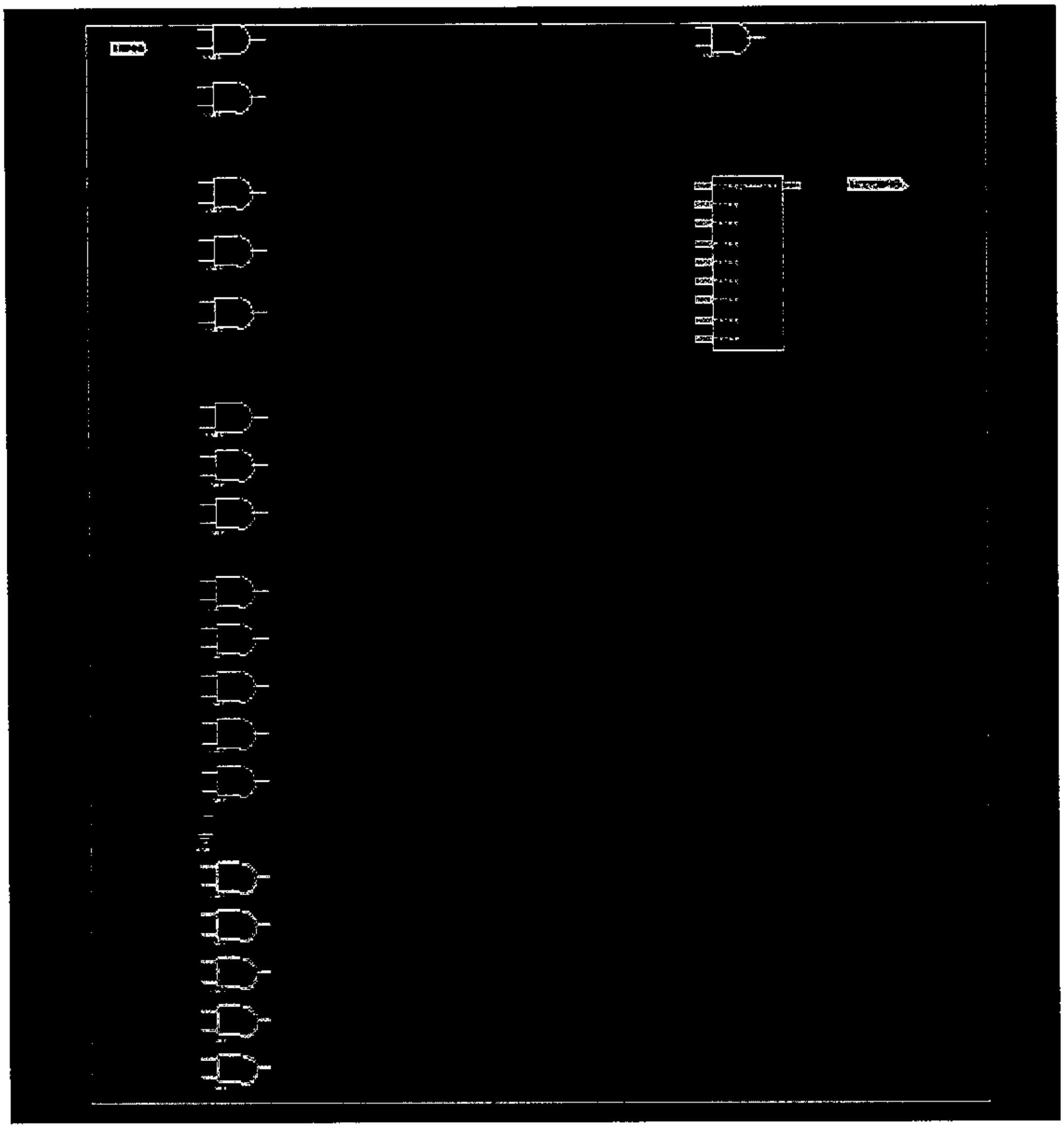


Fig 8.4 RTL Schematic of the Diffusion multiplier function Enlarged

8.5 RTL SCHEMATIC OF ENCRYPTION

The RTL schematic of encryption and its enlarged version are shown in the Fig 8.5 below. This indicates that when an input data of plain text (16-bits), round key (16-bits) and control inputs are provided to the module, the output cipher text (16-bits) is obtained. A four 16-bits input of data (64-bits) can be encrypted using this module.

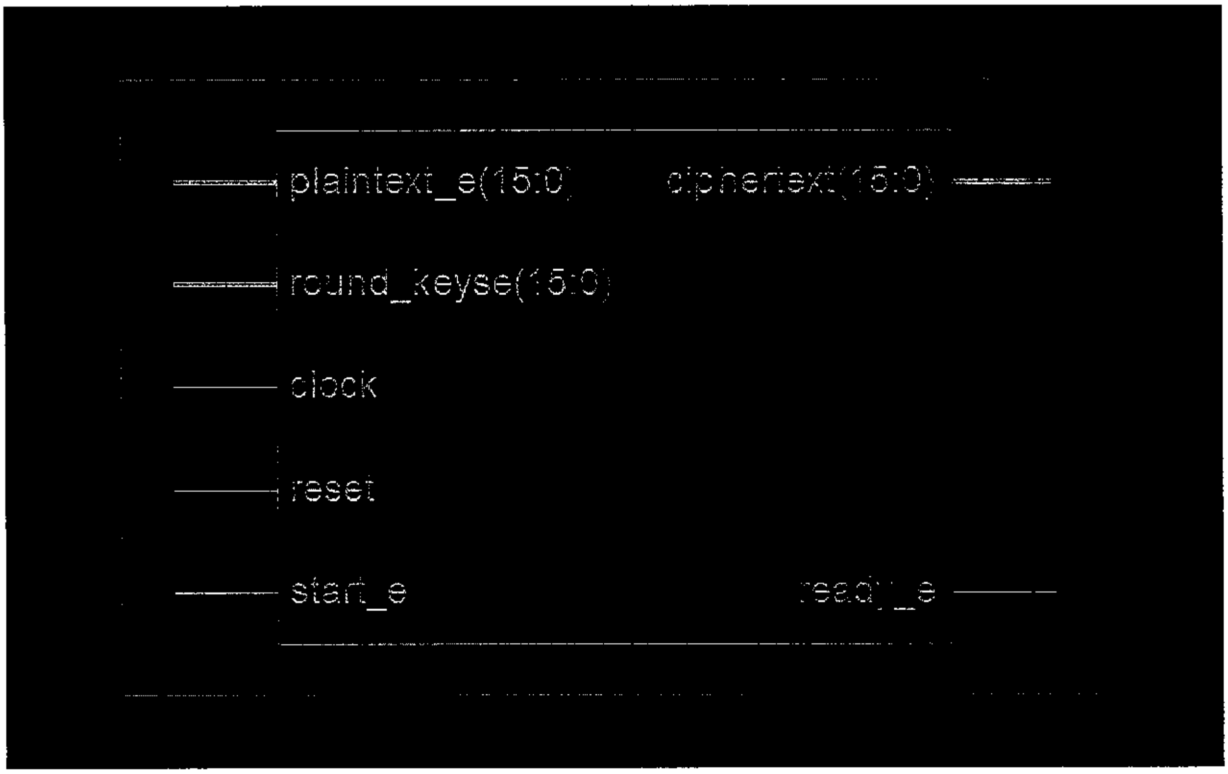


Fig 8.5 RTL Schematic of Encryption

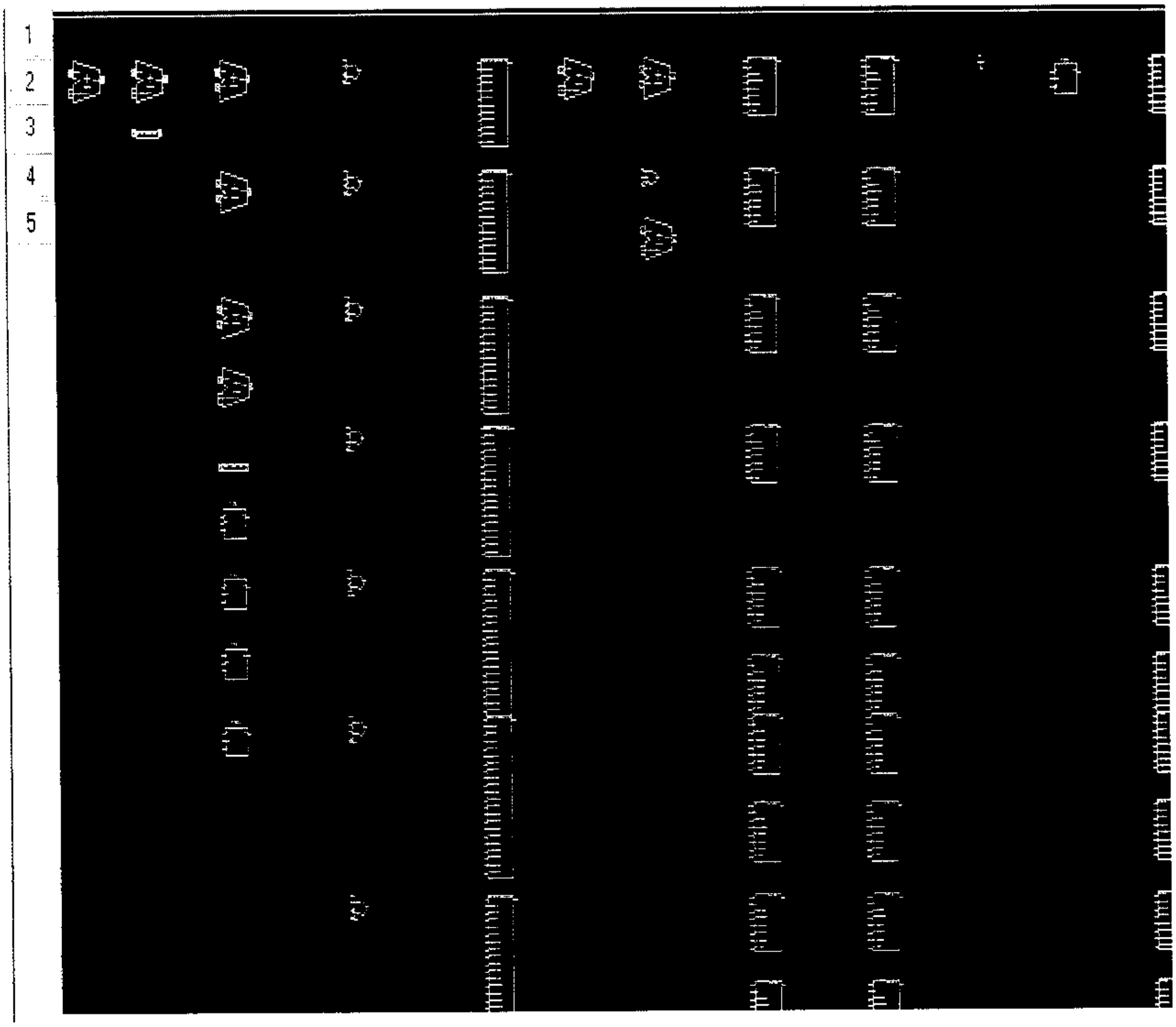


Fig 8.6 RTL Schematic of Encryption Enlarged-1

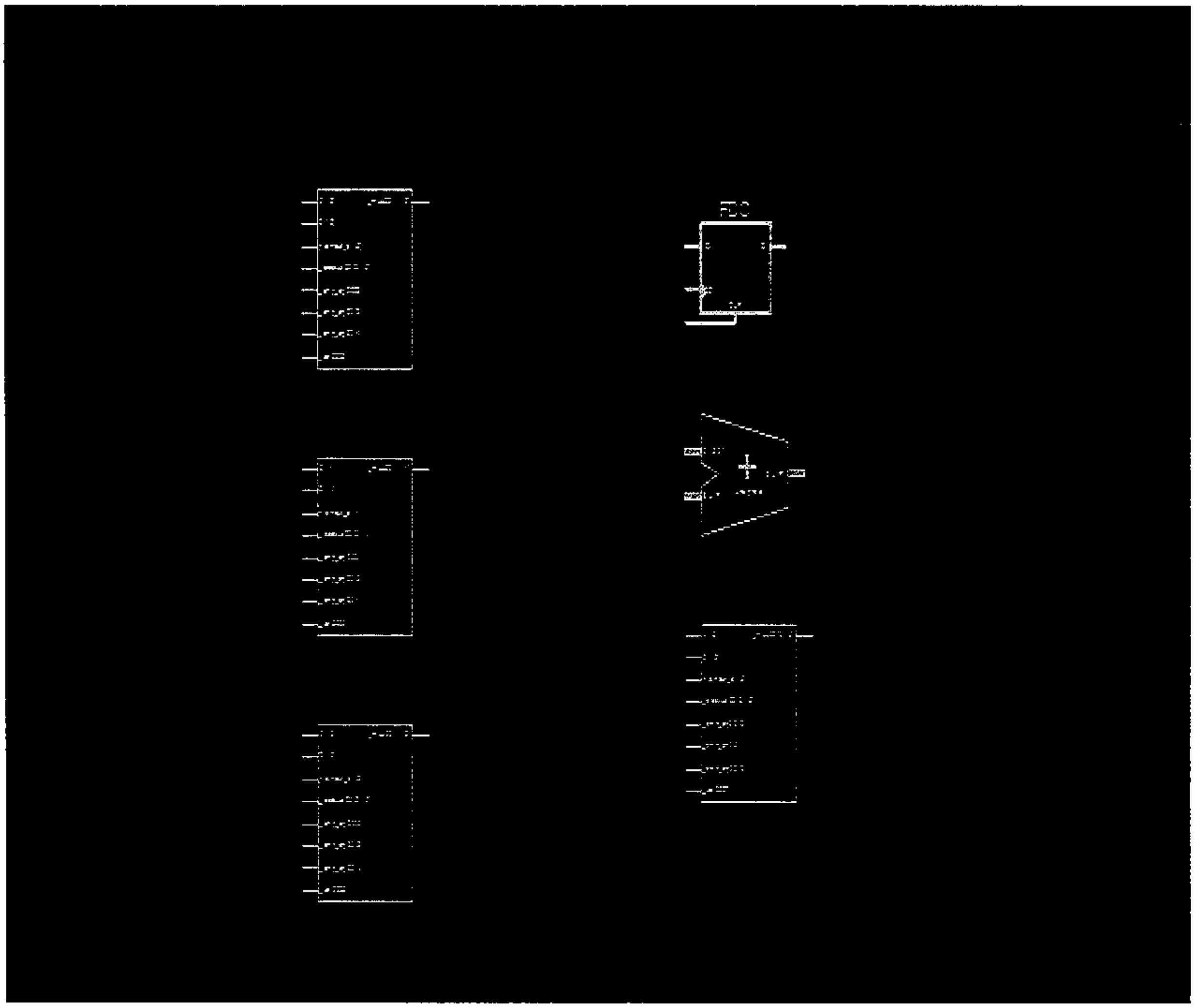


Fig 8.7 RTL Schematic of Encryption Enlarged-2

8.6 RTL SCHEMATIC OF DECRYPTION

The RTL schematic of decryption and its enlarged version are shown in the Fig 8.8 below. This indicates that when an input data of cipher text (16-bits), round key (16-bits) and control inputs are provided to the module, the output plain text (16-bits) is obtained. A four 16-bits input of data (64-bits) can be decrypted using this module.

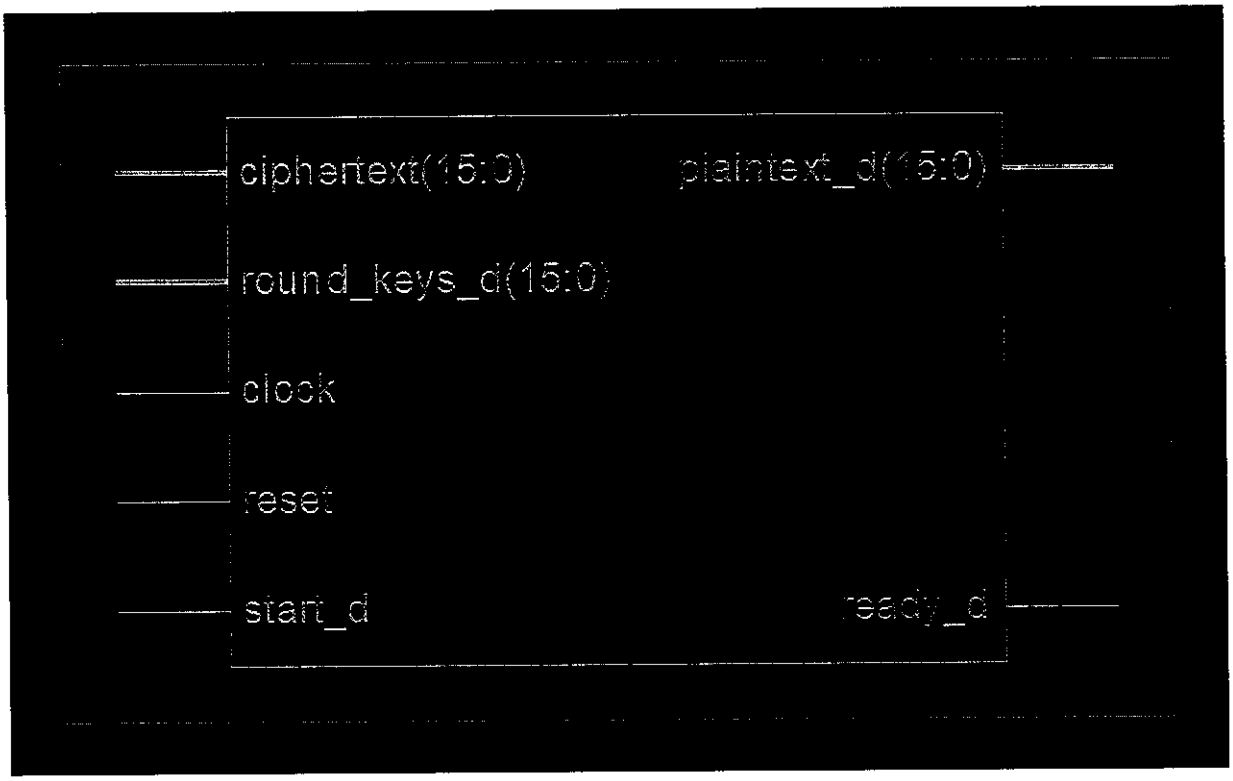


Fig 8.8 RTL Schematic of Decryption

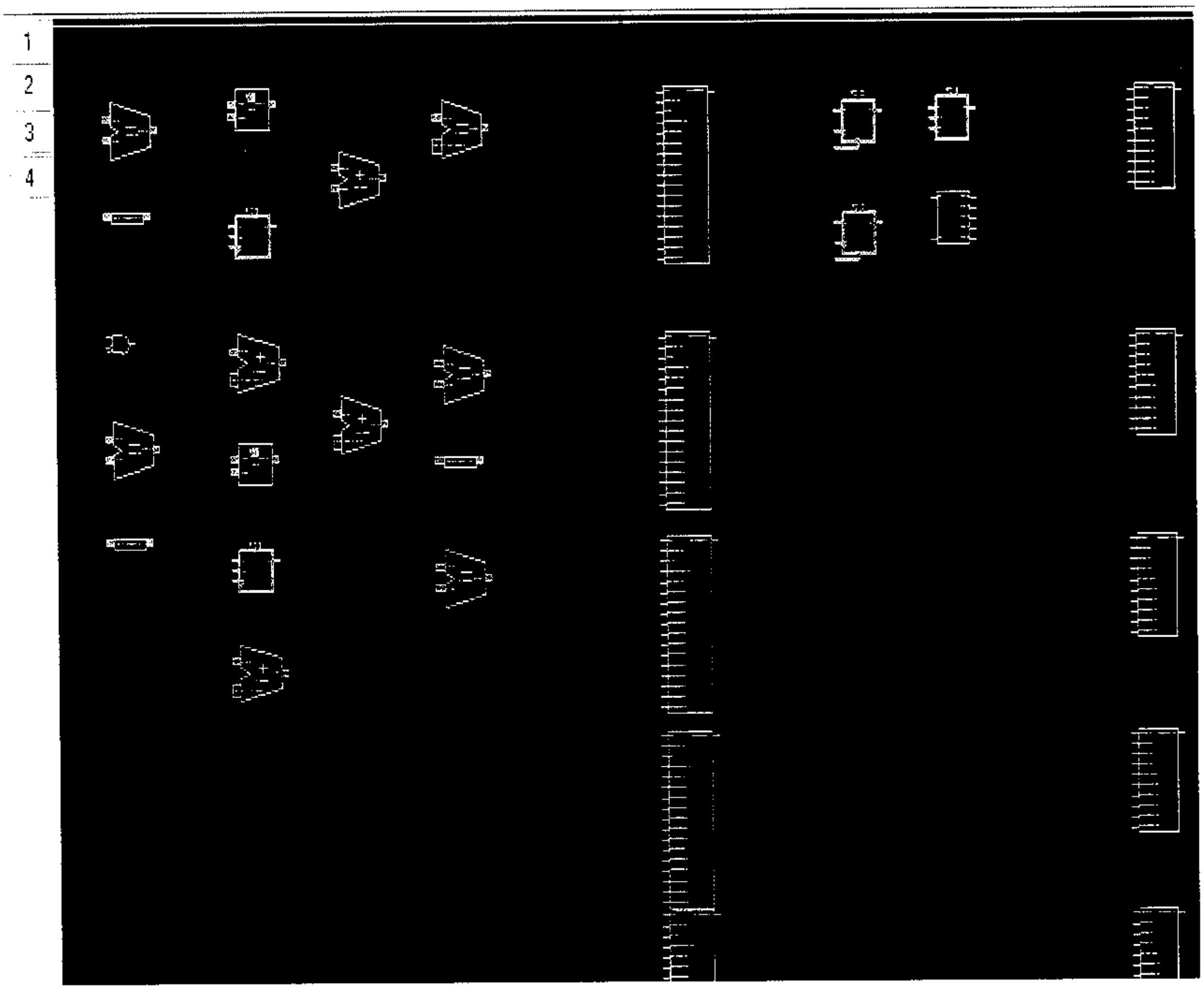


Fig 8.9 RTL Schematic of Decryption Enlarged-1

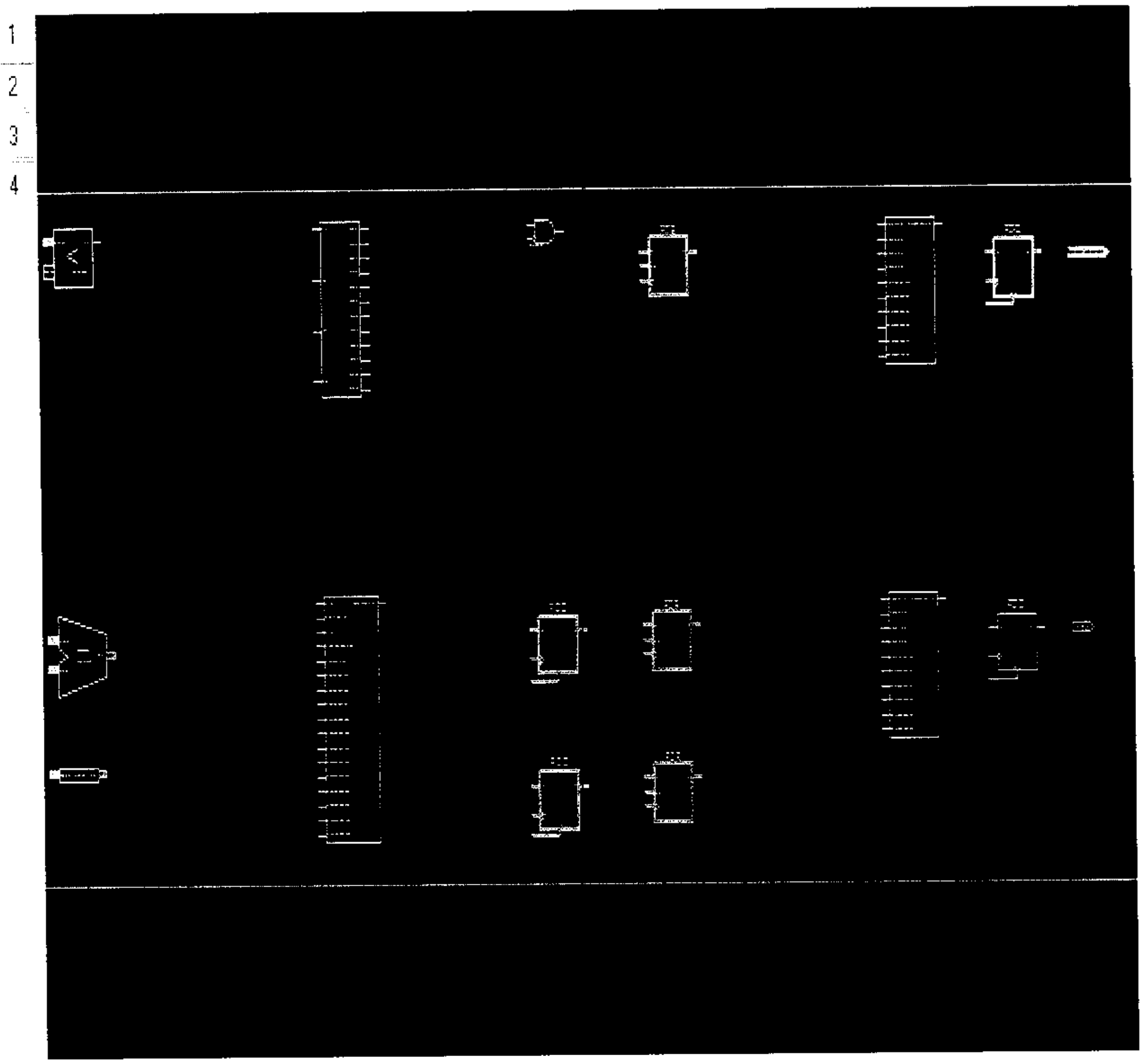


Fig 8.10 RTL Schematic of Decryption Enlarged-2

CHAPTER 9

CONCLUSION

Implementation of high performance RC6 algorithm with enhanced security; high speed and low memory cryptographic algorithm have been implemented. The Design is implemented in FPGA device. The Design uses pipeline technique shows an improvement in the speed of operation. From the above results it's evident that RC6 have best performance over other cryptographic algorithms.

The upcoming year will reveal the RC6 algorithm as the advanced encryption standard by NIST. The modern devices and smart cards and other security systems will be using RC6 algorithm in it for encryption and decryption of data.

CHAPTER 10

DISCUSSIONS

A distinguishing feature of RC6 algorithm is its heavy use of data-dependent rotations. The amount of rotation performed is dependent on the input data, and is not predetermined.

The encryption/decryption routines are very simple. While other operations (such as substitution operations) could have been included in the basic round operations, our objective is to focus on the data-dependent rotations as a source of cryptographic strength.

Some of the expanded key table S is initially added to the plaintext, and each round ends by adding expanded key from S to the intermediate values just computed. This assures that each round acts in a potentially different manner, in terms of the rotation amounts used.

The XOR operations done back and forth between the data provide some avalanche properties, causing a single-bit change in an input block to cause multiple-bit changes in following rounds which is use for Diffusion function.

Another distinguishing feature of RC6 algorithm is its simplicity and low memory requirement while other algorithms having high memory requirement cannot be implemented in all environments. Other algorithms like AES, DES, Triple DES, MARS, Serpent, Blowfish algorithms fails in its implementation in low memory requirement.

CHAPTER 11

FUTURE EXPANSION

The Implementation in hardware level with increased speed can be obtained by using the pipelined architecture in its structure. The bit size can be increased to increase more security. The implementation in the project is for fixed data length and key size. The data size and key size can be made variable for future implementation.

The RC6 algorithm can be made hard to be hacked by inserting a symmetric layer between encryption and decryption process. Insertion of symmetric layer improves security, speed and decreases the memory of the algorithm. This improved RC6 algorithm can be used in light mobile devices and RFIDs

CHAPTER 12

REFERENCES

1. N. Sklavos, C. Machas, O. Koufopavlou, "Area Optimized Architecture and VLSI Implementation of RC5 Encryption Algorithm", proceedings of the 10th IEEE conference on Electronics, circuits and systems, UAE, December 14-17, 2003
2. Rivest, R.L., Robshaw, M.J.B., Sidney, R., & Yin, Y.L (1998a). "The RC6 Block Cipher." . URL: ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/rc6v11.pdf
3. Rivest. R.L., Robshaw. M.J.B., Sidney, R., & Yin, Y.L (1998b). "The Security of the RC6 Block Cipher.".
   URL: ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/security.pdf
4. Nechvatal. J., Barker. E., Bassham.L., Burr.W., Dworkin.M., Foti. J., & Roback.E. (2000). "Report on the Development of the Advanced Encryption Standard (AES)." National Institute of Standards and Technology (Department of Commerce).
   URL: http://csrc.nist.gov/CryptoToolkit/aes/round2/r2report.pdf
5. Rivest, R.L (1997). "The RC5 Encryption Algorithm."
   URL: http://theory.lcs.mit.edu/%7Erivest/Rivest-rc5rev.pdf
6. R.L. Rivest, M.J.B. Robshaw, R.Sidney, and Y.L. Yin. The RC6 Block Cipher.
   URL:http://theory.lcs.mit.edu/~rivest/rc6.pdf
7. J. Beuchat FPGA Implementations of the RC6 Block Cipher
   URL:http://perso.ens-lyon.fr/jean-luc.beuchat/Publications/fpl2003.pdf