Cryptography Algorithm On Reconfigurable Hardware

Major Project Report

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology

In

VLSI Design

By

MANSUR RIYAZ LATIFBHAI

(11MECV07)

Department of Electrical Engineering

Electronics and Communication Programme

Institute of Technology,

Nirma University, Ahmedabad-382 481

MAY 2013

Cryptography Algorithm On Reconfigurable Hardware

Major Project Report

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology

In

Electronics & Communication

(Vlsi Design)

By

MANSUR RIYAZ LATIFBHAI

(11MECV07)

Under the Guidance of

Prof. N. P. Gajjar

Department of Electrical Engineering

Electronics and Communication Programme

Institute of Technology,

Nirma University, Ahmedabad-382 481

MAY 2013

Declaration

.

This is to certify that

(i) The thesis comprises my original work towards the degree of Master of Technology in VLSI Design at Nirma University and has not been submitted elsewhere for a degree.

(ii) Due acknowledgement has been made in the text to all other material used.

MANSUR RIYAZ LATIFBHAI

Certificate

This is to certify that the Major Project entitled **"Cryptography Algorithm On Reconfigurable Hardware"** submitted by **MANSUR RIYAZ LATIFBHAI**, towards the partial fulfilment of the requirements for the degree of Master of Technology in Communication Engineering of Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place:Ahmedabad

Internal Guide

Prof.N P Gajjar

(Associate Prof.)

Program Coordinator	**Head of Department**	**Director**
Dr. N M Devashrayee	Dr. P N Tekwani	Dr. K Kotecha

Acknowledgement

First of all I thank almighty God for providing me good health during the whole project work.

I would like to take the opportunity to extend my gratitude to the management of Institute of Technology, Nirma University for providing us healthy working environment throughout the project work. **Prof. N. P. Gajjar** is the main source of motivation for me in the project. He guided me at every step of the work I have carried out. I would like to thank Prof. N. P. Gajjar for his valuable guidance and support. I again would like to thank Prof. N. P. Gajjar for allowing me to use the resources of ISRO Respond Lab for carrying out the project. I would like to thank **Dr. N. M. Devashrayee** for his continuous support throughout the project.

I would also like to extend my gratitude to all teaching and non teaching staff members and my student mates who were helpful to me in direct or indirect way. At last but not least, I thank my family for their support and love.

Abstract

The Cryptographic Algorithm which is most widely used throughout the world for protecting information. Cryptography is the art of secret writing, followed by the guarantee to authenticate data and messages and protect the systems from valid attacks .It comprises of encryption and decryption operations each associated with a key which is supposed to be kept secret. We have implement RC6 Algorithm. Which is considered as a secured and elegant choice for AES. RC6 supports 32 bit and 64 bit processing. An eight step operation is used to encipher the 64 bit plain text block. The encrypted data is then decrypted by performing the reverse operations on the same.The hardware implementation of RC6 algorithm is done using VHDL Hardware Description Language.

The choice of reconfigurable logic as a target platform for cryptographic algorithm implementations appears to be a practical solution for embedded systems and high-speed applications. Both efficient and cost effective solutions of cryptographic algorithms are desired on reconfigurable logic platform. The term"efficient" normally refers to "high speed" solutions.That implies careful considerations of cryptographic algorithm formulations, which often will lead to modify the traditional specifications of those algorithms. That also implies knowledge of the target device: devicestructure, device resources, and device suitability to the given task.

Contents

Declaration **1**

Certificate **2**

Acknowledgement **3**

Abstract **4**

List of Figures **8**

List of Tables **10**

1 Introduction **11**

1.1 Main Goal 12

2 Background Information **14**

2.1 RC6 Algorythm 14

2.1.1 Why We have selected RC6 Algorythm 15

2.2 Strength of the Cryptosystem 16

2.3 Types of Cryptographic Algorythm 17

2.3.1 Symmetric Cryptographic Algorythm 17

2.3.2 Stream Cipher 18

2.3.3 Block Cipher 18

2.3.4 Drawbacks of Secret key Cryptography[4] 19

2.3.5 Asymmetric Cryptographic Algorythms (Public Key Algorithms): 21

2.3.6 Difference between Symmetric and Asymmetric Algorythms . 22

2.4 Basic of RC6 Algorythm 23

2.4.1 RC6 Block Diagram 23

2.4.2 Basic Operation 23

2.4.3 Basic steps for RC6 Algorythm 24

2.5 Encryption 25

2.5.1 Signal Specification 25

2.5.2 RC6 Encryption Algorythm 26

2.6 Decryption 27

3 Theoritical Analysis 30

3.1 Modules Block Diagrams 30

3.2 Key Scheduling[8] 31

3.2.1 Key scheduling operation 32

3.3 Finite State Machines 33

3.3.1 Introduction 33

3.3.2 Concept of Finite State Machine 33

3.4 Multiplier 37

3.4.1 Wallace Tree 38

3.4.2 Carry Save Adder 39

3.4.3 Carry Look Ahead Method 41

3.5 Swap Operation 42

4 Reconfigurable Hardware Technology 44

4.1 Introduction 44

4.1.1 Why Reconfigurable H/W Technology??[10] 44

4.1.2 Applications of FPGAs 45

4.2 FPGA Design Flow 45

4.3 AES Cipher Core 46

4.4 Timing Diagram 47

5 Implementation & Simmulation Result 48

5.1 How to Simulate & Implement RC6 Algorythm? 48

5.2 Testing Methodology 48

5.2.1 About RTL Schematic 49

5.2.2 RTL Schematic for Encryptior 49

5.2.3 RTL Schematic for Decryptior 50

5.3 About Test Bench Waveform 50

5.4 Steps for Hardware Implementation of RC6 Algorythm[15] 53

5.4.1 Synthesis Report for Encryption: 58

5.4.2 Synthesis Report for Decryption: 59

5.4.3 Synthesis Report for Encryption: 60

5.4.4 Synthesis Report for Decryption: 61

5.4.5 Synthesis Report for Encryption: 62

5.4.6 Synthesis Report for Decryption: 63

Application Area 64

Conclusion 65

Bibliography 66

List of Figures

2.1 RC6 Cipher[12] 23
2.2 Basic Steps[13] 24
2.3 Encryption block diagram 25
2.4 Decryption block diagram 27

3.1 Digital circuit block diagram for Encryption 30
3.2 Digital circuit block diagram for Decryption 31
3.3 Finite state machine for input Encryption 34
3.4 Finite State Machines for input Decryption 35
3.5 Finite State Machines for output Encryption 36
3.6 Finite State Machines for output Decryption 37
3.7 Wallace tree multiplier[4] 38
3.8 Wallace structure[4] 39
3.9 Carry Save Adder[4] 40
3.10 Carry Look Ahead Method[4] 41
3.11 Block diagram for swaping[4] 43

4.1 Fpga design Flow[10] 45
4.2 AES Cipher Core[10] 46
4.3 Timing diagram[10] 47

5.1 RTL Schematic for Encryptior 49
5.2 RTL Schematic for Decryptior 50

5.3 Simulation of Encryption 51
5.4 Simulation of Decryption 52
5.5 Select the device and design flow 53
5.6 Step for Generate programming file 54
5.7 Step for Configure device 54
5.8 Step for select Boundry-Scan 55
5.9 Boundary-Scan Mode Selection 55
5.10 Downloading program in Fpga 56
5.11 Operation Status 57

List of Tables

5.1 Device utilization summary Encryption Spartan 3 58

5.2 Timing summary Encryption Spartan 3 58

5.3 Device utilization summary Decryption Spartan 3 59

5.4 Timing summary Decryption Spartan 3 59

5.5 Device utilization summary Encryption Spartan 3E 60

5.6 Timing summary Encryption Spartan 3E 60

5.7 Device utilization summary Decryption Spartan 3E 61

5.8 Timing summary Decryption Spartan 3E 61

5.9 Device utilization summary Vertex 5 Encryption 62

5.10 Timing summary Vertex 5 Encryption 62

5.11 Device utilization summary Vertex 5 Decryption 63

5.12 Timing summary Vertex 5 for Decryption 63

Chapter 1

Introduction

Encryption is the transformation of plain data (known as plaintext) into unintelligible data (known as ciphertext) through an algorithm referred to as cipher. There are numerous encryption algorithms that are now commonly used in computation, but the U.S. government has adopted the Advanced Encryption Standard (AES) to be used by Federal departments and agencies for protecting sensitive information.

The National Institute of Standards and Technology (NIST) has published the specifications of this encryption standard in the Federal Information Processing Standards (FIPS) Publication.

Any conventional symmetric cipher, such as AES, requires a single key for both encryption and decryption, which is independent of the plaintext and the cipher itself. It should be impractical to retrieve the plaintext solely based on the ciphertext and the encryption algorithm, without knowing the encryption key. Thus, the secrecy of the encryption key is of high importance in symmetric ciphers such as AES.

Software implementation of encryption algorithms does not provide ultimate secrecy of the key since the operating system, on which the encryption software runs, is always vulnerable to attacks

There are other important drawbacks in software implementation of any encryption algorithm, including lack of CPU instructions operating on very large operands, word size mismatch on different operating systems and less parallelism in software. In addition, software implementation does not fulfill the required speed for time critical encryption applications. Thus, hardware implementation of encryption algorithms is an important alternative, since it provides ultimate secrecy of the encryption key, faster speed and more efficiency through higher levels of parallelism.

1.1 Main Goal

The local electron density measurement in plasma is important for:

- The choice of reconfigurable logic as a target platform for cryptographic algorithm implementations appears to be a practical solution for embedded systems and high- speed applications.
- Both efficient and cost effective solutions of cryptographic algorithms are desired on reconfigurable logic platform. The term "efficient" normally refers to "high speed" solutions.

- So here main objective is to find high speed and low area implementations of cryptographic algorithms using reconfigurable logic devices. That implies careful considerations of cryptographic algorithm formulations, which often will lead to modify the traditional specifications of those algorithms. That also implies knowledge of the target device: device structure, device resources, and device suitability to the given task.

- So in this Cryptography algorithm like AES will be developed by using verilog HDl language and uses a FPGA as a target

Chapter 2

Background Information

Cryptography

The Cryptographic Algorithm which is most widely used throughout the world for protecting information. Cryptography is the art of secret writing, followed by the guarantee to authenticate data and messages and protect the systems from valid attacks .It comprises of encryption and decryption operations each associated with a key which is supposed to be kept secret. We have implement RC6 Algorithm. Which is considered as a secured and elegant choice for AES due to its simplicity, security, performance and efficiency. RC6 supports 32 bit and 64 bit processing. An eight step operation is used to encipher the 64 bit plain text block. The encrypted data is then decrypted by performing the reverse operations on the same. The hardware implementation of RC6 algorithm is done using VHDL Hardware Description Language. For this implementation Xilinx foundation series 13.1i software

2.1 RC6 Algorythm

We introduce the RC6 TM block cipher. RC6 is an evolutionary improvement of RC5, designed to meet the requirements of the Advanced Encryption Standard (AES). Like RC5, RC6 makes essential use of data-dependent rotations. New features of RC6 include the use of four working registers instead of two, and the inclusion of integer multiplication as an additional primitive operation. The use multiplication greatly increases the diffusion achieved per round, allowing for greater security, fewer rounds, and increased throughput.

2.1.1 Why We have selected RC6 Algorythm

RC6 is considered as a secured and elegant choice due to its simplicity, security, performance and efficiency. It appears that RC6 is best suited for implementation in the targeted Xilinx FPGA (Spartan-3). Our studies reveal that multiplication and addition are the major bottlenecks as far as speed of encryption in the RC6 cipher is Concerned. Nevertheless, up to a great extent this shortcoming was Tackled using pipelining in our design.Consequently, since RC6 works best in non-feedback mode, the highest Speed/Area ratio can be achieved in the same RC6 is a symmetric key block cipher derived from RC5. It was designed by Ron Rivest,Matt Robshaw, Ray Sidney, and Yiqun Lisa Yin to meet the requirements of the Advanced Encryption Standard (AES) competition. The algorithm was one of the five finalists, and was also submitted to the NESSIE and CRYPTREC projects. It is a proprietary algorithm, patented by RSA Security.

RC6 proper has a block size of 64 OR 128 bits and supports key sizes of 128, 192 and 256 bits, but, like RC5, it can be parameterized to support a wide variety of word-lengths, key sizes and number of rounds. RC6 is very similar to RC5 in structure, using data-dependent rotations, modular addition and XOR operations; in fact, RC6 could be viewed as interweaving two parallel RC5 encryption processes. However, RC6 does use an extra multiplication operation not present in RC5 in order to make the rotation dependent on every bit in a word, and not just the least significant few bits.

2.2 Strength of the Cryptosystem

The strength of the encryption method comes from the algorithm, secrecy of the key,length of the key, initialization vectors, and how they all work together. When strength is discussed in encryption, it refers to how hard it is to figure out the algorithm or key, whic hever is not made public. Breaking a key has to do with processing an amazing number of possible values in the hopes of finding the one value that can be used to decrypt a specific message. The strength correlates to the amount of necessary processing power and time it takes to break the key or figure out the value of the key. Breaking a key can be accomplished by a brute force attack, which means trying every possible key value until the resulting plaintext is meaningful. Depending on the algorithm and length of the key, this can be a very easy task or a task that is close to impossible. If a key can be broken with a Pentium II processor in three hours, the cipher is not strong at all.If the key can only be broken with the use of a thousand multiprocessing systems, and it takes 1.2 million years, then it is pretty darn strong.

The goal of designing an encryption method is to make compromise too expensive or too time consuming. Another name for cryptography strength is work factor, which is an estimate of the effort it would take an attacker to penetrate an encryption method.

The strength of the protection mechanism should be used in correlation to the sensitivity of the data being encrypted. It is not necessary to encrypt information about a friends Saturday barbeque with a top secret NSA encryption algorithm, and it is not a good idea to send the intercepted KGB spy information using Pretty Good Privacy (PGP). Each type of encryption mechanism has its place and purpose.

Even if the algorithm is very complex and thorough, there are other issues within encryption that can weaken the strength of encryption methods. Because the key is usually the secret value needed to actually encrypt and decrypt messages, improper protection of the key can weaken the encryption strength. An extremely strong algorithm can be used, using a large keyspace, and a large and random key value, which are all the requirements for strong encryption, but if a user shares her key with others, these other pieces of the equation really dont matter.

An algorithm with no flaws, a large key, using all possible values within a keyspace,and protecting the actual key are important elements of encryption. If one is weak, it can prove to be the weak link that affects the whole process.

2.3 Types of Cryptographic Algorythm[11]

1. Symmetric encryption algorithms (secret or private key algorithms) and
2. Asymmetric encryption algorithms (or public key algorithms).

The difference is that symmetric encryption algorithms use the same key for encryption and decryption (or the decryption key is easily derived from the encryption key), whereas asymmetric encryption algorithms use a different key for encryption and decryption.

2.3.1 Symmetric Cryptographic Algorythm

Symmetric encryption algorithms can be divided into stream ciphers and block ciphers. Stream ciphers encrypt a single bit of plaintext at a time, whereas block ciphers take a number of bits (typically 64 bits in modern ciphers), and encrypt them as a single unit.

2.3.2 Stream Cipher

Stream ciphers are an important class of encryption algorithms. They encrypt individual characters (usually binary digits) of a plaintext message one at a time, using an encryption transformation which varies with time. By contrast, block ciphers tend to simultaneously encrypt groups of characters of a plaintext message using a fixed encryption transformation. Stream ciphers are generally faster than block ciphers in hardware, and have less complex hardware circuitry. They are also more appropriate, and in some cases mandatory (e.g., in some telecommunications applications), when buffering is limited or when characters must be individually processed as they are received. Because they have limited or no error propagation, stream ciphers may also be advantageous in situations where transmission errors are highly probable.

2.3.3 Block Cipher

Symmetric-key block ciphers are the most prominent and important elements in many cryptographic systems. Individually, they provide confidentiality. As a fundamental building block, their versatility allows construction of pseudorandom number generators, stream ciphers, MACs, and hash functions. They may furthermore serve as a central component in message authentication techniques, data integrity mechanisms, entity authentication protocols, and (symmetric-key) digital signature schemes. A block cipher is a function which maps n-bit plaintext blocks to n-bit ciphertext blocks; n is called the blocklength. It may be viewed as a simple substitution cipher with large character size. The function is parameterized by a k-bit key K,1 taking values from a subset K (the key space) of the set of all k-bit vectors Vk. It is generally assumed that the key is chosen at random. Use of plaintext and ciphertext blocks of equal size avoids data expansion.

To allow unique decryption, the encryption function must be one-to-one (i.e., invertible). For n-bit plaintext and ciphertext blocks and a fixed key, the encryption function is a bijection, defining a permutation on n-bit vectors. Each key potentially defines a different bijection. The number of keys is [K], and the effective key size is lg [K]; this equals the key length if all k-bit vectors are valid keys (K = V_k). If keys are equiprobable and each defines a different bijection, the entropy of the key space is also lg [K].

2.3.4 Drawbacks of Secret key Cryptography[4]

The Drawbacks of Secret Key Cryptography:

- **Key distribution and key exchange:** The master key used in this kind of cryptosystems must be known by the sender and receiver only. Hence, both parties should prevent that this key can get compromised by unauthorized entities.

- **Key management:** Those system having many users, must generate/manage many keys. For security given key should be changed frequently, even in every session.

- **Incompleteness:** It is impossible to implement some of the security services mentioned before. In particular, Authentication and non-repudiation cannot be fully implemented by only using secret key cryptography.

.

Some examples of popular symmetric encryption algorithms:

DES (Data Encrytion Standard): DES (Data Encrytion Standard)

DES accepts a 64-bit key, the key setup routines effectively discard 8 bits, giving DES a 56-bit effective keylength.. DES was designed to be implemented only in hardware, and is therefore extremely slow in software. A recent successful effort to crack DES took several thousand computers several months.

RC5:

RC5 is a group of algorithms designed by Ron Rivest of RSA Data Security that can take on a variable block size, key size, and number of rounds. The block size is generally dependent on the word size of the machine the particular version of RC5 was designed to run on; on 32-bit processors (with 32-bit words), RC5 generally has a 64-bit block size. David Wagner, John Kelsey, and Bruce Schneier have found weak keys in RC5, with the probability of selecting a weak key to be 2-10r, where r is the number of rounds. For sufficiently large r values (greater than 10), this is not a problem as long as you are not trying to build a hash function based on RC5. Kundsen has also found a differential attack on RC5. RC5 is described in this RSA document. RC5 is patented by RSA Security, Inc.

RC6:

RC6 is Ronald Rivest's AES submission. Like all AES ciphers, RC6 works on 128 bit blocks. It can accept variable length keys. It is very similar to RC5, incorporating the results of various studies on RC5 to improve the algorithm. The studies of RC5 found that not all bits of data are used to determine the rotation amount (rotation is used extensively in RC5); RC6 uses multiplication to determine the rotation amount and uses all bits of input data to determine the rotation amount, strengthening the avalanche effect.

RIJNDEAL:

Rijndael is an AES winner by Joan Daemen and Vincent Rijmen. The cipher has a variable block and key length, and the authors have demonstrated how to extend the block length and key length by multiples of 32 bits. The design of Rijndael was influenced by the SQUARE algorithm. The authors provide a Rijndael specification and a more theoretical paper on theirdesign principles. The authors have vowed to never patent Rijndael.

RC4:

The RC4 algorithm is a stream cipher from RSA Data Security, Inc. Though RC4 was originally a trade secret, the alleged source code was published anonymously in 1994. The published algorithm performs identically to RC4 implementations in official RSA products. RC4 is widely used in many applications and is generally regarded to be secure. There are no known attacks against RC4. RC4 is not patented by RSA Data Security, Inc; it is just protected as a trade secret. The 40-bit exportable version of RC4 has been broken by brute force! RC4 is implemented in Kremlin.

MD5:

While MD4 was designed for speed, a more conservative approach was taken in the design of MD5. However, applying the same techniques he used to attack MD4, Hans Dobbertin has shown that collisions can be found for the MD5 compression function in about 10 hours on a PC. While these attacks have not been extended to the full MD5 algorithm, they still do not inspire confidence in the algorithm. RSA is quick to point out that these collision attacks do not compromise the integrity of MD5 when used with existing digital signatures. MD5, like MD4, produces a 128-bit digest. An RFC describing MD5 in detail is available here. The use of MD5, as well as MD4, is not recommended in new applications.

2.3.5 Asymmetric Cryptographic Algorythms (Public Key Algorithms):

Asymmetric encryption algorithms (public key algorithms) use different keys for encryption and decryption, and the decryption key cannot (practically) be derived from the encryption key. Public key methods are important because they can be used for transmitting encryption keys or other data securely even when the parties have no opportunity to agree on a secret key in private.

2.3.6 Difference between Symmetric and Asymmetric Algorythms

Symmetric encryption algorithms encrypt and decrypt with the same key.Main advantages of symmetric encryption algorithms are its security and high speed. Asymmetric encryption algorithms encrypt and decrypt with different keys. Data is encrypted with a public key, and decrypted with a private key. Asymmetric encryption algorithms (also known as public-key algorithms) need at least a 3,000-bit key to achieve the same level of security of a 128-bit symmetric algorithm. Asymmetric algorithms are incredibly slow and it is impractical to use them to encrypt large amounts of data. Generally, symmetric encryption algorithms are much faster to execute on a computer than asymmetric ones. The keys used in public-key encryption algorithms are usually much longer than those used in symmetric encryption algorithms. This is caused by the extra structure that is available to the cryptanalyst. There the problem is not that of guessing the right key, but deriving the matching private key from the public key .In practice they are often used together, so that a public-key algorithm is used to encrypt a randomly generated encryption key, and the random key is used to encrypt the actual message using a symmetric algorithm. This is sometimes called hybrid encryption.

2.4 Basic of RC6 Algorythm

2.4.1 RC6 Block Diagram

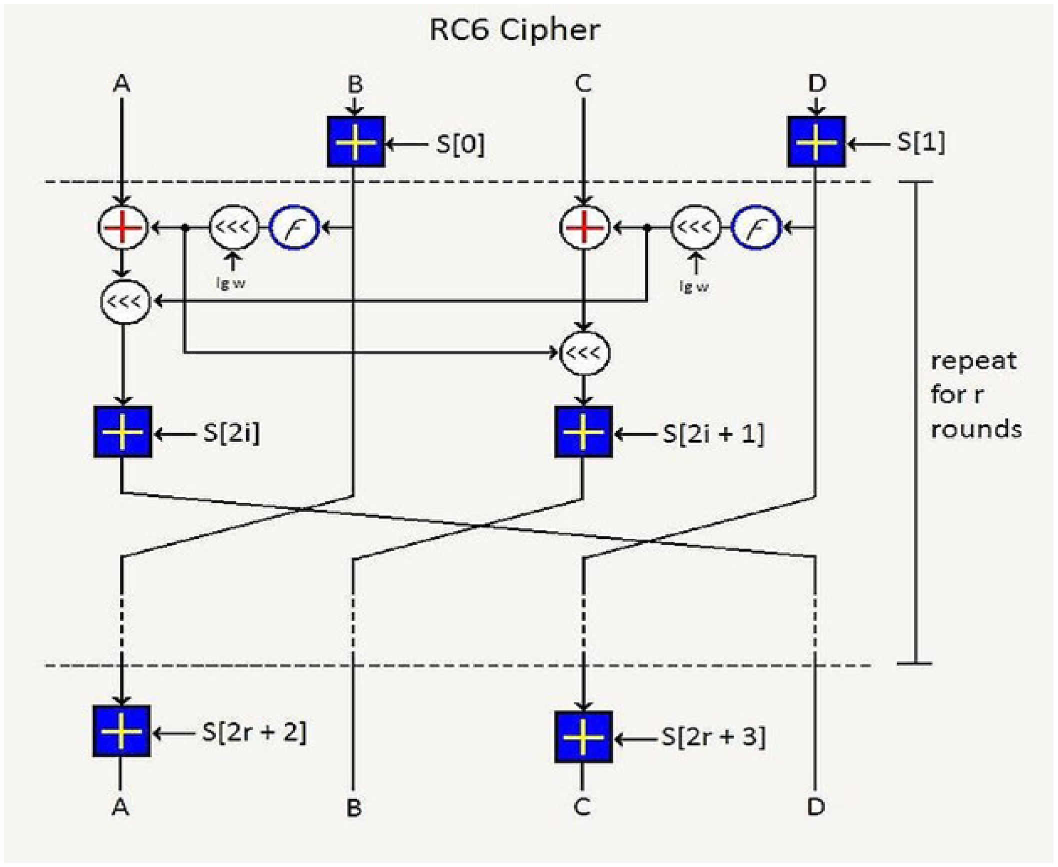


Figure 2.1: RC6 Cipher[12]

2.4.2 Basic Operation

RC6-w/r/b operates on units of four w-bit words using the following six basic operations. The base-two logarithm of w will be denoted by lg w.

- a + b integer addition modulo 2w
- a - b integer subtraction modulo 2w

- a XOR b bitwise exclusive-or of w-bit words
- a X b integer multiplication modulo
- a$\lll$ b rotate the w-bit word a to the left by the Amount given by the least significant lg w bits of b
- a$\ggg$b rotate the w-bit word a to the right by the Amount given by the least significant lg w bits of

2.4.3 Basic steps for RC6 Algorythm

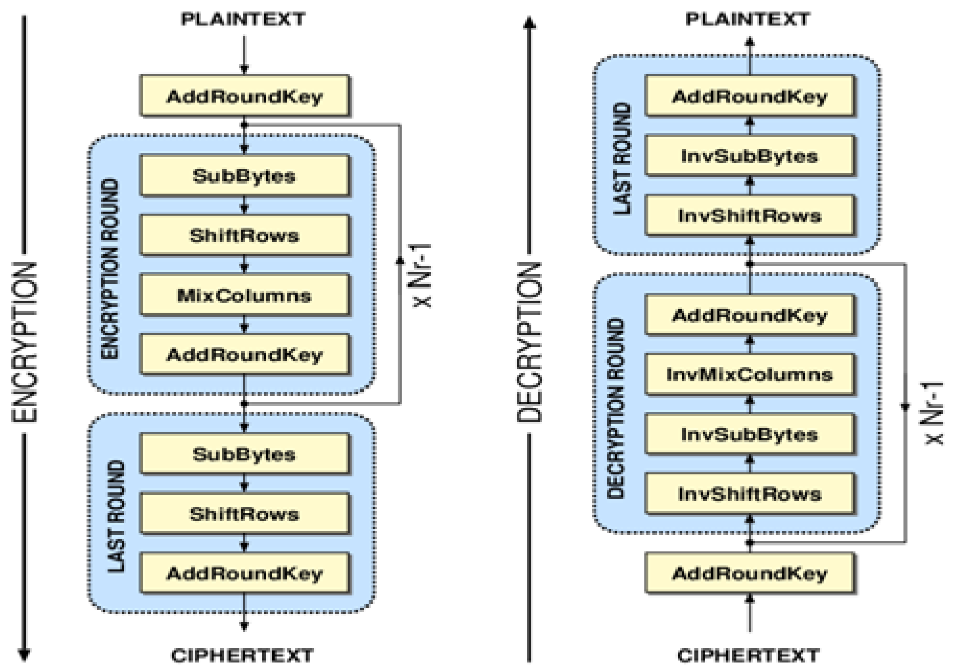


Figure 2.2: Basic Steps[13]

2.5 Encryption

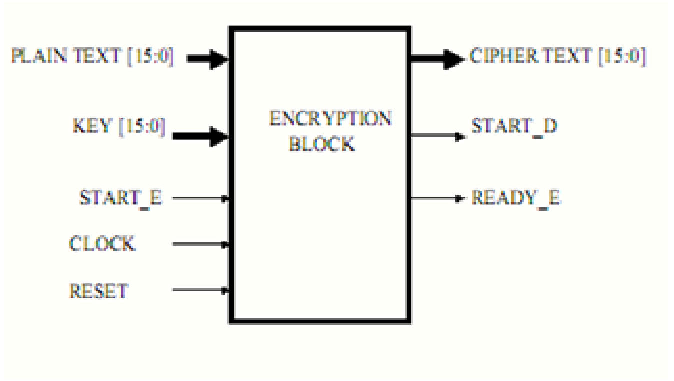


Figure 2.3: Encryption block diagram

2.5.1 Signal Specification

A block diagram of the 16-bit RC6 encryptor and decryptor is shown in Figure 1. Control signals are active high. The following gives a description of the input and output signals for the encryptor.

1. **Plaintext_e:** This 16-bit data input signal corresponds to a word of plaintext data that is to be encrypted. Four consecutive 16-bit Plaintext_evalues form a 64-bit block that is to be encrypted.

2. **Round_keys_e:** This 16-bit data input signal corresponds to one encryption round key. A total of 44 round keys are used to encrypt and decrypt the data. Although the encryptor and decryptor use identical round keys for a given block of data, separate encryptor and decryptor round keys are provided, since they receive the round keys at different times and processor the round keys in reverse order. This also allows the decryptor to be deciphering one block, while the encryptor is ciphering the next.

3. **Start_e:** This 1-bit control input signal tells the encryptor that it will start receiving Plaintext_e during the next cycle. This signal should only go high for one cycle.

4. **Reset:** This 1-bit control input signal resets both the encryptor and decryptor. When reset occurs, all registers are cleared and the controllers go to a known state.

5. **Clock :** This 1-bit input signal corresponds to the system clock for the encryptor and decryptor. Values are to be latched into registers at the positive edge of the clock.

6. **Ciphertext :** This 16-bit data output signal corresponds to a word of ciphertext data that has been encrypted and needs to be decrypted. Four consecutive 16-bit ciphertext values form a 64-bit block that has been encrypted. The same signal is used as an input to the decryptor.

7. **Ready_e:** This 1-bit control output signal indicates that the encryptor is ready to receive new plaintext. It goes high following a reset signal and stays high until the Start_e goes high. Once the encryptor has finished encrypting the dataReady_egoes high, until the next time Start_e goes high.

8. **Start_d :** This 1-bit control output signal tells the decryptor that the encryptor will start sending ciphertext during the next cycle. This signal should only go high for only one cycle. This same signal is used as an input to the decryptor

2.5.2 RC6 Encryption Algorythm

Input:

- Plain 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:

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

Procedure:

```
B = B + S [0]
D = D + S [1]
for i = 1 to r do
t = (B X (2B + 1)) $\ll$ lg w
u = (D X (2D + 1)) $\ll$ lg w
A = ((A XOR t) $\ll$ u) + S [2i]
C = ((C XOR u) $\ll$ t) + S [2i+ 1]
(A, B, C, D) = (B, C, D, A)
A = A + S [2r + 2]
C = C + S [2r + 3]
```

2.6 Decryption

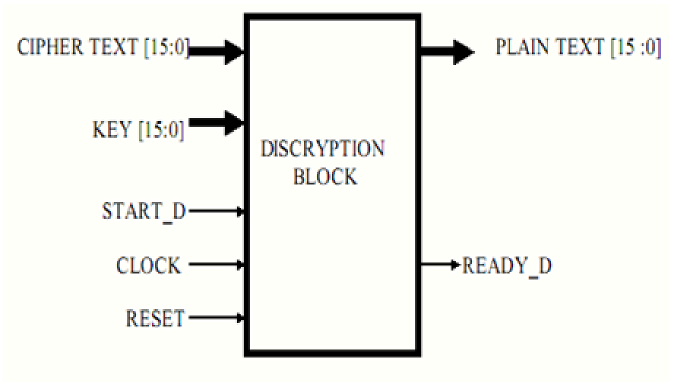


Figure 2.4: Decryption block diagram

The following gives a description of the input and output signals for the decryptor.

1. **Plaintext_d :** This 16-bit data output signal corresponds to a word of plaintext data that has been decrypted. Four consecutive 16-bit plaintext_d values form a 64-bit block that has been decrypted. When the same round_keys are used, the plaintext_d produced should be equivalent to the plaintext_e that was originally input.

2. **Round_keys_d :** This 16-bit data input signal corresponds to one decryption round key. (see the description of round_key_e for further details).

3. **Start_d :** This 1-bit control input signal tells the encryptor that it will start receiving Plaintext_e during the next cycle. This signal should only go high for one cycle.

4. **Reset :** This 1-bit control input signal resets both the encryptor and decryptor. When reset occurs, all registers are cleared and the controllers go to a known state.

5. **Clock :** This 1-bit input signal corresponds to the system clock for the encryptor and decryptor. Values are to be latched into registers at the positive edge of the clock.

6. **Ciphertext :** This 16-bit data output signal corresponds to a word of ciphertext data that has been encrypted and needs to be decrypted. Four consecutive 16-bit ciphertext values form a 64-bit block that has been encrypted. The same signal is used as an input to the decryptor.

7. **Ready_d :** This 1-bit control output signal indicates that the decryptor is ready to receive new ciphertext. It goes high following a reset signal and stays high until the start_d goes high. Once the decryptor has finished decrypting the data ready_d goes high, until the next time start_d goes high.

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 down to 1 do
{
(A, B, C, D) = (D, A, B, C)
u = (D X (2D + 1)) $\ll$ lg w
t = (B X (2B + 1)) $\ll$ lg w
C = ((C  S [2i + 1]) $\gg$ t) u
A = ((A  S [2i]) $\gg$ u)  t
}
D = D  S [1]
B = B  S [0]
```

Chapter 3

Theoritical Analysis

3.1 Modules Block Diagrams

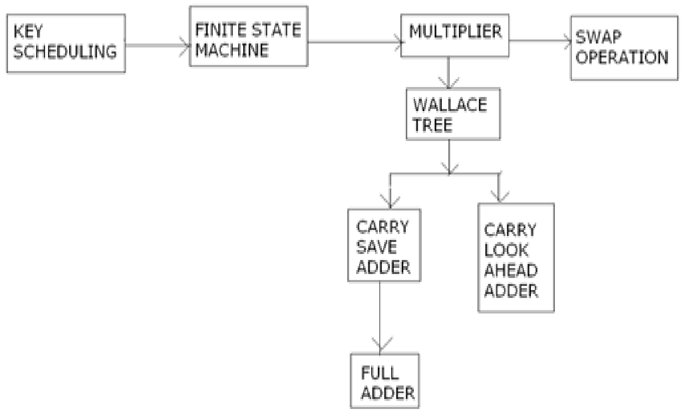


Figure 3.1: Digital circuit block diagram for Encryption

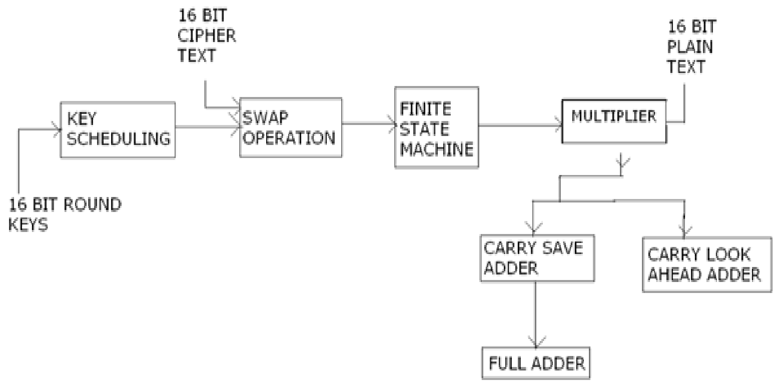


Figure 3.2: Digital circuit block diagram for Decryption

3.2 Key Scheduling[8]

The key schedule of RC6-w/r/b is practically identical to the key schedule of RC5-w/r/b. Indeed, the only difference is that for RC6-w/r/b, more words are derived from the user-supplied key for use during encryption and decryption.

The user supplies a key of b bytes, where $0 \leq b \leq 255$. From this key, 2r + 4 words (w bits each) are derived and stored in the array S[0,...,2r + 3]. This array is used in both encryption and decryption. The user supplies a key of b bytes. Sufficient zero bytes are appended to give a key length equal to a non-zero integral number of words; these key bytes are then loaded in little-endian fashion into an array of c w-bit (w = 32 bits in our case) words L [0],..., L [c - 1]. Thus the first byte of key is stored as the low-order byte of L [0], etc., and L [c - 1] is padded

With high-order zero bytes if necessary. The number of w bit (32 bit) words that will be generated for the additive round keys is 2r + 4 and these are stored in the array S [0;. . . ;2r+3]. the constants P32 = B7E15163 and Q32 = 9E3779B9 (hexadecimal) are the same magic constants" as used in the RC5. You may have wondered why there is array of 44. Because in encryption and decryption process we add round keys to A and C in

$A = ((A \text{ xor } t) \ll u) + S\ [2i]$

$C = ((C \text{ xor } u) \ll t) + S\ [2i + 1]$

So we need

20(for a as rounds are 20) +20(for C) +2(in starting for adding to B and D) +2(For adding in end in A and C) = 44

3.2.1 Key scheduling operation

Inputs:

User-supplied b byte key preload into the c-word array L[0,. . . ,c-1]

Number r of rounds

Output:

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

Procedure:

s(0)<= P ; – initialize constant array

l(0)<= (round_keyse +s(0));

s(1)<= (l(0)+ s(0)+ q);

l(1)<= (l(0)+s(1));

In general:

S(i) = l(i-1) + s(i-1) +q ;

L(i) = l(i) +s(i);

3.3 Finite State Machines

3.3.1 Introduction

Outputs are Function of State (and Inputs) Next States are Functions of State and Inputs Used to implement circuits that control other circuits "Decision Making" logic.

3.3.2 Concept of Finite State Machine

State diagram is a representation of different state and each state have different operation. The state diagram of cryptographic algorithms are shown below. The function in each state is also described in diagram. .

.

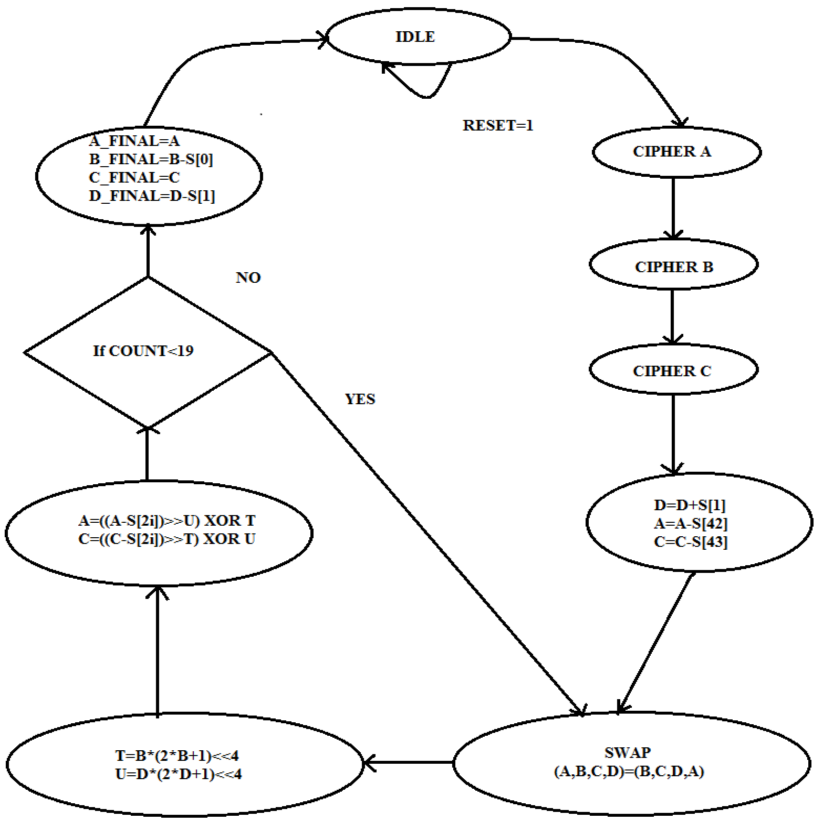


Figure 3.3: Finite state machine for input Encryption

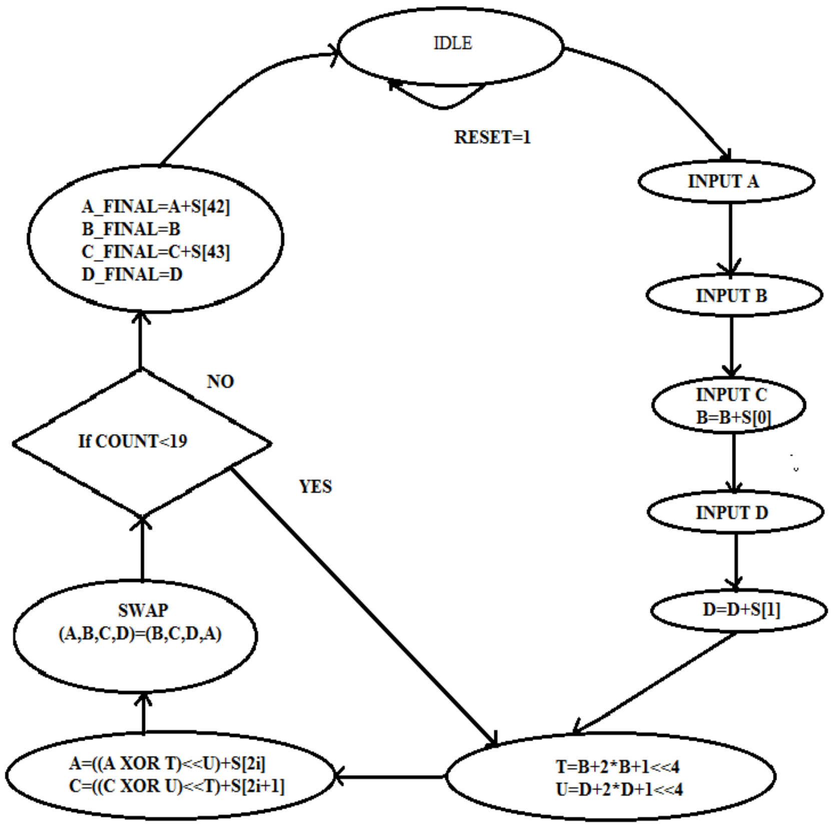


Figure 3.4: Finite State Machines for input Decryption

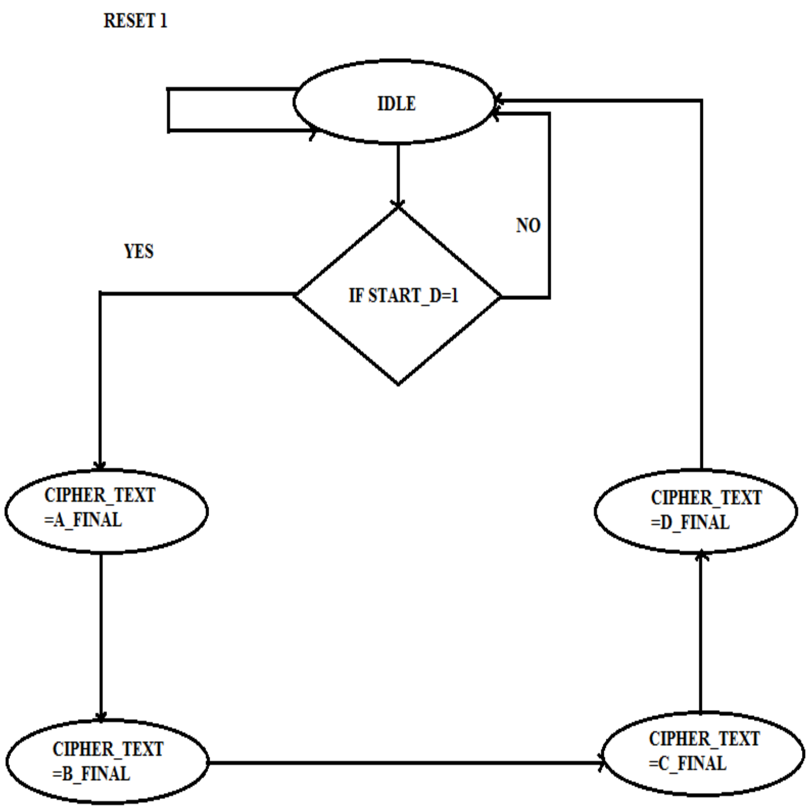


Figure 3.5: Finite State Machines for output Encryption

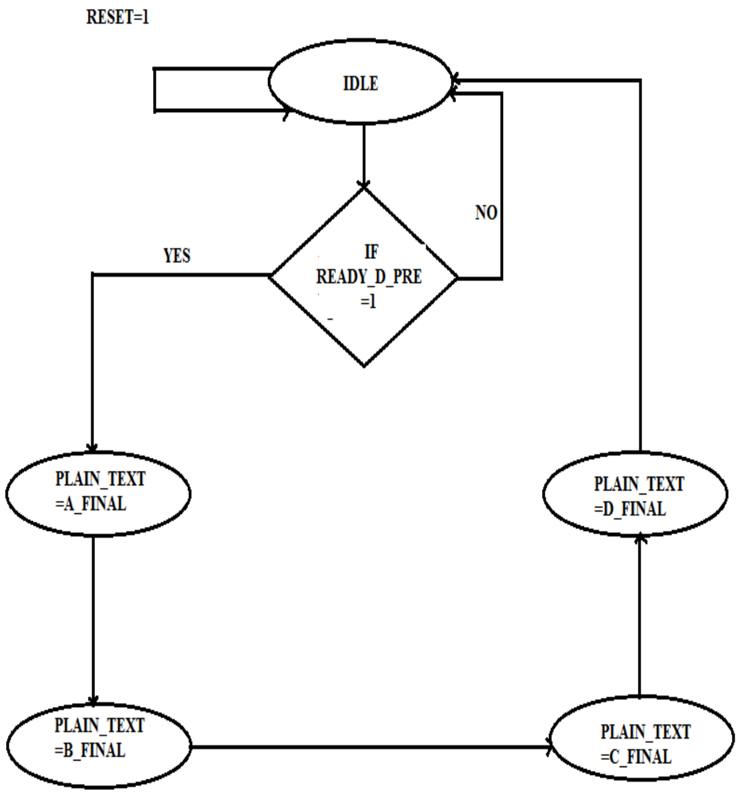


Figure 3.6: Finite State Machines for output Decryption

3.4 Multiplier

The multiplier is used to compute the operation B*(2B+1). It is implemented as $\mathbf{2}B^{\mathbf{2}}$+B. so we need to do a squaring and a addition. the partial product array for a parallel square using a multiplier The boxes indicate which terms may be combined using the equivalence aij+aji=2aij. In the lower portion of figure 2aij is represented by placing aij one column to the left which has a weighting of two times that of the current column. The square of operand a can be computed with the reduced partial product array. So for 2b we place whole product one place to left which has weighting 2 times that of previous. Our product is reduced to 9 from 16 due to this technique. In product1 (p1) we have value of B for +B operation in $\mathbf{2}B^{\mathbf{2}}$+B operation and normal 16 products formed are reduced to 8 products due to partial product reduction array. Now you may have wondered that why there are only 16 products while we are multiplying 16*16 so 32 products of 32 bits should be there. But there are only 16 bits and 16 products because we are using only lower 16 bits. This is so because we are having 20 rounds so for 20 rounds we are multiplying b*b so if we use all 32 bits and multiply at end of 20 rounds our product will be too large so we are using only lower 16 bits .

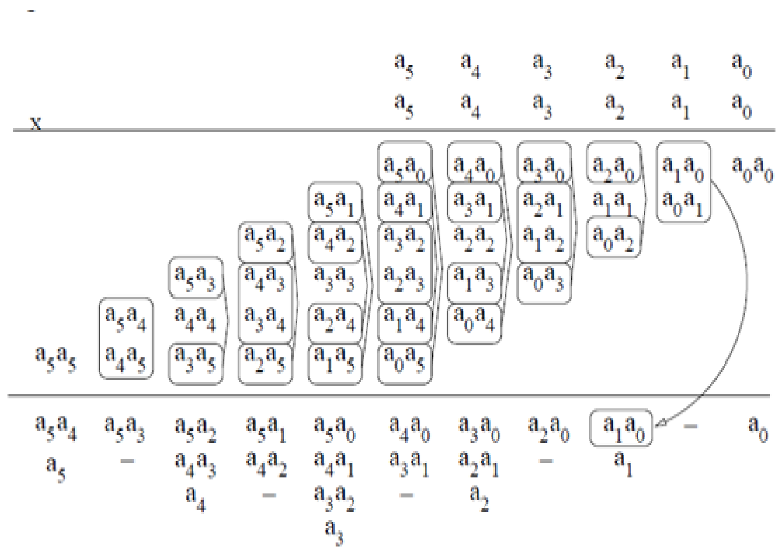


Figure 3.7: Wallace tree multiplier[4]

These 9 products formed are then input of Wallace tree multiplier. Wall ace tree will take 9 inputs and will produce final 16 bits one product. This is required by us to use as output of $\mathbf{2}B^{\mathbf{2}}$+B.

3.4.1 Wallace Tree

Figure below show the diagram of Wallace tree. Wallace tree have 2 components.

1. Carry save adder-CSA

2. Carry look ahead-CLA

A **Wallace tree** is an efficient hardware implementation of a digital circuit that multiplies two integers. The Wallace tree has three steps:

1. Multiply (that is - AND) each bit of one of the arguments, by each bit of the other, yielding n2 results. Depending on position of the multiplied bits, the wires carry different weights, for example wire of bit carrying result of a2b3 is 32 (see explanation of weights below).
2. Reduce the number of partial products to two by layers of full and half adders.
3. Group the wires in two numbers, and add them with a conventional adder.

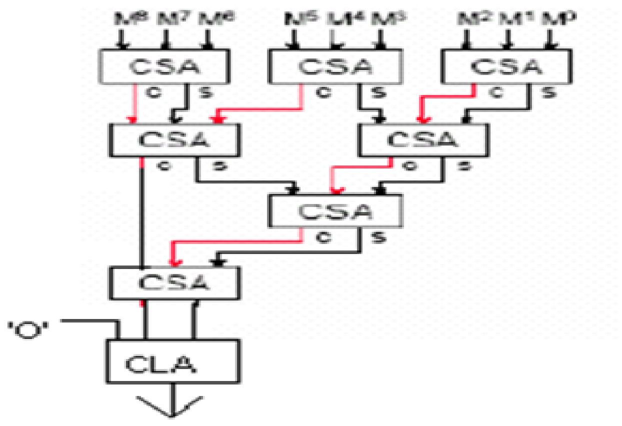


Figure 3.8: Wallace structure[4]

3.4.2 Carry Save Adder

The carry-save unit consists of n full adders, each of which computes a single sum and carry bit based solely on the corresponding bits of the three input numbers.

Given the three n - bit numbers a, b, and c, it produces a partial sum ps and a shift-carry sc:

$$ps_i = a_i \bigoplus b_i \bigoplus c_i$$

$$sc_i = (a_i \bigwedge b_i) \bigvee (a_i \bigwedge c_i) \bigvee (b_i \bigwedge c_i)$$

The entire sum can then be computed by:

1. Shifting the carry sequence sc left by one place.
2. Appending a 0 to the front (most significant bit) of the partial sum sequence ps.
3. Using a ripple carry adder to add these two together and produce the resulting n + 1-bit value. A ripple carry adder cannot compute a sum bit without waiting for the previous carry bit to be produced, and thus has a delay equal to that of n full adders. A carry-save adder produces all of its output values in parallel. we have seven full adder in our carry save adder.

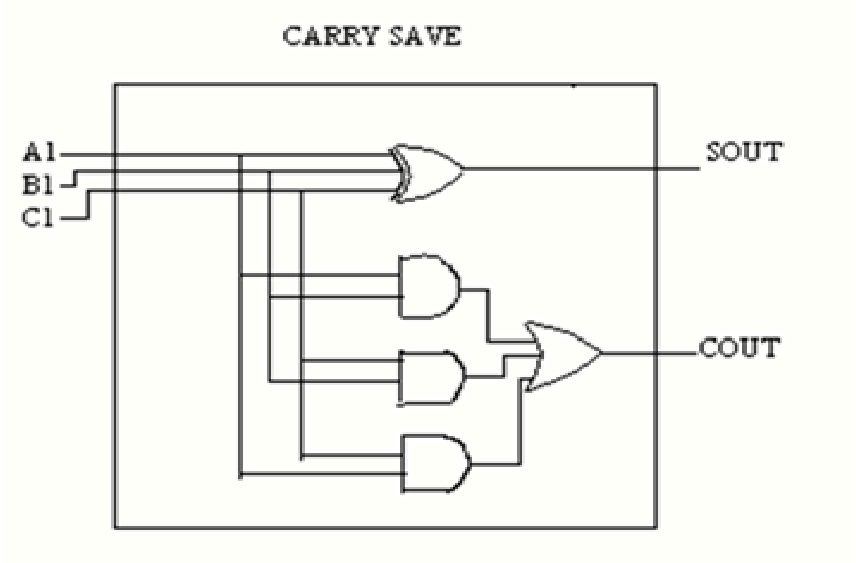


Figure 3.9: Carry Save Adder[4]

3.4.3 Carry Look Ahead Method

Carry look ahead logic uses the concepts of generating and propagating carries. Although in the context of a carry look ahead adder, it is most natural to think of generating and propagating in the context of binary addition, the concepts can be used more generally than this. In the descriptions below, the word digit can be replaced by bit when referring to binary addition.

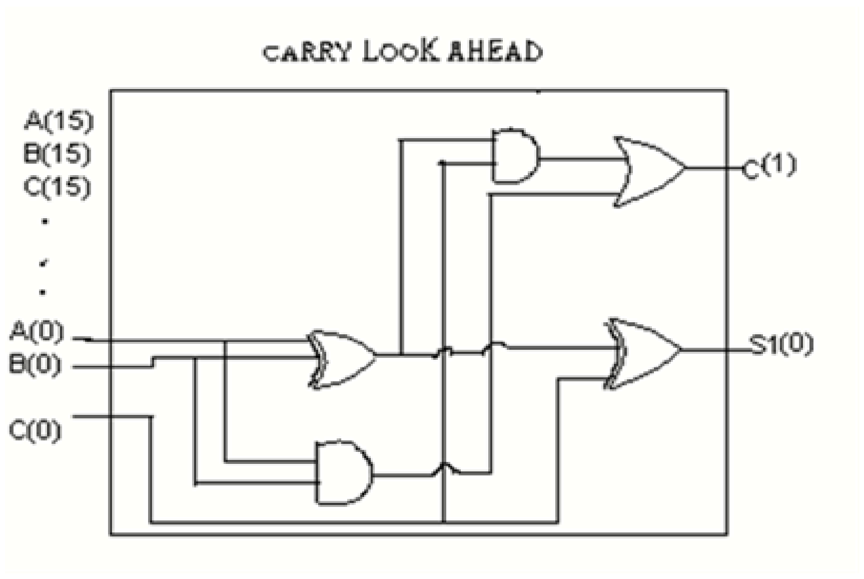


Figure 3.10: Carry Look Ahead Method[4]

The addition of two 1-digit inputs A and B is said to generate if the addition will always carry, regardless of whether there is an input carry (equivalently, regardless of whether any less significant digits in the sum carry). For example, in the decimal addition 52 + 67, the addition of the tens digits 5 and 6 generates because the result carries to the hundreds digit regardless of whether the ones digit carries (in the example, the ones digit does not carry (2+7=9)). In the case of binary addition, A + B generates if and only if both A and B are 1. If we write G(A,B) to represent the binary predicate that is true if and only if A + B generates, we have:

G(A,B) = A $\cdot$ B

The addition of two 1-digit inputs A and B is said to propagate if the addition will carry whenever there is an input carry (equivalently, when the next less significant digit in the sum carries).. In the case of binary addition, A + B propagates if and only if at least one of A or B is 1. If we write P(A,B) to represent the binary predicate that is true if and only if A + B propagates, In the case of binary addition, this definition is expressed by:

P'(A,B) = A $\bigoplus$ B

Carry is given by,

$C_{i+1} = G_i + P_i \cdot C_i$

Implementation details

Here we add sum and carry which are the output of carry save adder series and 3rd digit is taken as 0 as we have only two 16 bit arrays to add. We add individual bits for example

P0(A,B)=A0 XOR B0

G0=A0 AND B0

C1=G0+(P0·C0)

C0=0

3.5 Swap Operation

As shown in block diagram, first we perform XOR operation between two inputs then store the result into temporary variable. After that temporary variable is further XORed with any of two inputs.so we get swapped output.In RC6 algorithm by performing swap operation we get,

(A,B,C,D)=(B,C,D,A)

Formula for swapping can be written as,

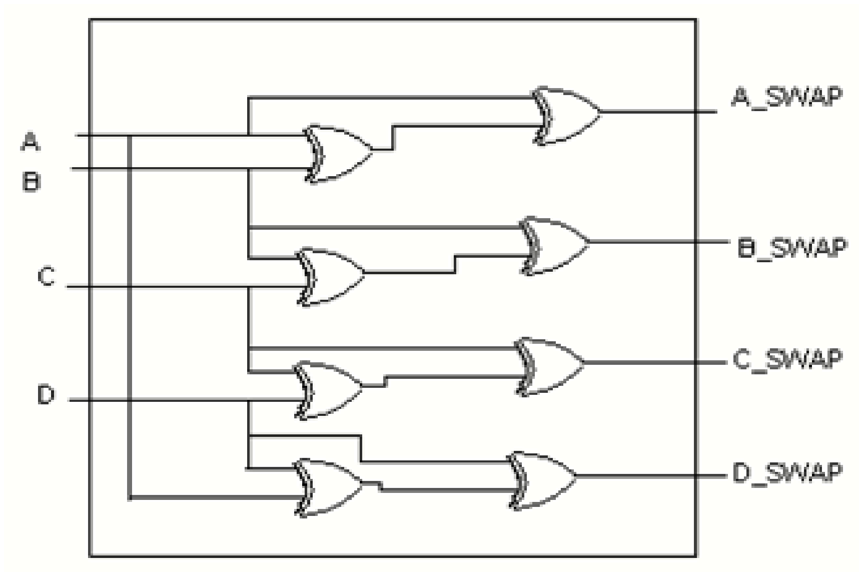


Figure 3.11: Block diagram for swaping[4]

temp_AB:= tempA xor tempB;

A$\leq$ temp_AB xor tempA;

Chapter 4

Reconfigurable Hardware Technology

4.1 Introduction

- An FPGA is an integrated circuit that belongs to a family of programmable devices called Programmable Logic Devices (PLDs).
- An FPGA contains tenths of thousands of building blocks, known as Configuration Logic Blocks (CLB) connected through programmable interconnections.
- In recent years, FPGAs have been used for reconfigurable computing when the main goal is to obtain high performance at a reasonable cost out of hardware implemented algorithms.

4.1.1 Why Reconfigurable H/W Technology??[10]

The main advantage of FPGAs is their reconfigurability, i.e., they can be used for different purposes at different stages of a computation and they can be, at least partially, reprogrammed on run-time.

4.1.2 Applications of FPGAs

1. Network processors
2. Real-time systems
3. Rapid ASIC prototyping
4. Digital signal processing
5. Robotics
6. Cryptography
7. Computer graphics etc...

4.2 FPGA Design Flow

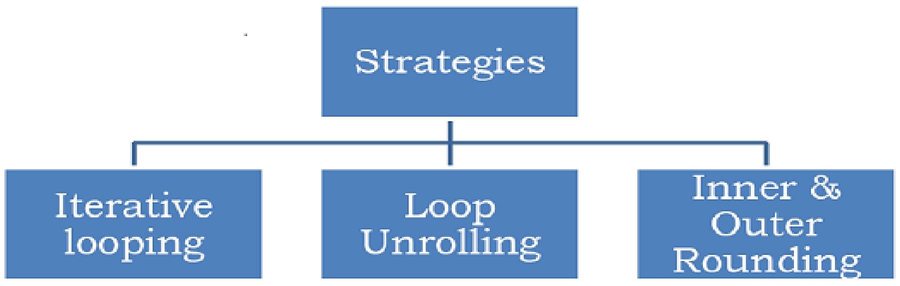


Figure 4.1: Fpga design Flow[10]

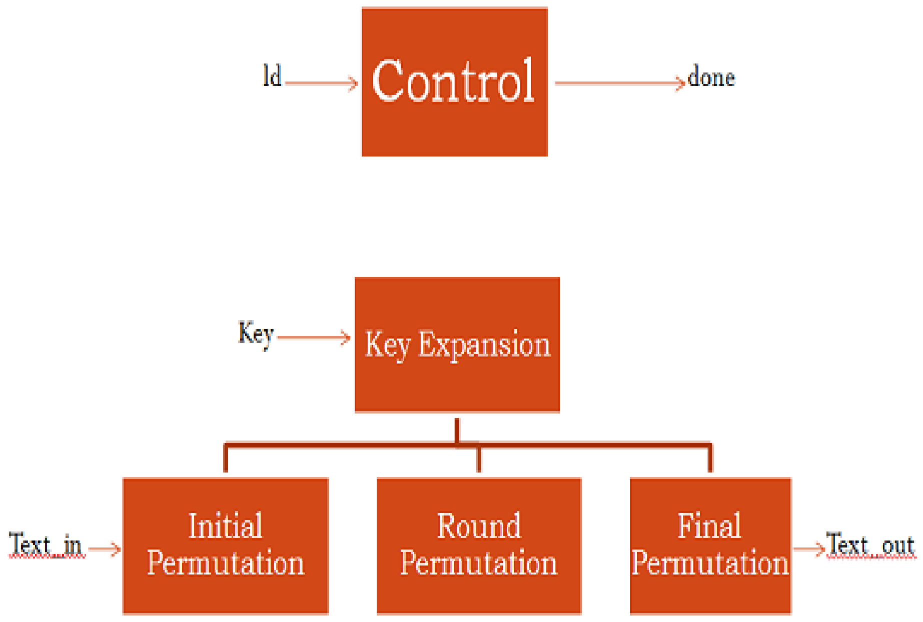


Figure 4.2: AES Cipher Core[10]

4.3 AES Cipher Core

The AES cipher core consists of a key expansion module, an initial permutation 0 module, a round permutation module and a final permutation module. The round permutation module will loop internally to perform 10 iteration (for 128 bit keys).

4.4 Timing Diagram

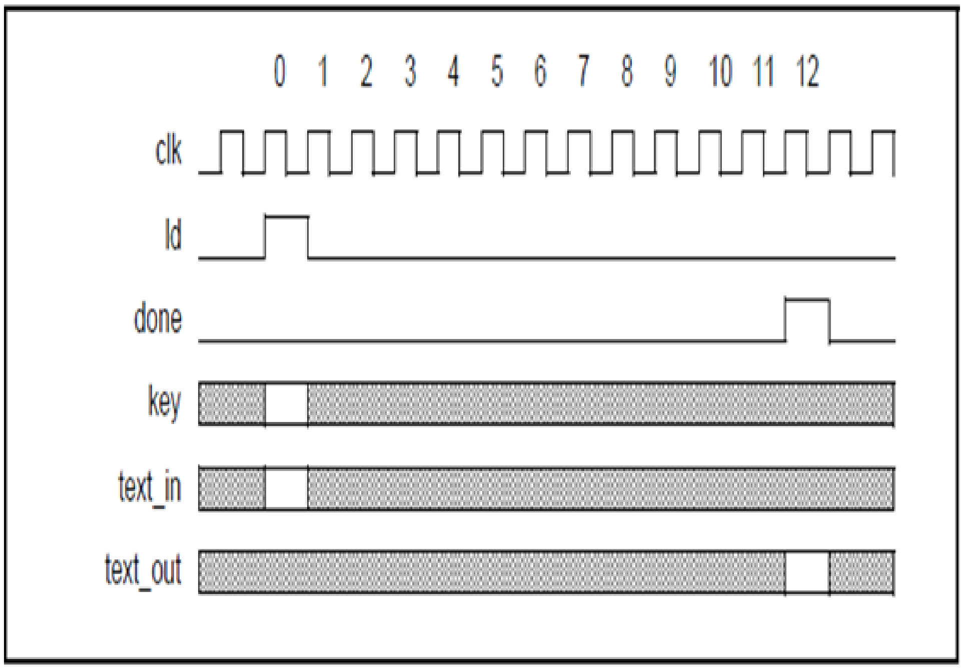


Figure 4.3: Timing diagram[10]

Chapter 5

Implementation & Simmulation Result

5.1 How to Simulate & Implement RC6 Algorythm?

We have implemented cryptography in Xilinx 13.1i edition software.We have seen how to make a program in Xilinx now we will see the results. And we will also come to know that how to give inputs and how to check test bench and see output. RTL schematic shows a functional digital circuit having inputs and outputs. Test bench waveform is used to check whethre our digital circuit works perfectly or not.for that specific steps must be followed as described in this section.

5.2 Testing Methodology

We can test our project by two ways.

1. Design project using RTL SCHEMETIC
2. Simulate project using TEST BENCH WAVEFORM

5.2.1 About RTL Schematic

Our design can be viewed using RTLschematic that,

1. A diagram showing the logical hierarchy of your modules including module names and brief descriptions of what each module does.

2. Detailed block diagrams for the encryptor and decryptor, which show registers, functional units, multiplexers, control signals, etc.

3. State transition graphs or ASM charts for the finite state machines that control your encryptor and decryptor.

4. VHDL code for your encryptor, decryptor, key scheduler, and testbench. This code should be commented well.

5.2.2 RTL Schematic for Encryptior

After checking syntex successfully, RTL schematic can be viewed from PROCESSES/ VIEW RTL SCHEMATIC .It shows digital circuit input and output diagram for RC6 ENCRYPTION algorithm.

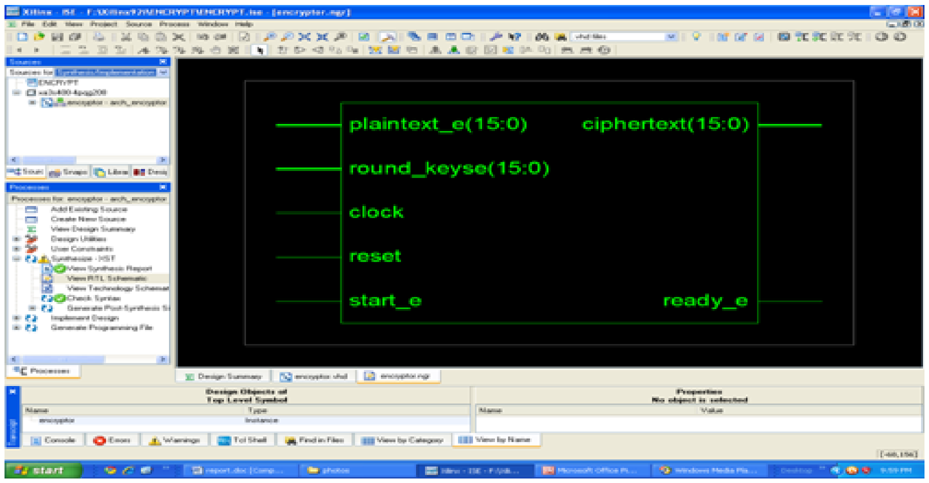


Figure 5.1: RTL Schematic for Encryptior

5.2.3 RTL Schematic for Decryptior

After checking syntex successfully, RTL schematic can be viewed from PROCESSES/ VIEW RTL SCHEMATIC .It shows digital circuit input and output diagram for RC6 DECRYPTION algorithm

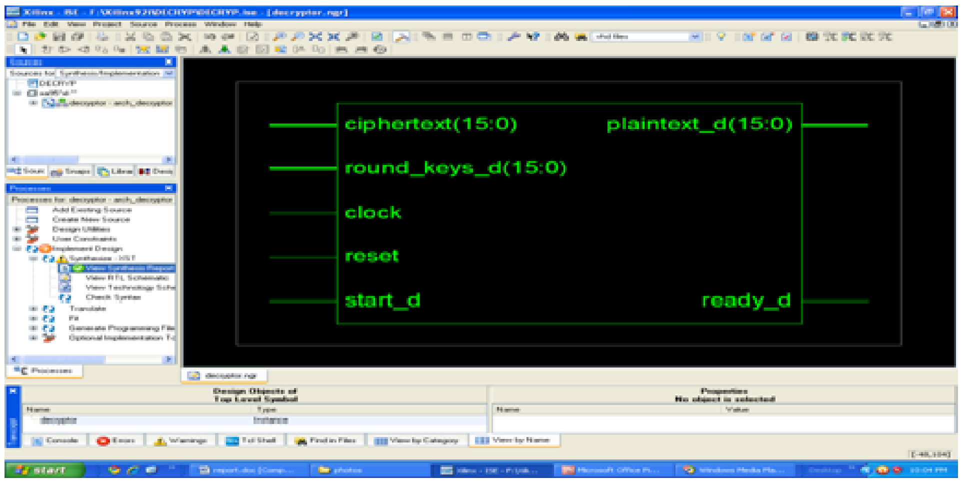


Figure 5.2: RTL Schematic for Decryptior

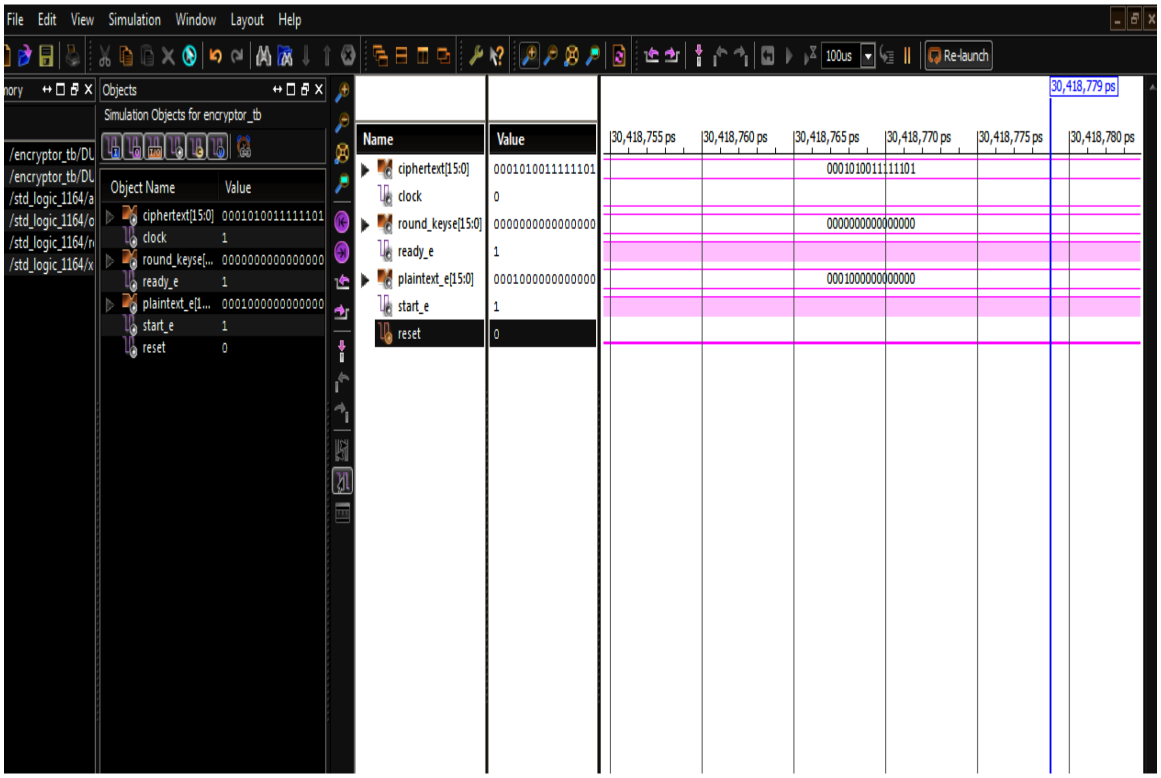
5.3 About Test Bench Waveform

Our design should be tested by developing a testbench that ,

1. Reads in plaintext, user-supplied keys, and expected ciphertext from a file.
2. Uses the key scheduler to take the user-supplied keys and generate the round keys.
3. Send the appropriate the control signals, plaintext, and round keys to the encryptor and decryptor.
4. Tests to ensure that the ciphertext is correctly generated and that the plaintext from the decryptor matches the plaintext to the encryptor. If discrepancies

occur between the expected results and actual results, errors should be reported.

In fig below we can see the output test bench of encryptor the cipher text is available after 1800ns. The cipher text is available after ready_e signal goes high.this cipher text is given to decryptor

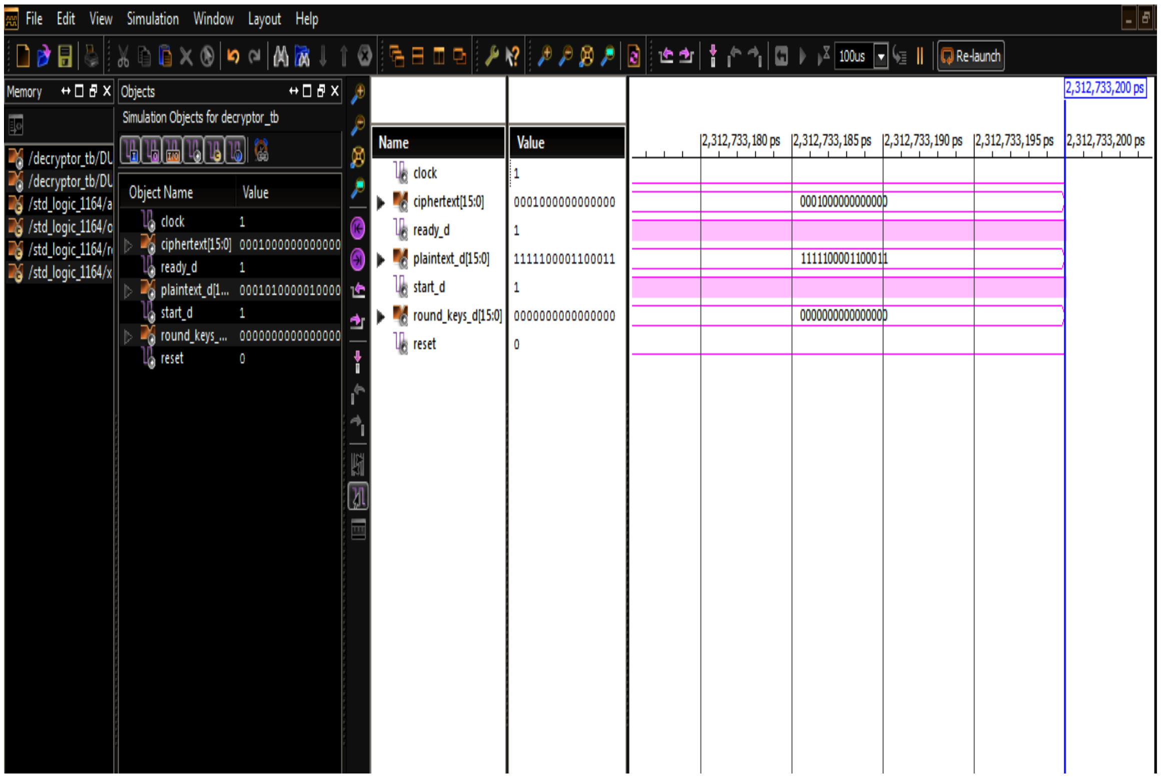


```
Name              Value
ciphertext[15:0]  0001010011111101
clock             0
round_keyse[15:0] 0000000000000000
ready_e           1
plaintext_e[15:0] 0001000000000000
start_e           1
reset             0
```

Figure 5.3: Simulation of Encryption

. When simulation is done we can see output of decryptor as shown below. The plain text which was given in encryptor is obtained in decryptor output.

.



Simulation Objects for decryptor_tb

Object Name	Value
clock	1
ciphertext[15:0]	0001000000000000
ready_d	1
plaintext_d[1...	0001010000010000
start_d	1
round_keys_...	0000000000000000
reset	0

Name	Value
clock	1
ciphertext[15:0]	0001000000000000
ready_d	1
plaintext_d[15:0]	1111100001100011
start_d	1
round_keys_d[15:0]	0000000000000000
reset	0

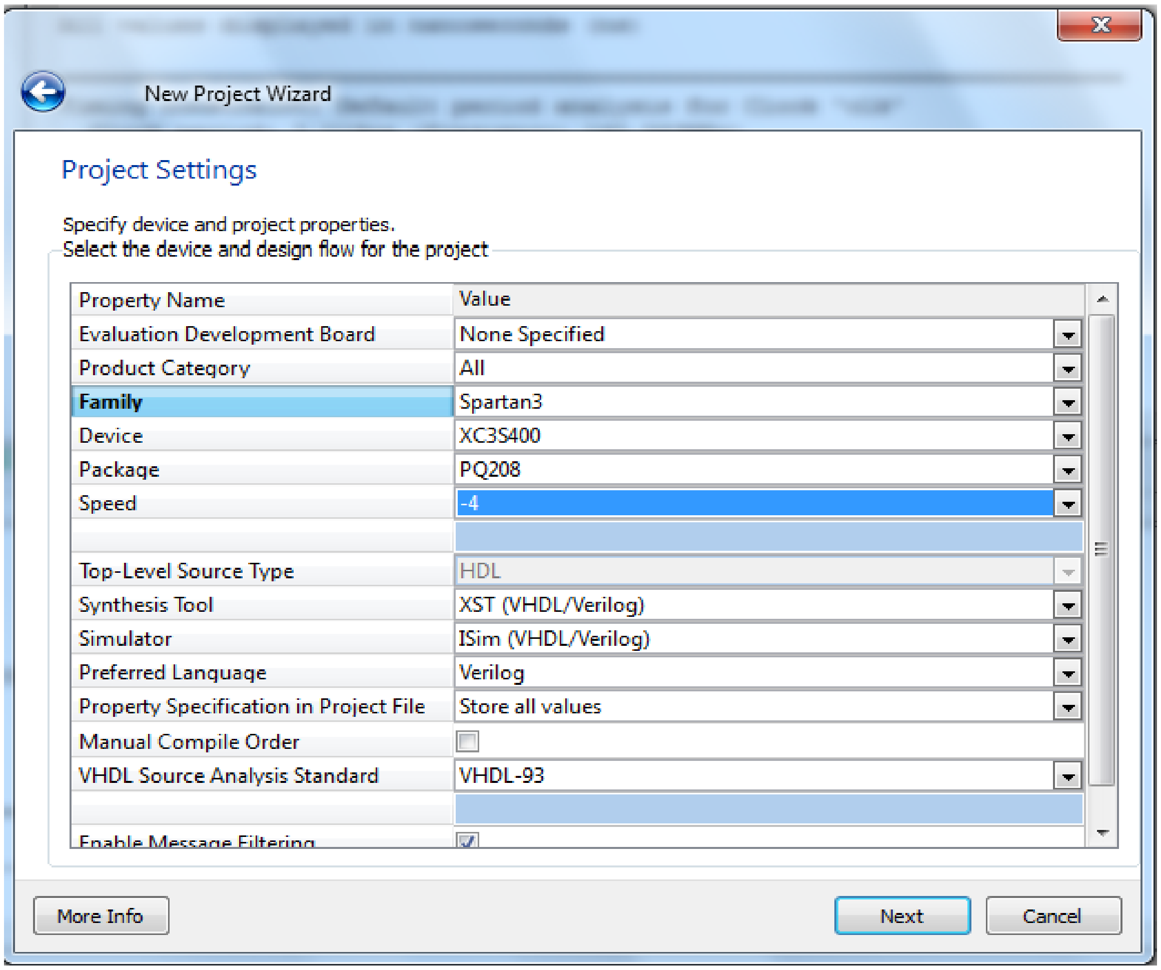
```
2,312,733,180 ps   2,312,733,185 ps   2,312,733,190 ps   2,312,733,195 ps   2,312,733,200 ps
                   0001000000000000
                   1111100001100011
                   0000000000000000
```

2,312,733,200 ps

Figure 5.4: Simulation of Decryption

5.4 Steps for Hardware Implementation of RC6 Algorythm[15]

STEP 1: Select the device and design flow for the project. .



Project Settings

Specify device and project properties.
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3
Device	XC3S400
Package	PQ208
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	✓

Figure 5.5: Select the device and design flow

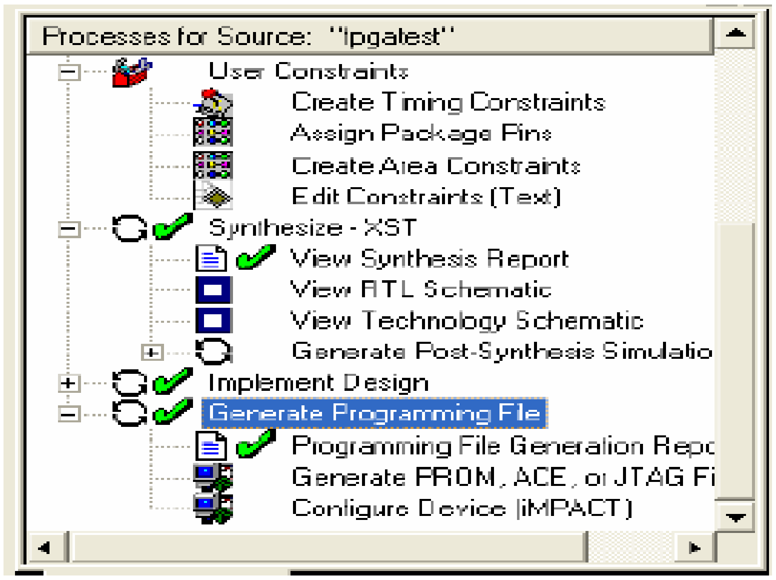
Device Family: Choose Spartan 3, the device we will be using.

Device: xs3s400, the specific Spartan 3 device we use.

Package: PQ208, this is the package type of our device (Ball Grid Array, 256 pins).

Speed Grade: The speed grade for this device is -4

STEP II: Create and save the user constraint file and run the Generate Programming File option.

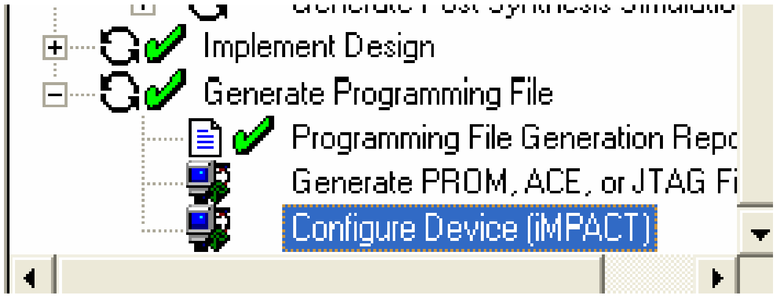


Processes for Source: "fpgatest"
User Constraints
Create Timing Constraints
Assign Package Pins
Create Area Constraints
Edit Constraints (Text)
Synthesize - XST
View Synthesis Report
View RTL Schematic
View Technology Schematic
Generate Post-Synthesis Simulatio
Implement Design
Generate Programming File
Programming File Generation Repo
Generate PROM, ACE, or JTAG Fi
Configure Device (iMPACT)

Figure 5.6: Step for Generate programming file

. **STEP III:** Hook up the FGPA to the PC using the supplied JTAG cable.

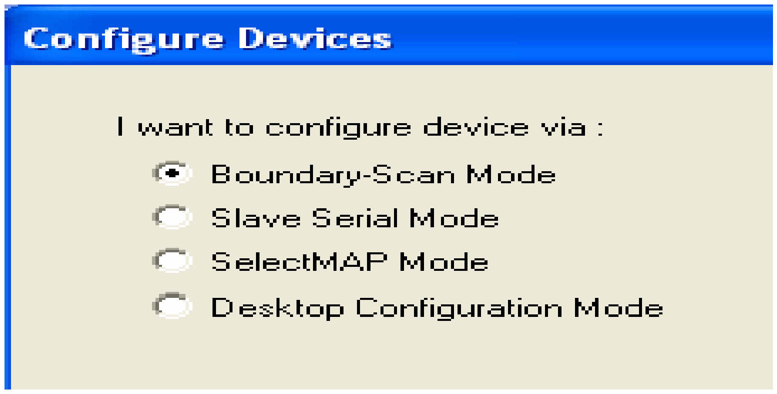
1. Start the iMPACT program by selecting it in the main Xilinx window under Generate Programming File. .



Implement Design
Generate Programming File
Programming File Generation Repo
Generate PROM, ACE, or JTAG Fi
Configure Device (iMPACT)

Figure 5.7: Step for Configure device

2. When the Configure Devices dialog appears, select Boundary-Scan Mode and press Next. .



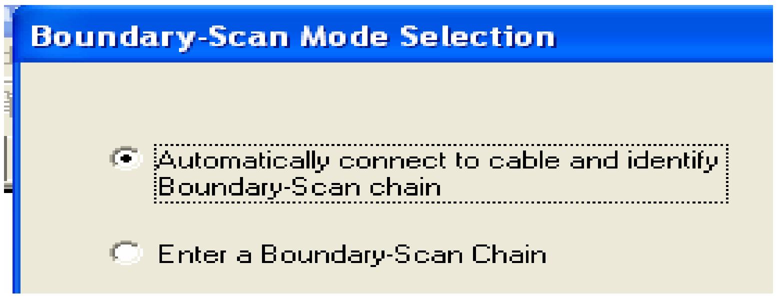
Configure Devices

I want to configure device via :

- [✓ Boundary-Scan Mode]
- Slave Serial Mode
- SelectMAP Mode
- Desktop Configuration Mode

Figure 5.8: Step for select Boundry-Scan

3. In the next dialog select Automatically connect to cable and press Next.



Boundary-Scan Mode Selection

- [✓ Automatically connect to cable and identify Boundary-Scan chain]
- Enter a Boundary-Scan Chain

Figure 5.9: Boundary-Scan Mode Selection

.

4. iMPACT should autodetect the FPGA and alert you that 2 devices were found, press OK.

5. When the select file menu appears, select the .bit file you generated earlier. You should have only one of these files in your project directory and it should have the same name as your schematic. Select that file and press Open.

6. If iMPACT warns you about the JTagClk being changed, simply press OK and move on.

7. Another Open File dialog will appear. This is used for programming the Flash-rom on the board and will not be used here. Select the Bypass button on this dialog.

8. With the programming files assigned, right click on the leftmost device and select Program..

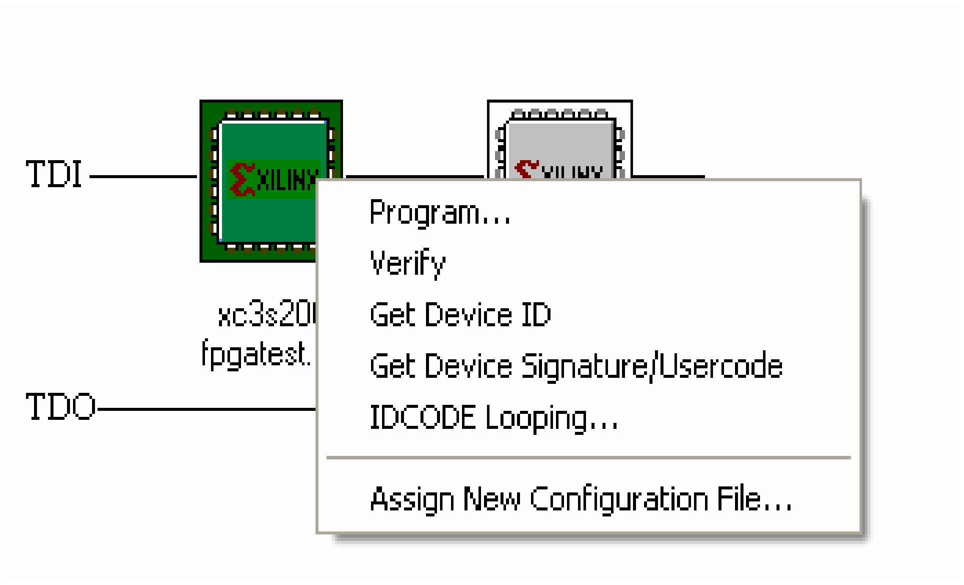


Figure 5.10: Downloading program in Fpga

.

9. Within the Program menu, unselect the Verify check if it is checked. If you try to program this FPGA with verify turned on, the programming will fail.

10. Select OK. The FPGA will take a moment to program, and should return with a Programming Successful message.

.

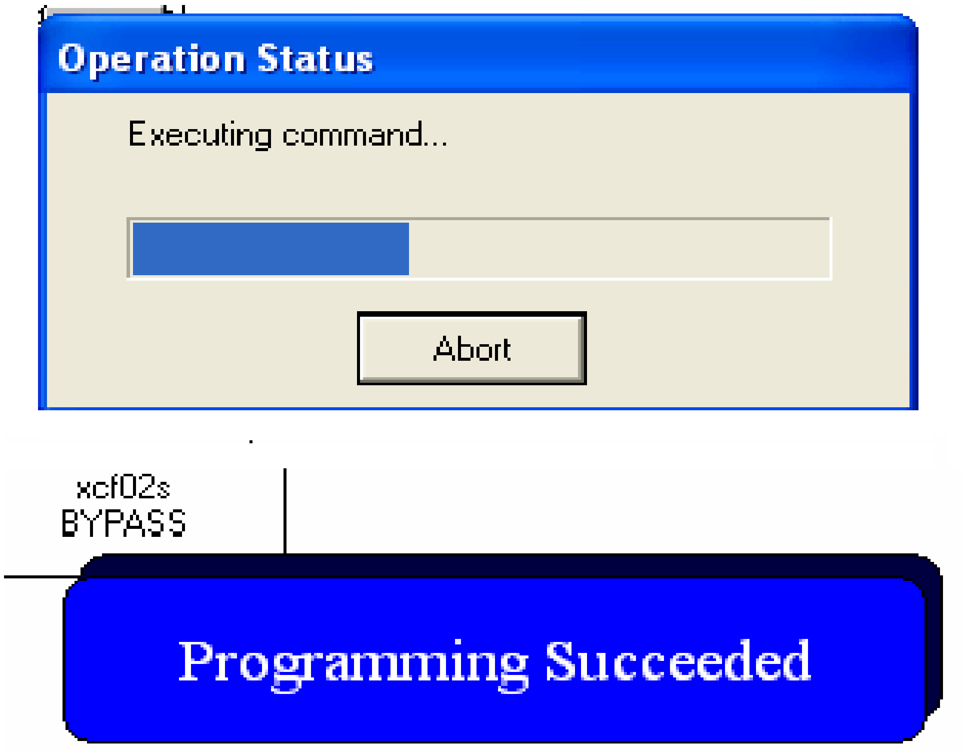


Figure 5.11: Operation Status

5.4.1 Synthesis Report for Encryption:

Device utilization summary for SPARTAN 3(3s400pq208-4):

	Used	**Available**	**Utilization**
Number of Slices:	**1273**	**3584**	**35**
Number of Slice Flip Flops:	**211**	**7168**	**2**
Number of 4 input LUTs:	**2460**	**7168**	**34**
Number of IOs:	**52**		
Number of bonded IOBs:	**52**	**141**	**36**
IOB Flip Flops::	**16**		
Number of GCLKs:	**1**	**8**	**12**

Table 5.1: Device utilization summary Encryption Spartan 3

Timing Summary(Speed Grade: -4):

Minimum period: (Maximum Frequency:74.571MHz)	**13.410ns**
Minimum input arrival time before clock:	**206.128ns**
Maximum output required time after clock:	**7.271ns**

Table 5.2: Timing summary Encryption Spartan 3

5.4.2 Synthesis Report for Decryption:

Device utilization summary for SPARTAN 3(3s400pq208-4):

	Used	**Available**	**Utilization**
Number of Slices:	**1285**	**3584**	**35**
Number of Slice Flip Flops:	**221**	**7168**	**3**
Number of 4 input LUTs:	**2490**	**7168**	**34**
Number of IOs:	**52**		
Number of bonded IOBs:	**52**	**141**	**36**
Number of GCLKs:	**1**	**8**	**24**

Table 5.3: Device utilization summary Decryption Spartan 3

Timing Summary(Speed Grade: -4):

Minimum period: (Maximum Frequency:78.339MHz)	**12.765ns**
Minimum input arrival time before clock:	**207.548ns**
Maximum output required time after clock:	**7.241ns**

Table 5.4: Timing summary Decryption Spartan 3

5.4.3 Synthesis Report for Encryption:

Device utilization summary for SPARTAN 3E(3s500efg320-4):

	Used	**Available**	**Utilization**
Number of Slices:	**1269**	**4656**	**35**
Number of Slice Flip Flops:	**203**	**9312**	**2**
Number of 4 input LUTs:	**2454**	**9312**	**26**
Number of IOs:	**52**		
Number of bonded IOBs:	**52**	**232**	**22**
IOB Flip Flops::	**16**		
Number of GCLKs:	**1**	**24**	**4**

Table 5.5: Device utilization summary Encryption Spartan 3E

Timing Summary(Speed Grade: -4):

Minimum period: (Maximum Frequency:90.38MHz)	**11.064ns**
Minimum input arrival time before clock:	**170.864ns**
Maximum output required time after clock:	**4.39ns**

Table 5.6: Timing summary Encryption Spartan 3E

5.4.4 Synthesis Report for Decryption:

Device utilization summary for SPARTAN 3E(3s500efg320-4):

	Used	Available	Utilization
Number of Slices:	**1280**	**4656**	**27**
Number of Slice Flip Flops:	**213**	**9312**	**2**
Number of 4 input LUTs:	**2482**	**9312**	**26**
Number of IOs:	**52**		
Number of bonded IOBs:	**52**	**232**	**22**
Number of GCLKs:	**1**	**24**	**4**

Table 5.7: Device utilization summary Decryption Spartan 3E

Timing Summary(Speed Grade: -4):

Minimum period: (Maximum Frequency:78.339MHz)	**12.765ns**
Minimum input arrival time before clock:	**207.548ns**
Maximum output required time after clock:	**7.241ns**

Table 5.8: Timing summary Decryption Spartan 3E

5.4.5 Synthesis Report for Encryption:

Device utilization summary for VERTEX 5(5vlx110tff1136-1):

	Used	**Available**	**Utilization**
Number of Slices:	**199**	**69120**	**0**
Number of Slice Flip Flops:	**2282**	**2481**	**91**
Number of 4 input LUTs:	**152**	**2481**	**6**
Number of IOs:	**52**		
Number of bonded IOBs:	**52**	**232**	**22**
IOB Flip Flops::	**16**		
Number of GCLKs:	**1**	**32**	**3**

Table 5.9: Device utilization summary Vertex 5 Encryption

Timing Summary(Speed Grade: -4):

Minimum period: (Maximum Frequency:194.704MHz)	**5.136ns**
Minimum input arrival time before clock:	**79.473ns**
Maximum output required time after clock:	**3.520ns**

Table 5.10: Timing summary Vertex 5 Encryption

5.4.6 Synthesis Report for Decryption:

Device utilization summary for VERTEX 5(5vlx110tff1136-1):

	Used	**Available**	**Utilization**
Number of Slices:	**201**	**69120**	**0**
Number of Slice Flip Flops:	**2278**	**2479**	**91**
Number of 4 input LUTs:	**120**	**2479**	**4**
Number of IOs:	**52**		
Number of bonded IOBs:	**52**	**232**	**22**
Number of GCLKs:	**1**	**32**	**3**

Table 5.11: Device utilization summary Vertex 5 Decryption

Timing Summary(Speed Grade: -4):

Minimum period: (Maximum Frequency:78.339MHz)	**12.765ns**
Minimum input arrival time before clock:	**207.548ns**
Maximum output required time after clock:	**7.241ns**

Table 5.12: Timing summary Vertex 5 for Decryption

Application Area

1. INTERNET E-COMMERCE
2. MOBILE TELEPHON ENETWORKS
3. BANK AUTOMATED TELLER MACHINE
4. SECRET COMMUNICATION

Conclusion

We have implemented the RC6 cipher core and RC6 inverse cipher core on FPGA spartan 3,spartan 3E and Vertex 5 reconfigurable hardware.

We have used two finite state machines for implementing pipeline in our design.One fsm controls the calculation of the various rounds. Another fsm is responsible for outputting the ciphertext so the encryptor can accept plain text while cipher text is still in the process of going out.

Multiplication and addition are the major bottleneck as far as speed of encryption in RC6 cipher is concerned. Nevertheless up to a great extent this short coming was tackled using pipelining in our design.

Bibliography

[1] E. Bach and J. Shallit. Algorithmic Number Theory, Volume I: Efficient Algorithms.Kluwer Academic Publishers, Boston, MA, 1996. 15. D. Bae, G. Kim, . Kim, S. Park, and O. Song. An Efficient.

[2] M. Bednara, M. Daldrup, J. Shokrollahi, J. Teich, and J. von zur Gathen. Reconfigurable Implementation of Elliptic Curve Crypto Algorithms. In 9 th Reconfigurable Architectures Workshop (RAW- 02), pages 157-164, Fort Lauderdale, Florida, U.S.A., April 2002.

[3] C. D.Walter, Q. K. Kog, and C. Paar, editors. Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings, volume 2779 of Lecture Notes in Computer Science. Springer, 2003.

[4] http://www.aoki.ecei.tohoku.ac.jp/arith/mg/algorithm.htmlitem

[5] N. A. Saqib, A. Diaz-Perez, and F. Rodriguez-Henriquez. Highly Optimized Single-Chip FPGA Implementations of AES Encryption and Decryption Cores In X Workshop Iberchip, pages 117-118, Cartagena-Colombia, March 2004.

[6] A. Rudra, P. K. Dubey, C. S. Julta, V. Kumar, J. R. Rao, and P. Rohatgi. Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. In Proceedings of the CHES 2001, volume 2162 of Lecture Notes in Computer Science, pages 171-184. Springer, 2001.

[7] V. A. Pedroni. Circuit Design with VHDL. The MIT Press, August 2004.

[8] National Institute of Standards and Technology. NIST Special Publication 800-57: Recommendation for Key Management Part 1: General, August 2005

[9] Douglas L. Perry, VHDL Programming by Example, Tata McGraw-Hill Edition 2002, 1-266.

[10] Adesara Ankit M. Prof. N.P. Gajjar Heeral P. Sheth."Cryptography algorithm on reconfigure hardware."Laljibhai Chaturbhai Institute of Technology,Bhandu-384120,13-14 Apirl,2012

[11] Atul Kahate, Cryptography and Network Security, Tata McGraw-Hill Edition 2003,2,29,40,43,63,75,98,107.

[12] Ioan Mang, Greda Erica Mang, Hardware Implementation with offline test capabilities of the RC6 block cipher 0-7803-7625-0/02/17.00 2002 IEEE. British Crown Copyright.

[13] Ronald L. Rivest,M.J.B. Robshaw *et al* "The RC6 Block Cipher",Version 1.1 - August 20, 1998

[14] L. Henzen VLSI Circuits for Cryptographic Authentication, PhD Thesis ETH-No. 19351, Hartung-Gorre Printing House, Konstanz, Germany, 2010

[15] Spartan 3 Tutorial

[16] S. Contini, R.L. Rivest, M.J.B. Robshaw and Y.L. Yin, The Security of the RC6 Block Cipher, Version 1.0, RSA laboratories, August 20, 1998.