FPGA Implementation and Analysis of DES and TWOFISH Encryption Algorithms

Thesis report submitted towards the partial fulfillment of

requirements for the award of the degree of

Master of Technology

In

VLSI Design & CAD

Submitted by

Amandeep Singh

Roll No. 600861002

Under the Guidance of

Mrs. Manu Bansal

Assistant Professor, ECED

Department of Electronics and Communication Engineering

THAPAR UNIVERSITY

PATIALA - 147004, INDIA

JULY - 2010

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"FPGA Implementation and Analysis of DES and TWOFISH Encryption Algorithms"** in partial fulfillment of the requirement for the award of degree of M.Tech (VLSI Design & CAD) at Electronic and Communication Department of Thapar University, Patiala, is an authentic record of my own carried out under the supervision of Mrs. Manu Bansal, Assistant Professor, ECED.

The matter embodied in this thesis has not been submitted in any other University/Institute for the award of any degree.

Date: 30/6/2010

Amandeep Singh
Roll No. 600861002

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Mrs. Manu Bansal
Assistant Professor
ECED, Thapar University

Counter signed by:

13.7.10

(Dr. A. K. Chatterjee)
Professor and Head
Electronics and Communication
Engineering Department,
Thapar University, Patiala

14.7.10

(Dr. R. K. Sharma)
Dean (Academic Affairs)
Thapar University,
Patiala.

ACKNOWLEDGEMENT

First of all, I thank the Almighty God, who gave me the opportunity and strength to carry out this work.

I would like to thank **Mrs. Manu Bansal, Assistant Professor (ECED)** for the opportunity to work with her, and also for her encouragement, trust and untiring support. **Mrs. Manu Bansal** has been an advisor in the true sense both academically and morally throughout this project work.

Much appreciation is expressed to **Prof. Abhijit Mukherjee, Director, Thapar University, and Prof. R.K. Sharma, Dean of Academic Affairs** to provide me moral support to go ahead with my M.Tech Thesis work.

I thank to **Mrs. Alpana Agarwal, Assistant Professor and PG Coordinator, ECED, Dr. A.K. Chatterjee, Professor and Head, ECED** and **all the faculty of ECED** for their continuous inspiration during this thesis work.

The paucity of words does not compromise for extending my thanks to my all family members and friends of M.Tech (VLSI & CAD Design) who were always there at the need of hour and helped me in completing this research report.

I am also thankful to the authors whose work has been consulted, utilized and cited in my dissertation.

Amandeep Singh
Roll No. 600861002

ABSTRACT

There are many aspects to security ranging from secure commerce and payments to private communications and protecting passwords. One essential aspect for secure communications is that of secret key Cryptography. It is the automated method in which security goals are accomplished. It includes the process of encryption that converts plain-text into cipher-text. The process of decryption reconverts the cipher-text into plain-text. Secure communication is the prime requirement of every organization. To achieve this, one can use many techniques or algorithms available for Cryptography. Various models were developed for the encryption in which keys were generated from the available data. Data Encryption Standard (DES) is the most commonly used algorithm for security purpose that was announced by National Bureau of Standard. It is the strongest algorithm but due to large processing time, it can be easily broken by the eavesdroppers. Twofish algorithm is nowadays a strongest algorithm due to its large key stream. An efficient encryption algorithm should consist of two factors – fast response and reduced complexity. Cryptosystems must satisfy three general requirements 1) the enciphering and deciphering transformations must be efficient for all keys. 2) System must be easy to use. 3) Security of the system should depend only on the secrecy of the keys and not on the secrecy of the algorithms.

The main objective of this dissertation is to analyze the various aspects of DES and Twofish algorithms.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER

1

INTRODUCTION

Security attacks against network are increasing significantly with time. Our communication media should also be secure and confidential. For this purpose, these three suggestions arrive in every one's mind: (i) one can transmit the message secretly, so that it can be saved from hackers, (ii) the sender ensures that the message arrives to the desired destination, and (iii) the receiver ensures that the received message is in its original form and coming from the right sender. For this, one can use two techniques, (i) one can use invisible ink for writing the message or can send the message through the confidential person, and (ii) one can use a scientific approach called "Cryptography".

Cryptography is the technique used to avoid unauthorized access of data. For example, data can be encrypted using a cryptographic algorithm in conjunction with the key management. It will be transmitted in an encrypted state, and later decrypted by the intended party. If a third party intercepts the encrypted data, it will be difficult to decipher. The security of modern cryptosystems is not based on the secrecy of the algorithm, but on the secrecy of a relatively small amount of information, called a secret key. The fundamental and classical task of cryptography is to provide confidentiality by encryption methods.

Cryptography is used in applications present in technologically advanced societies; examples include the security of ATM cards, computer passwords, and electronic commerce, which all depend on cryptography.

Cryptanalysis is the study used to describe the methods of code-breaking or cracking the code without using the security information, usually used by hackers.

1.1 Terms of Cryptography

The basic terms of Cryptography are:

Plain-text: the original message or data that is in readable form is known as plain-text.

Cipher-text: the encoded message is known as cipher-text.

Encryption: the process to convert the original message into coded form with the help of key, i.e., plain-text into cipher-text is known as encryption.

Decryption: the reverse process of encryption, i.e., to convert cipher-text into plain-text with the help of key is known as decryption.

Key: the key is used to encrypt or decrypt the message. It is of two types:

- Private key
- Public key

1.2 Objectives:

Cryptography is used to achieve the following goals:

Confidentiality: Protection against unauthorized disclosure of information. Confidentiality may be applied to whole messages, parts of messages, and even existence of messages [9]. Confidentiality is the protection of transmitted data from passive attacks.

Authentication: The authentication service is concerned with assuring that a communication is authentic. It is the corroboration of the claimed source of a message. Authentication is of two types: (i) Peer entity, and (ii) Data origin

Data integrity: The integrity can apply to a stream of messages, a single message, or selected fields within a message. It assures that messages are received as sent, with no duplication, insertion, modification, reordering, or replays. The destruction of data is also covered under this service.

Access control: It is the ability to limit and control the access to host systems and applications via communications links. To achieve this, each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual.

Nonrepudiation: Nonrepudiation prevents either sender or receiver from denying a transmitted message. When a message is sent, the receiver can prove that the alleged sender in fact sent the message.

1.3 Cryptography Techniques

There are two techniques used for data encryption and decryption, which are:

1. **Symmetric Cryptography**

   If sender and recipient use the same key then it is known as symmetrical or private key cryptography. It is always suitable for long data streams. Such system is difficult to use in practice because the sender and receiver must know the key. It also requires sending the keys over a secure channel from sender to recipient.

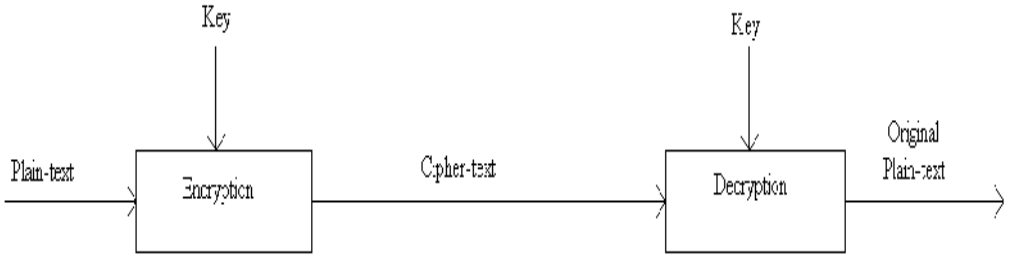


Figure 1.1: Symmetric cryptography

The main concern behind symmetric encryption is how to share the secret key securely between the two parties. If the key gets known for any reason, the whole system collapses. The key management for this type of encryption is troublesome when a unique secret key is used for each peer-to-peer connection. The total number of secret keys to be saved and managed for n-nodes will be $n(n-1)/2$.

There are two cipher methods that are used in symmetric key cryptography: Block cipher and Stream cipher. The block cipher method divides a large data set into blocks and encrypts each block separately. Finally the blocks are combined to produce encrypted data. The stream cipher method encrypts the data as a stream of bits without separating the data into blocks. The stream of bits from the data is encrypted sequentially using some of the results from the previous bit until all the bits in the data are encrypted as a whole.

2. **Asymmetric Cryptography**

   If sender and recipient use different keys then it is known as asymmetrical or public key cryptography. The key used for encryption is called the public key

and the key used for decryption is called the private key. Such technique is used for short data streams and also requires more time to encrypt the data.

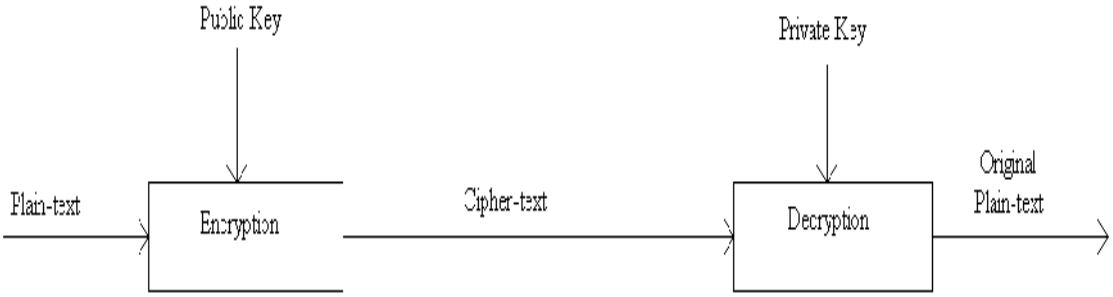


Figure 1.2: Asymmetric cryptography

To encrypt a message, a public key can be used by anyone, but the owner having private key can only decrypt it. There is no need for a secure communication channel for the transmission of the encryption key. Asymmetric algorithms cannot be applied to variable-length streams of data. Asymmetric keys are also called a key-exchange pair. Asymmetric encryption techniques are slower than symmetric techniques, because they require more computational processing power.

To get the benefits of both methods, a hybrid technique is usually used. In this technique, asymmetric encryption is used to exchange the secret key; symmetric encryption is then used to transfer data between sender and receiver. It is easy to convert the private key into public key, but the reverse is very difficult.

1.3.1 Digital Signatures

Public key cryptography gives a major benefit by providing a method for employment of digital signatures. Digital signatures and hand-written signatures both rely on the fact that it is very hard to find two people with the same signature. The authentication and data integrity are the two features of digital signatures by providing a seal over a document or a handwritten signature. Anyone with access to the public key of the signer may verify the signature. For example, in the field of E-commerce, an instruction to your bank to transfer money can be authenticated with a digital signature. Digital signatures cannot be copied to another document.

Digital Signature Generation:

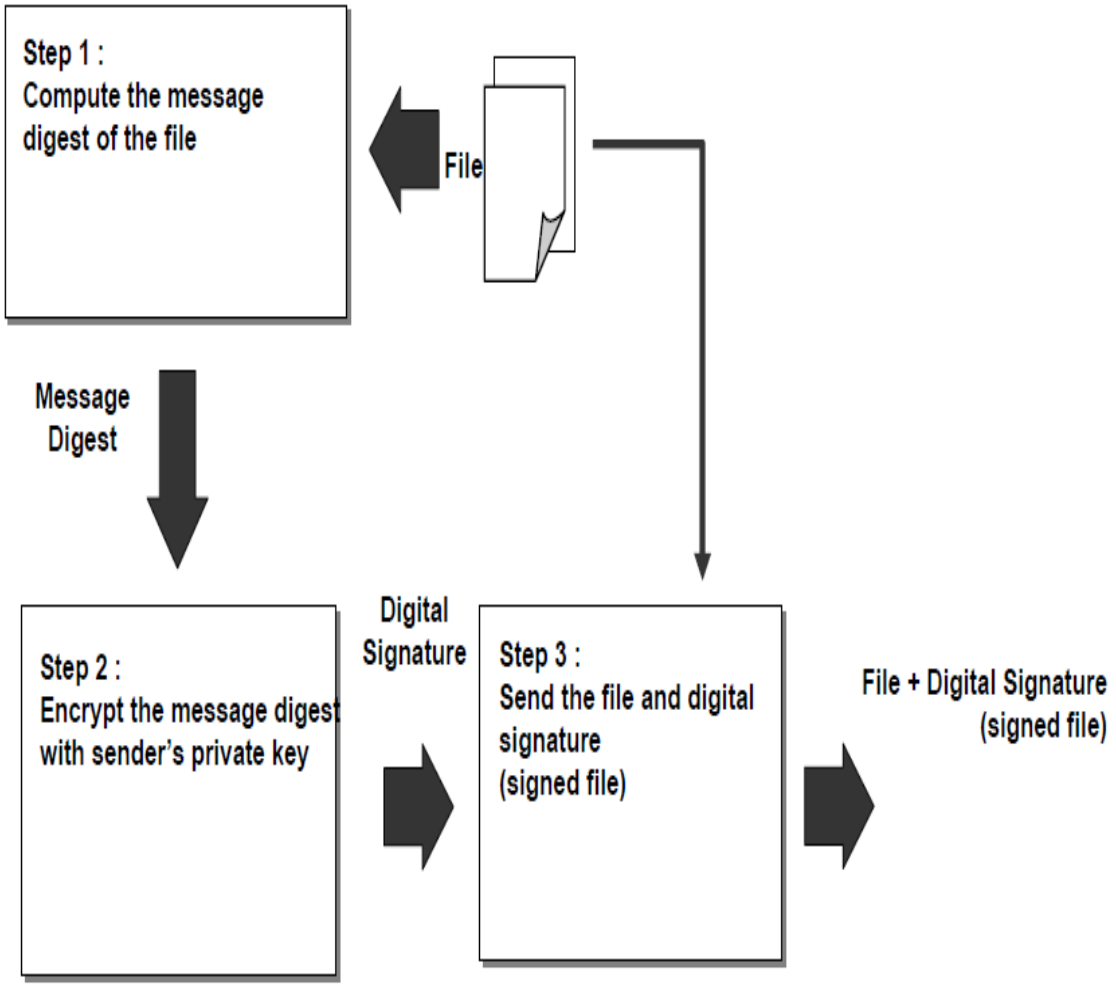


Figure 1.3: Steps for digital signature generation [48]

The steps for the digital signature generation are as follows:

1) The message digest of the plain-text is computed, i.e., what type of data is – text, video, image, *etc.* and what is the length of the data is, *etc.*
2) This message digest will be encrypted using key techniques and a digital signature is also attached before sending to the receiver.
3) The encrypted plain-text (including plain-text and digital signature) is sent to the receiver.

Digital Signature Verification:

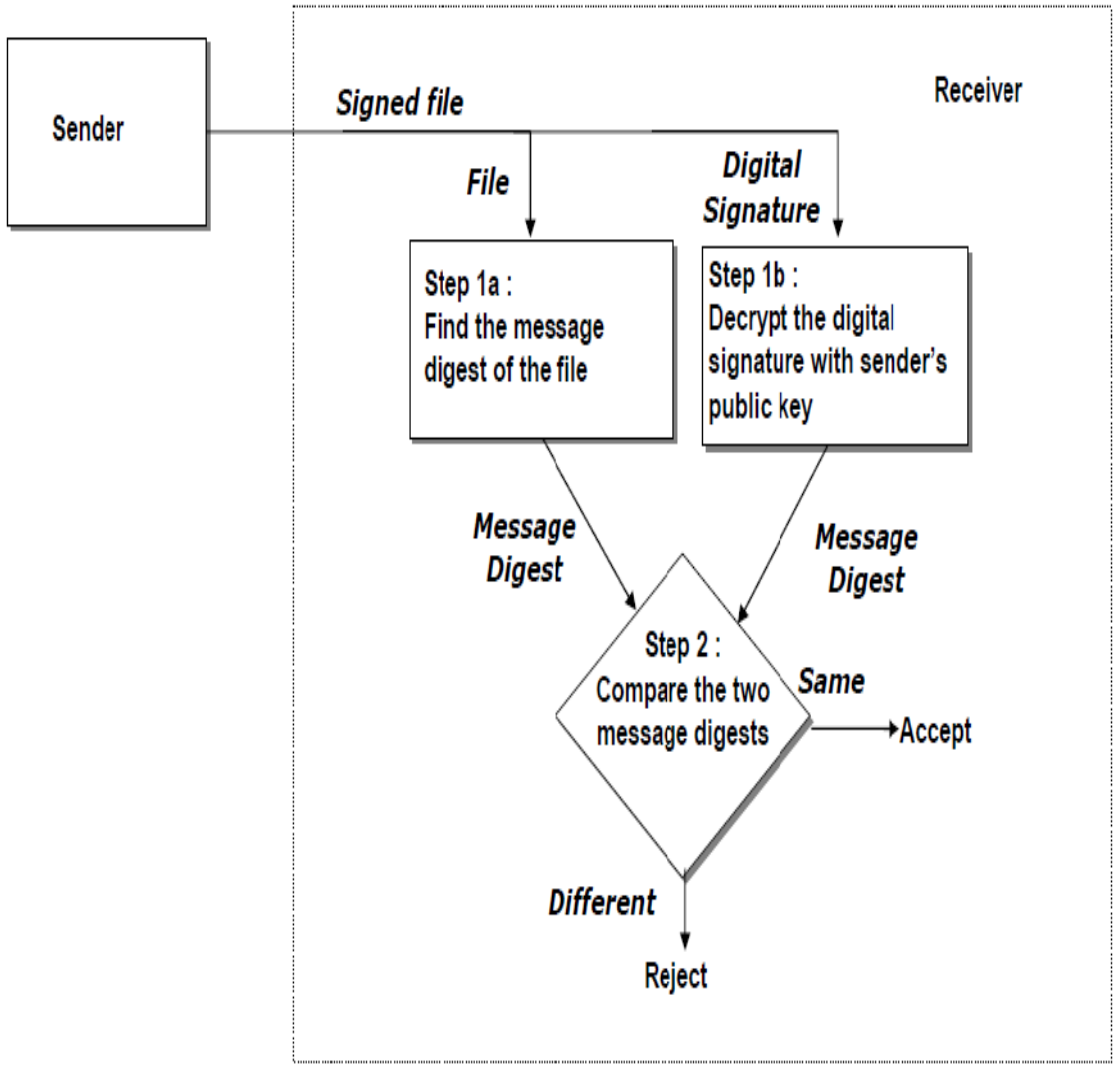


Figure 1.4: Steps for digital signature verification [48]

The steps for the digital signature verification are as follows:

1) The message digest of the file is evaluated for the verification purpose.
2) The encrypted file, i.e., cipher-text is decrypted using the key techniques.
3) After then the decrypted file and original file are compared for the verification. If two files are same then the decrypted file will be accepted, otherwise, it will be rejected.

1.4 Types of Ciphers

1. Substitution Cipher

A substitution cipher substitutes one piece of information for another. This is most frequently done by offsetting letters of the alphabet. For example, if we encode the word "SECRET" using Caesar's key value of 3, we offset the alphabet so that the 3rd letter down (D) begins the alphabet.

So starting with

ABCDEFGHIJKLMNOPQRSTUVWXYZ

and sliding everything up by 3, you get

DEFGHIJKLMNOPQRSTUVWXYZABC

where D=A, E=B, F=C, and so on.

Example:

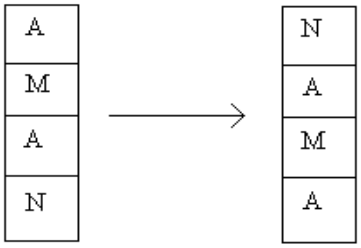
THIS IS A SECRET (key n=3)

WKLV LV D VHFUHW

2. Transposition Cipher

Transposition means rearranging elements in the plain-image. The permutation of bits decreases the perceptual information, whereas the permutation of pixels and blocks produce high-level security. In the bit permutation techniques, the bits in each pixel are permuted using the permutation keys with the key length equal to 8. The number of permutations is = 8! = 40320 and the number of keys are 121. In the pixel permutation, 8 pixels are taken as a group and permuted with the same size key. The block size is (8 × 8) then it is difficult to decrypt. To extract the image, a combinational sequence of permutations and the permutation keys using pseudo random index generators should be known.

Example:



1.5 Key Management

Cryptography can be used as a security mechanism to provide confidentiality, integrity, and authentication, but not if the keys are compromised in any way. The keys have to be distributed to the right entities and updated continuously. The keys need to be protected as they are being transmitted and while they are being stored on each workstation and server. The keys need to be generated, destroyed, and recovered properly. Key management can be handled through manual or automatic processes. The frequency of use of a cryptographic key can have a direct correlation to how often the key should be changed. The more a key is used, the more likely it is to be captured and compromised. Keeping keys secret is a challenging task. Keys should not be in clear-text outside the cryptography device

1.5.1 Rules for keys generation and their handling:

1. The key length should be of variable size for the highly secure communication.
2. Keys should be randomly selected by using the full spectrum of available key-space.
3. Multiple use of keys leads to short lifetime.
4. Keys should be properly destroyed when their lifetime is over.
5. For the secure communication, the keys are to be kept secret.

1.6 Organization of Thesis

The organization of the thesis is as follows:

Chapter 2: Gives the survey of literature conducted by various authors in the field of Cryptography.

Chapter 3: Describes the details of DES Algorithm. In this chapter, the overview and the steps of DES algorithm are discussed.

Chapter 4: Describes the details of Twofish Algorithm. In this chapter, the overview and the steps of Twofish algorithm are discussed.

Chapter 5: Gives the overview of FPGA Spartan 3-E kit used for the implementation of DES algorithm.

Chapter 6: Gives the results and conclusion of work done in thesis.

CHAPTER

2

LITERATURE SURVEY

Cryptography has a long and fascinating history. The most complete non-technical account of the subject is Kahn's "The Code breakers" that include cryptography from its initial and limited use by the Egyptians some 4000 years ago, to the twentieth century where it played a crucial role in the outcome of both world wars [1].

Beginning with the work of Feistel at IBM in the early 1970s and culminating in 1977 with the adoption as a U.S. Federal Information Processing Standard for encrypting unclassified information, DES, the Data Encryption Standard, is the most well-known cryptographic mechanism in history. The most striking development in the history of cryptography came in 1976 when Diffie and Hellman published a transaction "New Directions in Cryptography". This paper introduced the revolutionary concept of public-key cryptography and also provided a new and ingenious method. Before the modern era, cryptography was concerned solely with message confidentiality (i.e. encryption) — conversion of messages from a comprehensible form into an incomprehensible one and back again at the other end, rendering it unreadable without secret knowledge (namely, the key) [2]. In recent decades, the field has expanded beyond confidentiality concerns to include techniques for authentication, digital signatures, interactive proofs, and secure computation. For secure communication simple encryption devices such as Enigma machine were used during World War II having improvement using three rotors to substitute letters, a plug board, and a reflecting rotor. The advancement in encryption methods came with the existence of computers.

In 1978, Ruth M. Davis provides a hardware-implementable algorithm for enciphering data, which has been adopted as a Federal standard to provide a high level of cryptographic protection against various attacks [3].

In 1981, Whitfleld Diffie describes cryptographic technology, which examines the forces driving public development of cryptography [4]. The paper describes how one can secure the message over the telephone lines.

In 1988, Ingrid Verbauwhede describes Security and Performance Optimization of a New DES Data Encryption Chip. Novel CAD tools are used at different steps in the design process for simulation. The result is a single chip of 25 mm in 3-pm double-metal CMOS. Functionality tests show that a clock of 16.7 MHz can be applied, which means that a 32-Mbit/s data rate can be achieved for all eight byte modes [5].

In 1990, James E. Katz provides Social Aspects of Telecommunications Security Policy that describes a system that offers a variety of convenient and powerful services while meeting the legitimate needs of the individual for privacy and of society for security [6].

In 1991, H. Bonnenbergt describes the VLSI implementation of a new block cipher. The chip that runs with a maximum clock frequency of 33 MHz permitting a data conversion rate of more than 55 Mbits/s performs data encryption and decryption in a single hardware unit [7].

In 1992, K.H. Mundt “SUPERCRYPT” ASIC technology facilitates a new device family for data encryption in which semi-custom cell-based ASIC technology is described to get 100Mbits/s encryption speed on silicon applying 1 micron design rules [8].

In 1993, C. Boyd provides the modern data encryption in which proposed standard for digital signatures based on RSA were introduced [9]. A. Curigert describes VINCI: VLSI Implementation of the new secret-key block Cipher IDEA. VINCI’s IDEA premier silicon realization, integrates high-speed encryption and decryption, comprehensive key management functions, and all standardized cipher modes of operation in their ordinary and high-speed adapted versions [10].

In 1994, R. Zimmermann provides a 177 Mb/s VLSI implementation of the international data encryption algorithm in which the VLSI chip implements data

encryption and decryption in a single hardware unit. All-important standardized modes of operation of block ciphers, such as ECB, CBC, CFB, OFB, and MAC, are supported. Moreover, with a system clock frequency of 25 MHz the device permits a data conversion rate of more than 177 Mb/s [11].

In 1995, Stefan Wolter provides the implementation of the IDEA architecture that includes a concurrent self-test based on a mod3 residue code self-checking system. It allows the detection of permanent and temporary single and multiple-bit errors in the IDEA data-path. Hence it provides the secure prevention of faulty encrypted or decrypted data [12].

In 1996, Seung-Jo Han describes the improved DES algorithm in which a 96-bit data block is divided into three 32-bit sub-blocks to increase the Unicity Distance (UD) by performing different f-functions on each of the sub-block [13].

In 1997, Toby Schaffer describes a Flip-Chip implementation of the DES. The flip-chip/MCM environment provides advantages in driver performance, reduced die area requirements, and high bandwidth for triple DES operation [14]. Suan-Suan Chew provides IAuth: An Authentication System for Internet Applications that provides secure distribution of cryptographic keys while establishing authenticity between a user and a Web-based application [15].

In 1998, Hassina Guendouz describes rapid prototype of a fast data encryption standard with integrity processing for cryptographic applications. It implements the DES algorithm in the same silicon area with high-speed performance based on VHDL specifications [16]. K. Wong provides a single-chip FPGA implementation of the Data Encryption Standard (DES) algorithm describing the DES algorithm is secured in a single chip FPGA by loading the configuration data into the chip during the initialization cycle. Once the key is loaded, it is secured inside the chip [17].

In 1999, Yeong-Kang Lai provides a novel VLSI architecture for a variable-length key, 64-bit Blowfish block cipher. It performs thorough evaluation of initialized sub-keys, basic sub-keys generating, establishing a six-stage pipelined data path, and mapping of the algorithm onto the data path for the low power consumption to improve the chip performance [18].

In 2000, M.P. Leong described a bit-serial implementation of the International Data Encryption Algorithm (IDEA) using a novel bit-serial architecture to perform multiplication modulo $2^{16}+1$ by having minimal amount of hardware [19]. R. G. Sixel describes a high level language implementation of the DES and bit-slice architecture. This implementation had two objectives: (i) testing the whole algorithm prior to a VHDL description for future synthesis, and (ii) by making DES available for other applications requiring a software implementation [20]. Teo Pock Chueng provides implementation of pipelined DES using Altera CPLD. The architecture contains of three main parts, DES module, pipeline module and control unit module. Four segments pipeline is used in this architecture to burst the throughput of DES and Altera Hardware Description Language (AHDL) is used to implement the pipelined DES design for better output [21]. Cameron Patterson provides high performance DES encryption in Virtex FPGAs using Jbits. Jbits provides a Java-based Application Programming Interface (API) for the run-time creation and also for the modification of the configuration bit-stream. This allows dynamic circuit specialization based on a specific key and mode and when combined with a speed efficient layout, the result is a throughput of over 10 Gbits/s [22].

In 2001, Pui-Lam Siu provides a low power asynchronous DES that can be used in contactless smart cards with some advantages like low power consumption and no global clock [23]. N. Sklavos describes asynchronous low power VLSI implementation of IDEA. This approach proved efficiently the lowest power consumption of the asynchronous implementation compared to the existing synchronous [24]. Ahmet Eskicioglu provides a paper on cryptography. It concludes that early classical ciphers' substitution and transposition operations form the building blocks for today's powerful ciphers such as DES. Key agreement schemes allow communicating parties to establish a shared cipher key [25]. Deng Liang describes an efficient and scalable VLSI implementation of DES by using 1.2µm CMOS technology to implement a testing chip for testing the performance of the design [26].

In 2002, Yeong-Kang Lai represents the VLSI Architecture Design and Implementation for TWOFISH Block Cipher. In this paper the security of the Twofish Encryption algorithm was increased by using loop-folding technique with efficient hardware mapping [27]. Touria ARICH provides Hardware implementations of the Data

Encryption Standard in Electronic Code Book mode (ECB) using hardware description language VHDL [28]. Chih-Chung Lu describes an Integrated Design of Advanced Encryption Standard (AES) Encrypter and Decrypter. This method is used to make it a very low-complexity architecture, especially in saving the hardware resource in implementing the AES (Inv) Sub-Bytes module and (Inv) Mix columns module, etc [29].

In 2003, G. Catalini provides Modified Twofish Algorithm for increasing Security and Efficiency in the Encryption of video signals. The proposed system was tested on H 263+ coded signals [30]. M. McLoone provides high-performance FPGA implementation of DES using a novel method for implementing the key schedule based on Xilinx Virtex technology. Utilizing the novel method, a 16-stage pipelined DES design is achieved, which can run at an encryption rate of 3.87 Gbits/s [31].

In 2004, Bo Yang describes scan based side channel attack on dedicated hardware implementations of Data Encryption Standard. Scan chains can be used as a side channel to recover secret keys from a hardware implementation of the Data Encryption Standard [32]. Liakot Ali describes the implementation of triple data encryption algorithm and VHDL [33]. Tariq Jamil provides the Rijndael algorithm: A brief introduction to the new encryption standard. In overall performance, based on speed of encryption/decryption process and key set-up time, the Rijndael algorithm has attained top scores in tests conducted by NIST [34].

In 2005, Aamer Nadeem provides a performance comparison of data encryption algorithms in which various algorithms were compared and it was found that Blowfish algorithm is the best algorithm in view of processing time and security [35]. A. Ammar introduced random data encryption algorithm in pseudo-randomized cipher keys were used for greater security and higher throughput [36]. Jingmei Liu provides an AES S-box to increase complexity and cryptographic analysis. An improved AES S-box is presented to improve the complexity of AES S-box algebraic expression with terms increasing from 9 to 255 and algebraic degree invariable. The improved AES S-box also has better properties of Boolean functions in SAC and balance, and is capable of attacking against differential cryptanalysis with high reliable security [37].

In 2006, Chih-Hsu Yen described simple error detection methods for hardware implementation of AES by using a $(n+1,n)$ CRC where n can be 4,8,16 [38]. Alireza

Hodjat describes Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/s AES Processors. This paper describes different pipelined implementations of the AES algorithm as well as the design decisions and the area optimizations that lead to a low area and high throughput AES encryption processor are presented. Moreover, by pipelining the composite field implementation of the byte substitution phase of the AES algorithm (inner-round pipelining), the area consumption is reduced up to 35 percent [39].

In 2007, A.Chandra Sekhar provides data encryption technique using Random number generator using the recurrence matrices and a quadruple vector. It provides data encryption at two levels and hence security against crypto analysis is achieved at relatively low computational overhead using the mod function [40]. M.R.M. Rizk described an optimized area and optimized speed hardware implementations of AES on FPGA, the code was written in VHDL and synthesized and verified using Xilinx ISE 7.1 and simulated using Modelsim [41].

In 2008, Jing Wang provides improved DES algorithm based on irrational numbers. An improved scheme based on irrational numbers that enhances the randomness of sub-Key is proposed. The permutation is controlled by irrational number, i.e., considered as false chaos [42]. Md. Nazrul Islam describes the effect of security increment to symmetric data encryption through AES methodology. A new algorithm was proposed that was more securing than Rijndael algorithm but with less efficiency [43]. The diffusive property of three encryption algorithms – Rijndael, Twofish, and Safer+ is compared by using simple test vectors. It was conducted by Sapiee Haji Jamel and Mustafa Mat Deris [44].

The performance evaluation approach for a model was proposed in 2009 by Hung-Min Sun [45]. It provides the information about the Security of an Efficient Time-Bound Hierarchical Key Management Scheme. Tingyuan Nie describes a study of DES and Blowfish Encryption Algorithm. In this paper the function run time of both algorithms was compard [46]. Ashwini M. Deshpande describes FPGA Implementation of AES Encryption and Decryption by using VHDL language and ModelSim SE PLUS 5.7g software for Simulation. Encryption and decryption routines are fully functional at 50 and 100 MHz [47].

Many approaches for encryption have been studied in this chapter. The ideas of different authors have been explained. The different methods for reducing the processing time and for enhancing the security have also been explained. The need of Cryptography and the benefits of using it have been explained. As a result, the processing time of DES algorithm is very large. In order to reduce it, many techniques can be used such as, addition, subtraction, or any other logical implementation.

CHAPTER

3

DES ALGORITHM

Data Encryption Standard (DES) is a cryptographic standard that was proposed as the algorithm for secure and secret items in 1970 and was adopted as an American federal standard by National Bureau of Standards (NBS) in 1973. DES is a block cipher, which means that during the encryption process, the plain-text is broken into fixed length blocks and each block is encrypted at the same time. Basically it takes a 64 bit input plain text and a key of 64-bits (only 56 bits are used for conversion purpose and rest bits are used for parity checking) and produces a 64 bit cipher text by encryption and which can be decrypted again to get the message using the same key.

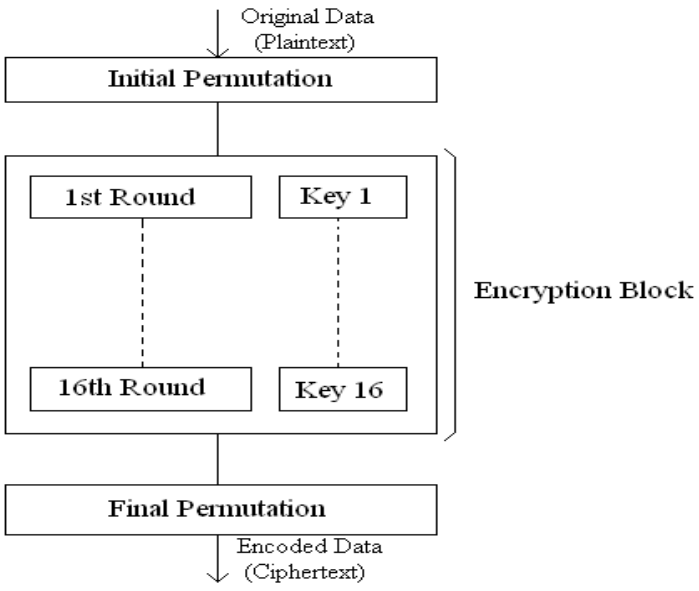


Figure 3.1: DES Flow

3.1 DES Core

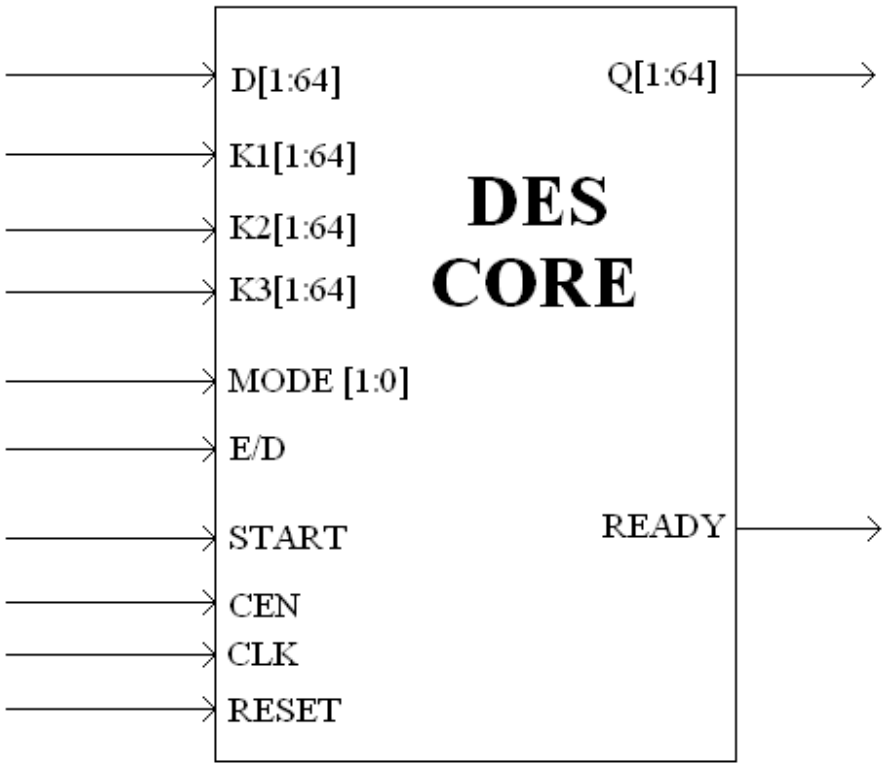


Figure 3.2: DES Core [52]

Pin name	I/O	Function
CLK	I	Core clock signal
RESET	I	Core reset signal
CEN	I	Synchronous enable signal. When LOW the core ignores all its inputs and all its outputs must be ignored
START	I	When goes HIGH, a cryptographic operation is started
E/D	I	Encrypt – 1 / Decrypt – 0
MODE [1:0]	I	DES Mode 0 – Single DES 1 – Double DES 2 – Triple DES

READY	O	Output data ready and valid
K1 [1:64]	I	First Encryption Key
K2 [1:64]	I	Second Encryption Key (used only in Double or triple mode)
K3 [1:64]	I	Third Encryption Key (used only in Triple mode)
D [1:64]	I	Input Plain or Cipher Text Data
Q [1:64]	O	Output Cipher or Plain Text Data

Table 3.1: Pin configuration of DES core [52]

3.2 DES Algorithm

The algorithm's overall structure is shown in fig 3.2 and the Fiestel diagram is shown in fig 3.3. The algorithm employs three different types of operations: Permutations, Rotations, and Substitutions. Between the initial and final transpositions, the algorithm performs 16 iterations of a function. The general depiction of DES encryption algorithm which consists of initial permutation of the 64 bit plain text and then goes through 16 rounds, where each round consists permutation and substitution of the text bit and the inputted key bit, and at last goes through a inverse initial permutation to get the 64 bit cipher text. DES standard is used very rarely because of its less key-space as the security of DES algorithm resides in the key-length. It is improved using irrational numbers as irrational numbers provide high security during encryption by controlling the permutation. The key-space can also be increased using this technique or performing the DES operation three times can also increase it. DES standard can also be made secure by loading the data configuration into single chip FPGA during the initialization cycle.

Basically, FPGAs provide a flexible, reliable and efficient way of synthesizing the complex logic in a regular structure consisting of predefined programmable blocks and predefined programmable routing resources. Encryption process in case of DES requires various operations and multiple iterations. It also includes sub-keys, which are the subset

of 56 bits key used in initial round. The section describes the steps for encryption used in DES. Utilizing the steps of DES, the model is proposed in which variable key length is used to encrypt the data. Key generation mechanism used in the model, generates the sub keys from the S-Boxes. Model also includes the recovery mechanisms in order to provide strength to the keys.

Modern applications of DES cover a wide variety of applications, such as secure internet (*ssl*), electronic financial transactions, remote access servers, cable modems, secure video surveillance and encrypted data storage.

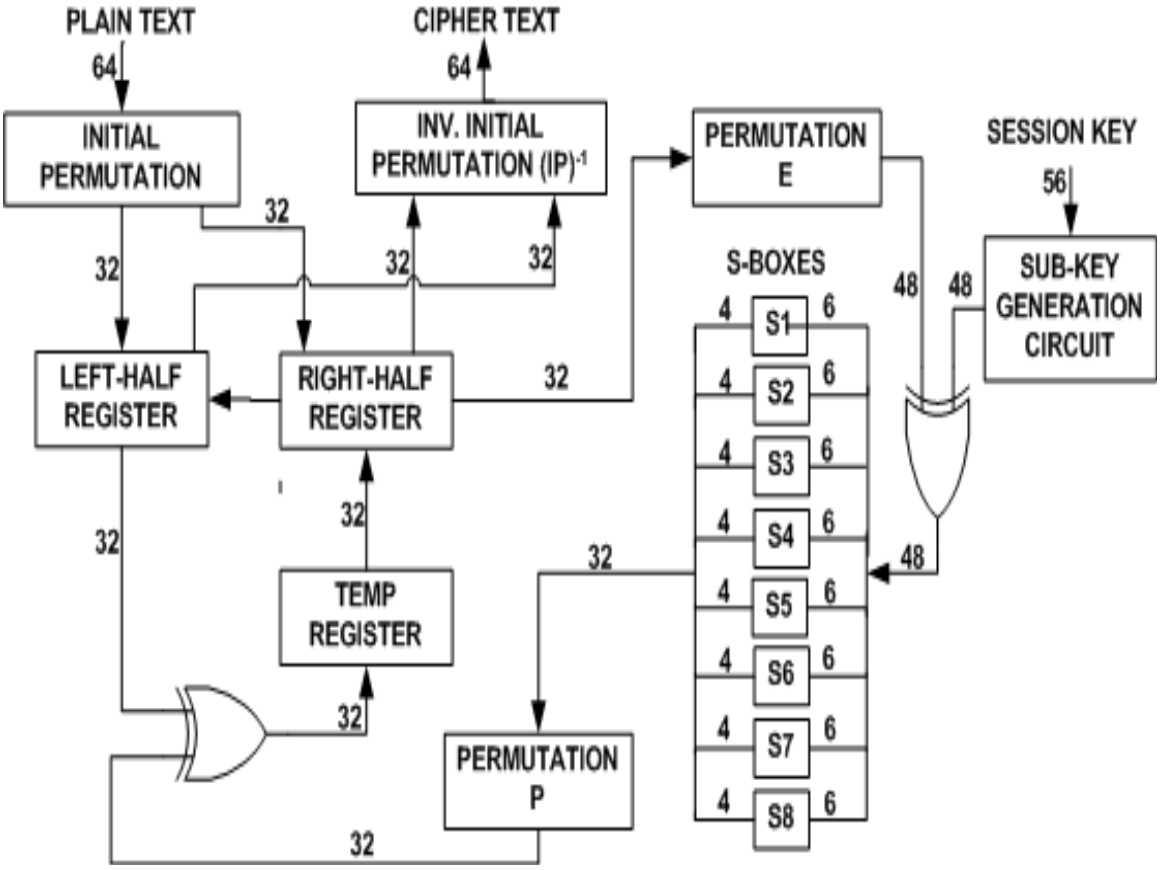


Figure 3.3: DES Algorithm [49]

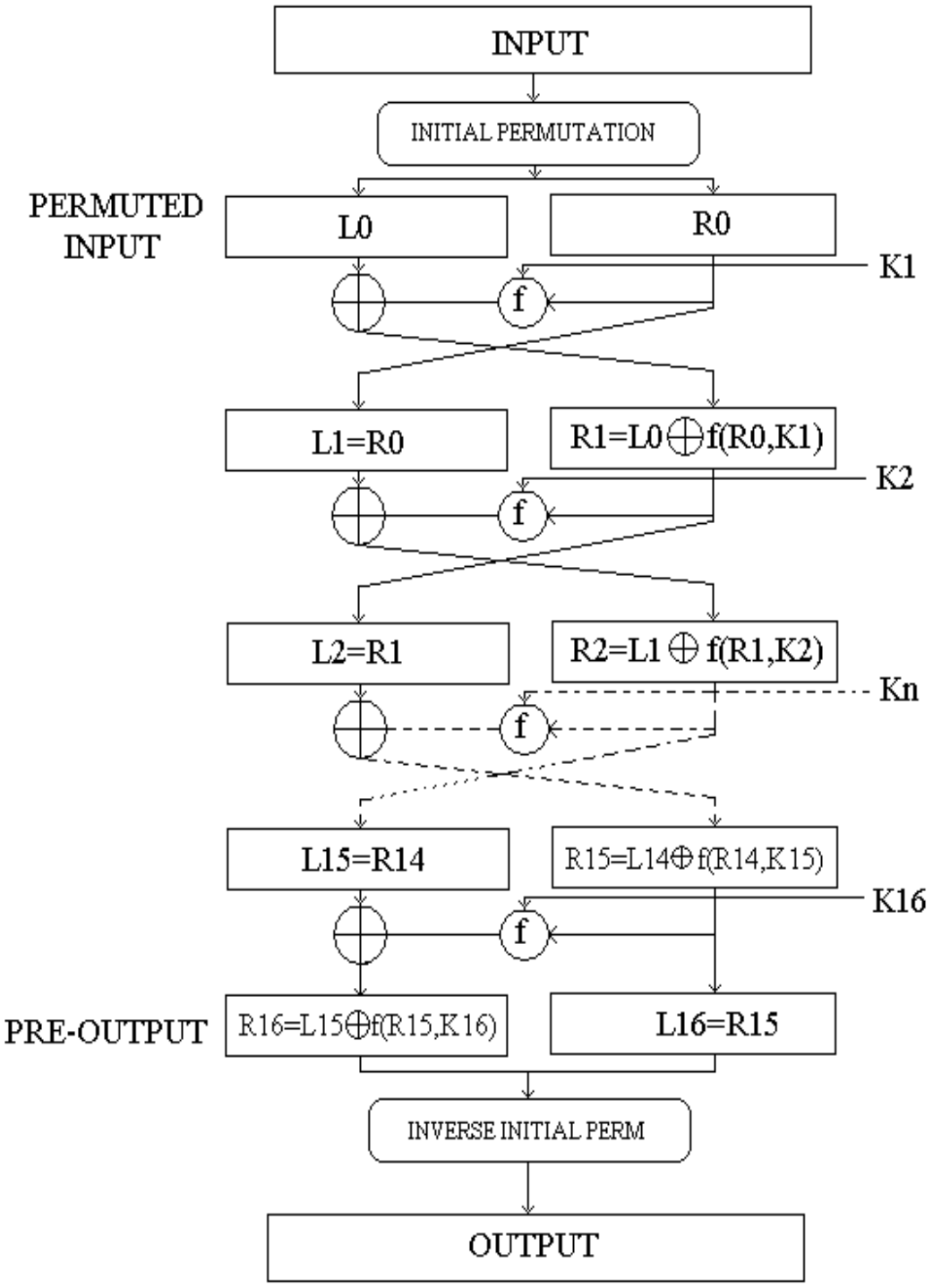


Figure 3.4: DES Fiestel Structure [51]

3.3 Steps for Algorithm [54]

Step 1: Create 16 sub-keys, each of which is 48-bits long.

The 64-bit key is permuted according to Permuted Choice – 1 table as shown in table 3.2. The first bit of K+ will be the 57th bit of K and the 2nd bit of K+ will be the 49th bit of K and so on. The last bit of K+ will be the 4th bit of K. Here it must be noted that K (i.e., original key) is of 64 bits and K+ (i.e., permuted key) is of 56 bits.

PC-1: Permuted Choice 1							
Bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**
1	57	49	41	33	25	17	9
8	1	58	50	42	34	26	18
15	10	2	59	51	43	35	27
22	19	11	3	60	52	44	36
29	63	55	47	39	31	23	15
36	7	62	54	46	38	30	22
43	14	6	61	53	45	37	29
50	21	13	5	28	20	12	4

Table 3.2: Permuted Choice 1

Example: The original 64-bit key

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11100001

we get the 56-bit permuted key

K+ = 11110000 11001100 10101010 01110101 01010110 01100111 10001111

Next split this key into left and right halves, C_0 and D_0, each having 28 bits.

Example: From the permuted key K+, we get

C_0 = 11110000 11001100 10101010 0111
D_0 = 0101 01010110 01100111 10001111

Now we can create sixteen blocks C_n and. D_n, where n ranges from 1 to 16. Each pair of blocks C_n and D_n is formed from the previous pair C_{n-1} and D_{n-1}, respectively, for n = 1, 2, ..., 16, using the sub-key rotation mechanism of "left shifts" of the previous block.

Sub-key Rotation Table																
Round Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Number of bits to rotate	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 3.3: Sub-key Rotation Table

This means, for example, C_3 and D_3 are obtained from C_2 and D_2, respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15}, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left

Example: From original pair C_0 and D_0 we obtain:

C_1 = 11100001100110010101010011111
D_1 = 10101010110011001111100011110

C_2 = 11000011001100101010100111111
D_2 = 01010101100110011111000111101

C_3 = 00001100110010101010011111111
D_3 = 01010110011001111100011110101

C_4 = 00110011001010101001111111100
D_4 = 01011001100111100011110101010

C_5 = 11001100101010100111111110000
D_5 = 01100110011110001111010101 01

C_6 = 00110010101010011111111000011
D_6 = 10011001111000111101010101 01

C_7 = 11001010101001111111100001100
D_7 = 01100111100011110101010110

C_8 = 00101010100111111110000110011
D_8 = 10011110001111010101011001

C_9 = 01010101001111111100001100110
D_9 = 00111100011110101010110011

C_{10} = 01010100111111100001100110 01
D_{10} = 11110001111010101010110011 00

C_{11} = 01010011111110000110011001 01
D_{11} = 11000111101010101011001100 11

C_{12} = 01001111111000011001100101 01
D_{12} = 00011110101010101100110011 11

C_{13} = 00111111100001100110010101 01
D_{13} = 01111010101010110011001111 00

C_{14} = 11111110000110011001010101 00
D_{14} = 11101010101011001100111100 01

C_{15} = 11111000011001100101010100 11
D_{15} = 10101010101100110011111000 111

C_{16} = 11110000110011001010101001 11
D_{16} = 01010101011001100111100011 11

Now we will obtain the keys K_n, for $1<=n<=16$, by using the permuted choice – 2 table to each of the concatenated pairs C_nD_n. Here the input to the table is of 56 bits, but the output will be of only 48 bits.

PC-2: Permuted Choice 2						
Bit	**0**	**1**	**2**	**3**	**4**	**5**
1	14	17	11	24	1	5
7	3	28	15	6	21	10
13	23	19	12	4	26	8
19	16	7	27	20	13	2
25	41	52	31	37	47	55
31	30	40	51	45	33	48
37	44	49	39	56	34	53
43	46	42	50	36	29	32

Table 3.4: Permuted Choice 2

Therefore, the first bit of K_n is the 14th bit of C_nD_n, the second bit the 17th, and so on, ending with the 48th bit of K_n being the 32th bit of C_nD_n.

Example: For the first key we have C_1D_1 = 11100001 10011001 01010100 11111010 10101100 11001111 00011110

which, after we apply the permutation PC-2, becomes

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

For the other keys we have

K_2 = 011110 011010 111011 011001 110110 111100 100111 100101
K_3 = 010101 011111 110010 001010 010000 101100 111110 011001
K_4 = 011100 101010 110111 010110 110110 110011 010100 011101
K_5 = 011111 001110 110000 000111 111010 110101 001110 101000

K_6 = 011000 111010 010100 111110 010100 000111 101100 101111

K_7 = 111011 001000 010010 110111 111101 100001 100010 111100

K_8 = 111101 111000 101000 111010 110000 010011 101111 111011

K_9 = 111000 001101 101111 101011 111011 011110 011110 000001

K_{10} = 101100 011111 001101 000111 101110 100100 011001 001111

K_{11} = 001000 010101 111111 010011 110111 101101 001110 000110

K_{12} = 011101 010111 000111 110101 100101 000110 011111 101001

K_{13} = 100101 111100 010111 010001 111110 101011 101001 000001

K_{14} = 010111 110100 001110 110111 111100 101110 011100 111010

K_{15} = 101111 111001 000110 001101 001111 010011 111100 001010

K_{16} = 110010 110011 110110 001011 000011 100001 011111 110101

Step 2: Encode each 64-bit block of data.

In this step, we have to obtain the initial permutation of data input D. This rearranges the bits according to the table as shown in table 3.5. The 58th bit of D becomes the first bit of IP. The 50th bit of D becomes the second bit of IP. The 7th bit of D is the last bit of IP.

IP: Initial Permutation								
Bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	58	50	42	34	26	18	10	2
9	60	52	44	36	28	20	12	4
17	62	54	46	38	30	22	14	6
25	64	56	48	40	32	24	16	8
33	57	49	41	33	25	17	9	1
41	59	51	43	35	27	19	11	3
49	61	53	45	37	29	21	13	5
57	63	55	47	39	31	23	15	7

Table 3.5: Initial Permutation

Example: Applying the initial permutation to the block of text M, given previously, we get

D = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

Here the 58th bit of D is "1", which becomes the first bit of IP. The 50th bit of D is "1", which becomes the second bit of IP. The 7th bit of D is "0", which becomes the last bit of IP.

Next divide the permuted block IP into a left half L_0 of 32 bits, and a right half R_0 of 32 bits.

Example: From IP, we get L_0 and R_0

L_0 = 1100 1100 0000 0000 1100 1100 1111 1111
R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

We now proceed through 16 rounds, for 1<=n<=16, using a function f which operates on two blocks--a data block of 32 bits and a key K_n of 48 bits--to produce a block of 32 bits. Let + denote XOR addition, (bit-by-bit addition modulo 2). Then for n going from 1 to 16 we calculate

$L_n = R_{n-1}$
$R_n = L_{n-1} + f(R_{n-1},K_n)$

This results in a final block, for n = 16, of $L_{16}R_{16}$, i.e., in each round, we take the right 32 bits of the previous result and make them the left 32 bits of the current step. For the right 32 bits in the current step, we XOR the left 32 bits of the previous step with the calculation f.

Example: For n = 1, we have

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

$L_1 = R_0$ = 1111 0000 1010 1010 1111 0000 1010 1010

$R_1 = L_0 + f(R_0,K_1)$

Procedure to compute *f* – Function:

To calculate *f*, we first expand each block R_{n-1} from 32 bits to 48 bits. This is done by using a E – bit selection table (shown in table 3.6) that repeats some of the bits in R_{n-1} . Thus E(R_{n-1}) has a 32 bit input block, and a 48 bit output block. Let E be such that the 48 bits of its output, written as 8 blocks of 6 bits each.

E-Bit Selection Table						
Bit	**0**	**1**	**2**	**3**	**4**	**5**
1	32	1	2	3	4	5
7	4	5	6	7	8	9
13	8	9	10	11	12	13
19	12	13	14	15	16	17
25	16	17	18	19	20	21
31	20	21	22	23	24	25
37	24	25	26	27	28	29
43	28	29	30	31	32	33

Table 3.6: E-bit Selection Table

Thus the first three bits of E(R_{n-1}) are the bits in positions 32, 1 and 2 of R_{n-1} while the last 2 bits of E (R_{n-1}) are the bits in positions 32 and 1.

Example: We calculate E(R_0) from R_0 as follows:

R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

E (R_0) = 011110 100001 010101 010101 011110 100001 010101 010101

Next in the *f* calculation, we XOR the output E(R_{n-1}) with the key K_n:

$$K_n + \mathrm{E}\,(R_{n-1}).$$

Example: For K_1 , E(R_0), we have

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010
E (R_0) = 011110 100001 010101 010101 011110 100001 010101 010101
K_1+E(R_0) = 011000 010001 011110 111010 100001 100110 010100 100111.

We have not yet finished calculating the function *f* . To this point we have expanded R_{n-1} from 32 bits to 48 bits, using the selection table, and XORed the result with the key K_n . We now have 48 bits, or eight groups of six bits. We now do something strange with each group of six bits: we use them as addresses in tables called "S boxes". Each group of six bits will give us an address in a different S box. Located at that address will be a 4 bit number. This 4 bit number will replace the original 6 bits. The net result is that the eight groups of 6 bits are transformed into eight groups of 4 bits (the 4-bit outputs from the S boxes) for 32 bits total.

Write the previous result, which is 48 bits, in the form:

$$K_n + \mathrm{E}\,(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8,$$

where each B_i is a group of six bits. We now calculate

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$$

where $S_i(B_i)$ referres to the output of the *i*-th S box and *B* is a block of 6 bits.

To repeat, each of the functions *S1, S2,..., S8*, takes a 6-bit block as input and yields a 4-bit block as output. The table to determine S_1 is shown and explained below:

S-Box 1: Substitution Box 1																
Row/ Column	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Table 3.7: S-box 1

The first and last bits of B represent in base 2 a number in the decimal range 0 to 3 (or binary 00 to 11). Let that number be i. The middle 4 bits of B represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Let that number be j. Look up in the table the number in the i-th row and j-th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input B. For example, for input block B = 011011 the first bit is "0" and the last bit "1" giving 01 as the row. This is row 1. The middle four bits are "1101". This is the binary equivalent of decimal 13, so the column is column number 13. In row 1, column 13 appears 5. This determines the output; 5 is binary 0101, so that the output is 0101. Hence S_1 (011011) = 0101. The tables defining the functions $S_1,...,S_8$ are the following:

S-Box 1: Substitution Box 1																
Row/ Column	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S-Box 2: Substitution Box 2																
Row/ Column	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10

1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S-Box 3: Substitution Box 3

Row/ Column	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S-Box 4: Substitution Box 4

Row/ Column	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S-Box 5: Substitution Box 5

Row/ Column	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S-Box 6: Substitution Box 6

Row/ Column	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	115	5	2	8	12	3	7	0	4	10	1	13	11	6

3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S-Box 7: Substitution Box 7

Row/ Column	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	1	113	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S-Box 8: Substitution Box 8

Row/ Column	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 3.8: S-boxes

Example: For the first round, we obtain as the output of the eight S boxes:

$K_1 + E(R_0) = 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111.$

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8) = 0101\ 1100\ 1000\ 0010\ 1011\ 0101\ 1001\ 0111$

The final stage in the calculation of *f* is to do a permutation P of the S-box output to obtain the final value of *f*:

$$f = P(S_1(B_1)\ S2(B_2)...S_8(B_8))$$

The permutation P is defined in the following table. P yields a 32-bit output from a 32-bit input by permuting the bits of the input block.

P Permutation				
Bit	**0**	**1**	**2**	**3**
1	16	7	20	21
5	29	12	28	17
9	1	15	23	26
13	5	18	31	10
17	2	8	24	14
21	32	27	3	9
25	19	13	30	6
29	22	11	4	25

Table 3.9: P-Permutation Table

Example: From the output of the eight S boxes:

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8) = 0101\ 1100\ 1000\ 0010\ 1011\ 0101\ 1001\ 0111$$

we get

$$f = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

$R_1 = L_0 + f(R_0 , K_1)$

= 1100 1100 0000 0000 1100 1100 1111 1111
+ 0010 0011 0100 1010 1010 1001 1011 1011
= 1110 1111 0100 1010 0110 0101 0100 0100

In the next round, we will have $L_2 = R_1$, which is the block we just calculated, and then we must calculate $R_2 = L_1 + f(R_1, K_2)$, and so on for 16 rounds. At the end of the sixteenth round we have the blocks L_{16} and R_{16}. We then *reverse* the order of the two blocks into the 64-bit block

$$R_{16}L_{16}$$

and apply a final permutation IP^{-1} as defined by the following table:

IP^(-1): Inverse Initial Permutation								
Bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	40	8	48	16	56	24	64	32
9	39	7	47	15	55	23	63	31
17	38	6	46	14	54	22	62	30
25	37	5	45	13	53	21	61	29
33	36	4	44	12	52	20	60	28
41	35	3	43	11	51	19	59	27
49	34	2	42	10	50	18	58	26
57	33	1	41	9	49	17	57	25

Table 3.10: Inverse Initial Permutation

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.

Example: If we process all 16 blocks using the method defined previously, we get, on the 16th round,

L_{16} = 0100 0011 0100 0010 0011 0010 0011 0100
R_{16} = 0000 1010 0100 1100 1101 1001 1001 0101

We reverse the order of these two blocks and apply the final permutation to

$R_{16}L_{16}$ = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100

IP^{-1} = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101

which in hexadecimal format is

69BE85B7C35D19EE.

This is the encrypted form of D = 0123456789ABCDEF

Decryption is simply the inverse of encryption, following the same steps as above, but reversing the order in which the sub-keys are applied.

CHAPTER

4

TWOFISH ALGORITHM

Twofish is a 128-bit block cipher proposed by Schneier et al. It can also work with 192- or 256-bit key lengths. A Feistel network structure, which was made specifically for DES algorithm and was a successful one, can also be used for Twofish algorithm having 16 rounds. A *Feistel network* is a general method of transforming any function (usually called the *F* function) into a permutation. It was invented by Horst Feistel. In a Feistel network, the round function consists of taking one part of the data being encrypted, feeding it into some key dependent function *F*, and then XORing the result into another part of the block. Two rounds of Feistel network are known to be as 'one cycle', therefore, Twofish algorithm is an 8-cycle algorithm having 16 rounds. The structure of the Twofish algorithm is shown in figure 4.1. The only difference between the two Feistel network structures is the two fixed rotations by one bit, performed together with the XOR operations on outputs of the F-function.

Twofish exhibits fast and versatile performance across most platforms; it performs well both in hardware and in memory-constrained environments. Twofish can be optimized for speed, key setup, memory, code size in software, or space in hardware. Encrypt data in less than 500 clock cycles per block on an Intel Pentium, Pentium Pro, and Pentium II, for a fully optimized version of the algorithm.

The Twofish algorithm consists of two types of operations (as shown in figure 4.1)

Byte-oriented: It includes the whitening operation and F-function.

Non-byte-oriented: It includes two sub-key additions modular 2^{32} comprising of 1-bit rotates, addition with sub-keys, and PHT (pseudo-Hadamard transform).

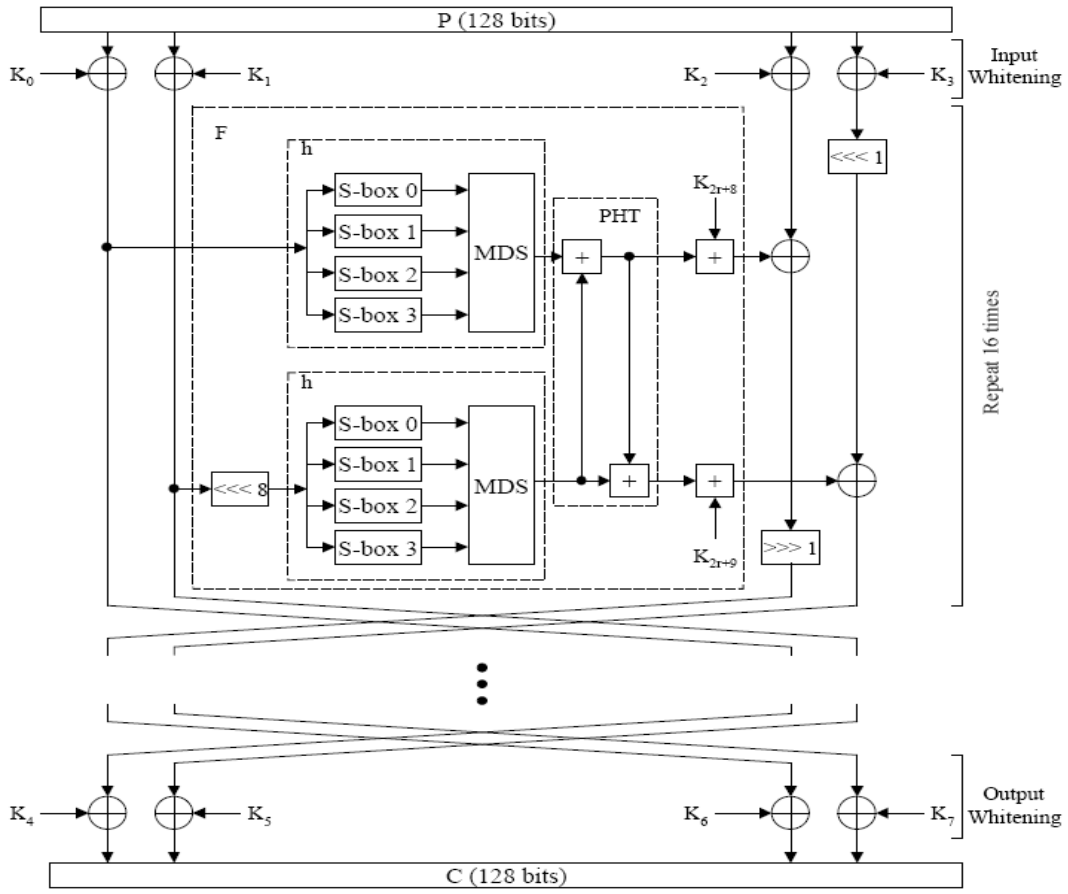


Figure 4.1: Feistel Structure for Twofish Algorithm [55]

4.1 Whitening Operation [55, 56]

It includes the XORing of input and output data with eight sub-keys (K0 – K7). Thus this operation is performed only at the input level, i.e., before 1st round, hence called input whitening and at the output level, i.e., after 16th round, hence called output whitening (as mentioned in figure 4.1). The whitening operation is actually used to increases the difficulty for the attackers, to search for key, by hiding the inputs to 1st and last round. The sub-keys (K0 – K7) used in this operation are also calculated in the same manner as the other round sub-keys, and are not be used in other operations.

4.2 F – function [55, 56]

It consists of five operations:

1. Fixed left rotation by 8 bits
2. Key-dependent S-boxes
3. Maximum Distance Separable (MDS) matrices
4. Pseudo-Hadamard Transform (PHT)
5. Two sub-key additions modular 2^{32}

4.3 S-boxes [55, 56]

An S-box is a table-driven substitution operation used in many algorithms. It can vary in both input and output size and can be made randomly or algorithmically. There are four kinds of S-boxes used in Twofish algorithm. Four different S-boxes together with the MDS matrix form an h-function. This h-function appears two times in the algorithm, which causes significant redundancy. In Twofish, each S-box consists of three 8-by-8-bit fixed permutations chosen from a set of two possible permutations, q0 and q1. The structure of all S-boxes is shown in figure 4.2.

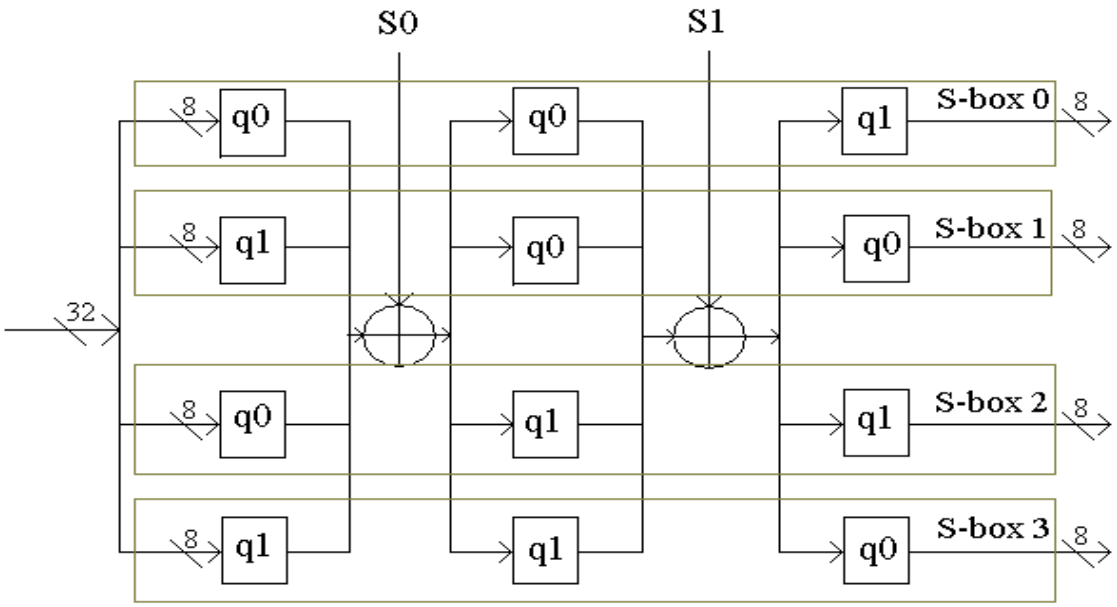


Figure 4.2: Structure for S-boxes

The XOR operations are performed with sub-keys S0, S1 between the three permutations. These sub-keys are computed only once for a particular global key, and stay fixed during the entire encryption and decryption process.

Each q-permutation represents a fixed function by having a regular structure as shown in figure 4.3. The main components of the q-permutations are 4-by-4-bit fixed S-boxes t0...t3. Both permutations q0 and q1 have the same internal structure, and differ only in the contents of the S-boxes t0...t3.

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
q0	t0	8	1	7	D	6	F	3	2	0	B	5	9	E	C	A	4
	t1	E	C	B	8	1	2	3	5	F	4	A	6	7	0	9	D
	t2	B	A	5	E	6	D	9	0	C	8	F	3	2	4	7	1
	t3	D	7	F	4	1	2	6	E	9	B	3	0	8	5	C	A
q1	t0	2	8	B	D	F	7	6	E	3	1	9	4	0	A	C	5
	t1	1	E	2	B	4	C	3	7	6	D	A	5	F	9	0	8
	t2	4	C	7	5	1	6	9	A	0	E	D	8	2	B	3	F
	t3	B	9	5	1	C	3	D	E	6	4	7	F	2	0	8	A

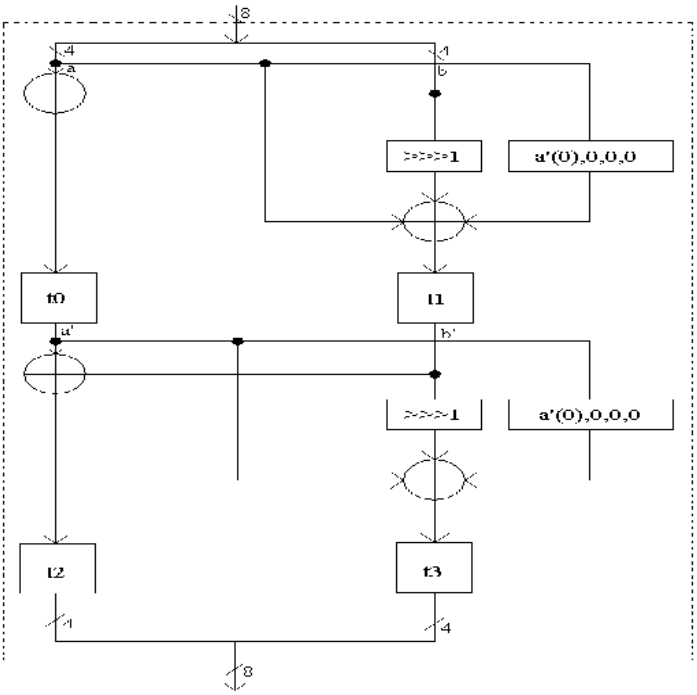


Figure 4.3: q-permutation Structure

4.4 MDS Matrices [55, 56]

Twofish algorithm uses a single 4-by-4 Maximum Distance Separable (MDS) matrix. The transformation performed by this matrix is described by the formula:

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \bullet \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

where y3...y0 are consecutive bytes of the input 32-bit word (y3 is the most significant byte), and z3...z0 form the output word.

This matrix multiplies a 32-bit input value by 8-bit constants, with all multiplications performed (byte by byte) in the Galois field GF (28). The primitive polynomial is x8 + x6 + x5 + x3 + 1. Only three different multiplications are used effectively in the MDS matrix, namely multiplication

- by 5B16 = 0101 10112 (represented in GF(28) by a polynomial x6 + x4 + x3 + x + 1),
- by EF16 = 1110 11112 (x7 + x6 + x5 + x3 + x2 + x + 1), and
- by 0116 = 0000 00012 (equivalent element in GF(28) is just 1) - obviously the result is equal to the input value.

MDS matrices are useful building blocks for ciphers because they guarantee a certain degree of diffusion. If one of the input elements is changed, all the output elements must change. If two input elements are changed, all but one of the output elements must change, etc.

4.5 Pseudo-Hadamard Transforms [55, 56]

A pseudo-Hadamard transform (PHT) is a simple mixing operation that is very efficient in software. Given two inputs, *a* and *b*, the 32-bit PHT is defined as:

$a' = a + b \bmod 2^{32}$

$b' = a + 2b \bmod 2^{32}$

Twofish uses a 32-bit PHT to mix the outputs from its two parallel 32-bit h-functions.

4.6 Steps for Twofish Algorithm [55, 56]

Step 1: The plaintext is split into four 32-bit words, i.e, P_0 - P_3 .

Step 2: In the input whitening step, these are XORed with four key words.

$$R_{0,i} = P_i \oplus K_i \; i = 0,...,3$$

This is followed by 16 rounds.

Step 3: In each round, the two words on the left are used as input to the h-functions; one of them is rotated by 8 bits first. The h-function consists of four byte-wide key-dependent S-boxes, followed by a linear mixing step based on an MDS matrix. The third word is XORed with the first output of F and then rotated right by one bit. The fourth word is rotated left by one bit and then XORed with the second output word of F. Finally, the two halves are exchanged. Thus,

$$\begin{aligned}
\left(F_{r,0}, F_{r,1}\right) &= F\left(R_{r,0}, R_{r,1}, r\right) \\
R_{r+1,0} &= \mathrm{ROR}\left(R_{r,2} \oplus F_{r,0}, 1\right) \\
R_{r+1,1} &= \mathrm{ROL}\left(R_{r,3}, 1\right) \oplus F_{r,1} \\
R_{r+1,2} &= R_{r,0} \\
R_{r+1,3} &= R_{r,1}
\end{aligned}$$

Step 4: The results of the two h-functions are combined using a Pseudo-Hadamard Transform (PHT), and two round key words are added.

Step 5: These two results are then XORed into the words on the right (one of which is rotated left by one bit first, the other is rotated right by one bit afterwards).

Step 6: The left and right halves are then swapped for the next round.

Step 7: After all the rounds, the swap of the last round is reversed, and the four words are XORed with four more key words to produce the cipher-text, which is known as output whitening.

CHAPTER

5

FPGA OVERVIEW

The Xilinx Spartan – 3E (XC3S500) FPGA kit is used for implementation of encryption algorithms [57]. The Xilinx ISE 9.2i tool is used for the design. Some specifications of Spartan – 3E kit are as following:

- Upto 232 user – I/O pins
- Over 10,000 logic cells
- 2 – line, 16 – character LCD screen
- PS/2 mouse or keyboard port
- VGA display port
- Two 9 – pin RS – 232 ports (DTE/DCE)
- 50 MHz clock oscillator
- Chipscope SoftTouch debugging port
- Eight discrete LEDs
- Four slide switches
- Four push-button switches
- Speed Grade -4
- FG320 package

5.1 Introduction to FPGA

A Field Programmable Gate Array (FPGA) is a reprogrammable logic device used as a reprogrammable alternative to ASIC chip devices. FPGAs can be used for specific

operational behavior, or general purpose CPU functionality depending on the complexity of the device. FPGA applications include DSP applications, imaging, speech recognition, cryptography, hardware emulation and for many other application specific uses. Many FPGAs are implementing shared general purpose CPUs on chip for shorter latency times between operations resulting in higher performance of the overall system.

FPGA is an integrated circuit (IC) that includes a two-dimensional array of

- Configurable Logic Blocks (CLBs)
- Configurable I/O Blocks (IOBs)

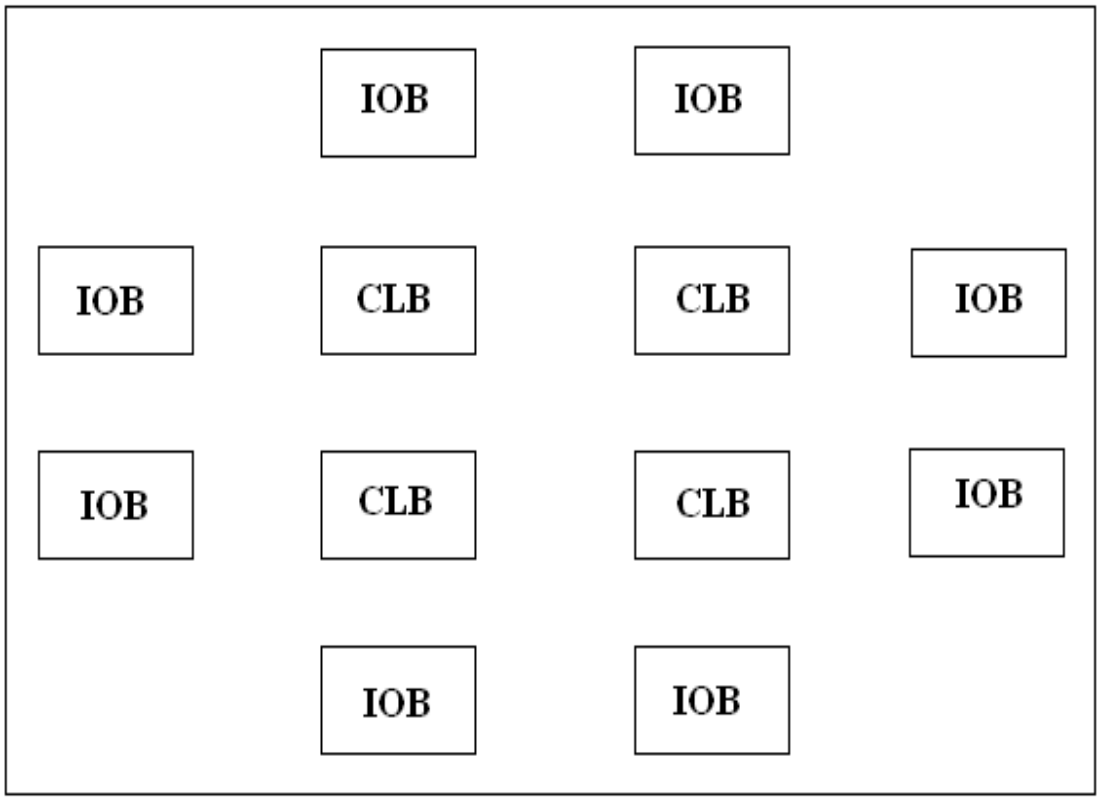


Figure 5.1: FPGA Architecture

FPGAs can be classified as:

- One time programmable
    - Fuses (destroy internal links with current)
    - Anti-fuses (grow internal links)
    - PROM
- Reprogrammable

- EPROM
- EEPROM
- Flash
- SRAM - volatile

5.2 Design Flow

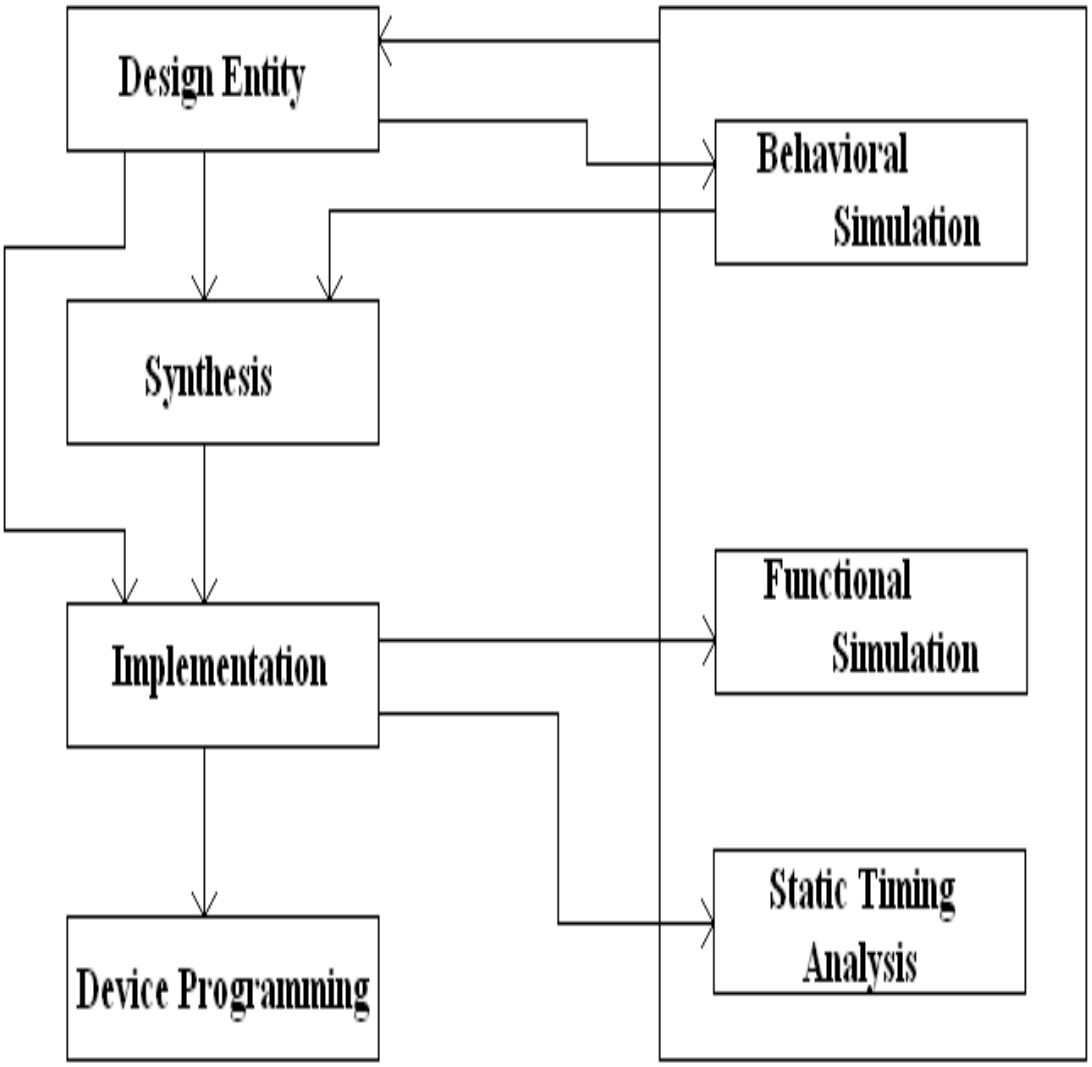


Figure 5.2: FPGA Design Flow

5.2.1 Design Entity

The HDL languages – Verilog or VHDL are used to design the architecture of the system. A VHDL design can be processed by its entity and architecture declarations. The architecture declaration can be of behavioral, dataflow, or structural modeling style.

5.2.2 Behavioral Simulation

By using behavioral simulation we can verify the functionality of the code designed for our system. Various modifications can be done in case of getting errors in the code.

5.2.3 Synthesis

The synthesis process includes the compilation of the sub-modules in the main module and the analysis of the hierarchy of the design. It checks the syntax of the design gives the output in the netlist format which is saved to an Native Generic Circuit (NGC).

5.2.4 Design Implementation

It includes the following three processes:

1. Translation
2. Mapping
3. Place & Route

Translation process compiles all the input netlists and constraints to a logic design file, namely – Native Generic Database (NGD) file having extension .ngd. In this process, the ports are assigned to physical elements such as pins, switches, buttons, etc. This information is stored in the UCF (User Constraint file) file.

Mapping process divides the whole circuit into sub-blocks so that they can fit into FPGA blocks. It includes the mapping of input NGD file logic into targeted FPGA elements such as CLBs and IOBs

Place and Route process places the sub-blocks from map process into the logic blocks according to the constraints and connects the logic blocks. This can be done by using Place And Route (PAR) program. This program takes the mapped NCD file as input and gives the completely routed NCD file as output.

5.2.5 Functional Simulation

This simulation process is used to verify the functionality of the design after the design implementation

5.2.6 Static Timing Analysis

The static timing analysis can be done by using following three types:

1. Post-fit Static Timing Analysis
2. Post-Map Static Timing Analysis
3. Post-Place and Route Static Timing Analysis

5.2.7 Device Programming

At last the FPGA device is to be programmed by configuring the device. In this process the .bit file is to be assigned to the device which is to be programmed and bye-passing the other devices.

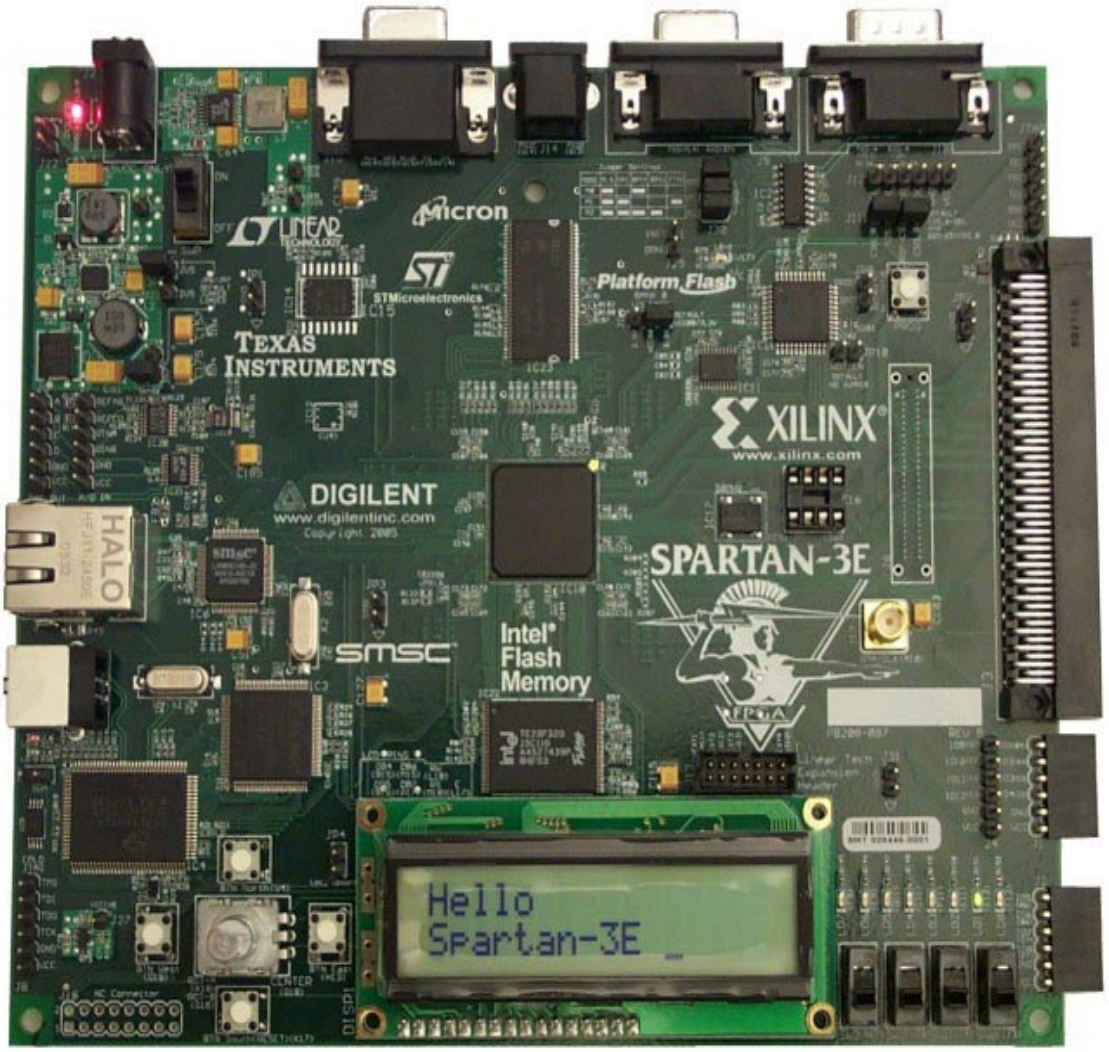


Figure 5.3: FPGA Spartan – 3E Kit

CHAPTER

6

EXPERIMENTAL RESULTS

6.1 Simulation Results of DES Algorithm

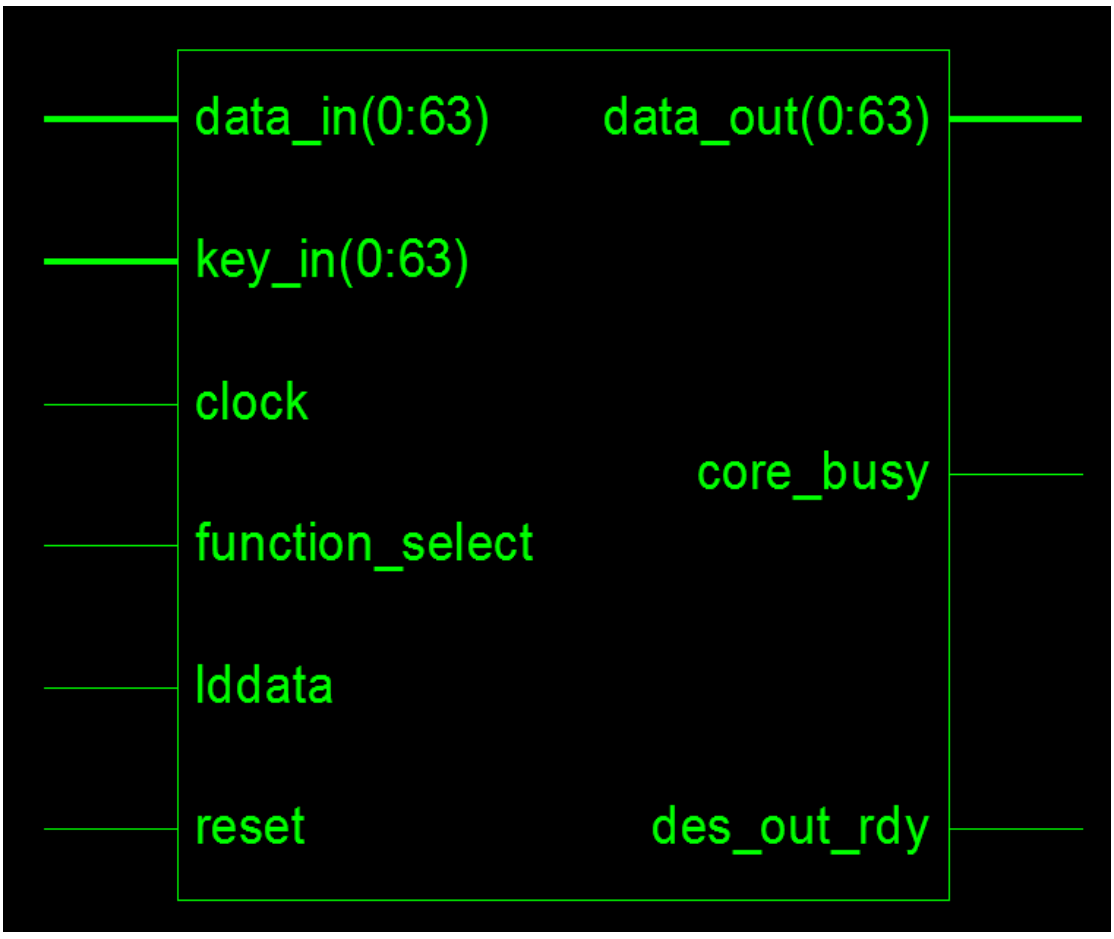


Figure 6.1: DES Block

Figure 6.1 shows the block of DES algorithm showing input and output pins. In this we give 64-bit data and 64-bit key, so that it gives us 64-bit encrypted data after processing through the whole encryption process including 16 rounds. In this, function select is to be 1 for the encryption process. The simulation results are shown in figure 6.2.

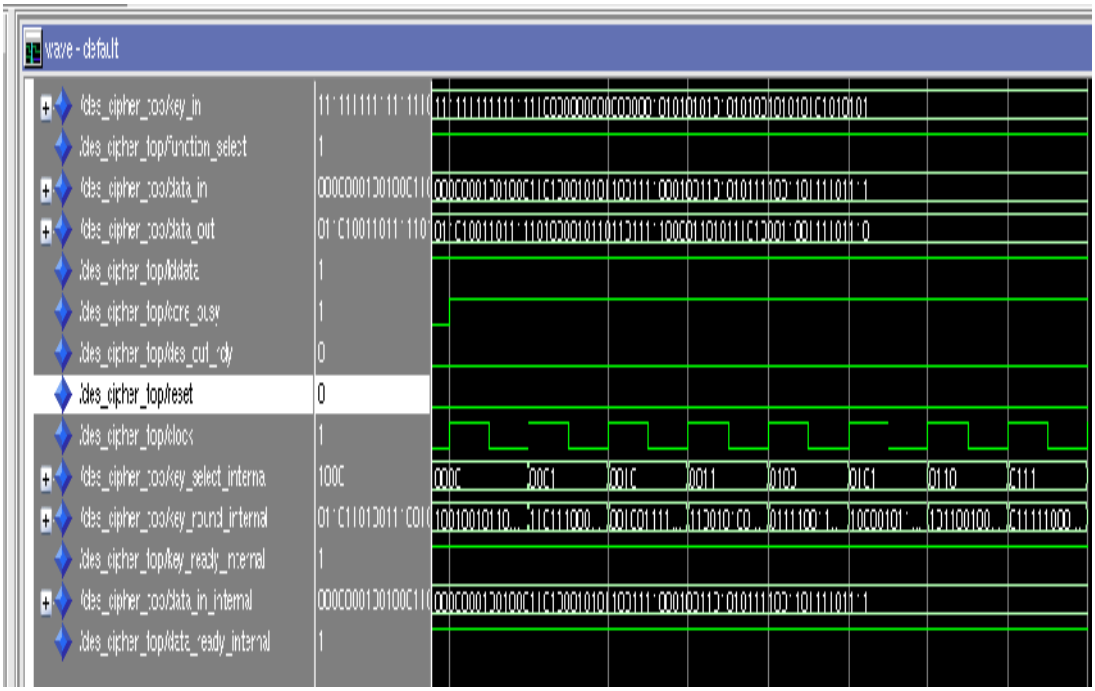


Figure 6.2: Simulation result of DES Encryption by Model Sim

Description:-

key_in : 64 – bit (1111111111111111000000000000000010101010101010100101010101010101)

function_select : '1' for encryption
'0' for decryption

data_in : 64 – bit (0000000100100011010001010110011110001001101010111100110111101111)

lddata : 1

data_out : 64 – bit (0110100110111110100001011011011011111000001101011101000110011110)

reset : 0

clock : 1

core_busy : 1

6.2 FPGA Implementation of DES Algorithm

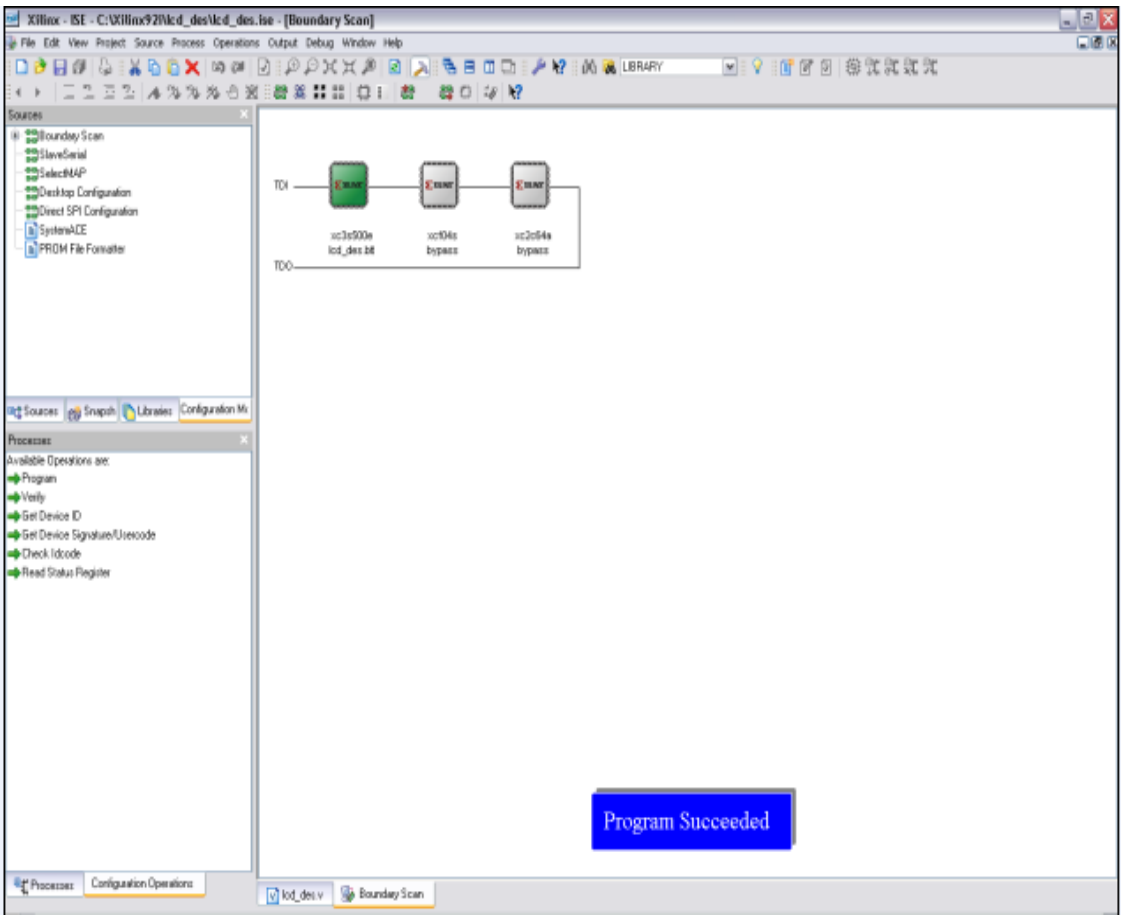


Figure 6.3: Programming Completion of DES Encryption

Figure 6.3 shows the successful completion of programming of device by showing a message “Program Succeeded”. The final output, i.e., the 64-bit encrypted data in the form of cipher-text is as shown in figure 6.4. The final output is shown on LCD of FPGA Spartan – 3E kit.

Figure 6.4: Output Display of DES Encrypted Data

6.3 Simulation Results of TWOFISH Algorithm

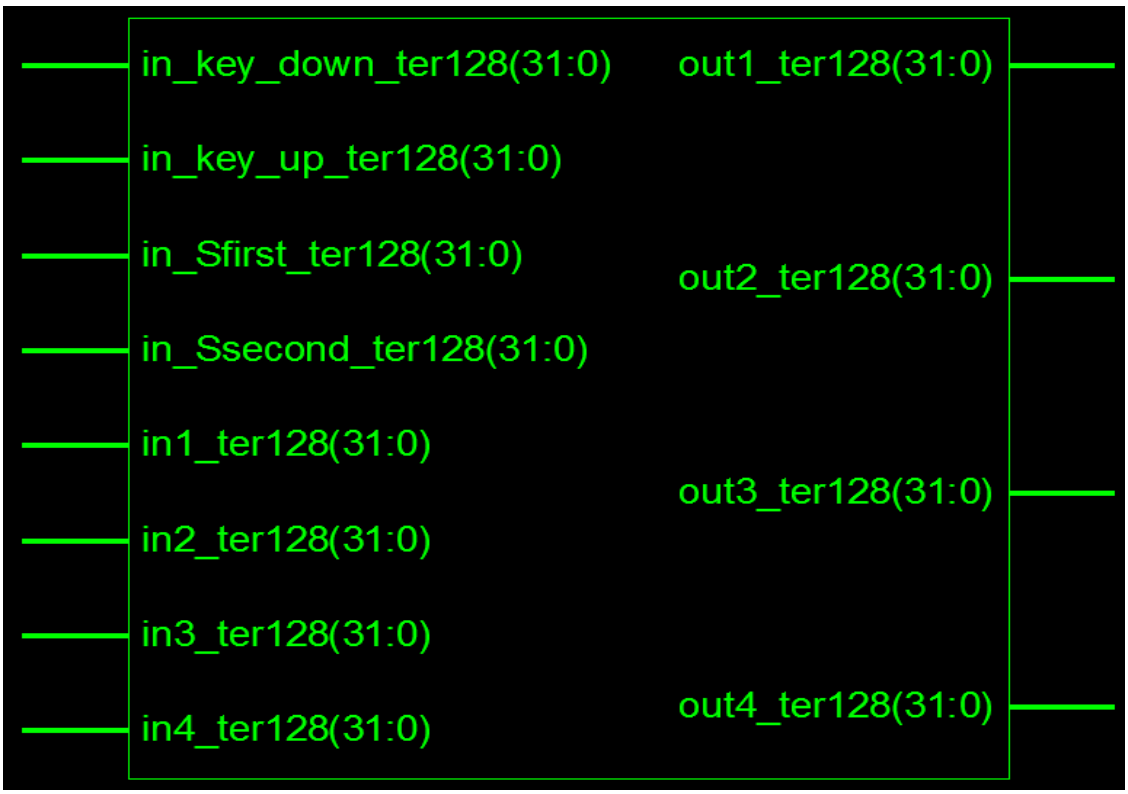


Figure 6.5: Twofish Encryption Block

Figure 6.5 shows the encryption block of Twofish algorithm showing input and output pins. In this we give 128-bit data in four parts each of 32-bit and 128-bit key, so that it gives us 128-bit encrypted data after processing through the whole encryption process including 16 rounds. The simulation results are shown in figure 6.6.

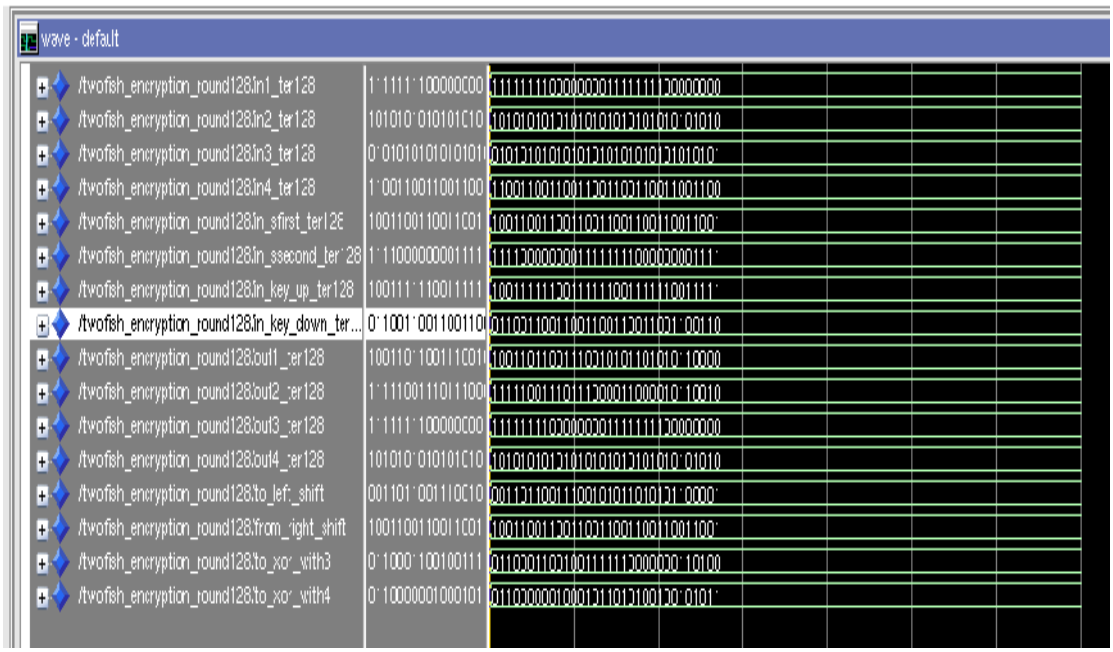


Figure 6.6: Simulation result of Twofish Encryption by Model Sim

Description:-

in1_ter128	:	32 – bit (11111111000000001111111100000000)
in2_ter128	:	32 – bit (10101010101010101010101010101010)
in3_ter128	:	32 – bit (01010101010101010101010101010101)
in4_ter128	:	32 – bit (11001100110011001100110011001100)
in_Sfirst_ter128	:	32 – bit (10011001100110011001100110011001)
in_Ssecond_ter128	:	32 – bit (11110000000011111111000000001111)
in_key_up_ter128	:	32 – bit (10011111100111111001111110011111)
in_key_down_ter128	:	32 – bit (01100110011001100110011001100110)
out1_ter128	:	32 – bit (10011011001110010101101010110000)
out2_ter128	:	32 – bit (11111001110111000011000010110010)
out3_ter128	:	32 – bit (11111111000000001111111100000000)

out4_ter128	:	32 – bit (10101010101010101010101010101010)
to_left_shift	:	32 – bit (00110110011100101011010101100001)
from_right_shift	:	32 – bit (10011001100110011001100110011001)
to_xor_with3	:	32 – bit (01100011001001111100000000110100)
to _xor_with4	:	32 – bit (01100000010001011010100100101011)

6.4 Synthesis Report of LCD Programming

---- Source Parameters

Input File Name : "lcd_des.prj"

Input Format : mixed

Ignore Synthesis Constraint File : No

---- Target Parameters

Output File Name : "lcd_des"

Output Format : NGC

Target Device : xc3s500e-4-fg320

---- Source Options

Top Module Name : lcd_des

Automatic FSM Extraction : Yes

FSM Encoding Algorithm : Auto

Safe Implementation : No

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
Asynchronous To Synchronous	: NO
Multiplier Style	: auto
Automatic Register Balancing	: No

---- Target Options

Add IO Buffers	: Yes
Global Maximum Fanout	: 500
Add Generic Clock Buffer(BUFG)	: 24
Register Duplication	: Yes
Slice Packing	: Yes
Optimize Instantiated Primitives	: No
Use Clock Enable	: Yes
Use Synchronous Set	: Yes
Use Synchronous Reset	: Yes
Pack IO Registers into IOBs	: auto
Equivalent register Removal	: Yes

---- General Options

Optimization Goal	: Speed
Optimization Effort	: 1
Library Search Order	: lcd_des.lso
Keep Hierarchy	: No
RTL Output	: Yes
Global Optimization	: All Clock Nets
Read Cores	: YES
Write Timing Constraints	: No
Cross Clock Analysis	: No

```
Hierarchy Separator                 : /
Bus Delimiter                       : <>
Case Specifier                      : Maintain
Slice Utilization Ratio             : 100
BRAM Utilization Ratio              : 100
Verilog 2001                        : Yes
Auto BRAM Packing                   : No
Slice Utilization Ratio Delta       : 5

Macro Statistics
 64x6-bit ROM                        : 1
26-bit up counter                   : 1
1-bit register                      : 8
 6-bit register                      : 1
 7-bit register                      : 1
1-bit xor2                          : 1
# IOs                               : 9
# BELS                              : 104
#      GND                          : 1
#      INV                          : 2
#      LUT1                         : 25
#      LUT2                         : 1
#      LUT4                         : 13
#      MUXCY                        : 25
#      MUXF5                        : 7
#      MUXF6                        : 4
#      VCC                          : 1
#      XORCY                        : 25
# FlipFlops/Latches                 : 34
#      FD                           : 33
#      FDR                          : 1
```

```
# Shift Registers                 : 7
#      SRL16                      : 7
# Clock Buffers                   : 1
#      BUFGP                      : 1
# IO Buffers                      : 8
#      OBUF                       : 8
Number of Slices                 : 26 out of   4656     0%
 Number of Slice Flip Flops      : 34 out of   9312     0%
 Number of 4 input LUTs          : 48 out of   9312     0%
 Number used as logic            : 41
 Number used as Shift registers  : 7
 Number of IOs                   : 9
 Number of bonded IOBs           : 9 out of    232     3%
 Number of GCLKs                 : 1 out of     24     4%
 NET clk PERIOD = 20 nS HIGH 10 nS
 Clock period                    : 4.823ns (frequency: 207.340MHz)
 Total number of paths / destination ports: 425 / 42
```

6.5 Design Statistics of DES Algorithm

Particulars	Des_cipher_top	Key_scheduling	Des_top	Block_top	Add_key	E_expansion_function
IOs	198	119	187	176	144	80
BELS	18	415	660	368	48	-
GND	1	1	1	1	-	
INV	1	1	1	-	-	-
LUT2	-	2	21	48	48	-
LUT2_D	-	-	27	-	-	-
LUT3	2	318	37	-	-	-
LUT3_D	-	-	2	-	-	-
LUT3_L	2	-	1	-	-	-
LUT4	10	11	347	190	-	-
LUT4_D	-	-	4	-	-	-
LUT4_L	-	-	65	-	-	-
MUXF5	1	70	121	96	-	-
MUXF6	-	12	32	32	-	-
VCC	1	1	1	1	-	-
Flip-Flops/Latches	11	57	144	-	-	-
FDR	10	1	11	-	-	-
FDE	-	56	132	-	-	-
FDRS	1	-	1	-	-	-
I/O Buffers	68	110	186	176	144	80
IBUF	2	61	116	112	96	32
OBUF	66	49	70	64	48	48
BUFGP (Clock)	1	1	1	-	-	-

Particulars	S_box	S1_box	S2_BOX	S3_box	S4_box	S5_box	S6_box	S7_box	S8_box
IOs	80	10	10	10	10	10	34	10	10
BELS	257	33	33	34	34	34	34	34	35
GND	1	1	1	1	1	1	1	1	1
INV	-	-	-	-	-	-	-	-	-
LUT2	-	-	-	-	-	-	-	-	-
LUT2_D	-	-	-	-	-	-	-	-	-
LUT3	-	-	-	-	-	-	-	-	-
LUT3_D	-	-	-	-	-	-	-	-	-
LUT3_L	-	-	-	-	-	-	-	-	-
LUT4	158	19	19	20	20	20	20	20	20
LUT4_D	-	-	-	-	-	-	-	-	--
LUT4_L	-	-	-	-	-	-	-	-	-
MUXF5	65	8	8	8	8	8	8	8	9
MUXF6	32	4	4	4	4	4	4	4	4
VCC	1	1	1	1	1	1	1	1	1
Flip-Flops/Latches	-	-	-	-	-	-	-	-	-
FDR	-	-	-	-	-	-	-	-	-
FDE	-	-	-	-	-	-	-	-	-
FDRS	-	-	-	-	-	-	-	-	-
I/O Buffers	80	10	10	10	10	10	10	10	10
IBUF	48	6	6	6	6	6	6	6	6
OBUF	32	4	4	4	4	4	4	4	4
BUFGP (Clock)	-	-	-	-	-	-	-	-	-

6.6 Device Utilization Summary of DES Algorithm

Particulars	Des_cipher_top	Key_scheduling	Des_top	Block_top	Add_key	E_expansion_function
No. of Slices (4656)	7	188	275	142	28	0
No. of Slice F/Fs (9312)	11	-	144	-	-	-
No. of 4-i/p LUTs (9312)	15	332	505	238	48	-
Bonded IOBs (232)	69	111	187	176	144	80
GCLKs (24)	1	1	1	-	-	-

Particulars	S_box	S1_box	S2_BOX	S3_box	S4_box	S5_box	S6_box	S7_box	S8_box
No. of Slices (4656)	13	13	13	13	13	13	13	13	13
No. of Slice F/Fs (9312)	-	-	-	-	-	-	-	-	-
No. of 4-i/p LUTs (9312)	20	20	20	20	20	20	20	20	20
Bonded IOBs (232)	10	10	10	10	10	10	10	10	10
GCLKs (24)	-	-	-	-	-	-	-	-	-

6.7 Macro Statistics of DES Algorithm

Particulars	Des_cipher_top	Key_scheduling	Des_top	Block_top	Add_key	E_expansion_function
FSMs	1	-	1	-	-	-
4-bit adder	1	-	1	-	-	-
4-bit add/sub	1	-	1	-	-	-
Registers/Flip-Flops	135	57	141	-	-	-
48-bit 16-1 MUX	1	1	-	-	-	-
32-bit XOR2	-	-	1	1	-	-
6-bit XOR2	-	-	8	8	8	-
64*4-bit ROMs	-	-	8	8	-	-

Particulars	S_box	S1_box	S2_BOX	S3_box	S4_box	S5_box	S6_box	S7_box	S8_box
FSMs	-	-	-	-	-	-	-	-	-
4-bit adder	-	-	-	-	-	-	-	-	-
4-bit add/sub	-	-	-	-	-	-	-	-	-
Registers/Flip-Flops	-	-	-	-	-	-	-	-	-
48-bit 16-1 MUX	-	-	-	-	-	-	-	-	-
32-bit XOR2	-	-	-	-	-	-	-	-	-
6-bit XOR2	-	-	-	-	-	-	-	-	-
64*4-bit ROMs	8	1	1	1	1	1	1	1	1

6.8 Timing Summary of DES Algorithm

Particulars	Des_cipher_top	Key_scheduling	Des_top	Block_top	Add_key	E_expansion_function
Timing Constraint (1)						
Source	round_counter_0 (FF)	reset (PAD)	r_in_internal_o (FF)	r_in (PAD)	key (PAD)	x_in (PAD)
Destination	des_out_ready (FF)	k3_47 (FF)	r_in_internal_20 (FF)	r_out (PAD)	x6_out (PAD)	block7_out (PAD)
Delay Time (ns)	4.45	4.194	6.735	10.513	6.209	4.937
Levels of Logic	2	2	6	8	3	2
Logic Time (ns)	2.910	2.47	4.174	2.748	5.194	4.490
Route Time (ns)	1.535	1.717	2.561	7.765	1.015	0.447
Paths/Destination Ports	59/10	113/113	3147/209	1864/64	96/48	48/48
Timing Constraint (2)						
Source	lddata (PAD)	k3_31 (FF)	key_round_in (PAD)	-	-	-
Destination	data_ready_internal (FF)	key_out (PAD)	r_in_internal (FF)	-	-	-
Delay Time (ns)	3.63	10.659	7.321	-	-	-
Levels of Logic	2	6	7	-	-	-
Logic Time (ns)	2.833	7.383	4.801	-	-	-
Route Time (ns)	0.840	3.276	2.520	-	-	-
Paths/Destination Ports	11/11	769/49	1886/249	-	-	-
Timing Constraint (3)						
Source	core_busy (FF)	key_select (PAD)	key_select_0 (FF)	-	-	-
Destination	core_busy (PAD)	key_out (PAD)	key_select (PAD)	-	-	-
Delay Time (ns)	4.31	12.083	4.571	-	-	-
Levels of Logic	1	1	1	-	-	-
Logic Time (ns)	3.68	8.010	3.863	-	-	-
Route Time (ns)	0.447	4.073	0.708	-	-	-
Paths/Destination Ports	2/2	739/48	70/70	-	-	-
CPU Time (s)	17.19/17.52	17.48/17.94	24.47/24.81	18.55/18.91	7.19/7.53	8.83/10.83
Elapsed Time (s)	17.00/17.00	18.00/18.00	24.00/25.00	19.00/19.00	7.00/7.00	9.00/11.00

Memory (kB)	157052	153980	158396	155004	148860	147836

Particulars	**S_box**	**S1_box**	**S2_BOX**	**S3_box**	**S4_box**	**S5_box**	**S6_box**	**S7_box**	**S8_box**
Timing Constraint (1)									
Source	Block7_in (PAD)	A (PAD)	A (PAD)	A (PAD)	A (PAD)	A (PAD)	A (PAD)	A (PAD)	A (PAD)
Destination	X7_out (PAD)	SPO (PAD)	SPO (PAD)	SPO (PAD)	SPO (PAD)	SPO (PAD)	SPO (PAD)	SPO (PAD)	SPO (PAD)
Delay Time (ns)	9.295	8.964	8.947	8.947	8.964	8.964	8.964	8.964	9.295
Levels of Logic	7	6	6	6	6	6	6	6	7
Logic Time (ns)	7.061	6.740	6.740	6.740	6.740	6.740	6.740	6.740	7.061
Route Time (ns)	2.234	2.224	2.207	2.207	2.224	2.224	2.224	2.224	2.254
Paths/Destination Ports	688/82	84/4	84/4	84/4	84/4	84/4	84/4	84/4	100/4
CPU Time (s)	11.36/11.70	7.75/8.09	9.02/9.34	7.98/8.34	7.73/8.08	7.53/7.88	7.58/7.94	8.92/9.28	8.67/9.02
Elapsed Time (s)	11.00/11.00	8.00/8.00	9.00/10.00	8.00/8.00	8.00/8.00	8.00/8.00	7.00/7.00	9.00/10.00	9.00/9.00
Memory (kB)	152756	148860	148860	148860	148860	148860	148860	148860	148860

6.10 Device Utilization Summary of TWOFISH Algorithm

Particulars →	No. of Slices (4656)	No. of 4-i/p LUTs (9312)	Bonded IOBs (232)
s128	394	692	192
reed-solomon128	374	655	192
data_input	0	-	256
data_output	0	-	256
f_128	704	1235	256
g_128	282	495	128
q0	18	32	16
q1	18	32	16
mds	57	102	64
mul_ef	5	9	16
mul_5b	4	8	16
pht	70	123	128
adder	1	2	5
encryption_round128	706	1239	384
decryption_round128	706	1239	384
h_128	240	423	104
keysched128	550	968	208
whit_keysched128	1803	3184	384

Particulars →	No. of Slices (4656)	No. of 4-i/p LUTs (9312)	Bonded IOBs (232)
mul01	0	-	16
mula4	3	6	16
mul55	4	8	16
mul87	4	7	16
mul5a	4	8	16
mul58	5	9	16
muldb	6	10	16
mul9e	5	10	16
mul56	4	8	16
mul82	5	9	16
mulf3	6	11	16
mul1e	6	11	16
mulc6	6	11	16
mul68	6	11	16
mule5	4	7	16
mul02	2	3	16
mula1	5	9	16
mulfc	6	11	16
mulc1	6	10	16
mul47	4	8	16
mulae	5	9	16
mul3d	5	9	16
mul19	5	9	16
mul03	4	8	16

6.11 Macro Statistics of TWOFISH Algorithm

Particulars →	**1-bit xor2**	**1-bit xor3**	1-bit xor4	**8-bit xor8**	**32-bit xor2**	**4-bit xor2**	**4-bit xor3**	**8-bit xor4**	**16*4-bit ROM**
s128	504	128	20	8	-	-	-	-	-
reed-solomon128	504	128	20	8	-	-	-	-	-
data_input	-	-	-	-	-	-	-	-	-
data_output	-	-	-	-	-	-	-	-	-
f_128	400	24	-	-	4	48	48	8	96
g_128	72	12	-	-	2	24	24	4	48
q0	-	-	-	-	-	2	2	-	4
q1	-	-	-	-	-	2	2	-	4
mds	72	12	-	-	-	-	-	4	-
mul_ef	8	3	-	-	-	-	-	-	-
mul_5b	10	-	-	-	-	-	-	-	-
pht	128	-	-	-	-	-	-	-	-
adder	2	-	-	-	-	-	-	-	-
encryption_round128	400	24	-	-	6	48	48	8	96
decryption_round128	400	24	-	-	6	48	48	8	96
h_128	72	12	-	-	2	24	24	4	48
keysched128	272	24	-	-	4	48	48	8	96
whit_keysched128	1088	96	-	-	16	192	192	32	384

Particulars	1-bit xor2	1-bit xor3	1-bit xor4
mul01	-	-	-
mula4	4	2	-
mul55	10	4	-
mul87	5	3	-
mul5a	7	3	-
mul58	9	-	1
muldb	11	4	1
mul9e	8	4	-
mul56	9	1	-
mul82	8	4	-
mulf3	11	3	1
mul1e	8	4	1
mulc6	12	3	1
mul68	11	2	1
mule5	7	2	-
mul02	3	-	-
mula1	10	1	1

mulfc	10	5	-
mulc1	9	4	1
mul47	8	2	-
mulae	8	3	-
mul3d	8	4	-
mul19	9	4	-
mul03	9	-	-

6.9 Design Statistics of TWOFISH Algorithm

Particulars →	IOs	BELS	LUT2	LUT3	LUT4	MUXF5	I/O Buffers	IBUF	OBUF
s128	192	715	119	144	429	23	192	128	64
reed-solomon128	192	673	151	113	391	18	192	128	64
data_input	256	-	-	-	-	-	256	128	128
data_output	256	-	-	-	-	-	256	128	128
f_128	256	1243	258	296	681	8	256	192	64
g_128	128	498	128	47	320	3	128	96	32
q0	16	32	14	2	16	-	16	8	8
q1	16	32	14	2	16	-	16	8	8
mds	64	112	8	34	60	10	64	32	32
mul_ef	16	10	-	4	5	1	16	8	8
mul_5b	16	8	3	4	1	-	16	8	8
pht	128	125	2	85	36	2	128	64	64
adder	5	2	-	2	-	-	5	3	2
encryption_round128	384	1251	257	237	745	12	384	256	128
decryption_round128	384	1251	257	237	745	12	384	256	128
h_128	104	426	91	42	290	3	104	72	32
keysched128	208	976	183	170	615	8	208	144	64
whit_keysched128	384	3227	585	633	1966	43	384	128	256

Particulars	IOs	BELS	LUT2	LUT3	LUT4	MUXF5	I/O Buffers	IBUF	OBUF
mul01	16	-	-	-	-	-	16	8	8
mula4	16	6	3	2	1	-	16	8	8
mul55	16	8	2	1	5	-	16	8	8
mul87	16	8	2	-	5	1	16	8	8
mul5a	16	8	-	4	4	-	16	8	8
mul58	16	10	2	2	5	1	16	8	8
muldb	16	10	-	5	5	-	16	8	8
mul9e	16	12	-	2	8	2	16	8	8
mul56	16	8	3	1	4	-	16	8	8
mul82	16	9	4	1	4	-	16	8	8
mulf3	16	11	3	-	8	-	16	8	8
mul1e	16	11	5	2	4	-	16	8	8
mulc6	16	11	-	3	8	-	16	8	8
mul68	16	12	3	2	6	1	16	8	8
mule5	16	8	1	1	5	1	16	8	8
mul02	16	3	3	-	-	-	16	8	8
mula1	16	9	3	2	4	-	16	8	8
mulfc	16	11	2	5	4	-	16	8	8
mulc1	16	10	1	1	8	-	16	8	8
mul47	16	8	3	4	1	-	16	8	8
mulae	16	10	1	4	4	1	16	8	8
mul3d	16	9	2	5	2	-	16	8	8
mul19	16	10	-	1	8	1	16	8	8
mul03	16	8	5	3	-	-	16	8	8

6.12 Timing Summary of TWOFISH Algorithm

Particulars →	Source	Destination	Delay Time (ns)	Levels of Logic	Logic Time (ns)	Route Time (ns)	Paths/ Destination Ports	CPU Time (s)	Elapsed Time (s)	Memory (kB)
s128	in_key_ts128(PAD)	out_Sfirst_ts128 (PAD)	10.764	6	7.306	3.458	1951/64	20.86 / 21.19	21.00 / 21.00	165564
reed-solomon128	in_rs128 (PAD)	out_Ssecond_rs 128 (PAD)	11.941	7	8.010	3.931	1939/64	19.30/ 19.64	19.00/ 19.00	161468
data_input	in_tdi (PAD)	out_tdi (PAD)	4.910	2	4.490	0.420	128/128	7.37/ 7.70	7.00/ 7.00	146812
data_output	in_tdo (PAD)	out_tdo (PAD)	4.910	2	4.490	0.420	128/128	7.69/ 8.03	8.00/ 8.00	147836
f_128	low_in_f128 (PAD)	up_out_f128 (PAD)	71.244	49	37.19 5	34.04 9	3891031348 68 / 64	93.83 / 94.16	94.00/ 94.00	185020
g_128	in_g128 (PAD)	out_g128 (PAD)	27.455	19	16.07 5	11.38 0	109530836 / 32	37.20/ 37.59	37.00/ 37.00	177468
q0	in_q0 (PAD)	out_q0 (PAD)	10.578	6	7.306	3.272	576/8	8.81/ 9.14	9.00/ 9.00	148860
q1	in_q1 (PAD)	out_q1 (PAD)	10.578	6	7.306	3.272	576/8	9.64/ 9.98	9.00/ 10.00	148860
mds	y2 (PAD)	z1 (PAD)	9.615	5	6.602	3.013	431/32	12.66/ 13.00	12.00/ 13.00	151932
mul_ef	in_ef (PAD)	out_ef (PAD)	7.823	4	5.898	1.925	35/8	7.45/ 7.80	8.00/ 8.00	146812
mul_5b	in_5b (PAD)	out_5b (PAD)	6.422	3	5.194	1.228	22/8	6.98/ 7.41	7.00/ 7.00	147836
pht	down_in_pht (PAD)	up_out_pht (PAD)	49.066	33	25.93 1	23.13 5	13915/64	10.42/ 10.77	10.00/ 10.00	149884
adder	in2_adder	out_carry_adder	6.236	3	5.194	1.042	6/2	6.84/	7.00/	147836

	(PAD)	(PAD)						7.20	7.00	
encryption_round128	in2_ter128 (PAD)	out2_ter128 (PAD)	72.623	50	37.899	34.724	433421211436 / 128	96.28/ 96.64	96.00/ 97.00	187068
decryption_round128	in2_tdr128 (PAD)	out2_tdr128 (PAD)	72.623	50	37.899	34.724	433421211436 / 128	102.64/ 102.98	103.00/ 103.00	187068
h_128	in_h128 (PAD)	out_h128 (PAD)	27.805	19	16.075	11.730	109530836 / 32	34.09/ 34.42	34.00/ 35.00	175804
keysched128	even_in_tk128 (PAD)	out_key_down_tk128 (PAD)	69.899	48	36.491	33.408	4583479222 4 / 64	85.20/ 85.55	85.00/ 85.00	192508
whit_keysched128	in_key_twk128 (PAD)	out_K7_twk128 (PAD)	63.544	43	32.971	30.573	2601538728 / 256	158.47/ 158.81	158.00/ 159.00	202812

Particulars →	Source	Destination	Delay Time (ns)	Levels of Logic	Logic Time (ns)	Route Time (ns)	Paths/ Destination Ports	CPU Time (s)	Elapsed Time (s)	Memory (kB)
mul01	in_mul01 (PAD)	out_mul01 (PAD)	4.910	2	4.490	0.420	8/8	6.59/6.92	7.00/7.00	147836
mula4	in_mula4 (PAD)	out_mula4 (PAD)	6.320	3	5.194	1.126	18/8	7.09/7.44	7.00/7.00	146812
mul55	in_mul55 (PAD)	out_mul55 (PAD)	7.481	4	5.898	1.583	33/8	8.20/8.55	8.00/9.00	148860
mul87	in_mul87 (PAD)	out_mul87 (PAD)	7.391	4	5.898	1.493	28/8	9.30/9.84	10.00/10.00	147836
mul5a	in_mul5a (PAD)	out_mul5a (PAD)	7.506	4	5.898	1.608	30/8	7.25/7.59	7.00/7.00	147836
mul58	in_mul58 (PAD)	out_mul58 (PAD)	6.697	4	5.515	1.182	31/8	7.41/7.73	7.00/7.00	147836
muldb	in_muldb (PAD)	out_muldb (PAD)	7.608	4	5.898	1.710	43/8	7.45/7.78	8.00/8.00	147836
mul9e	in_mul9e (PAD)	out_mul9e (PAD)	7.608	4	5.898	1.710	42/8	7.53/7.86	7.00/8.00	146812
mul56	in_mul56 (PAD)	out_mul56 (PAD)	7.475	4	5.898	1.577	28/8	7.25/7.59	7.00/7.00	147836
mul82	in_mul82 (PAD)	out_mul82 (PAD)	7.481	4	5.898	1.583	32/8	7.44/7.78	7.00/8.00	148860
mulf3	in_mulf3 (PAD)	out_mulf3 (PAD)	7.608	4	5.898	1.710	40/8	7.59/7.92	8.00/8.00	147836
mul1e	in_mul1e (PAD)	out_mul1e (PAD)	7.562	4	5.898	1.664	33/8	7.58/8.00	8.00/8.00	147836
mulc6	in_mulc6 (PAD)	out_mulc6 (PAD)	7.586	4	5.898	1.688	41/8	7.51/7.86	7.00/8.00	147836
mul68	in_mul68	out_mul68	7.992	5	6.219	1.773	39/8	7.45/7.80	8.00/8.00	147836

	(PAD)	(PAD)								
mule5	in_mule5 (PAD)	out_mule5 (PAD)	7.531	4	5.898	1.633	29/8	7.34/7.70	8.00/8.00	147836
mul02	in_mul02 (PAD)	out_mul02 (PAD)	6.280	3	5.194	1.086	11/8	6.86/7.22	7.00/7.00	147836
mula1	in_mula1 (PAD)	out_mula1 (PAD)	7.646	4	5.898	1.748	32/8	7.39/7.75	7.00/8.00	147836
mulfc	in_mulfc (PAD)	out_mulfc (PAD)	7.606	4	5.898	1.708	37/8	8.28/8.63	8.00/9.00	148860
mulc1	in_mulc1 (PAD)	out_mulc1 (PAD)	7.577	4	5.898	1.679	40/8	8.55/8.89	9.00/9.00	146812
mul47	in_mul47 (PAD)	out_mul47 (PAD)	6.376	3	5.194	1.182	22/8	7.44/7.77	8.00/8.00	147836
mulae	in_mulae (PAD)	out_mulae (PAD)	6.779	4	5.515	1.264	31/8	7.45/7.80	8.00/8.00	147836
mul3d	in_mul3d (PAD)	out_mul3d (PAD)	7.531	4	5.898	1.633	31/8	7.64/8.01	7.00/8.00	148860
mul19	in_mul19 (PAD)	out_mul19 (PAD)	6.818	4	5.515	1.303	36/8	7.53/7.89	8.00/8.00	147836
mul03	in_mul03 (PAD)	out_mul03 (PAD)	6.326	3	5.194	1.132	19/8	7.34/7.80	7.00/8.00	147836

CONCLUSION

The information security can be easily achieved by using Cryptography technique. A large number of encryption algorithms have been developed to secure our confidential data from the hackers. But some algorithms have been broken by using Cryptanalysis method. A key is the strongest point of any algorithm but it can become the weakest point if it is not secured. Our information can be secured if it is encrypted by using multiple keys or a large bit stream of key (i.e., 128 – bit, 256 – bit, etc.). But to achieve this a large computational time is required, giving a large delay which can be harmful to us. The hacker can hack the information during this time. The use of FPGAs can help us to improve this limitation because FPGAs can give enhanced speed. This is due to fact that the hardware implementation of most encryption algorithms can be done on FPGA.

In this thesis, DES encryption algorithm is implemented on Xilinx Spartan – 3E (XC3S500) FPGA kit. The Twofish encryption algorithm is analysed by using Xilinx ISE 9.2i tool.

REFERENCES

[1] D. Kahn: The Codebreakers: the story of secret writing, MacMillan publishing, 1996.

[2] W. Diffie and M. Hellman, "New Directions in Cryptography", IEEE Transaction on Information Theory, Vol. IT-22**,** Nov. 1976, pp. 644-654**.**

[3] Ruth M. Davis, "The Data Encryption Standard", Proceedings of Conference on Computer Security and the Data Encryption Standard, National Bureau of Standards, Gaithersburg, MD, Feb. 15, 1977, NBS Special Publication 500-27, pp 5-9.

[4] Whitfleld Diffie, "Cryptographic Technology: Fifteen Year Forecast" Reprinted by permission AAAS, 1981 from Secure Communications and Asymmetric Crypto Systems. AAAS Selecte8 Symposia. Editor: C.J. Simmons. Vol. 69, Westview Press, Boulder, Colorado, pp 38-57.

[5] Ingrid Verbauwhede, "Security and Performance Optimization of a New DES Data Encryption Chip", IEEE journal of Solid-State Circuits, Vol. 23, No. 3. June 1988, pp 647-656.

[6] James E. Katz, "Social Aspects of Telecommunications Security Policy", IEEE Technology and Society Magazine, June/July 1990, pp 16-24.

[7] H. Bonnenbergt, "VLSI Implementation of a New Block Cipher", IEEE 1991, pp 510-513.

[8] K.H. Mundt, "SUPERCRYPT, ASIC Technology facilitates a new Device Family for Data Encryption", IEEE 1992, pp 356-359.

[9] C. Boyd. "Modern Data Encryption," Electronics & Communication Engineering Journal, October 1993, Vol. 5, pp 271-278

[10] A. Curiger, "VINCI: VLSI Implementation of the New secret-key block Cipher IDEA", IEEE 1993 Custom Integrated Circuits Conference, pp 1-4.

[11] R. Zimmermann, "A 177 Mb/s VLSI Implementation of the International Data Encryption Algorithm", IEEE Journal of Solid-State Circuits. Vol. 29, No. 3, March 1994, pp 303-307.

[12] Stefan Wolter, "On the VLSI Implementation of the International Data encryption Algorithm IDEA", IEEE 1995, pp 397-400.

[13] Seung-Jo Han, "The Improved Data Encryption Standard (DES) Algorithm" IEEE 1996, Vol. 3, pp 1310-1314.

[14] Toby Schaffer, "A Flip-Chip Implementation of the Data Encryption Staiidard (DES)", IEEE 1997, pp 13-17.

[15] Suan-Suan Chew, "IAuth: An Authentication System for Internet Applications", Computer Software and Applications Conference, 1997 COMPSAC '97 Proceedings., The Twenty-First Annual International**,** IEEE 1997, pp 654-659.

[16] Hassina Guendouz, “Rapid Prototype of a Fast Data Encryption Standard with Integrity Processing for Cryptographic Applications”, IEEE 1998, pp 434-437.

[17] K. Wong, “A Single-Chip FPGA Implementation of the Data Encryption Standard (DES) Algorithm” Global Telecommunications Conference, 1998. GLOBECOM 98, IEEE, Vol. 2, pp 827-832

[18] Yeong-Kang Lai, “A Novel VLSI Architecture for a Variable-Length Key, 64-Bit Blowfish Block Cipher”, Signal Processing Systems, 1999 IEEE Workshop, pp 568-577.

[19] M.P. Leong, “A Bit-Serial Implementation of the International Data Encryption Algorithm IDEA”, 2000 IEEE Symposium on Field-Programmable Custom Computing Machines, pp 122-131.

[20] R. G. Sixel, “A High Level Language Implementation of the Data Encryption Standard and a Bit-Slice Architecture”, Roc 43rd IEEE Midwest Symp on Crcuits and Systems, Lansing MI, Aug 8-11, 2000, pp 266-269.

[21] Teo Pock Chueng, “Implementation of Pipelined Data Encryption Standard (DES) Using Altera CPLD”, TENCON 2000 Proceedings, Vol. 3, IEEE 2000, pp 17-21.

[22] Cameron Patterson,“High Performance DES Encryption in Virtex FPGAs using Jbits”, IEEE 2000, pp 113-121.

[23] Pui-Lam Siu, “A Low Power Asynchronous DES”, Circuits and Systems, ISCAS 2001 IEEE International Symposium, Vol. 4, pp 538-541.

[24] N. Sklavos, “Asynchronous Low Power VLSI Implementation of the International Data Encryption Algorithm”, Electronics Circuits and Systems ICECS 2001, 8th IEEE International Conference Vol. 3, pp 1425-1428.

[25] Ahmet Eskicioglu, “Cryptography”, IEEE Potentials 2001, pp 36-38.

[26] Deng Liang, “An Efficient and Scalable VLSI Implementation of DES”, ASIC 2001 Proceedings 4th International Conference IEEE 2001, pp 341-343.

[27] Yeong-Kang Lai, “VLSI Architecture Design and Implementation for Twofish Block Cipher”, IEEE 2002, pp 356-359.

[28] Touria ARICH, “Hardware implementations of the Data Encryption Standard”, IEEE 2002, pp 100-103.

[29] Chih-Chung Lu, “Integrated Design of AES (Advanced Encryption Standard) Encrypter and Decrypter”, Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP’02).

[30] G. Catalini, “Modified Twofish Algorithm for increasing Security and Efficiency in the Encryption of Video signals”, IEEE 2003, pp 525-528.

[31] M. McLoone, “High-performance FPGA implementation of DES using a novel method for implementing the key schedule”, IEE Proc.-Circuits Devices Syst., Vol. 150, No. 5, October 2003, pp 373-378.

[32] Bo Yang, “Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard”, ITC International Test Conference IEEE 2004, pp 339-344.

[33] Liakot Ali, “Implementation of Triple Data Encryption Algorithm using VHDL”, ICSE2004, Proc. 2004, Kuala Lumpur, Malaysia, pp 369-373.

[34] Tariq Jamil, “The Rijndael Algorithm: A brief introduction to the new encryption standard”, IEEE Potentials 2004, pp 36-38.

[35] Aamer Nadeem, “A Performance Comparison of Data Encryption Algorithms”, IEEE 2005, pp 84-89.

[36] A.Ammar, “Random Data Encryption Algorithm (RDEA)”, Twenty Second National Radio Science Conference (NRSC 2005), Cairo-Egypt, pp 1-8.

[37] Jingmei Liu, “An AES S-box to Increase Complexity and Cryptographic Analysis”, Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA’05) Vol. 1, pp.724-728.

[38] Chih-Hsu Yen, “Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard”, IEEE Transactions on Computers, Vol. 55, No. 6, June 2006, pp 720-731.

[39] Alireza Hodjat, “Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/s AES Processors”, IEEE Transactions on Computers, Vol. 55, No. 4, April 2006, pp 366-372.

[40] A.Chandra Sekhar, “Data Encryption technique using Random number generator”, 2007 IEEE International Conference on Granular Computing, pp 576-579.

[41] M.R.M. Rizk, “Optimized Area and Optimized Speed Hardware Implementations of AES on FPGA”, International Design and Test Workshop, 2007 2nd**,** IEEE 2007, pp 207-217.

[42] Jing Wang, “Improved DES Algorithm based on Irrational Numbers”, IEEE Int. Conference Neural Networks & Signal Processing Zhenjiang, China, June 8~10, 2008, pp 632-635.

[43] Md. Nazrul Islam, “Effect of Security Increment to Symmetric Data Encryption through AES Methodology”, Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing IEEE 2008, pp 291-294.

[44] Sapiee Haji Jamel, Mustafa Mat Deris, “Diffusive primitives in the Design of Modern Cryptographic Algorithms”, Proceedings of the International Conference on Computer and Communication Engineering 2008 May 13-15, 2008 Kuala Lumpur, Malaysia, pp 707-710.

[45] Hung-Min Sun, “On the Security of an Efficient Time-Bound Hierarchical Key Management Scheme”, IEEE transactions on dependable and secure computing, Vol. 6, No. 2, April-June 2009, pp 159-160.

[46] Tingyuan Nie, “A Study of DES and Blowfish Encryption Algorithm”, IEEE 2009, pp 1-4.

[47] Ashwini M. Deshpande, “FPGA Implementation of AES Encryption and Decryption”, International Conference on “Control, Automation, Communication and Energy Conservation - 2009, 4th-6th June 2009, pp 1-6.

[48] agents.csie.ntu.edu.tw/~yjhsu/courses/u2010/2004/040317.pdf

[49] Nazar A. Saqib, “A Compact and Efficient FPGA Implementation of the DES Algorithm”, islab.oregonstate.edu/papers/FRH/des (8).pdf

[50] CISSP All-in-One Certification Exam Guide, by Shon Harris

[51] William M. Daley, “Data Encryption Standard (DES)”, Federal Information Processing Standards Publication Reaffirmed 1999 October 25, pp 1-22.

[52] Ultra-Compact Data Encryption Standard Core, IP cores @ www.ipcores.com

[53] http://www.kremlinencrypt.com/algorithms.htm

[54] http://orlingrabbe.com/des.htm

[55] www.schneier.com/paper-twofish-fpga.pdf

[56] www.schneier.com/paper-twofish-paper.pdf