University of Cincinnati

Date: 6/18/2014

I, Lathika SriDatha Namburi, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Engineering.

It is entitled:
An Efficient VHDL Description and Hardware Implementation of the Triple DES Algorithm

Student’s name: Lathika SriDatha Namburi

This work and its defense approved by:

Committee chair: Carla Purdy, Ph.D.
Committee member: Raj Bhatnagar, Ph.D.
Committee member: George Purdy, Ph.D.
I, Lathika SriDatha Namburi, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Engineering.

It is entitled:
An Efficient VHDL Description and Hardware Implementation of the Triple DES Algorithm

Student’s name: Lathika SriDatha Namburi

This work and its defense approved by:

Committee chair: Carla Purdy, Ph.D.
Committee member: Raj Bhatnagar, Ph.D.
Committee member: George Purdy, Ph.D.
An Efficient VHDL Description and Hardware Implementation of the Triple DES Algorithm

A thesis submitted to the

Graduate School of the University of Cincinnati

In partial fulfillment of the requirements for the degree of

Master of Science

In the Department of Electrical and Computer Engineering

Of the College of Engineering and Applied Sciences

June 2014

By

Lathika SriDatha Namburi

B.Tech, Electronics and Communications Engineering,

Jawaharlal Nehru Technological University, Hyderabad, India, 2011

Thesis Advisor and Committee Chair: Dr. Carla Purdy
ABSTRACT

Data transfer is becoming more and more essential these days with applications ranging from everyday social networking to important banking transactions. The data that is being sent or received shouldn’t be in its original form but must be coded to avoid the risk of eavesdropping. A number of algorithms to encrypt and decrypt the data are available depending on the level of security to be achieved. Many of these algorithms require special hardware which makes them expensive for applications which require a low to medium level of data security. FPGAs are a cost effective way to implement such algorithms.

We briefly survey several encryption/decryption algorithms and then focus on one of these, the Triple DES. This algorithm is currently used in the electronic payment industry as well as in applications such as Microsoft OneNote, Microsoft Outlook and Microsoft system center configuration manager to password protect user content and data. We implement the algorithm in a Hardware Description Language, specifically VHDL and deploy it on an Altera DE1 board which uses a NIOS II soft core processor. The algorithm takes input encoded using a software based Huffman encoding to reduce its redundancy and compress the data. We analyze the results obtained from the implementation and discuss some methods to minimize attacks on the algorithm. Our VHDL implementation can also be ported to other hardware platforms.
Table of Contents

1. INTRODUCTION .......................................................................................................................... 1
   1.1 Motivation ............................................................................................................................. 1
   1.2 Thesis Goals ......................................................................................................................... 2
   1.3 Outline ................................................................................................................................. 2

2. BACKGROUND ............................................................................................................................ 3
   2.1 Cryptography ....................................................................................................................... 3
   2.2 Data Encryption Standard .................................................................................................... 4
   2.3 The Algorithm ..................................................................................................................... 5
   2.4 Feistel Function ................................................................................................................... 8
   2.5 Key Schedule ..................................................................................................................... 9
   2.6 Attacks ............................................................................................................................... 12
   2.7 Replacements .................................................................................................................... 14

3. OUTLINE OF RESEARCH ......................................................................................................... 16
   3.1 Triple DES ............................................................................................................................ 16
   3.2 Modes of Operation ............................................................................................................. 17
   3.3 Security .................................................................................................................................. 19
   3.4 HDL Implementation .......................................................................................................... 21
   3.5 Huffman Encoding ............................................................................................................ 23

4. RESULTS .................................................................................................................................... 27
   4.1 NIOS II Processor ............................................................................................................... 30
4.2 Implementation..................................................................................................................31
4.3 Simulation..........................................................................................................................32
4.4 Analysis..............................................................................................................................34
4.5 Summary............................................................................................................................37

5. CONCLUSIONS AND FUTURE WORK............................................................................38
  5.1 Conclusions......................................................................................................................38
  5.2 Future Work......................................................................................................................38

BIBLIOGRAPHY......................................................................................................................40

APPENDIX – 1: DETAILED DESCRIPTION OF VARIOUS MODULES OF DES.....................45

APPENDIX – 2: ALTERA QUARTUS II TUTORIAL..................................................................53
List of Figures

Figure 2.1: Encryption process of DES..............................................................7
Figure 2.2: Fiestel Function..............................................................................8
Figure 2.3: Key Scheduling............................................................................10
Figure 3.1: Encryption in ECB mode..............................................................18
Figure 3.2: Decryption in ECB mode..............................................................19
Figure 3.3: State Machine for the HDL implementation of DES..................21
Figure 3.4: Memory Initialization in the form of arrays..............................22
Figure 3.5: Binary tree generated through Huffman encoding....................25
Figure 4.1: Simulation results with the device EP2C2OF484C7....................32
Figure 4.2: RTL schematic of the system.....................................................33
Figure 4.3: Utilization of the logic circuitry and pins on the device...............34
Figure 4.4: Power estimation for the resulting implementation..................36
Figure 5.1: Sending the tree through RSA....................................................39
Figure A: DES Encryption..........................................................................54
Figure B: DES Decryption..........................................................................56
Figure C: TDES Encryption.......................................................................58
Figure D: TDES Decryption.......................................................................58
Figure E: Encryption of Example 1 (part1)................................................60
Figure F: Decryption of Example 1 (part1)................................................60
Figure G: Encryption of Example 1 (part2)................................................61
Figure H: Decryption of Example 1 (part2)………………………………………………………………………………61

Figure I: Encryption of Example 1 (part3)………………………………………………………………………………..62

Figure J: Decryption of Example 1 (part3)………………………………………………………………………………62

Figure K: Device Selection………………………………………………………………………………………………63

Figure L: Processor Specification…………………………………………………………………………………………65

Figure M: Finished NIOS II system……………………………………………………………………………………66

Figure N: Successful Compilation………………………………………………………………………………………..68
List of Tables

Table 2.1 : Number of shifts per round in Key scheduling..........................................................11
Table 3.1 : Symbols and corresponding codes..................................................................................25
Table 4.1 : Standard American letters and frequencies.................................................................29
Table 4.2 : Comparison with the previous implementation............................................................35
1. INTRODUCTION

1.1 Motivation

The Data Encryption Standard (DES) is a cryptographic algorithm which was extensively used in the encryption of data and laid the foundation for the advancement of modern cryptography in academia [1]. The algorithm is designed to encipher and decipher blocks of data, each 64 bit long with the help of a 64 bit key [2]. Of the 64 bits in the key, 8 bits are for the purpose of checking parity and provide no cryptographic importance bringing the effective key length down to 56 bits [2].

Due to the short key length, the algorithm is vulnerable to attacks theoretically, but the very first practical attack was demonstrated in June 1997 during the DESCHALL project [3] where a message encrypted with DES was cracked in public. In April of 2006, a machine called COPACOBANA [4] which consisted of 120 XILINX Spartan-3 1000 [5] FPGAs running in parallel and grouped in 20 DIMM modules each with 6 FPGAs [6] successfully cracked the DES in 9 days whereas the former took 96 days [7]. In 2008, its successor, the RIVYERA did the job in less than a day [7].

To increase the security further and to allow backward compatibility, advanced versions of the DES such as 2DES, 3DES and DESX were proposed. The 3DES or Triple DES with an increased effective key length became more successful and was standardized in 1999 [8]. Through the year 2030, Triple DES and the FIPS 197 Advanced Encryption Standard (AES) will coexist as approved algorithms thus, allowing for a gradual transition to AES (The AES is another symmetric-based encryption standard approved by NIST) [8].
The Triple DES cipher is currently used by the electronic payment industry and standards have been developed and propagated using the algorithm as the basis [9]. Other applications include Microsoft OneNote [10], Microsoft Outlook 2007 [11] and Microsoft System Center Configuration Manager 2012 [12] which protects user content with the help of Triple DES.

1.2 Thesis Goals

The aim of this thesis is to study Triple DES, to implement it efficiently on a FPGA equipped with an Altera Nios II processor [13] and to obtain the results using Altera Quartus II [14]. The message to be encoded is first compressed using a software based Huffman encoding algorithm to reduce redundancy.

1.3 Outline

The organization of the thesis is as follows

Chapter 2 deals with the background of Triple DES cipher and the relationship between DES and AES. It also provides information on the cryptographic performance of the DES block cipher and its cryptanalysis.

Chapter 3 deals with the detailed description of Triple DES, its internal components, modes of operation and its cryptanalysis.

Chapter 4 deals with Huffman encoding in software and the implementation of Triple DES on a Field Programmable Gate Array, specifically, one with an Altera NIOS II processor and analysis of the results obtained.

Chapter 5 gives the conclusion and the possibility of future work.
2. BACKGROUND

This chapter deals with the advent of cryptography into modern computing and the background of Triple DES. It also discusses the relation between DES and AES as well as the cryptographic performance of DES.

2.1 Cryptography

The term cryptography refers to the scientific study of techniques for securing digital information, transactions and distributed computations [15]. Plaintext refers to the data that can be read and understood without any special measures. Encryption is the process of converting a plaintext into an unreadable form called ciphertext. Decryption is the reverse process of encryption, converting ciphertext to plaintext [16].

A cryptographic algorithm also known as a cipher is a mathematical function used in the process of encryption and decryption and it works in combination with a key which can be a word, number or phrase to encrypt the plaintext [16]. Different ciphertexts can be formed with the same plaintext by employing different keys. The strength of the cipher and the secrecy of the key determine the security of the plaintext [16].

Depending on the keys used for encryption and decryption, cryptography is broadly classified into symmetric key and public key cryptography. If the same key is used for encryption as well as decryption, then it is symmetric key cryptography. Data Encryption Standard, its variations and Advanced Encryption standard fall under the category of symmetric key cryptography. In public key cryptography, two different keys called public and private key are used for encryption and decryption respectively [17].
In public key cryptographic systems, the two keys are mathematically related but calculating one key from the other is almost impossible. Diffie–Hellman algorithm and RSA [18] algorithm are examples of public key cryptographic algorithms. Of all these algorithms, both DES and AES have been standardized by the US government [19] which made them quite popular in public usage as well as in academia.

Initially, cryptography was mainly employed by military and intelligence organizations, but now it is everywhere. Security mechanisms that use cryptography are a part of any modern computing system [15]. Every time a secure website is accessed, users rely on cryptography with or without their knowledge. To protect valuable information in multi-user operating systems and also in personal laptops, cryptographic methods are used [15]. Electronic payment industry, user account protection and the list is endless. In short, cryptography has transformed itself from an art form that dealt with secret communication to a science which helps to secure systems for ordinary people all around the world [15].

2.2 Data Encryption Standard

The history of the DES dates back to the early 70s. In 1972, NBS (National Bureau of Standards), the then NIST (National Institute of Standards and Technology) decided that they need a standard for encrypting unclassified, sensitive information after surveying on the computer security needs of the US government [20]. IBM Corp., which developed a family of cryptographic algorithms for financial applications, submitted an algorithm in 1974 in response to a solicitation issued by NBS in the federal register [21].
This algorithm was published in 1975 after significant review within the government which requested comments on its technical aspects. In 1977, after in depth analysis of the comments as well as recommendations from workshops organized earlier, the DES was issued as a Federal Information Processing Standard 46 [22]. Although it received significant criticism, it was reaffirmed as a standard in 1983, 1988 (FIPS 46 – 1), 1993 (FIPS 46 – 2) and 1999 (FIPS 46 – 3) [1].

After its publication, DES was adopted by ANSI (American National Standards Institute) as a standard in 1981 [23] and since has become the most widely used encryption algorithm especially in the world of finance [21]. The algorithm was so extensively researched that Bruce Schneier, a famous cryptographer once said "DES did more to galvanize the field of cryptanalysis than anything else. Now there was an algorithm to study." [21].

2.3 The Algorithm

DES is a symmetric keyed block cipher. It employs a 64 bit key to encrypt a data block usually of 64 bits and uses the same key to decrypt the cipher block to retrieve the data block. The effective key length is reduced to 56 bits as 8 of the 64 bits are solely used for parity – “One bit in each 8-bit byte of the key may be utilized for error detection in key generation, distribution, and storage. Bits 8, 16, 24, 32, 40, 48, 56 and 64 are for use in ensuring that each byte is of odd parity” [24].

Before beginning the process of encryption, the plaintext is divided into 2 32 bit blocks and each block is processed separately thereafter as shown in figure 2.1. A series of operations which include various substitution and permutation primitives are used which were defined by
Horst Feistel that are applied to the 2 data halves iteratively for a specified number of times [25] [26]. Each iteration is termed as a round and DES contains 16 rounds to achieve adequate data scrambling and desired security [21].

This Feistel scheme serves an important purpose. With the exception of the application of the subkeys, it makes sure that the decryption is the same as encryption reducing the complexity of implementation especially in hardware because it is enough to implement the same algorithm to encrypt as well as decrypt the given data.

The Final Permutation is the inverse process of Initial permutation and though serving no cryptographic importance, they are included to load and unload data blocks onto the old existing hardware [27]. The Initial and Final Permutations are explained in detail in Appendix - 1. After 16 rounds and before the final permutation, the left (L) and the right (R) blocks are governed by the following expressions.

\[
L_{16} = R_{15}
\]

\[
R_{16} = L_{15} \oplus F(R_{15}, K_{16}) [2]
\]

Where, \(K_{16}\) denotes the 16th subkey generated through key scheduling.
Figure 2.1: Encryption process of DES
As shown in Figure 2.2, the Feistel (F) function consists of Expansion (E), Substitution (S) and Permutation (P) stages.

The 32 bit data block which is half of the original plaintext serves as input to the Expansion function. As the name indicates, it expands the 32 bits to 48 bits according to a fixed permutation. The resulting 48 bit output and a 48 bit subkey are mixed through an XOR operation which results in a 48 bit block that acts as an input to Substitution function. There are
8 substitution boxes each taking a 6 bit signal as input and replacing it with a 4 bit output according to a non–linear transformation which is provided in the form of a lookup table [1].

The core security for the algorithm is provided by the Substitution boxes which makes the cipher non-linear and rendering any attack ineffective [22]. The outputs of all the 8 boxes are then collected and rearranged according to a Permutation (P). This output is then xored with the other half data block and the process continues for 16 rounds. The lookup tables for the Expansion, substitution and Permutation boxes are provided and are explained in detail in Appendix -1.

2.5 Key Schedule

The key scheduling algorithm mainly consists of 2 functions- Permutation Choice (PC) 1 and Permutation Choice (PC) 2. Permutation Choice 1 deals with filtering the effective 56 bit key from the original 64 bit key which are thereafter divided into 2 equal parts and processed separately [2].

The two halves, each of 24 bit length are left shifted as shown in figure 2.3 and the resulting blocks are given as input to Permutation Choice 2 which then selects 24 bits from each half and rearranges them and outputs a subkey that is 48 bit long [1].

This process continues for 16 rounds and 16 subkeys are generated which are used in their respective rounds during encryption. The key schedule for decryption is similar to that of encryption with the only difference being the reverse ordering of the subkeys. The number of shifts vary for each iteration and are specified in Table – 1.
Figure 2.3: Key Schedule

Figure 2.3: Key Schedule

KEY (64 BITS)

<<<

PERMUTED CHOICE 1 (PC1)

<<<

PERMUTED CHOICE 2 (PC2)

<<<

SUBKEY 1 (16 BITS)

<<<

PERMUTED CHOICE 2 (PC2)

for 16 rounds

<<<

PERMUTED CHOICE 2 (PC2)

<<<

SUBKEY 2 (16 BITS)

<<<

PERMUTED CHOICE 2 (PC2)

<<<

SUBKEY 15 (16 BITS)

<<<

PERMUTED CHOICE 2 (PC2)

<<<

SUBKEY 16 (16 BITS)
<table>
<thead>
<tr>
<th>Iteration or Round Number</th>
<th>Number of left shifts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2. 1 : Number of Shifts per round in Key scheduling
2.6 Attacks

Due to the 56 bit key length, DES is susceptible to different kinds of attacks, the most common being the brute force attack. Brute force attack is the process of applying each and every possible key combination until we hit the correct one [28]. Attacks on DES were completely theoretical for 20 years after its standardization.

A proposition was made in 1977 by Diffie and Hellman for a machine that could find a key for DES in 1 day which costs 20 Million Dollars. Then in 1993, another proposition by Wiener for a key search machine brought down the price to 1 Million Dollars which can find a key within 7 hours [29]. These two were completely theoretical and because the hardware realization and the cost were impractical, they were ignored.

Differential Cryptanalysis [30], rediscovered by Eli Biham and Adi Shamir in the 1980s could crack the DES and its complexity is much less compared to a brute force attack. $2^{49}$ chosen plaintexts are required to perform the differential cryptanalysis and as the name indicates, chosen plaintexts are those data blocks which are chosen by the user deliberately to gain more information regarding the algorithm used for encryption [31].

Linear Cryptanalysis [32], discovered by Mitsuru Matsui is a form of cryptanalysis that finds affine [33] approximations to the action of a cipher. It needs $2^{43}$ known plaintexts to crack the DES [32]. Known plaintexts are data blocks whose plaintext and ciphertext combinations are known to the user and are used in the process of cryptanalysis.

A combination of both, Differential – Linear Cryptanalysis, proposed by Hellman and Langford in 1994 could recover 10 bits of the key corresponding to a DES algorithm comprising of 8 rounds
with just 512 chosen plaintexts and with a success rate of 80% [34]. All the attacks proposed above are completely theoretical due to the immense computational complexity.

In 1997, a group called DES Challenge (DESCHALL) became the first to publicly crack a message with DES encryption in a contest sponsored by RSA Security [35]. Matt Curtin, who was a member of the DESCHALL team later wrote a book entitled “Brute Force: Cracking the Data Encryption Standard” in which he describes the procedure they followed in achieving the success.

To implement an attack, a search among $2^{56}$ keys is needed which is impossible to perform on a single computer. Rocke Verser, the team lead for the DESCHALL project came up with the idea of utilizing the free idle cycles of the huge number of computers connected through internet [3]. A server setup on a 486 based PS2 PC [36] with 56 MB memory and the project was announced through Usenet [37] and software was quickly written for a wide variety of home machines and later included powerful 64 bit systems [3].

A single computer equipped with a Pentium 200 MHz processor could test 1 million keys each second approximately if idle with the software written [3]. Even at this rate, it takes a lot of time to search all the possible key combinations. So, there was a rapid rise in the number of computers involved, reaching 78,000 in the end. It took 96 days to find the correct key but only a quarter of the total key space has been searched and a rate of about 7 billion keys per second [3].

In 1998, the Electronic Frontier Foundation (EFF), a non-profit digital rights group built a custom DES cracker with the aim of performing a brute force search on the key space of DES [38]. It
comprised of 1856 custom chips and was worth 250,000 Dollars and was able to decrypt a message encrypted through DES in 56 hours [38]. This was a warning sign for both DES and NIST, the former trying to improve the algorithm and the latter, searching for a new standard.

In 2006, teams from the University of Bochum and University of Kiel built a machine called COPACOBANA (Code Optimized Parallel Code Breaker) [4]. The main difference between the machine built by the EFF and COPACOBANA is the latter had Field Programmable Gate Arrays (FPGA) at its core while the former had chips custom made for the process [6] [38].

Each machine is equipped with 120 Xilinx Spartan3-1000 [5] FPGAs which are grouped in 20 Dual Inline Memory Modules (DIMM) and fits into a 19” rack of 3 height units [6]. It costs about 10,000 Dollars [39] and had the capacity to perform an exhaustive search of DES’ key in less than 9 days [6]. Its later versions were able to achieve a throughput of up to 65 billion keys per second [7]. The decrease in the cost was mainly attributed to improvements in electronic hardware and the use of general purpose processors as its building blocks.

In 2009, the successor of COPACOBANA 5000, the RIVYERA, developed by the company SciEngines GmbH [40], using 128 Xilinx Spartan-3 5000 [5] FPGAs achieved a throughput of over 280 billion keys per second broke DES in less than a single day [7]. The advanced versions of RIVYERA achieved a throughput of 292 billion keys per second reducing the cracking time further [40].

2.7 Replacements

As the vulnerability of the DES was demonstrated publicly by the DESCHALL project, National Institute of Standards and Technology (NIST) made an announcement for a successor for DES
There was an international competition in which the finalists were RC6, Serpent, MARS, Twofish and Rijndael. Among these, the Rijndael algorithm won with 86 positive and 10 negative votes and also because it had the best combination of security, efficiency, implementability, performance, and flexibility.

The Rijndael cipher, popularly known as the Advanced Encryption Standard (AES) was announced in 2001. The AES algorithm based on the Rijndael cipher accepts a 128 bit data block and gives a choice of 3 key lengths, 128 bits, 192 bits and 256 bits. The algorithm with 128 bit key in CCM (“Counter Mode Cipher Block Chaining Message Authentication Code Protocol”) is used by IEEE 802.11i for Wireless LAN products.

This did not mark the end of the DES. The variants of the DES such as Triple DES, DES-X and GDES were proposed with the hope of succeeding the DES, but due to the increased security provided by the Rijndael cipher, the former were not considered. The Triple DES variant of the DES, which provides 3 times the security as that of DES when all the 3 keys are independent is considered to be appropriate through 2030 by NIST.
3. OUTLINE OF RESEARCH

This chapter deals with the description of Triple DES, its modes of operation, its cryptanalysis and Huffman Coding in detail. It also deals with the HDL implementation of different components of the TDES algorithm.

3.1 Triple DES

As its predecessor, the Triple Data Encryption Standard (Triple DES) is also a symmetric key block cipher [8]. It applies the DES algorithm thrice to each block of data. Hence, there are 3 different keys to perform each application of DES. Therefore, it raises the key length to 168 bits (excluding the 24 parity bits, 8 from each key) and the keys are collectively known as the key bundle [8].

The basic function of Triple DES is encrypting the data block with the first key, decrypting the result with the second key and finally, encrypting the obtained block with the third key. If we assume that k1, k2 and k3 are the 3 keys and C to be the ciphertext and P to be the plaintext and F to denote encryption and f to denote decryption, then the encryption process of the Triple DES can be represented by

\[ C = F_{k3} [f_{k2} (F_{k1} (P))] \]

In the same way, the decryption process of Triple DES can be represented by

\[ P = f_{k1} [F_{k2} (f_{k3} (C))] \]

Depending on the uniqueness of the keys k1, k2 and k3, we have 2 keying options available for the Triple DES. They are
• Keying Option 1: All the 3 keys are unique, i.e., \( k_1 \neq k_2 \neq k_3 \).

• Keying Option 2: Only \( k_1 \) and \( k_2 \) are unique and \( k_1 \) and \( k_3 \) are equal, i.e., \( k_1 \neq k_2 \) and \( k_1 = k_3 \).

If all the 3 keys are identical, then it is the same as DES.

Of the 3 keying options, option 1 is the strongest as the resulting effective key length is 168 bits and option 2 has a key length of 112 bits.

3.2 Modes of Operation

Any block cipher is not secure unless it is used in a mode of operation. “A block cipher mode, or mode, for short, is an algorithm that features the use of a symmetric key block cipher algorithm to provide an information service, such as confidentiality or authentication” [48]. A block cipher deals with the secure encryption and decryption of a single block of data whereas a mode of operation deals with the application of the cryptographic algorithm on a stream of data usually larger than a single block [48].

There are 12 modes of operation currently approved by NIST. These are classified into Confidentiality modes, Authentication modes and combined mode for confidentiality and Authentication. Electronic Codebook (ECB), Cipher-Block Chaining (CBC), Output Feedback (OFB), Cipher Feedback (CFB), Counter (CTR) and Xor-encrypt-xor based tweaked-codebook mode with ciphertext stealing – AES (XTS-AES) [49] are the 6 approved confidentiality modes [48].

Cipher based Message Authentication Code (CMAC) is the only type of authentication mode. Counter with CBC-MAC (CCM), Galois/Counter Mode (GCM), the AES Key Wrap (KW), AES Key
Wrap with Padding (KWP), Triple DES Key Wrap (TKW) are the five approved modes combining both confidentiality and authentication [50]. As the names indicate, confidentiality modes keep the data hidden but they offer no protection against data modification and tampering which is provided by the authentication mode [51].

The most common modes of operation used for Triple DES are Electronic Codebook and Cipher-Block Chaining modes. As a part of our research, the Electronic Codebook (ECB) mode is used to implement the Triple DES. It is the simplest of all the modes of encryption. It divides the message into data blocks and each block is separately worked on. Figures 3.1 and 3.2 show the encryption and decryption processes of the ECB mode respectively.

![Diagram of ECB mode]

Figure 3. 1 : Encryption in ECB mode
The Cipher Block Chaining (CBC) mode was invented by IBM in 1976. Since then, it has been the most widely used mode of operation [52]. The current ciphertext is XORed with the next plaintext before its encryption in this mode and an Initialization Vector is used for the first block. Due to the dependency of the current ciphertext on all the previous plaintexts, the encryption operation cannot be parallelized and even a change in a single bit of the plaintext or the Initialization Vector affects all the cipherblocks that follow.

3.3 Security

Although the Triple DES with 3 independent keys seems impervious to any attacks, it is prone to meet-in-the-middle attack [53]. “A meet-in-the-middle attack is a cryptographic attack, first
developed by Diffie and Hellman, employs a space-time tradeoff to drastically reduce the complexity of cracking a multiple-encryption scheme” [53].

If we consider the above used notation for the plaintext, ciphertext, encryption and decryption, then meet-in-the-middle attack can be described as

\[
C = F_{k3} \left[ f_{k2} \left( F_{k1}(P) \right) \right]
\]

\[
F_{k3}(C) = f_{k2} \left( F_{k1}(P) \right)
\]

Now, the attack forms 2 look-up tables, the first one containing pairs of all possible combinations of the key \( k1 \) and \( k2 \) \( \left( 2^{112} \right) \) combinations and the resulting block formed after performing the encryption and decryption process with \( k1 \) and \( k2 \) respectively \( \left( f_{k2} \left( F_{k1}(P) \right) \right) \) and the second one containing pairs of all possible combinations of the key \( k3 \) \( \left( 2^{56} \right) \) and the resulting block formed after decryption of the ciphertext \( C \).

Then, the two tables are compared for any matching keys which reveal the keys used in the process. As this is a tedious process the entries into the look-up tables are compared as soon as they are entered. Naturally, previous knowledge of the plaintext and ciphertext pair is required besides the huge memory space to accommodate the entries of the two tables as well as the computational power to perform the operation.

Usually, a brute force attack takes at most \( 2^{168} \) key combinations to crack the Triple DES whereas the meet-in-the-middle attack greatly reduces this complexity and brings it down to \( 2^{112} \). Hence, the effective security provided by the Triple DES when using 3 distinct keys is only 112 bits. Although all federal applications should use 3 independent keys [47], there are some applications where keying option 2 is used and in such cases, the effective security it provides is
only 80 bits and is vulnerable to chosen or known plaintext attacks \[54\] \[55\]. The MITM attack for the Triple DES algorithm can be avoided by changing the keys after every \(2^{31}\) encryptions.

### 3.4 HDL Implementation

The individual modules such as the S boxes, the P box, the E box and the key schedule are implemented separately from the lookup tables provided in the appendix 1. They are later structurally implemented forming bigger modules till they reach one single module. This module is used in writing the cipher code for DES.

The DES code consists of a State machine which is shown in figure 3.3.

![State machine and Truth table for the HDL implementation of DES](image)

**Figure 3.3:** State machine and Truth table for the HDL implementation of DES
There are 5 states in the code. In the waitforkey state, the code waits for the key to be entered. Once the key is fed, the control switches to the next state which is the waitfordata state.

In this state, the code waits for the data to be entered. Once the data is given to the code, the DES operation is commenced and as can be seen from figure 2.1, it continues for 16 rounds. This is indicated as the firstround, lastround and the repeatround in the state diagram. The control switches from waitfordata state to the firstround state when the data is fed and then after the firstround state, it switches to repeatround state where it iterates 14 times. At the end of the 14th iteration, the control is switched to lastround state where upon completion, it switches back to waitforkey state. In addition to these states and the key and data input, we have other control signals such as clock, reset and function_select(used for selection between encryption and decryption). The simulation results for DES are explained in detail in Appendix – 2.

The code for Triple DES contains the component of DES initialized thrice along with the initialization of memory in the form of arrays. As we can see from figure 3.4, we are declaring 3

```
169   TYPE MEM1 IS ARRAY (0 to 1) OF STD_LOGIC_VECTOR (63 DOWNTO 0);
170   TYPE MEM2 IS ARRAY (0 to 1) OF STD_LOGIC_VECTOR (63 DOWNTO 0);
171   TYPE MEM3 IS ARRAY (0 to 1) OF STD_LOGIC_VECTOR (63 DOWNTO 0);
172   SIGNAL ram1 : MEM1;
173   signal ram2 : MEM2;
174   signal ram3 : MEM3;
```

Figure 3.4: Memory initialization in the form of arrays.
different arrays instead of one single array to reduce the overhead caused by the addressing of the elements in the array thus reducing the complexity of the code. We are storing each key in its own array and using them as needed. The Triple DES code also has 2 different states—waitforkey and waitfordata. The control waits until the keys are fed and then switches to waitfordata state and it remains in this state until input is given. Once the input is fed, the Triple DES operation is performed on the data. Appendix – 2 gives a detailed explanation of the simulation of Triple DES in Modelsim and analysis of the results.

3.5 Huffman Encoding

Huffman encoding is employed to achieve lossless data compression [56]. David A. Huffman published a paper in 1952 entitled “A method for the construction of minimum redundancy codes” [57] with the main aim of minimizing the average number of coding digits per message.

The process of compression is performed by creating a binary tree which consists of nodes. The number of nodes depend on the number of symbols that are going to be compressed. At the beginning of the compression process, all the symbols are leaf nodes of the tree with their corresponding frequency as their respective weight. Conventionally, binary ‘1’ represents the right child and binary ‘0’ represents the left child in the tree. The final tree comprises of n-1 internal nodes and n leaf nodes where n is the number of symbols.

The Huffman’s algorithm is implemented using a min heap that stores the symbols and constructs a graph that becomes a binary tree eventually. The weight or the frequency of each symbol is its value. The following are the steps in the implementation
• All the n symbols are to be entered into a min heap. Being a min heap, the symbols are sorted in the ascending order with the symbols having the least weight lying at the top.

• As long as the heap has more than 1 symbol in it, the top two symbols from the heap are removed one after the other.

• A new symbol is created with the weight equal to the sum of the weights of the removed symbols.

• The newly created symbols becomes the parent to the two symbols from which it has been derived.

• This process is continued until there is one symbol left which becomes the root of the binary tree.

The time complexity of the algorithm will be O (n log n) where n represents the number of symbols because a heap time complexity is O (log n) and we are dealing with a tree comprising of 2n-1 nodes and n leaves [56].

The following example uses the above mentioned steps to generate binary codes for the given set of symbols

**Example:**

Consider a set of symbols {a, b, c, d} with frequency of occurrences {0.6, 0.25, 0.1, 0.05}. The Huffman algorithm first arranges them in the decreasing order of their frequencies. For simplicity of operation, the frequencies are taken such that they are in the decreasing order.

Now, the two symbols with the least frequencies are combined to form a new internal node and the resulting symbols are rearranged in descending order and this process repeats until
there is one symbol left. Now, a tree is formed and each node is assigned a code starting with the root node. The left child of the root node is assigned ‘0’ and the right child is assigned ‘1’. This assignment continues until we reach the leaf nodes. As each leaf node represent a symbol, the last remaining codes become the individual codes for each symbol. The following figure illustrates the process.

![Binary tree generated through Huffman encoding.](image)

**Figure 3.5**: Binary tree generated through Huffman encoding.

The resulting codes for each symbol are given in the table below

<table>
<thead>
<tr>
<th>symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
</tr>
<tr>
<td>c</td>
<td>110</td>
</tr>
<tr>
<td>d</td>
<td>111</td>
</tr>
</tbody>
</table>

**Table 3.1**: The symbols and the corresponding codes
As we can observe from the above codes that the symbols with the highest frequencies have the least number of bits in the code. This is one of the important features of data compression through Huffman coding.

The Huffman coding algorithm is implemented in C. Once the data is compressed through this algorithm, the resulting binary data is given to the Triple DES module for encryption and then it is transmitted. The main idea behind using the Huffman coding is to reduce redundancy and reduction in redundancy makes the message more secure [58] [59]. This allows us to use the same key for a longer period and this in turn is very useful as the keys are transmitted through RSA which is slow in operation [60].
4. RESULTS

This chapter deals with Huffman encoding, the implementation of Triple DES on a Field Programmable Gate Array, specifically, one with an Altera NIOS II processor. We are using an Altera DE1 board for our simulation in Quartus II version 13.0 sp1 [14].

Once a binary tree is generated using Huffman encoding and the data is compressed, at the decompressing end, we need to have the same tree to decompress the data and to find out what the original message was. There are two solutions to achieve this.

- We can use the standard letter frequencies for the language which we are sending the message in (in our case, American English). With this method, the tree never changes and it is a part of the decoding machinery.
- We can send the generated tree to the decompressing end using RSA. As we are sending the keys of Triple DES using RSA [18] or Diffie- Hellman anyway, the tree could be included with them.

For the purpose of our Thesis, we can implement the first solution and construct a tree with the standard frequencies of the American English alphabet and use the resulting codes for the symbols to encode our message.

The following table gives the alphabet and the corresponding frequencies.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency (relative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>8.12%</td>
</tr>
<tr>
<td>b</td>
<td>1.492%</td>
</tr>
<tr>
<td>---</td>
<td>--------</td>
</tr>
<tr>
<td>c</td>
<td>2.71%</td>
</tr>
<tr>
<td>d</td>
<td>4.32%</td>
</tr>
<tr>
<td>e</td>
<td>12.02%</td>
</tr>
<tr>
<td>f</td>
<td>2.30%</td>
</tr>
<tr>
<td>g</td>
<td>2.03%</td>
</tr>
<tr>
<td>h</td>
<td>5.92%</td>
</tr>
<tr>
<td>i</td>
<td>7.31%</td>
</tr>
<tr>
<td>j</td>
<td>0.10%</td>
</tr>
<tr>
<td>k</td>
<td>0.69%</td>
</tr>
<tr>
<td>l</td>
<td>3.98%</td>
</tr>
<tr>
<td>m</td>
<td>2.61%</td>
</tr>
<tr>
<td>n</td>
<td>6.95%</td>
</tr>
<tr>
<td>o</td>
<td>7.68%</td>
</tr>
<tr>
<td>p</td>
<td>1.82%</td>
</tr>
<tr>
<td>q</td>
<td>0.11%</td>
</tr>
<tr>
<td>r</td>
<td>6.02%</td>
</tr>
</tbody>
</table>
Table 4.1: Standard American English letters and frequencies [61]

The above discussed Huffman coding method relies on previous knowledge of the symbol frequencies thus creating trees based on that data. An improvement in the Huffman coding changes the frequencies of the symbols as they are being transmitted thus increasing the efficiency of the compression and transmission. This variation is known as adaptive or dynamic Huffman coding [62].

Unlike the normal Huffman coding where the tree structure is unchanged, the adaptive Huffman coding changes the structure of the tree as the frequencies of symbols vary. The most notable implementations are Faller-Gallager and Knuth (FGK) algorithm [63] and Vitter algorithm [63].
The main difference between both the algorithms is that the entire tree is checked when the frequencies are updated in Vitter algorithm. The FGK algorithm only checks the nodes against the right sibling of its parent or its right sibling [64]. This makes the FGK algorithm

4.1 NIOS II Processor

NIOS II processor is widely used in a variety of applications involving Field Programmable Gate arrays and is the most versatile processor in the world [65]. There are 3 varieties of NIOS II processors – the fast core (f), the economy core (e) and the standard core (s). The features of the fast core and standard core processors include a vector interrupt controller, tightly coupled memory, ability to customize instructions and support of Real Time operating Systems (RTOS) [65]. The ability to run embedded Linux with the help of a Memory management Unit (MMU) is an exclusive feature of the fast core processors. The economy core processors can be used for microcontroller applications as they contain a very less number of logic elements and they are inexpensive.

NIOS II has a 32 bit RISC architecture with 32 general purpose registers [13]. A system with a NIOS II can also be viewed as a microcontroller as it consists of the processor, on-chip memory and peripherals [13]. As the logic resources and pins are programmable in Altera devices, customization is possible in any manner seen fit. For example, the number of pins can be reduced to simplify the design of the system. Although the logic elements used in the system will be increased because of this, the board design becomes easier. Also, in instances where the entire resources of the processor are not utilized, the reminder can be used for implementing some processor unrelated functions, sometimes entire systems.
4.2 Implementation

We are employing an Altera Cyclone II device for implementing the Triple DES. Specifically, the model EP2C20F484C7 is used with a core voltage of 1.2 Volts, 315 pins, 4 PLLs and 16 global clocks. We are using Qsys system integration tool for generating the desired system. It allows us to choose the specific type of NIOS II processor, add or remove different input and output blocks and peripherals to our system [66]. Qsys is released as an update for the older SOPC builder which was the standard tool for configuring and generating an embedded system for Altera devices. There is a huge choice of embedded and DSP processors and of interface protocols such as Ethernet, PCI, Serial etc. There is a provision for external memory but the on-chip memory which is again divided into RAM and ROM is sufficient for our application. We can also select the base addresses for different components of our system making the process of design simple and fast. Once the design is finished, it can be synthesized in either Verilog or VHDL and can be simulated in Quartus II. We can also create a testbench system where the desired system is being instantiated and bus functional models can be added for driving the top level interfaces. These bus functional models can interact with the simulated system once generated. We are using the following components in our system.

- NIOS II processor
- Clock with a frequency of 50 MHz
- On-chip memory of 4 KB
- JTAG UART to connect the components used.
4.3 Simulation

Appendix – 2 guides through the steps involved in creating a design in Qsys, creating a project in Quartus II and including the required design files in the project. Once all of this is completed, we can compile the project. Compilation reveals the physical properties of the system that has been designed which is essential in optimizing it and making it more efficient.

Figure 4.1 shows the simulation results of our system. It gives the information about the number of combinational functions and dedicated logic registers and most importantly, the number of pins used in the design. We are using 3,278 of the available 18,753 logic elements of which 2,569 are combinational functions and 1,538 are registers. The former and latter utilize 14% and 8% of the available resources respectively bringing the total logic utilization to a mere 17%. 206 of the available 315 pins are used in the design which is about 65% of the total utilization.

![Compilation Report - TDES](image)

Figure 4.1: Simulation results with device EP2C20F484C7
Figure 4.2 shows the RTL schematic of the designed system. It contains the signals ldkey (key loading ready signal), lddata (data loading ready signal), data_in (data input), we (write enable), write_address (write address), function_select (encryption or decryption), clock, reset, main_key_in (key input) as the input signals and out_ready (output ready indicator) and data_out (data output) as the output signals. It also shows the three rams – ram1, ram2 and ram3 which were declared to store the incoming keys in successive clock cycles.
4.4 Analysis

The above figure shows the utilization of the logic on the FPGA during a normal implementation of Triple DES versus the specified implementation using memory. From the figure, it is evident that the combinational functions used for implementing with memory are more compared to that of the normal implementation but compared to the total amount of the available resource, the difference is very small.

During the normal implementation, the number of functions used are 1088 and our implementation costs us 1481 more bringing the amount to 2569. But the total available combinational logic is 18,752 which means that our implementation increases the combinational logic utilization by a mere 7.8%. This increase in combinational logic is attributed to the usage of functions which make use of the memory in the system.
Table 4.2: Comparison with the previous implementation

The figure also shows the number of pins utilized vs the available pins on the considered configuration in the normal implementation as well as the proposed implementation. In the normal implementation, 304 of the available 315 pins are used which amounts to 97% of pin usage whereas in the implementation with memory, only 206 pins are used which is 65% of the total pin count. The above table shows the difference between the previous implementation [67] and our implementation.

The saved pins can be used to implement a different module or an entirely different system or can be left alone to save power and to increase the performance of the embedded system. For
example, the saved pins and the rest of the combinational functions can be used for implementing other modes of operation like CBC mode or can be used to implement a module that changes the keys on a regular basis which greatly increases the efficiency of the system. The same method is implemented in different Xilinx devices and a significant reduction in the number of pins is observed.

Figure 4.5 gives us an estimate of the power utilization of the hardware implementation of Triple DES. From the analysis, we can know the ambient temperature of the circuit, the airflow required and also the capacity of the heat sink on the board.

![Power Estimation for the resulting Implementation](image)

Figure 4.4: Power Estimation for the resulting Implementation

The Device and Package give the details of the device in use which is EPC2C20F484C7. The C indicates that the device is of commercial grade. For optimum performance of the device, it
needs an airflow of about 1 m/s and a 23mm heat sink. Thermal Analysis indicates that the junction Temperature is 25.8°C and the maximum allowed temperature is 83.9°C. The power consumption by the Logic Components is 0.003 Watt and by the I/O is 0.049 W. The clock consumes about 0.002 W of power and combined with a static power of 0.047W, the total power consumption of the device will be 0.101 W.

4.5 Summary

Although the keying option 1 for Triple DES is very secure and all its cracking methods are highly impractical, there are some simple solutions to protect the data if there is a potential attack on the system. One solution is to use a new set of keys after every $2^{31}$ plaintext encryptions or decryptions. This is based on the fact that it requires $2^{32}$ known plaintexts, $2^{88}$ bits of memory and $2^{113}$ steps for the best known attack on Triple DES that has 3 independent keys. Which means by the end of the encryption of the $2^{32}$nd plaintext, the attacker can get hold of the keys. So, changing the keys frequently ensures data security before the algorithm itself.
5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

We have seen the history of modern cryptography and also the history of one of the important cryptographic algorithm, the DES. We discussed the structure of the algorithm in detail and also analyzed its advantages and disadvantages. Then we moved on to its successor, the Triple DES whose structure and functionality were explained in detail.

Then we analyzed the results of the HDL implementation of Triple DES and used this analysis to make modifications to our hardware design. The Algorithm was implemented on an Altera DE1 board with a NIOS II processor and the results obtained using Altera Quartus II are analyzed and a significant reduction in the pin usage was observed.

Although the pin usage was reduced, there was a minute increase in the amount of logic elements used due to the incorporation of memory in the implementation. Memory plays an important role in storing the keys in successive clock cycles and allowing them to be retrieved later.

Hence, by using different components of an embedded system, it is possible to implement a cryptographic algorithm efficiently which increases the possibility of manufacturing low cost embedded systems that can provide a high degree of security for a wide variety of applications.

5.2 Future Work

This work demonstrated that it is possible to increase the efficiency of implementing a cryptographic algorithm on an FPGA. On – chip memory is used for this purpose and this can be
eliminated by making modifications to the algorithm.

To further increase the security of the data through Huffman coding, the binary tree that has been generated at the source can be sent to the destination as it is in the clear or using RSA algorithm. Figure 5.1 shows the planned block diagram.

![Block Diagram](image)

Figure 5.1: Sending the tree through RSA

Adaptive Huffman coding which is an efficient variation to Huffman coding can be implemented using either the Faller-Gallager–Knuth algorithm or the Vitter algorithm.

Other factors that affect the performance like the number of logic elements and registers can be optimized in the future. Methods to avoid attacks on Triple DES such as Meet in the Middle (MITM) Attacks can be implemented making the algorithm much more secure. Extensions of this solution can be implemented.

Reduction in the pin count further by imparting different type of coding techniques is also of interest. Also, algorithms such as Advanced Encryption Standard (AES), Blowfish etc. which are much more advanced and complicated can be implemented on an FPGA as future work.
Bibliography


23, 2014.


This chapter gives a detailed description of various components of the DES algorithm which are

1. Initial Permutation (IP)
2. Final Permutation (FP)
3. Expansion (E)
4. Substitution (S)
5. Permutation (P)
6. Permuted Choice 1 (PC1)
7. Permuted Choice 2 (PC2)

1. INITIAL PERMUTATION (IP)

The data block which is 64 bit wide goes through the process of IP which basically shuffles the bits according to the following table.

<table>
<thead>
<tr>
<th>58</th>
<th>50</th>
<th>42</th>
<th>34</th>
<th>26</th>
<th>18</th>
<th>10</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>52</td>
<td>44</td>
<td>36</td>
<td>28</td>
<td>20</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>62</td>
<td>54</td>
<td>46</td>
<td>38</td>
<td>30</td>
<td>22</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>64</td>
<td>56</td>
<td>48</td>
<td>40</td>
<td>32</td>
<td>24</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>57</td>
<td>49</td>
<td>41</td>
<td>33</td>
<td>25</td>
<td>17</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>
So, the 1\textsuperscript{st} bit of the input will be shuffled to 58\textsuperscript{th} bit of the output, 2\textsuperscript{nd} bit to 50\textsuperscript{th} and so on.

2. **FINAL PERMUTATION (FP)**

After 16 rounds of Feistel scheme, the resulting data will undergo a final permutation which is again shuffling the bits. The following table gives the ordering.

<table>
<thead>
<tr>
<th></th>
<th>40</th>
<th>8</th>
<th>48</th>
<th>16</th>
<th>56</th>
<th>24</th>
<th>64</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>7</td>
<td>47</td>
<td>15</td>
<td>55</td>
<td>23</td>
<td>63</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>6</td>
<td>46</td>
<td>14</td>
<td>54</td>
<td>22</td>
<td>62</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>5</td>
<td>45</td>
<td>13</td>
<td>53</td>
<td>21</td>
<td>61</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>4</td>
<td>44</td>
<td>12</td>
<td>52</td>
<td>20</td>
<td>60</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>3</td>
<td>43</td>
<td>11</td>
<td>51</td>
<td>19</td>
<td>59</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>2</td>
<td>42</td>
<td>19</td>
<td>50</td>
<td>18</td>
<td>58</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>1</td>
<td>41</td>
<td>9</td>
<td>49</td>
<td>17</td>
<td>57</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>
3. **EXPANSION**

The Expansion function takes the 32 bit half-block as input and expands it to 48 bits according to the following table:

<table>
<thead>
<tr>
<th>32</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>1</td>
</tr>
</tbody>
</table>

The process of expansion is achieved through duplication of bits. The 1st bit of the input is duplicated twice – once as the 2nd bit of the output and once as the 48th bit.

4. **SUBSTITUTION**

The substitution boxes substitute each of their 6 bit input with a 4 bit output according to the following tables.

**S1**

<table>
<thead>
<tr>
<th>14</th>
<th>4</th>
<th>13</th>
<th>1</th>
<th>2</th>
<th>15</th>
<th>11</th>
<th>8</th>
<th>3</th>
<th>10</th>
<th>6</th>
<th>12</th>
<th>5</th>
<th>9</th>
<th>0</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>11</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
<td>14</td>
<td>8</td>
<td>13</td>
<td>6</td>
<td>2</td>
<td>11</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>14</td>
<td>10</td>
<td>0</td>
<td>6</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

**S2**

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>1</th>
<th>8</th>
<th>14</th>
<th>6</th>
<th>11</th>
<th>3</th>
<th>4</th>
<th>9</th>
<th>7</th>
<th>2</th>
<th>13</th>
<th>12</th>
<th>0</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>13</td>
<td>4</td>
<td>7</td>
<td>15</td>
<td>2</td>
<td>8</td>
<td>14</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>9</td>
<td>11</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>7</td>
<td>11</td>
<td>10</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>12</td>
<td>6</td>
<td>9</td>
<td>3</td>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>10</td>
<td>1</td>
<td>3</td>
<td>15</td>
<td>4</td>
<td>2</td>
<td>11</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>0</td>
<td>5</td>
<td>14</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

**S3**

<table>
<thead>
<tr>
<th>10</th>
<th>0</th>
<th>9</th>
<th>14</th>
<th>6</th>
<th>3</th>
<th>15</th>
<th>5</th>
<th>1</th>
<th>13</th>
<th>12</th>
<th>7</th>
<th>11</th>
<th>4</th>
<th>2</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>7</td>
<td>0</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>14</td>
<td>12</td>
<td>11</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>8</td>
<td>15</td>
<td>3</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>2</td>
<td>12</td>
<td>5</td>
<td>10</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>13</td>
<td>0</td>
<td>6</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>15</td>
<td>14</td>
<td>3</td>
<td>11</td>
<td>5</td>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>

**S4**

<table>
<thead>
<tr>
<th>7</th>
<th>13</th>
<th>14</th>
<th>3</th>
<th>0</th>
<th>6</th>
<th>9</th>
<th>10</th>
<th>1</th>
<th>2</th>
<th>8</th>
<th>5</th>
<th>11</th>
<th>12</th>
<th>4</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>8</td>
<td>11</td>
<td>5</td>
<td>6</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>12</td>
<td>1</td>
<td>10</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>9</td>
<td>0</td>
<td>12</td>
<td>11</td>
<td>7</td>
<td>13</td>
<td>15</td>
<td>1</td>
<td>3</td>
<td>14</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>12</td>
<td>7</td>
<td>2</td>
<td>14</td>
</tr>
</tbody>
</table>
### S5

<table>
<thead>
<tr>
<th>2</th>
<th>12</th>
<th>4</th>
<th>1</th>
<th>7</th>
<th>10</th>
<th>11</th>
<th>6</th>
<th>8</th>
<th>5</th>
<th>3</th>
<th>15</th>
<th>13</th>
<th>0</th>
<th>14</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>11</td>
<td>2</td>
<td>12</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>10</td>
<td>3</td>
<td>9</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>11</td>
<td>10</td>
<td>13</td>
<td>7</td>
<td>8</td>
<td>15</td>
<td>9</td>
<td>12</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>12</td>
<td>7</td>
<td>1</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>6</td>
<td>15</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

### S6

<table>
<thead>
<tr>
<th>12</th>
<th>1</th>
<th>10</th>
<th>5</th>
<th>9</th>
<th>2</th>
<th>6</th>
<th>8</th>
<th>0</th>
<th>13</th>
<th>3</th>
<th>4</th>
<th>14</th>
<th>7</th>
<th>5</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>15</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>12</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>13</td>
<td>14</td>
<td>0</td>
<td>11</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>15</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>12</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>12</td>
<td>9</td>
<td>5</td>
<td>15</td>
<td>10</td>
<td>11</td>
<td>14</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>8</td>
<td>13</td>
</tr>
</tbody>
</table>

### S7

<table>
<thead>
<tr>
<th>4</th>
<th>11</th>
<th>2</th>
<th>14</th>
<th>15</th>
<th>0</th>
<th>8</th>
<th>13</th>
<th>3</th>
<th>12</th>
<th>9</th>
<th>7</th>
<th>5</th>
<th>10</th>
<th>6</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>11</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>10</td>
<td>14</td>
<td>3</td>
<td>5</td>
<td>12</td>
<td>2</td>
<td>15</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>11</td>
<td>13</td>
<td>12</td>
<td>3</td>
<td>7</td>
<td>14</td>
<td>10</td>
<td>15</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>5</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>13</td>
<td>8</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>14</td>
<td>2</td>
<td>3</td>
<td>12</td>
</tr>
</tbody>
</table>
5. PERMUTATION

The 32 bit output of the substitution undergoes a shuffle according to the permutation table below. The 1st bit of the output of the Permutation is assigned to the 16th bit of the input and so on.
6. **PERMUTED CHOICE 1 (PC1)**

The PC1 function takes the 64 bit key as the input and gives a 56 bit key as the output which is divided into 2 28 bit halves. Hence the tables doesn’t have the parity bits included in them. There are two tables for PC1 as the output is divided.

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>49</td>
<td>41</td>
<td>33</td>
<td>25</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>58</td>
<td>50</td>
<td>42</td>
<td>34</td>
<td>26</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>59</td>
<td>51</td>
<td>43</td>
<td>35</td>
<td>27</td>
</tr>
<tr>
<td>19</td>
<td>11</td>
<td>3</td>
<td>60</td>
<td>52</td>
<td>44</td>
<td>36</td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>55</td>
<td>47</td>
<td>39</td>
<td>31</td>
<td>23</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>62</td>
<td>54</td>
<td>46</td>
<td>38</td>
<td>30</td>
<td>22</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>61</td>
<td>53</td>
<td>45</td>
<td>37</td>
<td>29</td>
</tr>
<tr>
<td>21</td>
<td>13</td>
<td>5</td>
<td>28</td>
<td>20</td>
<td>12</td>
<td>4</td>
</tr>
</tbody>
</table>
```

7. **PERMUTED CHOICE 2 (PC2)**

The PC2 function selects the 48 bit subkey from the 56 bit input according to the following table.

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>7</td>
<td>11</td>
<td>24</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>15</td>
<td>6</td>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td>23</td>
<td>19</td>
<td>12</td>
<td>4</td>
<td>26</td>
<td>8</td>
</tr>
</tbody>
</table>
```
TUTORIAL FOR SIMULATION IN QUARTUS II

The following steps are involved in the implementation of the algorithm on the FPGA.

- Implement the algorithm in a Hardware description language.
- Determine the number of pins required and vary the code accordingly.
- Using Altera Quartus II software, simulate the code on the target FPGA and obtain the results.

For this tutorial, we are using version 13.0 of Altera Quartus II and version 10.2b of Modelsim for the simulations. For newer versions, modifications to details may be required.

To implement the algorithm in a hardware description language, we are choosing VHDL and modelsim for simulation. First, we divide the algorithm into various modules and individually describe each module in VHDL. These modules include the Substitution boxes (S), the Permutation box (P), the Expansion function (E), the integration of S, P and E boxes which forms the Feistel function, Permutated Choice (PC) – 1, Permutated Choice (PC) – 2, the integration of PC1 and PC2 which forms the key scheduling and finally combining the Feistel function, the key scheduling, the Initial and final permutations to form the DES.
At this point, we need to verify whether the code written so far is yielding the correct result or not. So, a simulation is done for DES by writing a testbench to test the functionality of DES. For this purpose, we are using the following specifications

- Clock frequency = 50 MHz
- Key = 1111111111111111111111111111111111111111111111111111111111110000
- Data = 1010101010101010101010101010101010101010101010101010101010101

Then encryption is performed on the above given data with the specified key. The simulation result is shown in figure A.

![Figure A: DES Encryption](image-url)
The resulting data output is

Output = 001001000011100110010011110011011000011010101010101010000011

Now, the correctness of the code can be verified if this output produces the previously given input as output if it is given as input to the DES with the same key. Putting it plainly, if we decrypt the above obtained output with the same key, we should get the same data block which was given as the input for the encryption as the output. Using the same clock frequency and key as above, we are changing the input data to

Data = 001001000011100110011011110000110101010101010101000011

And performing decryption which yields the simulation result shown in figure B. As it is implied from the simulation, we have

Output = 1010101010101010101010101010101010101010101010101010101010101010

Hence, the code is correct so far as the same data block was yielded as output after performing successive encryption and decryption operations on that block of data. Now, this code is the building block for the Triple DES. This code is implemented thrice structurally in the Triple DES as per the algorithm and is simulated by writing a testbench to verify its correctness. The most important thing to consider here is that though the code by itself is correct, its implementation on the FPGA might yield erroneous results. The reason for this is the presence of a large number of input and output signals which are translated to pins on an FPGA.
For the code which we use, we have a 64 bit data input, a 192 bit key which is fed as 3 64 bit keys, a 64 bit data output and 10 other signals which is equal to 330 pins on the board which in some cases is not possible to implement. So, in the process of writing the code for the Triple DES, we are using the memory provided on the FPGA to store the keys first and use them later in the encryption process.

This technique is really efficient in terms of the amount of pins being saved on the FPGA and the resulting benefits from it. As every FPGA comes with some memory included in it, the memory used for storing the keys is not an addition to the existing hardware. There is an
increase in the amount of logic blocks used by this process but they also pose no addition to the hardware but using the existing hardware efficiently.

To implement the above proposed method, we are making use of a single input signal called Key_in where we are giving the 3 keys as input in successive clock cycles and storing them in the memory which we define in the code. When the keys are stored, we give the input data and perform the required encryption or decryption operation on it. The following specifications are used in the implementation.

- Clock frequency = 100 MHz
- Key 1 = 1010101010101010101010101010101010101010101010101010101010101010
- Key 2 = 0000000000000000000000000000000000000000000000000000000000000000
- Key 3 = 1111111111111111111111111000000000000000000000000000000000000000
- Data = 1010101010101010101010101010101010101010101010101010101010101010

The simulation result is shown in figure C. After encryption, the achieved result is

Output = 0001111000000110011011101110101101000110010110110111110.

To test the correctness of the code, we perform decryption on this block of data and check whether we can get the original Data input as its output. So, using the same specifications, we are changing the input to

Data = 00011110000001100110111011101011011011000110010110110111110
Figure C: Triple DES Encryption

Figure D: Triple DES Decryption
With the above data input, we perform decryption which yields the simulation result shown in figure D. From the figure, it is evident that we were able to achieve the initial input at the output which is

Output = 1010101010101010101010101010101010101010101010101010101010101010.

Another example for implementation that we are considering is the sentence “The quick brown fox jumped over the lazy dog”. After Huffman coding, this turns to a binary sequence which is as follows.

11110001011011001010011100101011000000101000001100101011101110010000101011111010100001110010101000000111011100110010110110111011100101101000100111111000101011010110010010100110011

The simulation results of this binary sequence are shown in figures E, F, G, H, I and J. E and F gives us the encryption and decryption of the first 64 bits of the sequence, G and H shows us the encryption and decryption of the next 64 bits of the sequence and I and J shows us the encryption and decryption of the last 64 bits of the sequence.

As we can observe from the results, the output of the decryption process yields the same result as that of the input of the encryption process. Now, when the output of the Triple DES decryption module is given to the Huffman decoding block, the tree is backtracked to reveal the original message which in this case is “The quick brown fox jumped over the lazy dog”. Thus, we can implement this code on a Field Programmable Gate Array of our choice. We are opting for an Altera DE1 board with a NIOS II softcore processor.
Figure E: Triple DES Encryption of the example (part 1)

Figure F: Triple DES Decryption of the example (part 1)
Figure G: Triple DES Encryption of the example (part 2)

Figure H: Triple DES Decryption of the example (part 2)
Figure I: Triple DES Encryption of the example (part 3)

Figure J: Triple DES Decryption of the example (part 3)
QUARTUS II TUTORIAL

Quartus II is a tool designed by Altera Corporation for the synthesis and analysis of designs described in a Hardware Description Language for programmable logic devices. Open the Quartus II tool installed in your system. Click on the File menu and select new Quartus II project from the list. Enter a new working directory (Example: D:\Thesis\Simulation) and give a new name to the project. By clicking next, the project wizard redirects us to a new window where design files can be added to the project. Add all the required design files and click next. Then, a selection for device specification appears where for the family, Cyclone II and FBGA for the package is chosen.
From the list of devices mentioned, select a device with the name EP2C20F484C7 as shown in figure E. In the next window, we will be asked to specify any other EDA tools that might be used in the project. As we are not using any of those tools, we can proceed to the next window. Then a summary of our settings appears for our final approval to create the project. After clicking finish, a new project is created.

At this point, we need to create a hardware model for our simulation with different components such as the processor, memory, I/O etc. For this task, we are using a tool called Qsys which is an upgrade for the SOPC Builder. To open Qsys, go to tools and select Qsys and select new design and save it. Upon initializing Qsys, we can observe that only the clock (clk) has been defined. To choose a processor, click on the embedded processors tab under the library section which is at the left corner of the window. In the embedded processors, select NIOS II processor and it will appear on the system contents list under the name nios2_qsys. A window pops up asking for the specifications of the NIOS II processor as shown in figure F. Select NIOS II/s as the core and leave the other settings as they are and hit finish.

To select memory, click on Memories and Memory Controllers tab and select on-chip. If the memory requirements are huge, we can make use of external memory interfaces. As we are dealing with memory requirements in the range of a few bytes, on-chip memory is sufficient. In the on-chip tab, select the on-chip memory RAM or ROM. In the memory specification window, select memory type as RAM, data width as 32 and total memory size as 4096 bytes.
Now, we need to connect these components on the FPGA. Select the Interface protocols tab and click on serial. Select JTAG UART from the list and leave the settings as they are and hit finish. Select system and click assign base addresses which assigns the base addresses for different components initialized in our project. Now, edit the base address of the memory to 0X00000000 and lock it so that it cannot be changed.

We gathered the components of our FPGA and specified their properties. Now we need to connect them. In the connections section, connect the clock output signal and the reset output of the clock to clock input and the reset input of the processor, memory and the UART respectively. Connect the data_master signal of the processor to the jtag_debug_module signal of the processor, s1 signal of the memory and Avalon_jtag_slave signal of the UART.
In the IRQ section, connect the IRQ signal which is the data_master signal of the processor to the Avalon_jtag_slave of the UART. The names of the components can be renamed as per our requirements. The final system is shown in figure G.

If we observe in the messages section, we will find 2 errors regarding the setting of reset vector and exception vector memory. To clear those, go to the properties of NIOS II processor and scroll down until the section of reset vector appears. Select onchip memory for reset vector memory and exception vector memory and hit finish. Now, save the design and click on the generation tab and select a synthesis language. As we are using VHDL, select VHDL and specify a path for output file and hit generate.
Once the generation is complete, we can close Qsys and proceed to our project. In the main window, select add/remove files from project in the project section. In the add files window, give the directory where the synthesis files of the Qsys are stored and select 2 files, both having the same name as the saved Qsys design; one is a VHDL file and the other is an IP variation file with extension .qip and add them to the current project.

Save the project and set the top level design entity in the project navigator window. Once this is done, our project is ready for compilation. Click on processing and select start compilation and observe the progress in the tasks window. As we can see, there are many stages in compilation – Analysis and synthesis, Analysis and elaboration, Fitting, Assembling and Timing analysis. Carefully check if each individual stage is being passed or not. If there is an error during compilation, we can easily determine where we went wrong by analyzing these stages. A successful compilation is shown in figure H.

In the event of a successful compilation, a compilation report is generated which contains information regarding the hardware used for the design, for example – total logic elements, registers, pins, PLLs etc. With the help of this report, the design can be optimized to reduce the amount of hardware required or to achieve the desired timing requirements.
![Figure N: Successful Compilation](image-url)