International Journal For Technological Research In Engineering
Vol. 1, Issue. 1, Sep – 2013 *ISSN (Online): 2347 - 4718*

HIGH SPEED PIPELINED DATA ENCRYPTION STANDARD WITH ERROR DETECTION AND CORRECTION

Praveenkumar[1], Pradeep[2] S R, Anupkumar[3]
[1, 2, 3]Department of VLSI and Embedded Systems
VTU RC-Gulbarga
[1]praveenkumarrc@yahoo.in
[2]pradeepsr09@gmail.com
[3]anupkumar.v.j@gmail.com

Abstract: The selective application of technological and related procedural safeguards is an important responsibility of every organization or industry in providing adequate security to its electronic data systems. The DES (algorithm) was developed at IBM and adopted by the National Bureau of Standards, and has successfully withstood all the attacks published so far in the open literature. This project specifies a cryptographic algorithm in order to protect sensitive data, increasing the speed by pipelining the architecture by A ZERO-DELAY DE-SYNCHRONIZATION MODEL. We also develop a error detection and correction mechanism with help of HAMMING CODES. To maintain the data confidentially and to protect it, we need to convert the data into different form that differs completely from input and then transmit it. That data has to be again decrypted (got back) at the receiver. The algorithm defines the steps needed to encrypt the data and also to decrypt it. This design is programmed in Verilog. By implementing triple DES, the security can be increased.

I. INTRODUCTION

Security is a prevalent concern in information and data systems of all types. One means of providing security in communications is through encryption. Encryption is the process of converting plain text unhidden to a cryptic text hidden by encryption algorithm like DES algorithm to secure data against data thieves. This process has another part where cryptic text needs to be decrypted by the same reverse algorithm on the other end to be understood in fig 1.

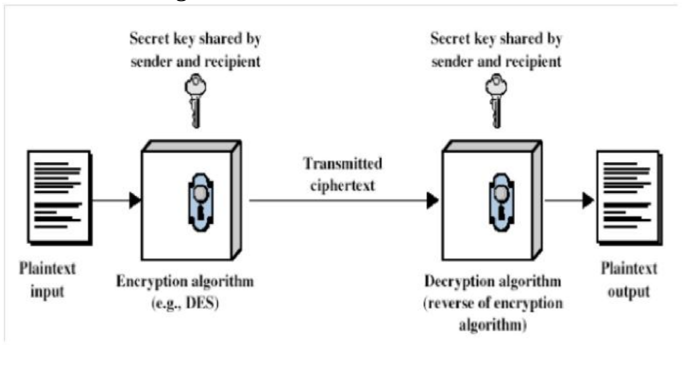

Secret key shared by sender and recipient
Secret key shared by sender and recipient
Transmitted ciphertext
Plaintext input
Encryption algorithm (e.g., DES)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

Fig. 1. The Simple block diagram of DES

There are two main types of cryptography in use today symmetric or secret key cryptography and asymmetric or public key cryptography. Secret key cryptography goes back to at least Egyptian times and is of concern here. It involves the use of only one key which is used for both encryption and decryption (hence the use of the term symmetric). It is necessary for security purposes that the secret key never be revealed.

II. THE DATA ENCRYPTION STANDARD (DES)

a. Brief history of DES

DES was the result of a research project set up by International Business Machines (IBM) Corporation in the late 1960s which resulted in a cipher known as LUCIFER. In the early 1970s it was decided to commercialize LUCIFER and a number of significant changes were introduced. IBM was not the only one involved in these changes as they sought technical advice from the National Security Agency (NSA) (other outside consultants were involved but it is likely that the NSA were the major contributors from a technical point of view). The altered version of LUCIFER was put forward as a proposal for the new national encryption standard requested by the National Bureau of Standards (NBS). It was finally adopted in 1977 as the Data Encryption Standard - DES [1].

b. Inner workings of DES

DES (and most of the other major symmetric ciphers) is based on a cipher known as the Feistel block cipher. This was a block cipher developed by the IBM cryptography researcher Horst Feistel in the early 70s. It consists of a number of rounds where each round contains bit-shuffling, non-linear substitutions (S-boxes) and exclusive OR operations. Most symmetric encryption schemes today are based on this structure (known as a feistel network). As with most encryption schemes, DES expects two inputs- the plaintext to be encrypted and the secret key[3]. The manner in which the plaintext is accepted, and the key arrangement used for encryption and decryption, both determine the type of cipher it is. DES is therefore a symmetric, 64 bit block cipher as it uses the same key for both encryption and decryption and only operates on 64 bit blocks of data at a time (be they plaintext or cipher text). The key size used is 56 bits, however a 64 bit (or eight-byte) key is actually input. The least significant bit of each byte is either used for parity (odd for DES) or set arbitrarily and does not increase the security in any way. All blocks are numbered from left to right which makes the eight bit of each byte the parity bit. Once a plain-text message is received to be encrypted, it is arranged into 64 bit blocks required for input. If the number of bits in the message is not evenly divisible by 64, then the last block will be padded. Multiple permutations and substitutions are incorporated throughout in order to increase the difficulty of performing a cryptanalysis on the cipher. However, it is generally accepted that the initial and final permutations offer little or no contribution to the security of DES and in fact some software implementations omit them (although strictly speaking these are not DES as they do not adhere to the standard). The algorithm is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64- bit key. Deciphering must be accomplished by using the same key as for enciphering, but with the schedule of addressing the key bits altered so that the deciphering process is the reverse of the enciphering process. A block to be enciphered is subjected to an initial permutation IP, then to a complex key- dependent computation and finally to a permutation which is the inverse of the initial permutation IP-1. The key-dependent computation can be simply defined in terms of a function f, called the cipher function, and a function KS, called the key schedule. A description of the computation is given first, along with details as to how the algorithm is used for encipherment. Next, the use of the algorithm for decipherment is described. Finally, a definition of the cipher function f is given in terms of primitive functions which are called the selection functions Si and the permutation function P.

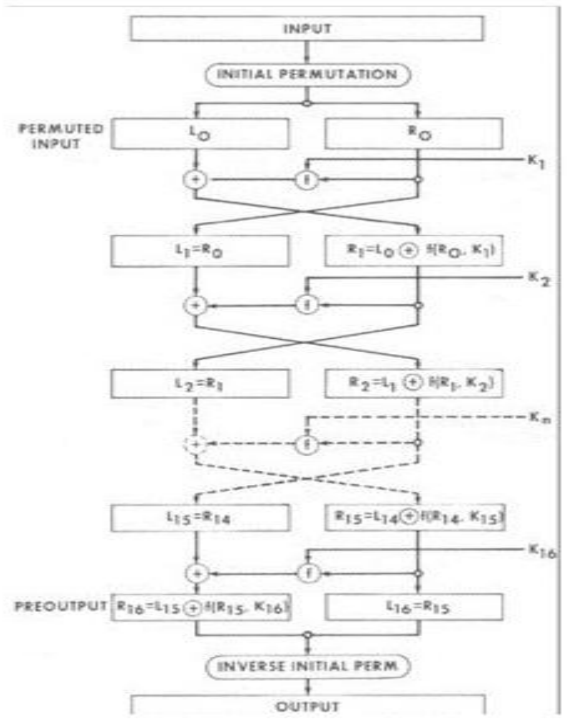
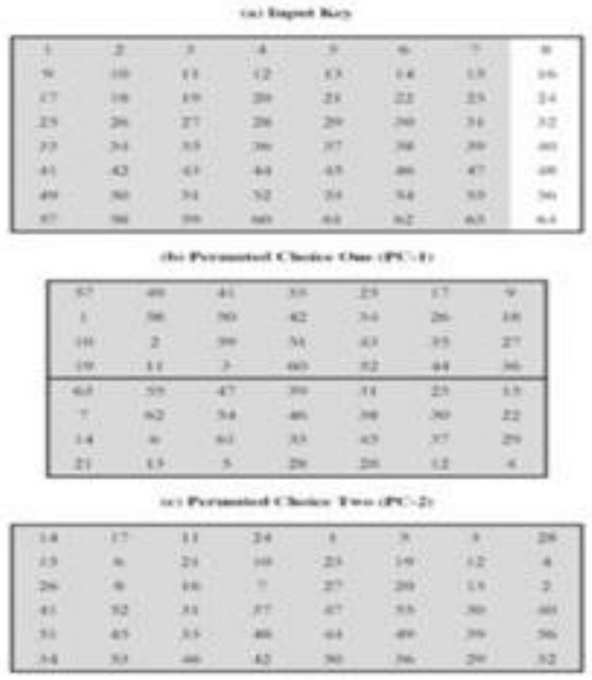


Fig. 2. Eciphering Computation

1. **Overall structure**

The following notation is convenient: Given two blocks L and R of bits, LR denotes the block consisting of the bits of L followed by the bits of R. Since concatenation is associative, B1B2...B8, for example, denotes the block consisting of the bits of B1 followed by the bits of B2...followed by the bits of B8.



(a) Input Key

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

(b) Permuted Choice One (PC-1)

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

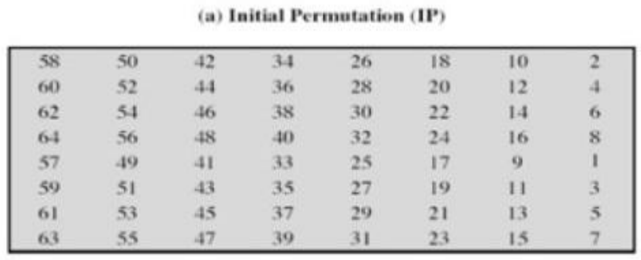
(c) Permuted Choice Two (PC-2)

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

Fig. 3. Permutation tables used in DES.

2. **Enciphering**

Initially the key is passed through a per mutation function (PC1 - defined in fig 3 b). For each of the 16 iterations, a sub key (Ki) is produced by a combination of a left circular shift and a permutation (PC2 - defined in fig 3 c) which is the same for each iteration. However, the resulting sub key is different for each iteration because of repeated shifts. The 64 bits of the input block to be enciphered are first subjected to the permutation, called the initial permutation IP of fig 4 a of DES key schedule. That is the permuted input has



(a) Initial Permutation (IP)

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

Fig. 4. Initial Permutation

bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit. The permuted input block is then the input to a complex key-dependent computation described above table. The output of that computation, called the preoutput, is then subjected to the permutation which is the inverse of the initial permutation that is as shown in Table (b) of DES key schedule. The computation which uses the permuted input block as its input to produce the preoutput block consists, but for a final interchanges of blocks, of 16 iterations of a calculation that is described below in terms of the cipher function f which operates on two blocks, one of 32 bits and one of 48bits, and produces a block of 32 bits. Let the 64 bits of the input block to an iteration consists of a 32 bit block L followed by a 32 bit block R. using the notation defined in the instruction, the input block is then LR. Let K be a block of 48 bits chosen from the 64-bit key. Then the output L'R' of iteration with input LR is defined by:

1) L' = R R' = L f (R, K) where d e n o t e s bit-by-bit addition modulo

2) As remarked before, the input of the first iteration of the calculation is the permuted input block. If L'R' is the output of the 16th iteration then R'L' is the preoutput block. At each iteration a different block K of key bits is chosen from the 64-bit key designated by KEY. With more notations we can describe the iterations of the computation in more detail. Let KS be a function which takes an integer n in the range from
1 to 16 and a 64-bit block KEY as input and yields as output a 48-bit block Kn which is a permuted selection of bits from KEY. That is (2) Kn = KS (n,KEY) with Kn determined by the bits in 48 distinct bit positions of KEY. KS is called the key schedule because the block K used in the n'th iteration of (1) is the block Kn determined by (2). As before, let the permuted input block be LR. Finally, let L() and R() be respectively L and R and let Ln and Rn be respectively L' and R' of (1) when L and R are respectively Ln-1 and Rn-1 and K is Kn; that is, when n is in the range from 1 to 16, (3) Ln = Rn-1 Rn = Ln-1 f(Rn-1,Kn) The preoutput block is then R16L16. The key schedule KS of the algorithm is described in detail. The key schedule produces the 16 Kn which are required for the algorithm.

3. Deciphering

The permutation IP-1 applied to the pre- output block is the inverse of the initial permutation IP applied to the input. Further, from (1) it follows that:
4) R = L' L= R' f(L',K) Consequently, to decipher it is only necessary to apply the very same algorithm to an enciphered message block, taking care that at each iteration of the computation the same block of key bits K is used during decipherment as was used during the encipherment of the block. Using the notation of the previous section, this can be expressed by the equations:
5) Rn-1 = Ln Ln-1 = Rn f(Ln,Kn) where now R16L16 is the permuted input block for the deciphering calculation and L0R0 is the preoutput block. That is, for the decipherment calculation with R16L16 as the permuted input, K16 is used in the first iteration, K15 in the second, and so on, with K1 used in the 16th iteration. The Cipher Function f A sketch of the calculation of f(R, K) is given in Figure of S box Let E denote a function which takes a block of 32 bits as input and yields a block of 48 bits as output. Let E be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the table Expansion permutation(c). Thus the first three bits of E(R) are the bits in positions 32, 1 and 2 of R while the last 2 bits of E(R) are the bits in positions 32 and 1. Each of the unique selection functions S1,S2,...,S8, takes a 6-bit block as input and yields a 4-bit block as output and is illustrated by using a table containing the recommended S1,S2,...,S8 : figure? If S1 is the function defined in this table and B is a block of 6 bits, then S1(B) is determined as follows: The

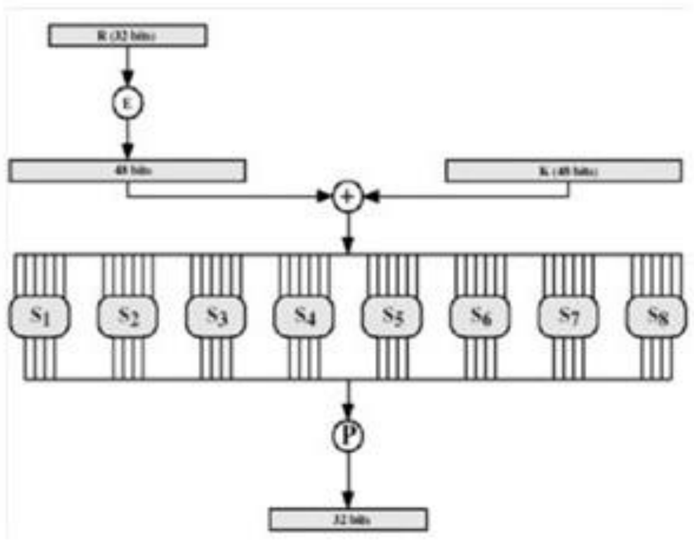


Fig. 5. Calculation of f(R, K)

first and last bits of B represent in base 2 a number in the range 0 to 3. Let that number be i. The middle 4 bits of B represent in base 2 a number in the range 0 to 15. 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 S1 (B) of S1 for the input B. For example, for input 011011 the row is 01, that is row 1, and the column is determined by 1101, that is column 13. In row 1 column 13 appears 5 so that the output is 0101. Selection functions S1, S2..., S8 of the algorithm appear as shown in the above table of S-box. The permutation function P yields a 32-bit output from a 32-bit input by permuting the bits of the input block. Such a function is defined by the table 2.1. The output P(L) for the function P defined by this table is obtained from the input L by taking the 16th bit of L as the first bit of P(L), the 7th bit as the second bit of P(L), and so on until the 25th bit of L is taken as the 32nd bit of P(L). The permutation function P of the algorithm is repeated. Now let S1,...,S8 be eight distinct selection functions, let P be the permutation function and let E be the function defined above.

6) B1B2...B8 = K E (R)

7) P (S1 (B1) S2 (B2)...S8 (B8))

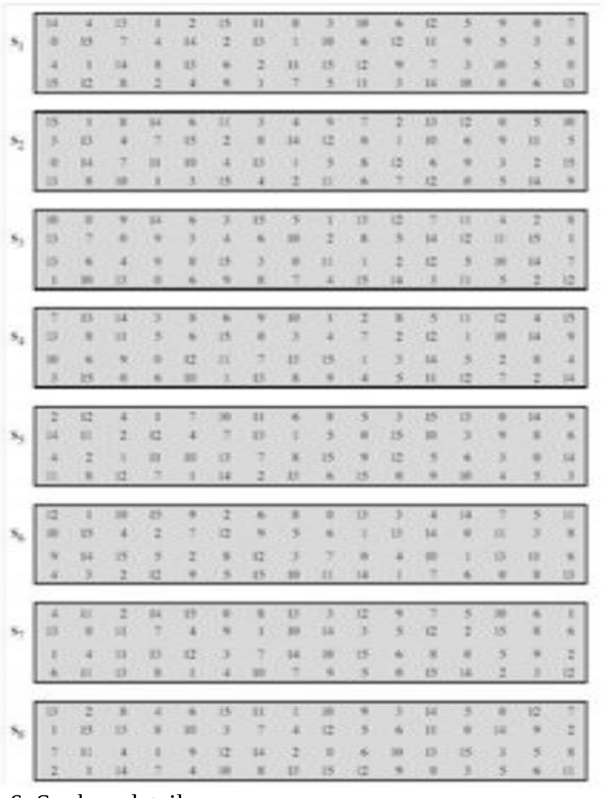
Thus K E(R) is first divided into the 8 blocks as indicated in (6). Then each Bi is taken as an input to Si and the 8 blocks S1 (B1), S2 (B2), S8 (B8) of 4 bits each are consolidated into a single block of 32 bits which forms the input to P. The output (7) is then the output of the function f for the inputs R and K.

III. KEY SCHEDULING

For example, C3 and D3 are obtained from C2 and D2, respectively, by two left shifts, and C16 and D16 are obtained from C15 and D15, 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, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1[5].

IV. OTHER POINTS OF NOTE

Having looked at DES in some detail a brief look at some other points is in order. These include decryption, modes of operation, security etc.



S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Fig. 6. S – box details

a. Modes of operation

The DES algorithm is a basic building block for providing data security. To apply DES in a variety of applications, five modes of operation have been defined which cover virtually all variation of use of the algorithm and these are shown in table

b. DES decryption

The decryption process with DES is essentially the same as the encryption process and is as follows: Use the cipher text as the input to the DES algorithm but use the keys Ki in reverse order. That is, use K16 on the first iteration, K15 on the second until K1 which is used on the 16th and last iteration.

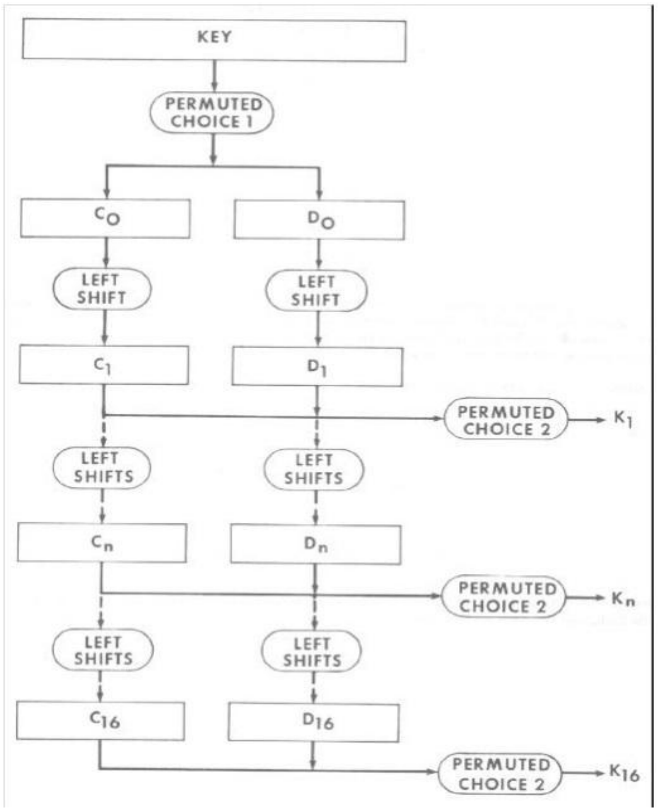
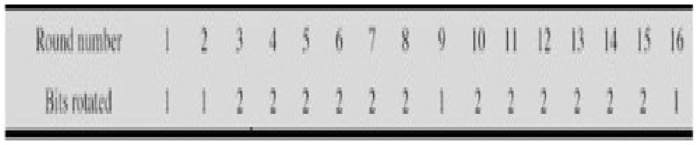


Fig. 7. Key scheduler calculation



Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Fig. 8. Iteration numbers with Numbers of shifts

c. Avalanche effect

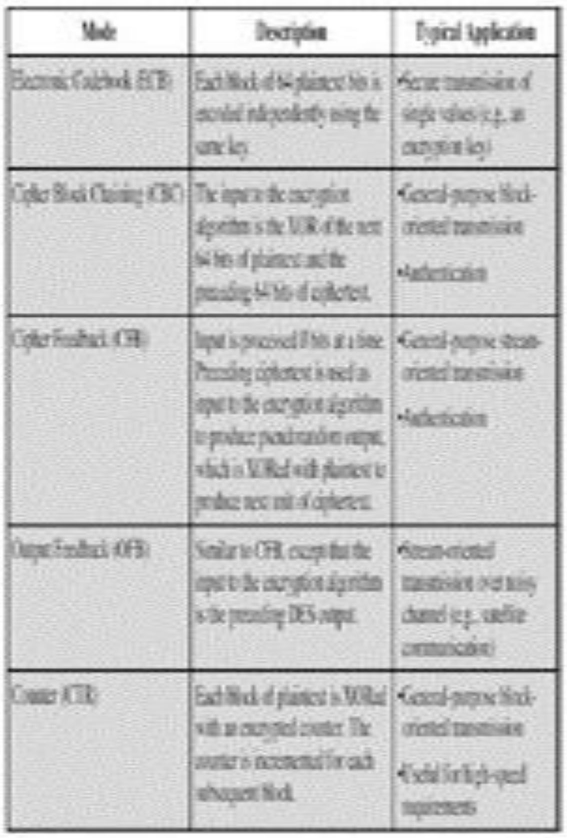
A desirable property of any encryption algorithm is that a small change in either plaintext or key should produce significant changes in the cipher text. DES exhibits a strong avalanche effect. Table 2.5 illustrates this.

V. PIPELINING THE DES ALGORITHM

Pipelining is wildly use method in large design for speed Enhancement. The iterative nature of the DES algorithm makes it ideally suited to pipelining and that can be 4, 6, 8 or 16 stages. The DES algorithm implementation presented in this paper is based on the ECB mode with 16 stages pipelining. Although the ECB mode is less secure than other modes of operation, it is commonly used and its operation can be pipelined[9] as shown in fig 9.

VI. SKEW CORE KEY-SCHEDULING

In the implementation of the DES algorithm key schedule employed here same as above stated. The direct mapping [6] of given key in required sub-keys but both ways using only wiring resources, So this part will be executed very fast and no optimizations would have any stage pipelined DES design and key-scheduling, it is necessary to control the time at which the sub-keys are effect. For maintaining proper synchronization in 16-available to each function f block. This is accomplished by the addition of a skew [6] that delays the individual sub keys by the required amount. The skew consists of 48 bits array of register shown in Fig 10. An outline of this key scheduling



Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key	•Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	•General-purpose block-oriented transmission •Authentication
Cipher Feedback (CFB)	Input is processed 8 bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	•General-purpose stream-oriented transmission •Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding DES output.	•Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	•General-purpose block-oriented transmission •Useful for high-speed requirements

Fig. 9. DES modes of operation

Method is provided in Fig.11, since the DES algorithm consists of 16 rounds, the skew core is set to loop 15 times since a register is not required to delay the first sub-key. The number of resister in each sub-key path same as plaintext passes the no. of register before reaching respective round as shown in Fig 11. It is noticeable that design of these registers same as used in Round blocks.

VII. A ZERO-DELAY DE-SYNCHRONIZATION MODEL

The de-synchronization model presented in this section aims at the substitution of the global clock by a set of asynchronous controllers that guarantee an equivalent behavior. The model assume that the circuit has combinational blocks (CL) and registers implemented with D flip-flops (FF), all of them working with the same clock edge (e.g. rising in Figure 14(a)). A. Steps in the de-synchronization method.

(a) Change in Plaintext			(b) Change in Key	
Round	Number of bits that differ		Round	Number of bits that differ
0	1		0	0
1	6		1	2
2	21		2	14
3	35		3	28
4	39		4	32
5	34		5	30
6	32		6	32
7	31		7	35
8	29		8	34
9	42		9	40
10	44		10	38
11	32		11	31
12	30		12	33
13	30		13	28
14	26		14	26
15	29		15	34
16	34		16	35

Fig. 10. Avalanche effect a small change in the plaintext produces a significant change in the ciphertext

The de-Synchronization method proceeds in three steps: 1) Conversion of the flip-flop-based synchronous circuit into latch-based one D-flip-flop is conceptually composed of master-slave latches. To perform de-synchronization, this internal structure is explicitly revealed (see Figure 14(b)) to: a) decouple local clocks for master and slave latches (in a D-flip-flop they are both derived from the same clock) and b) optionally improve performance through retiming, i.e. by moving latches across combinational logic. The conversion of a flip-flop-based circuit into a latch based one is not specific to the de-synchronization framework only. It is known to give an improvement in performance for synchronous systems [9] and, for this reason, it has a value by itself. 2) Generation of matched delays for the combinational logic (denoted by rounded rectangles in Figure 14(b)). Each matched delay must be greater than or equal to the delay of the critical path of the corresponding combinational block. Each matched delay serves as a completion detector for the corresponding combinational block. 3) Implementation of the local controllers. This is the main topic of this section.

VIII. ERROR CALCULATIONS

If exactly one bit error appears at the output state of the S-box and also in inverse S-box the presented fault detection scheme is able to detect it and error converges is about 100 percent. This is because in this case the error indication flag of the corresponding block alarms the fault. However due to the technological const0raints, single stuck –at fault may not be applicable for a mugger to gain more information. Thus, multiple bits will actually be flipped and hence multiple stuck-at errors are also considered in this paper covering both natural faults and fault attacks. For the calculation of the fault coverage for the multiple errors, Pi defines as the possibility of error detection in block in Figs 16 and 17. Then, the probability of not detecting the errors in block is (1-pi). For arbitrarily distributed errors in the S-box (respectively inverse S-box), this probability for each block is Independent of those of other blocks. Therefore, one can derive the equation for the error coverage of the randomly distributed errors as Where S is the set of the block numbers where the faults are injected. For randomly distributed errors, the error coverage for each block is Pi.

IX. HAMMING CODE

When data is transmitted from one location to another there is always the possibility that an error may occur. There are a number of reliable codes that can be used to encode data so that the error can be detected and corrected. With this project you will explore a simple error detection-correction technique called a Hamming Code. A Hamming Code can be used to detect and correct one-bit change in an encoded code word. This approach can be useful as a change in a single bit is more probable than a change in two bits or more bits.

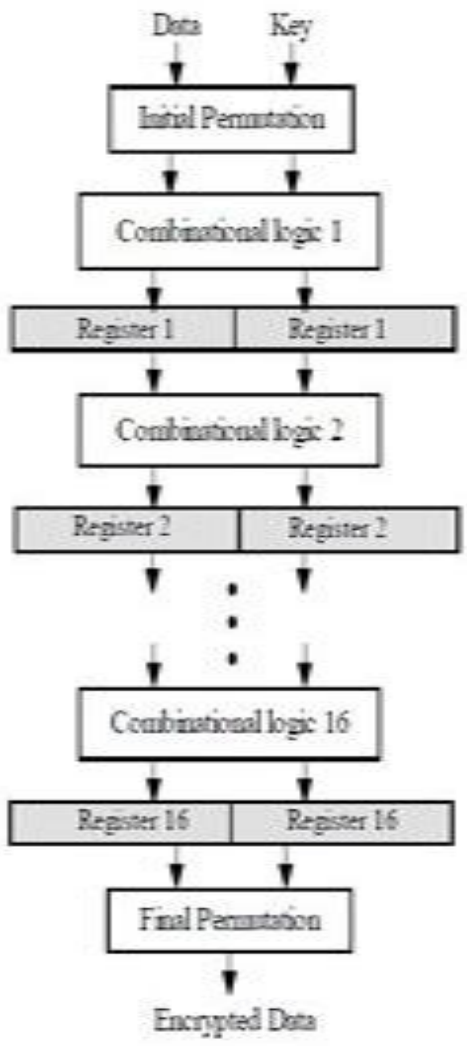


Fig. 11. DES pipeline Architecture

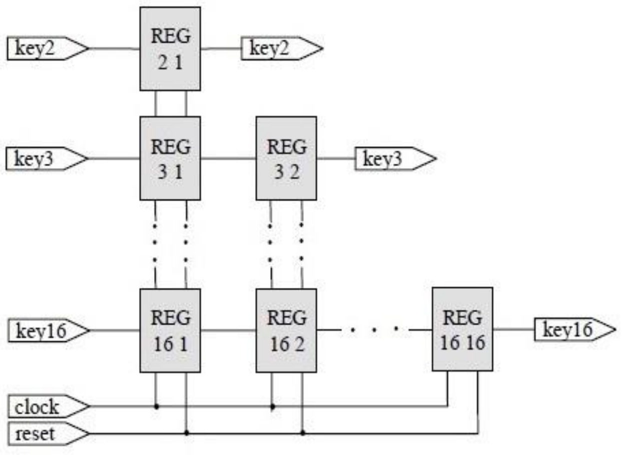


Fig. 12. Register arrays

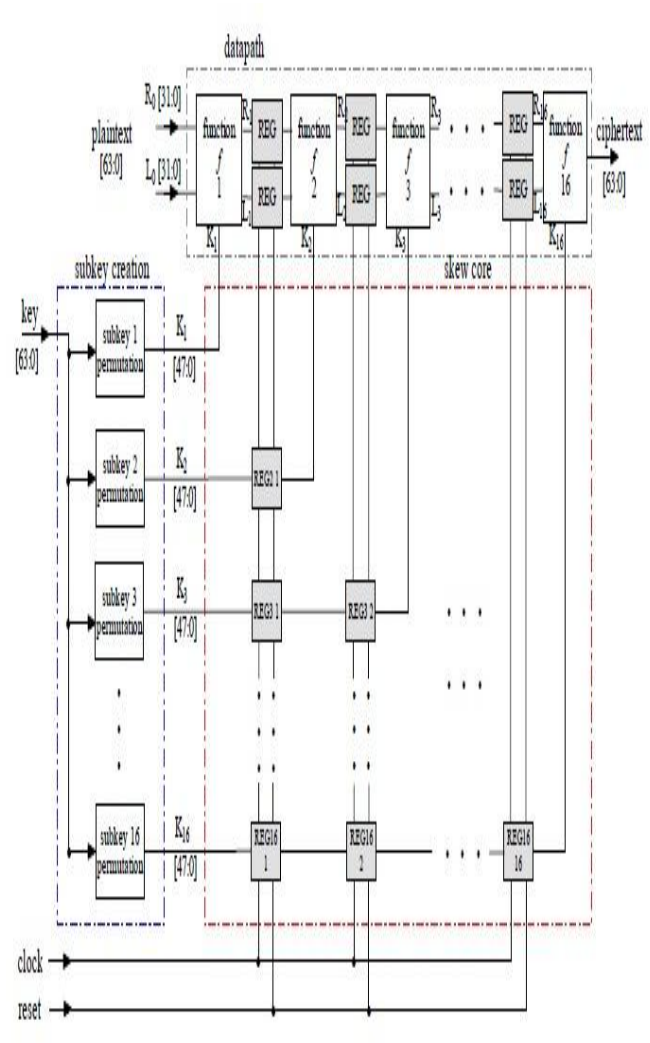


Fig. 13. Skew core Key-scheduling in DES design.

A. Calculating the Hamming Code

The key to the Hamming Code is the use of extra parity bits to allow the identification of a single error. Create the code word as follows: 1.Mark all bit positions that are powers of two as parity bits. (positions 1, 2, 4, 8, 16, 32, 64, etc.) 2. All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.) 3. Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips .Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1, 3,5,7,9,11,13,15...) Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (2,3,6,7,10,11,14,15,...) Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (4,5,6,7,12,13,14,15,20,21,22,23,...) Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15,24-31,40-47,...) Position 16: check 16 bits, skip 16 bits, check 16 bits, skip 16 bits, etc. (16-31,48-63,80-95,...) Position 32: check 32 bits, skip 32 bits, check 32 bits, skip 32 bits, etc. (32-63,96-127,160-191,...etc. 4.Set a parity bit to 1 if the total number of ones in the positions it checks is odd. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

$$EC\% = 100 \times \left(1 - \prod_{i \in S}(1-p_i)\right)\%$$

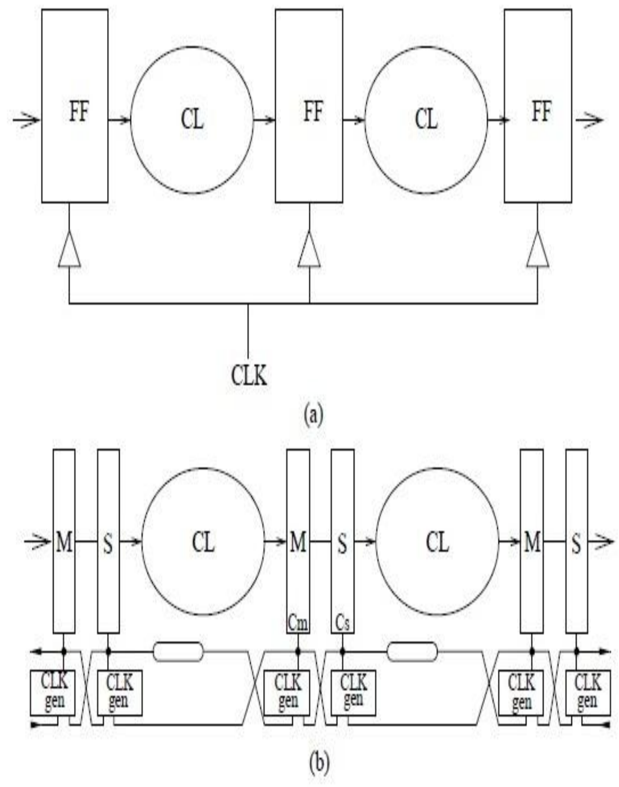


Fig. 14. (a) Synchronous circuit, (b) de-synchronized circuit.

X. APPLICATIONS

DES is utilized in various applications and environments. Cryptographic services are required across variety of platforms in a wide range of applications such as secure access to private networks, electronic commerce and health care. The security of conventional encryptions depends on several factors. DES can be used in intensive cryptographic computer application [4]. Applications such as electronic commerce, internet. bankings and electronic fund transfer, secure and private com- munication require better performance cryptographic system. The pipelined DES consumes less hardware resource than fully pipelined DES does, and provides more throughput than practical DES.

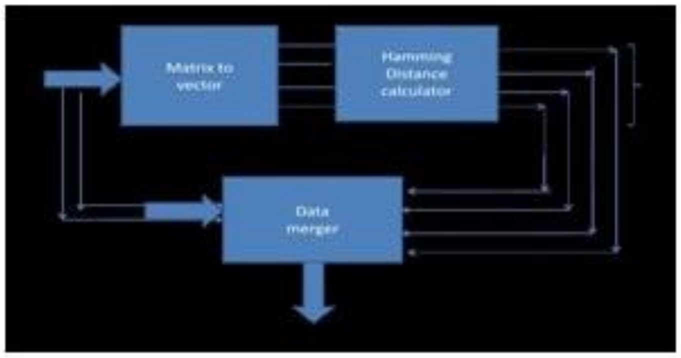


Fig. 16. Transmitter section

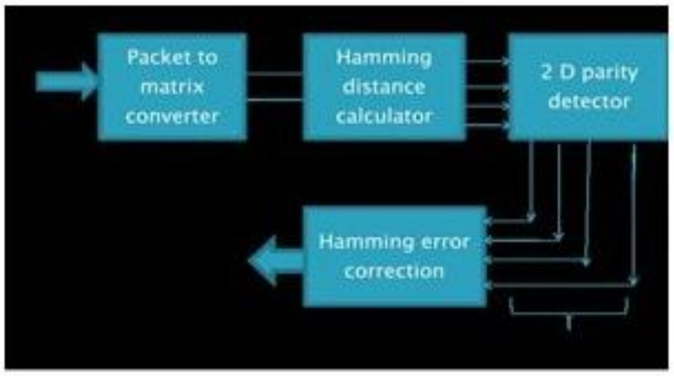
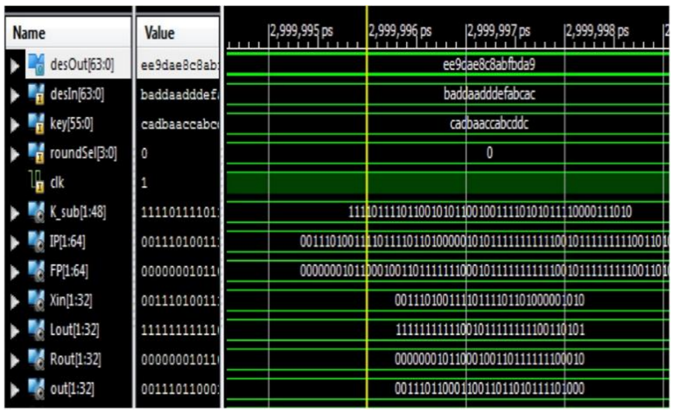


Fig. 17. Receiver section

The Triple DES consumes more hardware and also the frequency of operation is low compared to straight DES. But the basic advantage of using triple DES is that it is more secured than that of DES and pipelined DES.

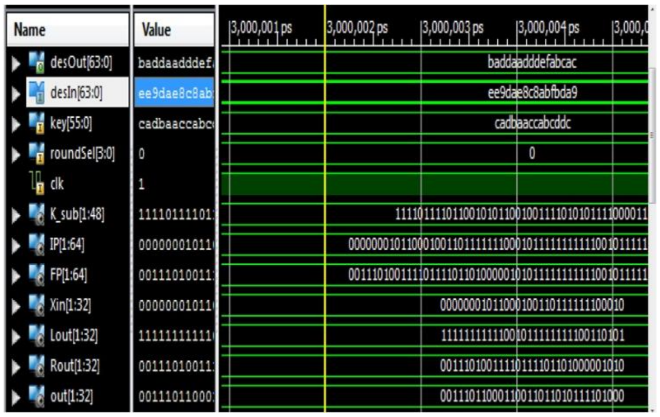
XI. RESULT

The encryption and decryption results are as shown below



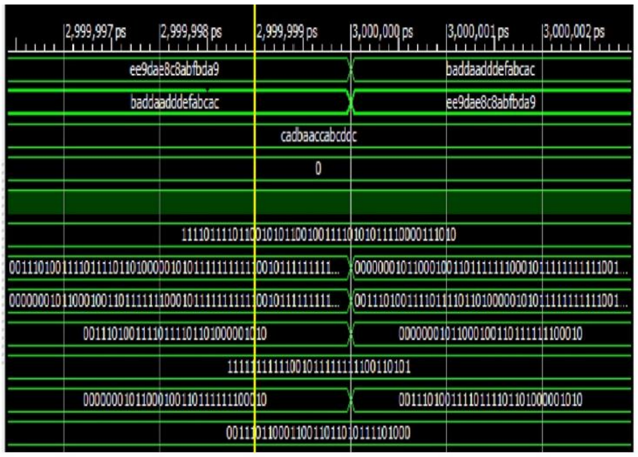
```
Name           Value          |2,999,995 ps  |2,999,996 ps  |2,999,997 ps  |2,999,998 ps
desOut[63:0]   ee9dae8c8ab    ee9dae8c8abfbda9
desIn[63:0]    baddaadddef    baddaadddefabcac
key[55:0]      cadbaaccabc    cadbaaccabcddc
roundSel[3:0]  0              0
clk            1
K_sub[1:48]    11110111101    111101111011001010110010011110101011110000111010
IP[1:64]       00111010011    00111010011110111101101000001010111111111111001011111111110011010
FP[1:64]       00000001011    00000001011000100110111111100010111111111111001011111111110011010
Xin[1:32]      00111010011    00111010011110111101101000001010
Lout[1:32]     11111111111    11111111111001011111111100110101
Rout[1:32]     00000001011    00000001011000100110111111100010
out[1:32]      00111011000    00111011000110011011010111101000
```

Fig. 18. Encryption



```
Name           Value          |3,000,001 ps  |3,000,002 ps  |3,000,003 ps  |3,000,004 ps  |3,000,0
desOut[63:0]   baddaadddef    baddaadddefabcac
desIn[63:0]    ee9dae8c8ab    ee9dae8c8abfbda9
key[55:0]      cadbaaccabc    cadbaaccabcddc
roundSel[3:0]  0              0
clk            1
K_sub[1:48]    11110111101    11110111101100101011001001111010101111000011
IP[1:64]       00000001011    000000010110001001101111111000101111111111110010111111
FP[1:64]       00111010011    001110100111101111011010000010101111111111110010111111
Xin[1:32]      00000001011    00000001011000100110111111100010
Lout[1:32]     11111111111    11111111111001011111111100110101
Rout[1:32]     00111010011    00111010011110111101101000001010
out[1:32]      00111011000    00111011000110011011010111101000
```

Fig. 19. Decryption



```
|2,999,997 ps  |2,999,998 ps  |2,999,999 ps  |3,000,000 ps  |3,000,001 ps  |3,000,002 ps
ee9dae8c8abfbda9                                          baddaadddefabcac
baddaadddefabcac                                          ee9dae8c8abfbda9
cadbaaccabcddc
0
111101111011001010110010011110101011110000111010
0011101001111011110110100000101011111111111001011111111...   000000010110001001101111111000101111111111111001...
0000000101100010011011111110001011111111111001011111111...   001110100111101111011010000010101111111111111001...
00111010011110111101101000001010                          00000001011000100110111111100010
11111111111001011111111100110101
00000001011000100110111111100010                          00111010011110111101101000001010
00111011000110011011010111101000
```

Fig.20. Both encryption and Decryption

REFERENCES:

[1] Data encryption standard (DES), National Bureau of Standards (U.S.), Federal Information Processing Standards Publication 46, National Technical Information Service , Springfield, VA, Apr. 1977.
[2] J. Orlin Grabbe. The DES Algorithm.
[3] Schneier, B. Applied Cryptography, Protocols, Algorithms, and Source Code , IEEE Int Symp circuits syst (ISCAS), 2005 pp.592-595,2003.
[4] Ahmed Zure Shameri, DES Cryptographic System for Information Security , IEEE Trans circuits syst vol.53, no.11,1165-1169, 2002.
[5] Kaps J, Fast DES implementations for FPGAs and its application to a Universal key-search machine. ,. In: Proc. 5th Annual Workshop on selected areas in cryptography.
[6] McLoone, M., McCanny, J, High-performance FPGA implementation of DES using a novel method for implementing the key schedule.
[7] J Wilcox, D., Pierson, L., Robertson, P., Witzke, E.L., Gass, K, A DES basic suitable for network encryption at 10 Gbs and

beyond ,.. In: CHESS 99, LNCS 1717 (1999) 3748
[8] Wong, K., Wark, M., Dawson, E, A Single-Chip FPGA Implementation of the Data Encryption Standard (des) Algorithm, In: IEEE Globecom Communication Conf., Sydney, Australia (2002) 827832.
[9] Biham, E A fast new DES implementation in software, Proc. 4th Int.Workshop on Fast software Encryption, FSE 97, Haifa, Israel, Jan. 1997 (Springer-Verlag, 1997), pp. 260 271.
[10] Wong, K., Wark, M., Dawson, E..: A Single- Chip FPGA Implementation of the Data Encryption Standard (des) Algorithm. , In: IEEE Globecom Communication Conf., Sydney, Australia (1998) 827832.
[11] J Wilcox, D., Pierson, L., Robertson, P., Witzke, E.L., Gass, K.: A DES asic suitable for network encryption at 10 Gbs and beyond, In: CHESS 99, LNCS 1717 (1999) 3748..
[12] Patterson, C. High Performance DES Encryption in Virtex FPGAs using Jbits. In: Field-programmable custom computing machines, In: FCCM00, Napa Valley, CA, USA, IEEE Computer. Soc., CA, USA, 2000 (2000) 11312.
[13] Wong, K., Wark, M., Dawson, E. .: A Single- Chip FPGA Implementation of the Data Encryption Standard (des) Algorithm, In: IEEE Globecom Communication Conf., Sydney, Australia (1998) 827832.