MANIPAL
ACADEMY of HIGHER EDUCATION
(Deemed to be University under Section 3 of the UGC Act, 1956)

Implementation of Hardware Accelerator for Cryptographic Applications

Mini-Project Mid Evaluation Report

submitted to

Manipal School of Information Sciences, MAHE, Manipal

Reg. Number	Name	Branch
201038004	RAGHAV K K	VLSI Design
201038005	SATISH NAYAK	VLSI Design

Under the guidance of

Dr. Shridhar Nayak
Assistant professor
Manipal School of Information Sciences,
MAHE, MANIPAL

10/06/2021

MANIPAL SCHOOL OF INFORMATION SCIENCES
MANIPAL
(A constituent unit of MAHE, Manipal)

DECLARATION

We declare that this mini project, entitled "**Implementation of Hardware Accelerator for Cryptographic Applications**" submitted for the evaluation of course work of Mini Project to Manipal School of Information Sciences, is our original work implemented by extending an existing idea of hardware accelerators, conducted under the supervision of our guide Dr. Shridhar Nayak, Assistant professor, MSIS, Manipal and panel members. References, help and material obtained from other sources have been duly acknowledged.

ABSTRACT

Hardware acceleration combines the flexibility of general-purpose processors, such as CPUs, with the efficiency of fully customized hardware, such as GPUs and ASICs, increasing efficiency by orders of magnitude when any application is implemented higher up the hierarchy of digital computing systems. In low power computing, they allow complex tasks such as computer vision or cryptography to be performed under a very tight power budget. Without a dedicated accelerator, these tasks would not be feasible especially in encryption algorithms such as DES (Data Encryption Standard). This is achieved by means of pipelining and parallelism of the design using Verilog.

Keywords

GPU, ASIC, cryptography, pipeline, Data Encryption Standard.

Contents

List of Figures

1. Introduction

Hardware acceleration is the use of computer hardware specially made to perform some functions more efficiently than is possible in software running on a general-purpose central processing unit. Graphics Processing Units (GPUs) originally designed for handling the motion of image, GPUs are now used for calculations involving massive amounts of data, accelerating portions of an application while the rest continues to run on the CPU.

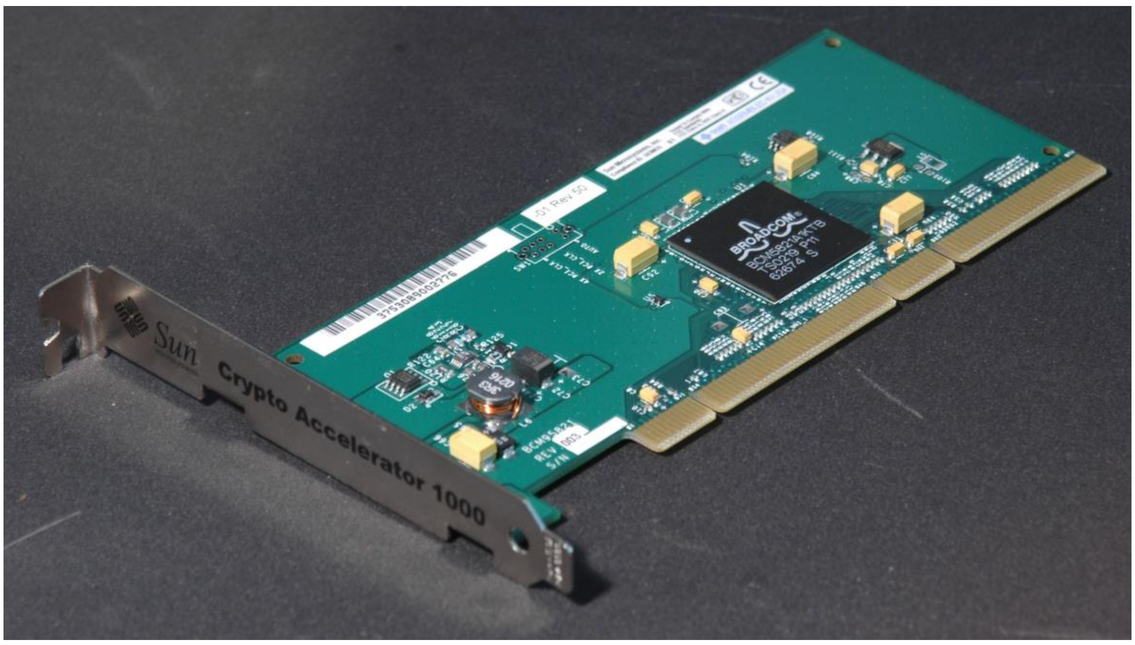


Figure 1. A Crypto Accelerator of SUN Corporation

The advent of contemporary tools such as field-programmable gate array (FPGAs) and application-specific integrated circuit (ASICs) have lifted the restriction of hardware acceleration to fully fixed algorithms, making hardware acceleration advantageous for a wider variety of common, graphically intensive tasks.

A Cryptographic Hardware Accelerator can be-

- Integrated into the SoC as a separate processor, as special purpose CPU (aka Core).
- Integrated in a Coprocessor on the circuit board
- Contained on a Chip on an extension circuit board, this can be connected to the mainboard via some BUS, e.g. PCI
- An ISA extension like e.g. AES instruction set and thus integral part of the CPU (in that case a kernel driver in not needed)

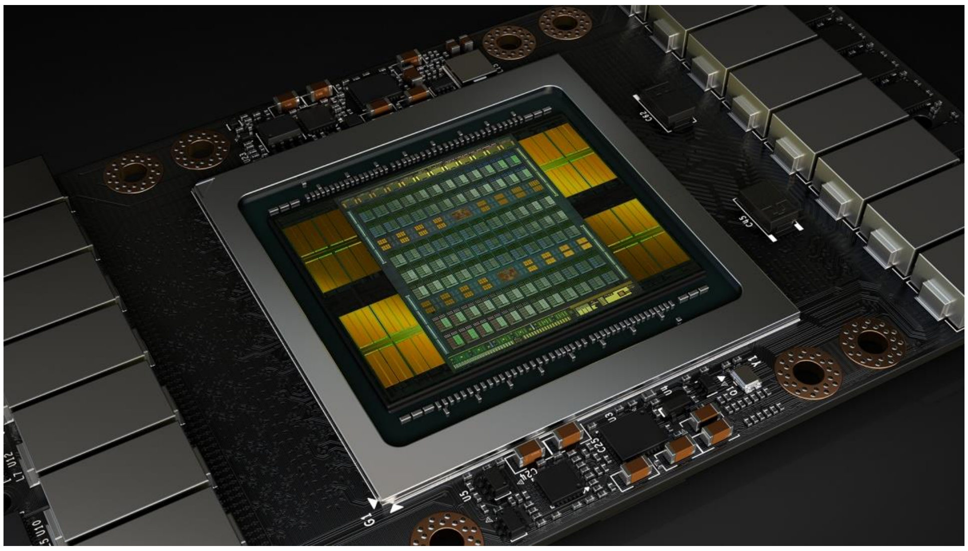


Figure 2. NVIDIA Tesla V100 GP-GPU

There is a wide variety of dedicated hardware acceleration systems. One popular form is tethering hardware acceleration, which, when acting as a WiFi hotspot, will offload operations involving tethering onto a WiFi chip, reducing system workload and increasing energy efficiency. Hardware graphics acceleration, also known as GPU rendering, works server-side using buffer caching and modern graphics APIs to deliver interactive visualizations of high-cardinality data. AI hardware acceleration is designed for such applications as artificial neural networks, machine vision, and machine learning hardware acceleration, often found in the fields of robotics and the Internet of Things.

Figure 3. Silicom’s Dual Hardware Accelerator (PE3IS2CO3)

The most common hardware used for acceleration include-

- Graphics Processing Units (GPUs): originally designed for handling the motion of image, GPUs are now used for calculations involving massive amounts of data, accelerating portions of an application while the rest continues to run on the CPU. The massive parallelism of modern GPUs allows users to process billions of records instantly.
- Field Programmable Gate Arrays (FPGAs): a hardware description language (HDL)-specified semiconductor integrated circuit designed to allow the user to configure a large majority of the electrical functionality. FPGAs can be used to accelerate parts of an algorithm, sharing part of the computation between the FPGA and a general-purpose processor.
- Application-Specific Integrated Circuits (ASICs): an integrated circuit customized specifically for a particular purpose or application, improving overall speed as it focuses solely on performing its one function. Maximum complexity in modern ASICs has grown to over 100 million logic gates.

Due to rapid development of internet and multimedia application, security is becoming an important issue to protect the ownership as well as integrity of information against third party attacks. Image encryption manipulates the conversion of an image to another one that is unseeable, while the decryption of an image is the process that retrieves the original image from the image that has been encrypted using appropriate password or key. Therefore, image encryption is among an ultimate solution in protecting the privacy of an image. Unlike text encryption, the image acts as the plaintext. Thus, specific algorithms for image encryption are needed. Few of the traditional cryptosystems are RSA, DES, and AES are used for encrypting.

Figure 4. Google's Cloud TPU (Tensor Processing Unit)

2. Material and methods

2.1 Architecture

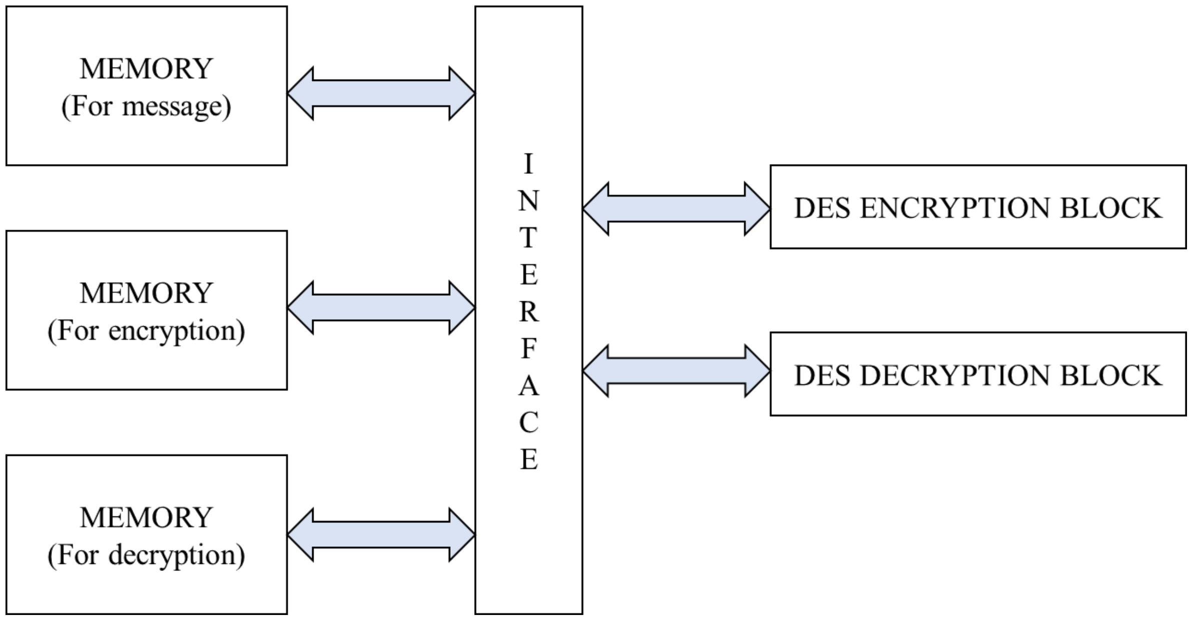


Figure 5. Hardware accelerator block diagram

The DES algorithm is a block cipher that uses the same binary key both to encrypt and decrypt data blocks, and thus is called a symmetric key cipher. DES operates on 64-bit plaintext data blocks, processing them under the control of a 56-bit key to produce 64 bits of encrypted ciphertext. Similarly, the DES decryption process operates on a 64-bit ciphertext block using the same 56-bit key to produce the original 64-bit plaintext block. DES uses a sequence of operations, including several substitution and permutation primitives, to encrypt a data block. These primitives are subsequently used to reverse the encryption operation.

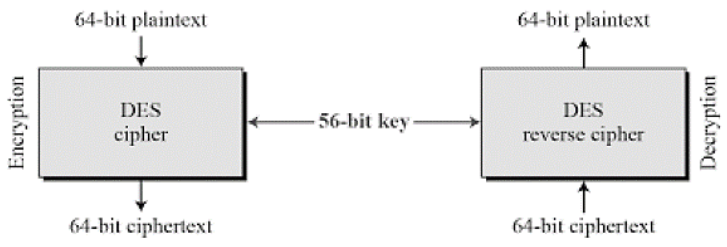


Figure 6. DES algorithm

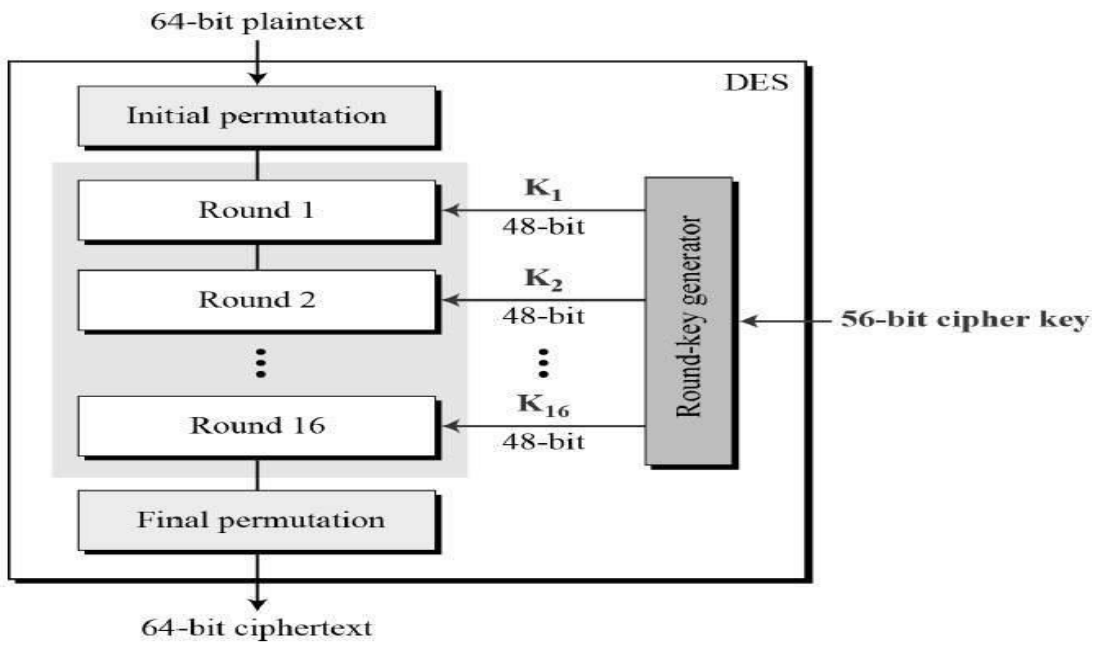


Figure 7. Block diagram of 64-bit encryption

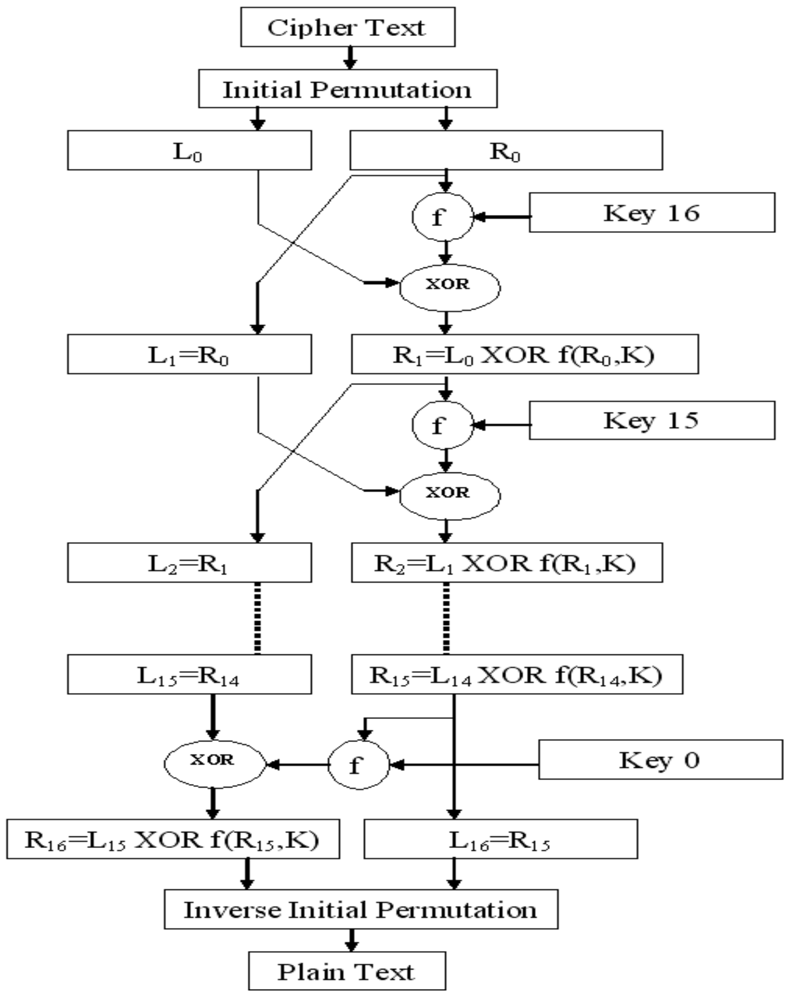


Figure 8. Block diagram of 64-bit decryption

It uses 16 rounds to ensure that the data are adequately scrambled to meet the security goals. The secret key is used to control the operation of the DES algorithm. Each key contains 56 bits of information, selected by each user to make the results of the encryption operations secret to that user. The message, encrypted data and decrypted data are each written and read from three different memory elements connected to each other by means of an interconnect. The process of encryption and decryption is sped up owing to the highly pipelined architecture used to implement it along with hardware parallelism for a wider range of concurrency.

DES Algorithm

The encryption process is made of two permutations (P-boxes), which we call initial and final permutations, and sixteen Feistel rounds. Each round uses a different 48-bit round key generated from the cipher key according to a predefined algorithm. Figure shows the elements of DES cipher at the encryption site.

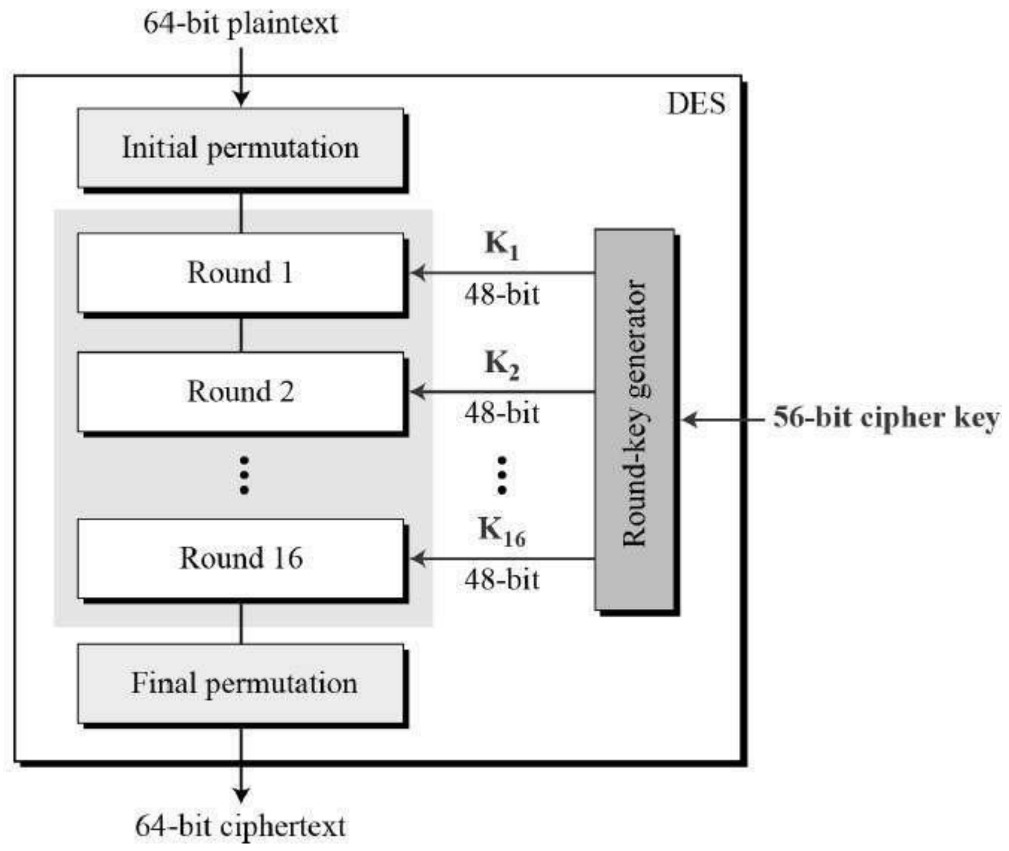


Figure 9. DES structure

Initial and Final Permutations

Figure shows the initial and final permutations (P-boxes). Each of these permutations takes a 64-bit input and permutes them according to a predefined rule. We have shown only a few input ports and the corresponding output ports. These permutations are keyless straight permutations that are the inverse of each other. For example, in the initial permutation, the 58th bit in the input becomes the first bit in the output. Similarly, in the final permutation, the first bit in the input becomes the 58th bit in the output. In other words, if the rounds between these two permutations do not exist, the 58th bit entering the initial permutation is the same as the 58th bit leaving the final permutation.

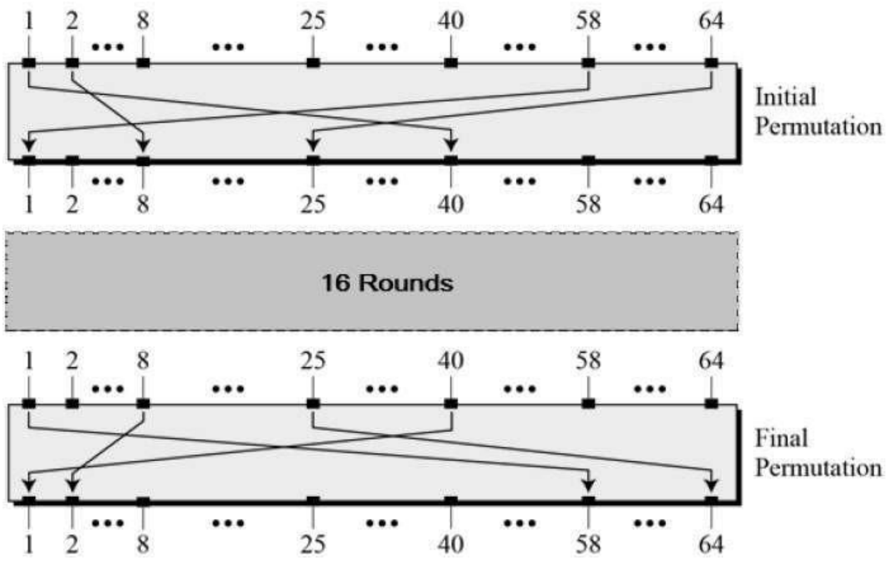
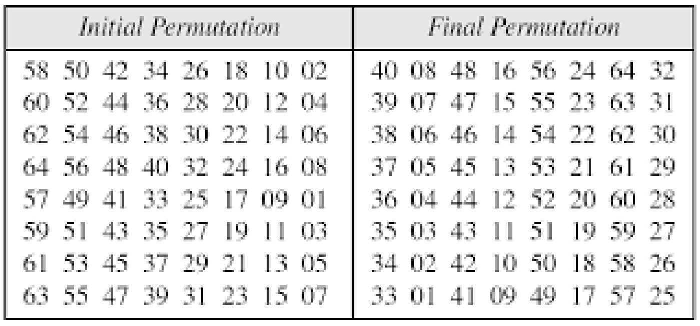


Figure 10. Initial and Final Permutations in DES

The permutation rules for these P-boxes are shown in Table. Each side of the table can be thought of as a 64-element array. Note that, as with any permutation table we have discussed so far, the value of each element defines the input port number, and the order (index) of the element defines the output port number.



Initial Permutation	*Final Permutation*
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

These two permutations have no cryptography significance in DES. Both permutations are keyless and predetermined. The reason they are included in DES is not clear and has not been revealed by the DES designers. The guess is that DES was designed to be implemented in hardware (on chips) and that these two complex permutations may thwart a software simulation of the mechanism.

FIESTAL Rounds

DES uses 16 rounds. Each round of DES is a Feistel cipher. The round takes LI−1 and RI−1 from previous round (or the initial permutation box) and creates LI and RI, which go to the next round (or final permutation box). We can assume that each round has two cipher elements (mixer and swapper). Each of these elements is invertible. The swapper is obviously invertible. It swaps the left half of the text with the right half. The mixer is invertible because of the XOR operation. All noninvertible elements are collected inside the function f (RI−1, KI).

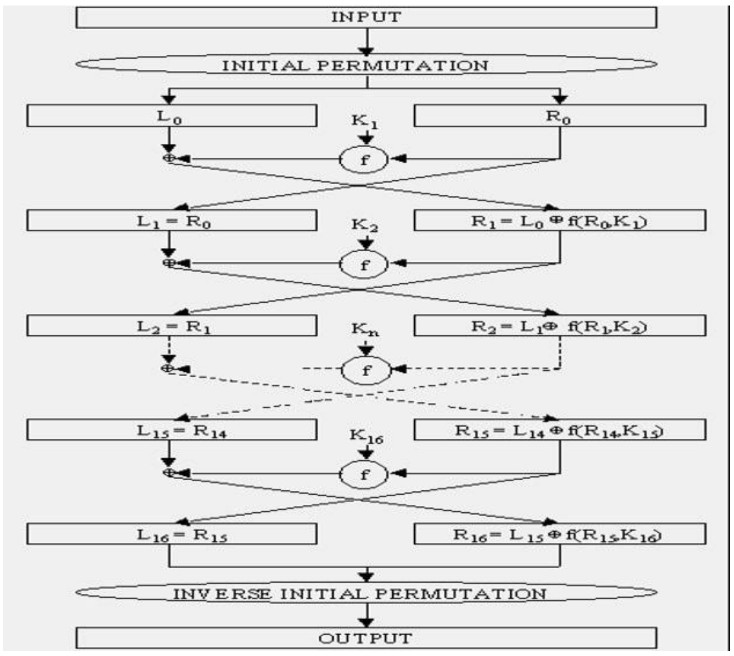

INPUT
INITIAL PERMUTATION
L_0
K_1
R_0
f
$L_1 = R_0$
K_2
$R_1 = L_0 \oplus f(R_0, K_1)$
f
$L_2 = R_1$
K_n
$R_2 = L_1 \oplus f(R_1, K_2)$
f
$L_{15} = R_{14}$
K_{16}
$R_{15} = L_{14} \oplus f(R_{14}, K_{15})$
f
$L_{16} = R_{15}$
$R_{16} = L_{15} \oplus f(R_{15}, K_{16})$
INVERSE INITIAL PERMUTATION
OUTPUT

Figure 11. DES Algorithm Flow of each round

DES Function

The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits (RI−1) to produce a 32-bit output. This function is made up of four sections: an expansion D-box, a whitener (that adds key), a group of S-boxes, and a straight D-box as shown below.

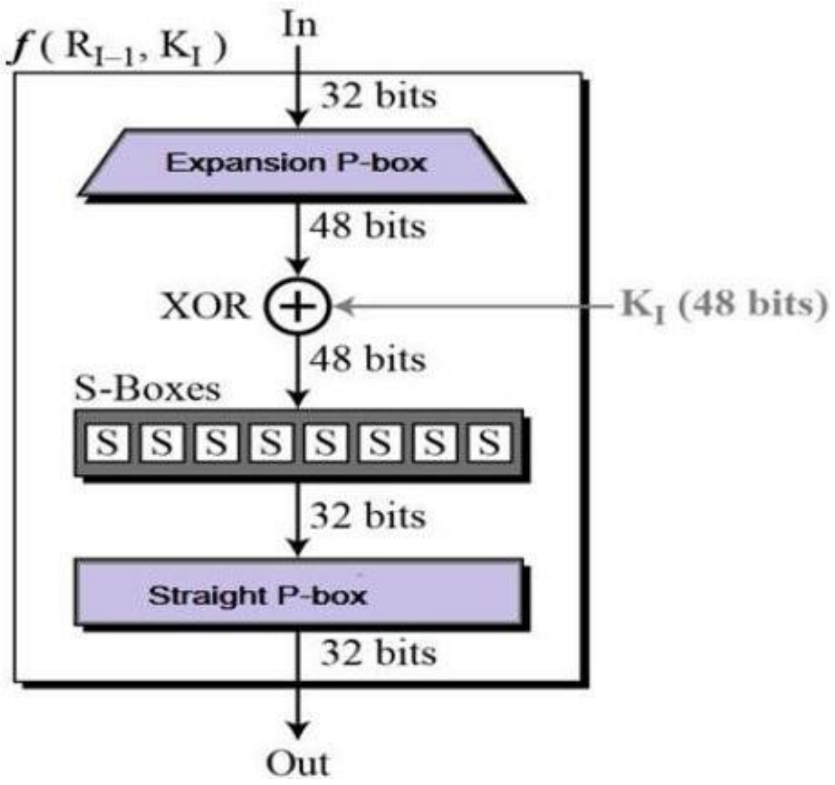


Figure 12. Expansion Box

Expansion D-box

Since RI−1 is a 32-bit input and KI is a 48-bit key, we first need to expand RI−1 to 48 bits. RI−1 is divided into 8 4-bit sections. Each 4-bit section is then expanded to 6 bits. This expansion permutation follows a predetermined rule. For each section, input bits 1, 2, 3, and 4 are copied to output bits 2, 3, 4, and 5, respectively. Output bit 1 comes from bit 4 of the previous section; output bit 6 comes from bit 1 of the next section. If sections 1 and 8 can be considered adjacent sections, the same rule applies to bits 1 and 32. The Fig. shown below shows the input and output in the expansion permutation.

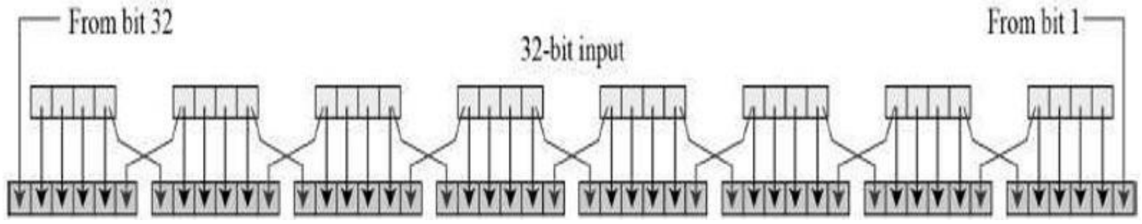
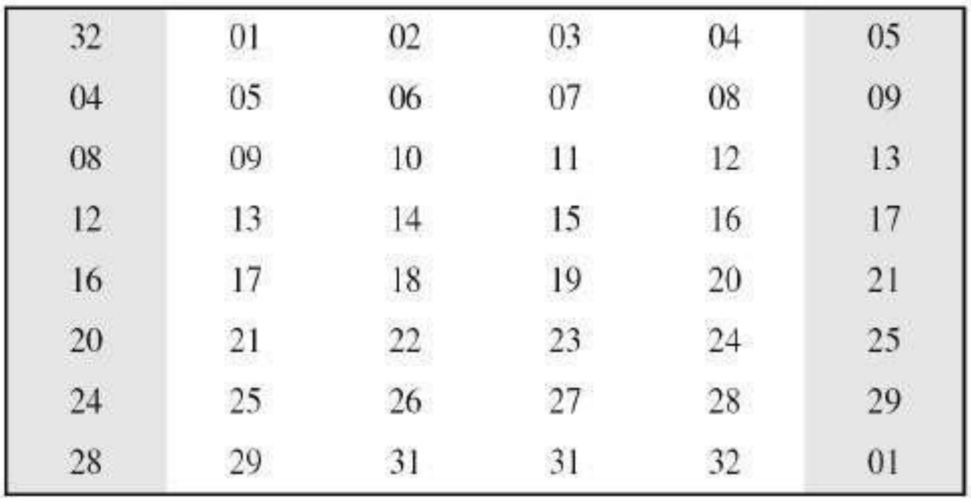


Figure 13. Expansion step

Although the relationship between the input and output can be defined mathematically, DES uses the Table shown below to define this D-box. Note that the number of output ports is 48, but the value range is only 1 to 32. Some of the inputs go to more than one output. For example, the value of input bit 5 becomes the value of output bits 6 and 8.



32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

Whitener (XOR)

After the expansion permutation, DES uses the XOR operation on the expanded right section and the round key. Note that both the right section and the key are 48-bits in length. Also note that the round key is used only in this operation.

S-Boxes

The S-boxes do the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. See Fig. below.

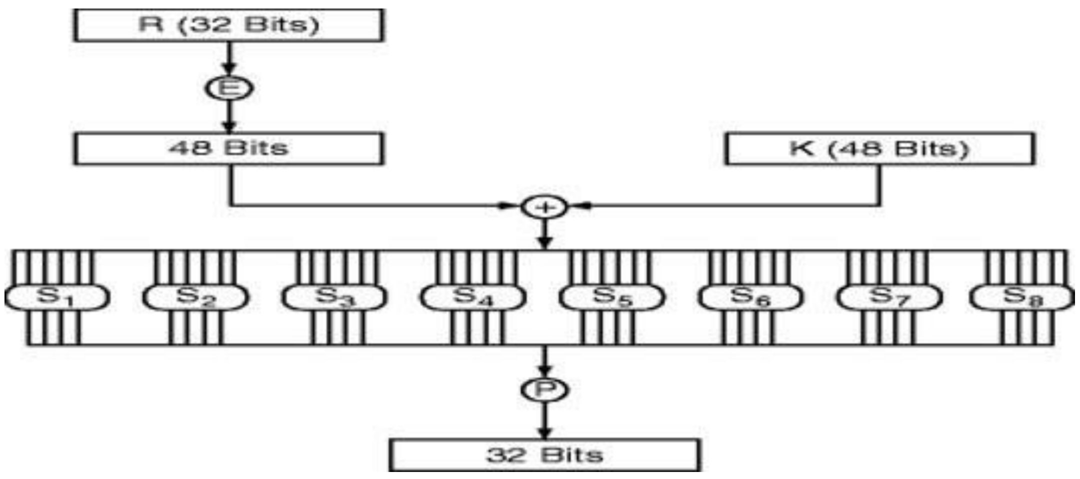


Figure 14. S-Boxes

After the expansion permutation, DES uses the XOR operation on the expanded right section and the round key. Note that both the right section and the key are 48-bits in length. Also note that the round key is used only in this operation.

S-Boxes

The S-boxes do the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. In Fig. below, the 48-bit data from the second operation is divided into eight 6-bit chunks, and each chunk is fed into a box. The result of each box is a 4-bit chunk; when these are combined the result is a 32-bit text. The substitution in each box follows a pre-determined rule based on a 4-row by 16-column table. The combination of bits 1 and 6 of the input defines one of four rows; the combination of bits 2 through 5 defines one of the sixteen columns as shown in Fig. Because each S-box has its own table, we need eight tables to define the output of these boxes. The values of the inputs (row number and column number) and the values of the outputs are given as decimal numbers to save space. These need to be changed to binary.

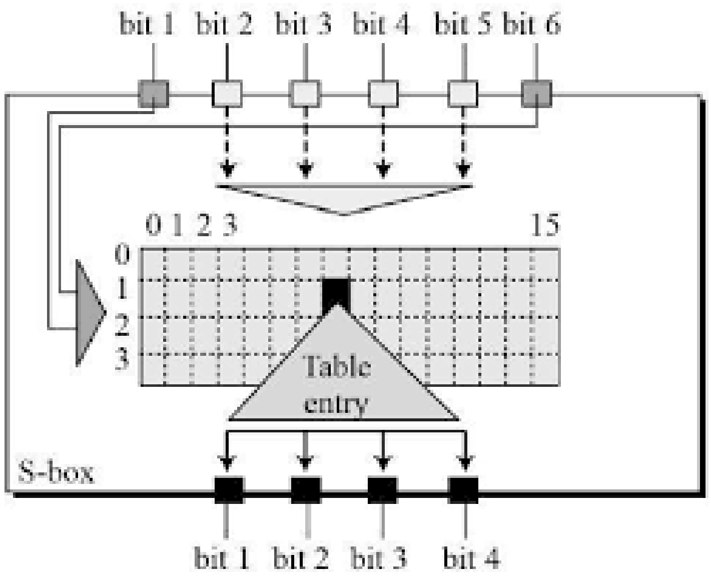
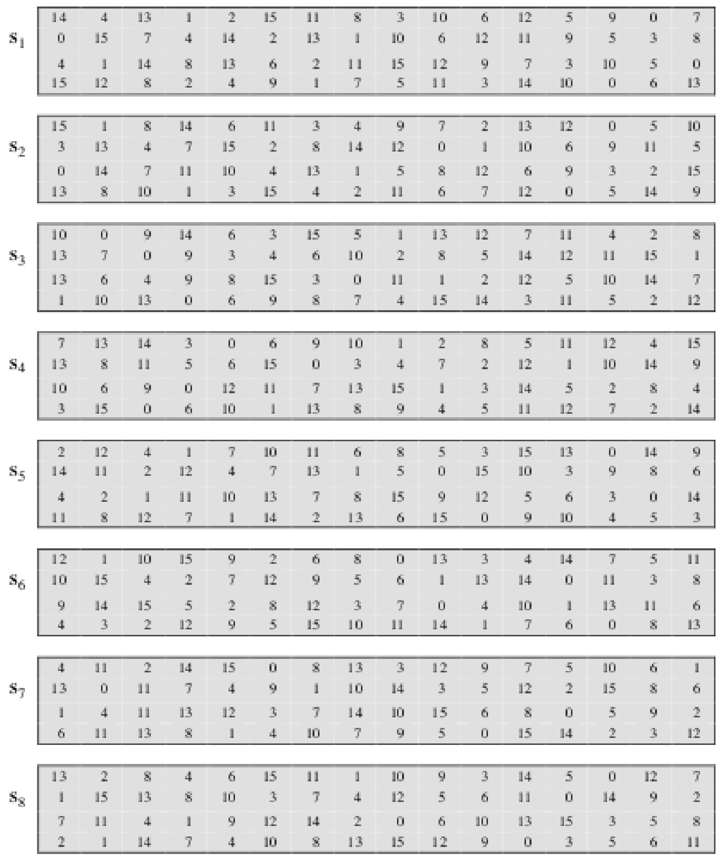


Figure 15. S-Box Calculation

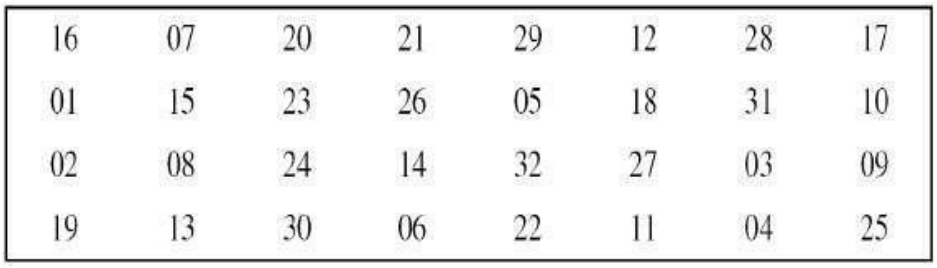


	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
S_1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
S_2	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11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

Figure 16. S-Box Lookup Table

Straight D-box:

The last operation in the DES function is a permutation with a 32-bit input and a 32-bit output. The input/output relationship for this operation is shown in Table below and follows the same general rule as previous tables. For example, the seventh bit of the input becomes the second bit of the output.



16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Cipher and reverse cipher:

Using mixers and swappers, we can create the cipher and reverse cipher, each having 16 rounds. The cipher is used at the encryption site; the reverse cipher is used at the decryption site. The whole idea is to make the cipher and the reverse cipher algorithms similar. To achieve this goal, one approach is to make the last round (round 16) different from the others; it has only a mixer and no swapper. Although the rounds are not aligned, the elements (mixer or swapper) are aligned. We know that a mixer is a self-inverse; so is a swapper. The final and initial permutations are also inversing of each other. The left section of the plaintext at the encryption site, L0, is enciphered as L16 at the encryption site; L16 at the decryption is deciphered as L0 at the decryption site. The situation is the same with R0 and R16.A very important point we need to remember about the ciphers is that the round keys (K1 to K16) should be applied in the reverse order. At the encryption site, round 1 uses K1 and round 16 uses K16; at the decryption site, round 1 uses K16 and round 16 uses K1.

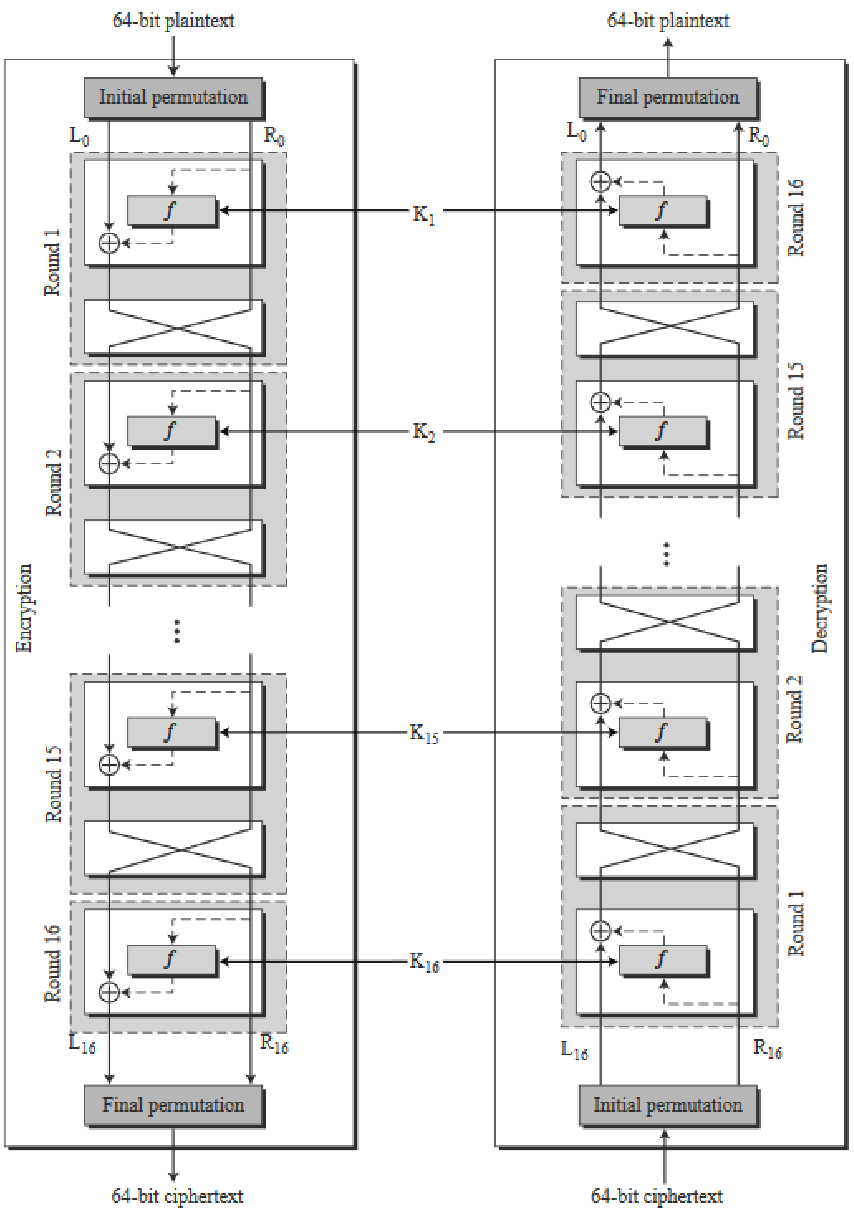


Figure 17. DES Encryption and Decryption

Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. However, the cipher key is normally given as a 64-bit key in which 8 extra bits are the parity bits, which are dropped before the actual key-generation process.

Parity Drop

The pre-process before key expansion is a compression transposition step that we call parity bit drop. It drops the parity bits (bits 8, 16, 24, 32, ..., 64) from the 64-bit key and permutes the rest of the bits. The remaining 56-bit value is the actual cipher key which is used to generate round keys. The parity drop step (a compression D-box) is shown in Table below.

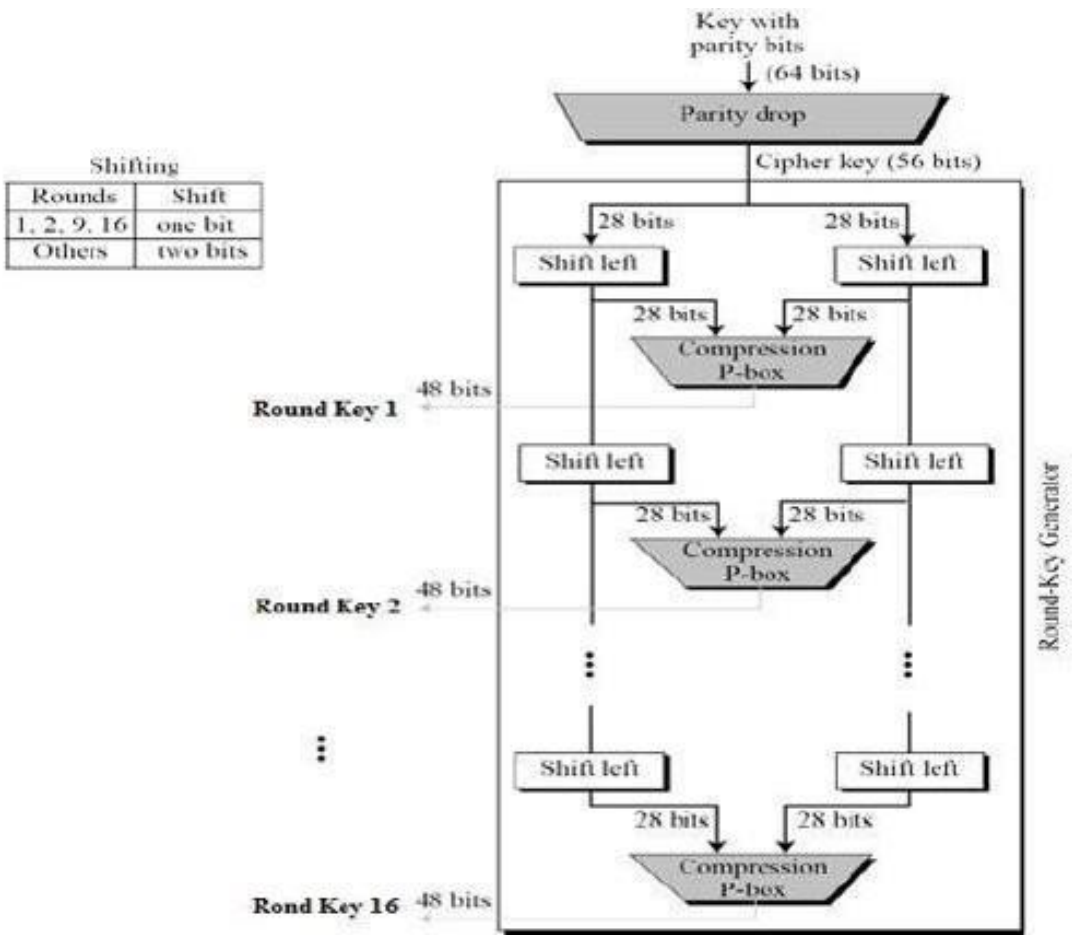
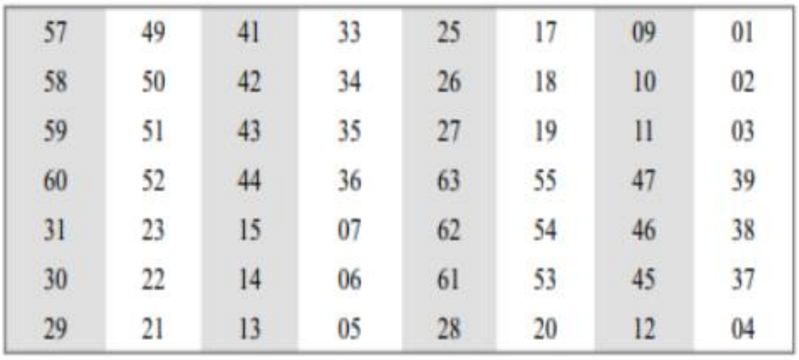


Figure 18. Key Generation

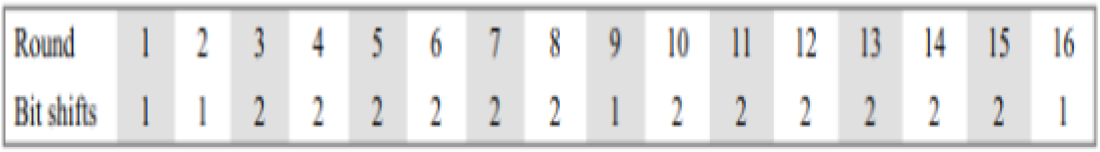


57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Figure 19. PC-1 Permuted Lookup Table

Shift Left

After the straight permutation, the key is divided into two 28-bit parts. Each part is shifted left (circular shift) one or two bits. In rounds 1, 2, 9, and 16, shifting is one bit; in the other rounds, it is two bits. The two parts are then combined to form a 56-bit part. Table below shows the number of shifts for each round.

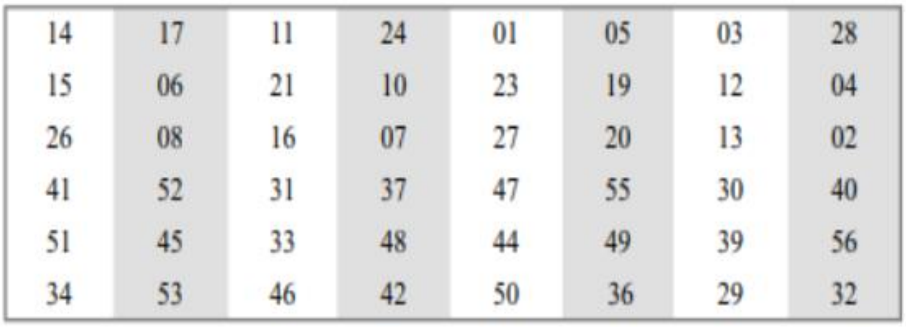


Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figure 20. Left Shift order for 16 Key blocks

Compression D-box

The compression D-box changes the 58 bits to 48 bits, which are used as a key for a round. The compression step is shown in Table below



14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Figure 21. PC-2 Permuted Lookup Table

2.2 Specifications

- 1GHz operating frequency
- 115 stage pipeline
- 64 bit data encryption & decryption
- 8KB memory

2.3 Implementation

The implementation is targeted towards creating a hardware accelerator for DES functionality of fixed bit size, memory functionalities to store the message, key and encrypted data, and communication via a simple single port dual channel RAM to connect each module.

2.3 Software & Hardware Requirements

1. **Verilog**
   Verilog, standardized as IEEE 1364, is a hardware description language (HDL) used to model electronic systems. It is most used in the design and verification of digital circuits at the register-transfer level of abstraction. It is also used in the verification of analog circuits and mixed-signal circuits, as well as in the design of genetic circuits. A Verilog design consists of a hierarchy of modules. Modules encapsulate design hierarchy and communicate with other modules through a set of declared input, output, and bidirectional ports. Internally, a module can contain any combination of the following: net/variable declarations (wire, reg, integer, etc.), concurrent and sequential statement blocks, and instances of other modules.

2. **Cadence NCSim**
   Incisive is a suite of tools from Cadence Design Systems related to the design and verification of ASICs, SoCs, and FPGAs. Incisive is commonly referred to by the name NCSim about the core simulation engine. In the late 1990s, the tool suite was known as LDV (logic design and verification). NCSim is a unified simulation engine for Verilog, VHDL, and System C. Loads snapshot images generated by NC Elaborator. This tool

can be run in GUI mode or batch command-line mode. In GUI mode, NCSim is like the debug features of Modelism's Vsim.

3. **HEX file conversion**

   A HEX file is a hexadecimal source file typically used by programmable logic devices, such as microcontrollers in remote controls, office machines, and automobile engine control systems. It contains settings, configuration information, or other data saved in a hexadecimal format. HEX files may be stored in either a binary or text format. The input to the block is bitmap image which must be converted to a hexadecimal code format. Since Verilog cannot read images directly in .bmp format we use Linux command to perform this conversion. The output of this block is the hex file which is then given to the System Verilog encryption blocks for processing.

3. Results

Extensive reading of literature, specifications and implementation methodology has been done. The implementation of the DES encryption, decryption and memory modules have been completed.

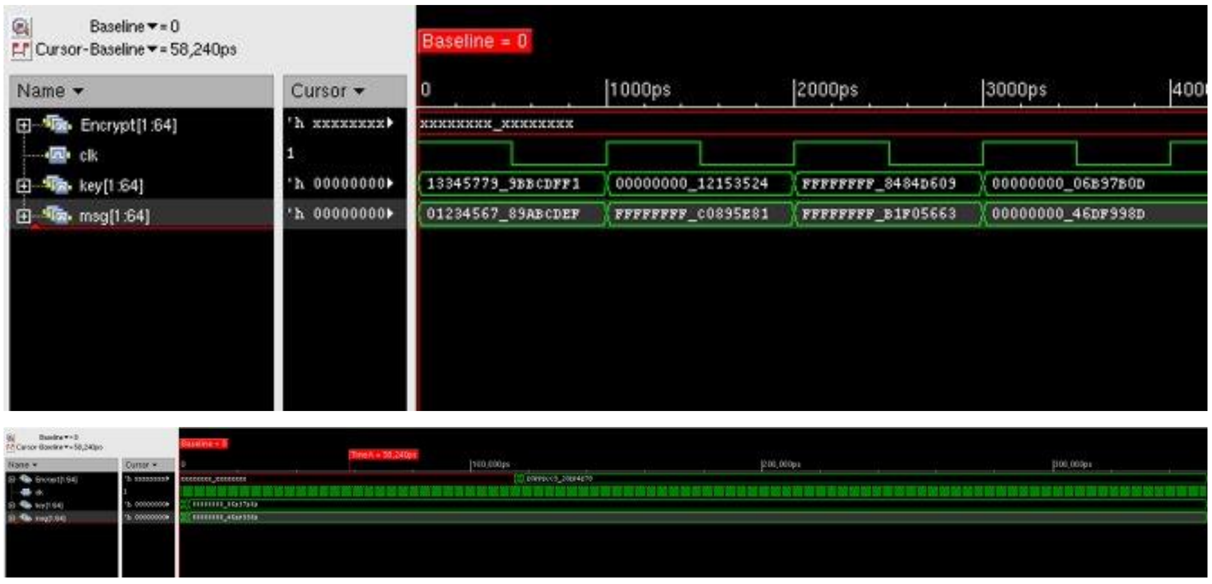


Figure 22. Message and Key inputs

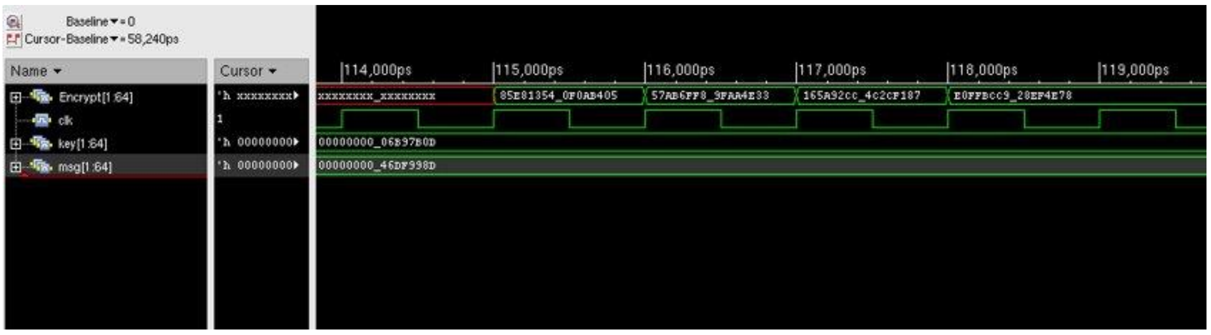


Figure 23. Encrypted message

Message = 0123456789ABCDEF: namely, Cipher = 85E813540F0AB405.

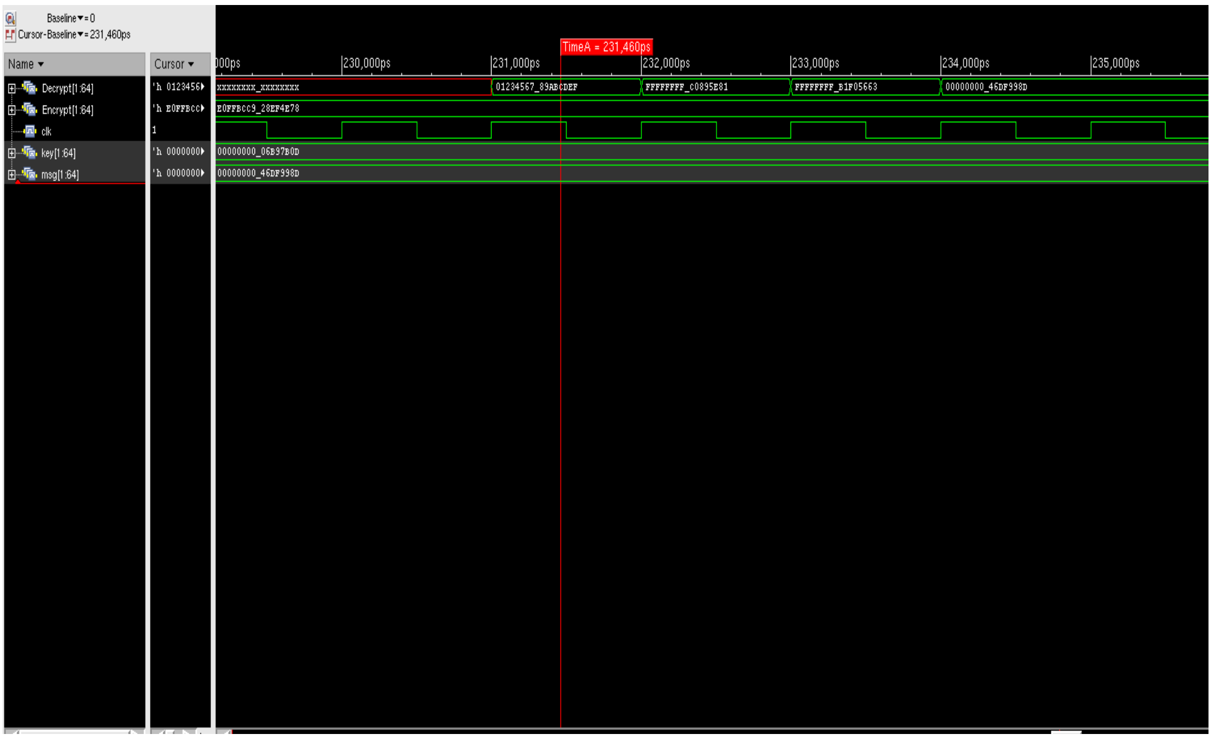


Figure 24. Decryption result

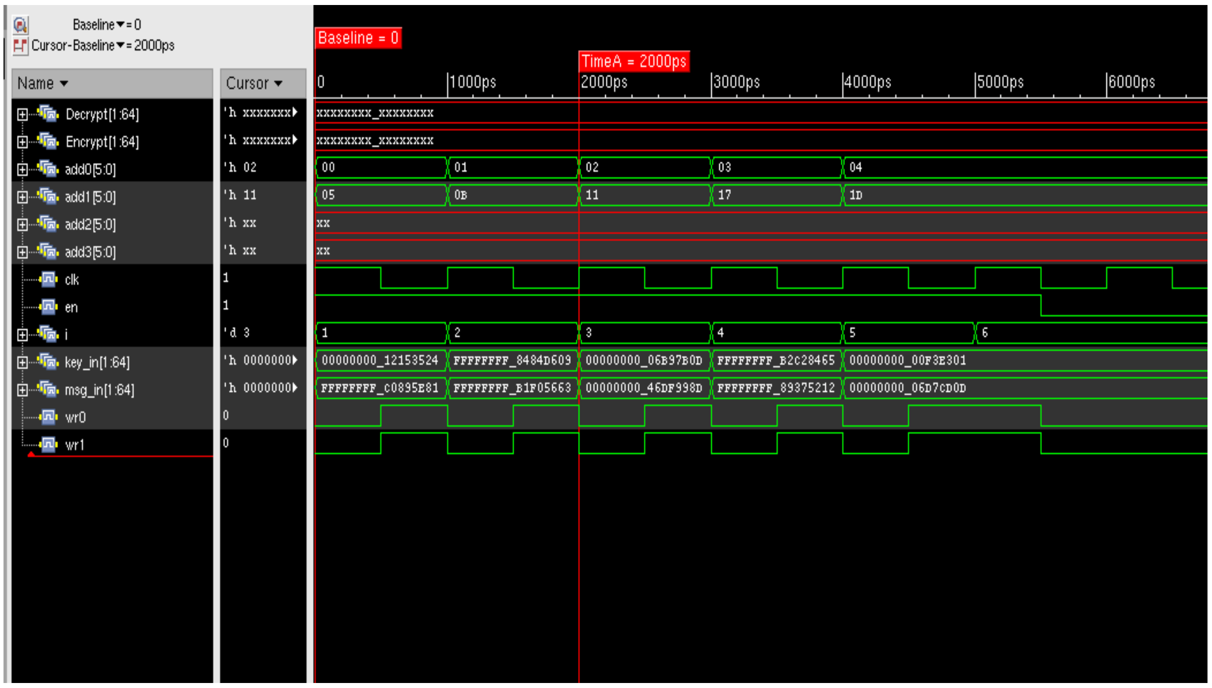


Figure 25. Memory operation

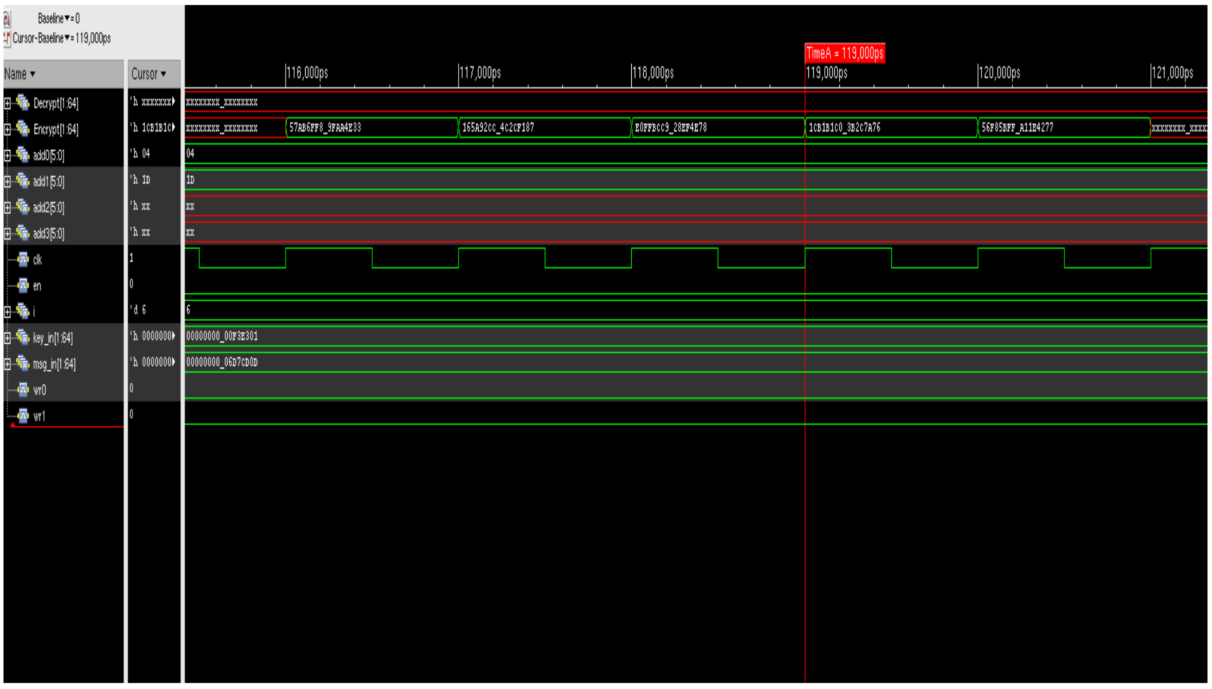


Figure 26. Encryption with memory integration

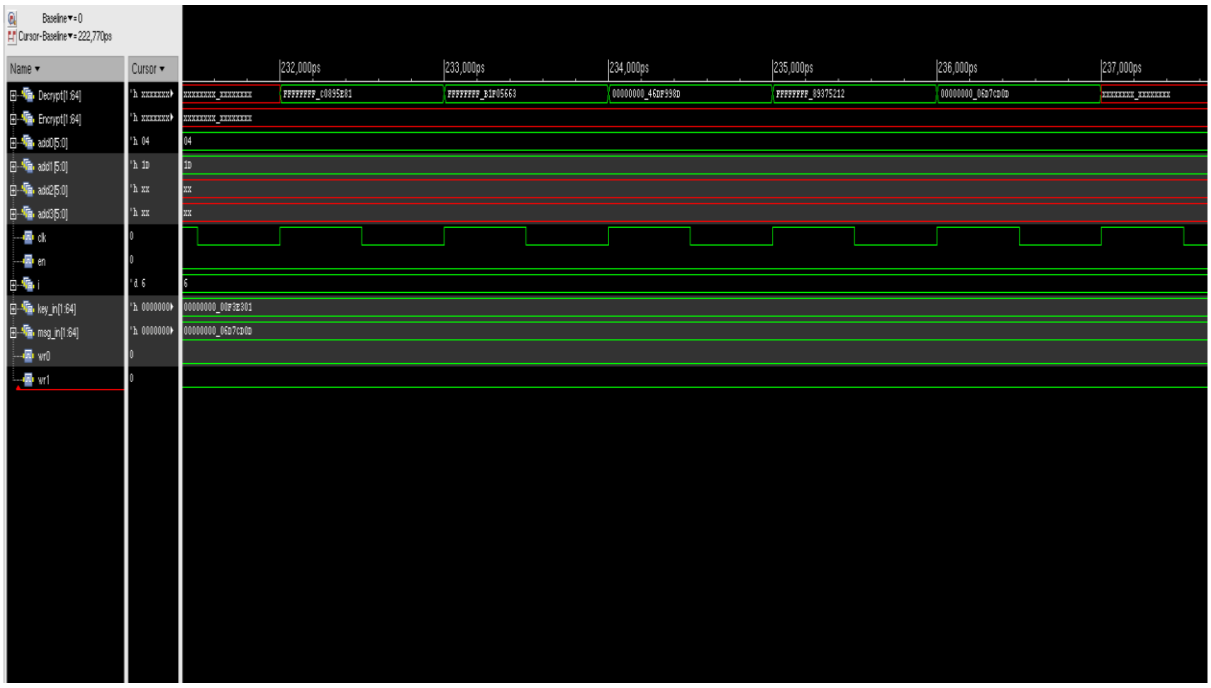


Figure 27. Decryption with memory integration

4. Discussion

Testing of the modelled architecture using a testbench and viewing the waveforms to ensure that the design specification and project objectives are met.

4.1. Challenges faced

- Understanding and finalizing of the specification from the wide variety of hardware accelerator implementations.
- Achieving pipelining of the encryption and decryption to get the output in a much shorter duration as compared to conventional DES hardware modules.
- Integrating the adequately sized memory to facilitate better performance of the desired system.
- Ensuring that the data is synchronized to ensure proper functioning of the encryption and decryption blocks.

5. Conclusions

- A Hardware Accelerator for Cryptographic Applications (DES) has been modelled using Verilog.

- Design has been done from scratch to simulate an efficient implementation of the algorithm with performance in focus.
- The high performance and low latency is attributed to the chosen methodology of a pipelined architecture.
- Utilizing this methodology, a 64-bit DES algorithm with an operating frequency of 1GHz is implemented.

6. Scope for further work

- Implementing a more streamlined memory unit.
- Upgrading current design to accommodate a robust encryption standard (i.e., AES 256).
- Achieving better hardware parallelism for further increase in performance.

7. References

[1] Design for Embedded Image Processing on FPGAs (Wiley) by Dr Donald Bailey, 1st edition book. Transaction Level Modeling with SystemC for System Level Design.

[2] https://oldwiki.archive.openwrt.org/doc/hardware/cryptographic.hardware.accelerat
ors

[3] http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm

[4] http://www.cs.cmu.edu/~djames/graphicsI/graphicsI_progHW_slides-1up.pdf

[5] https://en.wikibooks.org/wiki/Microprocessor_Design/Pipelined_Processors