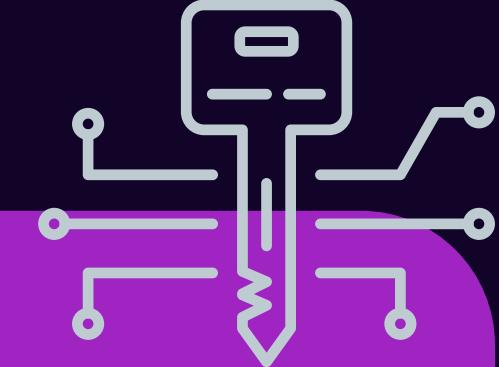


DATA ENCRYPTION STANDARD

Group PA-11



members of The Group



Aliya Rizqiningrum Salamun	2306161813
Anthonius Hendy Wirawan	2306161795
Axel Adrial Pazal Kalembang	2306161984
Jeremy Wijanarko Mulyono	2306267132



Background

Data Encryption Standard (DES) adalah algoritma kriptografi sejarah yang dikembangkan pada awal 1970-an oleh National Bureau of Standards (sekarang NIST) di Amerika Serikat. DES dirancang untuk melindungi data dengan menggunakan cipher blok berukuran 64-bit dengan ukuran kunci 56-bit. Serta menggunakan kombinasi operasi logika biner, permutasi bit, dan rotasi untuk proses enkripsi dan dekripsi.

Dalam era digital yang berkembang pesat, keamanan data menjadi krusial untuk melindungi informasi dari ancaman seperti pencurian atau manipulasi. DES yang merupakan salah satu algoritma enkripsi klasik dirancang untuk mengenkripsi data dengan menyandikannya ke dalam format yang tidak dapat dibaca tanpa kunci dekripsi yang tepat.

Proyek ini dipilih karena relevansinya yang terus bertahan sebagai salah satu algoritma enkripsi yang paling banyak digunakan, serta kemampuannya untuk diterapkan dalam berbagai aplikasi sistem keamanan data digital.

Description

Proyek ini bertujuan untuk mengimplementasikan DES menggunakan pendekatan berbasis perangkat keras dengan bahasa pemrograman VHDL. Desain ini tidak hanya mampu mengenkripsi data, tetapi juga mendekripsinya melalui penerapan Finite State Machine (FSM). Program ini dilengkapi dengan fitur untuk read dan write data dari file teks, memungkinkan pengguna untuk mengelola input data dan menyimpan hasil enkripsi dan dekripsi dengan mudah.

Objectives



Menerapkan algoritma DES menggunakan VHDL pada perangkat keras untuk enkripsi dan dekripsi.

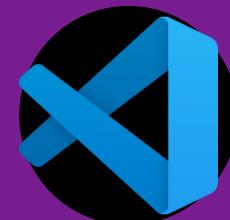


Menjelaskan teknik kriptografi seperti permutasi dan substitusi data.



Menyediakan fitur read-write file teks untuk pengelolaan data enkripsi.

Equipment



Visual Studio Code



ModelSim

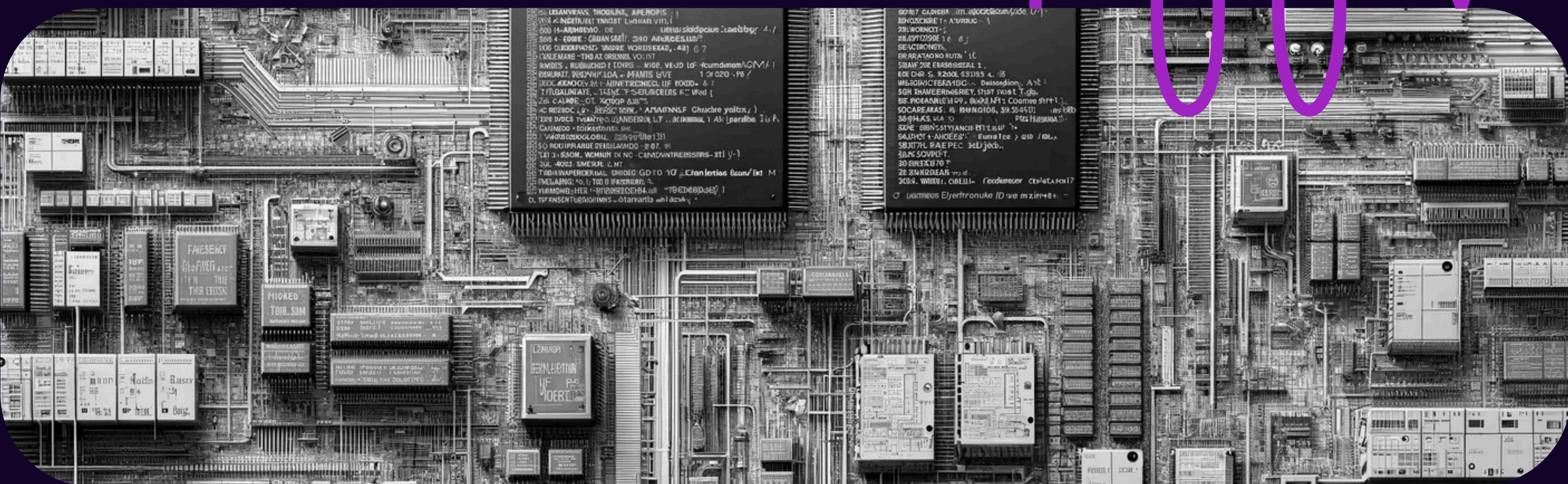


Quartus



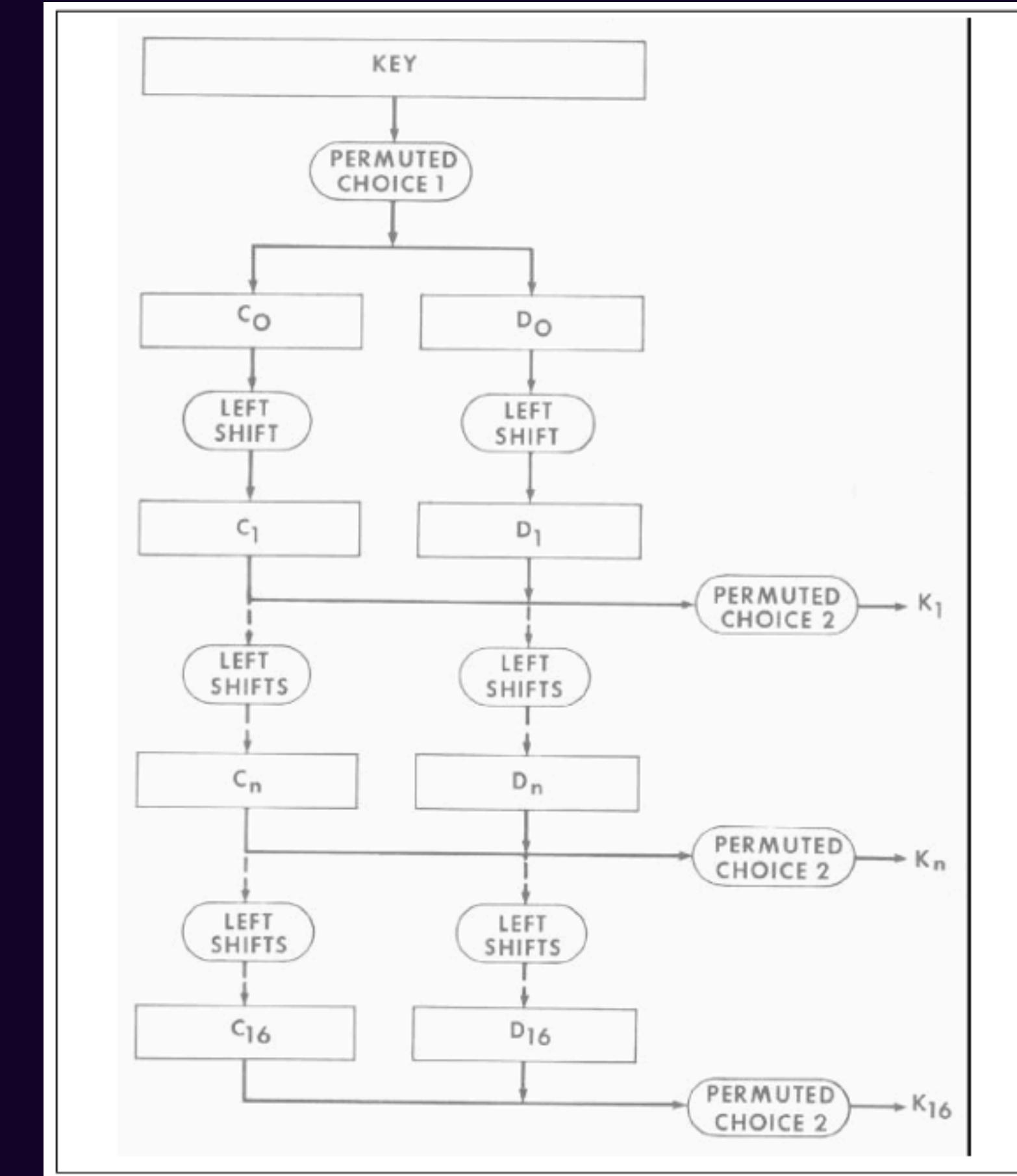
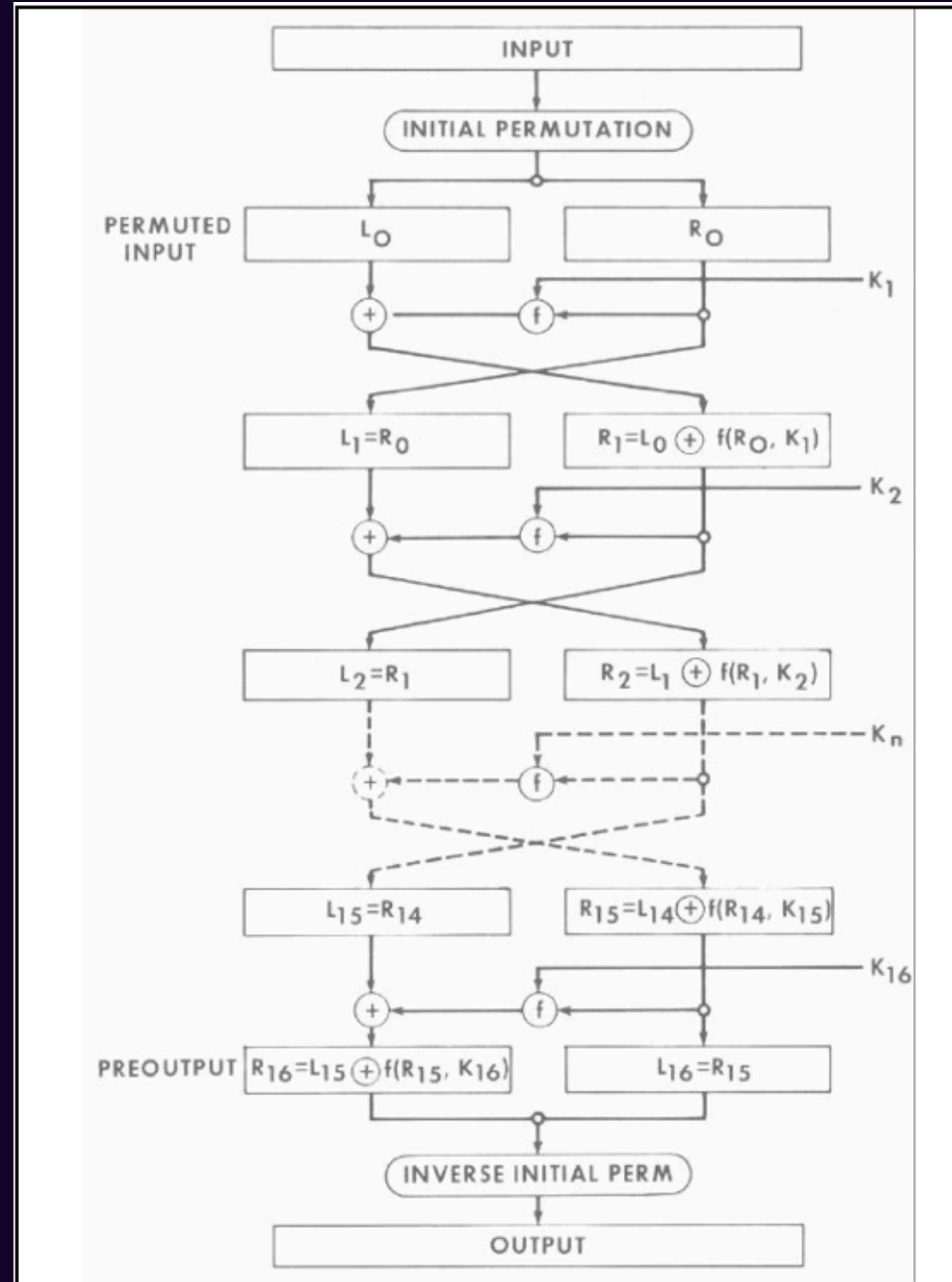
Github

Implementation



How it Works?

Data awalnya dipermutaskan (Initial Permutation), kemudian melalui 16 ronde Feistel, di mana setiap ronde melibatkan ekspansi bagian kanan data menjadi 48-bit, XOR dengan subkunci, substitusi menggunakan S-box untuk menghasilkan 32-bit, dan permutasi untuk menyebarkan bit. Setelah 16 ronde, hasilnya digabungkan kembali dan diproses melalui permutasi akhir (Final Permutation) untuk menghasilkan ciphertext. Dekripsi dilakukan dengan proses yang sama, tetapi subkunci diterapkan dalam urutan terbalik. Algoritma ini mengamankan data dengan kombinasi substitusi dan permutasi, menciptakan efek avalanche, di mana perubahan kecil pada input menghasilkan perubahan besar pada output.



Code

```
library ieee;
use ieee.std_logic_1164.all;

entity expansion is -- mengubah 32 bit menjadi 48 bit
port(
    input : in std_logic_vector(0 to 31); -- Input 32-bit data
    output : out std_logic_vector(0 to 47) -- Expanded 48-bit output
);
end expansion;

architecture behavior of expansion is

-- Define the expansion table as a constant array
type exp_array is array(0 to 47) of integer range 1 to 32;

constant ip_exp : exp_array :=
(
    32, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9, 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, 30, 31, 32, 1
);

begin

-- Process block to implement the expansion
process(input) is
begin
    for i in 0 to 47 loop
        -- Adjust the output to subtract 1 for zero-based indexing
        output(i) <= input(ip_exp(i) - 1);
    end loop;
end process;

```

Component - Expanding Unit

Expanding unit berfungsi sebagai salah satu step awal dalam proses enkripsi, proses memperluas data dengan menerima input 32-bit menjadi output 48-bit. Proses ini dilakukan pada bagian kanan (right half) dari blok data selama setiap ronde pada fungsi Feistel. DES menggunakan subkey 48-bit di setiap ronde. Expansion ini memungkinkan data 32-bit dapat di-XOR dengan subkey 48-bit. Fungsi utamanya adalah untuk meningkatkan kompleksitas

Code

```
5 -- Entity declaration for S-box 1
6 entity s_box_1 is
7 port(
8     input: in std_logic_vector(0 to 5); -- 6-bit input to the S-box
9     output: out std_logic_vector(0 to 3) -- 4-bit output from the S-box
10 );
11 end s_box_1;
12
13 architecture behavior of s_box_1 is
14
15 -- Define the S-box as a 2D array with integer values
16 type s_box is array(0 to 3, 0 to 15) of integer range 0 to 16;
17 constant box: s_box :=
18 (
19     (15, 5, 14, 2, 3, 16, 12, 9, 4, 11, 7, 13, 6, 10, 1, 8),
20     (1, 16, 8, 5, 15, 3, 14, 2, 11, 7, 13, 12, 10, 6, 4, 9),
21     (5, 2, 15, 9, 14, 7, 3, 12, 16, 13, 10, 8, 4, 11, 6, 1),
22     (16, 13, 9, 3, 5, 10, 2, 8, 6, 12, 4, 15, 11, 1, 7, 14)
23 );
24
25 begin
26     -- Process block to compute the S-box output
27 process(input) is
28     variable column: integer range 0 to 15;          -- Column index in the S-box
29     variable row: integer range 0 to 3;              -- Row index in the S-box
30     variable outer_bits: std_logic_vector(0 to 1); -- Extracted outer bits of the input
31     variable output_decimal: integer range 0 to 15; -- Final output in decimal
32 begin
33     -- Extract column index (middle 4 bits of the input)
34     column := to_integer(unsigned(input(1 to 4)));
35
36     -- Extract row index (concatenation of the outer 2 bits of the input)
37     outer_bits := input(0) & input(5);
38     row := to_integer(unsigned(outer_bits));
39
40     -- Lookup value in the S-box and subtract 1 to adjust for 0-based indexing
41     output_decimal := box(row, column) - 1;
```

Component - S-Box

S-Box (Substitution Box) adalah komponen kriptografi penting dalam algoritma DES. Fungsinya adalah untuk melakukan substitusi non-linear pada data yang masuk guna meningkatkan keamanan dan resistansi terhadap serangan seperti analisis linear atau diferensial.

Baris ditentukan oleh bit pertama dan terakhir dari input 6-bit (2 bit).

Kolom ditentukan oleh 4 bit tengah dari input 6-bit. Contoh: Jika input adalah 100011, maka: Baris = 11 (bit ke-0 dan ke-5, dalam desimal = 3). Kolom = 0001 (bit ke-1 hingga ke-4, dalam desimal = 1). Output adalah nilai di baris ke-2 dan kolom ke-1 dari tabel S-Box.

Code

```
initial_permutation.vhd
4  entity init_permuting is
5    port(
6      input: in std_logic_vector(0 to 63);
7      right_half_permutation: out std_logic_vector(0 to 31);
8      left_half_permutation: out std_logic_vector(0 to 31)
9    );
10 end init_permuting;
11
12 architecture behavior of init_permuting is
13
14   -- Define the initial permutation array
15   type init_per_array is array(0 to 63) of integer range 1 to 64;
16
17   constant ip: init_per_array := (
18     58, 50, 42, 34, 26, 18, 10, 2,
19     60, 52, 44, 36, 28, 20, 12, 4,
20     62, 54, 46, 38, 30, 22, 14, 6,
21     64, 56, 48, 40, 32, 24, 16, 8,
22     57, 49, 41, 33, 25, 17, 9, 1,
23     59, 51, 43, 35, 27, 19, 11, 3,
24     61, 53, 45, 37, 29, 21, 13, 5,
25     63, 55, 47, 39, 31, 23, 15, 7
26   );
27
28 begin
29   process(input) is
30     variable permuted_array : std_logic_vector(0 to 63);
31   begin
32     -- Apply permutation
33     for i in 0 to 63 loop
34       permuted_array(i) := input(ip(i) - 1); -- Adjust for 0-based indexing
35     end loop;
36
37     -- Split the permuted array into left and right halves
38     left_half_permutation <= permuted_array(0 to 31);
39     right_half_permutation <= permuted_array(32 to 63);
40   end process;
```

Component - Permutation

Initial Permutation (IP) adalah langkah pertama dalam algoritma DES yang bertujuan untuk mengacak urutan bit pada data input sebelum diproses lebih lanjut. Pada tahap ini, blok data 64-bit yang masuk akan dipermutasikan menggunakan tabel permutasi tetap yang telah ditentukan. Hasil dari IP adalah dua bagian: left_half (32 bit) dan right_half (32 bit). IP adalah salah satu dari beberapa algoritma permutasi yang ada dalam DES.

Code

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity permutation is
5 port(
6     input: in std_logic_vector(0 to 31);
7     output: out std_logic_vector(0 to 31)
8 );
9 end permutation;
10
11 architecture behavior of permutation is
12
13     -- Define the permutation array as a constant array
14     type ip_array is array(0 to 31) of integer range 1 to 32;
15     constant ip: ip_array := (
16         16, 7, 20, 21,
17         29, 12, 28, 17,
18         1, 15, 23, 26,
19         5, 18, 31, 10,
20         2, 8, 24, 14,
21         32, 27, 3, 9,
22         19, 13, 30, 6,
23         22, 11, 4, 25
24     );
25
26 begin
27
28     process(input) is
29     begin
30         -- Apply permutation
31         for i in 0 to 31 loop
32             output(i) <= input(ip(i) - 1); -- Adjust for 0-based indexing
33         end loop;
34     end process;
35
36 end behavior;
```

Component - Permutation

Permutasi dalam DES adalah proses reordering posisi bit berdasarkan fixed permutation table yang telah ditentukan. Pada modul ini, permutation melakukan permutasi pada blok data 32-bit dengan mengatur ulang posisi bit menggunakan tabel permutasi ip. Setiap bit dari input vector ditempatkan di lokasi baru pada output vector sesuai dengan nilai tabel.

Code

```
subkey_generation.vhd
21 architecture behavior of subkey_generation is
22     signal left_half_shifted: std_logic_vector(0 to 27);
23     signal right_half_shifted: std_logic_vector(0 to 27);
24
25 begin
26
27     -- Generate for shifting by 1-bit
28     shift_by_one: if (shifting_parameter = "01") generate
29         -- Left shifting logic
30         left_shifting: if (left_or_right = "0") generate
31             s11: left_rot_one_bit port map(
32                 input => left_key_in,
33                 output => left_half_shifted
34             );
35
36             s12: left_rot_one_bit port map(
37                 input => right_key_in,
38                 output => right_half_shifted
39             );
40         end generate left_shifting;
41
42         -- Right shifting logic
43         right_shifting: if (left_or_right = "1") generate
44             s13: right_rot_one_bit port map(
45                 input => left_key_in,
46                 output => left_half_shifted
47             );
48
49             s14: right_rot_one_bit port map(
50                 input => right_key_in,
51                 output => right_half_shifted
52             );
53         end generate right_shifting;
54     end generate shift_by_one;
55
56     -- Generate for shifting by 2-bits
57     shift_by_2: if (shifting_parameter = "10") generate
58
59         -- Left shifting logic
60         left_shifting: if (left_or_right = "0") generate
61             s15: left_double_rot_bit port map(
62                 input => left_key_in,
63                 output => left_half_shifted
64             );
65
66             s16: left_double_rot_bit port map(
67                 input => right_key_in,
68                 output => right_half_shifted
69             );
70         end generate left_shifting;
71
72         -- Right shifting logic
73         right_shifting: if (left_or_right = "1") generate
74             s17: right_double_rot_bit port map(
75                 input => left_key_in,
76                 output => left_half_shifted
77             );
78
79             s18: right_double_rot_bit port map(
80                 input => right_key_in,
81                 output => right_half_shifted
82             );
83         end generate right_shifting;
84     end generate shift_by_2;
85
86     -- Permutation logic
87     permutation: if (shifting_parameter = "11") generate
88         s19: pc2 port map(
89             input => left_half_shifted,
90             output => permuted_left
91         );
92
93         s20: pc2 port map(
94             input => right_half_shifted,
95             output => permuted_right
96         );
97     end generate permutation;
98
99 end architecture;
```

Component - Key Schedule

Subkey Generation adalah proses untuk menghasilkan subkunci unik yang digunakan di setiap ronde enkripsi atau dekripsi DES. Pada modul ini, kunci 56-bit dibagi menjadi dua bagian, yaitu left key (C) dan right key (D), masing-masing 28 bit. Kedua bagian ini mengalami shifting berdasarkan jumlah yang ditentukan oleh tabel shifting (1 atau 2 bit). Pergeseran bisa ke arah kiri atau kanan, tergantung pada mode operasi. Hasil shifting kemudian dipermutasikan kembali melalui Permutation Choice 2 (PC-2) untuk menghasilkan subkunci 48-bit.

Code

```
Decryption.vhd
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity decryption is
5    port(  input: in std_logic_vector(0 to 63);
6          key: in std_logic_vector(0 to 63);
7          output: out std_logic_vector(0 to 63));
8  end decryption;
9
10 architecture behavior of decryption is
11
12   component init_permuting
13     port(  input: in std_logic_vector(0 to 63);
14            right_half_permutation: out std_logic_vector(0 to 31);
15            left_half_permutation: out std_logic_vector(0 to 31));
16   end component;
17
18   component permutation_key_one
19     port(  key: in std_logic_vector(0 to 63);
20           left_key_permutation: out std_logic_vector(0 to 27);
21           right_key_permutation: out std_logic_vector(0 to 27));
22   end component;
23
24   component subkey_generation
25     generic(shifting_parameter: std_logic_vector(0 to 1);
26             left_or_right: std_logic_vector(0 to 0));
27     port(  left_key_in: in std_logic_vector(0 to 27);
28           right_key_in: in std_logic_vector(0 to 27);
29           subkey: out std_logic_vector(0 to 47);
30           left_key_out: out std_logic_vector(0 to 27);
31           right_key_out: out std_logic_vector(0 to 27));
32   end component;
33   component left_rot_one_bit
34     port(
35       input: in std_logic_vector(0 to 27);
36       output: out std_logic_vector(0 to 27));
37   end component;
```

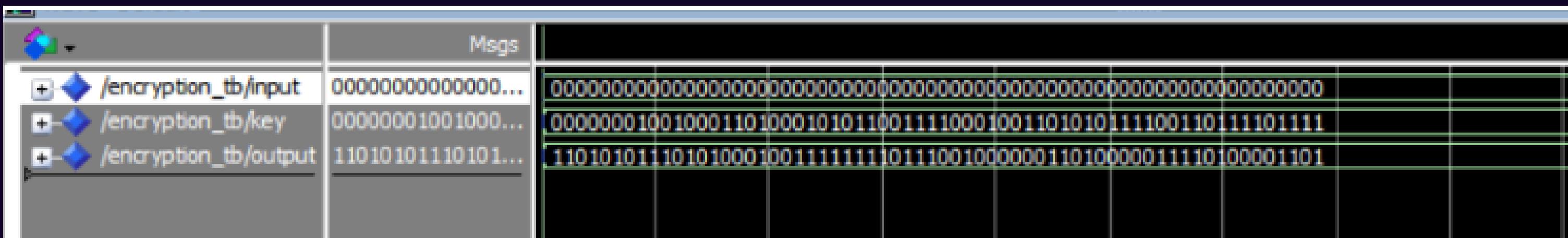
Component - Top Level

Decryption adalah salah satu komponen top level yang mendeskripsikan dan mengarahkan alur dekripsi secara penuh. Decryption akan menerima sebuah input dan key, dan menghasilkan output hasil dekripsi dari keduanya.

Testing and Result

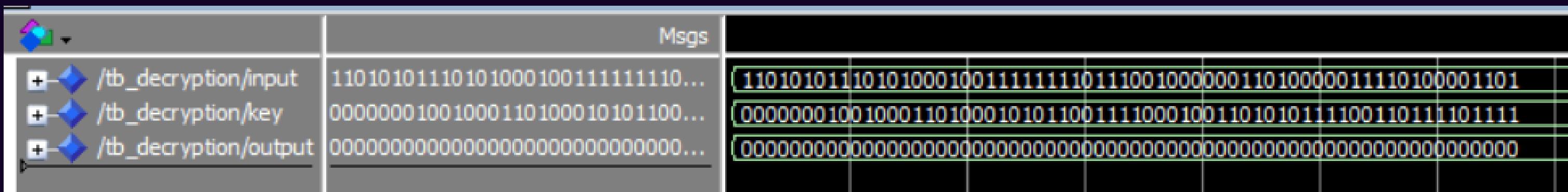


Testing using ModelSim



Encrypt sensitive data.

Testing using ModelSim



Decrypt
sensitive data.

Result

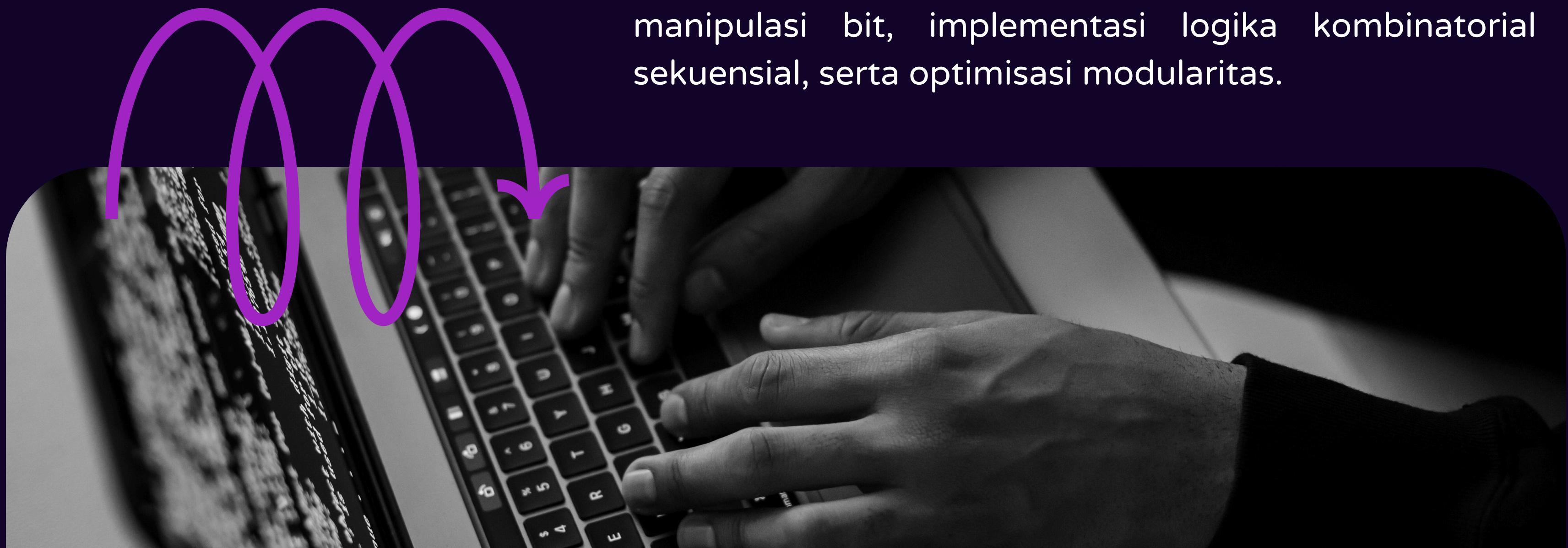
Hasil sesuai ekspektasi yaitu mengikuti algoritma Data Encryption Standard

Setelah enkripsi berhasil dilakukan, proses dekripsi berhasil mengembalikan ke key awal

Proses kompleks dengan substitusi, ekspansi, XOR dan lainnya memberikan kepercayaan akan keamanan data terenkripsi

Conclusion

Proyek implementasi algoritma Data Encryption Standard (DES) ini berhasil membangun sistem enkripsi dan dekripsi yang sesuai dengan standar DES. Melalui proyek implementasi DES, pemahaman tentang Perancangan Sistem Digital dan VHDL meningkat secara signifikan. Proyek ini mengintegrasikan berbagai aspek desain digital, seperti manipulasi bit, implementasi logika kombinatorial dan sekuensial, serta optimisasi modularitas.





THANK YOU!

