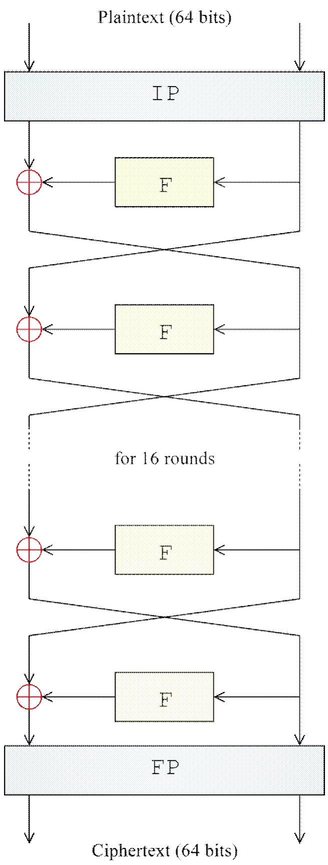
- DES

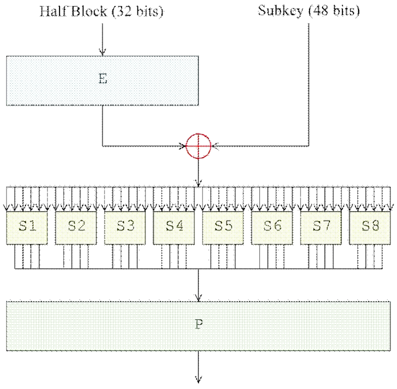
DES(Data Encryption Standard)was once a predominant symmetric-key algorithm for the encryption of electronic data. It was highly influential in the advancement of modern cryptograhpy in the academic world.

DES is now considered to be insecure for many applications. This is mainly due to the 56-bit key size being too small. There are some analytical results which demonstrate theoretical weaknesses in the cipher, although they are infeasible to mount in practice. The algorithm is believed to be practically secure in the form of Triple DES, although there are theoretical attacks.



DES is the archetypal block cipher –an algorithm that takes a fixed length string of plaintext bits and transforms it through a series of complicated operations into another ciphertext bitstring of the same length. Inthe case of DES, the block size is 64bits. DES uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64 bits. However, only 56 bits are actually used by the algorithm. 8bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56bits. Decryption uses the same structure as encryption but with the keys used in reverse order.

There are 16 identical stages of processing termed rounds. Before the main rounds, the block is divided into two 32bit nalves and processed alternately; this criss-crossing is known as the Feistel scheme(F function). The Feistel structuer ensures that decryption and encryption are very similar processes-the only difference is that he subkeys are applied in the reverse order when decrypting. The rest of the algorithm is identical. This greatly simplifies implementation, particularly in hardware, as there is no need for separate encryption and decryption algorithms.

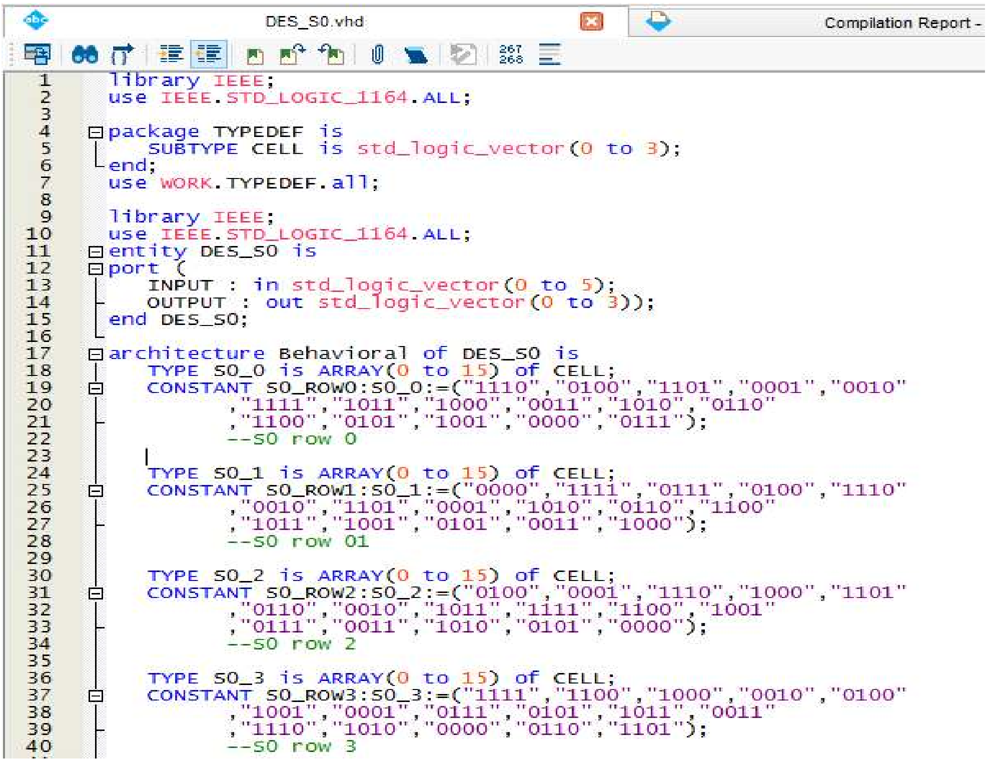


The ⊕symbol denotes the exclusive OR(XOR) operation. The F-function scrambles half a block together with some of the key. The output form the F-function is then combined with the other half of the block, and the halves are swapped before the next round.

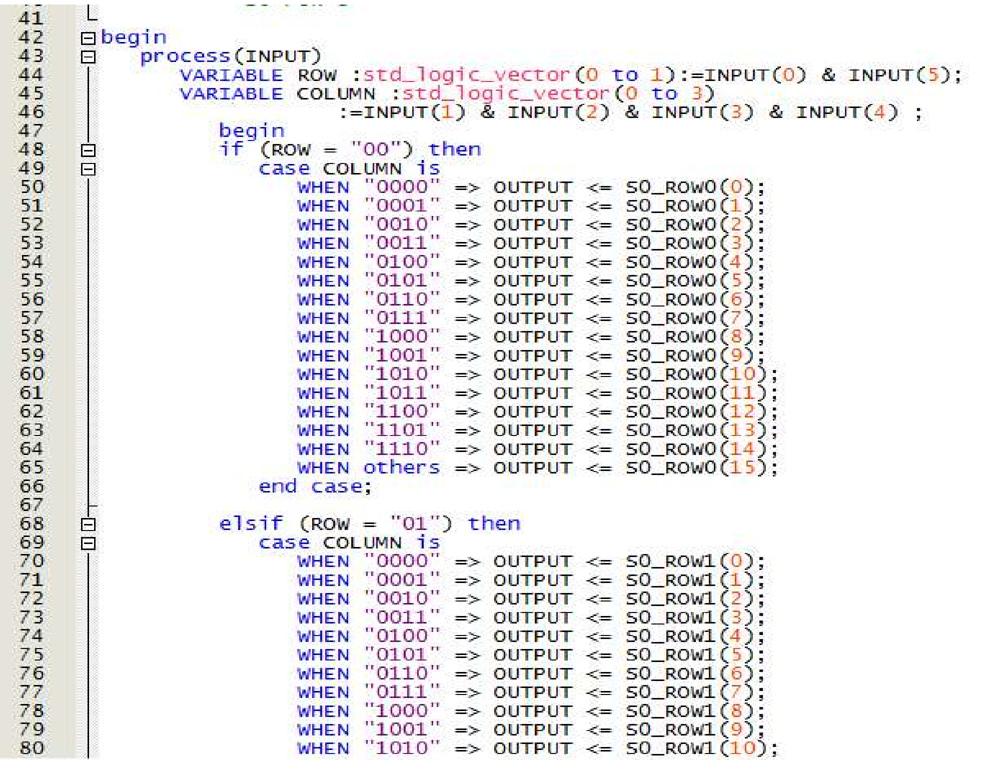
The F-function operates on half a block(32bits) at a time and consists of four stages, Expansion →Key mixing →Substitution →Permutation.

- Web link for DES I found
  https://en.wikipedia.org/wiki/Data_Encryption_Standard
  http://www.tutorialspoint.com/cryptography/data_encryption_standard.htm

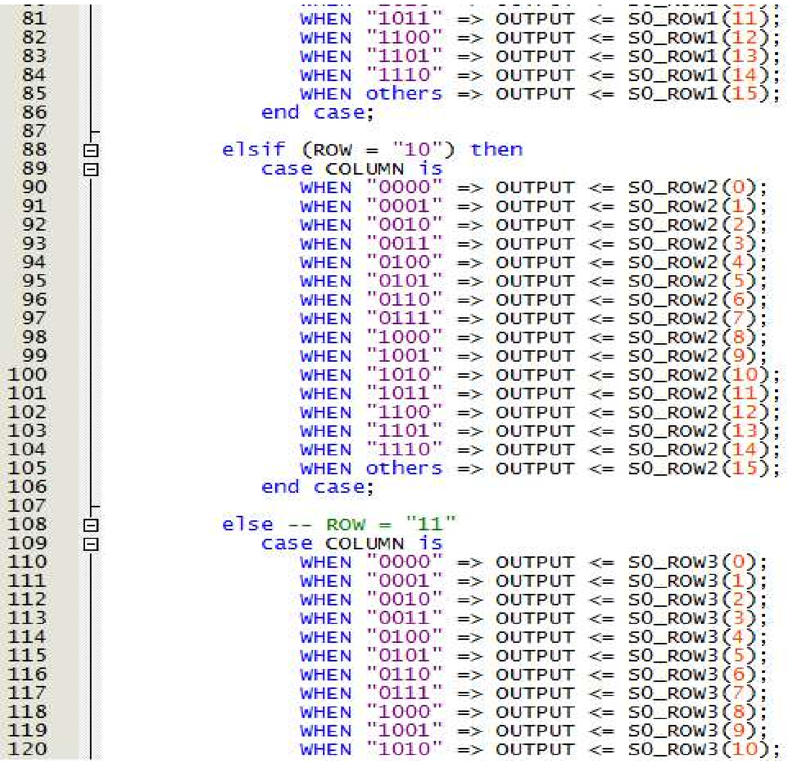
- VHDL source code for S[0]



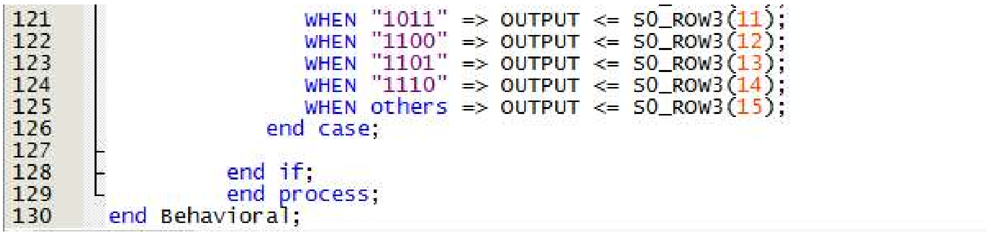
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

package TYPEDEF is
    SUBTYPE CELL is std_logic_vector(0 to 3);
end;
use WORK.TYPEDEF.all;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity DES_S0 is
port (
    INPUT : in std_logic_vector(0 to 5);
    OUTPUT : out std_logic_vector(0 to 3));
end DES_S0;

architecture Behavioral of DES_S0 is
    TYPE S0_0 is ARRAY(0 to 15) of CELL;
    CONSTANT S0_ROW0:S0_0:=("1110","0100","1101","0001","0010"
            ,"1111","1011","1000","0011","1010","0110"
            ,"1100","0101","1001","0000","0111");
            --S0 row 0

    TYPE S0_1 is ARRAY(0 to 15) of CELL;
    CONSTANT S0_ROW1:S0_1:=("0000","1111","0111","0100","1110"
            ,"0010","1101","0001","1010","0110","1100"
            ,"1011","1001","0101","0011","1000");
            --S0 row 01

    TYPE S0_2 is ARRAY(0 to 15) of CELL;
    CONSTANT S0_ROW2:S0_2:=("0100","0001","1110","1000","1101"
            ,"0110","0010","1011","1111","1100","1001"
            ,"0111","0011","1010","0101","0000");
            --S0 row 2

    TYPE S0_3 is ARRAY(0 to 15) of CELL;
    CONSTANT S0_ROW3:S0_3:=("1111","1100","1000","0010","0100"
            ,"1001","0001","0111","0101","1011","0011"
            ,"1110","1010","0000","0110","1101");
            --S0 row 3
```



```

begin
    process(INPUT)
        VARIABLE ROW :std_logic_vector(0 to 1):=INPUT(0) & INPUT(5);
        VARIABLE COLUMN :std_logic_vector(0 to 3)
                    :=INPUT(1) & INPUT(2) & INPUT(3) & INPUT(4) ;
            begin
            if (ROW = "00") then
                case COLUMN is
                    WHEN "0000" => OUTPUT <= S0_ROW0(0);
                    WHEN "0001" => OUTPUT <= S0_ROW0(1);
                    WHEN "0010" => OUTPUT <= S0_ROW0(2);
                    WHEN "0011" => OUTPUT <= S0_ROW0(3);
                    WHEN "0100" => OUTPUT <= S0_ROW0(4);
                    WHEN "0101" => OUTPUT <= S0_ROW0(5);
                    WHEN "0110" => OUTPUT <= S0_ROW0(6);
                    WHEN "0111" => OUTPUT <= S0_ROW0(7);
                    WHEN "1000" => OUTPUT <= S0_ROW0(8);
                    WHEN "1001" => OUTPUT <= S0_ROW0(9);
                    WHEN "1010" => OUTPUT <= S0_ROW0(10);
                    WHEN "1011" => OUTPUT <= S0_ROW0(11);
                    WHEN "1100" => OUTPUT <= S0_ROW0(12);
                    WHEN "1101" => OUTPUT <= S0_ROW0(13);
                    WHEN "1110" => OUTPUT <= S0_ROW0(14);
                    WHEN others => OUTPUT <= S0_ROW0(15);
                end case;

            elsif (ROW = "01") then
                case COLUMN is
                    WHEN "0000" => OUTPUT <= S0_ROW1(0);
                    WHEN "0001" => OUTPUT <= S0_ROW1(1);
                    WHEN "0010" => OUTPUT <= S0_ROW1(2);
                    WHEN "0011" => OUTPUT <= S0_ROW1(3);
                    WHEN "0100" => OUTPUT <= S0_ROW1(4);
                    WHEN "0101" => OUTPUT <= S0_ROW1(5);
                    WHEN "0110" => OUTPUT <= S0_ROW1(6);
                    WHEN "0111" => OUTPUT <= S0_ROW1(7);
                    WHEN "1000" => OUTPUT <= S0_ROW1(8);
                    WHEN "1001" => OUTPUT <= S0_ROW1(9);
                    WHEN "1010" => OUTPUT <= S0_ROW1(10);
```

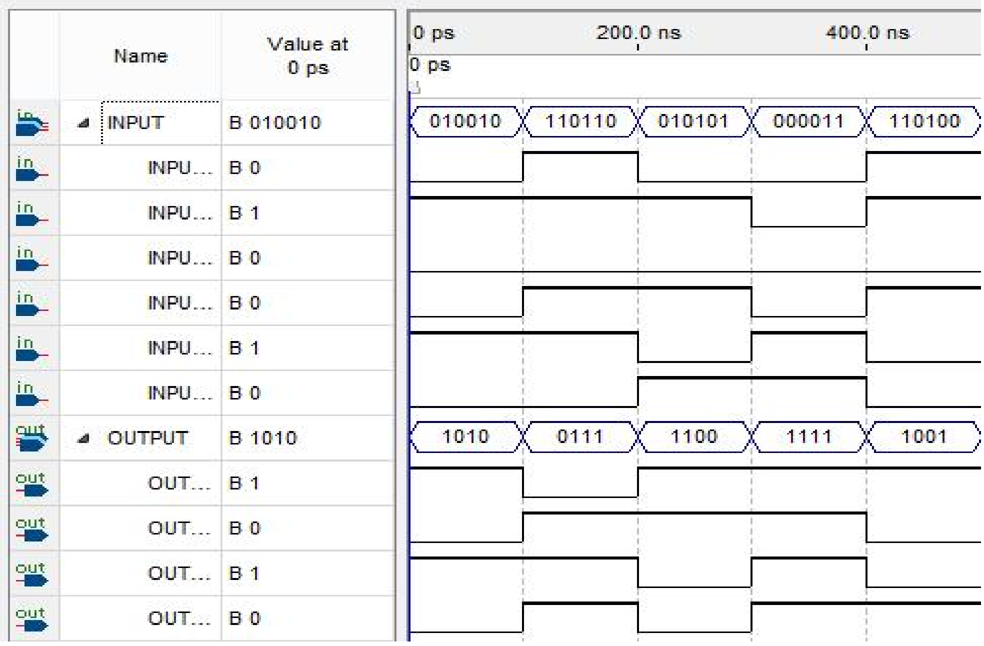


```
               WHEN "1011" => OUTPUT <= SO_ROW1(11);
               WHEN "1100" => OUTPUT <= SO_ROW1(12);
               WHEN "1101" => OUTPUT <= SO_ROW1(13);
               WHEN "1110" => OUTPUT <= SO_ROW1(14);
               WHEN others => OUTPUT <= SO_ROW1(15);
            end case;

         elsif (ROW = "10") then
            case COLUMN is
               WHEN "0000" => OUTPUT <= SO_ROW2(0);
               WHEN "0001" => OUTPUT <= SO_ROW2(1);
               WHEN "0010" => OUTPUT <= SO_ROW2(2);
               WHEN "0011" => OUTPUT <= SO_ROW2(3);
               WHEN "0100" => OUTPUT <= SO_ROW2(4);
               WHEN "0101" => OUTPUT <= SO_ROW2(5);
               WHEN "0110" => OUTPUT <= SO_ROW2(6);
               WHEN "0111" => OUTPUT <= SO_ROW2(7);
               WHEN "1000" => OUTPUT <= SO_ROW2(8);
               WHEN "1001" => OUTPUT <= SO_ROW2(9);
               WHEN "1010" => OUTPUT <= SO_ROW2(10);
               WHEN "1011" => OUTPUT <= SO_ROW2(11);
               WHEN "1100" => OUTPUT <= SO_ROW2(12);
               WHEN "1101" => OUTPUT <= SO_ROW2(13);
               WHEN "1110" => OUTPUT <= SO_ROW2(14);
               WHEN others => OUTPUT <= SO_ROW2(15);
            end case;

         else -- ROW = "11"
            case COLUMN is
               WHEN "0000" => OUTPUT <= SO_ROW3(0);
               WHEN "0001" => OUTPUT <= SO_ROW3(1);
               WHEN "0010" => OUTPUT <= SO_ROW3(2);
               WHEN "0011" => OUTPUT <= SO_ROW3(3);
               WHEN "0100" => OUTPUT <= SO_ROW3(4);
               WHEN "0101" => OUTPUT <= SO_ROW3(5);
               WHEN "0110" => OUTPUT <= SO_ROW3(6);
               WHEN "0111" => OUTPUT <= SO_ROW3(7);
               WHEN "1000" => OUTPUT <= SO_ROW3(8);
               WHEN "1001" => OUTPUT <= SO_ROW3(9);
               WHEN "1010" => OUTPUT <= SO_ROW3(10);
```



```
               WHEN "1011" => OUTPUT <= SO_ROW3(11);
               WHEN "1100" => OUTPUT <= SO_ROW3(12);
               WHEN "1101" => OUTPUT <= SO_ROW3(13);
               WHEN "1110" => OUTPUT <= SO_ROW3(14);
               WHEN others => OUTPUT <= SO_ROW3(15);
            end case;

         end if;
         end process;
end Behavioral;
```

- Screenshot of the waveform for test



Name	Value at 0 ps
INPUT	B 010010
INPU...	B 0
INPU...	B 1
INPU...	B 0
INPU...	B 0
INPU...	B 1
INPU...	B 0
OUTPUT	B 1010
OUT...	B 1
OUT...	B 0
OUT...	B 1
OUT...	B 0