Experiment 3: Combinational Circuit 3

Prateek Garg
20D070060
EE-214, WEL, IIT Bombay
September 8th, 2021

Overview of the experiment:

Main Objectives:

- Design a universal shifter circuit, which can perform logical right shift or left shift on 8-bit input by the specified number of bits.
- Describe the designed circuit in vhdl.
- Simulate the design using the generic testbench in ModelSim.
- Test the correctness of design using Scanchain.

The design of circuit was broken down into designing smaller parts which were:

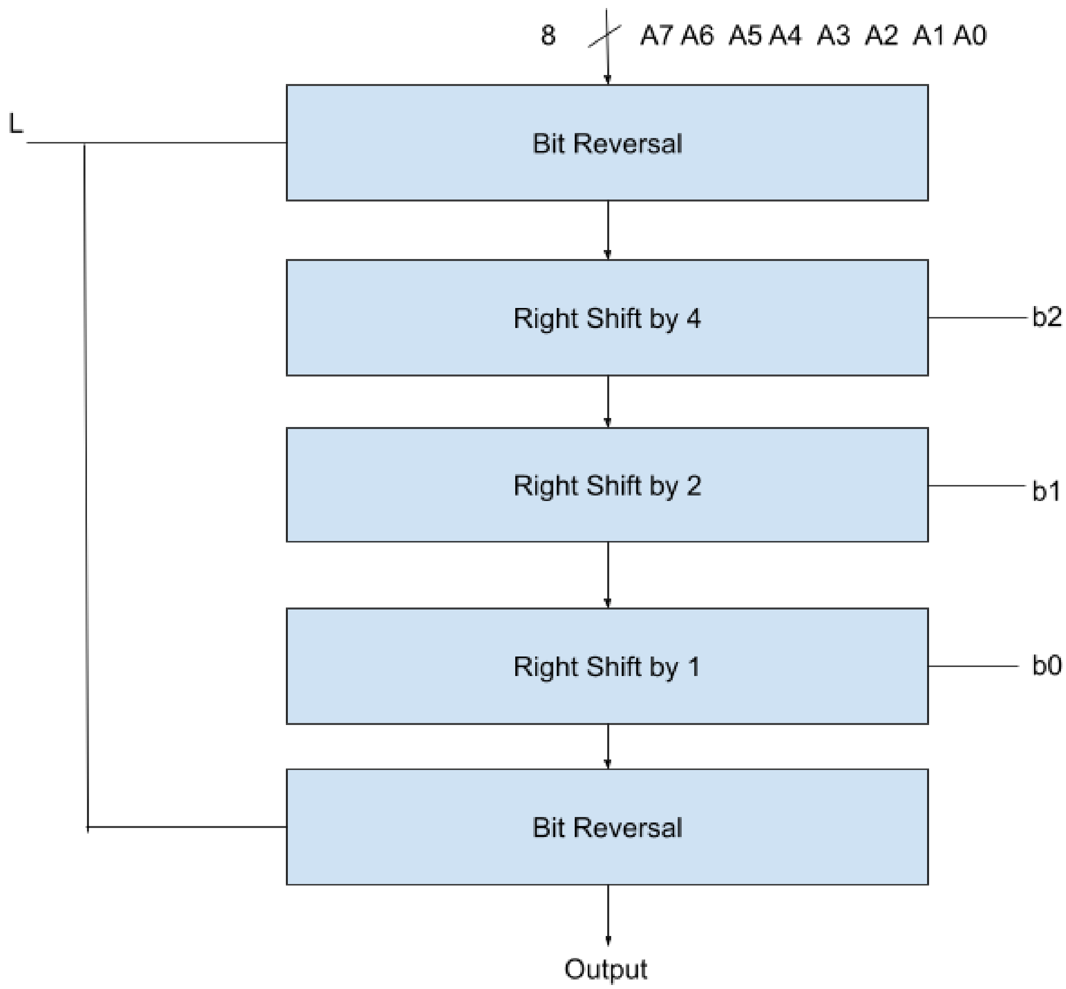
- 8-bit block to revert the bit sequence.
- 8-bit block to right shift input by 1 bits.
- 8-bit block to right shift input by 2 bits.
- 8-bit block to right shift input by 4 bits.

The design was described in VHDL using the software Quartus Prime. Structural modelling was used which means components were instantiated and port mapping is used to describe connections. The design was then simulated using ModelSim checked against every possible testcase. Then the design was simulated on Krypton board and checked against every possible test case using Scanchain.

Approach to the experiment:

Design:

Universal Shifter Circuit:



The objective is to design an universal shifter. The shifter is given 2 inputs, direction-right or left and amount of shift which can range from 0 to 7.

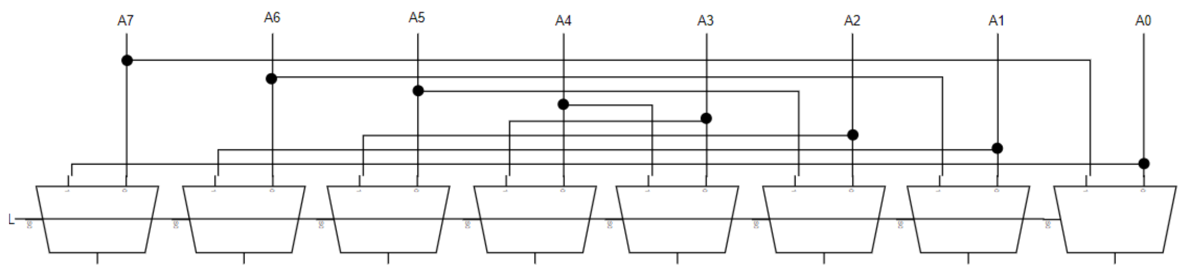
The direction is modelled using single bit L. L=0 implies right and L=1 implies left. Since amount of shift can vary from 0 to 7 it can be modelled using $\log_2(8)=3$ bits b0,b1 and b2. Each of which signifies a shift of 2^0, 2^1, 2^2 or 1,2 and 4. Every shift from 0 to 7 can be represented by 1,2 and 4 as:

0=1(0)+2(0)+4(0)
1=1(1)+2(0)+4(0)
2=1(0)+2(1)+4(0)
3=1(1)+2(1)+4(0)
4=1(0)+2(0)+4(1)

5=1(1)+2(0)+4(1)
6=1(0)+2(1)+4(1)
7=1(1)+2(1)+4(1)

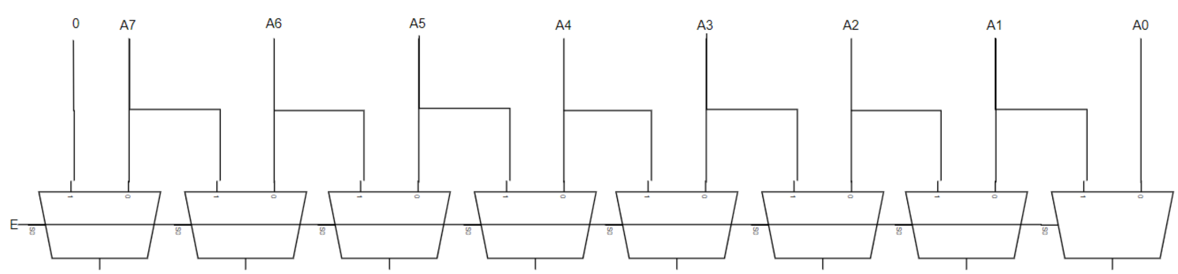
We see this is essentially a truth table for 3 bits.
The Right and Left functionality is achieved using an Bit reversal block which revert the input when E is high. We then right shift this reverted input and then revert again which achieves a left shift. We now design each block separately.

8-bit input bit reversal:



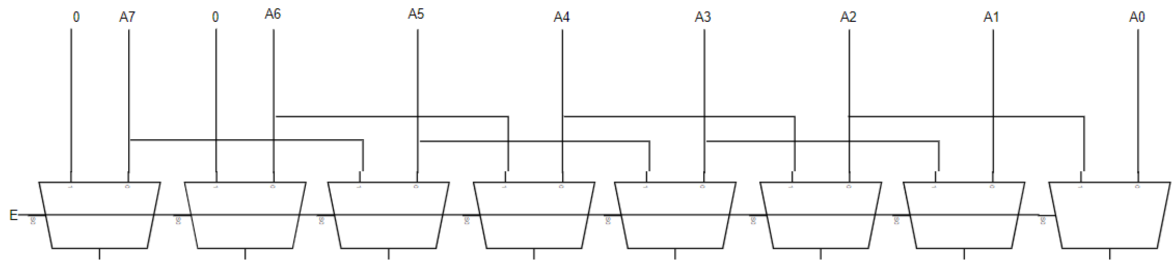
The bit reversal circuit uses 8 2x1 Multiplexers. The I_0 input of i^{th} multiplexer is A_i while I_1 is A_{7-i}. The S input of each multiplexer is E. When E is low each multiplexer outputs the I_0 which is i^{th} bit itself but when E is high each outputs A_{7-i} which is reversed bit from the other end. Thus we achieve a 8-input block which reverses the input when E is high and the same otherwise.

8-bit input right shift by 1:



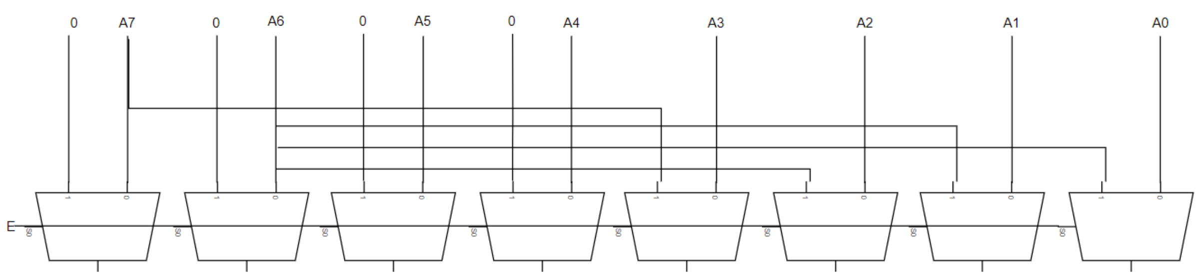
The shifter circuit uses 8 2x1 Multiplexers. The I_0 input of i^{th} multiplexer is A_i while I_1 is A_{i+1}. The I_1 input of 7^{th} multiplexer is 0. S input of each multiplexer is E. When E is low each multiplexer outputs the I_0 which is i^{th} bit itself but when E is high each outputs A_{i+1} which is shifted by 1 bit to right. Thus we achieve a 8-input block which shifts the input by 1 when E is high and the same otherwise.

8-bit input right shift by 2:



The shifter circuit uses 8 2x1 Multiplexers. The I_0 input of i^{th} multiplexer is A_i while I_1 is A_{i+2}. The I_1 input of the 6th and 7th multiplexer is 0. S input of each multiplexer is E. When E is low each multiplexer outputs the I_0 which is i^{th} bit itself but when E is high each outputs A_{i+2} which is shifted by 2 bit to right. Thus we achieve an 8-input block which shifts the input by 2 when E is high and the same otherwise.

8-bit input right shift by 4:



The shifter circuit uses 8 2x1 Multiplexers. The I_0 input of i^{th} multiplexer is A_i while I_1 is A_{i+4}. The I_1 input of the 4th, 5th, 6th and 7th multiplexer is 0. S input of each multiplexer is E. When E is low each multiplexer outputs the I_0 which is i^{th} bit itself but when E is high each outputs A_{i+4} which is shifted by 4 bit to right. Thus we achieve an 8-input block which shifts the input by 4 when E is high and the same otherwise.

Circuit is drawn using online tool: [Logic Diagram Software]

Design document and VHDL code:

Architecture of 8-bit Universal-Shifter:

```
entity Universal_Shifter is
  port (
          I : in std_logic_vector(7 downto 0);
          b : in std_logic_vector(2 downto 0);
          L : in std_logic;
          Y: out std_logic_vector(7 downto 0)

      );
end entity Universal_Shifter;

architecture Struct of Universal_Shifter is
  component Bit_Reversal is
    port (I : in std_logic_vector(7 downto 0); E : in std_logic; Y: out
std_logic_vector(7 downto 0));
  end component;

  component Right_Shift_4 is
    port (I : in std_logic_vector(7 downto 0); E : in std_logic; Y: out
std_logic_vector(7 downto 0));
  end component;

  component Right_Shift_2 is
  port (I : in std_logic_vector(7 downto 0); E : in std_logic; Y: out
std_logic_vector(7 downto 0));
  end component;

  component Right_Shift_1 is
  port (I : in std_logic_vector(7 downto 0); E : in std_logic; Y: out
std_logic_vector(7 downto 0));
  end component;

  signal Y1,Y2,Y3,Y4: std_logic_vector(7 downto 0);

begin
     rev0: Bit_Reversal  port map (I => I,  E => L,   Y => Y1);
   R4:   Right_Shift_4 port map (I => Y1, E => b(2),Y => Y2);
```

```
   R2:   Right_Shift_2 port map (I => Y2, E => b(1),Y => Y3);
   R1:   Right_Shift_1 port map (I => Y3, E => b(0),Y => Y4);
   rev1: Bit_Reversal  port map (I => Y4, E => L,    Y => Y);
end Struct;
```

Architecture of 8-bit input bit reversal:

```
architecture Struct of Bit_Reversal is
  component  MUX_2x1  is
  port (I: in std_logic_vector(1 downto 0); S: in std_logic; Y: out
std_logic);
  end component;

begin
  bit_Rev : for a in 0 to 7 generate

   mx: MUX_2x1 port map(I(0) => I(a), I(1) => I(7-a), S => E,Y =>Y(a));

  end generate ; --Bit_Reversal

end Struct;
```

Architecture of 8-bit input right shift by 1:

```
architecture Struct of Right_Shift_1 is
  component  MUX_2x1  is
  port (I: in std_logic_vector(1 downto 0); S: in std_logic; Y: out
std_logic);
  end component;

begin
  R_Shift_1: for a in 0 to 7 generate

   lsb: if a < 7 generate
            mx: MUX_2x1   port map(I(0) => I(a), I(1) => I(a+1), S => E,
Y => Y(a));
        end generate lsb;

        msb: if a > 6 generate
```

```
            mx: MUX_2x1   port map(I(0) => I(a), I(1) => '0', S => E, Y
=> Y(a));
        end generate msb;

  end generate ; --Right_Shift_1

end Struct;
```

Architecture of 8-bit input right shift by 2:

```
architecture Struct of Right_Shift_2 is
  component  MUX_2x1  is
  port (I: in std_logic_vector(1 downto 0); S: in std_logic; Y: out
std_logic);
  end component;

begin
  R_Shift_2: for a in 0 to 7 generate

   lsb: if a < 6 generate
            mx: MUX_2x1   port map(I(0) => I(a), I(1) => I(a+2), S => E,
Y => Y(a));
        end generate lsb;

        msb: if a > 5 generate
            mx: MUX_2x1   port map(I(0) => I(a), I(1) => '0', S => E, Y
=> Y(a));
        end generate msb;

  end generate ; --Right_Shift_2

end Struct;
```

Architecture of 8-bit input right shift by 4:.

```
architecture Struct of Right_Shift_4 is
  component  MUX_2x1  is
  port (I: in std_logic_vector(1 downto 0); S: in std_logic; Y: out
std_logic);
  end component;
```

```
begin
  R_Shift_4: for a in 0 to 7 generate

   lsb: if a < 4 generate
            mx: MUX_2x1 port map(I(0) => I(a), I(1) => I(a+4), S => E, Y
=> Y(a));
        end generate lsb;

        msb: if a > 3 generate
            mx: MUX_2x1 port map(I(0) => I(a), I(1) => '0', S => E, Y =>
Y(a));
        end generate msb;

  end generate ; --Right_Shift_4

end Struct;
```

RTL View:

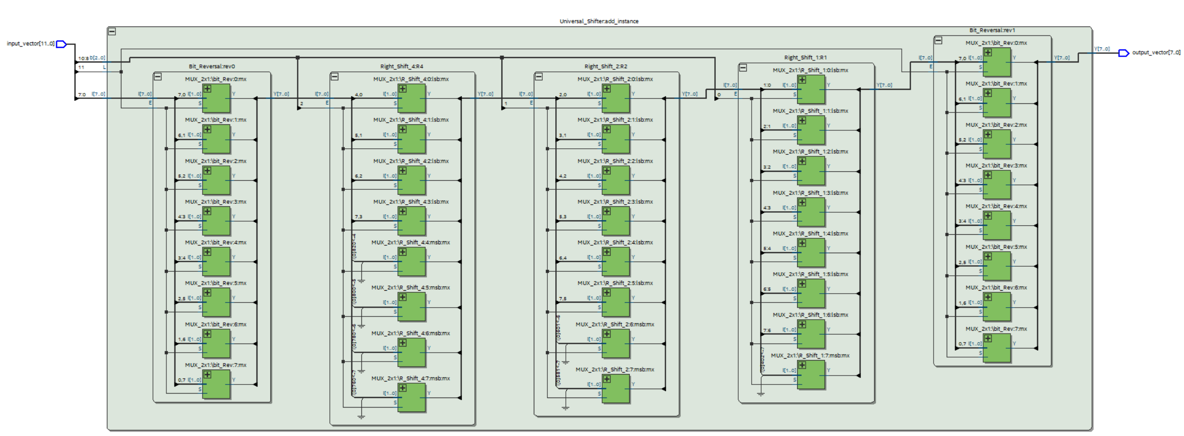


Fig: RTL View.

DUT Input/Output Format:

Port map:

```
-- order of inputs
L   => input_vector(11),
b   => input_vector(10 downto 8),
I   => input_vector(7 downto 0),

-- order of outputs
Y   => output_vector(7 downto 0)
```

```
Some test cases from TRACEFILE:
TRACEFILE format
<L B2 B1 B0 A7 A6 A5 A4 A3 A2 A1 A0> <S7 S6 S5 S4 S3 S2 S1 S0>
11111111

                    011010100001 00000010 11111111
                    011010100010 00000010 11111111
                    011010100011 00000010 11111111
                    011010100100 00000010 11111111
                    011010100101 00000010 11111111
                    011010100110 00000010 11111111
                    011010100111 00000010 11111111
                    011010101000 00000010 11111111
                    011010101001 00000010 11111111
                    011010101010 00000010 11111111
                    011010101011 00000010 11111111
                    011010101100 00000010 11111111
                    011010101101 00000010 11111111
                    011010101110 00000010 11111111
                    011010101111 00000010 11111111
                    011010110000 00000010 11111111
                    011010110001 00000010 11111111
                    011010110010 00000010 11111111
                    011010110011 00000010 11111111
                    011010110100 00000010 11111111
                    011010110101 00000010 11111111
                    011010110110 00000010 11111111
                    011010110111 00000010 11111111
                    011010111000 00000010 11111111
```

RTL Simulation:

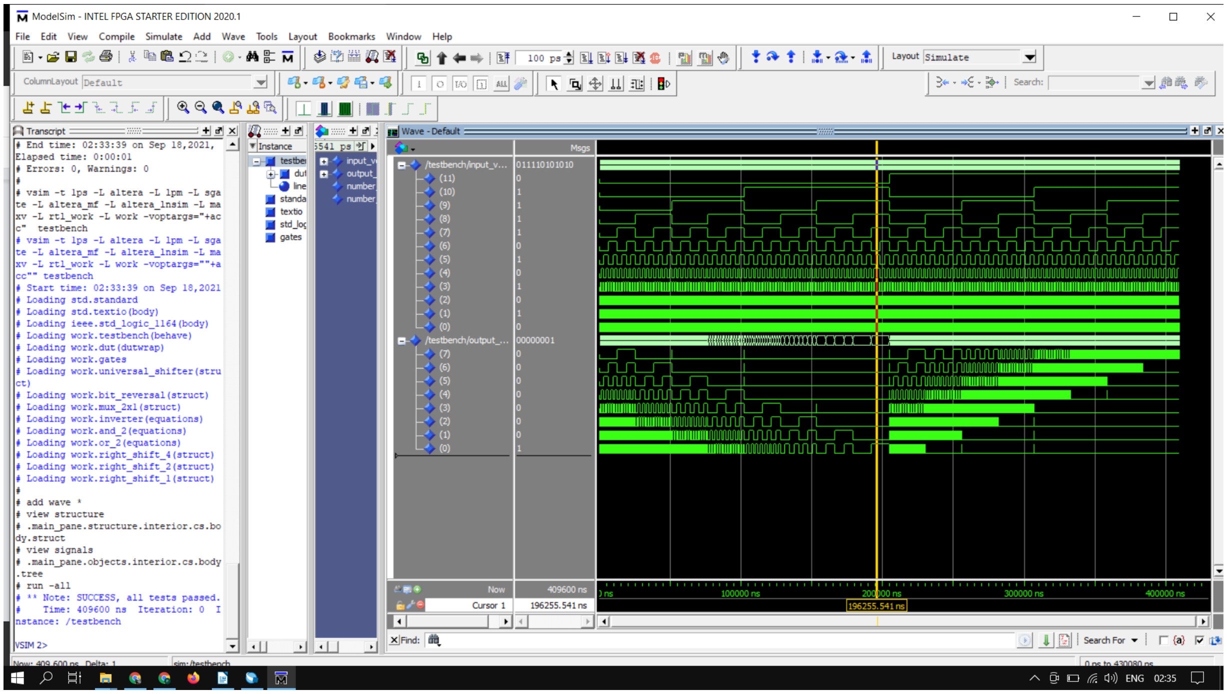


Fig: RTL Simulation

Gate-level Simulation:

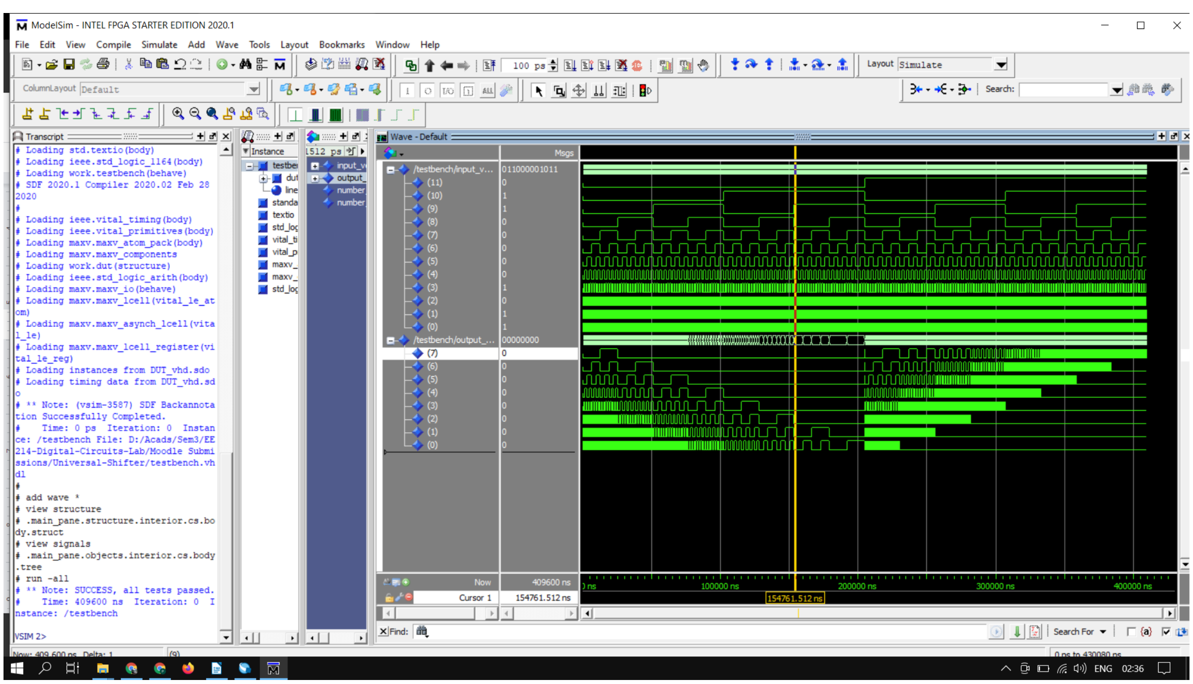


Fig: Gate Level Simulation

Observation:

Parts of the file out.txt generated by scanchain:

```
010111011000 00000110 Success
010111011001 00000110 Success
010111011010 00000110 Success
010111011011 00000110 Success
010111011100 00000110 Success
010111011101 00000110 Success
010111011110 00000110 Success
010111011111 00000110 Success
010111100000 00000111 Success
010111100001 00000111 Success
010111100010 00000111 Success
010111100011 00000111 Success
010111100100 00000111 Success
010111100101 00000111 Success
010111100110 00000111 Success
010111100111 00000111 Success
010111101000 00000111 Success
010111101001 00000111 Success
010111101010 00000111 Success
010111101100 00000111 Success
010111101101 00000111 Success
010111101110 00000111 Success
010111101111 00000111 Success
010111110000 00000111 Success
010111110001 00000111 Success
010111110010 00000111 Success
010111110011 00000111 Success
010111110100 00000111 Success
010111110101 00000111 Success
010111110110 00000111 Success
010111110111 00000111 Success
010111111000 00000111 Success
010111111001 00000111 Success
010111111010 00000111 Success
010111111011 00000111 Success
010111111100 00000111 Success
010111111101 00000111 Success
010111111110 00000111 Success
```

References:

1. J.F.Wakerly: Digital Design, Principles and Practices,4th Edition,Pearson Education, 2005