

Name: **Dhruv Jain**

Rollno.: **180020006**

Digital Systems (EE 204) Assignment-1

Project Link for all VHDL Files :-

https://github.com/Feetly/Course_Projects/tree/master/11.%20ModelSim%20Projects

Q.1 Write a VHDL code to design an arithmetic and logic unit using logic gates that performs the following three operations:

1. 4-bit addition
2. 4-bit subtraction
3. 4-bit multiplication

Write a test bench that includes 10 input combinations for the above program.

A.1 => Inputs A, B, S. Output R. S = 00 (add), S = 01 (Sub), S = 10 (Multiply), S = 11(Concatenate)

Code: -

```
library IEEE;
```

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

```
use IEEE.NUMERIC_STD.ALL;
```

```
entity full_sub is
```

```
Port(
```

```
  A : in STD_LOGIC;
```

```
  B : in STD_LOGIC;
```

```
  Cin : in STD_LOGIC;
```

```
  S : out STD_LOGIC;
```

```
  Cout : out STD_LOGIC);
```

```
end full_sub;
```

```
architecture dataflow of full_sub is
```

```
begin
```

```
  S <= A XOR B XOR Cin ;
```

```
  Cout <= ((not A) AND B) OR (Cin AND (not A)) OR (Cin AND B) ;
```

```
end dataflow;
```

```
library IEEE;
```

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

```
use IEEE.NUMERIC_STD.ALL;
```

```
entity full_adder is
```

```
Port(
```

```
A : in STD_LOGIC;
```

```
B : in STD_LOGIC;
```

```
Cin : in STD_LOGIC;
```

```
S : out STD_LOGIC;
```

```
Cout : out STD_LOGIC);
```

```
end full_adder;
```

```
architecture dataflow of full_adder is
```

```
begin
```

```
S <= A XOR B XOR Cin ;
```

```
Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;
```

```
end dataflow;
```

```
library IEEE;
```

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

```
use IEEE.NUMERIC_STD.ALL;
```

```
entity Multiplier is
```

```
port(
```

```
A,B: in std_logic_vector(3 downto 0);
```

```
R: out std_logic_vector(7 downto 0)
```

```
);
```

```
end entity Multiplier;
```

```
architecture Behavioral of Multiplier is
```

```
begin
```

```
R <= std_logic_vector(unsigned(A) * unsigned(B));
```

```
end architecture Behavioral;
```

```
library IEEE;
```

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

```
use IEEE.NUMERIC_STD.ALL;
```

```
entity calc is
```

```
Port(
```

```
A : in std_logic_vector(3 downto 0);
```

```
B : in std_logic_vector(3 downto 0);
```

```
S : in std_logic_vector(1 downto 0);
```

```
R : out std_logic_vector(7 downto 0));
```

```
end calc;
```

```
architecture Behavioral of calc is
```

```
component full_adder
```

```
Port(
```

```
A : in STD_LOGIC;
```

```
B : in STD_LOGIC;
```

```
Cin : in STD_LOGIC;
```

```
S : out STD_LOGIC;
```

```
Cout : out STD_LOGIC);
```

```
end component;
```

```
component full_sub
Port(
A : in STD_LOGIC;
B : in STD_LOGIC;
Cin : in STD_LOGIC;
S : out STD_LOGIC;
Cout : out STD_LOGIC);
end component;
```

```
COMPONENT Multiplier
port(
A , B: in std_logic_vector(3 downto 0);
R: out std_logic_vector(7 downto 0)
);
END COMPONENT;
```

```
signal RA : std_logic_vector(7 downto 0) := (others => '0');
signal RS : std_logic_vector(7 downto 0) := (others => '0');
signal RM : std_logic_vector(7 downto 0) := (others => '0');
signal tmp : std_logic := '0';
signal c1,c2,c3: STD_LOGIC;
signal f1,f2,f3: STD_LOGIC;
```

```
begin
```

```
FS1: full_sub port map( A(0), B(0), tmp, RS(0), f1);
FS2: full_sub port map( A(1), B(1), f1, RS(1), f2);
FS3: full_sub port map( A(2), B(2), f2, RS(2), f3);
FS4: full_sub port map( A(3), B(3), f3, RS(3), RS(4));
```

```
M1 : Multiplier port map(A,B,RM);
```

```
FA1: full_adder port map( A(0), B(0), tmp, RA(0), c1);
```

```
FA2: full_adder port map( A(1), B(1), c1, RA(1), c2);  
FA3: full_adder port map( A(2), B(2), c2, RA(2), c3);  
FA4: full_adder port map( A(3), B(3), c3, RA(3), RA(4));
```

```
process (S,RA,RS,RM,A,B)
```

```
begin
```

```
if S = "00" then
```

```
  R <= RA;
```

```
elseif S = "01" then
```

```
  R <= RS;
```

```
elseif S = "10" then
```

```
  R <= RM;
```

```
else
```

```
  R(0) <= B(0);
```

```
  R(1) <= B(1);
```

```
  R(2) <= B(2);
```

```
  R(3) <= B(3);
```

```
  R(4) <= A(0);
```

```
  R(5) <= A(1);
```

```
  R(6) <= A(2);
```

```
  R(7) <= A(3);
```

```
end if;
```

```
end process;
```

```
end Behavioral;
```

Test Banch Code:-

```
library ieee;
```

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

```
entity calc_tb is
```

```
end calc_tb;
```

architecture behavior OF calc_tb is

COMPONENT calc

Port(

A : in std_logic_vector(3 downto 0);

B : in std_logic_vector(3 downto 0);

S : in std_logic_vector(1 downto 0);

R : out std_logic_vector(7 downto 0));

END COMPONENT;

signal A_tb : std_logic_vector(3 downto 0) := (others => '0');

signal B_tb : std_logic_vector(3 downto 0) := (others => '0');

signal S_tb : std_logic_vector(1 downto 0) := (others => '0');

signal R_tb : std_logic_vector(7 downto 0);

BEGIN

uut: calc PORT MAP (

A => A_tb,

B => B_tb,

S => S_tb,

R => R_tb

);

process

begin

S_tb <= "00";

A_tb <= "1111";

B_tb <= "1111";

wait for 100 ns;

```
S_tb <= "01";  
A_tb <= "1111";  
B_tb <= "1111";
```

```
wait for 100 ns;  
S_tb <= "10";  
A_tb <= "1111";  
B_tb <= "1111";
```

```
wait for 100 ns;  
S_tb <= "11";  
A_tb <= "1111";  
B_tb <= "1111";
```

```
wait for 100 ns;  
S_tb <= "00";  
A_tb <= "1001";  
B_tb <= "1010";
```

```
wait for 100 ns;  
S_tb <= "01";  
A_tb <= "1001";  
B_tb <= "1010";
```

```
wait for 100 ns;  
S_tb <= "10";  
A_tb <= "1001";  
B_tb <= "1010";
```

```
wait for 100 ns;  
S_tb <= "11";  
A_tb <= "1001";  
B_tb <= "1010";
```

wait for 100 ns;

S_tb <= "00";

A_tb <= "1100";

B_tb <= "0011";

wait for 100 ns;

S_tb <= "01";

A_tb <= "1101";

B_tb <= "0010";

wait for 100 ns;

S_tb <= "10";

A_tb <= "1011";

B_tb <= "1011";

wait for 100 ns;

S_tb <= "11";

A_tb <= "0111";

B_tb <= "1110";

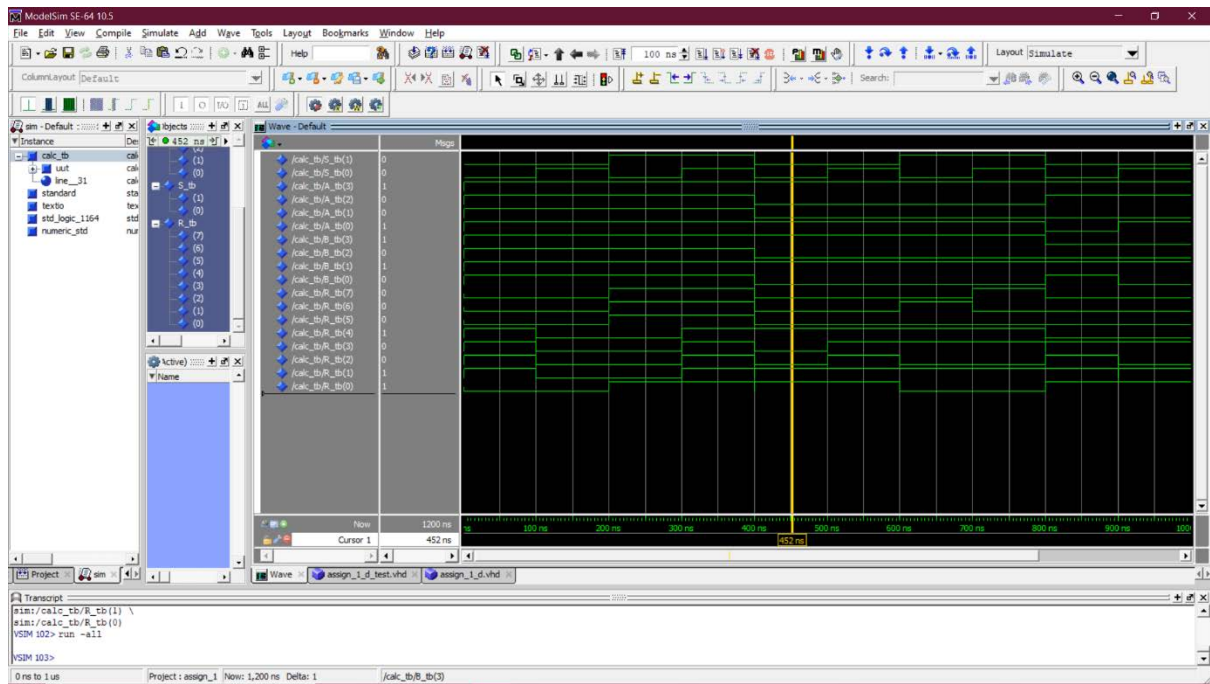
wait for 100 ns;

wait;

end process;

END;

Output:



Q.2 Write a VHDL code generates parity as 1 if the number of 1's a 4-bit long binary message is even and generates 0 otherwise. Use logic gates to design such parity generator. Also, write a VHDL code designing a circuit for parity checker which consider the 5-bit long binary signal (4-bit message + 1-bit parity) and generates 1 if the number of 1's in the 5-bit binary signal is even and 0 otherwise.

Motivation: If the parity checker generates '1' that means the message signal has error in it.

Assumption: Noise can change only 1-bit of the signal

A.2 =>

Part 1: 4-bit parity checker

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

entity parity is
port(
A: in std_logic_vector(3 downto 0);
R: out std_logic
);
end entity parity;

architecture Behavioral of parity is
begin

R <= NOT(A(3) XOR A(2) XOR A(1) XOR A(0));

end architecture Behavioral;
```

Test Bench:

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

entity parity_tb is
end parity_tb;
```

architecture behavior OF parity_tb is

COMPONENT parity

port(

A: in std_logic_vector(3 downto 0);

R: out std_logic

);

END COMPONENT;

signal A_tb : std_logic_vector(3 downto 0) := (others => '0');

signal R_tb : std_logic;

BEGIN

uut: parity PORT MAP (

A => A_tb,

R => R_tb

);

process

begin

A_tb <= "0000";

wait for 100 ns;

A_tb <= "0001";

wait for 100 ns;

A_tb <= "0010";

wait for 100 ns;

A_tb <= "0011";

```
wait for 100 ns;  
A_tb <= "0100";
```

```
wait for 100 ns;  
A_tb <= "0101";
```

```
wait for 100 ns;  
A_tb <= "0111";
```

```
wait for 100 ns;  
A_tb <= "1000";
```

```
wait for 100 ns;  
A_tb <= "1010";
```

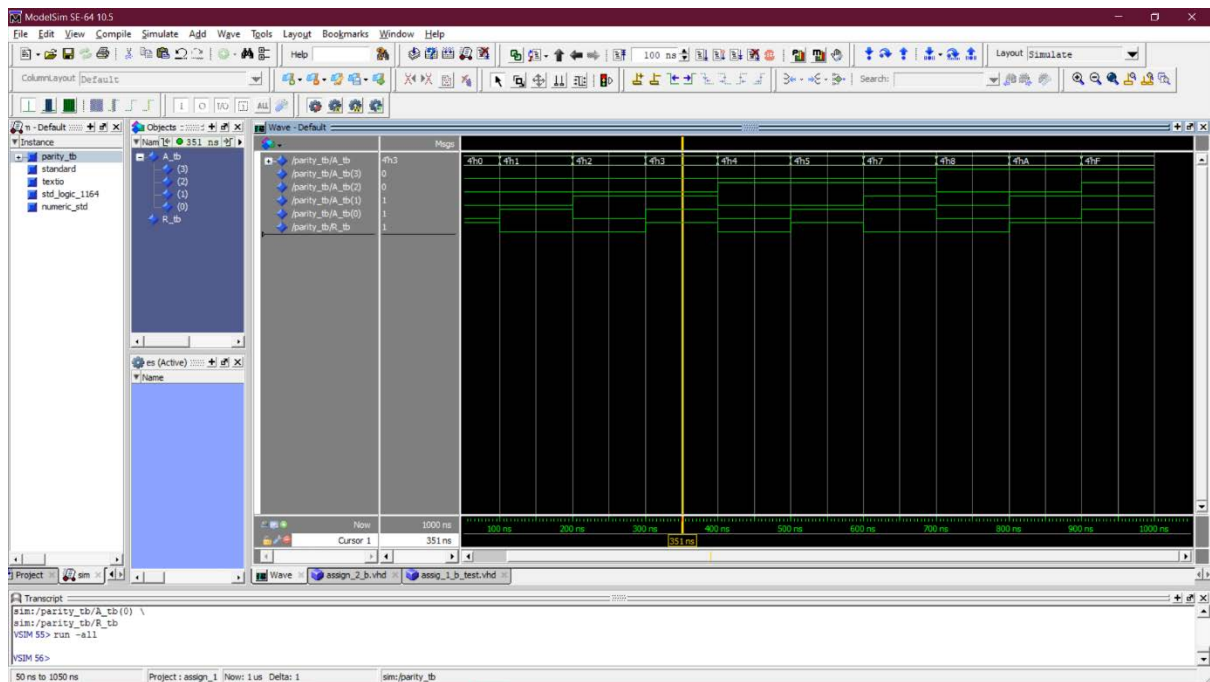
```
wait for 100 ns;  
A_tb <= "1111";
```

```
wait for 100 ns;  
wait;
```

```
end process;
```

```
END;
```

Output:



Part 2: 5-bit parity adder

```
library IEEE;
```

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

```
use IEEE.NUMERIC_STD.ALL;
```

```
entity parity2 is
```

```
port(
```

```
A: in std_logic_vector(3 downto 0);
```

```
R: out std_logic
```

```
);
```

```
end entity parity2;
```

```
architecture Behavioral of parity2 is
```

```
signal B : std_logic;
```

```
begin
```

```
B <= NOT(A(3) XOR A(2) XOR A(1) XOR A(0));
```

```
R <= NOT(B XOR A(3) XOR A(2) XOR A(1) XOR A(0));
```

```
end architecture Behavioral;
```

Test Bench:

```
library ieee;
```

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

```
entity parity2_tb is
```

```
end parity2_tb;
```

```
architecture behavior OF parity2_tb is
```

```
COMPONENT parity2
```

```
port(
```

```
A: in std_logic_vector(3 downto 0);
```

```
R: out std_logic
```

```
);
```

```
END COMPONENT;
```

```
signal A_tb : std_logic_vector(3 downto 0) := (others => '0');
```

```
signal R_tb : std_logic;
```

```
BEGIN
```

```
uut: parity2 PORT MAP (
```

```
A => A_tb,
```

```
R => R_tb
```

```
);
```

```
process
```

```
begin
```

```
A_tb <= "0000";
```

```
wait for 100 ns;
```

```
A_tb <= "0001";
```

```
wait for 100 ns;
```

```
A_tb <= "0010";
```

```
wait for 100 ns;
```

```
A_tb <= "0011";
```

```
wait for 100 ns;
```

```
A_tb <= "0100";
```

```
wait for 100 ns;
```

```
A_tb <= "0101";
```

```
wait for 100 ns;
```

```
A_tb <= "0111";
```

```
wait for 100 ns;
```

```
A_tb <= "1000";
```

```
wait for 100 ns;
```

```
A_tb <= "1010";
```

```
wait for 100 ns;
```

```
A_tb <= "1111";
```

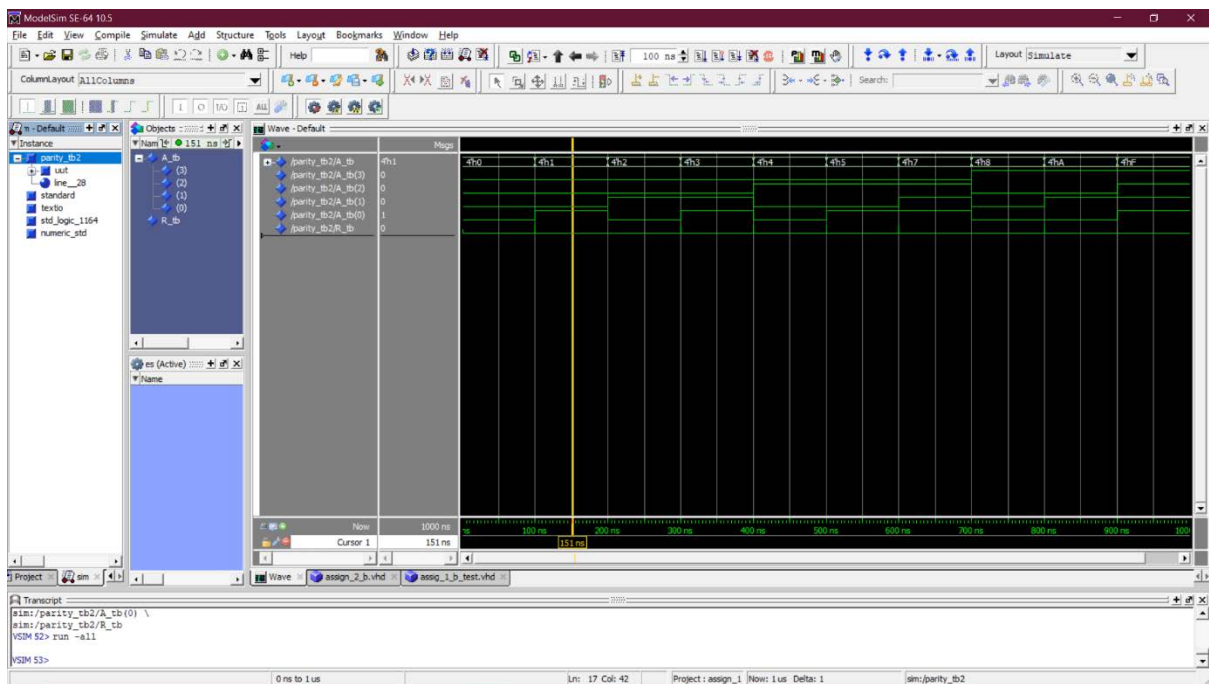
```
wait for 100 ns;
```

```
wait;
```

```
end process;
```

```
END;
```


Output:



Q.3 Write a VHDL code for a 4X1 MUX. Write a test bench and observe the output. Implement the following function using the 4X1 MUX as component in the code.

$$F(A,B,C,D) = \sum(1,2,4,7,10,12,14,15)$$

A.3 =>

Part 1: 4X1 MUX

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

entity MUX is

Port(

S : in STD_LOGIC_VECTOR (1 downto 0);

I : in STD_LOGIC_VECTOR (3 downto 0);

Y : out STD_LOGIC);

end MUX;

architecture Behavioral of MUX is

begin

process (S,I)

begin

if (S <= "00") then

Y <= I(0);

elsif (S <= "01") then

Y <= I(1);

```
elsif (S <= "10") then
```

```
Y <= I(2);
```

```
else
```

```
Y <= I(3);
```

```
end if;
```

```
end process;
```

```
end Behavioral;
```

Test Bench:

```
library ieee;
```

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

```
entity MUX_tb is
```

```
end MUX_tb;
```

```
architecture behavior OF MUX_tb is
```

```
COMPONENT MUX
```

```
Port(
```

```
S : in STD_LOGIC_VECTOR (1 downto 0);
```

```
I : in STD_LOGIC_VECTOR (3 downto 0);
```

```
Y : out STD_LOGIC);
```

```
END COMPONENT;
```

```
signal S_tb : std_logic_vector(1 downto 0) := (others => '0');
```

```
signal I_tb : std_logic_vector(3 downto 0) := (others => '0');
```

```
signal Y_tb : std_logic;
```

```
BEGIN
```

```
uut: MUX PORT MAP (
```

```
S => S_tb,
```

```
I => I_tb,
```

```
Y => Y_tb
```

```
);
```

```
process
```

```
begin
```

```
S_tb <= "00";
```

```
I_tb <= "0000";
```

```
wait for 100 ns;
```

```
S_tb <= "01";
```

```
I_tb <= "0000";
```

```
wait for 100 ns;
```

```
S_tb <= "10";
```

```
I_tb <= "0000";
```

```
wait for 100 ns;
```

```
S_tb <= "11";
```

```
I_tb <= "0000";
```

```
wait for 100 ns;
```

```
S_tb <= "00";
```

```
I_tb <= "1001";
```

```
wait for 100 ns;
```

```
S_tb <= "01";
```

```
I_tb <= "0111";
```

```
wait for 100 ns;
```

```
S_tb <= "10";
```

```
I_tb <= "0001";
```

```
wait for 100 ns;
```

```
S_tb <= "11";
```

```
I_tb <= "1010";
```

```
wait for 100 ns;
```

```
S_tb <= "11";
```

```
I_tb <= "0111";
```

```
wait for 100 ns;
```

```
S_tb <= "01";
```

```
I_tb <= "1111";
```

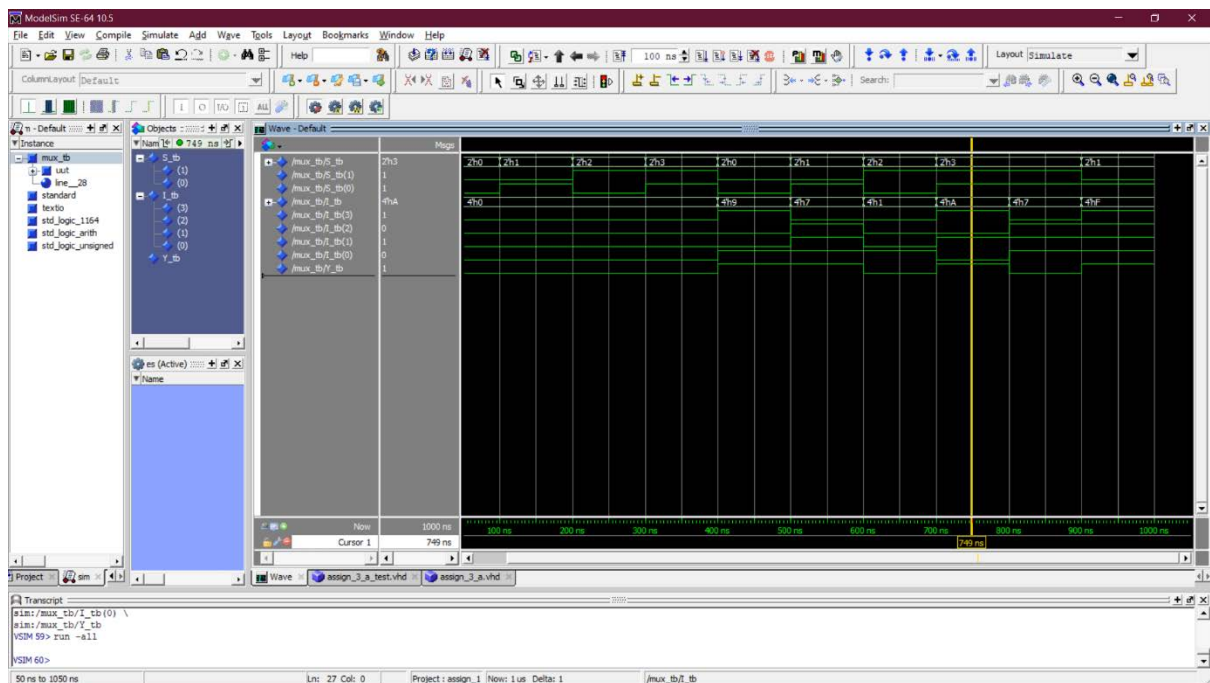
```
wait for 100 ns;
```

```
wait;
```

```
end process;
```

```
END;
```

Output:



Part 2: Implementing function

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

entity MUX is

Port(

S1 : in STD_LOGIC;

S0 : in STD_LOGIC;

I3 : in STD_LOGIC;

I2 : in STD_LOGIC;

I1 : in STD_LOGIC;

I0 : in STD_LOGIC;

Y : out STD_LOGIC);

end MUX;

architecture Behavioral of MUX is

begin

process (S1,S0,I3,I2,I1,I0)

begin

if (S1 = '0' AND S0 = '0') then

Y <= I0;

elsif (S1 = '0' AND S0 = '1') then

Y <= I1;

elsif (S1 = '1' AND S0 = '0') then

Y <= I2;

else

Y <= I3;

end if;

end process;

end Behavioral;

library ieee;

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

```
entity Func is
```

```
Port(
```

```
A : in STD_LOGIC;
```

```
B : in STD_LOGIC;
```

```
C : in STD_LOGIC;
```

```
D : in STD_LOGIC;
```

```
Y0 : out STD_LOGIC);
```

```
end Func;
```

```
architecture Behavioral of Func is
```

```
component MUXX
```

```
Port(
```

```
S1 : in STD_LOGIC;
```

```
S0 : in STD_LOGIC;
```

```
I3 : in STD_LOGIC;
```

```
I2 : in STD_LOGIC;
```

```
I1 : in STD_LOGIC;
```

```
I0 : in STD_LOGIC;
```

```
Y : out STD_LOGIC);
```

```
end component;
```

```
signal I1,I2,I3,I0: std_logic;
```

```
begin
```

```
I0 <= C xor D;
```

```
I1 <= C xnor D;
```

```
I2 <= C and (not D);
```

```
I3 <= C or (not D);
```

```
uut: MUXX port map( A , B, I3 , I2 , I1 , I0 , Y0 );
```


end Behavioral;

Test Bench:

library ieee;

use ieee.std_logic_1164.all;

entity Func_tb is

end Func_tb;

architecture behavior OF Func_tb is

COMPONENT Func

Port(

A : in STD_LOGIC;

B : in STD_LOGIC;

C : in STD_LOGIC;

D : in STD_LOGIC;

Y0 : out STD_LOGIC);

END COMPONENT;

signal A_tb : std_logic := '0';

signal B_tb : std_logic := '0';

signal C_tb : std_logic := '0';

signal D_tb : std_logic := '0';

signal Y0_tb : std_logic;

BEGIN

uut: Func PORT MAP (

A => A_tb,

B => B_tb,

```
C => C_tb,  
D => D_tb,  
Y0 => Y0_tb  
);  
  
process  
begin  
  
A_tb <= '0'; B_tb <= '0'; C_tb <= '0'; D_tb <= '0';  
  
wait for 100 ns;  
A_tb <= '0'; B_tb <= '0'; C_tb <= '0'; D_tb <= '1';  
  
wait for 100 ns;  
A_tb <= '0'; B_tb <= '0'; C_tb <= '1'; D_tb <= '1';  
  
wait for 100 ns;  
A_tb <= '1'; B_tb <= '0'; C_tb <= '0'; D_tb <= '0';  
  
wait for 100 ns;  
A_tb <= '0'; B_tb <= '0'; C_tb <= '0'; D_tb <= '1';  
  
wait for 100 ns;  
A_tb <= '0'; B_tb <= '1'; C_tb <= '0'; D_tb <= '1';  
  
wait for 100 ns;  
A_tb <= '0'; B_tb <= '1'; C_tb <= '1'; D_tb <= '0';  
  
wait for 100 ns;  
A_tb <= '1'; B_tb <= '0'; C_tb <= '0'; D_tb <= '1';  
  
wait for 100 ns;  
A_tb <= '1'; B_tb <= '0'; C_tb <= '1'; D_tb <= '0';
```

wait for 100 ns;

A_tb <= '1'; B_tb <= '1'; C_tb <= '1'; D_tb <= '1';

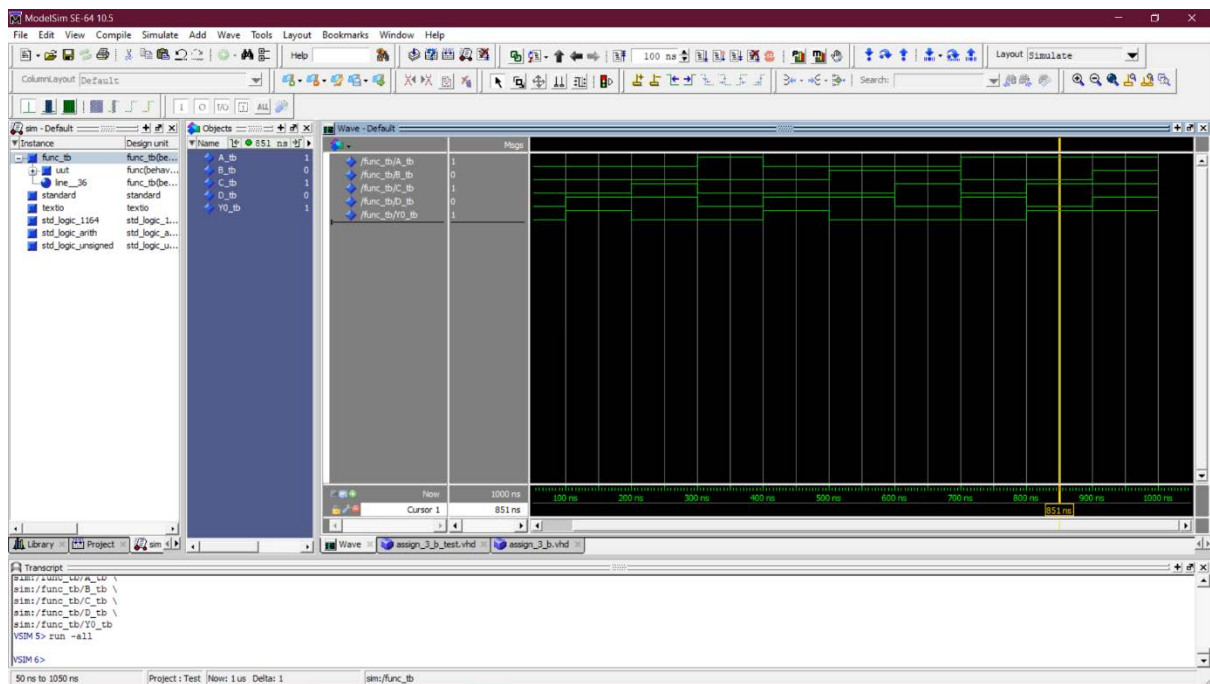
wait for 100 ns;

wait;

end process;

END;

Output:



Q.4 Write a VHDL code for a D-flipflop using behavioural modelling. Use the D-Flipflop as a component to design a 4-bit universal shift register that does the following operation:

A B Operation 1 1 Loads Parallel Data 1 0 Circular Left Shift 0 1 Circular Right Shift

Write a testbench along with a suitable clock input to observe the output.

A.4 =>

Part 1: D FlipFlop

```
library ieee;
```

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

```
entity D_FlipFlop is
```

```
port(
```

```
  I: in std_logic;
```

```
  clk: in std_logic;
```

```
  rst: in std_logic;
```

```
  prst: in std_logic;
```

```
  Y: out std_logic);
```

```
end D_FlipFlop;
```

```
architecture behavioral of D_FlipFlop is
```

```
begin
```

```
  process(prst,rst,clk,I)
```

```
  begin
```

```
    if (rst='1') then
```

```
      Y <= '0';
```

```
    elsif (prst='1') then
```

```
      Y <= '1' ;
```

```
    elsif(rising_edge(clk)) then
```

```
      Y <= I;
```

```
    end if;
```

```
  end process;
```

```
end behavioral;
```

Test Bench:

LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

ENTITY D_FlipFlop_tb IS

END D_FlipFlop_tb;

ARCHITECTURE behavior OF D_FlipFlop_tb IS

COMPONENT D_FlipFlop

port(

I: in std_logic;

clk: in std_logic;

rst: in std_logic;

prst: in std_logic;

Y: out std_logic);

END COMPONENT;

signal I_tb : std_logic := '0';

signal clk_tb : std_logic := '0';

signal rst_tb : std_logic := '1';

signal prst_tb : std_logic := '0';

signal Y_tb : std_logic;

constant clk_period : time := 20 ns;

signal ctr : integer := 0;

BEGIN

uut: D_FlipFlop PORT MAP (

I => I_tb,

```
clk => clk_tb,  
rst => rst_tb,  
prst => prst_tb,  
Y => Y_tb  
);
```

```
clk_process :process  
begin  
clk_tb <= '0';  
wait for clk_period/2;  
clk_tb <= '1';  
wait for clk_period/2;  
ctr <= ctr+20;  
if (ctr = 380) then  
wait;  
end if;  
end process;
```

```
stim_proc: process  
begin
```

```
rst_tb <= '1';  
prst_tb <= '0';  
wait for 100 ns;
```

```
rst_tb <= '0';  
prst_tb <= '1';  
wait for 100 ns;
```

```
rst_tb <= '0';  
prst_tb <= '0';  
l_tb <= '0';  
wait for 100 ns;
```

```

rst_tb <= '0';

prst_tb <= '0';

l_tb <= '1';

```

```

wait for 100 ns;

wait;

```

```

end process;

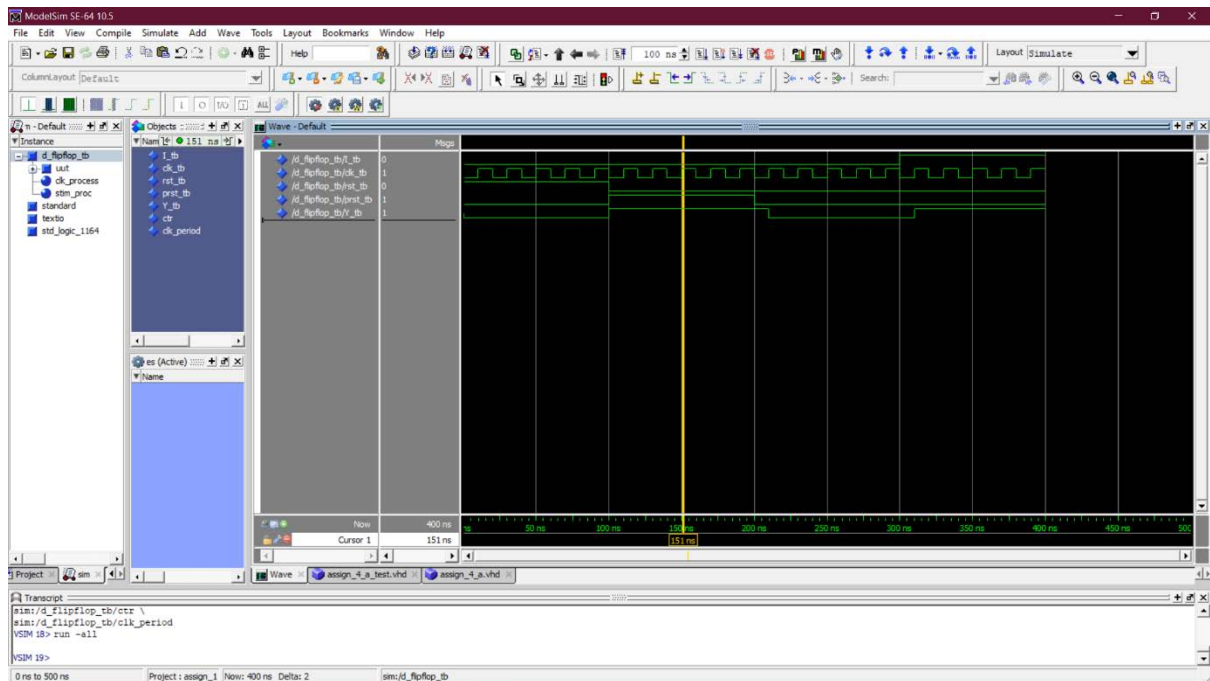
```

```

END;

```

Output:



Part 2: Circular Register

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

```
entity D_FlipFlop is  
port(  
    I: in std_logic;  
    clk: in std_logic;  
    rst: in std_logic;  
    prst: in std_logic;  
    YD: out std_logic);  
end D_FlipFlop;
```

architecture behavioral of D_FlipFlop is

```
begin  
    process(prst,rst,clk,I)  
    begin  
        if (rst='1') then  
            YD <= '0';  
        elsif (prst='1') then  
            YD <= '1' ;  
        elsif(rising_edge(clk)) then  
            YD <= I;  
        end if;  
    end process;  
  
end behavioral;
```

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



```
use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity MUX is
```

```
Port(
S1 : in STD_LOGIC;
S0 : in STD_LOGIC;
I3 : in STD_LOGIC;
I2 : in STD_LOGIC;
I1 : in STD_LOGIC;
I0 : in STD_LOGIC;
YM : out STD_LOGIC);
end MUX;
```

```
architecture Behavioral of MUX is
```

```
begin
process (S1,S0,I3,I2,I1,I0)
begin
if (S1 = '0' AND S0 = '0') then
YM <= I0;
elsif (S1 = '0' AND S0 = '1') then
YM <= I1;
elsif (S1 = '1' AND S0 = '0') then
YM <= I2;
else
YM <= I3;
end if;
end process;
end Behavioral;
```

```
library ieee;

use ieee.std_logic_1164.all;
```

entity Cir_Register is

Port(

Inp : in std_logic_vector(3 downto 0);

Q: out std_logic_vector(3 downto 0);

S : in std_logic_vector(1 downto 0);

clk: in std_logic);

end Cir_Register;

architecture Behavioral of Cir_Register is

component D_FlipFlop

port(

I: in std_logic;

clk: in std_logic;

rst: in std_logic;

prst: in std_logic;

YD: out std_logic);

end component;

component MUX

Port(

S1 : in STD_LOGIC;

S0 : in STD_LOGIC;

I3 : in STD_LOGIC;

I2 : in STD_LOGIC;

I1 : in STD_LOGIC;

I0 : in STD_LOGIC;

YM : out STD_LOGIC);

end component;

```

signal Y : std_logic_vector(3 downto 0) := (others => '0');
signal QM : std_logic_vector(3 downto 0) := (others => '0');

begin

M3: MUX port map( S(1) , S(0), Inp(3) , QM(2) , QM(0) , '0' , Y(3) );
M2: MUX port map( S(1) , S(0), Inp(2) , QM(1) , QM(3) , '0' , Y(2) );
M1: MUX port map( S(1) , S(0), Inp(1) , QM(0) , QM(2) , '0' , Y(1) );
M0: MUX port map( S(1) , S(0), Inp(0) , QM(3) , QM(1) , '0' , Y(0) );


D3: D_FlipFlop PORT MAP( Y(3) , clk , '0' , '0' , QM(3));
D2: D_FlipFlop PORT MAP( Y(2) , clk , '0' , '0' , QM(2));
D1: D_FlipFlop PORT MAP( Y(1) , clk , '0' , '0' , QM(1));
D0: D_FlipFlop PORT MAP( Y(0) , clk , '0' , '0' , QM(0));


Q <= QM;

end Behavioral;

```

Test Bench:

```

LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

ENTITY Cir_Register_tb IS

END Cir_Register_tb;

ARCHITECTURE behavior OF Cir_Register_tb IS

COMPONENT Cir_Register

Port(

Inp : in std_logic_vector(3 downto 0);

```

```
Q: out std_logic_vector(3 downto 0);
```

```
S : in std_logic_vector(1 downto 0);
```

```
clk: in std_logic);
```

```
END COMPONENT;
```

```
signal Inp_tb : std_logic_vector(3 downto 0) := (others => '0');
```

```
signal S_tb : std_logic_vector(1 downto 0) := (others => '0');
```

```
signal Q_tb: std_logic_vector(3 downto 0);
```

```
signal clk_tb: std_logic := '0';
```

```
constant clk_period : time := 20 ns;
```

```
signal ctr : integer := 0;
```

```
BEGIN
```

```
uut: Cir_Register PORT MAP(
```

```
Inp => Inp_tb,
```

```
clk => clk_tb,
```

```
S => S_tb,
```

```
Q => Q_tb
```

```
);
```

```
clk_process :process
```

```
begin
```

```
clk_tb <= '0';
```

```
wait for clk_period/2;
```

```
clk_tb <= '1';
```

```
wait for clk_period/2;
```

```
ctr <= ctr+20;
```

```
if (ctr = 480) then
```

```
wait;
```

```
end if;
```

```
end process;
```

```
stim_proc: process
```

```
begin
```

```
S_tb <= "11";
```

```
Inp_tb <= "1000";
```

```
wait for 50 ns;
```

```
S_tb <= "01";
```

```
wait for 100 ns;
```

```
S_tb <= "10";
```

```
wait for 100 ns;
```

```
S_tb <= "11";
```

```
Inp_tb <= "1001";
```

```
wait for 50 ns;
```

```
S_tb <= "01";
```

```
wait for 100 ns;
```

```
S_tb <= "10";
```

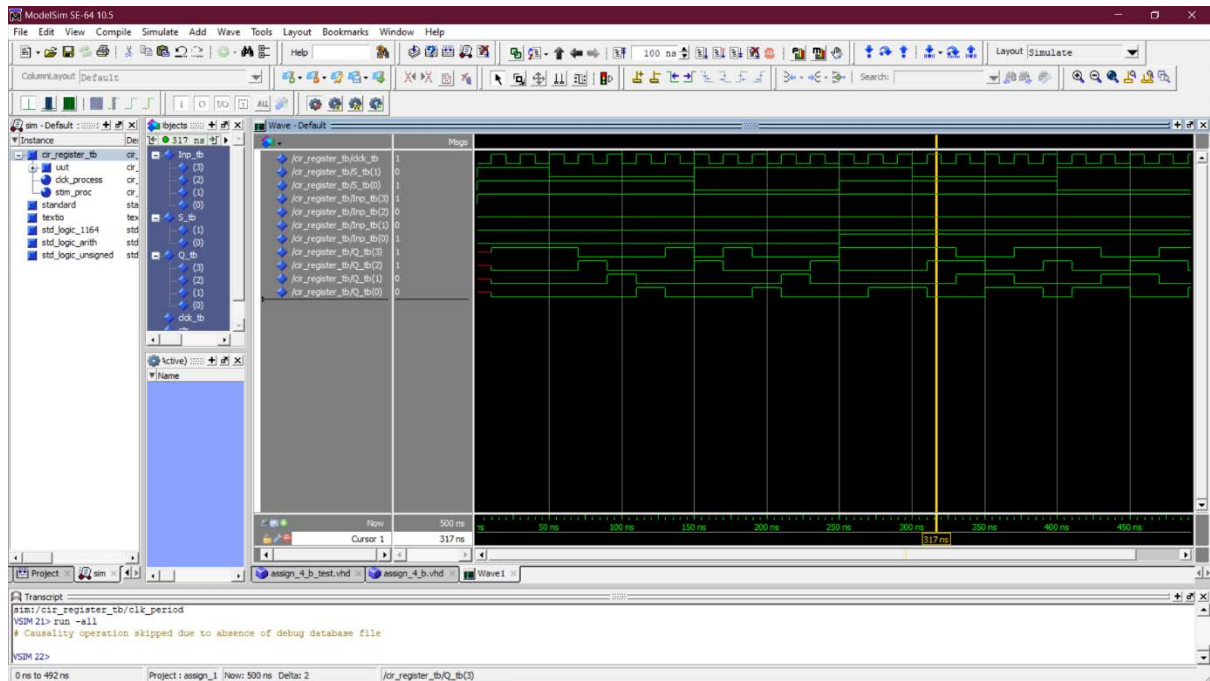
```
wait for 100 ns;
```

```
wait;
```

```
end process;
```

```
END;
```

Output:



Extras:

1. 4-bit Adder

Code

```
library ieee;

use ieee.std_logic_1164.all; -- Highlights

entity full_adder is
Port(
A : in STD_LOGIC;
B : in STD_LOGIC;
Cin : in STD_LOGIC;
S : out STD_LOGIC;
Cout : out STD_LOGIC);
end full_adder;

architecture dataflow of full_adder is
begin

S <= A XOR B XOR Cin ;
Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;

end dataflow;

library ieee;

use ieee.std_logic_1164.all; -- Highlights

entity adder is
Port ( A3 : in STD_LOGIC;
A2 : in STD_LOGIC;
A1 : in STD_LOGIC;
A0 : in STD_LOGIC;
```

```
B3 : in STD_LOGIC;
B2 : in STD_LOGIC;
B1 : in STD_LOGIC;
B0 : in STD_LOGIC;
S3 : out STD_LOGIC;
S2 : out STD_LOGIC;
S1 : out STD_LOGIC;
S0 : out STD_LOGIC;
Cout : out STD_LOGIC);
end adder;
```

architecture Behavioral of adder is

```
component full_adder
Port ( A : in STD_LOGIC;
B : in STD_LOGIC;
Cin : in STD_LOGIC;
S : out STD_LOGIC;
Cout : out STD_LOGIC);
end component;
```

```
signal c1,c2,c3: STD_LOGIC;
signal c0 : std_logic := '0';
```

```
begin
```

```
FA1: full_adder port map( A0, B0, c0, S0, c1);
FA2: full_adder port map( A1, B1, c1, S1, c2);
FA3: full_adder port map( A2, B2, c2, S2, c3);
FA4: full_adder port map( A3, B3, c3, S3, Cout);
```

```
end Behavioral;
```

Test bench:

```
library ieee;
```

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

```
entity adder_tb is
```

```
end adder_tb;
```

```
architecture behavior OF adder_tb is
```

```
COMPONENT adder
```

```
Port ( A3 : in STD_LOGIC;
```

```
A2 : in STD_LOGIC;
```

```
A1 : in STD_LOGIC;
```

```
A0 : in STD_LOGIC;
```

```
B3 : in STD_LOGIC;
```

```
B2 : in STD_LOGIC;
```

```
B1 : in STD_LOGIC;
```

```
B0 : in STD_LOGIC;
```

```
S3 : out STD_LOGIC;
```

```
S2 : out STD_LOGIC;
```

```
S1 : out STD_LOGIC;
```

```
S0 : out STD_LOGIC;
```

```
Cout : out STD_LOGIC);
```

```
END COMPONENT;
```

```
signal a3_tb : std_logic := '0';
```

```
signal a2_tb : std_logic := '0';
```

```
signal a1_tb : std_logic := '0';
```

```
signal a0_tb : std_logic := '0';
```

```
signal b3_tb : std_logic := '0';
```

```
signal b2_tb : std_logic := '0';
```

```
signal b1_tb : std_logic := '0';
```

```
signal b0_tb : std_logic := '0';  
signal s3_tb : std_logic;  
signal s2_tb : std_logic;  
signal s1_tb : std_logic;  
signal s0_tb : std_logic;  
signal cout_tb : std_logic;
```

```
BEGIN
```

```
uut: adder PORT MAP (
```

```
A3 => a3_tb,
```

```
A2 => a2_tb,
```

```
A1 => a1_tb,
```

```
A0 => a0_tb,
```

```
B3 => b3_tb,
```

```
B2 => b2_tb,
```

```
B1 => b1_tb,
```

```
B0 => b0_tb,
```

```
S3 => s3_tb,
```

```
S2 => s2_tb,
```

```
S1 => s1_tb,
```

```
S0 => s0_tb,
```

```
Cout => cout_tb
```

```
);
```

```
process
```

```
begin
```

```
a3_tb <= '0'; a2_tb <= '0'; a1_tb <= '0'; a0_tb <= '0';
```

```
b3_tb <= '0'; b2_tb <= '0'; b1_tb <= '0'; b0_tb <= '0';
```

```
wait for 100 ns;
```

```
a3_tb <= '1'; a2_tb <= '1'; a1_tb <= '1'; a0_tb <= '1';
```

b3_tb <= '0'; b2_tb <= '0'; b1_tb <= '0'; b0_tb <= '0';

wait for 100 ns;

a3_tb <= '1'; a2_tb <= '1'; a1_tb <= '1'; a0_tb <= '1';

b3_tb <= '1'; b2_tb <= '1'; b1_tb <= '1'; b0_tb <= '1';

wait for 100 ns;

a3_tb <= '1'; a2_tb <= '1'; a1_tb <= '1'; a0_tb <= '1';

b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '1'; b0_tb <= '0';

wait for 100 ns;

a3_tb <= '0'; a2_tb <= '0'; a1_tb <= '0'; a0_tb <= '1';

b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '1'; b0_tb <= '1';

wait for 100 ns;

a3_tb <= '0'; a2_tb <= '1'; a1_tb <= '0'; a0_tb <= '1';

b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '0'; b0_tb <= '1';

wait for 100 ns;

a3_tb <= '1'; a2_tb <= '1'; a1_tb <= '1'; a0_tb <= '0';

b3_tb <= '0'; b2_tb <= '1'; b1_tb <= '1'; b0_tb <= '0';

wait for 100 ns;

a3_tb <= '1'; a2_tb <= '1'; a1_tb <= '0'; a0_tb <= '0';

b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '1'; b0_tb <= '0';

wait for 100 ns;

a3_tb <= '0'; a2_tb <= '0'; a1_tb <= '1'; a0_tb <= '1';

b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '1'; b0_tb <= '1';

wait for 100 ns;

a3_tb <= '0'; a2_tb <= '1'; a1_tb <= '0'; a0_tb <= '1';

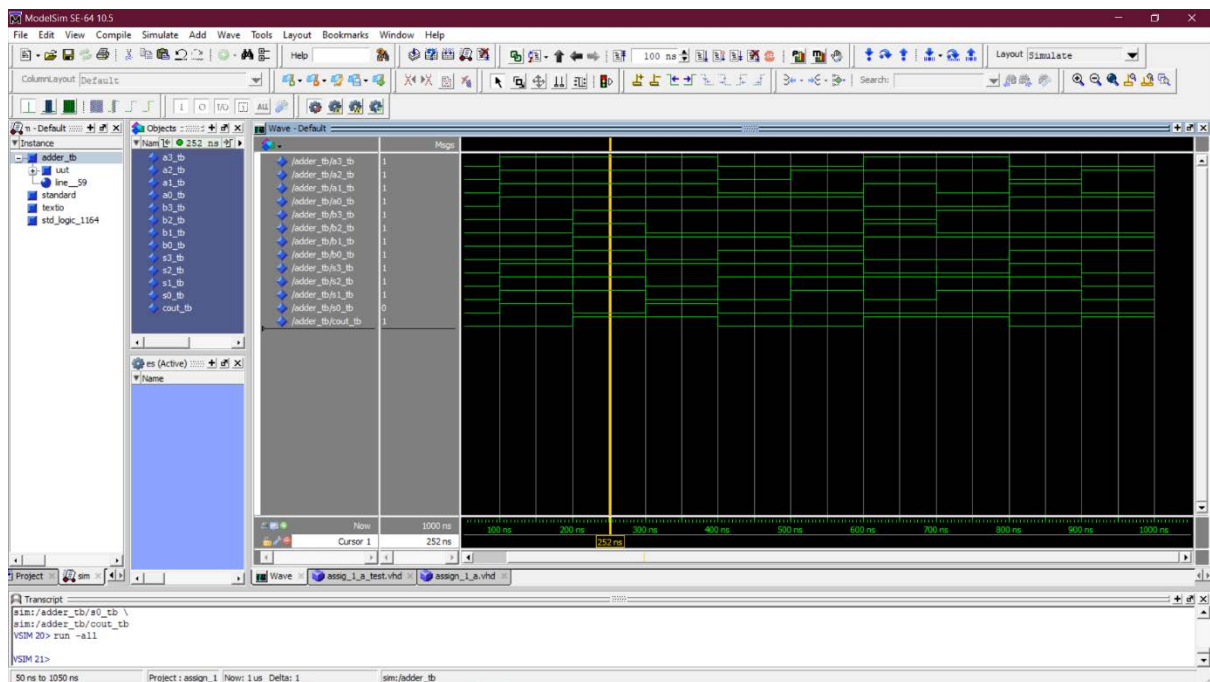
b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '1'; b0_tb <= '1';

```
wait;
```

end process;

END;

Output:



2. 4-bit Subtractor

Code

```
library ieee;
```

```
use ieee.std_logic_1164.all; -- Highlights
```

```
entity full_sub is
```

```
Port(
```

```
A : in STD_LOGIC;
```

```
B : in STD_LOGIC;
```

```
Cin : in STD_LOGIC;
```

```
S : out STD_LOGIC;
```

```
Cout : out STD_LOGIC);
```

```
end full_sub;
```

```
architecture dataflow of full_sub is
```

```
begin
```

```
S <= A XOR B XOR Cin ;
```

```
Cout <= ((not A) AND B) OR (Cin AND (not A)) OR (Cin AND B) ;
```

```
end dataflow;
```

```
library ieee;
```

```
use ieee.std_logic_1164.all; -- Highlights
```

```
entity sub is
```

```
Port(
```

```
A3 : in STD_LOGIC;
```

```
A2 : in STD_LOGIC;
```

```
A1 : in STD_LOGIC;
```

```
A0 : in STD_LOGIC;
```

```
B3 : in STD_LOGIC;
```

```
B2 : in STD_LOGIC;
B1 : in STD_LOGIC;
B0 : in STD_LOGIC;
S3 : out STD_LOGIC;
S2 : out STD_LOGIC;
S1 : out STD_LOGIC;
S0 : out STD_LOGIC;
Cout : out STD_LOGIC);
end sub;
```

architecture Behavioral of sub is

```
component full_sub
Port(
A : in STD_LOGIC;
B : in STD_LOGIC;
Cin : in STD_LOGIC;
S : out STD_LOGIC;
Cout : out STD_LOGIC);
end component;
```

```
signal c1,c2,c3: STD_LOGIC;
signal c0 : std_logic := '0';
```

```
begin
```

```
FS1: full_sub port map( A0, B0, c0, S0, c1);
FS2: full_sub port map( A1, B1, c1, S1, c2);
FS3: full_sub port map( A2, B2, c2, S2, c3);
FS4: full_sub port map( A3, B3, c3, S3, Cout);
```

```
end Behavioral;
```

Test bench:

```
library ieee;
```

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

```
entity sub_tb is
```

```
end sub_tb;
```

```
architecture behavior OF sub_tb is
```

```
COMPONENT sub
```

```
Port(
```

```
A3 : in STD_LOGIC;
```

```
A2 : in STD_LOGIC;
```

```
A1 : in STD_LOGIC;
```

```
A0 : in STD_LOGIC;
```

```
B3 : in STD_LOGIC;
```

```
B2 : in STD_LOGIC;
```

```
B1 : in STD_LOGIC;
```

```
B0 : in STD_LOGIC;
```

```
S3 : out STD_LOGIC;
```

```
S2 : out STD_LOGIC;
```

```
S1 : out STD_LOGIC;
```

```
S0 : out STD_LOGIC;
```

```
Cout : out STD_LOGIC);
```

```
END COMPONENT;
```

```
signal a3_tb : std_logic := '0';
```

```
signal a2_tb : std_logic := '0';
```

```
signal a1_tb : std_logic := '0';
```

```
signal a0_tb : std_logic := '0';
```

```
signal b3_tb : std_logic := '0';
```

```
signal b2_tb : std_logic := '0';
```

```
signal b1_tb : std_logic := '0';  
signal b0_tb : std_logic := '0';  
signal s3_tb : std_logic;  
signal s2_tb : std_logic;  
signal s1_tb : std_logic;  
signal s0_tb : std_logic;  
signal cout_tb : std_logic;
```

```
BEGIN
```

```
uut: sub PORT MAP (
```

```
A3 => a3_tb,
```

```
A2 => a2_tb,
```

```
A1 => a1_tb,
```

```
A0 => a0_tb,
```

```
B3 => b3_tb,
```

```
B2 => b2_tb,
```

```
B1 => b1_tb,
```

```
B0 => b0_tb,
```

```
S3 => s3_tb,
```

```
S2 => s2_tb,
```

```
S1 => s1_tb,
```

```
S0 => s0_tb,
```

```
Cout => cout_tb
```

```
);
```

```
process
```

```
begin
```

```
a3_tb <= '0'; a2_tb <= '0'; a1_tb <= '0'; a0_tb <= '0';
```

```
b3_tb <= '0'; b2_tb <= '0'; b1_tb <= '0'; b0_tb <= '0';
```

```
wait for 100 ns;
```



```
a3_tb <= '1'; a2_tb <= '1'; a1_tb <= '1'; a0_tb <= '1';  
b3_tb <= '0'; b2_tb <= '0'; b1_tb <= '0'; b0_tb <= '0';
```

```
wait for 100 ns;
```

```
a3_tb <= '1'; a2_tb <= '1'; a1_tb <= '1'; a0_tb <= '1';  
b3_tb <= '1'; b2_tb <= '1'; b1_tb <= '1'; b0_tb <= '1';
```

```
wait for 100 ns;
```

```
a3_tb <= '1'; a2_tb <= '1'; a1_tb <= '1'; a0_tb <= '1';  
b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '1'; b0_tb <= '0';
```

```
wait for 100 ns;
```

```
a3_tb <= '0'; a2_tb <= '0'; a1_tb <= '0'; a0_tb <= '1';  
b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '1'; b0_tb <= '1';
```

```
wait for 100 ns;
```

```
a3_tb <= '0'; a2_tb <= '1'; a1_tb <= '0'; a0_tb <= '1';  
b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '0'; b0_tb <= '1';
```

```
wait for 100 ns;
```

```
a3_tb <= '1'; a2_tb <= '1'; a1_tb <= '1'; a0_tb <= '0';  
b3_tb <= '0'; b2_tb <= '1'; b1_tb <= '1'; b0_tb <= '0';
```

```
wait for 100 ns;
```

```
a3_tb <= '1'; a2_tb <= '1'; a1_tb <= '0'; a0_tb <= '0';  
b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '1'; b0_tb <= '0';
```

```
wait for 100 ns;
```

```
a3_tb <= '0'; a2_tb <= '0'; a1_tb <= '1'; a0_tb <= '1';  
b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '1'; b0_tb <= '1';
```

```
wait for 100 ns;
```

```
a3_tb <= '0'; a2_tb <= '1'; a1_tb <= '0'; a0_tb <= '1';
```

```
b3_tb <= '1'; b2_tb <= '0'; b1_tb <= '1'; b0_tb <= '1';
```

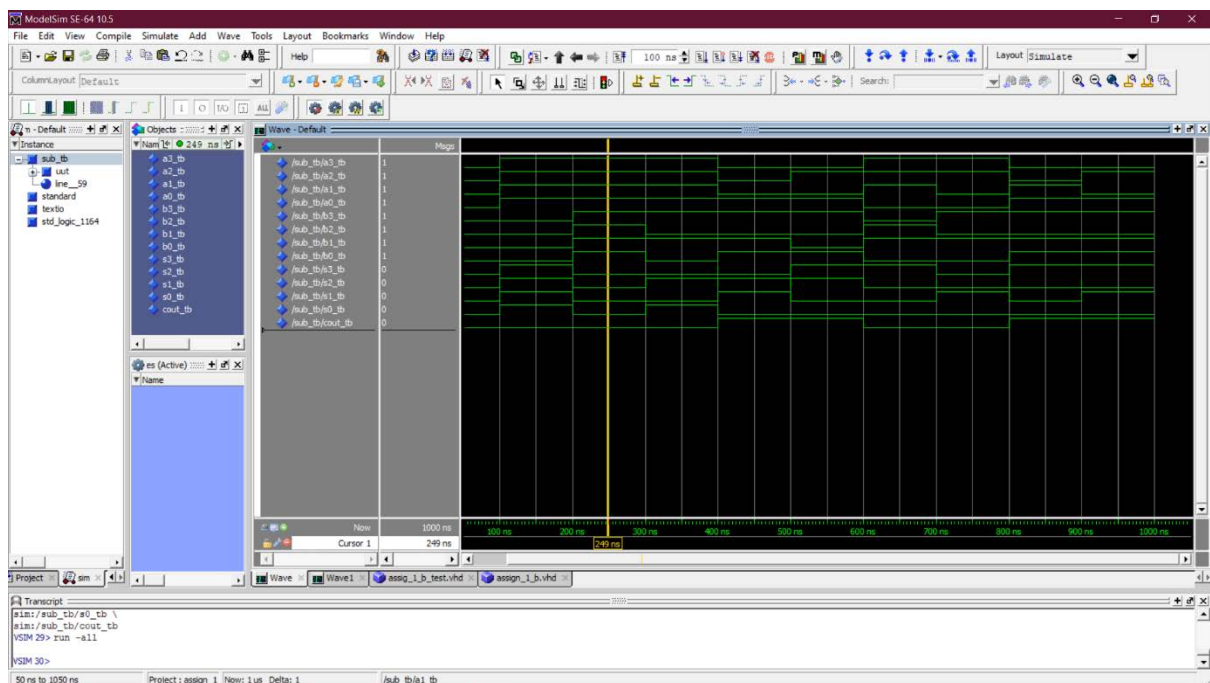
```
wait for 100 ns;
```

```
wait;
```

```
end process;
```

```
END;
```

Output:



3. 4-bit Multiplier

Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

entity Multiplier is

port(

A , B: in std_logic_vector(3 downto 0);

R: out std_logic_vector(7 downto 0)

);

end entity Multiplier;

architecture Behavioral of Multiplier is

begin

R <= std_logic_vector(unsigned(A) * unsigned(B));

end architecture Behavioral;
```

Test bench:

```
library ieee;

use ieee.std_logic_1164.all;

entity Multiplier_tb is

end Multiplier_tb;

architecture behavior OF Multiplier_tb is

COMPONENT Multiplier

port(

A , B: in std_logic_vector(3 downto 0);
```

```
R: out std_logic_vector(7 downto 0)
```

```
);
```

```
END COMPONENT;
```

```
signal A_tb : std_logic_vector(3 downto 0) := (others => '0');
```

```
signal B_tb : std_logic_vector(3 downto 0) := (others => '0');
```

```
signal R_tb : std_logic_vector(7 downto 0);
```

```
BEGIN
```

```
uut: Multiplier PORT MAP(
```

```
A => A_tb,
```

```
B => B_tb,
```

```
R => R_tb
```

```
);
```

```
process
```

```
begin
```

```
A_tb <= "0000";
```

```
B_tb <= "0000";
```

```
wait for 100 ns;
```

```
A_tb <= "1111";
```

```
B_tb <= "1111";
```

```
wait for 100 ns;
```

```
A_tb <= "1111";
```

```
B_tb <= "1111";
```

```
wait for 100 ns;
```

```
A_tb <= "1010";
```

```
B_tb <= "0101";
```

wait for 100 ns;

A_tb <= "1001";

B_tb <= "0110";

wait for 100 ns;

A_tb <= "1100";

B_tb <= "0110";

wait for 100 ns;

A_tb <= "0101";

B_tb <= "0110";

wait for 100 ns;

A_tb <= "0100";

B_tb <= "0101";

wait for 100 ns;

A_tb <= "1110";

B_tb <= "0110";

wait for 100 ns;

A_tb <= "0000";

B_tb <= "0110";

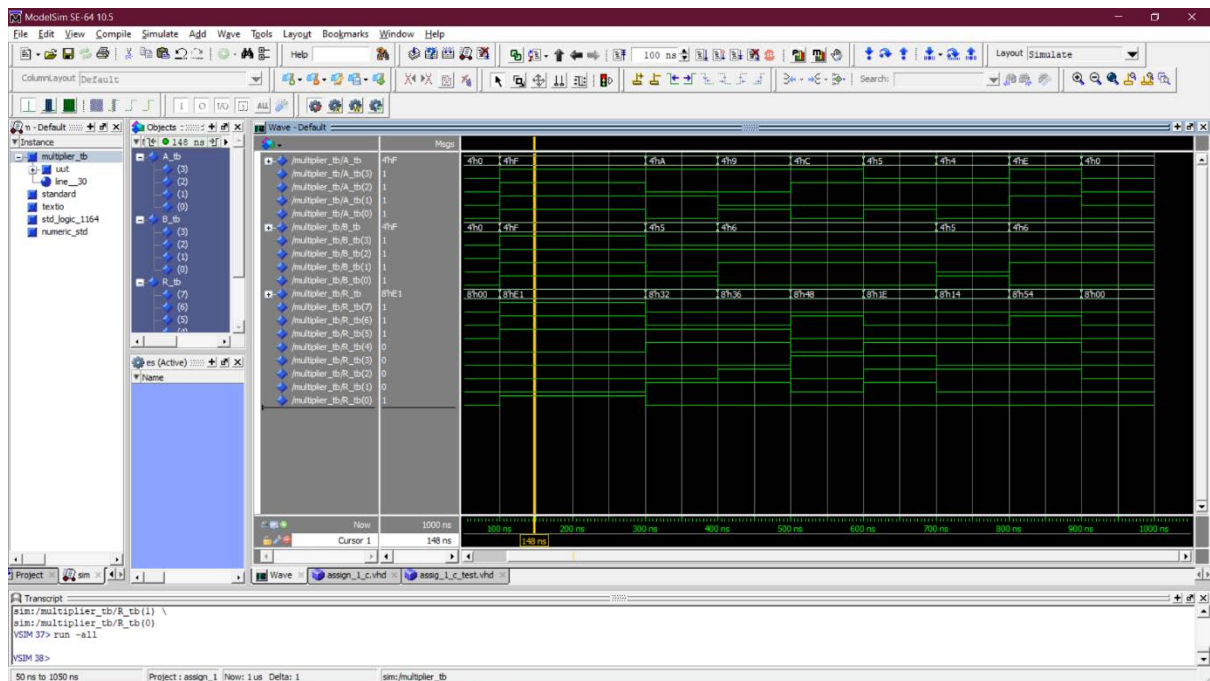
wait for 100 ns;

wait;

end process;

END;

Output:



The End