Name: Ahmed Usama Saad

Section: 1

Designed circuits

1-Adder Substractor Circuit

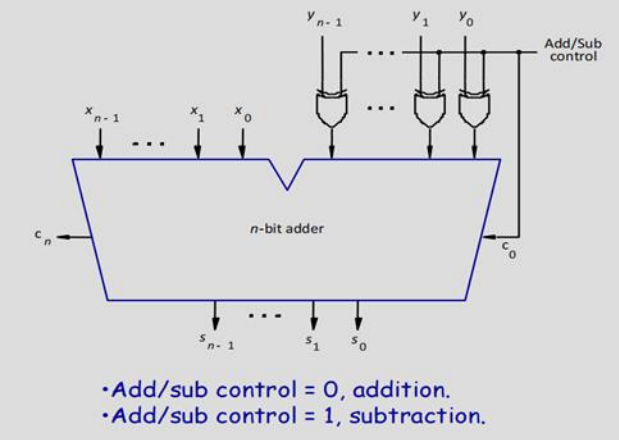
2- Carry Lookahead Adder Circuit

3- Sequential Multiplication Circuit

4- Non Restoring Division Circuit

5- ALU

1- Design a circuit of 8-bit adder/subtractor based on ripple carry adder and and prove the correctness of your code.



Full Adder :

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity \ent_FA\ is
        port(
                x : in STD_LOGIC;
                y : in STD_LOGIC;
                cin : in STD_LOGIC;
                sum : out STD_LOGIC;
                cout : out STD_LOGIC
            );
end \ent_FA\;
architecture \arch_FA\ of \ent_FA\ is
begin
    sum <= (x xor y) xor cin;
    Cout <= (x and (y or cin)) or (cin and y);
```

```
end \arch_FA\;
```

Adder Substractor :

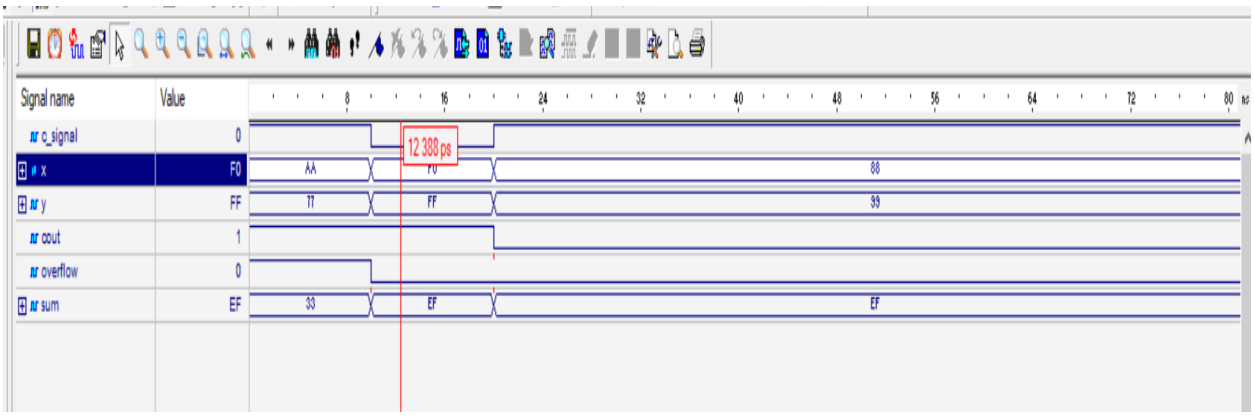
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity \ent_add sub\ is
        port(
                c_signal : in STD_LOGIC;
                x : in STD_LOGIC_VECTOR(7 downto 0);
                y : in STD_LOGIC_VECTOR(7 downto 0);
                cout : out STD_LOGIC;
                overflow : out STD_LOGIC;
                sum : out STD_LOGIC_VECTOR(7 downto 0)
            );
end \ent_add sub\;

architecture \arch_add sub\ of \ent_add sub\ is
  component \ent_FA\ is
        port(
                x : in STD_LOGIC;
                y : in STD_LOGIC;
                cin : in STD_LOGIC;
                sum : out STD_LOGIC;
                cout : out STD_LOGIC
            );
```

```
  end component;
signal C: std_logic_vector(8 downto 1);
signal temp: std_logic_vector(7 downto 0);
begin
temp(0) <= c_signal xor y(0);
temp(1) <= c_signal xor y(1);
temp(2) <= c_signal xor y(2);
temp(3) <= c_signal xor y(3);
temp(4) <= c_signal xor y(4);
temp(5) <= c_signal xor y(5);
temp(6) <= c_signal xor y(6);
temp(7) <= c_signal xor y(7);
FA0:\ent_FA\ port map(x(0),temp(0),c_signal, sum(0),C(1));
FA1:\ent_FA\ port map(x(1),temp(1),c(1), sum(1),C(2));
FA2:\ent_FA\ port map(x(2),temp(2),c(2), sum(2),C(3));
FA3:\ent_FA\ port map(x(3),temp(3),c(3), sum(3),C(4));
FA4:\ent_FA\ port map(x(4),temp(4),c(4), sum(4),C(5));
FA5:\ent_FA\ port map(x(5),temp(5),c(5), sum(5),C(6));
FA6:\ent_FA\ port map(x(6),temp(6),c(6), sum(6),C(7));
FA7:\ent_FA\ port map(x(7),temp(7),c(7), sum(7),C(8));
cout <= c(8);
overflow <= c(7) xor c(8);
end \arch_add sub\;
```

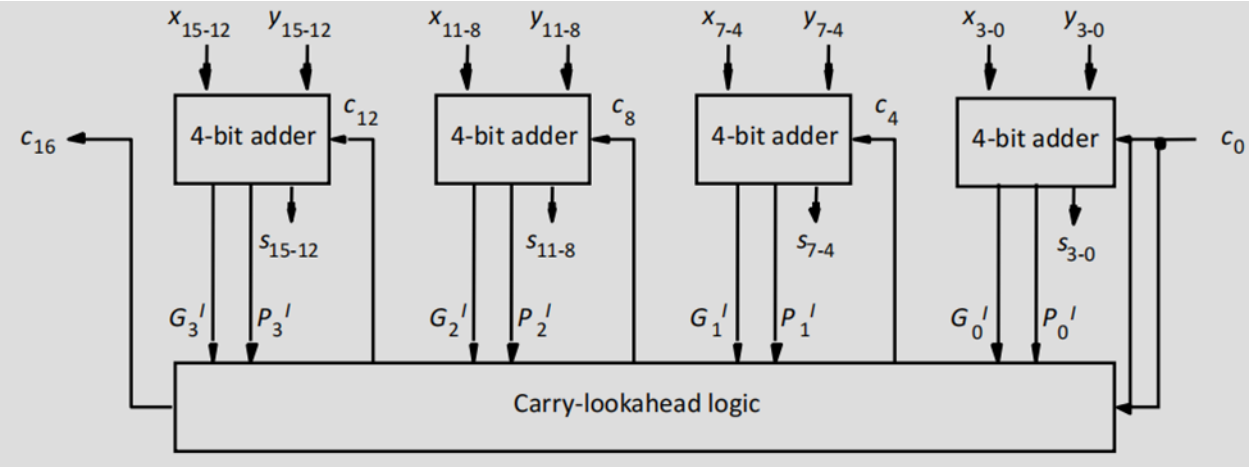
TestBench:

```
process
        begin
                x <= "10101010" ;
                y <= "01110111" ;
                c_signal <= '1' ;
                wait for 10 ns ;
                x <= "11110000" ;
                y <= "11111111" ;
                c_signal <= '0' ;
                wait for 10 ns ;
                x <= "10001000" ;
                 y <= "10011001" ;
                c_signal <= '1' ;
                wait for 10 ns ;
                wait;
  end process;
```

Simulation :



2- Design 16-bit carry look-ahead adder with minimum gate delay that adds A0-15 to B0-15 then produces S0-15 and C16. Write VHDL code required to design this adder using structural modelling and prove the correctness of your code.



Full Adder :

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity ent_FA is
	port(
		x : in STD_LOGIC;
		y : in STD_LOGIC;
		cin : in STD_LOGIC;
		sum : out STD_LOGIC;
		P : out STD_LOGIC;
		G : out STD_LOGIC
	);
end ent_FA;
```

```
architecture arch_FA of ent_FA is
begin
   sum <= x xor y xor cin;
    P <= x xor y;
    G <= x and y;
end arch_FA;
```

---

4 Bit Carry Lookahead Adder :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
 entity ent_carry_lookahead is
Port (
      x : in STD_LOGIC_VECTOR (3 downto 0);
      y : in STD_LOGIC_VECTOR (3 downto 0);
      cin : in STD_LOGIC;
      sum : out STD_LOGIC_VECTOR (3 downto 0);
      Cout : out STD_LOGIC ;
      P0 : out STD_LOGIC ;
      G0 : out STD_LOGIC
            );
end ent_carry_lookahead;

architecture arch_carry_lookahead of ent_carry_lookahead is
 component ent_FA
           Port (
```

```
        x : in STD_LOGIC;
        y : in STD_LOGIC;
        cin : in STD_LOGIC;
        sum : out STD_LOGIC;
        P : out STD_LOGIC;
        G : out STD_LOGIC
            );
 end component;

signal c1,c2,c3: STD_LOGIC;
signal Pi,Gi: STD_LOGIC_VECTOR(3 downto 0);
begin
 F1: ent_FA port map( x(0), y(0), cin, sum(0), Pi(0), Gi(0));
F2: ent_FA port map( x(1), y(1), c1, sum(1), Pi(1), Gi(1));
F3: ent_FA port map( x(2), y(2), c2, sum(2), Pi(2), Gi(2));
F4: ent_FA port map( x(3), y(3), c3, sum(3), Pi(3), Gi(3));
c1 <= Gi(0) OR (Pi(0) AND cin);
c2 <= Gi(1) OR (Pi(1) AND Gi(0)) OR (Pi(1) AND Pi(0) AND cin);
c3 <= Gi(2) OR (Pi(2) AND Gi(1)) OR (Pi(2) AND Pi(1) AND Gi(0)) OR (Pi(2) AND Pi(1)
AND Pi(0) AND cin);
Cout <= Gi(3) OR (Pi(3) AND Gi(2)) OR (Pi(3) AND Pi(2) AND Gi(1)) OR (Pi(3) AND Pi(2)
AND Pi(1) AND Gi(0)) OR (Pi(3) AND Pi(2) AND Pi(1) AND Pi(0) AND cin);
P0 <= Pi(3) AND Pi(2) AND Pi(1) AND Pi(0) AND cin ;
G0 <= Gi(3) OR (Pi(3) AND Gi(2)) OR (Pi(3) AND Pi(2) AND Gi(1)) OR (Pi(3) AND Pi(2)
AND Pi(1) AND Gi(0)) ;
end arch_carry_lookahead;
```

16 Bit Carry Lookahead Adder :

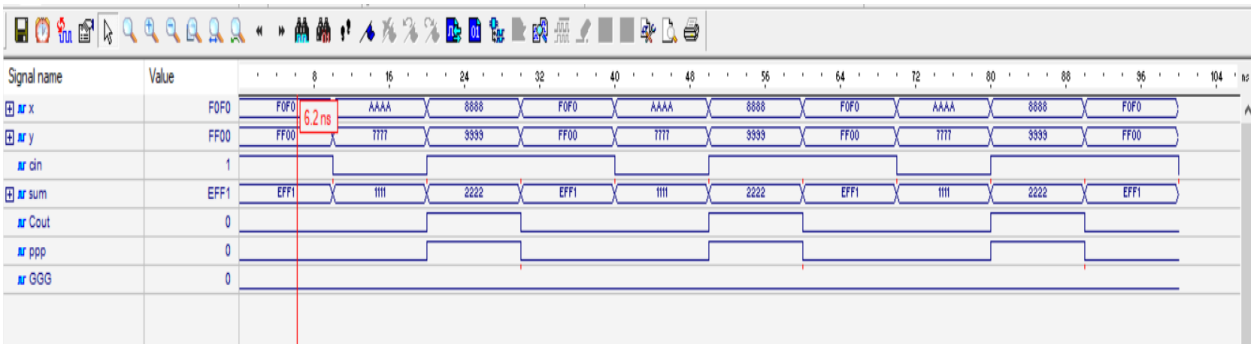
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
 entity ent_16bit_carry_lookahead is
Port (
     x : in STD_LOGIC_VECTOR (15 downto 0);
     y : in STD_LOGIC_VECTOR (15 downto 0);
     cin : in STD_LOGIC;
     sum : out STD_LOGIC_VECTOR (15 downto 0);
     Cout : out STD_LOGIC;
     ppp : out STD_LOGIC;
     GGG : out STD_LOGIC
          );
end ent_16bit_carry_lookahead;

architecture arch_16bit_carry_lookahead of ent_16bit_carry_lookahead is
component ent_carry_lookahead
          Port (
               x : in STD_LOGIC_VECTOR (3 downto 0);
               y : in STD_LOGIC_VECTOR (3 downto 0);
               cin : in STD_LOGIC;
                sum : out STD_LOGIC_VECTOR (3 downto 0);
                Cout : out STD_LOGIC ;
                P0 : out STD_LOGIC ;
```

```
            G0 : out STD_LOGIC

                );

end component;

 signal c4,c8,c12: STD_LOGIC;

signal Pi,Gi: STD_LOGIC_VECTOR(3 downto 0);

begin

B1: ent_carry_lookahead port map( x(3 downto 0), y(3 downto 0), cin, sum(3 downto
0),Pi(0),Gi(0));

B2: ent_carry_lookahead port map( x(7 downto 4), y(7 downto 4), c4, sum(7 downto
4),Pi(1),Gi(1));

B3: ent_carry_lookahead port map( x(11 downto 8), y(11 downto 8), c8, sum(11 downto
8),Pi(2),Gi(2));

B4: ent_carry_lookahead port map( x(15 downto 12), y(15 downto 12), c8, sum(15 downto
12),Pi(3),Gi(3));

 c4 <= Gi(0) OR (Pi(0) AND cin);

c8 <= Gi(1) OR (Pi(1) AND Gi(0)) OR (Pi(1) AND Pi(0) AND cin);

c12 <= Gi(2) OR (Pi(2) AND Gi(1)) OR (Pi(2) AND Pi(1) AND Gi(0)) OR (Pi(2) AND Pi(1)
AND Pi(0) AND cin);

Cout <= Gi(3) OR (Pi(3) AND Gi(2)) OR (Pi(3) AND Pi(2) AND Gi(1)) OR (Pi(3) AND Pi(2)
AND Pi(1) AND Gi(0)) OR (Pi(3) AND Pi(2) AND Pi(1) AND Pi(0) AND cin );

GGG <=Gi(3) OR (Pi(3) AND Gi(2)) OR (Pi(3) AND Pi(2) AND Gi(1)) OR (Pi(3) AND Pi(2)
AND Pi(1) AND Gi(0));

ppp <=Pi(3) AND Pi(2) AND Pi(1) AND Pi(0) AND cin;

end arch_16bit_carry_lookahead ;
```

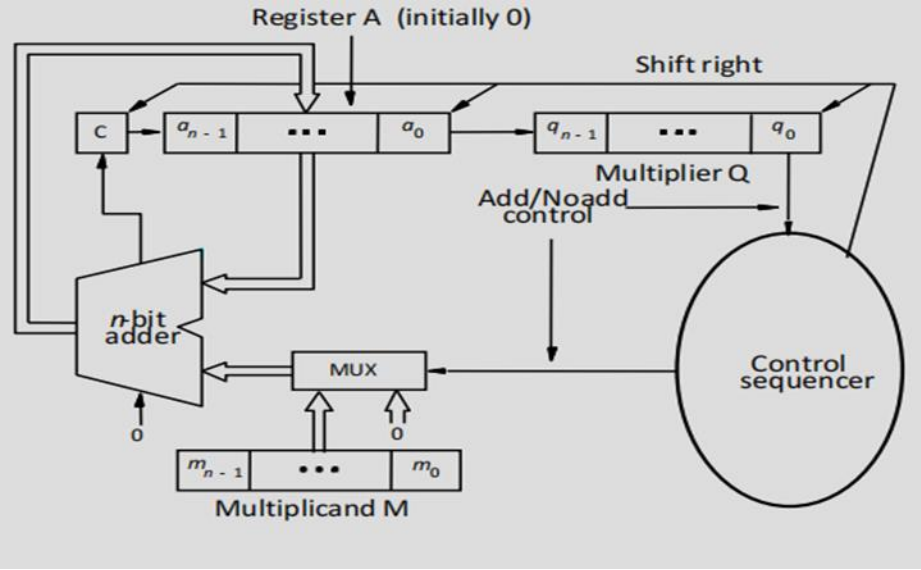
TestBench :

```
process
        begin
                x <= "1111000011110000";
                y <= "1111111100000000";
                cin <= '1' ;
               wait for 10 ns;
               x <= "1010101010101010";
               y <= "0111011101110111";
                cin <= '0' ;
                wait for 10 ns;
                x <= "1000100010001000";
                y <= "1001100110011001";
                cin <= '1' ;
               wait for 10 ns;
                end process ;
end TB_ARCHITECTURE;
```

Simulation :



5- Design a circuit for 8-bit sequential multiplication and prove the correctness of your code.



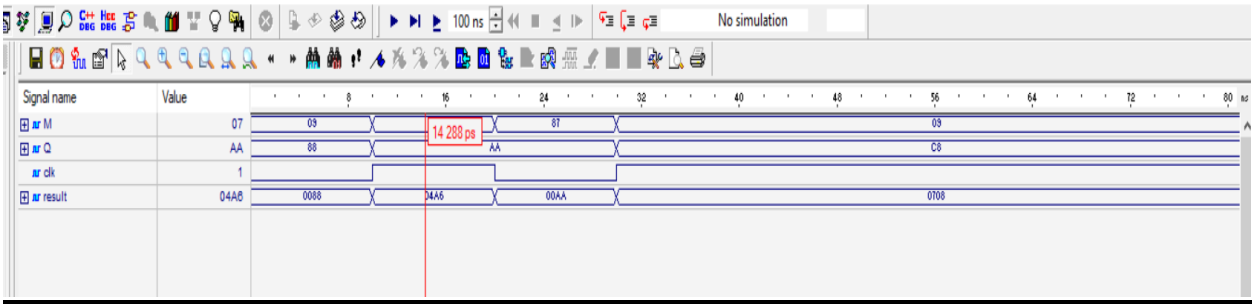

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity ent_mul is
port(
    M : in STD_LOGIC_VECTOR(7 downto 0);        -- Multiplicand
    Q : in STD_LOGIC_VECTOR(7 downto 0);       -- Multiplier
    result : out STD_LOGIC_VECTOR(15 downto 0);
    clk : in std_logic
     );
end ent_mul;
```

```
architecture ent_arch of ent_mul is
begin
   process (clk)
   variable x : std_logic_vector(8 downto 0);          -- 9 bit because of carry
   variable y : std_logic_vector(7 downto 0);
   variable z : std_logic_vector(8 downto 0);          -- 9 bit because of carry

   begin
     x := "000000000";
     y := Q;
     z := '0'& M;
          if clk = '1' and clk'event then
          for i in 0 to 7 loop
             if y(0) = '1' then        -- Q
               x := x + z;             -- Add ( A + M )
             end if;
          y := x(0)&y(7 downto 1);     -- Shift Right
          x := '0'&x(8 downto 1);                  -- Shift Right
          end loop;
          end if;
   result <= x(7 downto 0) & y ;
   end process;
end ent_arch;
```

TestBench :

```
process
        begin
            Q <= "10001000";
             M <= "00001001";
             clk <= '0';
             wait for 10 ns;
             Q <= "10101010";
             M <= "00000111";
             clk <= '1';
             wait for 10 ns;
            Q <= "10101010";
             M <= "10000111";
             clk <= '0';
             wait for 10 ns;
            Q <= "11001000";
             M <= "00001001";
             clk <= '1';
             wait for 10 ns;
             wait;
       end process;
```

Simulation :


100 ns
No simulation
Signal name
Value
8
16
24
32
40
48
56
64
72
80
ar M
07
09
14.288 ps
87
09
ar Q
AA
88
AA
C8
ar clk
1
ar result
04A6
0088
04A6
00AA
0708

7- Design a circuit for non-restoring division circuit and prove the correctness of your code.

Non-restoring algorithm is:
Set A to 0.
Repeat n times:
1- If the sign of A is positive:
Shift A and Q left and subtract M. Set q0 to 1.
Else if the sign of A is negative:
Shift A and Q left and add M. Set q0 to 0.
End of loop
Finally, If the sign of A is 1, add A to M to restore the remainder.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity \ent_non _rest_div\ is
        port(
        Q : in std_logic_vector(7 downto  0);     -- Dividend
        M : in std_logic_vector(7 downto  0);     -- Divisor
        quotient : out std_logic_vector(7 downto  0);
        reminder : out std_logic_vector(7 downto  0);
        clk :std_logic
        );
end \ent_non _rest_div\;
```

```
architecture arch_non_rest_div of \ent_non _rest_div\ is
begin
process(clk)
variable x : std_logic_vector(7 downto  0);
variable y : std_logic_vector(7 downto  0);
variable z : std_logic_vector(7 downto  0);
begin
x := "00000000";          -- empty register(A)
y := Q;              -- divident
z := M;              -- divisor

if clk = '1' and clk'event then
for i in 0 to 7 loop
                  if x(7) = '1' then                          -- A
                        x := x(6 downto 0)&y(7);          -- shift left
                        x := x + z;                                  -- add (M + A )
                 else
                        x := x(6 downto 0)&y(7);          -- shift left
                        x := x - z;                                  -- sub ( M - A )
                 end if;
                if x(7) = '1' then                          -- A
                        y := y(6 downto 0 )&'0';          -- set Q(0) = 0
                else
                        y := y(6 downto 0 )&'1';          -- ste Q(0) = 1
                end if;
```

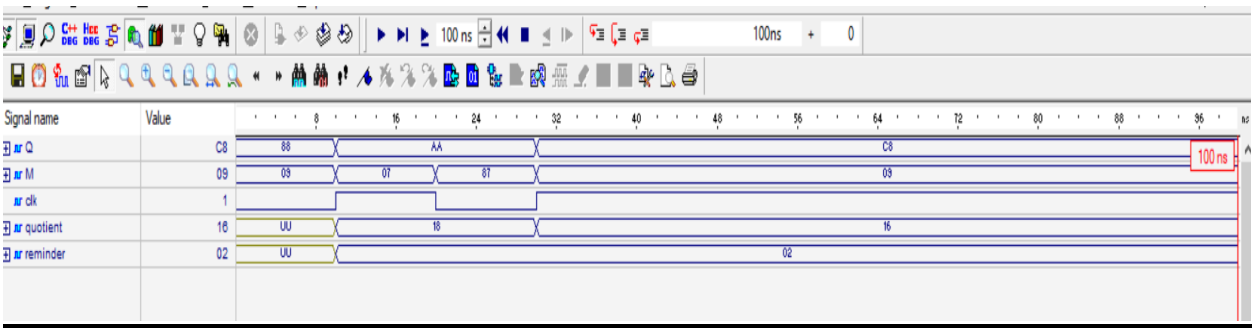
```
end loop;
                if x(7) = '1' then                    -- restoring for reminder
                        x := x + z;
                end if;
quotient <= y;
reminder <= x ;
end if;
end process;
end arch_non_rest_div;
```

TestBench :

```
 process
          begin
        Q <= "10001000";
          M <= "00001001";
          clk <= '0';
          wait for 10 ns;
        Q <= "10101010";
          M <= "00000111";
          clk <= '1';
          wait for 10 ns;
        Q <= "10101010";
          M <= "10000111";
          clk <= '0';
```

```
        wait for 10 ns;
      Q <= "11001000";
        M <= "00001001";
        clk <= '1';
        wait for 10 ns;
        wait;
    end process;
```

Simulation :



8- Design 8-bit ALU with inputs (A , B) and Z as output which is capable of carrying out 16 operation, but for now only implement the following operations:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.NUMERIC_STD.all;
entity ent_alu is
generic( constant N: natural := 1 );
    Port
    (
    x: in  STD_LOGIC_VECTOR(7 downto 0);
    y: in  STD_LOGIC_VECTOR(7 downto 0);
    select_alu : in  STD_LOGIC_VECTOR(3 downto 0);
    out_alu : out  STD_LOGIC_VECTOR(7 downto 0)
    );
end ent_alu;

architecture arch_alu of ent_alu is
signal r: std_logic_vector (7 downto 0);
begin
process(x,y,select_alu)
 begin
  case(select_alu) is
  when "0000" =>              -- no operation
```

Instruction	Function
No operation	Z=A
ADD	Z = A + B
SUB	Z = A – B
Increment	Z = A + 1
Decrement	Z = A – 1
Zero	Z = 0
SHL	Shift Left
SHR	Shift Right
ROL	Rotate Left
ROR	Rotate Right

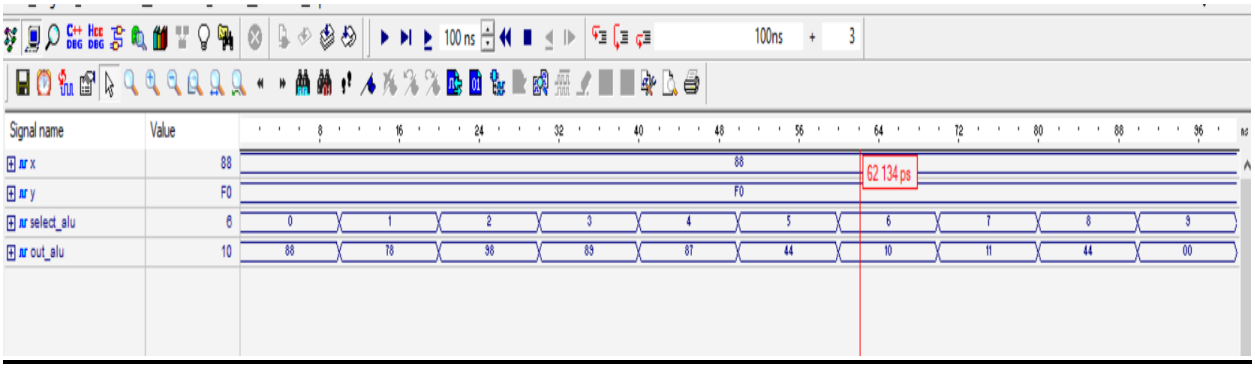
```
    r <= x ;
  when "0001" =>              -- addition
    r <= x + y ;
  when "0010" =>             -- substraction
    r <= x - y ;
  when "0011" =>              -- increment
    r <= x + 1 ;
  when "0100" =>            -- decriment
    r <= x - 1 ;
  when "0101" =>             -- shift right
    r <= std_logic_vector(unsigned(x) srl N);
  when "0110" =>              -- shift left
    r <= std_logic_vector(unsigned(x) sll N);
  when "0111" =>             -- rotate left
    r <= std_logic_vector(unsigned(x) rol N);
  when "1000" =>           -- rotate right
    r <= std_logic_vector(unsigned(x) ror N);
  when "1001" =>          -- zero
    r <= X"00" ;
  when others => r <= X"00" ;
  end case;
end process;
out_alu <= r ;
end arch_alu;
```

TestBench :

```
process
begin
        x <= "10001000" ;
        y <= "11110000" ;
        select_alu <= "0000" ;
        wait for 10 ns ;
        x <= "10001000" ;
        y <= "11110000" ;
        select_alu <= "0001" ;
        wait for 10 ns ;
        x <= "10001000" ;
        y <= "11110000" ;
        select_alu <= "0010" ;
        wait for 10 ns ;
        x <= "10001000" ;
        y <= "11110000" ;
        select_alu <= "0011" ;
        wait for 10 ns ;
        x <= "10001000" ;
        y <= "11110000" ;
        select_alu <= "0100" ;
        wait for 10 ns ;
        x <= "10001000" ;
        y <= "11110000" ;
```

```
select_alu <= "0101" ;
wait for 10 ns ;
x <= "10001000" ;
y <= "11110000" ;
select_alu <= "0110" ;
wait for 10 ns ;
x <= "10001000" ;
y <= "11110000" ;
select_alu <= "0111" ;
wait for 10 ns ;
x <= "10001000" ;
y <= "11110000" ;
select_alu <= "1000" ;
wait for 10 ns ;
x <= "10001000" ;
y <= "11110000" ;
select_alu <= "1001" ;
wait for 10 ns ;
end process ;
```

Simulation :


Signal name
Value
ar x
88
ar y
F0
ar select_alu
6
ar out_alu
10
62.134 ps