Lab 9-10 Nanoprocessor Design Competition

CS1050 Computer Organization and Digital Design

Group number – 14

Members:

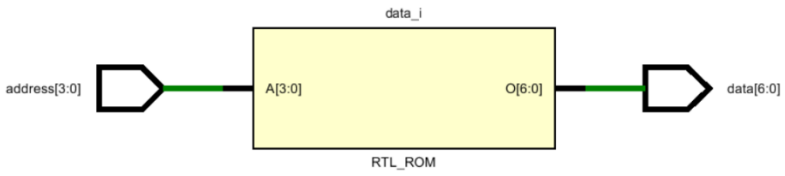
Name	***Index number***
Illangasinghe I.M.D.P	220234R
Dissanayake D.M.S.H	220138C
Balasuriya B.D.S.K	220056X
Fernando K.M.I	220166J

Lab Task

- Designing and developing a 4-bit arithmetic unit capable of completing addition and subtraction on signed integers.
- Designing and developing instruction decoder to decode 12 bit instructions and activate necessary components in the processor accordingly.
- Designing and developing other required components to create the processor.
    - Register Bank
    - Program Rom
    - Program Counter
    - Multiplexers (Using Tri-state Buffers)
    - 3-bit adder
    - Slow clock
- Verifying the processor functionality via Xilinx Vivado simulation.
- Testing the processor in Basys 3 FPGA board.

Design Diagrams and Simulation

- Lookup table for Seven segment display

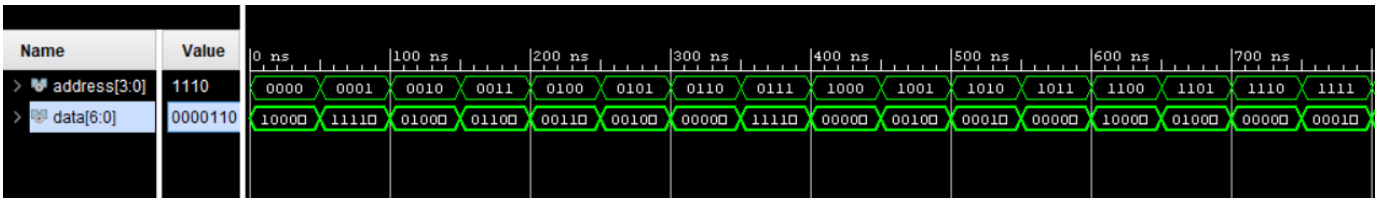



```
-- Module Name: LUT_16_7 - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use ieee.numeric_std.all;

-- seven segment LUT
entity LUT_16_7 is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end LUT_16_7;

architecture Behavioral of LUT_16_7 is

type rom_type is array (0 to 15) of std_logic_vector(6 downto 0);

signal sevenSegment_ROM : rom_type := (
    "1000000", -- 0
    "1111001", -- 1
    "0100100", -- 2
    "0110000", -- 3
    "0011001", -- 4
    "0010010", -- 5
    "0000010", -- 6
    "1111000", -- 7
    "0000000", -- 8
    "0010000", -- 9
    "0001000", -- a
    "0000011", -- b
    "1000110", -- c
    "0100001", -- d
    "0000110", -- e
    "0001110"  -- f
    );

begin
data <= sevenSegment_ROM(to_integer(unsigned(address)));
end Behavioral
```

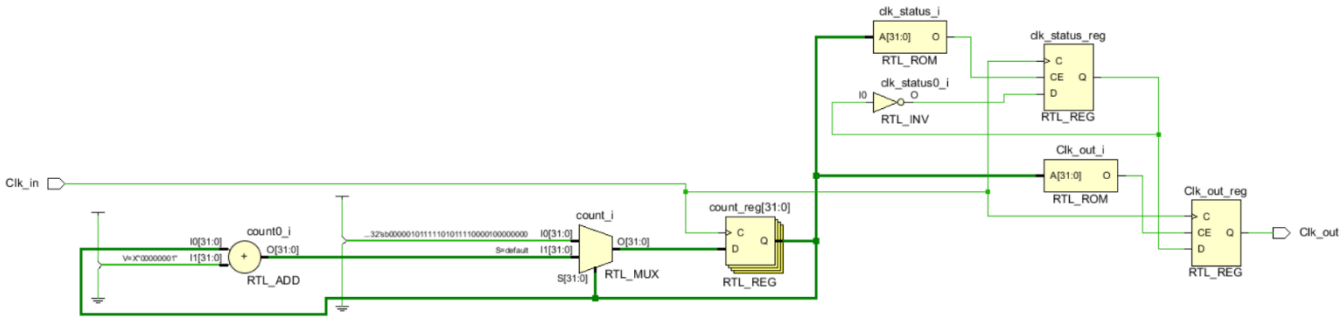
- Unit testing and Simulation

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

entity TB_SevenSeg is
--  Port ( );
end TB_SevenSeg;

architecture Behavioral of TB_SevenSeg is

component LUT_16_7 is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end component;

signal address :  STD_LOGIC_VECTOR (3 downto 0);
signal data    :  STD_LOGIC_VECTOR (6 downto 0);

begin
UUT:LUT_16_7 port map(address,data);

process begin

    address<="0000";
    wait for 50 ns;
    address<="0001";
    wait for 50 ns;
    address<="0010";
    wait for 50 ns;
    address<="0011";
    wait for 50 ns;
    address<="0100";
    wait for 50 ns;
    address<="0101";
    wait for 50 ns;
    address<="0110";
    wait for 50 ns;
    address<="0111";
    wait for 50 ns;
    address<="1000";
    wait for 50 ns;
    address<="1001";
    wait for 50 ns;
    address<="1010";
    wait for 50 ns;
    address<="1011";
    wait for 50 ns;
    address<="1100";
    wait for 50 ns;
    address<="1101";
    wait for 50 ns;
    address<="1110";
    wait for 50 ns;
    address<="1111";
    wait for 50 ns;
    address<="1110";
```

```
    wait;
end process;

end Behavioral;
```



- Slow Clock

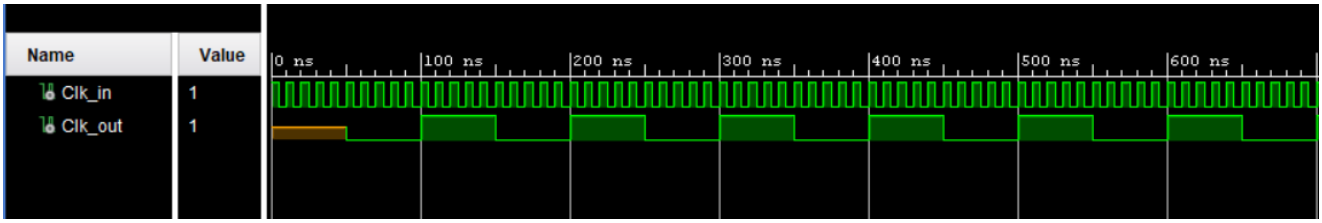



```
-- Module Name: Slow_Clk - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end Slow_Clk;

architecture Behavioral of Slow_Clk is

    signal count:integer := 1;
    signal clk_status : std_logic := '0';

begin
    process (Clk_in)
    begin
        if rising_edge(Clk_in) then
            count <= count + 1;
            if (count = 100000000) then
```

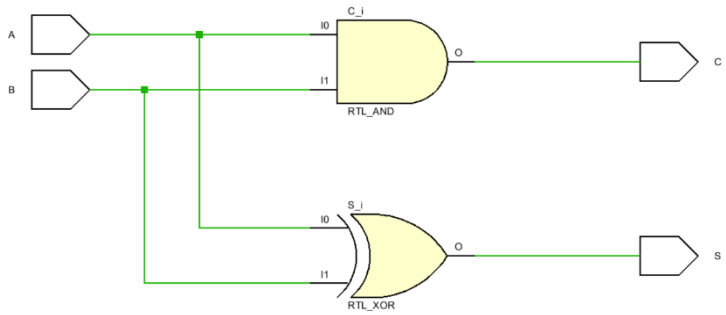
```
                clk_status <= not clk_status;
                Clk_out <= clk_status;
                count <= 1;
            end if;
        end if;
    end process;

end Behavioral;
```

- Unit testing and Simulation

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

entity TB_slowclock is
--  Port ( );
end TB_slowclock;

architecture Behavioral of TB_slowclock is
component Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);

end component;
signal   Clk_in : STD_LOGIC;
signal   Clk_out : STD_LOGIC;

begin

UUT:SLow_Clk port map(Clk_in,Clk_out);

process begin
    Clk_in<='1';
    wait for 5 ns;
    Clk_in<='0';
    wait for 5 ns;
end process;

end Behavioral;
```



- ***Half adder***

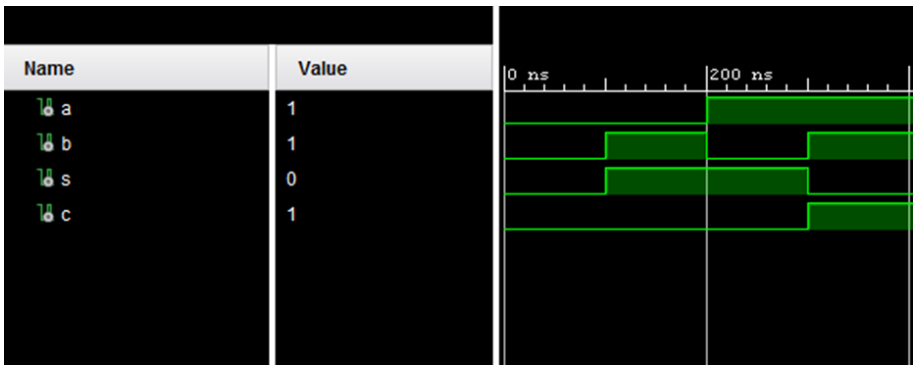



```
-- Module Name: HA - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Half adder;
entity HA is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           S : out STD_LOGIC;
           C : out STD_LOGIC);
end HA;

architecture Behavioral of HA is

begin
S<=A XOR B;
C<=A AND B;

end Behavioral;
```

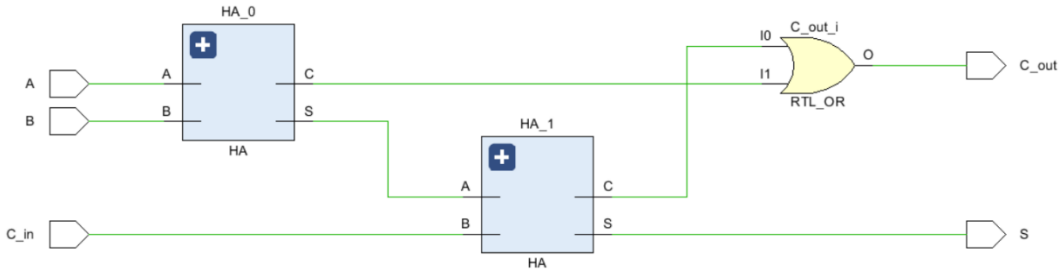
- Unit testing and Simulation

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

 entity TB_HA is
 --  Port ( );
 end TB_HA;

 architecture Behavioral of TB_HA is
 COMPONENT HA
     PORT(A,B :IN STD_LOGIC;
     S,C:OUT STD_LOGIC);
 END COMPONENT;

 SIGNAL a,b: std_logic;
 SIGNAL s,c  : std_logic;
```

```
begin

UUT: HA PORT MAP(
     A=>a,
     B=>b,
     S=>s,
     C=>c
     );

     process
     begin
          a<='0';
          b<='0';
          WAIT FOR 100 ns;
          b<='1';
          WAIT FOR 100 ns;
          a<='1';
          b<='0';
          WAIT FOR 100 ns;
          b<='1';
          WAIT;
     end process;

end Behavioral;
```



- Full adder



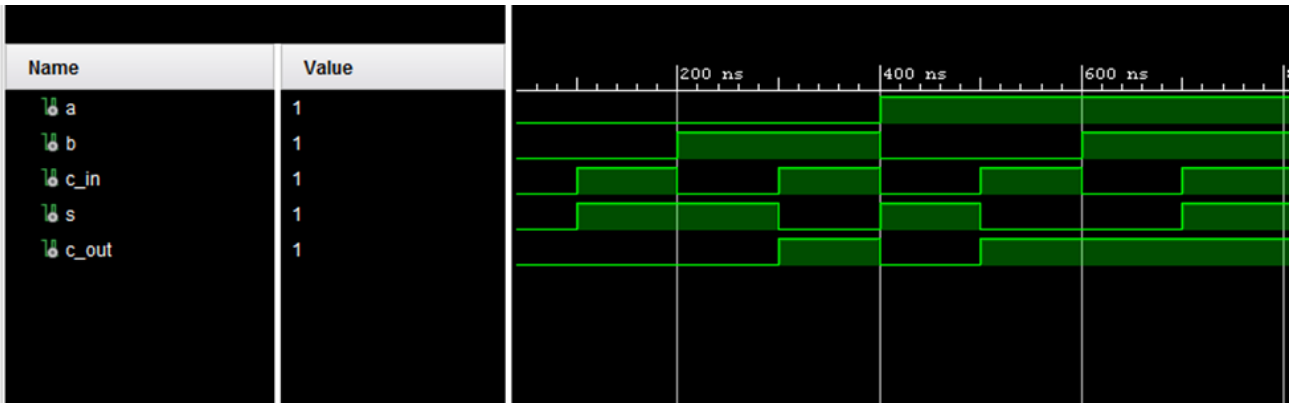

```
-- Module Name: FA - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Full Adder

entity FA is
    Port (
        A : in STD_LOGIC;
        B : in STD_LOGIC;
        C_in : in STD_LOGIC;
        S : out STD_LOGIC;
        C_out : out STD_LOGIC
    );
end FA;

architecture Behavioral of FA is
    COMPONENT HA
        PORT(
            A : IN STD_LOGIC;
            B : IN STD_LOGIC;
            S : OUT STD_LOGIC;
            C : OUT STD_LOGIC
        );
    END COMPONENT;

    SIGNAL HA0_S, HA0_C, HA1_S, HA1_C : STD_LOGIC;

begin
    HA_0 : HA port map (
        A => A,
        B => B,
        S => HA0_S,
        C => HA0_C
    );

    HA_1 : HA port map (
        A => HA0_S,
        B => C_in,
        S => HA1_S,
```

```
        C => HA1_C
    );

    S <= HA1_S;
    C_out <= HA1_C OR HA0_C;
end Behavioral;
```

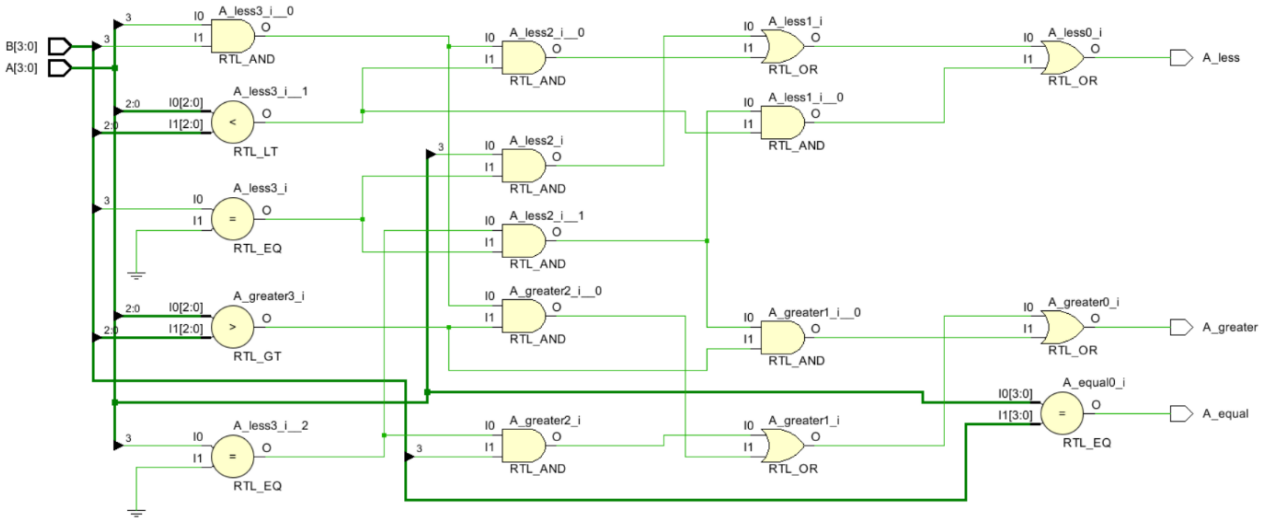
- Unit testing and Simulation

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

entity TB_FA is
--  Port ( );
end TB_FA;

architecture Behavioral of TB_FA is
COMPONENT FA
    PORT(A,B,C_in :IN STD_LOGIC;
    S,C_out:OUT STD_LOGIC);
END COMPONENT;

SIGNAL a,b,c_in: std_logic;
SIGNAL s,c_out  : std_logic;
begin

UUT: FA PORT MAP(
    A=>a,
    B=>b,
    C_in=>c_in,
    S=>s,
    C_out=>c_out
    );

    process
    begin
        a<='0';
        b<='0';
        c_in<='0';
        WAIT FOR 100 ns;
        c_in<='1';
        WAIT FOR 100 ns;
        b<='1';
        c_in<='0';
        WAIT FOR 100 ns;
        c_in<='1';
        WAIT FOR 100 ns;
        a<='1';
        b<='0';
        c_in<='0';
        WAIT FOR 100 ns;
        c_in<='1';
        WAIT FOR 100 ns;
        b<='1';
        c_in<='0';
```

```
        WAIT FOR 100 ns;
        c_in<='1';
          WAIT;
     end process;

end Behavioral;
```



- ***Comparator***



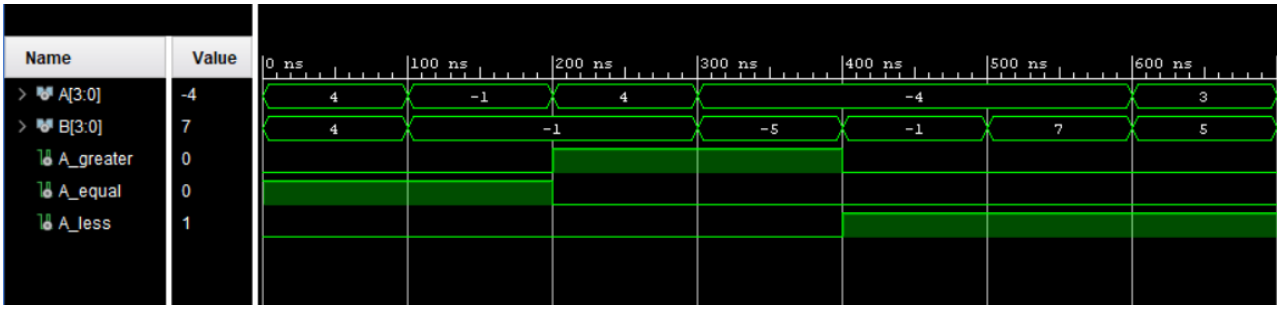

```
-- Module Name: comparator - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- comparator for comparision of two values selected by Data Bus
entity comparator is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
```

```
            A_less : out STD_LOGIC;
            A_equal : out STD_LOGIC;
            A_greater : out STD_LOGIC);
end comparator;

architecture Behavioral of comparator is

signal vA_equal:std_logic_vector (3 downto 0);
signal vA_less: std_logic;

begin

vA_less <= '1' when(A(3)='1' and B(3)='0') else '0';

A_less <='1' when(vA_less='1' or (A(3)='1' and B(3)='1' and    A(2 downto
0)<B(2 downto 0)) or ( A(3)='0' and B(3)='0' and A(2 downto 0)<B(2 downto
0))) else '0';

A_equal <='1' when (A=B) else '0';

A_greater <='1' when((A(3)='0' and B(3)='1') or (A(3)='1' and B(3)='1' and
A(2 downto 0)>B(2 downto 0)) or ( A(3)='0' and B(3)='0' and A(2 downto 0)>B(2
downto 0))) else '0';

end Behavioral;
```

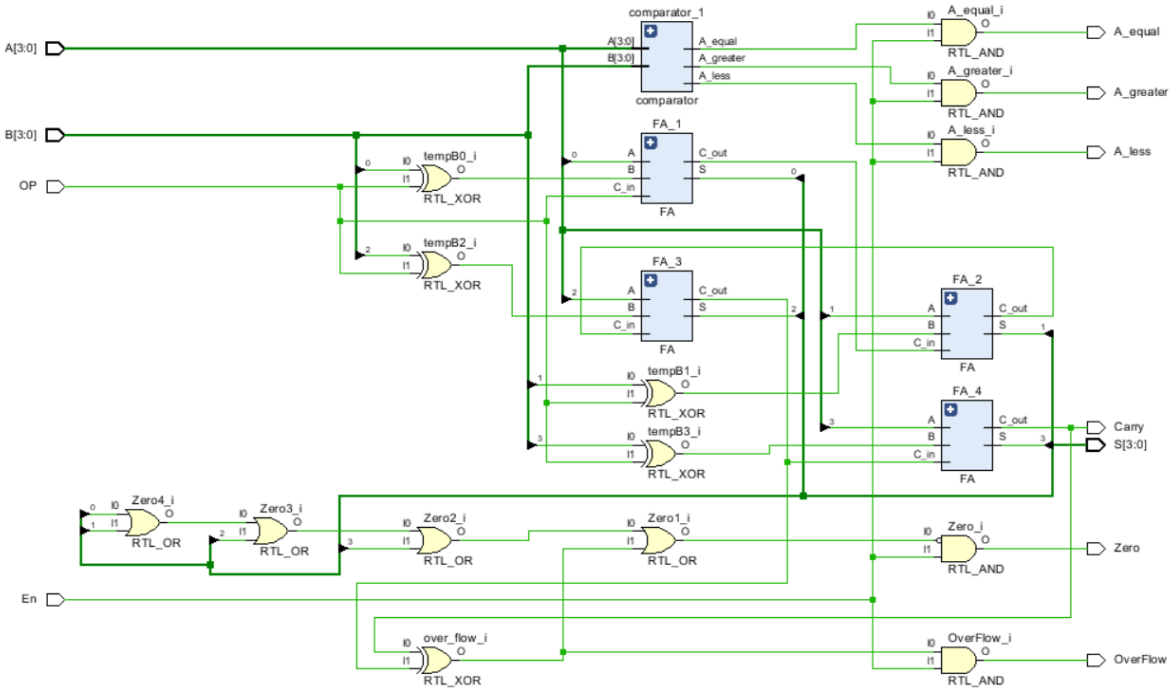
- Unit testing and Simulation

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

entity TB_FA is
--  Port ( );
end TB_FA;

architecture Behavioral of TB_FA is
COMPONENT FA
    PORT(A,B,C_in :IN STD_LOGIC;
    S,C_out:OUT STD_LOGIC);
END COMPONENT;

SIGNAL a,b,c_in: std_logic;
SIGNAL s,c_out  : std_logic;
begin

UUT: FA PORT MAP(
    A=>a,
    B=>b,
    C_in=>c_in,
    S=>s,
    C_out=>c_out
    );

    process
    begin
        a<='0';
```

```
        b<='0';
        c_in<='0';
        WAIT FOR 100 ns;
        c_in<='1';
        WAIT FOR 100 ns;
        b<='1';
        c_in<='0';
        WAIT FOR 100 ns;
        c_in<='1';
        WAIT FOR 100 ns;
        a<='1';
        b<='0';
        c_in<='0';
        WAIT FOR 100 ns;
        c_in<='1';
        WAIT FOR 100 ns;
        b<='1';
        c_in<='0';
        WAIT FOR 100 ns;
        c_in<='1';
          WAIT;
    end process;

end Behavioral;
```



- ***ALU***




```
-- Module Name: RCA_4 - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RCA_4 is
    Port (
             A: in STD_LOGIC_VECTOR (3 downto 0) ;
             B:in STD_LOGIC_VECTOR (3 downto 0);
             OP,En : in STD_LOGIC;
             S : out STD_LOGIC_VECTOR (3 downto 0);
           OverFlow,Zero,Carry,A_less,A_equal,A_greater : out STD_LOGIC);
end RCA_4;

architecture Behavioral of RCA_4 is
component comparator is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           A_less : out STD_LOGIC;
           A_equal : out STD_LOGIC;
           A_greater : out STD_LOGIC);
end component;

component FA
 port (
 A: in std_logic;
 B: in std_logic;
 C_in: in std_logic;
```

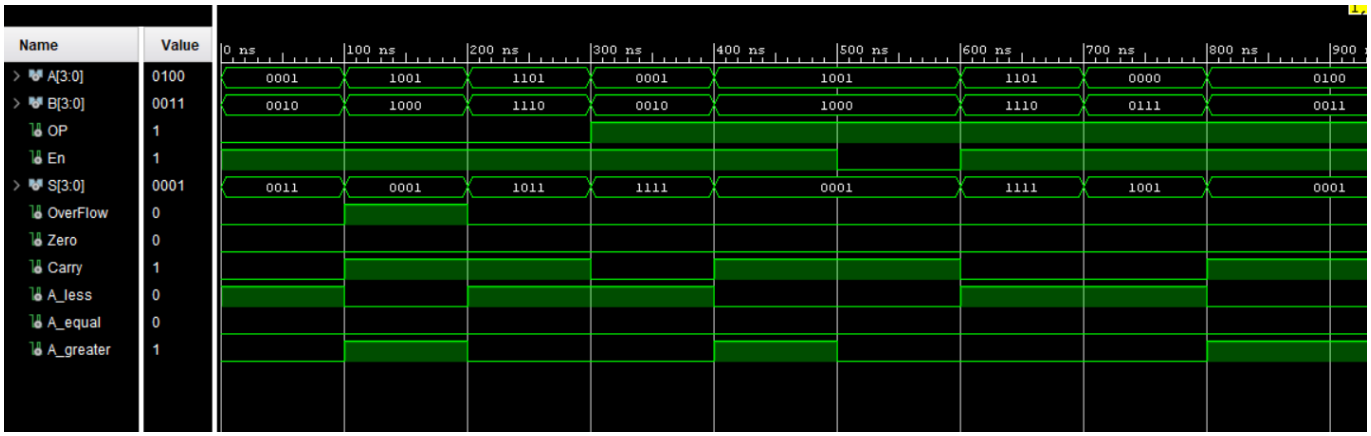
```
 S: out std_logic;
 C_out: out std_logic);
 end component;

 SIGNAL vA_less,vA_equal,vA_greater,over_flow,FA0_S, FA0_C, FA1_S, FA1_C,
FA2_S, FA2_C, FA3_S, FA3_C,tempB0,tempB1,tempB2,tempB3,cout : std_logic;
 signal S_temp : std_logic_vector (3 downto 0);

begin

    comparator_1:comparator port map(A,B,vA_less,vA_equal,vA_greater);

    A_less<=vA_less and En;
    A_equal<=vA_equal and En;
    A_greater<=vA_greater and En;
    tempB0<=B(0) XOR OP;
    tempB1<=B(1) XOR OP;
    tempB2<=B(2) XOR OP;
    tempB3<=B(3) XOR OP;

    FA_1 : FA
    port map (
        A => A(0),
        B =>tempB0,
        C_in => OP, -- Set to ground
        S => S_temp(0),
        C_Out => FA0_C);

    FA_2 : FA
    port map (
        A => A(1),
        B => tempB1,
        C_in => FA0_C,
        S => S_temp(1),
        C_Out => FA1_C);

    FA_3 : FA
    port map (
        A => A(2),
        B =>tempB2,
        C_in => FA1_C,
        S => S_temp(2),
        C_Out => FA2_C);

    FA_4 : FA
    port map (
        A => A(3),
        B => tempB3,
        C_in => FA2_C,
        S => S_temp(3),
        C_Out => cout);

    S<=S_temp;
    over_flow <= cout XOR FA2_c;
    Carry <= cout;
    Zero <= not (S_temp(0) or S_temp(1) or S_temp(2) or S_temp(3) or
over_flow) and En;
```

```
    OverFlow <= over_flow and En;

 end Behavioral;
```

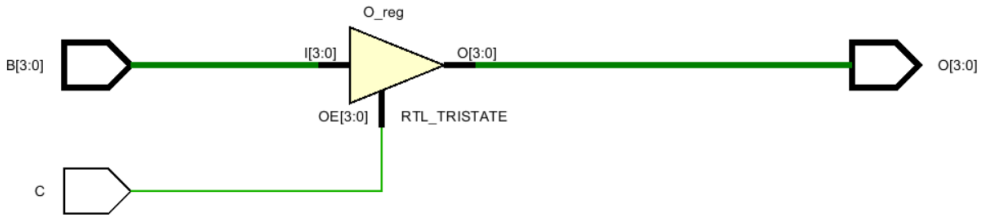
- Unit testing and Simulation

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

entity TB_ALU is
--  Port ( );
end TB_ALU;

architecture Behavioral of TB_ALU is
component RCA_4 is
    Port (
            A: in STD_LOGIC_VECTOR (3 downto 0) ;
            B:in STD_LOGIC_VECTOR (3 downto 0);
            OP,En : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           OverFlow,Zero,Carry,A_less,A_equal,A_greater : out STD_LOGIC);
end component;
signal A  :  STD_LOGIC_VECTOR (3 downto 0) ;
signal B  :  STD_LOGIC_VECTOR (3 downto 0);
signal OP,En :  STD_LOGIC;
signal S  :  STD_LOGIC_VECTOR (3 downto 0);
signal OverFlow,Zero,Carry,A_less,A_equal,A_greater :  STD_LOGIC;

begin

UUT:RCA_4 port map(A,B,OP,En,S,
OverFlow,Zero,Carry,A_less,A_equal,A_greater);
process begin
    En<='1';
    A<="0001";
    B<="0010";
    OP<='0';

    wait for 100 ns;
    A<="1001";
    B<="1000";
    OP<='0';

    wait for 100 ns;
    A<="1101";
    B<="1110";
    OP<='0';

    wait for 100 ns;
    A<="0001";
    B<="0010";
    OP<='1';

    wait for 100 ns;
```

```
        A<="1001";
        B<="1000";
        OP<='1';

        wait for 100 ns;
        En<='0';

        wait for 100 ns;
        En<='1';
        A<="1101";
        B<="1110";
        OP<='1';

        wait for 100 ns;
        A<="0000";
        B<="0111";

        wait for 100 ns;
        A<="0100";
        B<="0011";
        wait;

        wait;
    end process;

    end Behavioral;
```



- ***Tri State Buffer – 4 bit***

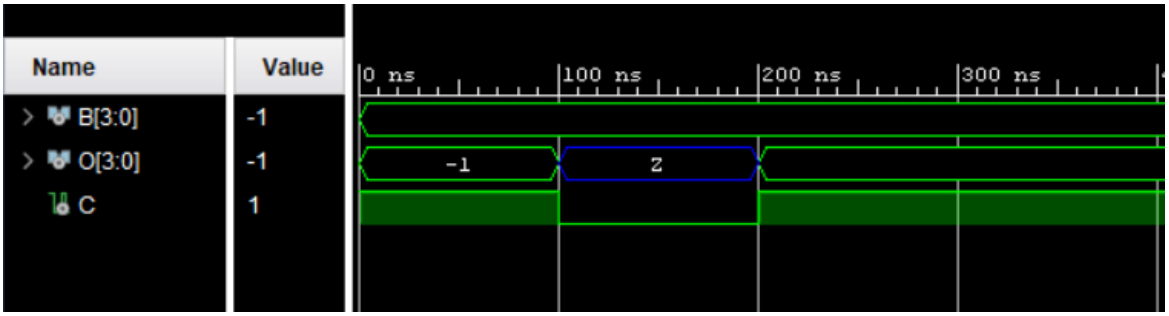



```
-- Module Name: TBuffer_2_1_4Bit - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- 4 bit normal singal controlled tri state buffer
entity TBuffer_2_1_4Bit is
    Port ( B : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (3 downto 0));
end TBuffer_2_1_4Bit;

architecture Behavioral of TBuffer_2_1_4Bit is

begin

O <= B when C='1' else "ZZZZ";

end Behavioral;
```

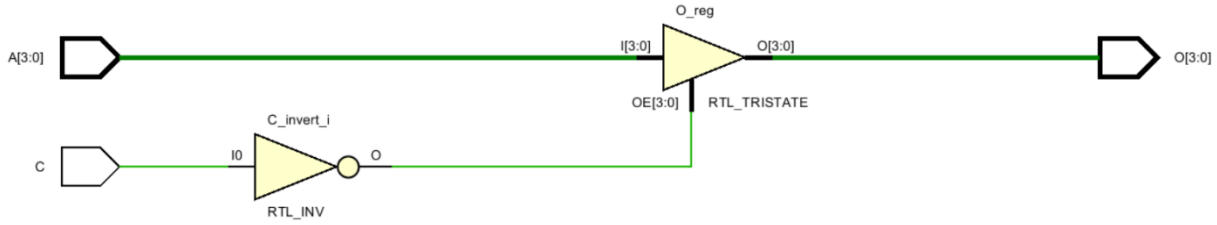
- Unit testing and Simulation

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

entity TBuff_4bit_n is
--  Port ( );
end TBuff_4bit_n;

architecture Behavioral of TBuff_4bit_n is
component TBuffer_2_1_4Bit is
    Port ( B : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal B,O : std_logic_vector(3 downto 0);
signal C:std_logic;
begin
UUT:TBuffer_2_1_4Bit port map(B,C,O);
process begin
B<="1111";
C<='1';
```

```
wait for 100 ns;
C<='0';
wait for 100 ns;
C<='1';
wait;

end process;
end Behavioral;
```



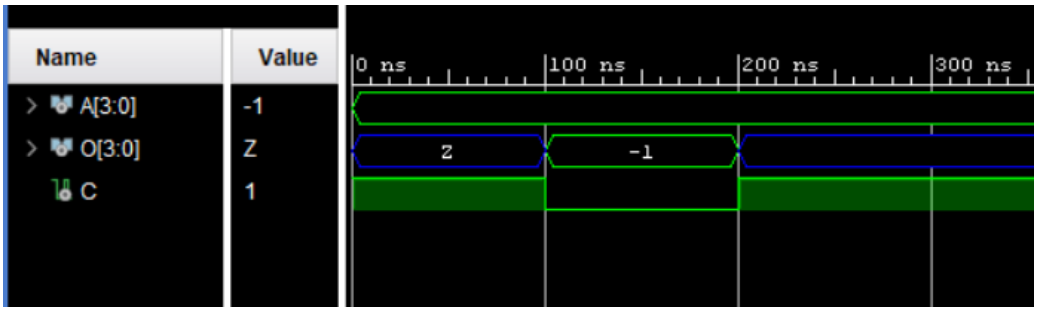
- Tri State Buffer (Inverted) – 4 bit



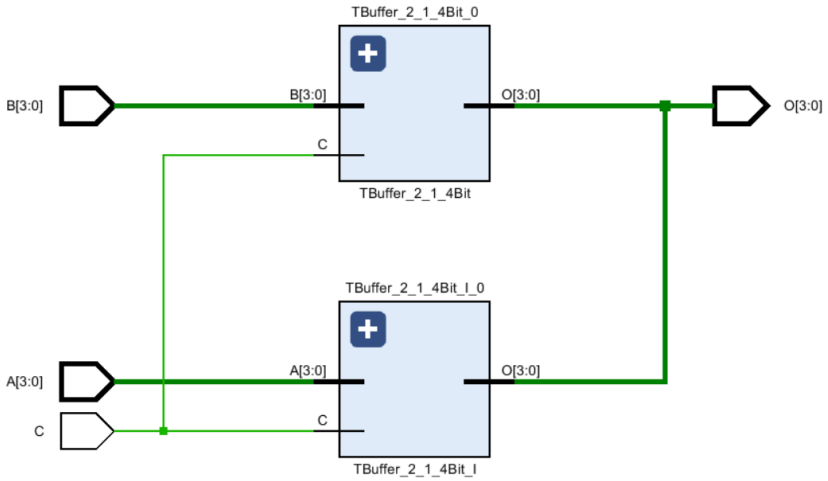

```
-- Module Name: TBuffer_2_1_4Bit_I - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- 4 bit inverted tri state buffer
entity TBuffer_2_1_4Bit_I is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (3 downto 0));
end TBuffer_2_1_4Bit_I;

architecture Behavioral of TBuffer_2_1_4Bit_I is
signal C_invert:std_logic;

begin
C_invert <= not C;
O <=A when C_invert='1' else "ZZZZ";

end Behavioral;
```

- Unit testing and Simulation

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

entity Tbuff_4bit_invert is
--  Port ( );
end Tbuff_4bit_invert;
architecture behavioral of Tbuff_4bit_invert is
component TBuffer_2_1_4Bit_I is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal A,O : std_logic_vector(3 downto 0);
signal C:std_logic;
begin
UUT:TBuffer_2_1_4Bit_I port map(A,C,O);
process begin
A<="1111";
C<='1';
wait for 100 ns;
C<='0';
wait for 100 ns;
C<='1';
wait;

end process;
end Behavioral;
```



- 2×1 Multiplexer

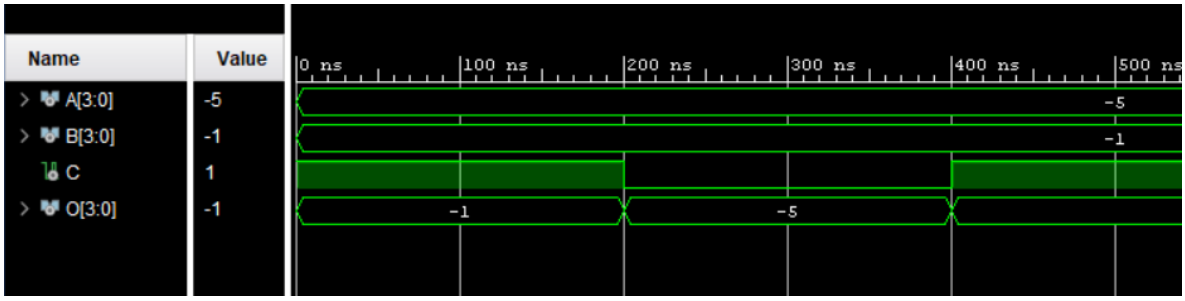



```
-- Module Name: Mux_2_1 - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- 2 way mux using two inverted and normal signal controlled tri state
buffers
entity Mux_2_1 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (3 downto 0));
end Mux_2_1;

architecture Behavioral of Mux_2_1 is

component TBuffer_2_1_4Bit_I is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component TBuffer_2_1_4Bit is
    Port ( B : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal A_out,B_out:STD_LOGIC_VECTOR (3 downto 0);

begin

TBuffer_2_1_4Bit_I_0: TBuffer_2_1_4Bit_I port map(A,C,A_out);
TBuffer_2_1_4Bit_0: TBuffer_2_1_4Bit PORT MAP(B,C,B_out);
O <= A_out;
```

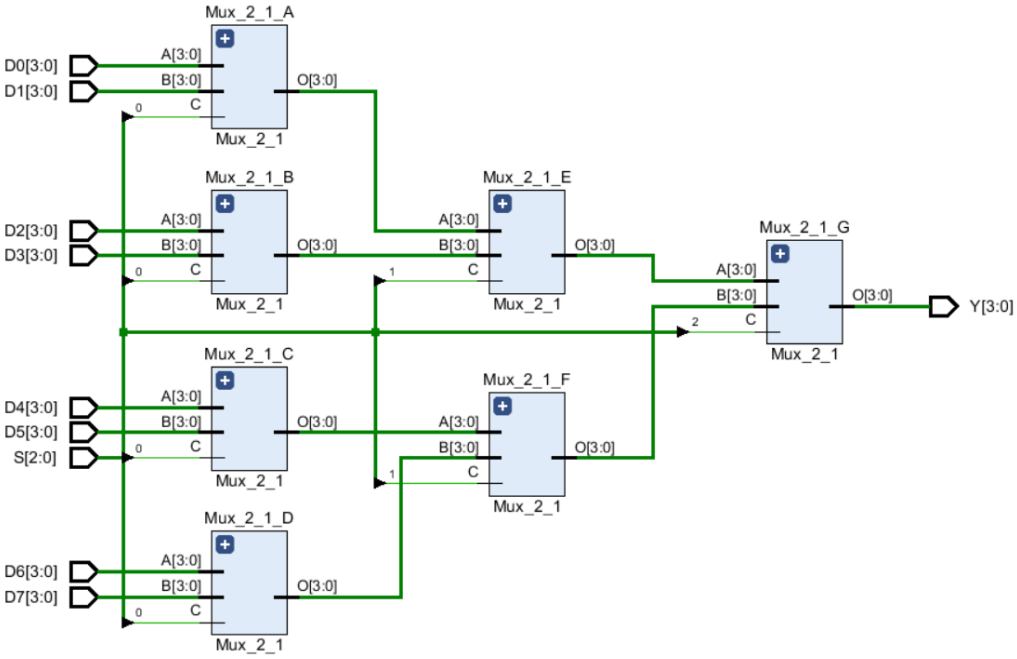
```
O <= B_out;

end Behavioral;
```

- Unit testing and Simulation

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

entity TB_Tbuffer_MUX_2_1 is
--  Port ( );
end TB_Tbuffer_MUX_2_1;

architecture Behavioral of TB_Tbuffer_MUX_2_1 is
component Mux_2_1 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal A :  STD_LOGIC_VECTOR (3 downto 0);
signal B :  STD_LOGIC_VECTOR (3 downto 0);
signal C :  STD_LOGIC;
signal O :  STD_LOGIC_VECTOR (3 downto 0);
begin
UUT:Mux_2_1 port map(A,B,C,O);
process begin
A<="1011";
B<="1111";
C<='1';
wait for 200 ns;
C<='0';
wait for 200 ns;
C<='1';
wait;
end process;
end Behavioral;
```



- 8×1 Multiplexer



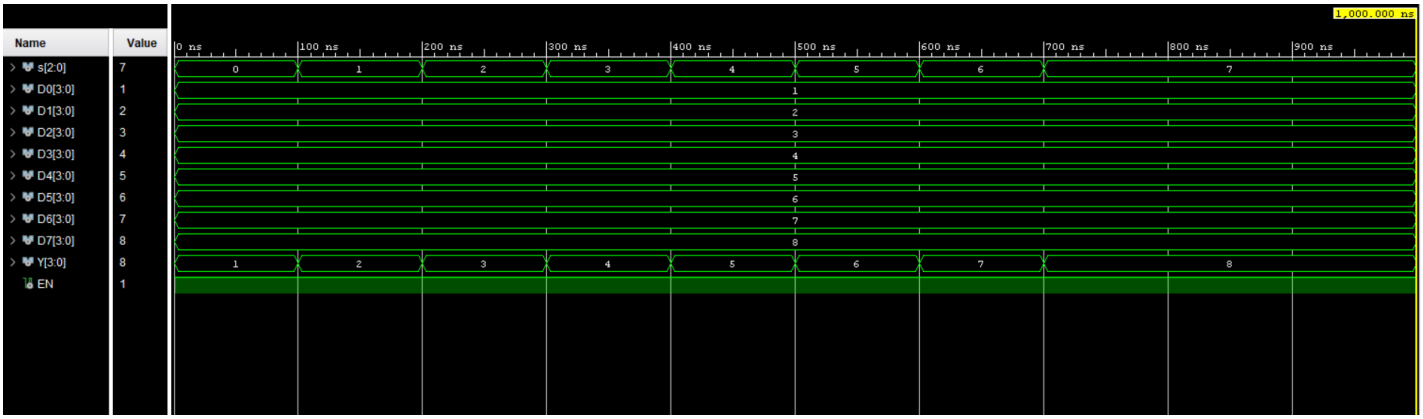

```
-- Module Name: Mux_8_to_1 - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
-- 8 way mux using 2 way bux all in 4 bits
entity Mux_8_to_1 is
    Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
           D0,D1,D2,D3,D4,D5,D6,D7 : in std_logic_vector (3 downto 0);
           Y : out STD_LOGIC_vector(3 downto 0));
end Mux_8_to_1;

architecture Behavioral of Mux_8_to_1 is


component Mux_2_1 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (3 downto 0));
end component;

SIGNAL d12, d23, d45, d67, p, q:STD_LOGIC_VECTOR(3 DOWNTO 0);

begin
Mux_2_1_A:Mux_2_1
```

```
    port map(D0,D1,S(0),d12);

Mux_2_1_B:Mux_2_1
    port map(D2,D3,S(0),d23);

Mux_2_1_C:Mux_2_1
        port map(D4,D5,S(0),d45);

Mux_2_1_D:Mux_2_1
        port map(D6,D7,S(0),d67);

Mux_2_1_E:Mux_2_1
        port map(d12,d23,S(1),p);

Mux_2_1_F:Mux_2_1
        port map(d45,d67,S(1),q);

Mux_2_1_G:Mux_2_1
        port map(p,q,S(2),Y);

end Behavioral;
```

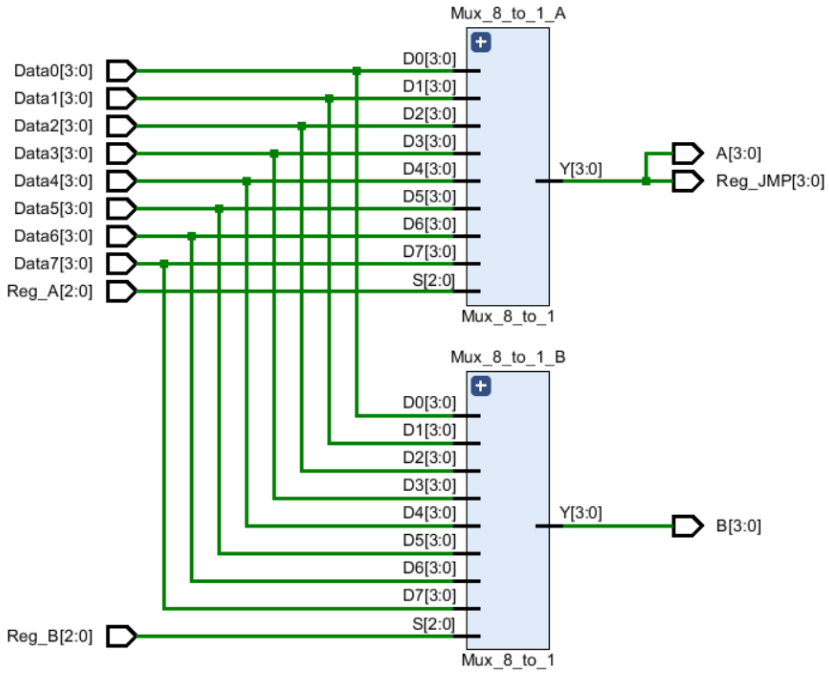
- Unit testing and Simulation

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

entity TB_MUX is
--  Port ( );
end TB_MUX;

architecture Behavioral of TB_MUX is

component Mux_8_to_1 is
    Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
           D0,D1,D2,D3,D4,D5,D6,D7 : in std_logic_vector (3 downto 0);

           EN : in STD_LOGIC;
           Y : out STD_LOGIC_vector(3 downto 0));
end component;

signal s:STD_LOGIC_VECTOR (2 downto 0);
signal D0,D1,D2,D3,D4,D5,D6,D7,Y:STD_LOGIC_VECTOR (3 downto 0);
signal EN:STD_LOGIC;

begin
UUT:Mux_8_to_1 port map(s,D0,D1,D2,D3,D4,D5,D6,D7,EN,Y);
process begin
    EN<='1';
    s<="000";
    D0<="0001";
    D1<="0010";
    D2<="0011";
    D3<="0100";
    D4<="0101";
```

```
        D5<="0110";
        D6<="0111";
        D7<="1000";

        wait for 100 ns;
        s<="001";
        wait for 100 ns;
        s<="010";
        wait for 100 ns;
        s<="011";
        wait for 100 ns;
        s<="100";
        wait for 100 ns;
        s<="101";
        wait for 100 ns;
        s<="110";
        wait for 100 ns;
        s<="111";
    wait;

    end process;

    end Behavioral;
```



- ***Mux Selector***



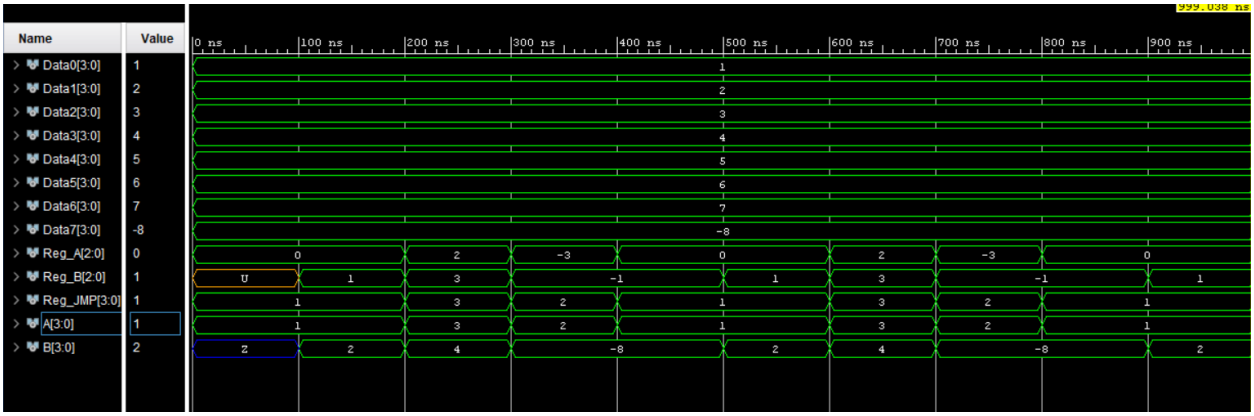

```
-- Module Name: Mux_Selector - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Use Regester select signal for A and B Register and get value
entity Data_Bus is
    Port ( Data0 : in STD_LOGIC_VECTOR (3 downto 0);
           Data1 : in STD_LOGIC_VECTOR (3 downto 0);
           Data2 : in STD_LOGIC_VECTOR (3 downto 0);
           Data3 : in STD_LOGIC_VECTOR (3 downto 0);
           Data4 : in STD_LOGIC_VECTOR (3 downto 0);
           Data5 : in STD_LOGIC_VECTOR (3 downto 0);
           Data6 : in STD_LOGIC_VECTOR (3 downto 0);
           Data7 : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_A : in STD_LOGIC_VECTOR (2 downto 0);
           Reg_B : in STD_LOGIC_VECTOR (2 downto 0);
           Reg_JMP : out STD_LOGIC_vector (3 downto 0);
           A : out STD_LOGIC_VECTOR (3 downto 0);
           B : out STD_LOGIC_VECTOR (3 downto 0));
end Data_Bus;

architecture Behavioral of Data_Bus is

component Mux_8_to_1 is
    Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
           D0,D1,D2,D3,D4,D5,D6,D7 : in std_logic_vector (3 downto 0);
           Y : out STD_LOGIC_vector(3 downto 0));
end component;

signal register_A:std_logic_vector (3 downto 0);

begin
```

```
    Mux_8_to_1_A:Mux_8_to_1 port
map(Reg_A,Data0,Data1,Data2,Data3,Data4,Data5,Data6,Data7,register_A);
    Mux_8_to_1_B:Mux_8_to_1 port
map(Reg_B,Data0,Data1,Data2,Data3,Data4,Data5,Data6,Data7,B);
    A <= register_A;
    Reg_JMP <= register_A;

end Behavioral;
```

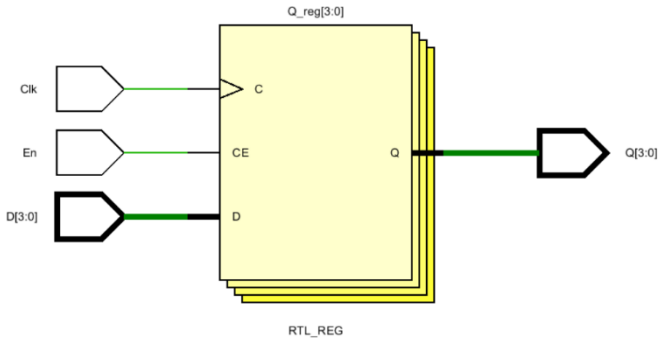
- Unit testing and Simulation

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

entity TB_MUXselector is
--  Port ( );
end TB_MUXselector;

architecture Behavioral of TB_MUXselector is

component Data_Bus is
    Port ( Data0 : in STD_LOGIC_VECTOR (3 downto 0);
           Data1 : in STD_LOGIC_VECTOR (3 downto 0);
           Data2 : in STD_LOGIC_VECTOR (3 downto 0);
           Data3 : in STD_LOGIC_VECTOR (3 downto 0);
           Data4 : in STD_LOGIC_VECTOR (3 downto 0);
           Data5 : in STD_LOGIC_VECTOR (3 downto 0);
           Data6 : in STD_LOGIC_VECTOR (3 downto 0);
           Data7 : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_A : in STD_LOGIC_VECTOR (2 downto 0);
           Reg_B : in STD_LOGIC_VECTOR (2 downto 0);
           Reg_JMP : out STD_LOGIC_vector (3 downto 0);
           A : out STD_LOGIC_VECTOR (3 downto 0);
           B : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal Data0 :  STD_LOGIC_VECTOR (3 downto 0);
signal Data1 :  STD_LOGIC_VECTOR (3 downto 0);
signal Data2 :  STD_LOGIC_VECTOR (3 downto 0);
signal Data3 :  STD_LOGIC_VECTOR (3 downto 0);
signal Data4 :  STD_LOGIC_VECTOR (3 downto 0);
signal Data5 :  STD_LOGIC_VECTOR (3 downto 0);
signal Data6 :  STD_LOGIC_VECTOR (3 downto 0);
signal Data7 :  STD_LOGIC_VECTOR (3 downto 0);
signal Reg_A :  STD_LOGIC_VECTOR (2 downto 0);
signal Reg_B :  STD_LOGIC_VECTOR (2 downto 0);
signal Reg_JMP : STD_LOGIC_vector (3 downto 0);
signal A :  STD_LOGIC_VECTOR (3 downto 0);
signal B :  STD_LOGIC_VECTOR (3 downto 0);

begin
UUT:Data_Bus port
map(Data0,Data1,Data2,Data3,Data4,Data5,Data6,Data7,Reg_A,Reg_B,Reg_JMP,A,B);

process begin
```

```
Data0<="0001";
Data1<="0010";
Data2<="0011";
Data3<="0100";
Data4<="0101";
Data5<="0110";
Data6<="0111";
Data7<="1000";
Reg_A<="000";
wait for 100 ns;
Reg_B<="001";
wait for 100 ns;
Reg_A<="010";
Reg_B<="011";
wait for 100 ns;
Reg_A<="101";
Reg_B<="111";
wait for 100 ns;
Reg_A<="101";
Reg_B<="111";
end process;

end Behavioral;
```



- ***Register***

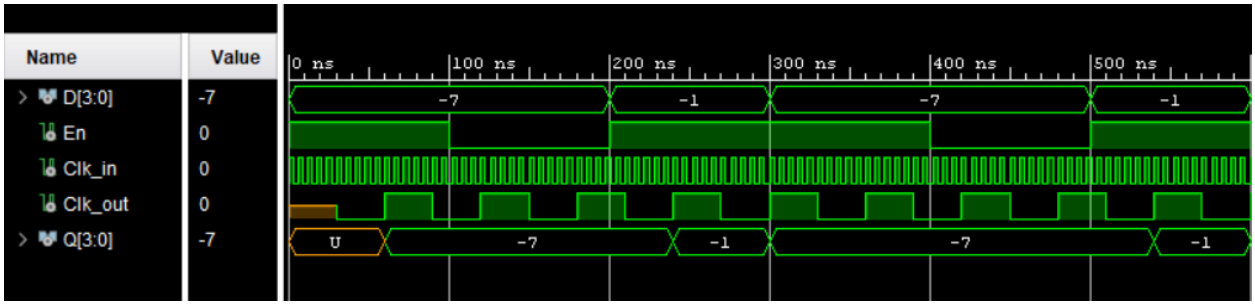



```
-- Module Name: Reg - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Reg is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           En : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end Reg;

architecture Behavioral of Reg is

begin
    -- Process triggered by the clock signal
    process (Clk)
    begin
        if (rising_edge(Clk)) then  -- Check for rising edge of the clock
            if En = '1' then        -- Check if enable signal is active
                Q <= D;             -- Transfer input D to output Q
            end if;
        end if;
    end process;

end Behavioral;
```

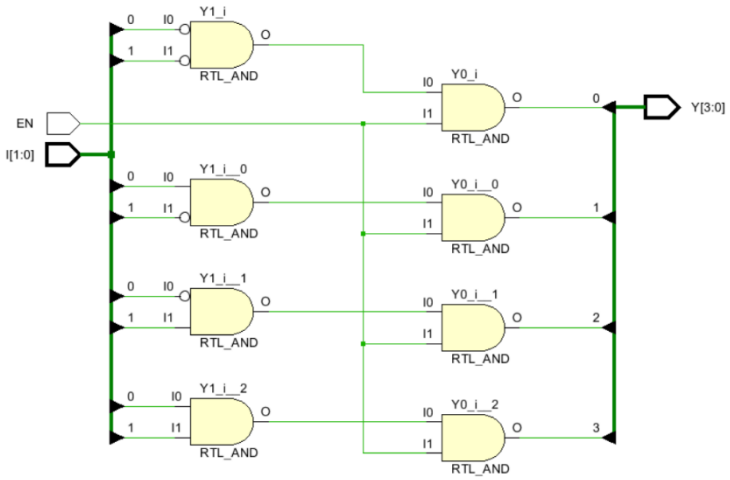
- Unit testing and Simulation

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

entity TB_REG is
--  Port ( );
end TB_REG;

architecture Behavioral of TB_REG is
COMPONENT Reg is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           En : in STD_LOGIC;
           Clk : in STD_LOGIC;
```

```
            Q : out STD_LOGIC_VECTOR (3 downto 0));
end COMPONENT;
COMPONENT Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end COMPONENT;

SIGNAL D :  STD_LOGIC_VECTOR (3 downto 0);
SIGNAL En :  STD_LOGIC;
SIGNAL  Clk_in,Clk_out: STD_LOGIC;
SIGNAL Q :  STD_LOGIC_VECTOR (3 downto 0);
begin
UUT:Reg PORT MAP(D,En,Clk_out,Q);
UUT1:Slow_Clk PORT MAP(Clk_in,Clk_out);
PROCESS BEGIN
Clk_in<='1';
wait for 3 ns;
Clk_in<='0';
wait for 3 ns;
end process;
process begin
D<="1001";
En<='1';
wait for 100 ns;
En<='0';
wait for 100 ns;
D<="1111";
En<='1';
wait for 100 ns;
En<='0';
END PROCESS;

end Behavioral;
```



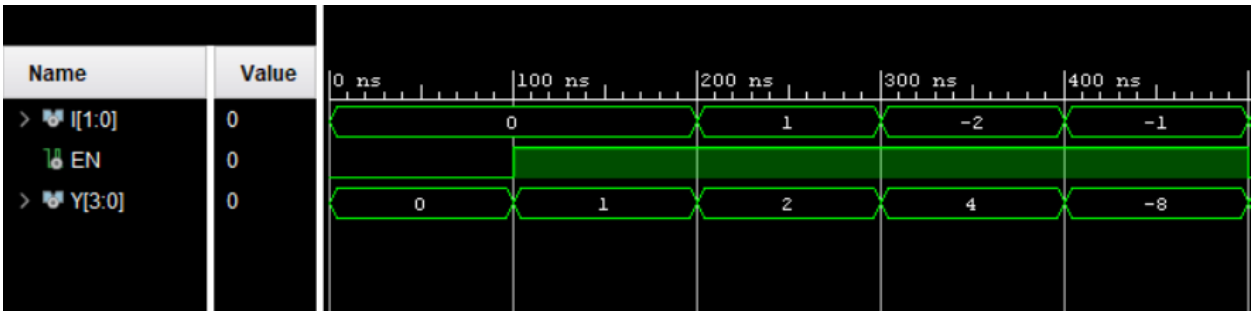
- 2 to 4 Decoder



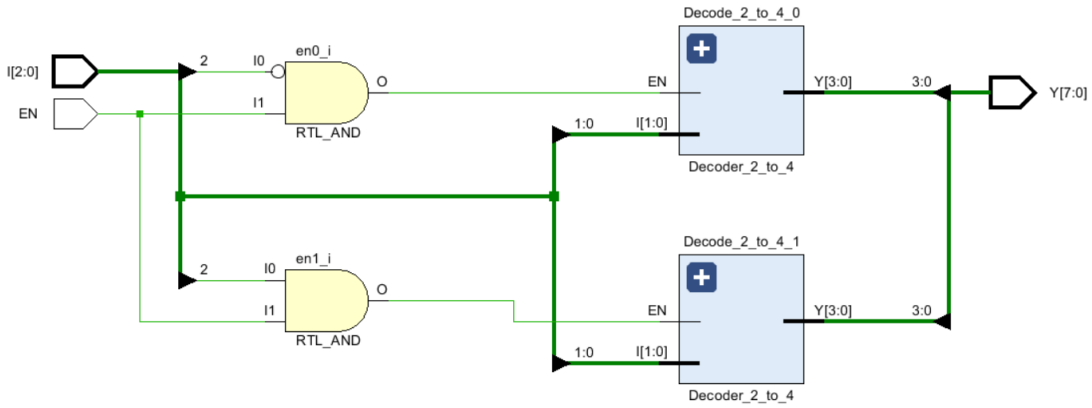

```
-- Module Name: Decoder_2_to_4 - Behavioral


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Decoder_2_to_4 is
    Port (
        I : in STD_LOGIC_VECTOR(1 downto 0);  -- Input signal
        EN : in STD_LOGIC;                    -- Enable signal
        Y : out STD_LOGIC_VECTOR(3 downto 0)  -- Output signals
    );
end Decoder_2_to_4;

architecture Behavioral of Decoder_2_to_4 is
begin
    -- Decode input I based on EN signal to produce a 4-bit output
    Y(0) <= (NOT I(0)) AND (NOT I(1)) AND EN; -- Active when I = "00" and EN
is high
    Y(1) <= I(0) AND (NOT I(1)) AND EN;       -- Active when I = "01" and EN
is high
    Y(2) <= (NOT I(0)) AND I(1) AND EN;       -- Active when I = "10" and EN
is high
    Y(3) <= I(0) AND I(1) AND EN;             -- Active when I = "11" and EN
is high
end Behavioral;
```

- Unit testing and Simulation

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

entity TB_Dec2to4 is
--  Port ( );
end TB_Dec2to4;

architecture Behavioral of TB_Dec2to4 is
component Decoder_2_to_4 is
    Port (
        I : in STD_LOGIC_VECTOR(1 downto 0);  -- Input signal
        EN : in STD_LOGIC;                    -- Enable signal
        Y : out STD_LOGIC_VECTOR(3 downto 0)  -- Output signals
    );
end component;
signal  I :  STD_LOGIC_VECTOR(1 downto 0);
signal EN :  STD_LOGIC;
signal Y :   STD_LOGIC_VECTOR(3 downto 0);
begin
UUT:Decoder_2_to_4 port map(I,EN,Y);
process begin
I<="00";
EN<='0';
WAIT FOR 100 NS;
EN<='1';
WAIT FOR 100 NS;
I<="01";
WAIT FOR 100 NS;
I<="10";
WAIT FOR 100 NS;
I<="11";
WAIT FOR 100 NS;
I<="00";
EN<='0';
wait;
end process;

end Behavioral;
```



- 3 to 8 Decoder



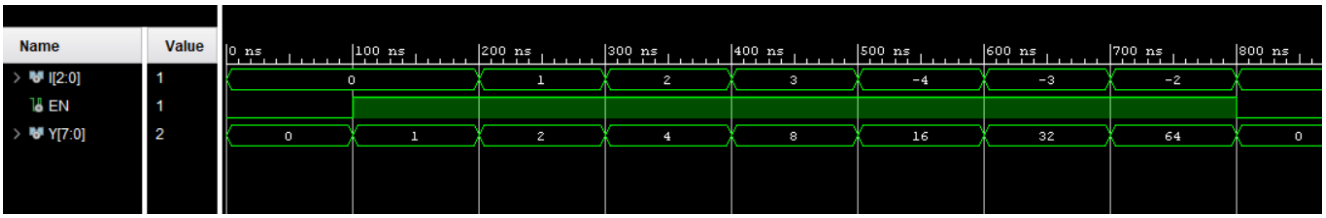

```
-- Module Name: Decoder_3_to_8 - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Decoder_3_to_8 is
    Port (
        I : in STD_LOGIC_VECTOR (2 downto 0);   -- 3-bit input
        EN : in STD_LOGIC;                      -- Enable input
        Y : out STD_LOGIC_VECTOR (7 downto 0)   -- 8-bit output
    );
end Decoder_3_to_8;

architecture Behavioral of Decoder_3_to_8 is

COMPONENT Decoder_2_to_4
    PORT(I:IN STD_LOGIC_VECTOR;
    EN:IN STD_LOGIC;
    Y:OUT STD_LOGIC_VECTOR);
END COMPONENT;

signal I0, I1 : STD_LOGIC_VECTOR(1 downto 0);  -- Signals for the inputs of
the nested decoders
signal Y0, Y1 : STD_LOGIC_VECTOR(3 downto 0);  -- Signals for the outputs of
the nested decoders
signal en0, en1, I2 : STD_LOGIC;               -- Enable signals for the
nested decoders and intermediate bit signal

begin
    Decode_2_to_4_0 :Decoder_2_to_4
        port map(
            I => I0,
            EN => en0,
            Y => Y0
        );
    Decode_2_to_4_1 : Decoder_2_to_4
        port map(
```

```
            I => I1,
            EN => en1,
            Y => Y1
        );

    -- Assignments for enable signals and connecting inputs to outputs
    en0 <= NOT(I(2)) AND EN;
    en1 <= I(2) AND EN;
    I0 <= I(1 downto 0);
    I1 <= I(1 downto 0);
    I2 <= I(2);
    Y(3 downto 0) <= Y0;
    Y(7 downto 4) <= Y1;

end Behavioral;
```

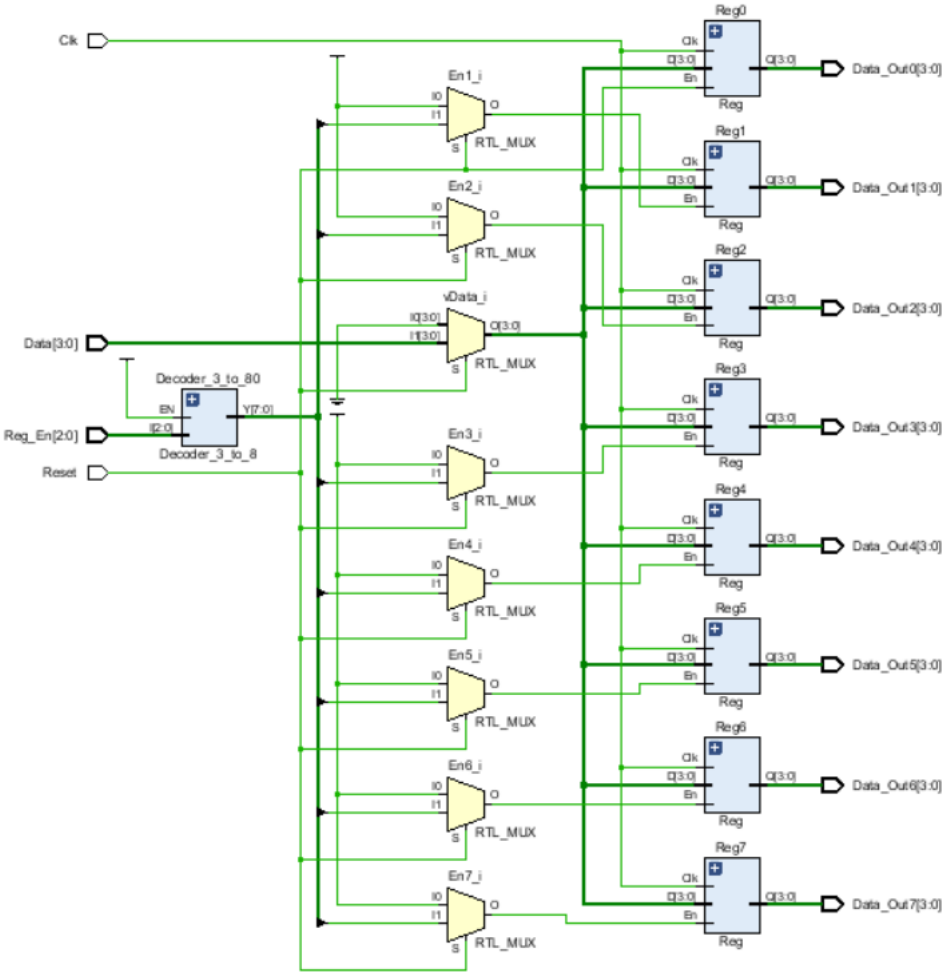
- Unit testing and Simulation

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

entity TB_DEC3TO8 is
--  Port ( );
end TB_DEC3TO8;

architecture Behavioral of TB_DEC3TO8 is
COMPONENT Decoder_3_to_8 is
    Port (
        I : in STD_LOGIC_VECTOR (2 downto 0);  -- 3-bit input
        EN : in STD_LOGIC;                     -- Enable input
        Y : out STD_LOGIC_VECTOR (7 downto 0)  -- 8-bit output
    );
end COMPONENT;
SIGNAL I  : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL EN :  STD_LOGIC;
SIGNAL Y  :  STD_LOGIC_VECTOR (7 downto 0);
begin
UUT:Decoder_3_to_8 PORT MAP(I,EN,Y);
PROCESS BEGIN
I<="000";
EN<='0';
WAIT FOR 100 NS;
EN<='1';
WAIT FOR 100 NS;
I<="001";
WAIT FOR 100 NS;
I<="010";
WAIT FOR 100 NS;
I<="011";
WAIT FOR 100 NS;
I<="100";
WAIT FOR 100 NS;
I<="101";
WAIT FOR 100 NS;
I<="110";
```

```
WAIT FOR 100 NS;
I<="111";
END PROCESS;

end Behavioral;
```



- ***Register Bank***



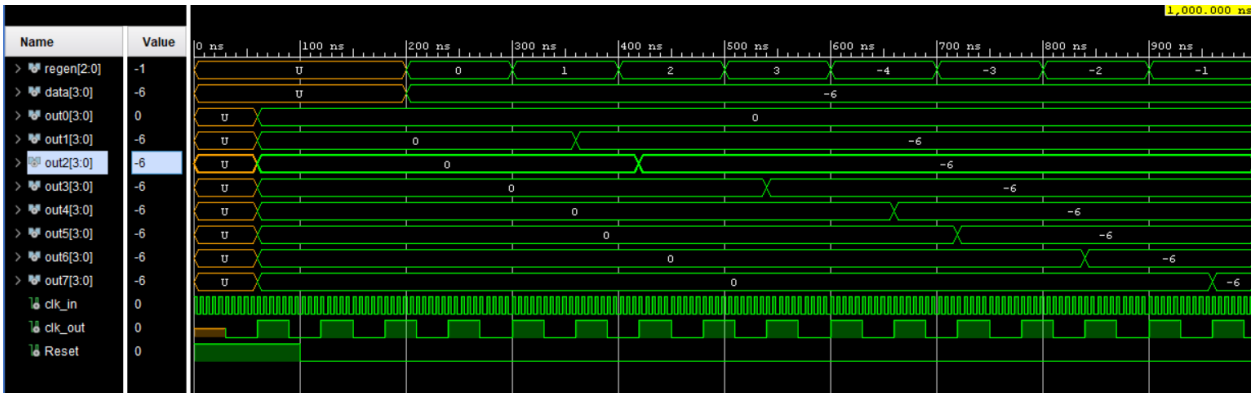
```
-- Module Name: Reg_Bank - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Reg_Bank is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Reg_En : in STD_LOGIC_VECTOR (2 downto 0);
           Data : in STD_LOGIC_VECTOR (3 downto 0);
           Data_Out0 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out1 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out2 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out3 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out4 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out5 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out6 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out7 : out STD_LOGIC_VECTOR (3 downto 0));

end Reg_Bank;

architecture Behavioral of Reg_Bank is
component Reg is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           En : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Decoder_3_to_8 is
    Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end component;
signal Dec_out : STD_LOGIC_VECTOR (7 downto 0);
signal En0,En1,En2,En3,En4,En5,En6,En7 : std_logic;
signal vData,vData_Out7 : std_logic_vector (3 downto 0);

begin
-- Assign 0000 in all register when reset button pushed
vData<= "0000" when (Reset='1') else Data;
En0<= '1' when (Reset='1') else '0';
En1<='1' when (Reset='1') else Dec_out(1);
En2<='1' when (Reset='1') else Dec_out(2);
En3<='1' when (Reset='1') else Dec_out(3);
En4<='1' when (Reset='1') else Dec_out(4);
En5<='1' when (Reset='1') else Dec_out(5);
En6<='1' when (Reset='1') else Dec_out(6);
En7<='1' when (Reset='1') else Dec_out(7);
Decoder_3_to_80:Decoder_3_to_8 port map(Reg_En,'1',Dec_out);
Reg0:Reg port map(vData,En0,Clk,Data_Out0);
Reg1:Reg port map(vData,En1,Clk,Data_Out1);
Reg2:Reg port map(vData,En2,Clk,Data_Out2);
```

```
Reg3:Reg port map(vData,En3,Clk,Data_Out3);
Reg4:Reg port map(vData,En4,Clk,Data_Out4);
Reg5:Reg port map(vData,En5,Clk,Data_Out5);
Reg6:Reg port map(vData,En6,Clk,Data_Out6);
Reg7:Reg port map(vData,En7,Clk,vData_Out7);
-- output R7 register for used in led seveng segment display
Data_Out7<=vData_Out7;

end Behavioral;
```

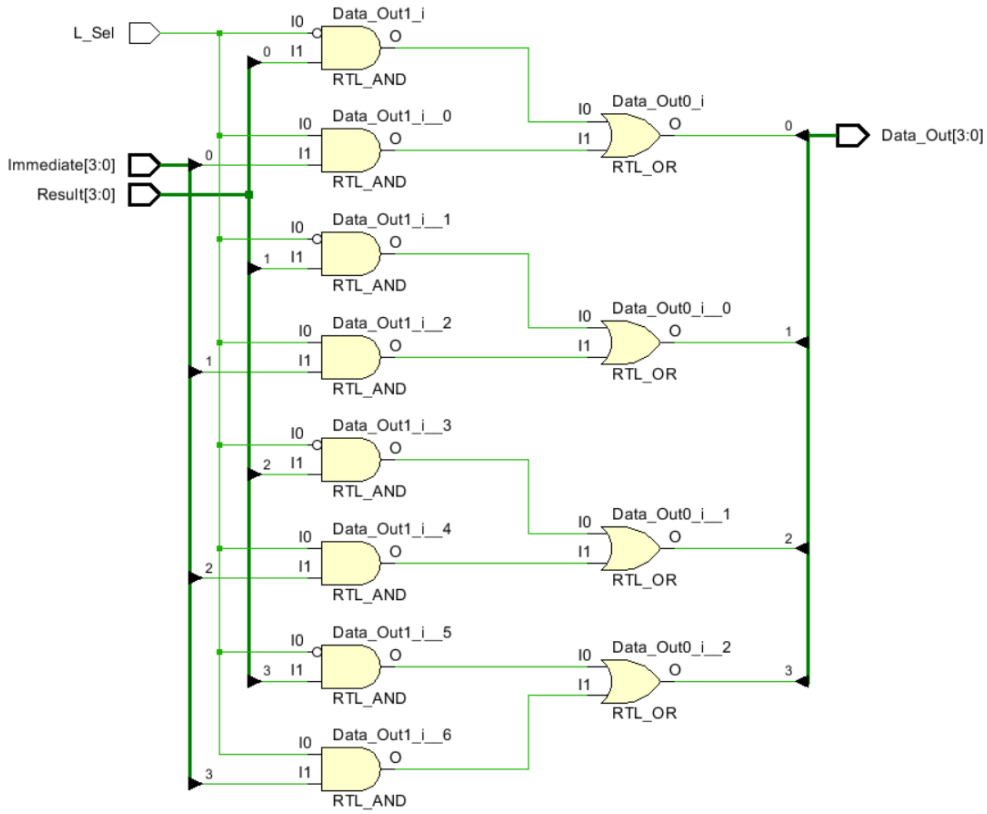
- Unit testing and Simulation

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


entity TB_Reg_Bank is
--  Port ( );
end TB_Reg_Bank;

architecture Behavioral of TB_Reg_Bank is
component Reg_Bank is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Reg_En : in STD_LOGIC_VECTOR (2 downto 0);
           Data : in STD_LOGIC_VECTOR (3 downto 0);
           Data_Out0 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out1 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out2 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out3 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out4 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out5 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out6 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end component;
signal regen: std_logic_vector (2 downto 0);
signal data , out0,out1,out2,out3,out4,out5,out6,out7 : std_logic_vector(3
downto 0);
signal clk_in,clk_out,Reset : std_logic;
begin
UUT_sc:Slow_Clk port map(clk_in,clk_out);
UUT_rb:Reg_Bank port
map(clk_out,Reset,regen,data,out0,out1,out2,out3,out4,out5,out6,out7);

process begin
clk_in<='1';
wait for 3 ns;
clk_in<='0';
wait for 3 ns;
end process;
```

```
process begin
Reset<='1';
wait for 100 ns;
Reset<='0';
wait for 100 ns;
Data<="1010";
regen<="000";
wait for 100 ns;
regen<="001";
wait for 100 ns;
regen<="010";
wait for 100 ns;
regen<="011";
wait for 100 ns;
regen<="100";
wait for 100 ns;
regen<="101";
wait for 100 ns;
regen<="110";
wait for 100 ns;
regen<="111";
wait;


end process;
end Behavioral;
```



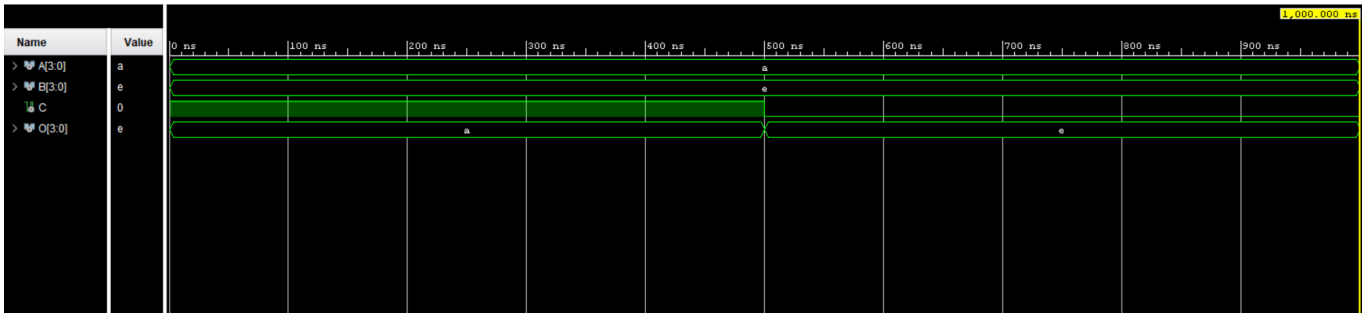
- Load Selector MUX




```
-- Module Name: MUX_2_way_4 - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- 2 way 4 bit mux as load selector
entity MUX_2_way_4 is
    Port (
        Immediate : in STD_LOGIC_VECTOR (3 downto 0);  -- Immediate input
data
        Result : in STD_LOGIC_VECTOR (3 downto 0);     -- Result input data
        L_Sel : in STD_LOGIC;                          -- Selector input
        Data_Out : out STD_LOGIC_VECTOR (3 downto 0)   -- Output data
    );
end MUX_2_way_4;

architecture Behavioral of MUX_2_way_4 is
    -- Logic to determine output based on selector L_Sel
begin
    Data_Out(0)<=(not L_Sel and Result(0)) or (L_Sel and Immediate(0));
    Data_Out(1)<=(not L_Sel and Result(1)) or (L_Sel and Immediate(1));
    Data_Out(2)<=(not L_Sel and Result(2)) or (L_Sel and Immediate(2));
    Data_Out(3)<=(not L_Sel and Result(3)) or (L_Sel and Immediate(3));
end Behavioral;
```

- Unit testing and Simulation

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

entity TB_MUX_4_T_Buf is
--  Port ( );
end TB_MUX_4_T_Buf;

architecture Behavioral of TB_MUX_4_T_Buf is

component MUX_2_way_4 is
    Port ( Immediate : in STD_LOGIC_VECTOR (3 downto 0);
           Result : in STD_LOGIC_VECTOR (3 downto 0);
           L_Sel : in STD_LOGIC;
           Data_Out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal A,B : STD_LOGIC_VECTOR (3 downto 0);
signal C : STD_LOGIC;
signal O : STD_LOGIC_VECTOR (3 downto 0);  -- Initialize O

begin
UUT: MUX_2_way_4 port map(Immediate=>A,Result => B, L_Sel => C, Data_Out =>
O);

process
begin
    A<="1010";
    B<="1110";
    C<='1';
    wait for 500 ns;
    C<='0';
    wait;
end process;
end Behavioral;
```



- Instruction Decoder

```
-- Module Name: Instruction_Dec - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Instruction_Dec is
    Port ( Ins : in STD_LOGIC_VECTOR (13 downto 0);
           Reg_JMP : in STD_LOGIC_VECTOR (3 downto 0);
           L_Sel : out STD_LOGIC;
           Immediate : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_Sel_A : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_Sel_B : out STD_LOGIC_VECTOR (2 downto 0);
           A_S_Sel : out STD_LOGIC;
           Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
           JMP_F : out STD_LOGIC;
           JMP_Add : out STD_LOGIC_VECTOR (2 downto 0);
           A_S_En:out std_logic);
end Instruction_Dec;

architecture Behavioral of Instruction_Dec is

signal insts:std_logic_vector(3 downto 0);
signal regA,regB:std_logic_vector(2 downto 0);

--signal value:std_logic_vector(3 downto 0);
```

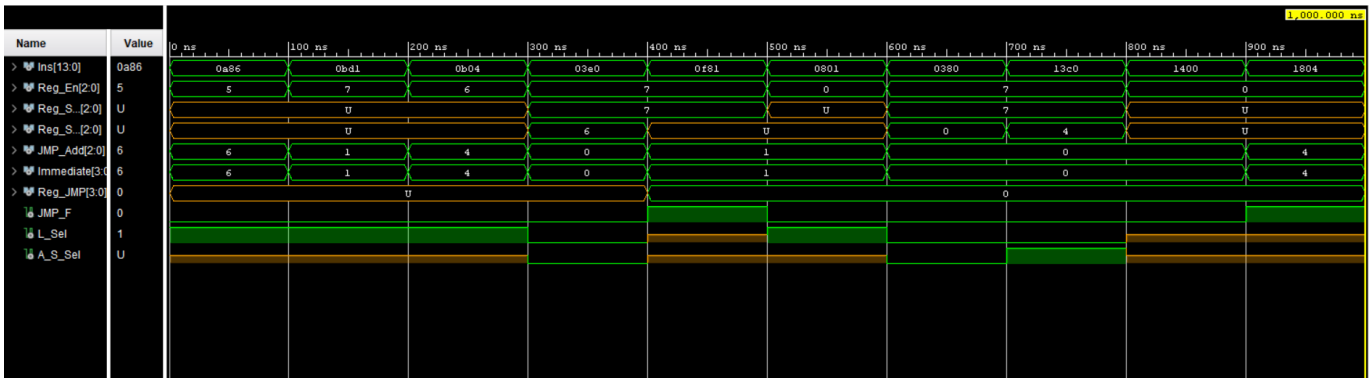
```
signal
vJMP_F,v1L_Sel,v2L_Sel,default_L_Sel,v1A_S_Sel,v2A_S_Sel,v3A_S_Sel:std_logic;
signal vReg_En,default_vector,vReg_Sel_B,vReg_Sel_A:std_logic_vector(2 downto
0);

begin
    -- Decode the instruction
    Immediate <= Ins(3 downto 0); -- This is always directly decoded from Ins
    insts <= Ins(13 downto 10); --OP_code extraction
    regA <= Ins(9 downto 7);    -- Register A signal extraction
    regB <= Ins(6 downto 4);    -- Register B signal extraction
    Reg_En<=regA;               -- Register enable output extraction

 -- Load selector between immediate and Adder/Subtractor output
    -- Load selector ''0' output selection
    v1L_Sel<=not(insts(3) or insts(2) or insts(1)) or ( not(insts(3) or
insts(1) or insts(0)) and insts(2) );
    -- Undefine Load selector output when not '0' using undefined signal
    v2L_Sel<=not v1L_Sel when (v1L_Sel='1') else default_L_Sel;
    -- Assign '1' to load selector for suitable operations. So It will only
output '1' and '0' for prefered instructions and undefined at other
instructions
    L_Sel <= insts(1) and not (insts(3) or insts(2) or insts(0))
when(insts(1)='1' and insts(3)='0' and insts(2)='0' and insts(0)='0')  else
v2L_Sel;

 -- JMP_Flag output
    JMP_F <= ( NOT(insts(3)) AND not insts(2) AND insts(1) AND (insts(0)) AND
( NOT ( Reg_JMP(3) or Reg_JMP(2) or Reg_JMP(1) or Reg_JMP(0) )) ) OR (not
insts(3) AND insts(2) AND  insts(1) AND NOT(insts(0)));

 -- JMP_Address get by last 3 bits
    JMP_Add<=Ins(2 downto 0);

 -- RegSel_A output to give to Register select from register bank to give
input to Adder/Subtractor
    vReg_Sel_A<="000" when ((not(insts(3) or insts(2) or insts(1)) and
insts(0))='1') else default_vector;
    Reg_Sel_A <= regA when (NOT insts(3) AND (( not insts(1) and not
insts(0)) or (not insts(2)and insts(1) and insts(0) ) ))='1' else vReg_Sel_A;

 -- Reg_Selector for B
    -- Get regA or undefined to a signal. For RegA for RegA selecting moments
and others as undefined
    vReg_Sel_B<=regA when (NOT(insts(3)) AND NOT(insts(2)) AND NOT insts(1)
AND (insts(0)))='1' else default_vector;
    -- Get regB or regA or undefined as suitable
    Reg_Sel_B <=regB when (NOT(insts(3)) AND NOT(insts(1)) AND NOT(insts(0))
AND (insts(2) or not insts(2)))='1' else vReg_Sel_B;

 --Adder Subtractor Selector
    v1A_S_Sel<= not (insts(3) or insts(2) or insts(1) or insts(0));
    v2A_S_Sel<= not v1A_S_Sel when (v1A_S_Sel='1') else default_L_Sel;
    v3A_S_Sel <= (NOT(insts(3)) AND NOT(insts(1)) and (insts(2) xor
insts(0)));
    A_S_Sel<= v3A_S_Sel when (v3A_S_Sel='1') else v2A_S_Sel;
    A_S_En<=NOT(insts(3) or insts(1) or insts(0));
```

```
end Behavioral;
```

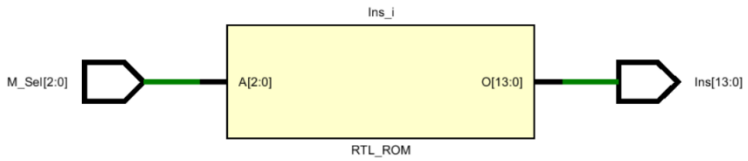
- Unit testing and Simulation

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

entity TB_Ins is
--  Port ( );
end TB_Ins;
architecture Behavioral of TB_Ins is
component Instruction_Dec is
    Port ( Ins : in STD_LOGIC_VECTOR (13 downto 0);
           Reg_JMP : in STD_LOGIC_VECTOR (3 downto 0);
           L_Sel : out STD_LOGIC;
           Immediate : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_Sel_A : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_Sel_B : out STD_LOGIC_VECTOR (2 downto 0);
           A_S_Sel : out STD_LOGIC;
           Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
           JMP_F : out STD_LOGIC;
           JMP_Add : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal Ins:STD_LOGIC_VECTOR (13 downto 0);
signal Reg_En,Reg_Sel_A,Reg_Sel_B,JMP_Add:STD_LOGIC_VECTOR (2 downto 0);
signal Immediate,Reg_JMP:STD_LOGIC_VECTOR (3 downto 0);
signal JMP_F,L_Sel,A_S_Sel:STD_LOGIC;

begin
UUT:Instruction_Dec port
map(Ins=>Ins,A_S_Sel=>A_S_Sel,L_Sel=>L_Sel,Reg_JMP=>Reg_JMP,Reg_En=>Reg_En,Im
mediate=>Immediate,Reg_Sel_A=>Reg_Sel_A,Reg_Sel_B=>Reg_Sel_B,JMP_F=>JMP_F,JMP
_Add=>JMP_Add);

process begin
    Ins<="00101010000110";
    wait for 100 ns;
    Ins<="00101111010001";
    wait for 100 ns;
    Ins<="00101100000100";
    wait for 100 ns;
    Ins<="00001111100000";
    wait for 100 ns;
    Reg_JMP<="0000";
    Ins<="00111110000001";
    wait for 100 ns;
    Ins<="00100000000001";
    wait for 100 ns;
    Ins<="00001110000000";
    wait for 100 ns;
    Ins<="01001111000000";
    wait for 100 ns;
    Ins<="01010000000000";
    wait for 100 ns;
    Ins<="01100000000100";
```

```
    wait for 100 ns;
end process;
end Behavioral;
```



- Program ROM



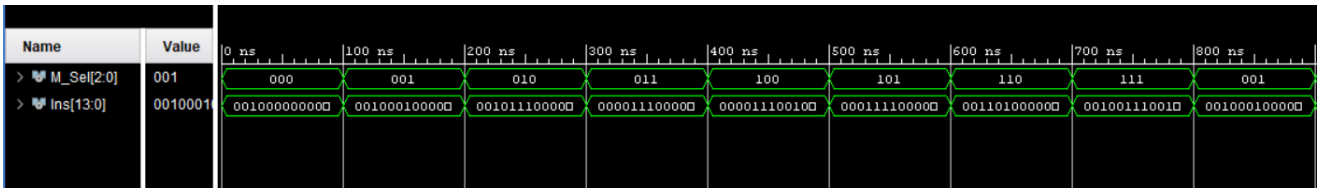

```
-- Module Name: Program_Rom - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Program_Rom is
    Port ( M_Sel : in STD_LOGIC_VECTOR (2 downto 0);
           Ins : out STD_LOGIC_VECTOR (13 downto 0));
end Program_Rom;

architecture Behavioral of Program_Rom is

    --         Instructions       --

-- NAME OP_CODE REG_A REG_B VALUE Description
-- movi    0010  111    000  1010   Move value to Register A
-- add     0000  111    101  0000   Final result will be saved in Register A
-- neg     0001  111    000  0000   Negative in two's complement is stored in
Register A
-- JZR     0011  111    000  0101   If Register A's value is Zero then jump to
instruction given by last 3 bit
-- sub     0100  111    101  0000   Subtract Register B's value from Register A
and stored in Register A
-- NOP     0101  000    000  0000   Pass clock cycle and do nothing
```

```
-- JMP    0110  000   000  0111   Jump to the instruction given by last 3
bits

type rom_type is array (0 to 7) of std_logic_vector(13 downto 0);
signal Instruction_ROM : rom_type := (
 "00100000000111",  -- 000
 "00101110000101",  -- 001
 "00001110000000",  -- 010
 "00100000001001",  -- 011
 "00101110001011",  -- 100
 "00001110000000",  -- 101
 "01010000000110",  -- 110
 "00100110001001"); -- 111
begin

Ins<=Instruction_ROM(to_integer(unsigned(M_Sel)));
end Behavioral;
```

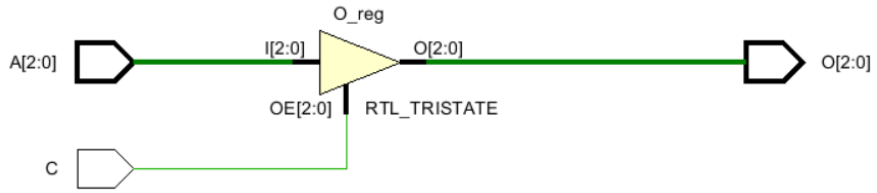
- Unit testing and Simulation

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

entity TB_programrom is
--  Port ( );
end TB_programrom;

architecture Behavioral of TB_programrom is
component Program_Rom is
    Port ( M_Sel : in STD_LOGIC_VECTOR (2 downto 0);
           Ins : out STD_LOGIC_VECTOR (13 downto 0));
end component;
signal M_Sel:std_logic_vector(2 downto 0);
signal Ins:std_logic_vector(13 downto 0);
begin
UUT:Program_Rom port map(M_Sel,Ins);
process begin
M_Sel<="000";
wait for 100 ns;
M_Sel<="001";
wait for 100 ns;
M_Sel<="010";
wait for 100 ns;
M_Sel<="011";
wait for 100 ns;
M_Sel<="100";
wait for 100 ns;
M_Sel<="101";
wait for 100 ns;
M_Sel<="110";
wait for 100 ns;
M_Sel<="111";
wait for 100 ns;
M_Sel<="001";
wait for 100 ns;
```

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



- ***Tri State Buffer – 3bit***

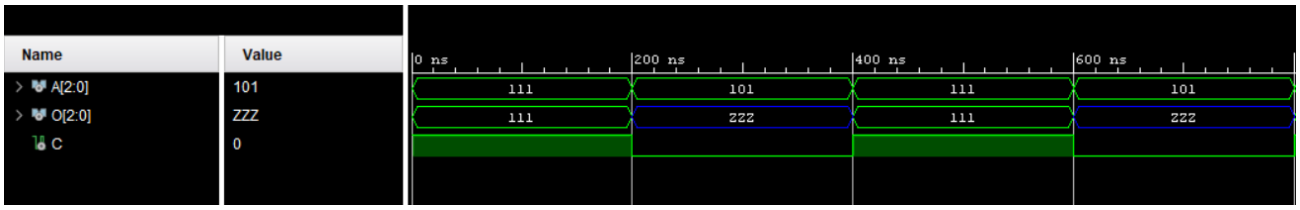



```
-- Module Name: TBuffer_3_Bit - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TBuffer_3_Bit is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (2 downto 0));
end TBuffer_3_Bit;

architecture Behavioral of TBuffer_3_Bit is

begin
O<=A when (C='1') else "ZZZ";

end Behavioral;
```

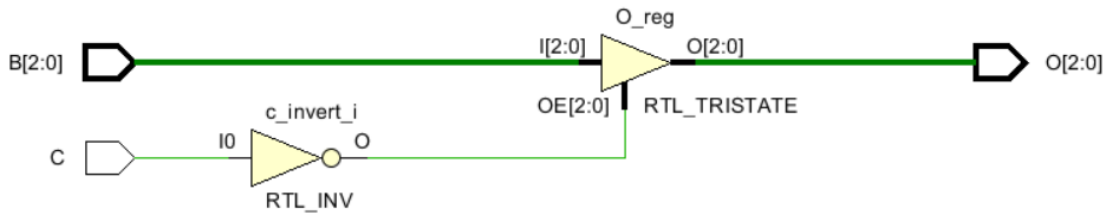
- Unit testing and Simulation

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

entity TB_Tri_normal is
--  Port ( );
end TB_Tri_normal;
```

```
architecture Behavioral of TB_Tri_normal is
component TBuffer_3_Bit is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal A,O:STD_LOGIC_VECTOR (2 downto 0);
signal C:std_logic;

begin
UUT:TBuffer_3_Bit port map(A,C,O);

process begin
    A<="111";
    C<='1';
    wait for 100 ns;
    A<="101";
    C<='0';
    wait;
end process;

end Behavioral;
```



- Tri State Buffer (Inverted) – 3bit

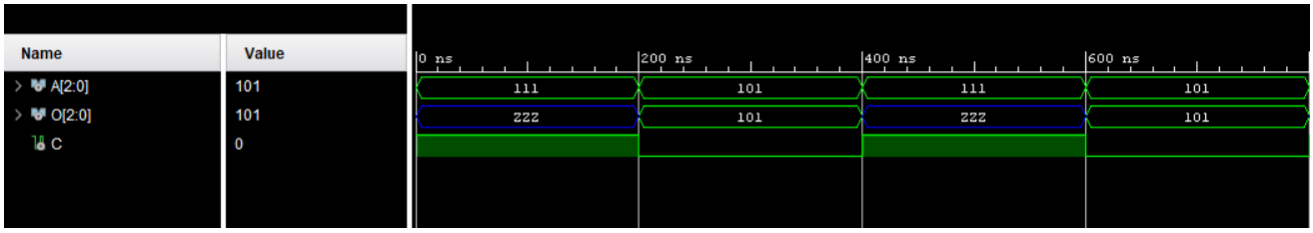



```
-- Module Name: TBuffer_3_Bit_I - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TBuffer_3_Bit_I is
    Port ( B : in STD_LOGIC_VECTOR (2 downto 0);
```

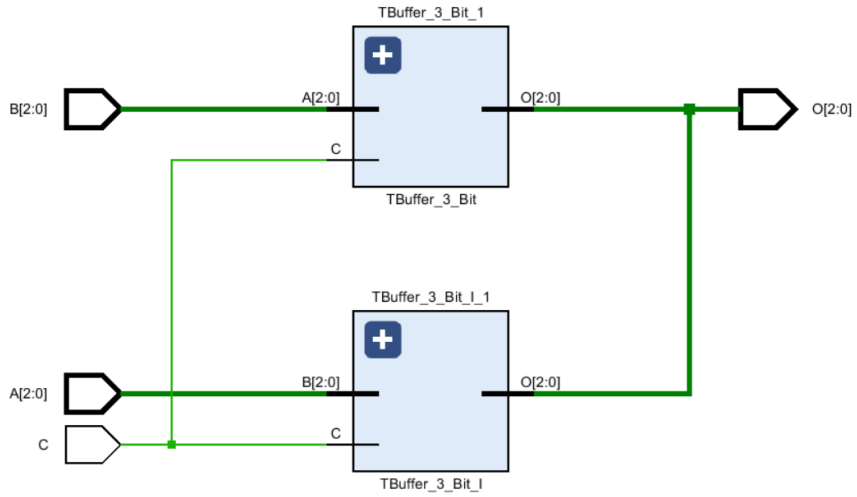
```
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (2 downto 0));
end TBuffer_3_Bit_I;

architecture Behavioral of TBuffer_3_Bit_I is

signal c_invert:std_logic;
begin
      c_invert <= not C;
      O <=B when (c_invert='1') else "ZZZ";
end Behavioral;
```

- Unit testing and Simulation

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

entity TB_Tri_Invert is
--  Port ( );
end TB_Tri_Invert;

architecture Behavioral of TB_Tri_Invert is

component TBuffer_3_Bit_I is
    Port ( B : in STD_LOGIC_VECTOR (2 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal A,O:STD_LOGIC_VECTOR (2 downto 0);
signal C:std_logic;

begin
UUT:TBuffer_3_Bit_I port map(A,C,O);

process begin
    A<="111";
    C<='1';
    wait for 200 ns;
    A<="101";
    C<='0';
    wait for 200 ns;

end process;

end Behavioral;
```



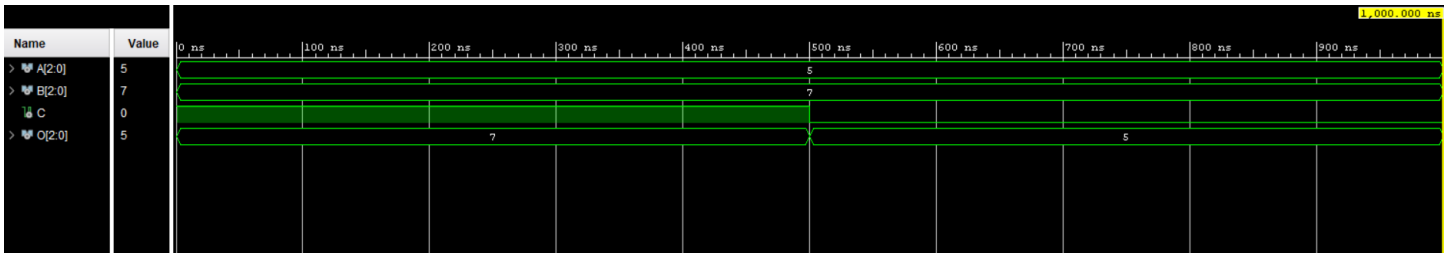
- 2×1 Multiplexer – 3 bit



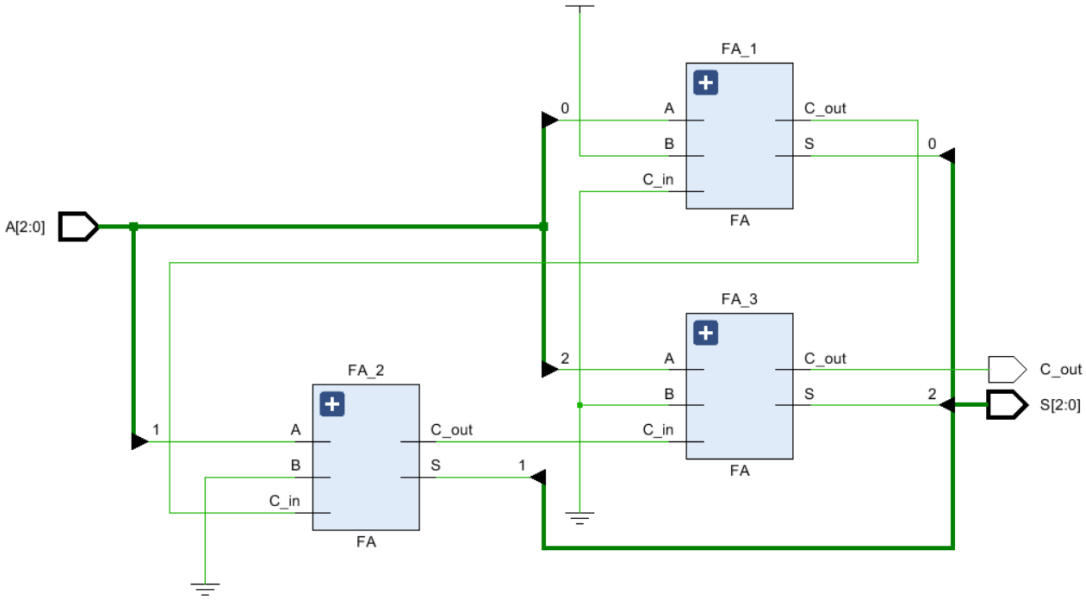

```
-- Module Name: TBuffer_2_way_3 - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- 2 way mux using 2 tri state buffers
entity TBuffer_2_way_3 is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           B : in STD_LOGIC_VECTOR (2 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (2 downto 0));
end TBuffer_2_way_3;

architecture Behavioral of TBuffer_2_way_3 is
-- Non inverted control tristate buffer to output the same input under '1'
signal
component TBuffer_3_Bit is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (2 downto 0));
end component;
-- Inverted control tristate buffer to output the same input under '0' signal
component TBuffer_3_Bit_I is
    Port ( B : in STD_LOGIC_VECTOR (2 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal A_out,B_out:STD_LOGIC_VECTOR (2 downto 0);
begin
TBuffer_3_Bit_1:TBuffer_3_Bit port map(B,C,B_out);
TBuffer_3_Bit_I_1:TBuffer_3_Bit_I PORT MAP(A,C,A_out);
O<= A_out;
O<= B_out;
end Behavioral;
```

- Unit testing and Simulation

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

entity TB_tristate is
--  Port ( );
end TB_tristate;

architecture Behavioral of TB_tristate is
component TBuffer_2_way_3 is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           B : in STD_LOGIC_VECTOR (2 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal A,B : STD_LOGIC_VECTOR (2 downto 0);
signal C : STD_LOGIC;
signal O : STD_LOGIC_VECTOR (2 downto 0);  -- Initialize O

begin
UUT: TBuffer_2_way_3 port map(A=>A,B => B, C => C, O => O);

process
begin
A<="101";
B<="111";
C<='1';
wait for 500 ns;
C<='0';
wait;
end process;
end Behavioral;
```



- Ripple Carry Adder – 3 bit



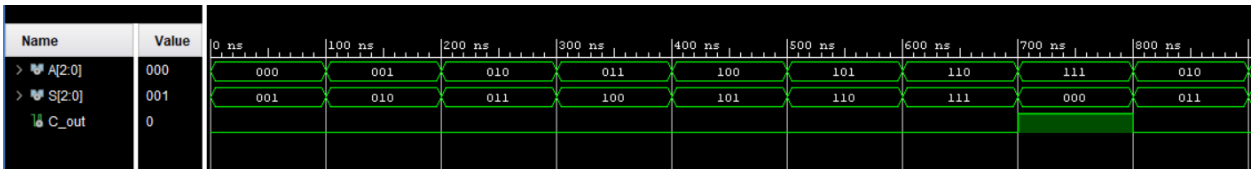

```
-- Module Name: RCA_3 - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Ripple carry adder that capable of adding 3 bit number
-- Used for incrementing instruction counter signal
entity RCA_3 is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           S : out STD_LOGIC_VECTOR (2 downto 0);
           C_out : out STD_LOGIC);
end RCA_3;

architecture Behavioral of RCA_3 is

-- Full adders used for creating RCA
component FA
 port (
 A: in std_logic;
 B: in std_logic;
 C_in: in std_logic;
 S: out std_logic;
 C_out: out std_logic);
 end component;

 SIGNAL FA0_S, FA0_C, FA1_S, FA1_C, FA2_S, FA2_C : std_logic;
begin

    FA_1 : FA
    port map (
    A => A(0),
    B => '1', -- Incrementing step
```

```
    C_in => '0', -- Set to ground
    S => S(0),
    C_Out => FA0_C);

    FA_2 : FA
    port map (
    A => A(1),
    B => '0',
    C_in => FA0_C,
    S => S(1),
    C_Out => FA1_C);

    FA_3 : FA
    port map (
    A => A(2),
    B => '0',
    C_in => FA1_C,
    S => S(2),
    C_Out => C_out);

  end Behavioral;
```

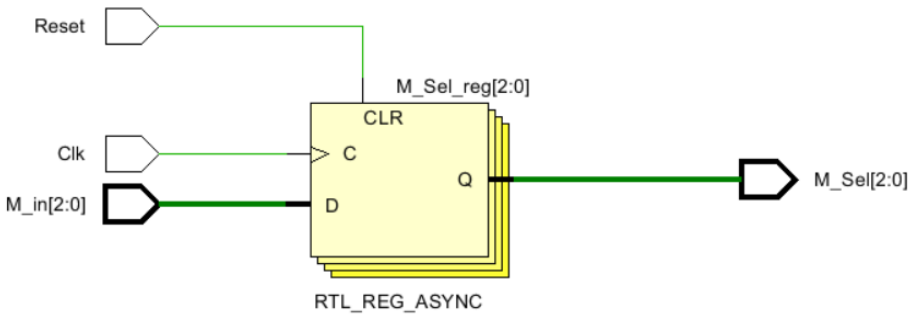
- Unit testing and Simulation

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

entity TB_REG is
--  Port ( );
end TB_REG;

architecture Behavioral of TB_REG is
COMPONENT Reg is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           En : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end COMPONENT;
COMPONENT Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end COMPONENT;

SIGNAL D :  STD_LOGIC_VECTOR (3 downto 0);
SIGNAL En :  STD_LOGIC;
SIGNAL  Clk_in,Clk_out: STD_LOGIC;
SIGNAL Q :  STD_LOGIC_VECTOR (3 downto 0);
begin
UUT:Reg PORT MAP(D,En,Clk_out,Q);
UUT1:Slow_Clk PORT MAP(Clk_in,Clk_out);
PROCESS BEGIN
Clk_in<='1';
wait for 3 ns;
Clk_in<='0';
wait for 3 ns;
```

```
end process;
process begin
D<="1001";
En<='1';
wait for 100 ns;
En<='0';
wait for 100 ns;
D<="1111";
En<='1';
wait for 100 ns;
En<='0';
END PROCESS;

end Behavioral;
```



- Program Counter



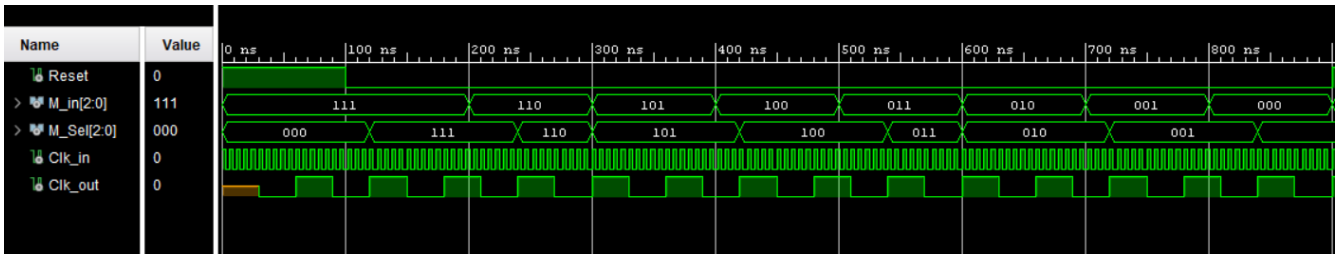

```
-- Module Name: Program_Counter - Behavioral

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

-- Instruction counter.
-- Press Reset button to start from 000 instruction.
-- Output Mux or normal counting accordingly
-- Synchronized by a slow clock
entity Program_Counter is
    Port ( Reset : in STD_LOGIC;
           Clk : in std_logic;
           M_in : in STD_LOGIC_VECTOR (2 downto 0);
           M_Sel : out STD_LOGIC_VECTOR (2 downto 0));
end Program_Counter;
```

```
architecture Behavioral of Program_Counter is
begin
process(Clk, Reset)
    begin
        if Reset = '1' then -- check for Reset press
            -- Reset value
            M_Sel<="000";
        elsif rising_edge(Clk) then
            M_Sel<=M_in; -- Update register input from mux output
        end if;
    end process;
end Behavioral;
```

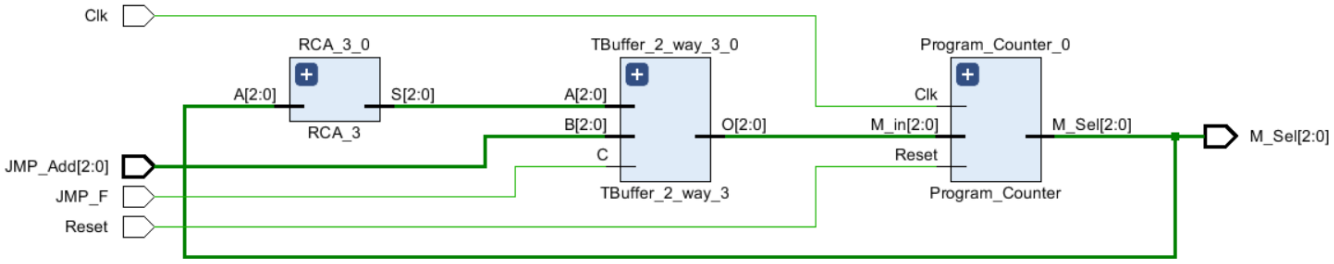
- Unit testing and Simulation

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

entity TB_programcounter is
--  Port ( );
end TB_programcounter;

architecture Behavioral of TB_programcounter is
component Program_Counter is
    Port ( Reset : in STD_LOGIC;
           Clk : in std_logic;
           M_in : in STD_LOGIC_VECTOR (2 downto 0);
           M_Sel : out STD_LOGIC_VECTOR (2 downto 0));
end component;
COMPONENT Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end COMPONENT;

signal Reset:std_logic;
signal M_in,M_Sel :  STD_LOGIC_VECTOR (2 downto 0);
SIGNAL  Clk_in,Clk_out: STD_LOGIC;
begin

UUT:Program_Counter port map(Reset,Clk_out,M_in,M_Sel);
UUT1:Slow_Clk PORT MAP(Clk_in,Clk_out);
PROCESS BEGIN
Clk_in<='1';
wait for 3 ns;
Clk_in<='0';
wait for 3 ns;
end process;
process begin
Reset<='1';
M_in<="111";
wait for 100 ns;
Reset<='0';
wait for 100 ns;
```

```
M_in<="110";
wait for 100 ns;
M_in<="101";
wait for 100 ns;
M_in<="100";
wait for 100 ns;
M_in<="011";
wait for 100 ns;
M_in<="010";
wait for 100 ns;
M_in<="001";
wait for 100 ns;
M_in<="000";
wait for 100 ns;


end process;
end Behavioral;
```



- ***Instruction Selector***



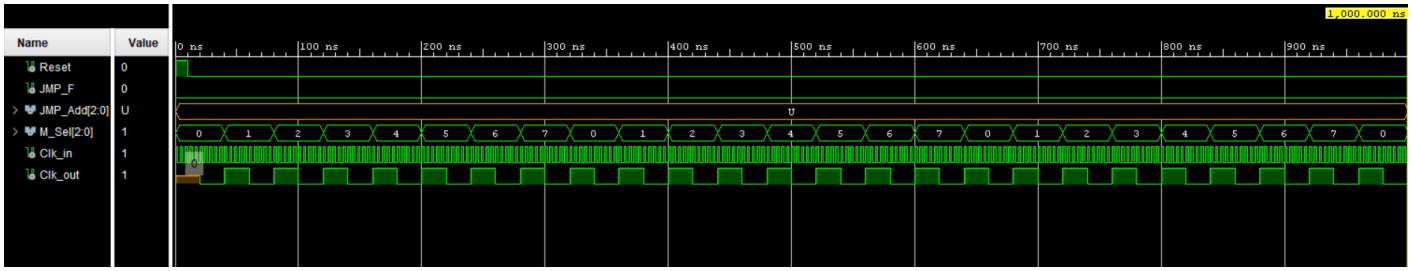

```
-- Module Name: Instruction_Selector - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Instruction_Selector is
    Port ( Reset : in STD_LOGIC;
           JMP_F : in STD_LOGIC;
           JMP_Add : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           M_Sel : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_Selector;

architecture Behavioral of Instruction_Selector is

--Tri state buffer for selecting instructions between jump address and normal
counting
component TBuffer_2_way_3 is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           B : in STD_LOGIC_VECTOR (2 downto 0);
           C : in STD_LOGIC;
           O : out STD_LOGIC_VECTOR (2 downto 0));
end component;

-- Get output and Reset when button pushed
component Program_Counter is
    Port ( Reset : in STD_LOGIC;
           Clk : in std_logic;
           M_in : in STD_LOGIC_VECTOR (2 downto 0);
           M_Sel : out STD_LOGIC_VECTOR (2 downto 0));
end component;

-- Increment instruction register
component RCA_3 is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           --B : in STD_LOGIC_VECTOR (2 downto 0);
           S : out STD_LOGIC_VECTOR (2 downto 0);
           C_out : out STD_LOGIC);
end component;
signal mem_out,adder_out,Mux_out:STD_LOGIC_VECTOR (2 downto 0);
```

```
begin
    Program_Counter_0:Program_Counter port map(Reset,Clk,Mux_out,mem_out);
    RCA_3_0:RCA_3 port map(mem_out,adder_out);
    TBuffer_2_way_3_0:TBuffer_2_way_3 port
map(adder_out,JMP_Add,JMP_F,Mux_out);
    M_Sel <= mem_out;
end Behavioral;
```

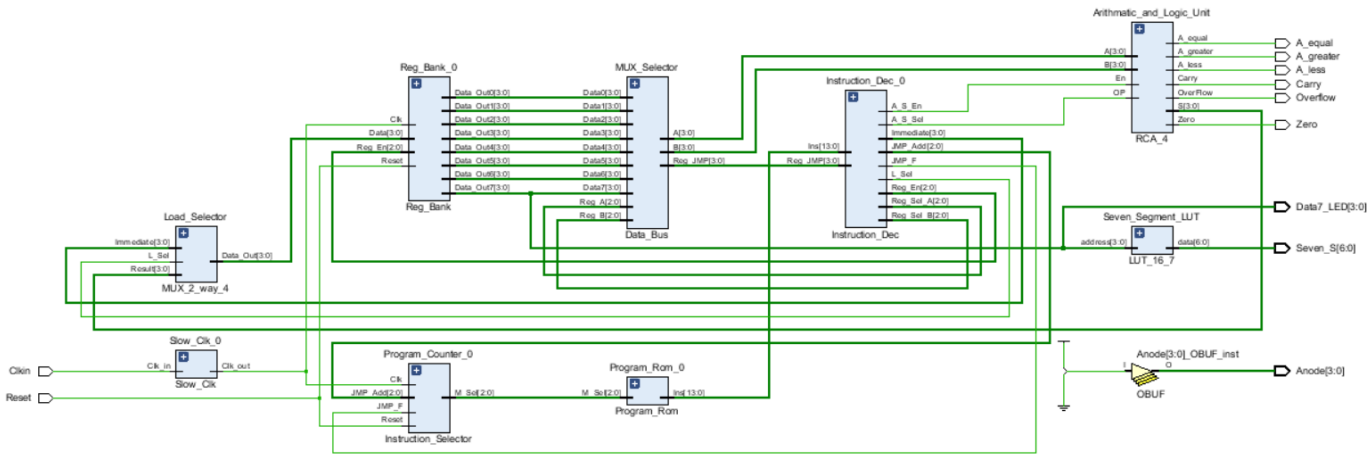
- Unit testing and Simulation

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

entity TB_PC is
--  Port ( );
end TB_PC;

architecture Behavioral of TB_PC is
component Instruction_Selector is
    Port ( Reset : in STD_LOGIC;
           JMP_F : in STD_LOGIC;
           JMP_Add : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           M_Sel : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end component;
signal Reset : std_logic := '1';
signal JMP_F : STD_LOGIC ;
signal   JMP_Add ,M_Sel: STD_LOGIC_VECTOR (2 downto 0);


 signal Clk_in,Clk_out :STD_LOGIC;

begin
UUT1:Slow_Clk port map(Clk_in,Clk_out);
UUT2:Instruction_Selector port map(Reset,JMP_F,JMP_Add,Clk_out,M_Sel);

process begin
Clk_in <= '1';
wait for 2 ns;
Clk_in <= '0';
wait for 2 ns;
end process;

process

    begin
        -- Test 1: Reset the program counter
        JMP_F <= '0';
        Reset <= '1';
        wait for 10 ns; -- Give time for the reset to take effect
```

```
        Reset <= '0';
        wait for 200 ns;
        JMP_F  <= '0';
        wait for 200 ns;
        JMP_F <= '0';
        wait for 200 ns;
        JMP_F<= '0';
        wait;
    end process;

end Behavioral;
```



- ***The Processor***




```
-- Module Name: Processor - Behavioral


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity Processor is
    Port ( Clkin : in STD_LOGIC;
           Data7_LED: out std_logic_vector (3 downto 0);
           Overflow ,Zero,Carry,A_less,A_equal,A_greater: out std_logic;
           Reset : in STD_LOGIC;
           Seven_S:out std_logic_vector (6 downto 0);
           Anode : out STD_LOGIC_VECTOR (3 downto 0));
end Processor;

architecture Behavioral of Processor is
component LUT_16_7 is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end component;

component Instruction_Selector is
    Port ( Reset : in STD_LOGIC;
           JMP_F : in STD_LOGIC;
           JMP_Add : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in std_logic;
           M_Sel : out STD_LOGIC_VECTOR (2 downto 0));
end component;
component Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end component;


component Instruction_Dec is
    Port ( Ins : in STD_LOGIC_VECTOR (13 downto 0);
```

```
            Reg_JMP : in STD_LOGIC_VECTOR (3 downto 0);
            L_Sel : out STD_LOGIC;
            Immediate : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_Sel_A : out STD_LOGIC_VECTOR (2 downto 0);
            Reg_Sel_B : out STD_LOGIC_VECTOR (2 downto 0);
            A_S_Sel : out STD_LOGIC;
            Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
            JMP_F : out STD_LOGIC;
            JMP_Add : out STD_LOGIC_VECTOR (2 downto 0);
            A_S_En:out std_logic);
end component;
component Program_Rom is
    Port ( --Reg_Ins:in STD_LOGIC_VECTOR (11 downto 0);
           M_Sel : in STD_LOGIC_VECTOR (2 downto 0);
           Ins : out STD_LOGIC_VECTOR (13 downto 0));
end component;
component MUX_2_way_4 is
    Port ( Immediate : in STD_LOGIC_VECTOR (3 downto 0);
           Result : in STD_LOGIC_VECTOR (3 downto 0);
           L_Sel : in STD_LOGIC;
           Data_Out : out STD_LOGIC_VECTOR (3 downto 0));
end component;
component Reg_Bank is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Reg_En : in STD_LOGIC_VECTOR (2 downto 0);
           Data : in STD_LOGIC_VECTOR (3 downto 0);
           Data_Out0 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out1 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out2 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out3 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out4 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out5 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out6 : out STD_LOGIC_VECTOR (3 downto 0);
           Data_Out7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;
component Data_Bus is
    Port ( Data0 : in STD_LOGIC_VECTOR (3 downto 0);
           Data1 : in STD_LOGIC_VECTOR (3 downto 0);
           Data2 : in STD_LOGIC_VECTOR (3 downto 0);
           Data3 : in STD_LOGIC_VECTOR (3 downto 0);
           Data4 : in STD_LOGIC_VECTOR (3 downto 0);
           Data5 : in STD_LOGIC_VECTOR (3 downto 0);
           Data6 : in STD_LOGIC_VECTOR (3 downto 0);
           Data7 : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_A : in STD_LOGIC_VECTOR (2 downto 0);
           Reg_B : in STD_LOGIC_VECTOR (2 downto 0);
           Reg_JMP : out STD_LOGIC_vector (3 downto 0);
           A : out STD_LOGIC_VECTOR (3 downto 0);
           B : out STD_LOGIC_VECTOR (3 downto 0));
end component;
component RCA_4 is
    Port (
            A: in STD_LOGIC_VECTOR (3 downto 0) ;
            B:in STD_LOGIC_VECTOR (3 downto 0);
            OP,En : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
```

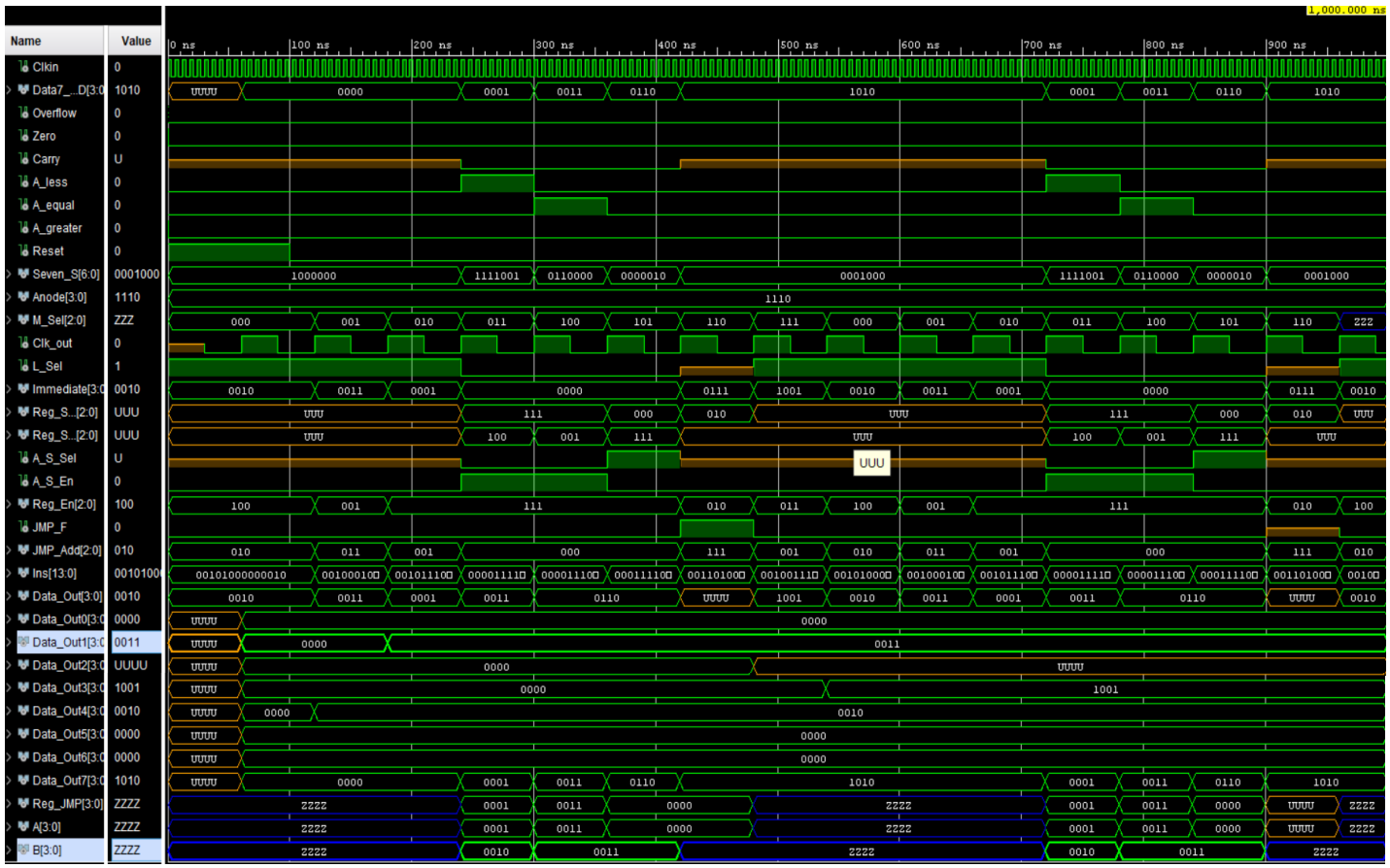
```
            OverFlow,Zero,Carry,A_less,A_equal,A_greater : out STD_LOGIC);
end component;


signal JMP_f,L_Sel,A_S_Sel,Clk,vA_S_En:std_logic;

signal A,B,Reg_check_JMP ,Immediate,Result,Data_Reg
,Reg_Data_Out0,Reg_Data_Out1,Reg_Data_Out2,Reg_Data_Out3,Reg_Data_Out4,Reg_Da
ta_Out5,Reg_Data_Out6,Reg_Data_Out7  :std_logic_vector(3 downto 0);

signal JMPadd,msel,Reg_Sel_A,Reg_Sel_B,Reg_En :std_logic_vector(2 downto 0);

signal insts :std_logic_vector(13 downto 0);

begin
-- Program counter mapping
    Program_Counter_0:Instruction_Selector port
map(Reset,JMP_f,JMPadd,Clk,msel);

-- Program Rom mapping
    Program_Rom_0:Program_Rom port map(msel,insts);

-- Instruction Decoder mapping
    Instruction_Dec_0:Instruction_Dec port
map(insts,Reg_check_JMP,L_Sel,Immediate,Reg_Sel_A,Reg_Sel_B,A_S_Sel,Reg_En,JM
P_f,JMPadd,vA_S_En);

-- Load selector mapping
    Load_Selector:MUX_2_way_4 port map(Immediate,Result,L_Sel,Data_Reg);

-- Reg Bank mapping
    Reg_Bank_0:Reg_Bank port
map(Clk,Reset,Reg_En,Data_Reg,Reg_Data_Out0,Reg_Data_Out1,Reg_Data_Out2,Reg_D
ata_Out3,Reg_Data_Out4,Reg_Data_Out5,Reg_Data_Out6,Reg_Data_Out7);

-- MUX Selector mapping
    MUX_Selector:Data_Bus port
map(Reg_Data_Out0,Reg_Data_Out1,Reg_Data_Out2,Reg_Data_Out3,Reg_Data_Out4,Reg
_Data_Out5,Reg_Data_Out6,Reg_Data_Out7,Reg_Sel_A,Reg_Sel_B,Reg_check_JMP,A,B)
;

-- ALU mapping
    Arithmatic_and_Logic_Unit:RCA_4 port
map(A,B,A_S_Sel,vA_S_En,Result,Overflow,Zero,Carry,A_less,A_equal,A_greater);

-- Slow Clock
    Slow_Clk_0:Slow_Clk port map(Clkin,Clk);

-- Seven segment lookup table
    Seven_Segment_LUT:LUT_16_7 port map(Reg_Data_Out7,Seven_S);

-- Led Output R7 register
Data7_LED<=Reg_Data_Out7;

-- Anode for seven segment
Anode <= "1110";
```

```
end Behavioral;
```

- System testing and Simulation

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

entity TB_nano is
--  Port ( );
end TB_nano;

architecture Behavioral of TB_nano is

component Processor is
    Port  ( Clkin : in STD_LOGIC;
            Data7_LED: out std_logic_vector (3 downto 0);
            Overflow ,Zero,Carry,A_less,A_equal,A_greater: out std_logic;
            Reset : in STD_LOGIC;
            Seven_S:out std_logic_vector (6 downto 0);
            Anode : out STD_LOGIC_VECTOR (3 downto 0);
            M_Sel : out STD_LOGIC_VECTOR (2 downto 0);
            Clk_out : out STD_LOGIC;
            L_Sel : out STD_LOGIC;
            Immediate : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_Sel_A : out STD_LOGIC_VECTOR (2 downto 0);
            Reg_Sel_B : out STD_LOGIC_VECTOR (2 downto 0);
            A_S_Sel : out STD_LOGIC;
            Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
            JMP_F : out STD_LOGIC;
            JMP_Add : out STD_LOGIC_VECTOR (2 downto 0);
            Ins : out STD_LOGIC_VECTOR (13 downto 0);
            Data_Out : out STD_LOGIC_VECTOR (3 downto 0);
            Data_Out0 : out STD_LOGIC_VECTOR (3 downto 0);
            Data_Out1 : out STD_LOGIC_VECTOR (3 downto 0);
            Data_Out2 : out STD_LOGIC_VECTOR (3 downto 0);
            Data_Out3 : out STD_LOGIC_VECTOR (3 downto 0);
            Data_Out4 : out STD_LOGIC_VECTOR (3 downto 0);
            Data_Out5 : out STD_LOGIC_VECTOR (3 downto 0);
            Data_Out6 : out STD_LOGIC_VECTOR (3 downto 0);
            Data_Out7 : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_JMP : out STD_LOGIC_vector (3 downto 0);
            A : out STD_LOGIC_VECTOR (3 downto 0);
            B : out STD_LOGIC_VECTOR (3 downto 0)
              );
end component;

signal Clkin : STD_LOGIC;
signal Data7_LED: std_logic_vector (3 downto 0);
signal Overflow ,Zero,Carry,A_less,A_equal,A_greater:std_logic;
signal Reset : STD_LOGIC;
signal Seven_S: std_logic_vector (6 downto 0);
signal Anode : STD_LOGIC_VECTOR (3 downto 0);

signal M_Sel :  STD_LOGIC_VECTOR (2 downto 0);
signal Clk_out :  STD_LOGIC;
```

```
signal L_Sel :  STD_LOGIC;
signal Immediate : STD_LOGIC_VECTOR (3 downto 0);
signal Reg_Sel_A : STD_LOGIC_VECTOR (2 downto 0);
signal Reg_Sel_B :  STD_LOGIC_VECTOR (2 downto 0);
signal A_S_Sel : STD_LOGIC;
signal Reg_En :  STD_LOGIC_VECTOR (2 downto 0);
signal JMP_F :  STD_LOGIC;
signal JMP_Add :  STD_LOGIC_VECTOR (2 downto 0);
signal Ins :  STD_LOGIC_VECTOR (13 downto 0);
signal Data_Out :  STD_LOGIC_VECTOR (3 downto 0);
signal Data_Out0 : STD_LOGIC_VECTOR (3 downto 0);
signal Data_Out1 : STD_LOGIC_VECTOR (3 downto 0);
signal Data_Out2 : STD_LOGIC_VECTOR (3 downto 0);
signal Data_Out3 : STD_LOGIC_VECTOR (3 downto 0);
signal Data_Out4 : STD_LOGIC_VECTOR (3 downto 0);
signal Data_Out5 : STD_LOGIC_VECTOR (3 downto 0);
signal Data_Out6 : STD_LOGIC_VECTOR (3 downto 0);
signal Data_Out7 : STD_LOGIC_VECTOR (3 downto 0);
signal Reg_JMP :  STD_LOGIC_vector (3 downto 0);
signal A :  STD_LOGIC_VECTOR (3 downto 0);
signal B :  STD_LOGIC_VECTOR (3 downto 0);

begin
    UUT:Processor port map(Clkin,Data7_LED,Overflow
,Zero,Carry,A_less,A_equal,A_greater,Reset,Seven_S,Anode,M_Sel,Clk_out,L_Sel,
Immediate,Reg_Sel_A,Reg_Sel_B,A_S_Sel,Reg_En,JMP_F,JMP_Add,Ins,Data_Out,Data_
Out0,Data_Out1,Data_Out2,Data_Out3,Data_Out4,Data_Out5,
Data_Out6,Data_Out7,Reg_JMP,A,B);

    process begin
    Clkin<='1';
    wait for 3 ns;
    Clkin<='0';
    wait for 3 ns;
    end process;
    process begin
    Reset<='1';
    wait for 100 ns;
    Reset<='0';
    wait;

end process;

end Behavioral;
```


Name
Value
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1,000.000 ns
Clkin
Data7_...D[3:0]
Overflow
Zero
Carry
A_less
A_equal
A_greater
Reset
Seven_S[6:0]
Anode[3:0]
M_Sel[2:0]
Clk_out
L_Sel
Immediate[3:0]
Reg_S...[2:0]
Reg_S...[2:0]
A_S_Sel
A_S_En
Reg_En[2:0]
JMP_F
JMP_Add[2:0]
Ins[13:0]
Data_Out[3:0]
Data_Out0[3:0]
Data_Out1[3:0]
Data_Out2[3:0]
Data_Out3[3:0]
Data_Out4[3:0]
Data_Out5[3:0]
Data_Out6[3:0]
Data_Out7[3:0]
Reg_JMP[3:0]
A[3:0]
B[3:0]

Constraint File

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports Clkin]
    set_property IOSTANDARD LVCMOS33 [get_ports Clkin]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports Clkin]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {Data7_LED[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Data7_LED[0]}]
set_property PACKAGE_PIN E19 [get_ports {Data7_LED[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Data7_LED[1]}]
set_property PACKAGE_PIN U19 [get_ports {Data7_LED[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Data7_LED[2]}]
set_property PACKAGE_PIN V19 [get_ports {Data7_LED[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Data7_LED[3]}]

set_property PACKAGE_PIN W3 [get_ports {A_greater}]
    set_property IOSTANDARD LVCMOS33 [get_ports {A_greater}]
set_property PACKAGE_PIN U3 [get_ports {A_equal}]
    set_property IOSTANDARD LVCMOS33 [get_ports {A_equal}]
set_property PACKAGE_PIN P3 [get_ports {A_less}]
    set_property IOSTANDARD LVCMOS33 [get_ports {A_less}]
set_property PACKAGE_PIN N3 [get_ports {Overflow}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Overflow}]
set_property PACKAGE_PIN P1 [get_ports {Zero}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Zero}]
set_property PACKAGE_PIN L1 [get_ports {Carry}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Carry}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {Seven_S[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_S[0]}]
set_property PACKAGE_PIN W6 [get_ports {Seven_S[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_S[1]}]
set_property PACKAGE_PIN U8 [get_ports {Seven_S[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_S[2]}]
set_property PACKAGE_PIN V8 [get_ports {Seven_S[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_S[3]}]
set_property PACKAGE_PIN U5 [get_ports {Seven_S[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_S[4]}]
set_property PACKAGE_PIN V5 [get_ports {Seven_S[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_S[5]}]
set_property PACKAGE_PIN U7 [get_ports {Seven_S[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_S[6]}]


set_property PACKAGE_PIN U2 [get_ports {Anode[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode[0]}]
set_property PACKAGE_PIN U4 [get_ports {Anode[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode[1]}]
set_property PACKAGE_PIN V4 [get_ports {Anode[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode[2]}]
set_property PACKAGE_PIN W4 [get_ports {Anode[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode[3]}]


##Buttons
set_property PACKAGE_PIN U18 [get_ports Reset]
    set_property IOSTANDARD LVCMOS33 [get_ports Reset]

```

Strategies used to optimize resource consumption

To optimize resource consumption in our Nano processor design, we employed several key strategies:

1. Avoidance of Lookup Tables for Instruction Decode - Instead of using lookup tables (LUTs) for instruction decoding, we implemented the instruction decoder using basic logic gates (AND, NOT, NOR, OR, XNOR). This approach significantly reduced the area and power consumption of the instruction decoding circuitry.

2. Utilization of Tristate Buffers Over Multiplexers - In our design, we chose to use tristate buffers for data selection instead of multiplexers. Tristate buffers helped simplify the circuitry and contributed to better resource utilization.

3. Optimized VHDL Implementation - When implementing the instruction decoder in VHDL, we avoided using if-else or case statements. Instead, we directly used logic gates, which allowed the synthesis tool to optimize the logic more efficiently. This approach resulted in a more compact and optimized hardware implementation.

These strategies were chosen to reduce the complexity of our Nano processor design, leading to a more efficient use of resources such as logic elements and power. By optimizing these aspects of our design, we aimed to create a Nano processor that delivers high performance while minimizing resource consumption.

Resource utilization report


```
------------------------------------------------------------------------------------
------------------------------------------------------------------
| Tool Version : Vivado v.2018.2.1 (win64) Build 2288692 Thu Jul 26 18:24:02
MDT 2018
| Date         : Sun May  5 01:24:29 2024
| Host         : DESKTOP-3045960 running 64-bit major release  (build 9200)
| Command      : report_utilization -file {C:/Users/DASUN/Downloads/Telegram
Desktop/latest update.xpr/utilization.txt} -name utilization_1
| Design       : Processor
| Device       : 7a35tcpg236-1
| Design State : Routed
------------------------------------------------------------------------------------
------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Slice Logic Distribution
3. Memory
4. DSP
5. IO and GT Specific
6. Clocking
7. Specific Feature
8. Primitives
9. Black Boxes
10. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs              |  117 |     0 |     20800 |  0.56 |
|   LUT as Logic          |  117 |     0 |     20800 |  0.56 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         |   57 |     0 |     41600 |  0.14 |
|   Register as Flip Flop |   57 |     0 |     41600 |  0.14 |
|   Register as Latch     |    0 |     0 |     41600 |  0.00 |
| F7 Muxes                |    0 |     0 |     16300 |  0.00 |
| F8 Muxes                |    0 |     0 |      8150 |  0.00 |
+-------------------------+------+-------+-----------+-------+


1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
```

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
3	Yes	-	Reset
0	Yes	Set	-
54	Yes	Reset	-

2. Slice Logic Distribution

Site Type	Used	Fixed	Available	Util%
Slice	47	0	8150	0.58
SLICEL	35	0		
SLICEM	12	0		
LUT as Logic	117	0	20800	0.56
using O5 output only	0			
using O6 output only	76			
using O5 and O6	41			
LUT as Memory	0	0	9600	0.00
LUT as Distributed RAM	0	0		
LUT as Shift Register	0	0		
LUT Flip Flop Pairs	8	0	20800	0.04
fully used LUT-FF pairs	0			
LUT-FF pairs with one unused LUT output	5			
LUT-FF pairs with one unused Flip Flop	8			
Unique Control Sets	6			

* Note: Review the Control Sets Report for more information regarding control sets.

3. Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	0	0	50	0.00
RAMB36/FIFO*	0	0	50	0.00
RAMB18	0	0	100	0.00

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

4. DSP

Site Type	Used	Fixed	Available	Util%
DSPs	0	0	90	0.00

5. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	23	23	106	21.70
IOB Master Pads	9			
IOB Slave Pads	14			
Bonded IPADs	0	0	10	0.00
Bonded OPADs	0	0	4	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	104	0.00
GTPE2_CHANNEL	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	106	0.00
OLOGIC	0	0	106	0.00

6. Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	1	0	32	3.13
BUFIO	0	0	20	0.00
MMCME2_ADV	0	0	5	0.00
PLLE2_ADV	0	0	5	0.00
BUFMRCE	0	0	10	0.00
BUFHCE	0	0	72	0.00
BUFR	0	0	20	0.00

7. Specific Feature

Site Type	Used	Fixed	Available	Util%
BSCANE2	0	0	4	0.00
CAPTUREE2	0	0	1	0.00
DNA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE2	0	0	1	0.00
ICAPE2	0	0	2	0.00
PCIE_2_1	0	0	1	0.00
STARTUPE2	0	0	1	0.00
XADC	0	0	1	0.00

8. Primitives

Ref Name	Used	Functional Category
LUT2	71	LUT
FDRE	54	Flop & Latch
LUT4	47	LUT
OBUF	21	IO
LUT3	15	LUT
LUT6	13	LUT
LUT5	11	LUT
CARRY4	8	CarryLogic
FDCE	3	Flop & Latch
IBUF	2	IO
LUT1	1	LUT
BUFG	1	Clock

9. Black Boxes

Ref Name	Used

10. Instantiated Netlists

Ref Name	Used

Implemented additional features

In our Nano processor design, we extended the given 12-bit instructions to 14-bit instructions to incorporate additional commands, namely SUB, NOP, and JMP, expanding the functionality of the processor.

1. Sub Command - For the SUB command, we added the opcode 0100 to the instructions and introduced new signals from the instruction decoder to the multiplexer selector to be used with the subtractor. The subtractor is a crucial component for subtracting values, as it first converts the second number to its two's complement and then adds it to the first number.

2. NOP Command - The nop command was implemented by adding logic to the instruction decoder and the opcode is 0101. This command effectively does nothing but pass the clock cycle, providing a way to introduce delays or synchronize operations.

3. JMP Command - Inspired by the JZR opcode, we implemented the jmp command to jump to a specific instruction in the program. This command is particularly useful for creating loop structures within programs, enhancing the flexibility and control flow of the processor.

4. Comparator Unit - We also added a comparator unit to the adder/subtractor unit, extending its functionality. The comparator unit includes additional LEDs to indicate whether the first value is greater, equal, or lower than the second value. This feature is especially useful for comparing values in the registers and is capable of handling both positive and negative numbers within the range of -8 to 7, given the 4-bit representation of numbers in our design.

By adding these additional features, we have enhanced the functionality and versatility of our Nano processor, providing more capabilities for executing a wider range of instructions and operations. Additionally, by extending the instruction set to 14 bits, we have ensured that the processor is easily scalable and can accommodate further enhancements and modifications in the future.

Individual contributions

(01) Illangasinghe I.M.D.P

- Designed and developed

	Deign time	Develop time
Program rom	30min	4h
Instruction decoder	2h	10h
Program counter	1h	3h
Instruction selector	30min	30min
3-bit ripple carry adder	30min	30min
2-way 3-bit multiplexer	30min	30min
2-way 4-bit multiplexer	30min	30min
4-bit Adder /subtractor	2h	3h
Comparator unit	1h	30min
Register bank	30min	30min
Mux selector	30min	2h

- Simulation and testing

  Testbenches - 2h

  4-bit Register

  Register bank

  ALU & Comparator

  Decoder 3 to 8

  Decoder 2 to 4

  MUX selector

  Seven Segment lookup table

  3-bit Ripple carry adder

  Program rom

(02) Dissanayake D.M.S.H

- Designed and developed

	Deign time	Develop time
Tri State Buffer	15min	30min
Tri State Buffer (Inverted)	15min	20min
Optimized 2-way 4-bit Mux	30min	--
8-way 4-bit multiplexer	1h	1h
Mux Selector	--	1h
2 to 4 Decoder 4-bit	30min	30min
3 to 8 Decoder 4-bit	30min	30min
Instruction Decoder	--	2h

- Simulation and testing

  Testbenches - 1h

  Tri State Buffer

  Tri State Buffer (Inverted)

  2-way 4-bit Mux

  8-way 4-bit multiplexer

  2 to 4 Decoder 4 bit

  3 to 8 Decoder 4 bit

- Creating the constraint file and mapping the outputs - 30min

- Assembling the design diagrams, VHDL codes and simulations and creating the lab report – 6h

(03) Balasuriya B.D.S.K

- Designed and developed

	Design time	Develop time
4-bit Add/Subtract unit	30min	30min
3-bit adder	20min	20min
3-bit Program Counter (PC)	40min	1h
k-way b-bit multiplexers	2h	1h
k-way b-bit Tri-state buffers	2h	1h
Register Bank	1h	30min
Program ROM	30min	30min
Instruction Decoder	--	2h

- Simulation and testing

  Testbenches - 2h

  4-bit Add/Subtract unit

  3-bit adder

  3-bit Program Counter (PC)

  k-way b-bit multiplexers

  k-way b-bit Tri-state buffers

  Register Bank

  Program ROM

(04) Fernando K.M.I

- Designed and developed

	Design time	Develop time
4-bit Add/Subtract unit	30min	30min
3-bit adder	20min	20min
3-bit Program Counter (PC)	40min	1h
k-way b-bit multiplexers	2h	1h
k-way b-bit Tri-state buffers	2h	1h
Register Bank	1h	30min
Program ROM	30min	30min
Instruction Decoder	--	2h

- Simulation and testing

  Testbenches - 2h

  4-bit Add/Subtract unit

  3-bit adder

  3-bit Program Counter (PC)

  k-way b-bit multiplexers

  k-way b-bit Tri-state buffers

  Register Bank

  Program ROM

➢ The testing on Basys 3 board was done by all the team members - 5h

CS1050 - Computer Organization and Digital Design

Nanoprocessor Design Competition

Group number – 14

Members:

Name	***Index number***
Illangasinghe I.M.D.P	220234R
Dissanayake D.M.S.H	220138C
Balasuriya B.D.S.K	220056X
Fernando K.M.I	220166J