Group members

Group 34

Name: Rathnayaka W.T
Index: 210536K

Name: Dewpura L.C.I.K
Index:210116A

Lab 9-10

1)Assigned lab task

The task was to design a 4-bit processor capable of executing 4 instruction types.Other sub tasks of this lab was to design a 4-bit arithmetic unit that can add and subtract signed integers, decode instructions to activate necessary components on the processor, design k-way b-bit multiplexers or tri-state busses and finally test their functionality via simulation.

2) Assembly program and its machine code representation

Assembly Program

```
MOVI R7,0
MOVI R1,1
MOVI R2,2
MOVI R3,3
ADD  R7,R1
ADD  R7,R2
ADD  R7,R3
JZR  R0,6
```

Machine code

```
101110000000
100010000001
100100000010
100110000011
001110010000
001110100000
```

001110110000
110000000111

3) All VHDL Codes

4-bit adder subtractor unit

Sources

ASUnit.vhd

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

entity ASUnit is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Selector : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC);
end ASUnit;

architecture Behavioral of ASUnit is

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_C, FA1_C, FA2_C, FA3_C : STD_LOGIC ; -- Carry out of each Full Adder
signal FA0_B, FA1_B, FA2_B, FA3_B : STD_LOGIC ; -- B input of each Full Adder
signal FA0_S, FA1_S, FA2_S, FA3_S : STD_LOGIC ; -- S out of each Full Adder

begin

FA_0 : FA
port map (
```

```
A => A(0),
B => FA0_B,
C_in => Selector,
S => FA0_S,
C_Out => FA0_C);

FA_1 : FA
port map (
A => A(1),
B => FA1_B,
C_in => FA0_C,
S => FA1_S,
C_Out => FA1_C);

FA_2 : FA
port map (
A => A(2),
B => FA2_B,
C_in => FA1_C,
S => FA2_S,
C_Out => FA2_C);

FA_3 : FA
port map (
A => A(3),
B => FA3_B,
C_in => FA2_C,
S => FA3_S,
C_Out => FA3_C);

-- define B input for each FA
FA0_B <= Selector XOR B(0) ;
FA1_B <= Selector XOR B(1) ;
FA2_B <= Selector XOR B(2) ;
FA3_B <= Selector XOR B(3) ;

-- Overflow = C3 XOR C4
Overflow <= FA2_C XOR FA3_C ;

-- If 0000 then Zero = 1
```

```
Zero <= (NOT(FA0_S)) AND (NOT(FA1_S)) AND (NOT(FA2_S)) AND (NOT(FA3_S)) ;

-- Define S outputs
S(0) <= FA0_S ;
S(1) <= FA1_S ;
S(2) <= FA2_S ;
S(3) <= FA3_S ;

end Behavioral;
```

FA.vhd

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

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 <= HA0_C OR HA1_C;

end Behavioral;
```

HA.vhd

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

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;
```

Simulations

TB_ASUnit.vhd

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

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

architecture Behavioral of TB_ASUnit is

component ASUnit
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
       B : in STD_LOGIC_VECTOR (3 downto 0);
       Selector : in STD_LOGIC;
       S : out STD_LOGIC_VECTOR (3 downto 0);
       Zero : out STD_LOGIC;
       Overflow : out STD_LOGIC);
end component;

signal A : STD_LOGIC_VECTOR (3 downto 0);
signal B : STD_LOGIC_VECTOR (3 downto 0);
signal Selector : STD_LOGIC;
signal S : STD_LOGIC_VECTOR (3 downto 0);
signal Zero : STD_LOGIC;
signal Overflow : STD_LOGIC;

begin

UUT : ASUnit PORT MAP (
    A => A,
    B => B,
    Selector => Selector,
    S => S,
    Zero => Zero,
    Overflow => Overflow
);

--Index 210536 - 11 0011 0110 0110 1000
--Index 210116 - 11 0011 0100 1100 0100

process

begin
```

```
-- 1000 + 0110 = 1110
A <= "1000";
B <= "0110";
Selector <= '0'; -- add

WAIT FOR 100ns ;

-- 1100 + 0110 = 1 0010
A <= "1100";
B <= "0110";
Selector <= '0'; -- add

WAIT FOR 100ns ;

-- 1100 - 0110 = 0110
A <= "1100";
B <= "0110";
Selector <= '1'; -- sub

WAIT FOR 100ns ;

--0011 - 0100 =  1111
A <= "0011";
B <= "0100";
Selector <= '1'; -- sub

WAIT FOR 100ns ;

-- 0011 - 0011 = 0000 -- check Zero flag
A <= "0011";
B <= "0011";
Selector <= '1'; -- add

WAIT ;



end process;
```

```
end Behavioral;
```

3-bit adder

Sources

Adder_3bit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Adder_3bit is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           S : out STD_LOGIC_VECTOR (2 downto 0));
end Adder_3bit;

architecture Behavioral of Adder_3bit is

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_C, FA1_C, FA2_C : STD_LOGIC ; -- Carry out of each Full Adder
signal B : STD_LOGIC_VECTOR (2 downto 0) ; -- B input 001 always as a 3 bit bus

begin

B <= "001";

FA_0 : FA
port map (
A => A(0),
B => B(0),
C_in => '0', -- No carry initially
```

```
S => S(0),
C_Out => FA0_C);

FA_1 : FA
port map (
A => A(1),
B => B(1) ,
C_in => FA0_C,
S => S(1) ,
C_Out => FA1_C);

FA_2 : FA
port map (
A => A(2),
B => B(2),
C_in => FA1_C,
S => S(2),
C_Out => FA2_C);
end Behavioral;
```

Simulations

TB_Adder_3bit.vhd

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


entity TB_Adder_3bit is
--  Port ( );
end TB_Adder_3bit;

architecture Behavioral of TB_Adder_3bit is

component Adder_3bit
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           S : out STD_LOGIC_VECTOR (2 downto 0));

end component;
```

```
signal A : STD_LOGIC_VECTOR (2 downto 0);
signal S : STD_LOGIC_VECTOR (2 downto 0);

begin

UUT : Adder_3bit PORT MAP (
	A => A,
	S => S
);

--Index 210536 - 110 011 011 001 101 000
--Index 210116 - 110 011 010 011 000 100

process

begin

--output should be 001
A <= "000";

WAIT FOR 100ns;

--output should be 010
A <= "001";

WAIT FOR 100ns;

--output should be 011
A <= "010";

WAIT FOR 100ns;

--output should be 100
A <= "011";

WAIT FOR 100ns;

--output should be 101
A <= "100";
```

```
WAIT FOR 100ns;

--output should be 110
A <= "101";

WAIT FOR 100ns;

--output should be 111
A <= "110";

WAIT;

end process;

end Behavioral;
```

3-bit Program Counter

Sources

ProgramCounter.vhd

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

entity ProgramCounter is
Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
       Res : in STD_LOGIC;
       Clk : in STD_LOGIC;
       Q : out STD_LOGIC_VECTOR (2 downto 0));
end ProgramCounter;

architecture Behavioral of ProgramCounter is

begin

 process (Clk) begin
```

```
    if(rising_edge(Clk)) then
        if Res='1' then
            Q<="000";

        else
            Q<=D;

        end if;
    end if;
 end process;

end Behavioral;
```

Simulations

TB_ProgramCounter.vhd

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

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

architecture Behavioral of TB_ProgramCounter is

component ProgramCounter

Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
       Res : in STD_LOGIC;
       Clk : in STD_LOGIC;
       Q : out STD_LOGIC_VECTOR (2 downto 0));

end component;

signal D : STD_LOGIC_VECTOR (2 downto 0);
signal Res : STD_LOGIC;
signal Clk : STD_LOGIC;
signal Q : STD_LOGIC_VECTOR(2 downto 0);
```

```
constant period : time := 80 ns;

begin

UUT : ProgramCounter PORT MAP (
        D => D,
        Res=>Res,
        Clk=> Clk,
        Q => Q
);

process
begin
    Clk <= '0';
    wait for period/2;

    Clk <= '1';
    wait for period/2 ;

end process;


--Index 210536 - 110 011 011 001 101 000
--Index 210116 - 110 011 010 011 000 100

process
begin

Res <= '0';

D <= "000";

wait for period;

D <= "100";

wait for period;

D <= "011";
```

```
wait for period;

D <= "111";

wait for period;

Res <= '1';

wait for period;

Res <= '0';

D <= "101";

wait;

end process;
```

k-way b-bit multiplexers

Sources

Mux_2_to_1_bit_3.vhd (2-way 3-bit multiplexer)

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

entity Mux_2_to_1_bit_3 is
    Port ( S : in STD_LOGIC;
           D0 : in STD_LOGIC_VECTOR (2 downto 0);
           D1 : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC_VECTOR (2 downto 0));
end Mux_2_to_1_bit_3;

architecture Behavioral of Mux_2_to_1_bit_3 is

component Mux_2_to_1
    Port ( S : in STD_LOGIC ;
```

```
        D : in STD_LOGIC_VECTOR (1 downto 0);
        EN : in STD_LOGIC;
        Y : out STD_LOGIC);
end component;

signal Mux_0_D,Mux_1_D,Mux_2_D : STD_LOGIC_VECTOR (1 downto 0);

begin

Mux_2_to_1_0 : Mux_2_to_1
port map(
   S => S ,
   D => Mux_0_D,
   EN=> '1',
   Y => Y(0)

);

Mux_2_to_1_1 : Mux_2_to_1
port map(
   S => S,
   D => Mux_1_D,
   EN=> '1',
   Y => Y(1)


);

Mux_2_to_1_2 :  Mux_2_to_1
port map(
   S => S,
   D => Mux_2_D,
   EN=> '1',
   Y => Y(2)

);



-- inputs for mu;tiplexer 0
```

```
Mux_0_D(0) <= D0(0);
Mux_0_D(1) <= D1(0);

-- inputs for mu;tiplexer 1
Mux_1_D(0) <= D0(1);
Mux_1_D(1) <= D1(1);

-- inputs for mu;tiplexer 2
Mux_2_D(0) <= D0(2);
Mux_2_D(1) <= D1(2);


end Behavioral;
```

Mux_2_to_1.vhd

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

entity Mux_2_to_1 is
    Port ( S : in STD_LOGIC;
           D : in STD_LOGIC_VECTOR (1 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC);
end Mux_2_to_1;

architecture Behavioral of Mux_2_to_1 is

--If S = '0' then
--      Y <= D(0)
--  else
--      Y <= D(1)

begin
Y <= ( (NOT(S) AND D(0) ) OR ( S AND D(1) ) )  AND EN ;

end Behavioral;
```

Mux_2_to_1_bit_4.vhd (2-way 4-bit multiplexer)

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

entity Mux_2_to_1_bit_4 is
   Port ( S : in STD_LOGIC;
          D0 : in STD_LOGIC_VECTOR (3 downto 0);
          D1 : in STD_LOGIC_VECTOR (3 downto 0);
          Y : out STD_LOGIC_VECTOR (3 downto 0));
end Mux_2_to_1_bit_4;

architecture Behavioral of Mux_2_to_1_bit_4 is

component Mux_2_to_1
   Port ( S : in STD_LOGIC ;
          D : in STD_LOGIC_VECTOR (1 downto 0);
          EN : in STD_LOGIC;
          Y : out STD_LOGIC);
end component;

signal Mux_0_D,Mux_1_D,Mux_2_D,Mux_3_D : STD_LOGIC_VECTOR (1 downto 0);

begin

Mux_2_to_1_0 : Mux_2_to_1
port map(
   S => S ,
   D => Mux_0_D,
   EN=> '1',
   Y => Y(0)

);

Mux_2_to_1_1 : Mux_2_to_1
port map(
   S => S,
   D => Mux_1_D,
```

```
    EN=> '1',
    Y => Y(1)


);

Mux_2_to_1_2 :  Mux_2_to_1
port map(
    S => S,
    D => Mux_2_D,
    EN=> '1',
    Y => Y(2)

);

Mux_2_to_1_3 :  Mux_2_to_1
port map(
    S => S,
    D => Mux_3_D,
    EN=> '1',
    Y => Y(3)

);



-- inputs for mu;tiplexer 0
Mux_0_D(0) <= D0(0);
Mux_0_D(1) <= D1(0);

-- inputs for mu;tiplexer 1
Mux_1_D(0) <= D0(1);
Mux_1_D(1) <= D1(1);

-- inputs for mu;tiplexer 2
Mux_2_D(0) <= D0(2);
Mux_2_D(1) <= D1(2);

-- inputs for mu;tiplexer 3
Mux_3_D(0) <= D0(3);
```

```
Mux_3_D(1) <= D1(3);


end Behavioral;
```

Mux_8_to_1_bit_4.vhd (8-way 4-bit multiplexer)

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

entity Mux_8_to_1_bit_4 is
    Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
           D0 : in STD_LOGIC_VECTOR (3 downto 0);
           D1 : in STD_LOGIC_VECTOR (3 downto 0);
           D2 : in STD_LOGIC_VECTOR (3 downto 0);
           D3 : in STD_LOGIC_VECTOR (3 downto 0);
           D4 : in STD_LOGIC_VECTOR (3 downto 0);
           D5 : in STD_LOGIC_VECTOR (3 downto 0);
           D6 : in STD_LOGIC_VECTOR (3 downto 0);
           D7 : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end Mux_8_to_1_bit_4;

architecture Behavioral of Mux_8_to_1_bit_4 is

component Mux_8_to_1
    Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
           D : in STD_LOGIC_VECTOR (7 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC);
end component;

signal Mux_0_D,Mux_1_D,Mux_2_D,Mux_3_D : STD_LOGIC_VECTOR (7 downto 0);

begin

Mux_8_to_1_0 : Mux_8_to_1
port map(
```

```
    S => S,
    D => Mux_0_D,
    EN=> '1',
    Y => Y(0)

);

Mux_8_to_1_1 : Mux_8_to_1
port map(
    S => S,
    D => Mux_1_D,
    EN=> '1',
    Y => Y(1)

);

Mux_8_to_1_2 : Mux_8_to_1
port map(
    S => S,
    D => Mux_2_D,
    EN=> '1',
    Y => Y(2)
);

Mux_8_to_1_3 : Mux_8_to_1
port map(
    S => S,
    D => Mux_3_D,
    EN=> '1',
    Y => Y(3)
);

-- inputs for mu;tiplexer 0
Mux_0_D(0) <= D0(0);
Mux_0_D(1) <= D1(0);
Mux_0_D(2) <= D2(0);
Mux_0_D(3) <= D3(0);
Mux_0_D(4) <= D4(0);
Mux_0_D(5) <= D5(0);
Mux_0_D(6) <= D6(0);
```

```
Mux_0_D(7) <= D7(0);

-- inputs for mu;tiplexer 1
Mux_1_D(0) <= D0(1);
Mux_1_D(1) <= D1(1);
Mux_1_D(2) <= D2(1);
Mux_1_D(3) <= D3(1);
Mux_1_D(4) <= D4(1);
Mux_1_D(5) <= D5(1);
Mux_1_D(6) <= D6(1);
Mux_1_D(7) <= D7(1);

-- inputs for mu;tiplexer 2
Mux_2_D(0) <= D0(2);
Mux_2_D(1) <= D1(2);
Mux_2_D(2) <= D2(2);
Mux_2_D(3) <= D3(2);
Mux_2_D(4) <= D4(2);
Mux_2_D(5) <= D5(2);
Mux_2_D(6) <= D6(2);
Mux_2_D(7) <= D7(2);

-- inputs for mu;tiplexer 3
Mux_3_D(0) <= D0(3);
Mux_3_D(1) <= D1(3);
Mux_3_D(2) <= D2(3);
Mux_3_D(3) <= D3(3);
Mux_3_D(4) <= D4(3);
Mux_3_D(5) <= D5(3);
Mux_3_D(6) <= D6(3);
Mux_3_D(7) <= D7(3);

end Behavioral;
```

Mux_8_to_1.vhd

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

```
entity Mux_8_to_1 is
   Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
          D : in STD_LOGIC_VECTOR (7 downto 0);
          EN : in STD_LOGIC;
          Y : out STD_LOGIC);
end Mux_8_to_1;

architecture Behavioral of Mux_8_to_1 is

component Decoder_3_to_8
port(
I: in STD_LOGIC_VECTOR;
EN: in STD_LOGIC;
Y: out STD_LOGIC_VECTOR );
end component;

signal S0 : STD_LOGIC_VECTOR (2 downto 0);
signal Y0 : STD_LOGIC_VECTOR (7 downto 0);
signal
en0,minTerm_0,minTerm_1,minTerm_2,minTerm_3,minTerm_4,minTerm_5,minTerm_6,minTer
m_7 : STD_LOGIC;

begin
Decoder_3_to_8_0 : Decoder_3_to_8
port map(
I => S0,
EN => en0,
Y => Y0 );

en0 <= EN;
S0 <= S;

minTerm_0 <= Y0(0) AND D(0);
minTerm_1 <= Y0(1) AND D(1);
minTerm_2 <= Y0(2) AND D(2);
minTerm_3 <= Y0(3) AND D(3);
minTerm_4 <= Y0(4) AND D(4);
minTerm_5 <= Y0(5) AND D(5);
minTerm_6 <= Y0(6) AND D(6);
minTerm_7 <= Y0(7) AND D(7);
```

```
Y <=  minTerm_0 OR minTerm_1 OR minTerm_2 OR minTerm_3 OR minTerm_4 OR
minTerm_5 OR minTerm_6 OR minTerm_7;

end Behavioral;
```

Decoder_3_to_8.vhd

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

entity 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 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);
 signal Y0,Y1 : STD_LOGIC_VECTOR (3 downto 0);
 signal en0,en1, I2 : STD_LOGIC;


begin

 Decoder_2_to_4_0 : Decoder_2_to_4
 port map(
 I => I0,
 EN => en0,
 Y => Y0 );
 Decoder_2_to_4_1 : Decoder_2_to_4
 port map(
```

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

Decoder_2_to_4.vhd

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

entity Decoder_2_to_4 is
    Port ( I : in STD_LOGIC_VECTOR (1 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end Decoder_2_to_4;

architecture Behavioral of Decoder_2_to_4 is

begin

Y(0) <= NOT(I(0)) AND NOT(I(1)) AND EN;
Y(1) <= I(0) AND NOT(I(1)) AND EN;
Y(2) <= NOT(I(0)) AND I(1) AND EN;
Y(3) <= I(0) AND I(1) AND EN;

end Behavioral;
```

Simulations

TB_Mux_2_to_1_bit_3.vhd

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

entity TB_Mux_2_to_1_bit_3 is
--  Port ( );
end TB_Mux_2_to_1_bit_3;

architecture Behavioral of TB_Mux_2_to_1_bit_3 is

component Mux_2_to_1_bit_3
    Port(  S : in STD_LOGIC;
           D0 : in STD_LOGIC_VECTOR (2 downto 0);
           D1 : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal S : STD_LOGIC;
signal D0: STD_LOGIC_VECTOR(2 downto 0);
signal D1: STD_LOGIC_VECTOR(2 downto 0);
signal Y : STD_LOGIC_VECTOR(2 downto 0);

begin

UUT : Mux_2_to_1_bit_3 PORT MAP (
       S  => S ,
       D0 => D0,
       D1 => D1,
       Y  => Y
);

--Index 210536 - 110 011 011 001 101 000
--Index 210116 - 110 011 010 011 000 100

process

begin

--output should be "011"
```

```
S <= '1' ;
D0 <= "110";
D1 <= "011";

WAIT FOR 100 ns;

--output should be "011"
S <= '0';
D0 <= "011";
D1 <= "001";

WAIT FOR 100 ns;

--output should be "000"
S <= '1';
D0 <= "101";
D1 <= "000";

WAIT FOR 100 ns;

--output should be "010"
S <= '0';
D0 <= "010";
D1 <= "011";

WAIT;

end process;

end Behavioral;
```

TB_Mux_2_to_1_bit_4.vhd

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

entity TB_Mux_2_to_1_bit_4 is
--  Port ( );
end TB_Mux_2_to_1_bit_4;
```

```
architecture Behavioral of TB_Mux_2_to_1_bit_4 is

component Mux_2_to_1_bit_4
    Port(  S : in STD_LOGIC;
           D0 : in STD_LOGIC_VECTOR (3 downto 0);
           D1 : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal S : STD_LOGIC;
signal D0: STD_LOGIC_VECTOR(3 downto 0);
signal D1: STD_LOGIC_VECTOR(3 downto 0);
signal Y : STD_LOGIC_VECTOR(3 downto 0);

begin

UUT : Mux_2_to_1_bit_4 PORT MAP (
       S  => S ,
       D0 => D0,
       D1 => D1,
       Y  => Y
);

--Index 210536 - 11 0011 0110 0110 1000
--Index 210116 - 11 0011 0100 1100 0100

process

begin

--output should be "0110"
S <= '1' ;
D0 <= "1000";
D1 <= "0110";

WAIT FOR 100 ns;

--output should be "0011"
S <= '0';
```

```
D0 <= "0011";
D1 <= "0110";

WAIT FOR 100 ns;

--output should be "1100"
S <= '1';
D0 <= "0100";
D1 <= "1100";

WAIT FOR 100 ns;

--output should be "1000"
S <= '0';
D0 <= "1000";
D1 <= "0100";

WAIT;

end process;

end Behavioral;
```

TB_Mux_8_to_1_bit_4.vhd

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

entity TB_Mux_8_to_1_bit_4 is
--  Port ( );
end TB_Mux_8_to_1_bit_4;

architecture Behavioral of TB_Mux_8_to_1_bit_4 is

component Mux_8_to_1_bit_4
    Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
           D0 : in STD_LOGIC_VECTOR (3 downto 0);
           D1 : in STD_LOGIC_VECTOR (3 downto 0);
           D2 : in STD_LOGIC_VECTOR (3 downto 0);
           D3 : in STD_LOGIC_VECTOR (3 downto 0);
```

```
        D4 : in STD_LOGIC_VECTOR (3 downto 0);
        D5 : in STD_LOGIC_VECTOR (3 downto 0);
        D6 : in STD_LOGIC_VECTOR (3 downto 0);
        D7 : in STD_LOGIC_VECTOR (3 downto 0);
        Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal S  : STD_LOGIC_VECTOR (2 downto 0);
signal D0 : STD_LOGIC_VECTOR (3 downto 0);
signal D1 : STD_LOGIC_VECTOR (3 downto 0);
signal D2 : STD_LOGIC_VECTOR (3 downto 0);
signal D3 : STD_LOGIC_VECTOR (3 downto 0);
signal D4 : STD_LOGIC_VECTOR (3 downto 0);
signal D5 : STD_LOGIC_VECTOR (3 downto 0);
signal D6 : STD_LOGIC_VECTOR (3 downto 0);
signal D7 : STD_LOGIC_VECTOR (3 downto 0);
signal Y  : STD_LOGIC_VECTOR (3 downto 0);

begin

UUT : Mux_8_to_1_bit_4 PORT MAP(
      S=>S,
      D0=>D0,
      D1=>D1,
      D2=>D2,
      D3=>D3,
      D4=>D4,
      D5=>D5,
      D6=>D6,
      D7=>D7,
      Y=>Y
);

--Index 210536 - 11 0011 0110 0110 1000
--Index 210116 - 11 0011 0100 1100 0100

process
begin

--select D0
```

```
S <= "000";
D0<= "0011";
D0<= "0110";
D1<= "1110";
D2<= "1000";
D3<= "1011";
D4<= "0100";
D5<= "1100";
D6<= "0101";
D7<= "0000";

wait for 100 ns;

--select D1
S <= "001";
D0<= "0011";
D0<= "0110";
D1<= "1110";
D2<= "1000";
D3<= "1011";
D4<= "0100";
D5<= "1100";
D6<= "0101";
D7<= "0000";

wait for 100 ns;

--select D2
S <= "010";
D0<= "0011";
D0<= "0110";
D1<= "1110";
D2<= "1000";
D3<= "1011";
D4<= "0100";
D5<= "1100";
D6<= "0101";
D7<= "0000";

wait for 100 ns;
```

```
--select D3
S <= "011";
D0<= "0011";
D0<= "0110";
D1<= "1110";
D2<= "1000";
D3<= "1011";
D4<= "0100";
D5<= "1100";
D6<= "0101";
D7<= "0000";

wait for 100 ns;

--select D4
S <= "100";
D0<= "0011";
D0<= "0110";
D1<= "1110";
D2<= "1000";
D3<= "1011";
D4<= "0100";
D5<= "1100";
D6<= "0101";
D7<= "0000";

wait for 100 ns;


--select D5
S <= "101";
D0<= "0011";
D0<= "0110";
D1<= "1110";
D2<= "1000";
D3<= "1011";
D4<= "0100";
D5<= "1100";
D6<= "0101";
```

```
D7<= "0000";

wait for 100 ns;


--select D6
S <= "110";
D0<= "0011";
D0<= "0110";
D1<= "1110";
D2<= "1000";
D3<= "1011";
D4<= "0100";
D5<= "1100";
D6<= "0101";
D7<= "0000";

wait for 100 ns;

--select D7
S <= "111";
D0<= "0011";
D0<= "0110";
D1<= "1110";
D2<= "1000";
D3<= "1011";
D4<= "0100";
D5<= "1100";
D6<= "0101";
D7<= "0000";

wait;


end process;

end Behavioral;
```

Register Bank

Sources

RegBank.vhd

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

entity RegBank is
    Port (
    RegBank_EN : in STD_LOGIC_VECTOR(2 downto 0); -- Register enable which selects the
register
    RegBank_Clk : in STD_LOGIC;
    RegBank_Res : in STD_LOGIC ;
    RegBank_D : in STD_LOGIC_VECTOR(3 downto 0); -- input of register bank

    -- output of register bank
    RegBank_Q_0 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 0
    RegBank_Q_1 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 1
    RegBank_Q_2 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 2
    RegBank_Q_3 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 3
    RegBank_Q_4 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 4
    RegBank_Q_5 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 5
    RegBank_Q_6 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 6
    RegBank_Q_7 : out STD_LOGIC_VECTOR(3 downto 0)  -- output of register 7
     );
end RegBank;

architecture Behavioral of RegBank is

component Reg_4_bit
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC;
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end component;
```

```
component Decoder_3_to_8
   Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
          EN : in STD_LOGIC;
          Y : out STD_LOGIC_VECTOR (7 downto 0));
end component ;

signal Decoder_output : STD_LOGIC_VECTOR(7 downto 0);
signal RegBank_Q_0_temp : STD_LOGIC_VECTOR(3 downto 0);

begin

RegBank_Decoder_3_to_8 : Decoder_3_to_8
port map(
   I => RegBank_EN ,
   EN=> '1',
   Y => Decoder_output
);


Reg_4_bit_0 : Reg_4_bit
port map(
   D => "0000", --hardcode Register 0 into "0000"
   EN=> Decoder_output(0),
   Res=>RegBank_Res,
   Clk=>RegBank_Clk,
   Q=>RegBank_Q_0_temp
);

Reg_4_bit_1 : Reg_4_bit
port map(
   D => RegBank_D,
   EN=> Decoder_output(1),
   Res=>RegBank_Res,
   Clk=>RegBank_Clk,
   Q=>RegBank_Q_1
);

Reg_4_bit_2 : Reg_4_bit
port map(
```

```
    D => RegBank_D,
    EN=> Decoder_output(2),
    Res=>RegBank_Res,
    Clk=>RegBank_Clk,
    Q=>RegBank_Q_2
);

Reg_4_bit_3 : Reg_4_bit
port map(
    D => RegBank_D,
    EN=> Decoder_output(3),
    Res=>RegBank_Res,
    Clk=>RegBank_Clk,
    Q=>RegBank_Q_3
);

Reg_4_bit_4 : Reg_4_bit
port map(
    D => RegBank_D,
    EN=> Decoder_output(4),
    Res=>RegBank_Res,
    Clk=>RegBank_Clk,
    Q=>RegBank_Q_4
);

Reg_4_bit_5 : Reg_4_bit
port map(
    D => RegBank_D,
    EN=> Decoder_output(5),
    Res=>RegBank_Res,
    Clk=>RegBank_Clk,
    Q=>RegBank_Q_5
);

Reg_4_bit_6 : Reg_4_bit
port map(
    D => RegBank_D,
    EN=> Decoder_output(6),
    Res=>RegBank_Res,
    Clk=>RegBank_Clk,
```

```
   Q=>RegBank_Q_6
);

Reg_4_bit_7 : Reg_4_bit
port map(
   D => RegBank_D,
   EN=> Decoder_output(7),
   Res=>RegBank_Res,
   Clk=>RegBank_Clk,
   Q=>RegBank_Q_7
);
RegBank_Q_0 <= "0000";
end Behavioral;
```

Reg_4_bit.vhd

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

entity Reg_4_bit is
   Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
          EN : in STD_LOGIC;
          Res : in STD_LOGIC;
          Clk : in STD_LOGIC;
          Q : out STD_LOGIC_VECTOR (3 downto 0));
end Reg_4_bit;

architecture Behavioral of Reg_4_bit is

begin

process (Clk) begin
   if(rising_edge(Clk)) then
      if En = '1' then
         if Res='1' then
            Q<="0000";
```

```
        else
            Q<=D;

        end if;
      end if;
   end if;
end process;

end Behavioral;
```

Simulations

TB_RegBank.vhd

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

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

architecture Behavioral of TB_RegBank is

component RegBank

   Port (
RegBank_EN : in STD_LOGIC_VECTOR(2 downto 0); -- Register enable which selects the
register
RegBank_Clk : in STD_LOGIC;
RegBank_Res : in STD_LOGIC ;
RegBank_D : in STD_LOGIC_VECTOR(3 downto 0); -- input of register bank

-- output of register bank
RegBank_Q_0 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 0
RegBank_Q_1 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 1
RegBank_Q_2 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 2
RegBank_Q_3 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 3
RegBank_Q_4 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 4
```

```
RegBank_Q_5 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 5
RegBank_Q_6 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 6
RegBank_Q_7 : out STD_LOGIC_VECTOR(3 downto 0)  -- output of register 7
 );
end component;

signal RegBank_EN : STD_LOGIC_VECTOR(2 downto 0); -- Register enable which selects the
register
signal RegBank_Clk : STD_LOGIC;
signal RegBank_Res : STD_LOGIC ;
signal RegBank_D : STD_LOGIC_VECTOR(3 downto 0); -- input of register bank

-- output of register bank
signal RegBank_Q_0 : STD_LOGIC_VECTOR(3 downto 0); -- output of register 0
signal RegBank_Q_1 : STD_LOGIC_VECTOR(3 downto 0); -- output of register 1
signal RegBank_Q_2 : STD_LOGIC_VECTOR(3 downto 0); -- output of register 2
signal RegBank_Q_3 : STD_LOGIC_VECTOR(3 downto 0); -- output of register 3
signal RegBank_Q_4 : STD_LOGIC_VECTOR(3 downto 0); -- output of register 4
signal RegBank_Q_5 : STD_LOGIC_VECTOR(3 downto 0); -- output of register 5
signal RegBank_Q_6 : STD_LOGIC_VECTOR(3 downto 0); -- output of register 6
signal RegBank_Q_7 : STD_LOGIC_VECTOR(3 downto 0);  -- output of register 7

constant period : time := 100 ns;

begin

UUT : RegBank PORT MAP (

   RegBank_EN => RegBank_EN,
   RegBank_Clk => RegBank_Clk,
   RegBank_Res => RegBank_Res,
   RegBank_D => RegBank_D,

   RegBank_Q_0 => RegBank_Q_0,
   RegBank_Q_1 => RegBank_Q_1,
   RegBank_Q_2 => RegBank_Q_2,
   RegBank_Q_3=>  RegBank_Q_3,
   RegBank_Q_4=>  RegBank_Q_4,
   RegBank_Q_5 => RegBank_Q_5,
   RegBank_Q_6 =>  RegBank_Q_6,
```

```
    RegBank_Q_7 => RegBank_Q_7
);



process
begin
    RegBank_Clk <= '0';
    wait for period/2;

    RegBank_Clk <= '1';
    wait for period/2 ;

end process;

--Index 210536 - 11 0011 0110 0110 1000
--Index 210116 - 11 0011 0100 1100 0100

process
begin

    RegBank_EN <= "000"; -- choose Register 0
    RegBank_Res <= '0';
    RegBank_D <= "1000";

    wait for period;

    RegBank_EN <= "001"; -- choose Register 1
    RegBank_Res <= '0';
    RegBank_D <= "1000";

    wait for period;

    RegBank_EN <= "010"; -- choose Register 2
    RegBank_Res <= '0';
    RegBank_D <= "0100";

    wait for period;

    RegBank_EN <= "011"; -- choose Register 3
```

```
    RegBank_Res <= '0';
    RegBank_D <= "0011";

    wait for period;

    RegBank_EN <= "100"; -- choose Register 4
    RegBank_Res <= '0';
    RegBank_D <= "1000";

    wait for period;

    RegBank_EN <= "101"; -- choose Register 5
    RegBank_Res <= '0';
    RegBank_D <= "0110";

    wait for period;

    RegBank_EN <= "101"; --  reset register 5
    RegBank_Res <= '1';

    wait for period;

    RegBank_EN <= "110"; -- choose Register 6
    RegBank_Res <= '0';
    RegBank_D <= "1100";

    wait for period;

    RegBank_EN <= "111"; -- choose Register 7
    RegBank_Res <= '0';
    RegBank_D <= "1000";

    wait ;

end process;

end Behavioral;
```

Program ROM

Sources

ROM.vhd

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

use ieee.numeric_std.all;

entity ROM is
    Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
           data : out STD_LOGIC_VECTOR (11 downto 0));
end ROM;

architecture Behavioral of ROM is
type rom_type is array (0 to 7) of std_logic_vector(11 downto 0);
    signal program_ROM : rom_type := (

    --main program
        "101110000000",  -- MOVI R7,0
        "100010000001", -- MOVI R1,1
        "100100000010", -- MOVI R2,2
        "100110000011", -- MOVI R3,3
        "001110010000", -- ADD  R7,R1
        "001110100000", -- ADD  R7,R2
        "001110110000", -- ADD  R7,R3
        "110000000111"  -- JZR  R0,6




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

Simulations

TB_ROM.vhd

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

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

architecture Behavioral of TB_ROM is

component ROM
Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
          data : out STD_LOGIC_VECTOR (11 downto 0));
end component;

signal address : STD_LOGIC_VECTOR (2 downto 0);
signal data : STD_LOGIC_VECTOR (11 downto 0);

begin

UUT : ROM PORT MAP (
       address => address,
       data => data
);
process
begin

address <= "000";
wait for 100 ns;

address <= "001";
wait for 100 ns;

address <= "010";
wait for 100 ns;

address <= "011";
```

```
wait for 100 ns;

address <= "100";
wait for 100 ns;

address <= "101";
wait for 100 ns;

address <= "110";
wait for 100 ns;

address <= "111";
wait ;


end process;


end Behavioral;
```

Instruction Decoder

Sources

InstructionDecoder.vhd

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

entity InstructionDecoder is
    Port ( InstructionBus : in STD_LOGIC_VECTOR (11 downto 0);
           RegJumpCheck : in STD_LOGIC_VECTOR (3 downto 0);
           RegisterEnable : out STD_LOGIC_VECTOR (2 downto 0);
           LoadSelector : out STD_LOGIC;
           ImmediateValue : out STD_LOGIC_VECTOR (3 downto 0);
           RegisterSelector_A : out STD_LOGIC_VECTOR (2 downto 0);
           RegisterSelector_B : out STD_LOGIC_VECTOR (2 downto 0);
```

```
        AddSubSelector : out STD_LOGIC;
        JumpFlag : out STD_LOGIC;
        JumpAddress : out STD_LOGIC_VECTOR (2 downto 0));
end InstructionDecoder;

architecture Behavioral of InstructionDecoder is

begin
process (InstructionBus) begin

-- MOVI
   if ((InstructionBus(11)='1') AND (InstructionBus(10)='0') ) then

      RegisterEnable(0) <= InstructionBus(7);
      RegisterEnable(1) <= InstructionBus(8);
      RegisterEnable(2) <= InstructionBus(9);

      LoadSelector <= '0'; -- Choose Immediate Value from MUX

      ImmediateValue(0) <= InstructionBus(0);
      ImmediateValue(1) <= InstructionBus(1);
      ImmediateValue(2) <= InstructionBus(2);
      ImmediateValue(3) <= InstructionBus(3);

      JumpFlag <= '0' ; -- not JMP.Just increment address by 1

   end if;

-- ADD
   if ((InstructionBus(11)='0') AND (InstructionBus(10)='0') ) then

      RegisterEnable(0) <= InstructionBus(7);
      RegisterEnable(1) <= InstructionBus(8);
      RegisterEnable(2) <= InstructionBus(9);

      RegisterSelector_A(0) <= InstructionBus(7);
      RegisterSelector_A(1) <= InstructionBus(8);
      RegisterSelector_A(2) <= InstructionBus(9);
```

```
        RegisterSelector_B(0) <= InstructionBus(4);
        RegisterSelector_B(1) <= InstructionBus(5);
        RegisterSelector_B(2) <= InstructionBus(6);

        LoadSelector <= '1'; -- Choose AddSubVAl from Mux
        AddSubSelector <= '0'; -- select addition
        JumpFlag <= '0' ; -- not JMP.Just increment address by 1

    end if;

-- NEG
    if ((InstructionBus(11)='0') AND (InstructionBus(10)='1') ) then

        RegisterEnable(0) <= InstructionBus(7);
        RegisterEnable(1) <= InstructionBus(8);
        RegisterEnable(2) <= InstructionBus(9);

        LoadSelector <= '1'; --Choose AddSumVal from Mux

        RegisterSelector_B(0) <= InstructionBus(9);
        RegisterSelector_B(1) <= InstructionBus(8);
        RegisterSelector_B(2) <= InstructionBus(7);

        RegisterSelector_A(0)<= '0';
        RegisterSelector_A(1)<= '0';
        RegisterSelector_A(2)<= '0';

        AddSubSelector <= '1';
        JumpFlag <= '0' ; -- not JMP.Just increment address by 1
    end if;

-- JZR
    if ((InstructionBus(11)='1') AND (InstructionBus(10)='1') ) then

        RegisterEnable(0) <= InstructionBus(7);
        RegisterEnable(1) <= InstructionBus(8);
        RegisterEnable(2) <= InstructionBus(9);

        RegisterSelector_A(0) <= InstructionBus(7);
        RegisterSelector_A(1) <= InstructionBus(8);
```

```
        RegisterSelector_A(2) <= InstructionBus(9);


        if (RegJumpCheck = "0000") then

            JumpFlag <= '1' ; -- JMP instruction

            JumpAddress(0) <= InstructionBus(0);
            JumpAddress(1) <= InstructionBus(1);
            JumpAddress(2) <= InstructionBus(2);
        else
            JumpFlag <= '0';
        end if;

    end if;
end process ;

end Behavioral;
```

Simulations

TB_InstructionDecoder.vhd

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

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

architecture Behavioral of TB_InstructionDecoder is
component InstructionDecoder
Port ( InstructionBus : in STD_LOGIC_VECTOR (11 downto 0);
           RegJumpCheck : in STD_LOGIC_VECTOR (3 downto 0);
           RegisterEnable : out STD_LOGIC_VECTOR (2 downto 0);
           LoadSelector : out STD_LOGIC;
           ImmediateValue : out STD_LOGIC_VECTOR (3 downto 0);
           RegisterSelector_A : out STD_LOGIC_VECTOR (2 downto 0);
           RegisterSelector_B : out STD_LOGIC_VECTOR (2 downto 0);
           AddSubSelector : out STD_LOGIC;
```

```
        JumpFlag : out STD_LOGIC;
        JumpAddress : out STD_LOGIC_VECTOR (2 downto 0));

end component;
signal InstructionBus : STD_LOGIC_VECTOR (11 downto 0);
signal RegJumpCheck :  STD_LOGIC_VECTOR (3 downto 0);
signal RegisterEnable :  STD_LOGIC_VECTOR (2 downto 0);
signal LoadSelector : STD_LOGIC;
signal ImmediateValue : STD_LOGIC_VECTOR (3 downto 0);
signal RegisterSelector_A : STD_LOGIC_VECTOR (2 downto 0);
signal RegisterSelector_B : STD_LOGIC_VECTOR (2 downto 0);
signal AddSubSelector : STD_LOGIC;
signal JumpFlag : STD_LOGIC;
signal JumpAddress : STD_LOGIC_VECTOR (2 downto 0);



begin

UUT : InstructionDecoder PORT MAP (
      InstructionBus => InstructionBus,
      RegJumpCheck => RegJumpCheck,
      RegisterEnable => RegisterEnable,
      LoadSelector => LoadSelector,
      ImmediateValue => ImmediateValue,
      RegisterSelector_A => RegisterSelector_A,
       RegisterSelector_B =>  RegisterSelector_B,
       AddSubSelector=> AddSubSelector,
       JumpFlag => JumpFlag,
       JumpAddress=> JumpAddress

);

process
begin

InstructionBus <= "100010000001";
wait for 100 ns;

InstructionBus <= "100100000010";
```

```
wait for 100 ns;

InstructionBus <= "100110000011";
wait for 100 ns;

InstructionBus <= "001110010000";
wait for 100 ns;

InstructionBus <= "001110100000";
wait for 100 ns;

InstructionBus <= "001110110000";
wait for 100 ns;

InstructionBus <= "110000000110";
RegJumpCheck <= "0000"; -- set manually to zero to eecute JMP
wait for 100 ns;

InstructionBus <= "110000000110";
RegJumpCheck <= "0001"; -- set manually to non zero to not JMP
wait for 100 ns;

InstructionBus <= "000000000000";
wait;

end process ;
```

System (slow clock+nano processor+7 segment ROM)

Sources

nano_processor.vhd

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

entity nano_processor is
    Port ( Clk : in STD_LOGIC;
```

```
        Res : in STD_LOGIC;
        Overflow : out STD_LOGIC;
        Zero : out STD_LOGIC;
        Output : out STD_LOGIC_VECTOR (3 downto 0));
end nano_processor;

architecture Behavioral of nano_processor is

component Mux_2_to_1_bit_3
 Port ( S : in STD_LOGIC;
        D0 : in STD_LOGIC_VECTOR (2 downto 0);
        D1 : in STD_LOGIC_VECTOR (2 downto 0);
        Y : out STD_LOGIC_VECTOR (2 downto 0));

end component;

component Mux_2_to_1_bit_4
Port ( S : in STD_LOGIC;
        D0 : in STD_LOGIC_VECTOR (3 downto 0);
        D1 : in STD_LOGIC_VECTOR (3 downto 0);
        Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Adder_3bit
Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
      S : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component Mux_8_to_1_bit_4
Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
      D0 : in STD_LOGIC_VECTOR (3 downto 0);
      D1 : in STD_LOGIC_VECTOR (3 downto 0);
      D2 : in STD_LOGIC_VECTOR (3 downto 0);
      D3 : in STD_LOGIC_VECTOR (3 downto 0);
      D4 : in STD_LOGIC_VECTOR (3 downto 0);
      D5 : in STD_LOGIC_VECTOR (3 downto 0);
      D6 : in STD_LOGIC_VECTOR (3 downto 0);
      D7 : in STD_LOGIC_VECTOR (3 downto 0);
      Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;
```

```
component ASUnit
Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
       B : in STD_LOGIC_VECTOR (3 downto 0);
       Selector : in STD_LOGIC;
       S : out STD_LOGIC_VECTOR (3 downto 0);
       Zero : out STD_LOGIC;
       Overflow : out STD_LOGIC);
end component;

component InstructionDecoder
    Port ( InstructionBus : in STD_LOGIC_VECTOR (11 downto 0);
       RegJumpCheck : in STD_LOGIC_VECTOR (3 downto 0);
       RegisterEnable : out STD_LOGIC_VECTOR (2 downto 0);
       LoadSelector : out STD_LOGIC;
       ImmediateValue : out STD_LOGIC_VECTOR (3 downto 0);
       RegisterSelector_A : out STD_LOGIC_VECTOR (2 downto 0);
       RegisterSelector_B : out STD_LOGIC_VECTOR (2 downto 0);
       AddSubSelector : out STD_LOGIC;
       JumpFlag : out STD_LOGIC;
       JumpAddress : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component ProgramCounter
Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
       Res : in STD_LOGIC;
       Clk : in STD_LOGIC;
       Q : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component ROM
   Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
        data : out STD_LOGIC_VECTOR (11 downto 0));
end component;

component RegBank
    Port (
RegBank_EN : in STD_LOGIC_VECTOR(2 downto 0); -- Register enable which selects the
register
RegBank_Clk : in STD_LOGIC;
```

```
RegBank_Res : in STD_LOGIC ;
RegBank_D : in STD_LOGIC_VECTOR(3 downto 0); -- input of register bank

-- output of register bank
RegBank_Q_0 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 0
RegBank_Q_1 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 1
RegBank_Q_2 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 2
RegBank_Q_3 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 3
RegBank_Q_4 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 4
RegBank_Q_5 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 5
RegBank_Q_6 : out STD_LOGIC_VECTOR(3 downto 0); -- output of register 6
RegBank_Q_7 : out STD_LOGIC_VECTOR(3 downto 0)  -- output of register 7
 );
end component;

signal Out_Mux_2_to_1_bit_3 : STD_LOGIC_VECTOR(2 downto 0) := "000";

signal MemorySelect : STD_LOGIC_VECTOR(2 downto 0) := "000";

signal Out_Adder_3bit : STD_LOGIC_VECTOR(2 downto 0) := "000";

signal JumpFlag : STD_LOGIC := '0';
signal JumpAddress : STD_LOGIC_VECTOR(2 downto 0) := "000";
signal InstructionBus : STD_LOGIC_VECTOR(11 downto 0) := "000000000000";
signal RegisterEnable : STD_LOGIC_VECTOR(2 downto 0) := "000";
signal LoadSelector : STD_LOGIC := '0';
signal ImmediateValue : STD_LOGIC_VECTOR(3 downto 0) := "0000" ;
signal RegisterSelector_A : STD_LOGIC_VECTOR(2 downto 0) := "000" ;
signal RegisterSelector_B : STD_LOGIC_VECTOR(2 downto 0) := "000";
signal AddSubSelector : STD_LOGIC := '0' ;
signal RegJumpCheck : STD_LOGIC_VECTOR(3 downto 0) := "0000" ;

signal RegBank_Q_0 : STD_LOGIC_VECTOR(3 downto 0) := "0000"; -- output of register 0
signal RegBank_Q_1 : STD_LOGIC_VECTOR(3 downto 0) := "0000"; -- output of register 1
signal RegBank_Q_2 : STD_LOGIC_VECTOR(3 downto 0) := "0000"; -- output of register 2
signal RegBank_Q_3 : STD_LOGIC_VECTOR(3 downto 0) := "0000"; -- output of register 3
signal RegBank_Q_4 : STD_LOGIC_VECTOR(3 downto 0) := "0000"; -- output of register 4
signal RegBank_Q_5 : STD_LOGIC_VECTOR(3 downto 0) := "0000"; -- output of register 5
signal RegBank_Q_6 : STD_LOGIC_VECTOR(3 downto 0) := "0000"; -- output of register 6
signal RegBank_Q_7 : STD_LOGIC_VECTOR(3 downto 0) := "0000";  -- output of register 7
```

```
signal Out_Mux_B   : STD_LOGIC_VECTOR(3 downto 0):= "0000";

signal RegBank_D : STD_LOGIC_VECTOR(3 downto 0) := "0000";

signal Out_ASUnit : STD_LOGIC_VECTOR(3 downto 0) := "0000";

begin

-- 2-way 3-bit Mux
Mux_2_to_1_bit_3_0 : Mux_2_to_1_bit_3
port map (
  S => JumpFlag,
  D0=> Out_Adder_3bit, -- JumpFlag = 0
  D1=> JumpAddress, -- JumpFlag = 1
  Y => Out_Mux_2_to_1_bit_3
);

Adder_3bit_0 : Adder_3bit
port map(
   A => MemorySelect,
   S => Out_Adder_3bit
);

ProgramCounter_0 : ProgramCounter
port map(
   D => Out_Mux_2_to_1_bit_3,
   Res => Res,
   Clk => Clk,
   Q => MemorySelect

);

ROM_0 : ROM
port map(
   address => MemorySelect,
   data => InstructionBus

);
```

```
InstructionDecoder_0 : InstructionDecoder
port map(
   InstructionBus => InstructionBus,
   RegJumpCheck => RegJumpCheck,
   RegisterEnable => RegisterEnable,
   LoadSelector => LoadSelector,
   ImmediateValue => ImmediateValue,
   RegisterSelector_A => RegisterSelector_A,
   RegisterSelector_B => RegisterSelector_B,
   AddSubSelector => AddSubSelector,
   JumpFlag => JumpFlag,
   JumpAddress => JumpAddress

);

 RegBank_0 : RegBank
    port map(

    RegBank_EN => RegisterEnable, -- Register enable which selects the register
    RegBank_Clk => Clk,
    RegBank_Res => Res,
    RegBank_D => RegBank_D , -- input of register bank

    -- output of register bank
    RegBank_Q_0 => RegBank_Q_0, -- output of register 0
    RegBank_Q_1 => RegBank_Q_1, -- output of register 1
    RegBank_Q_2 => RegBank_Q_2, -- output of register 2
    RegBank_Q_3 => RegBank_Q_3, -- output of register 3
    RegBank_Q_4 => RegBank_Q_4, -- output of register 4
    RegBank_Q_5 => RegBank_Q_5, -- output of register 5
    RegBank_Q_6 => RegBank_Q_6, -- output of register 6
    RegBank_Q_7 => RegBank_Q_7  -- output of register 7
    );


Mux_8_to_1_bit_4_A : Mux_8_to_1_bit_4
   port map(
      S => RegisterSelector_A,
```

```
        D0=> RegBank_Q_0,
        D1=> RegBank_Q_1,
        D2=> RegBank_Q_2,
        D3=> RegBank_Q_3,
        D4=> RegBank_Q_4,
        D5=> RegBank_Q_5,
        D6=> RegBank_Q_6,
        D7=> RegBank_Q_7,
        Y => RegJumpCheck
    );


Mux_8_to_1_bit_4_B : Mux_8_to_1_bit_4
        port map(
            S => RegisterSelector_B,
            D0=> RegBank_Q_0,
            D1=> RegBank_Q_1,
            D2=> RegBank_Q_2,
            D3=> RegBank_Q_3,
            D4=> RegBank_Q_4,
            D5=> RegBank_Q_5,
            D6=> RegBank_Q_6,
            D7=> RegBank_Q_7,
            Y => Out_Mux_B
        );

ASUnit_0 : ASUnit
port map (
    A => RegJumpCheck,
    B => Out_Mux_B,
    Selector => AddSubSelector,
    S => Out_ASUnit,
    Zero => Zero,
    Overflow => Overflow
);

Mux_2_to_1_bit_4_0 : Mux_2_to_1_bit_4
port map(
    S => LoadSelector,
    D0 => ImmediateValue,
```

```
    D1 => Out_ASUnit,
    Y => RegBank_D
);

Output <= RegBank_Q_7;

end Behavioral;
```

System.vhd

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

entity System is
    Port (
    Clk : in STD_LOGIC;
    Res : in STD_LOGIC;
    Zero: out STD_LOGIC;
    Overflow: out STD_LOGIC;
    S_7Seg : out STD_LOGIC_VECTOR(6 downto 0);
    S_out : out STD_LOGIC_VECTOR(3 downto 0);
    Anode : out STD_LOGIC_VECTOR(3 downto 0)
    );
end System;

architecture Behavioral of System is

component nano_processor
    Port ( Clk : in STD_LOGIC;
       Res : in STD_LOGIC;
       Overflow : out STD_LOGIC;
       Zero : out STD_LOGIC;
       Output : out STD_LOGIC_VECTOR (3 downto 0));

end component;

component LUT_16_7
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
```

```
      data : out STD_LOGIC_VECTOR (6 downto 0));
end component;

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

signal Slow_Clk_out : STD_LOGIC;
signal Reg7out : STD_LOGIC_VECTOR(3 downto 0);

begin

Slow_Clk_0 : Slow_Clk
port map(
Clk_in => Clk,
Clk_out => Slow_Clk_out
);

nano_processor_0 : nano_processor
port map(
Clk => Slow_Clk_out,
Res => Res,
Overflow => Overflow,
Zero => Zero,
Output => Reg7out
);

LUT_16_7_0 : LUT_16_7
port map(
address => Reg7out,
data => S_7Seg
);

S_out <= Reg7out;
Anode <= "1110";
end Behavioral;
```

Slow_Clk.vhd

```
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=30000000) then
            clk_status<= not clk_status;
            Clk_out<=clk_status;
            count<=1;
        end if;
    end if;

    end process;
```

7 segment ROM

LUT_16_7.vhd

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

use ieee.numeric_std.all;

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;
```

Simulations

TB_Slow_Clk.vhd

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

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

architecture Behavioral of TB_Slow_Clk is
component Slow_Clk
```

```
 Port ( Clk_in : in STD_LOGIC;
       Clk_out : out STD_LOGIC);
end component;

signal Clk_in : STD_LOGIC;
signal Clk_out : STD_LOGIC;

constant period : time := 10 ns;

begin
UUT : Slow_Clk PORT MAP(
    Clk_in => Clk_in,
    Clk_out => Clk_out

);

process
begin

Clk_in <= '0';
wait for period/2;

Clk_in <= '1';
wait for period/2;

end process;

end Behavioral;
```

TB_nano_processor.vhd

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

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

architecture Behavioral of TB_nano_processor is
```

```
component nano_processor
Port ( Clk : in STD_LOGIC;
       Res : in STD_LOGIC;
       Overflow : out STD_LOGIC;
       Zero : out STD_LOGIC;
       Output : out STD_LOGIC_VECTOR (3 downto 0));
end component;

 signal Clk : STD_LOGIC;
 signal Res : STD_LOGIC;
 signal Overflow : STD_LOGIC;
 signal Zero : STD_LOGIC;
 signal Output : STD_LOGIC_VECTOR (3 downto 0);

constant period : time := 100 ns;

begin
UUT : nano_processor PORT MAP (
       Clk => Clk,
       Res => Res,
       Overflow => Overflow,);

process
begin
Clk <= '0';
wait for period/2;

Clk <= '1';
wait for period/2;
end process;

process
begin
Res <= '1';
wait for period;

Res <= '0';
wait;
end process;
```

```
end Behavioral;
```

TB_System.vhd

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

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

architecture Behavioral of TB_System is

component System
    Port (
Clk : in STD_LOGIC;
Res : in STD_LOGIC;
Zero: out STD_LOGIC;
Overflow: out STD_LOGIC;
S_7Seg : out STD_LOGIC_VECTOR(6 downto 0);
S_out : out STD_LOGIC_VECTOR(3 downto 0)
);

end component;

signal Clk : STD_LOGIC;
signal Res : STD_LOGIC;
signal Zero: STD_LOGIC;
signal Overflow: STD_LOGIC;
signal S_7Seg : STD_LOGIC_VECTOR(6 downto 0);
signal S_out : STD_LOGIC_VECTOR(3 downto 0);

constant period : time := 10 ns;
begin
UUT : System PORT MAP(

        Clk => Clk,
        Res => Res,
```

```
        Zero => Zero,
        Overflow => Overflow,
        S_7Seg => S_7Seg,
        S_out => S_out
);

process
begin
Clk <= '0';
wait for period/2;

Clk <= '1';
wait for period/2;
end process;

process
begin
Res <= '1';
wait for period;

Res <= '0';
wait;
end process;

end Behavioral;
```

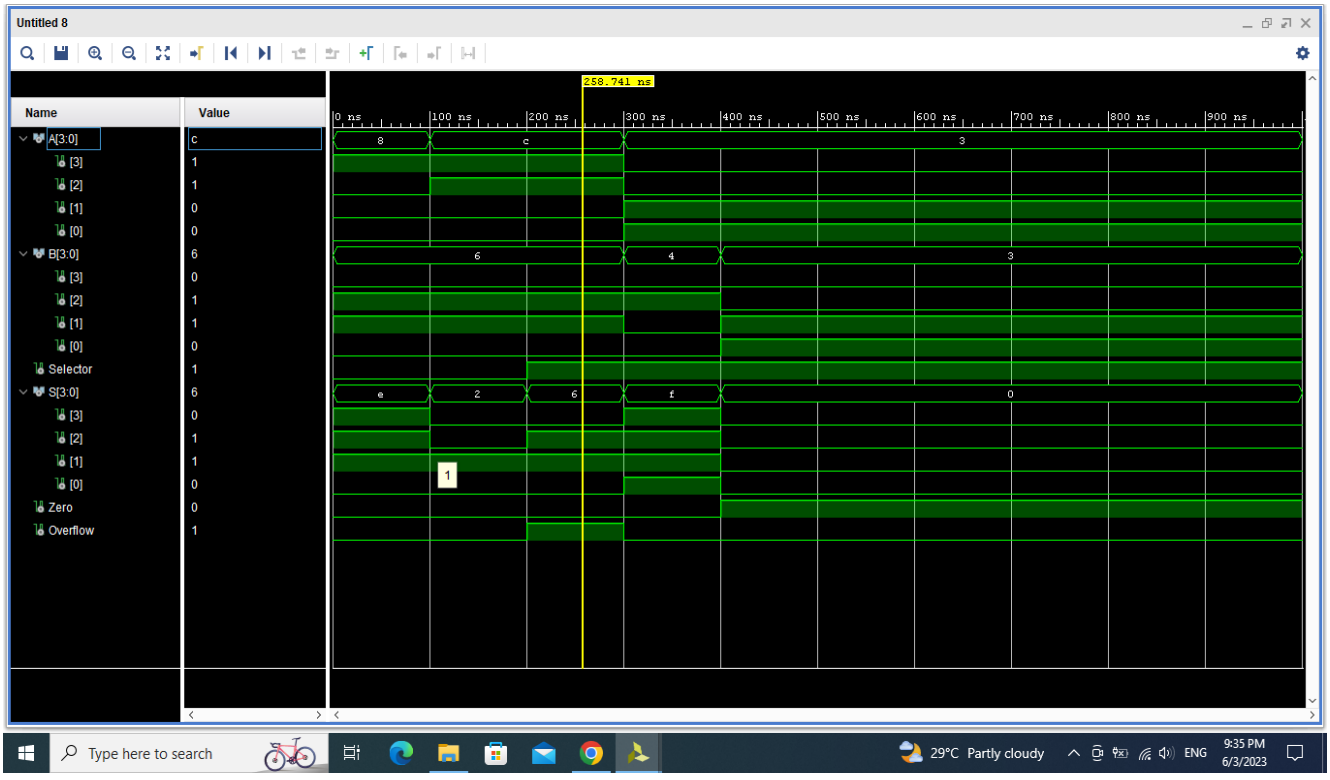
Constraint File

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports Clk]
	set_property IOSTANDARD LVCMOS33 [get_ports Clk]
	create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports Clk]
## LEDs
set_property PACKAGE_PIN U16 [get_ports {S_out[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_out[0]}]
set_property PACKAGE_PIN E19 [get_ports {S_out[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_out[1]}]
set_property PACKAGE_PIN U19 [get_ports {S_out[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_out[2]}]
set_property PACKAGE_PIN V19 [get_ports {S_out[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_out[3]}]
set_property PACKAGE_PIN P1 [get_ports {Zero}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Zero}]
set_property PACKAGE_PIN L1 [get_ports {Overflow}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Overflow}]
##7 segment display
set_property PACKAGE_PIN W7 [get_ports {S_7Seg[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_7Seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {S_7Seg[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_7Seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {S_7Seg[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_7Seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {S_7Seg[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_7Seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {S_7Seg[4]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_7Seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {S_7Seg[5]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_7Seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {S_7Seg[6]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {S_7Seg[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]}]
```

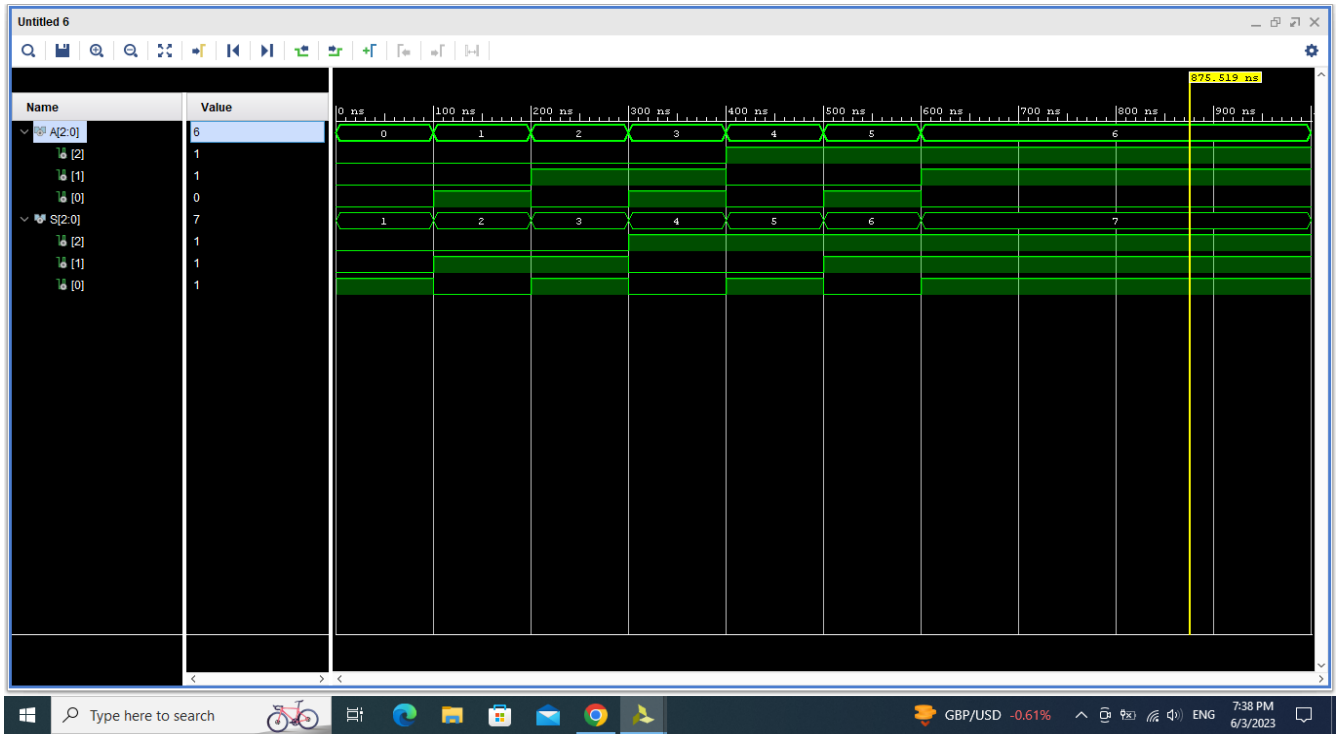
```
set_property PACKAGE_PIN W4 [get_ports {Anode[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {Anode[3]}]

##Buttons
set_property PACKAGE_PIN U18 [get_ports {Res}]
        set_property IOSTANDARD LVCMOS33 [get_ports {Res}]
```

4) All Timing Diagrams

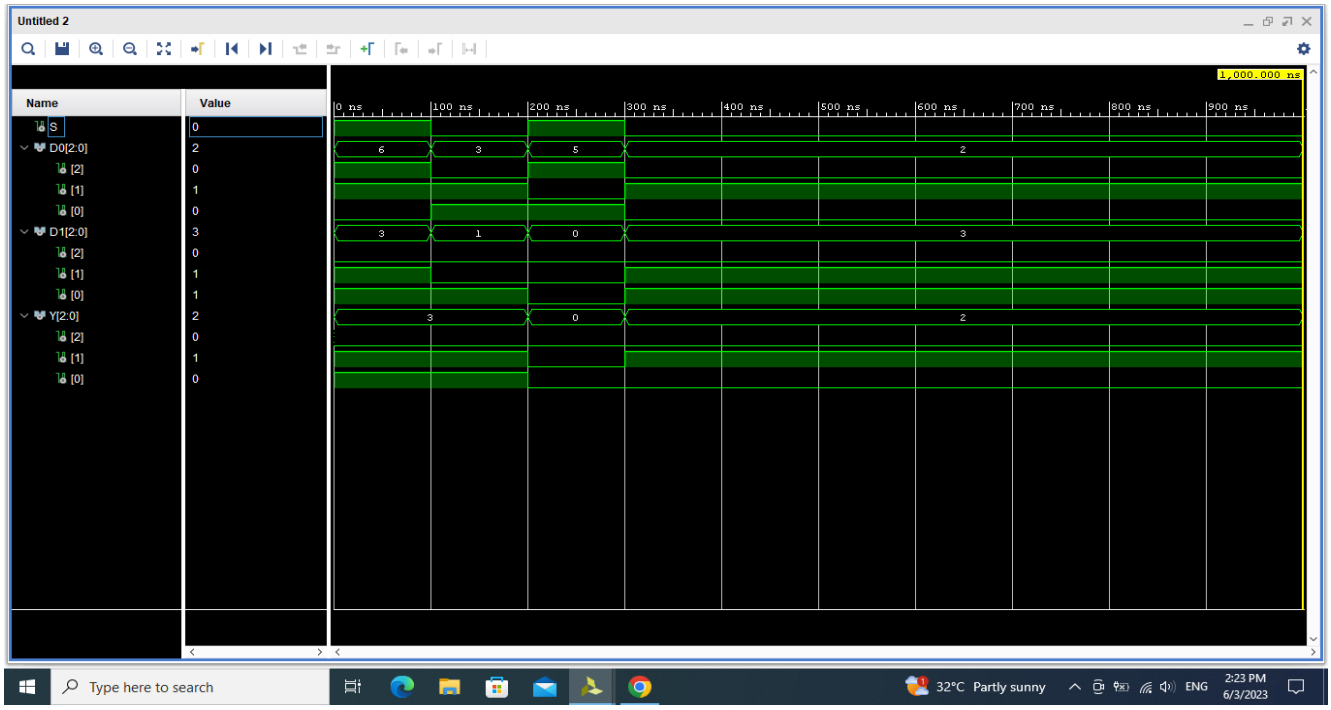
1) ASUnit



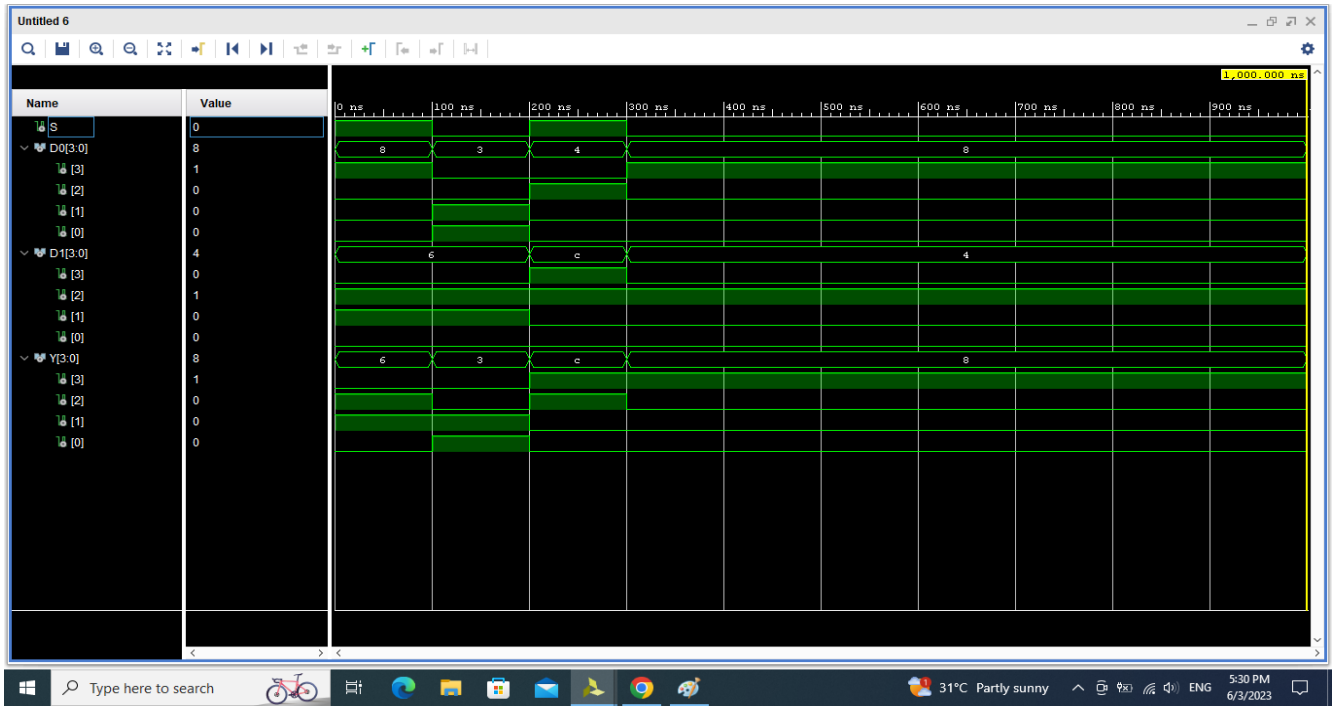
2) 3-bit adder



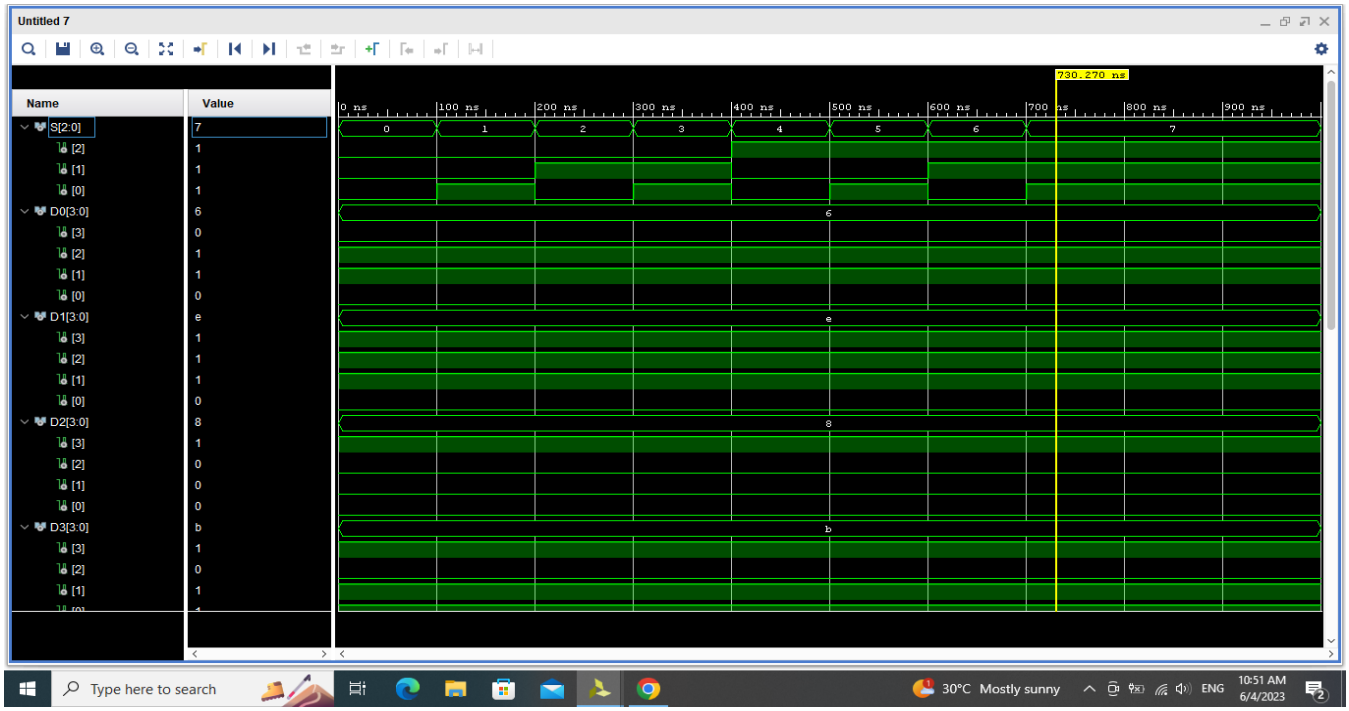
3) 2-way 3-bit MUX

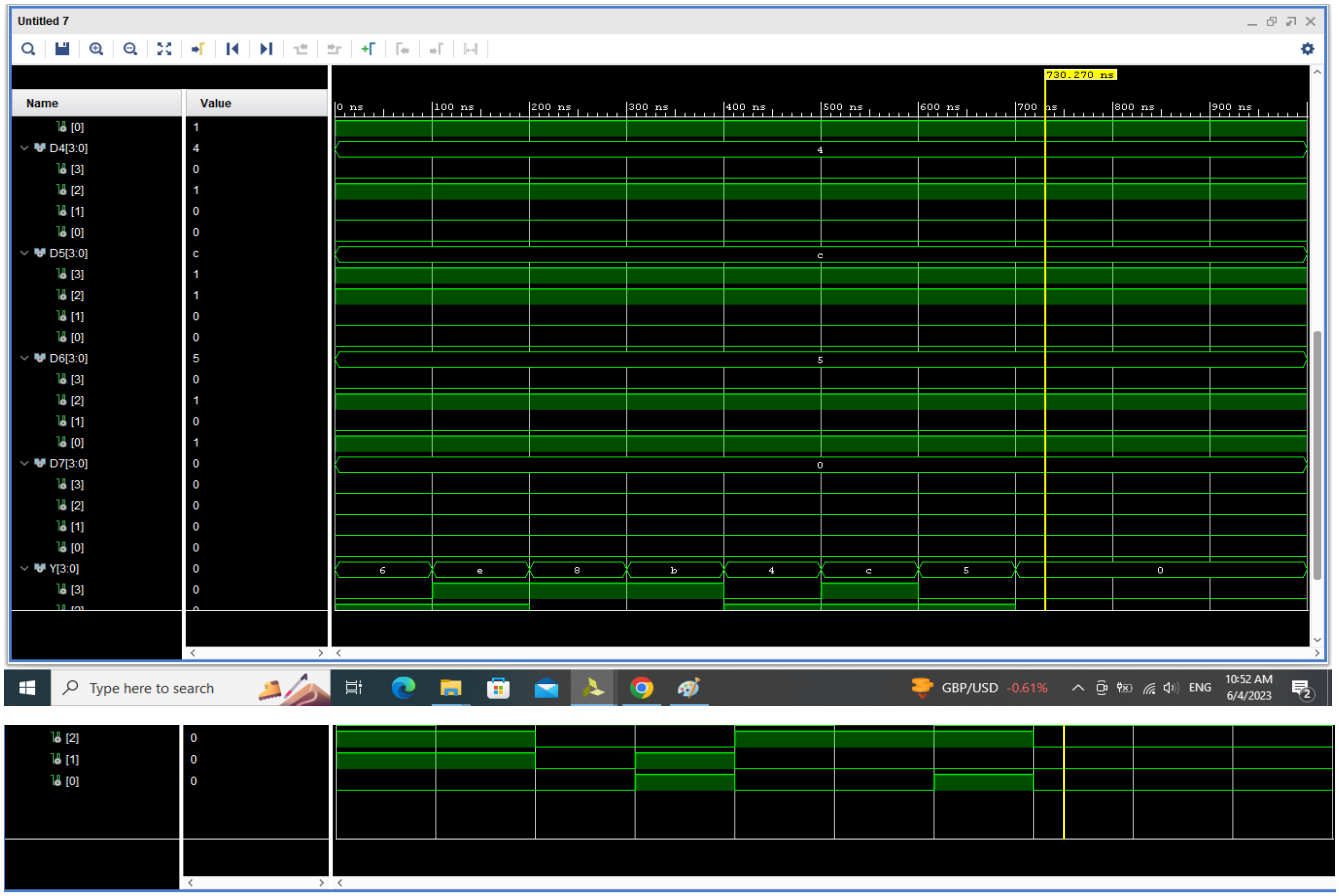


4) 2-way 4-bit MUX

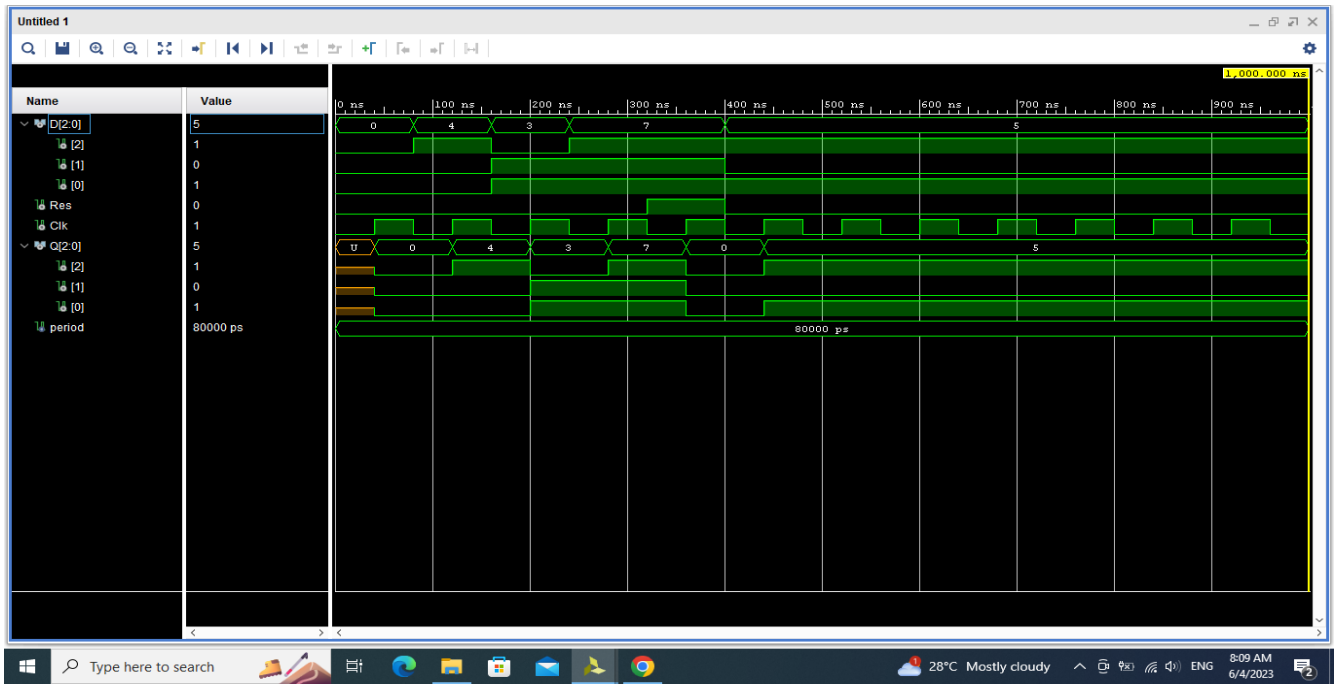


5) 8-way 4-bit MUX

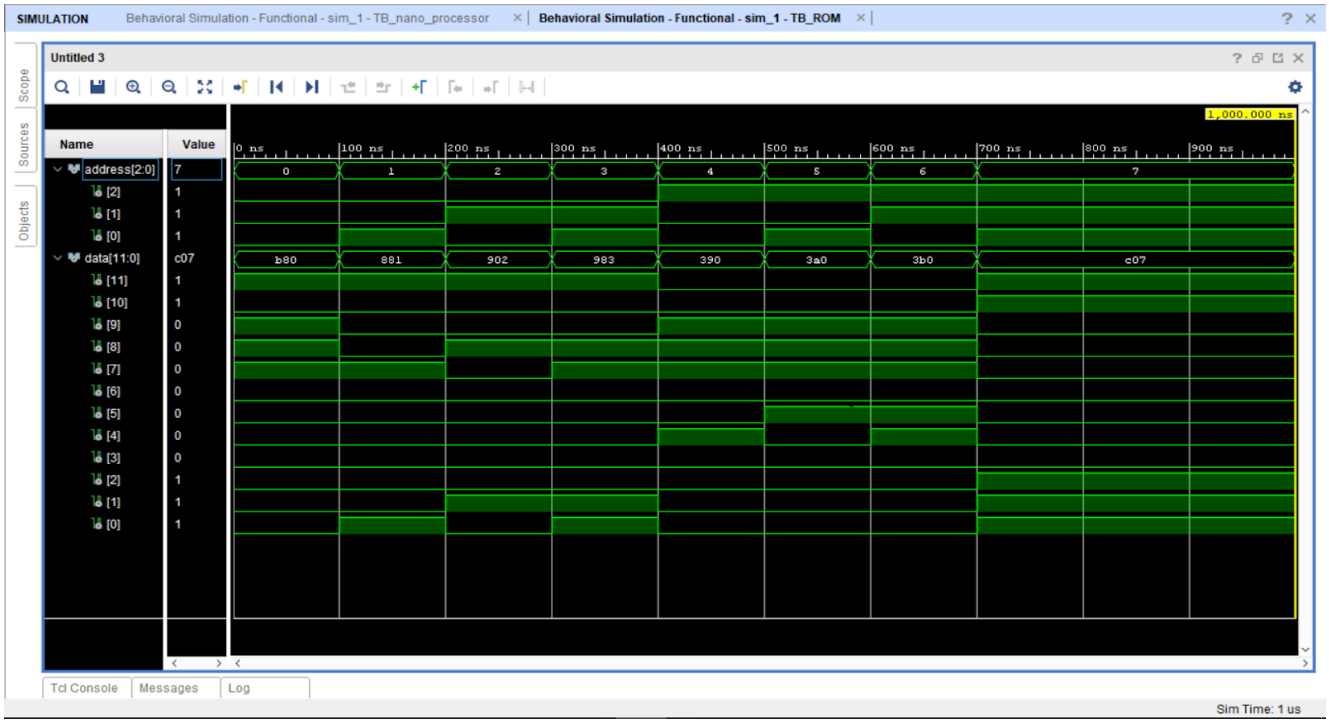




6) Program Counter

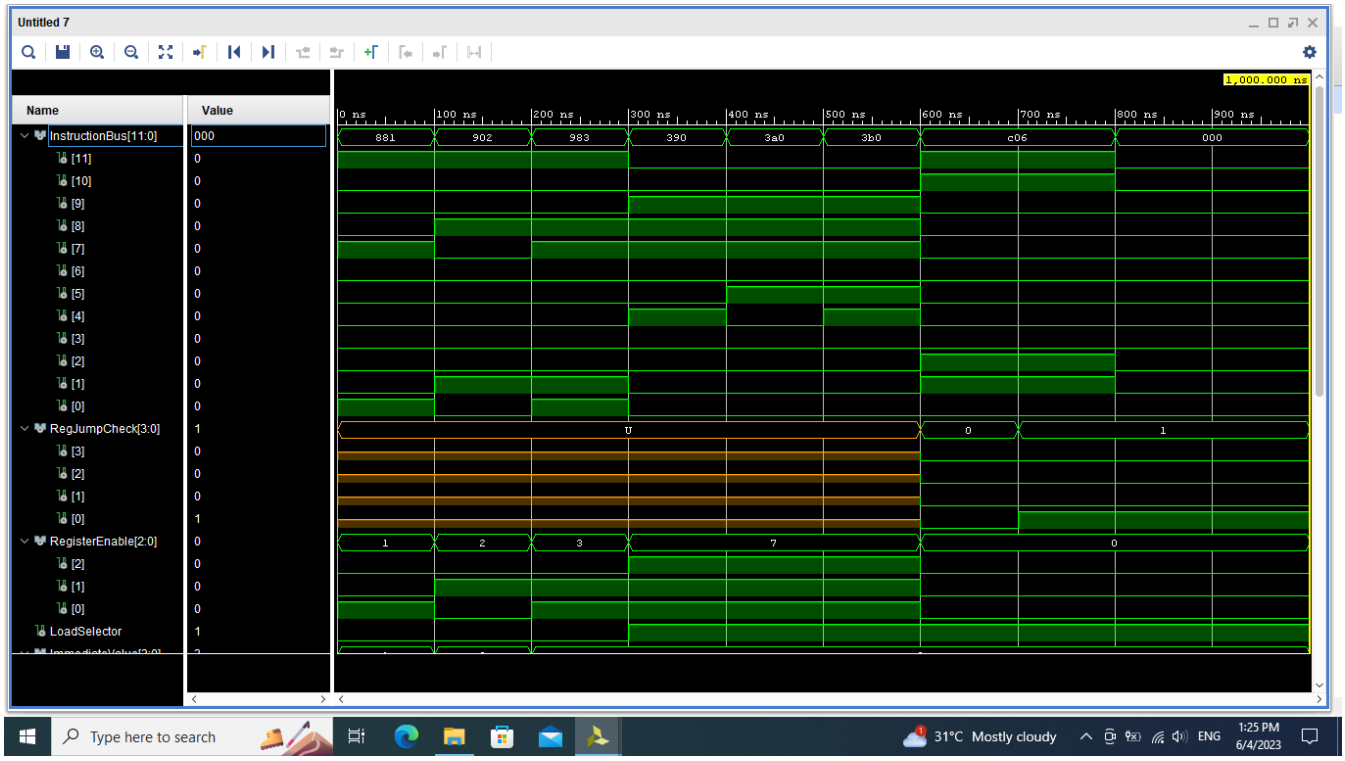


7) ROM

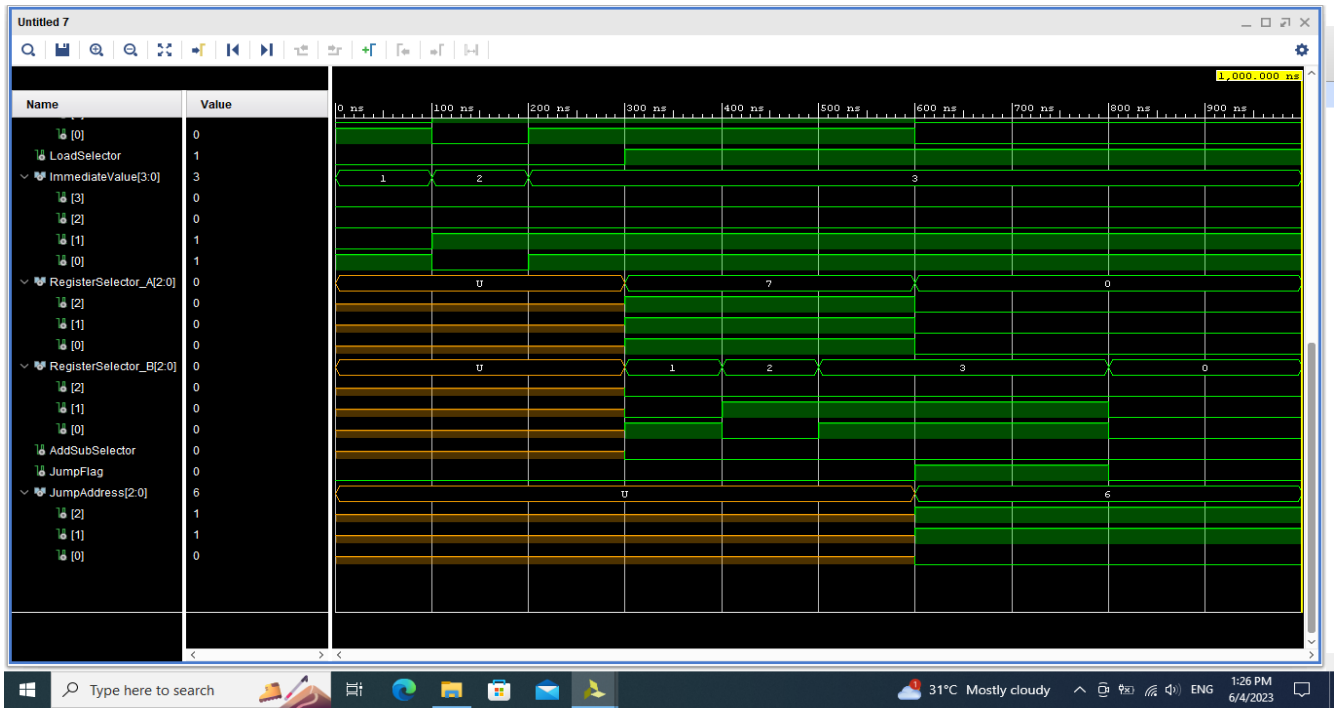


8) Instruction Decoder

<u>Part-1</u>

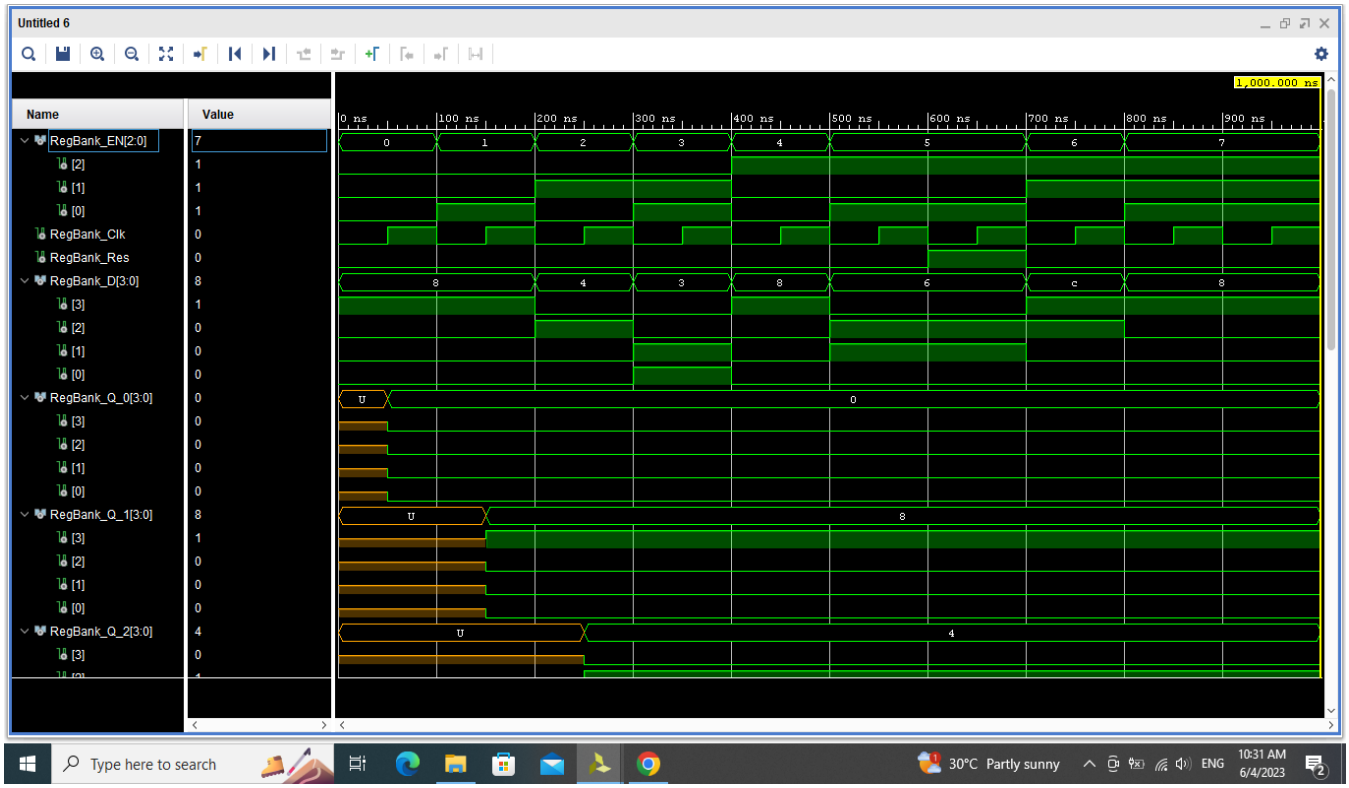


Part-2

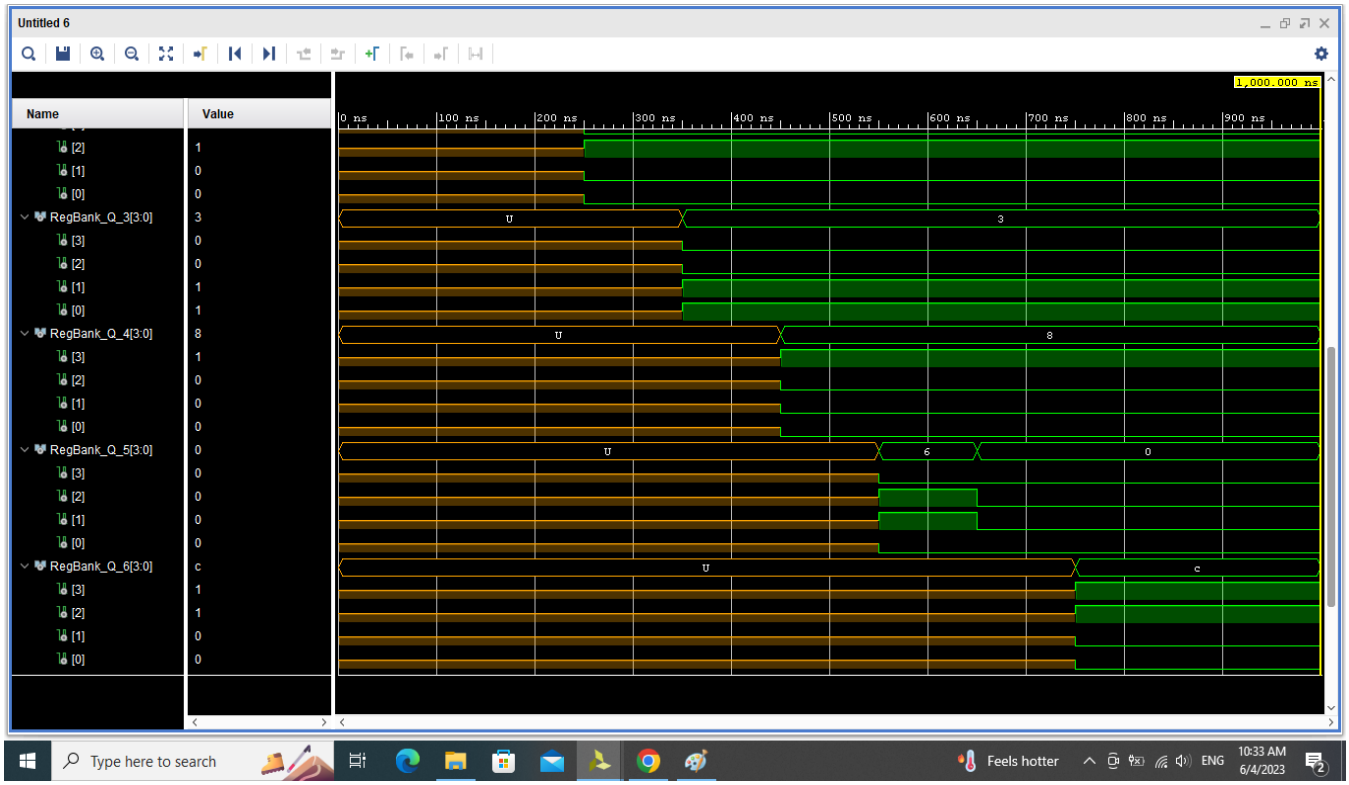


9) Register Bank

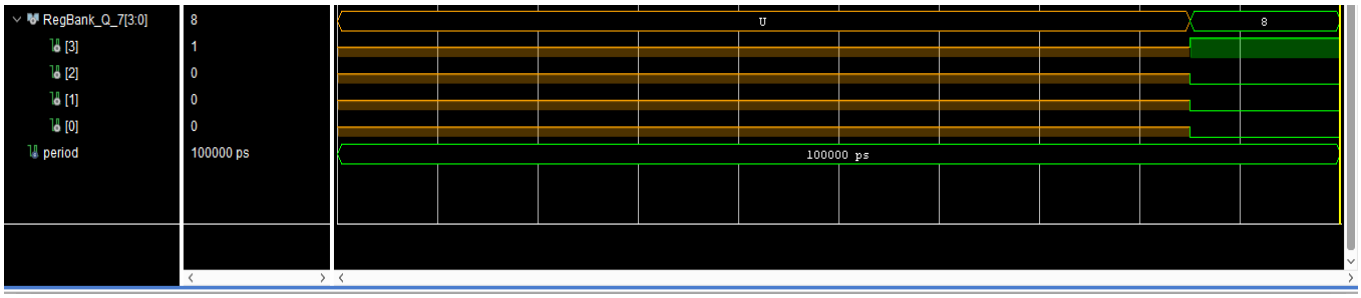
Part-1


Untitled 6
Name
Value
RegBank_EN[2:0]
RegBank_Clk
RegBank_Res
RegBank_D[3:0]
RegBank_Q_0[3:0]
RegBank_Q_1[3:0]
RegBank_Q_2[3:0]
1,000.000 ns
Type here to search
30°C Partly sunny
ENG
10:31 AM
6/4/2023

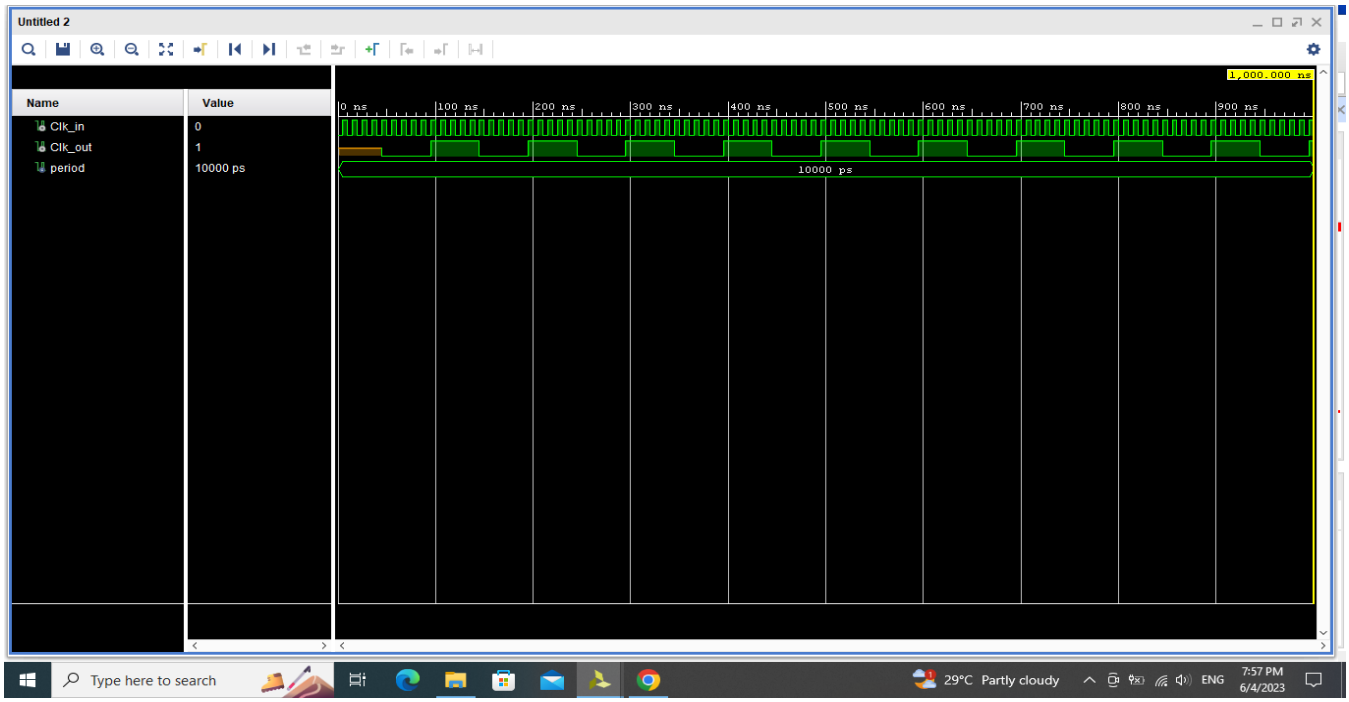
Part-2


Untitled 6
Name
Value
RegBank_Q_3[3:0]
RegBank_Q_4[3:0]
RegBank_Q_5[3:0]
RegBank_Q_6[3:0]
1,000.000 ns
Type here to search
Feels hotter
ENG
10:33 AM
6/4/2023

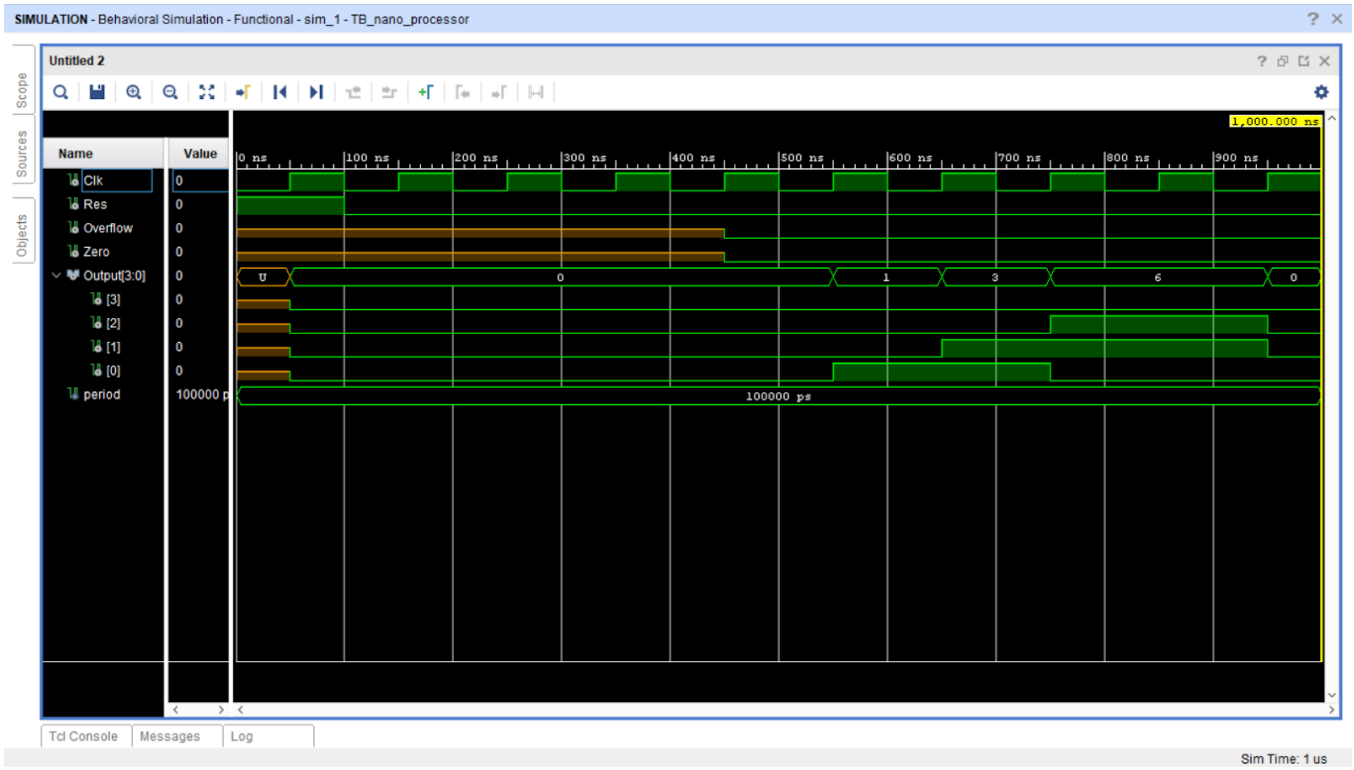
Part-3



10) Slow Clock



11) Nano Processor



Conclusion

- In this lab we were able to verify the accuracy of our design by simulating and testing every component.
- Instead of using many parallel wires we used buses to design the circuit.
- This simple nano processor design could be extended to build a complex processor which could be able to do complex operations.
- Even though this processor satisfied our work we cannot work on larger problems since the processor is 4-bit wide.
- In this exercise we had to hard code assembly instructions as binary values because the microprocessor only understands machine language.
- Finally this project helped us to build our team work ability and instructors guided us to achieve this task.
- We add a program to calculate 1+2+3 in to the ROM. We tested our processor in other different programs to check whether other instructions such as NEG are working well.

Contribution by each member

W.T Rathnayaka: 210536K - 30 hours

Wrote source code for each component. Wrote constraint file. Assembling different components to make **nano_processor.vhd**

Dewpura L.C.I.K: 210116A - 20 hours

Wrote source code for **System.vhd** assembling slow clock, 7 seven segment ROM and nano processor. Wrote test bench files for each component.

GitHub Link for all source codes

https://github.com/Ishdew/Nano-Processor