Lab 9-10 – Nano processor Report

- **Student names and Index numbers of Group 37**
    - Thanushanth K - 200636H
    - Lithurshan K - 200343G
    - Nithursika K - 200435P
    - Sabthana J - 200276D
    - Aathavan N - 200001H

- **Assigned Task**

Assigned task for this lab is to design a 4-bit Nano processor. First we have to develop a 4-bit arithmetic unit which can perform add and subtraction operations of signed integers. Then we have to build components of Nano processor such as a 3-bit adder, 3-bit Program Counter and k-way b-bit multiplexers. We also have to build a register bank contain 8 of 4-bit registers and a program ROM to store our hard coded Assembly program. Then we have to build 3,4 and 12-bit buses to connect the components. Instruction Decoder has to be designed in order to decode the instructions from the ROM and to activate the necessary components based on the instructions we execute.

As a group, we have to build these components of the Nano processor and have to simulate and verify each of their functionalities. Finally, we have to combine all these components and make a Nano processor which is capable of executing specific instructions saved in our ROM. After that we have to write an assembly code which will add from 1 to 3 and have to change it to the machine code. We have to hard code it to the ROM and have to simulate our Nano processor to show the output in a 7-segment display.

- **Machine Code and Assembly program**

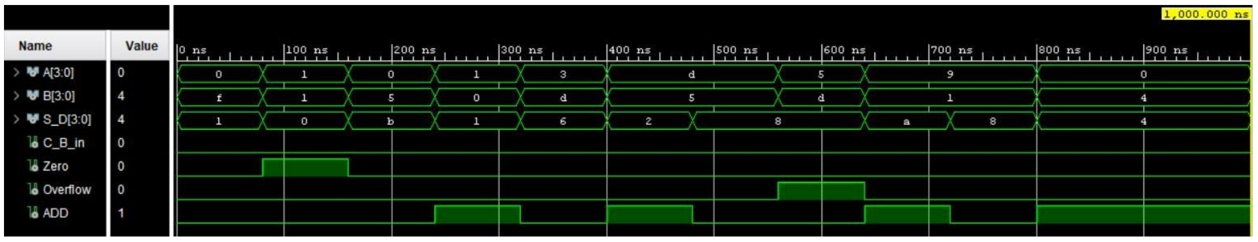
```
"100100000011", -- MOV R2,3

"100110000001", -- MOV R3,1

"010110000000", --NEG R3

"110100000111", --JZR R2,7

"001110100000", --ADD R7,R2  => 3,5,6

"000100110000", --ADD R2,R3

"110000000011", --JZR R0,3

"110000000111" --JZR R0,
```

- **Components of Nano processor**
  1. 4-bit Add/Subtract Unit
     - ❖VHDL Source

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

entity ADD_SUB is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           ADD : in STD_LOGIC;
           C_B_in : in STD_LOGIC :='0';
           S_D : out STD_LOGIC_VECTOR (3 downto 0);
 --          C_B_out : out STD_LOGIC;
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC);
end ADD_SUB;

architecture Behavioral of ADD_SUB is
component RCA
    Port (
        X : in STD_LOGIC_VECTOR (3 downto 0);
        Y : in STD_LOGIC_VECTOR (3 downto 0);
        C_in : in STD_LOGIC;
        S : out STD_LOGIC_VECTOR (3 downto 0);
        C_out : out STD_LOGIC);
end component;

component RCS
    Port (
        X : in STD_LOGIC_VECTOR (3 downto 0);
        Y : in STD_LOGIC_VECTOR (3 downto 0);
        B_in : in STD_LOGIC;
        D : out STD_LOGIC_VECTOR (3 downto 0);
        B_out : out STD_LOGIC);
end component;

SIGNAL  c_in, b_in, c_out, b_out: STD_LOGIC;
SIGNAL x_a, y_a, x_s, y_s,s, d, output: STD_LOGIC_VECTOR (3 downto 0);

begin
    RCA_0 : RCA
    port map (
         X => A,
         Y => B,
         C_in => C_B_in,
         S => s,
         C_out => c_out);

    RCS_0 : RCS
    port map (
        X => A,
        Y => B,
        B_in => C_B_in,
        D => d,
        B_out => b_out);

    output <= s when ADD = '1'  else d ;
    S_D <= output ;
 --   C_B_out <= c_out when ADD = '1'  else b_out;
```

```
    Zero <= NOT(output(0)) AND NOT(output(1)) AND NOT(output(2)) AND
NOT(output(3));

    Overflow <= (A(3) XNOR (NOT(ADD) XOR B(3))) AND (A(3) XOR output(3));

end Behavioral;
```

❖Timing diagram

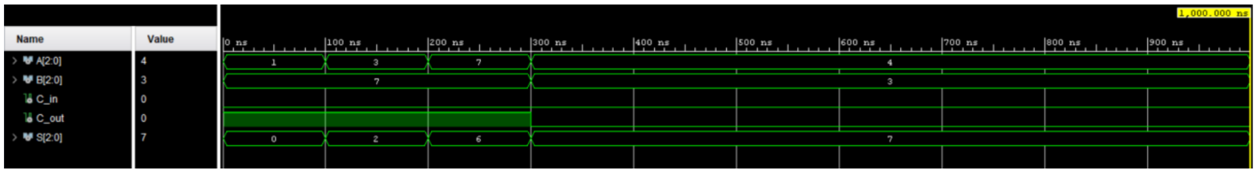


2. 3-bit Adder

❖Design Source

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

entity RCA_3 is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           B : in STD_LOGIC_VECTOR (2 downto 0);
           C_in : in STD_LOGIC;
           C_out : out STD_LOGIC;
           S : out STD_LOGIC_VECTOR (2 downto 0));
end RCA_3;

architecture Behavioral of RCA_3 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: std_logic;
begin

FA_0 : FA
    port map (
        A => A(0),
        B => B(0),
        C_in =>C_in,
        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 => C_out);
end Behavioral;
```

❖Timing diagram

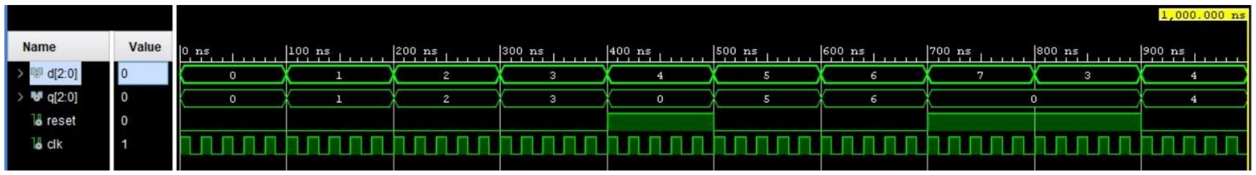


3. 3-bit Program Counter (PC)
    ❖VHDL Source

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

entity P_Counter is
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (2 downto 0):="000");
end P_Counter;
```

```
architecture Behavioral of P_Counter is

begin

    process (Clk,Reset) begin
    if Reset = '1' then   -- If register is reset PC valuse is 0
        if (rising_edge(Clk)) then
            Q <= "000" ;
        end if;
    else
        if (falling_edge(Clk)) then -- respond when clock rises
                Q <= D;
           end if;

        end if;

    end process;

end Behavioral;
```

❖Timing Diagram



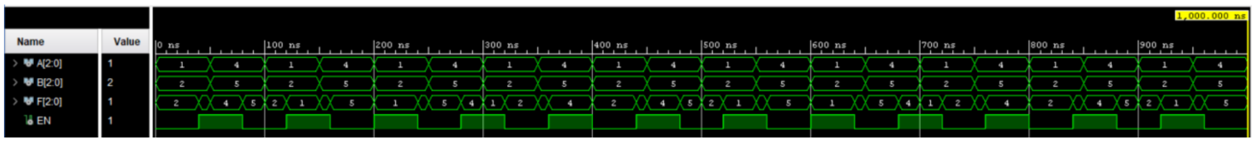
4. Multiplexers
   i. 2-way 3-bit multiplexer

❖Design Source

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

entity MUX2Way3Bit is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           B : in STD_LOGIC_VECTOR (2 downto 0);
           F : out STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC);
end MUX2Way3Bit;
```

```
architecture Behavioral of MUX2Way3Bit is

begin

F <= A when (EN = '1') else B;

end Behavioral;
```

❖Timing Diagram

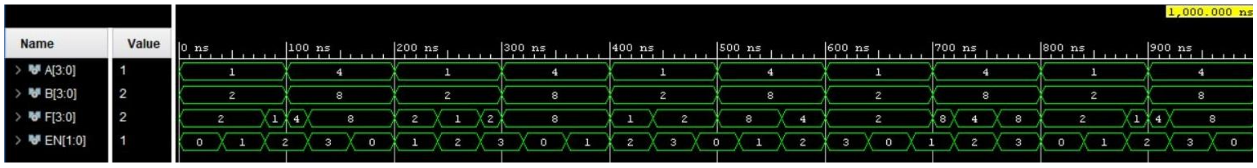


ii. 2-way 4-bit multiplexer

❖Design Source

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

entity MUX2Way4Bit is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          F : out STD_LOGIC_VECTOR (3 downto 0);
          EN : in STD_LOGIC_VECTOR (1 downto 0));
end MUX2Way4Bit;

architecture Behavioral of MUX2Way4Bit is
begin

F <= A when (EN = "10") else B;

end Behavioral;
```

❖Timing Diagram



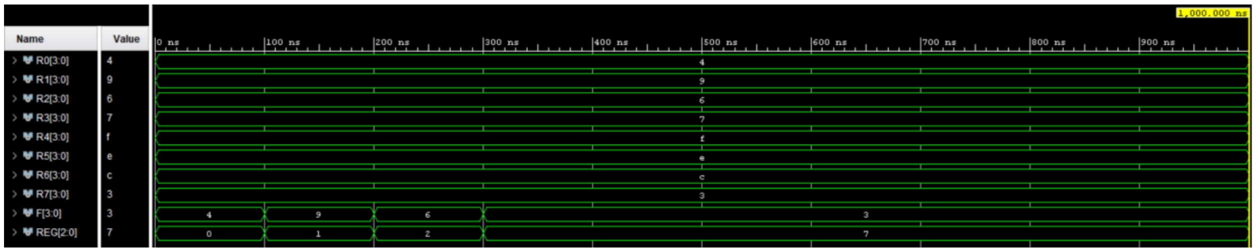
iii. 8-way 4-bit multiplexer

❖Design Source

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

entity MUX8Way4Bit is
    Port ( R0 : in STD_LOGIC_VECTOR (3 downto 0);
           R1 : in STD_LOGIC_VECTOR (3 downto 0);
           R2 : in STD_LOGIC_VECTOR (3 downto 0);
           R3 : in STD_LOGIC_VECTOR (3 downto 0);
           R4 : in STD_LOGIC_VECTOR (3 downto 0);
           R5 : in STD_LOGIC_VECTOR (3 downto 0);
           R6 : in STD_LOGIC_VECTOR (3 downto 0);
           R7 : in STD_LOGIC_VECTOR (3 downto 0);
           REG : in STD_LOGIC_VECTOR (2 downto 0);
           F : out STD_LOGIC_VECTOR (3 downto 0));
end MUX8Way4Bit;

architecture Behavioral of MUX8Way4Bit is

begin

process (REG)
begin

    if (REG = "000") then
        F <= R0;
    end if;
    if (REG = "001") then
        F <= R1;
    end if;
    if (REG = "010") then
        F <= R2;
    end if;
    if (REG = "011") then
        F <= R3;
    end if;
    if (REG = "100") then
        F <= R4;
    end if;
    if (REG = "101") then
        F <= R5;
    end if;
    if (REG = "110") then
        F <= R6;
    end if;
    if (REG = "111") then
```

```
        F <= R7;
    end if;

end process;

end Behavioral;
```

❖ Timing Diagram



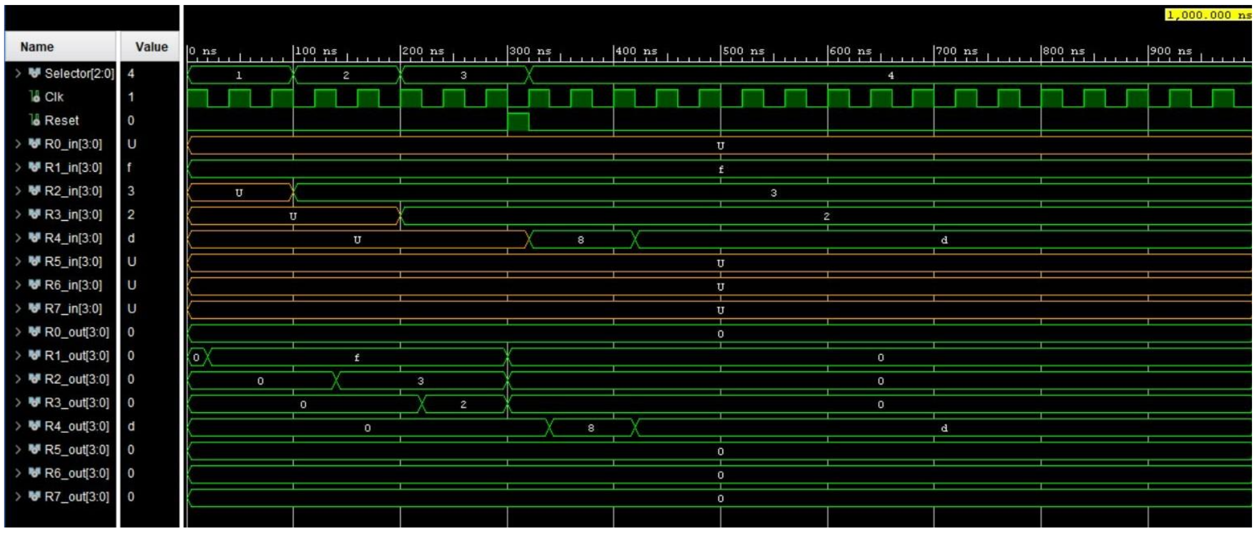
5. Register Bank

❖Design Source

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

entity RegisterBank is
    Port ( Selector : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           R0_in : in STD_LOGIC_VECTOR (3 downto 0);
           R1_in : in STD_LOGIC_VECTOR (3 downto 0);
           R2_in : in STD_LOGIC_VECTOR (3 downto 0);
           R3_in : in STD_LOGIC_VECTOR (3 downto 0);
           R4_in : in STD_LOGIC_VECTOR (3 downto 0);
           R5_in : in STD_LOGIC_VECTOR (3 downto 0);
           R6_in : in STD_LOGIC_VECTOR (3 downto 0);
           R7_in : in STD_LOGIC_VECTOR (3 downto 0);
           R0_out : out STD_LOGIC_VECTOR (3 downto 0);
           R1_out : out STD_LOGIC_VECTOR (3 downto 0);
           R2_out : out STD_LOGIC_VECTOR (3 downto 0);
           R3_out : out STD_LOGIC_VECTOR (3 downto 0);
           R4_out : out STD_LOGIC_VECTOR (3 downto 0);
           R5_out : out STD_LOGIC_VECTOR (3 downto 0);
           R6_out : out STD_LOGIC_VECTOR (3 downto 0);
           R7_out : out STD_LOGIC_VECTOR (3 downto 0));
end RegisterBank;

architecture Behavioral of RegisterBank is
component Reg
    port(
        D : in STD_LOGIC_VECTOR (3 downto 0);
       En : in STD_LOGIC;
       Reset : 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);
        Y : OUT STD_LOGIC_VECTOR (7 downto 0)
    );
end component;
SIGNAL P : STD_LOGIC_VECTOR (7 downto 0);
begin
Decoder_3_to_8_0: Decoder_3_to_8
    port map(
        I=>Selector,
        y =>P
    );


Reg_0: Reg
    port map(
        D=>R0_in,
        En=>P(0),
        Reset => Reset,
        Clk=>Clk,
        Q=>R0_out
    );

Reg_1: Reg
    port map(
        D=>R1_in,
        En=>P(1),
        Reset => Reset,
        Clk=>Clk,
        Q=>R1_out
    );

Reg_2: Reg
    port map(
        D=>R2_in,
        En=>P(2),
        Reset => Reset,
        Clk=>Clk,
        Q=>R2_out
        );

Reg_3: Reg
    port map(
        D=>R3_in,
        En=>P(3),
        Reset => Reset,
        Clk=>Clk,
        Q=>R3_out
    );

Reg_4: Reg
    port map(
        D=>R4_in,
        En=>P(4),
        Reset => Reset,
        Clk=>Clk,
        Q=>R4_out
    );

Reg_5: Reg
    port map(
        D=>R5_in,
        En=>P(5),
        Reset => Reset,
        Clk=>Clk,
        Q=>R5_out
        );

Reg_6: Reg
```

```
    port map(
        D=>R6_in,
        En=>P(6),
        Reset => Reset,
        Clk=>Clk,
        Q=>R6_out
    );

Reg_7: Reg
    port map(
        D=>R7_in,
        En=>P(7),
        Reset => Reset,
        Clk=>Clk,
        Q=>R7_out
    );


end Behavioral;
```

❖Timing Diagram



➢ Register

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

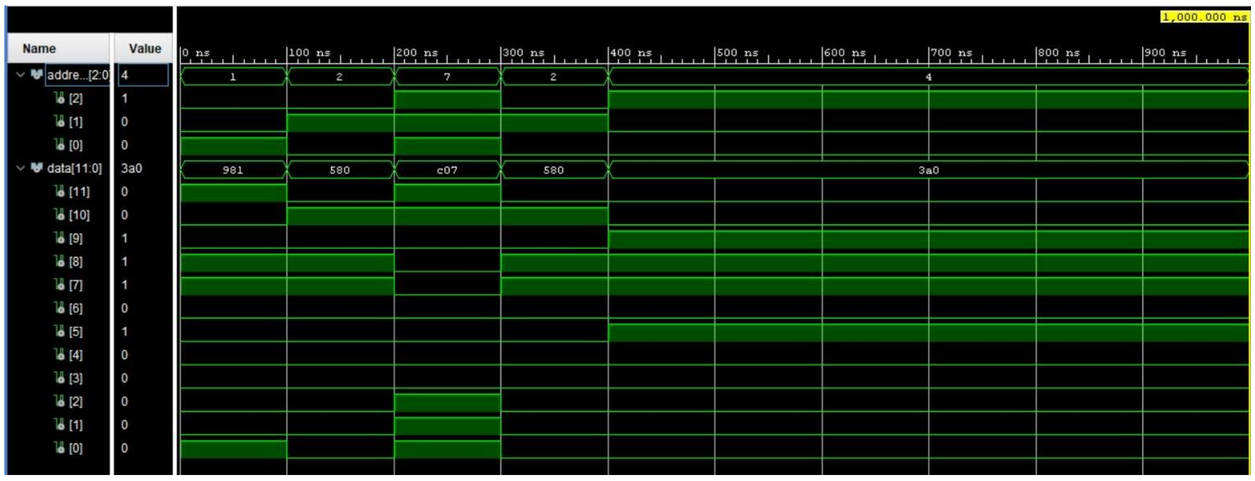
```
entity Reg is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           En : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end Reg;

architecture Behavioral of Reg is
SIGNAL S:STD_LOGIC_VECTOR (3 downto 0):="0000";

begin
process(Clk,Reset)
begin

    if (Reset = '1') then
        if(rising_edge(Clk)) then  --respond when clock rises
            Q <= "0000";
            S <= "0000";
        end if;
    else
        if(rising_edge(Clk)) then  --respond when clock rises
            if En ='1' then      --enable should be set
                Q<=D;
                S<=D;
             ELSE
                Q<=S;
             end if;
         end if;
    end if;
end process;

end Behavioral;
```

6. Program ROM
    ❖ Design Source

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

entity Program_ROM is
```

```
    Port ( PC : in STD_LOGIC_VECTOR (2 downto 0);
           Instruction : out STD_LOGIC_VECTOR (11 downto 0));
end Program_ROM;

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

    "100100000011", -- MOV R2,3
    "100110000001", -- MOV R3,1
    "010110000000", --NEG R3
    "110100000111", --JZR R2,7
    "001110100000", --ADD R7,R2  => 3,5,6
    "000100110000", --ADD R2,R3
    "110000000011", --JZR R0,3
    "110000000111" --JZR R0,7
    );
begin
    Instruction <= P_ROM(to_integer(unsigned(PC)));

end Behavioral;
```

❖Timing Diagram



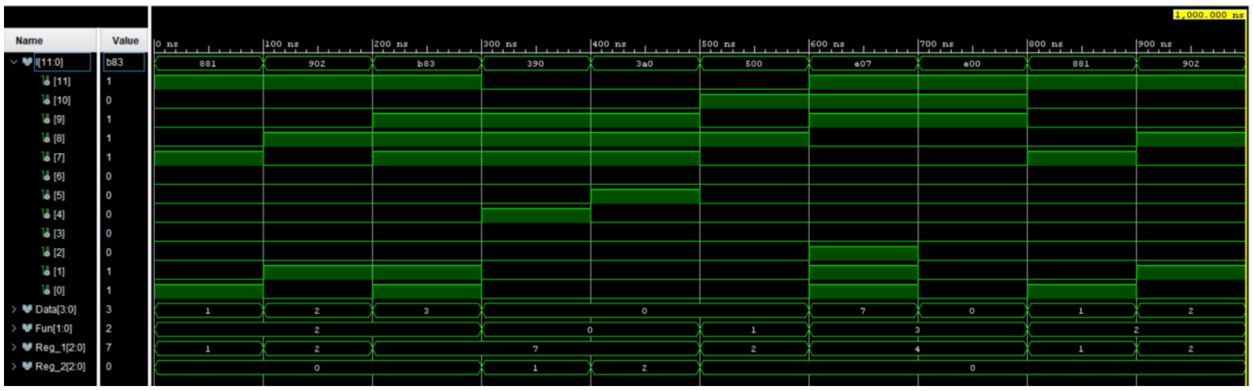
7. Buses

❖Design Source

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

```vhdl
entity BUSES is
    Port ( I : in STD_LOGIC_VECTOR (11 downto 0);
           Data : out STD_LOGIC_VECTOR (3 downto 0);
           Fun : out STD_LOGIC_VECTOR (1 downto 0);
           Reg_1 : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_2 : out STD_LOGIC_VECTOR (2 downto 0));
end BUSES;

architecture Behavioral of BUSES is

begin

    Fun(1) <= I(11);
    Fun(0) <= I(10);

    Reg_1(2) <= I(9);
    Reg_1(1) <= I(8);
    Reg_1(0) <= I(7);

    Reg_2(2) <= I(6);
    Reg_2(1) <= I(5);
    Reg_2(0) <= I(4);

    Data(3) <= I(3);
    Data(2) <= I(2);
    Data(1) <= I(1);
    Data(0) <= I(0);

end Behavioral;
```

❖Timing diagram



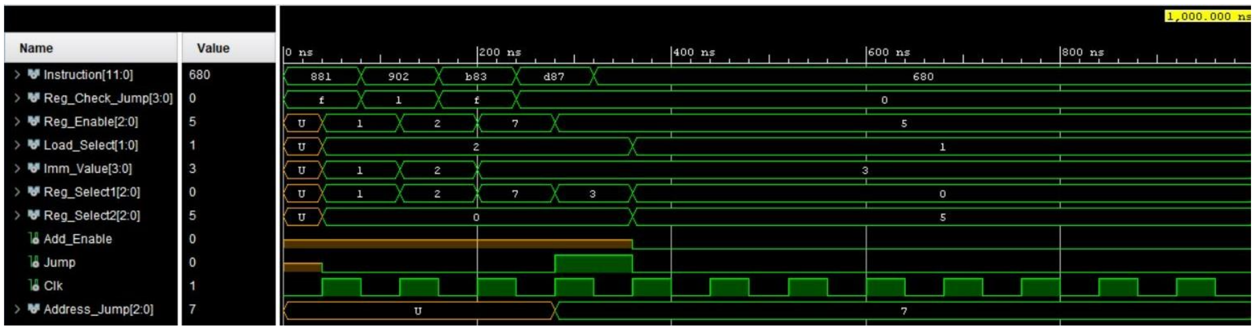
8. Instruction Decoder

❖Design Source

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

entity INS_DECODER is
    Port ( Clk : in STD_LOGIC;
           Instruction : in STD_LOGIC_VECTOR (11 downto 0);
           Reg_Check_Jump : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_Enable : out STD_LOGIC_VECTOR (2 downto 0);
           Load_Select : out STD_LOGIC_VECTOR (1 downto 0);
           Imm_Value : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_Select1 : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_Select2 : out STD_LOGIC_VECTOR (2 downto 0);
           Add_Enable : out STD_LOGIC;
           Jump : out STD_LOGIC;
           Address_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end INS_DECODER;

architecture Behavioral of INS_DECODER is
component BUSES
    Port ( I : in STD_LOGIC_VECTOR (11 downto 0);
       Data : out STD_LOGIC_VECTOR (3 downto 0);
       Fun : out STD_LOGIC_VECTOR (1 downto 0);
       Reg_1 : out STD_LOGIC_VECTOR (2 downto 0);
       Reg_2 : out STD_LOGIC_VECTOR (2 downto 0));
end component;

SIGNAL Reg_EN_S1, Reg_S2 : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL DATA : STD_LOGIC_VECTOR (3 downto 0);
SIGNAL Load_Select_temp : STD_LOGIC_VECTOR (1 downto 0);
signal count : integer := 1;
signal Clk_status : STD_LOGIC := '0';

begin
BUSES_0 : BUSES
port map(
    I =>Instruction,
    DATA =>DATA,
    FUN => Load_Select_temp,
    Reg_1 => Reg_EN_S1,
    --Reg_1 => Reg_Enable,
    Reg_2 => Reg_S2
    );

--    Load_Select <= Load_Select_temp;
Process (Load_Select_temp,Clk)
```

```
Begin
    if (rising_edge(Clk)) then
        if Load_Select_temp ="10" then  -- Load select is move
            Reg_Enable<= Reg_EN_S1;
            Imm_Value <= DATA;
            Reg_Select1 <= Reg_EN_S1;
            Reg_Select2 <= Reg_S2;
            JUMP <= '0';
            Load_Select <= Load_Select_temp;
         end if;

         if Load_Select_temp ="00" then   -- Load select is ADD
           -- Imm_Value <= "0011";
            Add_Enable <= '1';
            Reg_Select1 <= Reg_EN_S1;
            Reg_Enable <= Reg_EN_S1;
            Reg_Select2 <= Reg_S2;
            JUMP <= '0';
            Load_Select <= Load_Select_temp;
        end if;

        if Load_Select_temp ="01" then    --Load select is NEG
            Add_Enable <= '0';
            Reg_Select2 <= Reg_EN_S1 ;
            Reg_Select1 <= "000";
            --Reg_Select2 <= Reg_EN_S1 ;
            Reg_Enable <= Reg_EN_S1 ;
            JUMP <= '0';
            Load_Select <= Load_Select_temp;

        end if;

        if Load_Select_temp ="11" then    --Load select is JUMP
            Reg_Select1 <= Reg_EN_S1;
--            Reg_Select2 <= Reg_S2;
            if  Reg_Check_Jump = "0000" then
                JUMP <= '1';
                Address_Jump(0) <= DATA(0);
                Address_Jump(1) <= DATA(1);
                Address_Jump(2) <= DATA(2);
            else
                JUMP <= '0';
                Address_Jump(0) <= DATA(0);
                Address_Jump(1) <= DATA(1);
                Address_Jump(2) <= DATA(2);

            end if;
            Reg_Enable <= "101" ;
        end if;
    end if;
end process;

end Behavioral;
```

❖Timing Diagram



9. Slow clock
   i. For instruction decoder and register
      ❖Design source

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity Clk_Ins is

    Port ( Clk_in : in STD_LOGIC;

           Clk_out : out STD_LOGIC);

end Clk_Ins;


architecture Behavioral of Clk_Ins is

signal count : integer := 1;

signal Clk_status : STD_LOGIC := '1';


begin
```

```
    process (Clk_in) begin
        if (rising_edge (Clk_in)) then
            count <= count + 1;
            if (count = 2) then
--            if (count = 20000000) then
                Clk_status <= not Clk_status;
                Clk_out <= Clk_status;
                count <= 1;
            end if;
        end if;
    end process;
end Behavioral;
```

ii. For program counter

❖Design source

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

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

architecture Behavioral of Clk_P_Reg is
signal count : integer := 1;
signal Clk_status : STD_LOGIC := '1';

begin
    process (Clk_in) begin
        if (rising_edge (Clk_in)) then
```

```
            count <= count + 1;
            if (count = 5) then
--            if (count = 50000000) then
                Clk_status <= not Clk_status;
                Clk_out <= Clk_status;
                count <= 1;
            end if;
        end if;
    end process;
end Behavioral;
```

- **Nano processor**
    - ❖Design Source

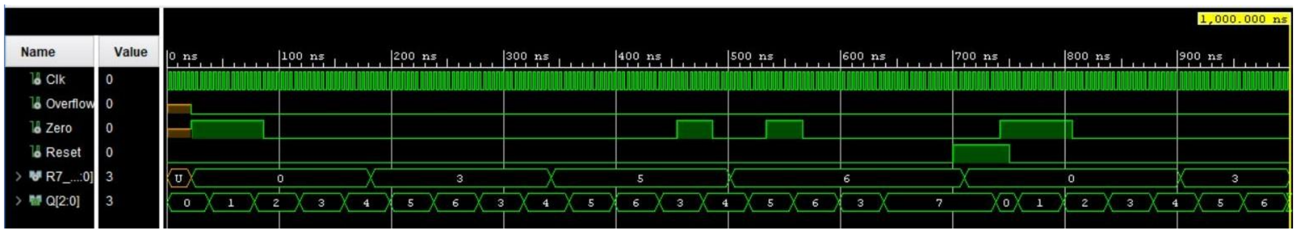
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NanoProcessor is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           R7_Data : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC);
end NanoProcessor;

architecture Behavioral of NanoProcessor is

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

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

----P_Counter-----------------------------------------------------------------------
component P_Counter
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
        Clk : in STD_LOGIC;
        Reset : in STD_LOGIC;
        Q : out STD_LOGIC_VECTOR (2 downto 0));
```

```
end component;

----RCA_3-----------------------------------------------------------------------
component RCA_3
   Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
      B : in STD_LOGIC_VECTOR (2 downto 0);
      C_in : in STD_LOGIC;
      C_out : out STD_LOGIC;
      S : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component MUX2Way3Bit
   Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
      B : in STD_LOGIC_VECTOR (2 downto 0);
      F : out STD_LOGIC_VECTOR (2 downto 0);
      EN : in STD_LOGIC);
end component;

component Program_ROM
   Port ( PC : in STD_LOGIC_VECTOR (2 downto 0);
         Instruction : out STD_LOGIC_VECTOR (11 downto 0));
end component;

component INS_DECODER
   Port ( Clk : in STD_LOGIC;
         Instruction : in STD_LOGIC_VECTOR (11 downto 0);
         Reg_Check_Jump : in STD_LOGIC_VECTOR (3 downto 0);
         Reg_Enable : out STD_LOGIC_VECTOR (2 downto 0);
         Load_Select : out STD_LOGIC_VECTOR (1 downto 0);
         Imm_Value : out STD_LOGIC_VECTOR (3 downto 0);
         Reg_Select1 : out STD_LOGIC_VECTOR (2 downto 0);
         Reg_Select2 : out STD_LOGIC_VECTOR (2 downto 0);
         Add_Enable : out STD_LOGIC;
         Jump : out STD_LOGIC;
         Address_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component MUX2Way4Bit is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
         B : in STD_LOGIC_VECTOR (3 downto 0);
         F : out STD_LOGIC_VECTOR (3 downto 0);
         EN : in STD_LOGIC_VECTOR (1 downto 0));
end component;

component RegisterBank is
   Port ( Selector : in STD_LOGIC_VECTOR (2 downto 0);
         Clk : in STD_LOGIC;
```

```
            Reset : in STD_LOGIC;
            R0_in : in STD_LOGIC_VECTOR (3 downto 0);
            R1_in : in STD_LOGIC_VECTOR (3 downto 0);
            R2_in : in STD_LOGIC_VECTOR (3 downto 0);
            R3_in : in STD_LOGIC_VECTOR (3 downto 0);
            R4_in : in STD_LOGIC_VECTOR (3 downto 0);
            R5_in : in STD_LOGIC_VECTOR (3 downto 0);
            R6_in : in STD_LOGIC_VECTOR (3 downto 0);
            R7_in : in STD_LOGIC_VECTOR (3 downto 0);
            R0_out : out STD_LOGIC_VECTOR (3 downto 0);
            R1_out : out STD_LOGIC_VECTOR (3 downto 0);
            R2_out : out STD_LOGIC_VECTOR (3 downto 0);
            R3_out : out STD_LOGIC_VECTOR (3 downto 0);
            R4_out : out STD_LOGIC_VECTOR (3 downto 0);
            R5_out : out STD_LOGIC_VECTOR (3 downto 0);
            R6_out : out STD_LOGIC_VECTOR (3 downto 0);
            R7_out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component MUX8Way4Bit
    Port ( R0 : in STD_LOGIC_VECTOR (3 downto 0);
            R1 : in STD_LOGIC_VECTOR (3 downto 0);
            R2 : in STD_LOGIC_VECTOR (3 downto 0);
            R3 : in STD_LOGIC_VECTOR (3 downto 0);
            R4 : in STD_LOGIC_VECTOR (3 downto 0);
            R5 : in STD_LOGIC_VECTOR (3 downto 0);
            R6 : in STD_LOGIC_VECTOR (3 downto 0);
            R7 : in STD_LOGIC_VECTOR (3 downto 0);
            REG : in STD_LOGIC_VECTOR (2 downto 0);
            F : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component ADD_SUB
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
        B : in STD_LOGIC_VECTOR (3 downto 0);
        ADD : in STD_LOGIC;
        C_B_in : in STD_LOGIC;
        S_D : out STD_LOGIC_VECTOR (3 downto 0);
        Zero : out STD_LOGIC;
        Overflow : out STD_LOGIC);
end component;

signal Q,Address_Jump,Reg_Enable,Reg_Select1,Reg_Select2 :
STD_LOGIC_VECTOR (2 downto 0);
signal S: STD_LOGIC_VECTOR (2 downto 0);
signal F : STD_LOGIC_VECTOR (2 downto 0);
signal C_out,JUMP,Add_Enable : STD_LOGIC;
signal Instruction : STD_LOGIC_VECTOR (11 downto 0);
signal Imm_Value : STD_LOGIC_VECTOR (3 downto 0);
signal Load_Select : STD_LOGIC_VECTOR (1 downto 0);
signal S_D :  STD_LOGIC_VECTOR (3 downto 0);
signal Reg_in : STD_LOGIC_VECTOR (3 downto 0);
signal
R0_out,R1_out,R2_out,R3_out,R4_out,R5_out,R6_out,R7_out,Mux_1_out,Mux_2_
out : STD_LOGIC_VECTOR (3 downto 0);
signal Carry, Clk_PC_Reg,Clk_Ins_decoder : STD_LOGIC;
begin

Clk_P_Reg_0 : Clk_P_Reg
port map(
   Clk_in => Clk,
   Clk_out => Clk_PC_Reg);


Clk_Ins_0 : Clk_Ins
   port map(
      Clk_in => Clk,
      Clk_out => Clk_Ins_decoder);

P_Counter_0 : P_Counter
port map (
   Reset => Reset,
   D =>F,
  Clk => Clk_PC_Reg,
   Q => Q
);
RCA_3_0: RCA_3
port map (
   A => Q,
   B => "001",
   C_in => '0',
   C_out => C_out,
   S => S
);

MUX2Way3Bit_0 : MUX2Way3Bit
port map (
   A =>Address_Jump ,
   B => S,
   EN => JUMP,
   F => F
);
```

```
Program_ROM_0 :Program_ROM
port map(
   PC => Q,
   Instruction => Instruction
);

INS_DECODER_0 : INS_DECODER
port map(
   Clk => Clk_Ins_decoder,
   Instruction  => Instruction,
   Reg_Check_Jump => Mux_1_out,
   Reg_Enable => Reg_Enable,
   Load_Select => Load_Select,
   Imm_Value => Imm_Value,
   Reg_Select1 => Reg_Select1,
   Reg_Select2 =>Reg_Select2,
   Add_Enable => Add_Enable,
   Jump => JUMP,
   Address_Jump => Address_Jump
   );

MUX2Way4Bit_0 : MUX2Way4Bit
port map(
   A => Imm_Value,
   B => S_D,
   F => Reg_in,
   EN => Load_Select
   );

RegisterBank_0 : RegisterBank
port map(
   Selector => Reg_Enable,
   Clk => Clk_Ins_decoder,
   Reset => Reset,
   R0_in => Reg_in,
   R1_in => Reg_in,
   R2_in => Reg_in,
   R3_in => Reg_in,
   R4_in => Reg_in,
   R5_in => Reg_in,
   R6_in => Reg_in,
   R7_in => Reg_in,
   R0_out=> R0_out,
   R1_out=> R1_out,
   R2_out=> R2_out,
   R3_out=> R3_out,
   R4_out=> R4_out,
   R5_out=> R5_out,
   R6_out=> R6_out,
   R7_out=> R7_out
   );

MUX8Way4Bit_A :MUX8Way4Bit
port map (
   R0 => R0_out,
   R1 => R1_out,
   R2 => R2_out,
   R3 => R3_out,
   R4 => R4_out,
   R5 => R5_out,
   R6 => R6_out,
   R7 => R7_out,
   REG => Reg_Select1,
   F => Mux_1_out
   );

MUX8Way4Bit_B :MUX8Way4Bit
port map (
   R0 => R0_out,
   R1 => R1_out,
   R2 => R2_out,
   R3 => R3_out,
   R4 => R4_out,
   R5 => R5_out,
   R6 => R6_out,
   R7 => R7_out,
   REG => Reg_Select2,
   F => Mux_2_out
   );

ADD_SUB_0 : ADD_SUB
port map (
   A => Mux_1_out,
   B => Mux_2_out,
   ADD => Add_Enable,
   C_B_in  => '0',
   S_D => S_D,
   Overflow => Overflow,
   Zero => Zero
   );
 R7_Data <=  R7_out;

end Behavioral;
```

❖Timing Diagram



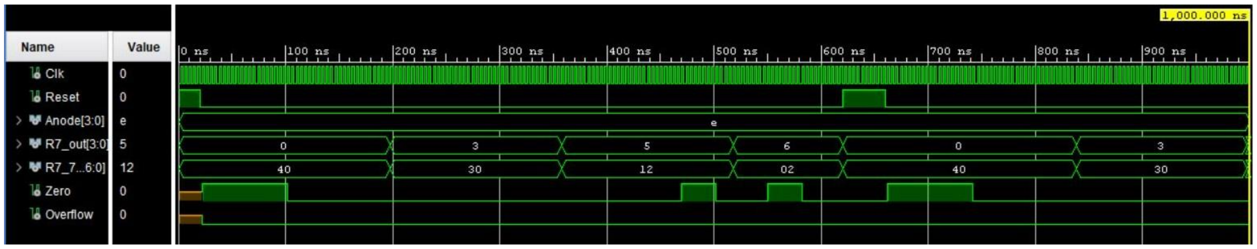
- **Display R7**

❖Design Source

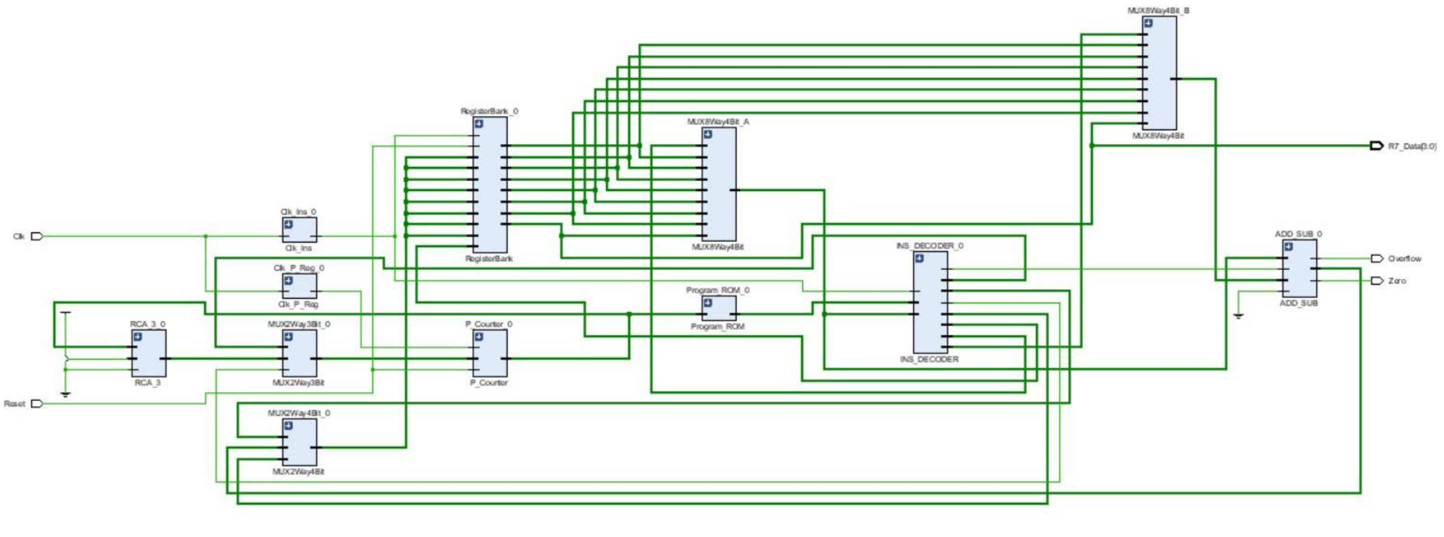
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Disply_R7 is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Anode : out STD_LOGIC_VECTOR (3 downto 0):= "1110";
           R7_out : out STD_LOGIC_VECTOR (3 downto 0);
           R7_7Seg : out STD_LOGIC_VECTOR (6 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC);
end Disply_R7;

architecture Behavioral of Disply_R7 is

component NanoProcessor
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           R7_Data : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC);
end component;


component LUT_16_7
    Port ( Data : in STD_LOGIC_VECTOR (3 downto 0);
           disply_data : out STD_LOGIC_VECTOR (6 downto 0));
end component;

signal R7_Data1 : STD_LOGIC_VECTOR (3 downto 0);
```

```
begin

NanoProcessor_0 :NanoProcessor
    Port map( Clk => Clk,
           Reset => Reset,
           R7_Data => R7_Data1,
           Zero => Zero,
           Overflow => Overflow
           );


LUT_16_7_0 : LUT_16_7
    port map(
        Data => R7_Data1,
        disply_data => R7_7Seg
        );

R7_out <= R7_Data1;


end Behavioral
```

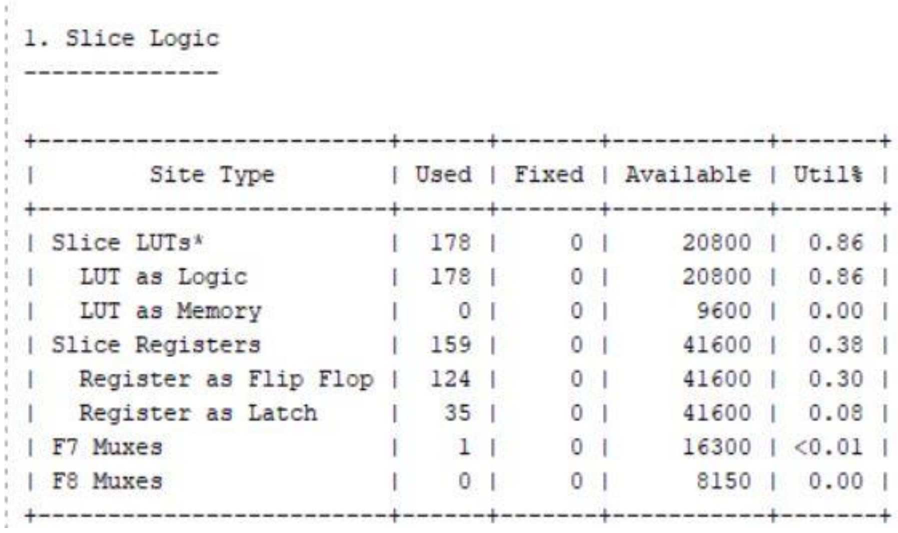
❖Timing diagram



- **Schematic Diagram of Nano processor**



- **Resource utilization (Slice LUTs and Slice Registers)**



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

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |  178 |     0 |     20800 |  0.86 |
|   LUT as Logic          |  178 |     0 |     20800 |  0.86 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         |  159 |     0 |     41600 |  0.38 |
|   Register as Flip Flop |  124 |     0 |     41600 |  0.30 |
|   Register as Latch     |   35 |     0 |     41600 |  0.08 |
| F7 Muxes                |    1 |     0 |     16300 | <0.01 |
| F8 Muxes                |    0 |     0 |      8150 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

- **Conclusions**

To build a functioning processor, we need build the components individually and test each of their functionalities. We can use busses to efficiently implement our design instead of letting so many wires to run around. As our processor won't understand assembly language instructions, we have to translate it into machine code, and have to hard code that into our ROM. Using the designs from previous labs will be helpful instead of building them from scratch. We all understood how the components of the processor works internally and about how the processor decodes and execute instructions. We also learned to work as a team and developed many interpersonal skills.

- **Contribution of Each Member**

Name	**Designed components**	**No. of hours spent**
Thanushanth K	Adder Subtractor Instruction Decoder	6
Lithurshan K	Program ROM k-way b-bit Mux	6
Nithursika K	Register Bank 3-bit Adder	5
Sabthana J	Program Counter Slow clock	5
Aathavan N	7-segment display Buses	5
All of us jointly involved in combining and debugging the final design.		