Lab 9-10 Report

210495G – Nipun Viraj

210450P – Thiwanka Roshen

ASSIGNED TASK

In this lab we designed a 4 – bit processor which can execute 4 instructions (MOVI, ADD, NEG and JZR). The nanoprocessor was built using the following components.

- 4 – bit add/sub unit.
- 3 – bit Adder
- 3 – bit Program Counter (PC).
- k – way b – bit Multiplexer.
- Register Bank.
- Program ROM.
- Instruction Decoder.

We have created VHDL codes for the key components mentioned above, and we have verified their functionality through simulations. Timing diagrams have been generated for each component, and an XDC VHDL code has been developed.

In the XDC VHDL code, the value of register 7 is connected as an output to three LEDs and a 7-segment display. Additionally, there are two LED indicators, LD14 and LD15, which indicate the zero and overflow flags. When the reset button is pressed, all register values and the program counter are reset to zero.

ASSEMBLY CODE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.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 ROM is
    Port ( MemSelect : in STD_LOGIC_VECTOR (2 downto 0);
           InstructBus : 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 Assembly_Code : rom_type := (
                    "100010000011", -- Move the value 2(binary 0011) to the register 1.
                    "100100000001", -- Move the value 1(binary 0011) to the register 2.
                    "010100000000", -- Negate the value in register 2.
                    "001110010000", -- Add the values in register 7 and register 1 and store result in Register 1.
                    "000010100000", -- Add the values in register 1 and register 2 and store result in Register 1.
                    "110010000111", -- Jump if value in register 1 is zero.(if R==0;Pc=7 || else;Pc=Pc+1)
                    "110000000011", -- Jump if value in register 0 is zero.(if R==0;Pc=3 || else;Pc=Pc+1)
                    "110000000111"  -- Jump if value in register 0 is zero.(if R==0;Pc=7 || else;Pc=Pc+1)
         );
begin
    InstructBus <= Assembly_Code(to_integer(unsigned(MemSelect)));
end Behavioral;
```

VHDL FILES

1. Slow Clock

```
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 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 = 50000000 ) then --50000000 for project (2 for test)
                clk_status <= not clk_status;
                Clk_out <= clk_status;
                count <= 1;
            end if;
        end if;
    end process;

end Behavioral;
```

2. LUT

```
entity LUT is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end LUT;

architecture Behavioral of LUT is

type rom_type is array (0 to 15) of std_logic_vector(6 downto 0);
    signal sevenSegment_ROM : rom_type := (
        "1000000", -- 0     0= on 1=off    g f e d c b a
        "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;
```

3. Program Counter

```
entity PC is
    Port ( Reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           MemorySelect : inout STD_LOGIC_VECTOR (2 downto 0);
           JumpAddr : in STD_LOGIC_VECTOR (2 downto 0);               -- If there is a jump instruct
           JumpFlag : in STD_LOGIC
           );
end PC;

architecture Behavioral of PC is
    COMPONENT DFF
    PORT (
        D : in STD_LOGIC;
        Res : in STD_LOGIC;
        Clk : in STD_LOGIC;
        Q : out STD_LOGIC;
        Qbar : out STD_LOGIC );
    END COMPONENT;

    --COMPONENT Slow_Clk
    --PORT (
        --Clk_in : in STD_LOGIC;
        --Clk_out : out STD_LOGIC );
    --END COMPONENT;

    COMPONENT RCA
    PORT (
        B : in STD_LOGIC_VECTOR (2 downto 0);       -- Always this is equals to 001
        S : out STD_LOGIC_VECTOR (2 downto 0);      -- Next memory address will out
        C_out : out STD_LOGIC);
    END COMPONENT;

    COMPONENT MUX_2W_3B
    PORT ( jmp_adrs : in STD_LOGIC_VECTOR (2 downto 0);
           adder_3bit : in STD_LOGIC_VECTOR (2 downto 0);
           jmp_flag : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (2 downto 0));
    END COMPONENT;

    signal D, A: std_logic_vector (2 downto 0);
    --signal Clk_slow : std_logic;
```

```
    begin

        --Slow_Clk0 : Slow_Clk
            --port map (
                --Clk_in => Clk,
                --Clk_out => Clk_slow );

        DFF0 : DFF
            port map (
                D => D(0),
                Q => MemorySelect(0),
                Res => Reset,
                Clk => Clk);--Clk_slow );

        DFF1 : DFF
            port map (
                D => D(1),
                Q => MemorySelect(1),
                Res => Reset,
                Clk => Clk);--Clk_slow );

        DFF2 : DFF
            port map (
                 D => D(2),
                 Q => MemorySelect(2),
                 Res => Reset,
                 Clk => Clk);--Clk_slow );

        RCA0 : RCA
            port map (
                 B => MemorySelect,
                 S => A );

        MUX_2W_3B0 : MUX_2W_3B
            port map (
                 jmp_adrs => JumpAddr,
                 adder_3bit => A,
                 jmp_flag => JumpFlag,
                 output => D  );


    end Behavioral;
```

4. ROM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.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 ROM is
    Port ( MemSelect : in STD_LOGIC_VECTOR (2 downto 0);
           InstructBus : 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 Assembly_Code : rom_type := (
                    "100010000011", -- Move the value 2(binary 0011) to the register 1.
                    "100100000001", -- Move the value 1(binary 0011) to the register 2.
                    "010100000000", -- Negate the value in register 2.
                    "001110010000", -- Add the values in register 7 and register 1 and store result in Register 1.
                    "000010100000", -- Add the values in register 1 and register 2 and store result in Register 1.
                    "110010000111", -- Jump if value in register 1 is zero.(if R==0;Pc=7 || else;Pc=Pc+1)
                    "110000000011", -- Jump if value in register 0 is zero.(if R==0;Pc=3 || else;Pc=Pc+1)
                    "110000000111"  -- Jump if value in register 0 is zero.(if R==0;Pc=7 || else;Pc=Pc+1)
         );
begin
    InstructBus <= Assembly_Code(to_integer(unsigned(MemSelect)));
end Behavioral;
```

5. Instruction Decoder

```vhdl
entity Instruction_Decoder is
    Port ( Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
           Load_Select : out STD_LOGIC;
           Immediate_Val : out STD_LOGIC_VECTOR (3 downto 0);
           MuxA_Select : out STD_LOGIC_VECTOR (2 downto 0);
           MuxB_Select : out STD_LOGIC_VECTOR (2 downto 0);
           AddSub_Select : out STD_LOGIC;
           JmpFlag : out STD_LOGIC;
           JmpAddr : out STD_LOGIC_VECTOR (2 downto 0);
           Instruct_Bus : in STD_LOGIC_VECTOR (11 downto 0);
           JumpCheck : in STD_LOGIC_VECTOR (3 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is

begin
   process(Instruct_Bus, JumpCheck)
   begin

       -- for ADD intruction
       if Instruct_Bus (11 downto 10) = "00" then
           JmpFlag <= '0';

           MuxA_Select <= Instruct_Bus (9 downto 7);
           MuxB_Select <= Instruct_Bus (6 downto 4);
           AddSub_Select <= '0'; -- Here we perform only addition.
           Load_Select <= '0';
           Reg_En <= Instruct_Bus (9 downto 7);  -- Store the result on Register A

       -- for NEG intruction
       elsif Instruct_Bus (11 downto 10) = "01" then
           JmpFlag <= '0';

           MuxA_Select <= "000";  -- For get -B we have to set A to zero(i.e. set to Register 0) because adder perform only A+B.
           MuxB_Select <= Instruct_Bus (9 downto 7);  -- For get a negative value of a particular number, it can be only done from Mux B
           AddSub_Select <= '1';  -- Here we perform only subtraction.
           Load_Select <= '0';
           Reg_En <= Instruct_Bus (9 downto 7);  -- Store the result on Register A

       -- for MOV intruction
       elsif Instruct_Bus (11 downto 10) = "10" then
           JmpFlag <= '0';
           MuxA_Select <= "000";
           MuxB_Select <= "000";

           Immediate_Val <= Instruct_Bus (3 downto 0);
           Reg_En <= Instruct_Bus (9 downto 7);
           Load_Select <= '1';  -- Select IMMEDIATE VALUE port via

       -- for JMP intruction
       else
           MuxB_Select <= "000";
           Immediate_Val <= "0000";
           Load_Select <= '1';
           Reg_En <= "000";

           MuxA_Select <= Instruct_Bus (9 downto 7);
           if JumpCheck = "0000" then
               JmpAddr <= Instruct_Bus (2 downto 0);
               JmpFlag <= '1';
           else
               JmpFlag <= '0';
           end if;
       end if;
   end process;

end Behavioral;
```

6. Mux 2 way 4 bit

```
entity MUX_2W_4B is
    Port ( AddSub : in STD_LOGIC_VECTOR (3 downto 0);
           Immediate_Val : in STD_LOGIC_VECTOR (3 downto 0);
           Load_Select : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (3 downto 0));
end MUX_2W_4B;

architecture Behavioral of MUX_2W_4B is

COMPONENT MUX_2_1
PORT ( D : in STD_LOGIC_VECTOR (1 downto 0);
       S : in STD_LOGIC;
       Y : out STD_LOGIC);
END COMPONENT;

begin
Mux_0: MUX_2_1 port map(
    D(1) => AddSub(0),
    D(0) => Immediate_Val(0),
    S => Load_Select,
    Y => output(0)
);

Mux_1: MUX_2_1 port map(
    D(1) => AddSub(1),
    D(0) => Immediate_Val(1),
    S => Load_Select,
    Y => output(1)
);

Mux_2: MUX_2_1 port map(
    D(1) => AddSub(2),
    D(0) => Immediate_Val(2),
    S => Load_Select,
    Y => output(2)
);

Mux_3: MUX_2_1 port map(
    D(1) => AddSub(3),
    D(0) => Immediate_Val(3),
    S => Load_Select,
    Y => output(3)
);  end Behavioral;
```

7. Mux 0 (Subfile of Mux 2 way 4 bit)

```
entity MUX_2_1 is
    Port ( D : in STD_LOGIC_VECTOR (1 downto 0);
           S : in STD_LOGIC;
           Y : out STD_LOGIC);
end MUX_2_1;

architecture Behavioral of MUX_2_1 is
Signal NOT_S : std_logic;

begin
    NOT_S <= NOT(S);

    Y <= (D(1) AND NOT_S) OR (D(0) AND S);
end Behavioral;
```

8. Add Sub Unit

```
21
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity ADD_SUB is
35      Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
36             B : in STD_LOGIC_VECTOR (3 downto 0);
37             S : inout STD_LOGIC_VECTOR (3 downto 0);
38             AddSubSelect : in STD_LOGIC;
39             Zero : out STD_LOGIC;
40             Overflow : out STD_LOGIC);
41  end ADD_SUB;
42
43  architecture Behavioral of ADD_SUB is
44
45      COMPONENT FA
46          PORT(A,B,C_in : IN STD_LOGIC;
47              S,C_out : OUT STD_LOGIC);
48              end component;
49
50         SIGNAL C0,C1,C2,C3,Z:
51         STD_LOGIC;
52         SIGNAL Q,Stemp: STD_LOGIC_VECTOR(3 downto 0);
53
54  begin
55      Q(0)<= AddSubSelect XOR B(0);
56      Q(1)<= AddSubSelect XOR B(1);
57      Q(2)<= AddSubSelect XOR B(2);
58      Q(3)<= AddSubSelect XOR B(3);
59
60       FA0:FA
61          PORT MAP(
62              A=>A(0),
```

```
FA0:FA
   PORT MAP(
       A=>A(0),
       B=>Q(0),
       C_in=>AddSubSelect,
       S=>Stemp(0),
       C_out=>C0);



FA1:FA
   PORT MAP(
       A=>A(1),
       B=>Q(1),
       C_in=>C0,
       S=>Stemp(1),
       C_out=>C1);

FA2 :FA
   PORT MAP(
       A=>A(2),
       B=>Q(2),
       C_in=>C1,
       S=>Stemp(2),
       C_out=>C2);

FA3 :FA
   PORT MAP(
       A=>A(3),
       B=>Q(3),
       C_in=>C2,
       S=>Stemp(3),
       C_out=>C3);


S<=Stemp;
Z<=C3 XOR C2;
Overflow<=Z;
Zero<=NOT(Stemp(0) OR Stemp(1) OR Stemp(2) OR Stemp(3) OR Z);


end Behavioral;
```

9. Mux 8Way 4 Bit

```
entity MUX_8W_4B is
    Port (  reg0 : in STD_LOGIC_VECTOR (3 downto 0);
            reg1 : in STD_LOGIC_VECTOR (3 downto 0);
            reg2 : in STD_LOGIC_VECTOR (3 downto 0);
            reg3 : in STD_LOGIC_VECTOR (3 downto 0);
            reg4 : in STD_LOGIC_VECTOR (3 downto 0);
            reg5 : in STD_LOGIC_VECTOR (3 downto 0);
            reg6 : in STD_LOGIC_VECTOR (3 downto 0);
            reg7 : in STD_LOGIC_VECTOR (3 downto 0);
            Reg_select : in STD_LOGIC_VECTOR (2 downto 0);
            output : out STD_LOGIC_VECTOR (3 downto 0));
end MUX_8W_4B;

architecture Behavioral of MUX_8W_4B is

begin
    process(reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7, Reg_select)
    begin
        case (Reg_select) is
            when "000" =>
                output <= reg0;
            when "001" =>
               output <= reg1;
            when "010" =>
               output <= reg2;
            when "011" =>
               output <= reg3;
            when "100" =>
               output <= reg4;
            when "101" =>
               output <= reg5;
            when "110" =>
               output <= reg6;
            when "111" =>
               output <= reg7;
            when others =>
               output <= "0000";  -- For handle any undefined cases
        end case;
    end process;
end Behavioral;
```

10. Register Bank

```
34  entity Reg_Bank is
35      Port ( D_input : in STD_LOGIC_VECTOR (3 downto 0);
36             Reg_En : in STD_LOGIC_VECTOR (2 downto 0);
37             Clock : in STD_LOGIC;
38             Reset : in STD_LOGIC;
39             D_out0 : out STD_LOGIC_VECTOR (3 downto 0);
40             D_out1 : out STD_LOGIC_VECTOR (3 downto 0);
41             D_out2 : out STD_LOGIC_VECTOR (3 downto 0);
42             D_out3 : out STD_LOGIC_VECTOR (3 downto 0);
43             D_out4 : out STD_LOGIC_VECTOR (3 downto 0);
44             D_out5 : out STD_LOGIC_VECTOR (3 downto 0);
45             D_out6 : out STD_LOGIC_VECTOR (3 downto 0);
46             D_out7 : out STD_LOGIC_VECTOR (3 downto 0));
47  end Reg_Bank;
48
49  architecture Behavioral of Reg_Bank is
50
51      component Dec_3_8 is
52          Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
53                 EN : in STD_LOGIC;
54                 Y : out STD_LOGIC_VECTOR (7 downto 0));
55      end component;
56     --####################THIS REG D FLIP FLOP MUST HAVE a reset PIN. ADD THIS FOR COMPLETION
57      component REG is
58          Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
59                 En : in STD_LOGIC;
60                 Clk : in STD_LOGIC;
61                 Reset : in STD_LOGIC;
62                 Q : out STD_LOGIC_VECTOR (3 downto 0));
63      end component;
64
65  signal y : STD_LOGIC_VECTOR (7 downto 0);
66
67  begin
68      decoder : Dec_3_8
69      port map (
70          I => Reg_En,
71          En => '1',
72          Y => y );
73
74      Reg0 : REG
75      port map (
76          D => "0000",
```

```
76          D => "0000",
77          En => y(0),
78          Clk => Clock,
79          Reset => Reset,
80          Q => D_out0 );
81
82      Reg1 : REG
83      port map (
84          D => D_input,
85          En => y(1),
86          Clk => Clock,
87          Reset => Reset,
88          Q => D_out1 );
89
90       Reg2 : REG
91       port map (
92          D => D_input,
93          En => y(2),
94          Clk => Clock,
95          Reset => Reset,
96          Q => D_out2 );
97
98       Reg3 : REG
99       port map (
100         D => D_input,
101         En => y(3),
102         Clk => Clock,
103         Reset => Reset,
104         Q => D_out3 );
105
106      Reg4 : REG
107      port map (
108         D => D_input,
109         En => y(4),
110         Clk => Clock,
111         Reset => Reset,
112         Q => D_out4 );
113
114      Reg5 : REG
115      port map (
116         D => D_input,
117         En => y(5),
118         Clk => Clock,
```

```
113
114         Reg5 : REG
115         port map (
116            D => D_input,
117            En => y(5),
118            Clk => Clock,
119            Reset => Reset,
120            Q => D_out5 );
121
122         Reg6 : REG
123         port map (
124            D => D_input,
125            En => y(6),
126            Clk => Clock,
127            Reset => Reset,
128            Q => D_out6 );
129
130         Reg7 : REG
131         port map (
132            D => D_input,
133            En => y(7),
134            Clk => Clock,
135            Reset => Reset,
136            Q => D_out7 );
137
138   end Behavioral;
139
```

11. Register Bank (3 to 8 Decoder)

```
entity Dec_3_8 is
    Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end Dec_3_8;

architecture Behavioral of Dec_3_8 is
Component Dec_2_4
port(I:in std_logic_vector(1 downto 0);
     EN:in std_logic;
     Y:out std_logic_vector(3 downto 0));
end component;
signal z1,z2 : std_logic;
signal l1,l2 : std_logic_vector(1 downto 0);
signal m1,m2 : std_logic_vector(3 downto 0);
begin

Dec_2_4_0:Dec_2_4 port map(
    I =>l1,
    EN =>z1,
    Y => m1
);
Dec_2_4_1:Dec_2_4 port map(
    I =>l2,
    EN=>z2,
    Y=>m2
);
l1 <= I(1 downto 0);
l2<= I(1 downto 0);
z1 <=  EN and not I(2);
z2 <=  EN and I(2);
Y(3 downto 0) <= m1;
Y(7 downto 4)<=m2;
end Behavioral;
```

12. Register Bank (2 to 4 Decoder)

```
entity Dec_2_4 is
    Port ( I : in STD_LOGIC_VECTOR (1 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end Dec_2_4;

architecture Behavioral of Dec_2_4 is

begin
Y(0) <= EN and not I(0) and not I(1);
Y(1) <= EN and  I(0) and  not I(1);
Y(2) <= EN and  not I(0) and  I(1);
Y(3) <= EN and  I(0) and  I(1);

end Behavioral;
```

13. Register Bank (Register)

```
entity REG is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           En : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;    -- add reset pin for reset output
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end REG;

architecture Behavioral of REG is

begin
    process(Clk) begin
    if (Reset = '1') then
        Q <= (others => '0');
    elsif (rising_edge(Clk)) then
        if En = '1' then
            Q <= D;
        end if;
    end if;
    end process;

end Behavioral;
```

14. NanoProcessor (Main Code)

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity NanoProcessor is
35      Port ( Reset : in STD_LOGIC;
36             clk : in STD_LOGIC;
37             Zero_LED : out STD_LOGIC;
38             Overflow_LED : out STD_LOGIC;
39             LED0 : out STD_LOGIC;
40             LED1 : out STD_LOGIC;
41             LED2 : out STD_LOGIC;
42             LED3 : out STD_LOGIC;
43             Anode : out STD_LOGIC_VECTOR (3 downto 0);
44             S_7Seg : out STD_LOGIC_VECTOR (6 downto 0)
45
46      );
47  end NanoProcessor;
48
49  architecture Behavioral of NanoProcessor is
50
51  component Reg_Bank is
52      Port ( D_input : in STD_LOGIC_VECTOR (3 downto 0);
53             Reg_En : in STD_LOGIC_VECTOR (2 downto 0);
54             Clock : in STD_LOGIC;
55             Reset : in STD_LOGIC;
56             D_out0 : out STD_LOGIC_VECTOR (3 downto 0);
57             D_out1 : out STD_LOGIC_VECTOR (3 downto 0);
58             D_out2 : out STD_LOGIC_VECTOR (3 downto 0);
59             D_out3 : out STD_LOGIC_VECTOR (3 downto 0);
60             D_out4 : out STD_LOGIC_VECTOR (3 downto 0);
61             D_out5 : out STD_LOGIC_VECTOR (3 downto 0);
62             D_out6 : out STD_LOGIC_VECTOR (3 downto 0);
63             D_out7 : out STD_LOGIC_VECTOR (3 downto 0));
64  end component;
```

```
64  end component;
65
66  component MUX_8W_4B is
67      Port (  reg0 : in STD_LOGIC_VECTOR (3 downto 0);
68              reg1 : in STD_LOGIC_VECTOR (3 downto 0);
69              reg2 : in STD_LOGIC_VECTOR (3 downto 0);
70              reg3 : in STD_LOGIC_VECTOR (3 downto 0);
71              reg4 : in STD_LOGIC_VECTOR (3 downto 0);
72              reg5 : in STD_LOGIC_VECTOR (3 downto 0);
73              reg6 : in STD_LOGIC_VECTOR (3 downto 0);
74              reg7 : in STD_LOGIC_VECTOR (3 downto 0);
75              Reg_select : in STD_LOGIC_VECTOR (2 downto 0);
76              output : out STD_LOGIC_VECTOR (3 downto 0));
77  end component;
78
79  component ADD_SUB is
80      Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
81             B : in STD_LOGIC_VECTOR (3 downto 0);
82             S : inout STD_LOGIC_VECTOR (3 downto 0);
83             AddSubSelect : in STD_LOGIC;
84             Zero : out STD_LOGIC;
85             Overflow : out STD_LOGIC);
86  end component;
87
88  component Instruction_Decoder is
89      Port ( Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
90             Load_Select : out STD_LOGIC;
91             Immediate_Val : out STD_LOGIC_VECTOR (3 downto 0);
92             MuxA_Select : out STD_LOGIC_VECTOR (2 downto 0);
93             MuxB_Select : out STD_LOGIC_VECTOR (2 downto 0);
94             AddSub_Select : out STD_LOGIC;
95             JmpFlag : out STD_LOGIC;
96             JmpAddr : out STD_LOGIC_VECTOR (2 downto 0);
97             Instruct_Bus : in STD_LOGIC_VECTOR (11 downto 0);
98             JumpCheck : in STD_LOGIC_VECTOR (3 downto 0));
99  end component;
100
101 component ROM is
102     Port ( MemSelect : in STD_LOGIC_VECTOR (2 downto 0);
103            InstructBus : out STD_LOGIC_VECTOR (11 downto 0));
104 end component;
105
106 component PC is
```

```
component PC is
    Port ( Reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           MemorySelect : inout STD_LOGIC_VECTOR (2 downto 0);
           JumpAddr : in STD_LOGIC_VECTOR (2 downto 0);               -- If there is a jump instruction, PC set MemorrySelect as the
           JumpFlag : in STD_LOGIC);
end component;

component MUX_2W_4B is
    Port ( AddSub : in STD_LOGIC_VECTOR (3 downto 0);
           Immediate_Val : in STD_LOGIC_VECTOR (3 downto 0);
           Load_Select : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (3 downto 0));
end component;

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

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

signal Clk_slow : std_logic;
signal Memory_Select : std_logic_vector (2 downto 0);  -- from program counter to program ROM
signal Jump_Address : std_logic_vector (2 downto 0);  -- from instruction decoder to program counter
signal Jump_Flag : std_logic; -- from instruction decoder to program counter
signal I : std_logic_vector (11 downto 0);  -- from program ROM to instruction decoder
signal Register_Enable : std_logic_vector (2 downto 0);  -- from instruction decoder to Register Bank
signal Load_Select : std_logic; -- from instruction decoder to Mux_2way_4bit
signal Immediate_Value : std_logic_vector (3 downto 0);  -- from instruction decoder to Mux_2way_4bit
signal Mux_A_Select : std_logic_vector (2 downto 0);  -- from instruction decoder to Mux_8way_4bit(A)
signal Mux_B_Select : std_logic_vector (2 downto 0);  -- from instruction decoder to Mux_8way_4bit(B)
signal ADD_SUB_Select : std_logic; -- from instruction decoder to Add_Sub_4bit
signal Mux_A : std_logic_vector (3 downto 0);  -- from Mux_8way_4bit(A) to Add_Sub_4bit / from instruction decoder to Mux_8way_4bit(A) output
signal Mux_B : std_logic_vector (3 downto 0);  -- from Mux_8way_4bit(B) to Add_Sub_4bit
signal Add_Sub_Result : std_logic_vector (3 downto 0);  -- from Add_Sub_4bit to Mux_2way_4bit
signal Register_Input : std_logic_vector (3 downto 0);  -- from Mux_2way_4bit to Register Bank
signal R0 : std_logic_vector (3 downto 0);  -- from R0 to Mux_8way_4bit
signal R1 : std_logic_vector (3 downto 0);  -- from R1 to Mux_8way_4bit
signal R2 : std_logic_vector (3 downto 0);  -- from R2 to Mux_8way_4bit
```

```
147  signal R1 : std_logic_vector (3 downto 0);  -- from R1 to Mux_8way_4bit
148  signal R2 : std_logic_vector (3 downto 0);  -- from R2 to Mux_8way_4bit
149  signal R3 : std_logic_vector (3 downto 0);  -- from R3 to Mux_8way_4bit
150  signal R4 : std_logic_vector (3 downto 0);  -- from R4 to Mux_8way_4bit
151  signal R5 : std_logic_vector (3 downto 0);  -- from R5 to Mux_8way_4bit
152  signal R6 : std_logic_vector (3 downto 0);  -- from R6 to Mux_8way_4bit
153  signal R7 : std_logic_vector (3 downto 0);  -- from R7 to Mux_8way_4bit
154
155  signal S_7s : STD_LOGIC_VECTOR (6 downto 0) := "0000000";
156  signal clk_7seg : STD_LOGIC;
157  signal selected_7seg : integer := 0;
158
159  begin
160
161  Slow_Clock : Slow_Clk
162  port map (
163      Clk_in => clk,
164      Clk_out => Clk_slow
165  );
166
167  LUT_0 : LUT
168  port map (
169      address => R7,
170      data => S_7s
171   );
172
173  PC_0 : PC
174  port map (
175      Reset => Reset,
176      Clk => Clk_slow,
177      MemorySelect => Memory_Select,
178      JumpAddr => Jump_Address,
179      JumpFlag => Jump_Flag
180  );
181
182  ROM0 : ROM
183  port map (
184      MemSelect => Memory_Select,
185      InstructBus => I
186  );
187
188  Instruction_Decoder0 : Instruction_Decoder
189  port map (
```

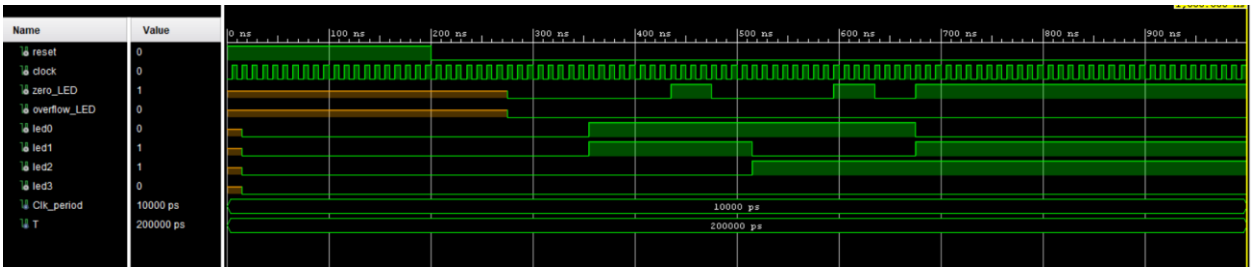
```
186 );
187
188 Instruction_Decoder0 : Instruction_Decoder
189 port map (
190     Reg_En => Register_Enable,
191     Load_Select => Load_Select,
192     Immediate_Val => Immediate_Value,
193     MuxA_Select => Mux_A_Select,
194     MuxB_Select => Mux_B_Select,
195     AddSub_Select => ADD_SUB_Select,
196     JmpFlag => Jump_Flag,
197     JmpAddr => Jump_Address,
198     Instruct_Bus => I,
199     JumpCheck => Mux_A
200 );
201
202 Mux_2way_4bit0 : Mux_2W_4B
203 port map (
204     AddSub =>  Add_Sub_Result,
205     Immediate_Val => Immediate_Value,
206     Load_Select => Load_Select,
207     output => Register_Input
208 );
209
210 ADD_SUB_Unit : Add_Sub
211 port map (
212     A => Mux_A,
213     B => Mux_B,
214     S => Add_Sub_Result,
215     AddSubSelect => ADD_SUB_Select,
216     Zero => Zero_LED,
217     Overflow => Overflow_LED
218 );
219
220 MuxA : MUX_8W_4B
221 port map (
222     reg0 => R0,
223     reg1 => R1,
224     reg2 => R2,
225     reg3 => R3,
226     reg4 => R4,
227     reg5 => R5,
228     reg6 => R6,
```

```
228         reg6 => R6,
229         reg7 => R7,
230         Reg_select => Mux_A_Select,
231         output => Mux_A
232     );
233 
234     MuxB : MUX_8W_4B
235     port map (
236         reg0 => R0,
237         reg1 => R1,
238         reg2 => R2,
239         reg3 => R3,
240         reg4 => R4,
241         reg5 => R5,
242         reg6 => R6,
243         reg7 => R7,
244         Reg_select => Mux_B_Select,
245         output => Mux_B
246     );
247 
248     RegisterBank : Reg_Bank
249     port map (
250         D_input => Register_Input,
251         Reg_En => Register_Enable,
252         Clock => Clk_slow,
253         Reset => Reset,
254         D_out0 => R0,
255         D_out1 => R1,
256         D_out2 => R2,
257         D_out3 => R3,
258         D_out4 => R4,
259         D_out5 => R5,
260         D_out6 => R6,
261         D_out7 => R7
262     );
263 
264     LED0 <= R7(0);
265     LED1 <= R7(1);
266     LED2 <= R7(2);
267     LED3 <= R7(3);
268 
269     process(clk_7seg)
270     begin
```

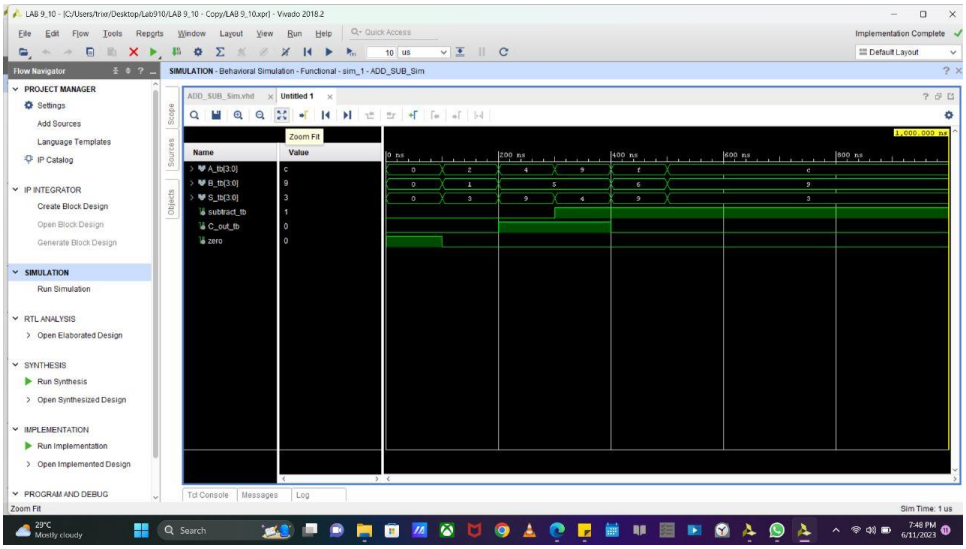
```
    reg7 => R7,
    Reg_select => Mux_B_Select,
    output => Mux_B
);

RegisterBank : Reg_Bank
port map (
    D_input => Register_Input,
    Reg_En => Register_Enable,
    Clock => Clk_slow,
    Reset => Reset,
    D_out0 => R0,
    D_out1 => R1,
    D_out2 => R2,
    D_out3 => R3,
    D_out4 => R4,
    D_out5 => R5,
    D_out6 => R6,
    D_out7 => R7
);

LED0 <= R7(0);
LED1 <= R7(1);
LED2 <= R7(2);
LED3 <= R7(3);

process(clk_7seg)
begin
    --if(selected_7seg = 0) then
        Anode <= "1110";
        S_7Seg <= S_7s;
    --elsif ( selected_7seg = 1 ) then
        --Anode <= "1101";
        --if(c = '0') then
            --S_7seg <= "0000001";
        --else
            --S_7seg <= "1001111";
        --end if;
    --end if;
end process;

end Behavioral;
```

3. TIMING DIAGRAMS

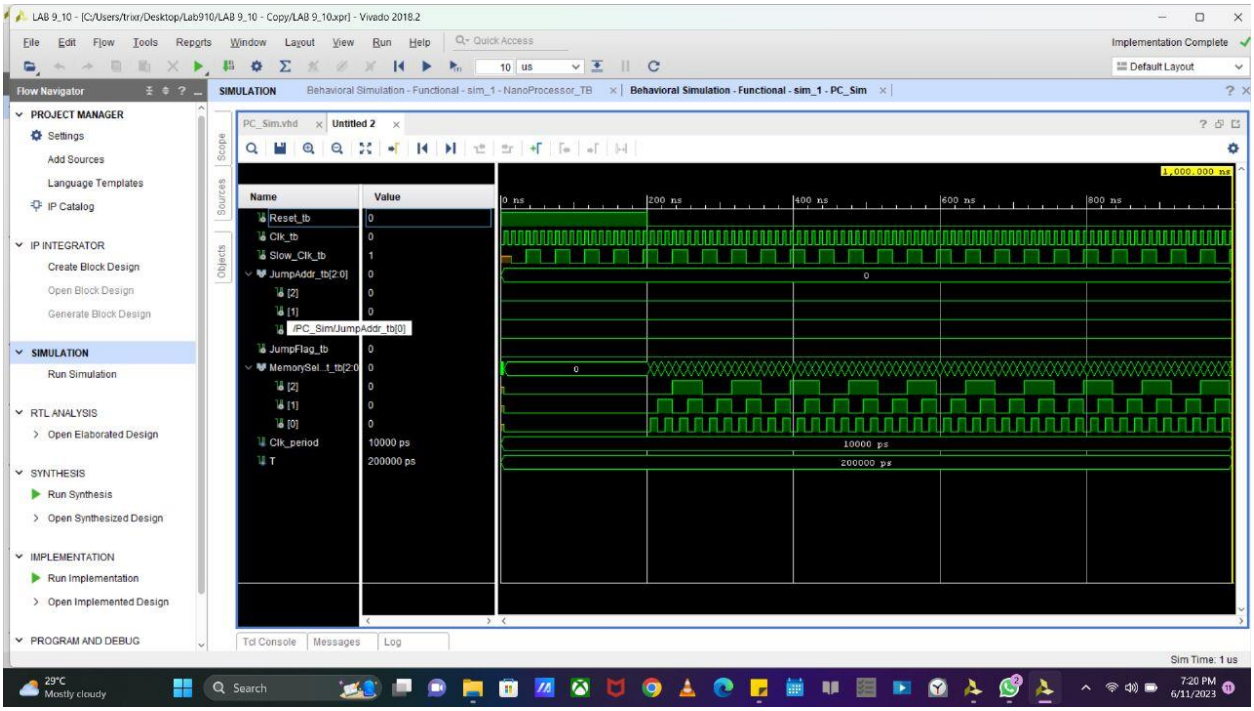
1. NanoProcessor



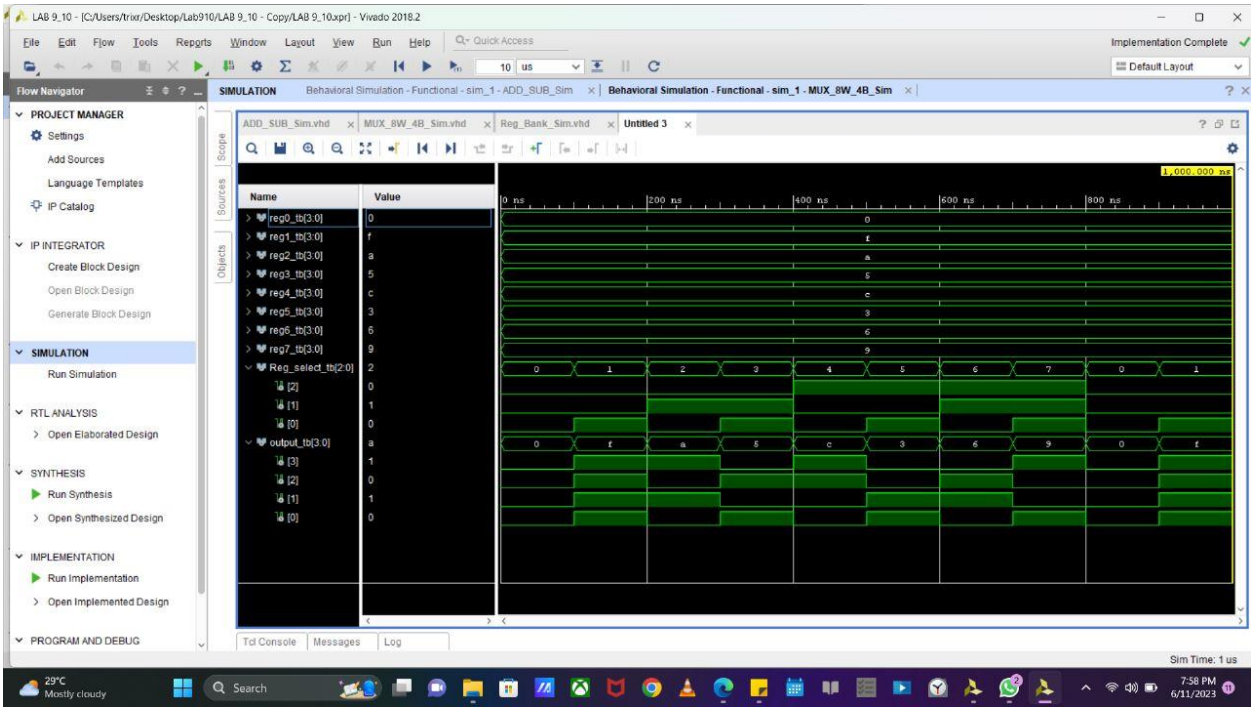
2. AddSub Unit



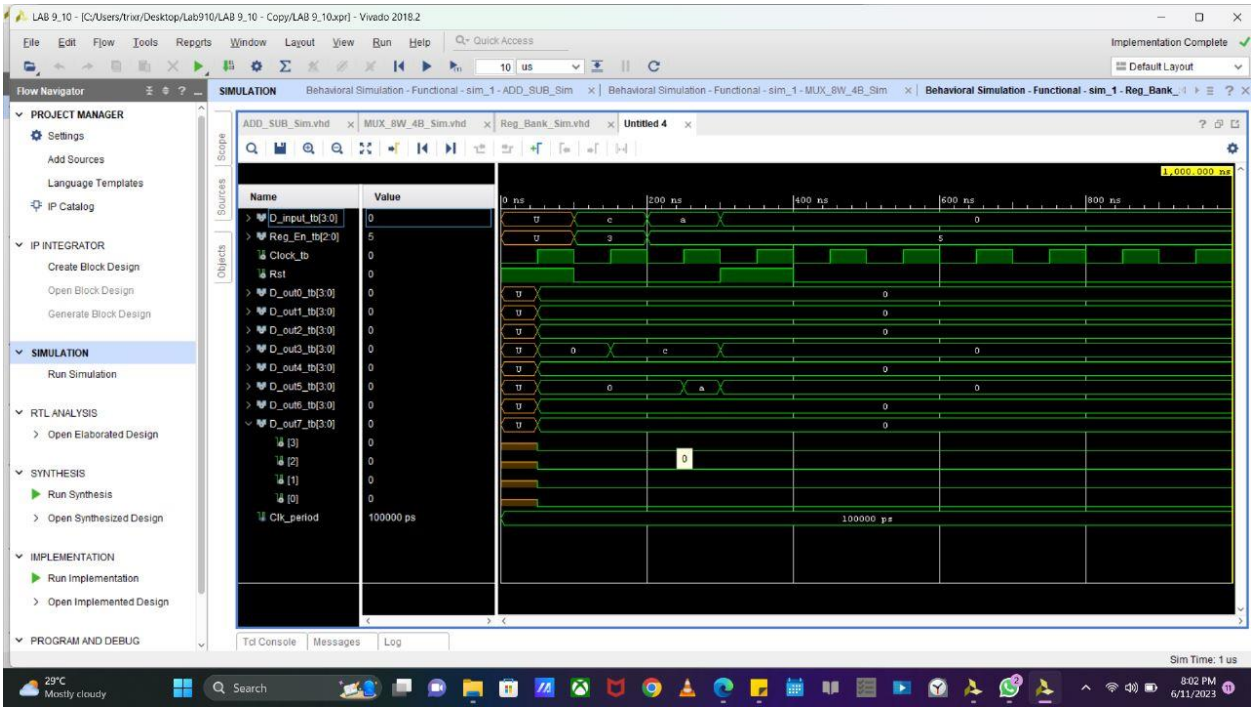
3. Program Counter



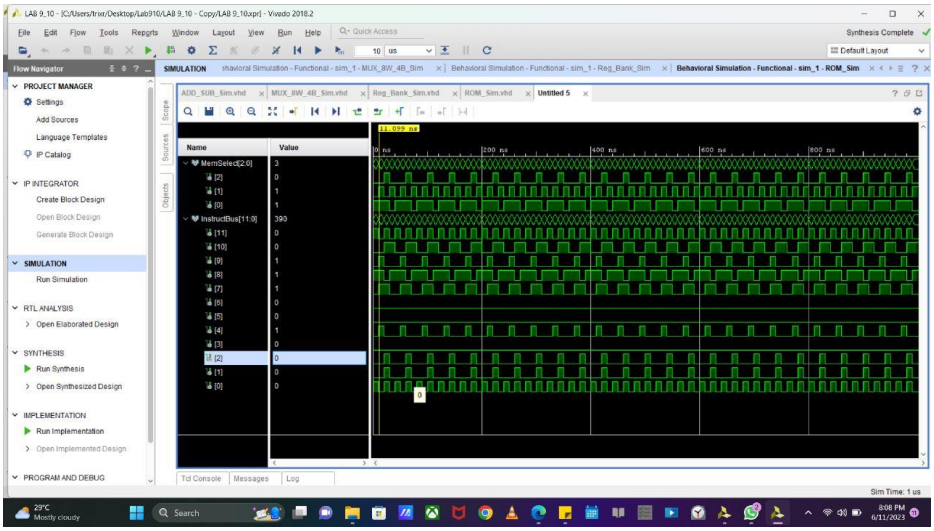
4. Mux 8 Way 4 Bit



5. Register Bank Sim



6. ROM



7. CONCLUSIONS

In this lab, we designed a nanoprocessor which was capable of doing 4 instructions MOVE, ADD, NEGATE & JUMP. For the processor, we had to build basic components including developing arithmetic unit, instruction decoder and multiplexers. By connecting all registers and all other components in their respective ways, we were able to verify the functionality using the Basys3 board as well. Overall the lab helped us learn a lot about planning and doing a project with good teamwork and time management.

8. CONTRIBUTION & TIME MANAGEMENT

InNipun Viraj → Creating basic components including all Muxes, Registers, Main Nanoprocessor file (halfway), TB files and Testing and Debugging. (19 hours)

Thiwanka Roshen → Instruction Decoder, Register Bank, Main Nanoprocessor file (halfway), ADD_SUB, Programmable ROM, 7 Seg Display (23 hours)