Lab 9-10 – Nano processor Design Competition

CS1050 Computer Organization and Digital Design

Group Members: -

- **Guruge S.M.L – 210201F**
- **Gunathunga W.S.D – 210196P**

- **Lab Task**
  - The objective of this task is to design a 4-bit processor capable of executing the following 4 instructions.

    1. MOVI R, d - Move immediate value d to register R.
    2. ADD Ra, Rb - Add values in register Ra & Rb and store the result in Ra.
    3. NEG R - 2's complement of register R.
    4. JZR R, d - Jump if value in register R is 0.

  - To build a certain processor, following components should be designed first.
    1. 4-bit Add/Subtract Unit
    2. 3-bit Adder
    3. 3-bit Program Counter
    4. 2-way 3-bit Multiplexer
    5. 2-way 4-bit Multiplexer
    6. 8-way 4-bit Multiplexer
    7. Register Bank
    8. Program ROM
    9. Instruction Decoder
    10. Slow Clock

  - After Creating above components, we tested their functionalities using simulation codes.
  - Then we connected these components using busses.
  - Finally, we tested their functionalities using BASYS3 board.

- **Assembly program**

```
MOVI R1,1        ; R1 ← 1
MOVI R2,2        ; R2 ←2
MOVI R3,3        ; R3 ←3
MOVI R7,0        ; R7 ←0
ADD R7,R1        ; R7 ←R7 + R1
ADD R7,R2        ; R7 ← R7 + R2
ADD R7,R3        ; R7 ← R7 + R3
JZR R0,7         ; If R0 = 0 jump to line 7
```

- **Machine code**
  - 100010000001
  - 100100000010
  - 100110000011
  - 101110000000
  - 001110010000
  - 001110100000
  - 001110110000
  - 110000000111

- **All VHDL Codes**
  - ❖ Source Codes
    - **4- bit adder subtracter**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity AdderSubtracter is
    Port ( A : in STD_LOGIC_Vector(3 downto 0);
           B : in STD_LOGIC_vector(3 downto 0);
           C_in : in STD_LOGIC_vector(1 downto 0);
           S : out STD_LOGIC_vector(3 downto 0);
           C_out : out STD_LOGIC;
           Zero : out std_logic;
          C_out_plus : out std_logic);
end AdderSubtracter;
architecture Behavioral of AdderSubtracter 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_S, FA0_C, FA1_S, FA1_C, FA2_S, FA2_C, FA3_S, FA3_C , output,
C_tmp : std_logic;
signal B_out,S0 : std_logic_vector(3 downto 0);
begin
B_out(0) <= (C_in(0) and (not C_in(1))) xor B(0);
B_out(1) <= (C_in(0) and (not C_in(1))) xor B(1);
B_out(2) <= (C_in(0) and (not C_in(1))) xor B(2);
B_out(3) <= (C_in(0) and (not C_in(1))) xor B(3);

C_tmp <= (C_in(0) and (not C_in(1)));

 FA_0 : FA
    port map (
        A => A(0),
        B => B_out(0),
        C_in => C_tmp,
        S => S0(0),
        C_Out => FA0_C);
 FA_1 : FA
    port map (
        A => A(1),
        B => B_out(1),
        C_in => FA0_C,
        S => S0(1),
        C_Out => FA1_C);
 FA_2 : FA
     port map (
         A => A(2),
         B => B_out(2),
         C_in => FA1_C,
         S => S0(2),
         C_Out => FA2_C);
 FA_3 : FA
     port map (
         A => A(3),
         B => B_out(3),
         C_in => FA2_C,
         S => S0(3),
         C_Out => output);
C_Out <= C_tmp AND (output xor FA2_C) ;
C_out_plus<= output and not(C_tmp);
Zero <= (NOT(C_in(1) OR C_in(0))) AND (not(S0(0) or S0(1) or S0(2) or S0(3)));
S<= S0;
```

- **3 – bit adder**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity adder_3_bit is
    Port ( A : in STD_LOGIC_Vector(2 downto 0);
           S : out STD_LOGIC_vector(2 downto 0));
end adder_3_bit;

architecture Behavioral of adder_3_bit 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;
begin

 FA_0 : FA
    port map (
        A => A(0),
        B => '1',
        C_in => '0',
        S => S(0),
        C_Out => FA0_C);
 FA_1 : FA
    port map (
        A => A(1),
        B => '0',
        C_in => FA0_C,
        S => S(1),
        C_Out => FA1_C);
 FA_2 : FA
     port map (
         A => A(2),
         B => '0',
         C_in => FA1_C,
         S => S(2),
         C_Out => FA2_C);
         end Behavioral;
```

- **3 – bit program counter**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity program_Counter is
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
           Reset : in STD_LOGIC:='0';
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_vector(2 downto 0):="000");
end program_Counter;

architecture Behavioral of program_Counter is
begin
    process(Clk) begin
    if(rising_edge(Clk)) then
        if Reset = '1' then
            Q <= "000";
        else
            Q <= D;
        end if;
     end if;
 end process;

end Behavioral;
```

- **2 – way 3 – bit Multiplexer**

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

entity MUX_2_way_3_bit is
 Port ( I0 : in STD_LOGIC_VECTOR (2 downto 0);
        I1 : in STD_LOGIC_VECTOR (2 downto 0);
        D  : out STD_LOGIC_VECTOR (2 downto 0);
        S  : in STD_LOGIC);
end MUX_2_way_3_bit;

architecture Behavioral of MUX_2_way_3_bit is
Signal Z1,Z2,Z3,Z4,Z5,Z6 : STD_LOGIC;

begin

Z1 <= (I0(0) AND not S);
Z2 <= (I1(0) AND S);
```

```
Z3 <= (I0(1) AND not S);
Z4 <= (I1(1) AND S);
Z5 <= (I0(2) AND not S);
Z6 <= (I1(2) AND S);

D(0) <= Z1 OR Z2;
D(1) <= Z3 OR Z4;
D(2) <= Z5 OR Z6;

end Behavioral;
```

- **2 – way 4- bit Multiplexer**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity MUX_2_way_4_bit is
   Port ( I0 : in STD_LOGIC_VECTOR (3 downto 0);
          I1 : in STD_LOGIC_VECTOR (3 downto 0);
          D : out STD_LOGIC_VECTOR (3 downto 0);
          S : in STD_LOGIC);
end MUX_2_way_4_bit;

architecture Behavioral of MUX_2_way_4_bit is
Signal Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8 : STD_LOGIC;


begin

Z1 <= (I0(0) AND not S);
Z2 <= (I1(0) AND S);
Z3 <= (I0(1) AND not S);
Z4 <= (I1(1) AND S);
Z5 <= (I0(2) AND not S);
Z6 <= (I1(2) AND S);
Z7 <= (I0(3) AND not S);
Z8 <= (I1(3) AND S);

D(0) <= Z1 OR Z2;
D(1) <= Z3 OR Z4;
D(2) <= Z5 OR Z6;
D(3) <= Z7 OR Z8;


end Behavioral;
```

- **8 – way 4-bit Multiplexer**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity MUX_8_way_4_bit is
   Port ( I0 : in STD_LOGIC_VECTOR (3 downto 0);
          I1 : in STD_LOGIC_VECTOR (3 downto 0);
          I2 : in STD_LOGIC_VECTOR (3 downto 0);
          I3 : in STD_LOGIC_VECTOR (3 downto 0);
          I4 : in STD_LOGIC_VECTOR (3 downto 0);
          I5 : in STD_LOGIC_VECTOR (3 downto 0);
          I6 : in STD_LOGIC_VECTOR (3 downto 0);
          I7 : in STD_LOGIC_VECTOR (3 downto 0);
          D : out STD_LOGIC_VECTOR (3 downto 0);
          S : in STD_LOGIC_VECTOR (2 downto 0);
          En : in std_logic);
end MUX_8_way_4_bit;

architecture Behavioral of MUX_8_way_4_bit is
Component Mux_8_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;

begin
Mux_0:Mux_8_1
port map(
S => S,
D(0) => I0(0),
D(1) => I1(0),
D(2) => I2(0),
D(3) => I3(0),
D(4) => I4(0),
D(5) => I5(0),
D(6) => I6(0),
D(7) => I7(0),
Y => D(0),
EN => En
);
Mux_1:Mux_8_1
port map(
S => S,
D(0) => I0(1),
D(1) => I1(1),
D(2) => I2(1),
D(3) => I3(1),
D(4) => I4(1),
D(5) => I5(1),
D(6) => I6(1),
D(7) => I7(1),
Y => D(1),
EN => En

);
Mux_2:Mux_8_1
port map(
S => S,
D(0) => I0(2),
D(1) => I1(2),
D(2) => I2(2),
D(3) => I3(2),
D(4) => I4(2),
D(5) => I5(2),
D(6) => I6(2),
D(7) => I7(2),
Y => D(2),
EN => En

);
Mux_3:Mux_8_1
port map(
S => S,
D(0) => I0(3),
D(1) => I1(3),
D(2) => I2(3),
D(3) => I3(3),
D(4) => I4(3),
D(5) => I5(3),
D(6) => I6(3),
D(7) => I7(3),
Y => D(3),
EN => En

);

end Behavioral;
```

- **Register Bank**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Register_Bank is
   Port ( Clk : in STD_LOGIC;
          Reg_En : in STD_LOGIC_VECTOR (2 downto 0);
          D : in STD_LOGIC_VECTOR (3 downto 0);
          S_out_0 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_1 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_2 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_3 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_4 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_5 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_6 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_7 : out STD_LOGIC_VECTOR (3 downto 0);
          reset : in std_logic);
end Register_Bank;

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

end component;

signal out_Dec : std_logic_vector(7 downto 0);

begin
   Decoder_3_to_8_0 : Decoder_3_to_8
      port map(
         I => Reg_En ,
         EN => '1',
         Y => out_Dec);
```

```
    Reg_0 :Reg
        port map(
            reset => reset,
            D => D,
            En => out_Dec(0),
            Clk => Clk,
            Q => S_out_0);
Reg_1 :Reg
     port map(
        reset => reset,
        D => D,
        En => out_Dec(1),
        Clk => Clk,
        Q => S_out_1);
Reg_2 :Reg
     port map(
        reset => reset,
        D => D,
        En => out_Dec(2),
        Clk => Clk,
        Q => S_out_2);
Reg_3 :Reg
     port map(
        reset => reset,
        D => D,
        En => out_Dec(3),
        Clk => Clk,
        Q => S_out_3);
Reg_4 :Reg
     port map(
        reset => reset,
        D => D,
        En => out_Dec(4),
        Clk => Clk,
        Q => S_out_4);
Reg_5 :Reg
      port map(
        reset => reset,
        D => D,
        En => out_Dec(5),
        Clk => Clk,
        Q => S_out_5);
Reg_6 :Reg
     port map(
        reset => reset,
        D => D,
        En => out_Dec(6),
        Clk => Clk,
        Q => S_out_6);
Reg_7 :Reg
     port map(
        reset => reset,
        D => D,
        En => out_Dec(7),
        Clk => Clk,
        Q => S_out_7);

end Behavioral;
```

- **Program Rom**

```
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 := (
                "100010000001",  --MOVI R1,1
                "100100000010", --MOVI R2,2
                "100110000011", --MOVI R3,3
                "101110000000", --MOVI R7,0
                "001110010000", --ADD R7,R1
                "001110100000", --ADD R7,R2
                "001110110000", --ADD R7,R3
                "110000000111"  --JZR R0,7

 );
begin
data <= program_ROM(to_integer(unsigned(address)));

end Behavioral;
```

- **Instruction Decoder**

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

entity Instruction_Decoder is
   Port ( data : in STD_LOGIC_VECTOR (11 downto 0);
         Reg_check_jump : in STD_LOGIC_VECTOR(3 downto 0);
         Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
         Reg_Sel_A : out STD_LOGIC_VECTOR (2 downto 0);
         Load_sel : out STD_LOGIC;
         Immediate_val : out STD_LOGIC_VECTOR (3 downto 0);
         Reg_Sel_B : out STD_LOGIC_VECTOR (2 downto 0);
         Add_Sub_Sel : out STD_LOGIC_VECTOR(1 downto 0);
         Jump_Flag : out STD_LOGIC;
         Address_To_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is

begin
   Reg_En <= data(9 downto 7);
   Reg_Sel_A <= data(9 downto 7);
   Reg_Sel_B <=  data(6 downto 4);
   Load_Sel <= data(11)and not(data(10));
   Immediate_val <= data(3 downto 0);
   Add_Sub_Sel <= data(11 downto 10);
   --Jump_Flag <= ((NOT Reg_check_jump(0)) AND (NOT Reg_check_jump(1))
AND (NOT Reg_check_jump(2)) AND (NOT Reg_check_jump(3))) AND data(11)
AND data(10);
   Jump_Flag <= NOT( Reg_check_jump(0) Or  Reg_check_jump(1) OR
Reg_check_jump(2) OR Reg_check_jump(3)) AND data(11) AND data(10);
   Address_To_Jump <= data(2 downto 0);

end Behavioral;
```

- **Slow Clock**

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


end Behavioral;
```

- **Nano Processor**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NanoProcessor is
       port(Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
```

```
          Reg7_Seg : out STD_LOGIC_VECTOR (6 downto 0);
          Zero : out STD_LOGIC;
          Overflow : out STD_LOGIC;
          Reg7_out : out std_logic_vector(3 downto 0);
          Anode : out std_logic_vector(3 downto 0));

end NanoProcessor;

architecture Behavioral of NanoProcessor is
component Slow_clk Port ( Clk_in : in STD_LOGIC;
          Clk_out : out STD_LOGIC);
end component;

component Register_Bank port(
          reset : in std_logic;
          Clk : in STD_LOGIC;
          Reg_En : in STD_LOGIC_VECTOR (2 downto 0);
          D : in STD_LOGIC_VECTOR (3 downto 0);
          S_out_0 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_1 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_2 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_3 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_4 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_5 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_6 : out STD_LOGIC_VECTOR (3 downto 0);
          S_out_7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component MUX_2_way_3_bit Port(
          I0 : in STD_LOGIC_VECTOR (2 downto 0);
          I1 : in STD_LOGIC_VECTOR (2 downto 0);
          S : in STD_LOGIC;
          D : out STD_LOGIC_VECTOR (2 downto 0));
end component;
component Instruction_Decoder
 Port ( data : in STD_LOGIC_VECTOR (11 downto 0);
         Reg_check_jump : in STD_LOGIC_VECTOR(3 downto 0);
         Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
         Reg_Sel_A : out STD_LOGIC_VECTOR (2 downto 0);
         Load_sel : out STD_LOGIC;
         Immediate_val : out STD_LOGIC_VECTOR (3 downto 0);
         Reg_Sel_B : out STD_LOGIC_VECTOR (2 downto 0);
         Add_Sub_Sel : out STD_LOGIC_VECTOR(1 downto 0);
         Jump_Flag : out STD_LOGIC;
```

```
          Address_To_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component MUX_2_way_4_bit Port(
          I0 : in STD_LOGIC_VECTOR (3 downto 0);
          I1 : in STD_LOGIC_VECTOR (3 downto 0);
          S : in STD_LOGIC;
          D : out STD_LOGIC_VECTOR (3 downto 0));
end component;
component Mux_8_Way_4_Bit port(
          I0 : in STD_LOGIC_VECTOR (3 downto 0);
          I1 : in STD_LOGIC_VECTOR (3 downto 0);
          I2 : in STD_LOGIC_VECTOR (3 downto 0);
          I3 : in STD_LOGIC_VECTOR (3 downto 0);
          I4 : in STD_LOGIC_VECTOR (3 downto 0);
          I5 : in STD_LOGIC_VECTOR (3 downto 0);
          I6 : in STD_LOGIC_VECTOR (3 downto 0);
          I7 : in STD_LOGIC_VECTOR (3 downto 0);
          D : out STD_LOGIC_VECTOR (3 downto 0);
          S : in STD_LOGIC_VECTOR (2 downto 0);
          En : in std_logic);
end component;
component ROM
Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
       data : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component adder_3_bit port(
         A : in STD_LOGIC_VECTOR (2 downto 0);
         S : out STD_LOGIC_VECTOR (2 downto 0)
);
end component;

component AdderSubtracter
Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          C_in : in STD_LOGIC_VECTOR(1 downto 0);
          S : out STD_LOGIC_VECTOR (3 downto 0);
          C_out : out STD_LOGIC;
          Zero : out Std_logic;
         C_out_plus : out std_logic);

end component;
component program_Counter
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
         Q : out STD_LOGIC_VECTOR (2 downto 0);
```

```
          Clk : in STD_LOGIC;
          Reset : in STD_LOGIC);

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;

signal
Reg_En0,Mux0_Out,Mux0_I0,Mux0_I1,Reg_Sel_Out_A,Reg_Sel_Out_B,Mem_Se
l : STD_LOGIC_VECTOR (2 downto 0);
signal
Reg_In,Reg_Out_0,Reg_Out_1,Reg_Out_2,Reg_Out_3,Reg_Out_4,Reg_Out_5,Re
g_Out_6,Reg_Out_7,Reg_Check_In,Im_Val, AddSubOut,Mux_B_Out:
STD_LOGIC_VECTOR (3 downto 0);
signal I_Dec_In : std_logic_vector(11 downto 0);
signal Mux0_Sel,I_Load_sel,slowClk, C_out_plus : std_logic;
signal AddSubSel : std_logic_vector(1 downto 0);

begin
    Anode <= "1110";
    Slow_Clock : Slow_clk

    Port map( Clk_in => Clk,
            Clk_out => slowClk );

    RegisterBank : Register_Bank
        port map(
            reset => Reset,
            Clk => slowClk,
            Reg_En => Reg_En0,
            D => Reg_In,
            S_out_0 => Reg_Out_0,
            S_out_1 => Reg_Out_1,
            S_out_2 => Reg_Out_2,
            S_out_3 => Reg_Out_3,
            S_out_4 => Reg_Out_4,
            S_out_5 => Reg_Out_5,
            S_out_6 => Reg_Out_6,
            S_out_7 => Reg_Out_7
    );
```

```
Mux_2_way_3_bit0 : MUX_2_way_3_bit
port map( I0 => Mux0_I0,
        I1 => Mux0_I1,
        S => Mux0_Sel,
        D => Mux0_Out
        );
InstructionDecorder : Instruction_Decoder
         Port map(data => I_Dec_In,
                Reg_check_jump => Reg_Check_In,
                Reg_En => Reg_En0,
                Reg_Sel_A => Reg_Sel_Out_A,
                Load_sel => I_Load_sel,
                Immediate_val => Im_Val,
                Reg_Sel_B => Reg_Sel_Out_B,
                Add_Sub_Sel => AddSubSel,
                Jump_Flag => Mux0_Sel,
                Address_To_Jump => Mux0_I1);

Mux2Way4Bit :  MUX_2_way_4_bit Port map(
                     I0 => AddSubOut,
                     I1 => Im_Val,
                     S => I_Load_sel,
                     D =>Reg_In);

Mux8Way4Bit_1 : Mux_8_Way_4_Bit port map(
        I0 => Reg_Out_0,
        I1 => Reg_Out_1,
        I2 => Reg_Out_2,
        I3 => Reg_Out_3,
        I4 => Reg_Out_4,
        I5 => Reg_Out_5,
        I6 => Reg_Out_6,
        I7 => Reg_Out_7,
        D => Reg_Check_In,
        S => Reg_Sel_Out_A,
        En => '1');

Mux8Way4Bit_2 : Mux_8_Way_4_Bit port map(
        I0 => Reg_Out_0,
        I1 => Reg_Out_1,
        I2 => Reg_Out_2,
        I3 => Reg_Out_3,
        I4 => Reg_Out_4,
        I5 => Reg_Out_5,
          I6 => Reg_Out_6,
          I7 => Reg_Out_7,
          D => Mux_B_Out,
          S => Reg_Sel_Out_B,
          En => '1');

 ROM0 : ROM
         Port map ( address => Mem_Sel,
               data => I_Dec_In);

Adder: adder_3_bit
          port map(
             A => Mem_Sel,
              S => Mux0_I0 );

 Adder_Sub : AdderSubtracter
    Port map ( A => Mux_B_Out,
          B => Reg_Check_In,
          C_in => AddSubSel,
          S => AddSubOut,
          C_out => Overflow,
          Zero => Zero);

 Program_Counter0 : program_Counter
             Port map( D => Mux0_Out,
                  Q => Mem_Sel,
                  Clk => slowClk,
                  Reset => Reset);

S_Seg_Out : LUT_16_7
            port map( address => Reg_Out_7,
                     data => Reg7_Seg);
Reg7_out <= Reg_Out_7;

end Behavioral;
```

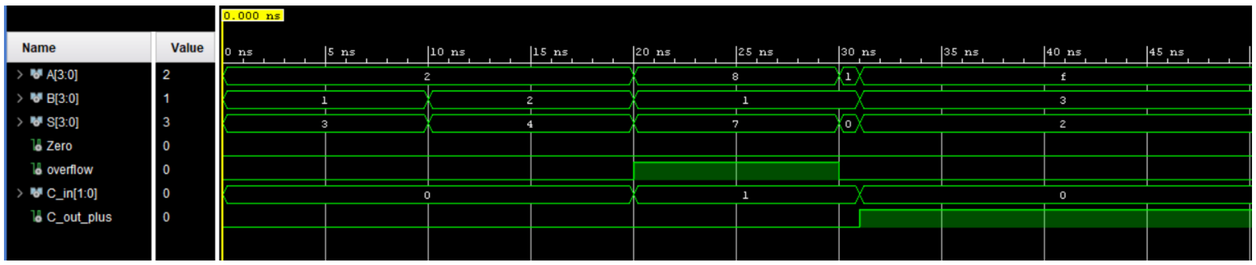
❖ **Simulation Codes**

You can get the respective simulation codes by following hyperlinks.

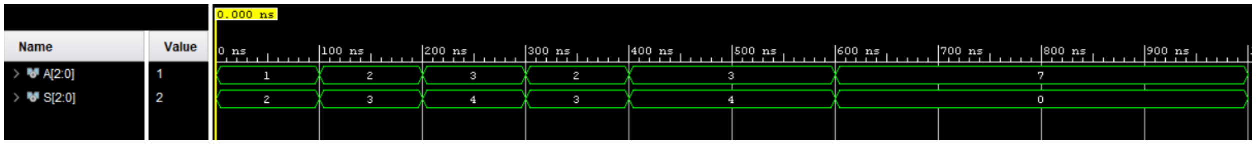
- adder_Subtractor_TB
- 3 - bit adder_TB
- 3 - bit program counter_TB
- 2-way 3-bit mux_TB
- 2-way 4-bit Mux_TB
- 8-way 4-bit Mux_TB
- RegisterBank_TB
- ROM_TB
- Instruction Decorder_TB
- NanoProcessor_TB

➢ Timing Diagrams

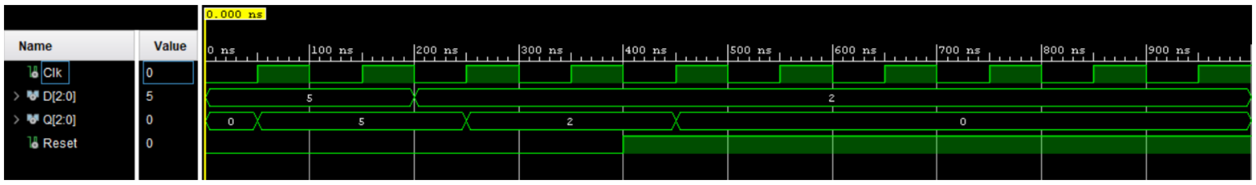
- **Adder_Subtractor**



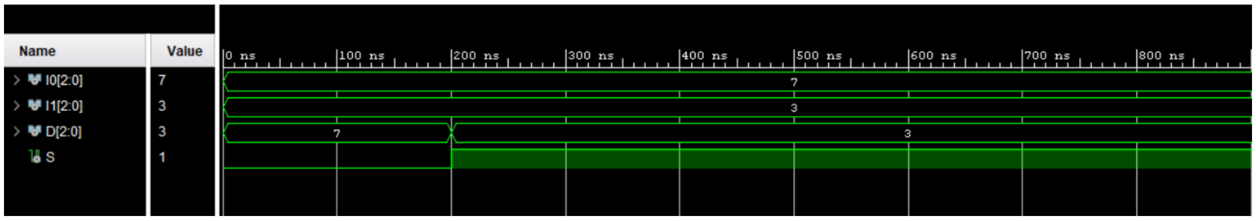
- **3-bit Adder**



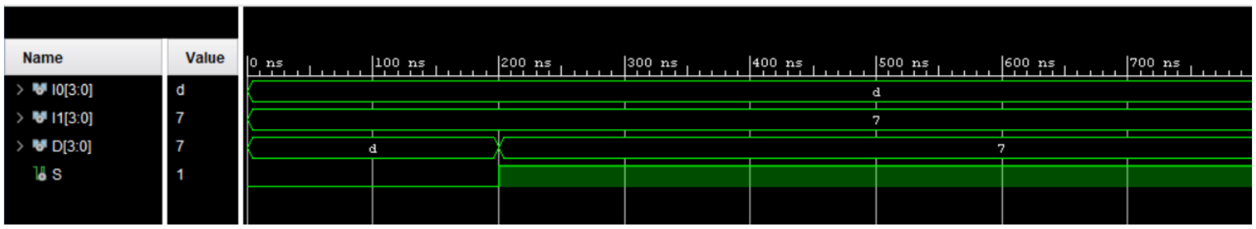
- **3- bit program counter**



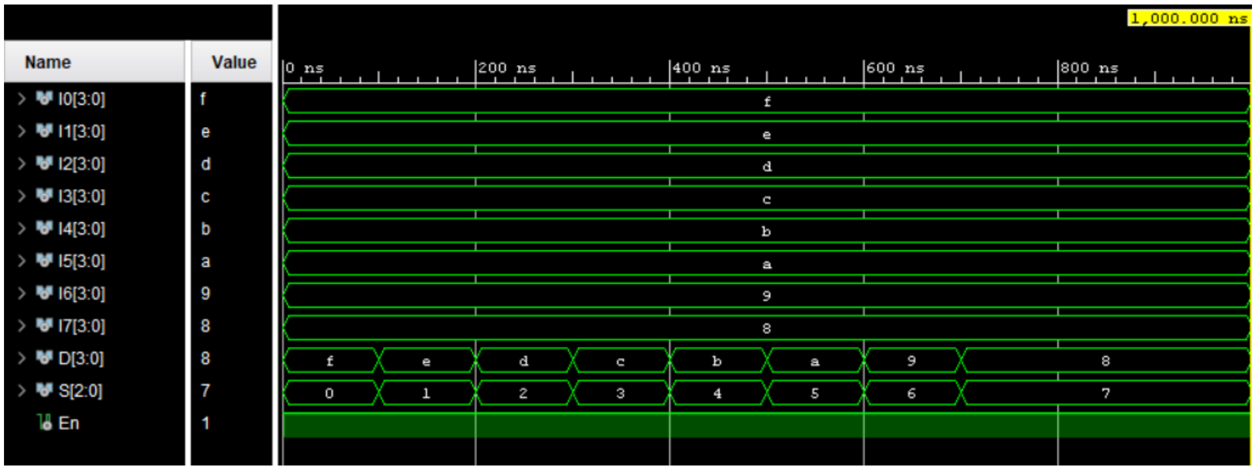
- **2-way 3-bit Multiplexer**



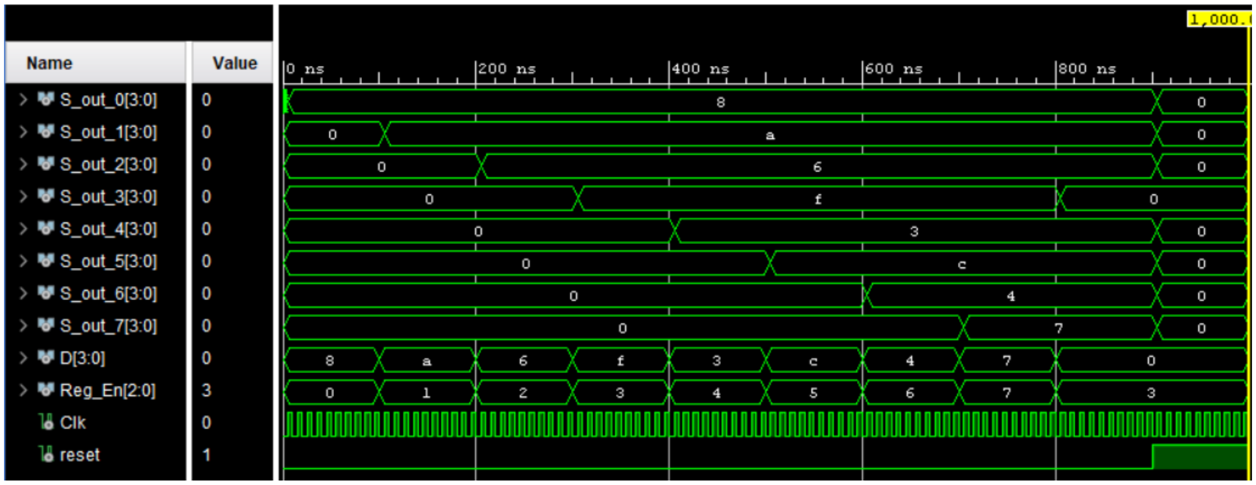
- **2 - way 4- bit Multiplexer**



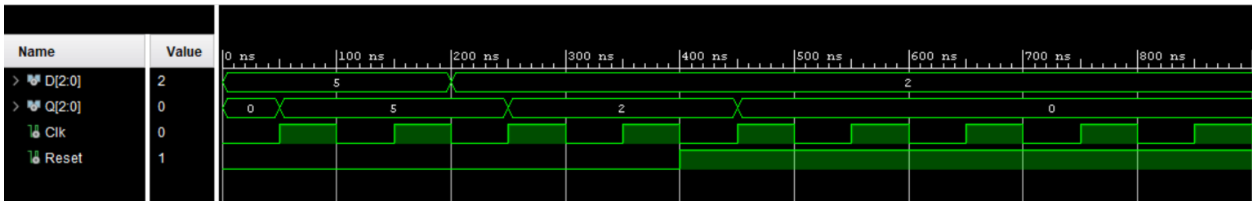
- **8 – way 4- bit Multiplexer**



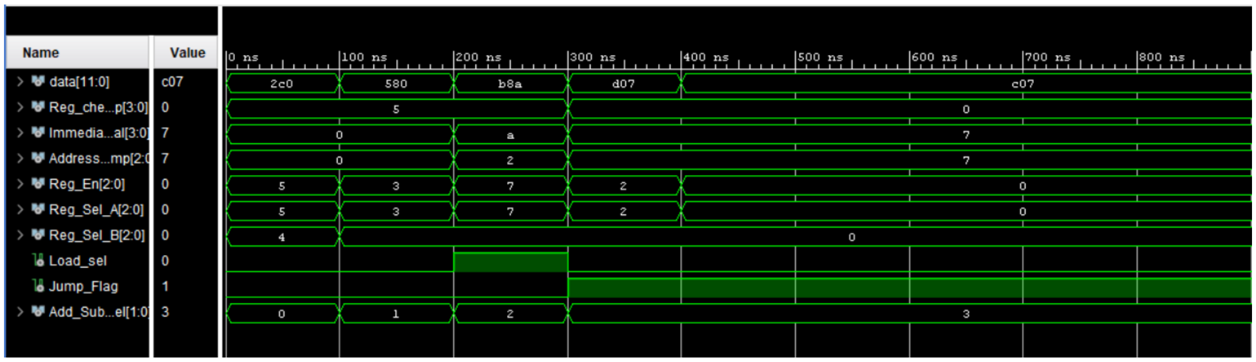
- **Register Bank**



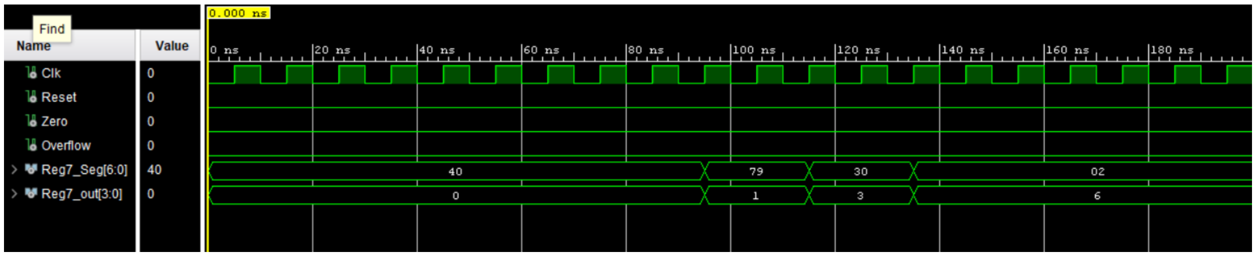
- **Program Counter**



- **Instruction Decoder**



- **Nano Processor**



➢ Conclusion

- Using buses instead of many parallel wires simplifies the design process for a microprocessor in VHDL.
- Simulating and testing each component is critical to ensure the reliability and correctness of the final product.
- Building a microprocessor in VHDL provides an opportunity to gain a deeper understanding of the internal structure and operation of such devices.

- The project can help develop valuable teamwork skills as multiple individuals work together to create a complex system.
- The project requires hardcoding assembly instructions as binary values into ROM since microprocessors only understand machine language.
- From this project we learn how to debug, test, and optimize their circuit designs using simulation tools and on the development board.

➢ Contribution Of Each Member

Name	Designing Parts	No. Of Hours Spend
Guruge S.M.L	Instruction Decoder 8-way 4-bit Multiplexer 4-bit Add/Subtract Unit Program ROM Register Bank	12 Hours
Gunathunga W.S.D	2-way 3-bit Multiplexer 2-way 4-bit Multiplexer 3-bit Adder 3-bit Program Counter Clock Controller	12 Hours