Lab 9-10 – Nano processor Design Competition

CS1050 Computer Organization and Digital Design

Members.

- K.M.L.I Kulathunga – 210307K
- K.D.R.P Rathnayake – 210537N

Lab Tasks.

- We have to create a simple microprocessor capable of executing a simple set of instructions.
- To do that we have to create these components,
  - ❖ 4-bit Add/Subtract unit
  - ❖ 3-bit adder
  - ❖ 3-bit program counter
  - ❖ k-way b-bit multiplexers
  - ❖ Register bank
  - ❖ Program Rom
  - ❖ Instruction decorder
- And also, we have to test these each component using simulation.
- We have to hardcode assembly program into Rom that we created.
- After that, we have to map each other and test it using simulation.
- Finally, we have to do port mapping accordingly and test the nano processor in BASYS 3 board.

Assembly program and its machine code representation.

Assembly Code	Binary Representation
MOVI R_6 , 3	10 110 000 0011
MOVI R_5 , 1	10 101 000 0001
NEG R_5	01 101 000 0000
ADD R_7 , R_6	00 111 110 0000
ADD R_6 , R_5	00 110 101 0000
JZR R_6 , 7	11 110 000 0111
JZR R_0 , 3	11 000 000 0011
JZR R_0 , 7	11 000 000 0111

All VHDL Codes.

01. Nano processor

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

entity NP is
    Port ( clock : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Zero : OUT STD_LOGIC;
           Carry : OUT STD_LOGIC;
           LED_out: OUT STD_LOGIC_VECTOR(3 downto 0);
           Seg_display: OUT STD_LOGIC_VECTOR(6 downto 0);
           anode : OUT STD_LOGIC_VECTOR(3 downto 0)
           );
end NP;

architecture Behavioral of NP is

component Multiplexer_8way_4bit
   Port ( input_0 : in  std_logic_vector(3 downto 0);
           input_1 : in  std_logic_vector(3 downto 0);
           input_2 : in  std_logic_vector(3 downto 0);
           input_3 : in  std_logic_vector(3 downto 0);
           input_4 : in  std_logic_vector(3 downto 0);
           input_5 : in  std_logic_vector(3 downto 0);
           input_6 : in  std_logic_vector(3 downto 0);
           input_7 : in  std_logic_vector(3 downto 0);
           selector  : in  std_logic_vector(2 downto 0);
           enable  : in  std_logic;
           output  : out std_logic_vector(3 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));
end component;

component Multiplexer_2way_4bit
    Port ( input_0 : in  std_logic_vector(3 downto 0);
           input_1 : in  std_logic_vector(3 downto 0);
           selector  : in  std_logic;
           enable  : in  std_logic;
           output  : out std_logic_vector(3 downto 0));
end component;

component Multiplexer_2way_3bit
  Port ( input_0 : in  std_logic_vector(2 downto 0);
           input_1 : in  std_logic_vector(2 downto 0);
           selector  : in  std_logic;
           enable  : in  std_logic;
           output  : out std_logic_vector(2 downto 0));
end component;

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;
           S : out STD_LOGIC_vector(2 downto 0);
           C_out : out STD_LOGIC);
end component;

component TO_7_seg
   Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end component;

component ROM
   Port ( selector : in STD_LOGIC_VECTOR (2 downto 0);
           instruction : out STD_LOGIC_VECTOR (11 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;

component Instruction_Decoder
Port ( instruction : in STD_LOGIC_VECTOR (11 downto 0);
       Reg_en : out STD_LOGIC_VECTOR (2 downto 0);
       Reg_sel_A : out STD_LOGIC_VECTOR (2 downto 0);
       Reg_sel_B : out STD_LOGIC_VECTOR (2 downto 0);
       Load_sel : out STD_LOGIC;
       Add_Sub : out STD_LOGIC;
       Reg_check : in STD_LOGIC_VECTOR (3 downto 0);
       Jump : out STD_LOGIC;
       Jump_to : out STD_LOGIC_VECTOR (2 downto 0);
       Load : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component RCA_4
   Port ( op : in std_logic;
        A : in STD_LOGIC_VECTOR (3 downto 0);
        B:  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;
        zero : out std_logic);
end component;

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

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


component PC
   Port ( Count : in STD_LOGIC_VECTOR (2 downto 0);
          Clock : in STD_LOGIC;
          Reset : in STD_LOGIC;
          PC_out : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal Slow_Clock_signal,Jump,load_sel,operator,RCA_3_carry,Delayed_Clock_signal :
STD_LOGIC;
signal Memory_select,Jump_to,PC_out,Reg_A,Reg_B,RCA_3bit_out
,Reg_sel,PC_count : STD_LOGIC_VECTOR(2 downto 0);
signal
Load,RCA_4bit_out,Reg_1,Reg_2,Reg_3,Reg_4,Reg_5,Reg_6,Reg_7,Reg_0,Reg_A_out,
Reg_B_out,To_registers : STD_LOGIC_VECTOR(3 downto 0);
signal Instruction : STD_LOGIC_VECTOR(11 downto 0);
signal Decoder_out : STD_LOGIC_VECTOR(7 downto 0);

begin

Slow_clock_01 : Slow_Clk
port map(
 Clk_in => clock ,
 Clk_out => Slow_Clock_signal
);

Slow_clock_02 : Delayed_clock
port map(
 Clk_in => clock ,
 Clk_out => Delayed_Clock_signal
);

PC_01 : PC
port map(
Count =>PC_count,
Clock =>Slow_Clock_signal,
Reset => Reset,
PC_out => Memory_select
);

Multiplexer_8way_4bit_A :Multiplexer_8way_4bit
port map(
        input_0 =>Reg_0,
        input_1 =>Reg_1,
        input_2 =>Reg_2,
        input_3=>Reg_3,
        input_4 =>Reg_4,
        input_5 =>Reg_5,
        input_6  =>Reg_6,
        input_7 =>Reg_7,
        selector =>Reg_A,
        enable  =>'1',
        output => Reg_A_out
);
Multiplexer_8way_4bit_B :Multiplexer_8way_4bit
port map(
        input_0 =>Reg_0,
        input_1 =>Reg_1,
        input_2 =>Reg_2,
        input_3=>Reg_3,
        input_4 =>Reg_4,
        input_5 =>Reg_5,
        input_6  =>Reg_6,
        input_7 =>Reg_7,
        selector =>Reg_B,
        enable  =>'1',
        output => Reg_B_out
```

```
);

ROM_01 : ROM
port map(
   selector => Memory_select,
   instruction => instruction
);

Instruction_Decoder_01 : Instruction_Decoder
Port map( instruction => instruction,
     Reg_en      =>  Reg_sel,
     Reg_sel_A    =>  Reg_A,
     Reg_sel_B    =>  Reg_B,
     Load_sel     =>  Load_sel,
     Add_Sub      =>  Operator,
     Reg_check    =>  Reg_A_out,
     Jump         =>  Jump,
     Jump_to      =>  Jump_to,
     Load         =>  Load);

RCA_4_01 : RCA_4
Port map(
   op      =>  operator,
   A       =>  Reg_A_out,
   B       =>  Reg_B_out,
   C_in    =>  '0',
   S       =>  RCA_4bit_out,
   C_out   => Carry,
   Zero    => Zero
);
RCA_3_01 : RCA_3
Port map(
   A       =>  Memory_select,
   B       =>  "001",
   C_in    =>  '0',
   S       =>  RCA_3bit_out,
   C_out   =>  RCA_3_carry
);

Decoder_01 : Decoder_3_to_8
port map(
   I       =>  Reg_sel,
   Y       =>  Decoder_out
);
```

```
-----------------------------------Register bank---------------------------------------------------
Register_00 : Reg
port map(
   D      =>  To_registers,
   En     =>  Decoder_out(0),
   Clk    =>  Delayed_Clock_signal,
   Q      =>  Reg_0
);
Register_01 : Reg
port map(
   D      =>  To_registers,
   En     =>  Decoder_out(1),
   Clk    =>  Delayed_Clock_signal,
   Q      =>  Reg_1
);
Register_02 : Reg
port map(
   D      =>  To_registers,
   En     =>  Decoder_out(2),
   Clk    =>  Delayed_Clock_signal,
   Q      =>  Reg_2
);
Register_03 : Reg
port map(
   D      =>  To_registers,
   En     =>  Decoder_out(3),
   Clk    =>  Delayed_Clock_signal,
   Q      =>  Reg_3
);
Register_04 : Reg
port map(
   D      =>  To_registers,
   En     =>  Decoder_out(4),
   Clk    =>  Delayed_Clock_signal,
   Q      =>  Reg_4
);
Register_05 : Reg
port map(
   D      =>  To_registers,
   En     =>  Decoder_out(5),
   Clk    =>  Delayed_Clock_signal,
   Q      =>  Reg_5
);
Register_06 : Reg
port map(
            D      =>  To_registers,
            En      =>  Decoder_out(6),
            Clk     =>  Delayed_Clock_signal,
            Q      =>  Reg_6
          );
          Register_07 : Reg
          port map(
            D      =>  To_registers,
            En      =>  Decoder_out(7),
            Clk     =>  Delayed_Clock_signal,
            Q      =>  Reg_7
          );
          -------------------------------------------------------------------------------------------------------------------
          ----
          Multiplexer_2way_4bit_01 :   Multiplexer_2way_4bit
          port map(
            input_0    =>  Load,
            input_1    =>  RCA_4bit_out,
            selector   =>  Load_sel,
            enable     =>  '1',
            output     =>  To_registers
          );
          Multiplexer_2way_3bit_01 :   Multiplexer_2way_3bit
          port map(
            input_0    =>  RCA_3bit_out,
            input_1    =>  Jump_to,
            selector   =>  jump,
            enable     =>  '1',
            output     =>  PC_count
          );

          Display_01 :   To_7_seg
          port map(
            address =>  Reg_7,
            data    =>  Seg_display
          );

          anode <="1110";
          Led_out <= Reg_7;
end Behavioral;
```

02. 4-bit Add/Subtract unit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity RCA_4 is
 Port ( op : in std_logic;
        A : in STD_LOGIC_VECTOR (3 downto 0);
        B:  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;
        zero : out std_logic);
end RCA_4;
architecture Behavioral of RCA_4 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 : std_logic;
 signal S_RCA ,B_RCA : STD_LOGIC_VECTOR (3 downto 0);

begin
 B_RCA(0)<=B(0)XOR op;
 B_RCA(1)<=B(1)XOR op;
 B_RCA(2)<=B(2)XOR op;
 B_RCA(3)<=B(3)XOR op;

 FA_0 : FA
 port map (
    A => A(0),
    B => B_RCA(0),
    C_in => op, -- Set to ground
    S => S_RCA(0),
    C_out => FA0_C);
 FA_1 : FA
 port map (
    A => A(1),
    B => B_RCA(1),
    C_in => FA0_C,
              S => S_RCA(1),
              C_out => FA1_C);

            FA_2 : FA
            port map (
              A => A(2),
              B => B_RCA(2),
              C_in => FA1_C,
              S => S_RCA(2),
              C_out => FA2_C);

            FA_3 : FA
            port map (
              A => A(3),
              B => B_RCA(3),
              C_in => FA2_C,
              S => S_RCA(3),
              C_out => FA3_C);
              C_out<= FA3_C;
              S<=S_RCA;
              Zero <= not(S_RCA(0))and not(S_RCA(1))and not(S_RCA(2))and not(S_RCA(3)) and
           not(FA3_C NAND op) ;
end Behavioral;
```

03. 3-bit adder.

```
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;
         S : out STD_LOGIC_vector(2 downto 0);
         C_out : out STD_LOGIC);
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_S, FA0_C, FA1_S, FA1_C, FA2_S, FA2_C : std_logic;

        begin

          FA_0 : FA
            port map (
              A => A(0),
              B => B(0),
              C_in => '0', -- Set to ground
              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;
```

04. 3-bit program counter

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

        entity PC is
          Port ( Count : in STD_LOGIC_VECTOR ( 2 downto 0);
                 Clock : in STD_LOGIC;
                 Reset : in STD_LOGIC;
                 PC_out : out STD_LOGIC_VECTOR (2 downto 0));
```

```
            end PC;

            architecture Behavioral of PC is
            component D_FF
            port (
            D : in STD_LOGIC;
            Res: in STD_LOGIC;
            Clk : in STD_LOGIC;
            Q : out STD_LOGIC;
            Qbar : out STD_LOGIC);

            end component;
            begin
            D_FF0 : D_FF
            port map (
               D => Count(0),
               Res => Reset,
               Clk => Clock,
               Q => PC_out(0));
            D_FF1 : D_FF
             port map (
                D => Count(1),
                Res => Reset,
                Clk => Clock,
                Q => PC_out(1));
             D_FF2 : D_FF
              port map (
                D => Count(2),
                Res => Reset,
                Clk => Clock,
                Q => PC_out(2));
end Behavioral;
```

05. 2-way 3-bit multiplexer

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

            entity Multiplexer_2way is
               Port ( input_0 : in  std_logic_vector(2 downto 0);
                     input_1 : in  std_logic_vector(2 downto 0);
                     selector  : in  std_logic;
                     enable  : in  std_logic;
                     output  : out std_logic_vector(2 downto 0));
```

```
        end Multiplexer_2way;

        architecture Behavioral of Multiplexer_2way is
        begin
          process(input_0, input_1, selector, enable)
          begin
            if enable = '1' then
              if selector = '0' then
                output <= input_0;
              else
                output <= input_1;
              end if;
            else
              output <= (others => '0');
            end if;
          end process;
end Behavioral;
```

06. 2-way 4-bit multiplexer

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

        entity Multiplexer_2way_4bit is
          Port ( input_0 : in  std_logic_vector(3 downto 0);
                 input_1 : in  std_logic_vector(3 downto 0);
                 selector  : in  std_logic;
                 enable  : in  std_logic;
                 output  : out std_logic_vector(3 downto 0));
        end Multiplexer_2way_4bit;

        architecture Behavioral of Multiplexer_2way_4bit is
        begin
          process(input_0, input_1, selector, enable)
          begin
            if enable = '1' then
              if selector = '0' then
                output <= input_0;
              else
                output <= input_1;
              end if;
            else
              output <= (others => '0');
            end if;
        end process;
end Behavioral;
```

07. 8-way 4-bit multiplexer

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

entity Multiplexer_8way_4bit is
   Port ( input_0 : in  std_logic_vector(3 downto 0);
          input_1 : in  std_logic_vector(3 downto 0);
          input_2 : in  std_logic_vector(3 downto 0);
          input_3 : in  std_logic_vector(3 downto 0);
          input_4 : in  std_logic_vector(3 downto 0);
          input_5 : in  std_logic_vector(3 downto 0);
          input_6 : in  std_logic_vector(3 downto 0);
          input_7 : in  std_logic_vector(3 downto 0);
          selector  : in  std_logic_vector(2 downto 0);
          enable  : in  std_logic;
          output  : out std_logic_vector(3 downto 0));
end Multiplexer_8way_4bit;

architecture Behavioral of Multiplexer_8way_4bit is
begin
   process(input_0, input_1, input_2, input_3, input_4, input_5, input_6, input_7,
selector, enable)
   begin
      if enable = '1' then
         case selector is
            when "000" =>
               output <= input_0;
            when "001" =>
               output <= input_1;
            when "010" =>
               output <= input_2;
            when "011" =>
               output <= input_3;
            when "100" =>
               output <= input_4;
            when "101" =>
               output <= input_5;
            when "110" =>
```

```
                          output <= input_6;
                       when "111" =>
                          output <= input_7;
                       when others =>
                          output <= (others => '0');
                    end case;
                 else
                    output <= (others => '0');
                 end if;
              end process;
end Behavioral;
```

08. 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;
                   Clk : in STD_LOGIC;
                   Q : out STD_LOGIC_VECTOR (3 downto 0) :=(others =>’0’));
           end Reg;

           architecture Behavioral of Reg is

           begin
           process (Clk) begin
            if (Rising_edge(Clk)) then -- respond when clock rises
            if En = '1' then -- Enable should be set
            Q <= D;
            end if;
            end if;
           end process;

end Behavioral;
```

09. Program ROM

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

```
        entity ROM is
          Port ( selector : in STD_LOGIC_VECTOR (2 downto 0);
               instruction : out STD_LOGIC_VECTOR (11 downto 0));
        end ROM;

        architecture Behavioral of ROM is
        component LUT_16_7
        Port ( address: in STD_LOGIC_VECTOR (2 downto 0);
         data : out STD_LOGIC_VECTOR (11 downto 0));
        end component;
        begin
        LUT_01 : LUT_16_7
         PORT MAP(
          address =>selector,
          data =>instruction
          );

end Behavioral;
```

10. Instruction decoder

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


        entity Instruction_Decoder is
          Port ( instruction : in STD_LOGIC_VECTOR (11 downto 0);
               Reg_en : out STD_LOGIC_VECTOR (2 downto 0);
               Reg_sel_A : out STD_LOGIC_VECTOR (2 downto 0);
               Reg_sel_B : out STD_LOGIC_VECTOR (2 downto 0);
               Load_sel : out STD_LOGIC;
               Add_Sub : out STD_LOGIC;
               Reg_check : in STD_LOGIC_VECTOR (3 downto 0);
               Jump : out STD_LOGIC;
               Jump_to : out STD_LOGIC_VECTOR (2 downto 0);
               Load : out STD_LOGIC_VECTOR (3 downto 0));
        end Instruction_Decoder;

        architecture Behavioral of Instruction_Decoder is

        begin
        process (instruction,Reg_check)
          begin
            case instruction(11 downto 10) is
```

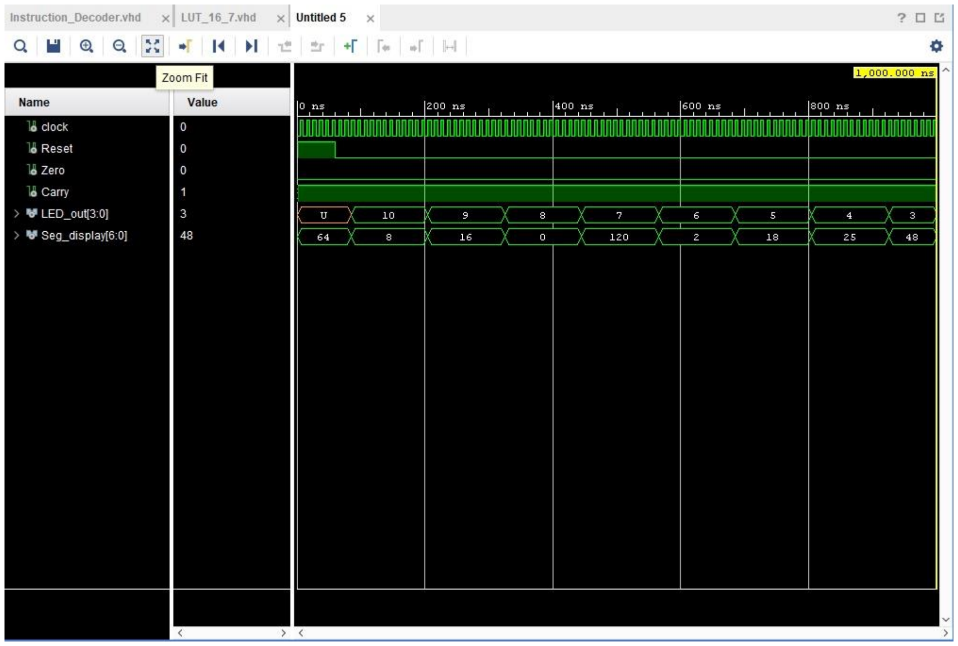
```
                    when "10" =>       -- WRITE
                       Load<= instruction(3 downto 0)  ;
                       Load_sel<= '0';
                       Reg_en<=instruction(9 downto 7);
                        Jump <='0';
                       jump_to <= "000";
                       Reg_sel_A<= "000";
                       Reg_sel_B<= "000";
                       Add_Sub <='0';

                     when "00" =>      -- ADD
                        Reg_sel_A<=instruction(9 downto 7);
                        Reg_sel_B<=instruction(6 downto 4);
                        Add_Sub<='0';
                        Load_sel<='1';
                        Reg_en<=instruction(9 downto 7);
                         Jump <='0';
                         jump_to <= "000";
                         Load <= "0000";
                     when "01" => --Neg
                        Reg_sel_A <="000";
                        Reg_sel_B <=instruction(9 downto 7);
                        Add_Sub<='1';
                        Load_sel<='1';
                        Reg_en<=instruction(9 downto 7);
                        Jump_to <= "000";
                         Jump <='0';
                        Load<="0000";
                     when "11" =>       --Jump
                        Reg_Sel_A <=instruction(9 downto 7);
                          if Reg_check="0000" then
                             Jump_to <= instruction(2 downto 0)  ;
                             Jump <='1';
                          else
                           Jump_to <= "000";
                           Jump <='0';
                          end if;
                          Reg_Sel_B <= "000";
                          Load_sel<='0';
                          Load <= "0000";
                          Reg_en <= "000";
                          Add_sub <= '0';
                      when others =>
                         Load<= "0000"  ;
                         Load_sel<= '0';
                    Reg_en<="000";
                    Reg_sel_A <= "000";
                    Reg_sel_B <= "000";
                    Jump_to <= "000";
                    Jump <='0';
                    Add_Sub <='0';

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

All Timing Diagrams.

01. Nano processor

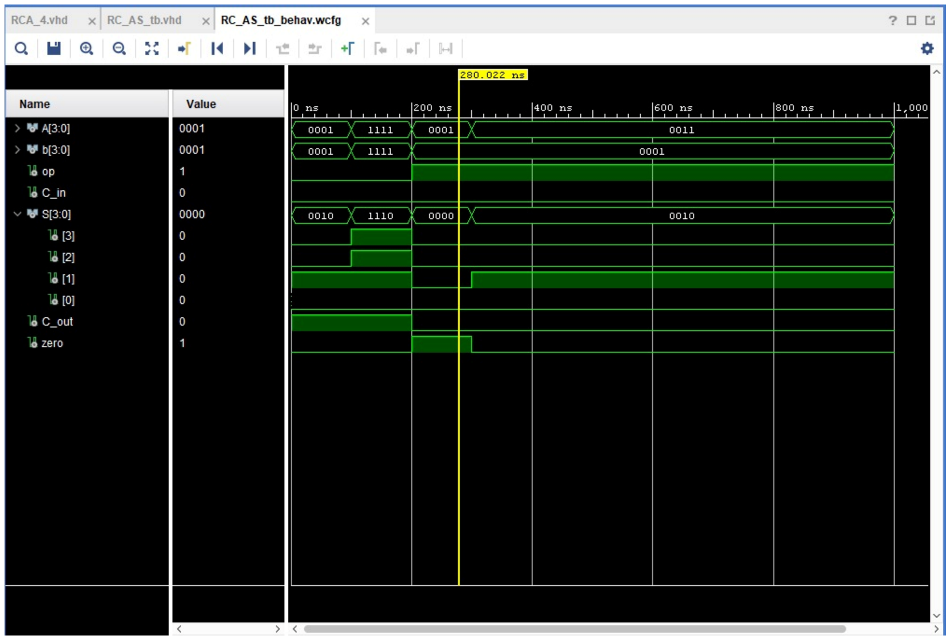


Name	Value
clock	0
Reset	0
Zero	0
Carry	1
LED_out[3:0]	3
Seg_display[6:0]	48

LED_out[3:0]: U, 10, 9, 8, 7, 6, 5, 4, 3

Seg_display[6:0]: 64, 8, 16, 0, 120, 2, 18, 25, 48

02. 4-bit adder/subtractor



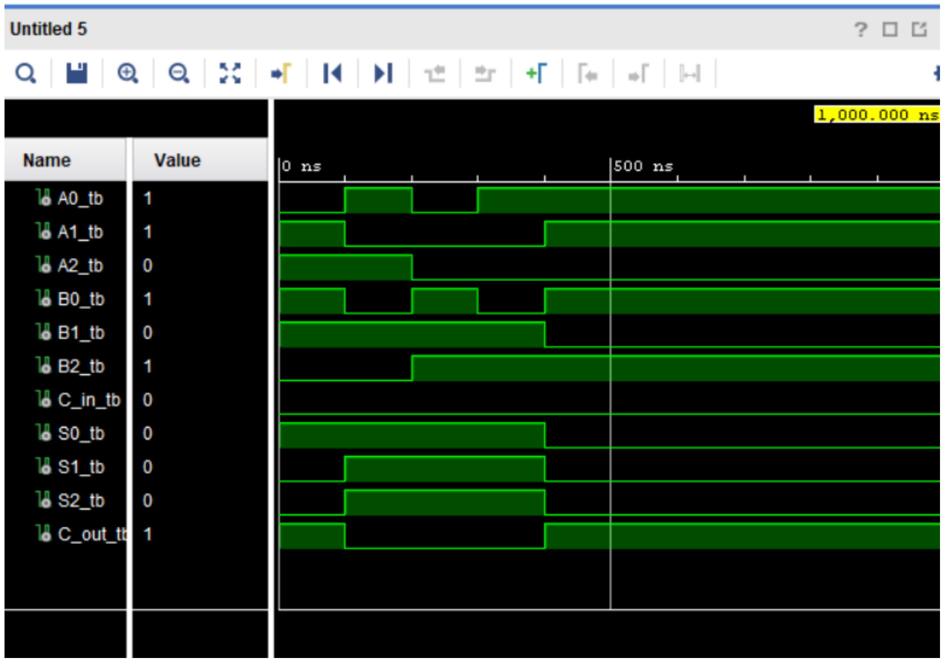
Name	Value
A[3:0]	0001
b[3:0]	0001
op	1
C_in	0
S[3:0]	0000
[3]	0
[2]	0
[1]	0
[0]	0
C_out	0
zero	1

A[3:0]: 0001, 1111, 0001, 0011

b[3:0]: 0001, 1111, 0001

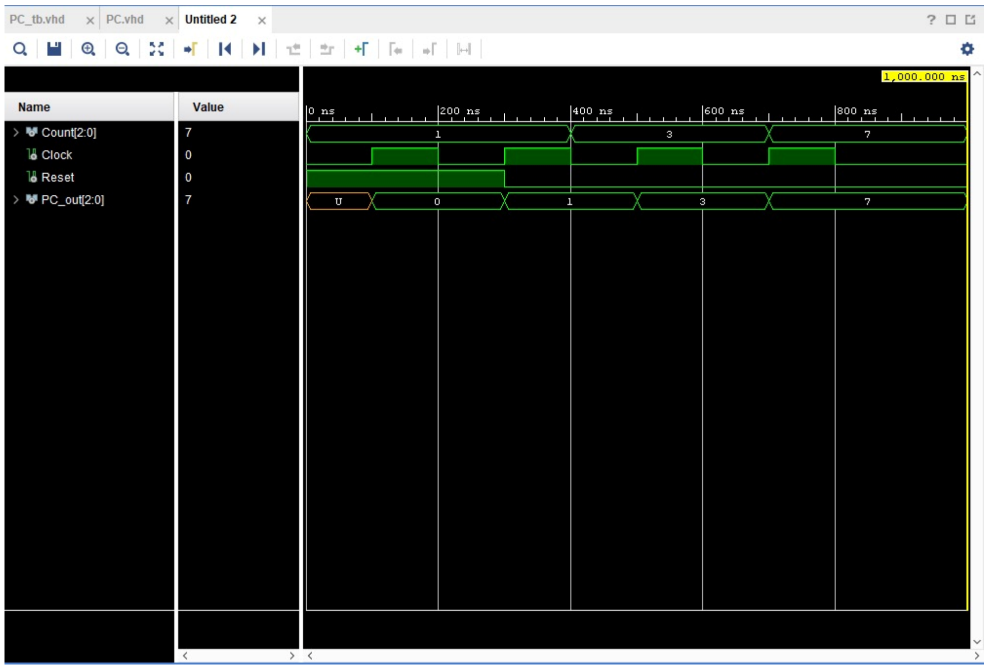
S[3:0]: 0010, 1110, 0000, 0010

03. 3-bit adder



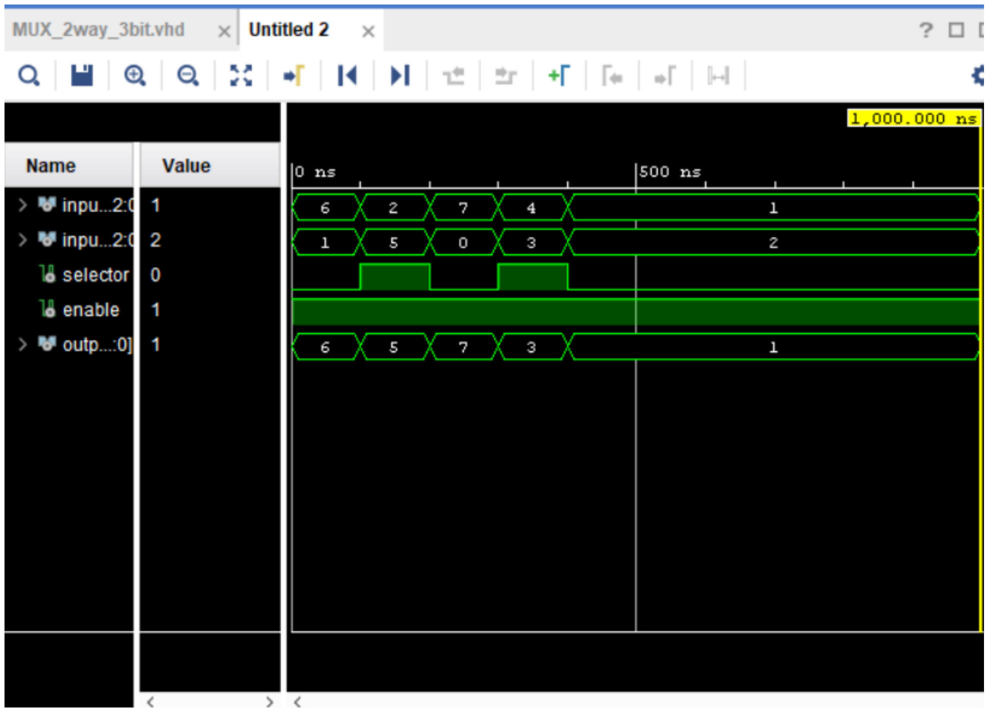
Name	Value
A0_tb	1
A1_tb	1
A2_tb	0
B0_tb	1
B1_tb	0
B2_tb	1
C_in_tb	0
S0_tb	0
S1_tb	0
S2_tb	0
C_out_tb	1

04. 3-bit program counter

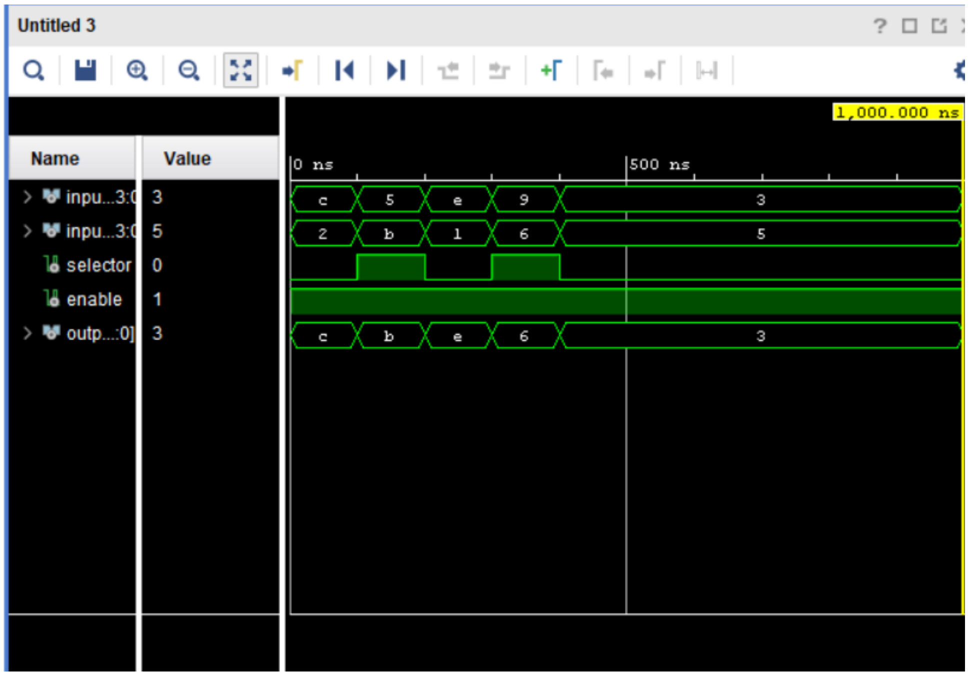


Name	Value
Count[2:0]	7
Clock	0
Reset	0
PC_out[2:0]	7

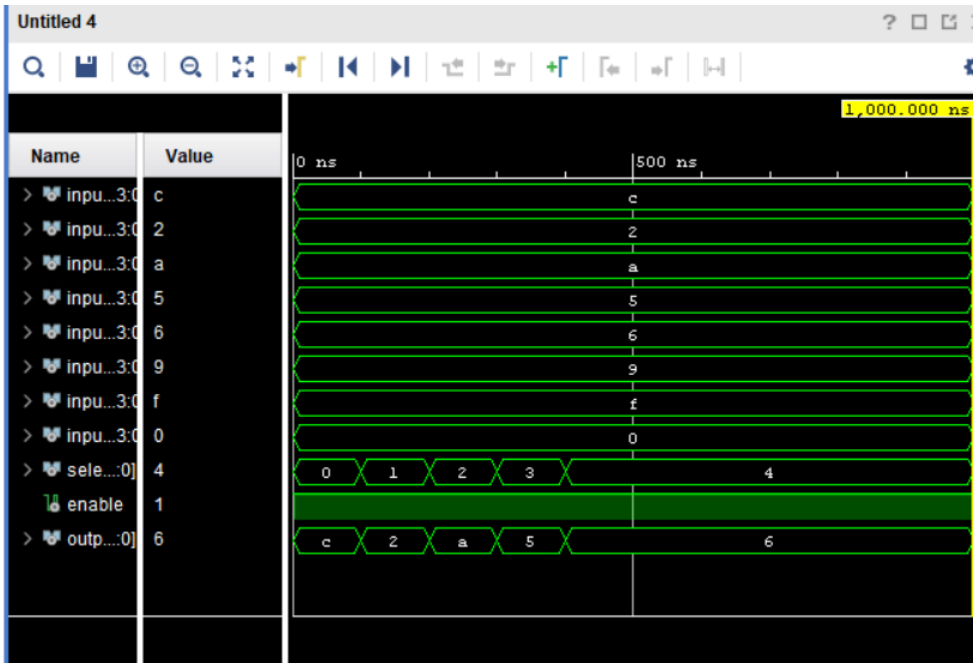
05. 2-way 3-bit multiplexer



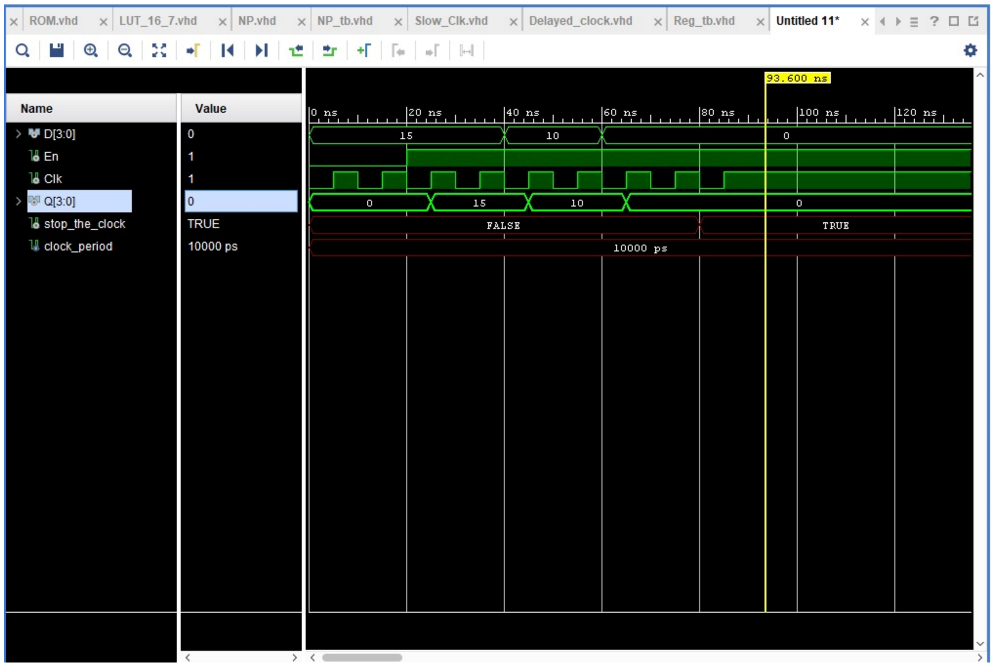
06. 2-way 4-bit multiplexer



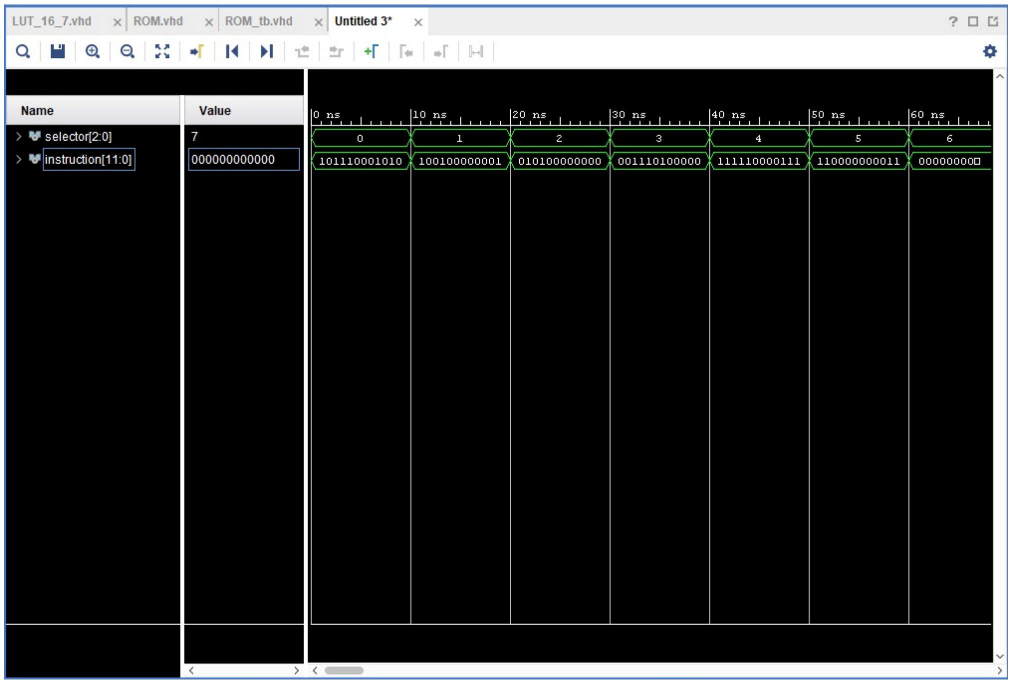
07. 8-way 4-bit multiplexer



08. Register bank

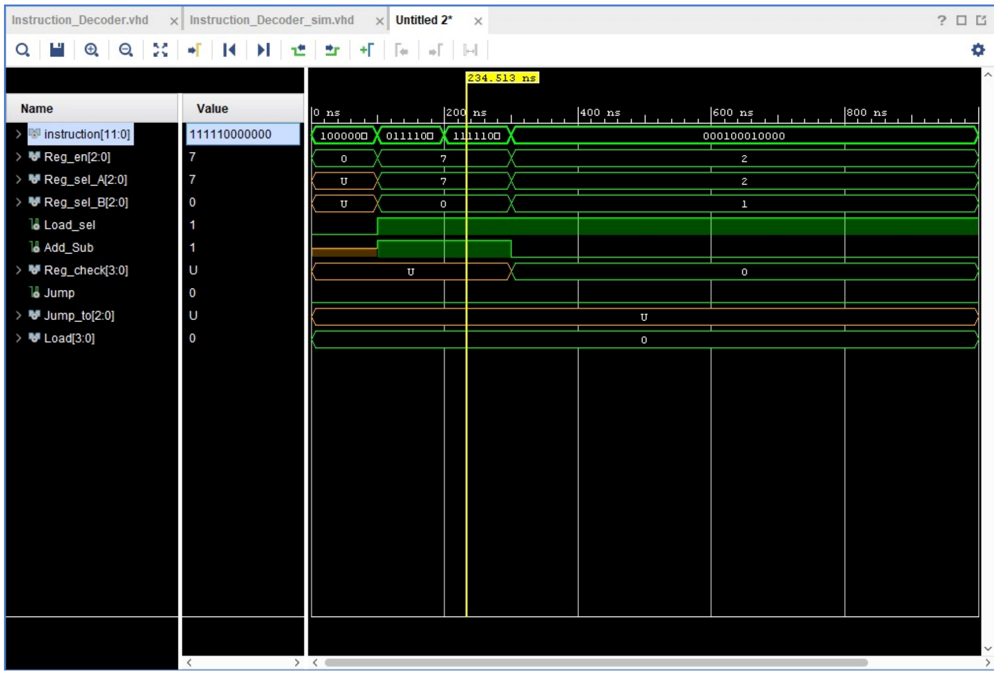


09. Program ROM



Name	Value
selector[2:0]	7
instruction[11:0]	000000000000

10. Instruction decoder



Name	Value
instruction[11:0]	111110000000
Reg_en[2:0]	7
Reg_sel_A[2:0]	7
Reg_sel_B[2:0]	0
Load_sel	1
Add_Sub	1
Reg_check[3:0]	U
Jump	0
Jump_to[2:0]	U
Load[3:0]	0

Conclusions By The Lab.

- We successfully created and tested all the required components, including the 4-bit Add/Subtract unit, 3-bit adder, program counter, multiplexers, register bank, program ROM, and instruction decoder, ensuring their functionality and reliability.

- By hardcoding an assembly program into the program ROM, we enabled the nano processor to execute a set of instructions accurately, showcasing its capability to process and execute programs.

- Through rigorous simulation testing, we verified the correct functionality of the integrated system, ensuring that the nano processor executes the assembly program accurately and produces the expected results.

- By mapping the integrated system and conducting hardware testing on the BASYS 3 board, we validated the compatibility and performance of the nano processor in a real hardware environment.

- Overall, our successful creation, integration, and testing of the nano processor demonstrate our proficiency in designing and implementing a functional microprocessor, while also enhancing our understanding of microprocessor architecture and its practical applications.

Contribution From Each Team Member.

01. K.D.R.P Rathnayake – 210537N
    - 3-bit program counter
    - Register bank
    - Program ROM
    - Instruction decorder
    - Nano processor (mapping all components)
    - Assembly code
    - Number of hours spent – 8 hours

02. K.M.L.I Kulathunga – 210307K
    - 4-bit adder/subtractor
    - 3-bit adder
    - 2-way 3-bit multiplexer
    - 2-way 4-bit multiplexer
    - 8-way 4-bit multiplexer
    - Number of hours spent – 5 hours