Lab 9-10 – Nano Processor Design Competition

CS1050 Computer Organization and Digital Design

Group 100:

- **210296X – Kopimenan L.**
- **210554M – Sajeev K.**

Lab Task:

Designing a simple nano processor which can execute the below assembly instructions.

1. ADD (Addition)
2. MOV (Move)
3. NEG (Negation) 2's complement
4. JZR (Jump) Jump to an instruction if condition is met.

The following components were asked to build and then they were

1. 4-bit Add/Subtract unit
2. 3-bit adder
3. 3-bit Program Counter (PC)
4. k-way b-bit multiplexers
5. Register Bank
6. Program ROM
7. Instruction Decoder

calculating the total between 1 to 3 and simulating the addition of 1,2 and 3 and load it to R7(8th register) were the tasks that the Program Rom contains and executes by the processor. Program contains these tasks as hard coded assembly language instructions.

ASSEMBLY CODE	MACHINE CODE REPRESENTATION
MOV R7,0	101110000000
MOV R1,1	100010000001
MOV R2,2	100100000010
MOV R3,3	100110000011
ADD R7,R1	001110010000
ADD R7,R2	001110100000
ADD R7,R3	001110110000
JZR R3,0	110110000000

Testing and verifying the functionality of the nano processor on the Basys 3 should be done.

VHDL codes

Half Adder

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

entity MUX_8_way_4_Bit 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);
           S : in STD_LOGIC_VECTOR (2 downto 0);
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end MUX_8_way_4_Bit;

architecture Behavioral of MUX_8_way_4_Bit is

begin
    process(R0,R1,R2,R3,R4,R5,R6,R7,S)
    begin
        case S is
            when "000" => Q <= R0;
            when "001" => Q <= R1;
            when "010" => Q <= R2;
            when "011" => Q <= R3;
            when "100" => Q <= R4;
            when "101" => Q <= R5;
            when "110" => Q <= R6;
            when "111" => Q <= R7;
```

```
        when others => Q <= "ZZZZ";
    end case;
  end process;
end Behavioral;
```

Full Adder

```
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 FA is
  Port ( A : in STD_LOGIC;
         B : in STD_LOGIC;
         C_in : in STD_LOGIC;
         S : out STD_LOGIC;
         C_out : out STD_LOGIC);
end FA;

architecture Behavioral of FA is
  component HA
    port(
    A : in std_logic;
    B : in std_logic;
    C : out std_logic;
    S : out std_logic);
```

```
    end component;
    SIGNAL HA0_S, HA0_C, HA1_S, HA1_C : std_logic;
begin
    HA_0 : HA
        port map (
        A => A,
        B => B,
        S => HA0_S,
        C => HA0_C);
    HA_1 : HA
        port map (
        A => HA0_S,
        B => C_in,
        S => HA1_S,
        C => HA1_C);
    S <= HA1_S;
    C_out <= HA0_C OR HA1_C;
end Behavioral;
```

Registrer

```
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;
           Reset : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end Reg;

architecture Behavioral of Reg is
```

```
begin
process(Clk, Reset)
begin
   if(Reset ='1') then
         Q<="0000";
   elsif(rising_edge(Clk)) then

         if(En = '1') then
            Q <= D;
         end if;
   end if;
end process;
end Behavioral;
```

4bit Ripple Carry Adder

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

entity RCA_4 is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
         B : in STD_LOGIC_VECTOR (3 downto 0);
         S : out STD_LOGIC_VECTOR (3 downto 0);
         C_in : in STD_LOGIC;
         C_out : 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;

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 => FA2_C);

   FA_3 : FA
      port map (
         A => A(3),
         B => B(3),
         C_in => FA2_C,
```

```
        S => S(3),
        C_Out => C_out);

end Behavioral;
```

Lookup table for 7 segmant display

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

entity LUT_16_7 is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end LUT_16_7;

architecture Behavioral of LUT_16_7 is

type rom_type is array (0 to 15) of std_logic_vector(6 downto 0);

signal sevenSegment_ROM : rom_type := (
    "0000001", -- 0
    "1001111", --1
    "0010010", --2
    "0000110", --3
    "1001100", --4
    "0100100", --5
    "0100000", --6
    "0001111", --7
    "0000000", --8
    "0000100", --9
    "0001000", -- a
    "1100000", --b
    "0110001", --c
```

```
   "1000010", --d
   "0110000", --e
   "0111000"  -- f
   );
begin
data <= sevenSegment_ROM(to_integer(unsigned(address)));
end Behavioral;
```

Decoder_3_to_8

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

entity Decoder_3_to_8 is
   Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
          EN : in STD_LOGIC := '1';
          Y : out STD_LOGIC_VECTOR (7 downto 0));
end Decoder_3_to_8;

architecture Behavioral of Decoder_3_to_8 is

begin

process (I)
begin
if I = "000" then
   Y <= "00000001";
elsif I = "001" then
   Y <= "00000010";
elsif I = "010" then
   Y <= "00000100";
elsif I = "011" then
   Y <= "00001000";
elsif I = "100" then
```

```
    Y <= "00010000";
elsif I = "101" then
    Y <= "00100000";
elsif I = "110" then
    Y <= "01000000";
else
    Y <= "10000000";
end if;
end process;

end Behavioral;
```

Muxes

2way_3bit

```
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);
           S : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (2 downto 0));
end MUX_2_way_3_Bit;

architecture Behavioral of MUX_2_way_3_Bit is

begin
    process(I0,I1,S)
    begin
       case S is
          when '0' => Q <= I0;
```

```
        when '1' => Q <= I1;
        when others => Q <= "ZZZ";
      end case;
    end process;
end Behavioral;
```

2way_4bit

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

entity MUX_2_way_4_Bit is
   Port ( Adder_Sub_Out : in STD_LOGIC_VECTOR (3 downto 0);
          Imd_Value : in STD_LOGIC_VECTOR (3 downto 0);
          S : in STD_LOGIC;
          Q : out STD_LOGIC_VECTOR (3 downto 0));
end MUX_2_way_4_Bit;

architecture Behavioral of MUX_2_way_4_Bit is

begin
   process(Adder_Sub_Out, Imd_Value, S)
   begin
      case S is
         when '0' => Q <= Adder_Sub_Out ;
         when '1' => Q <= Imd_Value ;
         when others => Q <= "ZZZZ";
      end case;
    end process;
end Behavioral;
```

8way_4bit

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

entity MUX_8_way_4_Bit 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);
           S : in STD_LOGIC_VECTOR (2 downto 0);
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end MUX_8_way_4_Bit;

architecture Behavioral of MUX_8_way_4_Bit is

begin
    process(R0,R1,R2,R3,R4,R5,R6,R7,S)
    begin
        case S is
            when "000" => Q <= R0;
            when "001" => Q <= R1;
            when "010" => Q <= R2;
            when "011" => Q <= R3;
            when "100" => Q <= R4;
            when "101" => Q <= R5;
            when "110" => Q <= R6;
            when "111" => Q <= R7;
            when others => Q <= "ZZZZ";
        end case;
```

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

Program Rom

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

entity Program_Rom is
    Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
   data : 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 program_ROM : rom_type := (

    "101110000000", -- MOVI R7, 0
    "101100000001", -- MOVI R6, 1
    "101010000010", -- MOVI R5, 2
    "101000000011", -- MOVI R4, 3
    "001111100000", -- ADD R7, R6
    "001111010000", -- ADD R7, R5
    "001111000000", -- ADD R7, R4
    "110110000000" -- JZR R3, 0
   );
begin
data <= program_ROM(to_integer(unsigned(address)));

end Behavioral;
```

Slowclock for xsim simulation

```
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;
```

Program counter

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


entity Program_Counter is
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
           En : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (2 downto 0));
end Program_Counter;

architecture Behavioral of Program_Counter is

begin
process (Clk, Reset) begin
       if (rising_edge(Clk)) then
          if (Reset = '0') then
              if (NOT(D="000")) then
                  Q <= D;
              end if;
         else
              Q <= "000";
          end if;
       end if;

end process;

end Behavioral;
```

Register bank

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

entity Register_Bank is
   Port ( Reg_en : in STD_LOGIC_VECTOR (2 downto 0);
          Clk : in STD_LOGIC;
          Reset : in STD_LOGIC;
          Inp : in STD_LOGIC_VECTOR (3 downto 0);
          Q0 : out STD_LOGIC_VECTOR (3 downto 0);
          Q1 : out STD_LOGIC_VECTOR (3 downto 0);
          Q2 : out STD_LOGIC_VECTOR (3 downto 0);
          Q3 : out STD_LOGIC_VECTOR (3 downto 0);
          Q4 : out STD_LOGIC_VECTOR (3 downto 0);
          Q5 : out STD_LOGIC_VECTOR (3 downto 0);
          Q6 : out STD_LOGIC_VECTOR (3 downto 0);
          Q7 : out STD_LOGIC_VECTOR (3 downto 0));
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;
          Reset : in STD_LOGIC;
          Q : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end component;

SIGNAL Y : STD_LOGIC_VECTOR (7 downto 0);
begin

Decoder_3_to_8_0: Decoder_3_to_8
PORT MAP(
   I=>Reg_en,
   EN =>'1',
   Y=>Y);

R0 : Reg
PORT MAP(
   D=>"0000",
   En => '0',
   Clk => Clk,
   Reset => Reset,
   Q=> Q0 );

R1 : Reg
PORT MAP(
   D=>Inp,
   En => Y(1),
   Clk => Clk,
   Reset => Reset,
   Q=> Q1);

R2 : Reg
PORT MAP(
   D=>Inp,
   En => Y(2),
   Clk => Clk,
   Reset => Reset,
   Q=> Q2);
```

```
R3 : Reg
PORT MAP(
   D=>Inp,
   En => Y(3),
   Clk => Clk,
   Reset => Reset,
   Q=> Q3);

R4 : Reg
PORT MAP(
   D=>Inp,
   En => Y(4),
   Clk => Clk,
   Reset => Reset,
   Q=> Q4);

R5 : Reg
PORT MAP(
   D=>Inp,
   En => Y(5),
   Clk => Clk,
   Reset => Reset,
   Q=> Q5);

R6 : Reg
PORT MAP(
   D=>Inp,
   En => Y(6),
   Clk => Clk,
   Reset => Reset,
   Q=> Q6);

R7 : Reg
PORT MAP(
```

```
    D=>Inp,
    En => Y(7),
    Clk => Clk,
    Reset => Reset,
    Q=> Q7);
end Behavioral;
```

Adder Sub

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

entity Add_Sub_Unit is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Sel : in STD_LOGIC; -- Add/Sub Selector --sel=0 ADD , sel=1 SUBSTRACT
           S : out STD_LOGIC_VECTOR (3 downto 0);
           C_Out : out STD_LOGIC; -- Carry flag
           Z_Out : out STD_LOGIC; -- Zero flag
           N_Out : out STD_LOGIC; -- Negetive flag
           P_Out : out STD_LOGIC );-- Parity flag (Odd parity detector)
end Add_Sub_Unit;

architecture Behavioral of Add_Sub_Unit is
component RCA_4
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           S : out STD_LOGIC_VECTOR (3 downto 0);
           C_in : in STD_LOGIC;
           C_out : out STD_LOGIC );
end component;

signal B_Sel, S_out  : STD_LOGIC_VECTOR (3 downto 0);
```

```
begin
B_Sel(0) <= B(0) XOR Sel;
B_Sel(1) <= B(1) XOR Sel;
B_Sel(2) <= B(2) XOR Sel;
B_Sel(3) <= B(3) XOR Sel;

RCA_4_0 : RCA_4
   port map (
      B => B_Sel,
      A => A,
      C_in => Sel,
      S => S_out,
      C_Out => C_Out);

S <= S_out;
Z_Out <= NOT (S_out(0) OR S_out(1) OR S_out(2) OR S_out(3));
N_Out <= S_out(3);
P_Out <= S_out(0) XOR S_out(1) XOR S_out(2) XOR S_out(3);

end Behavioral;
```

Instruction decoder

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

entity Instuction_Decoder is
   Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
          Reg_en : out STD_LOGIC_VECTOR (2 downto 0);
          Load_sel : out STD_LOGIC;
          Imd_Val : out STD_LOGIC_VECTOR (3 downto 0);
          Reg_SelA : out STD_LOGIC_VECTOR (2 downto 0);
          Reg_SelB : out STD_LOGIC_VECTOR (2 downto 0);
          Add_SubSel : out STD_LOGIC;
```

```
        Jump_add : out STD_LOGIC_VECTOR (2 downto 0);
        Jump_flag : out STD_LOGIC;
        Reg_Check_Jmp : in STD_LOGIC_VECTOR (3 downto 0));

end Instuction_Decoder;

architecture Behavioral of Instuction_Decoder is

begin
process(Instruction, Reg_Check_Jmp)
   begin
   Reg_en <= "000";
   Load_sel <= '0';
   Imd_Val <= "0000";
   Reg_SelA <= "000";
   Reg_SelB <= "000";
   Add_SubSel <= '0';
   Jump_add <= "000";
   Jump_flag <= '0';

 If (Instruction(11)='0' and Instruction(10)='0') then

      Reg_en<= Instruction (9 downto 7);
      Load_sel <= '0';
      Reg_SelA <= Instruction (9 downto 7);
      Reg_SelB <= Instruction (6 downto 4);
      Add_SubSel <= '0';
    elsIf (Instruction(11)='0' and Instruction(10)='1') then
        Reg_en <= Instruction(9 downto 7);
        Load_sel <= '0';
        Reg_SelA <= "000";
        Reg_SelB <= Instruction (9 downto 7);
        Add_SubSel <= '1';
```

```
    elsIf (Instruction(11)='1' and Instruction(10)='0') then
            Reg_en <= Instruction(9 downto 7);
            Load_sel <= '1';
            Imd_Val <= Instruction(3 downto 0);
    elsIf (Instruction(11)='1' and Instruction(10)='1') then
            Reg_en <= "000";
            Reg_SelA <= Instruction (9 downto 7);
            if (Reg_Check_Jmp = "0000") then
                Jump_flag <= '1';
                Jump_add <= Instruction(2 downto 0);
            else
                Jump_flag <= '0';
            end if;
   end if;
```

3bit Adder

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

entity Adder_3_Bit is
   Port ( A : 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 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_S, FA0_C, FA1_S, FA1_C, FA2_S : std_logic;
    SIGNAL B : STD_LOGIC_VECTOR (2 downto 0) := "001";

begin

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

Nano Processor

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

entity NanoProcessor is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC;
           R7 : out STD_LOGIC_VECTOR (3 downto 0);
           seg7_out : out STD_LOGIC_VECTOR (6 downto 0);
           anode : out STD_LOGIC_VECTOR (3 downto 0)
           );
end NanoProcessor;

architecture Behavioral of NanoProcessor is


component LUT_16_7
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end component;

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

component Register_Bank
```

```
    Port ( Reg_en : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Inp : in STD_LOGIC_VECTOR (3 downto 0);
           Q0 : out STD_LOGIC_VECTOR (3 downto 0);
           Q1 : out STD_LOGIC_VECTOR (3 downto 0);
           Q2 : out STD_LOGIC_VECTOR (3 downto 0);
           Q3 : out STD_LOGIC_VECTOR (3 downto 0);
           Q4 : out STD_LOGIC_VECTOR (3 downto 0);
           Q5 : out STD_LOGIC_VECTOR (3 downto 0);
           Q6 : out STD_LOGIC_VECTOR (3 downto 0);
           Q7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component MUX_8_way_4_Bit
    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);
           S : in STD_LOGIC_VECTOR (2 downto 0);
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Add_Sub_Unit
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Sel : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
```

```
        C_Out : out STD_LOGIC;
        Z_Out : out STD_LOGIC;
        N_Out : out STD_LOGIC;
        P_Out : out STD_LOGIC );
end component;


component MUX_2_way_4_Bit is
    Port ( Adder_Sub_Out : in STD_LOGIC_VECTOR (3 downto 0);
        Imd_Value : in STD_LOGIC_VECTOR (3 downto 0);
        S : in STD_LOGIC;
        Q : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Instuction_Decoder
    Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
        Reg_en : out STD_LOGIC_VECTOR (2 downto 0);
        Load_sel : out STD_LOGIC;
        Imd_Val : out STD_LOGIC_VECTOR (3 downto 0);
        Reg_SelA : out STD_LOGIC_VECTOR (2 downto 0);
        Reg_SelB : out STD_LOGIC_VECTOR (2 downto 0);
        Add_SubSel : out STD_LOGIC;
        Jump_add : out STD_LOGIC_VECTOR (2 downto 0);
        Jump_flag : out STD_LOGIC;
        Reg_Check_Jmp : in STD_LOGIC_VECTOR (3 downto 0));

end component;

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

```
component Program_Counter
   Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
          En : in STD_LOGIC;
          Reset : in STD_LOGIC;
          Clk : in STD_LOGIC;
          Q : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component Adder_3_Bit
   Port ( A : 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 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;
          Q : out STD_LOGIC_VECTOR (2 downto 0));
end component;
```

```
signal Imd_Val_Ins_to_Mux, RCA_out, Reg_Check_Jmp, Add_Sub_Out, Data_Mux_to_Reg_Bank,
Data_bus0, Data_bus1, Data_bus2, Data_bus3, Data_bus4, Data_bus5, Data_bus6, Data_bus7:
STD_LOGIC_VECTOR(3 downto 0);

signal Ins_Bus : STD_LOGIC_VECTOR(11 downto 0);

signal Reg_bus, Mem_Bus, Reg_SelA , Reg_SelB, Count_Mux_to_PC, Mem_RCA3_to_Mux, Jmp_Add:
STD_LOGIC_VECTOR (2 downto 0);

signal Add_Sub_Sel, Load_Select_Ins_to_Mux, C_out_RCA_3, Jmp_Flag: STD_LOGIC;
```

```
signal Slow_Clk_Signal: STD_LOGIC;

signal Mux_A_out , Mux_B_out :std_logic_vector(3 downto 0);
signal carry_Flag, zero_Flag, negative_Flag, parity_Flag : std_logic;

begin

LUT_16_7_0 : LUT_16_7
port map(
address => Data_bus7,
data => seg7_out);

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

Program_Rom_0 :Program_Rom
   port map (
   address => Mem_Bus,
   data => Ins_Bus);

Program_Counter_0  : Program_Counter
port map (
   D => Count_Mux_to_PC,
   EN => '1',
   Reset => Reset,
   Clk => Slow_Clk_Signal,
   Q => Mem_Bus);


Instuction_Decoder_0 : Instuction_Decoder
```

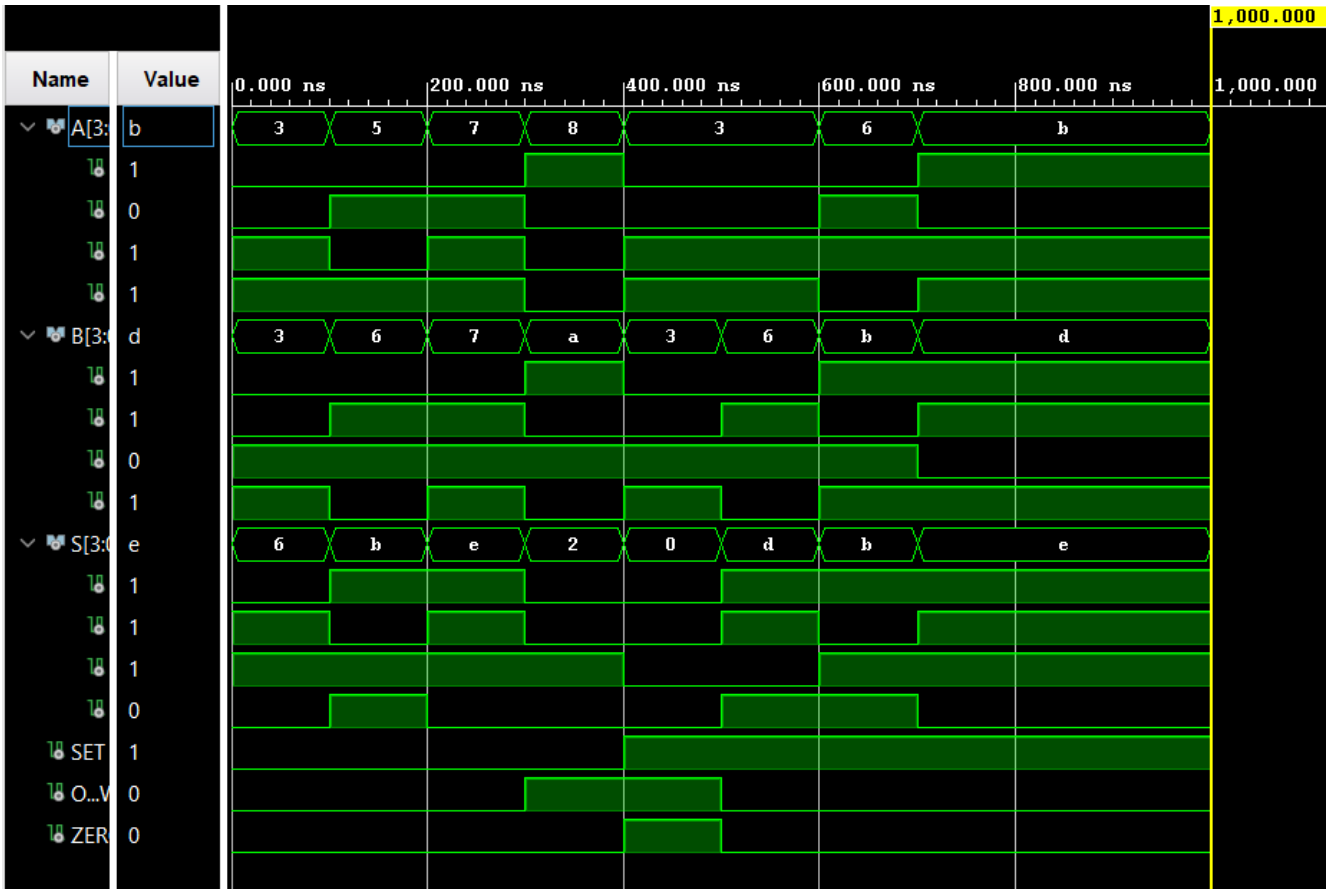
```
port map (
   Instruction => Ins_Bus,
   Reg_en => Reg_bus,
   Load_sel => Load_Select_Ins_to_Mux,
   Imd_Val => Imd_Val_Ins_to_Mux,
   Reg_SelA => Reg_SelA,
   Reg_SelB => Reg_SelB,
   Add_SubSel => Add_Sub_Sel,
   Jump_add => Jmp_Add,
   Jump_flag => Jmp_Flag,
   Reg_Check_Jmp => Reg_Check_Jmp);

Reg_Check_Jmp <= Mux_A_out;

Reg_Bank_0 : Register_Bank
port map (
     Reg_en => Reg_bus,
     Clk => Slow_Clk_Signal,
     Reset => Reset,
     Inp => Data_Mux_to_Reg_Bank,
     Q0 => Data_bus0,
     Q1 => Data_bus1,
     Q2 => Data_bus2,
     Q3 => Data_bus3,
     Q4 => Data_bus4,
     Q5 => Data_bus5,
     Q6 => Data_bus6,
     Q7 => Data_bus7 );

 Add_Sub_Unit_0 : Add_Sub_Unit
     port map( A => Mux_A_out,
    B => Mux_B_out,
```

```
    Sel => Add_Sub_Sel,
    S =>  Add_Sub_Out,
    C_Out => carry_Flag,
    Z_Out => zero_Flag,
    N_Out => negative_Flag,
    P_Out => parity_Flag );

Mux_2_way_4_Bit_0 : Mux_2_way_4_Bit
     port map (
     S => Load_Select_Ins_to_Mux,
     Adder_Sub_Out => Add_Sub_Out,
     Imd_Value => Imd_Val_Ins_to_Mux,
     Q => Data_Mux_to_Reg_Bank);


MUX_8_way_4_Bit_A :  MUX_8_way_4_Bit
   port map (
         R0 => Data_bus0,
         R1 => Data_bus1,
         R2 => Data_bus2,
         R3 => Data_bus3,
         R4 => Data_bus4,
         R5 => Data_bus5,
         R6 => Data_bus6,
         R7 => Data_bus7,
         S => Reg_SelA,
         Q => Mux_A_out );


 MUX_8_way_4_Bit_B :  MUX_8_way_4_Bit
  port map (
        R0 => Data_bus0,
```

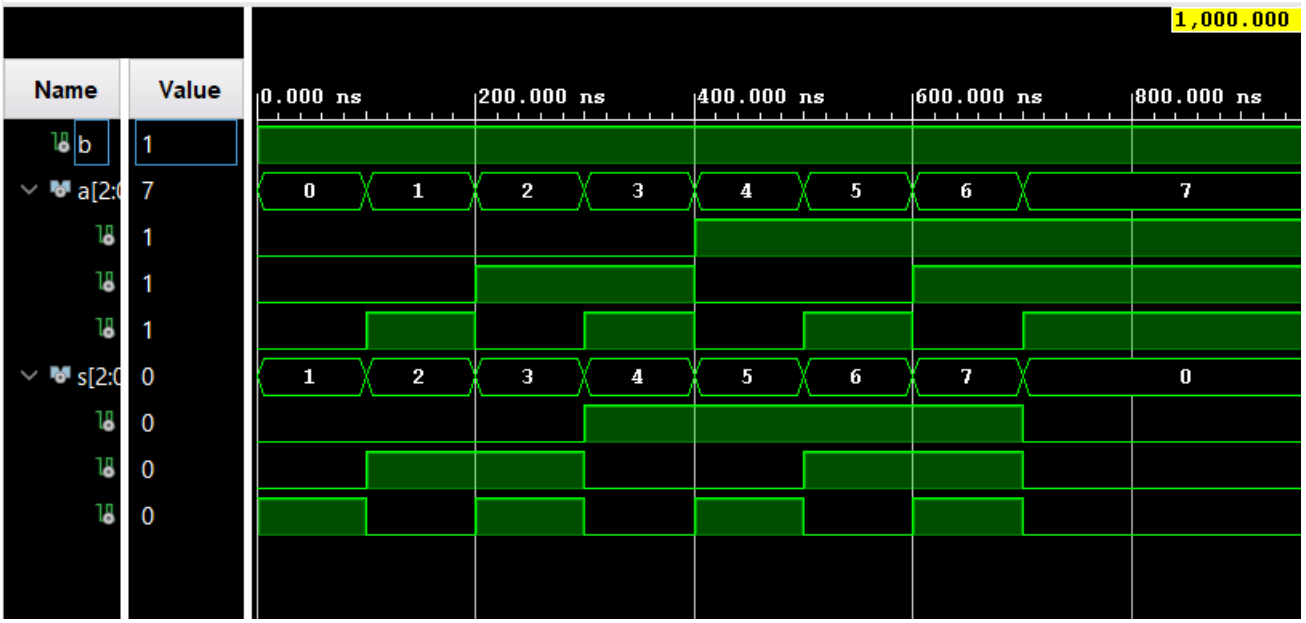
```
        R1 => Data_bus1,
        R2 => Data_bus2,
        R3 => Data_bus3,
        R4 => Data_bus4,
        R5 => Data_bus5,
        R6 => Data_bus6,
        R7 => Data_bus7,
        S => Reg_SelB,
        Q => Mux_B_out );

Adder_3_Bit_0  : Adder_3_Bit
port map (
   A => Mem_Bus,
   C_in => '0',
   S => Mem_RCA3_to_Mux,
   C_out => C_out_RCA_3);

MUX_2_way_3_Bit_0 : MUX_2_way_3_Bit
port map (
   S => Jmp_Flag,
   I0 => Mem_RCA3_to_Mux,
   I1 => Jmp_Add,
   Q => Count_Mux_to_PC);

Overflow <= carry_Flag;
Zero <= zero_Flag;

R7 <= Data_bus7;

anode <= "1110";

end Behavioral;
```

Timing Diagrams

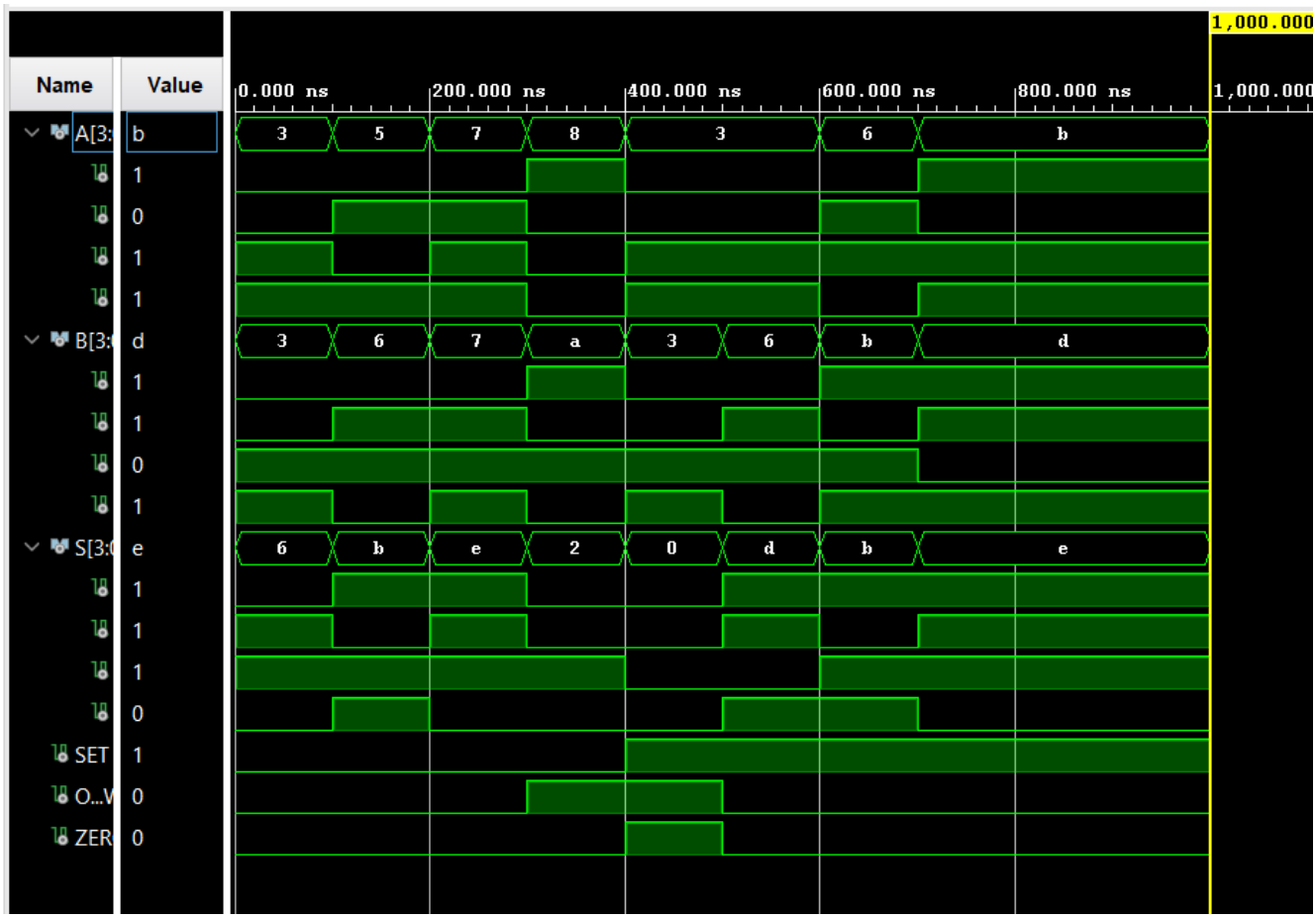
Instruction Decoder



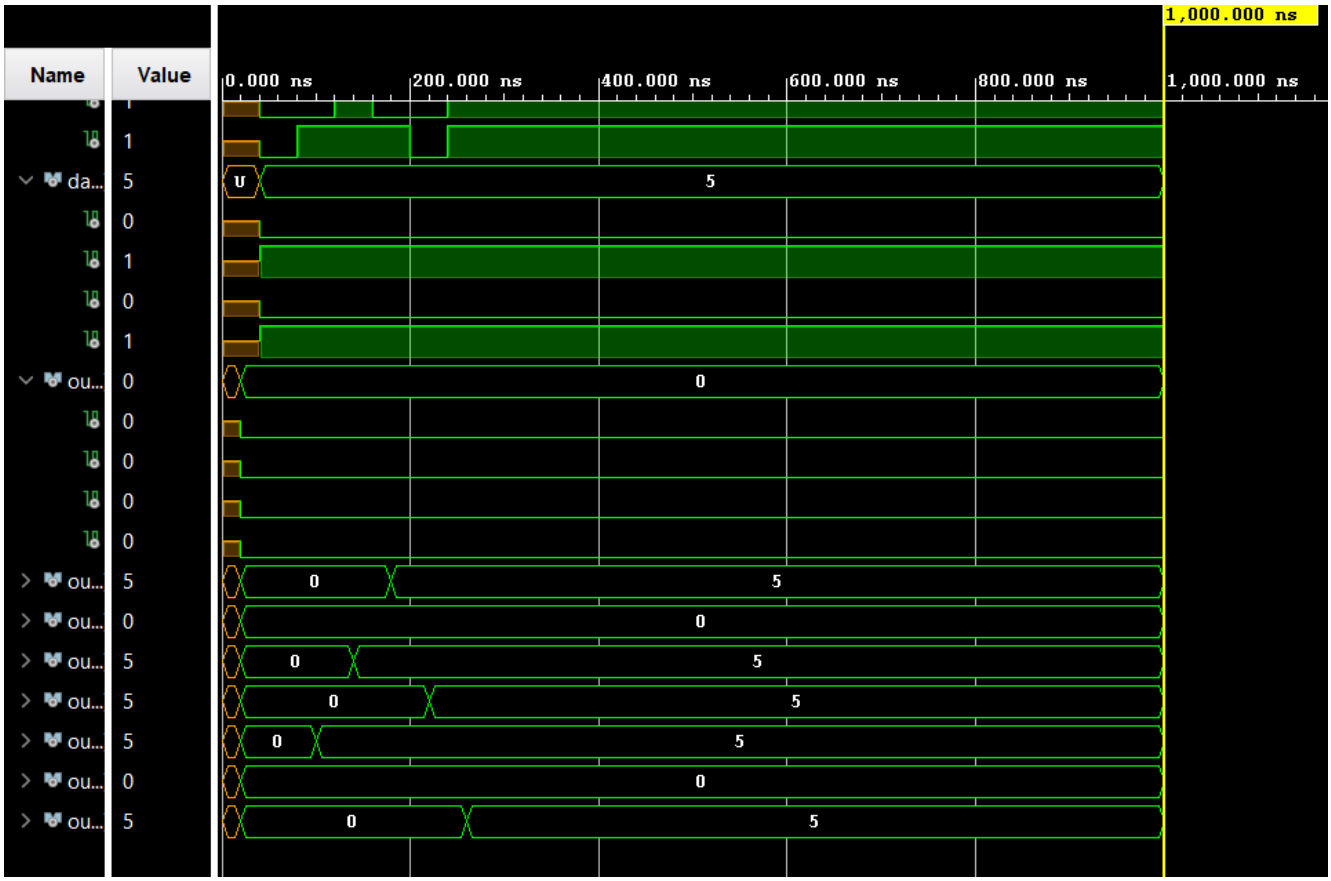
3 bit Adder

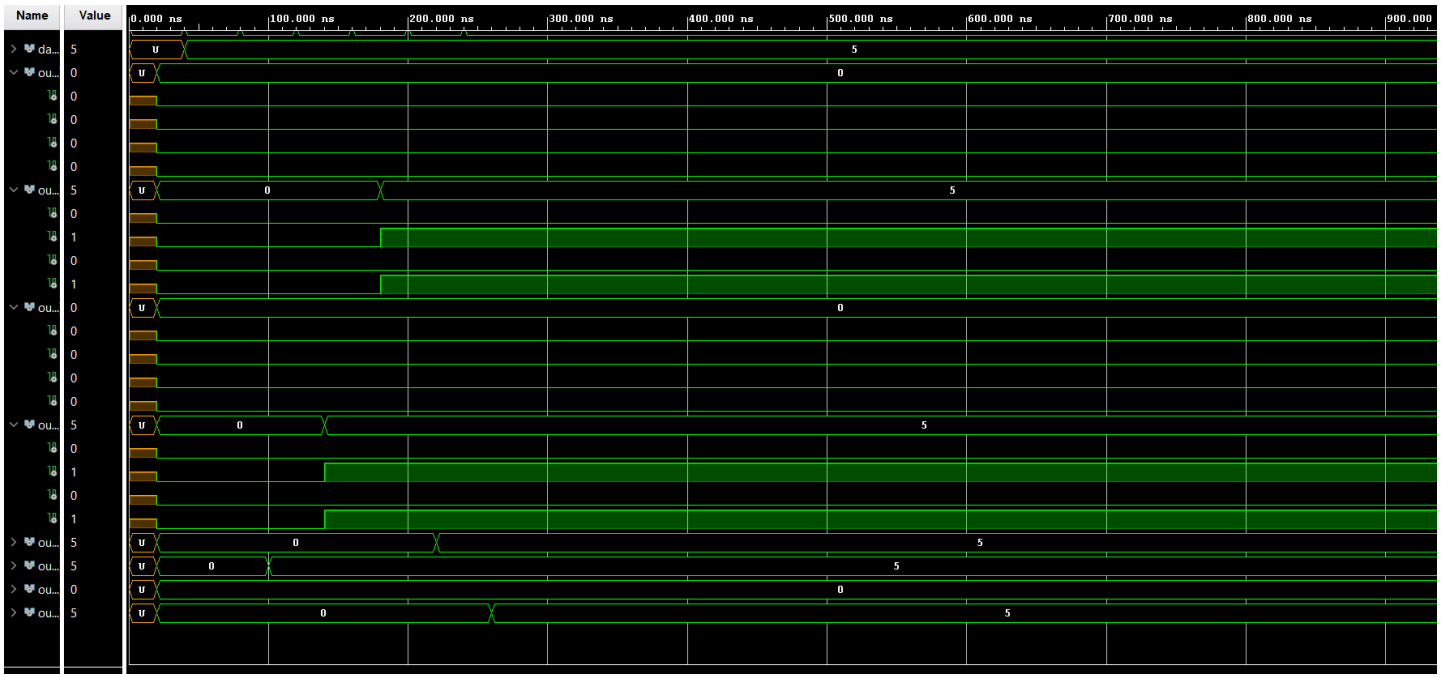


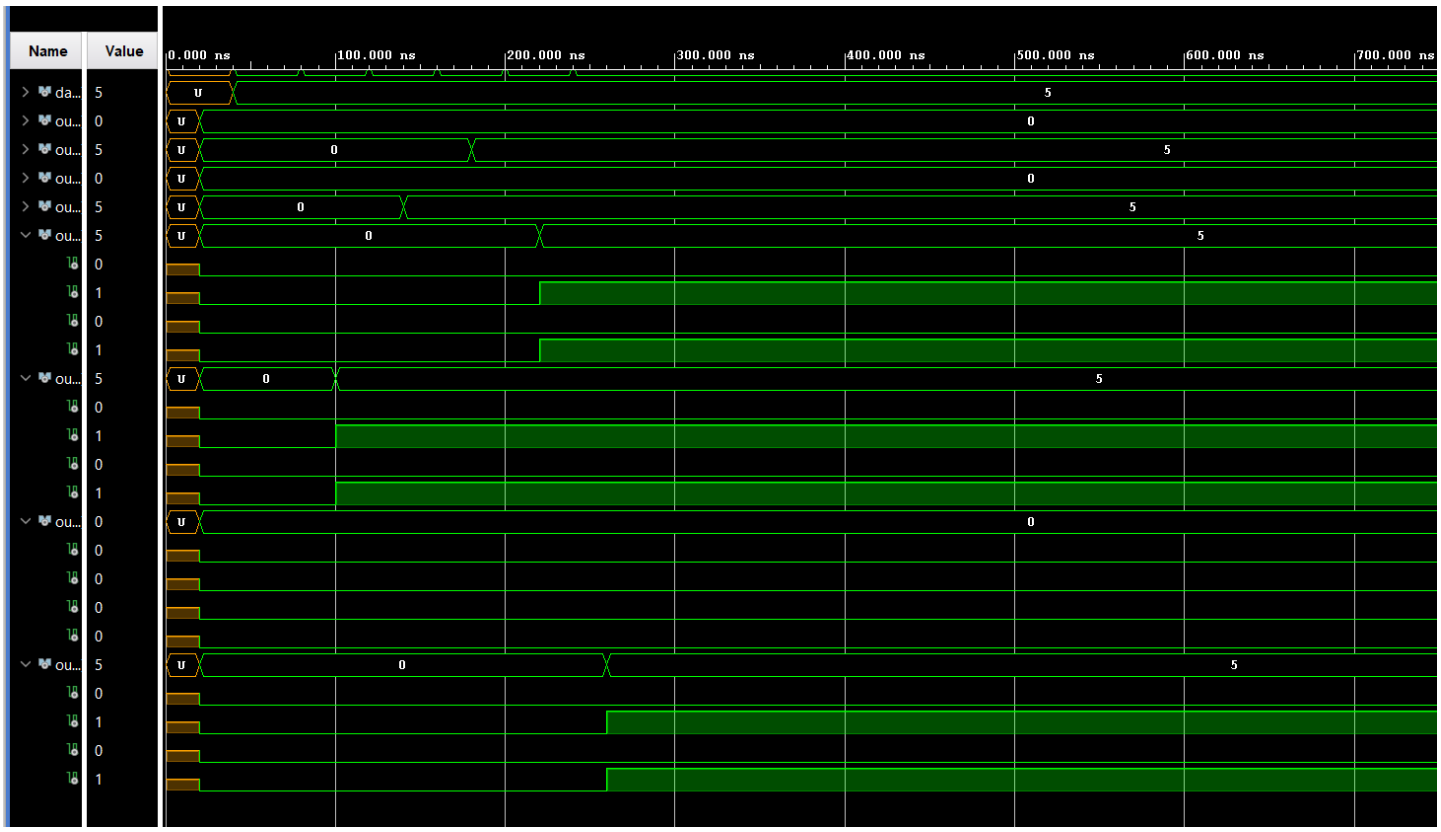
Adder Substraction Unit



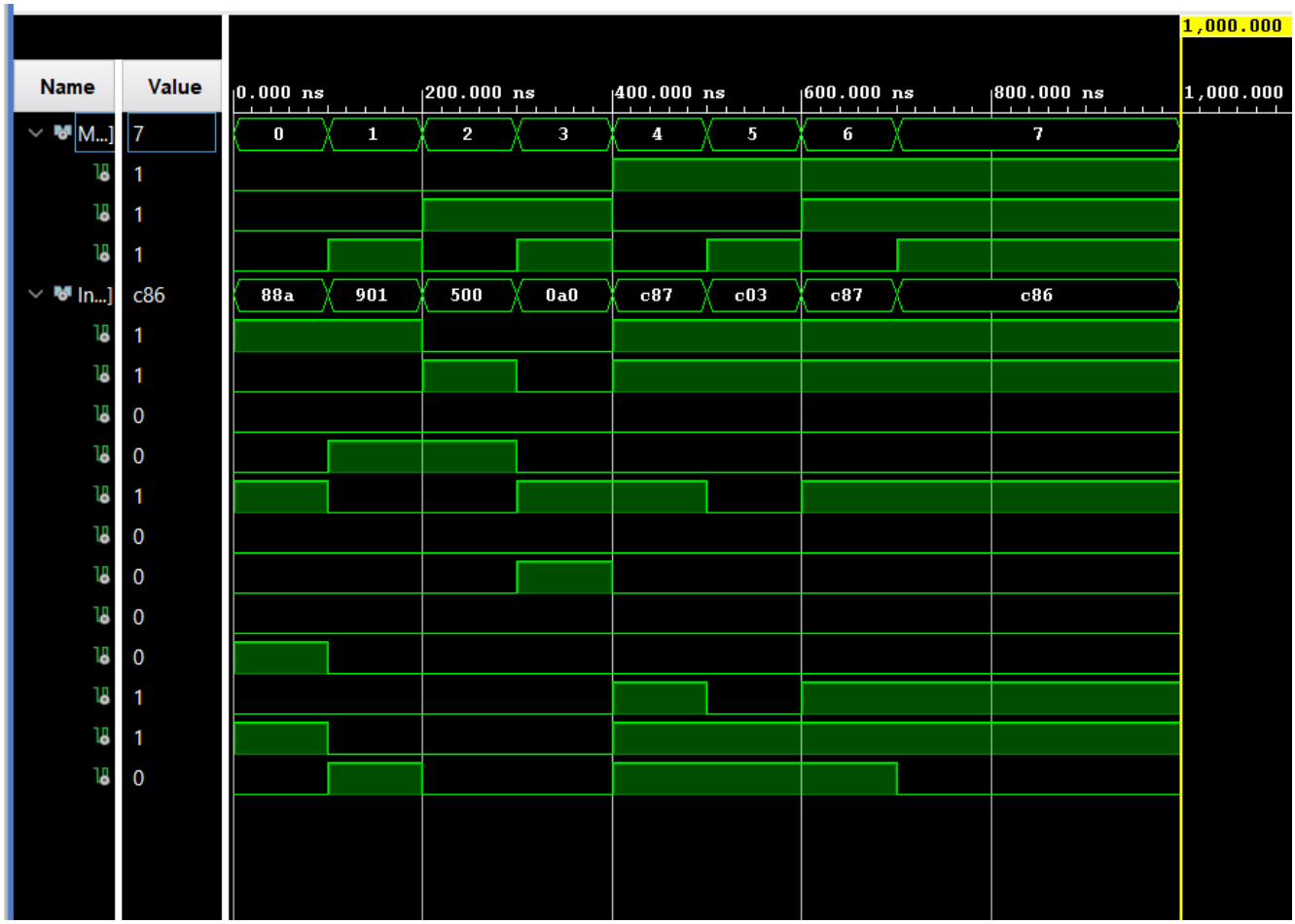
Register bank




Name
Value
0.000 ns
100.000 ns
200.000 ns
300.000 ns
400.000 ns
500.000 ns
600.000 ns
700.000 ns
800.000 ns
900.000

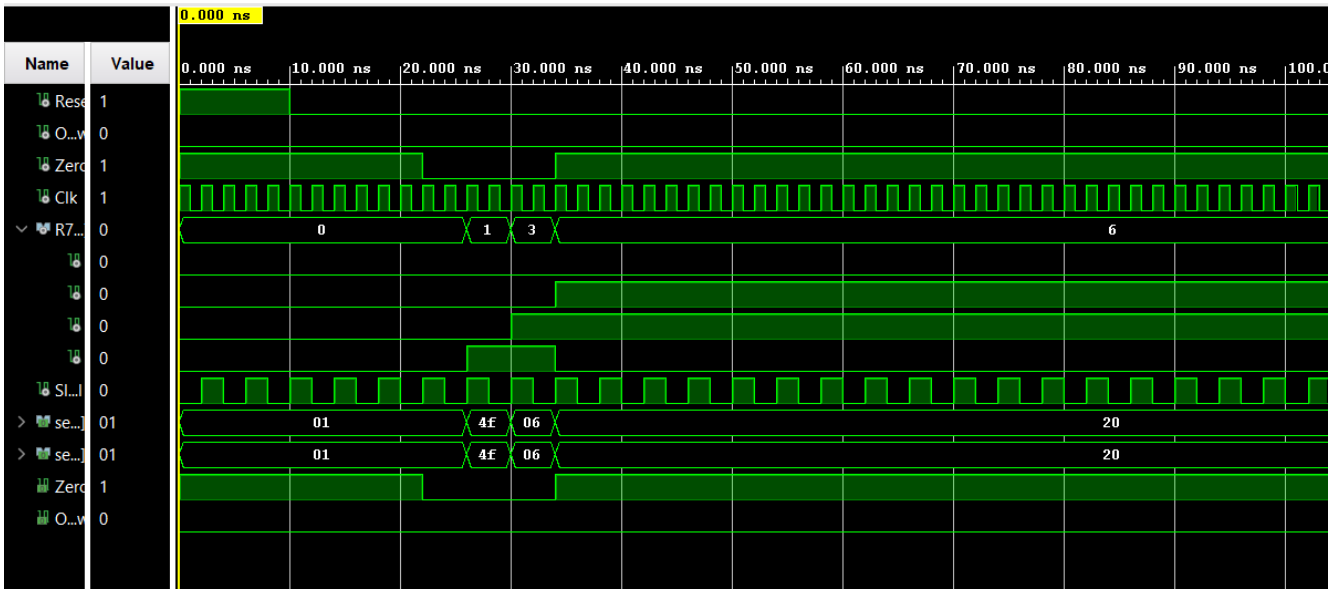

Name
Value
0.000 ns
100.000 ns
200.000 ns
300.000 ns
400.000 ns
500.000 ns
600.000 ns
700.000 ns

Program Rom



Program Counter

Nano Processor



Conclusion

- Learnt to develop a functional nano-processor that could execute simple assembly instructions like ADD, MOV, NEG and JNZ and display the output value on a seven segment display.
- In addition we were able to further optimize the design by using a behavioral approach to design the MUXs instead of decoders.
- Understood the importance of simulation of the circuits before programming on the boards.

Contribution: We totally spent 6 hours each and shared the work among us.

Kopimenan	Instruction Decoder, Program Counter, Program ROM, Register Bank
Sajeev	Adder Substraction Unit, Muxes, 3 bit Adder