Lab 9-10 Microprocessor

Group 46

C.S.M. Fernando: 210159G

M.T. Tharushika: 210636R

Lab Task

In this lab, we have to design a 4-bit processor capable of executing 4 instructions. For that we have to;

- Design and develop a 4-bit arithmetic unit that can add and subtract signed integers
- Decode instructions to activate necessary components on the processor
- Design and develop k-way b-bit multiplexers or tri-state busses

Finally, we have to verify their functionality via simulation and on the development board.

Assembly program and its machine code representation

Assembly Program	Machine Code Representation
MOVI R7, 0	101110000000
MOVI R6, 3	101100000011
MOVI R5, 1	101010000001
NEG R5	011010000000
ADD R7, R6	001111100000
ADD R6, R5	001101010000
JZR R6, 6	111100000110
JZR R0, 4	110000000100

VHDL codes

Nano Processor

Design Source

```
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);
           Seven_seg_out : out STD_LOGIC_VECTOR (6 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0)
           );
end NanoProcessor;

architecture Behavioral of NanoProcessor is



component SevenSeg_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 Reg_Bank
    Port ( Reg_en : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           D : in STD_LOGIC_VECTOR (3 downto 0);
           R0 : out STD_LOGIC_VECTOR (3 downto 0);
           R1 : out STD_LOGIC_VECTOR (3 downto 0);
```

```
        R2 : out STD_LOGIC_VECTOR (3 downto 0);
        R3 : out STD_LOGIC_VECTOR (3 downto 0);
        R4 : out STD_LOGIC_VECTOR (3 downto 0);
        R5 : out STD_LOGIC_VECTOR (3 downto 0);
        R6 : out STD_LOGIC_VECTOR (3 downto 0);
        R7 : out STD_LOGIC_VECTOR (3 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);
        Load_sel : out STD_LOGIC;
        Imm_Val : out STD_LOGIC_VECTOR (3 downto 0);
        Reg_Sel_A : out STD_LOGIC_VECTOR (2 downto 0);
        Reg_Sel_B : out STD_LOGIC_VECTOR (2 downto 0);
        Add_Sub_Sel : out STD_LOGIC;
        Jump_add : out STD_LOGIC_VECTOR (2 downto 0);
        Jump_flag : out STD_LOGIC;
        Reg_Chk_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);
        Reset : in STD_LOGIC;
        Clk : in STD_LOGIC;
        Q : out STD_LOGIC_VECTOR (2 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;  --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
        Overflow : out STD_LOGIC );   --Overflow flag
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));
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 MUX_2_way_4_Bit is
    Port ( I0 : in STD_LOGIC_VECTOR (3 downto 0);
        I1 : in STD_LOGIC_VECTOR (3 downto 0);
        S : in STD_LOGIC;
        Q : 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;
        Q : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal Imd_Value, Reg_Check, ASU_Out, Reg_Bank_Input, Data_bus_0, Data_bus_1, Data_bus_2, Data_bus_3,
Data_bus_4, Data_bus_5, Data_bus_6, Data_bus_7: STD_LOGIC_VECTOR(3 downto 0);
signal Inst_Bus : STD_LOGIC_VECTOR(11 downto 0);
signal Reg_en, Mem_Sel, Reg_Sel_A , Reg_Sel_B, Mux_to_PC, Adder_3_bit_out, Jmp_Address: STD_LOGIC_VECTOR (2
downto 0);
signal AS_Sel, Load_Select, Jump_Flag: STD_LOGIC;
signal Slow_Clk_out: STD_LOGIC;

signal Mux_A_out , Mux_B_out :std_logic_vector(3 downto 0);
signal Carry_Flag, Zero_Flag,Overflow_Flag: std_logic;

begin

SevenSeg_LUT_16_7_0 : SevenSeg_LUT_16_7
port map(
    address => Data_bus_7,
    data => Seven_seg_out);

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

Reg_Bank_0 : Reg_Bank
    port map (
        Reg_en => Reg_en,
        Clk => Slow_Clk_out,
        Reset => Reset,
        D => Reg_Bank_Input,
        R0 => Data_bus_0,
        R1 => Data_bus_1,
```

```
        R2 => Data_bus_2,
        R3 => Data_bus_3,
        R4 => Data_bus_4,
        R5 => Data_bus_5,
        R6 => Data_bus_6,
        R7 => Data_bus_7 );

Instruction_Decoder_0 : Instruction_Decoder
        port map (
            Instruction => Inst_Bus,
            Reg_en => Reg_en,
            Load_sel => Load_Select,
            Imm_Val => Imd_Value,
            Reg_Sel_A => Reg_Sel_A,
            Reg_Sel_B => Reg_Sel_B,
            Add_Sub_Sel => AS_Sel,
            Jump_add => Jmp_Address,
            Jump_flag => Jump_Flag,
            Reg_Chk_Jmp => Reg_Check);

Program_Rom_0 :Program_Rom
port map (
    address => Mem_Sel,
    data => Inst_Bus);

Program_Counter_0  : Program_Counter
port map (
    D => Mux_to_PC,
    Reset => Reset,
    Clk => Slow_Clk_out,
    Q => Mem_Sel);

Reg_Check <= Mux_A_out;

 Add_Sub_Unit_0 : Add_Sub_Unit
 port map(
```

```
    A => Mux_A_out,
    B => Mux_B_out,
    Sel => AS_Sel,
    S =>  ASU_Out,
    C_Out => Carry_Flag,
    Z_Out => Zero_Flag,
    Overflow => Overflow_Flag    );

Adder_3_Bit_0  : Adder_3_Bit
    port map (
        A => Mem_Sel,
        C_in => '0',
        S => Adder_3_bit_out);

MUX_8_way_4_Bit_A :  MUX_8_way_4_Bit
port map (
          R0 => Data_bus_0,
          R1 => Data_bus_1,
          R2 => Data_bus_2,
          R3 => Data_bus_3,
          R4 => Data_bus_4,
          R5 => Data_bus_5,
          R6 => Data_bus_6,
          R7 => Data_bus_7,
          S => Reg_Sel_A,
          Q => Mux_A_out );

MUX_8_way_4_Bit_B :  MUX_8_way_4_Bit
port map (
          R0 => Data_bus_0,
          R1 => Data_bus_1,
          R2 => Data_bus_2,
          R3 => Data_bus_3,
          R4 => Data_bus_4,
          R5 => Data_bus_5,
          R6 => Data_bus_6,
```

```
        R7 => Data_bus_7,
        S => Reg_Sel_B,
        Q => Mux_B_out );

Mux_2_way_4_Bit_0 : Mux_2_way_4_Bit
        port map (
            S => Load_Select,
            I0 => ASU_Out,
            I1 => Imd_Value,
            Q => Reg_Bank_Input);

MUX_2_way_3_Bit_0 : MUX_2_way_3_Bit
port map (
    S => Jump_Flag,
    I0 => Adder_3_bit_out,
    I1 => Jmp_Address,
    Q => Mux_to_PC);

Zero <= Zero_Flag;
Overflow <= Overflow_Flag;
R7 <= Data_bus_7;

an <= "1110";

end Behavioral;
```

Test Bench file

```
entity TB_Nanoprocessor is
--  Port ( );
end TB_Nanoprocessor;

architecture Behavioral of TB_Nanoprocessor is

component Nanoprocessor
    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));
    end component;

    signal Reset, Overflow, Zero : STD_LOGIC;
    signal Clk : STD_LOGIC := '0';
    signal R7 : STD_LOGIC_VECTOR (3 downto 0);

    begin

    UUT: NanoProcessor PORT MAP(
        Clk => Clk,
        Reset => Reset,
        Overflow => Overflow,
        Zero => Zero,
        R7 => R7
    );

    process
        begin
        wait for 1ns;
        Clk <= NOT(Clk);
    end process;

    process
    begin

    Reset <= '1';
    wait for 100 ns;

    Reset <= '0';
    wait;
    end process;
```

```
end Behavioral;
```

7 Segments Display

Design Source

```
entity SevenSeg_LUT_16_7 is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end SevenSeg_LUT_16_7;

architecture Behavioral of SevenSeg_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;
```

Test Bench file

```
entity TB_SevenSeg_LUT is
--  Port ( );
end TB_SevenSeg_LUT;

architecture Behavioral of TB_SevenSeg_LUT is
component SevenSeg_LUT_16_7 is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end component;

signal address : STD_LOGIC_VECTOR (3 downto 0);
signal data : STD_LOGIC_VECTOR (6 downto 0);

begin
UUT : SevenSeg_LUT_16_7
    port map (
      address => address,
      data    => data
);

process

begin
    address <= "1100";
    WAIT FOR 250 ns;

    address<= "1101";
    WAIT FOR 250 ns;

    address<= "0011";
```

```
    WAIT FOR 250 ns;

    address<= "1011";
    WAIT;

end process;

end Behavioral;
```

Slow Clock

Design Source

```
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 = 30000000) then
            --if (count = 4) then
                clk_status <= not clk_status;
                Clk_out <= clk_status;
                count <= 1;
            end if;
        end if;
    end process;
end Behavioral;
```

Register Bank

Design Source

```
entity Reg_Bank is
    Port ( Reg_en : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           D : in STD_LOGIC_VECTOR (3 downto 0);
           R0 : out STD_LOGIC_VECTOR (3 downto 0);
           R1 : out STD_LOGIC_VECTOR (3 downto 0);
           R2 : out STD_LOGIC_VECTOR (3 downto 0);
           R3 : out STD_LOGIC_VECTOR (3 downto 0);
           R4 : out STD_LOGIC_VECTOR (3 downto 0);
           R5 : out STD_LOGIC_VECTOR (3 downto 0);
           R6 : out STD_LOGIC_VECTOR (3 downto 0);
           R7 : out STD_LOGIC_VECTOR (3 downto 0));
end Reg_Bank;

architecture Behavioral of Reg_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
);

Reg_0 : Reg
PORT MAP(
    D =>"0000",
    En => '0',
    Clk => Clk,
    Reset => Reset,
    Q=> R0
);

Reg_1 : Reg
PORT MAP(
    D =>D,
    En => Y(1),
    Clk => Clk,
    Reset => Reset,
    Q=> R1
);

Reg_2: Reg
PORT MAP(
    D => D,
    En => Y(2),
    Clk => Clk,
    Reset => Reset,
    Q=> R2
);
```

```
Reg_3: Reg
PORT MAP(
    D => D,
    En => Y(3),
    Clk => Clk,
    Reset => Reset,
    Q=> R3
);

Reg_4 : Reg
PORT MAP(
    D => D,
    En => Y(4),
    Clk => Clk,
    Reset => Reset,
    Q=> R4 );

Reg_5 : Reg
PORT MAP(
    D => D,
    En => Y(5),
    Clk => Clk,
    Reset => Reset,
    Q=> R5
);

Reg_6 : Reg
PORT MAP(
    D => D,
    En => Y(6),
    Clk => Clk,
    Reset => Reset,
    Q=> R6
);

Reg_7 : Reg
```

```
PORT MAP(
     D => D,
    En => Y(7),
    Clk => Clk,
    Reset => Reset,
    Q=> R7
);
end Behavioral;
```

Test Bench file

```
entity TB_Reg_Bank is
--  Port ( );
end TB_Reg_Bank;

architecture Behavioral of TB_Reg_Bank is
component Reg_Bank
    Port ( Reg_en : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           D : in STD_LOGIC_VECTOR (3 downto 0);
           R0 : out STD_LOGIC_VECTOR (3 downto 0);
           R1 : out STD_LOGIC_VECTOR (3 downto 0);
           R2 : out STD_LOGIC_VECTOR (3 downto 0);
           R3 : out STD_LOGIC_VECTOR (3 downto 0);
           R4 : out STD_LOGIC_VECTOR (3 downto 0);
           R5 : out STD_LOGIC_VECTOR (3 downto 0);
           R6 : out STD_LOGIC_VECTOR (3 downto 0);
           R7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal Reg_en : STD_LOGIC_VECTOR (2 downto 0);
signal Clk : STD_LOGIC := '0';
signal Reset : STD_LOGIC;
signal Input_D : STD_LOGIC_VECTOR (3 downto 0);
signal R0,R1,R2,R3,R4,R5,R6,R7 : STD_LOGIC_VECTOR (3 downto 0);

begin
UUT: Reg_Bank
```

```
Port Map(
    Reg_en => Reg_en,
    Clk => Clk,
    Reset => Reset,
    D => Input_D,
    R0 => R0,
    R1 => R1,
    R2 => R2,
    R3 => R3,
    R4 => R4,
    R5 => R5,
    R6 => R6,
    R7 => R7 );


process
    begin
    wait for 10ns;
    Clk <= NOT(Clk);
end process;

process
begin

                    Input_D <= "1111";
                    Reset <= '0';
        Reg_En <= "100";
                    WAIT FOR 100 ns;

                    Reg_En <= "001";
                    WAIT FOR 100 ns;

                    Reg_En <= "010";
                    WAIT FOR 100 ns;

                    Reg_En <= "101";
                    WAIT FOR 100 ns;

                    Reg_En <= "111";
                    WAIT FOR 100 ns;
```

```
                Reg_En <= "011";
                WAIT FOR 100 ns;

                Reg_En <= "110";
                WAIT FOR 100 ns;

                Reset <='1';
        WAIT FOR 100ns;

        Input_D <="0110";
        Reg_En <="001";
        Reset <='0';
        WAIT;

end process;

end behavioral;
```

Instruction Decoder

Design Source

```
entity Instruction_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;
           Imm_Val : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_Sel_A : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_Sel_B : out STD_LOGIC_VECTOR (2 downto 0);
           Add_Sub_Sel : out STD_LOGIC;
           Jump_add : out STD_LOGIC_VECTOR (2 downto 0);
           Jump_flag : out STD_LOGIC;
           Reg_Chk_Jmp : in STD_LOGIC_VECTOR (3 downto 0));

end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is
```

```
begin



process(Instruction, Reg_Chk_Jmp)
    begin

    Reg_en <= "000";
    Load_sel <= '0';
    Imm_Val <= "0000";
    Reg_Sel_A <= "000";
    Reg_Sel_B <= "000";
    Add_Sub_Sel <= '0';
    Jump_add <= "000";
    Jump_flag <= '0';



    If (Instruction(11)='0' and Instruction(10)='0') then     --ADD
        Reg_en <= Instruction (9 downto 7);
        Load_sel <= '0';
        Reg_Sel_A <= Instruction (9 downto 7);
        Reg_Sel_B <= Instruction (6 downto 4);
        Add_Sub_Sel <= '0';

    elsIf (Instruction(11)='0' and Instruction(10)='1') then --NEG
            Reg_en <= Instruction(9 downto 7);
            Load_sel <= '0';
            Reg_Sel_A <= "000";
            Reg_Sel_B <= Instruction (9 downto 7);
            Add_Sub_Sel <= '1';



     elsIf (Instruction(11)='1' and Instruction(10)='0') then --MOVI
                Reg_en <= Instruction(9 downto 7);
                Load_sel <= '1';
                Imm_Val <= Instruction(3 downto 0);
```

```
    elsIf (Instruction(11)='1' and Instruction(10)='1') then --JUMP
          Reg_en <= "000";
          Reg_Sel_A <= Instruction (9 downto 7);
         if (Reg_Chk_Jmp = "0000") then
            Jump_flag <= '1';
            Jump_add <= Instruction(2 downto 0);

         else
            Jump_flag <= '0';
         end if;

   end if;

end process;
end Behavioral;
```

Test Bench file

```
entity TB_Instruction_Decoder is
--  Port ( );
end TB_Instruction_Decoder;

architecture Behavioral of TB_Instruction_Decoder is
component Instruction_Decoder
    Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
       Reg_en : out STD_LOGIC_VECTOR (2 downto 0);
       Load_sel : out STD_LOGIC;
       Imm_Val : out STD_LOGIC_VECTOR (3 downto 0);
       Reg_Sel_A : out STD_LOGIC_VECTOR (2 downto 0);
       Reg_Sel_B : out STD_LOGIC_VECTOR (2 downto 0);
       Add_Sub_Sel : out STD_LOGIC;
       Jump_add : out STD_LOGIC_VECTOR (2 downto 0);
       Jump_flag : out STD_LOGIC;
       Reg_Chk_Jmp : in STD_LOGIC_VECTOR (3 downto 0));
```

```
end component;

signal Instruction : STD_LOGIC_VECTOR (11 downto 0);
signal Reg_en : STD_LOGIC_VECTOR (2 downto 0);
signal Load_sel : STD_LOGIC;
signal Imm_Val : STD_LOGIC_VECTOR (3 downto 0);
signal Reg_Sel_A : STD_LOGIC_VECTOR (2 downto 0);
signal Reg_Sel_B : STD_LOGIC_VECTOR (2 downto 0);
signal Add_Sub_Sel : STD_LOGIC;
signal Jump_add : STD_LOGIC_VECTOR (2 downto 0);
signal Jump_flag : STD_LOGIC;
signal Reg_Chk_Jmp : STD_LOGIC_VECTOR (3 downto 0);

begin
UUT: Instruction_Decoder
PORT MAP(
    Instruction => Instruction,
    Reg_en => Reg_en,
    Load_sel => Load_sel,
    Imm_Val => Imm_Val,
    Reg_Sel_A => Reg_Sel_A,
    Reg_Sel_B => Reg_Sel_B,
    Add_Sub_Sel => Add_Sub_Sel,
    Jump_add => Jump_add,
    Jump_flag => Jump_flag,
    Reg_Chk_Jmp => Reg_Chk_Jmp );

--210159G => 1100110100111101111
--210636R => 1100110110110011100

process
    begin
        Instruction <= "111101110010";
        Reg_Chk_Jmp <= "1100";
        WAIT FOR 500ns;
```

```
        Instruction <= "001100110110";
        Reg_Chk_Jmp <= "1111";
        WAIT;

end process;

end Behavioral;
```

Program ROM

Design Source

```
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
    "101100000011", -- MOVI R6, 3
    "101010000001", -- MOVI R5, 1
    "011010000000", -- NEG R5
    "001111100000", -- ADD R7, R6
    "001101010000", -- ADD R6, R5
    "111100000110", -- JZR R6, 6
    "110000000100" -- JZR R0, 4
    );

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

```
end Behavioral;
```

Test Bench file

```
entity TB_Program_ROM is
--  Port ( );
end TB_Program_ROM;

architecture Behavioral of TB_Program_ROM is

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

signal address: STD_LOGIC_VECTOR (2 downto 0);
signal data: STD_LOGIC_VECTOR (11 downto 0);

begin
UUT: Program_ROM
Port Map(
    address => address,
    data => data);

process
begin

    address <= "110";
    wait for 200ns;

    address <= "011";
    wait for 200ns;

    address <= "101";
    wait for 200ns;
```

```
    address <= "001";
    wait for 200ns;

    address <= "111";
    wait;
end process;

end Behavioral;
```

4-bit Add/Subtract unit

Design Source

```
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; --sel=0 ADD , sel=1 SUBSTRACT
           S : out STD_LOGIC_VECTOR (3 downto 0);
           C_Out : out STD_LOGIC;
           Z_Out : out STD_LOGIC;
           Overflow : out STD_LOGIC
            );

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;
           Overflow : 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,
        Overflow => Overflow);

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

end Behavioral;
```

Test Bench file

```
entity TB_ASU is
--  Port ( );
end TB_ASU;

architecture Behavioral of TB_ASU is
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; --sel=0 ADD , sel=1 SUBSTRACT
           S : out STD_LOGIC_VECTOR (3 downto 0);
           C_Out : out STD_LOGIC;
           Z_Out : out STD_LOGIC;
           Overflow : out STD_LOGIC
```

```
        );
end component;

signal A,B,S: STD_LOGIC_VECTOR(3 downto 0);
signal Sel, C_Out, Z_Out, Overflow : STD_LOGIC ;

begin
    UUT: Add_Sub_Unit
    Port Map(
    A => A,
    B => B,
    Sel => Sel,
    S => S,
    C_Out => C_Out,
    Z_Out => Z_Out,
    Overflow => Overflow );

process
begin

A <= "1111";
B <= "1110";
Sel <= '0';
wait for 200ns;

A <= "1110";
B <= "1111";
Sel <= '1';
wait for 200ns;

A <= "1100";
B <= "1000";
Sel <= '1';
wait for 200ns;

A <= "0100";
```

```
B <= "0011";
Sel <= '0';
wait for 200ns;

A <= "0110";
B <= "0011";
Sel <= '0';
wait;

end process;

end Behavioral;
```

3-bit Program Counter (PC)

Design Source

```
entity Program_Counter is
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
           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 (Reset = '1') then
        Q <= "000";
    else
        if (rising_edge(Clk)) then
          Q <= D;
          end if;
    end if;
```

```
end process;

end Behavioral;
```

Test Bench file

```
entity TB_Program_Counter is
--  Port ( );
end TB_Program_Counter;

architecture Behavioral of TB_Program_Counter is
component Program_Counter
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
           Reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal D,Q : STD_LOGIC_VECTOR(2 downto 0);
signal CLK,Reset : STD_LOGIC := '0';


begin
UUT: Program_Counter
Port Map(
    D => D,
    Reset => Reset,
    CLK => CLK,
    Q => Q);

process
begin

wait for 10ns;
Clk <= NOT(Clk);
```

```
end process;

process
begin
   D <= "110";
   WAIT FOR 100 ns;

   D <= "011";
   WAIT FOR 100 ns;

   D <= "001";
   WAIT FOR 100 ns;

   D <= "010";
   WAIT FOR 100 ns;

   D <= "101";
   WAIT FOR 100 ns;

   D <= "111";
   WAIT;
end process;

end Behavioral;
```

3-bit adder

Design Source

```
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));
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,FA2_C : 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 => FA2_C);

end Behavioral;
```

Test Bench file

```
entity TB_Adder_3bit is
--  Port ( );
end TB_Adder_3bit;

architecture Behavioral of TB_Adder_3bit is
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));
end component;

signal A, S : STD_LOGIC_VECTOR(2 downto 0);
signal C_in : STD_LOGIC := '0';

begin
UUT: Adder_3_Bit
Port Map(
    A => A,
    C_in => C_in,
    S =>S );

process
    begin
        A <= "101";
        WAIT FOR 200ns;

        A <= "011";
```

```
        WAIT FOR 200ns;

        A <= "001";
        WAIT FOR 200ns;

        A <= "100";
        WAIT FOR 200ns;

        A <= "110";
        WAIT;

end process;
end Behavioral;
```

8-way 4-bit multiplexer

Design Source

```
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;
```

Test Bench file

```
entity TB_Mux_8way_4bit is
--  Port ( );
end TB_Mux_8way_4bit;

architecture Behavioral of TB_Mux_8way_4bit is
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;

signal R0, R1, R2, R3, R4, R5, R6, R7, Q : STD_LOGIC_VECTOR( 3 downto 0);
signal S : STD_LOGIC_VECTOR(2 downto 0);
begin
```

```
UUT: MUX_8_way_4_Bit
Port Map(
    R0 => R0,
    R1 =>  R1,
    R2 =>  R2,
    R3 =>  R3,
    R4 =>  R4,
    R5 =>  R5,
    R6 =>  R6,
    R7 =>  R7,
    S  =>  S ,
    Q  =>  Q );

process
   begin
      R0 <= "1100";
      R1 <= "1001";
      R2 <= "0011";
      R3 <= "0110";
      R4 <= "1101";
      R5 <= "1010";
      R6 <= "0100";
      R7 <= "1001";
      S  <= "001";
      WAIT FOR 500ns;

      R0 <= "1100";
      R1 <= "1101";
      R2 <= "1011";
      R3 <= "0011";
      R4 <= "1100";
      R5 <= "1001";
      R6 <= "0110";
      R7 <= "1101";
      S  <= "011";
      WAIT;
```

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

2-way 4-bit multiplexer

Design Source

```
entity MUX_2_way_4_Bit is
    Port ( I0 : in STD_LOGIC_VECTOR (3 downto 0);
           I1 : 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(I0, I1, S)
    begin
        case S is
            when '0' => Q <= I0;
            when '1' => Q <= I1;
            when others => Q <= "ZZZZ";
        end case;

     end process;
end Behavioral;
```

Test Bench file

```
entity TB_Mux_2way_4bit is
--  Port ( );
end TB_Mux_2way_4bit;

architecture Behavioral of TB_Mux_2way_4bit is
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;
       Q : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal Adder_Sub_Out, Imd_Value,Q : STD_LOGIC_VECTOR(3 downto 0);
signal S : STD_LOGIC;

begin
UUT: MUX_2_way_4_Bit
Port Map(
    I0 => Adder_Sub_Out,
    I1 =>  Imd_Value,
    S => S,
    Q => Q
    );

process

begin
    Adder_Sub_Out <= "1100";
    Imd_Value <= "1101";
    S <= '0';
    WAIT FOR 200ns;

    S <= '1';
    WAIT FOR 200ns;

    Adder_Sub_Out <= "1011";
    Imd_Value <= "0011";
    S <= '0';
    WAIT FOR 200ns;

    Adder_Sub_Out <= "0011";
    Imd_Value <= "1111";
    S <= '0';
```

```
    WAIT FOR 200ns;

    S <= '1';
    WAIT;

end process;

end Behavioral;
```

2-way 3-bit multiplexer

Design Source

```
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;
```

Test Bench file

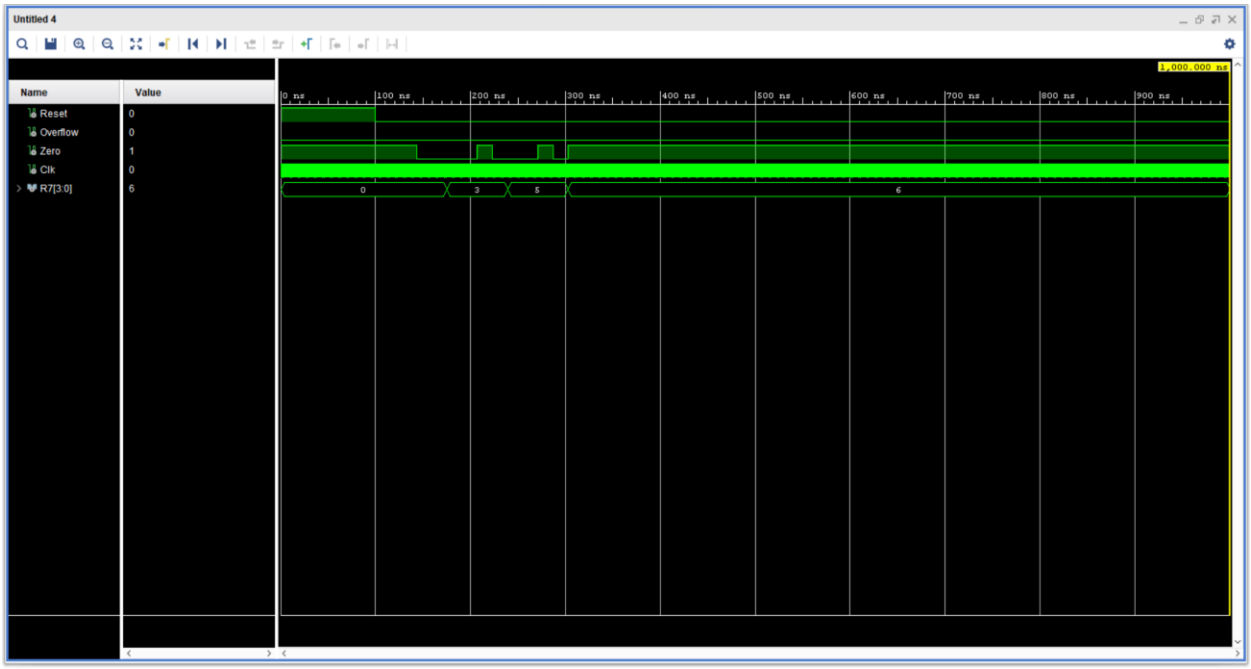
```
entity TB_Mux_2way_3bit is
```

```
--  Port ( );
end TB_Mux_2way_3bit;

architecture Behavioral of TB_Mux_2way_3bit is

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 I0,I1,Q : STD_LOGIC_VECTOR(2 downto 0);
signal S : STD_LOGIC;

begin
UUT: MUX_2_way_3_Bit
Port Map(
    I0 => I0,
    I1 => I1,
    S => S,
    Q => Q
    );

process

begin
    I0 <= "110";
    I1 <= "011";
    S <= '0';
    WAIT FOR 250ns;

    S <= '1';
    WAIT FOR 250ns;

    I0 <= "101";
```

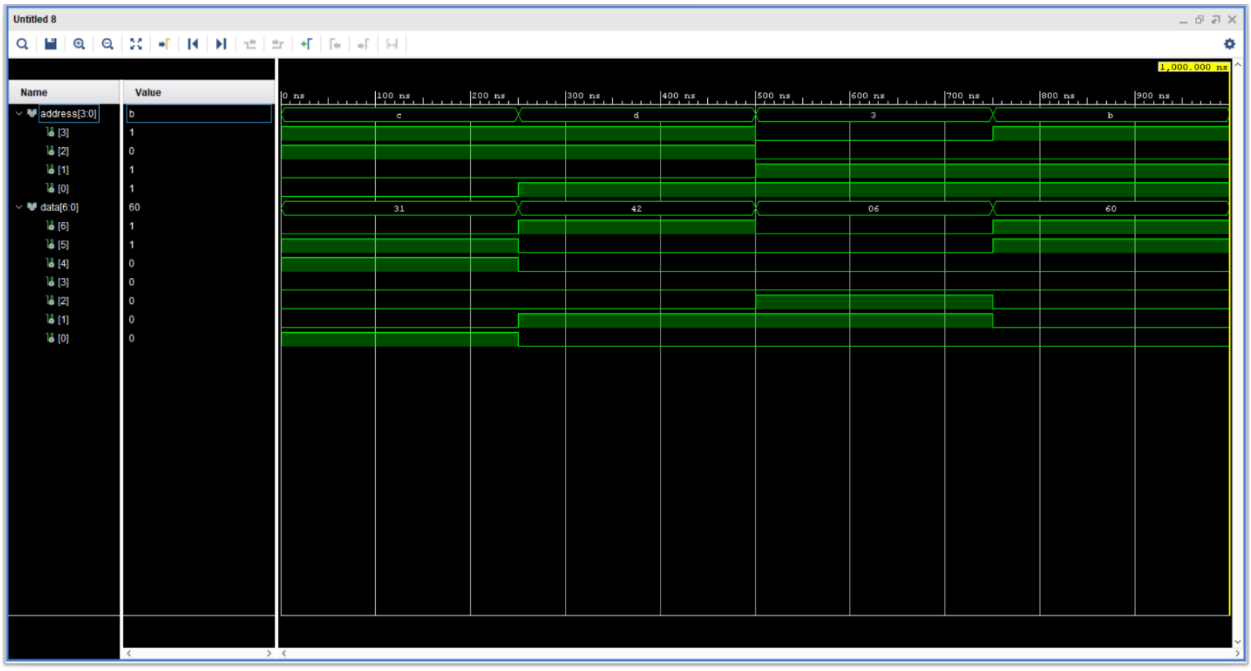
```
    I1 <= "001";
    S <= '0';
    WAIT FOR 250ns;

    I0 <= "100";
    I1 <= "111";
    S <= '1';
    WAIT FOR 250ns;

    S <= '1';
    WAIT;

end process;
end Behavioral;
```

Timing diagrams

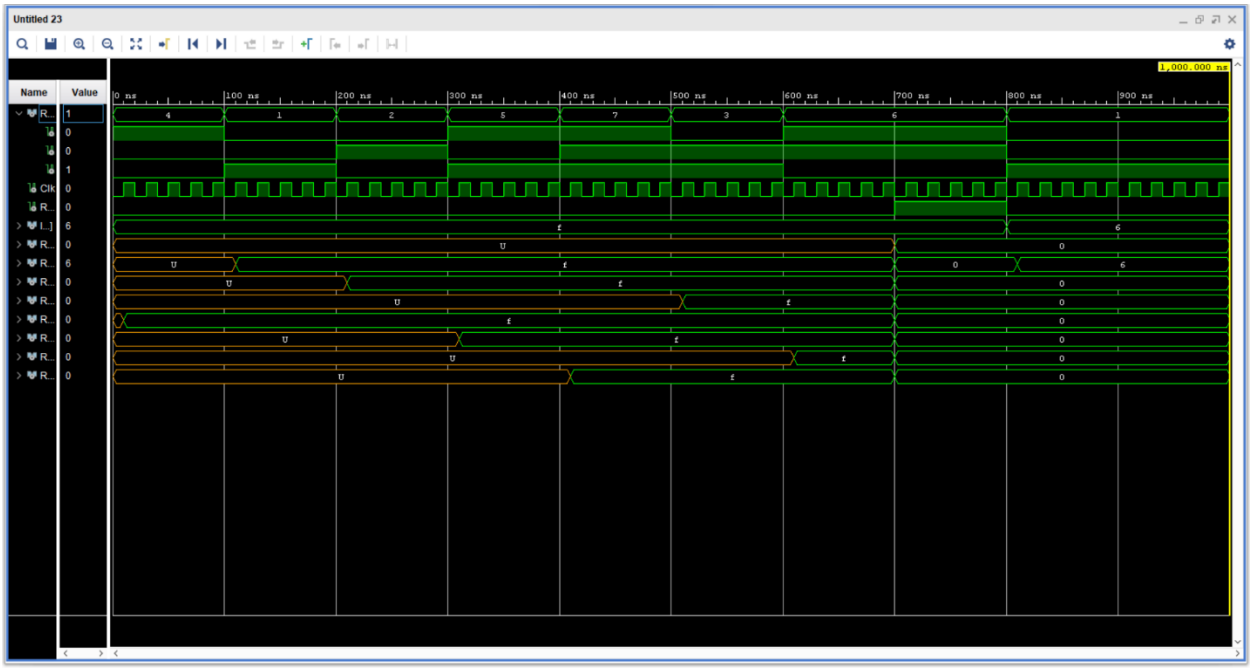
Nano Processor



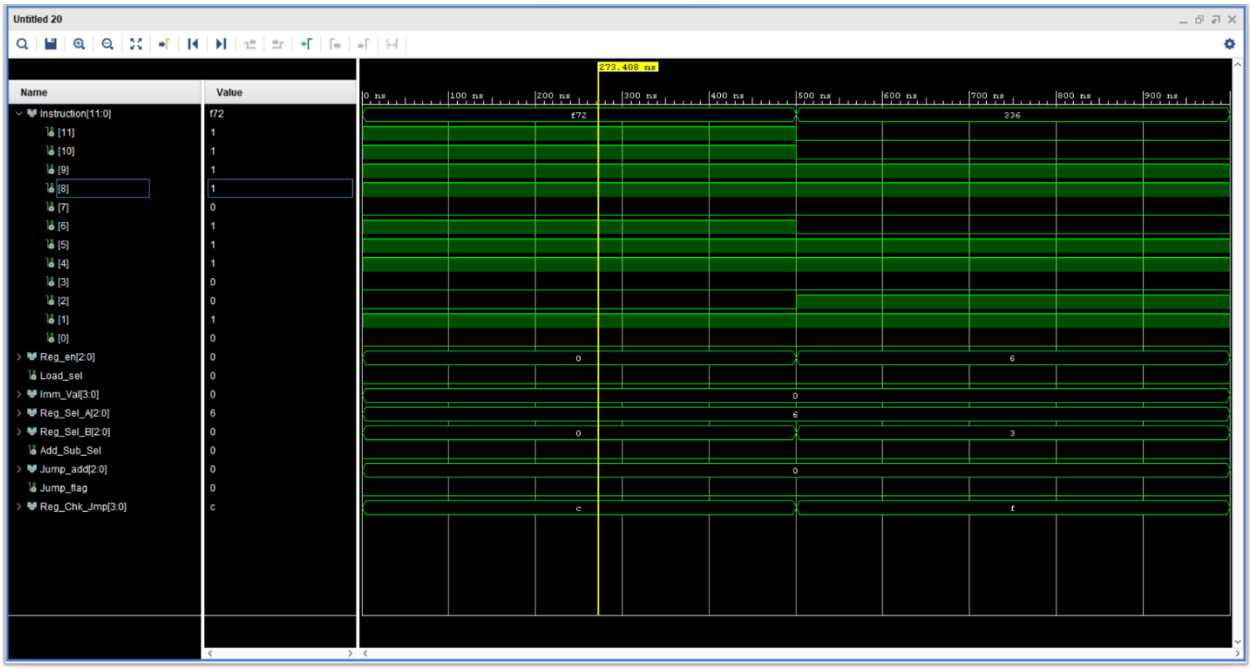
7 Segments Display



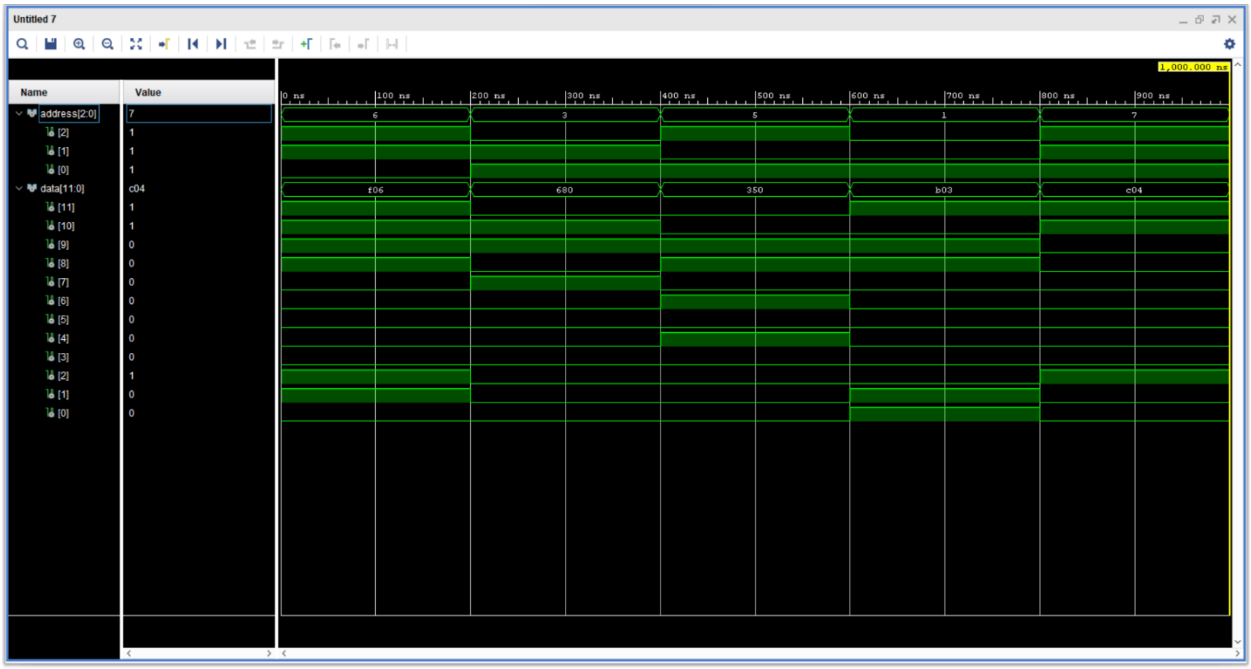
Register Bank



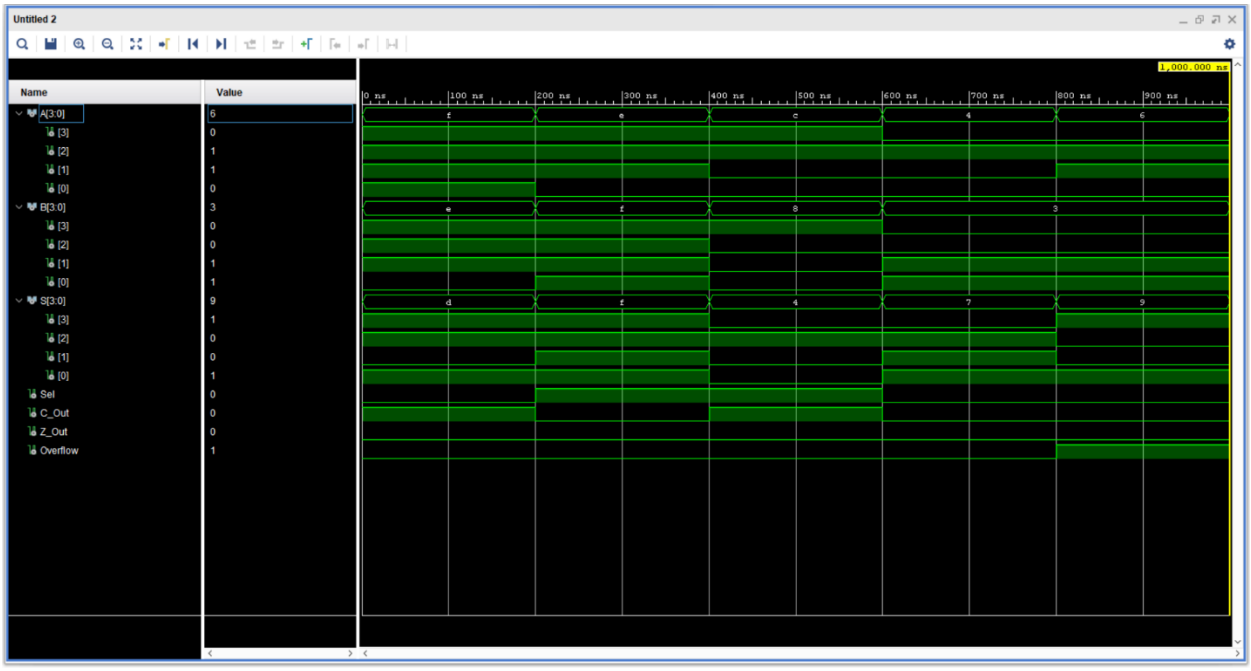
Instruction Decoder



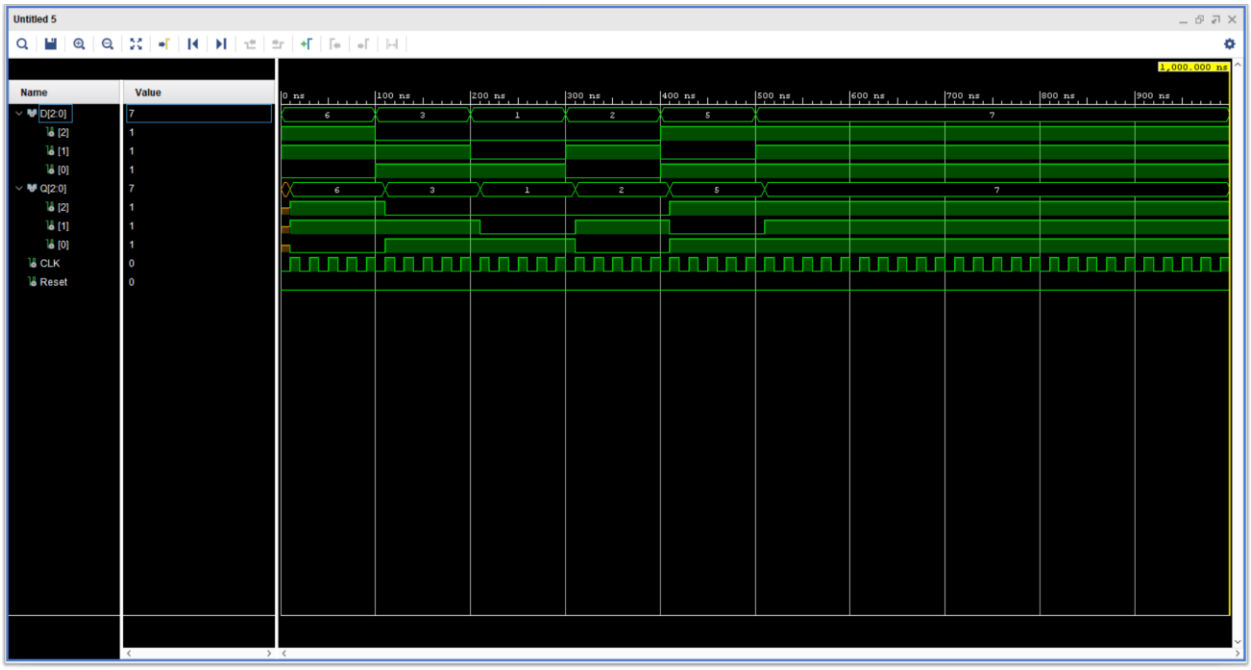
Program ROM



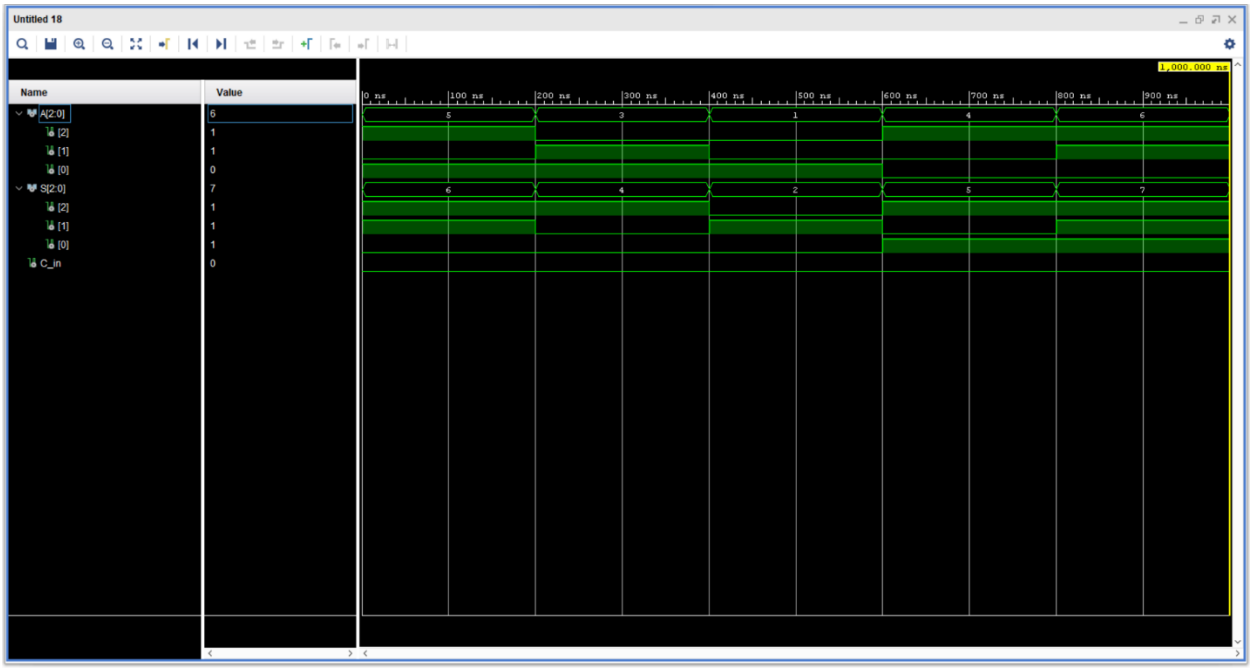
4-bit Add/Subtract unit



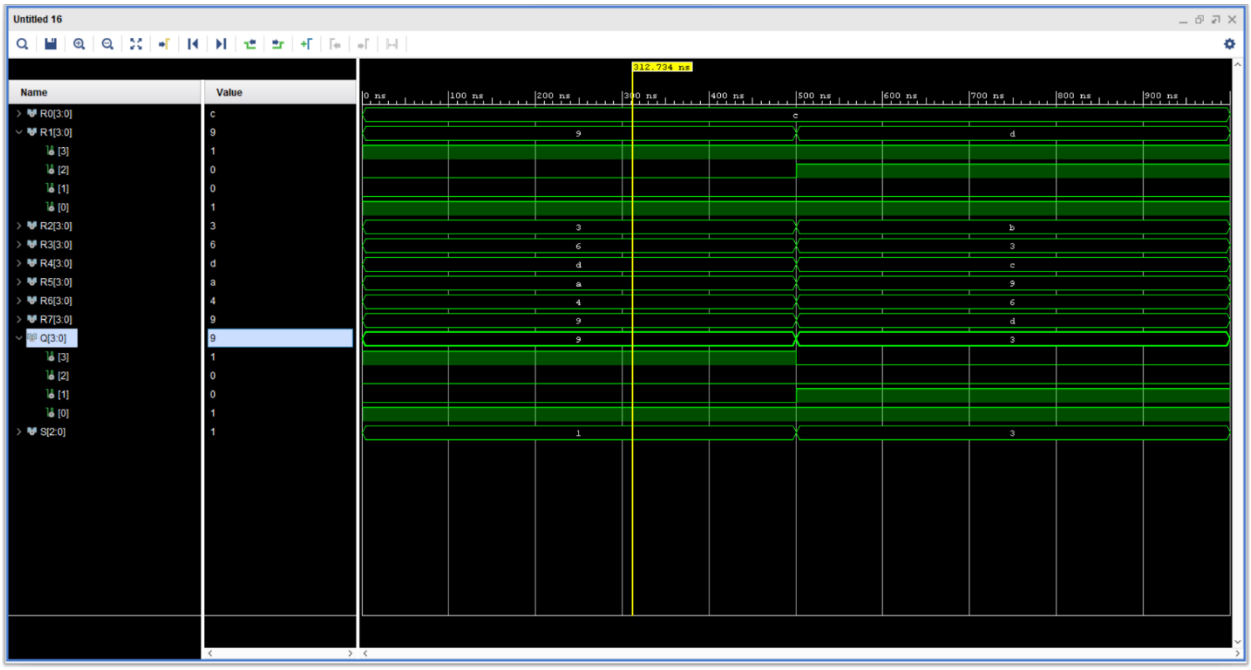
3-bit Program Counter (PC)



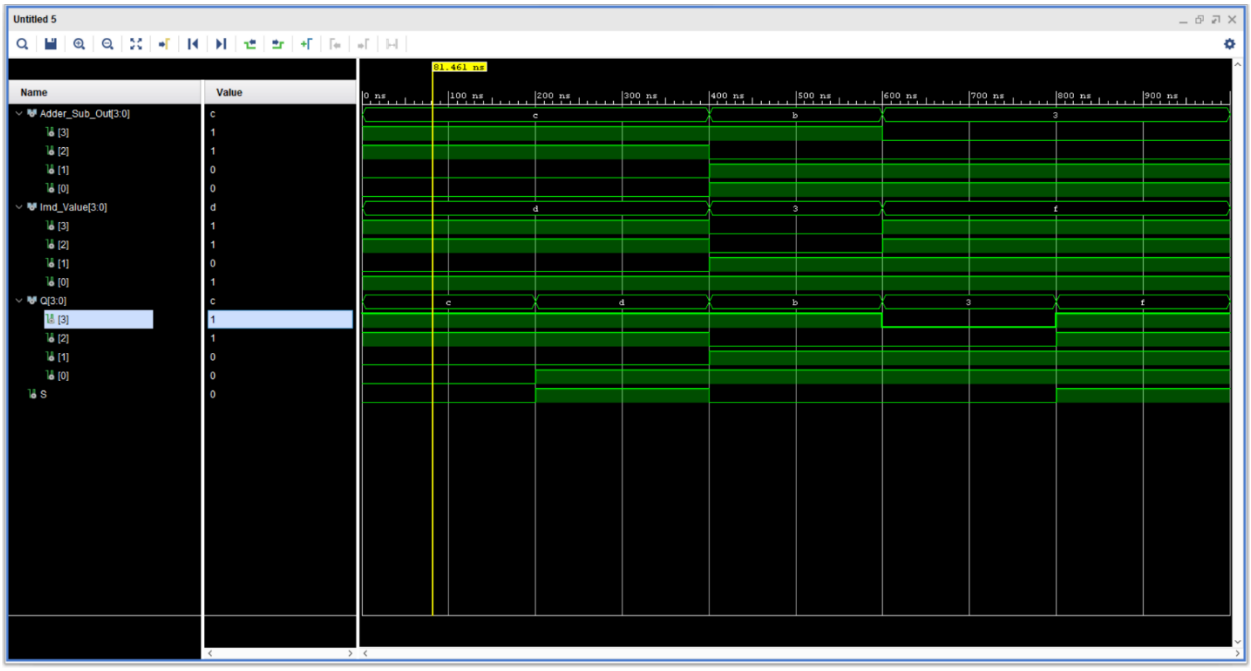
3-bit Adder



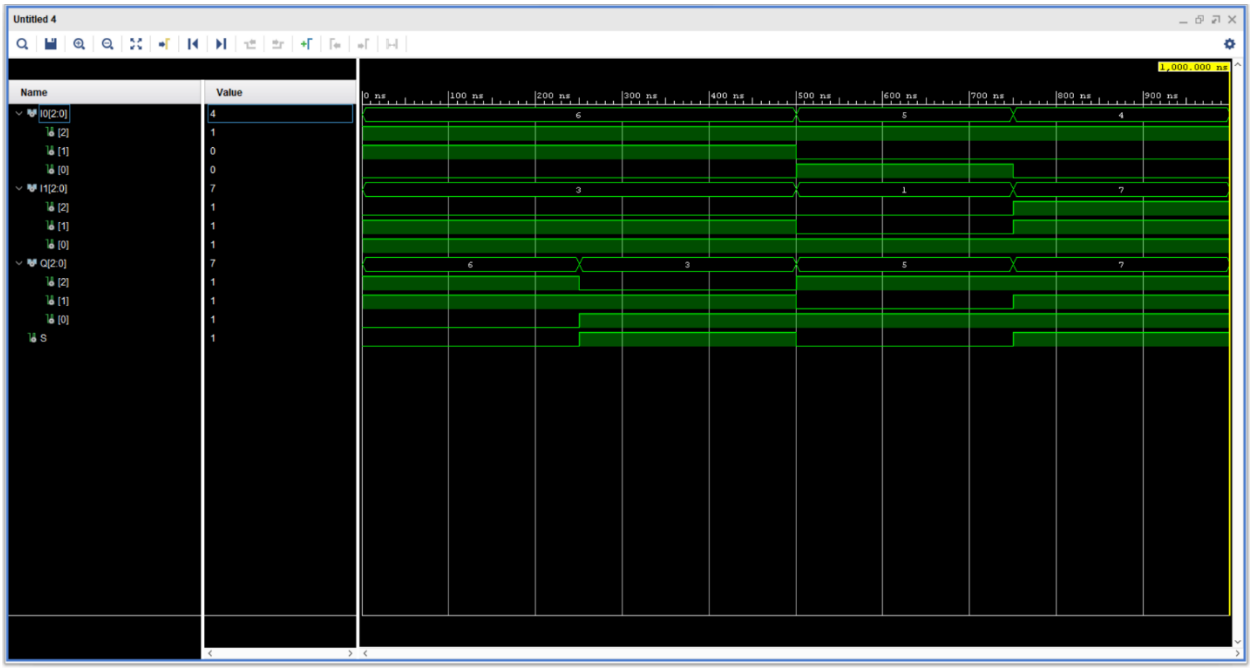
8-way 4-bit multiplexer



2-way 4-bit multiplexer



2-way 3-bit multiplexer



Conclusions

Through this project, we could get valuable experience in designing and developing a 4-bit nano processor. We learned about the significance of comprehensive planning, design, testing, and verification. Additionally, we learned about many processor parts, including the arithmetic unit, multiplexers, and tri-state busses, and how they work together to execute instructions.

Overall, this project was a great learning experience that provided us with a solid foundation in processor design and development.

The contribution of each team member

The project was done by both of us together. We first discussed how each component of the processor works and got a brief idea about them. Then we started coding the design source file and the test bench file of each component and simulated them one by one. Since both of us are in the same boarding place we worked on the same machine and coded together taking turns, discussing the problems we encounter and finding solutions together. We did the project for about three weeks and worked about approximately 80 hours.