CS1050 Computer Organization and Digital Design
Lab 9-10 – Nanoprocessor Design Competition

Team members(Group 81):

- Akshiya R. 210025T
- Praveenasarma B. 210493A

Lab task

This lab focused on the design of a 4-bit Nanoprocessor capable of efficiently executing four instructions. In order to create the circuit, we extended and improved several components that were developed in the previous labs. These components include a 4-bit Add/Subtract unit, a 3-bit adder, a 3-bit Program Counter (PC), k-way b-bit multiplexers, a Register Bank, a Program ROM, an Instruction Decoder, a 7-Segment Display, and a slow clock.

To simplify the design, we utilized 3, 4, and 12-bit buses for connecting the various components instead of using numerous wires. Since the nanoprocessor only understands machine language, we provided the instructions as binary values and hardcoded the program into the Program ROM. Additionally, we used a slow clock to drive the nanoprocessor, enabling us to observe the changes as the program executed at a reduced clock rate. We also conducted simulations to verify the functionality of each component by comparing their expected inputs and outputs.

As this was a team project, we distributed the workload among the two members equally. If one person did one component, otherone verified whether it is correct or not. We made some mistakes and by discussing them, we corrected them. Upon completion of the lab, we successfully designed and developed a 4-bit Arithmetic Unit capable of adding and subtracting signed integers, k-way b-bit Multiplexers, a Program ROM, and an Instruction Decoder to activate the necessary components within the Nano Processor. We verified the functionality through simulations and implemented the design on a BASYS3 board.

Assembly program and its machine code representation

	Assembly program	**Machine code representation**

Add 3 by looping	MOVI R4,3
	MOVI R6,1
	NEG R6
	ADD R7,R4
	ADD R4,R6
	JZR R4,7
	JZR R0,3
	JZR R4,7
		"101000000011"
		"101100000001"
		"011100000000"
		"001111000000"
		"001001100000"
		"111000000111"
		"110000000011"
		"111000000111"

Program to display 10 to 0	MOVI R7,10
	MOVI R2,1
	NEG R2
	ADD R7,R2
	JZR R7,7
	JZR R0,3
	JZR R1,7
	JZR R1,6
		"101110001010"
		"100100000001"
		"010100000000"
		"001110100000"
		"111110000111"
		"110000000011"
		"110010000111"
		"110010000110"

Move 1,2,3 to three registers R7,R6,R5 respectively. Add R6 to R7 and store in R7. Add R5 to R7 and store in R7. Adding 1,2,3 by storing each value in each register.	MOVI R7,1
	MOVI R6,2
	MOVI R5,3
	ADD R7,R6
	ADD R7,R5
	JZR R0,5
	JZR R0,5
	JZR R0,7
		"101110000001"
		"101100000010"
		"101010000011"
		"001111100000"
		"001111010000"
		"110000000101"
		"110000000101"
		"110000000111"

Contribution of team members

Team Member	Contribution	Time spent
Akshiya R.	3-bit adder 3-bit Program Counter (PC) 2-way 3-bit multiplexer Program ROM	25 hours
Praveenasarma B.	2-way 4-bit multiplexer 8-way 4-bit multiplexers 4-bit Add/Subtract unit Register Bank	25 hours
Together	Instruction Decoder NanoProcessor(Combined all components and debugged)	20 hours

Components of Nanoprocessor and their functions

Component	**Function**	**Structure**
4-bit Add/Subtract unit	When two four bits are given as input, it will add or subtract one from another and give the output. It will indicate if there is an overflow or if the output is zero.	It has four full adders within it. Add_Sub_Select signal decides whether to do addition or subtraction.
3-bit adder	It increments the address by one.	It has 3 full adders inside it.
3-bit Program Counter (PC)	Carries the address of next instruction to be executed	It has 3 D-Flip Flops inside.
2-way 3-bit multiplexer	It will decide whether to select either address from 3-bit adder or jump address from the instruction decoder.	
2-way 4-bit multiplexer	It takes either immediate value or value from the	

	Add/Sub Unit based on the signal from load select. Then the value is passed to the register bank.	
8-way 4-bit multiplexer	It will select, from which register the data has to be taken and passed to the add/sub unit.	It has three 8-to-1 Multiplexers.
Register Bank	4 bit data get stored into the registers in the register bank	There are 8 registers and a 3-to-8 decoder inside it. 3-to-8 decoder selects the register in which the input values should be stored.
Program ROM	It stores the assembly program. In each ROM, 12 bit instructions are stored.	It has 8 ROMs inside. Memory select selects which program has to be executed and sends the instruction to the instruction decoder.
Instruction Decoder	It decodes the instructions and transmits signals to control various components of the system. These signals include register enable for 3-to-8 decoder in the register bank, load select for managing a 2-way 4-bit multiplexer, add/sub select for a 4-bit add/sub unit, register select for choosing a specific 8-way 4-bit multiplexer, and jump flag for a 2-way 3-bit multiplexer. It also sends the values such as immediate value and address to jump.	It has a 2-to-4 decoder to decode and find which operation has to be executed.

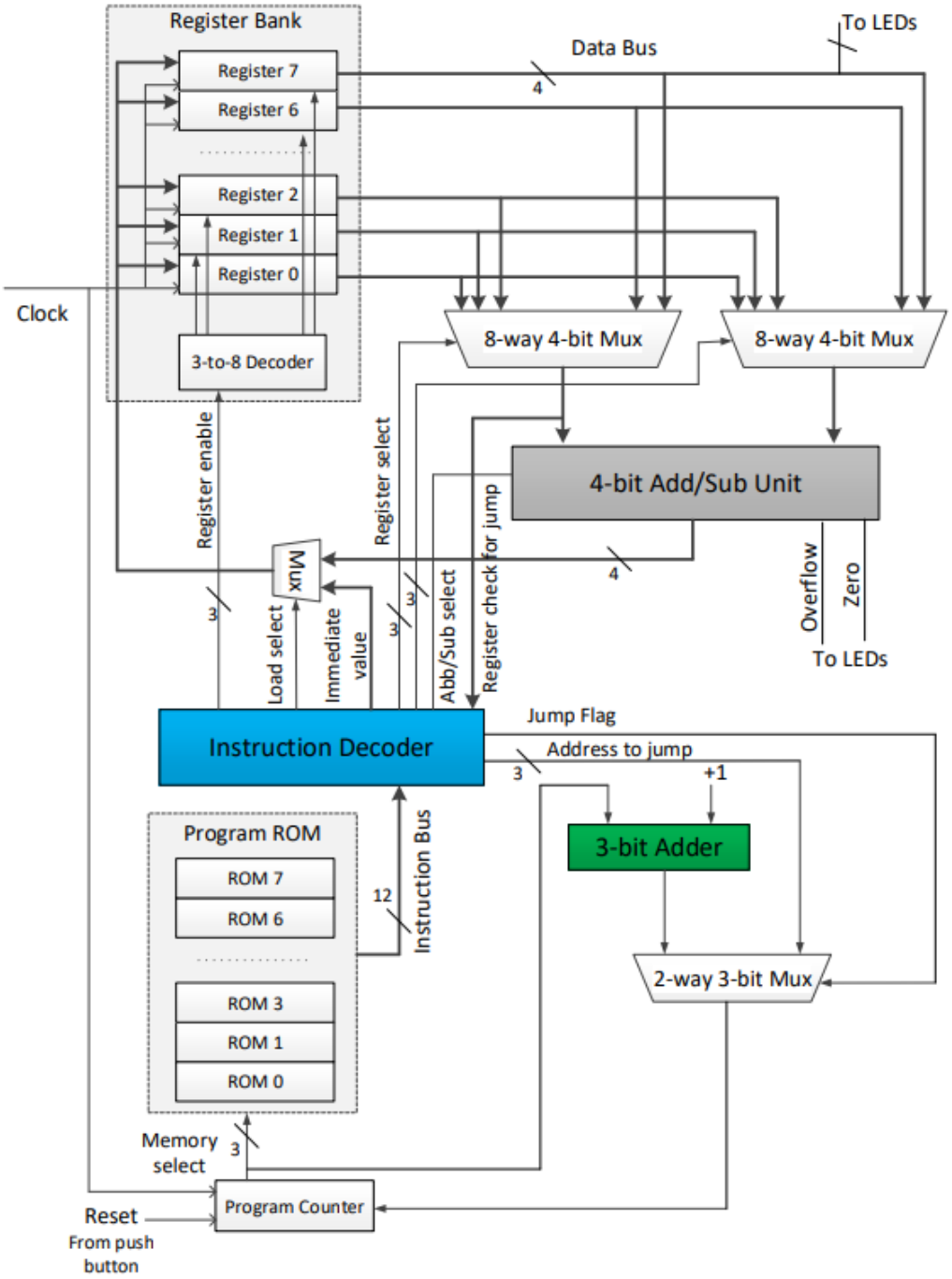
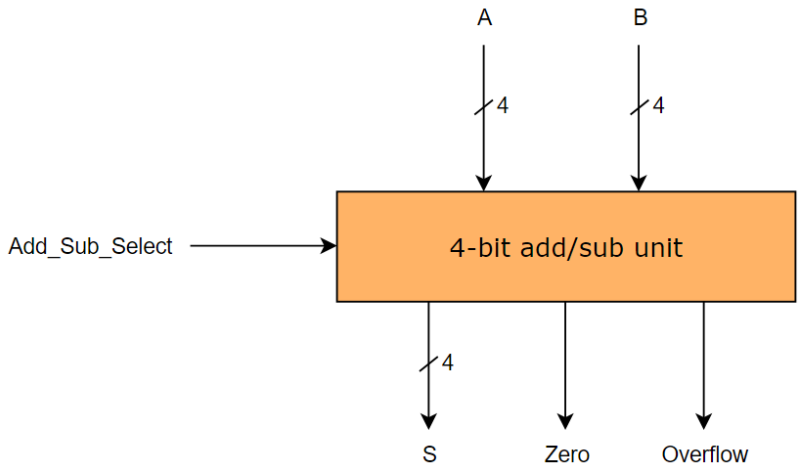


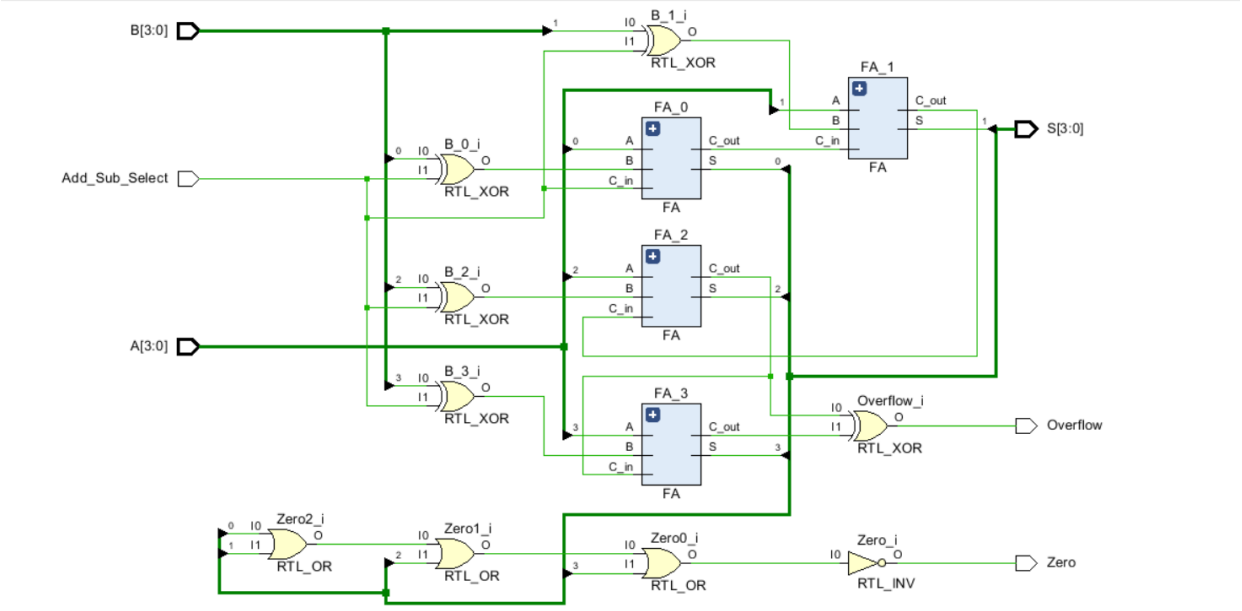
Figure 1 – High-level diagram of the nanoprocessor.

Image from: Lab file

1. 4-bit add/sub unit



Schematic



Add_Sub_4.vhd

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

entity Add_Sub_4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Add_Sub_Select : in STD_LOGIC;
           Zero:out std_logic;
           Overflow :out STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0));
end Add_Sub_4;

architecture Behavioral of Add_Sub_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_C,FA1_C,FA2_C,FA3_C: STD_LOGIC;
SIGNAL B_0,B_1,B_2,B_3: STD_LOGIC;
SIGNAL FA_Sum : STD_LOGIC_VECTOR (3 downto 0);

begin
    B_0 <= B(0) XOR Add_Sub_Select;
    B_1 <= B(1) XOR Add_Sub_Select;
    B_2 <= B(2) XOR Add_Sub_Select;
    B_3 <= B(3) XOR Add_Sub_Select;

    FA_0 : FA
    PORT MAP( A => A(0),
              B => B_0,
              C_in => Add_Sub_Select,
              S => FA_Sum(0),
              C_out => FA0_C);

    FA_1 : FA
    PORT MAP( A => A(1),
              B => B_1,
```

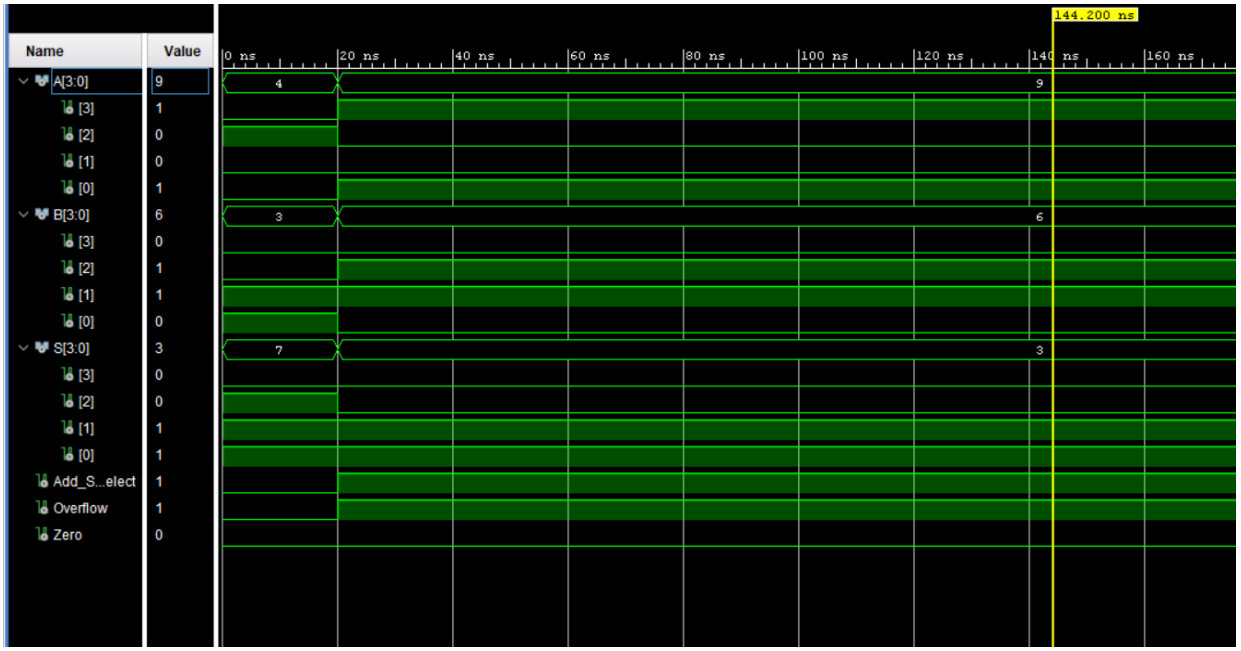
```
            C_in => FA0_C,
            S => FA_Sum(1),
            C_out => FA1_C);

    FA_2 : FA
    PORT MAP( A => A(2),
            B => B_2,
            C_in => FA1_C,
            S => FA_Sum(2),
            C_out => FA2_C);

   FA_3 : FA
   PORT MAP( A => A(3),
            B => B_3,
            C_in => FA2_C,
            S => FA_Sum(3),
            C_out => FA3_C);

   Overflow <= FA2_C xor FA3_C;
   Zero <= not(FA_Sum(0) or FA_Sum(1) or FA_Sum(2) or FA_Sum(3));
   S <= FA_Sum;
end Behavioral;
```

TB_Add_Sub_4.vhd

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

entity TB_Add_Sub_4 is
--  Port ( );
end TB_Add_Sub_4;

architecture Behavioral of TB_Add_Sub_4 is
component Add_Sub_4
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Add_Sub_Select : in STD_LOGIC;
           Zero:out std_logic;
           Overflow :out STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0));
end component;
SIGNAL A,B,S : STD_LOGIC_VECTOR (3 downto 0);
    SIGNAL Add_Sub_Select,Overflow,Zero : STD_LOGIC;
```

```
begin
    uut: Add_Sub_4 PORT MAP (
          A => A,
          B => B,
          Add_Sub_Select => Add_Sub_Select,
          S => S,
          Zero => Zero,
          Overflow => Overflow
        );
process
   begin
-- Group Members Index numbers' binary form
-- 210025T      11 0011 0100 0110 1001
-- 210493A      11 0011 0110 0011 1101
-- unique 4bit numbers from index numbers=> 0011, 0100, 0110, 1001,1101
        A <= "0100";
        B <= "0011";
        Add_Sub_Select <= '0';
        wait for 20ns;

        A <= "1001";
        B <= "0110";
        Add_Sub_Select <= '1';
        wait;
        end process;
end Behavioral;
```

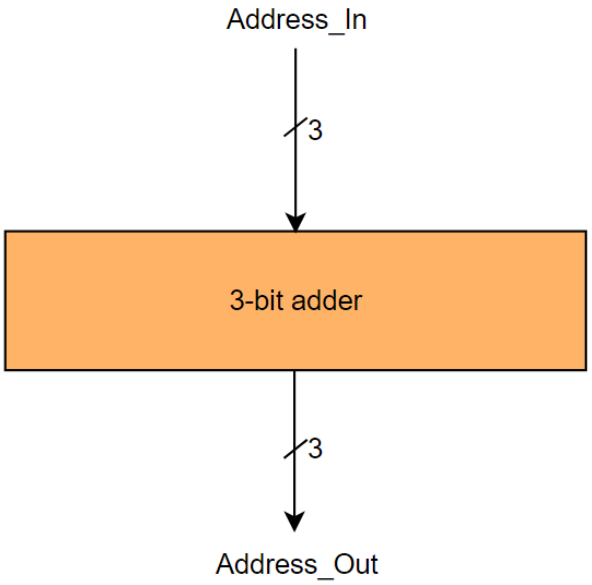
Timing diagram



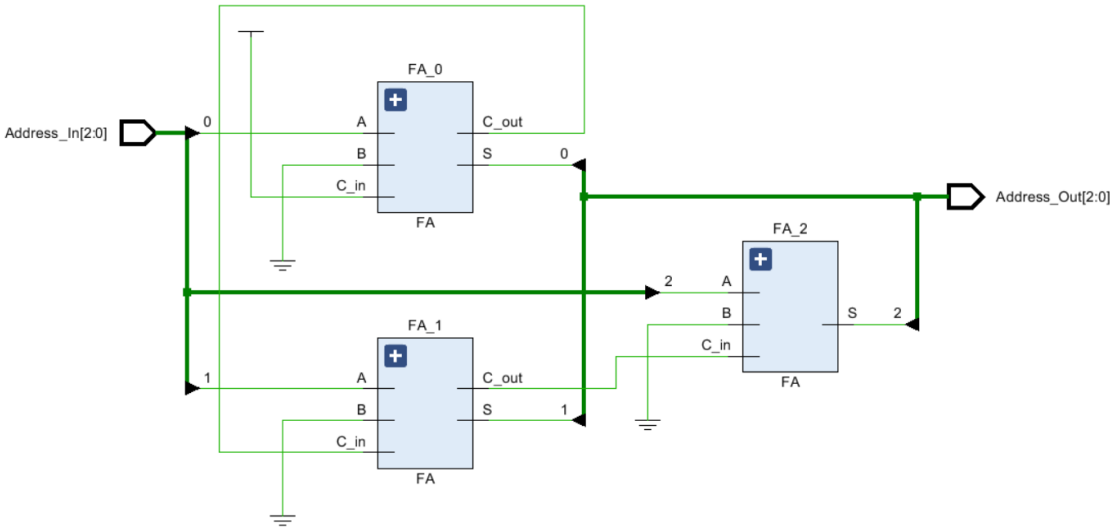
Note: Timing diagram can be verified by the following way:

If Add_Sub_Select is zero, inputs A and B will be added. If Add_Sub_Select is one, B will be subtracted from A.

2. 3-bit adder



Schematic



Adder_3.vhd

```
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 Adder_3 is
    Port ( Address_In : in STD_LOGIC_VECTOR (2 downto 0);
           Address_Out : out STD_LOGIC_VECTOR (2 downto 0));
end Adder_3;

architecture Behavioral of Adder_3 is
COMPONENT FA
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC := '0';
           C_in : in STD_LOGIC;
           S : out STD_LOGIC;
           C_out : out STD_LOGIC);
END COMPONENT;

signal FA_S, FA_C : std_logic_vector(2 downto 0);

begin
    FA_0 : FA
        PORT MAP( A => Address_In(0),
                  C_in => '1',
                  S => FA_S(0),
                  C_out => FA_C(0));
    FA_1 : FA
        PORT MAP( A => Address_In(1),
                  C_in => FA_C(0),
                  S => FA_S(1),
                  C_out => FA_C(1));
    FA_2 : FA
        PORT MAP( A => Address_In(2),
```

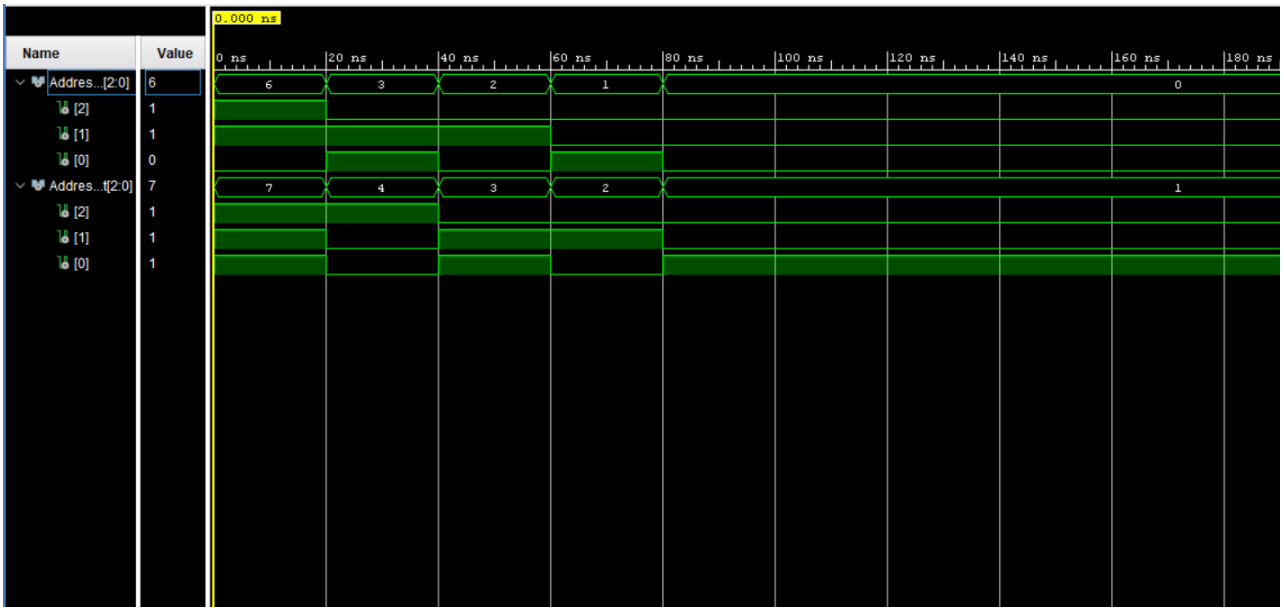
```
                C_in => FA_C(1),
                S => FA_S(2),
                C_out => FA_C(2));

Address_Out <= FA_S;

end Behavioral;
```

TB_Adder_3.vhd

```
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 TB_Adder_3 is
--  Port ( );
end TB_Adder_3;

architecture Behavioral of TB_Adder_3 is
component Adder_3
    Port ( Address_In : in STD_LOGIC_VECTOR (2 downto 0);
           Address_Out : out STD_LOGIC_VECTOR (2 downto 0));
end component;
SIGNAL Address_In,Address_Out : STD_LOGIC_VECTOR (2 downto 0);

begin
    uut: Adder_3 PORT MAP (
          Address_In => Address_In,
          Address_Out => Address_Out
        );

        process
        begin
        -- Group Members Index numbers' binary form
```

```
        -- 210025T      110 011 010 001 101 001
        -- 210493A      110 011 011 000 111 101
        -- unique 3bit numbers from index numbers for Address_In=>110,
011, 010, 001, 000, 101, 111

        Address_In <= "110";
        wait for 20ns;

        Address_In <= "011";
        wait for 20ns;

        Address_In <= "010";
        wait for 20ns;

        Address_In <= "001";
        wait for 20ns;

        Address_In <= "000";
        wait;
        end process;


end Behavioral;
```

Timing diagram

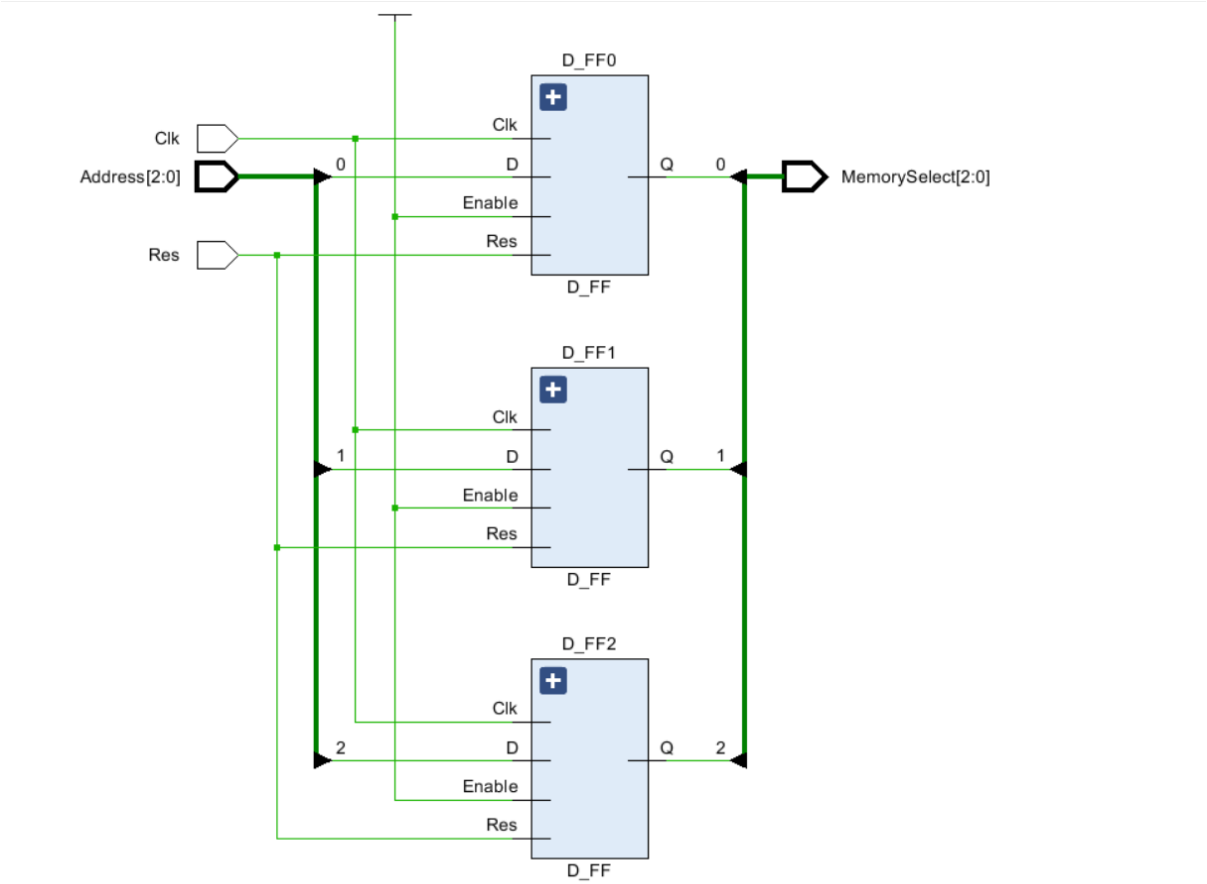


Note: Timing diagram can be verified by the following way:

Address_Out will be one more than Address_In.

3. 3-bit Program Counter (PC)

Schematic



Program_Counter_3.vhd

```
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;
```

```vhdl
--use UNISIM.VComponents.all;

entity Program_Counter_3 is
    Port ( Address : in STD_LOGIC_VECTOR (2 downto 0);
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           MemorySelect : out STD_LOGIC_VECTOR (2 downto 0));
end Program_Counter_3;

architecture Behavioral of Program_Counter_3 is

component D_FF
    port (
    D : in STD_LOGIC;
    Res: in STD_LOGIC;
    Clk : in STD_LOGIC;
    Enable: in std_logic:='1';
    Q : out STD_LOGIC;
    Qbar : out STD_LOGIC);
end component;


begin
    D_FF0 : D_FF
        port map (
            D => Address(0),
            Res => Res,
            Clk => Clk,
            Q => MemorySelect(0)
            );

    D_FF1 : D_FF
        port map (
            D => Address(1),
            Res => Res,
            Clk => Clk,
            Q => MemorySelect(1)
            );

    D_FF2 : D_FF
         port map (
            D => Address(2),
            Res => Res,
```

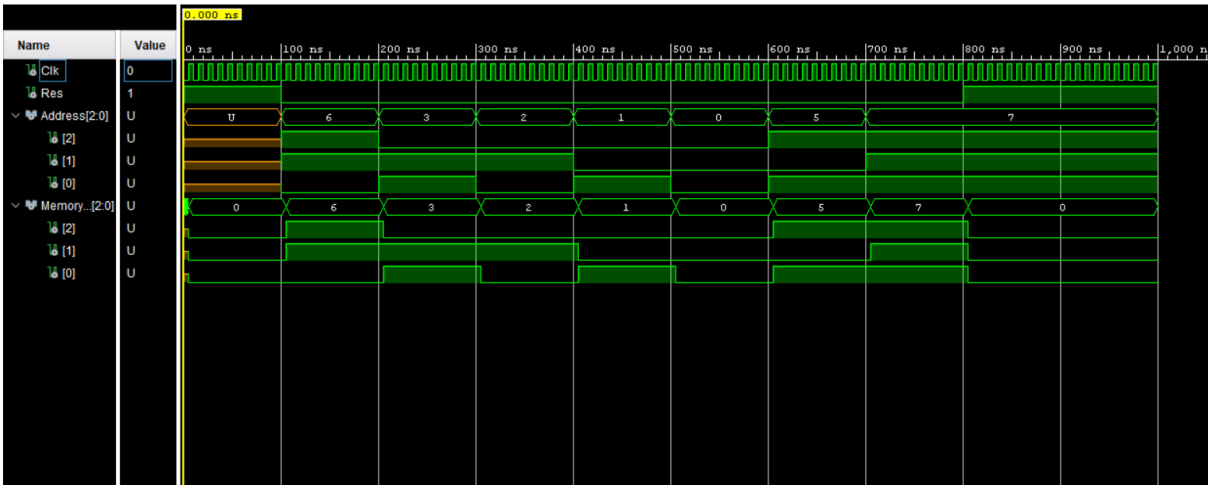
```
            Clk => Clk,
            Q => MemorySelect(2)
            );

end Behavioral;
```

TB_Program_Counter_3.vhd

```
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 TB_Program_Counter_3 is
--  Port ( );
end TB_Program_Counter_3;

architecture Behavioral of TB_Program_Counter_3 is
component Program_Counter_3 is
    Port ( Address : in STD_LOGIC_VECTOR (2 downto 0);
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           MemorySelect : out STD_LOGIC_VECTOR (2 downto 0));
 end component;
 signal Clk : STD_LOGIC := '0';
 signal Res: STD_LOGIC;
 signal Address,MemorySelect: STD_LOGIC_VECTOR(2 downto 0);
begin
    UUT: Program_Counter_3
        PORT MAP(
        Clk => Clk,
        Address => Address,
        Res => Res,
        MemorySelect => MemorySelect
```

```
        );
    process
            begin
            wait for 5ns;
            Clk <= Not(Clk);
            end process;

-- Group Members Index numbers' binary form
-- 210025T      110 011 010 001 101 001
-- 210493A      110 011 011 000 111 101
-- unique 3 bit numbers from index numbers for  =>110, 011, 010, 001,
000, 101, 111
process
      begin
            Res <= '1';
            wait for 100ns;

            Res <= '0';
            Address <= "110";
            wait for 100ns;

            Address <= "011";
            wait for 100ns;

            Address <= "010";
            wait for 100ns;

            Address <= "001";
            wait for 100ns;

            Address <= "000";
            wait for 100ns;

            Address <= "101";
            wait for 100ns;

            Address <= "111";
            wait for 100ns;

            Res <= '1';
            wait;
end process;
end Behavioral;
```

Timing diagram

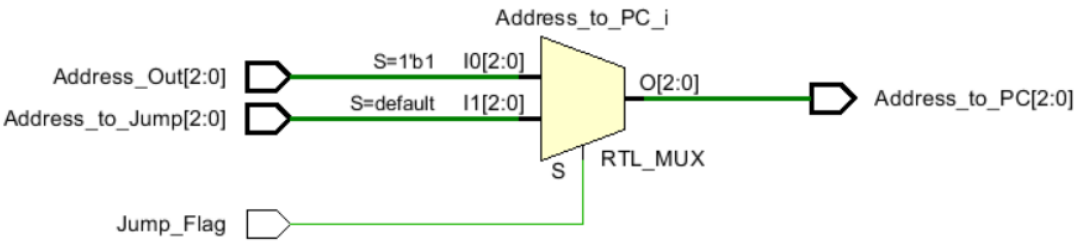


Note: Timing diagram can be verified by the following way:

Input has to be equal to the output. Address and MemorySelect values have to be the same.

4. 2-way 3-bit multiplexer

Schematic



Multiplexer_2_3.vhd

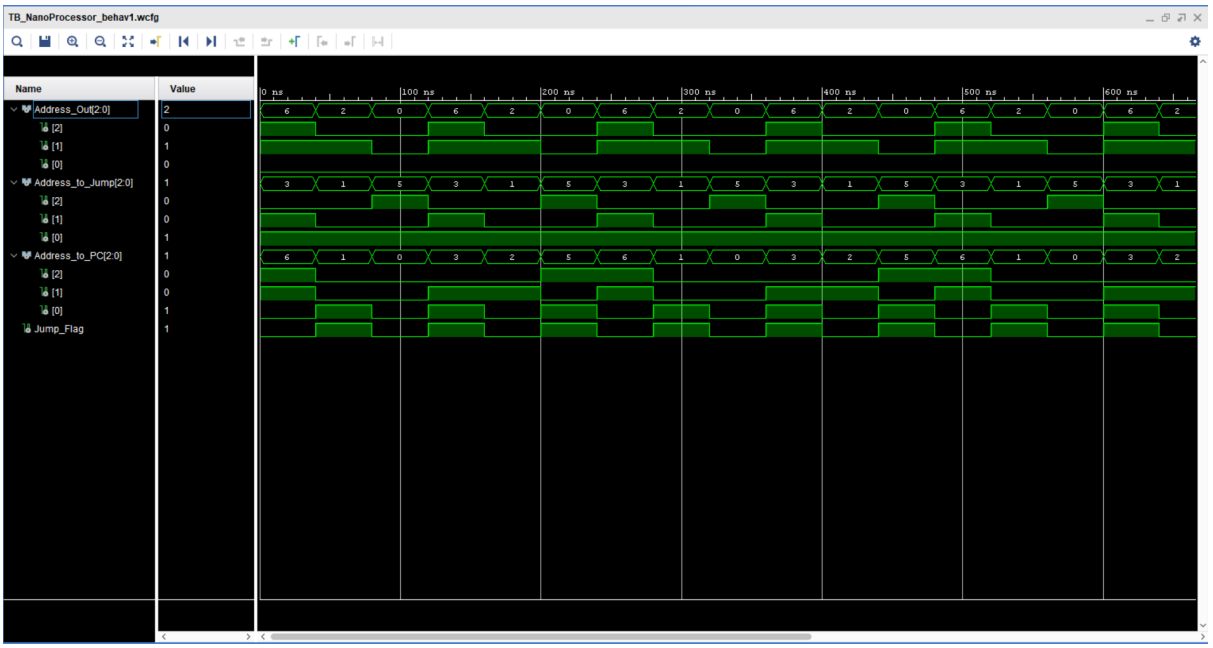
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- 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 Multiplexer_2_3 is
    Port ( Address_Out : in STD_LOGIC_VECTOR (2 downto 0);
           Address_to_Jump : in STD_LOGIC_VECTOR (2 downto 0);
           Jump_Flag : in STD_LOGIC;
           Address_to_PC : out STD_LOGIC_VECTOR (2 downto 0));
end Multiplexer_2_3;

architecture Behavioral of Multiplexer_2_3 is

begin
--    process(Jump_Flag,Address_Out,Address_to_Jump)
--        begin
--            if (Jump_Flag = '0') then
--                Address_to_PC <= Address_Out;
--            else
--                Address_to_PC <= Address_to_Jump;
--            end if;
--    end process;

--Simplified code
   Address_to_PC <= Address_Out when Jump_Flag = '0' else
Address_to_Jump;
end Behavioral;
```

TB_Multiplexer_2_3.vhd

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

entity TB_Multiplexer_2_3 is
--  Port ( );
end TB_Multiplexer_2_3;

architecture Behavioral of TB_Multiplexer_2_3 is
component Multiplexer_2_3 is
    Port ( Address_Out : in STD_LOGIC_VECTOR (2 downto 0);
           Address_to_Jump : in STD_LOGIC_VECTOR (2 downto 0);
           Jump_Flag : in STD_LOGIC;
           Address_to_PC : out STD_LOGIC_VECTOR (2 downto 0));
end component;
signal Address_Out,Address_to_Jump,Address_to_PC : STD_LOGIC_VECTOR (2
downto 0);
signal Jump_Flag :  STD_LOGIC;
begin
    UUT : Multiplexer_2_3
    PORT MAP( Address_Out => Address_Out,
              Address_to_Jump => Address_to_Jump,
              Jump_Flag => Jump_Flag,
              Address_to_PC => Address_to_PC);
      process
      begin
      Jump_Flag <= '0';
      wait for 40ns;
      Jump_Flag <= '1';
      wait for 40ns;
      end process;

      process
      begin
         -- Group Members Index numbers' binary form
         -- 210025T      110 011 010 001 101 001
         -- 210493A      110 011 011 000 111 101
         -- unique 3 bit numbers from index numbers=>110, 011, 010,
001, 000, 101, 111
         Address_Out<="110";
         Address_to_Jump<="011";
         wait for 40ns;
```

```
        Address_Out<="010";
        Address_to_Jump<="001";
        wait for 40ns;

        Address_Out<="000";
        Address_to_Jump<="101";
        wait for 40ns;
      end process;

end Behavioral;
```

Timing diagram

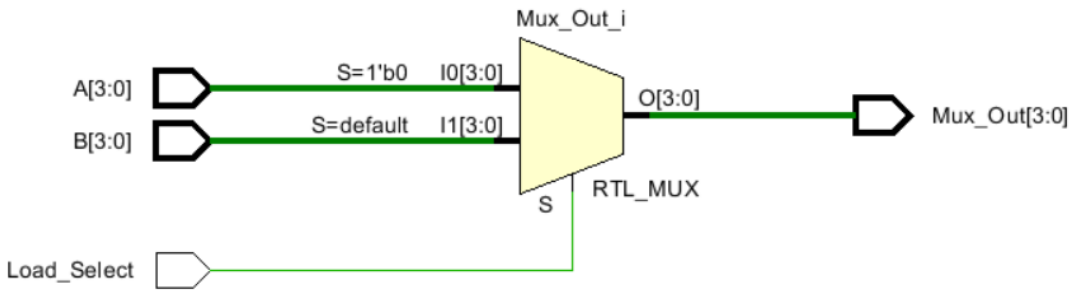


Note: Timing diagram can be verified by the following way:

When the jump flag is low(0), Address_Out has to be equal to the Address_to_PC. When the jump flag is high(1), Address_Out has to be equal to the Address_to_Jump.

5. 2-way 4-bit multiplexer

Schematic



Multiplexer_2_4.vhd

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

entity Multiplexer_2_4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Load_Select : in STD_LOGIC;
           Mux_Out : out STD_LOGIC_VECTOR (3 downto 0));
end Multiplexer_2_4;

architecture Behavioral of Multiplexer_2_4 is

begin
--process(Load_Select,A,B)
--    begin
--        if (Load_Select = '1') then
--            Mux_Out <= B;
--        else
--            Mux_Out <= A;
--        end if;
--    end process;

    --Simplified code
    Mux_Out <= A when Load_Select = '0' else B;
```

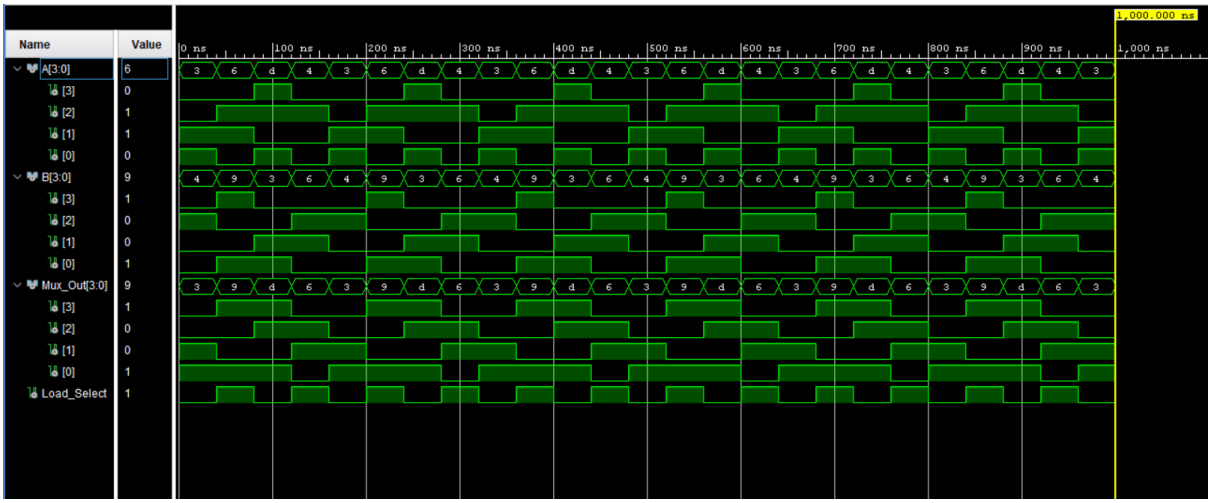
```
end Behavioral;
```

TB_Multiplexer_2_4.vhd

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

entity TB_Multiplexer_2_4 is
--  Port ( );
end TB_Multiplexer_2_4;

architecture Behavioral of TB_Multiplexer_2_4 is
component Multiplexer_2_4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Load_Select : in STD_LOGIC;
           Mux_Out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal A,B,Mux_Out : STD_LOGIC_VECTOR (3 downto 0);
signal Load_Select :  STD_LOGIC;

begin
    UUT : Multiplexer_2_4
    PORT MAP( A => A,
              B => B,
              Load_Select => Load_Select,
              Mux_Out => Mux_Out);
      process
      begin
      Load_Select <= '0';
      wait for 40ns;
      Load_Select <= '1';
      wait for 40ns;
      end process;

      process
      begin
         -- Group Members Index numbers' binary form
         -- 210025T      11 0011 0100 0110 1001
         -- 210493A      11 0011 0110 0011 1101
         -- unique 4bit numbers from index numbers=> 0011, 0100, 0110,
1001, 1101
```

```
        A<="0011";
        B<="0100";
        wait for 40ns;

        A<="0110";
        B<="1001";
        wait for 40ns;

        A<="1101";
        B<="0011";
        wait for 40ns;

        A<="0100";
        B<="0110";
        wait for 40ns;

      end process;


end Behavioral;
```

Timing diagram

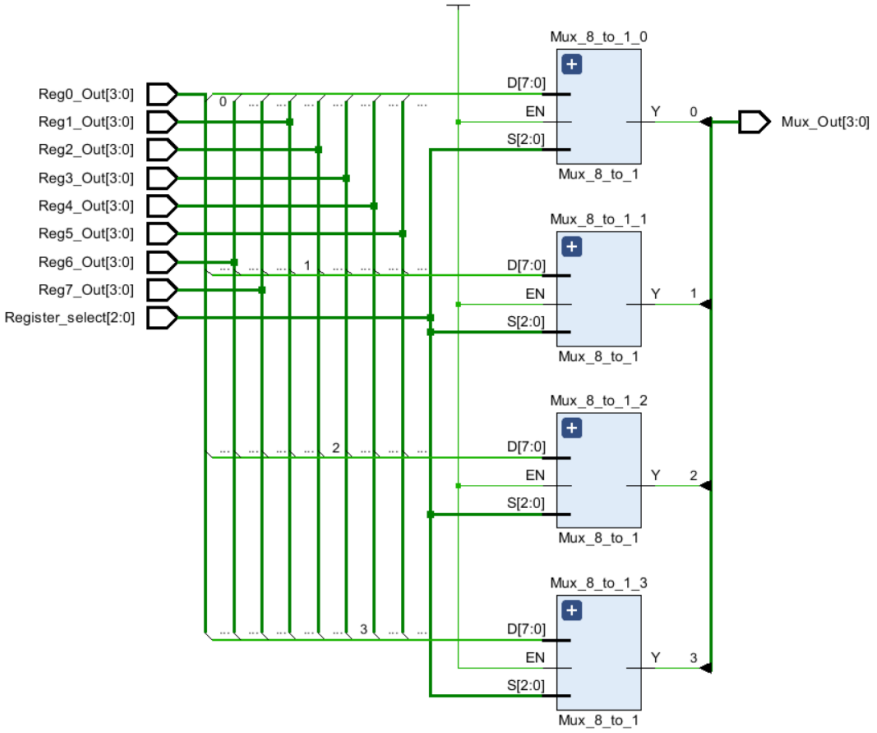


Note: Timing diagram can be verified by the following way:

When Load_Select is low(0), Mux_Out is equal to A and when Load_Select is high(1), Mux_Out is equal to B.

6. 8-way 4-bit multiplexer

Schematic



Multiplexer_8_4.vhd

```
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 Multiplexer_8_4 is
    Port ( Reg0_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg1_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg2_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg3_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg4_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg5_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg6_Out : in STD_LOGIC_VECTOR (3 downto 0);
```

```
           Reg7_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Register_select : in STD_LOGIC_VECTOR (2 downto 0);
           Mux_Out : out STD_LOGIC_VECTOR (3 downto 0));
end Multiplexer_8_4;

architecture Behavioral of Multiplexer_8_4 is

    component Mux_8_to_1
    Port ( S : in std_logic_vector(2 downto 0);
           D : in std_logic_vector(7 downto 0);
           Y : out std_logic
          );
end component;

begin
Mux_8_to_1_0: Mux_8_to_1
    port map(
        D(0) => Reg0_Out(0),
        D(1) => Reg1_Out(0),
        D(2) => Reg2_Out(0),
        D(3) => Reg3_Out(0),
        D(4) => Reg4_Out(0),
        D(5) => Reg5_Out(0),
        D(6) => Reg6_Out(0),
        D(7) => Reg7_Out(0),
        S => Register_select,
        Y => Mux_Out(0)
    );

Mux_8_to_1_1: Mux_8_to_1
    port map(
        D(0) => Reg0_Out(1),
        D(1) => Reg1_Out(1),
        D(2) => Reg2_Out(1),
        D(3) => Reg3_Out(1),
        D(4) => Reg4_Out(1),
        D(5) => Reg5_Out(1),
        D(6) => Reg6_Out(1),
        D(7) => Reg7_Out(1),
        S => Register_select,
        Y => Mux_Out(1)
    );
```

```
Mux_8_to_1_2: Mux_8_to_1
    port map(
        D(0) => Reg0_Out(2),
        D(1) => Reg1_Out(2),
        D(2) => Reg2_Out(2),
        D(3) => Reg3_Out(2),
        D(4) => Reg4_Out(2),
        D(5) => Reg5_Out(2),
        D(6) => Reg6_Out(2),
        D(7) => Reg7_Out(2),
        S => Register_select,
        Y => Mux_Out(2)
    );

Mux_8_to_1_3: Mux_8_to_1
    port map(
        D(0) => Reg0_Out(3),
        D(1) => Reg1_Out(3),
        D(2) => Reg2_Out(3),
        D(3) => Reg3_Out(3),
        D(4) => Reg4_Out(3),
        D(5) => Reg5_Out(3),
        D(6) => Reg6_Out(3),
        D(7) => Reg7_Out(3),
        S => Register_select,
        Y => Mux_Out(3)
    );
end Behavioral;
```

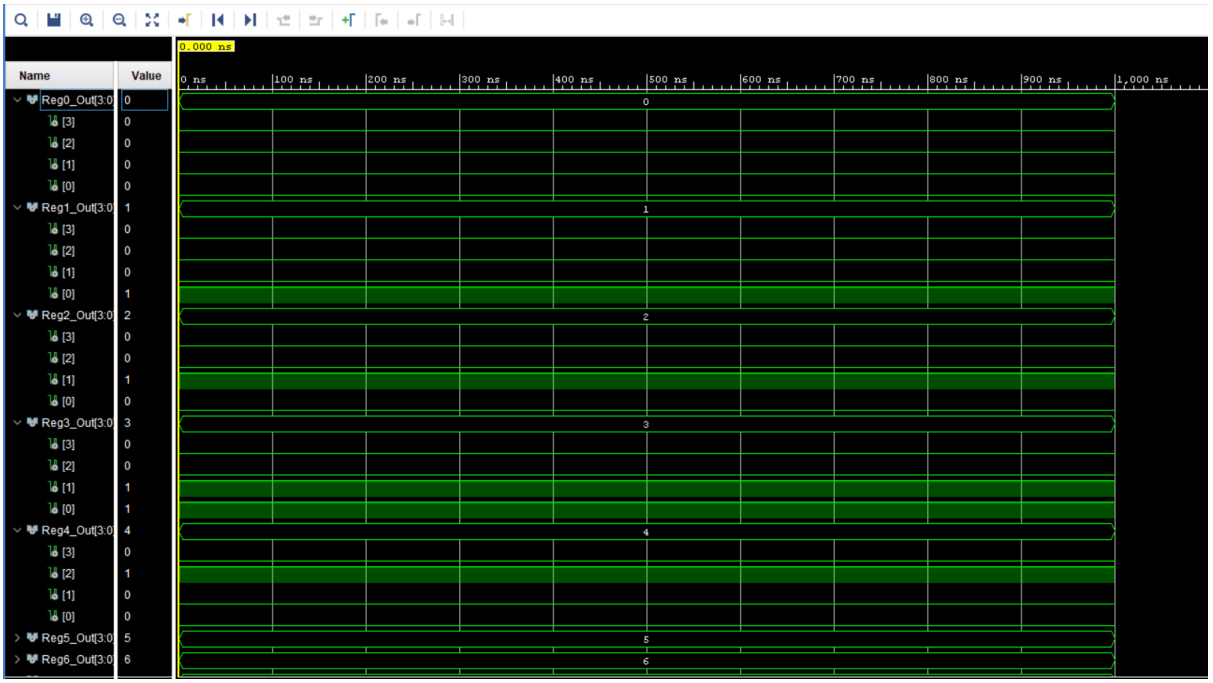
TB_Multiplexer_8_4.vhd

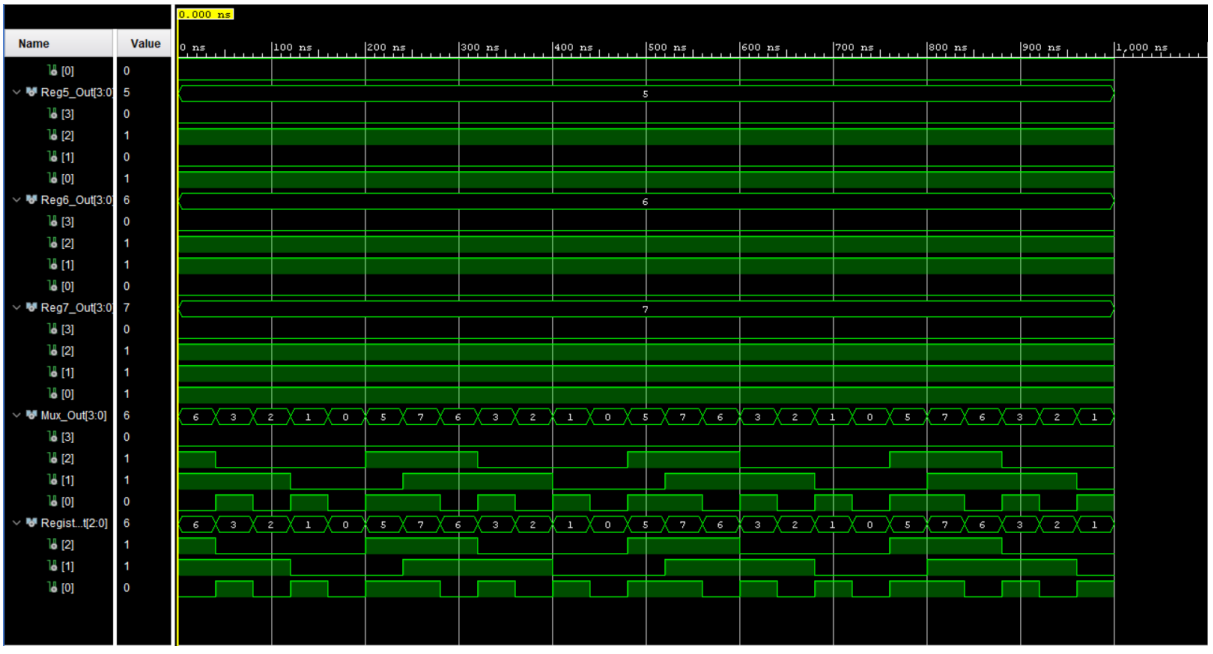
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- 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 TB_Multiplexer_8_4 is
--  Port ( );
end TB_Multiplexer_8_4;

architecture Behavioral of TB_Multiplexer_8_4 is
component Multiplexer_8_4 is
    Port ( Reg0_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg1_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg2_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg3_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg4_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg5_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg6_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg7_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Register_select : in STD_LOGIC_VECTOR (2 downto 0);
           Mux_Out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

SIGNAL
Reg0_Out,Reg1_Out,Reg2_Out,Reg3_Out,Reg4_Out,Reg5_Out,Reg6_Out,Reg7_Out
,Mux_Out :STD_LOGIC_VECTOR (3 downto 0);
signal Register_select : STD_LOGIC_VECTOR (2 downto 0);

begin
    UUT: Multiplexer_8_4
    PORT MAP( Reg0_Out =>Reg0_Out,
              Reg1_Out =>Reg1_Out,
              Reg2_Out =>Reg2_Out,
              Reg3_Out =>Reg3_Out,
              Reg4_Out =>Reg4_Out,
              Reg5_Out =>Reg5_Out,
              Reg6_Out =>Reg6_Out,
              Reg7_Out =>Reg7_Out,
              Register_select =>Register_select,
              Mux_Out => Mux_Out);


        process
        begin
        Reg0_Out<="0000";
        Reg1_Out<="0001";
```

```
        Reg2_Out<="0010";
        Reg3_Out<="0011";
        Reg4_Out<="0100";
        Reg5_Out<="0101";
        Reg6_Out<="0110";
        Reg7_Out<="0111";
        wait;
        end process;

        process
        begin
            -- Group Members Index numbers' binary form
            -- 210025T      110 011 010 001 101 001
            -- 210493A      110 011 011 000 111 101
            -- unique 3 bit numbers from index numbers=>110, 011, 010,
001, 000, 101, 111
            Register_select <= "110";
            wait for 40ns;

            Register_select <= "011";
            wait for 40ns;

            Register_select <= "010";
            wait for 40ns;

            Register_select <= "001";
            wait for 40ns;

            Register_select <= "000";
            wait for 40ns;

            Register_select <= "101";
            wait for 40ns;

            Register_select <= "111";
            wait for 40ns;
        end process;

end Behavioral;
```

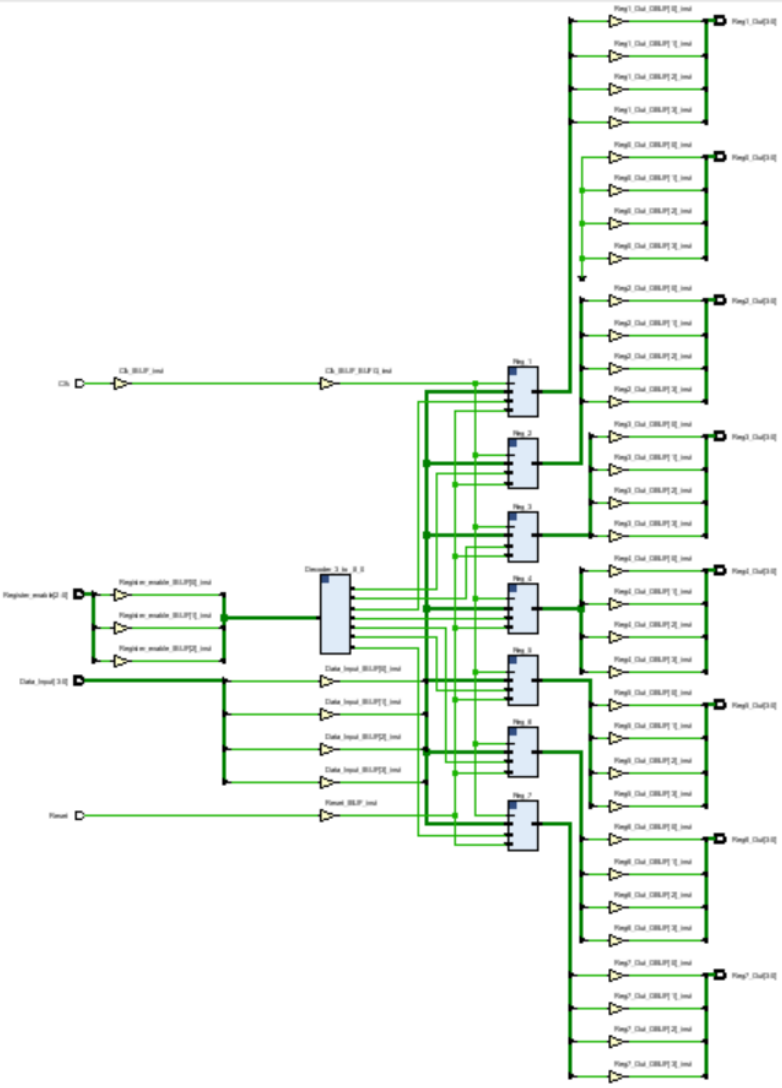
Timing diagram





7. Register Bank

Schematic



RegisterBank.vhd

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

entity RegisterBank is
    Port ( Register_enable : in STD_LOGIC_VECTOR (2 downto 0); --To
select the register
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Data_Input : in STD_LOGIC_VECTOR (3 downto 0);
           Reg0_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg1_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg2_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg3_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg4_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg5_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg6_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg7_Out : out STD_LOGIC_VECTOR (3 downto 0)
           );

end RegisterBank;

architecture Behavioral of RegisterBank is

component Register_4
    port(
       D : in STD_LOGIC_VECTOR (3 downto 0);
       En : in STD_LOGIC;
       Reset : in STD_LOGIC;
       Clk : in STD_LOGIC;
       Q : out STD_LOGIC_VECTOR (3 downto 0)
    );
end component;

component Decoder_3_8
    port(
        I : in STD_LOGIC_VECTOR (2 downto 0);
        En : in STD_LOGIC;
        Y : OUT STD_LOGIC_VECTOR (7 downto 0)
    );
end component;
```

```
SIGNAL Y : STD_LOGIC_VECTOR (7 downto 0);

begin
Decoder_3_to_8_0: Decoder_3_8
    port map(
        I => Register_enable,
        En => '1',
        Y => Y
    );


Reg_0: Register_4
    port map(
        D=>"0000",
        En=>Y(0),
        Reset => Reset,
        Clk=>Clk,
        Q=>Reg0_Out
    );

Reg_1: Register_4
    port map(
        D=>Data_Input,
        En=>Y(1),
        Reset => Reset,
        Clk=>Clk,
        Q=>Reg1_Out
    );

Reg_2: Register_4
    port map(
        D=>Data_Input,
        En=>Y(2),
        Reset => Reset,
        Clk=>Clk,
        Q=>Reg2_Out
        );

Reg_3: Register_4
    port map(
        D=>Data_Input,
        En=>Y(3),
        Reset => Reset,
```

```
        Clk=>Clk,
        Q=>Reg3_Out
    );

Reg_4: Register_4
    port map(
        D=>Data_Input,
        En=>Y(4),
        Reset => Reset,
        Clk=>Clk,
        Q=>Reg4_Out
    );

Reg_5: Register_4
    port map(
        D=>Data_Input,
        En=>Y(5),
        Reset => Reset,
        Clk=>Clk,
        Q=>Reg5_Out
        );

Reg_6: Register_4
    port map(
        D=>Data_Input,
        En=>Y(6),
        Reset => Reset,
        Clk=>Clk,
        Q=>Reg6_Out
    );

Reg_7: Register_4
    port map(
        D=>Data_Input,
        En=>Y(7),
        Reset => Reset,
        Clk=>Clk,
        Q=>Reg7_Out
    );


end Behavioral;
```

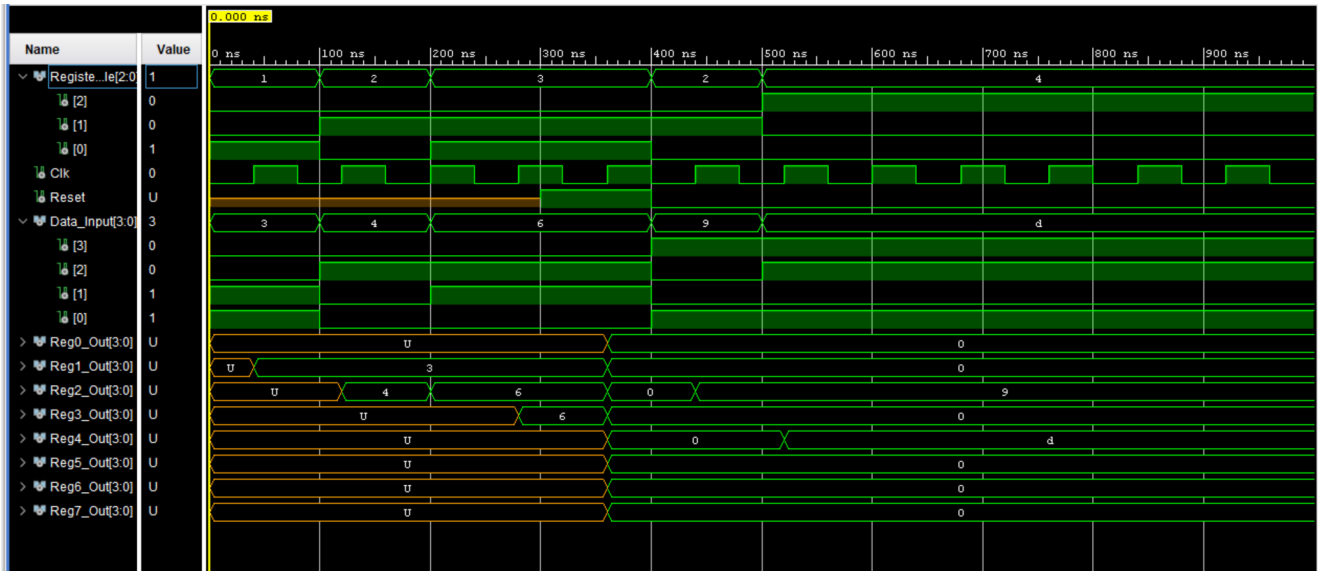
TB_RegisterBank.vhd

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

entity TB_RegisterBank is
--  Port ( );
end TB_RegisterBank;

architecture Behavioral of TB_RegisterBank is

component RegisterBank
    Port ( Register_enable : in STD_LOGIC_VECTOR (2 downto 0); --To
select the register
       Clk : in STD_LOGIC;
       Reset : in STD_LOGIC;
       Data_Input : in STD_LOGIC_VECTOR (3 downto 0);
       Reg0_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg1_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg2_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg3_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg4_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg5_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg6_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg7_Out : out STD_LOGIC_VECTOR (3 downto 0)
       );
end component;

Signal Register_enable : STD_LOGIC_VECTOR (2 downto 0);
Signal Clk :  STD_LOGIC:='0';
Signal Reset :  STD_LOGIC;
Signal Data_Input : STD_LOGIC_VECTOR (3 downto 0);
signal
Reg0_Out,Reg1_Out,Reg2_Out,Reg3_Out,Reg4_Out,Reg5_Out,Reg6_Out,Reg7_Out
: STD_LOGIC_VECTOR(3 downto 0);

begin
UUT:RegisterBank
    port map(
    Clk => Clk,
    Register_enable => Register_enable,
    Reset => Reset,
```

```
    Data_Input => Data_Input,
    Reg0_Out=> Reg0_Out,
    Reg1_Out=> Reg1_Out,
    Reg2_Out=> Reg2_Out,
    Reg3_Out=> Reg3_Out,
    Reg4_Out=> Reg4_Out,
    Reg5_Out=> Reg5_Out,
    Reg6_Out=> Reg6_Out,
    Reg7_Out=> Reg7_Out
    );

process
  begin
      Wait for 40ns;
      Clk<=NOT(Clk);
end process;

process
    begin
-- Group Members Index numbers' binary form
-- 210025T      11 0011 0100 0110 1001
-- 210493A      11 0011 0110 0011 1101
-- unique 4bit numbers from index numbers=> 0011, 0100, 0110, 1001,
1101

        Register_enable<="001";
        Data_Input<="0011";
        wait for 100ns;

        Register_enable<="010";
        Data_Input<="0100";
        wait for 100ns;

        Register_enable<="011";
        Data_Input<="0110";
        wait for 100ns;

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

        Reset <= '0';
        Register_enable<="010";
        Data_Input<="1001";
```

```
        wait for 100ns;

        Register_enable<="100";
        Data_Input<="1101";
        wait ;
end process;

end Behavioral;
```

Timing diagram

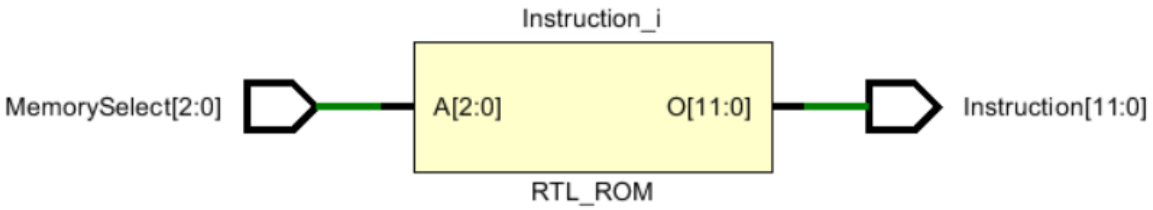


Note: Timing diagram can be verified by the following way:

If Reset equals to 1, data in all registers have to become zero. If Data_Input is 3 and Register-enable is 1, Register1_Out will be 3. If Data_Input is 4 and Register-enable is 2, Register2_Out will be 4.

8. Program ROM

Schematic



Program_ROM.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Program_ROM is
    Port ( MemorySelect : in STD_LOGIC_VECTOR (2 downto 0);
           Instruction : 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 P_ROM : rom_type := (
          --Add 3 by looping
          "101000000011", -- MOVI R4,3
```

```
          "101100000001", -- MOVI R6,1
          "011100000000", -- NEG R6
          "001111000000", -- ADD R7,R4
          "001001100000", -- ADD R4,R6
          "111000000111", -- JZR R4,7
          "110000000011", -- JZR R0,3
          "111000000111" -- JZR R4,7

        --Program to display 10 to 0
        --"101110001010", --0  Move R7 10
        --"100100000001", --1  Move R2 01
        --"010100000000", --2  Neg R2
        --"001110100000", --3  R7<- R7+R2
        --"111110000111", --4  JMP R7=0 PR7
        --"110000000011", --5  JMP R0=0 PR3
        --"110010000111", --6
        --"110010000110"  --7

        -- Add 1,2,3 to three registers R7,R6,R5 respectively.
        -- Add R6 to R7 and store in R7
        -- Add R5 to R7 and store in R7
        -- Adding 1,2,3 by storing each value in each registers.
        --"101110000001", -- 0-- MOVI R7,1
        --"101100000010", -- 1-- MOVI R6,2
        --"101010000011", -- 2-- MOVI R5,3
        --"001111100000", -- 3-- ADD R7,R6
        --"001111010000", -- 4-- ADD R7,R5
        --"110000000101", -- 5-- JZR R0,5
        --"110000000101", -- 6-- JZR R0,5
        --"110000000111"  -- 7-- JZR R0,7
    );
begin
    Instruction <= P_ROM(to_integer(unsigned(MemorySelect)));

end Behavioral;
```

TB_Program_Rom.vhd

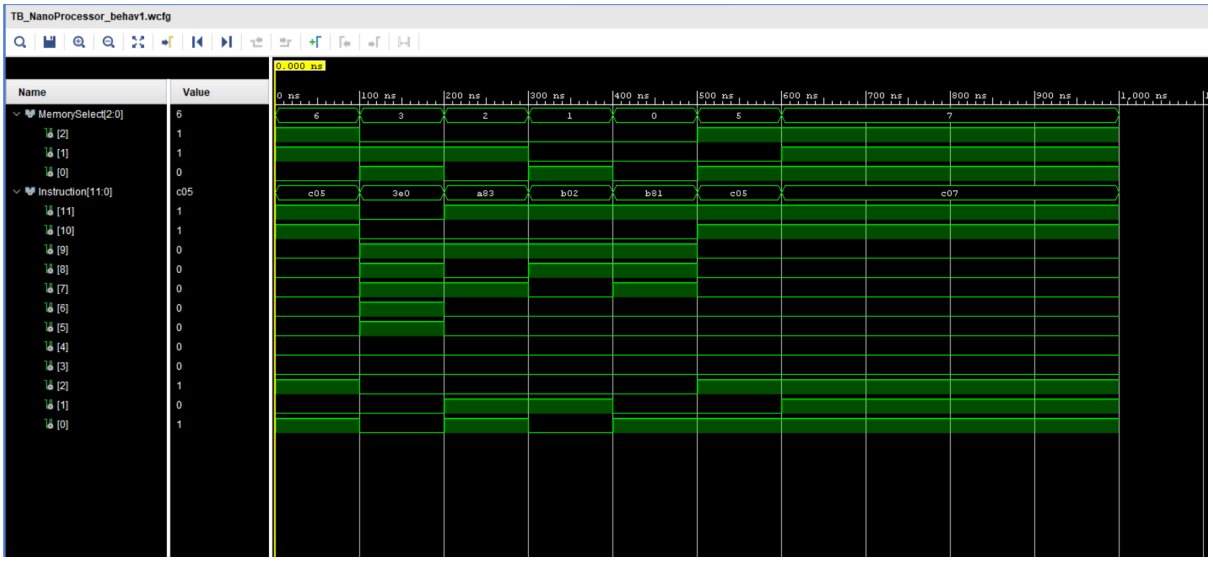
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- 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 TB_Program_Rom is
    --  Port ( );
end TB_Program_Rom;

architecture Behavioral of TB_Program_Rom is

component Program_ROM
     Port ( MemorySelect : in STD_LOGIC_VECTOR (2 downto 0);
     Instruction : out STD_LOGIC_VECTOR (11 downto 0));
end component;

signal MemorySelect:STD_LOGIC_VECTOR (2 downto 0);
signal Instruction: STD_LOGIC_VECTOR (11 downto 0);

begin
    UUT : Program_ROM
        port map(
            MemorySelect => MemorySelect,
            Instruction => Instruction
        );

process
    begin
        -- Group Members Index numbers' binary form
        -- 210025T      110 011 010 001 101 001
        -- 210493A      110 011 011 000 111 101
        -- unique 3bit numbers from index numbers=> 110, 011, 010, 001,
000, 101, 111
        MemorySelect <= "110";
        wait for 100ns;

        MemorySelect <= "011";
        wait for 100ns;

        MemorySelect <= "010";
```

```
        wait for 100ns;

        MemorySelect <= "001";
        wait for 100ns;

        MemorySelect <= "000";
        wait for 100ns;

        MemorySelect <= "101";
        wait for 100ns;

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

end Behavioral;
```

Timing diagram

1)If 1,2,3 are stored in three registers and then added



Note: Timing diagram can be verified by the following way:

Instruction ‘c05’ means ‘110000000101’.

c => 1100
0 => 0000
5 => 0101

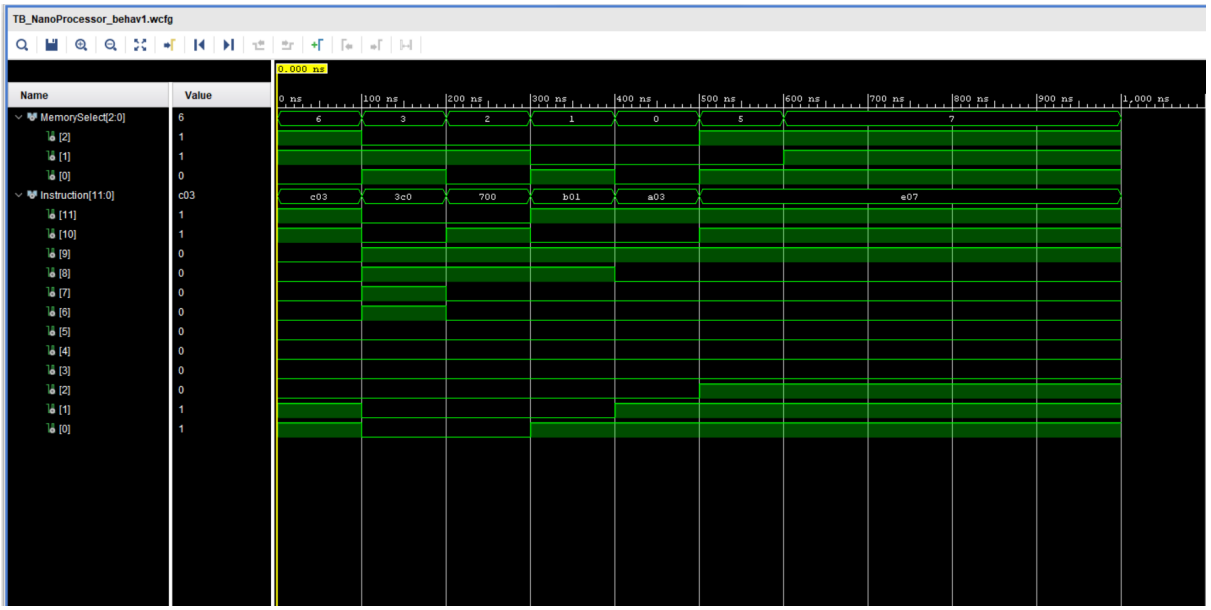
The following instruction set is given in the lab sheet.

Table 1 – Instruction Set.

Instruction	Description	Format (12-bit instruction)
MOVI R, *d*	Move immediate value *d* to register R, i.e., *R* ← *d* R ∈ [0, 7], *d* ∈ [0, 15]	1 0 R R R 0 0 0 d d d d
ADD Ra, Rb	Add values in registers Ra and Rb and store the result in Ra, i.e., Ra ← Ra + Rb Ra, Rb ∈ [0, 7]	0 0 Ra Ra Ra Rb Rb Rb 0 0 0 0
NEG R	2's complement of registers R, i.e., R ← – R R ∈ [0, 7]	0 1 R R R 0 0 0 0 0 0 0
JZR R, d	Jump if value in register R is 0, i.e., If R == 0 PC ← d; Else PC ← PC + 1; R ∈ [0, 7], *d* ∈ [0, 7]	1 1 R R R 0 0 0 0 d d d

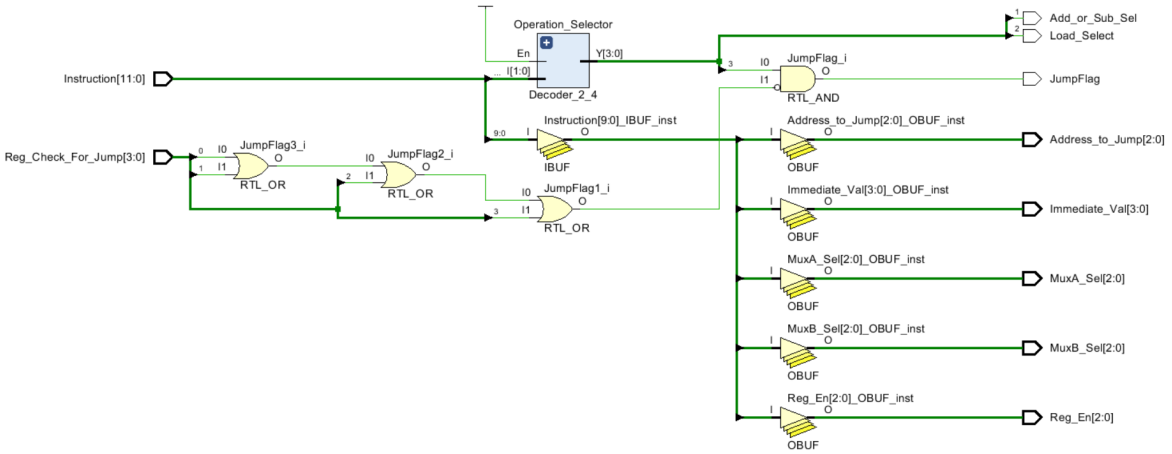
So '110000000101' means jump to the 5th line if value in Register0 is zero.

2) If 1,2,3 are added by looping



9. Instruction Decoder

Schematic



Instruction_Decoder.vhd

```
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 Instruction_Decoder is
     Port (
          Instruction : in STD_LOGIC_VECTOR (11 downto 0);
          Reg_Check_For_Jump : in STD_LOGIC_VECTOR (3 downto 0);
          Reg_En : out STD_LOGIC_VECTOR(2 downto 0);
          Load_Select : out STD_LOGIC;
          Immediate_Val : out STD_LOGIC_VECTOR(3 downto 0);
          MuxA_Sel : out STD_LOGIC_VECTOR (2 downto 0);
          MuxB_Sel : out STD_LOGIC_VECTOR (2 downto 0);
          Add_or_Sub_Sel : out STD_LOGIC;
```

```
        JumpFlag : out STD_LOGIC;
        Address_to_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is

component Decoder_2_4
    PORT (
    I : in STD_LOGIC_VECTOR (1 downto 0);
    EN : in STD_LOGIC;
    Y : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    signal jump_Instruction, Add_Instruction:std_logic;
begin

    Operation_Selector : Decoder_2_4
port map(
    I  => Instruction(11 downto 10),
    EN => '1',
    Y(0) => Add_Instruction, --add --we will not use this
    Y(1) => Add_or_Sub_Sel, -- neg
    Y(2) => Load_Select,     --movi
    Y(3) => jump_Instruction);   --jzr

Immediate_Val <= Instruction(3 downto 0);
Reg_En <= Instruction(9 downto 7);
Address_to_Jump <= Instruction(2 downto 0);
MuxB_Sel <= Instruction(9 downto 7);
MuxA_Sel <= Instruction(6 downto 4);

JumpFlag <= jump_Instruction and (not(Reg_Check_For_Jump(0) or
Reg_Check_For_Jump(1) or Reg_Check_For_Jump(2) or
Reg_Check_For_Jump(3)));

end Behavioral;
```

TB_Instruction_Decoder.vhd

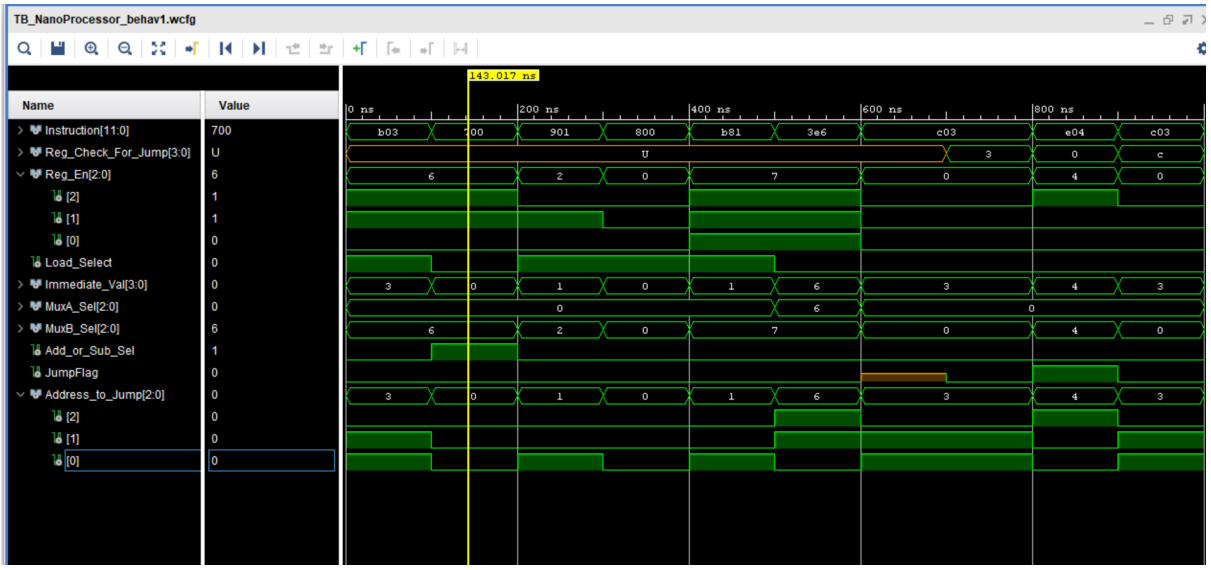
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
-- 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 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_Check_For_Jump : in STD_LOGIC_VECTOR (3 downto 0);
    Reg_En : out STD_LOGIC_VECTOR(2 downto 0);
    Load_Select : out STD_LOGIC;
    Immediate_Val : out STD_LOGIC_VECTOR(3 downto 0);
    MuxA_Sel : out STD_LOGIC_VECTOR (2 downto 0);
    MuxB_Sel : out STD_LOGIC_VECTOR (2 downto 0);
    Add_or_Sub_Sel : out STD_LOGIC;
    JumpFlag : out STD_LOGIC;
    Address_to_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal Instruction : STD_LOGIC_VECTOR (11 downto 0);
signal Reg_Check_For_Jump : STD_LOGIC_VECTOR (3 downto 0);
signal Reg_En : STD_LOGIC_VECTOR (2 downto 0);
signal Load_Select : STD_LOGIC;
signal Immediate_Val : STD_LOGIC_VECTOR (3 downto 0);
signal MuxA_Sel : STD_LOGIC_VECTOR (2 downto 0);
signal MuxB_Sel : STD_LOGIC_VECTOR (2 downto 0);
signal Add_or_Sub_Sel : STD_LOGIC;
signal JumpFlag : STD_LOGIC;
signal Address_to_Jump : STD_LOGIC_VECTOR (2 downto 0);


begin
UUT : Instruction_Decoder
    port map(
```

```
        Instruction =>Instruction,
        Reg_Check_For_Jump =>Reg_Check_For_Jump,
        Reg_En =>Reg_En,
        Load_Select => Load_Select,
        Immediate_Val =>Immediate_Val,
        MuxA_Sel =>MuxA_Sel,
        MuxB_Sel =>MuxB_Sel,
        Add_or_Sub_Sel =>Add_or_Sub_Sel,
        JumpFlag =>JumpFlag,
        Address_to_Jump => Address_to_Jump
        );
process
begin
         -- Group Members Index numbers' binary form
         -- 210025T      110 011 010 001 101 001
         -- 210493A      110 011 011 000 111 101
         -- unique 3bit numbers from index numbers=>
110,011,010,001,000,111,101
        Instruction <= "101100000011"; -- MOVI R6, 3
        wait for 100ns;

        Instruction <= "011100000000"; -- NEG R6
        wait for 100 ns;

        Instruction <= "100100000001"; -- MOVI R2, 1
        wait for 100ns;
        Instruction <= "100000000000"; -- MOVI R0, 0
        wait for 100ns;
        Instruction <= "101110000001"; -- MOVI R7, 5
        wait for 100ns;

        Instruction <= "001111100110"; -- ADD R7, R6
        wait for 100ns;

        Instruction <= "110000000011"; -- JZR R0,3
        wait for 100 ns;
        Reg_Check_For_Jump <= "0011";
        wait for 100ns;
        Instruction <= "111000000100"; -- JZR R4,4
        Reg_Check_For_Jump <= "0000";
        wait for 100ns;
        Instruction <= "110000000011"; -- JZR R0,3
        Reg_Check_For_Jump <= "1100";
```

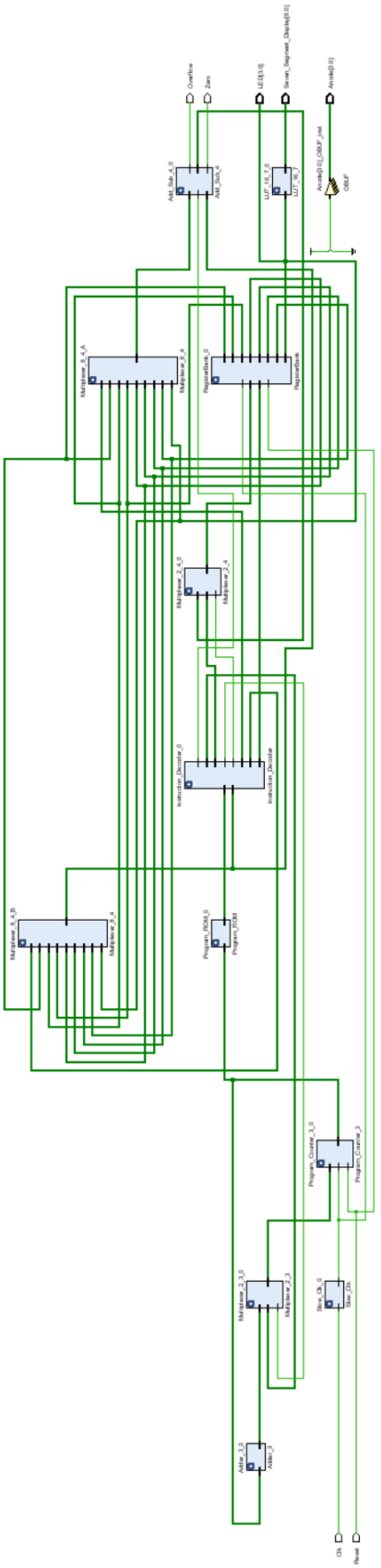
```
        wait;
        end process;

        end Behavioral;
```

Timing diagram



Nano Processor

Schematic



NanoProcessor.vhd

```
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 NanoProcessor is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Seven_Segment_Display : out STD_LOGIC_VECTOR (6 downto 0);
           LED : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC;
           Anode : out STD_LOGIC_VECTOR(3 downto 0));
end NanoProcessor;

architecture Behavioral of NanoProcessor is

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

-----------------------------------Program
Counter----------------------------------------
component Program_Counter_3
    Port ( Address : in STD_LOGIC_VECTOR (2 downto 0);
       Res : in STD_LOGIC;
       Clk : in STD_LOGIC;
       MemorySelect : out STD_LOGIC_VECTOR (2 downto 0));
end component;
```

```
---------------------------------3-bit
Adder-------------------------------------------------
component Adder_3
    Port ( Address_In : in STD_LOGIC_VECTOR (2 downto 0);
       Address_Out : out STD_LOGIC_VECTOR (2 downto 0));
end component;

---------------------------------2-way 3-bit
MUX----------------------------------------------
component Multiplexer_2_3
    Port ( Address_Out : in STD_LOGIC_VECTOR (2 downto 0);
       Address_to_Jump : in STD_LOGIC_VECTOR (2 downto 0);
       Jump_Flag : in STD_LOGIC;
       Address_to_PC : out STD_LOGIC_VECTOR (2 downto 0));
end component;

---------------------------------Program_ROM---------------------------
-----------------
component Program_ROM
    Port ( MemorySelect : in STD_LOGIC_VECTOR (2 downto 0);
       Instruction : out STD_LOGIC_VECTOR (11 downto 0));
end component;

---------------------------------Instruction
Decoder-----------------------------------------------
component Instruction_Decoder
     Port (
        Instruction : in STD_LOGIC_VECTOR (11 downto 0);
        Reg_Check_For_Jump : in STD_LOGIC_VECTOR (3 downto 0);
        Reg_En : out STD_LOGIC_VECTOR(2 downto 0);
        Load_Select : out STD_LOGIC;
        Immediate_Val : out STD_LOGIC_VECTOR(3 downto 0);
        MuxA_Sel : out STD_LOGIC_VECTOR (2 downto 0);
        MuxB_Sel : out STD_LOGIC_VECTOR (2 downto 0);
        Add_or_Sub_Sel : out STD_LOGIC;
        JumpFlag : out STD_LOGIC;
        Address_to_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end component;

---------------------------------2 way 4 bit
MUX----------------------------------------------
component Multiplexer_2_4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
```

```
       B : in STD_LOGIC_VECTOR (3 downto 0);
       Load_Select : in STD_LOGIC;
       Mux_Out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

---------------------------------Register
Bank-------------------------------------------
component RegisterBank is
    Port ( Register_enable : in STD_LOGIC_VECTOR (2 downto 0); --To
select the register
       Clk : in STD_LOGIC;
       Reset : in STD_LOGIC;
       Data_Input : in STD_LOGIC_VECTOR (3 downto 0);
       Reg0_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg1_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg2_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg3_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg4_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg5_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg6_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg7_Out : out STD_LOGIC_VECTOR (3 downto 0)
       );
end component;

---------------------------------Multiplexer_8_4-----------------------
---------------------
component Multiplexer_8_4
    Port ( Reg0_Out : in STD_LOGIC_VECTOR (3 downto 0);
       Reg1_Out : in STD_LOGIC_VECTOR (3 downto 0);
       Reg2_Out : in STD_LOGIC_VECTOR (3 downto 0);
       Reg3_Out : in STD_LOGIC_VECTOR (3 downto 0);
       Reg4_Out : in STD_LOGIC_VECTOR (3 downto 0);
       Reg5_Out : in STD_LOGIC_VECTOR (3 downto 0);
       Reg6_Out : in STD_LOGIC_VECTOR (3 downto 0);
       Reg7_Out : in STD_LOGIC_VECTOR (3 downto 0);
       Register_select : in STD_LOGIC_VECTOR (2 downto 0);
       Mux_Out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

-------------------------------------Add_Sub_4-----------------------
-----------------------
component Add_Sub_4
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
```

```
       B : in STD_LOGIC_VECTOR (3 downto 0);
       Add_Sub_Select : in STD_LOGIC;
       Zero:out std_logic;
       Overflow :out STD_LOGIC;
       S : out STD_LOGIC_VECTOR (3 downto 0));
end component;

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

signal Slow_Clock, JumpFlag, Load_Select, Add_or_Sub_Sel: STD_LOGIC;
signal AddressToPC, MemorySelect, Adder_Out: STD_LOGIC_VECTOR(2 downto
0);
signal JumpAddress, Register_enable, MuxA_Sel, MuxB_Sel:
STD_LOGIC_VECTOR(2 downto 0);
signal PROM_Instruction : STD_LOGIC_VECTOR (11 downto 0);
signal MuxA_out,MuxB_out, Sum, Mux_Out,Immediate_Val:
STD_LOGIC_VECTOR(3 downto 0);
signal
Reg0_Out,Reg1_Out,Reg2_Out,Reg3_Out,Reg4_Out,Reg5_Out,Reg6_Out,Reg7_Out
: STD_LOGIC_VECTOR(3 downto 0);

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

Program_Counter_3_0 : Program_Counter_3
port map (
    Address => AddressToPC,
    Res => Reset,
    Clk => Slow_Clock,
    MemorySelect => MemorySelect
);

Adder_3_0: Adder_3
port map (
    Address_In => MemorySelect,
```

```
    Address_Out => Adder_Out

);

Multiplexer_2_3_0 : Multiplexer_2_3
port map (
    Address_Out => Adder_Out ,
    Address_to_Jump => JumpAddress,
    Jump_Flag => JumpFlag,
    Address_to_PC => AddressToPC
);

Program_ROM_0 :Program_ROM
port map(
    MemorySelect => MemorySelect,
    Instruction => PROM_Instruction
);

Instruction_Decoder_0 : Instruction_Decoder
port map(
    Instruction => PROM_Instruction,
    Reg_Check_For_Jump => MuxB_out,
    Reg_En => Register_enable,
    Load_Select  => Load_Select,
    Immediate_Val => Immediate_Val,
    MuxA_Sel => MuxA_Sel,
    MuxB_Sel => MuxB_Sel,
    Add_or_Sub_Sel => Add_or_Sub_Sel,
    JumpFlag => JumpFlag,
    Address_to_Jump => JumpAddress
    );

Multiplexer_2_4_0 : Multiplexer_2_4
port map(
    A => Sum,
    B => Immediate_Val,
    Load_Select => Load_Select,
    Mux_Out => Mux_Out
    );

RegisterBank_0 : RegisterBank
port map(
    Clk => Slow_Clock,
```

```
    Register_enable => Register_enable,
    Reset => Reset,
    Data_Input => Mux_Out,
    Reg0_Out=> Reg0_Out,
    Reg1_Out=> Reg1_Out,
    Reg2_Out=> Reg2_Out,
    Reg3_Out=> Reg3_Out,
    Reg4_Out=> Reg4_Out,
    Reg5_Out=> Reg5_Out,
    Reg6_Out=> Reg6_Out,
    Reg7_Out=> Reg7_Out
    );

Multiplexer_8_4_A :Multiplexer_8_4
port map (
        Reg0_Out=> Reg0_Out,
        Reg1_Out=> Reg1_Out,
        Reg2_Out=> Reg2_Out,
        Reg3_Out=> Reg3_Out,
        Reg4_Out=> Reg4_Out,
        Reg5_Out=> Reg5_Out,
        Reg6_Out=> Reg6_Out,
        Reg7_Out=> Reg7_Out,
        Register_select => MuxA_Sel,
        Mux_Out => MuxA_Out
        );

Multiplexer_8_4_B :Multiplexer_8_4
port map (
    Reg0_Out=> Reg0_Out,
    Reg1_Out=> Reg1_Out,
    Reg2_Out=> Reg2_Out,
    Reg3_Out=> Reg3_Out,
    Reg4_Out=> Reg4_Out,
    Reg5_Out=> Reg5_Out,
    Reg6_Out=> Reg6_Out,
    Reg7_Out=> Reg7_Out,
    Register_select => MuxB_Sel,
    Mux_Out => MuxB_Out
    );

Add_Sub_4_0 : Add_Sub_4
port map (
```

```
    A => MuxA_Out,
    B => MuxB_Out,
    Add_Sub_Select => Add_or_Sub_Sel,
    S => Sum,
    Overflow => Overflow,
    Zero => Zero
    );

LUT_16_7_0 : LUT_16_7
port map(
    address => Reg7_Out,
    data => Seven_Segment_Display
);

LED <=  Reg7_Out;
Anode <= "1110";

end Behavioral;
```

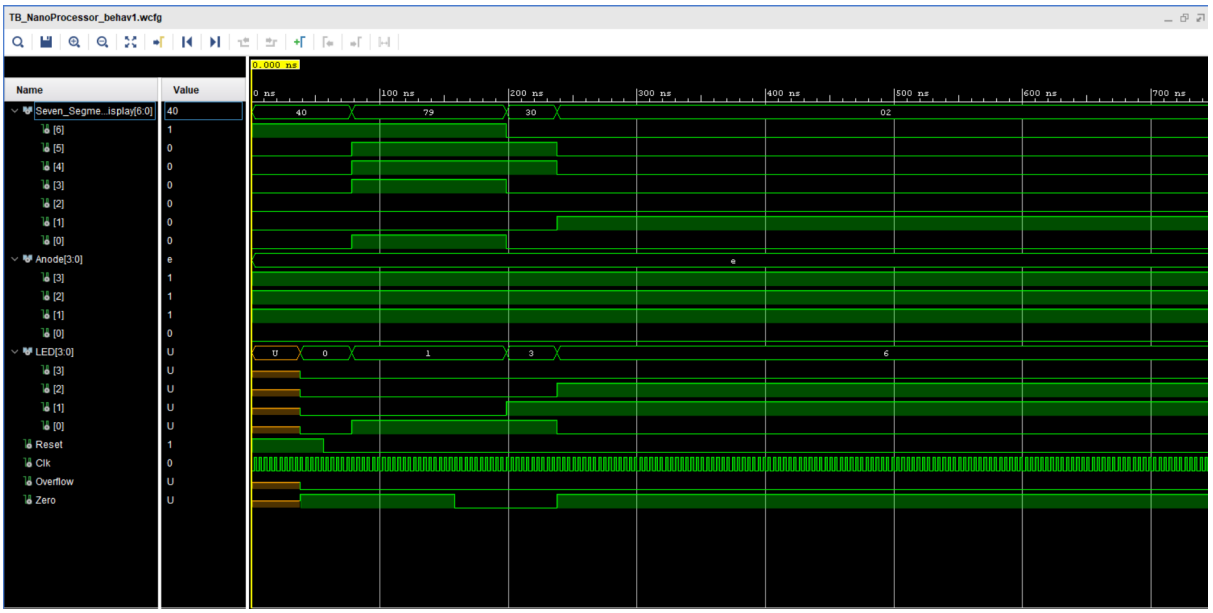
TB_NanoProcessor.vhd

```
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 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;
```

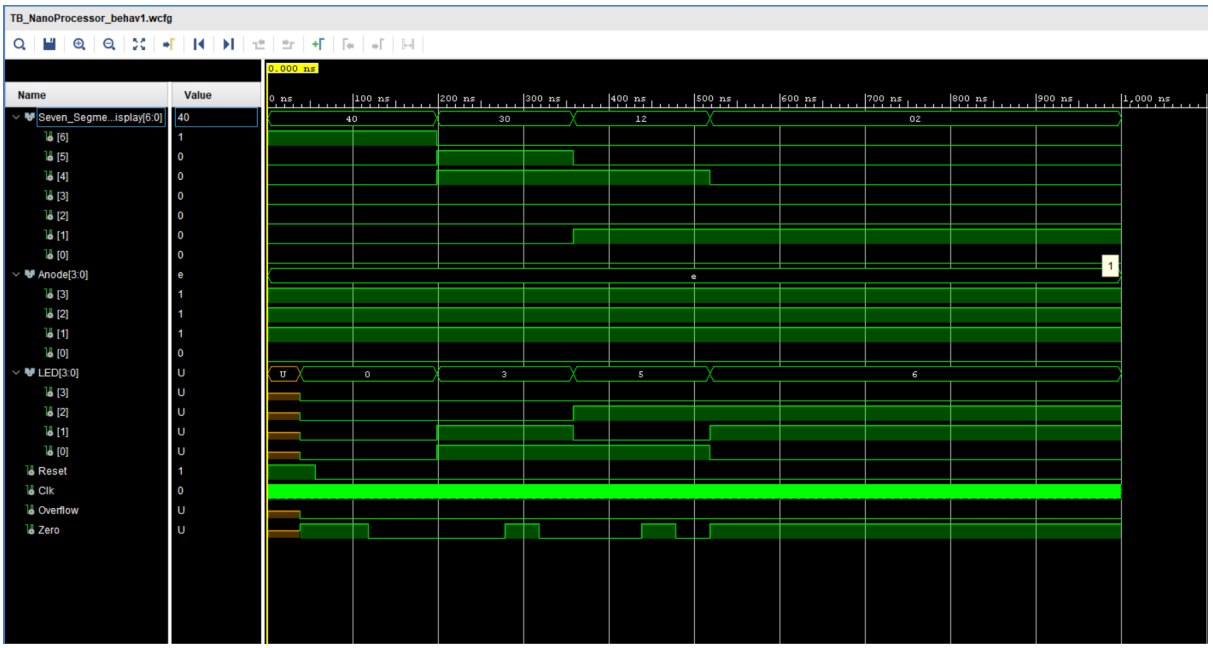
```
        Zero : out STD_LOGIC;
        LED : out STD_LOGIC_VECTOR (3 downto 0);
        Seven_Segment_Display: out STD_LOGIC_VECTOR (6 downto 0);
        Anode: out STD_LOGIC_VECTOR ( 3 downto 0) := "0001");

end component;
signal Overflow,Zero : std_logic;
signal Reset:std_logic := '1';
signal Clk:std_logic := '0';
signal LED,Anode: std_logic_vector(3 downto 0);
signal Seven_Segment_Display: std_logic_vector(6 downto 0);

begin

UUT: NanoProcessor
port map(
        Clk => Clk,
        Reset => Reset,
        Overflow => Overflow,
        Zero => Zero,
        LED =>LED,
        Seven_Segment_Display => Seven_Segment_Display,
        Anode => Anode
        );

process
begin
    wait for 2ns;
    Clk <= NOT(Clk);
end process;

process
begin
    Reset <= '1';
    wait for 56 ns;

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

end Behavioral;
```

Timing diagrams

1)If 1,2,3 are stored in three registers and then added



2) If 1,2,3 are added by looping



Slow_Clk.vhd

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

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

architecture Behavioral of Slow_Clk is
signal count : integer :=1;
signal clk_status : std_logic := '0';

begin
    process (Clk_in) begin
    if (rising_edge(Clk_in)) then
        count <= count + 1; --Incrementing the counter
        if (count = 5) then --100000000
            clk_status <= not clk_status; --Inverting the clock status
            Clk_out <= clk_status;
            count <=1; --Reset counter
        end if;
    end if;
    end process;

end Behavioral;
```

Basys3.xdc

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports Clk]
    set_property IOSTANDARD LVCMOS33 [get_ports Clk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports Clk]


## LEDs
set_property PACKAGE_PIN U16 [get_ports {LED[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]
set_property PACKAGE_PIN E19 [get_ports {LED[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]
set_property PACKAGE_PIN U19 [get_ports {LED[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]
set_property PACKAGE_PIN V19 [get_ports {LED[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]
set_property PACKAGE_PIN P1 [get_ports {Zero}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Zero}]
set_property PACKAGE_PIN L1 [get_ports {Overflow}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Overflow}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {Seven_Segment_Display[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports
{Seven_Segment_Display[0]}]
set_property PACKAGE_PIN W6 [get_ports {Seven_Segment_Display[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports
{Seven_Segment_Display[1]}]
set_property PACKAGE_PIN U8 [get_ports {Seven_Segment_Display[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports
{Seven_Segment_Display[2]}]
set_property PACKAGE_PIN V8 [get_ports {Seven_Segment_Display[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports
{Seven_Segment_Display[3]}]
set_property PACKAGE_PIN U5 [get_ports {Seven_Segment_Display[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports
{Seven_Segment_Display[4]}]
set_property PACKAGE_PIN V5 [get_ports {Seven_Segment_Display[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports
{Seven_Segment_Display[5]}]
```

```
set_property PACKAGE_PIN U7 [get_ports {Seven_Segment_Display[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports
{Seven_Segment_Display[6]}]


set_property PACKAGE_PIN U2 [get_ports {Anode[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode[0]}]
set_property PACKAGE_PIN U4 [get_ports {Anode[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode[1]}]
set_property PACKAGE_PIN V4 [get_ports {Anode[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode[2]}]
set_property PACKAGE_PIN W4 [get_ports {Anode[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode[3]}]


##Buttons
set_property PACKAGE_PIN U18 [get_ports Reset]
    set_property IOSTANDARD LVCMOS33 [get_ports Reset]
```

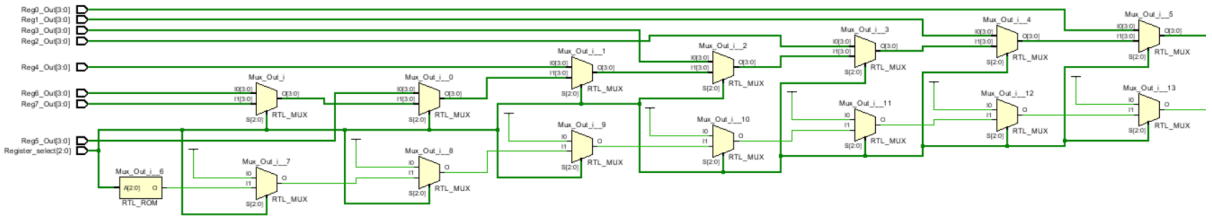
Additional

A. Changed Multiplexer_8_4 from the below code to code given under ***Multiplexer_8_4 .vhd***

If-else conditions created many multiplexers. To reduce the circuit components, we simplified the circuit by adding 4 multiplexers each for each bit. You can verify by seeing both schematic diagrams.

```
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 Multiplexer_8_4 is
    Port ( Reg0_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg1_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg2_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg3_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg4_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg5_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg6_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Reg7_Out : in STD_LOGIC_VECTOR (3 downto 0);
           Register_select : in STD_LOGIC_VECTOR (2 downto 0);
           Mux_Out : out STD_LOGIC_VECTOR (3 downto 0));
end Multiplexer_8_4;

architecture Behavioral of Multiplexer_8_4 is

begin
process (Register_select, Reg0_Out, Reg1_Out, Reg2_Out, Reg3_Out,
Reg4_Out, Reg5_Out, Reg6_Out, Reg7_Out)
begin
    if Register_select = "000" then
        Mux_Out <= Reg0_Out;
    elsif Register_select = "001" then
```

```
        Mux_Out <= Reg1_Out;
    elsif Register_select = "010" then
        Mux_Out <= Reg2_Out;
    elsif Register_select = "011" then
        Mux_Out <= Reg3_Out;
    elsif Register_select = "100" then
        Mux_Out <= Reg4_Out;
    elsif Register_select = "101" then
        Mux_Out <= Reg5_Out;
    elsif Register_select = "110" then
        Mux_Out <= Reg6_Out;
    elsif Register_select = "111" then
        Mux_Out <= Reg7_Out;
    end if;
end process;

end Behavioral;
```



B. Different programs give different LUT values which shows how much the circuit is optimized.

1) Storing 1,2,3 in three registers and adding

```
signal P_ROM : rom_type := (
          -- Adding 1,2,3 by storing each value in each registers.
          "101110000001", -- 0-- MOVI R7,1
          "101100000010", -- 1-- MOVI R6,2
          "101010000011", -- 2-- MOVI R5,3
          "001111100000", -- 3-- ADD R7,R6
          "001111010000", -- 4-- ADD R7,R5
          "110000000101", -- 5-- JZR R0,5
          "110000000101", -- 6-- JZR R0,5
          "110000000111"  -- 7-- JZR R0,7
```

```
    );
```

LUT: 35

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								35	49	0.00	0	0
impl_1	constrs_1	route_design Complete!	5.247	0.000	0.262	0.000	0.000	0.070	0	35	49	0.00	0	0

2) Adding 1 to 3 by looping

```
signal P_ROM : rom_type := (
          --Add 3 by looping
          "101000000011", -- MOVI R4,3
          "101100000001", -- MOVI R6,1
          "011100000000", -- NEG R6
          "001111000000", -- ADD R7,R4
          "001001100000", -- ADD R4,R6
          "111000000111", -- JZR R4,7
          "110000000011", -- JZR R0,3
          "111000000111" -- JZR R4,7
);
```

LUT: 40

Tcl Console | Messages | Log | Reports | Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								40	49	0.00	0	0
impl_1	constrs_1	route_design Complete!	5.711	0.000	0.263	0.000	0.000	0.076	0	40	50	0.00	0	0

C. Adding many numbers requires many registers to store each value. For example if we want to add from 1 to 10, we have to store them in 10 registers and to add them.

But if we add them by looping, we require less number of registers.

D. Program_Rom code to display 10 to 0 in 7 segment display

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity Program_ROM is
    Port ( MemorySelect : in STD_LOGIC_VECTOR (2 downto 0);
           Instruction : 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 P_ROM : rom_type := (

         --Program to display 10 to 0
         "101110001010", --0  Move R7 10
         "100100000001", --1  Move R2 01
         "010100000000", --2  Neg R2
         "001110100000", --3  R7<- R7+R2
         "111110000111", --4  JMP R7=0 PR7
         "110000000011", --5  JMP R0=0 PR3
         "110010000111", --6
         "110010000110"  --7

    );
begin
    Instruction <= P_ROM(to_integer(unsigned(MemorySelect)));

end Behavioral;
```

E. We used a slow clock while verifying the functionality on the development board. Slow clock is used to better observe the changes in the LEDs and seven segment display

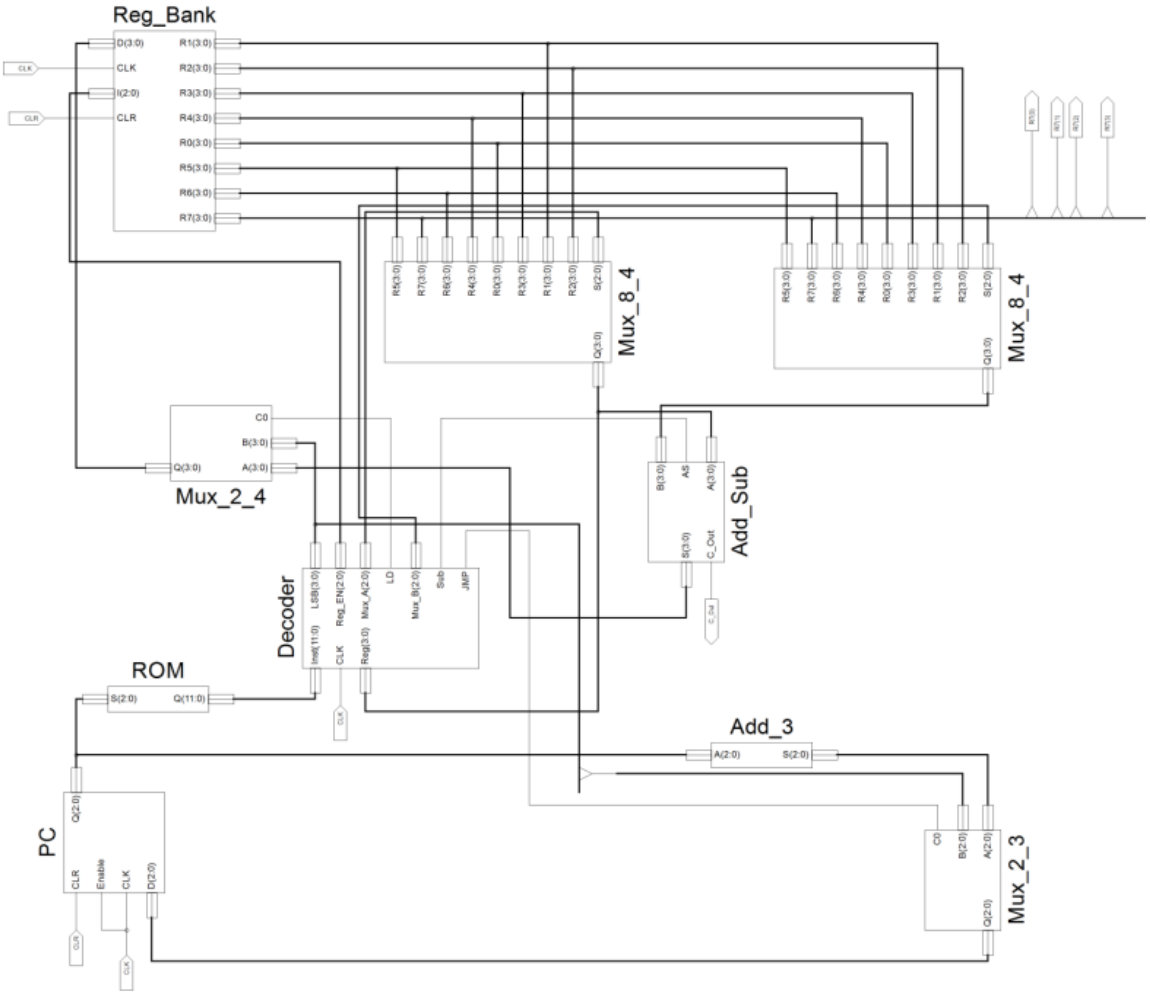


Figure 2 – A possible design of the nanoprocessor.

Image Source:
http://dilum.bandara.lk/wp-content/uploads/CourseNotes/CS2052CA/Lab-9-10-%E2%80%93-Nanoprocessor-Design-Competition.pdf

Resource utilization

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	40	0	20800	0.19
LUT as Logic	40	0	20800	0.19
LUT as Memory	0	0	9600	0.00
Slice Registers	49	0	41600	0.12
Register as Flip Flop	49	0	41600	0.12
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

Conclusion

In this lab project, we designed a 4-bit nanoprocessor capable of executing a simple set of instructions. The goal was to gain practical experience in designing and developing a nanoprocessor and understanding the components.

To build a functioning processor, we need to build the components individually and test their functionalities. We can use buses to efficiently implement our design instead of letting so many wires run around. As our processor won't understand assembly language instructions, we have to translate it into machine code, and have to hardcode that into our ROM.

Using the designs from previous labs will be helpful instead of building them from scratch. We understood how each component of the processor works internally and about how the processor decodes and executes instructions. We also gained practical experience in designing and developing a simple nanoprocessor, working collaboratively in a team, and integrating components developed by both team members. We also improved our communication, coordination, and responsibility-sharing skills.

Overall, this lab project provided valuable hands-on experience in digital design, computer organization, and teamwork, helping the team members enhance their understanding and skills in these areas.