CS-1050 Computer Organization and Digital Design

Lab 9-10 – Nano processor Design Competition

⇒ Suriya G. 220628E
⇒ Pratheep S. 220489C
⇒ Vithurshan S. 220672G
⇒ Vinushaanth S. 220668B **GROUP-24**

Contents

Lab Task

In this project, our team collaborated to develop a 4-bit Nanoprocessor aimed at executing a predefined set of instructions. We refined several core components learned from previous Labs, including 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, a 2-bit Multiplier, and a slow clock. To streamline the chip's design, we used buses, featuring 3, 4, and 12-bit configurations, to interconnect various modules, simplifying the wiring complexity.

As the Nanoprocessor operates in machine language, we hardcoded instructions in binary format and embedded them into the Program ROM, serving as the chip's operational blueprint. The deliberate pacing of the clock cycle enabled us to observe the Nanoprocessor's sequential execution steps, offering valuable insights into its operational behavior. Additionally, rigorous simulation procedures were conducted to ensure the functional integrity of each module, aligning with expected inputs and outputs. As a cohesive team effort, tasks were equally distributed among members, creating a collaborative environment where components were developed, designed, and validated by peers. This approach facilitated a culture of constructive feedback and continuous refinement.

This lab resulted in the successful development of a 4-bit Arithmetic Unit, capable of performing addition, subtraction, and multiplication operations on signed integers, alongside k-way b-bit Multiplexers, Program ROM, and Instruction Decoder functionalities. These accomplishments were validated through simulation and implementation on a BASYS3 board.

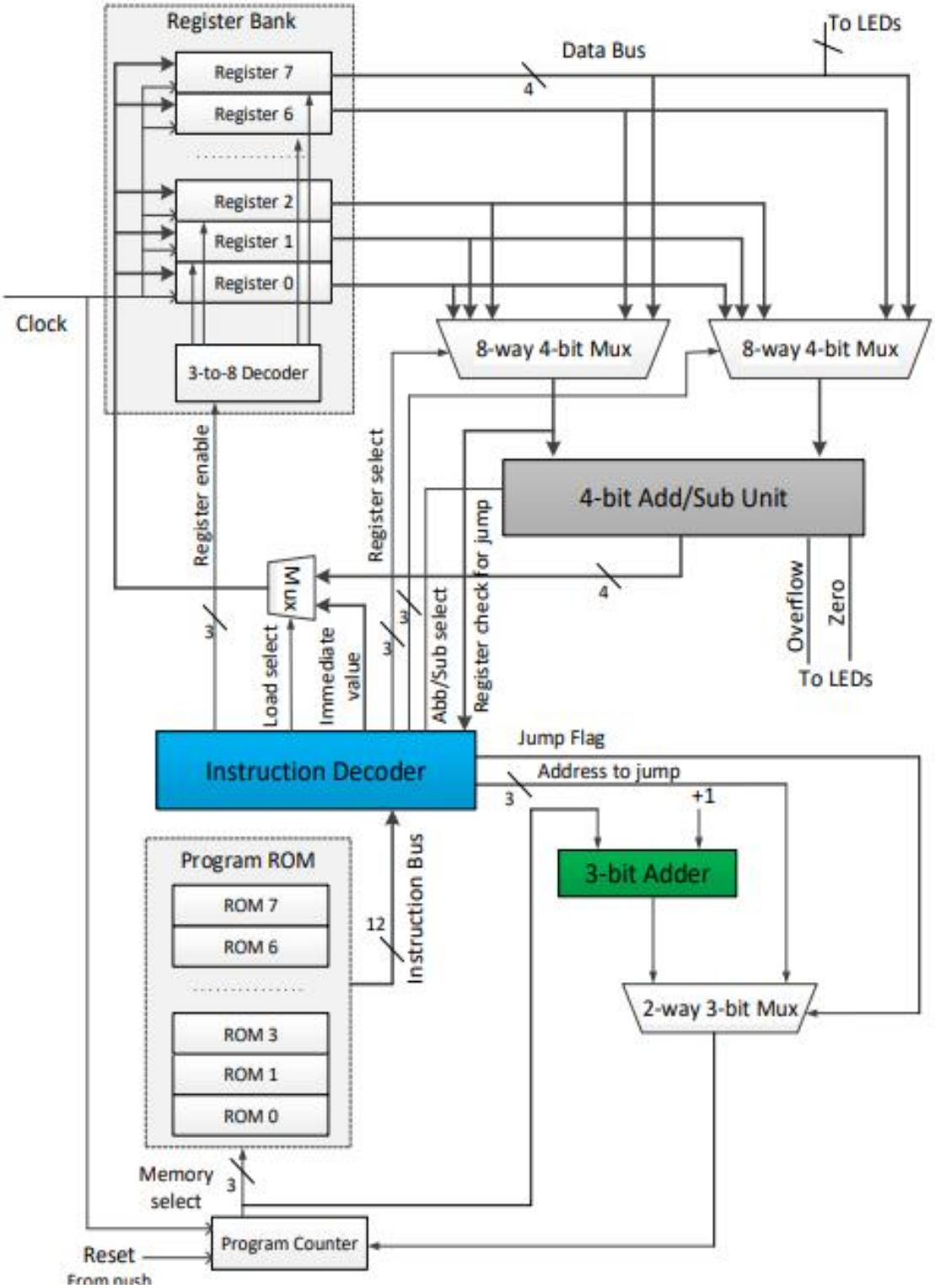


Figure 1 High level Diageam of Expected Nanoprocessor (Src-Lab instruction file)

Contribution of Team Members

Team Member	Contribution	Time Spent
Suriya G.	Instruction Decoder, 2-bit Multiplier, 13-bit program ROM (Designing), Modifying some components for adding multiplier.	14 hours
Pratheep S.	Nano processor with seven-segment display,4-bit adder/subtractor, Register with Reset, Register bank (Designing and simulation) Constraint file Report Preparation	18 hours
Vithurshan S.	n-way b-bit multiplexers, Program Decoder, ROM Incrementor (Designing and simulation) Assembling Nano processor for basic instructions	20 hours
Vinushaanth S.	3-bit adder, Program Counter, Program ROM (Designing and simulation) 13-bit Program ROM (Simulation)	15 hours
Together	Debugging, Testing on the Basys3 board , Assembling Nanoprocessor with Multiplier	6 hours

Nano Processor V1

Instructions and their Machine Code Representations

	Assembly Instruction	Machine Code

Calculation of total between the numbers 1 and 3 (Store 3 in R1 and add it with R7 reduce the value in R1 and add with R7 till it becomes zero)	MOVI R1,3
	MOVI R2,1
	NEG R2
	ADD R7,R1
	ADD R1,R2
	JZR R1,7
	JZR R0,3
	JZR R0,7
		"100010000011"
		"100100000001"
		"010100000000"
		"001110010000"
		"000010100000"
		"110010000111"
		"110000000011"
		"110000000111"

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"

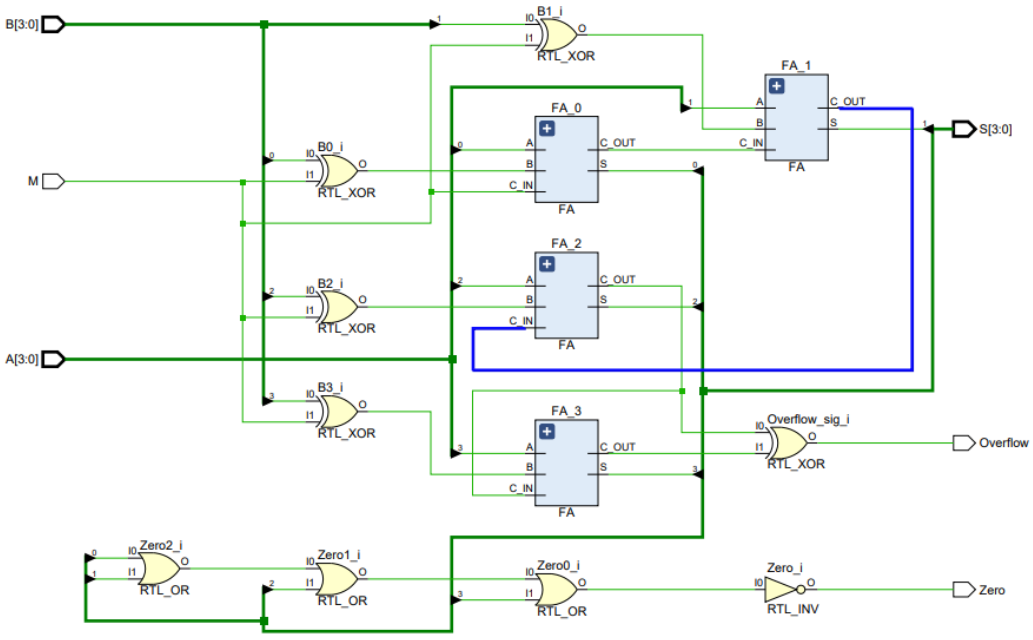
Components of Nano Processor and their Functions

Component	Function	Structure
4-bit Adder / Subtractor	It will add or subtract 4-bit inputs. (Subtraction is done by adding two's complement) This circuit can be used to negate a number. This has flags to indicate if there is an overflow in the process or if the output is zero.	It has 4 Full Adders in it. It has an input to select whether the circuit acts as an Adder or Subtractor.
3-bit Adder	The main function of the 3-bit adder is incrementing the current address stored in the program counter one by one.	It is implemented by using three full adders.
3-bit Program Counter	Here, it provides the address to access the next instruction to be executed.	It is implemented by using 3 D-Flipflops, a Clock signal, and a Reset option.
Register Bank	4-bit data can be stored in registers. The register can be selected through input.	It has 8 4-bit registers in it. A 3 to 8 Decoder is used to select a specific register in which the data should be stored.

Program ROM	In each ROM, 12-bit instructions(assembly program) are stored. The 13 bits are divided into various fields such as opcode and operands	It has 8 ROMs inside. Memory select selects which program has to be executed and sends the instruction to the instruction decoder.
2-way 3-bit Multiplexer	Used in Rom_Incrementor for choose one of the 3 bit values from Jumpaddress and 3 bit adder's output	
2-way 4-bit Multiplexer	Used for choose one of the values from immediate value and output from add/sub unit	
8-way 4-bit Multiplexer	it choose one out of 8 register and read data from that register and send it to the add/sub unit or program_decoder(RCJ)	
Instruction Decoder	Decodes instruction fetched from program rom and activates necessary units	It has a 2 to 4 Decoder to decode the first two bits (from the MSB side)

1. 4-bit Adder-Subtractor Unit

Schematic



Design Source File

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

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

architecture Behavioral of Add_Sub 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 B0,B1,B2,B3 : std_logic; --signals to bring input
signal C0,C1,C2,C3,Overflow_sig : std_logic;
signal S_sig : std_logic_vector(3 downto 0);
begin
B0 <= B(0) xor M;
B1 <= B(1) xor M;
B2 <= B(2) xor M;
B3 <= B(3) xor M;

FA_0 : FA
    port map (
```

```
        A => A(0),
        B => B0,
        C_IN => M,
        S => S_sig(0),
        C_OUT => C0);
FA_1 : FA
    port map (
        A => A(1),
        B => B1,
        C_IN => C0,
        S => S_sig(1),
        C_OUT => C1);

FA_2 : FA
    port map (
        A => A(2),
        B => B2,
        C_IN => C1,
        S => S_sig(2),
        C_OUT => C2);
FA_3 : FA
    port map (
        A => A(3),
        B => B3,
        C_IN => C2,
        S => S_sig(3),
        C_OUT => C3);
S <= S_sig;
Overflow_sig <= C2 xor C3;
Zero <= not (S_sig(0) or S_sig(1) or S_sig(2) or S_sig(3));
--Zero <= '1' when (S_sig = "0000") else '0';
Overflow <= Overflow_sig;
```

```
end Behavioral;
```

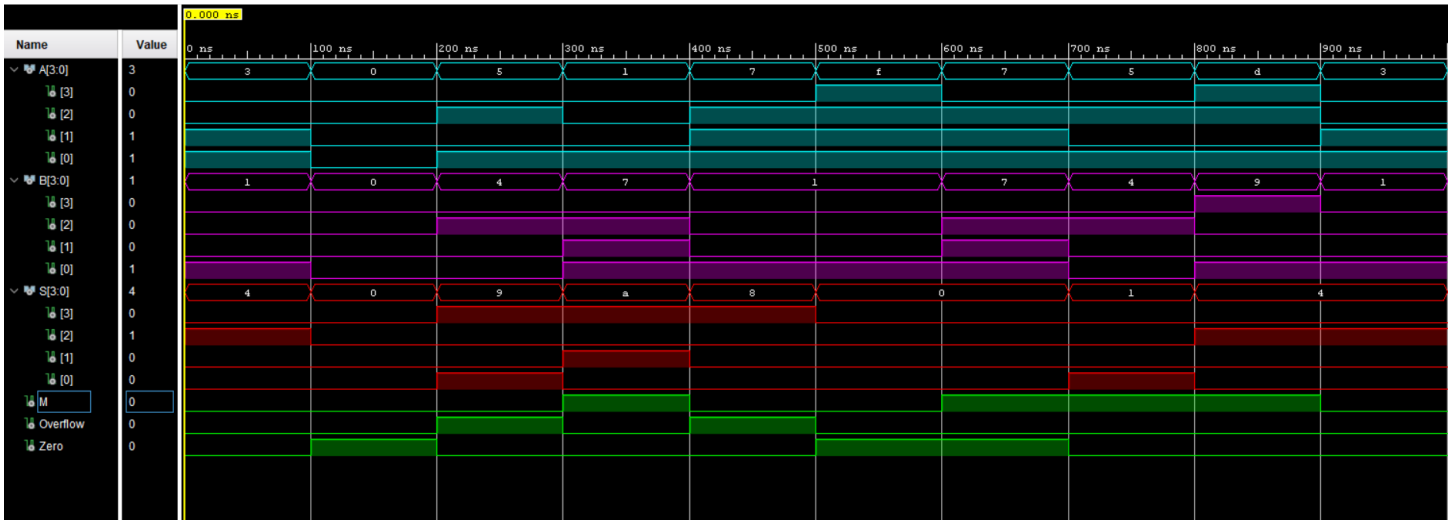
Simulation File

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

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

architecture Behavioral of TB_Add_Sub is
component Add_Sub
 Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
      B : in STD_LOGIC_VECTOR (3 downto 0);
      M : in STD_LOGIC;
      S : out STD_LOGIC_VECTOR (3 downto 0);
      Overflow : out STD_LOGIC;
      Zero : out STD_LOGIC);
end component;

signal A,B,S : std_logic_vector (3 downto 0);
signal M,Overflow,Zero : std_logic;

begin
    UUT : Add_Sub port map (
        A => A,
        B => B,
        M => M,
        S => S,
        Overflow => Overflow,
        Zero => Zero);
process
```

```
begin
    M <= '0';
    A <= "0011";
    B <= "0001";
    wait for 100 ns;

    A <= "0000";
    B <= "0000";
    wait for 100 ns;
    --220489C = 11 0101 1101 0100 1001
    A <= "0101";
    B <= "0100";
    wait for 100 ns;

    M <= '1';
    B <= "0111";
    A <= "0001";

    wait for 100 ns;

    M <= '0';
    A <= "0111";
    B <= "0001";
    wait for 100 ns;

    A <= "1111";
    B <= "0001";
    wait for 100 ns;

    M <= '1';
    A <= "0111";
    B <= "0111";
```

```
        wait for 100 ns;

        --220489C =11 0101 1101 0100 1001
        A <= "0101";
        B <= "0100";
        wait for 100 ns;
        A <= "1101";
        B <= "1001";
        wait for 100 ns;
    end process;

    end Behavioral;
```

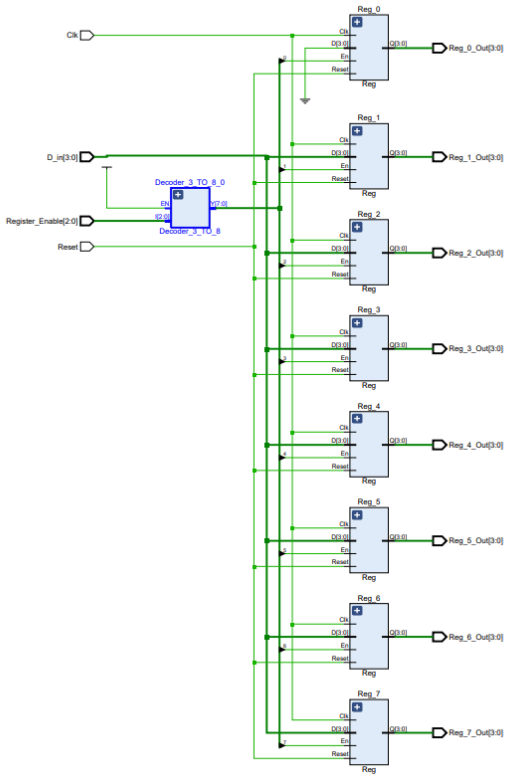
Timing Diagram



If M = 1 then the circuit will act as a subtractor and if M = 0 then the circuit will act as an Adder.

2. Register Bank

Schematic



Design Source File

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Reg_Bank is
    Port ( Register_Enable : in STD_LOGIC_VECTOR (2 downto 0);
           D_in : in STD_LOGIC_VECTOR (3 downto 0);
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Reg_0_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_1_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_2_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_3_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_4_Out : out STD_LOGIC_VECTOR (3 downto 0);
```

```
            Reg_5_Out : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_6_Out : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_7_Out : out STD_LOGIC_VECTOR (3 downto 0));
end Reg_Bank;

architecture Behavioral of Reg_Bank is
component Reg
    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_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;
signal Y : std_logic_vector (7 downto 0);
begin
Decoder_3_TO_8_0 : Decoder_3_TO_8
    port map (
        I => Register_Enable,
        EN => '1',
        Y => Y);
Reg_0 : Reg
    port map (
        D => "0000",
        Clk => Clk,
        EN => Y(0),
        Reset => Reset,
```

```
        Q => Reg_0_Out);
Reg_1 : Reg
    port map (
        D => D_in,
        Clk => Clk,
        EN => Y(1),
        Reset => Reset,
        Q => Reg_1_Out);
Reg_2 : Reg
    port map (
        D => D_in,
        Clk => Clk,
        EN => Y(2),
        Reset => Reset,
        Q => Reg_2_Out);
Reg_3 : Reg
    port map (
        D => D_in,
        Clk => Clk,
        EN => Y(3),
        Reset => Reset,
        Q => Reg_3_Out);
Reg_4 : Reg
    port map (
        D => D_in,
        Clk => Clk,
        EN => Y(4),
        Reset => Reset,
        Q => Reg_4_Out);
Reg_5 : Reg
    port map (
        D => D_in,
```

```
        Clk => Clk,
        EN => Y(5),
        Reset => Reset,
        Q => Reg_5_Out);
Reg_6 : Reg
    port map (
        D => D_in,
        Clk => Clk,
        EN => Y(6),
        Reset => Reset,
        Q => Reg_6_Out);
Reg_7 : Reg
    port map (
        D => D_in,
        Clk => Clk,
        EN => Y(7),
        Reset => Reset,
        Q => Reg_7_Out);
end Behavioral;
```

Simulation Source File

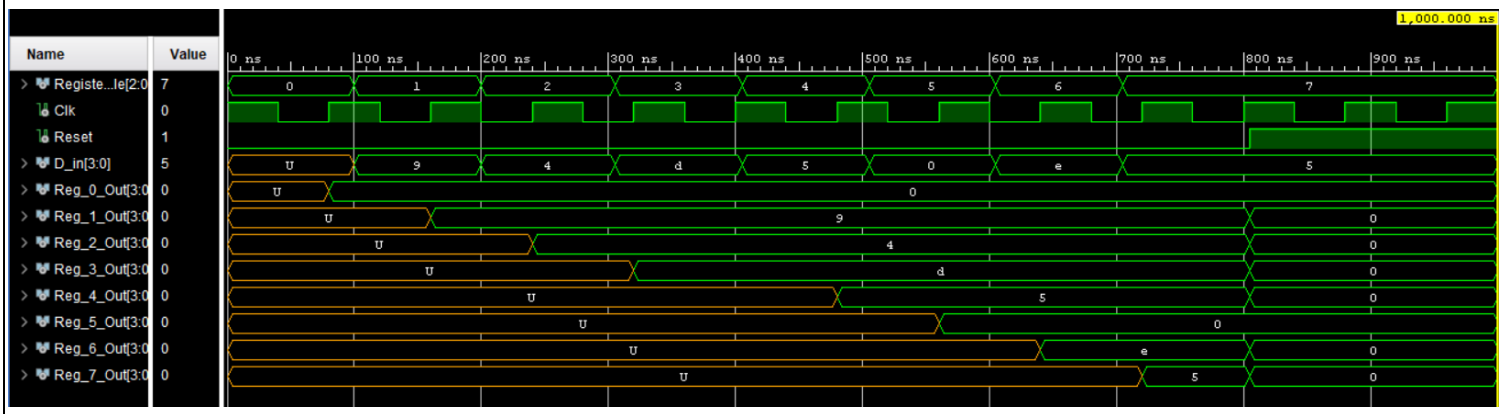
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity TB_Reg_Bank is
--  Port ( );
end TB_Reg_Bank;

architecture Behavioral of TB_Reg_Bank is
component Reg_Bank
    Port ( Register_Enable : in STD_LOGIC_VECTOR (2 downto 0);
           D_in : in STD_LOGIC_VECTOR (3 downto 0);
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
```

```
            Reg_0_Out : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_1_Out : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_2_Out : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_3_Out : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_4_Out : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_5_Out : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_6_Out : out STD_LOGIC_VECTOR (3 downto 0);
            Reg_7_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 D_in : STD_LOGIC_VECTOR (3 downto 0);
signal
Reg_0_Out,Reg_1_Out,Reg_2_Out,Reg_3_Out,Reg_4_Out,Reg_5_Out,Reg_6_Out,Reg_7_Out :
STD_LOGIC_VECTOR(3 downto 0);


begin
UUT : Reg_Bank
    port map (
        Register_Enable => Register_Enable,
        D_in => D_in,
        Clk => Clk,
        Reset => Reset,
        Reg_0_Out => Reg_0_Out,
        Reg_1_Out => Reg_1_Out,
        Reg_2_Out => Reg_2_Out,
        Reg_3_Out => Reg_3_Out,
        Reg_4_Out => Reg_4_Out,
        Reg_5_Out => Reg_5_Out,
        Reg_6_Out => Reg_6_Out,
        Reg_7_Out => Reg_7_Out);
process
```

```
begin
    Clk <= not (Clk);
    wait for 40 ns;
end process;
process
begin
    --220489C = 11 0101 1101 0100 1001
    --220672G = 11 0101 1110 0000 0000
    Reset <= '0';
    Register_Enable <= "000";
    wait for 100 ns;
    Register_Enable <= "001";
    D_in <= "1001";
    wait for 100 ns;
    Register_Enable <= "010";
    D_in <= "0100";
    wait for 105 ns;
    Register_Enable <= "011";
    D_in <= "1101";
    wait for 100 ns;
    Register_Enable <= "100";
    D_in <= "0101";
    wait for 100 ns;
    Register_Enable <= "101";
    D_in <= "0000";
    wait for 100 ns;
    Register_Enable <= "110";
    D_in <= "1110";
    wait for 100 ns;
    Register_Enable <= "111";
    D_in <= "0101";
    wait for 100 ns;
```

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

Timing Diagram

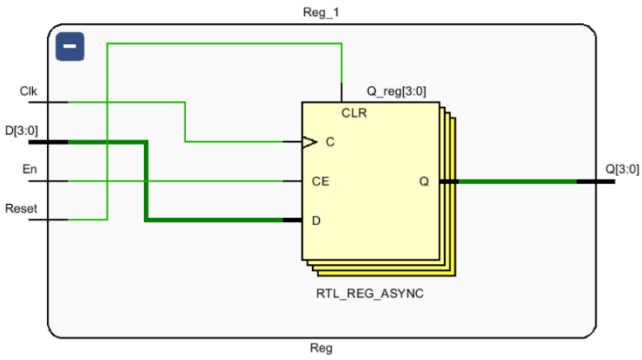


Design Source File for 4-bit Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Reg is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           En : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end Reg;
architecture Behavioral of Reg is
begin
process (Clk,Reset) begin
    if Reset = '1' then
        Q <= "0000";
    elsif (rising_edge(Clk)) then
        if En ='1' then
```

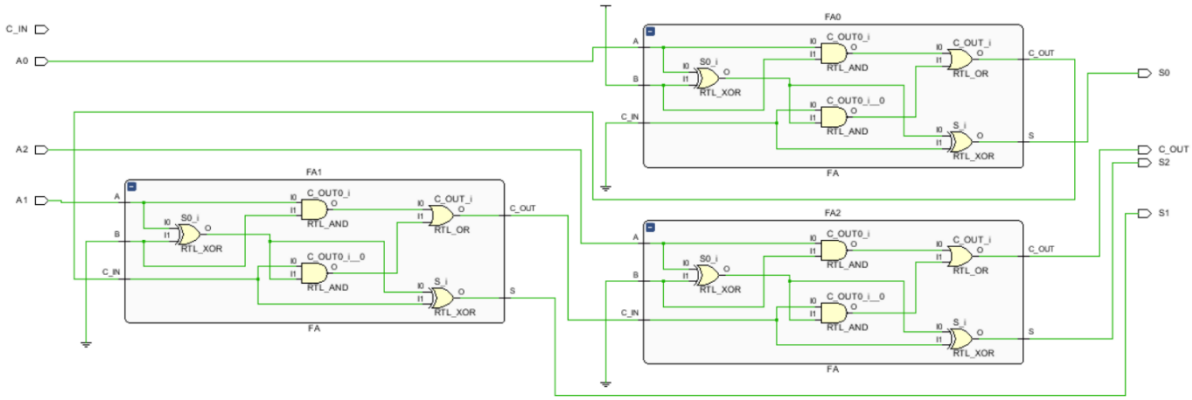
```
            Q <= D;
        end if;
    end if;
end process;
end Behavioral;
```

Schematic of 4-bit Register



3. 3-bit Adder

Schematic



Design Source File

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

entity RCA3 is
    Port ( A0 : in STD_LOGIC;
           A1 : in STD_LOGIC;
           A2 : in STD_LOGIC;
```

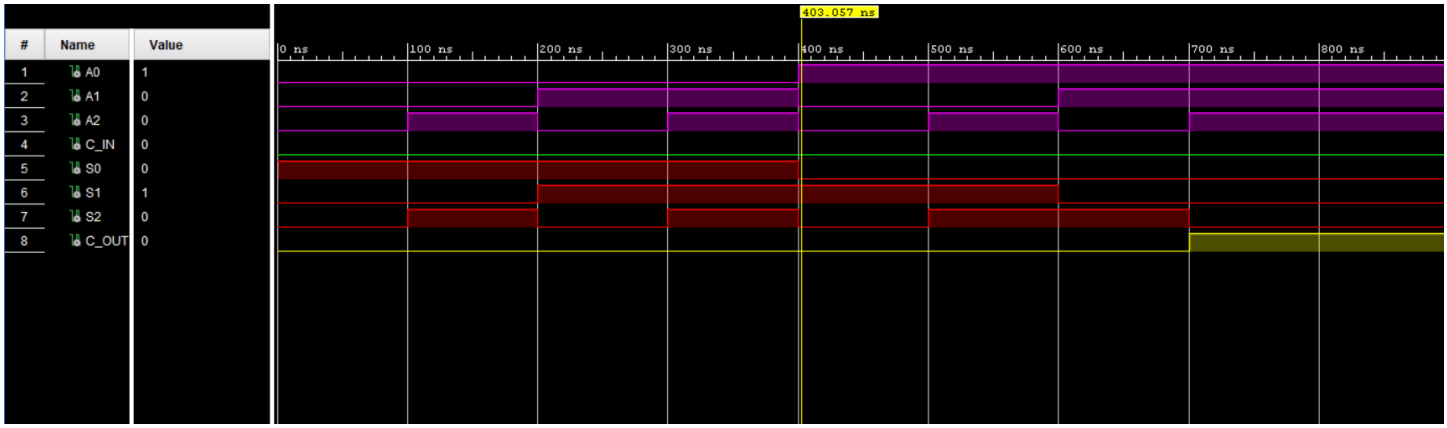
```
            S0 : out STD_LOGIC;
            S1 : out STD_LOGIC;
            S2 : out STD_LOGIC;
            C_IN : in STD_LOGIC;
            C_OUT : out STD_LOGIC);
end RCA3;

architecture Behavioral of RCA3 is
COMPONENT FA
PORT(
A,B,C_IN :IN STD_LOGIC;
S,C_OUT :OUT STD_LOGIC);
END COMPONENT;
SIGNAL FA0_C_OUT,FA1_C_OUT,FA2_C_OUT:STD_LOGIC;

begin
FA0:FA
PORT MAP(
A=>A0,
B=>'1',
C_IN=>'0',
S=>S0,
C_OUT=>FA0_C_OUT

);
FA1:FA
PORT MAP(
A=>A1,
B=>'0',
C_IN=>FA0_C_OUT,
S=>S1,
C_OUT=>FA1_C_OUT
);
FA2:FA
PORT MAP(
A=>A2,
B=>'0',
C_IN=>FA1_C_OUT,
S=>S2,
C_OUT=>FA2_C_OUT
);
C_OUT<=FA2_C_OUT;

end Behavioral;
```

Simulation Source File

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

entity RCA3TB is
--  Port ( );
end RCA3TB;

architecture Behavioral of RCA3TB is
COMPONENT RCA3
PORT(
A0,A1,A2,C_IN:IN STD_LOGIC;
S0,S1,S2,C_OUT:OUT STD_LOGIC);
END COMPONENT;
SIGNAL A0,A1,A2,C_IN:STD_LOGIC;
SIGNAL S0,S1,S2,C_OUT:STD_LOGIC;

begin
UUT:RCA3 PORT MAP(
A0=>A0,
A1=>A1,
A2=>A2,
S0=>S0,
S1=>S1,
S2=>S2,
C_IN=>C_IN,
C_OUT=>C_OUT
);
PROCESS
BEGIN
C_IN<='0';
A0<='0';
A1<='0';
A2<='0';

WAIT FOR 100 NS;
A0<='0';
A1<='0';
A2<='1';

WAIT FOR 100 NS;
A0<='0';
A1<='1';
A2<='0';
```

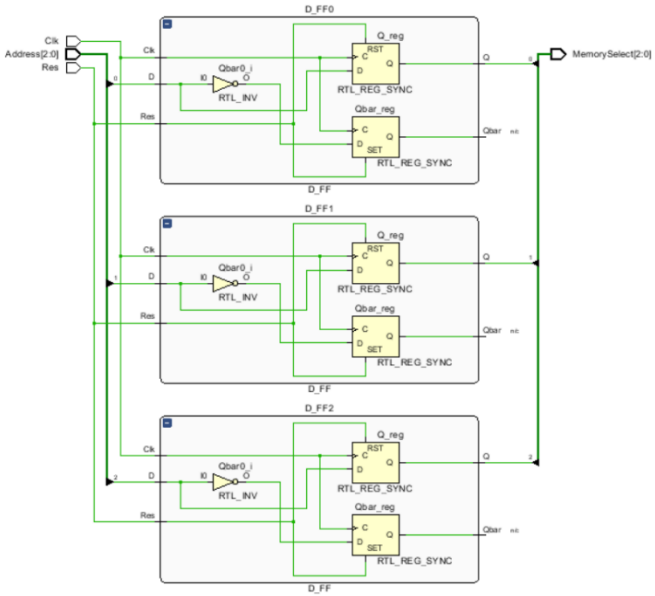
```
WAIT FOR 100 NS;
A0<='0';
A1<='1';
A2<='1';
WAIT FOR 100 NS;
A0<='1';
A1<='0';
A2<='0';
WAIT FOR 100 NS;
A0<='1';
A1<='0';
A2<='1';
WAIT FOR 100 NS;
A0<='1';
A1<='1';
A2<='0';
WAIT FOR 100 NS;
A0<='1';
A1<='1';
A2<='1';

WAIT;
END PROCESS;

end Behavioral;
```

Timing Diagram



4. Program Counter

Schematic



Design Source File

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

entity program_counter_3bit is
    Port ( Address : in STD_LOGIC_VECTOR (2 downto 0);
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           MemorySel : out STD_LOGIC_VECTOR (2 downto 0));
end program_counter_3bit;

architecture Behavioral of program_counter_3bit is
component D_FF
port(
           D : in STD_LOGIC;
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC;
           Qbar : out STD_LOGIC);
```

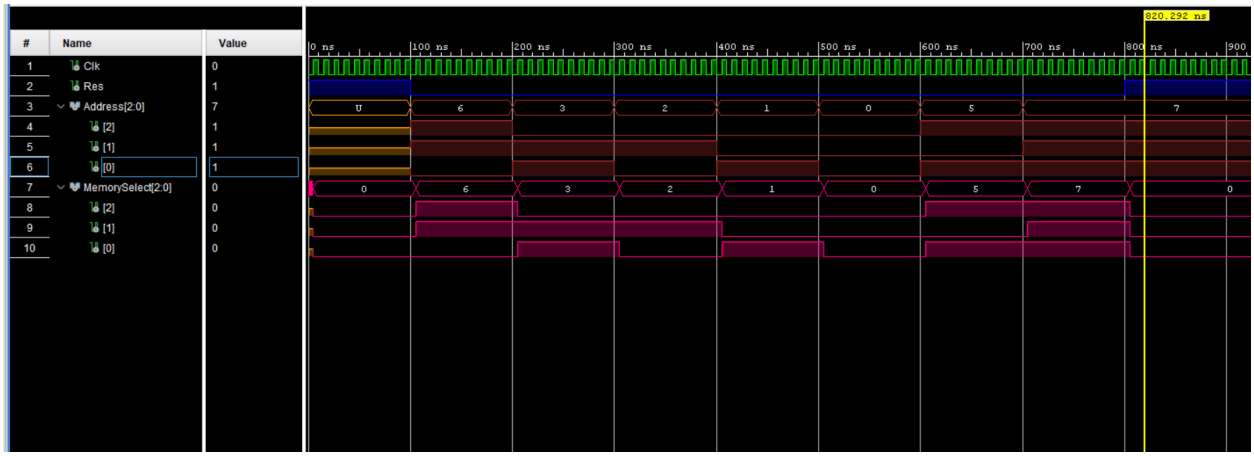
```
end component;

begin

D_FF_0:D_FF PORT MAP(
D=>ADDRESS(0),
Res=>Res,
Clk=>Clk,
Q=>MEMORYSEL(0));

D_FF_1:D_FF PORT MAP(
D=>ADDRESS(1),
Res=>Res,
Clk=>Clk,
Q=>MEMORYSEL(1));

D_FF_2:D_FF PORT MAP(
D=>ADDRESS(2),
Res=>Res,
Clk=>Clk,
Q=>MEMORYSEL(2));

end Behavioral;
```

Simulation Source File

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

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

architecture Behavioral of PC_COU_TB is
COMPONENT program_counter_3bit
port(
           Address : in STD_LOGIC_VECTOR (2 downto 0);
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           MemorySel : out STD_LOGIC_VECTOR (2 downto 0));
end component;
signal Address,MemorySel:std_logic_vector(2 downto 0);
signal Res:std_logic;
signal Clk:std_logic:='0';
```

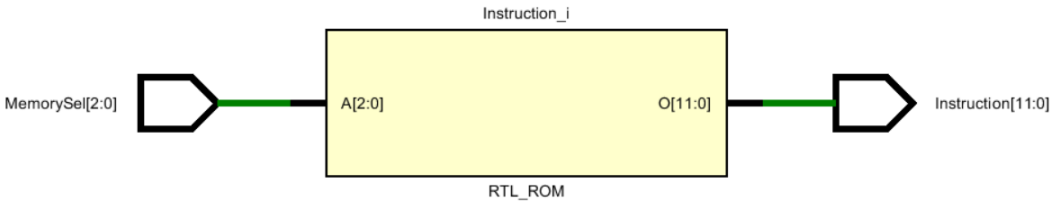
```
begin
uut:program_counter_3bit port map(
Address=>Address,
Res=>Res,
Clk=>clk,
MemorySel=>MemorySel);

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

 process
 begin
 Res<='1';
 wait for 100 ns;

 Res<='0';
 Address<="000";
 wait for 100 ns;
 Address<="001";
 wait for 100 ns;
 Address<="010";
 wait for 100 ns;
 Address<="011";
 wait for 100 ns;
 Address<="100";
 wait for 100 ns;
 Address<="101";
 wait for 100 ns;
 Address<="110";
 wait for 100 ns;
 Address<="111";
 wait for 100 ns;

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

Timing Diagram



5. Program ROM

Schematic



Design Source File

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

entity program_rom is
    Port ( MemorySel : 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);
```

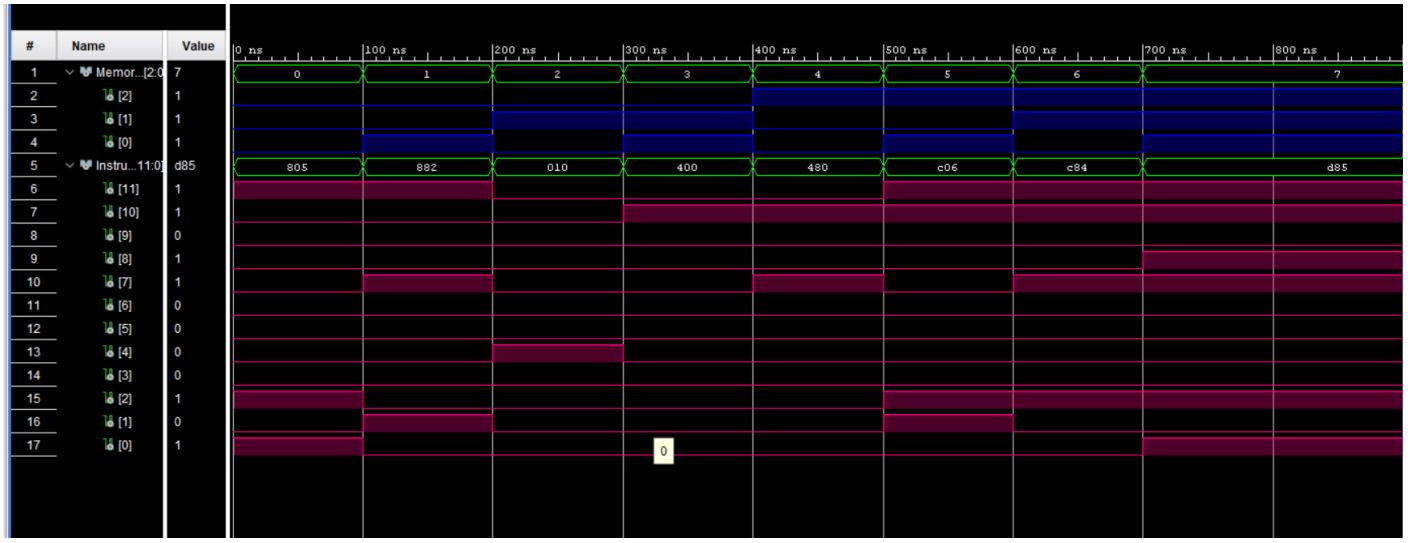
```
signal p_rom :rom_type:=(
 "100010000011", -- MOVI R1,3 --line->0
 "100100000001", -- MOVI R2,1
 "010100000000", -- NEG R2
 "001110010000", -- ADD R7,R1 --line->3
 "000010100000", -- ADD R1,R2
 "110010000111", -- JZR R1,7
 "110000000011", -- JZR R0,3
 "110000000111"  -- JZR R0,7 --line->7
);

begin
instruction<=p_rom(to_integer(unsigned(MemorySel)));
end Behavioral;
```

Simulation Source File

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

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

architecture Behavioral of program_rom_tb is
component program_rom
port(
          MemorySel : in STD_LOGIC_VECTOR (2 downto 0);
          Instruction : out STD_LOGIC_VECTOR (11 downto 0));
end component;
signal MemorySel:std_logic_vector(2 downto 0);
signal Instruction:std_logic_vector(11 downto 0);

begin
uut:program_rom port map(
MemorySel=>MemorySel,
Instruction=>Instruction);
process
begin
MemorySel<="000";
wait for 100 ns;
MemorySel<="001";
wait for 100 ns;
MemorySel<="010";
```

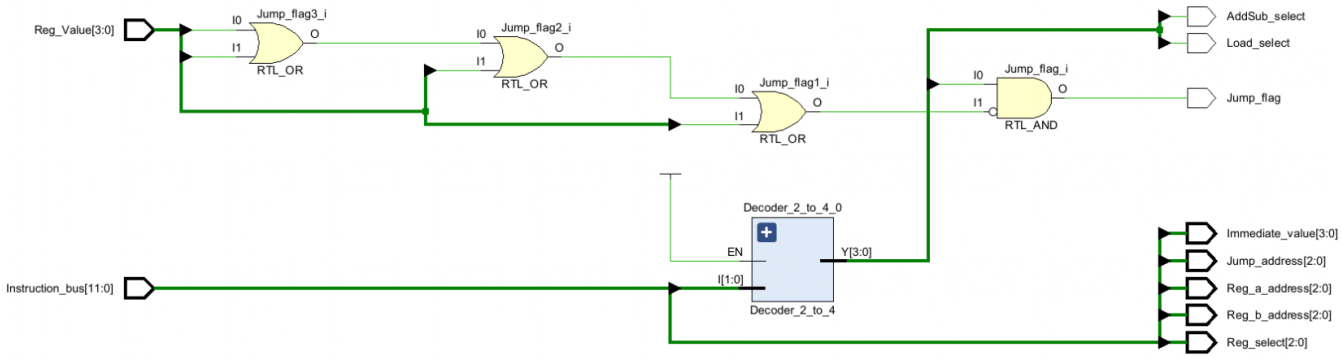
```
wait for 100 ns;
MemorySel<="011";
wait for 100 ns;
MemorySel<="100";
wait for 100 ns;
MemorySel<="101";
wait for 100 ns;
MemorySel<="110";
wait for 100 ns;
MemorySel<="111";
wait;
end process;


end Behavioral;
```

Timing Diagram



6. Instruction Decoder

Schematic



Design Source File

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

entity Instruction_decoder is
    Port ( Instruction_bus : in STD_LOGIC_VECTOR (11 downto 0); -- input from
program rom
           Reg_Value : in STD_LOGIC_VECTOR (3 downto 0); -- to check if a
register has value zero stored in it
           -- reg which we check is given with JZR instruction
           Reg_a_address : out STD_LOGIC_VECTOR (2 downto 0); -- selecting input
for AU via 8 way 2 bit mux
           Reg_b_address : out STD_LOGIC_VECTOR (2 downto 0); -- selecting input
for AU via 8 way 2 bit mux
           Reg_select : out STD_LOGIC_VECTOR (2 downto 0); -- to select Reg from
reg bank to write to a reg
           AddSub_select : out STD_LOGIC; -- selecting Add or Subtract (if neg
instruction sub will be selceted)

           Jump_flag : out STD_LOGIC; -- to  jump to another instruction in
PC(Program counter),
           -- by checking Reg_Value and this output connected to 2 way 3 bit MUX
selector (PC area)
           Jump_address : out STD_LOGIC_VECTOR (2 downto 0); -- which instruction
to execute, this output connected to 2 way 3 bit MUX
```

```
          Load_select : out STD_LOGIC; -- select between AU output and immediate
value (connected to 2 way 4 bit MUX)
          Immediate_value : out STD_LOGIC_VECTOR (3 downto 0)); -- immediate
value given for movi (a number) or JZR (a address in program rom)
end Instruction_decoder;

architecture Behavioral of Instruction_decoder is

component Decoder_2_to_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 ADD, NEG, MOVI, JZR :STD_LOGIC;

begin
Decoder_2_to_4_0: Decoder_2_to_4
port map( I(0)=>Instruction_bus(10),
          I(1)=>Instruction_bus(11),
          EN=>'1',
          Y(0) => ADD,
          Y(1) => NEG,
          Y(2) => MOVI,
          Y(3) => JZR);

Reg_a_address <= Instruction_bus(9 downto 7);
Reg_b_address <= Instruction_bus(6 downto 4);
Reg_select <= Instruction_bus(9 downto 7);
AddSub_select <= NEG; -- NEG is substracting from zero
Jump_Flag <= JZR AND (NOT (Reg_Value(0) OR Reg_Value(1) OR Reg_Value(2) OR
Reg_Value(3))) ;
Jump_Address <=Instruction_bus(2 downto 0);
Load_select <= MOVI;
Immediate_value <= Instruction_bus(3 downto 0);

end Behavioral;
```

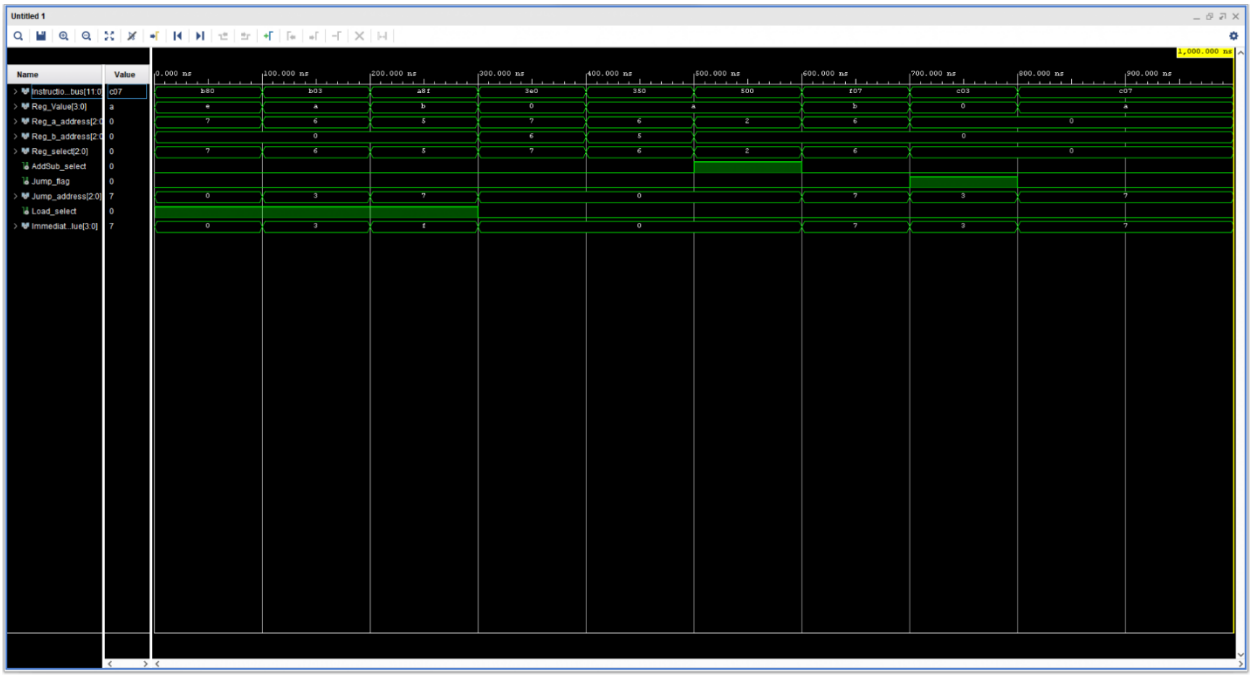
Simulation Source File

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

entity Instruction_decoder_Sim is
```

```
--  Port ( );
end Instruction_decoder_Sim;

architecture Behavioral of Instruction_decoder_Sim is
component Instruction_Decoder
    Port ( Instruction_bus : in STD_LOGIC_VECTOR (11 downto 0);
           Reg_Value : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_a_address : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_b_address : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_select : out STD_LOGIC_VECTOR (2 downto 0);
           AddSub_select : out STD_LOGIC;
           Jump_flag : out STD_LOGIC;
           Jump_address : out STD_LOGIC_VECTOR (2 downto 0);
           Load_select : out STD_LOGIC;
           Immediate_value : out STD_LOGIC_VECTOR (3 downto 0));
end component;

SIGNAL Instruction_bus : STD_LOGIC_VECTOR (11 downto 0);
SIGNAL Reg_Value : STD_LOGIC_VECTOR (3 downto 0);
SIGNAL Reg_a_address : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL Reg_b_address : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL Reg_select : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL AddSub_select : STD_LOGIC;
SIGNAL Jump_flag : STD_LOGIC;
SIGNAL Jump_address : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL Load_select : STD_LOGIC;
SIGNAL Immediate_value : STD_LOGIC_VECTOR (3 downto 0);


begin
UUT: Instruction_Decoder PORT MAP(
    Instruction_bus => Instruction_bus,
    Reg_Value => Reg_Value,
    Reg_a_address => Reg_a_address,
    Reg_b_address => Reg_b_address,
    Reg_select => Reg_select,
    AddSub_select => AddSub_select,
    Jump_flag => Jump_flag,
    Jump_address => Jump_address,
    Load_select => Load_select,
    Immediate_value => Immediate_value
    );
process
begin
    --1
```

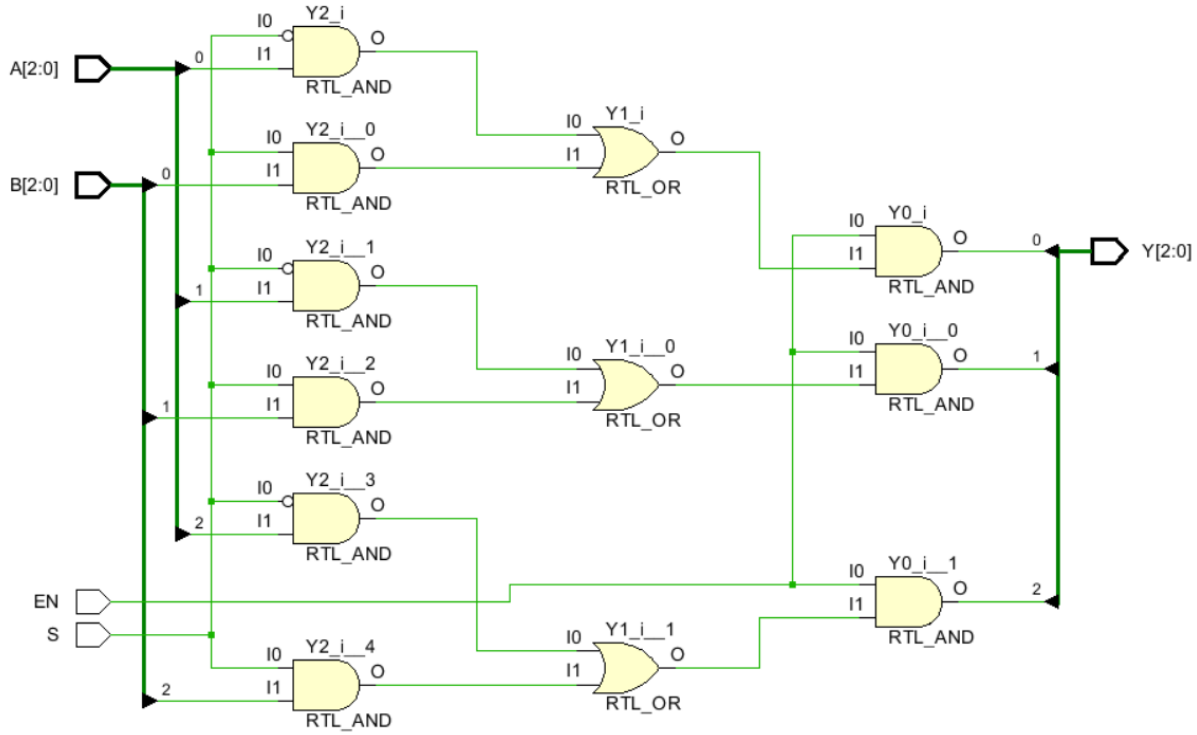
```
        Reg_Value <= "1110";
        Instruction_bus <= "101110000000";
        wait for 100ns;
        --2
        Reg_Value <= "1010";
        Instruction_bus <= "101100000011";
        wait for 100ns;
        --3
        Reg_Value <= "1011";
        Instruction_bus <= "101010001111";
        wait for 100ns;
        --4
        Reg_Value<= "0000";
        Instruction_bus <= "001111100000";
        wait for 100ns;
        --5
        Reg_Value <= "1010";
        Instruction_bus <= "001101010000";
        wait for 100ns;
        --6
        Reg_Value <= "1010";
        Instruction_bus<="010100000000";
        wait for 100ns;
        --7
        Reg_Value <= "1011";
        Instruction_bus<="111100000111";
        wait for 100ns;
        --8
        Reg_Value <= "0000";
        Instruction_bus<="110000000011";
        wait for 100ns;
        --9
        Reg_Value <= "1010";
        Instruction_bus<="110000000111";
        wait for 100ns;
        wait;

    end process;

    end Behavioral;
```

Timing Diagram



7. 2-way 3-bit Multiplexer

Schematic



Design Source File

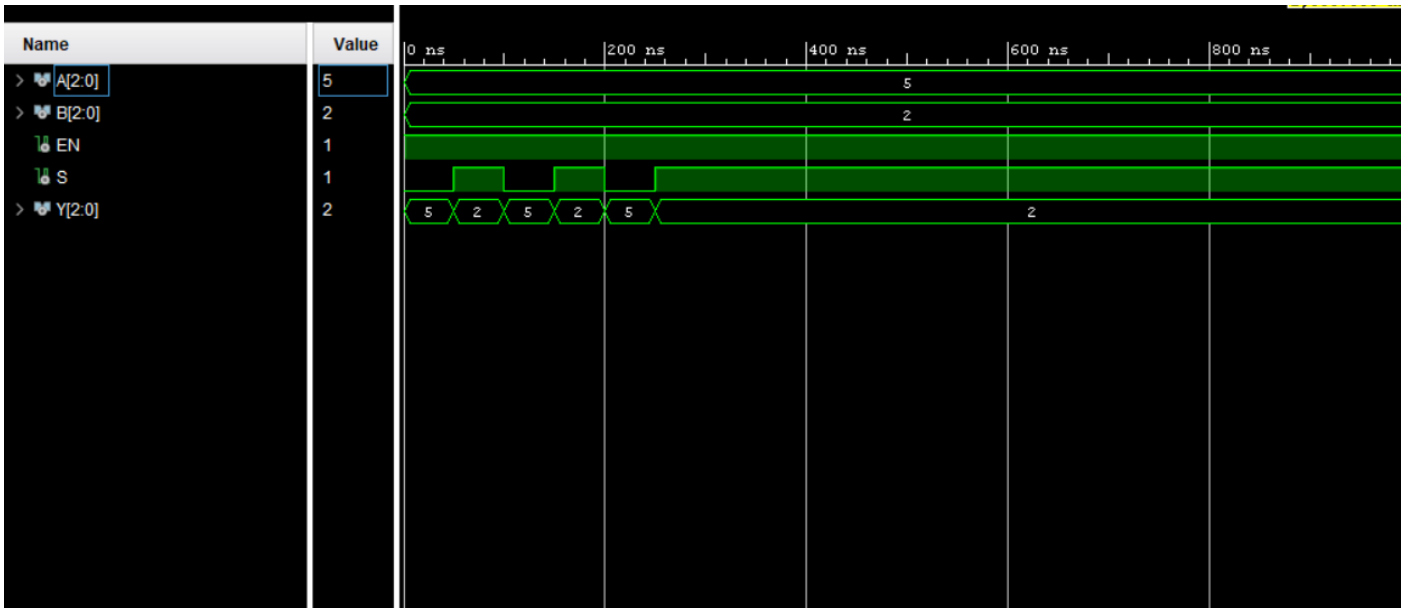
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux2way3bit is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           B : in STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC;
           S : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (2 downto 0));
end mux2way3bit;

architecture Behavioral of mux2way3bit is

begin
Y(0) <= EN and ( (not S and A(0)) OR (S and B(0)));
Y(1) <= EN and ((not S and A(1)) OR (S and B(1)));
Y(2) <= EN and ((not S and A(2)) OR (S and B(2)));

end Behavioral;
```

Simulation Source File

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

entity TB_mux2way3bit is
--  Port ( );
end TB_mux2way3bit;

architecture Behavioral of TB_mux2way3bit is
component mux2way3bit
port ( A : in STD_LOGIC_VECTOR (2 downto 0); -- immediate value
          B : in STD_LOGIC_VECTOR (2 downto 0);
          EN : in STD_LOGIC;
          S : in STD_LOGIC;
          Y : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal A,B : std_logic_vector (2 downto 0);
signal EN , S : std_logic ;
signal Y : std_logic_vector (2 downto 0);

begin
UUT: mux2way3bit
```

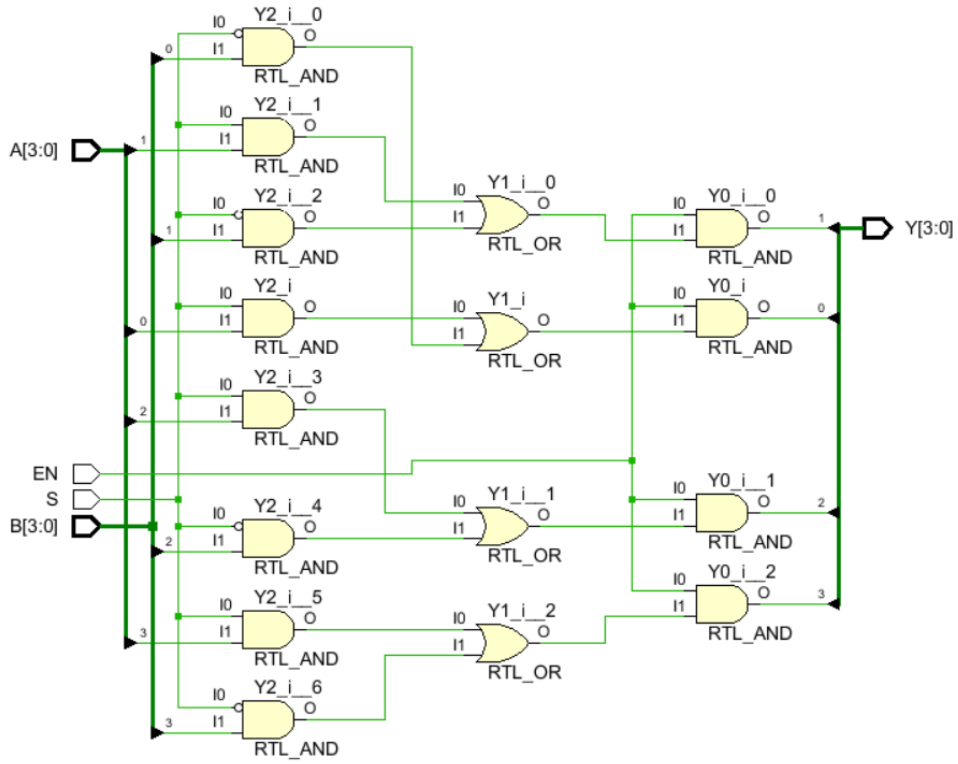
```
port map (
A => A,
B => B,
EN => EN,
S => S,
Y => Y);

process begin
EN <= '1';
A <= "101";
B <= "010";

S <= '0';
wait for 50ns;
S <= '1';
wait for 50ns;
S <= '0';
wait for 50ns;
S <= '1';
wait for 50ns;
S <= '0';
wait for 50ns;
S <= '1';
wait;
end process;
end Behavioral;
```

Timing Diagra



8. 2-way 4-bit Multiplexer

Schematic



Design Source File

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

entity mux2way4bit is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC;
           S : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end mux2way4bit;

architecture Behavioral of mux2way4bit is

begin
Y(0) <= EN and ( ( S and A(0)) OR (not S and B(0)));
Y(1) <= EN and (( S and A(1)) OR (not S and B(1)));
Y(2) <= EN and (( S and A(2)) OR (not S and B(2)));
Y(3) <= EN and (( S and A(3)) OR (not S and B(3)));
```

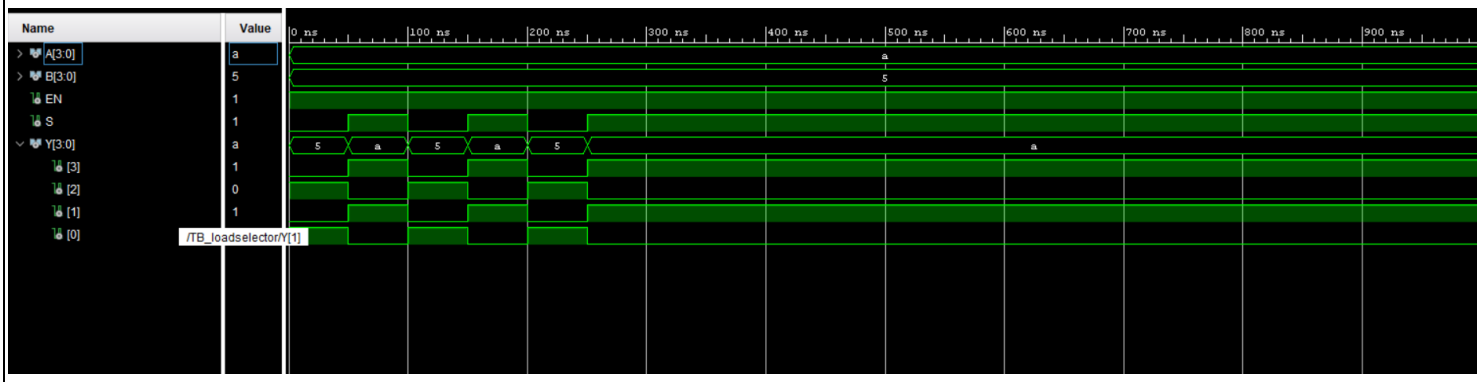
```
end Behavioral;
```

Simulation Source File

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

entity TB_mux2way4bit is
--  Port ( );
end TB_mux2way4bit;

architecture Behavioral of TB_mux2way4bit is
component mux2way4bit
port ( A : in STD_LOGIC_VECTOR (3 downto 0); -- immediate value
          B : in STD_LOGIC_VECTOR (3 downto 0);
          EN : in STD_LOGIC;
          S : in STD_LOGIC;
          Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;


signal A,B : std_logic_vector (3 downto 0);
signal EN , S : std_logic ;
signal Y : std_logic_vector (3 downto 0);

begin
UUT: mux2way4bit
port map (
A => A,
B => B,
EN => EN,
S => S,
Y => Y);

process begin
EN <= '1';
A <= "1010";
B <= "0101";


S <= '0';
wait for 50ns;
S <= '1';
```

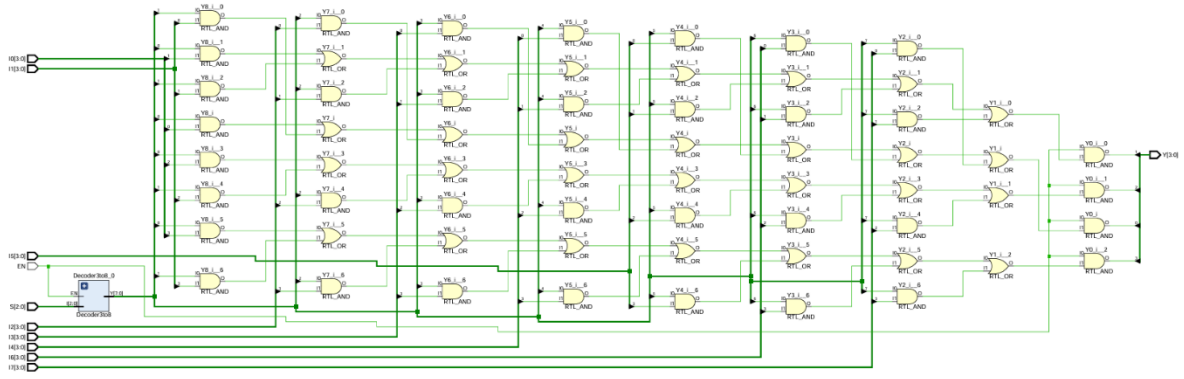
```
wait for 50ns;
S <= '0';
wait for 50ns;
S <= '1';
wait for 50ns;
S <= '0';
wait for 50ns;
S <= '1';
wait;
end process;
end Behavioral;
```

Timing Diagram



9. 8-way 4-bit Multiplexer

Schematic



Design Source File

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

entity mux8way4bit is
    Port ( I0 : in STD_LOGIC_VECTOR (3 downto 0); -- input 4 bit values I0 to I7
           I1 : in STD_LOGIC_VECTOR (3 downto 0);
           I2 : in STD_LOGIC_VECTOR (3 downto 0);
           I3 : in STD_LOGIC_VECTOR (3 downto 0);
           I4 : in STD_LOGIC_VECTOR (3 downto 0);
           I5 : in STD_LOGIC_VECTOR (3 downto 0);
           I6 : in STD_LOGIC_VECTOR (3 downto 0);
           I7 : in STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC;
           S : in STD_LOGIC_VECTOR (2 downto 0); --3 bit value selector
           Y : out STD_LOGIC_VECTOR (3 downto 0)); --selected 4 bit value
end mux8way4bit;

architecture Behavioral of mux8way4bit is
component Decoder_3_to_8
port(EN: in std_logic;
I : in std_logic_vector;
Y : out std_logic_vector);
end component;

signal sel : std_logic_vector (7 downto 0);
begin
Decoder3to8_0 : Decoder_3_to_8
port map(
EN => EN,
I => S,
Y => sel);

Y(0) <= EN and ( (sel(0) and I0(0)) OR (sel(1) and I1(0)) OR (sel(2) and I2(0))
OR (sel(3) and I3(0)) OR (sel(4) and I4(0)) OR (sel(5) and I5(0)) OR (sel(6) and
I6(0)) OR (sel(7) and I7(0)));
Y(1) <= EN and ( (sel(0) and I0(1)) OR (sel(1) and I1(1)) OR (sel(2) and I2(1))
OR (sel(3) and I3(1)) OR (sel(4) and I4(1)) OR (sel(5) and I5(1)) OR (sel(6) and
I6(1)) OR (sel(7) and I7(1)));
Y(2) <= EN and ( (sel(0) and I0(2)) OR (sel(1) and I1(2)) OR (sel(2) and I2(2))
OR (sel(3) and I3(2)) OR (sel(4) and I4(2)) OR (sel(5) and I5(2)) OR (sel(6) and
I6(2)) OR (sel(7) and I7(2)));
```

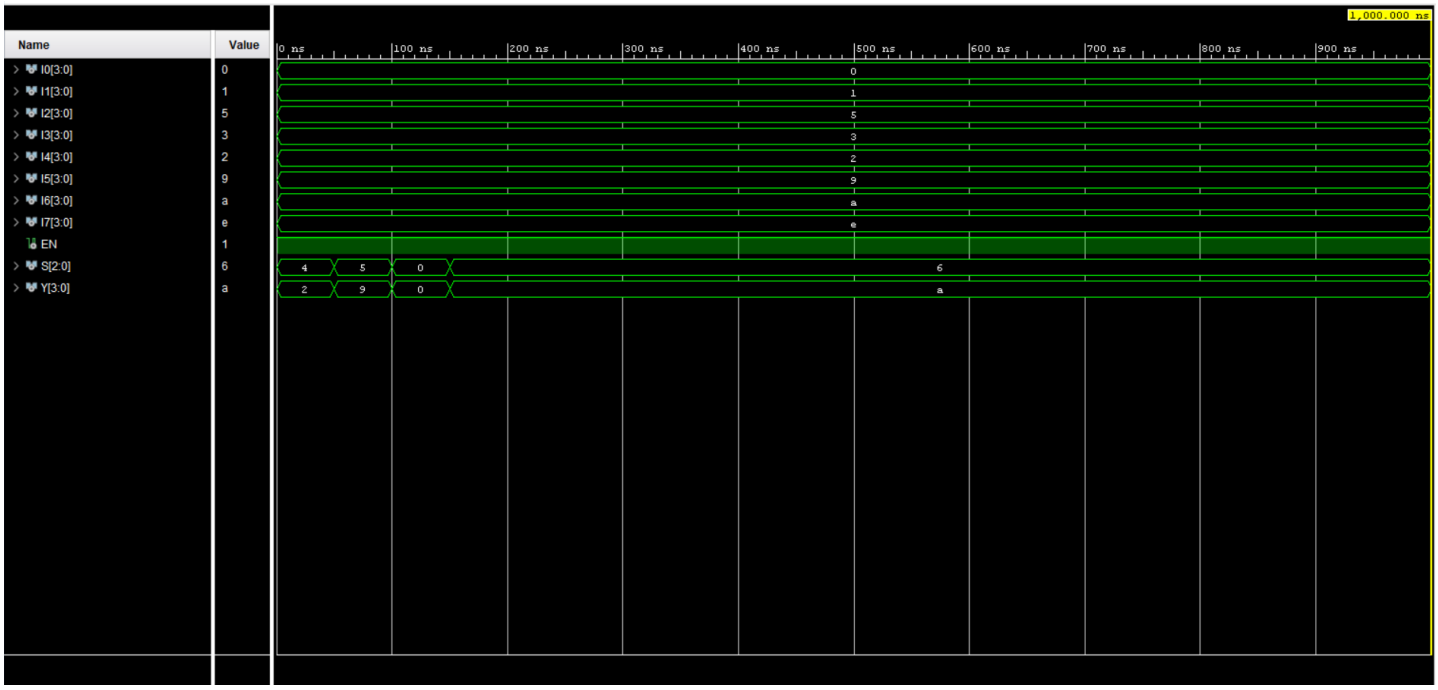
```
Y(3) <= EN and ( (sel(0) and I0(3)) OR (sel(1) and I1(3)) OR (sel(2) and I2(3))
OR (sel(3) and I3(3)) OR (sel(4) and I4(3)) OR (sel(5) and I5(3)) OR (sel(6) and
I6(3)) OR (sel(7) and I7(3)));

end Behavioral;
```

Simulation Source File

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

entity TB2W is
--  Port ( );
end TB2W;

architecture Behavioral of TB2W is
component mux8way4bit
port(
I0 : in STD_LOGIC_VECTOR;
           I1 : in STD_LOGIC_VECTOR (3 downto 0);
           I2 : in STD_LOGIC_VECTOR (3 downto 0);
           I3 : in STD_LOGIC_VECTOR (3 downto 0);
           I4 : in STD_LOGIC_VECTOR (3 downto 0);
           I5 : in STD_LOGIC_VECTOR (3 downto 0);
           I6 : in STD_LOGIC_VECTOR (3 downto 0);
           I7 : in STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC;
           S : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal I0,I1,I2,I3,I4,I5,I6,I7 : std_logic_vector (3 downto 0);
signal EN : STD_LOGIC;
signal S : std_logic_vector (2 downto 0);
signal Y : STD_LOGIC_VECTOR (3 downto 0);



begin
UUT : mux8way4bit
port map (
I1 => I1,
I2 => I2,
I3 => I3,
I4 => I4,
I5 => I5,
```

```
I6 => I6,
I7 => I7,
I0 => I0,
EN => EN,
S => S,
Y => Y);

process
begin
EN <= '1';
I0 <= "0000";
I1 <= "0001";
I2 <= "0101";
I3 <= "0011";
I4 <= "0010";
I5 <= "1001";
I6 <= "1010";
I7 <= "1110";
S <= "100";
wait for 50ns;
S <= "101";
wait for 50ns;
S <= "000";
wait for 50ns;
S <= "110";
wait;
end process;
end Behavioral;
```

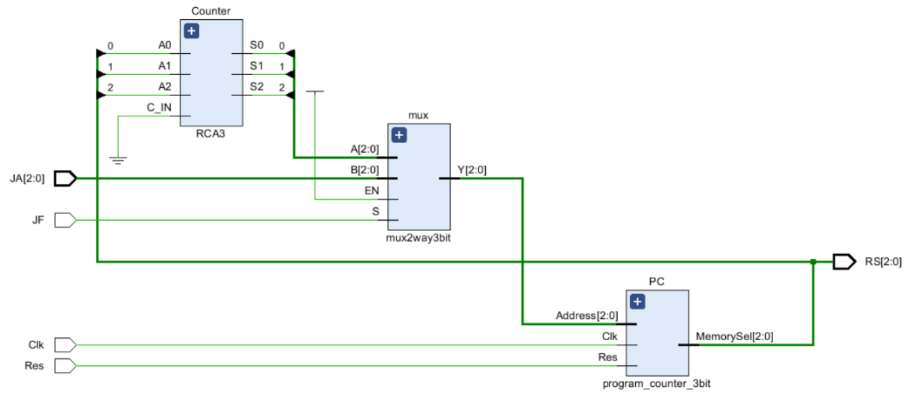
Timing Diagram



10. ROM Incrementor

It is a combinational circuit made up of 3-bit counter, 2-way 3-bit multiplexer and program counter . It will take Clock_signal , Reset , Jump_flag and Jump_Address as inputs and it will give 3-bit binary code as output for corresponding inputs . It works like a counter and its output indicates the ROM address that executed in next clock cycle. If the value of Jump_flag turns to '1' then It will loads Jump_Address to its output. otherwise, it works like a counter (0 -> 7 -> 0).

Schematic



Design Source File

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

entity Rom_Incrementor is
    Port ( Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           JumpFlag : in STD_LOGIC;
           JumpAddress : in STD_LOGIC_VECTOR (2 downto 0);
           SelectedRom : out STD_LOGIC_VECTOR (2 downto 0));
end Rom_Incrementor;

architecture Behavioral of Rom_Incrementor is
component program_counter_3bit
port (Address : in STD_LOGIC_VECTOR (2 downto 0);
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           MemorySel : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component RCA3
    Port ( A0 : in STD_LOGIC;
       A1 : in STD_LOGIC;
       A2 : in STD_LOGIC;
       S0 : out STD_LOGIC;
       S1 : out STD_LOGIC;
       S2 : out STD_LOGIC;
       C_IN : in STD_LOGIC;
       C_OUT : out STD_LOGIC);

end component;
component mux2way3bit
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
       B : in STD_LOGIC_VECTOR (2 downto 0);
       EN : in STD_LOGIC;
       S : in STD_LOGIC;
       Y : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal muxout,pcout,addout : std_logic_vector(2 downto 0);
begin

PC:program_counter_3bit
port map(
```

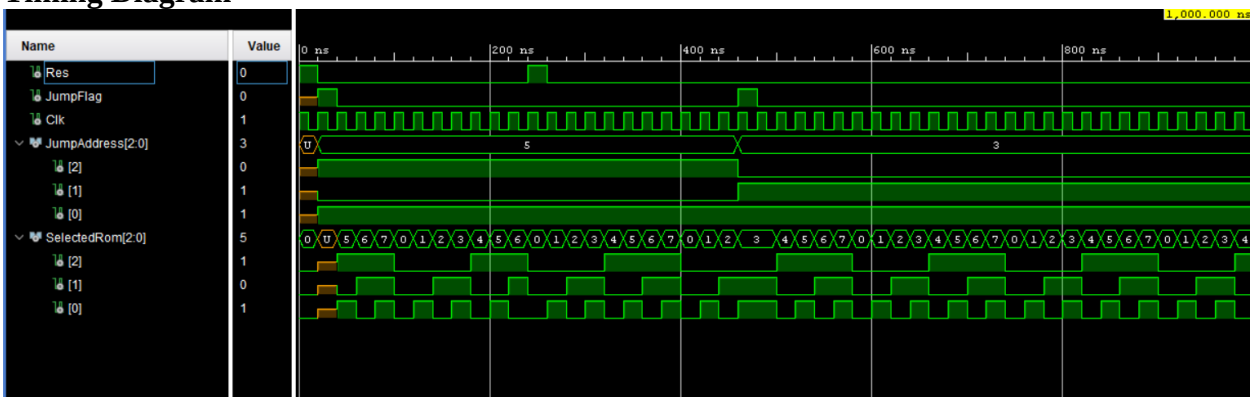
```
Address => muxout,
Res => Res,
Clk => Clk,
MemorySel => pcout);

SelectedRom <= pcout;

Counter: RCA3
port map(
A0 => pcout(0),
A1 => pcout(1),
A2 => pcout(2),
S0 => addout(0),
S1 => addout(1),
S2 => addout(2),
C_IN => '0');

mux: mux2way3bit
port map (
A => addout,
B => JumpAddress,
EN => '1',
S => JumpFlag,
Y => muxout);

end Behavioral;
```

Simulation Source File

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

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

architecture Behavioral of TB_PI is
component Rom_Incrementor
    Port ( Res : in STD_LOGIC;
       Clk : in STD_LOGIC;
       JumpFlag : in STD_LOGIC;
       JumpAddress : in STD_LOGIC_VECTOR (2 downto 0);
       SelectedRom : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal Res,JumpFlag : std_logic;
```

```
signal Clk : std_logic:='0';
signal JumpAddress,SelectedRom:std_logic_vector (2 downto 0);

begin
UUT: Rom_Incrementor
port map (
Res => Res,
Clk => Clk,
Jumpflag => Jumpflag,
JumpAddress => JumpAddress,
SelectedRom => SelectedRom);

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

process begin
Res <= '1';
wait for 20ns;
Res <= '0';
JumpAddress <= "101";
Jumpflag <= '1';
wait for 20 ns;
Jumpflag <= '0';
wait for 200ns;
Res <= '1';
wait for 20ns;
Res <= '0';
Jumpflag <= '0';
wait for 200 ns;
JumpAddress <= "011";
Jumpflag <= '1';
wait for 20 ns;
Jumpflag <= '0';
wait;

end process;

end Behavioral;
```

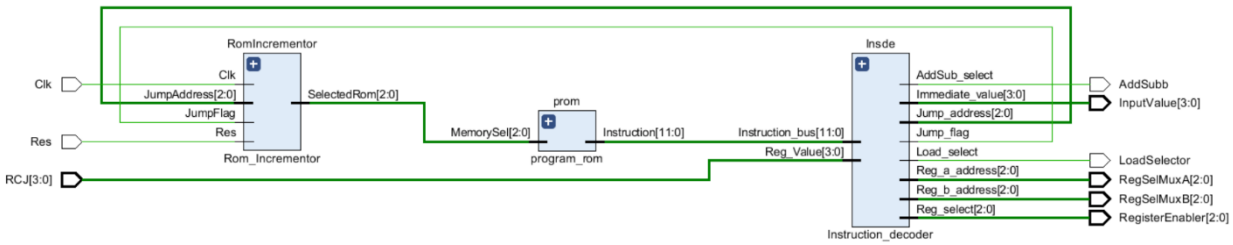
Timing Diagram



11. Program Decoder

It is made with RomIncrementor, ProgramRom, and Instruction_decoder. it takes clock signal , reset command, and a 4-bit value (Register check for jump that comes from 8-way 4-bit mux that connected with add/sub unit) as inputs. it will give output as same as the output that from the instruction decoder except jump address and jump flag (these values are used by itself). The function of this circuit is it will automatically select the instruction from program Rom that execute in the next clock cycle and decode that instruction and send it as output. simply said Program decoder Encapsulates the Romincrementer, program rom, instruction decoder

Schematic



Design Source File

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

entity program_decoder is
    Port ( Clk : in STD_LOGIC;
           Res : in STD_LOGIC;
           RCJ : in STD_LOGIC_VECTOR (3 downto 0);
           RegisterEnabler : out STD_LOGIC_VECTOR (2 downto 0);
           LoadSelector : out STD_LOGIC;
           InputValue : out STD_LOGIC_VECTOR (3 downto 0);
```

```
            RegSelMuxA : out STD_LOGIC_VECTOR (2 downto 0);
            RegSelMuxB : out STD_LOGIC_VECTOR (2 downto 0);
            AddSubb : out STD_LOGIC);
end program_decoder;

architecture Behavioral of program_decoder is
component Rom_Incrementor
    Port ( Res : in STD_LOGIC;
       Clk : in STD_LOGIC;
       JumpFlag : in STD_LOGIC;
       JumpAddress : in STD_LOGIC_VECTOR (2 downto 0);
       SelectedRom : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component program_rom
    Port ( MemorySel : in STD_LOGIC_VECTOR (2 downto 0);
       Instruction : out STD_LOGIC_VECTOR (11 downto 0));

end component;

component Instruction_decoder
    Port ( Instruction_bus : in STD_LOGIC_VECTOR (11 downto 0); -- input from
program rom
       Reg_Value : in STD_LOGIC_VECTOR (3 downto 0); -- to check if a register
has value zero stored in it
       -- reg which we check is given with JZR instruction
       Reg_a_address : out STD_LOGIC_VECTOR (2 downto 0); -- selecting input for
AU via 8 way 2 bit mux
       Reg_b_address : out STD_LOGIC_VECTOR (2 downto 0); -- selecting input for
AU via 8 way 2 bit mux
       Reg_select : out STD_LOGIC_VECTOR (2 downto 0); -- to select Reg from reg
bank to write to a reg
       AddSub_select : out STD_LOGIC; -- selecting Add or Subtract (if neg
instruction sub will be selceted)

       Jump_flag : out STD_LOGIC; -- to  jump to another instruction in
PC(Program counter),
       -- by checking Reg_Value and this output connected to 2 way 3 bit MUX
selector (PC area)
       Jump_address : out STD_LOGIC_VECTOR (2 downto 0); -- which instruction to
execute, this output connected to 2 way 3 bit MUX

       Load_select : out STD_LOGIC; -- select between AU output and immediate
value (connected to 2 way 4 bit MUX)
       Immediate_value : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end component;

signal JumpFlag:std_logic;
signal JumpAddress,SelectedRom:std_logic_vector (2 downto 0);
signal Ins:std_logic_vector (11 downto 0);

begin
RomIncrementor:Rom_Incrementor
port map(
Res => Res,
Clk => Clk,
JumpFlag => JumpFlag,
JumpAddress => JumpAddress,
SelectedRom => SelectedRom);

prom:program_rom
port map(
MemorySel => SelectedRom,
Instruction => Ins);


Insde:Instruction_decoder
port map(
Instruction_bus => Ins, -- input from program rom
       Reg_Value => RCJ, -- to check if a register has value zero stored in it
       -- reg which we check is given with JZR instruction
       Reg_a_address => RegSelMuxA, -- selecting input for AU via 8 way 2 bit mux
       Reg_b_address => RegSelMuxB, -- selecting input for AU via 8 way 2 bit mux
       Reg_select => RegisterEnabler, -- to select Reg from reg bank to write to
a reg
       AddSub_select => AddSubb, -- selecting Add or Subtract (if neg instruction
sub will be selceted)

       Jump_flag => JumpFlag, -- to  jump to another instruction in PC(Program
counter),
       -- by checking Reg_Value and this output connected to 2 way 3 bit MUX
selector (PC area)
       Jump_address => JumpAddress, -- which instruction to execute, this output
connected to 2 way 3 bit MUX

       Load_select => LoadSelector,-- select between AU output and immediate
value (connected to 2 way 4 bit MUX)
       Immediate_value => InputValue);
```

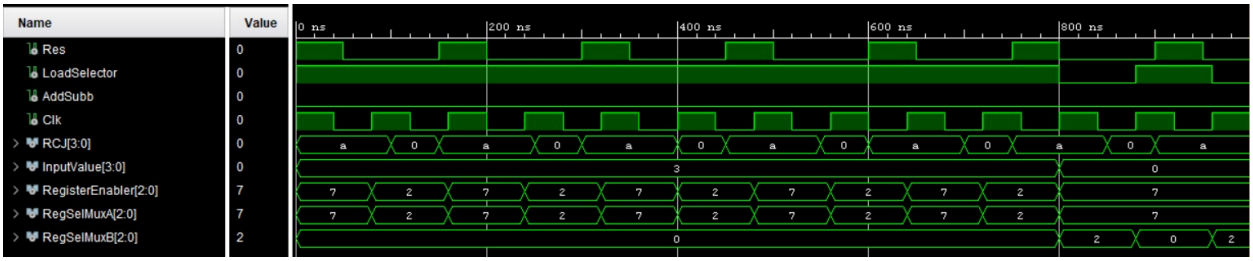
```
end Behavioral;
```

Simulation Source File

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

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

architecture Behavioral of prodecodetb is
component program_decoder
port(Clk : in STD_LOGIC;
           Res : in STD_LOGIC;
           RCJ : in STD_LOGIC_VECTOR (3 downto 0);
           RegisterEnabler : out STD_LOGIC_VECTOR (2 downto 0);
           LoadSelector : out STD_LOGIC;
           InputValue : out STD_LOGIC_VECTOR (3 downto 0);
           RegSelMuxA : out STD_LOGIC_VECTOR (2 downto 0);
           RegSelMuxB : out STD_LOGIC_VECTOR (2 downto 0);
           AddSubb : out STD_LOGIC);
end component;
signal Res,LoadSelector,AddSubb:std_logic;
signal Clk : std_logic:='0';
signal RCJ,InputValue : std_logic_vector (3 downto 0);
signal RegisterEnabler,RegSelMuxA,RegSelMuxB : std_logic_vector (2 downto 0);

begin
UUT : program_decoder
port map(
```

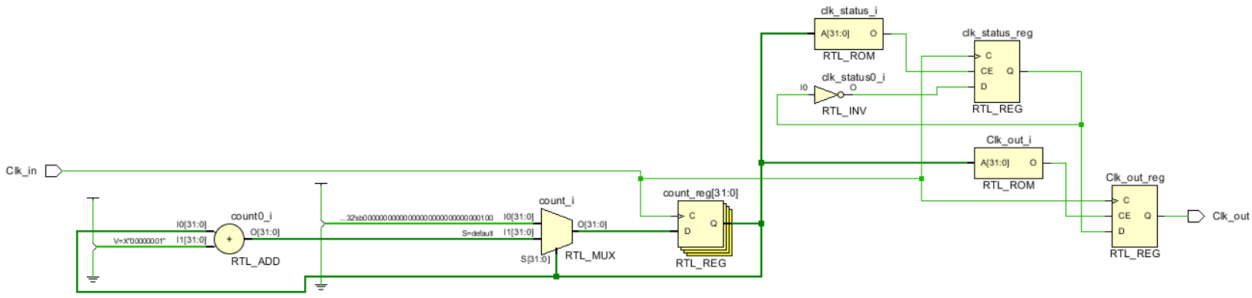
```
Clk => Clk,
Res => Res,
RCJ => RCJ,
RegisterEnabler => RegisterEnabler,
LoadSelector => LoadSelector,
InputValue => InputValue,
RegSelMuxA => RegSelMuxA,
RegSelMuxB => RegSelMuxB,
AddSubb => AddSubb);

process begin
Clk <= not (Clk);
wait for 40 ns;
end process;

process begin
Res <= '1';
RCJ <= "1010";
wait for 50 ns;
Res <= '0';
wait for 50 ns;
RCJ <= "0000";
wait for 50 ns;
end process;

end Behavioral;
```

Timing Diagram



12. Slow Clock

Schematic

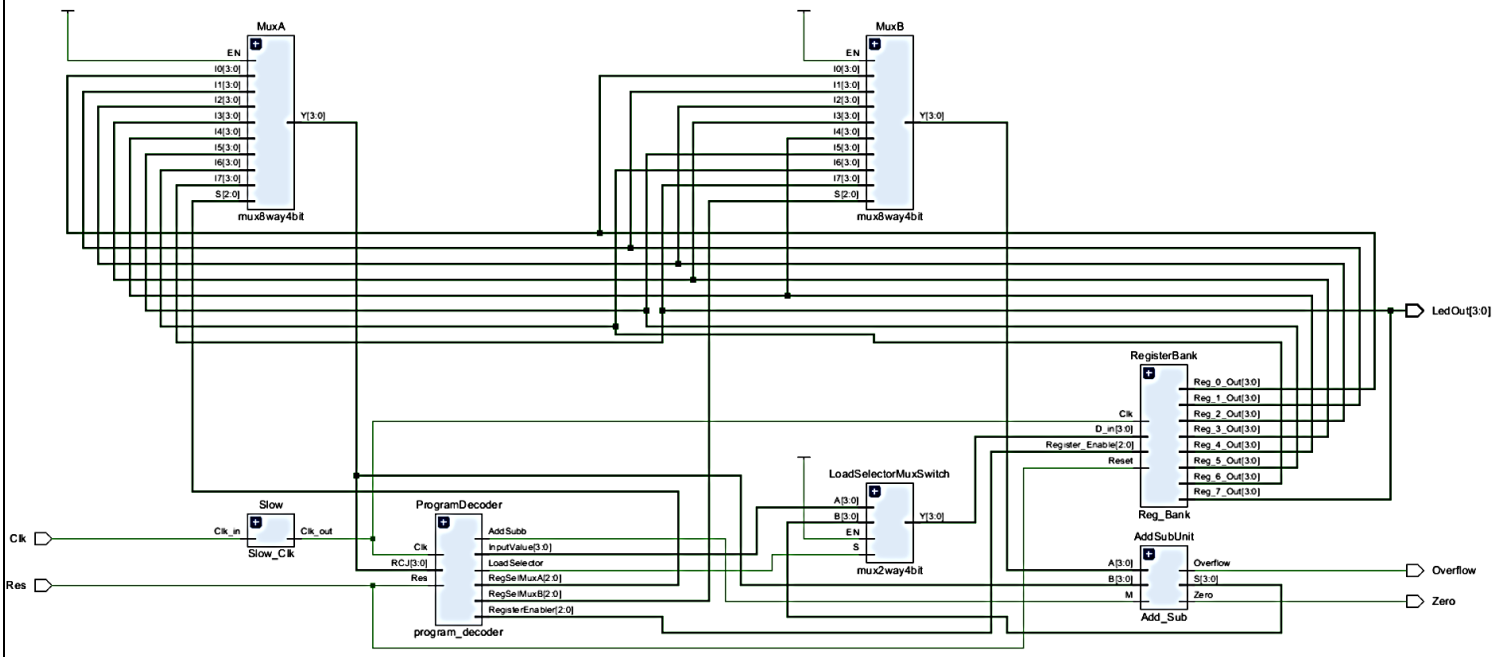


Design Source File

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Slow_Clk is
     Port ( Clk_in : in STD_LOGIC;
     Clk_out : out STD_LOGIC);
end Slow_Clk;

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

begin

process(Clk_in) begin
 if (rising_edge(Clk_in)) then
 count<=count+1; --Increment counter
     if (count=4) then --count 50M pluses(1/2 of period)
```

```
        clk_status<=NOT (clk_status); --invert clock status
        Clk_out<=clk_status;
        count<=1; --Reset counter
        end if;
    end if;
end process;
end Behavioral;
```

13. Nano Processor



Schematic

Design Source File

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

entity NanoProcessor is
    Port ( Res : in std_logic;
           LedOut : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC;
           Clk : in STD_LOGIC);
```

```
end NanoProcessor;

architecture Behavioral of Nanoprocessor is
component program_decoder
    Port ( Clk : in STD_LOGIC;
           Res : in STD_LOGIC;
           RCJ : in STD_LOGIC_VECTOR (3 downto 0);
           RegisterEnabler : out STD_LOGIC_VECTOR (2 downto 0);
           LoadSelector : out STD_LOGIC;
           InputValue : out STD_LOGIC_VECTOR (3 downto 0);
           RegSelMuxA : out STD_LOGIC_VECTOR (2 downto 0);
           RegSelMuxB : out STD_LOGIC_VECTOR (2 downto 0);
           AddSubb : out STD_LOGIC);
end component;
component Reg_Bank
    Port ( Register_Enable : in STD_LOGIC_VECTOR (2 downto 0);
       D_in : in STD_LOGIC_VECTOR (3 downto 0);
       Clk : in STD_LOGIC;
       Reset : in STD_LOGIC;
       Reg_0_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg_1_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg_2_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg_3_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg_4_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg_5_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg_6_Out : out STD_LOGIC_VECTOR (3 downto 0);
       Reg_7_Out : out STD_LOGIC_VECTOR (3 downto 0));

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

component mux8way4bit
    Port ( I0 : in STD_LOGIC_VECTOR (3 downto 0);
       I1 : in STD_LOGIC_VECTOR (3 downto 0);
       I2 : in STD_LOGIC_VECTOR (3 downto 0);
       I3 : in STD_LOGIC_VECTOR (3 downto 0);
       I4 : in STD_LOGIC_VECTOR (3 downto 0);
       I5 : in STD_LOGIC_VECTOR (3 downto 0);
       I6 : in STD_LOGIC_VECTOR (3 downto 0);
       I7 : in STD_LOGIC_VECTOR (3 downto 0);
       EN : in STD_LOGIC;
       S : in STD_LOGIC_VECTOR (2 downto 0);
```

```
        Y : out STD_LOGIC_VECTOR (3 downto 0));

end component;
component Add_Sub
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
       B : in STD_LOGIC_VECTOR (3 downto 0);
       M : in STD_LOGIC;
       S : out STD_LOGIC_VECTOR (3 downto 0);
       Overflow : out STD_LOGIC;
       Zero : out STD_LOGIC);

end component;
component mux2way4bit
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
       B : in STD_LOGIC_VECTOR (3 downto 0);
       EN : in STD_LOGIC;
       S : in STD_LOGIC;
       Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal calout,immivalue,corrvalue : std_logic_vector (3 downto 0);
signal RIA,RIB,RE,RSS : std_logic_vector (2 downto 0);
signal loadsel,AS,Clki : std_logic;
signal R0,R1,R2,R3,R4,R5,R6,R7,NA,NB : std_logic_vector (3 downto 0);
begin
Slow:Slow_Clk
port map(
Clk_in => Clk,
Clk_out => Clki);

ProgramDecoder:program_decoder
port map(
Clk => Clki,
Res => Res,
RCJ => NA,
RegisterEnabler => RE,
LoadSelector => loadsel,
InputValue => immivalue,
RegSelMuxA => RIA,
RegSelMuxB => RIB,
AddSubb => AS);

LoadSelectorMuxSwitch : mux2way4bit
port map(
B => calout,
A => immivalue,
```

```
EN => '1',
S => loadsel,
Y => corrvalue);

RegisterBank : Reg_Bank
port map(
D_in => corrvalue,
Clk => Clki,
Reset => Res,
Reg_0_Out => R0,
Reg_1_Out => R1,
Reg_2_Out => R2,
Reg_3_Out => R3,
Reg_4_Out => R4,
Reg_5_Out => R5,
Reg_6_Out => R6,
Reg_7_Out => R7,
Register_Enable => RE);

MuxA: mux8way4bit
port map(
I0 => R0,
I1 => R1,
I2 => R2,
I3 => R3,
I4 => R4,
I5 => R5,
I6 => R6,
I7 => R7,
EN => '1',
S => RIA,
Y => NA);

MuxB: mux8way4bit
port map(
I0 => R0,
I1 => R1,
I2 => R2,
I3 => R3,
I4 => R4,
I5 => R5,
I6 => R6,
I7 => R7,
EN => '1',
S => RIB,
```

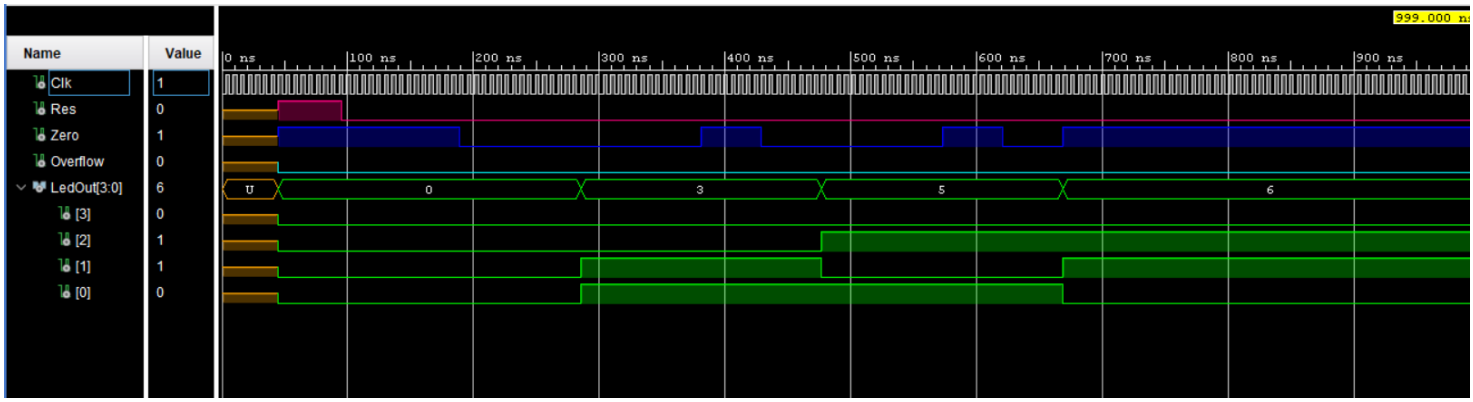
```
Y => NB);
AddSubUnit : Add_Sub
port map(
A => NB, -- value from Mux B
B => NA, -- value from Mux A
M => AS,
S => calout,
Overflow => Overflow,
Zero => Zero);
LedOut <= R7
end Behavioral;
```

Simulation Source File

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

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

architecture Behavioral of NanoProcessor_TB is
component NanoProcessor
    Port ( Res : in std_logic;
      LedOut : out STD_LOGIC_VECTOR (3 downto 0);
       Zero : out STD_LOGIC;
       Overflow : out STD_LOGIC;
       Clk : in STD_LOGIC);
end component;
signal Clk : std_logic:='0';
signal Res,Zero,Overflow : std_logic;
signal LedOut: std_logic_vector(3 downto 0);


begin
UUT : NanoProcessor
port map(
Res => Res,
LedOut => LedOut,
Zero => Zero,
Overflow => Overflow,
Clk => Clk);

process begin
wait for 3 ns;
Clk <= not (Clk);
```

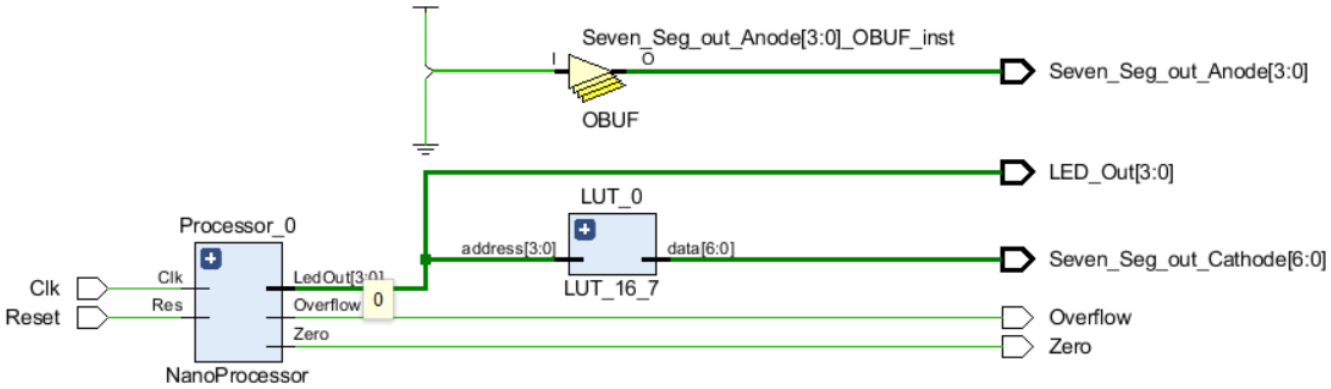
```
end process;

process begin
wait for 45ns;
Res <= '1';
wait for 50ns;
Res <= '0';
wait;
end process;
end Behavioral;
```

Timing Diagram



14. Nano Processor with 7-Segment Display

Schematic



Design Source File

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

entity NanoProcessor_7Seg is
    Port ( Reset : in std_logic;
           Clk : in STD_LOGIC;
           LED_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC;
           Seven_Seg_out_Anode : out STD_LOGIC_VECTOR (3 downto 0);
           Seven_Seg_out_Cathode : out STD_LOGIC_VECTOR (6 downto 0)
       );
end NanoProcessor_7Seg;

architecture Behavioral of NanoProcessor_7Seg is
component LUT_16_7
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end component;

component NanoProcessor
    Port ( Res : in std_logic;
           LedOut : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC;
           Clk : in STD_LOGIC);
end component;

signal OutSig : std_logic_vector (3 downto 0);
signal To7Seg : std_logic_vector (6 downto 0);

begin

Processor_0 : NanoProcessor
    port map (
        Res => Reset,
        Clk => Clk,
        LedOut => OutSig,
        Overflow => Overflow,
        Zero => Zero);

LUT_0 : LUT_16_7
    port map (
        Address => OutSig,
```

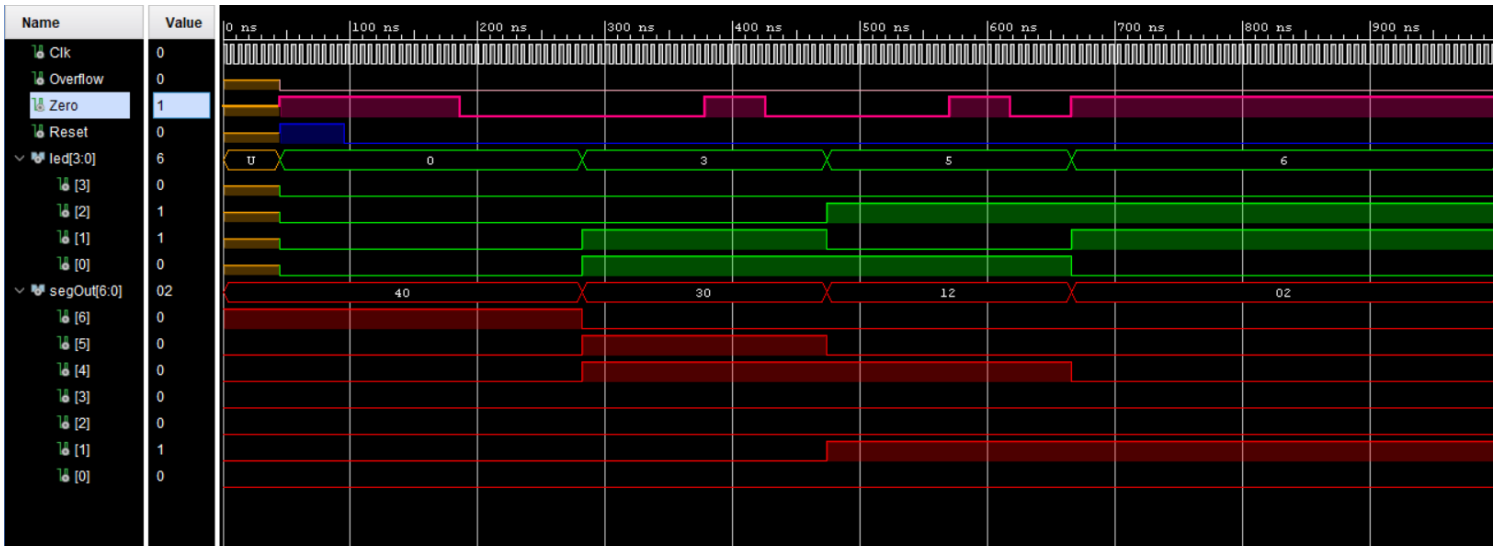
```
        Data => To7Seg);
LED_Out <= OutSig;

Seven_Seg_out_Cathode <= To7Seg;
Seven_Seg_out_Anode <= "1110";
end Behavioral
```

Simulation Source File

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

entity TB_NP_7Seg is
--  Port ( );
end TB_NP_7Seg;

architecture Behavioral of TB_NP_7Seg is
component NanoProcessor_7Seg
    Port ( Reset : in std_logic;
           Clk : in STD_LOGIC;
           LED_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC;
           Seven_Seg_out_Anode : out STD_LOGIC_VECTOR (3 downto 0);
           Seven_Seg_out_Cathode : out STD_LOGIC_VECTOR (6 downto 0)
       );
end component;

signal Clk : std_logic := '0';
signal Overflow, Zero, Reset : std_logic;
signal led : std_logic_vector (3 downto 0);
signal segOut : std_logic_vector (6 downto 0);

begin
UUT : NanoProcessor_7Seg
    port map (
        Reset => Reset,
        Clk => Clk,
        Zero => Zero,
        Overflow => Overflow,
        LED_Out => led,
        Seven_Seg_out_Cathode => segOut);

process
begin
```

```
    Clk <= not(Clk);
    wait for 3 ns;
end process;

process
begin
    wait for 45 ns;
    Reset <= '1';
    wait for 50 ns;
    Reset <= '0';
    wait;
end process;
end Behavioral;
```

Timing Diagram



Resource Utilization

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (106)	BUFGCTRL (32)
NanoProcessor_7Seg	39	50	25	39	6	19	1
Processor_0 (NanoPr...	39	50	25	39	6	0	0
ProgramDecoder (p...	21	3	8	21	2	0	0
RegisterBank (Reg...	7	13	6	7	0	0	0
Slow (Slow_Clk)	11	34	16	11	3	0	0

Resource	Utilization	Available	Utilization %
LUT	39	20800	0.19
FF	50	41600	0.12
IO	19	106	17.92

Constraint File

```
# 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_Out[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED_Out[0]}]
set_property PACKAGE_PIN E19 [get_ports {LED_Out[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED_Out[1]}]
set_property PACKAGE_PIN U19 [get_ports {LED_Out[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED_Out[2]}]
set_property PACKAGE_PIN V19 [get_ports {LED_Out[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED_Out[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_Seg_out_Cathode[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_out_Cathode[0]}]
set_property PACKAGE_PIN W6 [get_ports
{Seven_Seg_out_Cathode[1]}]
```

```
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_out_Cathode[1]}]
set_property PACKAGE_PIN U8 [get_ports
{Seven_Seg_out_Cathode[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_out_Cathode[2]}]
set_property PACKAGE_PIN V8 [get_ports
{Seven_Seg_out_Cathode[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_out_Cathode[3]}]
set_property PACKAGE_PIN U5 [get_ports
{Seven_Seg_out_Cathode[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_out_Cathode[4]}]
set_property PACKAGE_PIN V5 [get_ports
{Seven_Seg_out_Cathode[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_out_Cathode[5]}]
set_property PACKAGE_PIN U7 [get_ports
{Seven_Seg_out_Cathode[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_out_Cathode[6]}]


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

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

Nano Processor V2- With 2-bit Multiplier

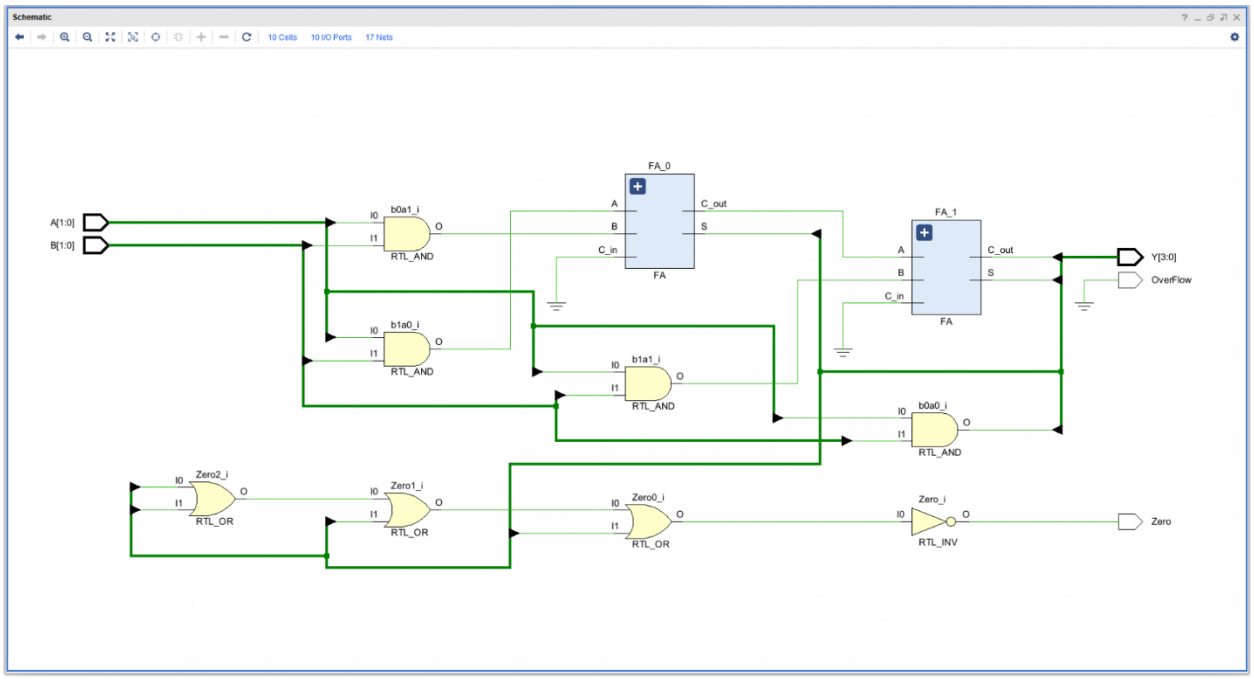
In addition to the components in the nano processor, we have added a 2-bit multiplier circuit.

To implement multiplication, we used “Mul” instruction to it. To support additional instruction, we have increased the instruction size from 12 bits to 13 bits and changed the implementation of Program Rom, Instruction Decoder and Program Decoder accordingly.

Component	Function	Structure
2-bit Multiplier	When two 2-bit numbers are given as input, it will multiply and give output. It will indicate if there is an overflow or if the output is zero	It has two Full adders to carry out the process
Program ROM	In each ROM, 13-bit instructions(assembly program) are stored. The 13 bits are divided into various fields such as opcode and operands.	It has 8 ROMs inside. Memory select selects which program has to be executed and sends the instruction to the instruction decoder.
Instruction Decoder	Decodes instruction fetched from program rom and activates necessary units	It has a 2 to 4 Decoder to decode the first two bits (from the MSB side)

1. 2-bit Multiplier

Schematic



Design Source File

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

entity Multiplier_2 is
    Port ( A : in STD_LOGIC_VECTOR (1 downto 0);
           B : in STD_LOGIC_VECTOR (1 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           OverFlow : out STD_LOGIC);
end Multiplier_2;

architecture Behavioral of Multiplier_2 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 b0a0,b1a0,b0a1,b1a1: STD_LOGIC;
signal c1,s1,c2,s2: STD_LOGIC;

begin
b0a0 <= A(0) AND B(0);
b1a0 <= A(0) AND B(1);
b0a1 <= A(1) AND B(0);
b1a1 <= A(1) AND B(1);
```

```
FA_0 : FA
    port map(
         A => b1a0,
         B => b0a1,
         C_in => '0',
         S => s1,
         C_out => c1 );

FA_1 : FA
    port map(
         A => c1,
         B => b1a1,
         C_in => '0',
         S => s2,
         C_out => c2 );

Y(0) <= b0a0;
Y(1) <= s1;
Y(2) <= s2;
Y(3) <= c2;

Zero <= NOT (b0a0 OR s1 OR s2 Or c2);
OverFlow <= '0'; -- since maximum 3 x 3 result 9 which can be reprecented by 4
bit so no overflow

end Behavioral;
```

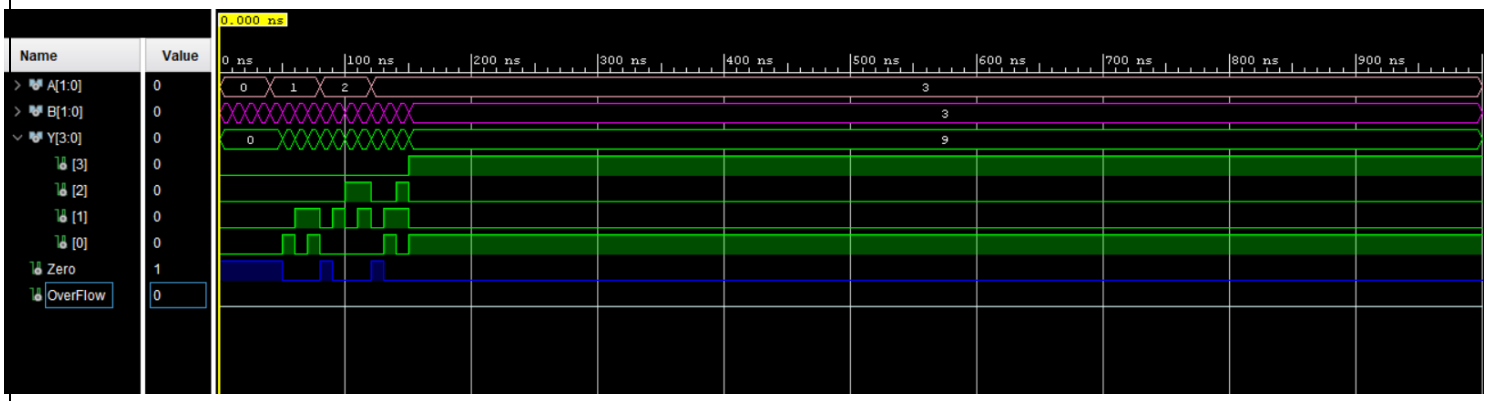
Simulation Source File

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


entity TB_Multiplier_2 is
--  Port ( );
end TB_Multiplier_2;


architecture Behavioral of TB_Multiplier_2 is
component Multiplier_2
    Port ( A : in STD_LOGIC_VECTOR (1 downto 0);
           B : in STD_LOGIC_VECTOR (1 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           OverFlow : out STD_LOGIC);
end component;
signal A,B :STD_LOGIC_VECTOR (1 downto 0);
signal Y :STD_LOGIC_VECTOR (3 downto 0);
signal Zero, OverFlow : STD_LOGIC;

begin
UUT: Multiplier_2
    port map( A => A,
              B =>B,
              Y=> Y,
              Zero => Zero,
              OverFlow => OverFlow);
process
begin
```

```
A <= "00";
B <= "00";
WAIT FOR 10ns;
B <= "01";
WAIT FOR 10ns;
B <= "10";
WAIT FOR 10ns;
B <= "11";
WAIT FOR 10ns;

A <= "01";

B <= "00";
WAIT FOR 10ns;
B <= "01";
WAIT FOR 10ns;
B <= "10";
WAIT FOR 10ns;
B <= "11";
WAIT FOR 10ns;

A <= "10";

B <= "00";
WAIT FOR 10ns;
B <= "01";
WAIT FOR 10ns;
B <= "10";
WAIT FOR 10ns;
```

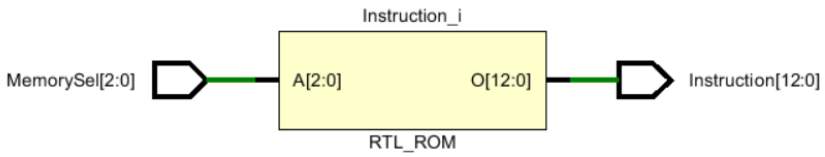
```
        B <= "11";
        WAIT FOR 10ns;

        A <= "11";

        B <= "00";
        WAIT FOR 10ns;
        B <= "01";
        WAIT FOR 10ns;
        B <= "10";
        WAIT FOR 10ns;
        B <= "11";
        WAIT FOR 10ns;
        WAIT;
    end process;
    end Behavioral;
```

Timing Diagram



2. Program ROM

Schematic



Design Source File

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

entity program_rom is
    Port ( MemorySel : in STD_LOGIC_VECTOR (2 downto 0);
           Instruction : out STD_LOGIC_VECTOR (12 downto 0));
end program_rom;

architecture Behavioral of program_rom is
type rom_type is array(0 to 7) of std_logic_vector(12 downto 0);
signal p_rom :rom_type:=(
    "0101110000011", -- MOVI R7,3 --line->0
    "0100100000010", -- MOVI R2,2
    "1001110100000", -- MUL R7,R2
    "0101010000011", -- MOVI R5,3 --line->0
    "0001111010000", -- ADD R7,R5
    "0011010000000", -- NEG R5 --line->0
    "0001111010000", -- ADD R7,R5
    "0110000000111" --  JZR R0,7 --
); -- output R7 : 3 6 9 6 3 0

begin
Instruction <= p_rom(to_integer(unsigned(MemorySel)));
```

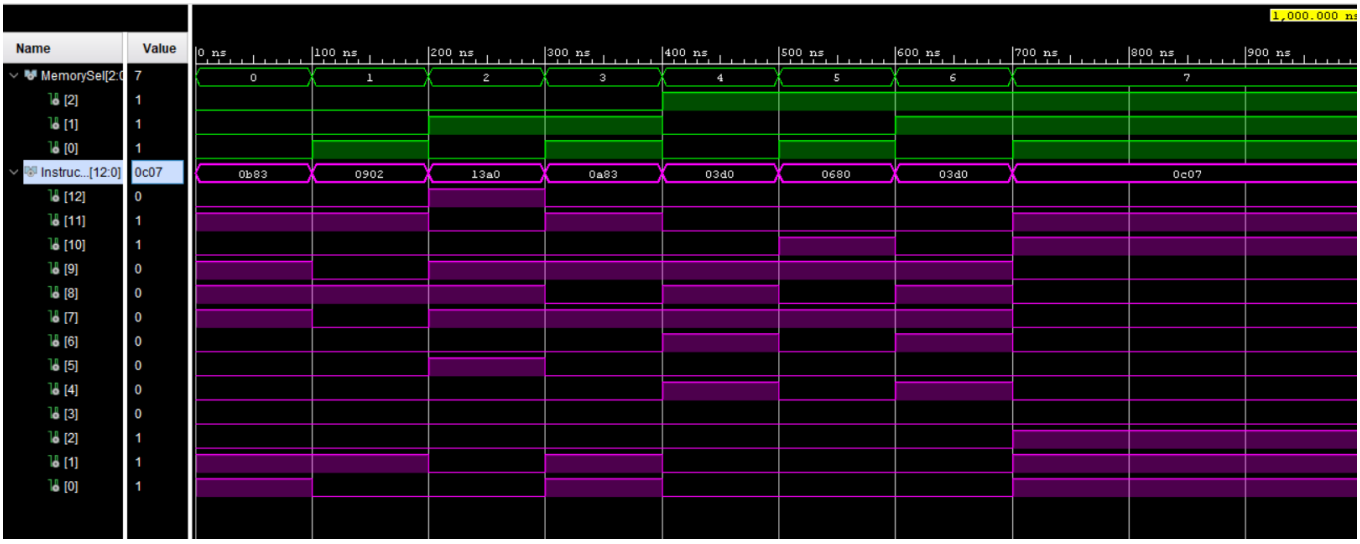
```
end Behavioral;
```

Simulation Source File

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

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

architecture Behavioral of program_rom_tb is
component program_rom
port(
          MemorySel : in STD_LOGIC_VECTOR (2 downto 0);
          Instruction : out STD_LOGIC_VECTOR (12 downto 0));
end component;
signal MemorySel:std_logic_vector(2 downto 0);
signal Instruction:std_logic_vector(12 downto 0);

begin
uut:program_rom port map(
MemorySel=>MemorySel,
Instruction=>Instruction);
process
begin
MemorySel<="000";
wait for 100 ns;
MemorySel<="001";
wait for 100 ns;
MemorySel<="010";
wait for 100 ns;
MemorySel<="011";
wait for 100 ns;
MemorySel<="100";
wait for 100 ns;
MemorySel<="101";
wait for 100 ns;
MemorySel<="110";
wait for 100 ns;
MemorySel<="111";
wait;
end process;
```

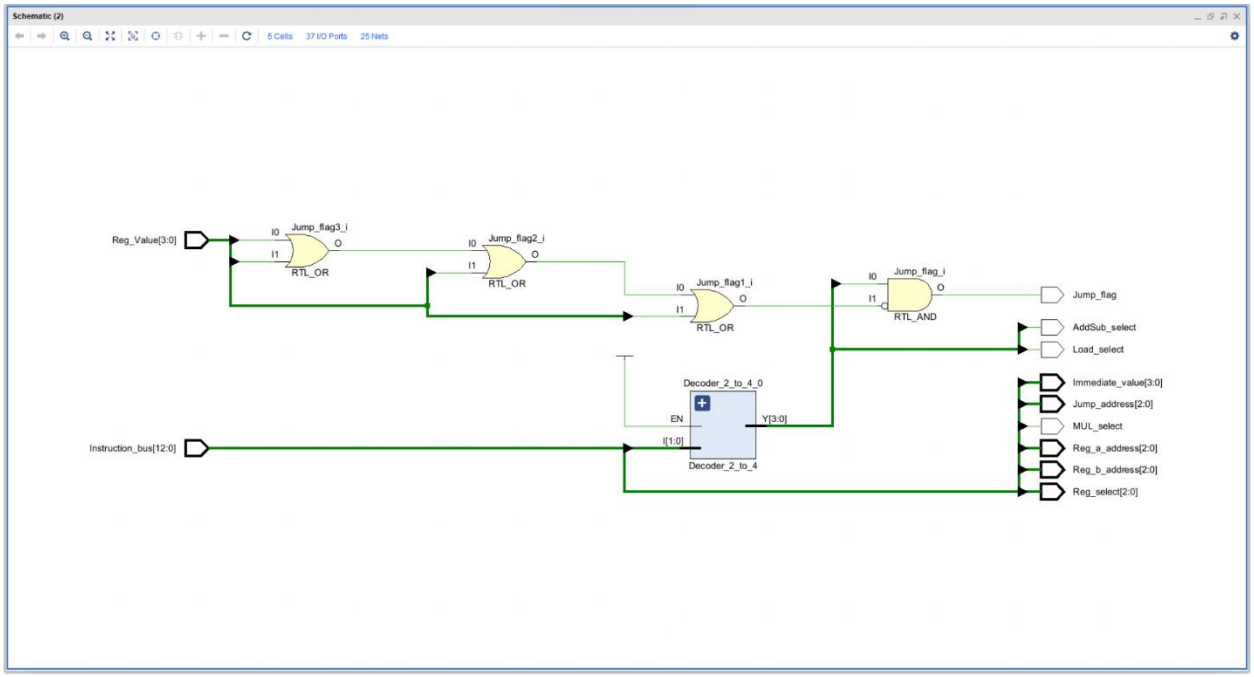
```
end Behavioral;
```

Timing Diagram



3. Instruction Decoder

Schematic



Design Source File

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity Instruction_decoder is

    Port ( Instruction_bus : in STD_LOGIC_VECTOR (12 downto 0); -- input from
program rom

           Reg_Value : in STD_LOGIC_VECTOR (3 downto 0); -- to check if a
register has value zero stored in it

           -- reg which we check is given with JZR instruction

           Reg_a_address : out STD_LOGIC_VECTOR (2 downto 0); -- selecting input
for AU via 8 way 2 bit mux

           Reg_b_address : out STD_LOGIC_VECTOR (2 downto 0); -- selecting input
for AU via 8 way 2 bit mux

           Reg_select : out STD_LOGIC_VECTOR (2 downto 0); -- to select Reg from
reg bank to write to a reg
```

```
           AddSub_select : out STD_LOGIC; -- selecting Add or Subtract (if neg
instruction sub will be selceted)


           Jump_flag : out STD_LOGIC; -- to  jump to another instruction in
PC(Program counter),
           -- by checking Reg_Value and this output connected to 2 way 3 bit MUX
selector (PC area)
           Jump_address : out STD_LOGIC_VECTOR (2 downto 0); -- which instruction
to execute, this output connected to 2 way 3 bit MUX


           Load_select : out STD_LOGIC; -- select between AU output and immediate
value (connected to 2 way 4 bit MUX)
           Immediate_value : out STD_LOGIC_VECTOR (3 downto 0); -- immediate
value given for movi (a number) or JZR (a address in program rom)
           MUL_select :  out STD_LOGIC); -- select between output from 2 way 4
bit mux and Multiplier output
end Instruction_decoder;


architecture Behavioral of Instruction_decoder is


component Decoder_2_to_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 ADD, NEG, MOVI, JZR :STD_LOGIC;


begin
Decoder_2_to_4_0: Decoder_2_to_4
port map( I(0)=>Instruction_bus(10),
          I(1)=>Instruction_bus(11),
```

```
            EN=>'1',
            Y(0) => ADD,
            Y(1) => NEG,
            Y(2) => MOVI,
            Y(3) => JZR);

Reg_a_address <= Instruction_bus(9 downto 7);
Reg_b_address <= Instruction_bus(6 downto 4);
Reg_select <= Instruction_bus(9 downto 7);
AddSub_select <= NEG; -- NEG is substracting from zero
Jump_Flag <= JZR AND (NOT (Reg_Value(0) OR Reg_Value(1) OR Reg_Value(2) OR
Reg_Value(3))) ;
Jump_Address <=Instruction_bus(2 downto 0);
Load_select <= MOVI;
MUL_select <= Instruction_bus(12);
Immediate_value <= Instruction_bus(3 downto 0);

end Behavioral;
```

Simulation Source File

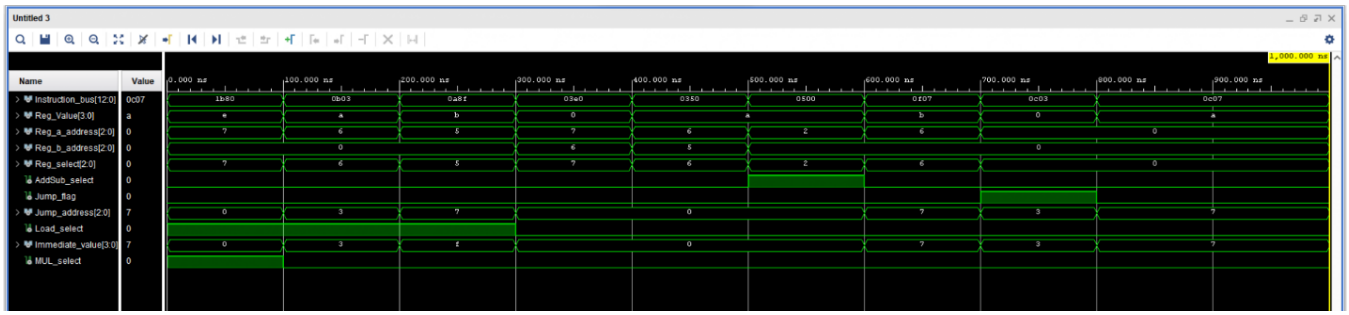
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Instruction_Decoder_13bit_Sim is
--  Port ( );
end Instruction_Decoder_13bit_Sim;

architecture Behavioral of Instruction_Decoder_13bit_Sim is
component Instruction_Decoder_13bit
```

```
    Port ( Instruction_bus : in STD_LOGIC_VECTOR (12 downto 0);
           Reg_Value : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_a_address : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_b_address : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_select : out STD_LOGIC_VECTOR (2 downto 0);
           AddSub_select : out STD_LOGIC;
           Jump_flag : out STD_LOGIC;
           Jump_address : out STD_LOGIC_VECTOR (2 downto 0);
           Load_select : out STD_LOGIC;
           Immediate_value : out STD_LOGIC_VECTOR (3 downto 0);
           MUL_select : out STD_LOGIC);
end component;

SIGNAL Instruction_bus : STD_LOGIC_VECTOR (12 downto 0);
SIGNAL Reg_Value : STD_LOGIC_VECTOR (3 downto 0);
SIGNAL Reg_a_address : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL Reg_b_address : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL Reg_select : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL AddSub_select : STD_LOGIC;
SIGNAL Jump_flag : STD_LOGIC;
SIGNAL Jump_address : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL Load_select : STD_LOGIC;
SIGNAL Immediate_value : STD_LOGIC_VECTOR (3 downto 0);
SIGNAL MUL_select : STD_LOGIC;

begin
UUT: Instruction_Decoder_13bit PORT MAP(
    Instruction_bus => Instruction_bus,
    Reg_Value => Reg_Value,
```

```
    Reg_a_address => Reg_a_address,
    Reg_b_address => Reg_b_address,
    Reg_select => Reg_select,
    AddSub_select => AddSub_select,
    Jump_flag => Jump_flag,
    Jump_address => Jump_address,
    Load_select => Load_select,
    Immediate_value => Immediate_value,
    MUL_select => MUL_select
    );
process
begin

    --1
    Reg_Value <= "1110";
    Instruction_bus <= "1101110000000";
    wait for 100ns;
--    2
    Reg_Value <= "1010";
    Instruction_bus <= "0101100000011";
    wait for 100ns;
    --3
    Reg_Value <= "1011";
    Instruction_bus <= "0101010001111";
    wait for 100ns;
    --4
    Reg_Value<= "0000";
    Instruction_bus <= "0001111100000";
    wait for 100ns;
```

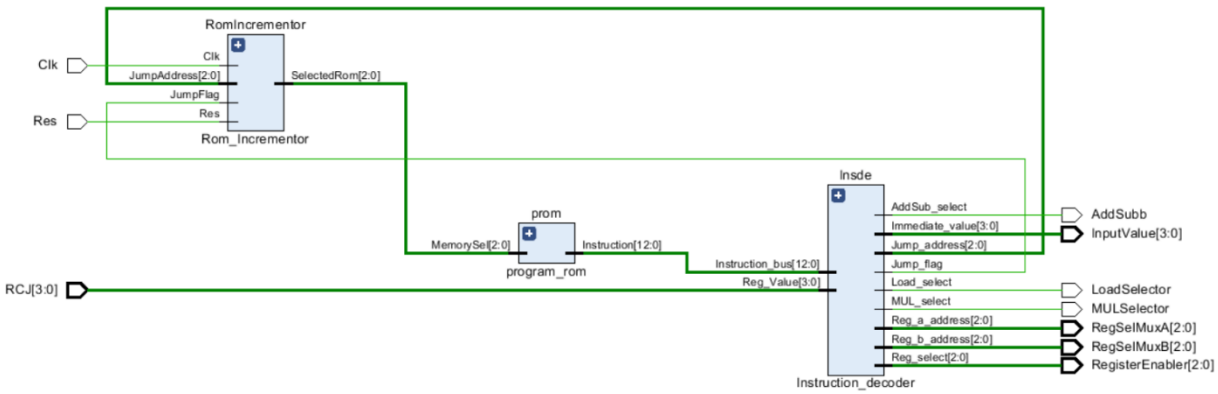
```
    --5
    Reg_Value <= "1010";
    Instruction_bus <= "0001101010000";
    wait for 100ns;
    --6
    Reg_Value <= "1010";
    Instruction_bus<="0010100000000";
    wait for 100ns;
    --7
    Reg_Value <= "1011";
    Instruction_bus<="0111100000111";
    wait for 100ns;
    --8
    Reg_Value <= "0000";
    Instruction_bus<="0110000000011";
    wait for 100ns;
    --9
    Reg_Value <= "1010";
    Instruction_bus<="0110000000111";
--    wait for 100ns;
    wait;

end process;

end Behavioral;
```

Timing Diagram



4. Program Decoder

Schematic



Design Source File

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity program_decoder is
    Port ( Clk : in STD_LOGIC;

           Res : in STD_LOGIC;
```

```
            RCJ : in STD_LOGIC_VECTOR (3 downto 0);
            RegisterEnabler : out STD_LOGIC_VECTOR (2 downto 0);
            LoadSelector : out STD_LOGIC;
            MULSelector : out STD_LOGIC;
            InputValue : out STD_LOGIC_VECTOR (3 downto 0);
            RegSelMuxA : out STD_LOGIC_VECTOR (2 downto 0);
            RegSelMuxB : out STD_LOGIC_VECTOR (2 downto 0);
            AddSubb : out STD_LOGIC);
end program_decoder;

architecture Behavioral of program_decoder is
component Rom_Incrementor
    Port ( Res : in STD_LOGIC;
       Clk : in STD_LOGIC;
       JumpFlag : in STD_LOGIC;
       JumpAddress : in STD_LOGIC_VECTOR (2 downto 0);
       SelectedRom : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component program_rom
    Port ( MemorySel : in STD_LOGIC_VECTOR (2 downto 0);
       Instruction : out STD_LOGIC_VECTOR (12 downto 0));

end component;

component Instruction_decoder
    Port ( Instruction_bus : in STD_LOGIC_VECTOR (12 downto 0); -- input from
program rom
       Reg_Value : in STD_LOGIC_VECTOR (3 downto 0); -- to check if a register
has value zero stored in it
```

```
        -- reg which we check is given with JZR instruction
        Reg_a_address : out STD_LOGIC_VECTOR (2 downto 0); -- selecting input for
AU via 8 way 2 bit mux
        Reg_b_address : out STD_LOGIC_VECTOR (2 downto 0); -- selecting input for
AU via 8 way 2 bit mux
        Reg_select : out STD_LOGIC_VECTOR (2 downto 0); -- to select Reg from reg
bank to write to a reg
        AddSub_select : out STD_LOGIC; -- selecting Add or Subtract (if neg
instruction sub will be selceted)

        Jump_flag : out STD_LOGIC; -- to  jump to another instruction in
PC(Program counter),
        -- by checking Reg_Value and this output connected to 2 way 3 bit MUX
selector (PC area)
        Jump_address : out STD_LOGIC_VECTOR (2 downto 0); -- which instruction to
execute, this output connected to 2 way 3 bit MUX

        Load_select : out STD_LOGIC; -- select between AU output and immediate
value (connected to 2 way 4 bit MUX)
        Immediate_value : out STD_LOGIC_VECTOR (3 downto 0);
        MUL_select :  out STD_LOGIC); -- select between output from 2 way 4 bit
mux and Multiplier output

end component;

signal JumpFlag:std_logic;
signal JumpAddress,SelectedRom:std_logic_vector (2 downto 0);
signal Ins:std_logic_vector (12 downto 0);

begin
RomIncrementor:Rom_Incrementor
port map(
```

```
Res => Res,
Clk => Clk,
JumpFlag => JumpFlag,
JumpAddress => JumpAddress,
SelectedRom => SelectedRom);


prom:program_rom
port map(
MemorySel => SelectedRom,
Instruction => Ins);


Insde:Instruction_decoder
port map(
Instruction_bus => Ins, -- input from program rom
       Reg_Value => RCJ, -- to check if a register has value zero stored in it
       -- reg which we check is given with JZR instruction
       Reg_a_address => RegSelMuxA, -- selecting input for AU via 8 way 2 bit mux
       Reg_b_address => RegSelMuxB, -- selecting input for AU via 8 way 2 bit mux
       Reg_select => RegisterEnabler, -- to select Reg from reg bank to write to
a reg
       AddSub_select => AddSubb, -- selecting Add or Subtract (if neg instruction
sub will be selceted)


       Jump_flag => JumpFlag, -- to  jump to another instruction in PC(Program
counter),
       -- by checking Reg_Value and this output connected to 2 way 3 bit MUX
selector (PC area)
       Jump_address => JumpAddress, -- which instruction to execute, this output
connected to 2 way 3 bit MUX
```

```
        Load_select => LoadSelector,-- select between AU output and immediate
value (connected to 2 way 4 bit MUX)
        Immediate_value => InputValue,
        MUL_select => MULSelector);


end Behavioral;
```

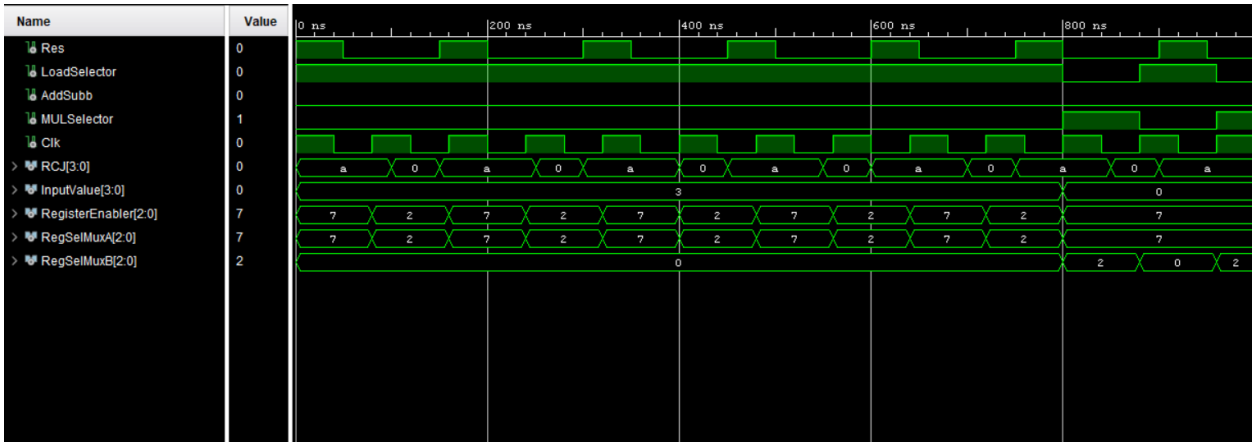
Simulation Source File

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

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

architecture Behavioral of prodecodetb is
component program_decoder
port(Clk : in STD_LOGIC;
            Res : in STD_LOGIC;
            RCJ : in STD_LOGIC_VECTOR (3 downto 0);
            RegisterEnabler : out STD_LOGIC_VECTOR (2 downto 0);
            LoadSelector : out STD_LOGIC;
            MULSelector : out STD_LOGIC;
            InputValue : out STD_LOGIC_VECTOR (3 downto 0);
            RegSelMuxA : out STD_LOGIC_VECTOR (2 downto 0);
            RegSelMuxB : out STD_LOGIC_VECTOR (2 downto 0);
            AddSubb : out STD_LOGIC);
end component;
```

```
signal Res,LoadSelector,AddSubb,MULSelector:std_logic;
signal Clk : std_logic:='0';
signal RCJ,InputValue : std_logic_vector (3 downto 0);
signal RegisterEnabler,RegSelMuxA,RegSelMuxB : std_logic_vector (2 downto 0);

begin
UUT : program_decoder
port map(
Clk => Clk,
Res => Res,
RCJ => RCJ,
RegisterEnabler => RegisterEnabler,
LoadSelector => LoadSelector,
InputValue => InputValue,
RegSelMuxA => RegSelMuxA,
RegSelMuxB => RegSelMuxB,
MULSelector => MULSelector,
AddSubb => AddSubb);

process begin
Clk <= not (Clk);
wait for 40 ns;
end process;

process begin
Res <= '1';
RCJ <= "1010";
wait for 50 ns;
Res <= '0';
```

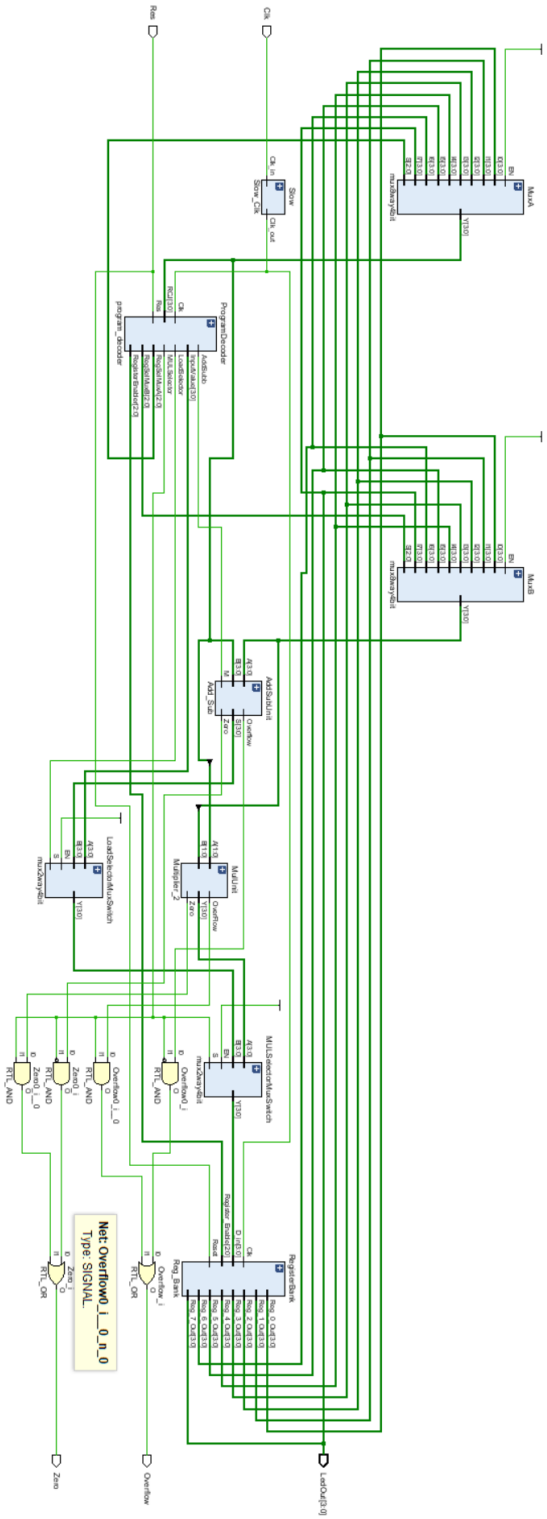
```
wait for 50 ns;
RCJ <= "0000";
wait for 50 ns;
end process;

end Behavioral;
```

Timing Diagram



Nano Processor with Multiplier

Schematic



Design Source File

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

entity NanoProcessor is
    Port ( Res : in std_logic;
           LedOut : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC;
           Clk : in STD_LOGIC);
end NanoProcessor;

architecture Behavioral of Nanoprocessor is
component program_decoder
    Port ( Clk : in STD_LOGIC;
           Res : in STD_LOGIC;
           RCJ : in STD_LOGIC_VECTOR (3 downto 0);
           RegisterEnabler : out STD_LOGIC_VECTOR (2 downto 0);
           LoadSelector : out STD_LOGIC;
           MULSelector : out STD_LOGIC;
           InputValue : out STD_LOGIC_VECTOR (3 downto 0);
           RegSelMuxA : out STD_LOGIC_VECTOR (2 downto 0);
           RegSelMuxB : out STD_LOGIC_VECTOR (2 downto 0);
           AddSubb : out STD_LOGIC);
end component;
component Reg_Bank
    Port ( Register_Enable : in STD_LOGIC_VECTOR (2 downto 0);
       D_in : in STD_LOGIC_VECTOR (3 downto 0);
       Clk : in STD_LOGIC;
```

```
        Reset : in STD_LOGIC;
        Reg_0_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Reg_1_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Reg_2_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Reg_3_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Reg_4_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Reg_5_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Reg_6_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Reg_7_Out : out STD_LOGIC_VECTOR (3 downto 0));

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

component mux8way4bit
    Port ( I0 : in STD_LOGIC_VECTOR (3 downto 0);
       I1 : in STD_LOGIC_VECTOR (3 downto 0);
       I2 : in STD_LOGIC_VECTOR (3 downto 0);
       I3 : in STD_LOGIC_VECTOR (3 downto 0);
       I4 : in STD_LOGIC_VECTOR (3 downto 0);
       I5 : in STD_LOGIC_VECTOR (3 downto 0);
       I6 : in STD_LOGIC_VECTOR (3 downto 0);
       I7 : in STD_LOGIC_VECTOR (3 downto 0);
       EN : in STD_LOGIC;
       S : in STD_LOGIC_VECTOR (2 downto 0);
       Y : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end component;
component Add_Sub
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
       B : in STD_LOGIC_VECTOR (3 downto 0);
       M : in STD_LOGIC;
       S : out STD_LOGIC_VECTOR (3 downto 0);
       Overflow : out STD_LOGIC;
       Zero : out STD_LOGIC);
end component;

component Multiplier_2
    Port ( A : in STD_LOGIC_VECTOR (1 downto 0);
           B : in STD_LOGIC_VECTOR (1 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           OverFlow : out STD_LOGIC);
end component;

component mux2way4bit
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
       B : in STD_LOGIC_VECTOR (3 downto 0);
       EN : in STD_LOGIC;
       S : in STD_LOGIC;
       Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal calout, MULout, immivalue,corrvalue, Data : std_logic_vector (3 downto 0);
signal RIA,RIB,RE,RSS : std_logic_vector (2 downto 0);
signal loadsel, mulsel, AS, Clki, MulZeroFlag, MulOverFlowFlag, AddSubZeroFlag,
AddSubOverFlowFlag : std_logic;
```

```
signal R0,R1,R2,R3,R4,R5,R6,R7,NA,NB : std_logic_vector (3 downto 0);

signal NA_2Bit, NB_2Bit :  std_logic_vector (1 downto 0); -- input for 2 bit
multiplier

begin

Slow:Slow_Clk

port map(

Clk_in => Clk,

Clk_out => Clki);


ProgramDecoder:program_decoder

port map(

Clk => Clki,

Res => Res,

RCJ => NA,

RegisterEnabler => RE,

LoadSelector => loadsel,

MULSelector => mulsel,

InputValue => immivalue,

RegSelMuxA => RIA,

RegSelMuxB => RIB,

AddSubb => AS);


LoadSelectorMuxSwitch : mux2way4bit

port map(

B => calout,

A => immivalue,

EN => '1',

S => loadsel,

Y => corrvalue);
```

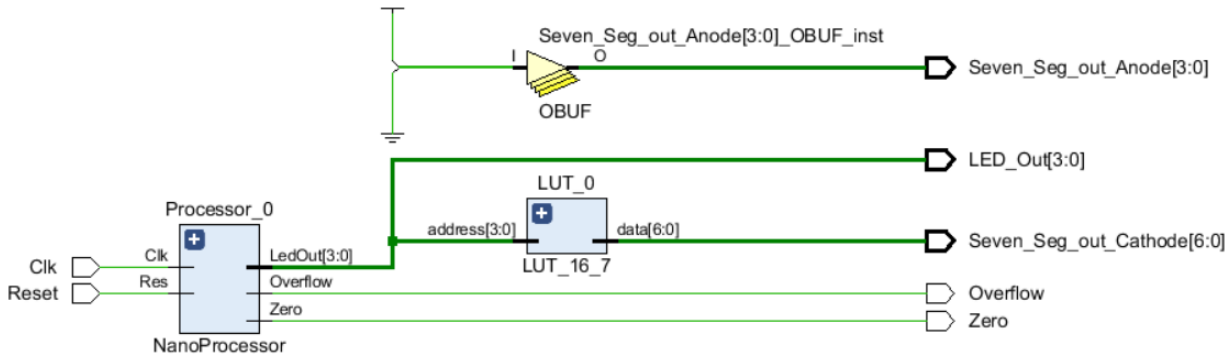
```
MULSelectorMuxSwitch : mux2way4bit
port map(
B => corrvalue,
A => MULout,
EN => '1',
S => mulsel, -- if it is '0' B will be selected which is addsub or immidiate
value
Y => Data);

RegisterBank : Reg_Bank
port map(
D_in => Data,
Clk => Clki,
Reset => Res,
Reg_0_Out => R0,
Reg_1_Out => R1,
Reg_2_Out => R2,
Reg_3_Out => R3,
Reg_4_Out => R4,
Reg_5_Out => R5,
Reg_6_Out => R6,
Reg_7_Out => R7,
Register_Enable => RE);

MuxA: mux8way4bit
port map(
I0 => R0,
I1 => R1,
```

```
I2 => R2,
I3 => R3,
I4 => R4,
I5 => R5,
I6 => R6,
I7 => R7,
EN => '1',
S => RIA,
Y => NA);

MuxB: mux8way4bit
port map(
I0 => R0,
I1 => R1,
I2 => R2,
I3 => R3,
I4 => R4,
I5 => R5,
I6 => R6,
I7 => R7,
EN => '1',
S => RIB,
Y => NB);

AddSubUnit : Add_Sub
port map(
A => NB, -- value from Mux B
B => NA, -- value from Mux A
M => AS,
```

```
S => calout,

Overflow => AddSubOverFlowFlag,

Zero => AddSubZeroFlag);


NA_2Bit <= NA(1 downto 0);
NB_2Bit <= NB(1 downto 0);

MulUnit : Multiplier_2

port map(

            A => NA_2Bit,

            B => NB_2Bit,

            Y => MULout,

            Zero => MulZeroFlag,

            OverFlow => MulOverFlowFlag

);


Zero <= (AddSubZeroFlag AND (NOT mulsel)) OR (MulZeroFlag AND mulsel);

Overflow <= (AddSubOverFlowFlag AND (NOT mulsel)) OR (MulOverFlowFlag AND
mulsel);


LedOut <= R7;


end Behavioral;
```

Nano Processor with Multiplier and 7-Segmet Display

Schematic



Design Source File

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NanoProcessor_7Seg is
    Port ( Reset : in std_logic;
           Clk : in STD_LOGIC;
           LED_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC;
           Seven_Seg_out_Anode : out STD_LOGIC_VECTOR (3 downto 0);
           Seven_Seg_out_Cathode : out STD_LOGIC_VECTOR (6 downto 0)
       );
end NanoProcessor_7Seg;

architecture Behavioral of NanoProcessor_7Seg is
component LUT_16_7
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end component;
```

```
component NanoProcessor
    Port ( Res : in std_logic;
           LedOut : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC;
           Clk : in STD_LOGIC);
end component;

signal OutSig : std_logic_vector (3 downto 0);
signal To7Seg : std_logic_vector (6 downto 0);

begin

Processor_0 : NanoProcessor
    port map (
        Res => Reset,
        Clk => Clk,
        LedOut => OutSig,
        Overflow => Overflow,
        Zero => Zero);

LUT_0 : LUT_16_7
    port map (
        Address => OutSig,
        Data => To7Seg);
LED_Out <= OutSig;

Seven_Seg_out_Cathode <= To7Seg;
```

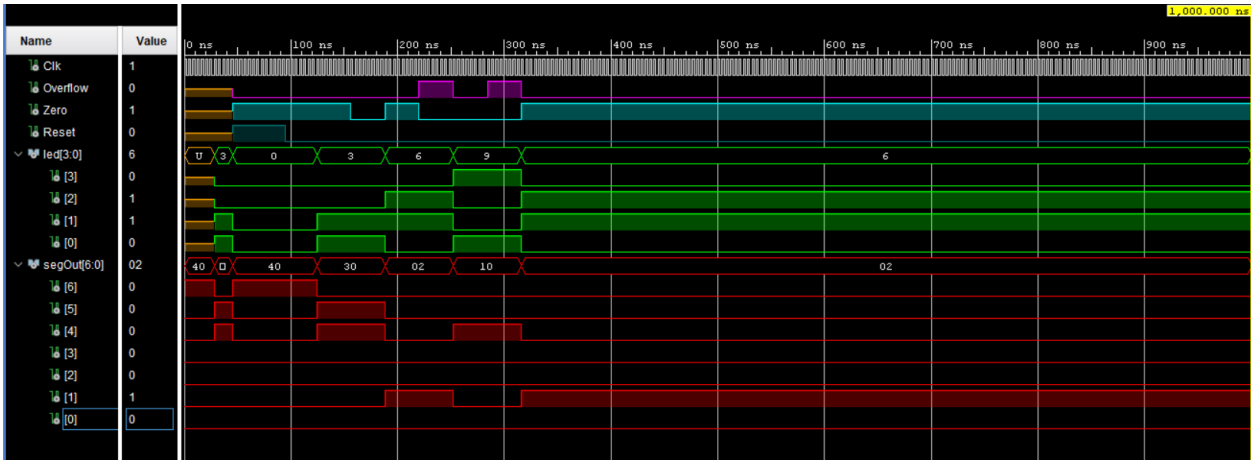
```
Seven_Seg_out_Anode <= "1110";
end Behavioral;
```

Simulation Source File

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

entity TB_7Seg_With_Multi is
--  Port ( );
end TB_7Seg_With_Multi;

architecture Behavioral of TB_7Seg_With_Multi is
component NanoProcessor_7Seg
    Port ( Reset : in std_logic;
           Clk : in STD_LOGIC;
           LED_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC;
           Seven_Seg_out_Anode : out STD_LOGIC_VECTOR (3 downto 0);
           Seven_Seg_out_Cathode : out STD_LOGIC_VECTOR (6 downto 0)
       );
end component;

signal Clk : std_logic := '0';
signal Overflow, Zero, Reset : std_logic;
signal led : std_logic_vector (3 downto 0);
signal segOut : std_logic_vector (6 downto 0);

begin
```

```
UUT : NanoProcessor_7Seg
    port map (
        Reset => Reset,
        Clk => Clk,
        Zero => Zero,
        Overflow => Overflow,
        LED_Out => led,
        Seven_Seg_out_Cathode => segOut);
process
begin
    Clk <= not(Clk);
    wait for 2 ns;
end process;

process
begin
    wait for 45 ns;
    Reset <= '1';
    wait for 50 ns;
    Reset <= '0';
    wait;
end process;
end Behavioral;
```

Timing Diagram



Resource Utilization

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (106)	BUFGCTRL (32)
NanoProcessor_7Seg	50	49	27	50	8	19	1
Processor_0 (NanoPr...	50	49	27	50	8	0	0
ProgramDecoder (p...	27	3	10	27	2	0	0
RegisterBank (Reg...	12	12	10	12	1	0	0
Slow (Slow_Clk)	11	34	16	11	3	0	0

Resource	Utilization	Available	Utilization %
LUT	50	20800	0.24
FF	49	41600	0.12
IO	19	106	17.92

Instructions to Operate the Nano Processor

Center Button – RESET – Pushing this button will execute the instructions again (Use this button to observe the process)

LED 0 – LED 3 => These LEDs depict the output of Register 7 (Two's complement of numbers from -8 to 7)

LED 14 – Zero – Will light Up when the Output of Multiplier, Adder/Subtractor from current instruction is Zero

LED 15 – Overflow – Will light if there is an Overflow fromAdder/Subtractor

Seven Segment Display – It will display the value stored in Register 7 (similar to LED0-LED3)

Problems Faced and solutions

Problem- Adding an instruction to multiplier without affecting other parts' functionality of the nano processor

Solution - Use 13-bit instruction and use its MSD as a multiplier unit selector and use this bit to activate proper units and output zero and overflow correctly

Problem – If we want to add or remove components we had to change the whole code of nano processor.

Solution – Instead of Implementing Nano processor as a single component we divided it into sub components if we need to add/remove any components we just need to change smaller components (Nano Processor with seven segment remains same unless we need to add more output flags).
For Example, To add multiplier we needed increased instruction size and changed program rom and instruction decoder but the change was encapsulated in Program Decoder)

Problem - Assuring that the program counter updates its address value one by one synchronously. When selecting a Rom(R0-R7) from the program counter we can use a 3 to 8 decoder or LUT.

Solution - Use a Synchronous Process-Define a clock signal. This ensures that the operations inside the process occur only when the clock transitions from low to high(0->1).Rom Selector is implemented by using LUT for flexibility and programmability

Suggestions for Future Development

- **Adding Divider Component**

  Like Multiplication circuit we can add divider circuit and perform division operation (Div). We need to design a Divider Circuit (4-bit by 2-bit Divider) and we need modify Instruction Decoder and Program Decoder according to that.

- **Improving the functionality of Seven Segment Display**

  Now we only utilize one of the four displays provided on Basys3 board. We can make use of the other displays as well. For example, we can use the second display to show whether the result is negative or not (use second display to show the "-" sign and the first display and the magnitude in the first display)

- **Multiple ROMs**

  Instead of changing the VHDL code of Program ROM for each instruction set we can store multiple instruction sets in different ROMs and select the ROMs from external inputs.

- **Increase bit size of components**

  Instead of 4-bit Registers and Adder-Subtractor unit we can use higher number of bits (6-bit or 8-bit) to support large calculations and more instructions

Conclusion

In this lab project, our objective was to design a 4-bit nano processor capable of executing basic instructions. Our aim was to gain practical knowledge into nano processor design and component functionality. To construct a functional processor, we approached the task by developing individual components and validating their operations. Utilizing buses for connectivity in our design, reducing the complexity associated with increased wiring.

Given that our processor operates on machine code, we manually translated assembly language instructions into machine code and embedded them into ROM. Knowledge from our previous lab designs made the process easier, allowing us to use existing components rather than starting from scratch. This approach facilitated a deeper understanding of each component's internal workings and the processor's instruction decoding and execution mechanisms. Working collaboratively as a team, we developed our skills in component integration, communication, coordination, and responsibility sharing. This project served as a hands-on learning experience in digital design, computer organization, and teamwork, enabling us to enhance our proficiency and understanding in these domains.