NANO PROCESSOR

CS1050 - Computer Organization and Digital Design

Group – **69**

Prepared by:

K.V. Kahapola	– 210266G
A.S. Galappaththi	– 210172N

Table of Contents

INTRODUCTION

The final lab assignment involves the design of a microprocessor with specific functionalities. It includes a 4-bit Add/Subtract unit that performs addition and subtraction using the 2's complement method. Additionally, a 3-bit Program Counter (PC) is designed using a 3-bit adder, a 3-bit register, and a 2-way 3-bit multiplexer. The microprocessor utilizes an Instruction Decoder to activate the necessary components based on the instructions provided by the program ROM, which stores the Assembly program. The use of buses helps in combining components and reducing code complexity. A Register Bank is implemented using registers and a 3-to-8 decoder from previous labs. The nano processor supports four operations: direct value-to-register transfer, addition of two values, computation of the 2's complement of a value and jumping to a specific instruction. To perform subtraction, it is necessary to negate one value and add them together.

NANO PROCESSOR

The processor system consists of two input signals: the Reset push button and the Clock. It generates three output signals: Reg7, overflow, and zero flags. These output signals provide information about the status and values of the corresponding components within the processor.

DESIGN SOURCE CODE

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

entity Nano_Processor is
    Port ( Clk          : in STD_LOGIC;
           Reset        : in STD_LOGIC;
           Overflow     : out STD_LOGIC;
           Zero         : out STD_LOGIC;
           LED_Out        : out STD_LOGIC_VECTOR (3 downto 0);
           Out_7Seg  : out STD_LOGIC_VECTOR(6 downto 0));
end Nano_Processor;

architecture Behavioral of Nano_Processor is

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

signal SlowClk : std_logic;

component Instruction_Decoder is
    Port ( I      : in STD_LOGIC_VECTOR (11 downto 0);
           JMP_Check        : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_En           : out STD_LOGIC_VECTOR (2 downto 0);
           Load_Sel         : out STD_LOGIC;
           Im_Val          : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_Sel_0        : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_Sel_1        : out STD_LOGIC_VECTOR (2 downto 0);
           Add_Sub_Sel        : out STD_LOGIC;
           JMP_Flag        : out STD_LOGIC;
           JMp_Address          : out STD_LOGIC_VECTOR (2 downto 0);
           Neg          : out STD_LOGIC);
end component;

signal ins                                  : std_logic_vector(11 downto 0);
signal imVal                               : std_logic_vector(3 downto 0);
signal regEn, regSelA, regSelB, jmpAdd      : std_logic_vector(2 downto 0);
signal loadSel, addSubSel, jmpFlag, negSel  : std_logic;

component PC is
    Port ( Clk      : in STD_LOGIC;
           Reset    : in STD_LOGIC;
           Jump_Flag : in STD_LOGIC;
           Jump_Address  : in STD_LOGIC_VECTOR (2 downto 0);
           Sel   : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal memSel : std_logic_vector(2 downto 0);

component ROM is
    Port ( Sel      : in STD_LOGIC_VECTOR (2 downto 0);
           Ins  : out STD_LOGIC_VECTOR (11 downto 0));
end component;

component Reg_bank is
    Port ( Reg_Bank_in   : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_EN        : in STD_LOGIC_VECTOR (2 downto 0);
           Clk          : in STD_LOGIC;
           R0   : out STD_LOGIC_VECTOR (3 downto 0);
           R1   : out STD_LOGIC_VECTOR (3 downto 0);
           R2   : out STD_LOGIC_VECTOR (3 downto 0);
           R3   : out STD_LOGIC_VECTOR (3 downto 0);
           R4   : out STD_LOGIC_VECTOR (3 downto 0);
           R5   : out STD_LOGIC_VECTOR (3 downto 0);
           R6   : out STD_LOGIC_VECTOR (3 downto 0);
           R7   : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal regBankIn, r0, r1, r2, r3, r4, r5, r6, r7 : std_logic_vector(3 downto
0);

component MUX_8_way_4 is
    Port ( R0       : in STD_LOGIC_VECTOR (3 downto 0);
           R1       : in STD_LOGIC_VECTOR (3 downto 0);
           R2       : in STD_LOGIC_VECTOR (3 downto 0);
           R3       : in STD_LOGIC_VECTOR (3 downto 0);
           R4       : in STD_LOGIC_VECTOR (3 downto 0);
           R5       : in STD_LOGIC_VECTOR (3 downto 0);
           R6       : in STD_LOGIC_VECTOR (3 downto 0);
           R7       : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_Sel   : in STD_LOGIC_VECTOR (2 downto 0);
           Q        : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal muxOut0, muxOut1 : std_logic_vector(3 downto 0);

component Add_Sub_4 is
    Port ( Mux1_out     : in STD_LOGIC_VECTOR (3 downto 0);
           Mux2_out     : in STD_LOGIC_VECTOR (3 downto 0);
           Add_Sub_sel  : in STD_LOGIC;
           Neg_sel      : in STD_LOGIC;
           Add_Sub_out  : out STD_LOGIC_VECTOR (3 downto 0);
           overflow     : out STD_LOGIC;
           zero         : out STD_LOGIC);
end component;

signal addSubOut        : std_logic_vector(3 downto 0);

component Mux_2_way_4 is
    Port ( im_Val           : in STD_LOGIC_VECTOR (3 downto 0);
           add_Sub_Out    : in STD_LOGIC_VECTOR (3 downto 0);
           load_Sel          : in STD_LOGIC;
           mux_out          : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component LUT_7seg is
    Port ( Address  : in STD_LOGIC_VECTOR (3 downto 0);
           Data     : out STD_LOGIC_VECTOR (6 downto 0));
end component;

begin

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

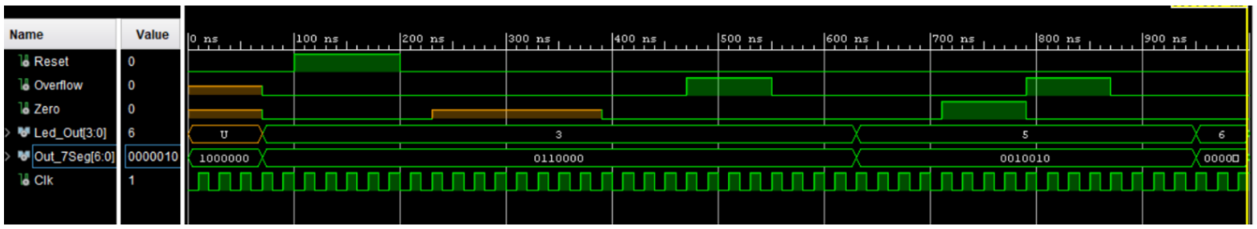
```
RegisterBank_0 : Reg_bank
    port map(
        Reg_Bank_in  => regBankIn,
        Reg_EN       => regEn,
        Clk         => SlowClk,
        R0    => r0,
        R1    => r1,
        R2    => r2,
        R3    => r3,
        R4    => r4,
        R5    => r5,
        R6    => r6,
        R7    => r7);

Mux8_4bit_A : MUX_8_way_4
    port map(
        R0 => r0,
        R1 => r1,
        R2 => r2,
        R3 => r3,
        R4 => r4,
        R5 => r5,
        R6 => r6,
        R7 => r7,
        Reg_Sel => regSelA,
        Q => muxOut0);

Mux8_4bit_B : MUX_8_way_4
    port map(
        R0      => r0,
        R1      => r1,
        R2      => r2,
        R3      => r3,
        R4      => r4,
        R5      => r5,
        R6      => r6,
        R7      => r7,
        Reg_Sel  => regSelB,
        Q       => muxOut1);

AddSubUnit_0 : Add_Sub_4
    port map(
        Mux1_out    => muxOut0,
        Mux2_out    => muxOut1,
        Add_Sub_sel => addSubSel,
        Neg_Sel     => negSel,
        Add_Sub_out => addSubOut,
        overflow    => Overflow,
        zero        => Zero);

Mux2_4bit_0 : Mux_2_way_4
    port map(
        im_Val => imVal,
        add_Sub_Out => addSubOut,
        load_Sel => loadSel,
        mux_out => regBankIn);

Ins_Decoder_0 : Instruction_Decoder
    port map(
        I => ins,
        JMP_Check   => muxOut0,
        Reg_En      => regEn,
        Load_Sel    => loadSel,
        Im_Val     => imVal,
        Reg_Sel_0   => regSelA,
        Reg_Sel_1   => regSelB,
        Add_Sub_Sel  => addSubSel,
        JMP_Flag   => jmpFlag,
        JMP_Address    => jmpAdd,
        NEG     => negSel);

Program_ROM_0 : ROM
    port map(
        Sel     => memSel,
        Ins => ins);

ProgramCounter_0 : PC
    port map(
        Clk         => SlowClk,
        Reset       => Reset,
        Jump_Flag    => jmpFlag,
        Jump_Address     => jmpAdd,
        Sel      => memSel);

LUT_7seg_0 : LUT_7seg
    port map(
        Address => r7,
        Data    => Out_7Seg);

    LED_Out <= r7;

end Behavioral;
```

TEST BENCH CODE

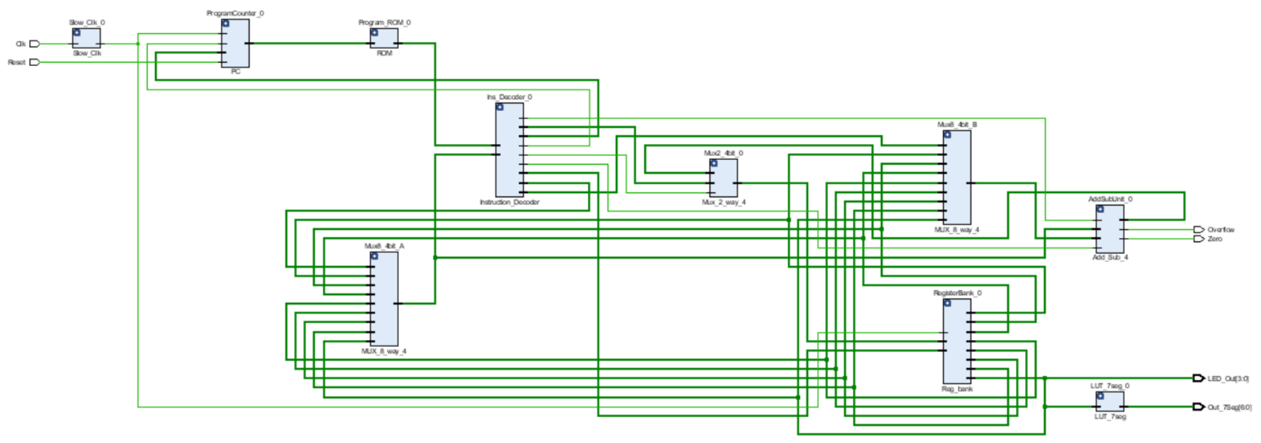
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

architecture Behavioral of TB_Nano_Processor is

component Nano_Processor is
    Port ( Clk          : in STD_LOGIC;
           Reset        : in STD_LOGIC;
           Overflow     : out STD_LOGIC;
           Zero         : out STD_LOGIC;
           Led_Out         : out STD_LOGIC_VECTOR (3 downto 0);
           Out_7Seg  : out STD_LOGIC_VECTOR(6 downto 0));
```

```
end component;

signal Reset, Overflow, Zero   : std_logic;
signal Led_Out                        : std_logic_vector(3 downto 0);
signal Out_7Seg                  : std_logic_vector(6 downto 0);
signal Clk : std_logic :='0';


begin

UUT : Nano_Processor port map(
        Clk         => Clk,
        Reset       => Reset,
        Overflow    => Overflow,
        Zero        => Zero,
        Led_Out       => Led_Out,
        Out_7Seg => Out_7Seg);

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

sim : process
    begin
        Reset <= '0';
        wait for 100ns;
        Reset <= '1';
        wait for 100ns;
        Reset <= '0';
    wait;
end process;

end behavioral;
```

TIMING DIAGRAM



RTL ANALYSIS - SCHEMATIC



ROM

The program ROM stores machine codes for each instruction required to be executed by the processor. Each instruction is assigned a separate location in the ROM, with each location consisting of 12 bits. The first two bits indicate the opcode of the instruction. The subsequent six bits represent the register addresses, with each address allocated 3 bits. The remaining four bits are utilized for the immediate value, while the last three bits are dedicated to the jump address. This structured format ensures proper encoding and decoding of instructions within the processor.

DESIGN SOURCE CODE

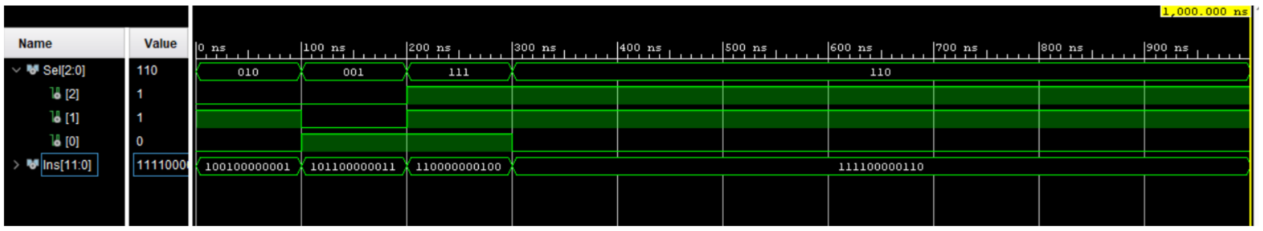
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ROM is
    Port ( Sel       : in STD_LOGIC_VECTOR (2 downto 0);
           Ins   : out STD_LOGIC_VECTOR (11 downto 0));
end ROM;

architecture Behavioral of ROM is

type rom_type is array( 0 to 7 ) of std_logic_vector (11 downto 0);
    signal Ins_ROM : rom_type := (
    "101110000011", --MOVI R7, 3
    "101100000011", --MOVI R1, 3
    "100100000001", --MOVI R2, 1
    "010100000000", --NEG R2
    "001100100000", --ADD R1, R2
    "001111100000", --ADD R7, R1
    "111100000110", --JZR R1, 6
    "110000000100"  --JZR R0, 4
     --"100010001010", --MOVI R1, 10 ; R1 ? 10
     --"100100000001", --MOVI R2, 1 ; R2 ? 1
     --"010100000000", --NEG R2 ; R2 ? -R2
     --"000010100000", --ADD R1, R2 ; R1 ? R1 + R2
     --"110010000111", --JZR R1, 7 ; If R1 = 0 jump to line 7
     --"110000000011", --JZR R0, 3 ; If R0 = 0 jump to line 3
     --"000000000000",
     --"000000000000"
        );

begin

Ins <= Ins_ROM(to_integer(unsigned(Sel)));

end Behavioral;
```

TEST BENCH CODE

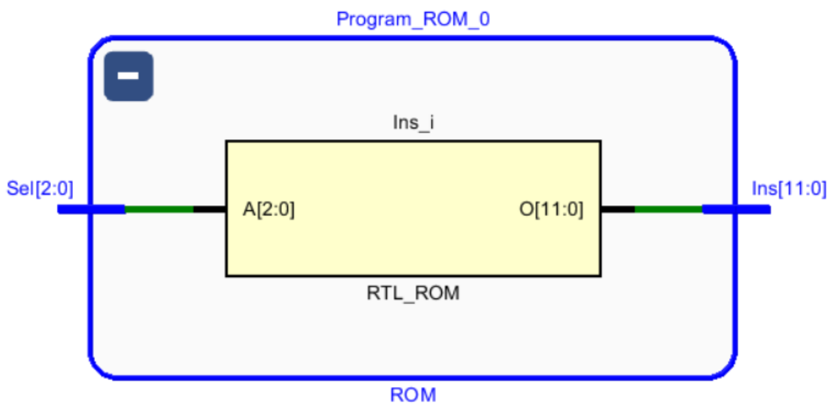
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


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

architecture Behavioral of TB_ROM is

component ROM is
    Port ( Sel      : in STD_LOGIC_VECTOR (2 downto 0);
           Ins   : out STD_LOGIC_VECTOR (11 downto 0));
end component;

signal Sel : std_logic_vector(2 downto 0);
signal Ins : std_logic_vector(11 downto 0);

begin

UUT : ROM port map(
        Sel     => Sel,
        Ins => Ins);

process
    begin
        Sel <= "010";
        wait for 100ns;

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

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

        Sel <= "110";
    wait;
```

```
end process;

end Behavioral;
```

TIMING DIAGRAM



RTL ANALYSIS - SCHEMATIC



INSTRUCTION DECODER

The instruction decoder receives the instruction from the program ROM during each clock cycle. Its purpose is to determine the required components that need activation and the corresponding signals to be transmitted to each activated component. It incorporates a 2-to-4 decoder to recognize the instruction opcode and appropriately distribute the provided register addresses and immediate values.

DESIGN SOURCE CODE

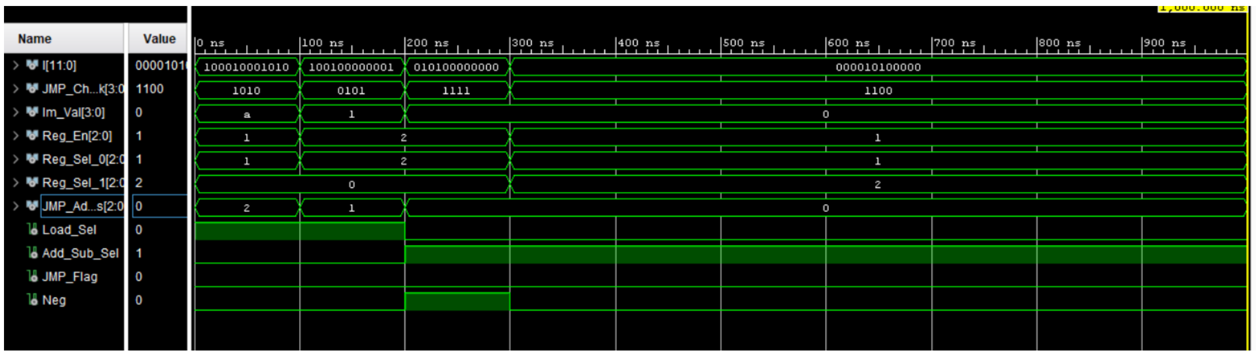
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Instruction_Decoder is
    Port ( I        : in STD_LOGIC_VECTOR (11 downto 0);
           JMP_Check         : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_En            : out STD_LOGIC_VECTOR (2 downto 0);
           Load_Sel          : out STD_LOGIC;
           Im_Val           : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_Sel_0         : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_Sel_1         : out STD_LOGIC_VECTOR (2 downto 0);
           Add_Sub_Sel        : out STD_LOGIC;
           JMP_Flag         : out STD_LOGIC;
           JMP_Address          : out STD_LOGIC_VECTOR (2 downto 0);
           NEG           : out STD_LOGIC);
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is

component Decoder_2_to_4 is
    Port ( I : in STD_LOGIC_VECTOR (1 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal Address, Neg0, Mov, Jmp, En_Sel : std_logic;

begin

    Decoder_2_to_4_0: Decoder_2_to_4
        port map(
                I(0)     => I(10),
                I(1)     => I(11),
                EN       => '1',
                Y(0)     => Address,
                Y(1)     => Neg0,
                Y(2)     => Mov,
                Y(3)     => Jmp);

    Load_Sel     <= Mov;
    Add_Sub_Sel   <= Address or Neg0;
    NEG      <= Neg0;

    En_Sel       <= Address or Mov or Neg0;

    Im_Val      <= I(3 downto 0);

    Reg_En(0)    <= En_Sel and I(7);
    Reg_En(1)    <= En_Sel and I(8);
    Reg_En(2)    <= En_Sel and I(9);

    Reg_Sel_0    <= I(9 downto 7);
    Reg_Sel_1    <= I(6 downto 4);

    JMP_Flag    <= Jmp and (not (JMP_Check(0) or JMP_Check(1) or JMP_Check(2)
or JMP_Check(3)));
    JMP_Address     <= I(2 downto 0);

end Behavioral;
```

TEST BENCH CODE

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

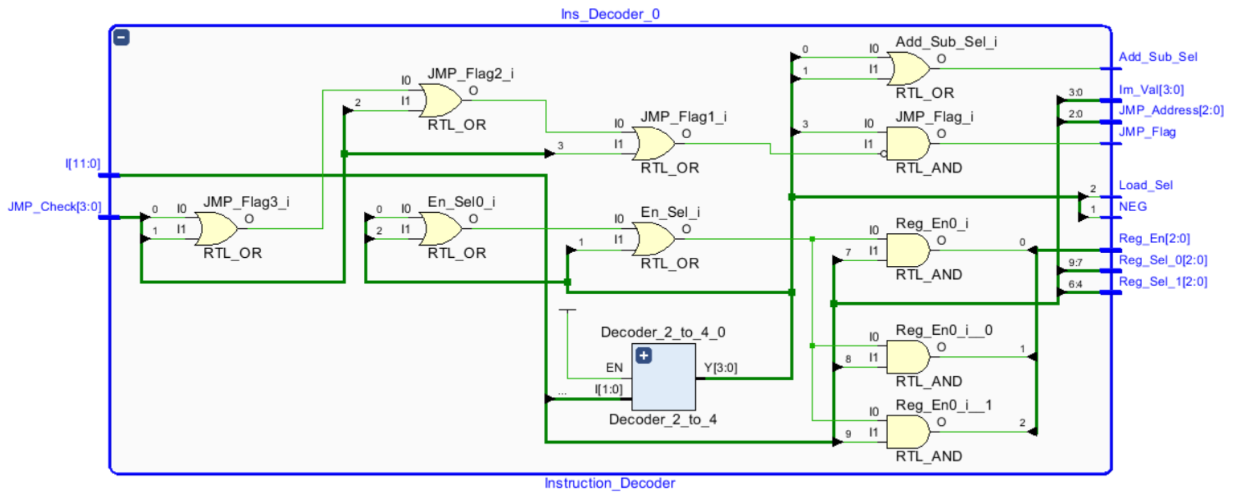
```
entity TB_Instruction_Decoder is
--  Port ( );
end TB_Instruction_Decoder;

architecture Behavioral of TB_Instruction_Decoder is

component Instruction_Decoder is
    Port ( I      : in STD_LOGIC_VECTOR (11 downto 0);
           JMP_Check        : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_En           : out STD_LOGIC_VECTOR (2 downto 0);
           Load_Sel         : out STD_LOGIC;
           Im_Val          : out STD_LOGIC_VECTOR (3 downto 0);
           Reg_Sel_0        : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_Sel_1        : out STD_LOGIC_VECTOR (2 downto 0);
           Add_Sub_Sel       : out STD_LOGIC;
           JMP_Flag        : out STD_LOGIC;
           JMP_Address          : out STD_LOGIC_VECTOR (2 downto 0);
           Neg          : out STD_LOGIC);
end component;

signal I                             : std_logic_vector(11 downto 0);
signal JMP_Check, Im_Val                         : std_logic_vector(3 downto 0);
signal Reg_En, Reg_Sel_0, Reg_Sel_1, JMP_Address   : std_logic_vector(2 downto
0);
signal Load_Sel, Add_Sub_Sel, JMP_Flag, Neg : std_logic;

begin

UUT : Instruction_Decoder port map(
        I => I,
        JMP_Check   => JMP_Check,
        Reg_En      => Reg_En,
        Load_Sel    => Load_Sel,
        Im_Val     => Im_Val,
        Reg_Sel_0   => Reg_Sel_0,
        Reg_Sel_1   => Reg_Sel_1,
        Add_Sub_Sel  => Add_Sub_Sel,
        JMP_Flag   => JMP_Flag,
        JMP_Address    => JMP_Address,
        Neg     => Neg);

process
    begin
        I <= "100010001010";
        JMP_Check   <= "1010";
        wait for 100ns;

        I <= "100100000001";
        JMP_Check   <= "0101";
        wait for 100ns;

        I <= "010100000000";
        JMP_Check   <= "1111";
        wait for 100ns;
        I <= "000010100000";
        JMP_Check    <= "1100";
    wait;
end process;

end Behavioral;
```

TIMING DIAGRAM



Name	Value	0–100 ns	100–200 ns	200–300 ns	300–1000 ns
I[11:0]	0000101	100010001010	100100000001	010100000000	000010100000
JMP_Ch...k[3:0]	1100	1010	0101	1111	1100
Im_Val[3:0]	0	a	1	0	0
Reg_En[2:0]	1	1	2	2	1
Reg_Sel_0[2:0]	1	1	2	2	1
Reg_Sel_1[2:0]	2	0	0	0	2
JMP_Ad...s[2:0]	0	2	1	0	0
Load_Sel	0				
Add_Sub_Sel	1				
JMP_Flag	0				
Neg	0				

RTL ANALYSIS - SCHEMATIC



REGISTER BANK

The register bank comprises eight 4-bit registers (Reg0 to Reg7) implemented using D flip-flops. To enable each register individually based on the instruction, a 3-to-8 decoder is employed. Reg0 is preset to the value "0000" within the register bank, simplifying the implementation of the NEG instruction. Additionally, Reg7, which stores the final output, is directly connected to the Basys3 LEDs and the 7-segment display through the utilization of a lookup table.

DESIGN SOURCE CODE

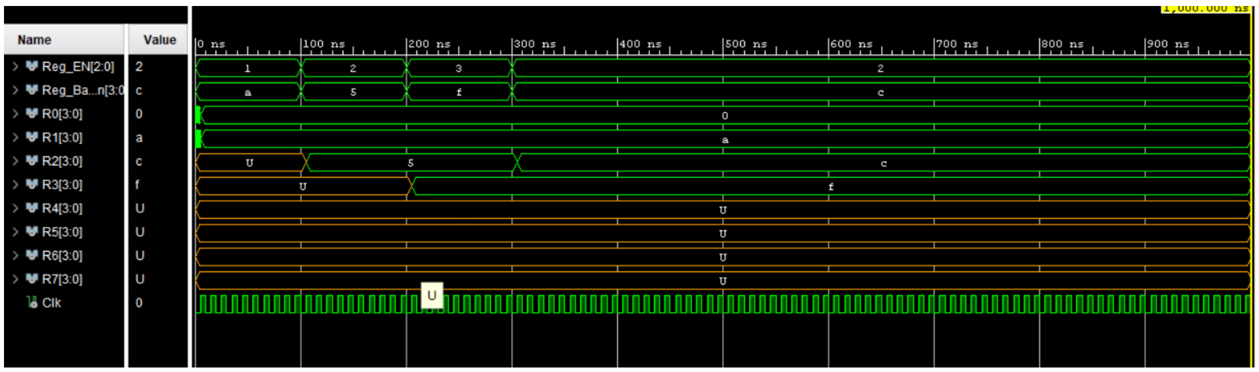
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Reg_bank is
    Port ( Reg_Bank_in   : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_EN        : in STD_LOGIC_VECTOR (2 downto 0);
           Clk          : in STD_LOGIC;
           R0     : out STD_LOGIC_VECTOR (3 downto 0);
           R1     : out STD_LOGIC_VECTOR (3 downto 0);
           R2     : out STD_LOGIC_VECTOR (3 downto 0);
           R3     : out STD_LOGIC_VECTOR (3 downto 0);
           R4     : out STD_LOGIC_VECTOR (3 downto 0);
           R5     : out STD_LOGIC_VECTOR (3 downto 0);
           R6     : out STD_LOGIC_VECTOR (3 downto 0);
           R7     : out STD_LOGIC_VECTOR (3 downto 0));
end Reg_bank;

architecture Behavioral of Reg_bank is

component Reg
    Port ( D    : in STD_LOGIC_VECTOR (3 downto 0);
           En   : in STD_LOGIC;
           Clk  : in STD_LOGIC;
           Q    : out STD_LOGIC_VECTOR (3 downto 0));
end component;

 --works as register enabling unit
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 in_Reg_EN : STD_LOGIC_VECTOR (7 downto 0);

begin

    Decoder_3_to_8_0 : Decoder_3_to_8
        port map (
        I(2 downto 0)    => Reg_EN(2 downto 0),
        EN               => '1',
        Y(7 downto 0)    => in_Reg_EN(7 downto 0));

    Reg_0 : Reg
        port map(
            D(3 downto 0)    => "0000", -- Reg_0 is readOnly
            En               => '1',
            Clk              => Clk,
            Q(3 downto 0)    => R0(3 downto 0));

    Reg_1 : Reg
        port map(
            D(3 downto 0)    => Reg_Bank_in(3 downto 0),
            En              => in_Reg_EN(1),
            Clk             => Clk,
            Q(3 downto 0)   => R1(3 downto 0));

    Reg_2 : Reg
        port map(
            D(3 downto 0)   => Reg_Bank_in(3 downto 0),
            En              => in_Reg_EN(2),
            Clk             => Clk,
            Q(3 downto 0)   => R2(3 downto 0));

    Reg_3 : Reg
        port map(
            D(3 downto 0)   => Reg_Bank_in(3 downto 0),
            En              => in_Reg_EN(3),
            Clk             => Clk,
            Q(3 downto 0)   => R3(3 downto 0));

    Reg_4 : Reg
        port map(
            D(3 downto 0)   => Reg_Bank_in(3 downto 0),
            En              => in_Reg_EN(4),
            Clk             => Clk,
            Q(3 downto 0)   => R4(3 downto 0));

    Reg_5 : Reg
        port map(
            D(3 downto 0)   => Reg_Bank_in(3 downto 0),
            En              => in_Reg_EN(5),
            Clk             => Clk,
            Q(3 downto 0)   => R5(3 downto 0));

    Reg_6 : Reg
        port map(
            D(3 downto 0)   => Reg_Bank_in(3 downto 0),
            En              => in_Reg_EN(6),
            Clk             => Clk,
            Q(3 downto 0)   => R6(3 downto 0));

    Reg_7 : Reg
        port map(
            D(3 downto 0)   => Reg_Bank_in(3 downto 0),
            En              => in_Reg_EN(7),
            Clk             => Clk,
            Q(3 downto 0)   => R7(3 downto 0));

end Behavioral;
```

TEST BENCH CODE

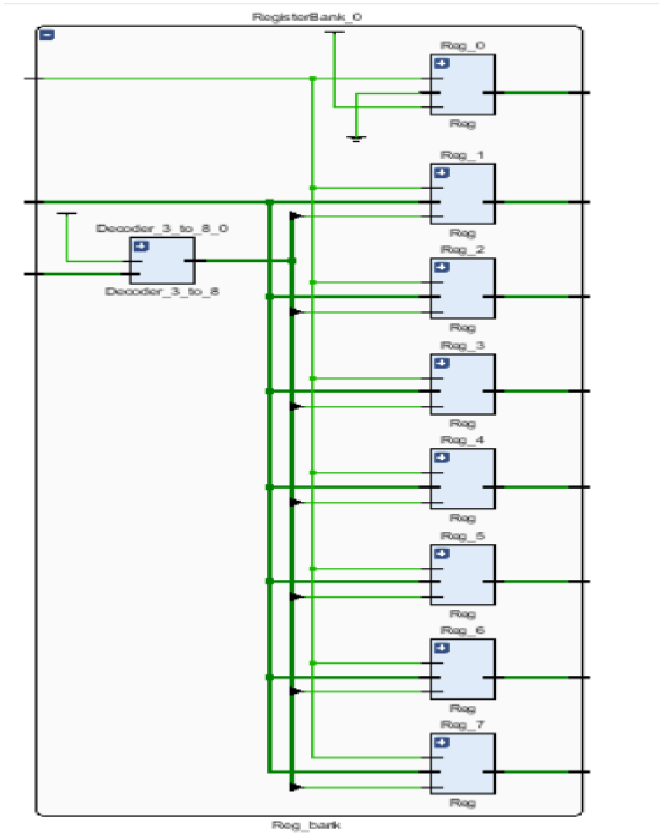
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

```
end TB_Reg_Bank;

architecture Behavioral of TB_Reg_Bank is


component Reg_Bank is
    Port ( Reg_Bank_in   : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_EN        : in STD_LOGIC_VECTOR (2 downto 0);
           Clk          : in STD_LOGIC;
           R0     : out STD_LOGIC_VECTOR (3 downto 0);
           R1     : out STD_LOGIC_VECTOR (3 downto 0);
           R2     : out STD_LOGIC_VECTOR (3 downto 0);
           R3     : out STD_LOGIC_VECTOR (3 downto 0);
           R4     : out STD_LOGIC_VECTOR (3 downto 0);
           R5     : out STD_LOGIC_VECTOR (3 downto 0);
           R6     : out STD_LOGIC_VECTOR (3 downto 0);
           R7     : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal Reg_EN : std_logic_vector(2 downto 0);
signal Reg_Bank_in, R0, R1, R2, R3, R4, R5, R6, R7 : std_logic_vector(3 downto
0);
signal Clk : std_logic:='0';

begin

UUT : Reg_Bank port map(
        Reg_Bank_in => Reg_Bank_in,
        Reg_EN => Reg_EN,
        Clk => Clk,
        R0 => R0,
        R1 => R1,
        R2 => R2,
        R3 => R3,
        R4 => R4,
        R5 => R5,
        R6 => R6,
        R7 => R7);

clock_process : process
begin
    wait for 5ns;
    CLk<=NOT(CLk);
end process;

sim : process
    begin
        Reg_Bank_in <= "1010";
        Reg_EN      <= "001";
        wait for 100ns;

        Reg_Bank_in <= "0101";
        Reg_EN      <= "010";
        wait for 100ns;

        Reg_Bank_in <= "1111";
        Reg_EN      <= "011";
        wait for 100ns;

        Reg_Bank_in <= "1100";
        Reg_EN      <= "010";

    wait;
end process;

end Behavioral;
```

TIMING DIAGRAM



RTL ANALYSIS - SCHEMATIC



REGISTER

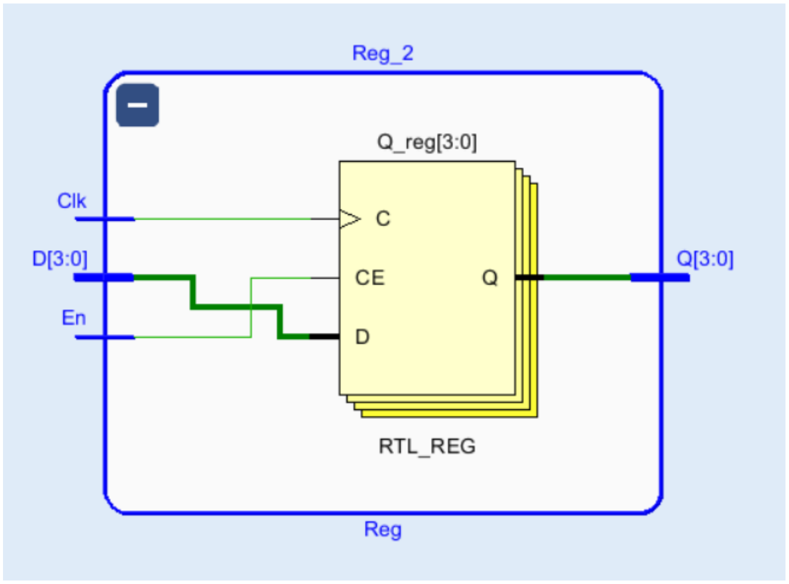
The existing register component is enhanced to include a Reset functionality. The Reset input is linked to the same push button utilized for the program counter. When the reset button is pressed, all registers within the system are initialized to store the value "0000". This ensures that a uniform reset state is achieved across all registers in response to the reset signal.

DESIGN SOURCE CODE

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

entity Reg is
    Port ( D     : in STD_LOGIC_VECTOR (3 downto 0);
           En    : in STD_LOGIC;
           Clk   : in STD_LOGIC;
           Q     : out STD_LOGIC_VECTOR (3 downto 0));
end Reg;

architecture Behavioral of Reg is

begin
```

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

end Behavioral;
```

RTL ANALYSIS - SCHEMATIC



ADD/SUBTRACT UNIT – 4 BITS

The Add/Subtract unit incorporates a ripple carry adder, previously designed using four full adders in lab 03. The numbers being processed are represented as signed integers, resulting in an output range of (-8) to 7. This unit can add two 4-bit numbers obtained from two 8-to-4 multiplexers. It also determines the carryout and zero conditions, indicated by the overflow and zero flags, respectively. The overflow flag is calculated by performing an XOR operation between the carry outs of the 3rd and 4th full adders. An overflow occurs if either of the carry outs is 1, while an overflow is absent if both carry outs are either 0 or 1. Moreover, the unit can compute the negation of a given 4-bit value. This is achieved by performing a bitwise XOR operation between the output of the mux A and Neg_Sel, which is set to 1 only in a negate instruction. The carry-in of the adder is also set as Neg_Sel. By adding the 1's complement of the value and 1, the add/subtract unit generates the 2's complement, which represents the negation.

DESIGN SOURCE CODE

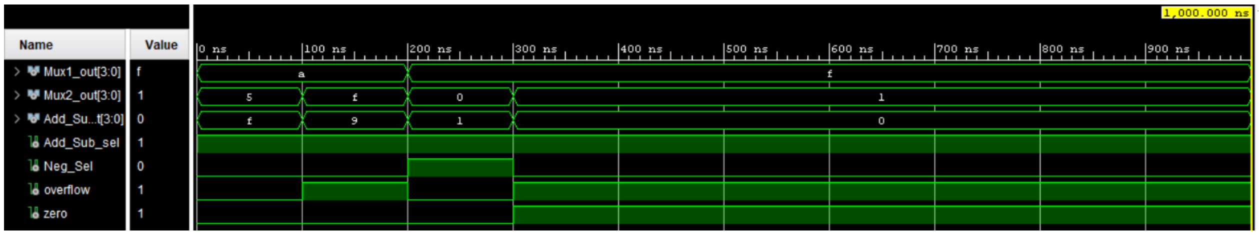
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Add_Sub_4 is
    Port ( Mux1_out     : in STD_LOGIC_VECTOR (3 downto 0);
           Mux2_out     : in STD_LOGIC_VECTOR (3 downto 0);
           Add_Sub_sel  : in STD_LOGIC;
           Neg_Sel       : in STD_LOGIC;
           Add_Sub_out  : out STD_LOGIC_VECTOR (3 downto 0);
           overflow     : out STD_LOGIC;
           zero         : out STD_LOGIC);     --zero flag :- '1' if output is
zero
end Add_Sub_4;

architecture Behavioral of Add_Sub_4 is

component RCA_4
    Port ( A0       : in STD_LOGIC;
           A1       : in STD_LOGIC;
           A2       : in STD_LOGIC;
           A3       : in STD_LOGIC;
           B0       : in STD_LOGIC;
           B1       : in STD_LOGIC;
           B2       : in STD_LOGIC;
           B3       : in STD_LOGIC;
           C_in     : in STD_LOGIC;
           S0       : out STD_LOGIC;
           S1       : out STD_LOGIC;
           S2       : out STD_LOGIC;
           S3       : out STD_LOGIC;
           C_out    : out STD_LOGIC);
end component;

signal FA0_in, FA1_in, FA2_in, FA3_in   : STD_LOGIC;

signal RCA_out       : STD_LOGIC_VECTOR (3 downto 0);
signal RCA_Carryout  : STD_LOGIC;

begin
    FA0_in <= Mux1_out(0) xor Neg_Sel;
    FA1_in <= Mux1_out(1) xor Neg_Sel;
    FA2_in <= Mux1_out(2) xor Neg_Sel;
    FA3_in <= Mux1_out(3) xor Neg_Sel;

    RCA_4_0 : RCA_4
        port map(
            A0      => FA0_in,
            A1      => FA1_in,
            A2      => FA2_in,
            A3      => FA3_in,
            B0      => Mux2_out(0),
            B1      => Mux2_out(1),
            B2      => Mux2_out(2),
            B3      => Mux2_out(3),
            C_in    => Neg_Sel,
            S0      => RCA_out(0),
            S1      => RCA_out(1),
            S2      => RCA_out(2),
            S3      => RCA_out(3),
            C_out   => overflow);

        process(Add_Sub_sel, RCA_out)
```

```
            begin

                if (Add_Sub_sel = '1') then
                    Add_Sub_out <= RCA_out;
                end if;

            end process;

        zero <= not(RCA_out(0) or RCA_out(1) or RCA_out(2) or RCA_out(3));

end Behavioral;
```

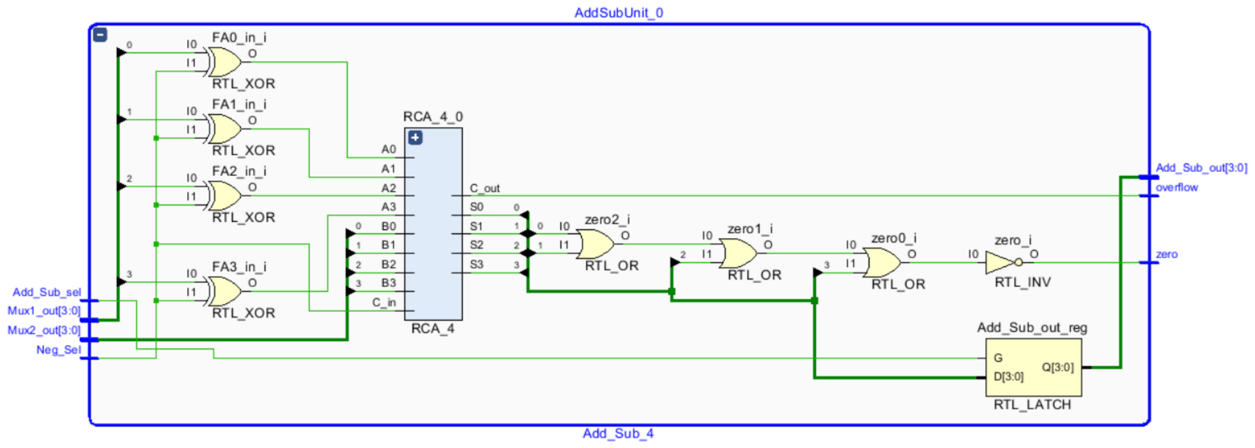
TEST BENCH CODE

```
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 is
    Port (
        Mux1_out     : in STD_LOGIC_VECTOR (3 downto 0);
        Mux2_out     : in STD_LOGIC_VECTOR (3 downto 0);
        Add_Sub_sel  : in STD_LOGIC;
        Neg_Sel      : in STD_LOGIC;
        Add_Sub_out  : out STD_LOGIC_VECTOR (3 downto 0);
        overflow     : out STD_LOGIC;
        zero         : out STD_LOGIC
    );
end component;

signal Mux1_out, Mux2_out, Add_Sub_out      : std_logic_vector(3 downto 0);
signal Add_Sub_sel, Neg_Sel, overflow, zero  : std_logic;

begin

UUT : Add_Sub_4 port map(
        Mux1_out     => Mux1_out,
        Mux2_out     => Mux2_out,
        Add_Sub_sel  => Add_Sub_sel,
        Neg_Sel      => Neg_Sel,
        Add_Sub_out  => Add_Sub_out,
        overflow     => overflow,
        zero         => zero);

process
    begin
        Mux1_out    <= "1010";
        Mux2_out    <= "0101";
        Add_Sub_sel <= '1';
        Neg_Sel     <= '0';
        wait for 100ns;

        Mux2_out     <= "1111";
        wait for 100ns;

        Mux1_out     <= "1111";
        Mux2_out     <= "0000";
        Neg_Sel      <= '1';

        wait for 100ns;

        Mux1_out     <= "1111";
        Mux2_out     <= "0001";
        Neg_Sel      <= '0';
        wait;
    end process;

end Behavioral;
```

TIMING DIAGRAM



RTL ANALYSIS - SCHEMATIC



ADDER – 3 BITS

The ripple carry adder employed in the system consists of three full adders. Its purpose is to increase the value of the PC-register by 1. The output of the adder is directed to a 2-way 3-bit multiplexer (mux). This mux is responsible for selecting the output of the adder as required by the system's operation.

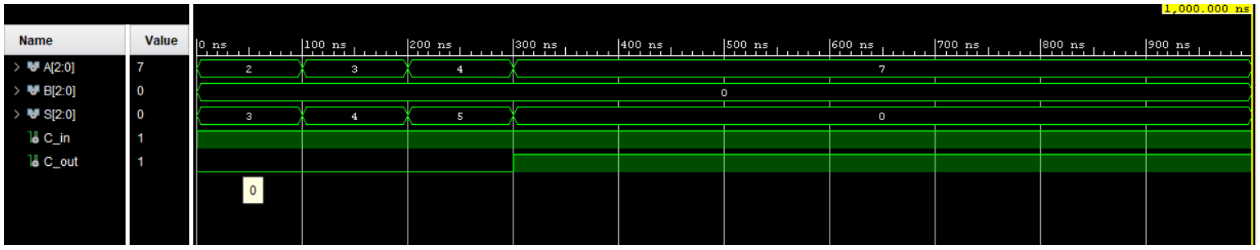
DESIGN SOURCE CODE

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

entity Adder_3 is
    Port ( A        : in STD_LOGIC_VECTOR(2 downto 0);
           B        : in STD_LOGIC_VECTOR (2 downto 0);
           C_in      : in STD_LOGIC;
           S        : out STD_LOGIC_VECTOR (2 downto 0);
           C_out     : out STD_LOGIC);
end Adder_3;

architecture Behavioral of Adder_3 is

    component FA
        port (
            A        : in std_logic;
            B        : in std_logic;
            C_in     : in std_logic;
            S        : out std_logic;
            C_out    : out std_logic);
    end component;

signal FA0_C, FA1_C          : std_logic;

begin

    FA_0 : FA
        port map (
            A => A(0),
            B => B(0),
            C_in => C_in,
            S => S(0),
            C_Out => FA0_C);

    FA_1 : FA
        port map (
            A => A(1),
            B => B(1),
            C_in => FA0_C,
            S => S(1),
            C_Out => FA1_C);

    FA_2 : FA
        port map (
            A => A(2),
            B => B(2),
            C_in => FA1_C,
            S => S(2),
            C_Out => C_Out);

end Behavioral;
```

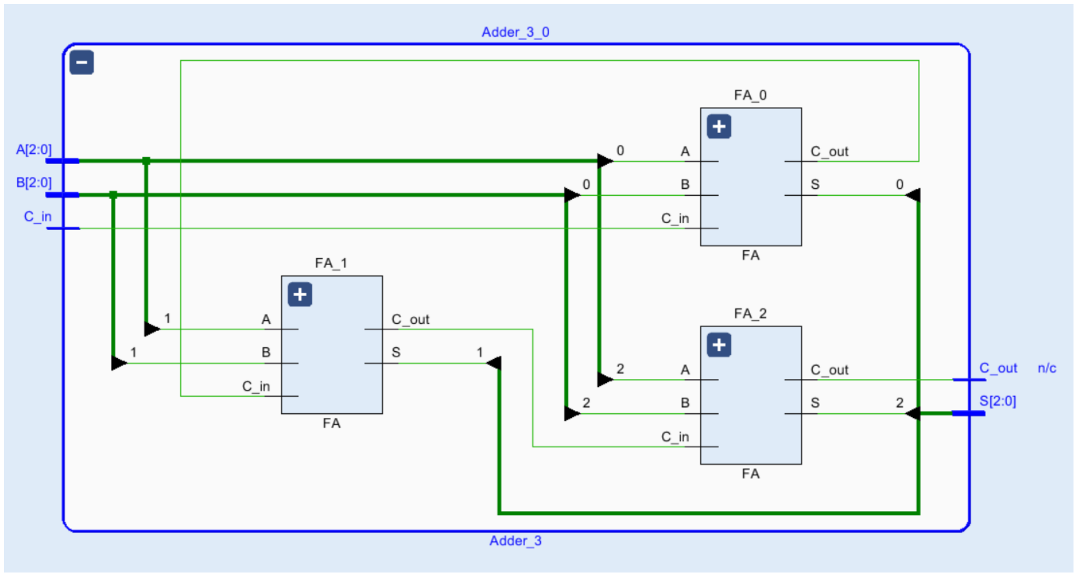
TEST BENCH CODE

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

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

architecture Behavioral of TB_Adder_3 is

component Adder_3 is
    Port ( A       : in STD_LOGIC_VECTOR (2 downto 0);

           B       : in STD_LOGIC_VECTOR (2 downto 0);

           C_in     : in STD_LOGIC;

           S       : out STD_LOGIC_VECTOR (2 downto 0);

           C_out    : out STD_LOGIC);
end component;

signal A,B,S : STD_LOGIC_VECTOR (2 downto 0);
signal C_in,C_out : STD_LOGIC;

begin

UUT : Adder_3 port map(
        A      => A,
        B      => B,
        C_in    => C_in,
        S      => S,
        C_out   => C_out);

process
    begin
        B <= "000";
        C_in <= '1';
        A <= "010";
        wait for 100ns;

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

        A <= "100";
        wait for 100ns;

        A <= "111";
    wait;
end process;
end Behavioral;
```

TIMING DIAGRAM



RTL ANALYSIS - SCHEMATIC



PROGRAM COUNTER

The program counter (PC) plays a crucial role in determining the next instruction to be executed. It continually increments the pc-register in response to clock pulses. Additionally, a 2-way 3-bit multiplexer is utilized to examine the status of the jump flag. If the jump flag is enabled, the pc-register will transition to the address specified by the instruction decoder. However, if the jump flag is not enabled, the pc-register will be set to the incremented address, ensuring the sequential execution of instructions.

DESIGN SOURCE CODE

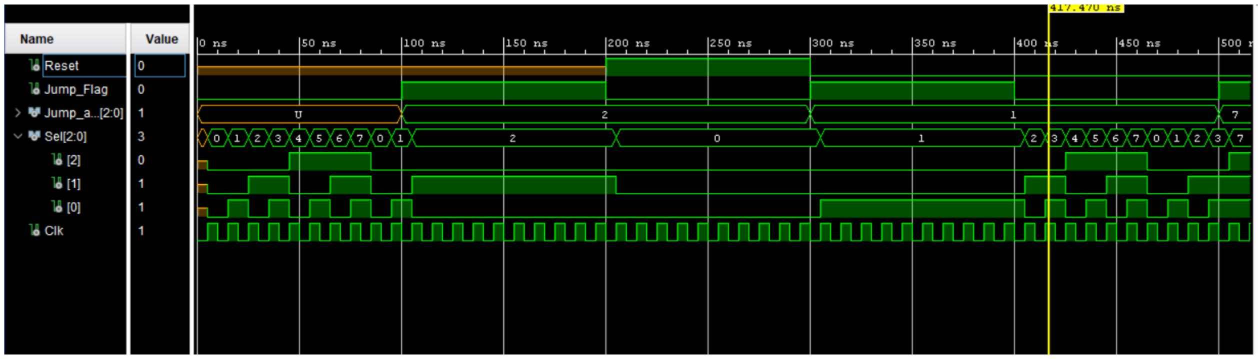
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PC is
    Port ( Clk       : in STD_LOGIC;
           Reset     : in STD_LOGIC;
           Jump_Flag : in STD_LOGIC;
           Jump_address   : in STD_LOGIC_VECTOR (2 downto 0);
           Sel    : out STD_LOGIC_VECTOR (2 downto 0));
end PC;
```

```
architecture Behavioral of PC is

component PC_Reg is
    Port (    Reg_in     : in STD_LOGIC_VECTOR (2 downto 0);
              reset     : in STD_LOGIC;
              Clk   : in STD_LOGIC;
              Reg_out     : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component Adder_3 is
    Port ( A      : in STD_LOGIC_VECTOR(2 downto 0);

           B      : in STD_LOGIC_VECTOR(2 downto 0);

           C_in     : in STD_LOGIC;

           S      : out STD_LOGIC_VECTOR(2 downto 0);

           C_out    : out STD_LOGIC);
end component;

component Mux_2_way_3 is
    Port ( Adder_out  : in STD_LOGIC_VECTOR (2 downto 0);
           JMP_address  : in STD_LOGIC_VECTOR (2 downto 0);
           JMP_Flag     : in STD_LOGIC;
           mux_out      : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal Mux_in            : std_logic_vector(2 downto 0);
signal Ins_Next           : std_logic_vector(2 downto 0);
signal Ins_Curr           : std_logic_vector(2 downto 0);

begin

    PC_Reg_0 : PC_Reg
        port map(
            Reg_in   => Ins_Next,
            reset   => Reset,
            Clk => Clk,
            Reg_out   => Ins_Curr);

    Adder_3_0 : Adder_3
        port map(
            A      => Ins_Curr,
            B      => "000",
            C_in    => '1',
            S    => Mux_in
            );
    Mux_2_way_3_0 : Mux_2_way_3
        port map(
            Adder_out => Mux_in,
            JMP_address => Jump_address,
            JMP_Flag    => JUMP_Flag,
            mux_out     => Ins_Next);
    Sel <= Ins_Curr;

end Behavioral;
```

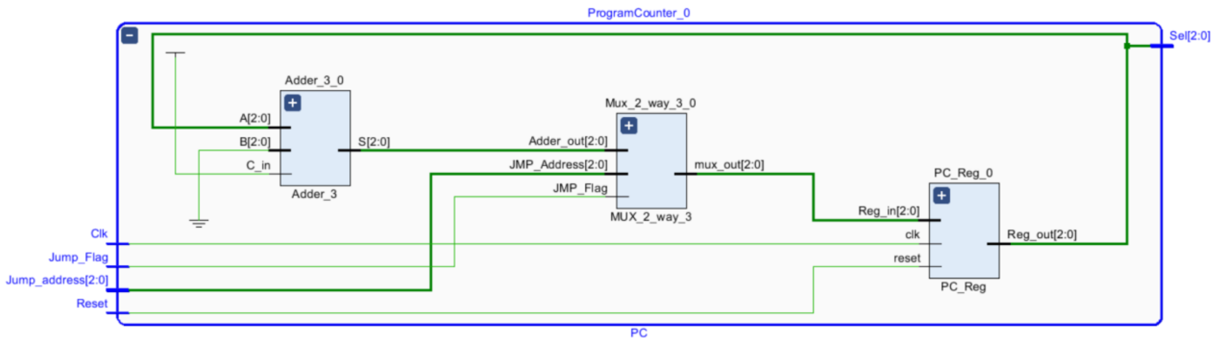
TEST BENCH CODE

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

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

architecture Behavioral of TB_PC is

component PC is
    Port (
        Clk             : in STD_LOGIC;
        Reset           : in STD_LOGIC;
        Jump_Flag       : in STD_LOGIC;
        Jump_address    : in STD_LOGIC_VECTOR (2 downto 0);
        Sel             : out STD_LOGIC_VECTOR (2 downto 0)
    );
end component;

signal Reset, Jump_Flag : std_logic;
signal Jump_address, Sel : std_logic_vector(2 downto 0);
signal Clk : std_logic := '0';


begin

UUT : PC port map (
        Clk             => Clk,
        Reset           => Reset,
        Jump_Flag       => Jump_Flag,
        Jump_address    => Jump_address,
        Sel             => Sel
);

process
begin
    wait for 5ns;
    Clk <= NOT(Clk);
end process;
stimulus_process : process
    begin
        Jump_Flag <= '0';
        wait for 100ns;

        Jump_Flag <= '1';
        Jump_Address  <= "010";
        wait for 100ns;

        Jump_Flag <= '0';
        Reset <= '1';
        wait for 100ns;
        Reset <= '0';

        Jump_Flag <= '1';
        Jump_Address  <= "001";
        wait for 100ns;

        Jump_Flag <= '0';
        wait for 100ns;

        Jump_Flag <= '1';
        Jump_Address  <= "111";
    wait;
end process;

end Behavioral;
```

TIMING DIAGRAM



RTL ANALYSIS - SCHEMATIC



K-WAY B-BIT MUX

The design of the system was built upon the multiplexers implemented in lab 4. The key distinction lies in the utilization of k-way b-bit multiplexers, which incorporate k input buses and one output bus containing b bits. Each multiplexer possesses a path selector determined by the value of k. This design choice allows for efficient selection and routing of data within the system, enabling the proper flow of information between components.

8-WAY-4-BIT MUX

DESIGN SOURCE CODE

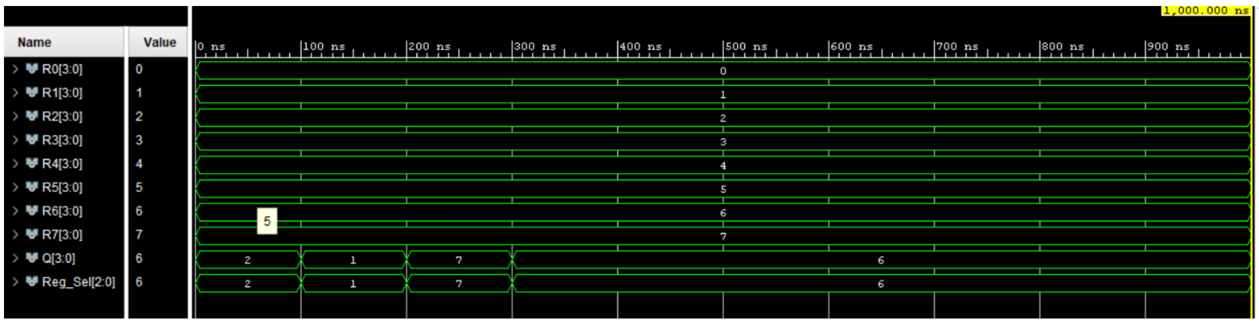
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_8_way_4 is
    Port ( R0       : in STD_LOGIC_VECTOR (3 downto 0);
           R1       : in STD_LOGIC_VECTOR (3 downto 0);
           R2       : in STD_LOGIC_VECTOR (3 downto 0);
           R3       : in STD_LOGIC_VECTOR (3 downto 0);
           R4       : in STD_LOGIC_VECTOR (3 downto 0);
           R5       : in STD_LOGIC_VECTOR (3 downto 0);
           R6       : in STD_LOGIC_VECTOR (3 downto 0);
           R7       : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_sel   : in STD_LOGIC_VECTOR (2 downto 0);
           Q        : out STD_LOGIC_VECTOR (3 downto 0));
end MUX_8_way_4;

architecture Behavioral of MUX_8_way_4 is

begin

    process(R0, R1, R2, R3, R4, R5, R6, R7, Reg_sel)
        begin
            case Reg_sel is

                when "000" => Q <= R0;
                when "001" => Q <= R1;
                when "010" => Q <= R2;
                when "011" => Q <= R3;
                when "100" => Q <= R4;
                when "101" => Q <= R5;
                when "110" => Q <= R6;
                when "111" => Q <= R7;
                when others => Q <= "0000";

            end case;
        end process;

end Behavioral;
```

TEST BENCH CODE

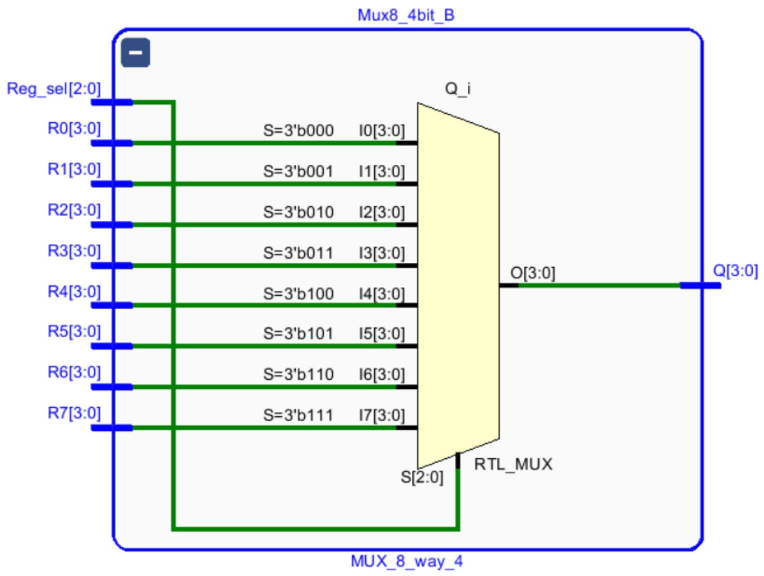
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TB_MUX_8_way_4 is
--  Port ( );
end TB_MUX_8_way_4;
```

```
architecture Behavioral of TB_MUX_8_way_4 is

component MUX_8_way_4 is
        Port ( R0       : in STD_LOGIC_VECTOR (3 downto 0);
       R1       : in STD_LOGIC_VECTOR (3 downto 0);
       R2       : in STD_LOGIC_VECTOR (3 downto 0);
       R3       : in STD_LOGIC_VECTOR (3 downto 0);
       R4       : in STD_LOGIC_VECTOR (3 downto 0);
       R5       : in STD_LOGIC_VECTOR (3 downto 0);
       R6       : in STD_LOGIC_VECTOR (3 downto 0);
       R7       : in STD_LOGIC_VECTOR (3 downto 0);
       Reg_sel   : in STD_LOGIC_VECTOR (2 downto 0);
       Q        : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal R0, R1, R2, R3, R4, R5, R6, R7, Q : std_logic_vector(3 downto 0);
signal Reg_Sel                            : std_logic_vector(2 downto 0);

begin

UUT : MUX_8_way_4 port map(
        R0      => R0,
        R1      => R1,
        R2      => R2,
        R3      => R3,
        R4      => R4,
        R5      => R5,
        R6      => R6,
        R7      => R7,
        Reg_sel  => Reg_sel,
        Q       => Q);

process
    begin
        R0 <= "0000";
        R1 <= "0001";
        R2 <= "0010";
        R3 <= "0011";
        R4 <= "0100";
        R5 <= "0101";
        R6 <= "0110";
        R7 <= "0111";
        Reg_sel <= "010";
        wait for 100ns;

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

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

        Reg_sel <= "110";
        wait for 100ns;
    wait;
end process;
end Behavioral;
```

TIMING DIAGRAM



RTL ANALYSIS - SCHEMATIC



2-WAY-4-BIT MUX

DESIGN SOURCE CODE

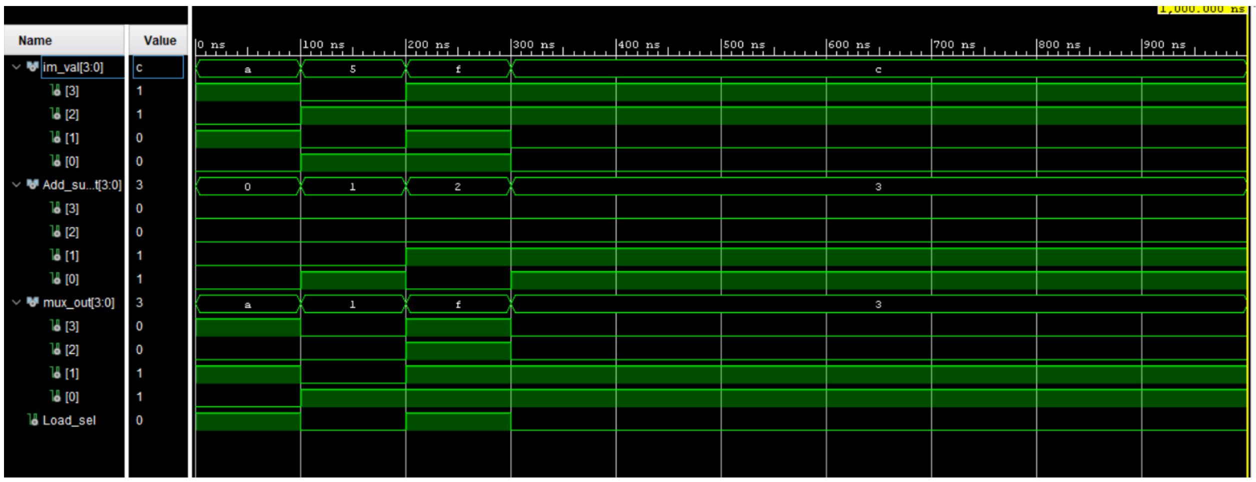
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux_2_way_4 is
    Port ( Im_val           : in STD_LOGIC_VECTOR (3 downto 0);
           Add_Sub_Out    : in STD_LOGIC_VECTOR (3 downto 0);
           Load_Sel         : in STD_LOGIC; --Load_Sel <= Mov
           mux_out          : out STD_LOGIC_VECTOR (3 downto 0));
end Mux_2_way_4;
```

```
architecture Behavioral of Mux_2_way_4 is

begin

    process(Im_val, Add_Sub_Out, Load_Sel)
        begin
            case Load_Sel is

                when '0'        => mux_out <= Add_Sub_Out;
                when '1'        => mux_out <= Im_val;
                when others     => mux_out <= "0000";

            end case;
        end process;

end Behavioral;
```

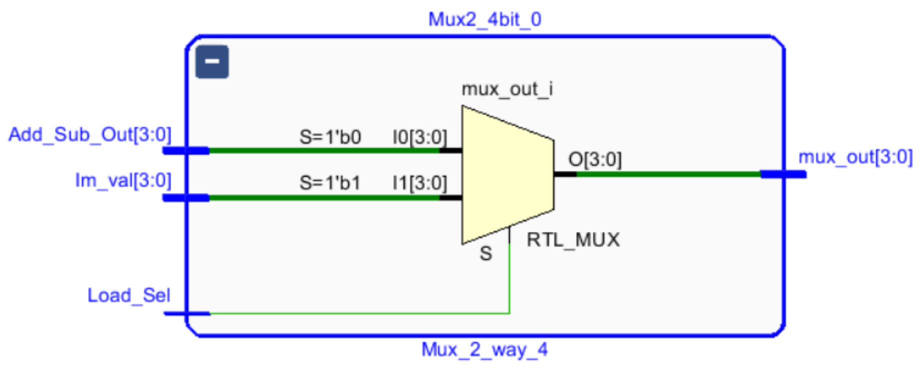
TEST BENCH CODE

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

entity TB_MUX_2_way_4 is
--  Port ( );
end TB_MUX_2_way_4;

architecture Behavioral of TB_MUX_2_way_4 is

component Mux_2_way_4 is
    Port ( im_val           : in STD_LOGIC_VECTOR (3 downto 0);
           Add_sub_out    : in STD_LOGIC_VECTOR (3 downto 0);
           Load_sel           : in STD_LOGIC;
           mux_out          : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal im_val, Add_sub_out, mux_out   : std_logic_vector(3 downto 0);
signal Load_sel                          : std_logic;

begin

UUT : Mux_2_way_4 port map(
        im_val          => im_val,
        Add_sub_out   => Add_sub_out,
        Load_sel         => Load_sel,
        mux_out         => mux_out);

process
    begin
        -- Test case 1
        im_val    <= "1010";
        Load_sel  <= '1';
        Add_sub_out <= "0000";
        wait for 100 ns;

        -- Test case 2
        im_val    <= "0101";
        Load_sel  <= '0';
        Add_sub_out <= "0001";
        wait for 100 ns;

        -- Test case 3
        im_val    <= "1111";
        Load_sel  <= '1';
        Add_sub_out <= "0010";
        wait for 100 ns;

        -- Test case 4
        im_val    <= "1100";
        Load_sel  <= '0';
        Add_sub_out <= "0011";
        wait for 100 ns;

        wait;
    end process;


end Behavioral;
```

TIMING DIAGRAM



RTL ANALYSIS - SCHEMATIC



2-WAY-3-BIT MUX

DESIGN SOURCE CODE

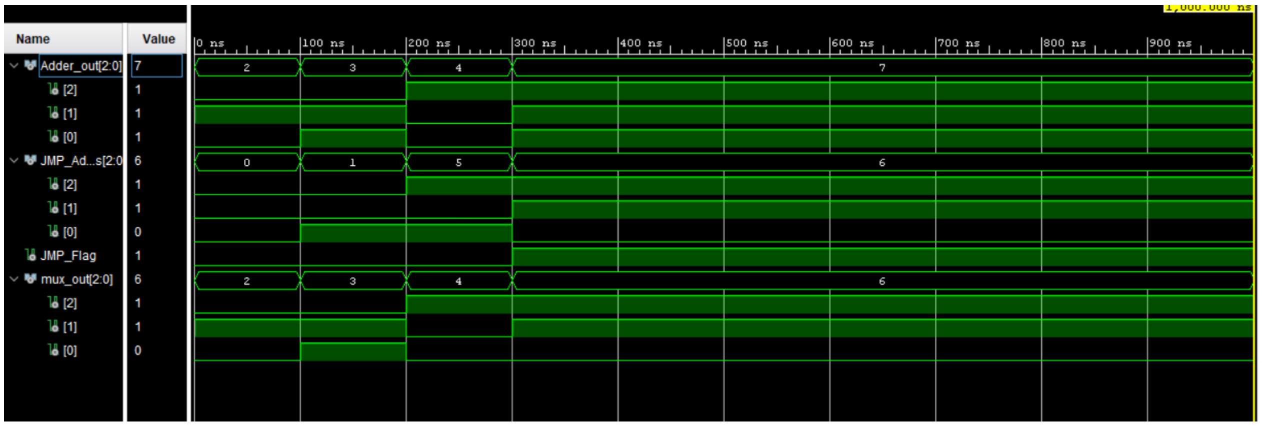
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_2_way_3 is
    Port ( Adder_out   : in STD_LOGIC_VECTOR (2 downto 0);
           JMP_Address  : in STD_LOGIC_VECTOR (2 downto 0);
           JMP_Flag     : in STD_LOGIC;
           mux_out      : out STD_LOGIC_VECTOR (2 downto 0));
end MUX_2_way_3;

architecture Behavioral of MUX_2_way_3 is

begin

    process(Adder_out, JMP_Address, JMP_Flag)
        begin
            case JMP_Flag is

                when '1'    => mux_out <= JMP_Address;
                when '0'    => mux_out <= Adder_out;
                when others => mux_out <= "000";

            end case;
        end process;

end Behavioral;
```

TEST BENCH CODE

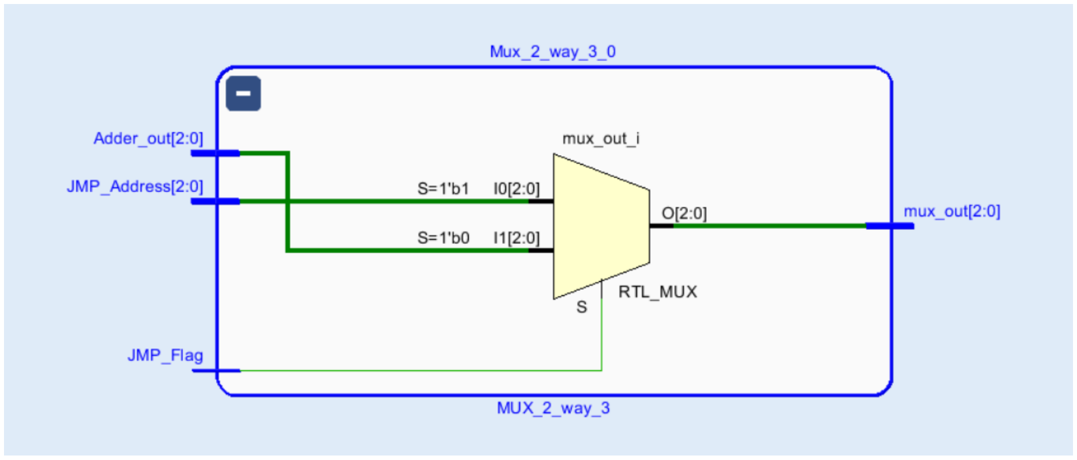
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity TB_MUX_2_way_3 is
end TB_MUX_2_way_3;

architecture Behavioral of TB_MUX_2_way_3 is
    component MUX_2_way_3 is
        Port (
            Adder_out   : in STD_LOGIC_VECTOR (2 downto 0);
            JMP_Address : in STD_LOGIC_VECTOR (2 downto 0);
            JMP_Flag    : in STD_LOGIC;
            mux_out     : out STD_LOGIC_VECTOR (2 downto 0)
        );
    end component;

    signal Adder_out, JMP_Address : STD_LOGIC_VECTOR (2 downto 0);
    signal JMP_Flag              : STD_LOGIC;
    signal mux_out               : STD_LOGIC_VECTOR (2 downto 0);
begin
    UUT : MUX_2_way_3
        port map (
            Adder_out   => Adder_out,
            JMP_Address => JMP_Address,
            JMP_Flag    => JMP_Flag,
            mux_out     => mux_out
        );

    stimulus_process : process
    begin
        -- Test case 1
        Adder_out   <= "010";
        JMP_Address <= "000";
        JMP_Flag    <= '0';
        wait for 100 ns;

        -- Test case 2
        Adder_out   <= "011";
        JMP_Address <= "001";
        JMP_Flag    <= '0';
        wait for 100 ns;

        -- Test case 3
        Adder_out   <= "100";
        JMP_Address <= "101";
        JMP_Flag    <= '0';
        wait for 100 ns;

        -- Test case 4
        Adder_out   <= "111";
        JMP_Address <= "110";
        JMP_Flag    <= '1';
        wait for 100 ns;

        wait;
    end process;
end Behavioral;
```

TIMING DIAGRAM



RTL ANALYSIS - SCHEMATIC



XDC 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]}]

##7 segment display
set_property PACKAGE_PIN W7 [get_ports {Out_7Seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Out_7Seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {Out_7Seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Out_7Seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {Out_7Seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Out_7Seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {Out_7Seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Out_7Seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {Out_7Seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Out_7Seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {Out_7Seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Out_7Seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {Out_7Seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Out_7Seg[6]}]

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

CONCLUSION

The lab project involved designing a nano processor capable of executing a set of simple instructions. It incorporated components such as an Add/Subtract unit, a Program Counter, an Instruction Decoder, and a Register Bank. By leveraging pre-developed components from previous labs, the integration process was made smoother. Through simulations, the functionality of the components and the overall processor was verified. The project provided an opportunity to enhance skills in communication, coordination, task distribution, and component integration.