Lab 9-10 – Nanoprocessor Design Competition

CS1050 Computer Organization and Digital Design

Dept. of Computer Science and Engineering, University of Moratuwa

Group --34

- Waduge S.S.- 220673K
- Wijesooriya J.M.I. – 220724U
- Gallage A.H. –220172A
- Wijeratne E.N.K. – 220717C

Lab task and introduction

The main objective of this lab is to design and implement a simple 4-bit processor, referred to as a "nano processor", that can execute a basic set of instructions. This involves designing and integrating several key components:

- 4-bit Arithmetic Unit: Develop an add/subtract unit capable of performing signed integer arithmetic.
- Program Counter: Implement a 3-bit program counter with reset functionality.
- Multiplexers: Design 2-way 3-bit, 2-way 4-bit, and 8-way 4-bit multiplexers.
- Register Bank: Implement an 8-register, 4-bit wide register bank, with one register hardwired to 0.
- Instruction Decoder: Develop the logic to decode the provided 12-bit instruction set and activate the necessary components.
- Program ROM: Implement a ROM to store the assembly program that will be executed.

A team of 4 students were tasked to design, implement, and test these components. we verified the functionality through simulation and finally test the overall nano processor on the BASYS 3 development board.

The key tasks include:

- Designing the internal logic of the instruction decoder
- Building and testing the various sub-components
- Integrating the components into the top-level nano processor design
- Writing an assembly program, converting it to machine code, and hardcoding it into the program ROM

- Connecting the inputs/outputs and verifying the nano processor's operation on the BASYS 3 board.

Assembly Code

```
MOVI R1,01 – Move 01 value to R1
MOVI R2,02 – Move 02 value to R2
ADD R1, R2 – ADD Values in R1 and R2 then save the answer in to R1
MOVI R3,03 – Move 03 value to R3
SUB R3,R1 --  Sub  R1 from R3 then save it to R2
```

Machine Code

"0000010000001" -- Load 1 int register 1

"0000100000010" -- Load 2 into register 2

"0010010100000" -- Add the contents of registers 1 and 2,

"0000110000101" -- Load 5 int register 3

"0001000000011" -- Load 3 int register 4

Contents

Base Model

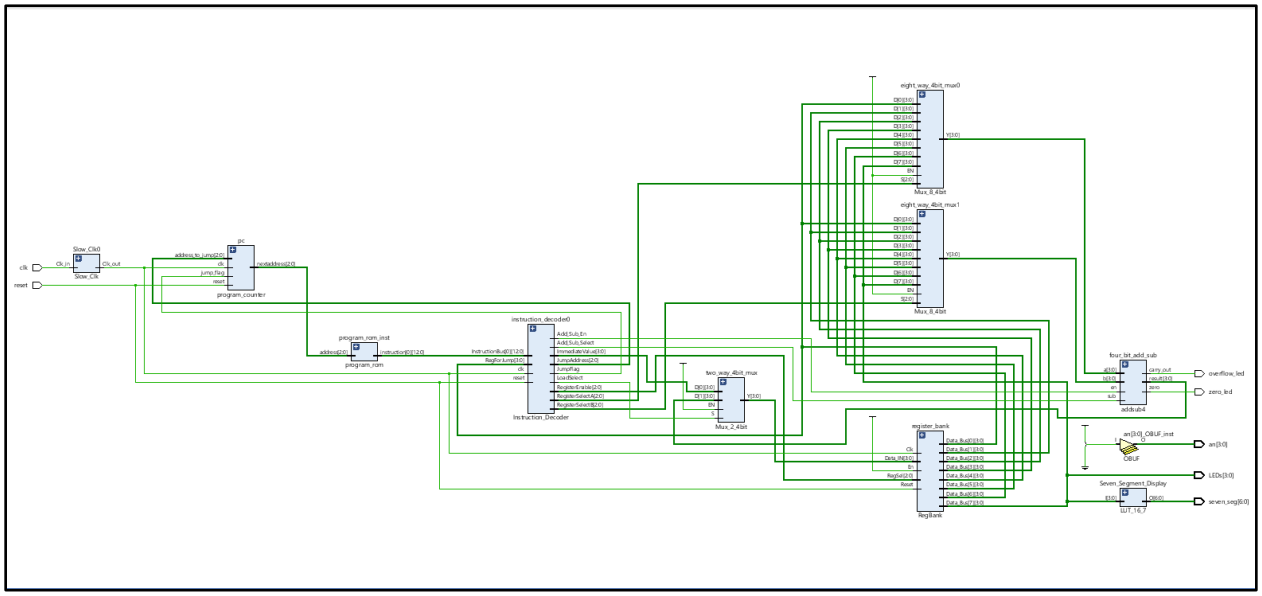
- **Description**

This is a nano processor with only 4-bit adder and subtractor. The components are Seven segment Display, Slow clock, 4bit Adder/Subtractor, program counter, 2-way 4 bit mux, 8-way 4 bit mux, Register Bank, Instruction Decoder, Program rom.

Components

Nano Processor

Elaborated Design Schematic



Design Source Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use WORK.Bus_System.ALL;

entity nanoprocessor is
    port (
        clk : in std_logic;
        reset : in std_logic;
```

```
        overflow_led : out std_logic;
        zero_led : out std_logic;
        seven_seg:out std_logic_vector(6 downto 0);
        LEDs :out std_logic_vector(3 downto 0);
        an : out std_logic_vector(3 downto 0):="1110"



    );
end entity nanoprocessor;

architecture rtl of nanoprocessor is
    component LUT_16_7 is
        port(
            I : in STD_LOGIC_VECTOR (3 downto 0);
           O : out STD_LOGIC_VECTOR (6 downto 0)
           );
    end component;

    component slow_Clk is
        port(
        Clk_in : in STD_LOGIC;
        Clk_out : out STD_LOGIC
        );
    end component;
    component addsub4 is
        port (
        a : in std_logic_vector(3 downto 0);
        b : in std_logic_vector(3 downto 0);
        sub : in std_logic;
        en: in std_logic;
        result : out std_logic_vector(3 downto 0);
        carry_out : out std_logic;
        zero:out std_logic

    );
    end component;

    component adder3 is
        port (
        a : in std_logic_vector(2 downto 0);
        b : in std_logic_vector(2 downto 0);
        result : out std_logic_vector(2 downto 0);
        carry_out : out std_logic
    );
```

```
    end component;


    component program_counter is
        port (
        clk :in std_logic;
        reset : in std_logic;
--        currentaddress: in STD_LOGIC_VECTOR(2 downto 0);
        nextaddress : out std_logic_vector(2 downto 0);
        jump_flag : in std_logic;
        address_to_jump :  in std_logic_vector(2 downto 0)
    );
    end component;



    component Mux_2_4bit is
        port (D : in BUS_2_4bit;
           S : in STD_LOGIC;
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR(3 downto 0));
    end component;

    component Mux_8_4bit is
        port ( D : in BUS_8_4bit;
           S : in STD_LOGIC_VECTOR(2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR(3 downto 0));
    end component;

    component RegBank is
        Port ( RegSel :  in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           En : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Data_IN : in STD_LOGIC_VECTOR(3 downto 0);
           Data_Bus : out BUS_8_4bit);
    end component;

    component instruction_decoder is
        Port (
        clk : in std_logic;
        reset : in std_logic;
        InstructionBus : in BUS_13bit;
        RegForJump : in STD_LOGIC_VECTOR(3 downto 0);
```

```
        RegisterEnable :out STD_LOGIC_VECTOR (2 downto 0);
        LoadSelect : out std_logic;
        ImmediateValue : out std_logic_vector(3 downto 0);
        RegisterSelectA : out std_logic_vector(2 downto 0);
        RegisterSelectB : out std_logic_vector(2 downto 0);
        Add_Sub_Select : out std_logic;
        JumpFlag : out std_logic;
        JumpAddress : out std_logic_vector(2 downto 0);
        Add_Sub_En : out std_logic


    );
    end component;

    component program_rom is
        port (
            address : in std_logic_vector(2 downto 0);
            instruction : out BUS_13bit
        );
    end component;

    signal muxoutput0 : std_logic_vector(3 downto 0);
    signal muxoutput1: std_logic_vector(3 downto 0);
    signal instruction_bus : BUS_13bit;

    signal reg_enable : STD_LOGIC_VECTOR (2 downto 0);
    signal add_sub_select : std_logic;
    signal mux3_select : bus_1_4bit;
    signal mux33_select : std_logic;
    signal load_select: std_logic;
    signal jump_flag : std_logic;
    signal zero_flag : std_logic;
    signal reg_a_data : bus_1_4bit;
    signal reg_b_data : bus_1_4bit;
    signal alu_result : bus_1_4bit;
    signal alu_overflow : std_logic;
    signal pc_next : std_logic_vector(2 downto 0);
    --signal pc_current :std_logic_vector(2 downto 0);
    signal regA:std_logic_vector(2 downto 0);
    signal regB:std_logic_vector(2 downto 0);
    signal jumpAddress:std_logic_vector(2 downto 0);
    signal regdata: bus_1_4bit;
    signal Databus0 :std_logic_vector(3 downto 0);
    signal regoutput: BUS_8_4bit;
    signal mux_2_4bus : BUS_2_4bit;
```

```
    signal mux_2_3bus : BUS_2_3bit;
    signal slw_clk : STD_LOGIC;
    signal seven_seg_out : std_logic_vector(6 downto 0);
    signal seven_seg_in : STD_LOGIC_VECTOR (3 downto 0);
    signal compare0 : std_logic_vector(2 downto 0);
    signal add_sub_enable:std_logic;
    signal comparator_enable : std_logic;
begin
    Seven_Segment_Display : LUT_16_7
    port map(
           I =>seven_seg_in,
           O =>seven_seg_out
           );

    Slow_Clk0 : slow_Clk
    port map(
        Clk_in => clk,
        Clk_out => slw_clk
        );


    four_bit_add_sub : addsub4
        port map (
            a =>muxoutput0,
            b =>muxoutput1,
            sub =>add_sub_select,
            en=>add_sub_enable,
            result =>mux_2_4bus(1),
            carry_out =>alu_overflow,
            zero=>zero_flag
        );



    pc : program_counter
        port map (
            clk=> slw_clk ,
            reset => reset ,
            nextaddress => pc_next,
            jump_flag => jump_flag ,
            address_to_jump => jumpAddress
        );

    two_way_4bit_mux : Mux_2_4bit
        port map
```

```
      ( D =>mux_2_4bus,
        S =>load_select,
        EN=>'1',
        Y => databus0);


eight_way_4bit_mux0 : Mux_8_4bit
    port map (
       D => regoutput,
       S =>regA,
       EN =>'1',
       Y=>muxoutput0
    );

   eight_way_4bit_mux1 : Mux_8_4bit
    port map (
       D => regoutput,
       S =>regB,
       EN =>'1',
       Y=>muxoutput1
    );

register_bank : RegBank
    port map (
       RegSel => reg_enable,
       Clk =>slw_clk,
       En =>'1',
       Reset =>reset,
       Data_IN =>databus0,
       Data_Bus =>regoutput
       );

instruction_decoder0 : instruction_decoder
    port map (
    clk =>slw_clk,
    reset=>reset,
    InstructionBus=>instruction_bus,
    RegForJump =>regoutput(0),
    RegisterEnable=>reg_enable,
    LoadSelect =>load_select,
    ImmediateValue =>mux_2_4bus(0),
    RegisterSelectA=>regA,
    RegisterSelectB=>regB,
    Add_Sub_Select =>add_sub_select,
    JumpFlag =>jump_flag,
```

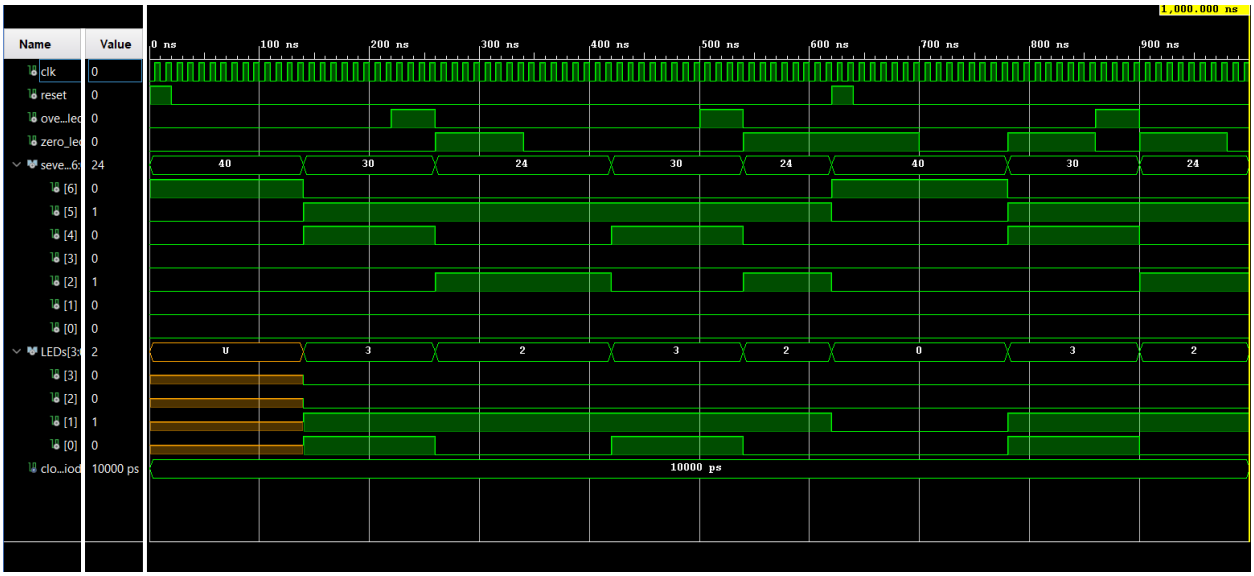
```
        JumpAddress =>jumpAddress,
        Add_Sub_En=>add_sub_enable
        );

    program_rom_inst : program_rom
        port map (
            address => pc_next,
            instruction => instruction_bus
        );

    overflow_led <= alu_overflow;
    zero_led <= zero_flag;
    seven_seg_in <= regoutput(7);

    LEDs <= regoutput(7);
    seven_seg <= seven_seg_out;



end architecture rtl;
```

VHDL code for simulation file

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.bus_system.all;

entity nanoprocessor_tb is
end entity nanoprocessor_tb;

architecture tb of nanoprocessor_tb is
    component nanoprocessor is
        port (
            clk : in std_logic;
            reset : in std_logic;
            overflow_led : out std_logic;
            zero_led : out std_logic;
            seven_seg : out std_logic_vector(6 downto 0);
            LEDs : out std_logic_vector(3 downto 0);
            an : out std_logic_vector(3 downto 0):="1110"
```

```
        );
    end component nanoprocessor;

    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal overflow_led : std_logic;
    signal zero_led : std_logic;
    signal seven_seg : std_logic_vector(6 downto 0);
    signal LEDs : std_logic_vector(3 downto 0);

    constant clock_period : time := 10 ns;
begin
    uut : nanoprocessor
        port map (
            clk => clk,
            reset => reset,
            overflow_led => overflow_led,
            zero_led => zero_led,
            seven_seg => seven_seg,
            LEDs => LEDs



        );

    clk_process : process
    begin
        clk <= '0';
        wait for clock_period / 2;
        clk <= '1';
        wait for clock_period / 2;
    end process;

    Reset_Process : PROCESS BEGIN
        Reset <= '1';
        WAIT FOR 20 NS;
        Reset <= '0';
        WAIT FOR 600 NS;
        Reset <= '1';
    END PROCESS;
end architecture tb;
```

Time simulation for nano processor



Slow Clock

Building and testing a nano processor using the Vivado software, the slow clock would refer to a clock signal with a reduced frequency that is utilized during the development and testing process. We set count ==2 when we test in Vivado and when we test in board, we set count == 50 million.

VHDL code for design 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_state : STD_LOGIC := '0';

begin
    PROCESS (Clk_in)
        BEGIN
            IF (rising_edge(Clk_in)) THEN
                count <= count+1;
                IF (count = 2) THEN  --count 50M pulses (1/2 of period) ; 100 MHz
/ 2 = 50 MHz
                    clk_state <= NOT clk_state;
                    count <= 1;
                END IF;
            END IF;
            Clk_out <= clk_state;
    END PROCESS;

end Behavioral;
```

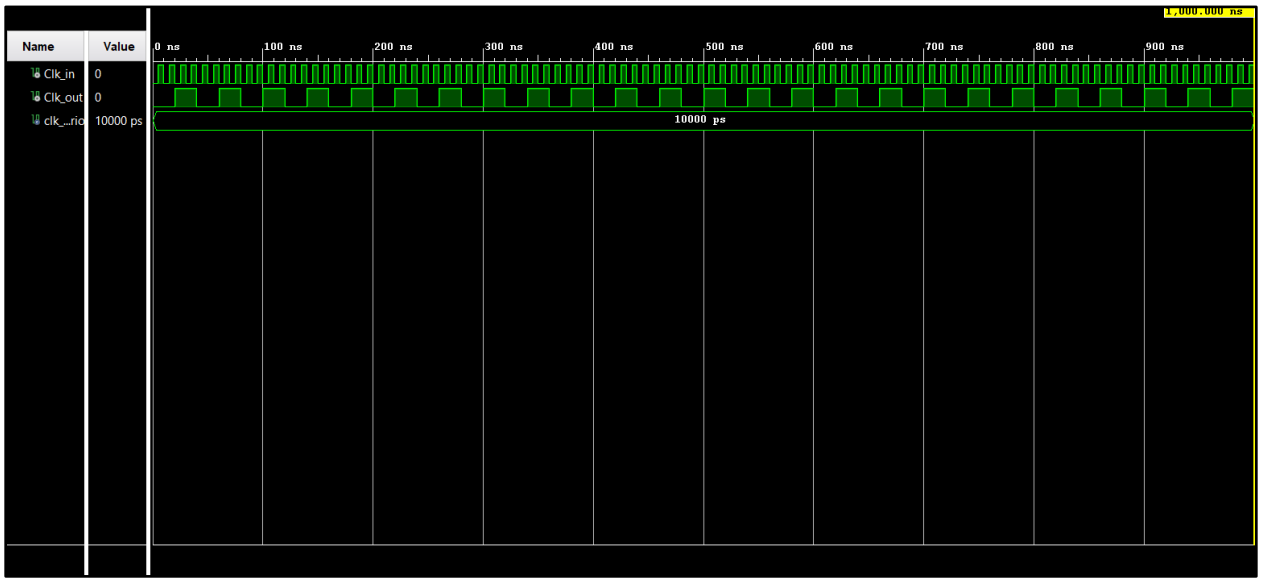
VHDL code for simulation file

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

entity Slow_Clk_tb is
```

```
end Slow_Clk_tb;

architecture Behavioral of Slow_Clk_tb is

    -- Component Declaration for the Unit Under Test (UUT)
    component Slow_Clk
        Port ( Clk_in : in STD_LOGIC;
               Clk_out : out STD_LOGIC);
    end component;

    -- Input signal declaration
    signal Clk_in : std_logic := '0';

    -- Output signal declaration
    signal Clk_out : std_logic;

    -- Clock period definitions
    constant clk_period : time := 10 ns; -- 100 MHz clock

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: Slow_Clk port map (
        Clk_in => Clk_in,
        Clk_out => Clk_out
    );

    -- Clock process definition
    clk_process: process
    begin
        Clk_in <= '0';
        wait for clk_period/2;
        Clk_in <= '1';
        wait for clk_period/2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        -- Wait for some time to allow the clock to stabilize
        wait for 100 ns;

        -- Check if the output clock is half the frequency of the input clock
        for i in 1 to 10 loop
            wait until rising_edge(Clk_out);
```

```
            wait for clk_period;
        end loop;

        -- Finish the simulation
        wait;
    end process;

end Behavioral;
```

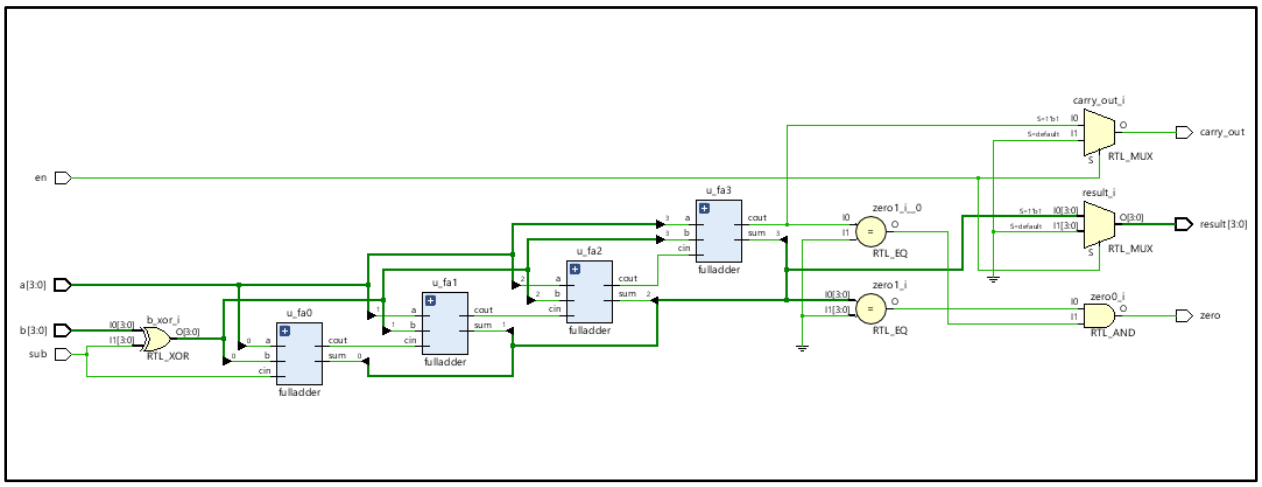
Time simulation for slow clock



4-bit Adder/Subtractor

This is 4-bit adder/subtractor. It's task is add or subtract two 4-bit numbers. When add_sub_enable =='0' it's a adder and add_sub_enable=='1', It's a subtractor. We can get outputs in 2's compliment.

Elaborated Design Schematic



VHDL code for design file

```
library ieee;
use ieee.std_logic_1164.all;

entity addsub4 is
    port (
        a      : in  std_logic_vector(3 downto 0);
        b      : in  std_logic_vector(3 downto 0);
        sub    : in  std_logic;
        en : in std_logic;
        result: out std_logic_vector(3 downto 0);
        carry_out: out std_logic;
        zero   : out std_logic
    );
end addsub4;

architecture behavioral of addsub4 is
    component fulladder
        port (
            a    : in  std_logic;
```

```
            b   : in  std_logic;
            cin : in  std_logic;
            sum : out std_logic;
            cout: out std_logic
        );
    end component;

    signal c   : std_logic_vector(4 downto 0);
    signal b_xor: std_logic_vector(3 downto 0);
    signal sum : std_logic_vector(3 downto 0);
    signal zeroflag : std_logic;

begin
    -- XOR b with the sub signal to perform addition or subtraction
    b_xor <= b xor (sub & sub & sub & sub);

    -- Instantiate full adders
    u_fa0 : fulladder
        port map (
            a   => a(0),
            b   => b_xor(0),
            cin => sub,
            sum => sum(0),
            cout=> c(1)
        );

    u_fa1 : fulladder
        port map (
            a   => a(1),
            b   => b_xor(1),
            cin => c(1),
            sum => sum(1),
            cout=> c(2)
        );

    u_fa2 : fulladder
        port map (
            a   => a(2),
            b   => b_xor(2),
            cin => c(2),
            sum => sum(2),
            cout=> c(3)
        );

    u_fa3 : fulladder
```

```
        port map (
            a   => a(3),
            b   => b_xor(3),
            cin => c(3),
            sum => sum(3),
            cout=> c(4)
        );

    -- Assign outputs

     result   <= sum when en='1' else (others => '0');
    carry_out<= c(4) when en='1' else '0';

    -- Optimize zero flag calculation
    process(Sum, C(4))
    begin
        if (Sum = "0000" and C(4) = '0') then
            Zero <= '1';
        else
            Zero <= '0';
        end if;
    end process;



end behavioral;
```

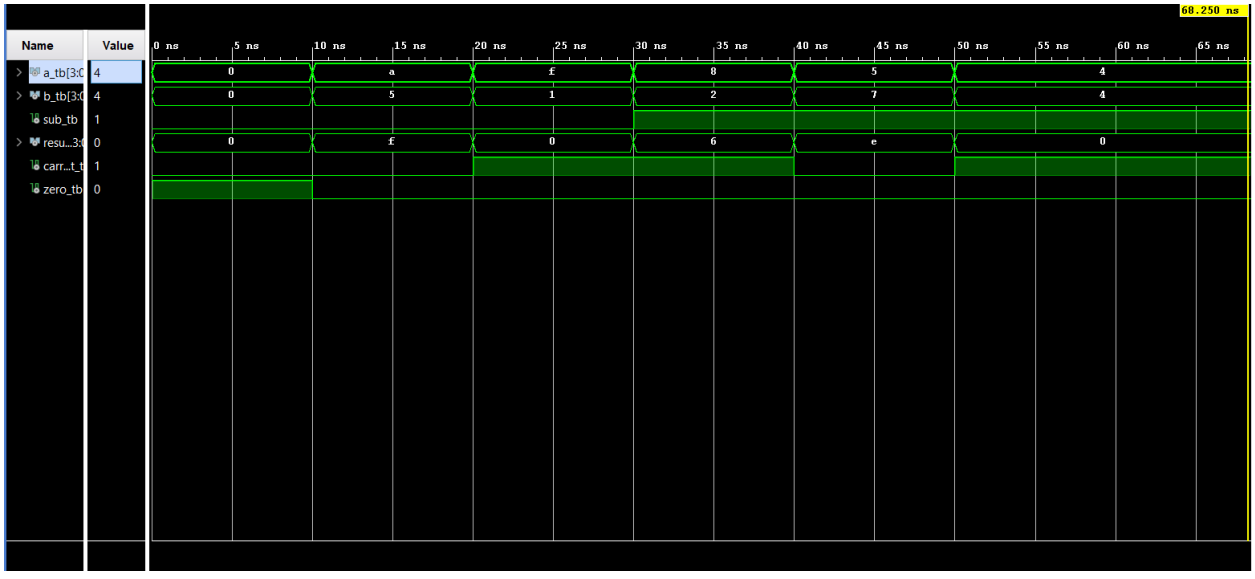
VHDL code for simulation file

```
library ieee;
use ieee.std_logic_1164.all;

entity addsub4_tb is
end addsub4_tb;

architecture testbench of addsub4_tb is
    component addsub4
        port (
            a        : in  std_logic_vector(3 downto 0);
            b        : in  std_logic_vector(3 downto 0);
            sub      : in  std_logic;
            result   : out std_logic_vector(3 downto 0);
            carry_out: out std_logic;
            zero     : out std_logic
        );
```

```
    end component;

    signal a_tb        : std_logic_vector(3 downto 0);
    signal b_tb        : std_logic_vector(3 downto 0);
    signal sub_tb      : std_logic;
    signal result_tb   : std_logic_vector(3 downto 0);
    signal carry_out_tb: std_logic;
    signal zero_tb     : std_logic;

begin
    -- Instantiate the Unit Under Test (UUT)
    uut : addsub4
        port map (
            a        => a_tb,
            b        => b_tb,
            sub      => sub_tb,
            result   => result_tb,
            carry_out=> carry_out_tb,
            zero     => zero_tb
        );

    -- Stimulus process
    stim_proc : process
    begin
        -- Test cases for addition
        sub_tb <= '0';

        -- Test case 1: 0000 + 0000
        a_tb    <= "0000";
        b_tb    <= "0000";
        wait for 10 ns;
        assert result_tb = "0000" and carry_out_tb = '0' and zero_tb = '1'
            report "Test case 1 failed" severity error;

        -- Test case 2: 1010 + 0101
        a_tb    <= "1010";
        b_tb    <= "0101";
        wait for 10 ns;
        assert result_tb = "1111" and carry_out_tb = '0' and zero_tb = '0'
            report "Test case 2 failed" severity error;

        -- Test case 3: 1111 + 0001
        a_tb    <= "1111";
        b_tb    <= "0001";
        wait for 10 ns;
```

```
        assert result_tb = "0000" and carry_out_tb = '1' and zero_tb = '1'
            report "Test case 3 failed" severity error;

        -- Test cases for subtraction
        sub_tb <= '1';

        -- Test case 4: 1000 - 0010
        a_tb    <= "1000";
        b_tb    <= "0010";
        wait for 10 ns;
        assert result_tb = "0110" and carry_out_tb = '0' and zero_tb = '0'
            report "Test case 4 failed" severity error;

        -- Test case 5: 0101 - 0111
        a_tb    <= "0101";
        b_tb    <= "0111";
        wait for 10 ns;
        assert result_tb = "1110" and carry_out_tb = '1' and zero_tb = '0'
            report "Test case 5 failed" severity error;

        -- Test case 6: 0100 - 0100
        a_tb    <= "0100";
        b_tb    <= "0100";
        wait for 10 ns;
        assert result_tb = "0000" and carry_out_tb = '0' and zero_tb = '1'
            report "Test case 6 failed" severity error;

        wait; -- End the simulation
    end process;
end testbench;
```

Time simulation for add/sub.



Program counter

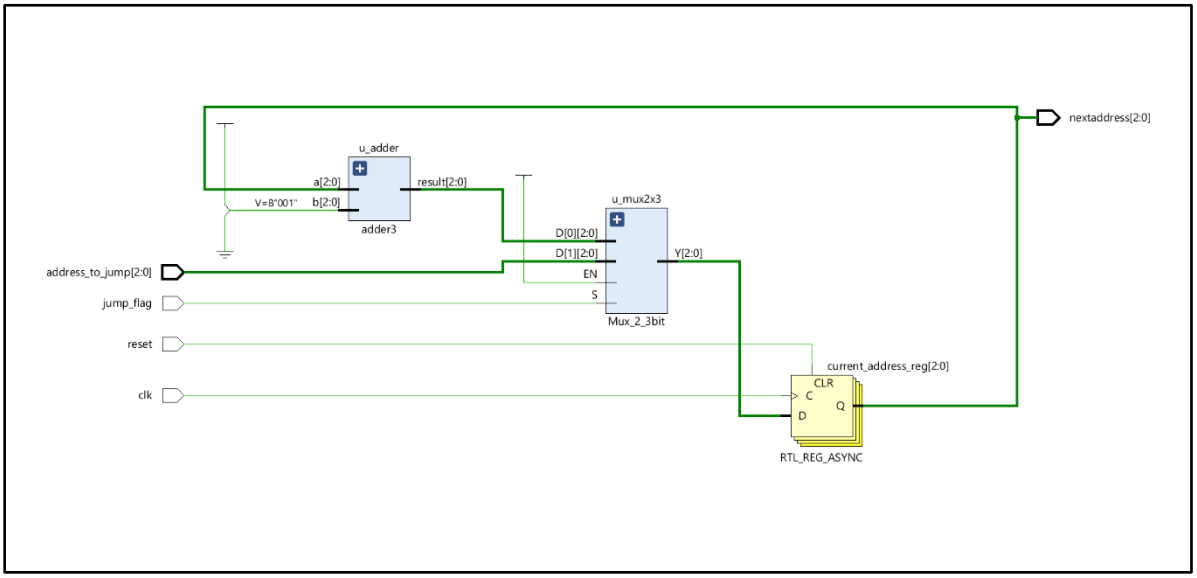
Program counter stores next address of instruction. When a program is running, the CPU fetches instructions from memory sequentially, using the program counter as a pointer to determine the address of the next instruction. In our PC addresses are 3 bit and so we can have maximum 8 addresses.

VHDL file for design file

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.Bus_System.all;

entity program_counter is
    port (
        clk : in std_logic;
        reset : in std_logic;
        nextaddress : out std_logic_vector(2 downto 0);
        jump_flag : in std_logic;
        address_to_jump : in std_logic_vector(2 downto 0)
    );
end program_counter;

architecture behavioral of program_counter is

    component adder3
        port (
            a : in std_logic_vector(2 downto 0);
            b : in std_logic_vector(2 downto 0);
            result : out std_logic_vector(2 downto 0);
            carry_out : out std_logic
        );
    end component;

    component Mux_2_3bit is
        port (
            D : in BUS_2_3bit;
            S : in STD_LOGIC;
            EN : in STD_LOGIC;
            Y : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;

    signal current_address : std_logic_vector(2 downto 0);
    signal next_address : std_logic_vector(2 downto 0);
```

```
    signal jflag : std_logic;
    signal mux_in : BUS_2_3bit;
    signal mux_out : std_logic_vector(2 downto 0);

begin

    u_adder : adder3
        port map (
            a => current_address,
            b => "001",
            result => next_address,
            carry_out => open
        );

    u_mux2x3 : Mux_2_3bit
        port map (
            D => mux_in,
            S => jflag,
            EN => '1',
            Y => mux_out
        );

    mux_in(0) <= next_address;
    mux_in(1) <= address_to_jump;
    jflag <= jump_flag;

    process (clk, reset)
    begin
        if reset = '1' then
            current_address <= "000";
        elsif rising_edge(clk) then
            current_address <= mux_out;
        end if;
    end process;

    nextaddress <= current_address;

end behavioral;
```

Elaborated Design Schematic for Program Counter



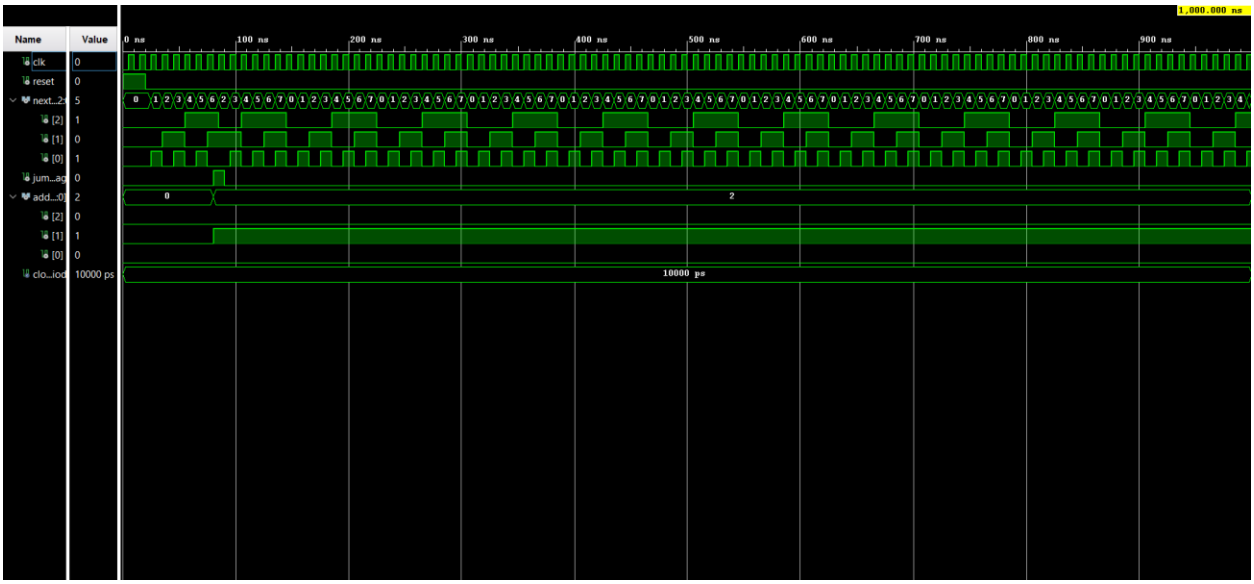
VHDL file for Program Counter test bench

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.Bus_System.all;

entity program_counter_tb is
end program_counter_tb;

architecture behavioral of program_counter_tb is
    component program_counter
        port (
            clk : in std_logic;
            reset : in std_logic;
            nextaddress : out std_logic_vector(2 downto 0);
            jump_flag : in std_logic;
            address_to_jump : in std_logic_vector(2 downto 0)
        );
    end component;

    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal nextaddress : std_logic_vector(2 downto 0);
    signal jump_flag : std_logic := '0';
    signal address_to_jump : std_logic_vector(2 downto 0) := (others => '0');
```

```
    constant clock_period : time := 10 ns;
begin
    uut : program_counter
        port map (
            clk => clk,
            reset => reset,
            nextaddress => nextaddress,
            jump_flag => jump_flag,
            address_to_jump => address_to_jump
        );

    clk_process : process
    begin
        clk <= '0';
        wait for clock_period / 2;
        clk <= '1';
        wait for clock_period / 2;
    end process;

    test_process : process
    begin
        -- Reset the program counter
        reset <= '1';
        wait for 2 * clock_period;
        reset <= '0';

        -- Test normal operation
        wait for 6 * clock_period;
        assert nextaddress = "110" report "Normal operation test failed" severity
error;

        -- Test jump operation
        jump_flag <= '1';
        address_to_jump <= "010";
        wait for clock_period;
        assert nextaddress = "010" report "Jump operation test failed" severity
error;

        -- Test incrementing after jump
        jump_flag <= '0';
        wait for 3 * clock_period;
        assert nextaddress = "011" report "Increment after jump test failed"
severity error;
```

```
        wait;
    end process;
end behavioral;
```

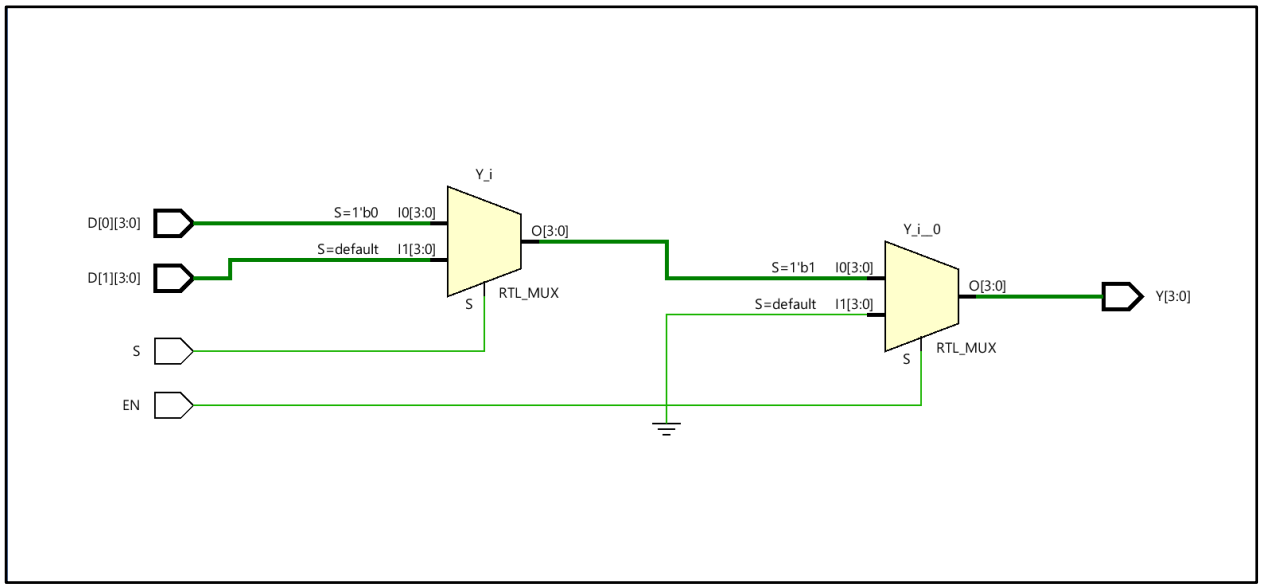
Timing Diagram for Program Counter



Two way 4-bit mux

2 way 4-bit mux is for selection when execute a add or subtract instructions. Load select is decide what is the value should go though mux and store in register, immediate value or result from Adder.

Elaborated Design Schematic



VHDL code for design file

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

entity Mux_2_4bit is
    port ( --D0 : in BUS_1_4bit;
            --D1 : in BUS_1_4bit;
            D: in BUS_2_4bit;
           S : in STD_LOGIC;
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR(3 downto 0));
end Mux_2_4bit;

architecture Behavioral of Mux_2_4bit is
begin
    process(D, S, EN)
```

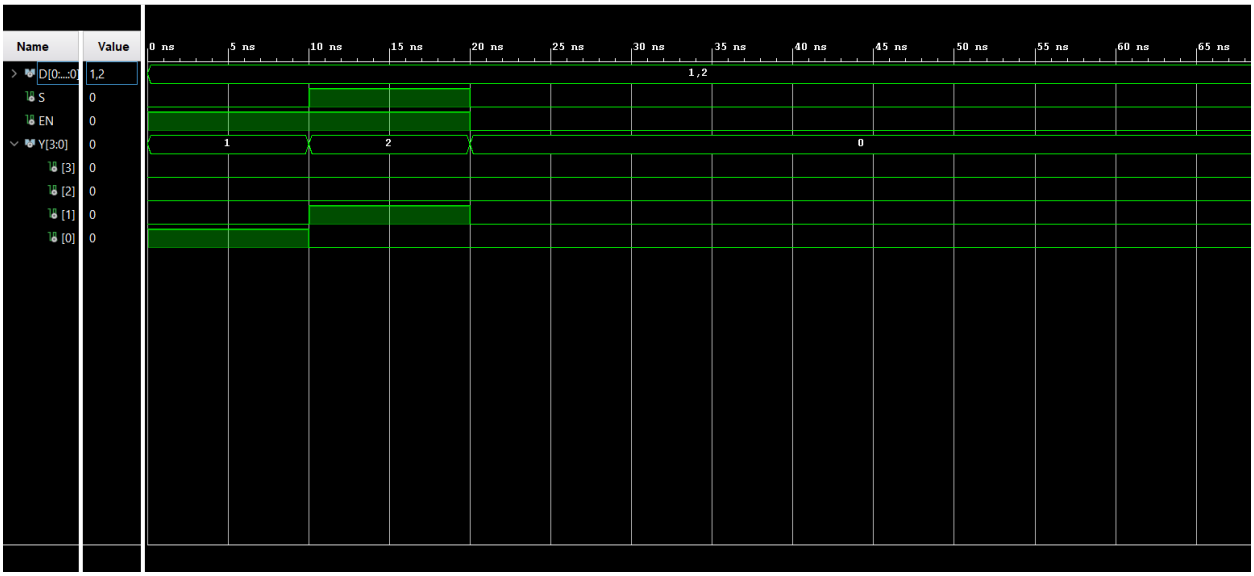
```
    begin
        if (EN = '1') then
            if (S = '0') then
                Y <= D(0);
            else
                Y <= D(1);
            end if;
        else
            Y <= "0000";
        end if;
    end process;
end Behavioral;
```

VHDL file for simulation file

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

entity Mux_2_4bit_TB is
end Mux_2_4bit_TB;

architecture Behavioral of Mux_2_4bit_TB is
    component Mux_2_4bit
        port (
            D : in BUS_2_4bit;
            S : in STD_LOGIC;
            EN : in STD_LOGIC;
            Y : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    signal D : BUS_2_4bit;
    signal S : STD_LOGIC;
    signal EN : STD_LOGIC;
    signal Y : STD_LOGIC_VECTOR(3 downto 0);

begin
    UUT : Mux_2_4bit
        port map (
            D => D,
            S => S,
            EN => EN,
            Y => Y
        );
```

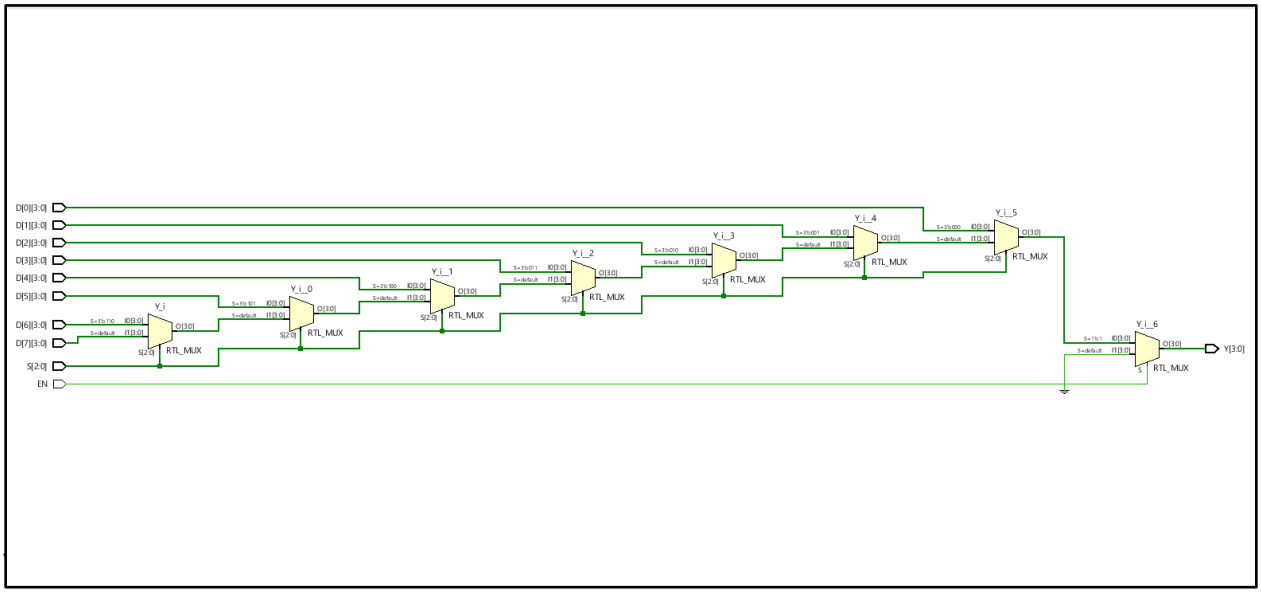
```
    stimulus : process
    begin
        -- Test case 1: S = '0', EN = '1'
        D <= ("0001", "0010");
        S <= '0';
        EN <= '1';
        wait for 10 ns;
        assert (Y = "0001") report "Test case 1 failed" severity error;

        -- Test case 2: S = '1', EN = '1'
        D <= ("0001", "0010");
        S <= '1';
        EN <= '1';
        wait for 10 ns;
        assert (Y = "0010") report "Test case 2 failed" severity error;

        -- Test case 3: EN = '0'
        D <= ("0001", "0010");
        S <= '0';
        EN <= '0';
        wait for 10 ns;
        assert (Y = "0000") report "Test case 3 failed" severity error;

        wait;
    end process;
end Behavioral;
```

Time simulation for 2-way 4 bit mux



Eight-way 4-bit mux

Elaborated Design Schematic




```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use WORK.Bus_System.ALL;

entity Mux_8_4bit is
    port ( D : in BUS_8_4bit;
           S : in STD_LOGIC_VECTOR(2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR(3 downto 0));
end Mux_8_4bit;

architecture Behavioral of Mux_8_4bit is
begin
    process(D, S, EN)
    begin
        if (EN = '1') then
            if (S = "000") then
                Y <= D(0);
            else if (S = "001") then
                Y <= D(1);
            else if (S = "010") then
                Y <= D(2);
            else if (S = "011") then
                Y <= D(3);
            else if (S = "100") then
                Y <= D(4);
```

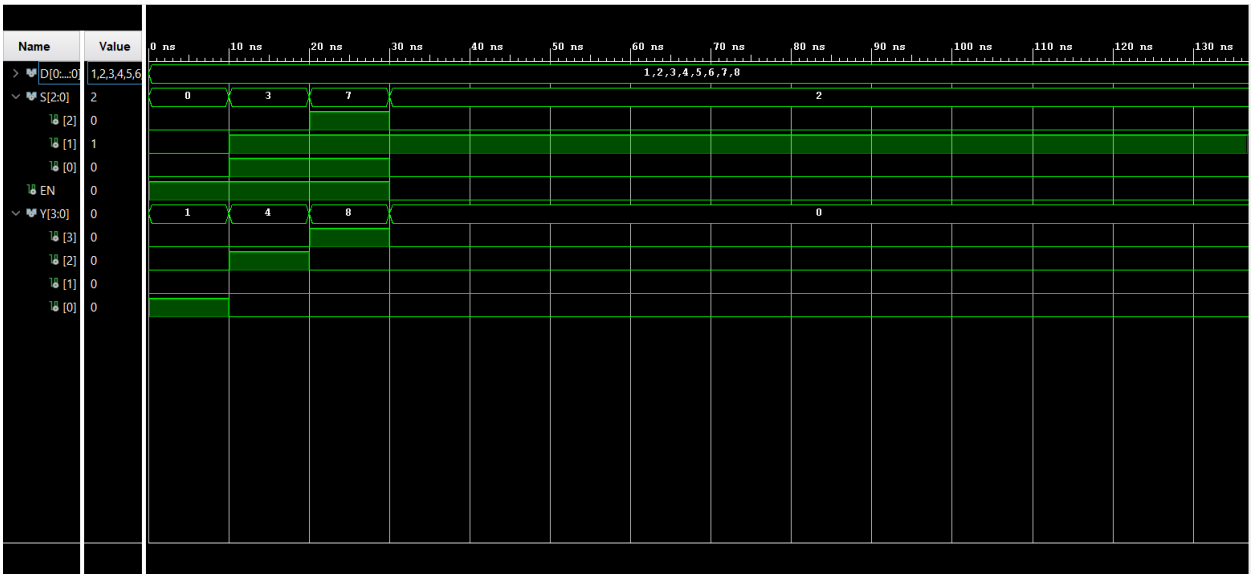
```
            else if (S = "101") then
                Y <= D(5);
            else if (S = "110") then
                Y <= D(6);
            else
                Y <= D(7);
            end if;
            end if;
            end if;
            end if;
            end if;
            end if;
            end if;
        else
            Y <= "0000";
        end if;
    end process;
end Behavioral;
```

VHDL code for simulation file

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

entity Mux_8_4bit_TB is
end Mux_8_4bit_TB;

architecture Behavioral of Mux_8_4bit_TB is
    component Mux_8_4bit
        port (
            D : in BUS_8_4bit;
            S : in STD_LOGIC_VECTOR(2 downto 0);
            EN : in STD_LOGIC;
            Y : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    signal D : BUS_8_4bit;
    signal S : STD_LOGIC_VECTOR(2 downto 0);
    signal EN : STD_LOGIC;
    signal Y : STD_LOGIC_VECTOR(3 downto 0);

begin
    UUT : Mux_8_4bit
```

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

    stimulus : process
    begin
        -- Test case 1: S = "000", EN = '1'
        D <= ("0001", "0010", "0011", "0100", "0101", "0110", "0111", "1000");
        S <= "000";
        EN <= '1';
        wait for 10 ns;
        assert (Y = "0001") report "Test case 1 failed" severity error;

        -- Test case 2: S = "011", EN = '1'
        D <= ("0001", "0010", "0011", "0100", "0101", "0110", "0111", "1000");
        S <= "011";
        EN <= '1';
        wait for 10 ns;
        assert (Y = "0100") report "Test case 2 failed" severity error;

        -- Test case 3: S = "111", EN = '1'
        D <= ("0001", "0010", "0011", "0100", "0101", "0110", "0111", "1000");
        S <= "111";
        EN <= '1';
        wait for 10 ns;
        assert (Y = "1000") report "Test case 3 failed" severity error;

        -- Test case 4: EN = '0'
        D <= ("0001", "0010", "0011", "0100", "0101", "0110", "0111", "1000");
        S <= "010";
        EN <= '0';
        wait for 10 ns;
        assert (Y = "0000") report "Test case 4 failed" severity error;

        wait;
    end process;
end Behavioral;
```

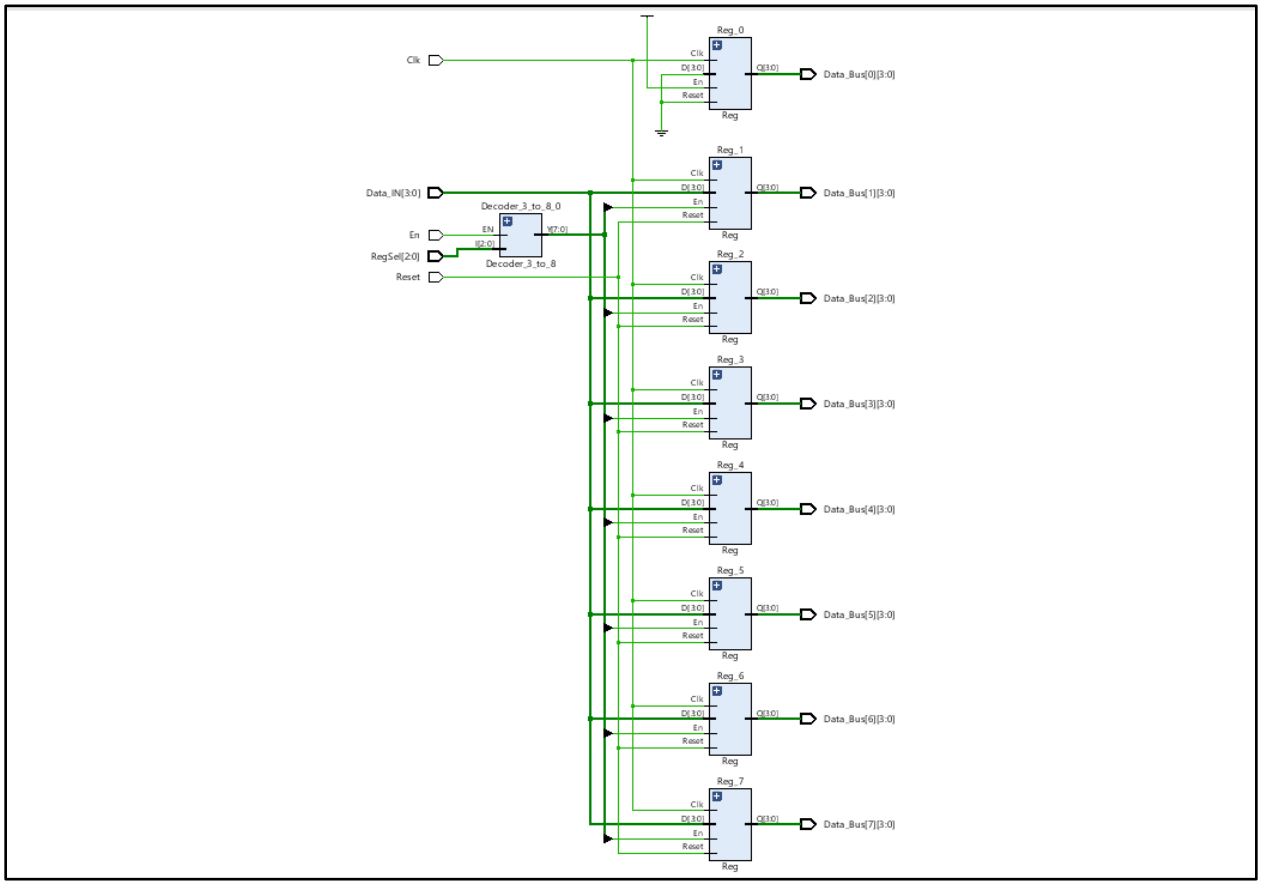
Time simulation for 8-way 4bit mux



Register Bank

In the register bank there are 7 registers. We already have hard coded register 0 to “0000” and we save adder results in register 7.

Elaborated Design Schematic



VHDL code for design file

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

entity RegBank is
    Port (RegSel : in STD_LOGIC_VECTOR (2 downto 0);
          Clk : in STD_LOGIC;
          En : in STD_LOGIC;
          Reset : in STD_LOGIC;
          Data_IN : in STD_LOGIC_VECTOR (3 downto 0);
          Data_Bus : out BUS_8_4bit);
end RegBank;
```

```
architecture Behavioral of RegBank 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 regsel_signal : STD_LOGIC_VECTOR (7 DOWNTO 0);
    signal reg_outputs : BUS_8_4bit; -- Array to hold register outputs

begin
    Decoder_3_to_8_0 : Decoder_3_to_8
    port map (
        I => RegSel,
        En => En,
        Y => regsel_signal
    );

    -- Register R0 is to be hardcoded to 0000
    Reg_0 : Reg
    port map (
        D => "0000",
        En => '1',
        Reset => '0',
        Clk => Clk,
        Q => reg_outputs(0)
    );

    Reg_1 : Reg
    port map (
        D => Data_IN,
        En => regsel_signal(1),
        Reset => Reset,
        Clk => Clk,
        Q => reg_outputs(1)
    );
```

```
Reg_2 : Reg
port map (
    D => Data_IN,
    En => regsel_signal(2),
    Reset => Reset,
    Clk => Clk,
    Q => reg_outputs(2)
);

Reg_3 : Reg
port map (
    D => Data_IN,
    En => regsel_signal(3),
    Reset => Reset,
    Clk => Clk,
    Q => reg_outputs(3)
);

Reg_4 : Reg
port map (
    D => Data_IN,
    En => regsel_signal(4),
    Reset => Reset,
    Clk => Clk,
    Q => reg_outputs(4)
);

Reg_5 : Reg
port map (
    D => Data_IN,
    En => regsel_signal(5),
    Reset => Reset,
    Clk => Clk,
    Q => reg_outputs(5)
);

Reg_6 : Reg
port map (
    D => Data_IN,
    En => regsel_signal(6),
    Reset => Reset,
    Clk => Clk,
    Q => reg_outputs(6)
);
```

```
    Reg_7 : Reg
    port map (
        D => Data_IN,
        En => regsel_signal(7),
        Reset => Reset,
        Clk => Clk,
        Q => reg_outputs(7)
    );

    Data_Bus <= reg_outputs; -- Assign the array of register outputs to Data_Bus

end Behavioral;
```

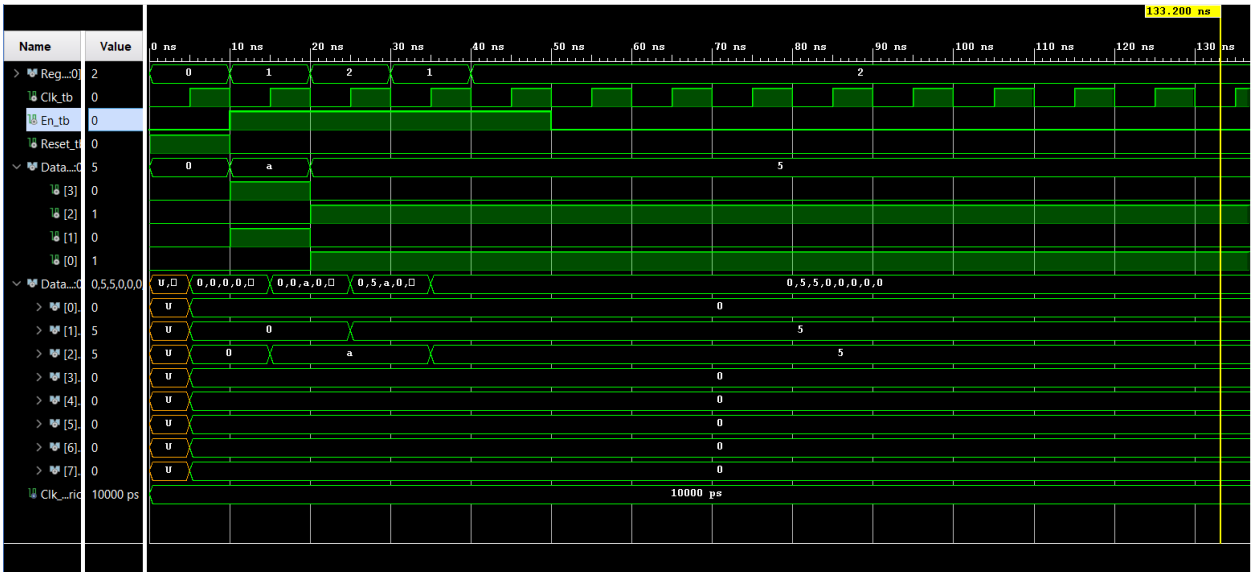
VHDL code for simulation file

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

entity RegBank_tb is
end RegBank_tb;

architecture Behavioral of RegBank_tb is
    -- Component Declaration
    component RegBank
        Port (RegSel  : in STD_LOGIC_VECTOR (2 downto 0);
              Clk     : in STD_LOGIC;
              En      : in STD_LOGIC;
              Reset   : in STD_LOGIC;
              Data_IN : in STD_LOGIC_VECTOR (3 downto 0);
              Data_Bus: out BUS_8_4bit);
    end component;

    -- Signals
    signal RegSel_tb   : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
    signal Clk_tb      : STD_LOGIC := '0';
    signal En_tb       : STD_LOGIC := '0';
    signal Reset_tb    : STD_LOGIC := '0';
    signal Data_IN_tb  : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
    signal Data_Bus_tb : BUS_8_4bit;

    -- Clock Period
```

```
    constant Clk_period : time := 10 ns;

begin
    -- Instantiate the Unit Under Test (UUT)
    uut : RegBank
        port map (
            RegSel  => RegSel_tb,
            Clk     => Clk_tb,
            En      => En_tb,
            Reset   => Reset_tb,
            Data_IN => Data_IN_tb,
            Data_Bus => Data_Bus_tb
        );

    -- Clock process
    Clk_process : process
    begin
        Clk_tb <= '0';
        wait for Clk_period / 2;
        Clk_tb <= '1';
        wait for Clk_period / 2;
    end process;

    -- Stimulus process
    stim_proc : process
    begin
        -- Reset the register bank
        Reset_tb <= '1';
        wait for Clk_period;
        Reset_tb <= '0';

        -- Write data to register 1
        RegSel_tb <= "001";
        Data_IN_tb <= "1010";
        En_tb <= '1';
        wait for Clk_period;
        En_tb <= '0';

        -- Write data to register 2
        RegSel_tb <= "010";
        Data_IN_tb <= "0101";
        En_tb <= '1';
        wait for Clk_period;
        En_tb <= '0';
```

```
        -- Read data from register 1
        RegSel_tb <= "001";
        En_tb <= '1';
        wait for Clk_period;
        En_tb <= '0';

        -- Read data from register 2
        RegSel_tb <= "010";
        En_tb <= '1';
        wait for Clk_period;
        En_tb <= '0';

        wait; -- Wait forever
    end process;

end Behavioral;
```

Time simulation for reg bank



Instruction decoder

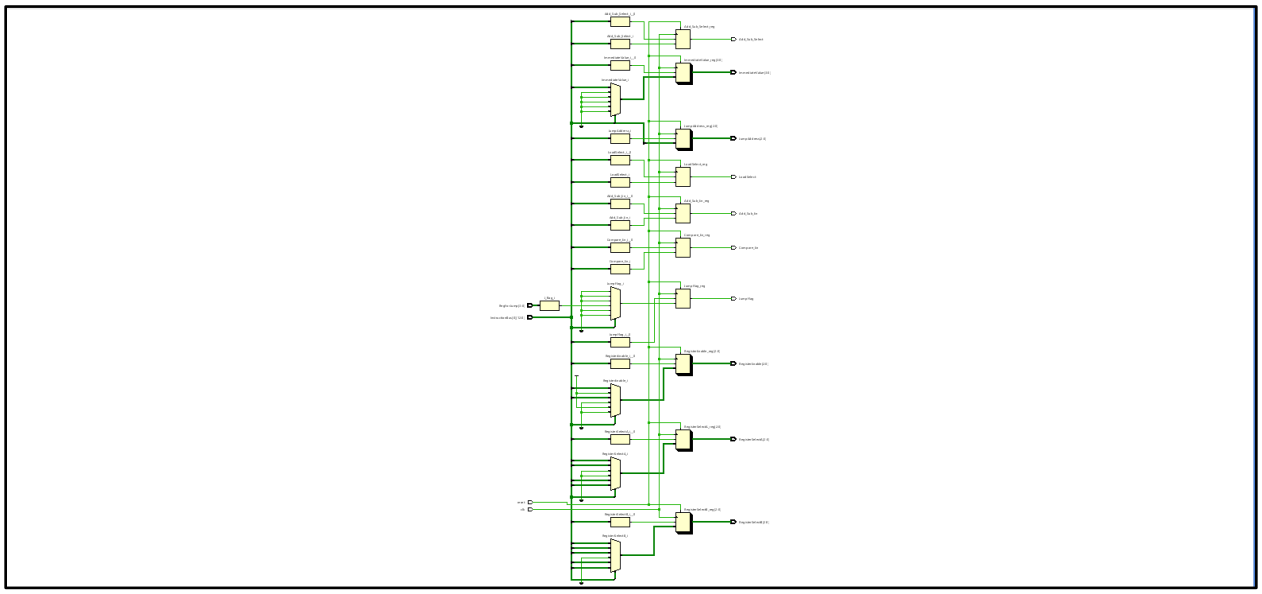
Instruction decoder is the component which execute instructions. We send instruction from program rom using a bus.

So instruction has few components too.

"0010010100000" -----→"001|001|010|0000"

- First three bits from left --→ opcode, which decide what is the operation(Load, Add, Sub)
- Next three is about first register address.
- Next three about other register address.
- Final 4 bits about immediate value(value we load).

Elaborated Design Schematic



VHDL code for design file

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

entity Instruction_Decoder is
```

```
    Port (
        clk            : in std_logic;
        reset          : in std_logic;
        InstructionBus : in BUS_13bit;
        RegForJump     : in STD_LOGIC_VECTOR(3 downto 0);
        RegisterEnable : out STD_LOGIC_VECTOR(2 downto 0);
        LoadSelect     : out std_logic;
        ImmediateValue : out std_logic_vector(3 downto 0);
        RegisterSelectA: out std_logic_vector(2 downto 0);
        RegisterSelectB: out std_logic_vector(2 downto 0);
        Add_Sub_Select : out std_logic;
        JumpFlag       : out std_logic;
        JumpAddress    : out std_logic_vector(2 downto 0);
        Add_Sub_En     : out std_logic;
        Compare_En     : out std_logic
        --Mul_En         : out std_logic
    );
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is
    signal opcode    : std_logic_vector(2 downto 0);
    signal reg_a     : std_logic_vector(2 downto 0);
    signal reg_b     : std_logic_vector(2 downto 0);
    signal Jaddress  : std_logic_vector(2 downto 0);
    signal J_flag    : std_logic;
begin
    opcode <= InstructionBus(0)(12 downto 10);
    reg_a  <= InstructionBus(0)(9 downto 7);
    reg_b  <= InstructionBus(0)(6 downto 4);

    -- Calculate jump address
    J_flag    <=  '1' when RegForJump = "0000" else '0';
    Jaddress  <=  InstructionBus(0)(2 downto 0);

    process (clk, reset)
    begin
        if reset = '1' then
            JumpFlag        <= '0';
            JumpAddress     <= (others => '0');
            RegisterEnable  <= (others => '0');
            LoadSelect      <= '0';
            ImmediateValue  <= (others => '0');
            RegisterSelectA <= (others => '0');
            RegisterSelectB <= (others => '0');
            Add_Sub_Select  <= '0';
```

```
        Add_Sub_En      <= '0';
        Compare_En      <= '0';

    elsif rising_edge(clk) then
        case opcode is
            when "000" => -- MOVI Instruction
                RegisterEnable  <= reg_a;
                LoadSelect      <= '0';
                ImmediateValue  <= InstructionBus(0)(3 downto 0);
                RegisterSelectA <= reg_a;
                RegisterSelectB <= reg_b;
                Add_Sub_Select  <= '0';
                Add_Sub_En      <= '0';
                Compare_En      <= '0';
                JumpFlag        <= '0';

            when "001" => -- ADD Instruction
                RegisterEnable  <= "111";
                LoadSelect      <= '1';
                ImmediateValue  <= (others => '0');
                RegisterSelectA <= reg_a;
                RegisterSelectB <= reg_b;
                Add_Sub_Select  <= '0';
                Add_Sub_En      <= '1';
                Compare_En      <= '0';
                JumpFlag        <= '0';

            when "010" => -- NEG Instruction
                RegisterEnable  <= reg_a;
                LoadSelect      <= '1';
                ImmediateValue  <= (others => '0');
                RegisterSelectA <= (others => '0');
                RegisterSelectB <= reg_a;
                Add_Sub_Select  <= '1';
                Add_Sub_En      <= '1';
                Compare_En      <= '0';
                JumpFlag        <= '0';

            when "011" => -- JZR Instruction
                RegisterEnable  <= "000";
                LoadSelect      <= '0';
                ImmediateValue  <= (others => '0');
                RegisterSelectA <= (others => '0');
                RegisterSelectB <= (others => '0');
                Add_Sub_Select  <= '0';
```

```
                    JumpFlag        <= J_flag;
                    JumpAddress     <= Jaddress;
                    Add_Sub_En      <= '0';
                    Compare_En      <= '0';

                when "100" => -- SUB Instruction
                    RegisterEnable  <= "111";
                    LoadSelect      <= '1';
                    ImmediateValue  <= (others => '0');
                    RegisterSelectA <= reg_a;
                    RegisterSelectB <= reg_b;
                    Add_Sub_Select  <= '1';
                    Add_Sub_En      <= '1';
                    Compare_En      <= '0';
                    JumpFlag        <= '0';

                when "101" => -- COM Instruction
                    RegisterEnable  <= "000";
                    LoadSelect      <= '1';
                    ImmediateValue  <= (others => '0');
                    RegisterSelectA <= reg_a;
                    RegisterSelectB <= reg_b;
                    Add_Sub_Select  <= '0';
                    Add_Sub_En      <='0';
                    Compare_En      <='1';
                    JumpFlag        <= '0';

--                when "110" => -- MUL Instruction
--                    RegisterEnable  <= "111";
--                    LoadSelect      <= '1';
--                    ImmediateValue  <= (others => '0');
--                    RegisterSelectA <= reg_a;
--                    RegisterSelectB <= reg_b;
--                    Add_Sub_Select  <= '0';
--                    Add_Sub_En      <= '0';
--                    Mul_En          <= '0';
--                    Compare_En      <= '0';
--                    JumpFlag        <= '0';

                when others =>
                    null;
            end case;
        end if;
    end process;
end Behavioral;
```

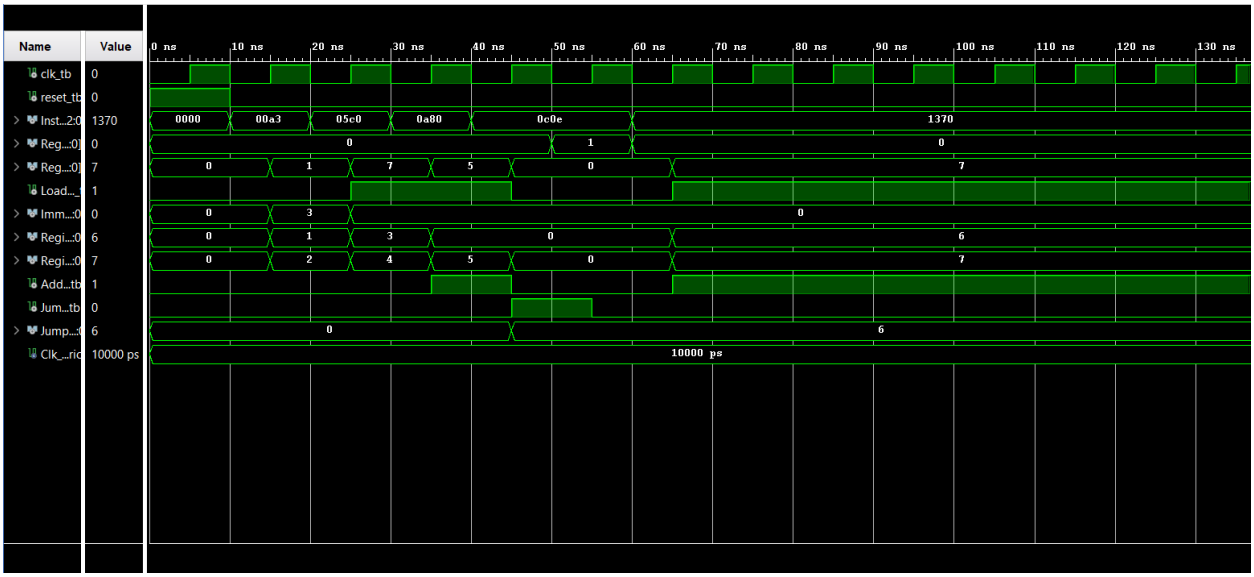
VHDL code for simulation file

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use WORK.Bus_System.ALL;

entity Instruction_Decoder_tb is
end Instruction_Decoder_tb;

architecture Testbench of Instruction_Decoder_tb is
    -- Component Declaration
    component Instruction_Decoder
        Port (
            clk            : in std_logic;
            reset          : in std_logic;
            InstructionBus : in BUS_13bit;
            RegForJump     : in STD_LOGIC_VECTOR(3 downto 0);
            RegisterEnable : out STD_LOGIC_VECTOR(2 downto 0);
            LoadSelect     : out std_logic;
            ImmediateValue : out std_logic_vector(3 downto 0);
            RegisterSelectA: out std_logic_vector(2 downto 0);
            RegisterSelectB: out std_logic_vector(2 downto 0);
            Add_Sub_Select : out std_logic;
            JumpFlag       : out std_logic;
            JumpAddress    : out std_logic_vector(2 downto 0)
        );
    end component;

    -- Input Signals
    signal clk_tb         : std_logic := '0';
    signal reset_tb       : std_logic := '0';
    signal InstructionBus_tb: BUS_13bit := (others => (others => '0'));
    signal RegForJump_tb  : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');

    -- Output Signals
    signal RegisterEnable_tb: STD_LOGIC_VECTOR(2 downto 0);
    signal LoadSelect_tb  : std_logic;
    signal ImmediateValue_tb: std_logic_vector(3 downto 0);
    signal RegisterSelectA_tb: std_logic_vector(2 downto 0);
    signal RegisterSelectB_tb: std_logic_vector(2 downto 0);
    signal Add_Sub_Select_tb: std_logic;
    signal JumpFlag_tb    : std_logic;
    signal JumpAddress_tb : std_logic_vector(2 downto 0);
```

```
    -- Clock Period
    constant Clk_period : time := 10 ns;

begin
    -- Instantiate the Unit Under Test (UUT)
    uut : Instruction_Decoder
        port map (
            clk            => clk_tb,
            reset          => reset_tb,
            InstructionBus => InstructionBus_tb,
            RegForJump     => RegForJump_tb,
            RegisterEnable => RegisterEnable_tb,
            LoadSelect     => LoadSelect_tb,
            ImmediateValue => ImmediateValue_tb,
            RegisterSelectA => RegisterSelectA_tb,
            RegisterSelectB => RegisterSelectB_tb,
            Add_Sub_Select => Add_Sub_Select_tb,
            JumpFlag       => JumpFlag_tb,
            JumpAddress    => JumpAddress_tb
        );

    -- Clock process
    Clk_process : process
    begin
        clk_tb <= '0';
        wait for Clk_period / 2;
        clk_tb <= '1';
        wait for Clk_period / 2;
    end process;

    -- Stimulus process
    stim_proc : process
    begin
        -- Reset the instruction decoder
        reset_tb <= '1';
        wait for Clk_period;
        reset_tb <= '0';

        -- Test case 1: MOVI Instruction
        InstructionBus_tb(0) <= "000" & "001" & "010" & "0011";
        RegForJump_tb <= "0000";
        wait for Clk_period;
        -- Add your assertions here
```

```
        -- Test case 2: ADD Instruction
        InstructionBus_tb(0) <= "001" & "011" & "100" & "0000";
        RegForJump_tb <= "0000";
        wait for Clk_period;
        -- Add your assertions here

        -- Test case 3: NEG Instruction
        InstructionBus_tb(0) <= "010" & "101" & "000" & "0000";
        RegForJump_tb <= "0000";
        wait for Clk_period;
        -- Add your assertions here

        -- Test case 4: JZR Instruction (Jump condition true)
        InstructionBus_tb(0) <= "011" & "000" & "000" & "111";
        RegForJump_tb <= "0000";
        wait for Clk_period;
        -- Add your assertions here

        -- Test case 5: JZR Instruction (Jump condition false)
        InstructionBus_tb(0) <= "011" & "000" & "000" & "111";
        RegForJump_tb <= "0001";
        wait for Clk_period;
        -- Add your assertions here

        -- Test case 6: SUB Instruction
        InstructionBus_tb(0) <= "100" & "110" & "111" & "0000";
        RegForJump_tb <= "0000";
        wait for Clk_period;
        -- Add your assertions here

        wait; -- End the simulation
    end process;
end Testbench;
```

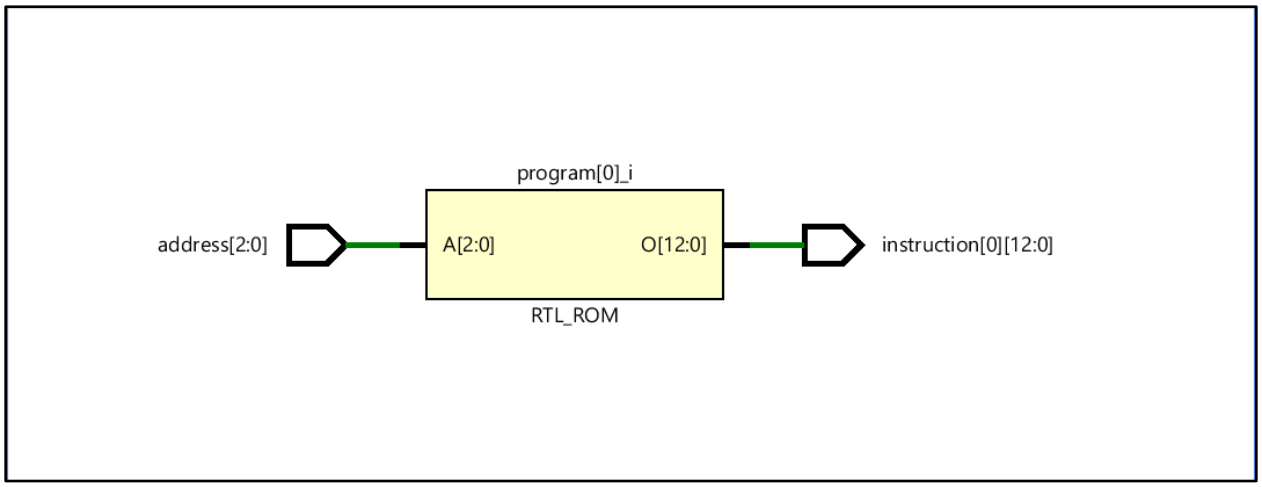
Time simulation for instruction decoder



Program rom.

Program rom is a component which store the instructions we have to execute in process.

Elaborated Design Schematic



VHDL code for design file

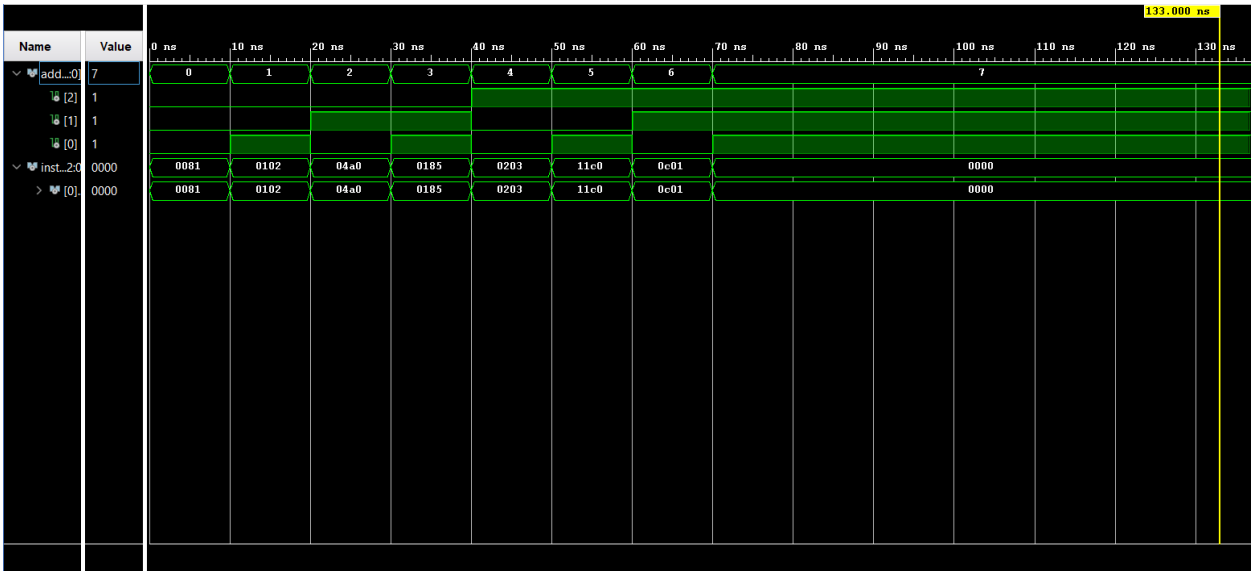
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use WORK.Bus_System.ALL;

entity program_rom is
    port (
        address : in std_logic_vector(2 downto 0);
        instruction : out BUS_13bit
    );
end entity program_rom;

architecture rtl of program_rom is
    type program_memory is array (0 to 7) of std_logic_vector(12 downto 0);
    constant program : program_memory := (


        "0000010000001",  -- Load 1 int register 1
        "0000100000010",  -- Load 2 into register 2
        "0010010100000",  -- Add the contents of registers 1 and 2,


        "0000110000101",  -- Load 5 int register 3
        "0001000000011",  -- Load 3 int register 4
        "1000111000000",  -- SUB the contents of registers 1 and 2,
        "0110000000001",  -- Jump to address,
        "0000000000000" -- comparator









    );
begin
    instruction(0) <= program(to_integer(unsigned(address)));
end architecture rtl;
```

VHDL code for simulation file

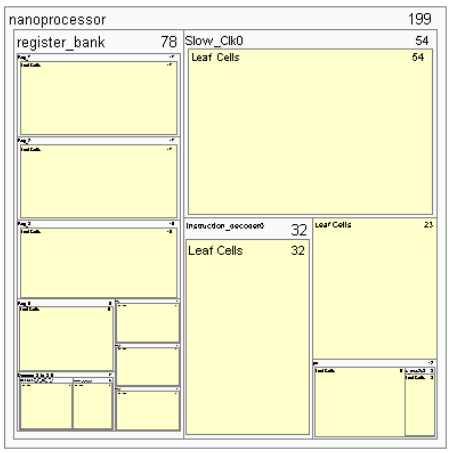
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.bus_system.all;

entity program_rom_tb is
end entity program_rom_tb;

architecture tb of program_rom_tb is
    component program_rom is
        port (
            address : in std_logic_vector(2 downto 0);
            instruction : out BUS_13bit
        );
    end component program_rom;

    signal address : std_logic_vector(2 downto 0) := (others => '0');
    signal instruction : BUS_13bit;

begin
    uut : program_rom
        port map (
            address => address,
            instruction => instruction
        );

    test_proc : process
    begin
        -- Test the program ROM
        address <= "000";
        wait for 10 ns;


        address <= "001";
        wait for 10 ns;

        address <= "010";
        wait for 10 ns;

        address <= "011";
        wait for 10 ns;

        address <= "100";
```

```
        wait for 10 ns;

        address <= "101";
        wait for 10 ns;

        address <= "110";
        wait for 10 ns;

        address <= "111";
        wait for 10 ns;

        wait;
    end process test_proc;
end architecture tb;
```

Time simulation for program rom



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
synth_1	constrs_1	synth_design Complete!								56	81	0.0	0	0	5/5/24, 9:16 AM	00:00:17	Vivado Synthesis Defaults (V
impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	0.553	0	59	83	0.0	0	0	5/5/24, 9:17 AM	00:00:37	Vivado Implementation De



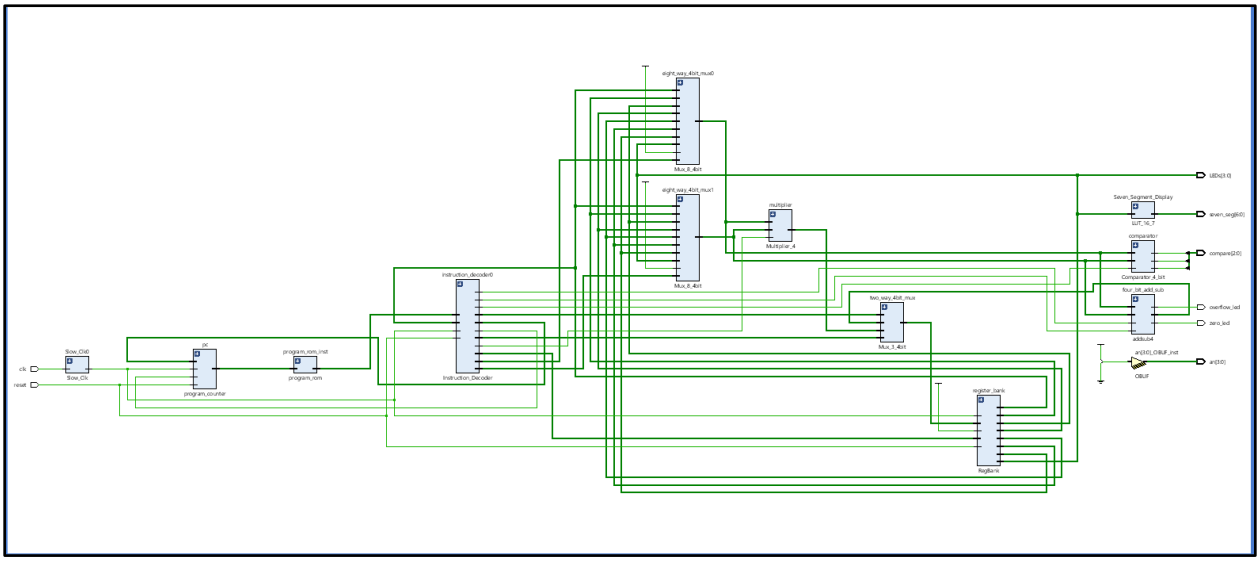
Modified version of nano processor with additional features.

We added some features such as comparator, Multiplier for above base nano processor and we removed 2-way 4 bit mux and replaced it from 3- way 4 bit mux.

Components

Nano Processor

Elaborated Design Schematic



VHDL code for Design file.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use WORK.Bus_System.ALL;

entity nanoprocessor is
    port (
        clk : in std_logic;
        reset : in std_logic;
        overflow_led : out std_logic;
        zero_led : out std_logic;
        seven_seg:out std_logic_vector(6 downto 0);
        LEDs :out std_logic_vector(3 downto 0);
        compare:out std_logic_vector(2 downto 0);
        an : out std_logic_vector(3 downto 0):="1110"
```

```
--        j_flag: out std_logic;
--        j_address: out std_logic_vector(2 downto 0);
--        Reg_Zero_out : out std_logic_vector(3 downto 0)


    );
end entity nanoprocessor;

architecture rtl of nanoprocessor is
    component LUT_16_7 is
        port(
            I : in STD_LOGIC_VECTOR (3 downto 0);
           O : out STD_LOGIC_VECTOR (6 downto 0)
           );
    end component;

    component slow_Clk is
        port(
        Clk_in : in STD_LOGIC;
        Clk_out : out STD_LOGIC
        );
    end component;

    component Seven_Segment_Driver is
    Port (
        clk : in std_logic;
        reset : in std_logic;
        input : in std_logic_vector(4 downto 0);
        segments : out std_logic_vector(6 downto 0);
        an : out std_logic_vector(1 downto 0) -- Anode enable signals for the two
displays
    );
    end component;

    component Multiplier_4 is
        port (
            A : in STD_LOGIC_VECTOR (3 downto 0);
            B : in STD_LOGIC_VECTOR (3 downto 0);
            Y : out STD_LOGIC_VECTOR (7 downto 0);
            EN : in STD_LOGIC
           );
    end component;

    component addsub4 is
        port (
```

```
        a : in std_logic_vector(3 downto 0);
        b : in std_logic_vector(3 downto 0);
        sub : in std_logic;
        en: in std_logic;
        result : out std_logic_vector(3 downto 0);
        carry_out : out std_logic;
        zero:out std_logic

    );
    end component;

    component adder3 is
        port (
        a : in std_logic_vector(2 downto 0);
        b : in std_logic_vector(2 downto 0);
        result : out std_logic_vector(2 downto 0);
        carry_out : out std_logic
    );
    end component;

    component Comparator_4_bit is
        Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC;
           Equal : out STD_LOGIC;
           Greater : out STD_LOGIC;
           Smaller : out STD_LOGIC);
    end component;

    component program_counter is
        port (
        clk :in std_logic;
        reset : in std_logic;
--        currentaddress: in STD_LOGIC_VECTOR(2 downto 0);
        nextaddress : out std_logic_vector(2 downto 0);
        jump_flag : in std_logic;
        address_to_jump :  in std_logic_vector(2 downto 0)
    );
    end component;



    component Mux_3_4bit is
        port (D : in BUS_3_4bit;
           S : in STD_LOGIC_VECTOR(1 downto 0);
```

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

component Mux_8_4bit is
    port ( D : in BUS_8_4bit;
        S : in STD_LOGIC_VECTOR(2 downto 0);
        EN : in STD_LOGIC;
        Y : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component RegBank is
    Port ( RegSel :  in STD_LOGIC_VECTOR (2 downto 0);
        Clk : in STD_LOGIC;
        En : in STD_LOGIC;
        Reset : in STD_LOGIC;
        Data_IN : in STD_LOGIC_VECTOR(3 downto 0);
        Data_Bus : out BUS_8_4bit);
end component;

component instruction_decoder is
    Port (
    clk : in std_logic;
    reset : in std_logic;
    InstructionBus : in BUS_13bit;
    RegForJump : in STD_LOGIC_VECTOR(3 downto 0);
    RegisterEnable :out STD_LOGIC_VECTOR (2 downto 0);
    LoadSelect : out std_logic_vector(1 downto 0);
    ImmediateValue : out std_logic_vector(3 downto 0);
    RegisterSelectA : out std_logic_vector(2 downto 0);
    RegisterSelectB : out std_logic_vector(2 downto 0);
    Add_Sub_Select : out std_logic;
    JumpFlag : out std_logic;
    JumpAddress : out std_logic_vector(2 downto 0);
    Add_Sub_En : out std_logic;
    Compare_En : out std_logic;
    Mul_En     : out std_logic
);
end component;

component program_rom is
    port (
        address : in std_logic_vector(2 downto 0);
        instruction : out BUS_13bit
    );
end component;
```

```
    signal muxoutput0               : std_logic_vector(3 downto 0);
    signal muxoutput1               : std_logic_vector(3 downto 0);

    signal instruction_bus          : BUS_13bit;
    signal reg_enable               : STD_LOGIC_VECTOR (2 downto 0);
    signal add_sub_select           : std_logic;
    signal mux3_select              : bus_1_4bit;
    signal mux33_select             : std_logic;
    signal load_select              : std_logic_vector(1 downto 0);
    signal jump_flag                : std_logic;
    signal zero_flag                : std_logic;
    signal reg_a_data               : bus_1_4bit;
    signal reg_b_data               : bus_1_4bit;

    signal alu_result               : bus_1_4bit;
    signal alu_overflow             : std_logic;
    signal pc_next                  : std_logic_vector(2 downto 0);

    signal regA                     :std_logic_vector(2 downto 0);
    signal regB                     :std_logic_vector(2 downto 0);
    signal jumpAddress              :std_logic_vector(2 downto 0);
    signal regdata                  : bus_1_4bit;
    signal Databus0                 :std_logic_vector(3 downto 0);
    signal regoutput                : BUS_8_4bit;

    signal mux_3_4bus               : BUS_3_4bit;
--    signal mux_2_3bus               : BUS_2_3bit;
    signal slw_clk                  : STD_LOGIC;
    signal seven_seg_out            : std_logic_vector(6 downto 0);
    signal seven_seg_in             : STD_LOGIC_VECTOR (3 downto 0);
    signal compare0                 : std_logic_vector(2 downto 0);

    signal add_sub_enable           : std_logic;
    signal comparator_enable        : std_logic;
    signal Mul_enable               : std_logic;
    signal Mul_out                  : std_logic_vector(7 downto 0);

begin

    Seven_Segment_Display : LUT_16_7
    port map(
           I =>seven_seg_in,
           O =>seven_seg_out
           );
```

```
    Slow_Clk0 : slow_Clk
    port map(
        Clk_in => clk,
        Clk_out => slw_clk
        );

    multiplier : Multiplier_4
    port map(
            A => muxoutput0,
            B => muxoutput1,
            Y => Mul_out,
            EN=> Mul_enable
            );
mux_3_4bus(2)<= Mul_out(3 downto 0);

    four_bit_add_sub : addsub4
        port map (
            a =>muxoutput0,
            b =>muxoutput1,
            sub =>add_sub_select,
            en=>add_sub_enable,
            result =>mux_3_4bus(1),
            carry_out =>alu_overflow,
            zero=>zero_flag
        );



    pc : program_counter
        port map (
            clk=> slw_clk ,
            reset => reset ,
            nextaddress => pc_next,
            jump_flag => jump_flag ,
            address_to_jump => jumpAddress
        );

    two_way_4bit_mux : Mux_3_4bit
        port map
          ( D =>mux_3_4bus,
           S =>load_select,
           Y => databus0);
```

```
eight_way_4bit_mux0 : Mux_8_4bit
    port map (
       D => regoutput,
       S =>regA,
       EN =>'1',
       Y=>muxoutput0
    );

   eight_way_4bit_mux1 : Mux_8_4bit
    port map (
       D => regoutput,
       S =>regB,
       EN =>'1',
       Y=>muxoutput1
    );

register_bank : RegBank
    port map (
       RegSel => reg_enable,
       Clk =>slw_clk,
       En =>'1',
       Reset =>reset,
       Data_IN =>databus0,
       Data_Bus =>regoutput
       );

instruction_decoder0 : instruction_decoder
    port map (
    clk             =>slw_clk,
    reset           =>reset,
    InstructionBus  =>instruction_bus,
    RegForJump      =>regoutput(0),
    RegisterEnable  =>reg_enable,
    LoadSelect      =>load_select,
    ImmediateValue  =>mux_3_4bus(0),
    RegisterSelectA =>regA,
    RegisterSelectB =>regB,
    Add_Sub_Select  =>add_sub_select,
    JumpFlag        =>jump_flag,
    JumpAddress     =>jumpAddress,
    Add_Sub_En      =>add_sub_enable,
    Compare_En      =>comparator_enable,
    Mul_En          =>Mul_enable
    );
comparator: Comparator_4_bit
```

```
    port map(
        A =>muxoutput0,
        B =>muxoutput1,
        EN =>comparator_enable,
        Equal=>compare0(0),
        Greater =>compare0(1),
        Smaller =>compare0(2)
    );
    program_rom_inst : program_rom
        port map (
            address => pc_next,
            instruction => instruction_bus
        );

    overflow_led <= alu_overflow;
    zero_led <= zero_flag;
    seven_seg_in <= regoutput(7);

    LEDs <= regoutput(7);
    seven_seg <= seven_seg_out;
    compare<=compare0;

--    j_address <= pc_next;
--    j_flag <= jump_flag;
--    Reg_Zero_out <= regoutput(0);
end architecture rtl;
```

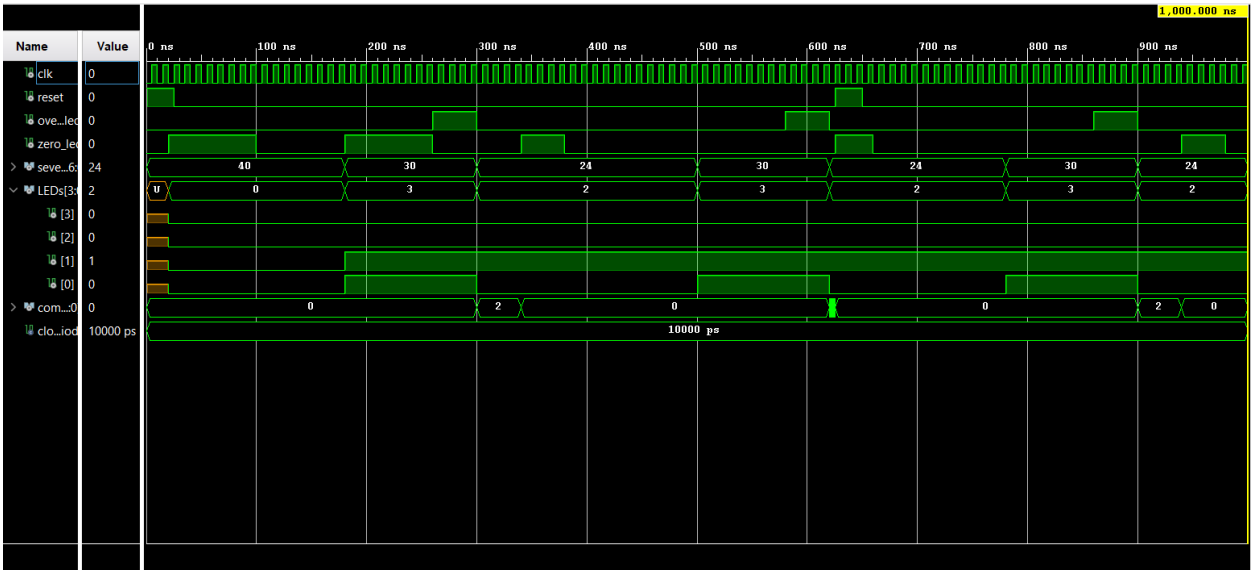
Simulation File for Nano Processor

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.bus_system.all;

entity nanoprocessor_tb is
end entity nanoprocessor_tb;

architecture tb of nanoprocessor_tb is
    component nanoprocessor is
        port (
            clk : in std_logic;
            reset : in std_logic;
            overflow_led : out std_logic;
            zero_led : out std_logic;
```

```
            seven_seg : out std_logic_vector(6 downto 0);
            LEDs : out std_logic_vector(3 downto 0);
            compare:out std_logic_vector(2 downto 0);
            an : out std_logic_vector(3 downto 0):="1110"
--            j_flag: out std_logic;
--            j_address: out std_logic_vector(2 downto 0);
--            Reg_Zero_out : out std_logic_vector(3 downto 0)

        );
    end component nanoprocessor;

    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal overflow_led : std_logic;
    signal zero_led : std_logic;
    signal seven_seg : std_logic_vector(6 downto 0);
    signal LEDs : std_logic_vector(3 downto 0);
    signal compare:std_logic_vector( 2 downto 0);
--    signal j_flag:  std_logic;
--    signal j_address: std_logic_vector(2 downto 0);
--    signal Reg_Zero_out : std_logic_vector(3 downto 0);
    constant clock_period : time := 10 ns;
begin
    uut : nanoprocessor
        port map (
            clk => clk,
            reset => reset,
            overflow_led => overflow_led,
            zero_led => zero_led,
            seven_seg => seven_seg,
            LEDs => LEDs,
            compare=>compare
--            j_flag=>j_flag,
--            j_address=>j_address,
--            Reg_Zero_out=>Reg_Zero_out

        );

    clk_process : process
    begin
        clk <= '0';
        wait for clock_period / 2;
        clk <= '1';
        wait for clock_period / 2;
    end process;
```

```
    Reset_Process : PROCESS BEGIN
        Reset <= '1';
        WAIT FOR 25 NS;
        Reset <= '0';
        WAIT FOR 600 NS;
        Reset <= '1';
    END PROCESS;
end architecture tb;
```

Timing Diagram for nano processor

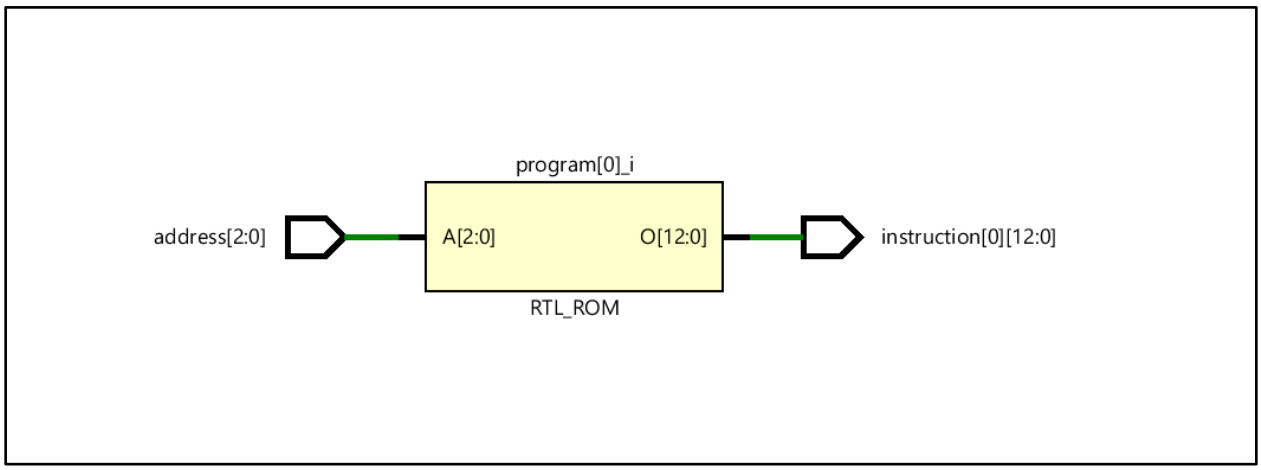


Bus System(Bus_system.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
PACKAGE Bus_System IS
        TYPE BUS_8_4bit IS ARRAY(0 to 7) of STD_LOGIC_VECTOR(3 downto 0);
        TYPE BUS_2_3bit IS ARRAY(0 to 1) of STD_LOGIC_VECTOR(2 downto 0);
        TYPE BUS_2_4bit IS ARRAY(0 to 1) of STD_LOGIC_VECTOR(3 downto 0);
        TYPE BUS_1_4bit IS ARRAY(0 to 3) of STD_LOGIC;
        TYPE BUS_3_4bit IS ARRAY(0 to 2) of STD_LOGIC_VECTOR(3 downto 0);

        TYPE BUS_13bit IS ARRAY(0 to 0) of   STD_LOGIC_VECTOR(12 downto 0);
        TYPE BUS_1_12bit IS ARRAY(0 to 11) of   STD_LOGIC;
        TYPE BUS_2bit IS ARRAY(0 to 1) of STD_LOGIC;
        TYPE BUS_1_3bit IS ARRAY(0 to 2) of STD_LOGIC;

END PACKAGE;
```

Program ROM

Elaborated Design Schematic



Design Source Code (program_rom.vhd)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use WORK.Bus_System.ALL;

entity program_rom is
    port (
        address : in std_logic_vector(2 downto 0);
        instruction : out BUS_13bit
    );
end entity program_rom;

architecture rtl of program_rom is
    type program_memory is array (0 to 7) of std_logic_vector(12 downto 0);
    constant program : program_memory := (


        "0000010000001",  -- Load 1 int register 1
```

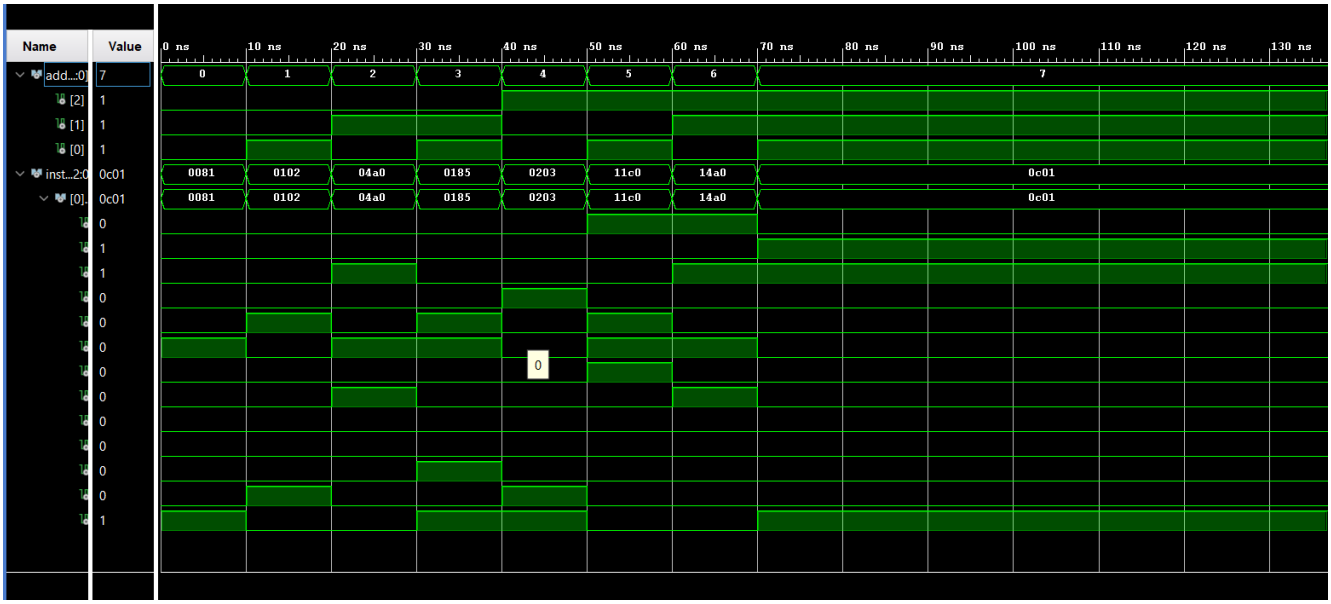
```
        "0000100000010",  -- Load 2 into register 2
        "0010010100000",  -- Add the contents of registers 1 and 2,


        "0000110000101",  -- Load 5 int register 3
        "0001000000011",  -- Load 3 int register 4
        "1000111000000",  -- SUB the contents of registers 3 and 4,
        --"1100111000000",  -- MUL the contents of registers 1 and 2,
        "1010010100000", -- comparator
        "0110000000001"  -- Jump to address,


    );
begin
    instruction(0) <= program(to_integer(unsigned(address)));
end architecture rtl;
```

Simulation File for Nano Processor(program_rom_tb.vhd)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.bus_system.all;

entity program_rom_tb is
end entity program_rom_tb;

architecture tb of program_rom_tb is
    component program_rom is
        port (
            address : in std_logic_vector(2 downto 0);
            instruction : out BUS_13bit
        );
    end component program_rom;

    signal address : std_logic_vector(2 downto 0) := (others => '0');
    signal instruction : BUS_13bit;

begin
    uut : program_rom
        port map (
            address => address,
```

```
            instruction => instruction
        );

    test_proc : process
    begin
        -- Test the program ROM
        address <= "000";
        wait for 10 ns;


        address <= "001";
        wait for 10 ns;


        address <= "010";
        wait for 10 ns;


        address <= "011";
        wait for 10 ns;


        address <= "100";
        wait for 10 ns;


        address <= "101";
        wait for 10 ns;

        address <= "110";
        wait for 10 ns;

        address <= "111";
        wait for 10 ns;

        wait;
    end process test_proc;
end architecture tb;
```

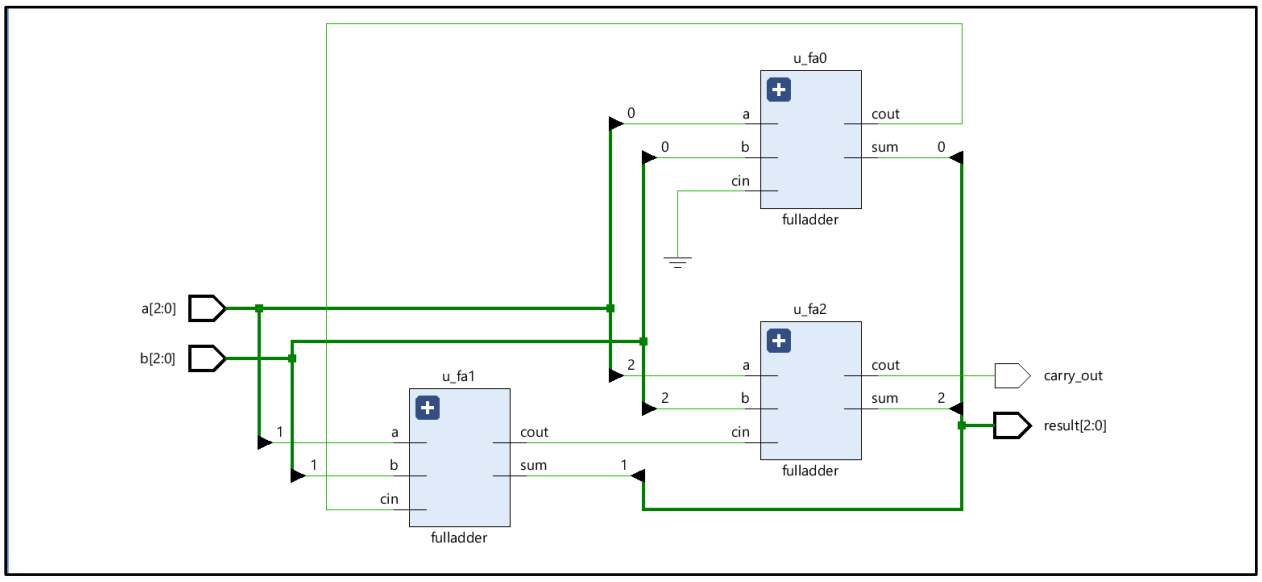
Timing Diagram for program rom



3-Bit Adder

3-bit adder is for program counter to create next address. After PC sends current address to rom then using 3-bit adder add 1 for the current address and store in PC.

Elaborated Design Schematic



Design Source Code for 3-Bit Adder (adder3.vhd)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity adder3 is
    port (
        a : in std_logic_vector(2 downto 0);
        b : in std_logic_vector(2 downto 0);
        result : out std_logic_vector(2 downto 0);
        carry_out : out std_logic
    );
end adder3;
```

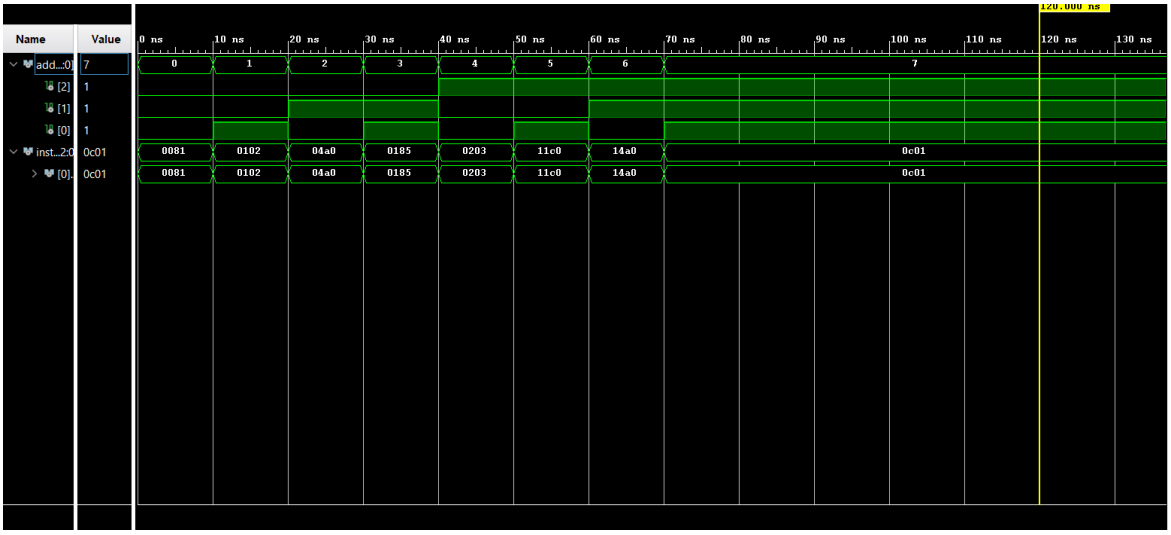
```
architecture behavioral of adder3 is

    component fulladder
        port (
            a, b, cin : in std_logic;
            sum, cout : out std_logic
        );
    end component;

    signal c : std_logic_vector(3 downto 0);

begin

    u_fa0 : fulladder
        port map (
            a => a(0),
            b => b(0),
            cin => '0',
            sum => result(0),
            cout => c(1)
        );

    u_fa1 : fulladder
        port map (
            a => a(1),
            b => b(1),
            cin => c(1),
            sum => result(1),
            cout => c(2)
        );

    u_fa2 : fulladder
        port map (
            a => a(2),
            b => b(2),
            cin => c(2),
            sum => result(2),
            cout => c(3)
        );

    carry_out <= c(3);

end behavioral;
```

Simulation File for 3-bit Adder(adder3_tb.vhd)

```
library ieee;
use ieee.std_logic_1164.all;

entity adder3_tb is
end adder3_tb;

architecture behavioral of adder3_tb is

    component adder3
        port (
            a : in std_logic_vector(2 downto 0);
            b : in std_logic_vector(2 downto 0);
            result : out std_logic_vector(2 downto 0);
            carry_out : out std_logic
        );
    end component;

    signal a_tb : std_logic_vector(2 downto 0);
    signal b_tb : std_logic_vector(2 downto 0);
    signal result_tb : std_logic_vector(2 downto 0);
    signal carry_out_tb : std_logic;

begin

    uut : adder3
        port map (
            a => a_tb,
            b => b_tb,
            result => result_tb,
            carry_out => carry_out_tb
        );

    stimulus : process
    begin
        -- Test case 1
        a_tb <= "000";
        b_tb <= "000";
        wait for 10 ns;

        -- Test case 2
```

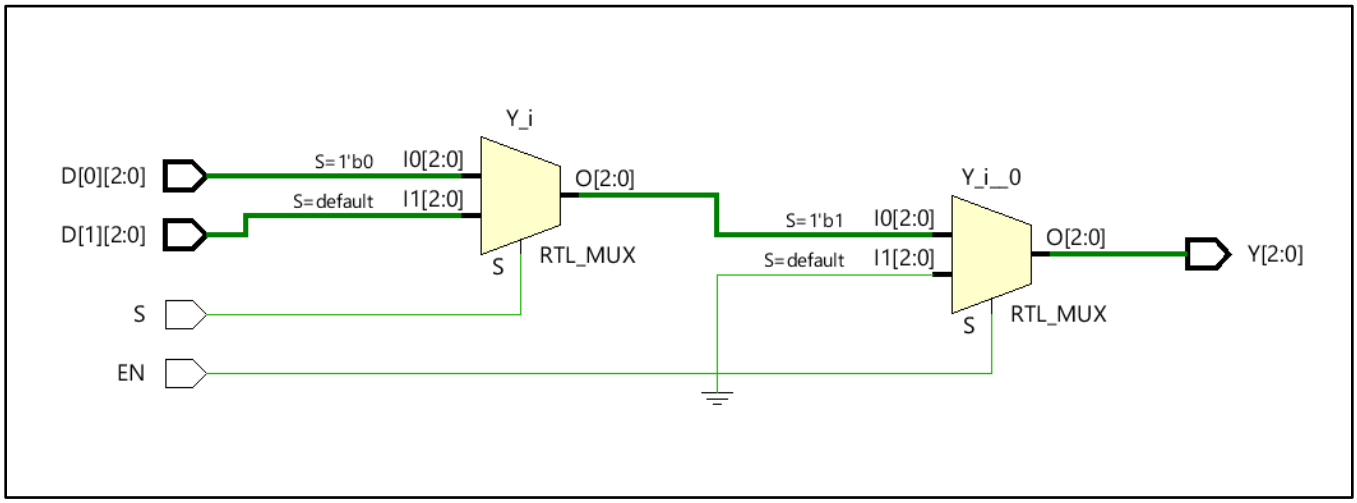
```
        a_tb <= "001";
        b_tb <= "010";
        wait for 10 ns;

        -- Test case 3
        a_tb <= "111";
        b_tb <= "001";
        wait for 10 ns;

        -- Add more test cases as needed

        -- End the simulation
        wait;
    end process stimulus;

end behavioral;
```

Timing Diagram for 3-bit Adder



2-Way 3-Bit Multiplexer

Elaborated Design Schematic



Design Source Code (Mux_2_3bit.vhd)

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

entity Mux_2_3bit is
    port (
           D: in BUS_2_3bit;
           S : in STD_LOGIC;
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR(2 downto 0));
end Mux_2_3bit;

architecture Behavioral of Mux_2_3bit is
signal D0: STD_LOGIC_VECTOR(2 downto 0);
signal D1: STD_LOGIC_VECTOR(2 downto 0);
```

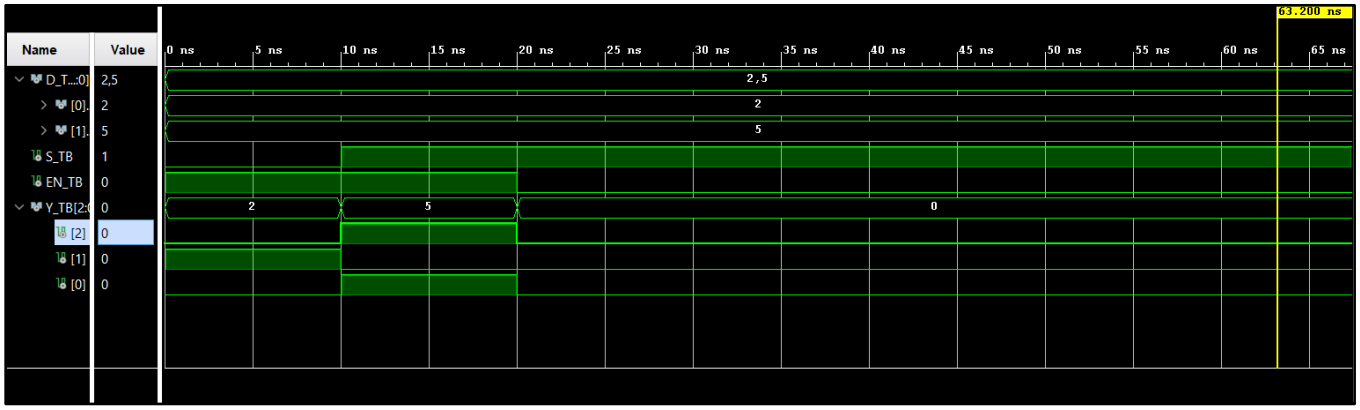
```
begin
    D0 <= D(0);
    D1 <= D(1);

    process(D0,D1, S, EN)

    begin
        if (EN = '1') then
            if (S = '0') then
                Y <=D0;
            else if (S='1') then
                Y <= D1;
            end if;
            end if;
        else
            Y<="000";
        end if;
    end process;
end Behavioral;
```

Simulation File for 2-Way 3-Bit Multiplexer (Mux_2_3bit_TB.vhd)

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

entity Mux_2_3bit_TB is
end Mux_2_3bit_TB;

architecture Behavioral of Mux_2_3bit_TB is
    component Mux_2_3bit
        port (
            D: in BUS_2_3bit;
            S: in STD_LOGIC;
            EN: in STD_LOGIC;
            Y: out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;

    signal D_TB: BUS_2_3bit;
    signal S_TB: STD_LOGIC;
```

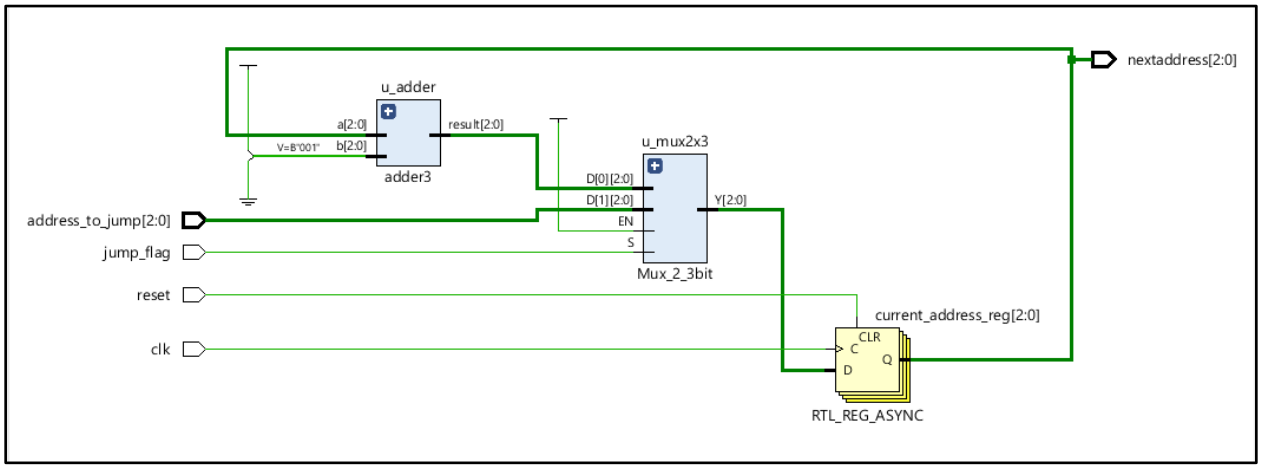
```
    signal EN_TB: STD_LOGIC;
    signal Y_TB: STD_LOGIC_VECTOR(2 downto 0);

begin
    UUT: Mux_2_3bit
        port map(
            D => D_TB,
            S => S_TB,
            EN => EN_TB,
            Y => Y_TB
        );

    -- Stimulus process
    stimulus_proc: process
    begin
        -- Test case 1: Enable = '1', S = '0'
        D_TB <= ("010", "101");
        S_TB <= '0';
        EN_TB <= '1';
        wait for 10 ns;
        assert(Y_TB = "010") report "Test case 1 failed" severity error;

        -- Test case 2: Enable = '1', S = '1'
        D_TB <= ("010", "101");
        S_TB <= '1';
        EN_TB <= '1';
        wait for 10 ns;
        assert(Y_TB = "101") report "Test case 2 failed" severity error;

        -- Test case 3: Enable = '0'
        D_TB <= ("010", "101");
        S_TB <= '1';
        EN_TB <= '0';
        wait for 10 ns;
        assert(Y_TB = "000") report "Test case 3 failed" severity error;

        wait;
    end process;
end Behavioral;
```

Timing Diagram for 2-Way 3-Bit Multiplexer



Program Counter

Elaborated Design Schematic



Design Source Code (program_counter.vhd)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.Bus_System.all;

entity program_counter is
    port (
        clk : in std_logic;
        reset : in std_logic;
        nextaddress : out std_logic_vector(2 downto 0);
        jump_flag : in std_logic;
        address_to_jump : in std_logic_vector(2 downto 0)
    );
end program_counter;

architecture behavioral of program_counter is

    component adder3
        port (
            a : in std_logic_vector(2 downto 0);
            b : in std_logic_vector(2 downto 0);
            result : out std_logic_vector(2 downto 0);
            carry_out : out std_logic
        );
    end component;
```

```
    component Mux_2_3bit is
        port (
            D : in BUS_2_3bit;
            S : in STD_LOGIC;
            EN : in STD_LOGIC;
            Y : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;

    signal current_address : std_logic_vector(2 downto 0);
    signal next_address : std_logic_vector(2 downto 0);
    signal jflag : std_logic;
    signal mux_in : BUS_2_3bit;
    signal mux_out : std_logic_vector(2 downto 0);

begin

    u_adder : adder3
        port map (
            a => current_address,
            b => "001",
            result => next_address,
            carry_out => open
        );

    u_mux2x3 : Mux_2_3bit
        port map (
            D => mux_in,
            S => jflag,
            EN => '1',
            Y => mux_out
        );

    mux_in(0) <= next_address;
    mux_in(1) <= address_to_jump;
    jflag <= jump_flag;

    process (clk, reset)
    begin
        if reset = '1' then
            current_address <= "000";
        elsif rising_edge(clk) then
            current_address <= mux_out;
        end if;
    end process;
```

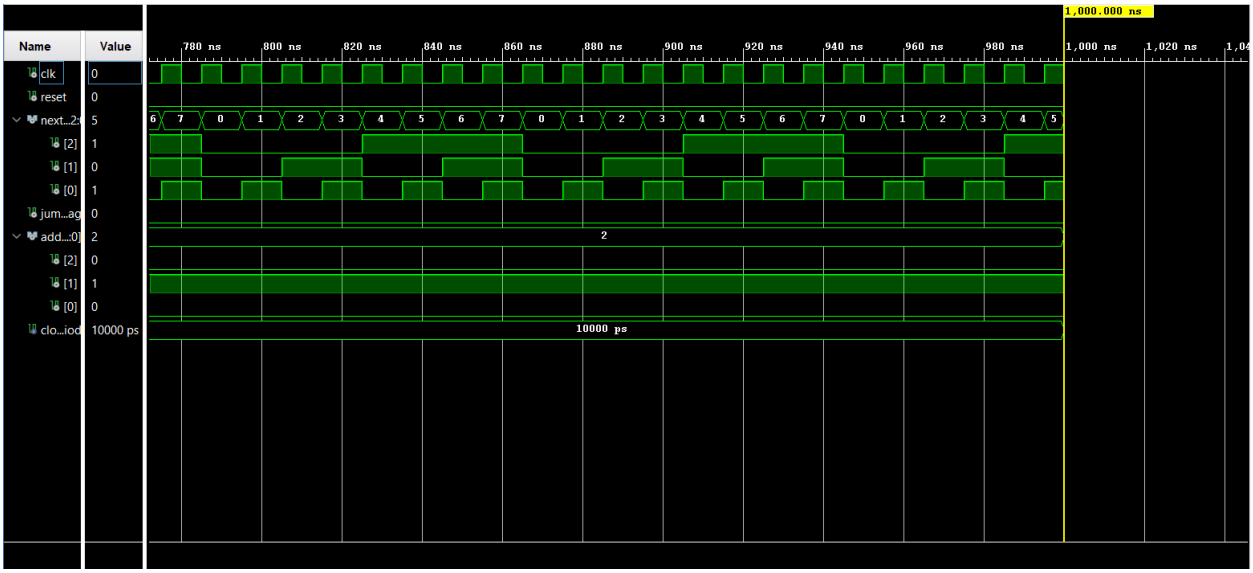
```
    nextaddress <= current_address;

end behavioral;
```

Simulation File for Program Counter(program_counter_tb.vhd)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.Bus_System.all;

entity program_counter_tb is
end program_counter_tb;

architecture behavioral of program_counter_tb is
    component program_counter
        port (
            clk : in std_logic;
            reset : in std_logic;
            nextaddress : out std_logic_vector(2 downto 0);
            jump_flag : in std_logic;
            address_to_jump : in std_logic_vector(2 downto 0)
        );
    end component;

    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal nextaddress : std_logic_vector(2 downto 0);
    signal jump_flag : std_logic := '0';
    signal address_to_jump : std_logic_vector(2 downto 0) := (others => '0');

    constant clock_period : time := 10 ns;
begin
    uut : program_counter
        port map (
            clk => clk,
            reset => reset,
            nextaddress => nextaddress,
            jump_flag => jump_flag,
            address_to_jump => address_to_jump
        );
```

```
    clk_process : process
    begin
        clk <= '0';
        wait for clock_period / 2;
        clk <= '1';
        wait for clock_period / 2;
    end process;

    test_process : process
    begin
        -- Reset the program counter
        reset <= '1';
        wait for 2 * clock_period;
        reset <= '0';

        -- Test normal operation
        wait for 6 * clock_period;
        assert nextaddress = "110" report "Normal operation test failed" severity
error;

        -- Test jump operation
        jump_flag <= '1';
        address_to_jump <= "010";
        wait for clock_period;
        assert nextaddress = "010" report "Jump operation test failed" severity
error;

        -- Test incrementing after jump
        jump_flag <= '0';
        wait for 3 * clock_period;
        assert nextaddress = "011" report "Increment after jump test failed"
severity error;

        wait;
    end process;
end behavioral;
```

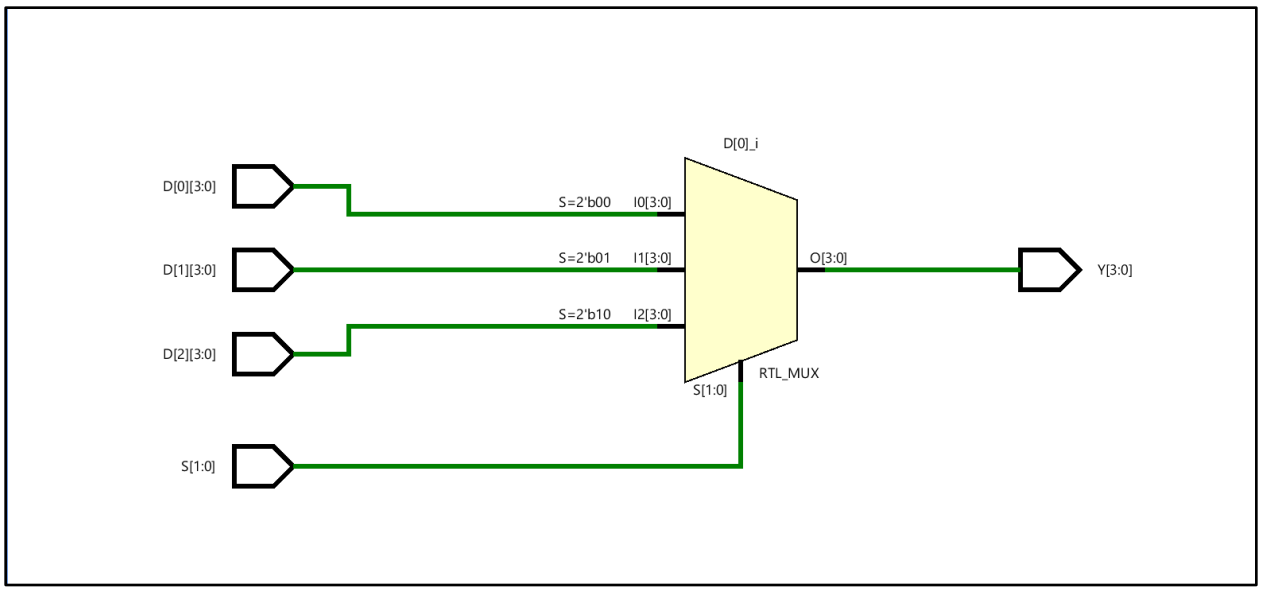
Timing Diagram for Program Counter



3-Way 4-Bit Multiplexer

We replaced 2-way 4-bit Mux using 3 way 4-bit mux because we add comparator and multiplier.

Elaborated Design Schematic



Design Source Code (Mux_3_4bit.vhd)

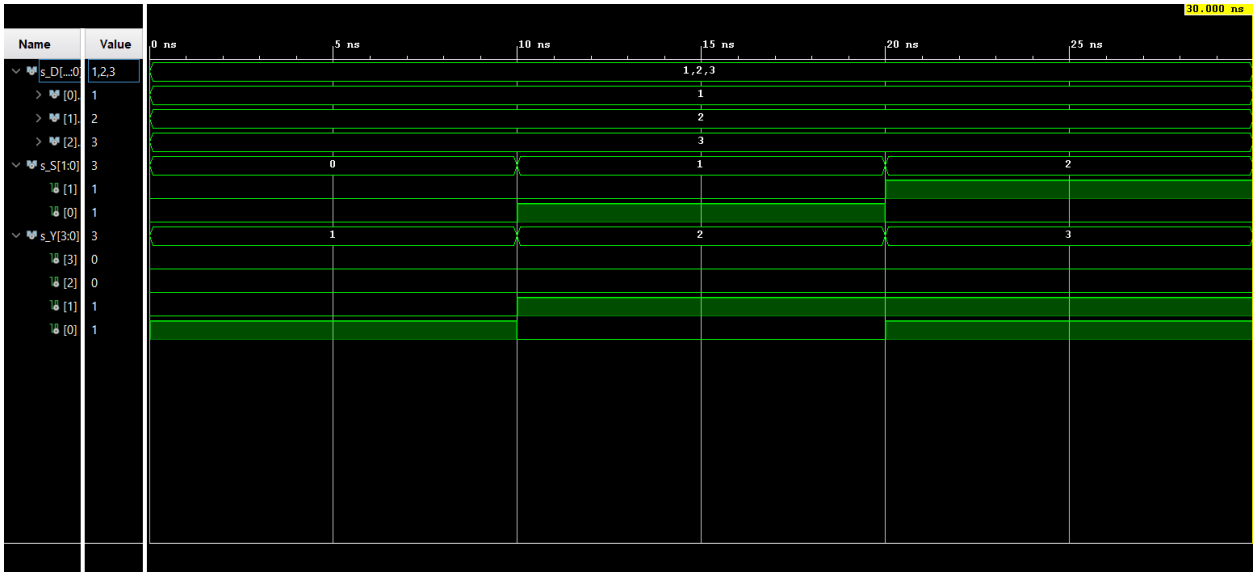
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Numeric_STD.ALL;
use WORK.Bus_System.ALL;

entity Mux_3_4bit is
    port (
           D: in BUS_3_4bit;
           S : in STD_LOGIC_VECTOR(1 downto 0);
             --EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR(3 downto 0));
end Mux_3_4bit;

architecture Behavioral of Mux_3_4bit is
begin

Y <= D(TO_INTEGER(UNSIGNED(S)));
end Behavioral;
```

Simulation File for 3-Way 4-Bit Multiplexer (Mux_3_4bit_TB.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Numeric_STD.ALL;
use WORK.Bus_System.ALL;

entity Mux_3_4bit_tb is
end Mux_3_4bit_tb;

architecture Behavioral of Mux_3_4bit_tb is
    signal s_D : BUS_3_4bit := (others => (others => '0'));
    signal s_S : STD_LOGIC_VECTOR(1 downto 0) := (others => '0');
    signal s_Y : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
    --signal EN:std_logic;
    component Mux_3_4bit is
        port (
            D : in BUS_3_4bit;
            S : in STD_LOGIC_VECTOR(1 downto 0);
            --EN:in STD_logic;
            Y : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

begin
    uut : Mux_3_4bit port map (
        D => s_D,
        S => s_S,
        Y => s_Y
    );

    stim_proc : process
    begin

        -- Test case 1: Select D(0)
        s_D <= (X"1", X"2", X"3");
        s_S <= "00";
        wait for 10 ns;
        assert s_Y = X"1" report "Test case 1 failed" severity error;

        -- Test case 2: Select D(1)
        s_S <= "01";
        wait for 10 ns;
        assert s_Y = X"2" report "Test case 2 failed" severity error;
```

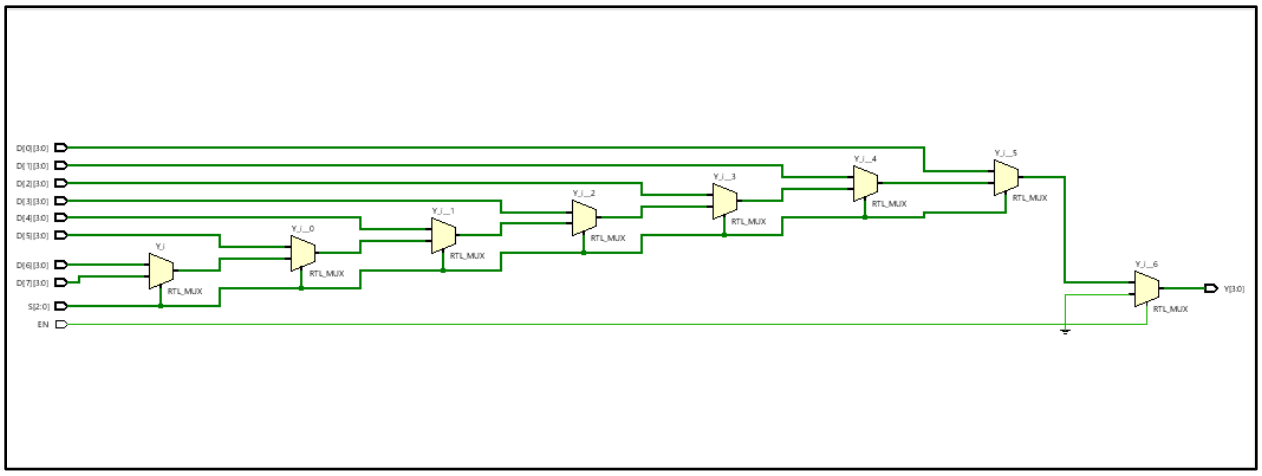
```
        -- Test case 3: Select D(2)
        s_S <= "10";
        wait for 10 ns;
        assert s_Y = X"3" report "Test case 3 failed" severity error;

        -- Test case 4: Invalid selection
        s_S <= "11";
        wait for 10 ns;
        assert s_Y = X"0" report "Test case 4 failed" severity error;

        report "All tests passed";
        wait;
    end process;
end Behavioral;
```

Timing Diagram for 3-Way 4-Bit Multiplexer



8-Way 4-Bit Multiplexer

Elaborated Design Schematic



Design Source Code (Mux_8_4bit)

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

entity Mux_8_4bit is
    port ( D : in BUS_8_4bit;
           S : in STD_LOGIC_VECTOR(2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR(3 downto 0));
end Mux_8_4bit;

architecture Behavioral of Mux_8_4bit is
begin
    process(D, S, EN)
    begin
        if (EN = '1') then
            if (S = "000") then
                Y <= D(0);
            else if (S = "001") then
                Y <= D(1);
            else if (S = "010") then
```

```
                Y <= D(2);
            else if (S = "011") then
                Y <= D(3);
            else if (S = "100") then
                Y <= D(4);
            else if (S = "101") then
                Y <= D(5);
            else if (S = "110") then
                Y <= D(6);
            else
                Y <= D(7);
            end if;
            end if;
            end if;
            end if;
            end if;
            end if;
            end if;
        else
            Y <= "0000";
        end if;
    end process;
end Behavioral;
```

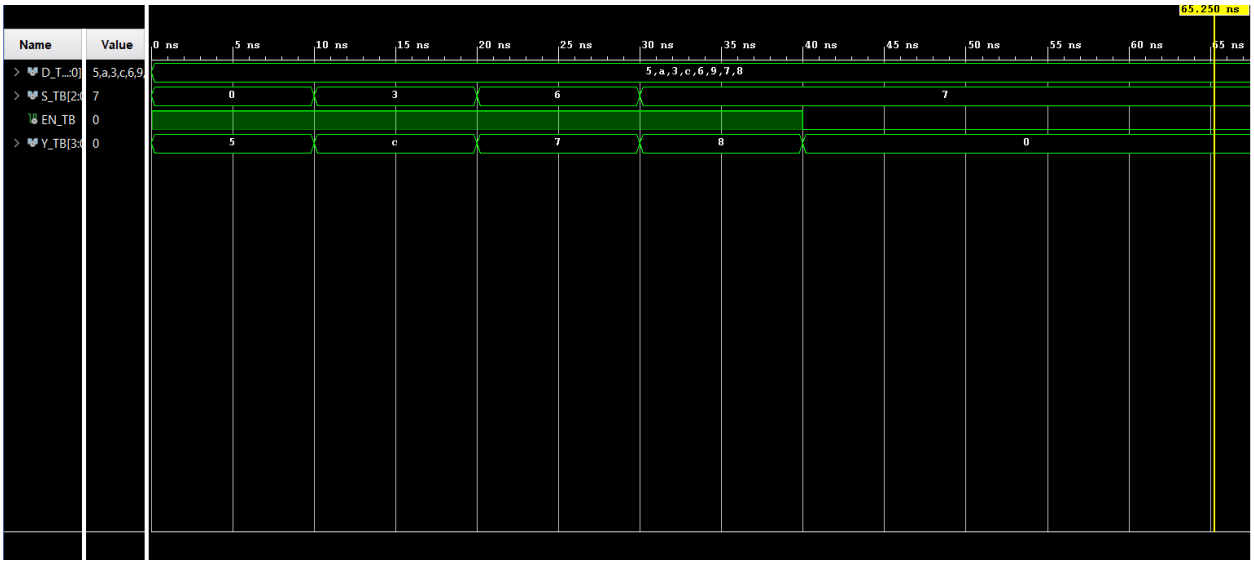
Simulation File for 8-Way 4-Bit Multiplexer(Mux_8_4bit_TB.vhd)

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

entity Mux_8_4bit_TB is
end Mux_8_4bit_TB;

architecture Behavioral of Mux_8_4bit_TB is
    component Mux_8_4bit
        port (
            D: in BUS_8_4bit;
            S: in STD_LOGIC_VECTOR(2 downto 0);
            EN: in STD_LOGIC;
            Y: out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    signal D_TB: BUS_8_4bit;
    signal S_TB: STD_LOGIC_VECTOR(2 downto 0);
```

```
    signal EN_TB: STD_LOGIC;
    signal Y_TB: STD_LOGIC_VECTOR(3 downto 0);

begin
    UUT: Mux_8_4bit
        port map(
            D => D_TB,
            S => S_TB,
            EN => EN_TB,
            Y => Y_TB
        );

    -- Stimulus process
    stimulus_proc: process
    begin
        -- Test case 1: Enable = '1', S = "000"
        D_TB <= ("0101", "1010", "0011", "1100", "0110", "1001", "0111", "1000");
        S_TB <= "000";
        EN_TB <= '1';
        wait for 10 ns;
        assert(Y_TB = "0101") report "Test case 1 failed" severity error;

        -- Test case 2: Enable = '1', S = "011"
        D_TB <= ("0101", "1010", "0011", "1100", "0110", "1001", "0111", "1000");
        S_TB <= "011";
        EN_TB <= '1';
        wait for 10 ns;
        assert(Y_TB = "1100") report "Test case 2 failed" severity error;

        -- Test case 3: Enable = '1', S = "110"
        D_TB <= ("0101", "1010", "0011", "1100", "0110", "1001", "0111", "1000");
        S_TB <= "110";
        EN_TB <= '1';
        wait for 10 ns;
        assert(Y_TB = "0111") report "Test case 3 failed" severity error;

        -- Test case 4: Enable = '1', S = "111"
        D_TB <= ("0101", "1010", "0011", "1100", "0110", "1001", "0111", "1000");
        S_TB <= "111";
        EN_TB <= '1';
        wait for 10 ns;
        assert(Y_TB = "1000") report "Test case 4 failed" severity error;

        -- Test case 5: Enable = '0'
        D_TB <= ("0101", "1010", "0011", "1100", "0110", "1001", "0111", "1000");
```

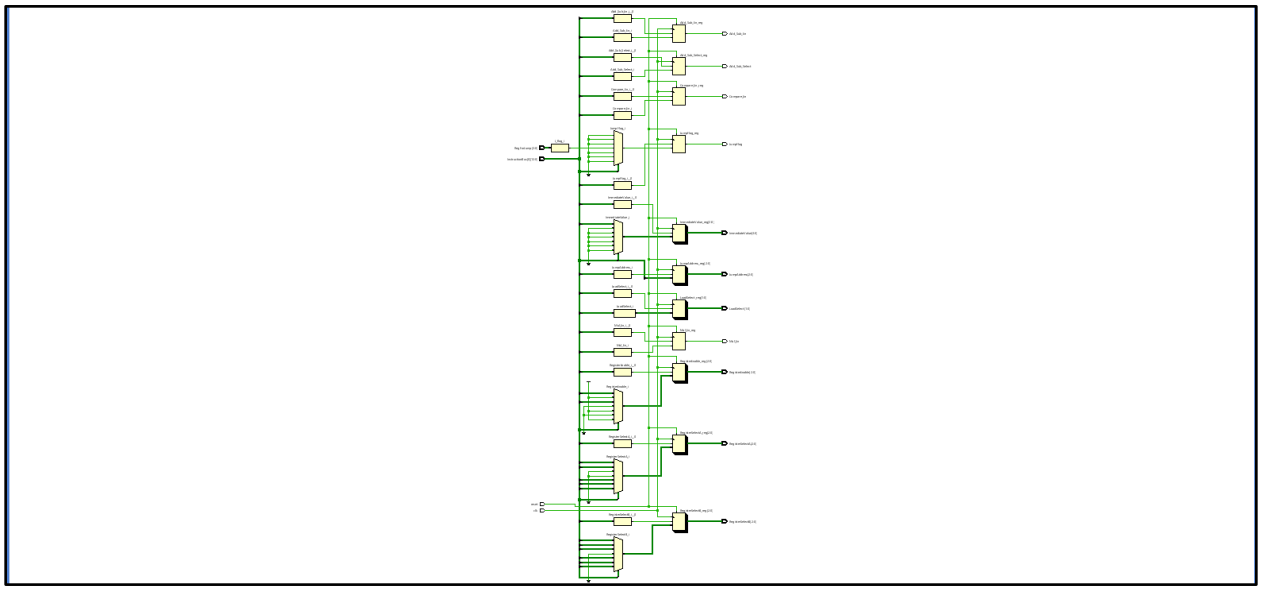
```
        S_TB <= "111";
        EN_TB <= '0';
        wait for 10 ns;
        assert(Y_TB = "0000") report "Test case 5 failed" severity error;

        wait;
    end process;
end Behavioral;
```

Timing Diagram for 2-Way 3-Bit Multiplexer



Instruction Decoder

Elaborated Design Schematic



Design Source Code (Instruction_Decoder.vhd)

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

entity Instruction_Decoder is
    Port (
        clk            : in std_logic;
        reset          : in std_logic;
        InstructionBus : in BUS_13bit;
        RegForJump     : in STD_LOGIC_VECTOR(3 downto 0);
        RegisterEnable : out STD_LOGIC_VECTOR(2 downto 0);
        LoadSelect     : out std_logic_vector(1 downto 0);
        ImmediateValue : out std_logic_vector(3 downto 0);
        RegisterSelectA: out std_logic_vector(2 downto 0);
        RegisterSelectB: out std_logic_vector(2 downto 0);
        Add_Sub_Select : out std_logic;
        JumpFlag       : out std_logic;
```

```
        JumpAddress    : out std_logic_vector(2 downto 0);
        Add_Sub_En     : out std_logic;
        Compare_En     : out std_logic;
        Mul_En         : out std_logic
    );
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is
    signal opcode    : std_logic_vector(2 downto 0);
    signal reg_a     : std_logic_vector(2 downto 0);
    signal reg_b     : std_logic_vector(2 downto 0);
    signal Jaddress  : std_logic_vector(2 downto 0);
    signal J_flag    : std_logic;
begin
    opcode <= InstructionBus(0)(12 downto 10);
    reg_a  <= InstructionBus(0)(9 downto 7);
    reg_b  <= InstructionBus(0)(6 downto 4);

    -- Calculate jump address
    J_flag    <=  '1' when RegForJump = "0000" else '0';
    Jaddress  <=  InstructionBus(0)(2 downto 0);

    process (clk, reset)
    begin
        if reset = '1' then
            JumpFlag        <= '0';
            JumpAddress     <= (others => '0');
            RegisterEnable  <= (others => '0');
            LoadSelect      <= "00";
            ImmediateValue  <= (others => '0');
            RegisterSelectA <= (others => '0');
            RegisterSelectB <= (others => '0');
            Add_Sub_Select  <= '0';
            Add_Sub_En      <= '0';
            Compare_En      <= '0';
            Mul_En          <= '0';

        elsif rising_edge(clk) then
            case opcode is
                when "000" => -- MOVI Instruction
                    RegisterEnable  <= reg_a;
                    LoadSelect      <= "00";
                    ImmediateValue  <= InstructionBus(0)(3 downto 0);
                    RegisterSelectA <= reg_a;
                    RegisterSelectB <= reg_b;
```

```
        Add_Sub_Select  <= '0';
        Add_Sub_En      <= '0';
        Compare_En      <= '0';
        JumpFlag        <= '0';
        Mul_En          <= '0';

    when "001" => -- ADD Instruction
        RegisterEnable  <= "111";
        LoadSelect      <= "01";
        ImmediateValue  <= (others => '0');
        RegisterSelectA <= reg_a;
        RegisterSelectB <= reg_b;
        Add_Sub_Select  <= '0';
        Add_Sub_En      <= '1';
        Compare_En      <= '0';
        Mul_En          <= '0';
        JumpFlag        <= '0';

    when "010" => -- NEG Instruction
        RegisterEnable  <= reg_a;
        LoadSelect      <= "01";
        ImmediateValue  <= (others => '0');
        RegisterSelectA <= (others => '0');
        RegisterSelectB <= reg_a;
        Add_Sub_Select  <= '1';
        Add_Sub_En      <= '1';
        Compare_En      <= '0';
        Mul_En          <= '0';
        JumpFlag        <= '0';

    when "011" => -- JZR Instruction
        RegisterEnable  <= "000";
        LoadSelect      <= "00";
        ImmediateValue  <= (others => '0');
        RegisterSelectA <= (others => '0');
        RegisterSelectB <= (others => '0');
        Add_Sub_Select  <= '0';
        JumpFlag        <= J_flag;
        JumpAddress     <= Jaddress;
        Add_Sub_En      <= '0';
        Compare_En      <= '0';
        Mul_En          <= '0';

    when "100" => -- SUB Instruction
        RegisterEnable  <= "111";
```

```
                    LoadSelect      <= "01";
                    ImmediateValue  <= (others => '0');
                    RegisterSelectA <= reg_a;
                    RegisterSelectB <= reg_b;
                    Add_Sub_Select  <= '1';
                    Add_Sub_En      <= '1';
                    Compare_En      <= '0';
                    Mul_En          <= '0';
                    JumpFlag        <= '0';

                when "101" => -- COM Instruction
                    RegisterEnable  <= "000";
                    LoadSelect      <= "01";
                    ImmediateValue  <= (others => '0');
                    RegisterSelectA <= reg_a;
                    RegisterSelectB <= reg_b;
                    Add_Sub_Select  <= '0';
                    Add_Sub_En      <='0';
                    Compare_En      <='1';
                    Mul_En          <= '0';
                    JumpFlag        <= '0';

                when "110" => -- MUL Instruction
                    RegisterEnable  <= "111";
                    LoadSelect      <= "10";
                    ImmediateValue  <= (others => '0');
                    RegisterSelectA <= reg_a;
                    RegisterSelectB <= reg_b;
                    Add_Sub_Select  <= '0';
                    Add_Sub_En      <= '0';
                    Mul_En          <= '0';
                    Compare_En      <= '0';
                    Mul_En          <= '1';
                    JumpFlag        <= '0';

                when others =>
                    null;
            end case;
        end if;
    end process;
end Behavioral;
```

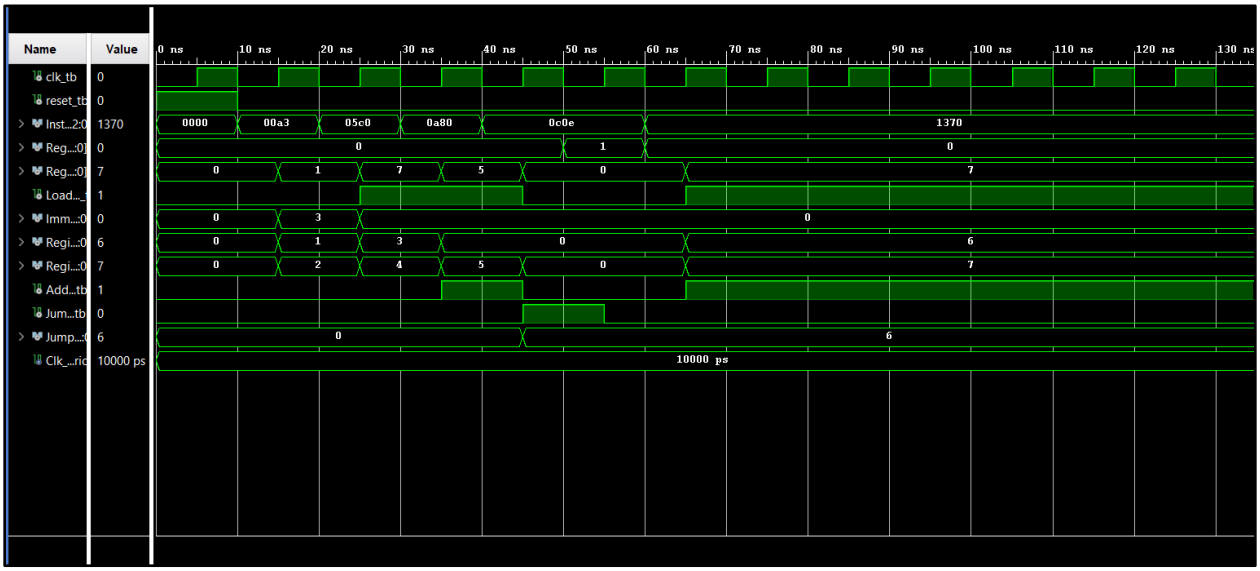
Simulation File for Instruction Decoder (Instruction_Decoder_tb.vhd)

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

entity Instruction_Decoder_tb is
end Instruction_Decoder_tb;

architecture Testbench of Instruction_Decoder_tb is
    -- Component Declaration
    component Instruction_Decoder
        Port (
            clk            : in std_logic;
            reset          : in std_logic;
            InstructionBus : in BUS_13bit;
            RegForJump     : in STD_LOGIC_VECTOR(3 downto 0);
            RegisterEnable : out STD_LOGIC_VECTOR(2 downto 0);
            LoadSelect     : out std_logic_vector(1 downto 0);
            ImmediateValue : out std_logic_vector(3 downto 0);
            RegisterSelectA: out std_logic_vector(2 downto 0);
            RegisterSelectB: out std_logic_vector(2 downto 0);
            Add_Sub_Select : out std_logic;
            JumpFlag       : out std_logic;
            JumpAddress    : out std_logic_vector(2 downto 0)
        );
    end component;

    -- Input Signals
    signal clk_tb         : std_logic := '0';
    signal reset_tb       : std_logic := '0';
    signal InstructionBus_tb: BUS_13bit := (others => (others => '0'));
    signal RegForJump_tb  : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');

    -- Output Signals
    signal RegisterEnable_tb: STD_LOGIC_VECTOR(2 downto 0);
    signal LoadSelect_tb  : std_logic_vector(1 downto 0);
    signal ImmediateValue_tb: std_logic_vector(3 downto 0);
    signal RegisterSelectA_tb: std_logic_vector(2 downto 0);
    signal RegisterSelectB_tb: std_logic_vector(2 downto 0);
    signal Add_Sub_Select_tb: std_logic;
    signal JumpFlag_tb    : std_logic;
    signal JumpAddress_tb : std_logic_vector(2 downto 0);
```

```
    -- Clock Period
    constant Clk_period : time := 10 ns;

begin
    -- Instantiate the Unit Under Test (UUT)
    uut : Instruction_Decoder
        port map (
            clk            => clk_tb,
            reset          => reset_tb,
            InstructionBus => InstructionBus_tb,
            RegForJump     => RegForJump_tb,
            RegisterEnable => RegisterEnable_tb,
            LoadSelect     => LoadSelect_tb,
            ImmediateValue => ImmediateValue_tb,
            RegisterSelectA => RegisterSelectA_tb,
            RegisterSelectB => RegisterSelectB_tb,
            Add_Sub_Select => Add_Sub_Select_tb,
            JumpFlag       => JumpFlag_tb,
            JumpAddress    => JumpAddress_tb
        );

    -- Clock process
    Clk_process : process
    begin
        clk_tb <= '0';
        wait for Clk_period / 2;
        clk_tb <= '1';
        wait for Clk_period / 2;
    end process;

    -- Stimulus process
    stim_proc : process
    begin
        -- Reset the instruction decoder
        reset_tb <= '1';
        wait for Clk_period;
        reset_tb <= '0';

        -- Test case 1: MOVI Instruction
        InstructionBus_tb(0) <= "000" & "001" & "010" & "0011";
        RegForJump_tb <= "0000";
        wait for Clk_period;
        -- Add your assertions here

        -- Test case 2: ADD Instruction
```

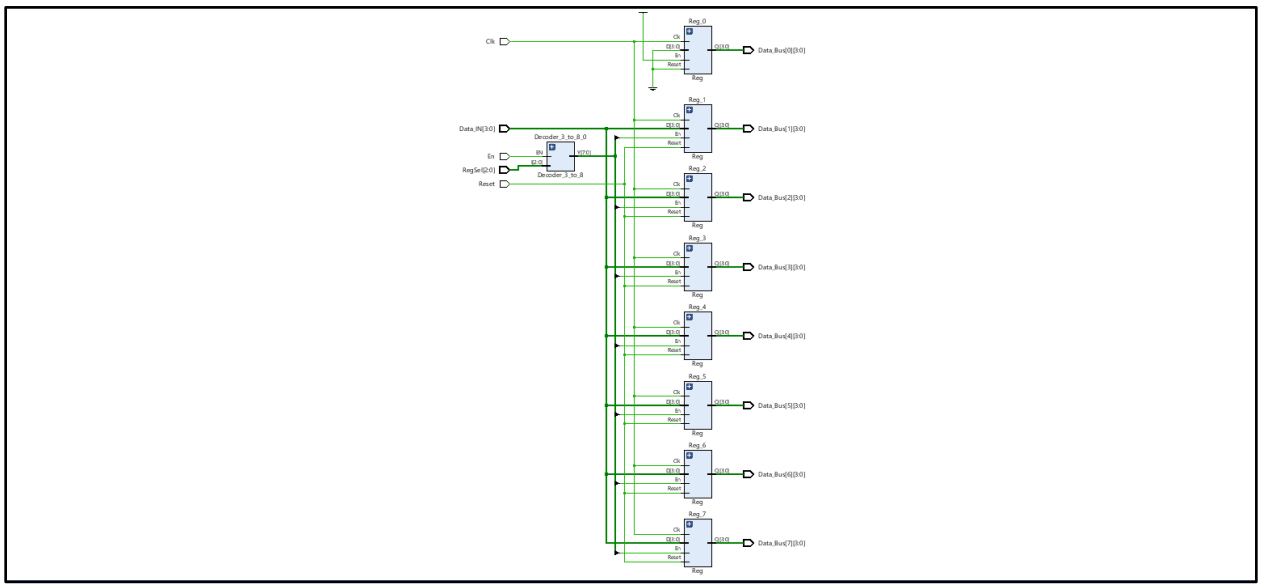
```
        InstructionBus_tb(0) <= "001" & "011" & "100" & "0000";
        RegForJump_tb <= "0000";
        wait for Clk_period;
        -- Add your assertions here

        -- Test case 3: NEG Instruction
        InstructionBus_tb(0) <= "010" & "101" & "000" & "0000";
        RegForJump_tb <= "0000";
        wait for Clk_period;
        -- Add your assertions here

        -- Test case 4: JZR Instruction (Jump condition true)
        InstructionBus_tb(0) <= "011" & "000" & "000" & "111";
        RegForJump_tb <= "0000";
        wait for Clk_period;
        -- Add your assertions here

        -- Test case 5: JZR Instruction (Jump condition false)
        InstructionBus_tb(0) <= "011" & "000" & "000" & "111";
        RegForJump_tb <= "0001";
        wait for Clk_period;
        -- Add your assertions here

        -- Test case 6: SUB Instruction
        InstructionBus_tb(0) <= "100" & "110" & "111" & "0000";
        RegForJump_tb <= "0000";
        wait for Clk_period;
        -- Add your assertions here

        wait; -- End the simulation
    end process;
end Testbench;
```

Timing Diagram for Instruction Decoder



Register Bank

Elaborated Design Schematic



Design Source Code (RegBank.vhd)

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

entity RegBank is
    Port (RegSel : in STD_LOGIC_VECTOR (2 downto 0);
          Clk : in STD_LOGIC;
          En : in STD_LOGIC;
          Reset : in STD_LOGIC;
          Data_IN : in STD_LOGIC_VECTOR (3 downto 0);
          Data_Bus : out BUS_8_4bit);
end RegBank;

architecture Behavioral of RegBank 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 regsel_signal : STD_LOGIC_VECTOR (7 DOWNTO 0);
    signal reg_outputs : BUS_8_4bit; -- Array to hold register outputs

begin
    Decoder_3_to_8_0 : Decoder_3_to_8
    port map (
        I => RegSel,
        En => En,
        Y => regsel_signal
    );

    -- Register R0 is to be hardcoded to 0000
    Reg_0 : Reg
    port map (
        D => "0000",
        En => '1',
        Reset => '0',
        Clk => Clk,
        Q => reg_outputs(0)
    );

    Reg_1 : Reg
    port map (
        D => Data_IN,
        En => regsel_signal(1),
        Reset => Reset,
        Clk => Clk,
        Q => reg_outputs(1)
    );

    Reg_2 : Reg
    port map (
        D => Data_IN,
        En => regsel_signal(2),
        Reset => Reset,
        Clk => Clk,
        Q => reg_outputs(2)
```

```
);

Reg_3 : Reg
port map (
    D => Data_IN,
    En => regsel_signal(3),
    Reset => Reset,
    Clk => Clk,
    Q => reg_outputs(3)
);

Reg_4 : Reg
port map (
    D => Data_IN,
    En => regsel_signal(4),
    Reset => Reset,
    Clk => Clk,
    Q => reg_outputs(4)
);

Reg_5 : Reg
port map (
    D => Data_IN,
    En => regsel_signal(5),
    Reset => Reset,
    Clk => Clk,
    Q => reg_outputs(5)
);

Reg_6 : Reg
port map (
    D => Data_IN,
    En => regsel_signal(6),
    Reset => Reset,
    Clk => Clk,
    Q => reg_outputs(6)
);

Reg_7 : Reg
port map (
    D => Data_IN,
    En => regsel_signal(7),
    Reset => Reset,
    Clk => Clk,
    Q => reg_outputs(7)
```

```
    );

    Data_Bus <= reg_outputs; -- Assign the array of register outputs to Data_Bus

end Behavioral;
```

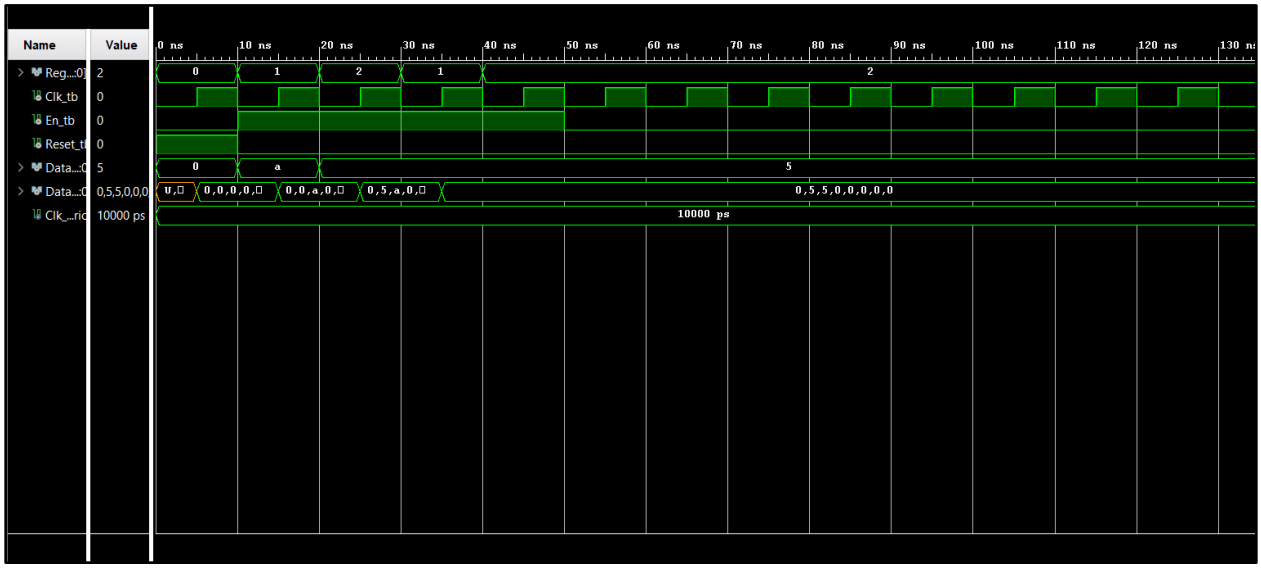
Simulation File for Register Bank (RegBank_tb.vhd)

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

entity RegBank_tb is
end RegBank_tb;

architecture Behavioral of RegBank_tb is
    -- Component Declaration
    component RegBank
        Port (RegSel  : in STD_LOGIC_VECTOR (2 downto 0);
              Clk     : in STD_LOGIC;
              En      : in STD_LOGIC;
              Reset   : in STD_LOGIC;
              Data_IN : in STD_LOGIC_VECTOR (3 downto 0);
              Data_Bus: out BUS_8_4bit);
    end component;

    -- Signals
    signal RegSel_tb   : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
    signal Clk_tb      : STD_LOGIC := '0';
    signal En_tb       : STD_LOGIC := '0';
    signal Reset_tb    : STD_LOGIC := '0';
    signal Data_IN_tb  : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
    signal Data_Bus_tb : BUS_8_4bit;

    -- Clock Period
    constant Clk_period : time := 10 ns;

begin
    -- Instantiate the Unit Under Test (UUT)
    uut : RegBank
        port map (
            RegSel  => RegSel_tb,
            Clk     => Clk_tb,
            En      => En_tb,
            Reset   => Reset_tb,
```

```
        Data_IN => Data_IN_tb,
        Data_Bus => Data_Bus_tb
    );

-- Clock process
Clk_process : process
begin
    Clk_tb <= '0';
    wait for Clk_period / 2;
    Clk_tb <= '1';
    wait for Clk_period / 2;
end process;

-- Stimulus process
stim_proc : process
begin
    -- Reset the register bank
    Reset_tb <= '1';
    wait for Clk_period;
    Reset_tb <= '0';

    -- Write data to register 1
    RegSel_tb <= "001";
    Data_IN_tb <= "1010";
    En_tb <= '1';
    wait for Clk_period;
    En_tb <= '0';

    -- Write data to register 2
    RegSel_tb <= "010";
    Data_IN_tb <= "0101";
    En_tb <= '1';
    wait for Clk_period;
    En_tb <= '0';

    -- Read data from register 1
    RegSel_tb <= "001";
    En_tb <= '1';
    wait for Clk_period;
    En_tb <= '0';

    -- Read data from register 2
    RegSel_tb <= "010";
    En_tb <= '1';
    wait for Clk_period;
```

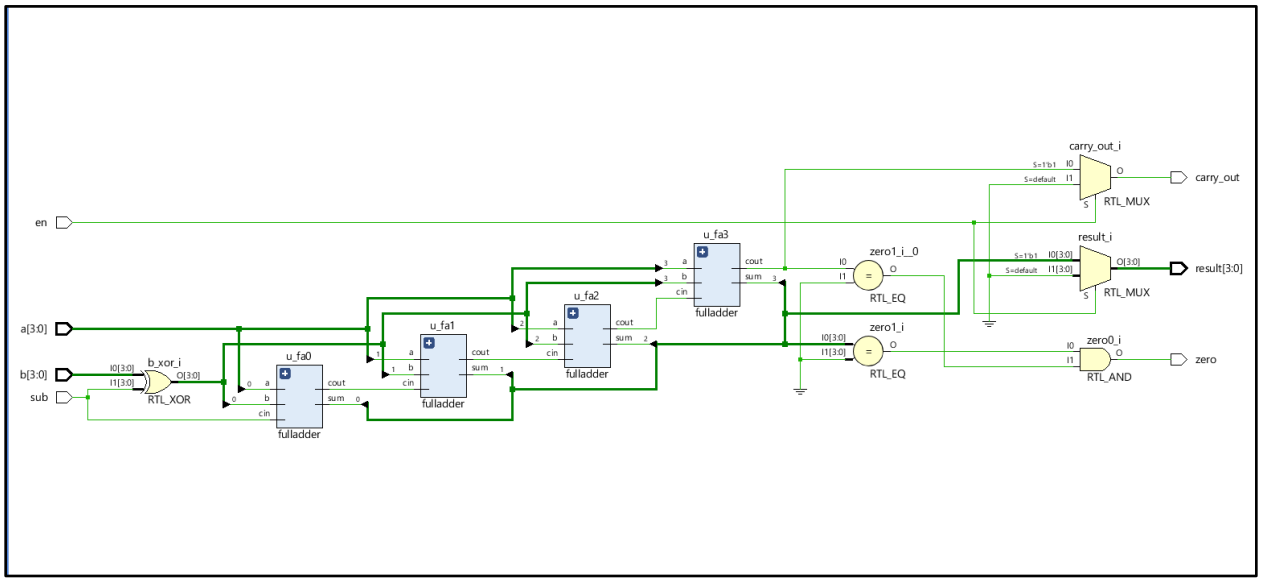
```
        En_tb <= '0';

        wait; -- Wait forever
    end process;

end Behavioral;
```

Timing Diagram for Register Bank



4-bit Adder/ Subtractor

Elaborated Design Schematic



Design Source Code (Mux_2_3bit.vhd)

```
library ieee;
use ieee.std_logic_1164.all;

entity addsub4 is
    port (
        a      : in  std_logic_vector(3 downto 0);
        b      : in  std_logic_vector(3 downto 0);
        sub    : in  std_logic;
        en : in std_logic;
        result: out std_logic_vector(3 downto 0);
        carry_out: out std_logic;
        zero   : out std_logic
    );
end addsub4;

architecture behavioral of addsub4 is
    component fulladder
        port (
            a   : in  std_logic;
            b   : in  std_logic;
            cin : in  std_logic;
            sum : out std_logic;
            cout: out std_logic
```

```
        );
    end component;

    signal c   : std_logic_vector(4 downto 0);
    signal b_xor: std_logic_vector(3 downto 0);
    signal sum : std_logic_vector(3 downto 0);
    signal zeroflag : std_logic;

begin
    -- XOR b with the sub signal to perform addition or subtraction
    b_xor <= b xor (sub & sub & sub & sub);

    -- Instantiate full adders
    u_fa0 : fulladder
        port map (
            a   => a(0),
            b   => b_xor(0),
            cin => sub,
            sum => sum(0),
            cout=> c(1)
        );

    u_fa1 : fulladder
        port map (
            a   => a(1),
            b   => b_xor(1),
            cin => c(1),
            sum => sum(1),
            cout=> c(2)
        );

    u_fa2 : fulladder
        port map (
            a   => a(2),
            b   => b_xor(2),
            cin => c(2),
            sum => sum(2),
            cout=> c(3)
        );

    u_fa3 : fulladder
        port map (
            a   => a(3),
            b   => b_xor(3),
            cin => c(3),
```

```
            sum => sum(3),
            cout=> c(4)
        );

    -- Assign outputs

     result   <= sum when en='1' else (others => '0');
    carry_out<= c(4) when en='1' else '0';

    -- Optimize zero flag calculation
    process(Sum, C(4))
    begin
        if (Sum = "0000" and C(4) = '0') then
            Zero <= '1';
        else
            Zero <= '0';
        end if;
    end process;



end behavioral;
```

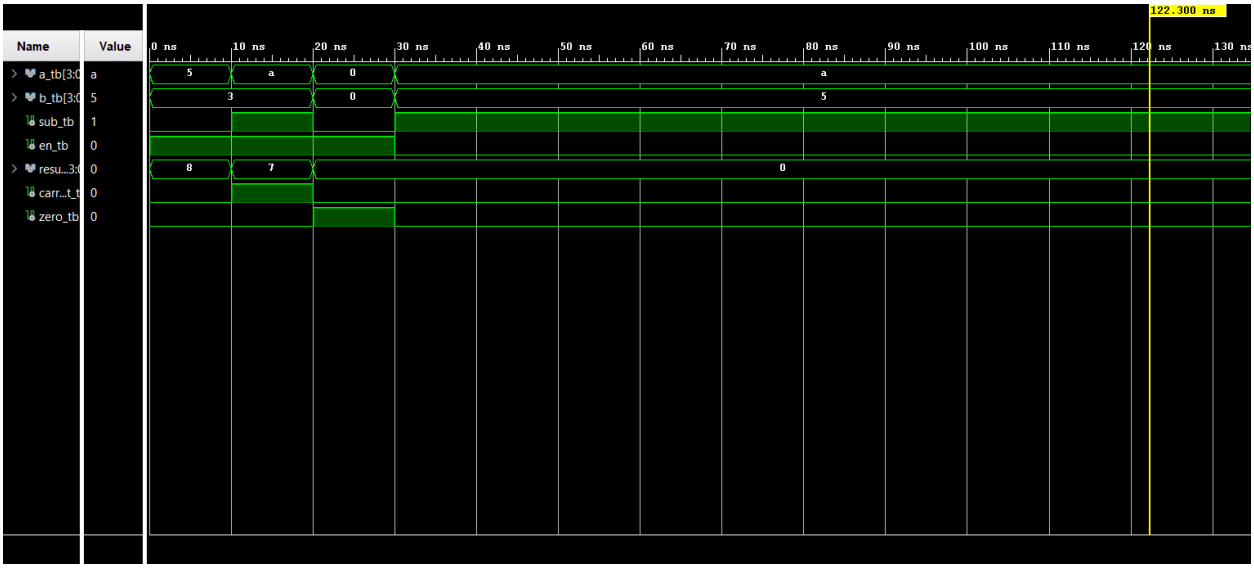
Simulation File for 4-bit Adder/ Subtractor(Mux_2_3bit_TB.vhd)

```
library ieee;
use ieee.std_logic_1164.all;

entity addsub4_tb is
end addsub4_tb;

architecture testbench of addsub4_tb is
    component addsub4
        port (
            a        : in  std_logic_vector(3 downto 0);
            b        : in  std_logic_vector(3 downto 0);
            sub      : in  std_logic;
            result   : out std_logic_vector(3 downto 0);
            carry_out: out std_logic;
            zero     : out std_logic
        );
    end component;
```

```
    signal a_tb        : std_logic_vector(3 downto 0);
    signal b_tb        : std_logic_vector(3 downto 0);
    signal sub_tb      : std_logic;
    signal result_tb   : std_logic_vector(3 downto 0);
    signal carry_out_tb: std_logic;
    signal zero_tb     : std_logic;

begin
    -- Instantiate the Unit Under Test (UUT)
    uut : addsub4
        port map (
            a        => a_tb,
            b        => b_tb,
            sub      => sub_tb,
            result   => result_tb,
            carry_out=> carry_out_tb,
            zero     => zero_tb
        );

    -- Stimulus process
    stim_proc : process
    begin
        -- Test cases for addition
        sub_tb <= '0';

        -- Test case 1: 0000 + 0000
        a_tb    <= "0000";
        b_tb    <= "0000";
        wait for 10 ns;
        assert result_tb = "0000" and carry_out_tb = '0' and zero_tb = '1'
            report "Test case 1 failed" severity error;

        -- Test case 2: 1010 + 0101
        a_tb    <= "1010";
        b_tb    <= "0101";
        wait for 10 ns;
        assert result_tb = "1111" and carry_out_tb = '0' and zero_tb = '0'
            report "Test case 2 failed" severity error;

        -- Test case 3: 1111 + 0001
        a_tb    <= "1111";
        b_tb    <= "0001";
        wait for 10 ns;
        assert result_tb = "0000" and carry_out_tb = '1' and zero_tb = '1'
            report "Test case 3 failed" severity error;
```

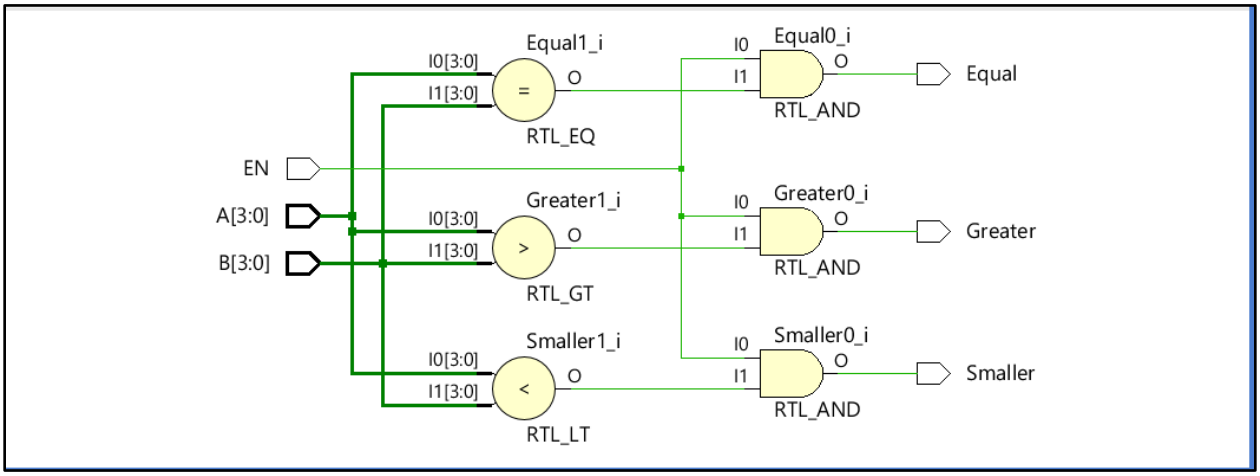
```
        -- Test cases for subtraction
        sub_tb <= '1';

        -- Test case 4: 1000 - 0010
        a_tb    <= "1000";
        b_tb    <= "0010";
        wait for 10 ns;
        assert result_tb = "0110" and carry_out_tb = '0' and zero_tb = '0'
            report "Test case 4 failed" severity error;

        -- Test case 5: 0101 - 0111
        a_tb    <= "0101";
        b_tb    <= "0111";
        wait for 10 ns;
        assert result_tb = "1110" and carry_out_tb = '1' and zero_tb = '0'
            report "Test case 5 failed" severity error;

        -- Test case 6: 0100 - 0100
        a_tb    <= "0100";
        b_tb    <= "0100";
        wait for 10 ns;
        assert result_tb = "0000" and carry_out_tb = '0' and zero_tb = '1'
            report "Test case 6 failed" severity error;

        wait; -- End the simulation
    end process;
end testbench;
```

Timing Diagram for 4-bit Adder/ Subtractor



Comparator

Elaborated Design Schematic



Design Source Code (Comparator_4_bit.vhd)

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


entity Comparator_4_bit is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           EN:in std_logic;
           Equal : out STD_LOGIC;
           Greater : out STD_LOGIC;
           Smaller : out STD_LOGIC);
end Comparator_4_bit;

architecture Behavioral of Comparator_4_bit is


begin

Greater <= '1' when (en = '1') AND(A > B)
else '0';
Equal <= '1' when (en = '1') AND(A = B)
else '0';
Smaller <= '1' when (en = '1') AND (A < B)
else '0';
```

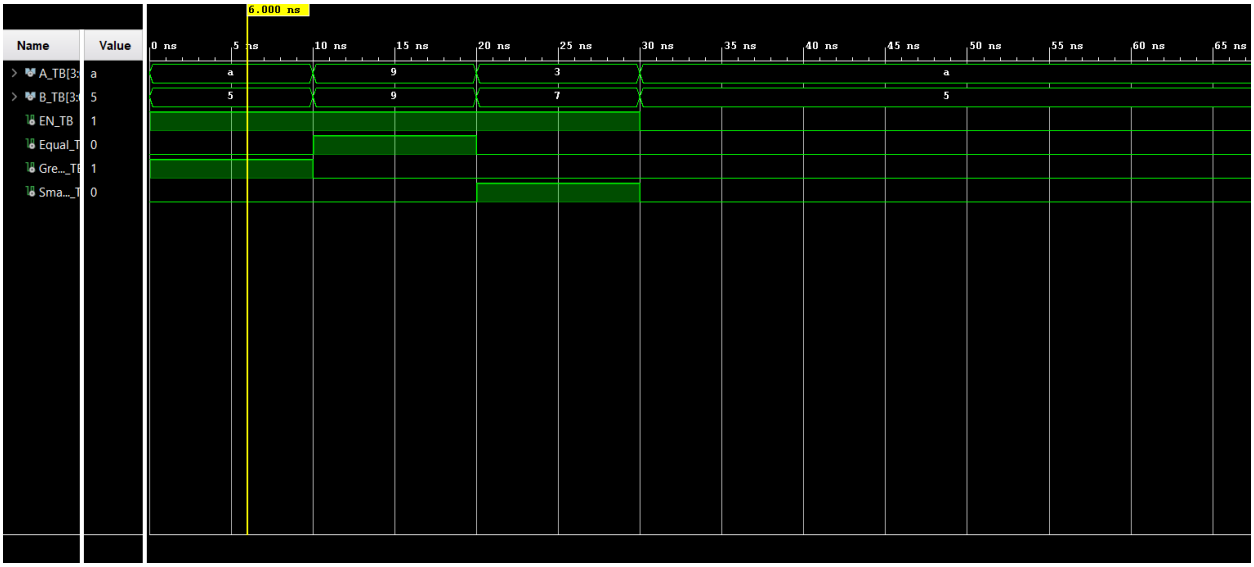
```
end Behavioral;
```

Simulation File for Comparator(Comparator_4_bit_TB.vhd)

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

entity Comparator_4_bit_TB is
end Comparator_4_bit_TB;

architecture Behavioral of Comparator_4_bit_TB is
    component Comparator_4_bit
        Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
               B : in STD_LOGIC_VECTOR (3 downto 0);
               EN: in std_logic;
               Equal : out STD_LOGIC;
               Greater : out STD_LOGIC;
               Smaller : out STD_LOGIC);
    end component;

    signal A_TB: STD_LOGIC_VECTOR (3 downto 0);
    signal B_TB: STD_LOGIC_VECTOR (3 downto 0);
    signal EN_TB: std_logic;
    signal Equal_TB: STD_LOGIC;
    signal Greater_TB: STD_LOGIC;
    signal Smaller_TB: STD_LOGIC;

begin
    UUT: Comparator_4_bit
        Port Map (
            A => A_TB,
            B => B_TB,
            EN => EN_TB,
            Equal => Equal_TB,
            Greater => Greater_TB,
            Smaller => Smaller_TB
        );

    -- Stimulus process
    stimulus_proc: process
    begin
        -- Test case 1: A > B, EN = '1'
        A_TB <= "1010";
```

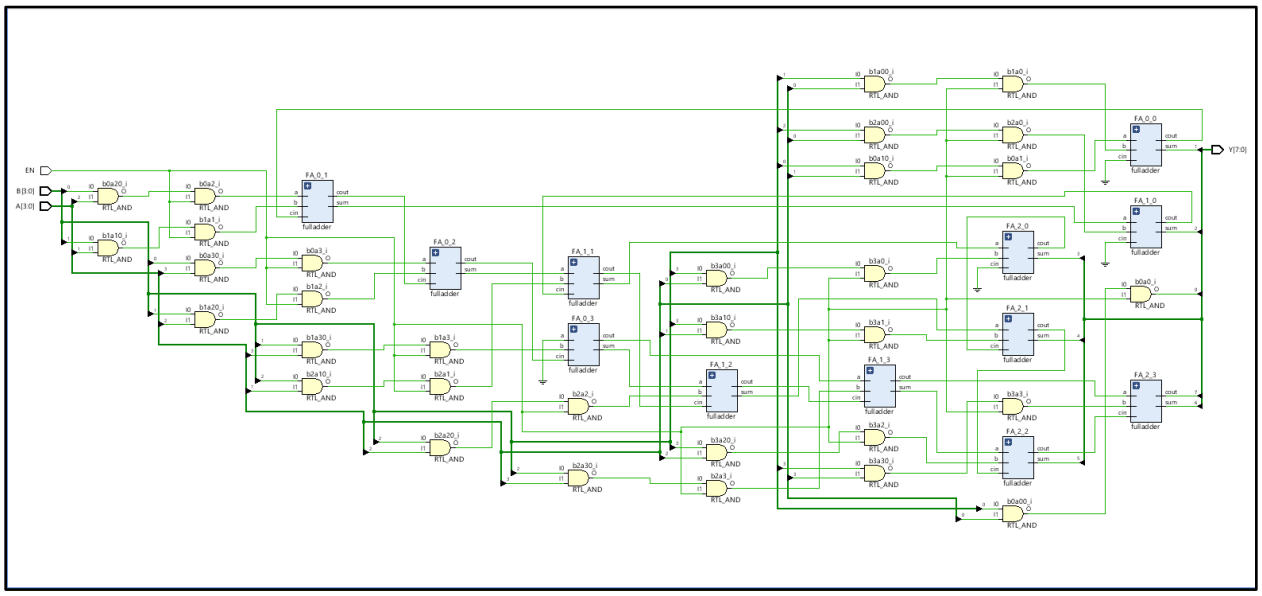
```
        B_TB <= "0101";
        EN_TB <= '1';
        wait for 10 ns;
        assert(Equal_TB = '0') report "Test case 1 failed" severity error;
        assert(Greater_TB = '1') report "Test case 1 failed" severity error;
        assert(Smaller_TB = '0') report "Test case 1 failed" severity error;

        -- Test case 2: A = B, EN = '1'
        A_TB <= "1001";
        B_TB <= "1001";
        EN_TB <= '1';
        wait for 10 ns;
        assert(Equal_TB = '1') report "Test case 2 failed" severity error;
        assert(Greater_TB = '0') report "Test case 2 failed" severity error;
        assert(Smaller_TB = '0') report "Test case 2 failed" severity error;

        -- Test case 3: A < B, EN = '1'
        A_TB <= "0011";
        B_TB <= "0111";
        EN_TB <= '1';
        wait for 10 ns;
        assert(Equal_TB = '0') report "Test case 3 failed" severity error;
        assert(Greater_TB = '0') report "Test case 3 failed" severity error;
        assert(Smaller_TB = '1') report "Test case 3 failed" severity error;

        -- Test case 4: EN = '0'
        A_TB <= "1010";
        B_TB <= "0101";
        EN_TB <= '0';
        wait for 10 ns;
        assert(Equal_TB = '0') report "Test case 4 failed" severity error;
        assert(Greater_TB = '0') report "Test case 4 failed" severity error;
        assert(Smaller_TB = '0') report "Test case 4 failed" severity error;

        wait;
    end process;
end Behavioral;
```

Timing Diagram for 4-bit Adder/ Subtractor



Multiplier

Elaborated Design Schematic



VHDL code for design file

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

entity Multiplier_4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end Multiplier_4;

architecture Behavioral of Multiplier_4 is

component fulladder
        Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Cin : in STD_LOGIC;
           Cout : out STD_LOGIC;
           Sum : out STD_LOGIC);
end component;

signal b0a0, b0a1, b0a2, b0a3 : std_logic;
signal b1a0, b1a1, b1a2, b1a3 : std_logic;
```

```
signal b2a0, b2a1, b2a2, b2a3 : std_logic;
signal b3a0, b3a1, b3a2, b3a3 : std_logic;

signal  C_0_0, S_0_0 : std_logic;
signal  C_0_1, S_0_1 : std_logic;
signal  C_0_2, S_0_2 : std_logic;
signal  C_0_3, S_0_3 : std_logic;

signal  C_1_0, S_1_0 : std_logic;
signal  C_1_1, S_1_1 : std_logic;
signal  C_1_2, S_1_2 : std_logic;
signal  C_1_3, S_1_3 : std_logic;

signal  C_2_0, S_2_0 : std_logic;
signal  C_2_1, S_2_1 : std_logic;
signal  C_2_2, S_2_2 : std_logic;
signal  C_2_3, S_2_3 : std_logic;

begin

FA_0_0 : fulladder
    port map(
    A => b0a1,
    B => b1a0,
    Cin => '0',
    Cout => C_0_0,
    Sum => S_0_0);

FA_0_1 : fulladder
    port map(
    A => b0a2,
    B => b1a1,
    Cin => C_0_0,
    Cout => C_0_1,
    Sum => S_0_1);

FA_0_2 : fulladder
    port map(
    A => b0a3,
    B => b1a2,
    Cin => C_0_1,
    Cout => C_0_2,
    Sum => S_0_2);

FA_0_3 : fulladder
```

```
    port map(
    A => '0',
    B => b1a3,
    Cin => C_0_2,
    Cout => C_0_3,
    Sum => S_0_3);

FA_1_0 : fulladder
    port map(
    A => S_0_1,
    B => b2a0,
    Cin => '0',
    Cout => C_1_0,
    Sum => S_1_0);

FA_1_1 : fulladder
    port map(
    A => S_0_2,
    B => b2a1,
    Cin => C_1_0,
    Cout => C_1_1,
    Sum => S_1_1);

FA_1_2 : fulladder
    port map(
    A => S_0_3,
    B => b2a2,
    Cin => C_1_1,
    Cout => C_1_2,
    Sum => S_1_2);

FA_1_3 : fulladder
    port map(
    A => C_0_3,
    B => b2a3,
    Cin => C_1_2,
    Cout => C_1_3,
    Sum => S_1_3);

FA_2_0 : fulladder
    port map(
    A => S_1_1,
    B => b3a0,
    Cin => '0',
    Cout => C_2_0,
```

```
    Sum => S_2_0);

FA_2_1 : fulladder
    port map(
    A => S_1_2,
    B => b3a1,
    Cin => C_2_0,
    Cout => C_2_1,
    Sum => S_2_1);

FA_2_2 : fulladder
    port map(
    A => S_1_3,
    B => b3a2,
    Cin => C_2_1,
    Cout => C_2_2,
    Sum => S_2_2);

FA_2_3 : fulladder
    port map(
    A => C_1_3,
    B => b3a3,
    Cin => C_2_2,
    Cout => C_2_3,
    Sum => S_2_3);

    b0a0 <= B(0) AND A(0) AND EN;
    b0a1 <= B(0) AND A(1) AND EN;
    b0a2 <= B(0) AND A(2) AND EN;
    b0a3 <= B(0) AND A(3) AND EN;
    b1a0 <= B(1) AND A(0) AND EN;
    b1a1 <= B(1) AND A(1) AND EN;
    b1a2 <= B(1) AND A(2) AND EN;
    b1a3 <= B(1) AND A(3) AND EN;
    b2a0 <= B(2) AND A(0) AND EN;
    b2a1 <= B(2) AND A(1) AND EN;
    b2a2 <= B(2) AND A(2) AND EN;
    b2a3 <= B(2) AND A(3) AND EN;
    b3a0 <= B(3) AND A(0) AND EN;
    b3a1 <= B(3) AND A(1) AND EN;
    b3a2 <= B(3) AND A(2) AND EN;
    b3a3 <= B(3) AND A(3) AND EN;

    Y(0) <= b0a0;
    Y(1) <= S_0_0;
```

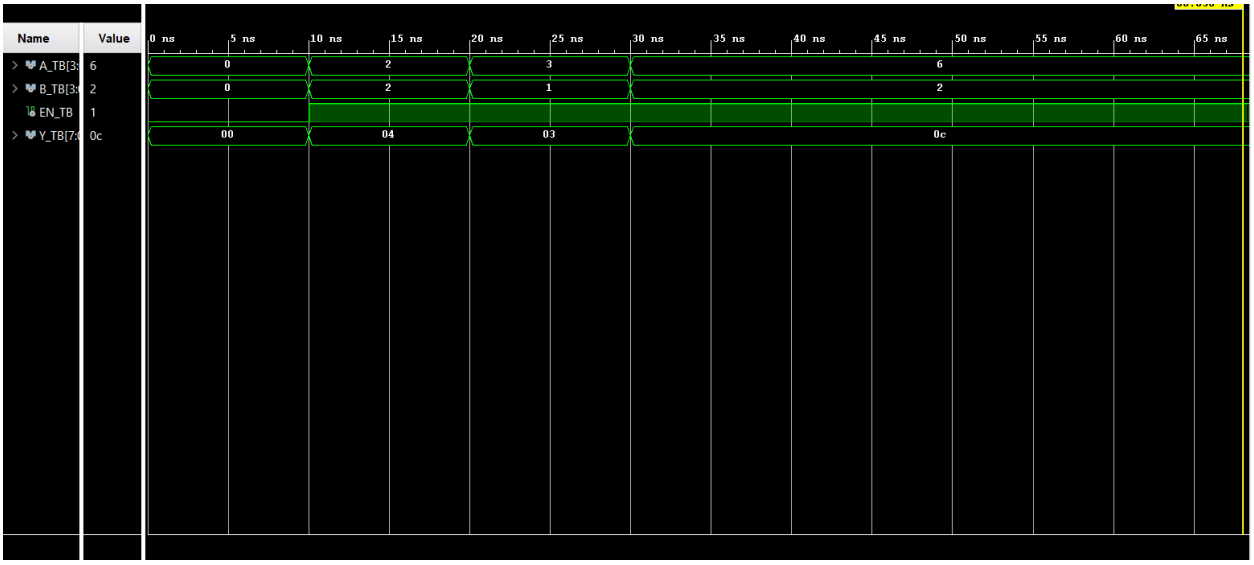
```
    Y(2) <= S_1_0;
    Y(3) <= S_2_0;
    Y(4) <= S_2_1;
    Y(5) <= S_2_2;
    Y(6) <= S_2_3;
    Y(7) <= C_2_3;

end Behavioral;
```

VHDL code for simulation file

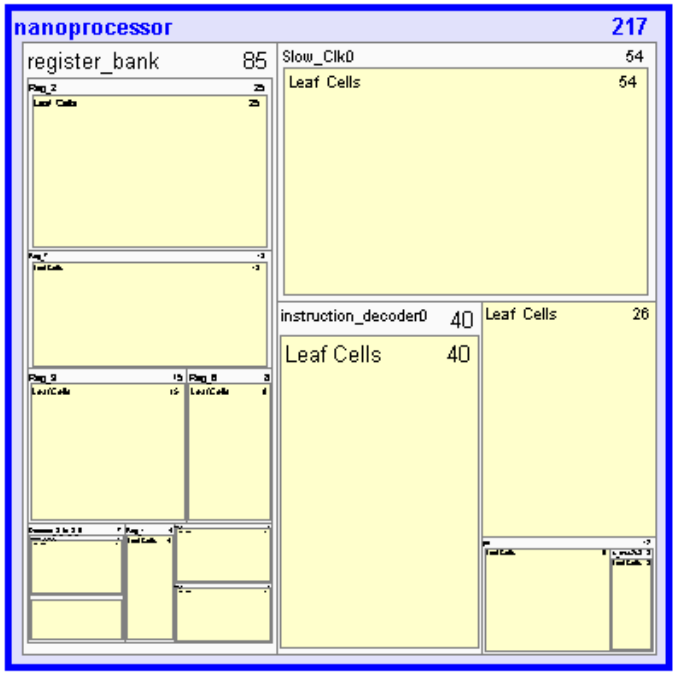
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Multiplier_4_TB is
end Multiplier_4_TB;

architecture Behavioral of Multiplier_4_TB is

    -- Component declaration for the DUT (Device Under Test)
    component Multiplier_4
        Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
               B : in STD_LOGIC_VECTOR (3 downto 0);
               EN : in STD_LOGIC;
               Y : out STD_LOGIC_VECTOR (7 downto 0));
    end component;

    -- Signal declarations
    signal A_TB : STD_LOGIC_VECTOR (3 downto 0);
    signal B_TB : STD_LOGIC_VECTOR (3 downto 0);
    signal EN_TB : STD_LOGIC;
    signal Y_TB : STD_LOGIC_VECTOR (7 downto 0);

begin

    -- Instantiate the DUT
    uut : Multiplier_4
        port map (A => A_TB,
                  B => B_TB,
                  EN => EN_TB,
                  Y => Y_TB);

    -- Stimulus process
    stimulus_proc: process
    begin
```

```
        -- Initialize inputs
        A_TB <= "0000";
        B_TB <= "0000";
        EN_TB <= '0';

        -- Apply stimulus
        wait for 10 ns;
        A_TB <= "0010";
        B_TB <= "0010";
        EN_TB <= '1';

        wait for 10 ns;
        A_TB <= "0011";
        B_TB <= "0001";

        wait for 10 ns;
        A_TB <= "0110";
        B_TB <= "0010";

        -- Add more test cases as needed

        wait;
    end process;

end Behavioral;
```

Time simulation for multiplier



✓ synth_1	constrs_1	synth_design Complete!								63	83	0.0	0	0	5/5/24, 9:25 AM	00:00:19	Vivado Synthesis Defaults (V
✓ impl_1	constrs_1	route_design Complete!	5.493	0.000	0.195	0.000	0.000	0.070	0	67	86	0.0	0	0	5/5/24, 9:28 AM	00:00:35	Vivado Implementation Def



Conclusion

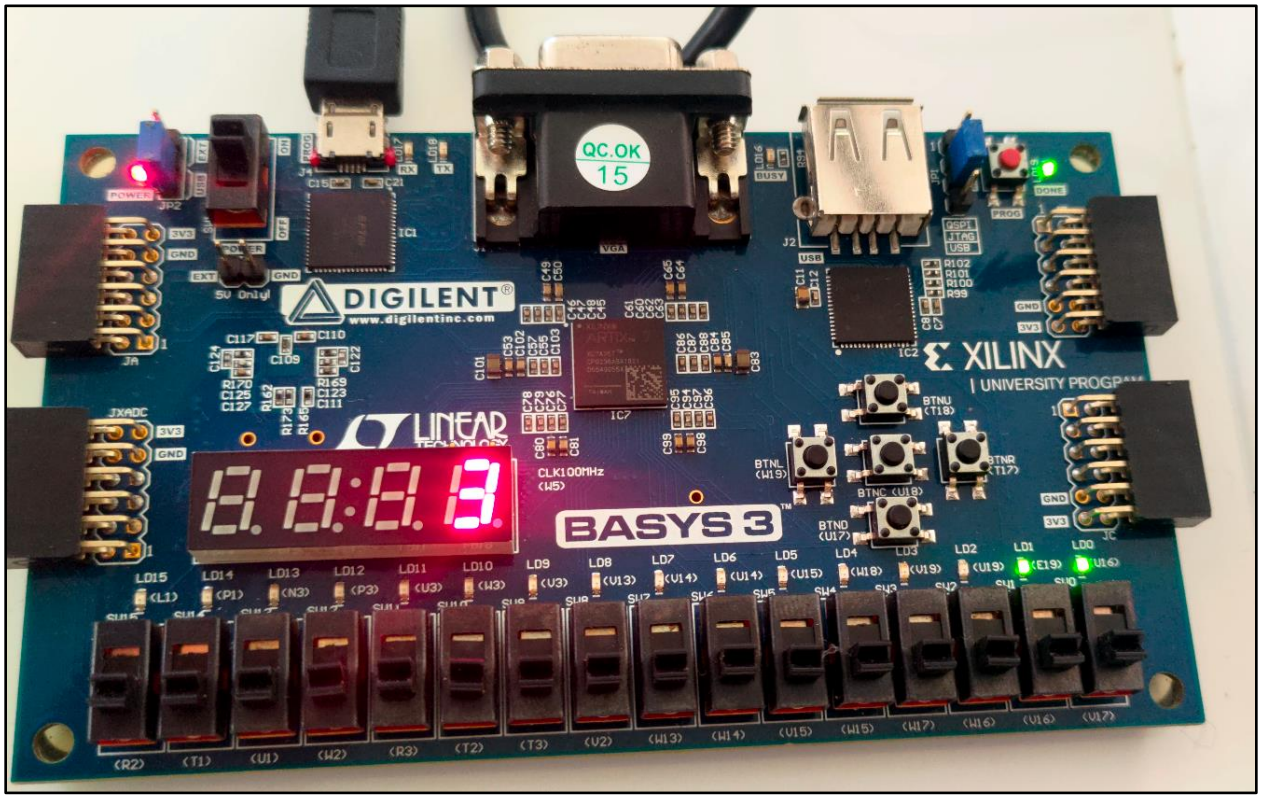
This lab provided a valuable opportunity to design and implement a 4-bit nanoprocessor capable of executing a simple set of instructions. Through this project, the team was able to apply the concepts learned in the course and gain practical experience in designing and integrating various digital logic components, such as the arithmetic unit, program counter, register bank, instruction decoder, and program ROM.

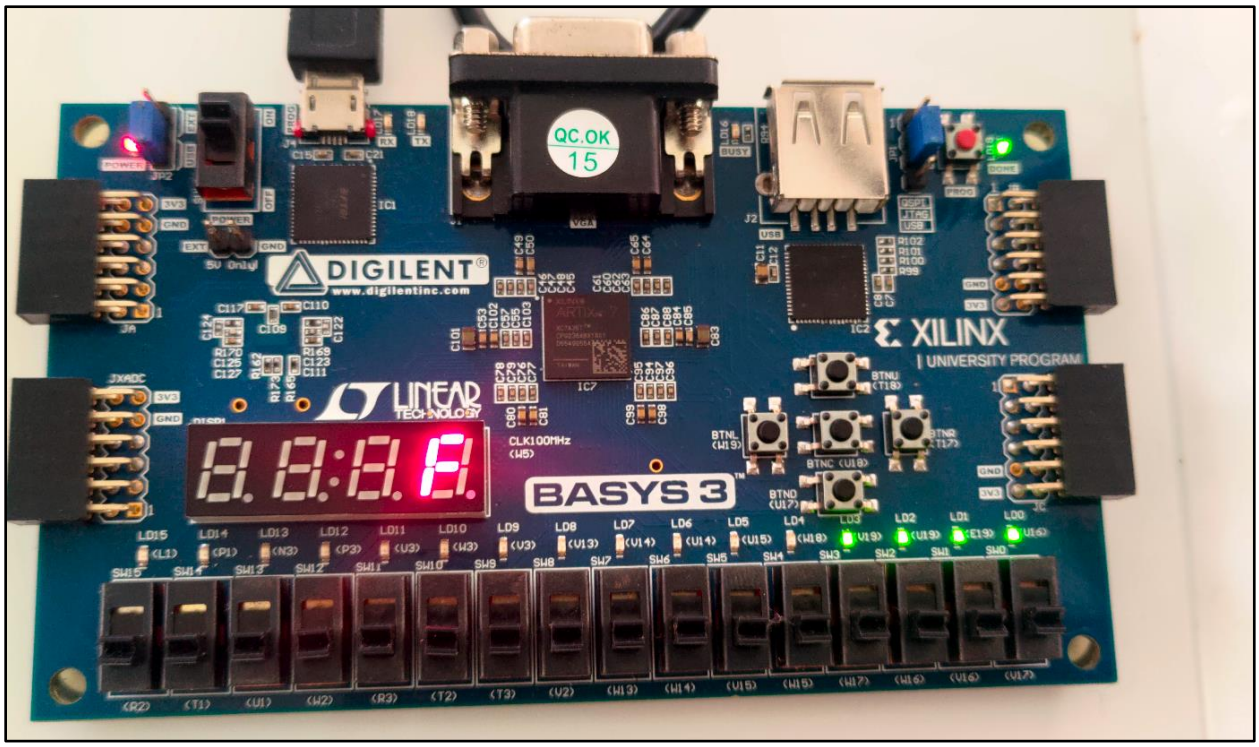
The design process involved breaking down the overall system into manageable sub-components, developing and testing each component individually, and then integrating them into the final nano processor design. This hands-on experience allowed the team to develop skills in digital design, VHDL programming, simulation, and troubleshooting.

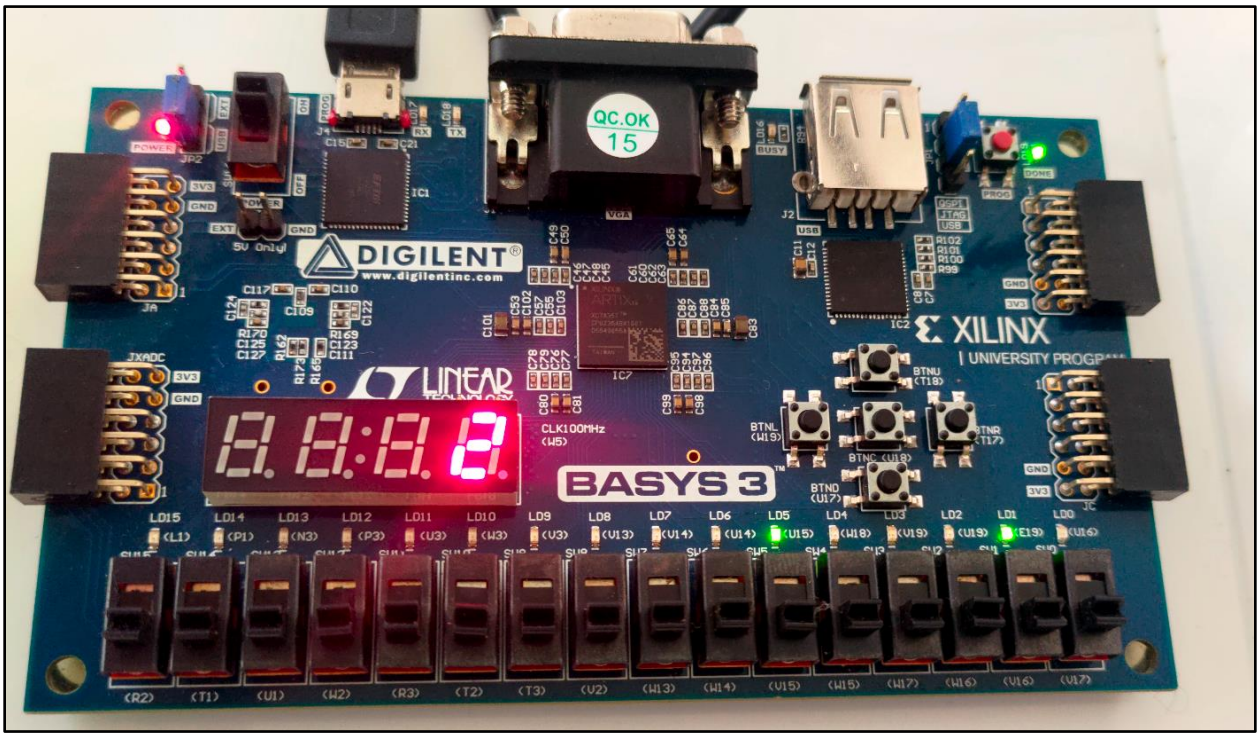
Additionally, the team-based nature of the project facilitated the development of important soft skills, including communication, collaboration, task management, and problem-solving. By dividing the workload and coordinating the integration of the sub-components, the team was able to efficiently complete the project and demonstrate the functionality of the nano processor.

Testing on Basys3

We tested a modified version of nano processor in Basys3 board.






QC.OK
15
DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGR
LINEAR
BASYS 3
CLK100MHz
(W5)
BTNU
(T18)
BTNL
(W19)
BTNC (U18)
BTNR
(T17)
BTND
(U17)
LD15 LD14 LD13 LD12 LD11 LD10 LD9 LD8 LD7 LD6 LD5 LD4 LD3 LD2 LD1 LD0
(R2) (T1) (U1) (W2) (R3) (T2) (T3) (V2) (W13) (W14) (V15) (W15) (W17) (W16) (V16) (V17)

Team Contribution

Waduge S.S.	• Bus System • Instruction Decoder • Program Counter • Constraints file
Wijesooriya J.M.I.	• Multiplier • Program Rom • Register Bank
Gallage A.H.	• Adder/Subtractor • Multiplexer • Test Benches (Instruction Decoder, Multiplexer, PC)
Wijeratne E.N.K.	• Nano Processor • Comparator • Test Benches (Multiplier, Register Bank, Nano Processor, Comparator)