CS1050 - Computer Organization and Digital Design
Lab 9 & 10 Nano Processor Design Competition

- **Group 20**
  - Kariyapperuma K.M.N.S 230317J
  - Kirindage S.S 230336P
  - Jayasinghe J.D.D.S 230281P
  - Rafi M.A.A 230505J

♦ Introduction

- The objective of the lab task was to design a nano processor using the components we had previously studied in earlier lab sessions. It was a group project, with each team consisting of four members.
- We developed two different versions of the nano processor. Initially, we created a version that met only the basic requirements outlined in the lab manual. For this version, we incorporated components such as a 4-bit Add/Subtract unit, a 3-bit adder, a 3-bit Program Counter, 8-way 2-bit multiplexers, 2-way 3-bit multiplexers, 2-way 4-bit multiplexers, a Register Bank, a Program ROM, and an Instruction Decoder.
- For the extended version of the nano processor, we upgraded several key components to enhance functionality. The Register Bank, 8-way multiplexers, and Add/Subtract unit were expanded to 8 bits. We also integrated a multiplier unit to support more complex arithmetic operations. For better output representation, three 7-segment display units were used, with the ability to display negative values using a minus sign. Additionally, we designed a Program ROM capable of storing up to 16 instructions, allowing for more versatile and extended program execution.
- We used 2's complement representation for signed register values, while the basic version displays them as unsigned on the 7-segment display.

Table of Contents

1) Design with Minimum qualifications

♦ Executable Instruction Set

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

♦ Assembly Code & Machine Code for the Design

```
Process  =>  3 +2 + 1 = 6


   "101110000001", -- 0          MOVI R7,1
   "100010000010", -- 1          MOVI R1.2
   "100100000011", -- 2          MOVI R2,3
   "001110010000", -- 3          ADD R7,R1
   "001110100000", -- 4          ADD R7,R2
  "110000000111", -- 5           JZR R0,7
  "110000000111", -- 6           JZR R0,7
  "110000000111" -- 7            JZR R0,7
```

Design Components

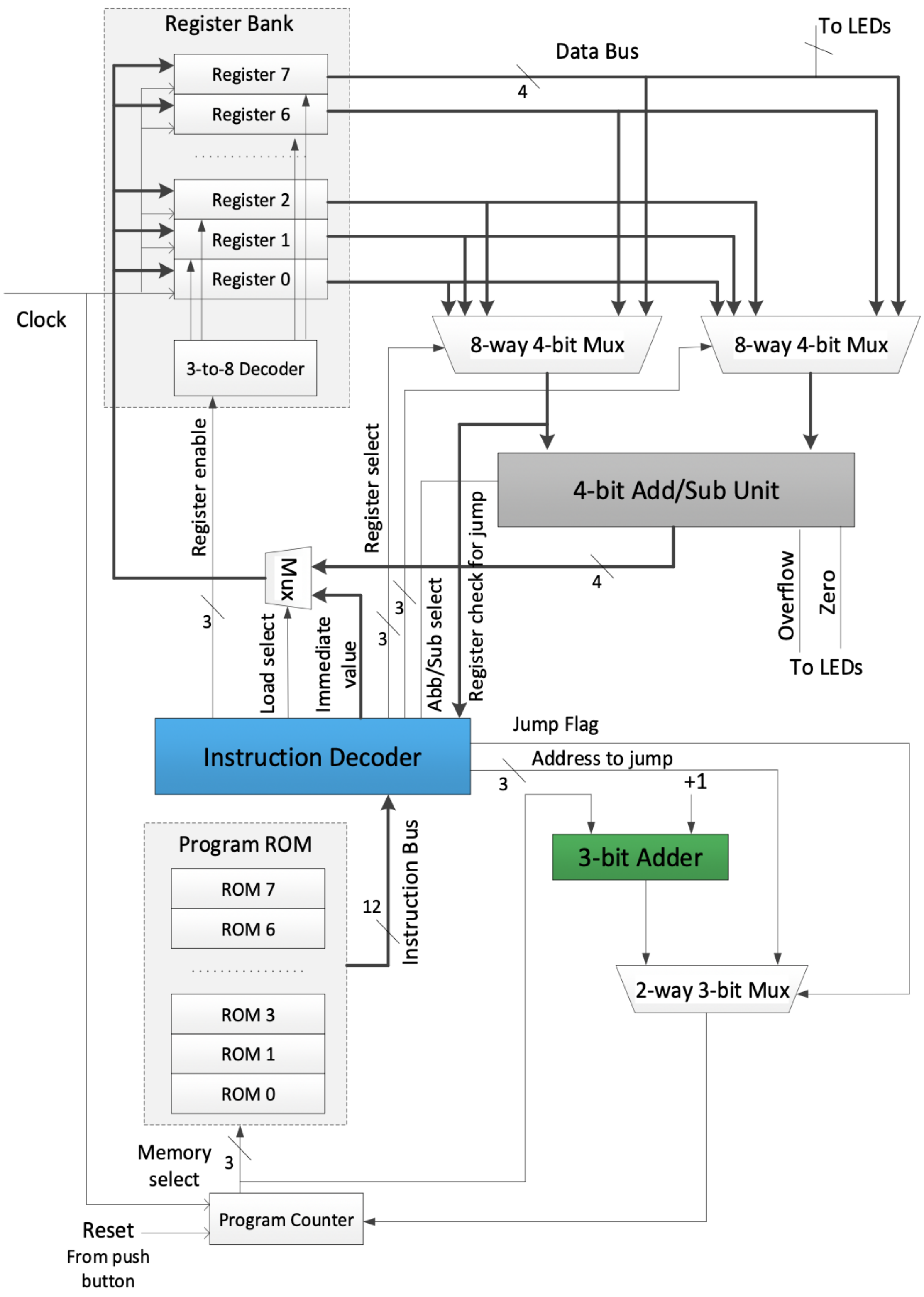


Figure 1 – High-level diagram of the nanoprocessor.

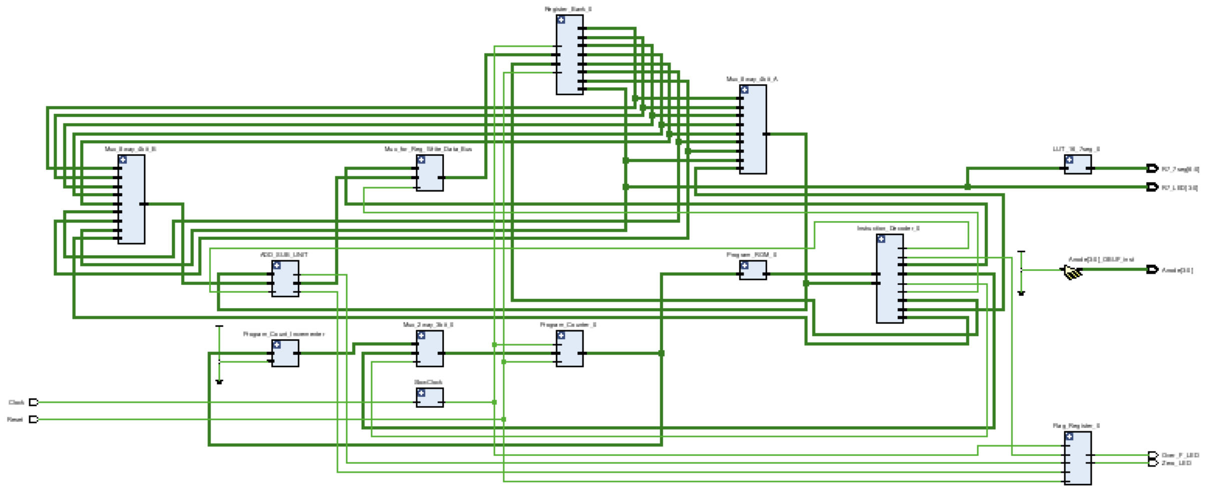
1. Nano Processor

- Design Source VHDL Code of Nano Processor

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

entity Nanoprocessor is
    Port ( Reset : in STD_LOGIC;
           Clock : in STD_LOGIC;
           R7_LED : out STD_LOGIC_VECTOR (3 downto 0);
           Zero_LED : out STD_LOGIC;
           Over_F_LED : out STD_LOGIC;
           Anode : out STD_LOGIC_VECTOR (3 downto 0);
           R7_7seg : out STD_LOGIC_VECTOR (6 downto 0));
end Nanoprocessor;

architecture Behavioral of Nanoprocessor is

----- Components ------

----- Slow Clock >>>> Slowed Clock signal so easy to observe --------
component Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end component;

-----Instruction Decoder----------------------------
component Instruction_Decoder is
    Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
           Jump_reg_val : in STD_LOGIC_VECTOR (3 downto 0); --register value for jump check
           Reg_Sel_A: out STD_LOGIC_VECTOR (2 downto 0);    --Mux Select signal
           Reg_Sel_B: out STD_LOGIC_VECTOR (2 downto 0);    --Muc Selct signal
           Reg_En :out STD_LOGIC_VECTOR (2 downto 0);       --Register Bank write enable signal
           Flag_Reg_En : out STD_LOGIC;     -- Enable for ADD_SUB_Units Flag
           Jump_Address : out STD_LOGIC_VECTOR (2 downto 0);
           Jump_Flag : out STD_LOGIC;
           Add_Sub_Sel : out STD_LOGIC; -- 0 for add , 1 for subtract
           Load_Sel : out STD_LOGIC;    -- 0 for immediarte value, 1 for add_sub  unit value
           Immediate_Value : out STD_LOGIC_VECTOR (3 downto 0));
end component;

---- Register Bank (8 4-bit registers) -------
component Reg_Bank is
    Port ( RegSel : in STD_LOGIC_VECTOR (2 downto 0);   --Register Select signal
           D : in STD_LOGIC_VECTOR (3 downto 0);    --Data line
           Reset: in STD_LOGIC;
           Clk : in STD_LOGIC;
           R0 : out STD_LOGIC_VECTOR (3 downto 0);
           R1 : out STD_LOGIC_VECTOR (3 downto 0);
           R2 : out STD_LOGIC_VECTOR (3 downto 0);
           R3 : out STD_LOGIC_VECTOR (3 downto 0);
           R4 : out STD_LOGIC_VECTOR (3 downto 0);
           R5 : out STD_LOGIC_VECTOR (3 downto 0);
           R6 : out STD_LOGIC_VECTOR (3 downto 0);
           R7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

---- Flag Register  (Zero, Overflow) --------
component Flag_register is
    Port ( Zero : in STD_LOGIC;     --zero signal from ADD_SUB_Unit
           Over_f : in STD_LOGIC;   --overflow signal from ADD_SUB_Unit
           En : in STD_LOGIC;
           reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Zero_Flag : out STD_LOGIC;
           OverFlow_Flag : out STD_LOGIC);
end component;

---- Program Counter (register) --------
component Program_Counter is
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
       Reset : in STD_LOGIC;
       Clk : in STD_LOGIC;
       P_count : out STD_LOGIC_VECTOR (2 downto 0));
end component;

---- 4-bit ADD_SUB_Unit -----------
component ADD_SUB_4bit is
    Port ( Ins : in STD_LOGIC; -- instruction (ADD =0, SUB = 1) (also works as C_in)
        A : in STD_LOGIC_VECTOR (3 DOWNTO 0);
        B : in STD_LOGIC_VECTOR (3 DOWNTO 0);
        S : out STD_LOGIC_VECTOR (3 DOWNTO 0);
        O_flow : out STD_LOGIC;
        Zero: out STD_LOGIC);
end component;

---- 3-bit Adder -----------------
component Adder_3bit is
    Port (
    A : in STD_LOGIC_VECTOR (2 DOWNTO 0);
    B : in STD_LOGIC_VECTOR (2 DOWNTO 0);
    S : out STD_LOGIC_VECTOR (2 DOWNTO 0));
end component;


---- 8 way 4bit Multiplexer ----------
component Mux_8way_4bit is
    Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
       L0 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 0
       L1 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 1
       L2 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 2
       L3 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 3
       L4 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 4
       L5 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 5
       L6 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 6
       L7 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 7
       EN : in STD_LOGIC;
       OL: out STD_LOGIC_VECTOR (3 downto 0)); -- 4-bit output line
end component;

---- 2 way 4bit Multiplexer --------
component Mux_2way_4bit is
    Port (
   Sel : in STD_LOGIC;
   A : in STD_LOGIC_VECTOR (3 downto 0);
   B : in STD_LOGIC_VECTOR (3 downto 0);
   Mux_out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

---- 2 way 3bit Multiplexer --------
component Mux_2way_3bit is
    Port (
       Sel : in STD_LOGIC;
       A : in STD_LOGIC_VECTOR (2 downto 0);
       B : in STD_LOGIC_VECTOR (2 downto 0);
       Mux_out : out STD_LOGIC_VECTOR (2 downto 0));
end component;

----- Program ROM >>>> Stored Machine language Assembly code -------
component Program_ROM is
    Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
           data : out STD_LOGIC_VECTOR (11 downto 0));
end component;

---- Lookup table for 7 Segment display outputs ----
component LUT_16_7seg is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
```

```
end component;



-----Signals-------------------

signal Slow_clock :std_logic;
signal instruction : std_logic_vector (11 downto 0);
signal Mux_A_out ,Mux_B_out : std_logic_vector (3 downto 0); -- 8way MUX outputs
signal Mux_A_Sel, Mux_B_Sel : std_logic_vector (2 downto 0); -- 8way MUX Select signal
signal Reg_EN : std_logic_vector (2 downto 0); -- Register bank select reg for write enable signal
signal Flag_Reg_EN : std_logic; --flag register enable signal
signal Address_to_jump : std_logic_vector (2 downto 0);
signal Jump_Flag : std_logic;
signal Add_Sub_Sel : std_logic;
signal Load_Sel : std_logic;    -- load select for register write data bus
signal Immediate_Value:  std_logic_vector (3 downto 0);

signal Reg_Data_in: std_logic_vector (3 downto 0); --Register Data in Bus
--Register Data outs
signal Reg_0, Reg_1, Reg_2, Reg_3, Reg_4, Reg_5, Reg_6, Reg_7: std_logic_vector (3 downto 0);
signal Add_Sub_Out : std_logic_vector (3 downto 0);
signal O_flow, Zero : std_logic;

signal Program_C_in, Program_C_out: std_logic_vector (2 downto 0);
signal Program_C_Incrementer_out : std_logic_vector (2 downto 0);


begin


    SlowClock : Slow_Clk
        port map(
        Clk_in => Clock,
        Clk_out => Slow_clock );

    Instruction_Decoder_0: Instruction_Decoder
        port map(
        Instruction     => instruction,
        Jump_reg_val    => Mux_A_out,
        Reg_Sel_A       => Mux_A_Sel,
        Reg_Sel_B       => Mux_B_Sel,
        Reg_En          => Reg_EN,
        Flag_Reg_En     => Flag_Reg_EN,
        Jump_Address    => Address_to_jump,
        Jump_Flag       => Jump_Flag,
        Add_Sub_Sel     => Add_Sub_Sel,
        Load_Sel        => Load_Sel,
        Immediate_Value => Immediate_Value);

    Register_Bank_0 : Reg_Bank
        port map(
        RegSel  => Reg_EN,
        D       => Reg_Data_in,
        Reset   => Reset,
        Clk     => Slow_clock,
        R0      => Reg_0,
        R1      => Reg_1,
        R2      => Reg_2,
        R3      => Reg_3,
        R4      => Reg_4,
        R5      => Reg_5,
        R6      => Reg_6,
        R7      => Reg_7);

    Mux_8way_4bit_A: Mux_8way_4bit
        port map(
        S => Mux_A_Sel,
        L0  => Reg_0,
        L1  => Reg_1,
        L2  => Reg_2,
        L3  => Reg_3,
        L4  => Reg_4,
        L5  => Reg_5,
        L6  => Reg_6,
        L7  => Reg_7,
        EN => '1',
        OL  => Mux_A_out);

    Mux_8way_4bit_B: Mux_8way_4bit
        port map(
        S => Mux_B_Sel,
        L0  => Reg_0,
        L1  => Reg_1,
        L2  => Reg_2,
        L3  => Reg_3,
        L4  => Reg_4,
        L5  => Reg_5,
        L6  => Reg_6,
        L7  => Reg_7,
        EN => '1',
        OL  => Mux_B_out);

    ADD_SUB_UNIT: Add_Sub_4bit
        port map(
        Ins => Add_Sub_Sel,
        A   => Mux_A_out,
        B   => Mux_B_out,
        S   => Add_Sub_Out,
        O_flow  => O_flow,
        Zero    => Zero);

    Mux_for_Reg_Write_Data_Bus: Mux_2way_4bit
        port map(
        A   => Immediate_Value,
        B   => Add_Sub_Out,
        Sel => Load_Sel,
        Mux_out   => Reg_Data_in);

    Flag_Register_0 :Flag_register
        port map(
        Zero    => Zero,
        Over_f  => O_flow,
        En      => Flag_Reg_EN,
        reset   => Reset,
        Clk     => Slow_clock,
        Zero_Flag  =>   Zero_LED,
        OverFlow_Flag => Over_F_LED);

    Program_Counter_0: Program_Counter
        port map(
        D       => Program_C_in,
        Reset   => Reset,
        Clk     => Slow_clock,
        P_count => Program_C_out);

    Program_Count_Incrementer: Adder_3bit
        port map(
        A   => Program_C_out,
        B   => "001",
        S   => Program_C_Incrementer_out);

    Mux_2way_3bit_0 : Mux_2way_3bit
        port map(
        A   => Program_C_Incrementer_out,
        B   => Address_to_jump,
        Sel => Jump_Flag,
        Mux_out  => Program_C_in);

    Program_ROM_0: Program_ROM
        Port map(
        address => Program_C_out,
        data    => instruction);


---- Output Settings ----------------

    -- Zero and Overflow outputs done in above
    LUT_16_7seg_0 : LUT_16_7seg
        Port map(
        address => Reg_7,
        data    => R7_7seg);

    R7_LED  <= Reg_7;

    ----enable left most 7 segment display panel---
    anode <= "1110";


end Behavioral;
```

- <u>RTL Schematic Diagram of the Nano Processor</u>

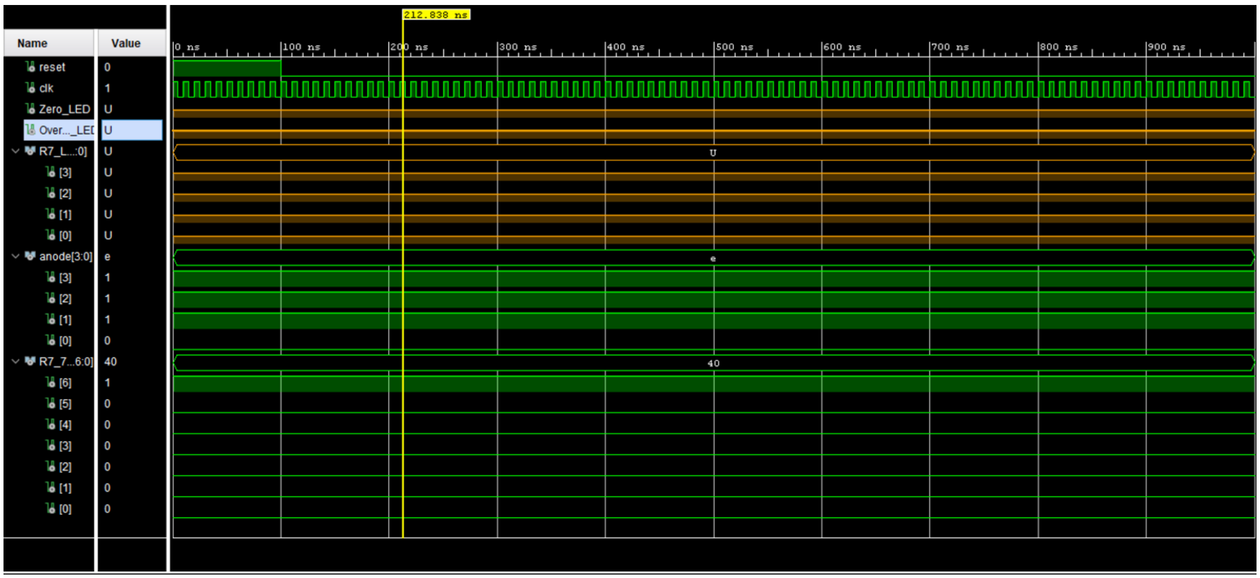


- <u>Behavioral Simulation Code for Nano Processor</u>

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


entity TB_Nanoprocessor is
end TB_Nanoprocessor;

architecture Behavioral of TB_Nanoprocessor is

component Nanoprocessor is
    Port ( Reset : in STD_LOGIC;
           Clock : in STD_LOGIC;
           R7_LED : out STD_LOGIC_VECTOR (3 downto 0);
           Zero_LED : out STD_LOGIC;
           Over_F_LED : out STD_LOGIC;
           Anode : out STD_LOGIC_VECTOR (3 downto 0);
           R7_7seg : out STD_LOGIC_VECTOR (6 downto 0));
end component;

signal reset : std_logic;
signal clk : std_logic := '0';
signal Zero_LED, Over_F_LED : std_logic;
signal R7_LED : STD_LOGIC_VECTOR (3 downto 0);
signal anode : STD_LOGIC_VECTOR (3 downto 0);
signal R7_7seg : STD_LOGIC_VECTOR (6 downto 0);
```

```
begin

UUT: Nanoprocessor
    port map(
    Reset    => reset,
    Clock    => clk,
    R7_LED   => R7_LED,
    Zero_LED    =>  Zero_LED,
    Over_F_LED => Over_F_LED,
    Anode    => anode,
    R7_7seg => R7_7seg);

    process
    begin
        clk <= not clk;
        wait for 5 ns;
    end process;

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

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


end Behavioral;
```

● Timing Diagram for Nano Processor



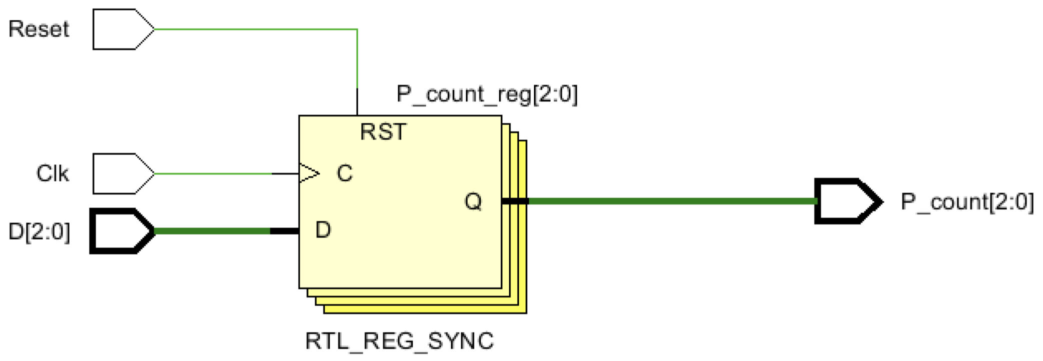
Name	Value
reset	0
clk	1
Zero_LED	U
Over..._LED	U
R7_L...:0]	U
[3]	U
[2]	U
[1]	U
[0]	U
anode[3:0]	e
[3]	1
[2]	1
[1]	1
[0]	0
R7_7...6:0]	40
[6]	1
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	0

2. Program Counter

- Design Source VHDL Code of Program Counter

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

entity Program_Counter is
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
       Reset : in STD_LOGIC;
       Clk : in STD_LOGIC;
       P_count : out STD_LOGIC_VECTOR (2 downto 0));
end Program_Counter;

architecture Behavioral of Program_Counter is

begin

process (Clk) begin
    if (rising_edge(Clk)) then
        if Reset = '1' then
            P_count <="000";
        else
            P_count <= D;
        end if;
    end if;
end process;

end Behavioral;
```

- RTL Schematic Diagram of the Program Counter



Behavioral Simulation Code for Program Counter

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

architecture Behavioral of TB_Program_Counter is

component Program_Counter is
    Port (
        D : in STD_LOGIC_VECTOR (2 downto 0);
       Reset : in STD_LOGIC;
       Clk : in STD_LOGIC;
       P_count : out STD_LOGIC_VECTOR (2 downto 0));

end component;

signal D, P_count : STD_LOGIC_VECTOR (2 downto 0);
signal  Reset,Clk: STD_LOGIC;



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


  signal Clk_in : STD_LOGIC:= '0' ;
  signal Clk_out : STD_LOGIC;



begin

UUT: Program_Counter
    port map (
    D=>D,
    Reset=>Reset,
    Clk=>Clk,
    P_count=>P_count

    );


  -- Instantiate the Unit Under Test (UUT)
 CLock_UUT: Slow_Clk
      Port map (
        Clk_in => Clk_in,
        Clk_out => Clk_out


      );

    -- Generate 5 rising edges (5 clock cycles)
    Clock_process: process
      begin
        Clk_in <= not Clk_in;
        wait for 5 ns;  -- 10ns full period = 100 MHz
      end process;
   Clk <= Clk_out;
    process
    begin

     -- Test case 1: SEL = 0, Y should be A
           D <= "011";
           Reset<= '0';
           wait for 200 ns;

           -- Test case 2: SEL = 1, Y should be B
           D <= "101";
           wait for 200 ns;
```

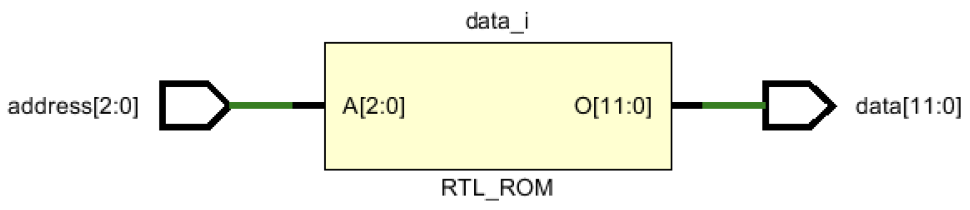
```
         D <= "110";
           wait for 200 ns;
          Reset<= '1';
           wait for 200 ns;
           Reset<= '0';
             D <= "010";
        wait for 200 ns;
        D <= "100";

    wait;
    end process;


end Behavioral;
```

3. Program ROM

- Design Source VHDL Code of Program ROM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;


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

architecture Behavioral of Program_ROM is

type rom_type is array (0 to 7) of std_logic_vector(11 downto 0);

    signal Program_ROM : rom_type := (
        "101110000001", -- 0        MOVI R7,1
        "100010000010", -- 1    MOVI R1.2
        "100100000011", -- 2        MOVI R2,3
        "001110010000", -- 3        ADD R7,R1
        "001110100000", -- 4        ADD R7,R2
        "110000000111", -- 5JZR R0,7
        "110000000111", -- 6        JZR R0,7
        "110000000111" -- 7 JZR R0,7
        );


begin
    data <= Program_ROM(to_integer(unsigned(address)));


end Behavioral;
```

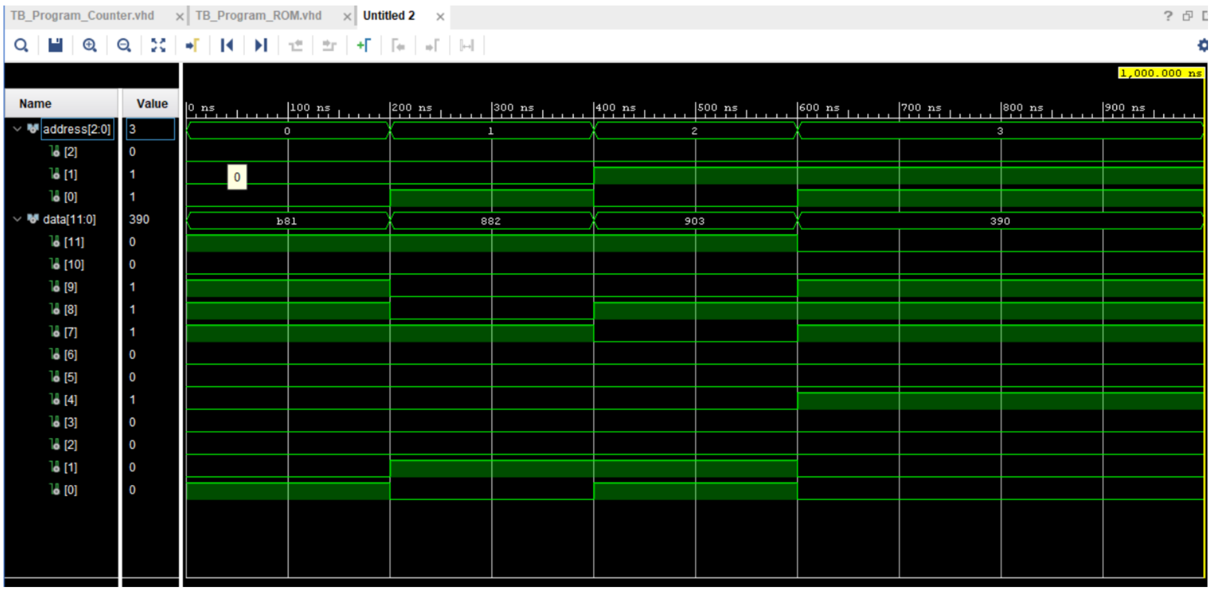
- RTL Schematic Diagram of the Program ROM



- <u>Behavioral Simulation Code for Program ROM</u>

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

    architecture Behavioral of TB_Program_ROM is

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

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



begin


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

process
begin

address <= "000";
wait for 200 ns;

address <= "001";
wait for 200 ns;
address <= "010";

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


wait;
end process;

end Behavioral;
```

- Timing Diagram for MUX_ Program ROM



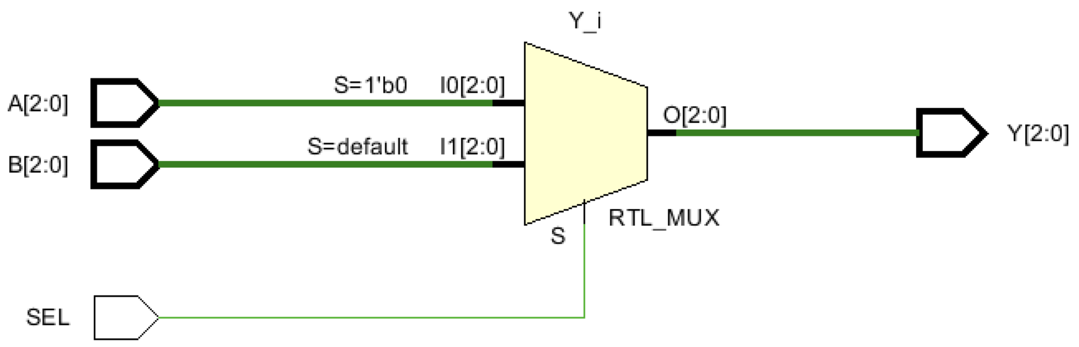
Name	Value	0–200 ns	200–400 ns	400–600 ns	600–1000 ns
address[2:0]	3	0	1	2	3
[2]	0				
[1]	1				
[0]	1				
data[11:0]	390	b81	882	903	390
[11]	0				
[10]	0				
[9]	1				
[8]	1				
[7]	1				
[6]	0				
[5]	0				
[4]	1				
[3]	0				
[2]	0				
[1]	0				
[0]	0				

4. MUX_2way_3bit

- Design Source VHDL Code of MUX_2way_3bit

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

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

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

entity Mux_2way_3bit is
    Port ( Sel : in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (2 downto 0);
           B : in STD_LOGIC_VECTOR (2 downto 0);
           Mux_out : out STD_LOGIC_VECTOR (2 downto 0));
end Mux_2way_3bit;

architecture Behavioral of Mux_2way_3bit is

begin
--Sel=0 for A, Sel=1 for B

    Mux_out(0) <= (A(0) AND (NOT Sel)) or (B(0) AND Sel);
    Mux_out(1) <= (A(1) AND (NOT Sel)) or (B(1) AND Sel);
    Mux_out(2) <= (A(2) AND (NOT Sel)) or (B(2) AND Sel);


end Behavioral;
```

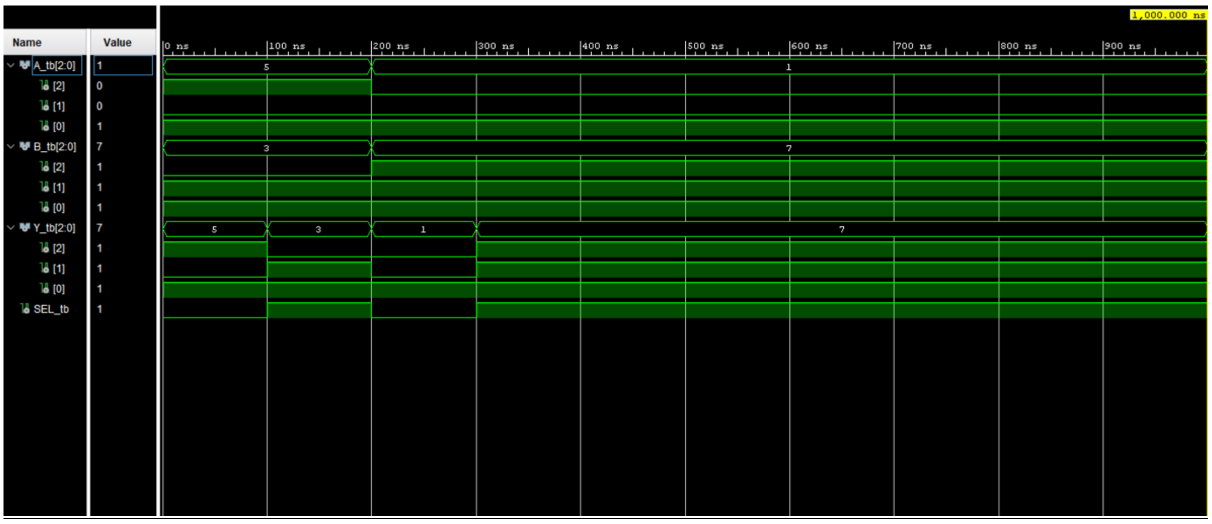
- RTL Schematic Diagram of the MUX_2way_3bit



- <u>Behavioral Simulation Code for MUX_2way_3bit</u>

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


entity TB_Mux_2way_3bit is
end TB_Mux_2way_3bit;

architecture Behavioral of TB_Mux_2way_3bit is

    -- Component declaration
    component Mux_2way_3bit is
        Port (
            A   : in  STD_LOGIC_VECTOR(2 downto 0);
            B   : in  STD_LOGIC_VECTOR(2 downto 0);
            SEL : in  STD_LOGIC;
            Y   : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;

    -- Testbench signals
    signal A_tb, B_tb, Y_tb : STD_LOGIC_VECTOR(2 downto 0);
    signal SEL_tb : STD_LOGIC;

begin

    -- Instantiate the Unit Under Test (UUT)
    UUT: Mux_2way_3bit
        port map (
            A   => A_tb,
            B   => B_tb,
            SEL => SEL_tb,
            Y   => Y_tb
        );

    -- Stimulus process
    process
    begin
        -- Test case 1: SEL = 0, Y should be A
        A_tb <= "101";
        B_tb <= "011";
        SEL_tb <= '0';
        wait for 100 ns;

        -- Test case 2: SEL = 1, Y should be B
        SEL_tb <= '1';
        wait for 100 ns;

        -- Test case 3: Change inputs
        A_tb <= "001";
        B_tb <= "111";
        SEL_tb <= '0';
        wait for 100 ns;

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

        -- Finish
        wait;
    end process;

end Behavioral;
```

- <u>Timing Diagram for MUX_2way_3bit</u>

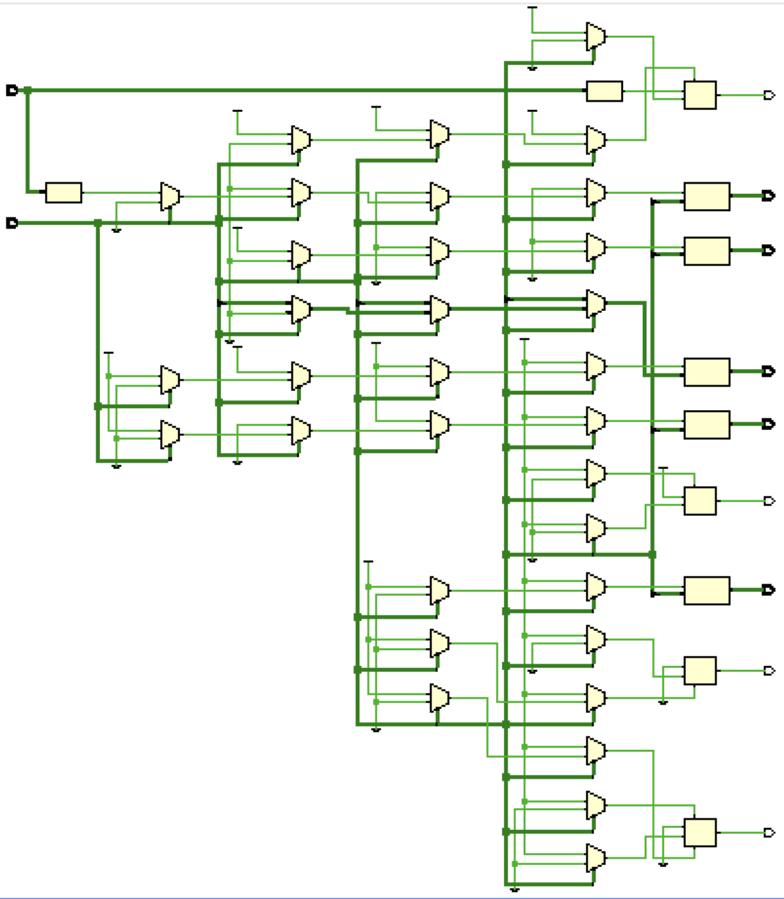


5. Instruction Decoder

- Design Source VHDL Code of Instruction Decoder

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


entity Instruction_Decoder is
    Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
           Jump_reg_val : in STD_LOGIC_VECTOR (3 downto 0); --register value for jump check
           Reg_Sel_A: out STD_LOGIC_VECTOR (2 downto 0);    --Mux Select signal
           Reg_Sel_B: out STD_LOGIC_VECTOR (2 downto 0);    --Muc Selct signal
           Reg_En :out STD_LOGIC_VECTOR (2 downto 0);       --Register Bank write enable signal
           Flag_Reg_En : out STD_LOGIC;     -- Enable for ADD_SUB_Units Flag
           Jump_Address : out STD_LOGIC_VECTOR (2 downto 0);
           Jump_Flag : out STD_LOGIC;
           Add_Sub_Sel : out STD_LOGIC; -- 0 for add , 1 for subtract
           Load_Sel : out STD_LOGIC;    -- 0 for immediarte value, 1 for add_sub  unit value
           Immediate_Value : out STD_LOGIC_VECTOR (3 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is

--signal Op_code : STD_LOGIC_VECTOR (1 downto 0);

begin

    Process (Instruction, Jump_reg_val)
    begin


        if Instruction (11 downto 10) = "00" then -- ADD
            Jump_Flag <= '0';
            Reg_Sel_A <= Instruction(9 downto 7);
            Reg_Sel_B <= Instruction(6 downto 4);
            Add_Sub_Sel <= '0';
            Load_Sel <= '1';
            Reg_En <= Instruction(9 downto 7);
            Flag_Reg_En <='1';

        elsif Instruction (11 downto 10) = "01" then -- Neg
            Jump_Flag <= '0';
            Reg_Sel_A <= Instruction(9 downto 7);
            Reg_Sel_B <= Instruction(6 downto 4);
            Add_Sub_Sel <= '1';
            Load_Sel <= '1';
            Reg_En <= Instruction(9 downto 7);
            Flag_Reg_En <='1';

        elsif Instruction (11 downto 10) = "10" then -- MOVI
            Jump_Flag <= '0';
            Load_Sel <= '0';
            Reg_En <= Instruction(9 downto 7);
            Flag_Reg_En <= '0';
            Immediate_value <= Instruction(3 downto 0);

        elsif Instruction (11 downto 10) = "11" then --JZR
            Reg_En <= "000";
            Flag_Reg_En <= '0';
            Reg_Sel_A <= Instruction(9 downto 7);
            if Jump_reg_val ="0000" then
                Jump_Flag <= '1';
                Jump_Address <= Instruction (2 downto 0);
            else
                Jump_Flag <= '0';
            end if;
        end if;
    end process;
end Behavioral;
```

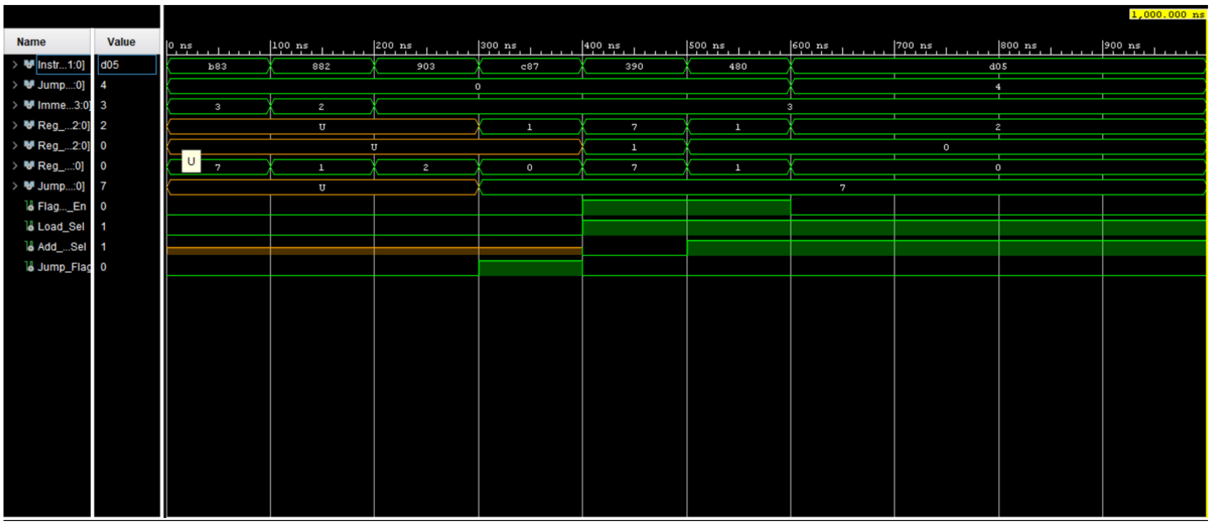
● RTL Schematic Diagram of the Instruction Decoder



● Behavioral Simulation Code for Instruction Decoder

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

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

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

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

architecture Behavioral of TB_Instruction_Decorder is

component Instruction_Decoder is
     Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
          Jump_reg_val : in STD_LOGIC_VECTOR (3 downto 0); --register value for jump check
          Reg_Sel_A: out STD_LOGIC_VECTOR (2 downto 0);    --Mux Selct signal
          Reg_Sel_B: out STD_LOGIC_VECTOR (2 downto 0);    --Muc Selct signal
          Reg_En :out STD_LOGIC_VECTOR (2 downto 0);       --Register Bank write enable signal
          Flag_Reg_En : out STD_LOGIC;     -- Enable for ADD_SUB_Units Flag
          Jump_Address : out STD_LOGIC_VECTOR (2 downto 0);
          Jump_Flag : out STD_LOGIC;
          Add_Sub_Sel : out STD_LOGIC; -- 0 for add , 1 for subtract
          Load_Sel : out STD_LOGIC;    -- 0 for immediarte value, 1 for add_sub  unit value
          Immediate_Value : out STD_LOGIC_VECTOR (3 downto 0));

end component;

signal Instruction : STD_LOGIC_VECTOR (11 downto 0);
signal Jump_reg_val ,Immediate_Value: STD_LOGIC_VECTOR (3 downto 0);
signal  Reg_Sel_A, Reg_Sel_B,Reg_En ,Jump_Address : STD_LOGIC_VECTOR (2 downto 0);
signal  Flag_Reg_En,Load_Sel,Add_Sub_Sel,Jump_Flag  : STD_LOGIC;


begin

UUT:Instruction_Decoder
    port map (
    Instruction=>Instruction,
    Jump_reg_val=>Jump_reg_val,
    Reg_Sel_A=> Reg_Sel_A,
    Reg_Sel_B=> Reg_Sel_B,
    Reg_En=> Reg_En,
    Flag_Reg_En=>Flag_Reg_En,
    Jump_Address=>Jump_Address,
    Jump_Flag=>Jump_Flag,
    Add_Sub_Sel=>Add_Sub_Sel,
    Load_Sel =>Load_Sel ,
    Immediate_Value=>Immediate_Value );

    process
    begin

        Jump_reg_val <= "0000";
        Instruction <="101110000011"; -- MOVI R7,1
        wait for 100ns;

        Instruction <="100010000010"; -- MOVI R1,2
        wait for 100ns;

        Instruction <= "100100000011"; --MOVI R2,3
        wait for 100ns;

        Instruction <="110010000111"; -- JZR R0,7
        wait for 100ns;

        Instruction <="001110010000"; -- ADD R7,R1
        wait for 100ns;

        Instruction <="010010000000"; -- NEG R1
        wait for 100ns;

        Jump_reg_val <= "0100";
        Instruction <="110100000101"; -- JZR R2,5

        wait;

    end process;

end Behavioral;
```

- <u>Timing Diagram for Instruction Decoder</u>

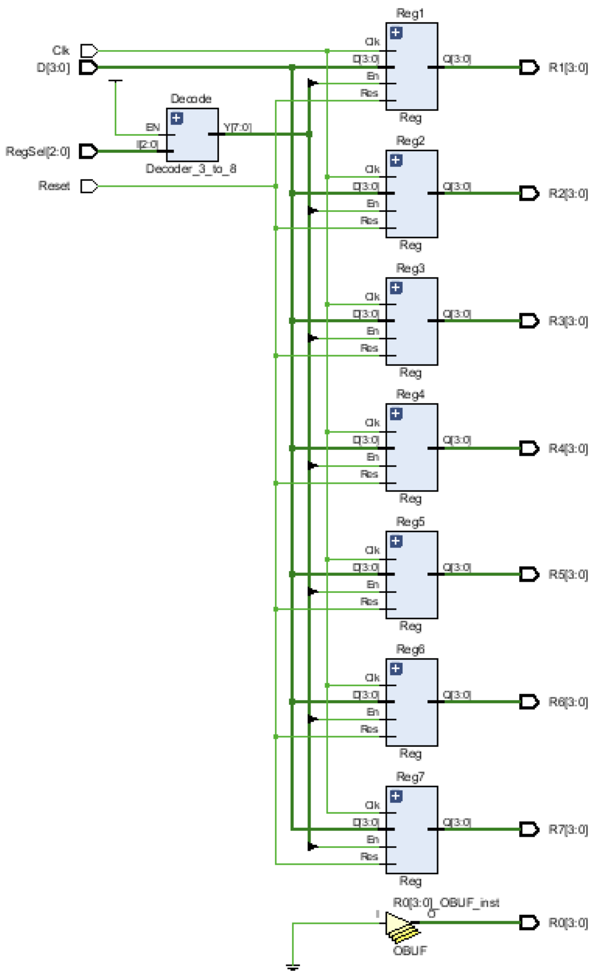


6. Register Bank

- <u>Design Source VHDL Code of Register Bank</u>

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

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

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

entity Reg_Bank is
    Port ( RegSel : in STD_LOGIC_VECTOR (2 downto 0);   --Register Select signal
           D : in STD_LOGIC_VECTOR (3 downto 0);    --Data line
           Reset: in STD_LOGIC;
           Clk : in STD_LOGIC;
           R0 : out STD_LOGIC_VECTOR (3 downto 0);
           R1 : out STD_LOGIC_VECTOR (3 downto 0);
           R2 : out STD_LOGIC_VECTOR (3 downto 0);
           R3 : out STD_LOGIC_VECTOR (3 downto 0);
           R4 : out STD_LOGIC_VECTOR (3 downto 0);
           R5 : out STD_LOGIC_VECTOR (3 downto 0);
           R6 : out STD_LOGIC_VECTOR (3 downto 0);
           R7 : out STD_LOGIC_VECTOR (3 downto 0));
end Reg_Bank;

architecture Behavioral of Reg_Bank is

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

component Decoder_3_to_8 is
    Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end component;

signal line_en : STD_LOGIC_VECTOR (7 downto 0);

begin

    Decode: Decoder_3_to_8
        port map(
        I => RegSel,
        En => '1',
        Y => line_en);

    R0 <= "0000";   --hard code R0 to 0000

    Reg1: Reg
        port map(
        D => D,
        En => line_en(1),
        Res => Reset,
        Clk => Clk,
        Q => R1);

    Reg2: Reg
        port map(
        D => D,
        En => line_en(2),
        Res => Reset,
        Clk => Clk,
        Q => R2);

    Reg3: Reg
        port map(
        D => D,
        En => line_en(3),
        Res => Reset,
        Clk => Clk,
        Q => R3);

    Reg4: Reg
        port map(
        D => D,
        En => line_en(4),
        Res => Reset,
        Clk => Clk,
        Q => R4);

    Reg5: Reg
        port map(
        D => D,
        En => line_en(5),
        Res => Reset,
        Clk => Clk,
        Q => R5);

    Reg6: Reg
        port map(
        D => D,
        En => line_en(6),
        Res => Reset,
        Clk => Clk,
        Q => R6);

    Reg7: Reg
        port map(
        D => D,
        En => line_en(7),
        Res => Reset,
        Clk => Clk,
        Q => R7);

end Behavioral;
```

- <u>RTL Schematic Diagram of the Register Bank</u>



- <u>Behavioral Simulation Code for Register Bank</u>

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

entity TB_Register_Bank is
end TB_Register_Bank;

architecture Behavioral of TB_Register_Bank is
component Reg_Bank is
    Port ( RegSel : in STD_LOGIC_VECTOR (2 downto 0);   --Register Select signal
           D : in STD_LOGIC_VECTOR (3 downto 0);    --Data line
           Reset: in STD_LOGIC;
           Clk : in STD_LOGIC;
           R0 : out STD_LOGIC_VECTOR (3 downto 0);
           R1 : out STD_LOGIC_VECTOR (3 downto 0);
           R2 : out STD_LOGIC_VECTOR (3 downto 0);
           R3 : out STD_LOGIC_VECTOR (3 downto 0);
           R4 : out STD_LOGIC_VECTOR (3 downto 0);
           R5 : out STD_LOGIC_VECTOR (3 downto 0);
           R6 : out STD_LOGIC_VECTOR (3 downto 0);
           R7 : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end component;

signal RegSel:STD_LOGIC_VECTOR (2 downto 0);
signal Reset,Clk:  STD_LOGIC;
signal R0,D,R1,R2,R3,R4,R5,R6,R7 :STD_LOGIC_VECTOR (3 downto 0);


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


  signal Clk_in : STD_LOGIC:= '0' ;
  signal Clk_out : STD_LOGIC;


begin

UUT: Reg_Bank
port map (
RegSel=>RegSel,
D=>D,
Reset=>Reset,
Clk=>Clk,
R0=>R0,
R1=>R1,
R2=>R2,
R3=>R3,
R4=>R4,
R5=>R5,
R6=>R6,
R7=>R7

);


  -- Instantiate the Unit Under Test (UUT)
 CLock_UUT: Slow_Clk
      Port map (
        Clk_in => Clk_in,
        Clk_out => Clk_out


      );

    -- Generate 5 rising edges (5 clock cycles)
    Clock_process: process
      begin
        Clk_in <= not Clk_in;
        wait for 5 ns;  -- 10ns full period = 100 MHz
      end process;
   Clk <= Clk_out;
    process
    begin
    RegSel<="001";
    Reset<='0';
    D<="1101";
     wait for 100 ns;

          RegSel<="010";
        D<="1001";
        wait for 100 ns;
```

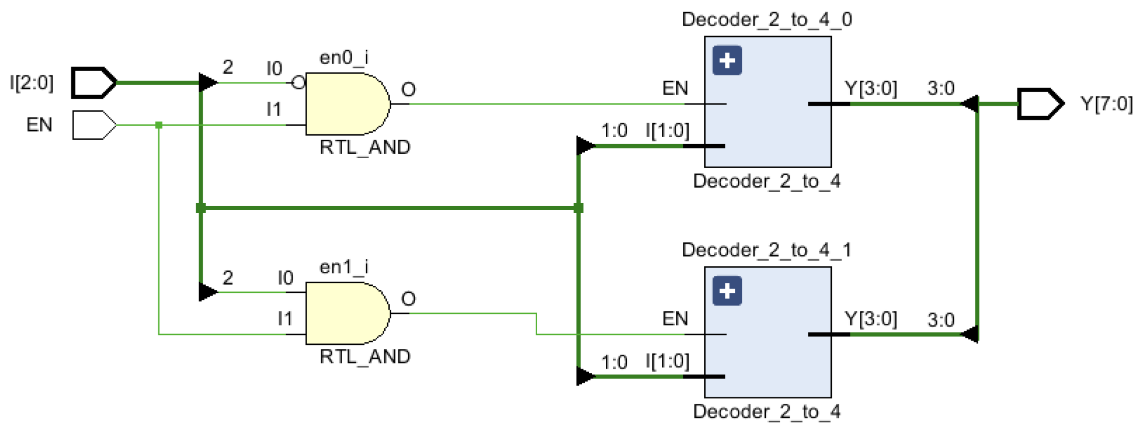
```
          RegSel<="110";
              D<="1011";
              wait for 100 ns;
               Reset<='1';
        wait for 100 ns;
        Reset<='0';
          RegSel<="011";
              D<="0001";
              wait for 100 ns;

         RegSel<="111";
             D<="0011";
  wait;
      end process;


end Behavioral;
```

7. Decoder_3_to_8

- Design Source VHDL Code of Decoder_3_to_8

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

entity Decoder_3_to_8 is
    Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end Decoder_3_to_8;

architecture Behavioral of Decoder_3_to_8 is

    component Decoder_2_to_4
        Port ( I : in STD_LOGIC_VECTOR;
               EN : in STD_LOGIC;
               Y : out STD_LOGIC_VECTOR );
    end component;

    signal I0,I1 :std_logic_vector(1 downto 0);
    signal Y0, Y1 :std_logic_vector(3 downto 0);
    signal en0,en1 :std_logic;

begin

    Decoder_2_to_4_0:Decoder_2_to_4
    port map(
        I=>I0,
        EN=>en0,
        Y=>Y0);

    Decoder_2_to_4_1:Decoder_2_to_4
    port map(
        I=>I1,
        EN=>en1,
        Y=>Y1);

    en0<= NOT(I(2)) AND EN;
    en1<= I(2) AND EN;
    I0<= I(1 downto 0);
    I1<= I(1 downto 0);
    --I2<=I(2);
    Y(3 downto 0) <= Y0;
    Y(7 downto 4) <= Y1;

end Behavioral;
```

- <u>RTL Schematic Diagram of the Decoder_3_to_8</u>

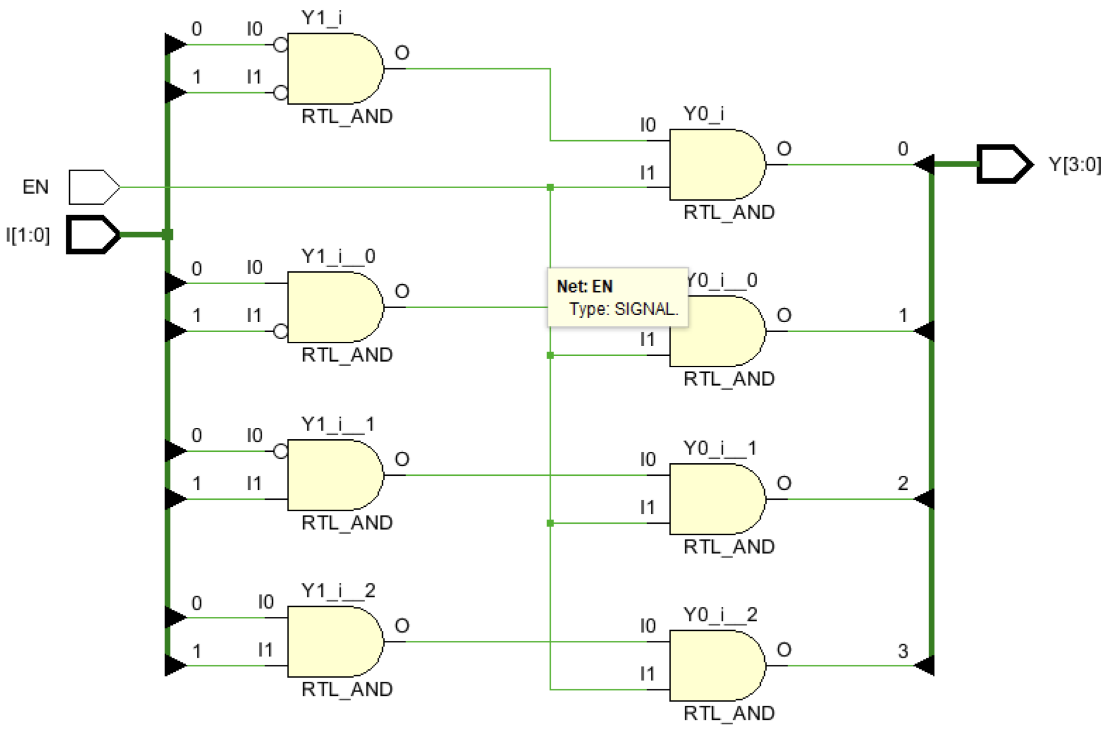


8. Decoder_2_to_4

- Design Source VHDL Code of Decoder_2_to_4

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

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

architecture Behavioral of Decoder_2_to_4 is

signal not_I0, not_I1 :std_logic;

begin
    not_I0 <= NOT I(0);
    not_I1 <= NOT I(1);

    Y(0) <= not_I0 AND not_I1 AND EN;
    Y(1) <= I(0) AND not_I1 AND EN;
    Y(2) <= not_I0 AND I(1) AND EN;
    Y(3) <= I(0) AND I(1) AND EN;

end Behavioral;
```

- RTL Schematic Diagram of the Decoder_2_to_4

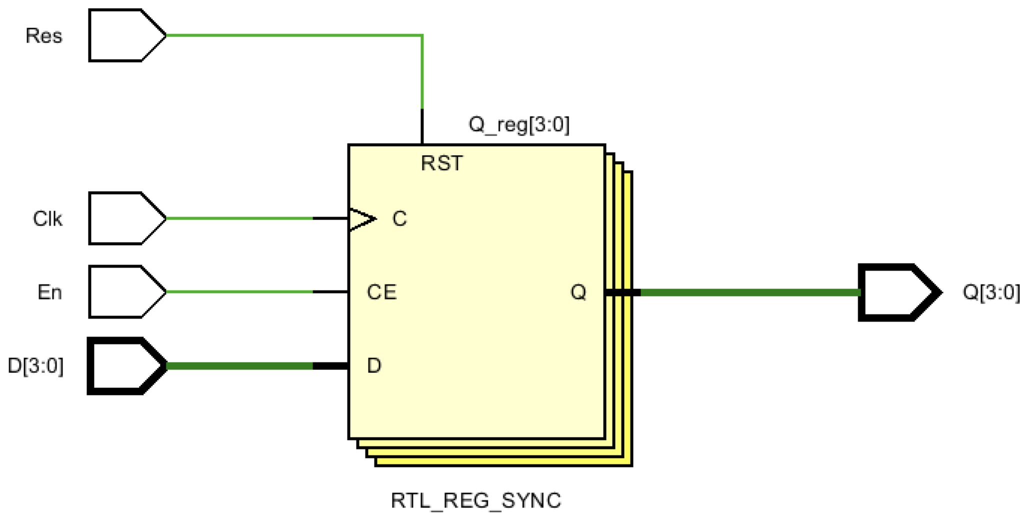


9. Register 4 bit

- <u>Design Source VHDL Code of Register 4 bit</u>

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

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

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

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

architecture Behavioral of Reg is

begin

process (Clk) begin
    if (rising_edge(Clk)) then
        if Res = '1' then
            Q <="0000";
        elsif En = '1' then
            Q <= D;
        end if;
    end if;
end process;


end Behavioral;
```

- <u>RTL Schematic Diagram of the Register 4 bit</u>

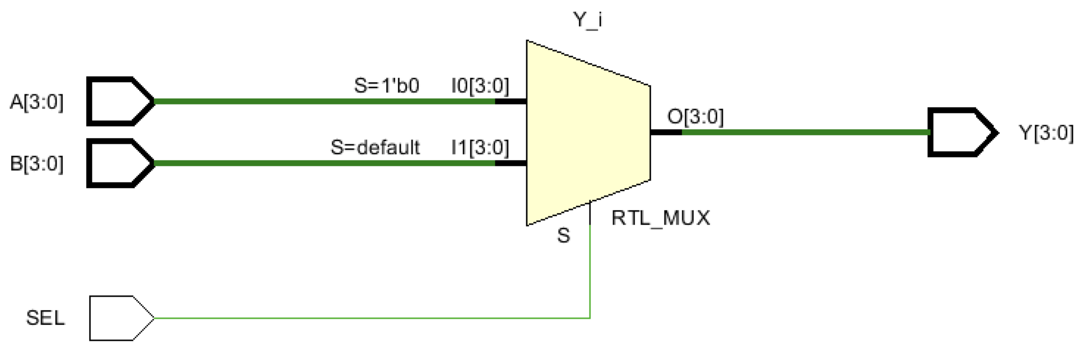


10. Mux_2_way_4bit

- Design Source VHDL Code of Mux_2_way_4bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Mux_2way_4bit is
    Port ( Sel : in STD_LOGIC;
       A : in STD_LOGIC_VECTOR (3 downto 0);
       B : in STD_LOGIC_VECTOR (3 downto 0);
       Mux_out : out STD_LOGIC_VECTOR (3 downto 0));
end Mux_2way_4bit;

architecture Behavioral of Mux_2way_4bit is

begin
--Sel=0 for A, Sel=1 for B

    Mux_out(0) <= (A(0) AND (NOT Sel)) or (B(0) AND Sel);
    Mux_out(1) <= (A(1) AND (NOT Sel)) or (B(1) AND Sel);
    Mux_out(2) <= (A(2) AND (NOT Sel)) or (B(2) AND Sel);
    Mux_out(3) <= (A(3) AND (NOT Sel)) or (B(3) AND Sel);

end Behavioral;
```

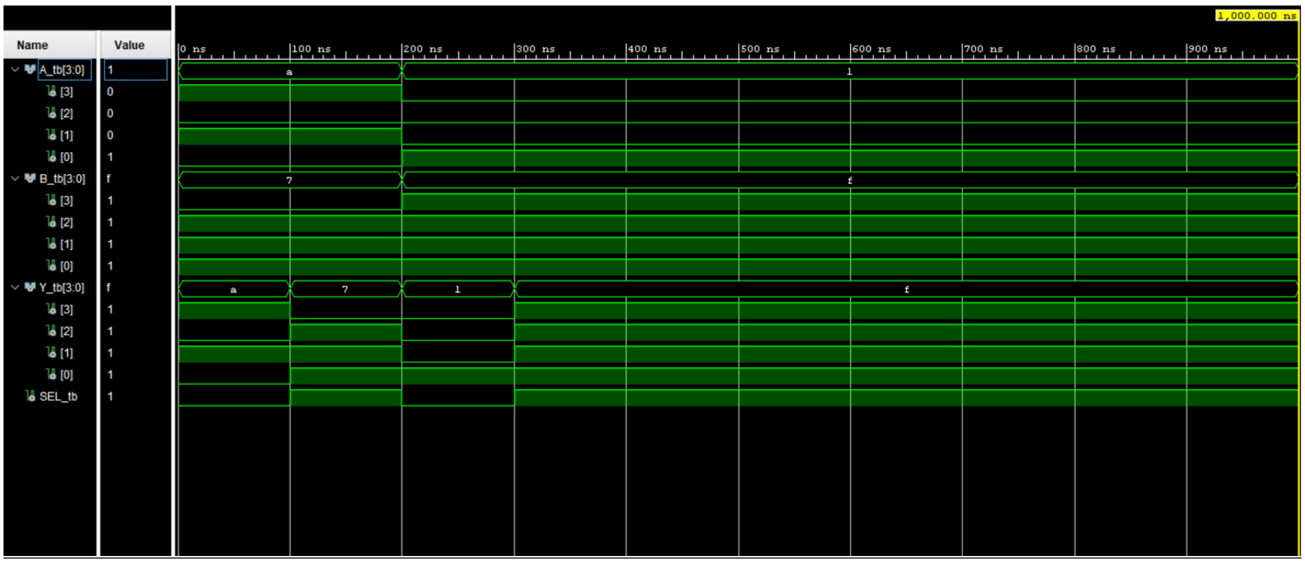
- RTL Schematic Diagram of the MUX_2_way_4bit



- <u>Behavioral Simulation Code for MUX_2_way_4bit</u>

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity TB_Mux_2way_4bit is
end TB_Mux_2way_4bit;

architecture Behavioral of TB_Mux_2way_4bit is

    -- Component declaration
    component Mux_2way_4bit is
        Port (
            A   : in  STD_LOGIC_VECTOR(3 downto 0);
            B   : in  STD_LOGIC_VECTOR(3 downto 0);
            SEL : in  STD_LOGIC;
            Y   : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    -- Testbench signals
    signal A_tb, B_tb, Y_tb : STD_LOGIC_VECTOR(3 downto 0);
    signal SEL_tb : STD_LOGIC;

begin

    -- Instantiate the Unit Under Test (UUT)
    UUT: Mux_2way_4bit
        port map (
            A   => A_tb,
            B   => B_tb,
            SEL => SEL_tb,
            Y   => Y_tb
        );

    -- Stimulus process
    process
    begin
        -- Test case 1: SEL = 0, Y should be A
        A_tb <= "1010";
        B_tb <= "0111";
        SEL_tb <= '0';
        wait for 100 ns;

        -- Test case 2: SEL = 1, Y should be B
        SEL_tb <= '1';
        wait for 100 ns;

        -- Test case 3: Change inputs
        A_tb <= "0001";
        B_tb <= "1111";
        SEL_tb <= '0';
        wait for 100 ns;

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

        -- Finish
        wait;
    end process;

end Behavioral;
```

- <u>Timing Diagram for MUX_2_way_4bit</u>

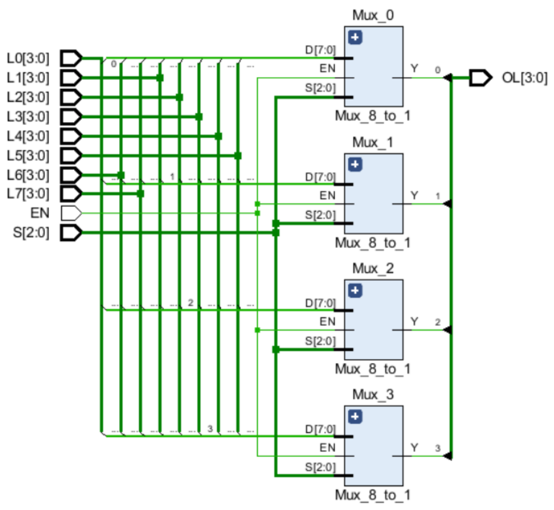


11. Mux_8_way_4bit

- Design Source VHDL Code of Mux_8_way_4bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Mux_8way_4bit is
    Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
           L0 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 0
           L1 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 1
           L2 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 2
           L3 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 3
           L4 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 4
           L5 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 5
           L6 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 6
           L7 : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit Input Line 7
           EN : in STD_LOGIC;
           OL: out STD_LOGIC_VECTOR (3 downto 0)); -- 4-bit output line
end Mux_8way_4bit;

architecture Behavioral of Mux_8way_4bit is

component Mux_8_to_1 is
    Port ( S : in STD_LOGIC_VECTOR (2 downto 0);
           D : in STD_LOGIC_VECTOR (7 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC);
end component;

signal D0 : STD_LOGIC_VECTOR (7 downto 0);
signal D1 : STD_LOGIC_VECTOR (7 downto 0);
signal D2 : STD_LOGIC_VECTOR (7 downto 0);
signal D3 : STD_LOGIC_VECTOR (7 downto 0);

begin

    D0 <= L7(0) & L6(0) & L5(0) & L4(0) & L3(0) & L2(0) & L1(0) & L0(0);
    D1 <= L7(1) & L6(1) & L5(1) & L4(1) & L3(1) & L2(1) & L1(1) & L0(1);
    D2 <= L7(2) & L6(2) & L5(2) & L4(2) & L3(2) & L2(2) & L1(2) & L0(2);
    D3 <= L7(3) & L6(3) & L5(3) & L4(3) & L3(3) & L2(3) & L1(3) & L0(3);

    Mux_0 : Mux_8_to_1
        port map(
        S => S,
        D => D0,
        EN => EN,
        Y => OL(0));

    Mux_1 : Mux_8_to_1
        port map(
        S => S,
        D => D1,
        EN => EN,
        Y => OL(1));

    Mux_2 : Mux_8_to_1
        port map(
        S => S,
        D => D2,
        EN => EN,
        Y => OL(2));

    Mux_3 : Mux_8_to_1
        port map(
        S => S,
        D => D3,
        EN => EN,
        Y => OL(3));

end Behavioral;
```

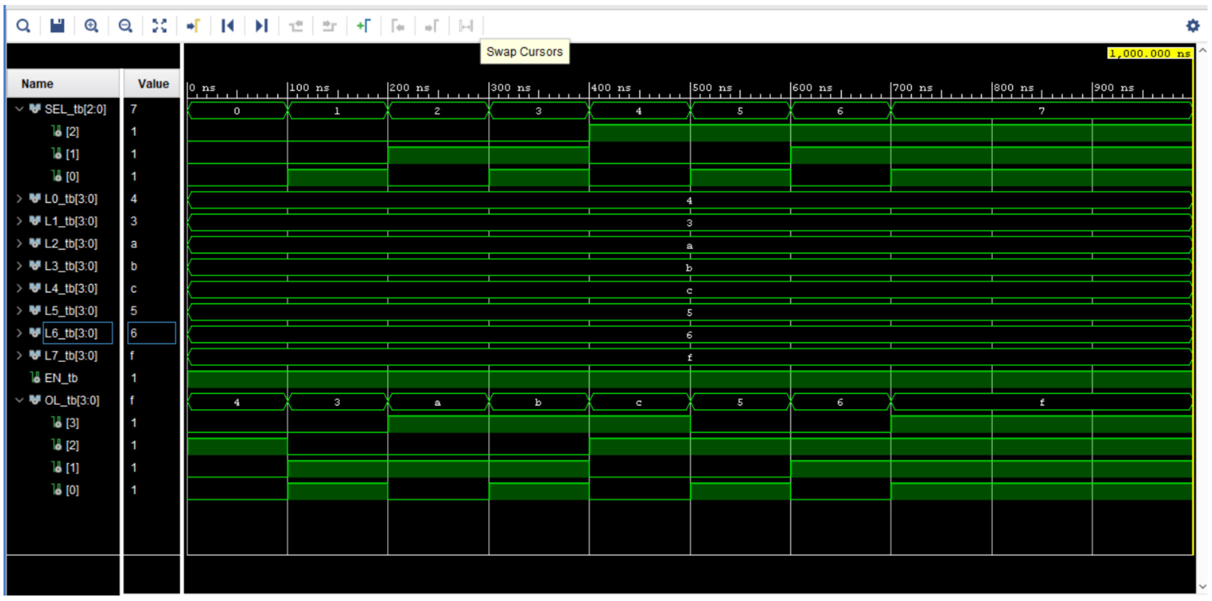
- <u>RTL Schematic Diagram of the MUX_8_way_4bit</u>



- <u>Behavioral Simulation Code for MUX_8_way_4bit</u>

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

entity TB_Mux_8way_4bit is
end TB_Mux_8way_4bit;

architecture Behavioral of TB_Mux_8way_4bit is

    component Mux_8way_4bit is
        Port (
            S  : in STD_LOGIC_VECTOR (2 downto 0);
            L0 : in STD_LOGIC_VECTOR (3 downto 0);
            L1 : in STD_LOGIC_VECTOR (3 downto 0);
            L2 : in STD_LOGIC_VECTOR (3 downto 0);
            L3 : in STD_LOGIC_VECTOR (3 downto 0);
            L4 : in STD_LOGIC_VECTOR (3 downto 0);
            L5 : in STD_LOGIC_VECTOR (3 downto 0);
            L6 : in STD_LOGIC_VECTOR (3 downto 0);
            L7 : in STD_LOGIC_VECTOR (3 downto 0);
            EN : in STD_LOGIC;
            OL : out STD_LOGIC_VECTOR (3 downto 0)
        );
    end component;

    signal SEL_tb : std_logic_vector(2 downto 0);
    signal L0_tb, L1_tb, L2_tb, L3_tb, L4_tb, L5_tb, L6_tb, L7_tb : std_logic_vector(3 downto 0);
    signal EN_tb : std_logic := '1';  -- Enable signal set to '1'
    signal OL_tb : std_logic_vector(3 downto 0);

begin

    UUT: Mux_8way_4bit
        port map(
            L0 => L0_tb,
            L1 => L1_tb,
            L2 => L2_tb,
            L3 => L3_tb,
            L4 => L4_tb,
            L5 => L5_tb,
            L6 => L6_tb,
            L7 => L7_tb,
            S  => SEL_tb,
            EN => EN_tb,
            OL => OL_tb
        );

    -- Stimulus process
    process
    begin
        -- Initialize inputs
        L0_tb <= "0100";
        L1_tb <= "0011";
        L2_tb <= "1010";
        L3_tb <= "1011";
        L4_tb <= "1100";
        L5_tb <= "0101";
        L6_tb <= "0110";
        L7_tb <= "1111";

        EN_tb <= '1';  -- Enable the multiplexer

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

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

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

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

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

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

        SEL_tb <= "110";
         wait for 100 ns;

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

        wait;
    end process;

end Behavioral;
```

- <u>Timing Diagram for MUX_8_way_4bit</u>

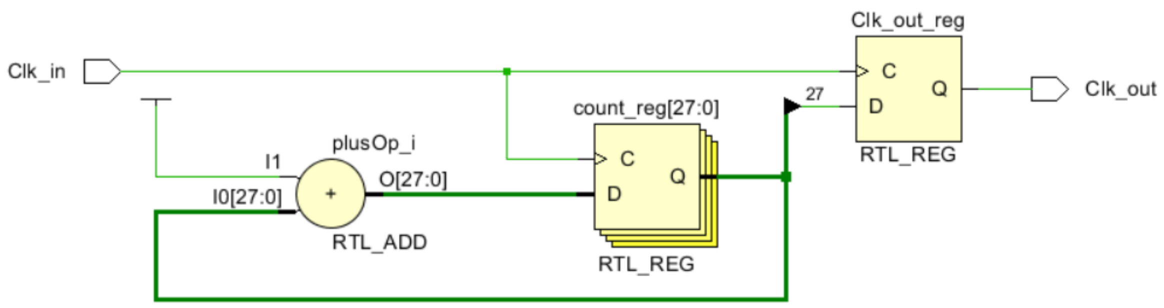


12. Slow_Clk

- <u>Design Source VHDL Code of Slow_Clk</u>

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.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: unsigned(27 downto 0) := (others => '0'); --28 bit counter
--signal clk_status: std_logic := '0';

begin

    process (Clk_in) begin
        if (rising_edge (Clk_in)) then
            count <= count+1;
            Clk_out <= count(27);
            --for simulation use count(2) => output clock frequency = 1/4 *orginal
clk frequence
            --Basys board use count(27) => clk_out will change for every
134,217,728 clk cycles
            --which means slow clk cycle is ~2.685 seconds  , frequency is ~0.3725
Hz
        end if;
    end process;
end Behavioral;
```

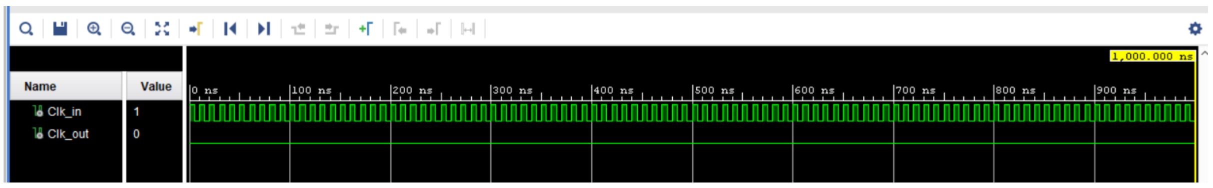
- <u>RTL Schematic Diagram of the Slow_Clk</u>



- <u>Behavioral Simulation Code for Slow_Clk</u>

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

architecture Behavioral of Slow_Clk_Sim is

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

  -- Testbench Signals

  signal Clk_in : STD_LOGIC:= '0' ;
  signal Clk_out : STD_LOGIC;

begin

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

  -- Generate 5 rising edges (5 clock cycles)
  Clock_process: process
    begin
      Clk_in <= not Clk_in;
      wait for 5 ns;  -- 10ns full period = 100 MHz
    end process;

end Behavioral;
```

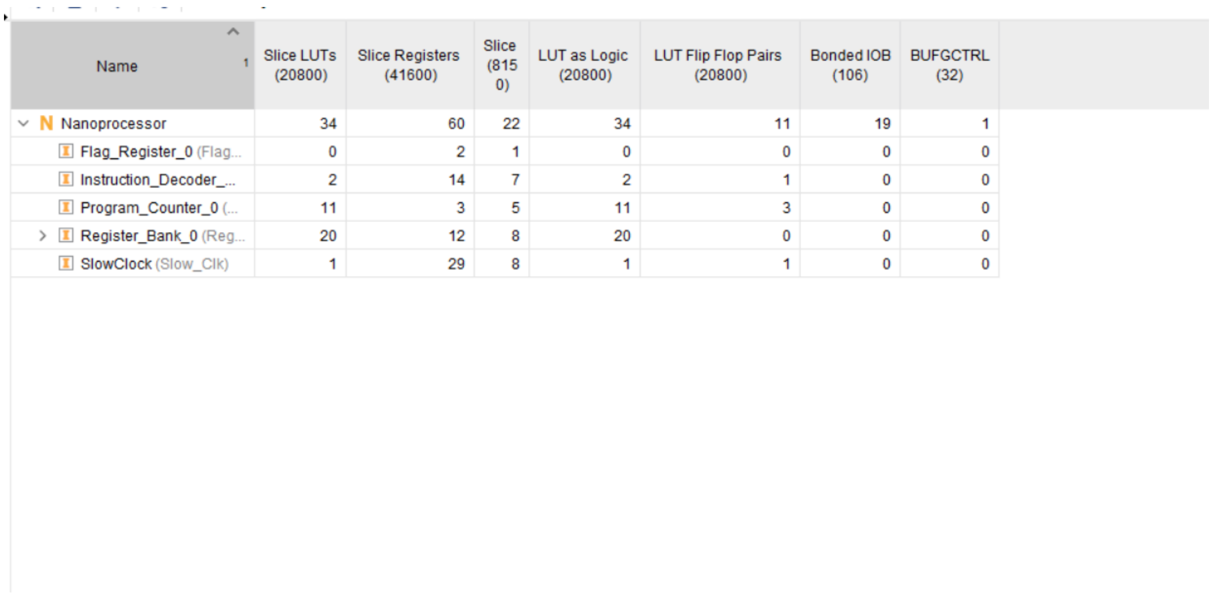
- <u>Timing Diagram for slow clock</u>



13. Constraint File

```
level signal names in the project

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



## LEDs
set_property PACKAGE_PIN U16 [get_ports {R7_LED[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[0]}]
set_property PACKAGE_PIN E19 [get_ports {R7_LED[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[1]}]
set_property PACKAGE_PIN U19 [get_ports {R7_LED[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[2]}]
set_property PACKAGE_PIN V19 [get_ports {R7_LED[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[3]}]
set_property PACKAGE_PIN P1 [get_ports {Zero_LED}]
        set_property IOSTANDARD LVCMOS33 [get_ports {Zero_LED}]
set_property PACKAGE_PIN L1 [get_ports {Over_F_LED}]
        set_property IOSTANDARD LVCMOS33 [get_ports {Over_F_LED}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {R7_7seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {R7_7seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {R7_7seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {R7_7seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {R7_7seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {R7_7seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {R7_7seg[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[6]}]

#set_property PACKAGE_PIN V7 [get_ports dp]
        #set_property IOSTANDARD LVCMOS33 [get_ports dp]

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


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

♦ Resource Utilization



Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (106)	BUFGCTRL (32)
Nanoprocessor	34	60	22	34	11	19	1
Flag_Register_0 (Flag...	0	2	1	0	0	0	0
Instruction_Decoder_...	2	14	7	2	1	0	0
Program_Counter_0 (...	11	3	5	11	3	0	0
Register_Bank_0 (Reg...	20	12	8	20	0	0	0
SlowClock (Slow_Clk)	1	29	8	1	1	0	0

2) **Design with extended qualifications**

♦ Executable Instruction Set

Instruction	Description	Format (17-bit instruction)
MOVI R, d	Move immediate value d to register R, i.e., R ← d, R ∈ [0, 7], d ∈ [0, 255]	0 1 0 R R R 0 0 0 d d d d d d d d
ADD Ra, Rb	Add values in registers Ra and Rb and store the result in Ra, i.e., Ra ←Ra + Rb Ra, Rb ∈ [0, 7]	0 0 0 Ra Ra Ra Rb Rb Rb 00000000
NEG R	2's complement of registers R, i.e., R ← – R R ∈ [0, 7]	0 0 1 R R R 000 0 0 0 0 0 0 0 0
JZR R, d	Jump if value in register R is 0, i.e., If R == 0 PC ←d; Else PC ← PC + 1; R ∈ [0, 7], d ∈ [0, 15]	0 1 1 R R R 0 0 0 0 0 0 0 d d d d
MUL Ra, Rb	Multiply values in registers Ra and Rb and store the result in Ra, i.e., Ra ← Ra x Rb Ra, Rb ∈ [0, 127] (2's complement positive numbers only)	1 0 0 Ra Ra Ra Rb Rb Rb 00000000
JUMP d	Jump to the address d d ∈ [0, 15]	1 0 1 0 0 0 0 0 0 0 0 0 0 d d d d
JNZ R, d	Jump if value in register R is not 0, i.e., If R != 0 PC ←d; Else PC ← PC + 1; R ∈ [0, 7], d ∈ [0, 15]	1 1 0 R R R 0 0 0 0 0 0 0 d d d d
COPY Ra Rb	Copy the value in Rb to Ra Ra ← Rb; R ∈ [0, 7]	1 1 1 Ra Ra Ra Rb Rb Rb 00000000

♦ Design Components

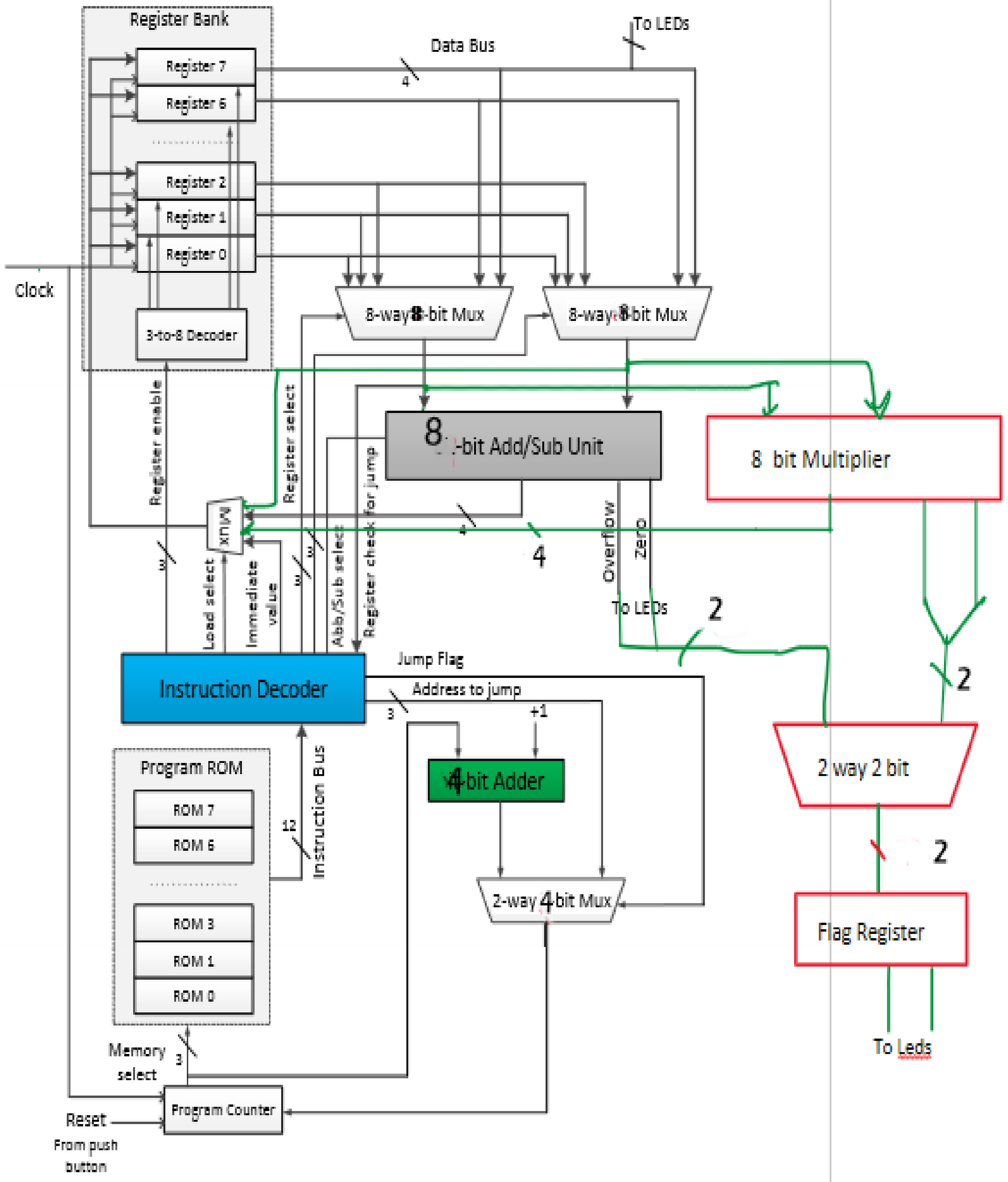


Figure 1 – High-level diagram of the nanoprocessor.

Page 2 of 5

1. Improved Nano Processor

- Design Source VHDL Code of improved Nano Processor

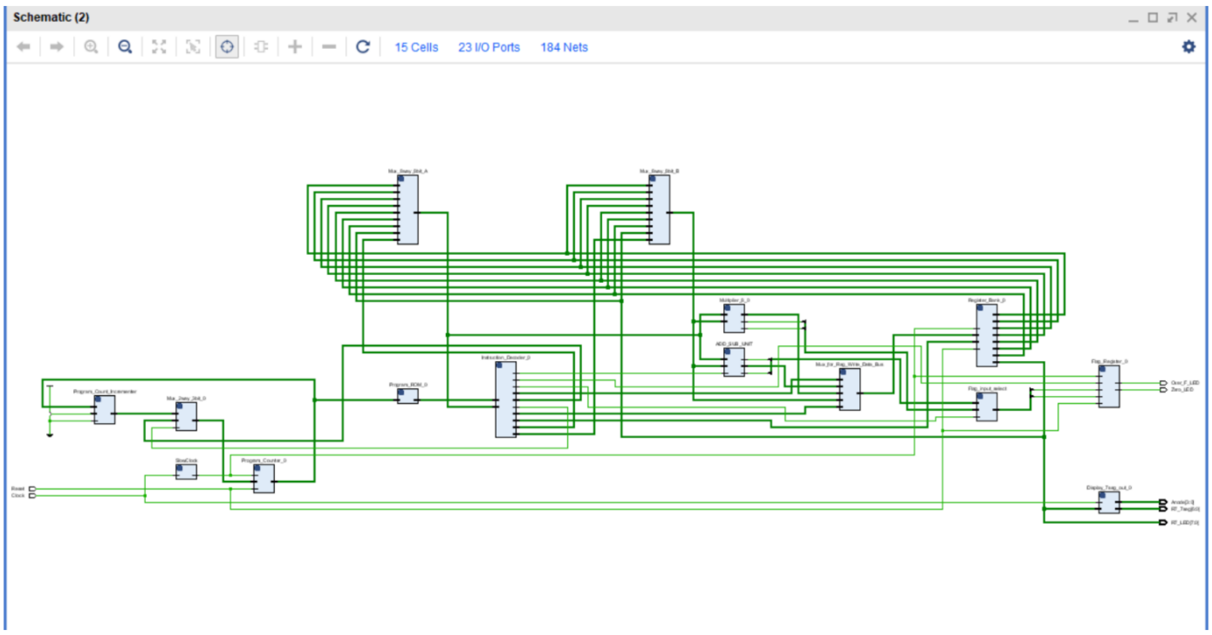
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

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

entity NanoProcessor_Improved is
    Port ( Reset : in STD_LOGIC;
       Clock : in STD_LOGIC;
       R7_LED : out STD_LOGIC_VECTOR (7 downto 0);
       Zero_LED : out STD_LOGIC;
       Over_F_LED : out STD_LOGIC;
       Anode : out STD_LOGIC_VECTOR (3 downto 0);
       R7_7seg : out STD_LOGIC_VECTOR (6 downto 0));
end NanoProcessor_Improved;

architecture Behavioral of NanoProcessor_Improved is

----- Components ------

----- Slow Clock >>>> Slowed Clock signal so easy to observe --------
component Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end component;

-----Instruction Decoder-----------------------------
component Instruction_Decoder is
    Port ( Instruction : in STD_LOGIC_VECTOR (16 downto 0);
           Jump_reg_val : in STD_LOGIC_VECTOR (7 downto 0); --register value for jump check
           Reg_Sel_A: out STD_LOGIC_VECTOR (2 downto 0);    --Mux Select signal
           Reg_Sel_B: out STD_LOGIC_VECTOR (2 downto 0);    --Muc Selct signal
           Reg_En :out STD_LOGIC_VECTOR (2 downto 0);       --Register Bank write enable signal
           Flag_Reg_En : out STD_LOGIC;     -- Enable for ADD_SUB_Units and Multiplier Flag
           Flag_Reg_Input_Sel: out STD_LOGIC; -- 0 for ADD_SUB_unit ,1 for multiplier
           Jump_Address : out STD_LOGIC_VECTOR (3 downto 0);
           Jump_Flag : out STD_LOGIC;
           Add_Sub_Sel : out STD_LOGIC; -- 0 for add , 1 for subtract
           Load_Sel : out STD_LOGIC_VECTOR (1 downto 0);  -- 00 for immediarte value, 01 for add_sub
unit value
                                                         --10 for multiplier,  11 for Copy line
           Immediate_Value : out STD_LOGIC_VECTOR (7 downto 0));
end component;

---- Register Bank (8 8-bit registers) -------
component Reg_Bank is
    Port ( RegSel : in STD_LOGIC_VECTOR (2 downto 0);   --Register Select signal
           D : in STD_LOGIC_VECTOR (7 downto 0);    --Data line
           Reset: in STD_LOGIC;
           Clk : in STD_LOGIC;
           R0 : out STD_LOGIC_VECTOR (7 downto 0);
           R1 : out STD_LOGIC_VECTOR (7 downto 0);
           R2 : out STD_LOGIC_VECTOR (7 downto 0);
           R3 : out STD_LOGIC_VECTOR (7 downto 0);
           R4 : out STD_LOGIC_VECTOR (7 downto 0);
           R5 : out STD_LOGIC_VECTOR (7 downto 0);
           R6 : out STD_LOGIC_VECTOR (7 downto 0);
           R7 : out STD_LOGIC_VECTOR (7 downto 0));
end component;
```

```
---- Flag Register  (Zero, Overflow) --------
component Flag_register is
    Port ( Zero : in STD_LOGIC;     --zero signal from ADD_SUB_Unit
           Over_f : in STD_LOGIC;   --overflow signal from ADD_SUB_Unit
           En : in STD_LOGIC;
           reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Zero_Flag : out STD_LOGIC;
           OverFlow_Flag : out STD_LOGIC);
end component;

---- Program Counter (register) --------
component Program_Counter is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
       Reset : in STD_LOGIC;
       Clk : in STD_LOGIC;
       P_count : out STD_LOGIC_VECTOR (3 downto 0));
end component;

---- 8-bit ADD_SUB_Unit -----------
component ADD_SUB_8bit is
    Port ( Ins : in STD_LOGIC; -- instruction (ADD =0, SUB = 1) (also works as C_in)
    A : in STD_LOGIC_VECTOR (7 DOWNTO 0);
    B : in STD_LOGIC_VECTOR (7 DOWNTO 0);
    S : out STD_LOGIC_VECTOR (7 DOWNTO 0);
    O_flow : out STD_LOGIC;
    Zero: out STD_LOGIC);
end component;

---- 8 bit Multiplier -------------------
component Multiplier_8bit is
    Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           Ans : out STD_LOGIC_VECTOR (7 downto 0);
           Over_f : out STD_LOGIC;
           Zero : out STD_LOGIC);
end component;

---- 4-bit RCA --(Adder)-----------------
component RCA_4bit is
    Port (
    A : in STD_LOGIC_VECTOR (3 DOWNTO 0);
    B : in STD_LOGIC_VECTOR (3 DOWNTO 0);
    S : out STD_LOGIC_VECTOR (3 DOWNTO 0);
    C_in :in STD_LOGIC;
    C_out: out STD_LOGIC);
end component;

---- 8 way 8-bit Multiplexer ----------
component Mux_8way_8bit is
    Port (
        D0, D1, D2, D3 : in  STD_LOGIC_VECTOR(7 downto 0);
        D4, D5, D6, D7 : in  STD_LOGIC_VECTOR(7 downto 0);
        SEL             : in  STD_LOGIC_VECTOR(2 downto 0);  -- 3-bit selection
        Y               : out STD_LOGIC_VECTOR(7 downto 0)   -- 8-bit output
    );
end component;

---- 4 way 8-bit Multiplexer ----------
component Mux_4way_8bit is
    Port (
        D0, D1, D2, D3 : in  STD_LOGIC_VECTOR(7 downto 0);
        SEL             : in  STD_LOGIC_VECTOR(1 downto 0);  -- 2-bit selection
        Y               : out STD_LOGIC_VECTOR(7 downto 0)   -- 8-bit output
    );
end component;

---- 2 way 4-bit Multiplexer ----------
component Mux_2way_4bit is
    Port (
        A   : in  STD_LOGIC_VECTOR(3 downto 0);  -- 4-bit input A
        B   : in  STD_LOGIC_VECTOR(3 downto 0);  -- 4-bit input B
        SEL : in  STD_LOGIC;                      -- Selection line
        Y   : out STD_LOGIC_VECTOR(3 downto 0)   -- 4-bit output
```

```vhdl
    );
end component;

---- 2 way 2-bit Multiplexer ----------
component Mux_2way_2bit is
    Port ( A : in STD_LOGIC_VECTOR (1 downto 0);
           B : in STD_LOGIC_VECTOR (1 downto 0);
           SEL: in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (1 downto 0));
end component;

----- Program ROM >>>> Stored Machine language Assembly code -------
component Program_ROM is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (16 downto 0));
end component;

---- 7 Segment display outputs setup----
component Display_7seg_out is
    Port ( Number : in STD_LOGIC_VECTOR (7 downto 0);
           Clock : in STD_LOGIC;
           Cathode : out STD_LOGIC_VECTOR (6 downto 0);
           Anode : out STD_LOGIC_VECTOR (3 downto 0));
end component;


-----Signals--------------------
signal Slow_clock :std_logic;
signal instruction : std_logic_vector (16 downto 0);
signal Mux_A_out ,Mux_B_out : std_logic_vector (7 downto 0); -- 8way MUX outputs
signal Mux_A_Sel, Mux_B_Sel : std_logic_vector (2 downto 0); -- 8way MUX Select signal
signal Reg_EN : std_logic_vector (2 downto 0); -- Register bank select reg for write enable signal
signal Flag_Reg_EN : std_logic; --flag register enable signal
signal Flag_Reg_Input_Sel: std_logic;
signal Address_to_jump : std_logic_vector (3 downto 0);
signal Jump_Flag : std_logic;
signal Add_Sub_Sel : std_logic;
signal Load_Sel : std_logic_vector (1 downto 0);   -- load select for register write data bus
signal Immediate_Value:  std_logic_vector (7 downto 0);

signal Reg_Data_in: std_logic_vector (7 downto 0); --Register Data in Bus
--Register Data outs
signal Reg_0, Reg_1, Reg_2, Reg_3, Reg_4, Reg_5, Reg_6, Reg_7: std_logic_vector (7 downto 0);
signal Add_Sub_Out : std_logic_vector (7 downto 0);
signal Multiplier_Out : std_logic_vector (7 downto 0);
signal Flag_reg_in_2bit :  std_logic_vector (1 downto 0);
signal adder_O_flow_Zero : std_logic_vector (1 downto 0);
signal mul_O_flow_Zero : std_logic_vector (1 downto 0);
signal O_flow_Zero : std_logic_vector (1 downto 0);

signal Program_C_in, Program_C_out: std_logic_vector (3 downto 0);
signal Program_C_Incrementer_out : std_logic_vector (3 downto 0);

begin

    SlowClock : Slow_Clk
        port map(
        Clk_in => Clock,
        Clk_out => Slow_clock );

    Instruction_Decoder_0: Instruction_Decoder
        port map(
        Instruction     => instruction,
        Jump_reg_val    => Mux_A_out,
        Reg_Sel_A       => Mux_A_Sel,
        Reg_Sel_B       => Mux_B_Sel,
        Reg_En          => Reg_EN,
        Flag_Reg_En     => Flag_Reg_EN,
        Flag_Reg_Input_Sel => Flag_Reg_Input_Sel,
        Jump_Address    => Address_to_jump,
        Jump_Flag       => Jump_Flag,
        Add_Sub_Sel     => Add_Sub_Sel,
        Load_Sel        => Load_Sel,
        Immediate_Value => Immediate_Value);
```

```
Register_Bank_0 : Reg_Bank
    port map(
    RegSel  => Reg_EN,
    D       => Reg_Data_in,
    Reset   => Reset,
    Clk     => Slow_clock,
    R0      => Reg_0,
    R1      => Reg_1,
    R2      => Reg_2,
    R3      => Reg_3,
    R4      => Reg_4,
    R5      => Reg_5,
    R6      => Reg_6,
    R7      => Reg_7);


Mux_8way_8bit_A: Mux_8way_8bit
    port map(
    D0  => Reg_0,
    D1  => Reg_1,
    D2  => Reg_2,
    D3  => Reg_3,
    D4  => Reg_4,
    D5  => Reg_5,
    D6  => Reg_6,
    D7  => Reg_7,
    SEL => Mux_A_Sel,
    Y   => Mux_A_out);


Mux_8way_8bit_B: Mux_8way_8bit
    port map(
    D0  => Reg_0,
    D1  => Reg_1,
    D2  => Reg_2,
    D3  => Reg_3,
    D4  => Reg_4,
    D5  => Reg_5,
    D6  => Reg_6,
    D7  => Reg_7,
    SEL => Mux_B_Sel,
    Y   => Mux_B_out);

ADD_SUB_UNIT: Add_Sub_8bit
    port map(
    Ins => Add_Sub_Sel,
    A   => Mux_A_out,
    B   => Mux_B_out,
    S   => Add_Sub_Out,
    O_flow  => adder_O_flow_Zero(1),
    Zero    => adder_O_flow_Zero(0));

Multiplier_8_0: Multiplier_8bit
    Port map(
    A => Mux_A_out,
    B => Mux_B_out,
    Ans => Multiplier_Out,
    Over_f => mul_O_flow_Zero(1),
    Zero  => mul_O_flow_Zero(0));

Mux_for_Reg_Write_Data_Bus: Mux_4way_8bit
    port map(
    D0   => Immediate_Value,
    D1  => Add_Sub_Out,
    D2  => Multiplier_Out,
    D3  => Mux_B_out,
    SEL => Load_Sel,
    Y   => Reg_Data_in);

Flag_input_select :Mux_2way_2bit
    port map(
    A   => adder_O_flow_Zero,
    B   => mul_O_flow_Zero,
        SEL => Flag_Reg_Input_Sel,
        Y   => O_flow_Zero);

    Flag_Register_0 :Flag_register
        port map(
        Zero     => O_flow_Zero(0),
        Over_f   => O_flow_Zero(1),
        En       => Flag_Reg_EN,
        reset    => Reset,
        Clk      => Slow_clock,
        Zero_Flag  =>   Zero_LED,
        OverFlow_Flag => Over_F_LED);

    Program_Counter_0: Program_Counter
        port map(
        D        => Program_C_in,
        Reset    => Reset,
        Clk      => Slow_clock,
        P_count => Program_C_out);

    Program_Count_Incrementer: RCA_4bit
        port map(
        A    => Program_C_out,
        B    => "0001",
        C_in => '0',
        S    => Program_C_Incrementer_out,
        C_out => open);

    Mux_2way_3bit_0 : Mux_2way_4bit
        port map(
        A    => Program_C_Incrementer_out,
        B    => Address_to_jump,
        SEL => Jump_Flag,
        Y    => Program_C_in);

    Program_ROM_0: Program_ROM
        Port map(
        address => Program_C_out,
        data     => instruction);

---- Output Settings ----------------
    -- Zero and Overflow outputs done in above

    --LED output
    R7_LED   <= Reg_7;

    --7 segmented Display out
    Display_7seg_out_0 :Display_7seg_out
        port map(
        Number   => Reg_7,
        Clock    => Clock,
        Cathode => R7_7seg,
        Anode    => Anode);



end Behavioral;
```

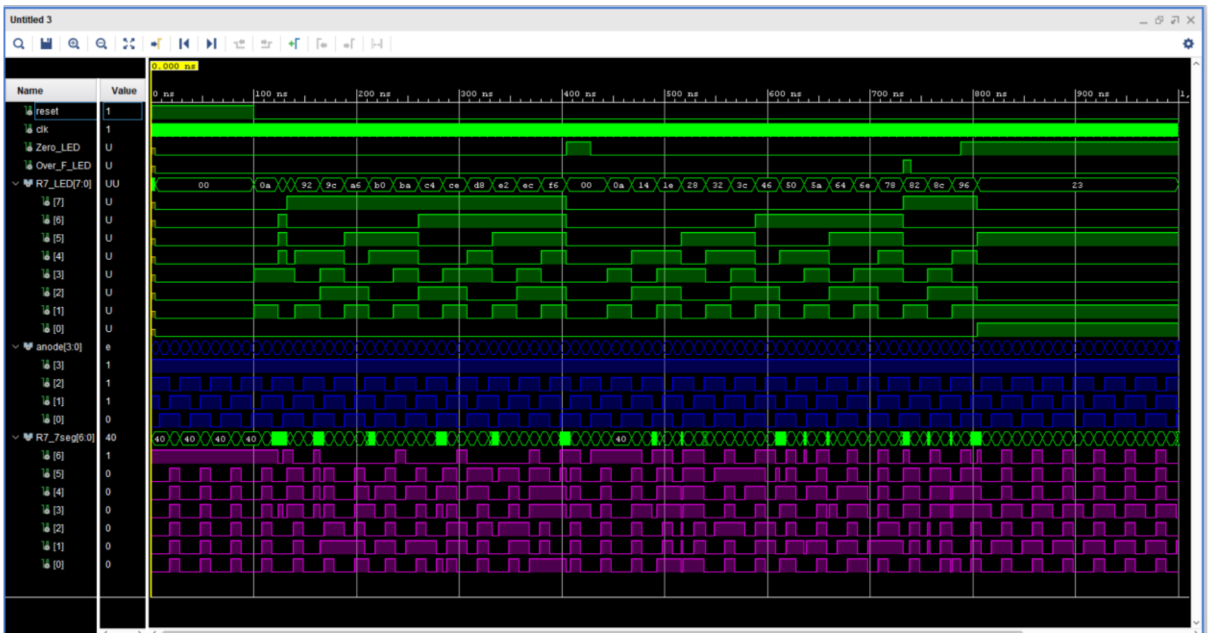
- <u>RTL Schematic Diagram of the improved Nano Processor</u>



- <u>Behavioral Simulation Code for improved Nano Processor</u>

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

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

architecture Behavioral of TB_NanoProcessor_Improved is

component NanoProcessor_Improved is
    Port ( Reset : in STD_LOGIC;
       Clock : in STD_LOGIC;
       R7_LED : out STD_LOGIC_VECTOR (7 downto 0);
       Zero_LED : out STD_LOGIC;
       Over_F_LED : out STD_LOGIC;
       Anode : out STD_LOGIC_VECTOR (3 downto 0);
       R7_7seg : out STD_LOGIC_VECTOR (6 downto 0));
end component;

signal reset : std_logic;
signal clk : std_logic := '0';
signal Zero_LED, Over_F_LED : std_logic;
signal R7_LED : STD_LOGIC_VECTOR (7 downto 0);
signal anode : STD_LOGIC_VECTOR (3 downto 0);
signal R7_7seg : STD_LOGIC_VECTOR (6 downto 0);

begin

    UUT: NanoProcessor_Improved
        port map(
        Reset    => reset,
        Clock    => clk,
        R7_LED   => R7_LED,
        Zero_LED    =>  Zero_LED,
        Over_F_LED => Over_F_LED,
        Anode    => anode,
        R7_7seg => R7_7seg);

        process
        begin
            clk <= not clk;
            wait for 1 ns;
        end process;

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

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


end Behavioral;
```

- <u>Timing Diagram for improved Nano Processor</u>


Untitled 3

Name	Value
reset	1
clk	1
Zero_LED	U
Over_F_LED	U
R7_LED[7:0]	UU
[7]	U
[6]	U
[5]	U
[4]	U
[3]	U
[2]	U
[1]	U
[0]	U
anode[3:0]	e
[3]	1
[2]	1
[1]	1
[0]	0
R7_7seg[6:0]	40
[6]	1
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	0

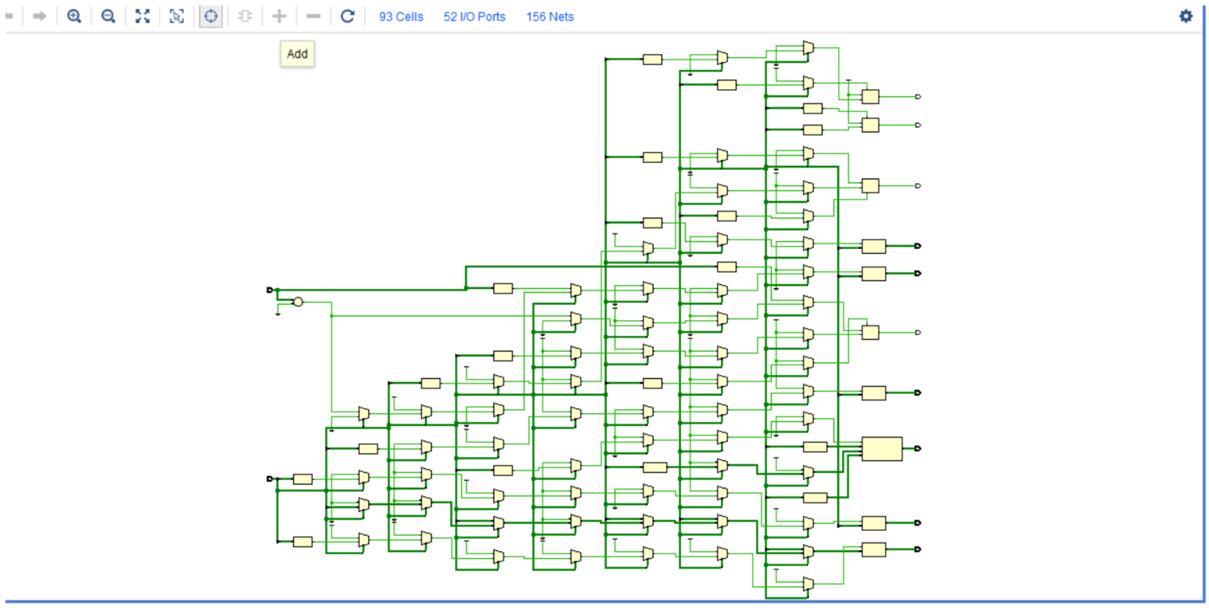
0.000 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
00 0a 92 9c a6 b0 ba c4 ce d8 e2 ec f6 00 0a 14 1e 28 32 3c 46 50 5a 64 6e 78 82 8c 96 23
40 40 40 40 40

2. Instruction_Decoder

- Design Source VHDL Code of improved Instruction_Decoder

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

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

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

entity Instruction_Decoder is
    Port ( Instruction : in STD_LOGIC_VECTOR (16 downto 0);
           Jump_reg_val : in STD_LOGIC_VECTOR (7 downto 0); --register value for jump check
           Reg_Sel_A: out STD_LOGIC_VECTOR (2 downto 0);    --Mux Select signal
           Reg_Sel_B: out STD_LOGIC_VECTOR (2 downto 0);    --Muc Selct signal
           Reg_En :out STD_LOGIC_VECTOR (2 downto 0);       --Register Bank write enable signal
           Flag_Reg_En : out STD_LOGIC;     -- Enable for ADD_SUB_Units and Multiplier Flag
           Flag_Reg_Input_Sel: out STD_LOGIC; -- 0 for ADD_SUB_unit ,1 for multiplier
           Jump_Address : out STD_LOGIC_VECTOR (3 downto 0);
           Jump_Flag : out STD_LOGIC;
           Add_Sub_Sel : out STD_LOGIC; -- 0 for add , 1 for subtract
           Load_Sel : out STD_LOGIC_VECTOR (1 downto 0);  -- 00 for immediarte value, 01 for add_sub
unit value
                                                         --10 for multiplier,  11 for Copy line
           Immediate_Value : out STD_LOGIC_VECTOR (7 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is

--signal Op_code : STD_LOGIC_VECTOR (1 downto 0);

begin

    Process (Instruction, Jump_reg_val)
    begin

        if Instruction (16 downto 14) = "000" then -- ADD
            Jump_Flag <= '0';
            Reg_Sel_A <= Instruction(13 downto 11);
            Reg_Sel_B <= Instruction(10 downto 8);
            Add_Sub_Sel <= '0';
            Load_Sel <= "01";
            Reg_En <= Instruction(13 downto 11);
            Flag_Reg_Input_Sel <= '0';
            Flag_Reg_En <='1';

        elsif Instruction (16 downto 14) = "001" then -- Neg
            Jump_Flag <= '0';
            Reg_Sel_A <= Instruction(13 downto 11);
            Reg_Sel_B <= Instruction(10 downto 8);
            Add_Sub_Sel <= '1';
            Load_Sel <= "01";
            Reg_En <= Instruction(13 downto 11);
            Flag_Reg_Input_Sel <= '0';
            Flag_Reg_En <='1';

        elsif Instruction (16 downto 14) = "010" then -- MOVI
            Jump_Flag <= '0';
            Load_Sel <= "00";
            Reg_En <= Instruction(13 downto 11);
            Flag_Reg_En <= '0';
            Immediate_value <= Instruction(7 downto 0);
        elsif Instruction (16 downto 14) = "011" then --JZR  (jump if Reg is zero)
            Reg_En <= "000";
            Flag_Reg_En <= '0';
            Reg_Sel_A <= Instruction(13 downto 11);
            if Jump_reg_val ="00000000" then
                Jump_Flag <= '1';
                Jump_Address <= Instruction (3 downto 0);
            else
                Jump_Flag <= '0';
            end if;

        elsif Instruction (16 downto 14) = "100" then  --MUL
            Jump_Flag <= '0';
            Reg_Sel_A <= Instruction(13 downto 11);
            Reg_Sel_B <= Instruction(10 downto 8);
            Load_Sel <= "10";
            Reg_En <= Instruction(13 downto 11);
            Flag_Reg_Input_Sel <= '1';
            Flag_Reg_En <='1';

        elsif Instruction (16 downto 14) = "101" then   --JUMP
            Reg_En <= "000";
            Flag_Reg_En <= '0';
            Jump_Flag <= '1';
            Jump_Address <= Instruction (3 downto 0);

        elsif Instruction (16 downto 14) = "110" then   --JNZ (jump if Reg not zero)
            Reg_En <= "000";
            Flag_Reg_En <= '0';
            Reg_Sel_A <= Instruction(13 downto 11);
            if Jump_reg_val /="00000000" then
                Jump_Flag <= '1';
                Jump_Address <= Instruction (3 downto 0);
            else
                Jump_Flag <= '0';
            end if;
        elsif Instruction (16 downto 14) = "111" then   --- COPY   (copy reg B to reg A)
            Jump_Flag <= '0';
            Flag_Reg_En <='0';
            Reg_Sel_B <= Instruction(10 downto 8);
            Load_Sel <= "11";
            Reg_En <= Instruction(13 downto 11);

        end if;

    end process;
```

- RTL Schematic Diagram of the improved Instruction_Decoder

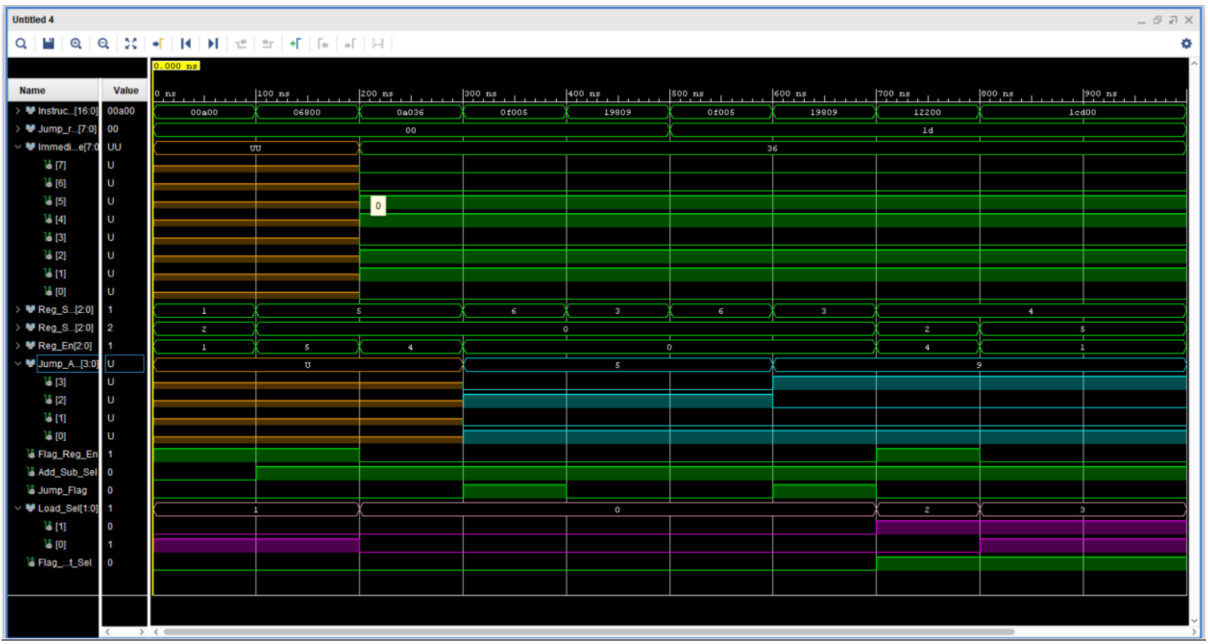


- Behavioral Simulation Code for improved Instruction_Decoder

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

architecture Behavioral of TB_Instruction_Decoder is

component Instruction_Decoder is
    Port ( Instruction : in STD_LOGIC_VECTOR (16 downto 0);
           Jump_reg_val : in STD_LOGIC_VECTOR (7 downto 0); --register value for jump check
           Reg_Sel_A: out STD_LOGIC_VECTOR (2 downto 0);    --Mux Select signal
           Reg_Sel_B: out STD_LOGIC_VECTOR (2 downto 0);    --Muc Selct signal
           Reg_En :out STD_LOGIC_VECTOR (2 downto 0);       --Register Bank write enable signal
           Flag_Reg_En : out STD_LOGIC;     -- Enable for ADD_SUB_Units and Multiplier Flag
           Flag_Reg_Input_Sel: out STD_LOGIC; -- 0 for ADD_SUB_unit ,1 for multiplier
           Jump_Address : out STD_LOGIC_VECTOR (3 downto 0);
           Jump_Flag : out STD_LOGIC;
           Add_Sub_Sel : out STD_LOGIC; -- 0 for add , 1 for subtract
           Load_Sel : out STD_LOGIC_VECTOR (1 downto 0);  -- 00 for immediarte value, 01 for add_sub
unit value
                                                        --10 for multiplier,  11 for Copy line
           Immediate_Value : out STD_LOGIC_VECTOR (7 downto 0));
end component;

signal Instruction : STD_LOGIC_VECTOR (16 downto 0);
signal Jump_reg_val ,Immediate_Value: STD_LOGIC_VECTOR (7 downto 0);
signal  Reg_Sel_A, Reg_Sel_B, Reg_En : STD_LOGIC_VECTOR (2 downto 0);
signal Jump_Address : STD_LOGIC_VECTOR (3 downto 0);
signal Flag_Reg_En,Add_Sub_Sel,Jump_Flag  : STD_LOGIC;
signal Load_Sel : STD_LOGIC_VECTOR (1 downto 0);
signal Flag_Reg_Input_Sel: STD_LOGIC;
```

```
begin

    UUT:Instruction_Decoder
        port map (
        Instruction=>Instruction,
        Jump_reg_val=>Jump_reg_val,
        Reg_Sel_A=> Reg_Sel_A,
        Reg_Sel_B=> Reg_Sel_B,
        Reg_En=> Reg_En,
        Flag_Reg_En=>Flag_Reg_En,
        Flag_Reg_Input_Sel => Flag_Reg_Input_Sel,
        Jump_Address=>Jump_Address,
        Jump_Flag=>Jump_Flag,
        Add_Sub_Sel=>Add_Sub_Sel,
        Load_Sel =>Load_Sel ,
        Immediate_Value=>Immediate_Value );

    process
    begin

    Jump_reg_val <= "00000000";

    Instruction <= "000001010000000000";
    wait for 100ns;

    Instruction <= "001101000000000000";
    wait for 100ns;

    Instruction <= "010100000000110110";
    wait for 100ns;

    Instruction <= "011110000000000101"; --JZR
    wait for 100ns;

    Instruction <= "110011000000001001"; --JNZ
    wait for 100ns;

    Jump_reg_val <= "00011101";

    Instruction <= "011110000000000101"; --JZR
    wait for 100ns;

    Instruction <= "110011000000001001"; --JNZ
    wait for 100ns;

    Instruction <= "100100010000000000"; --MUL
    wait for 100ns;

    Instruction <= "111001101000000000"; --COPY
    wait ;

    end process;

end Behavioral;
```

- <u>Timing Diagram for improved Instruction_Decoder</u>

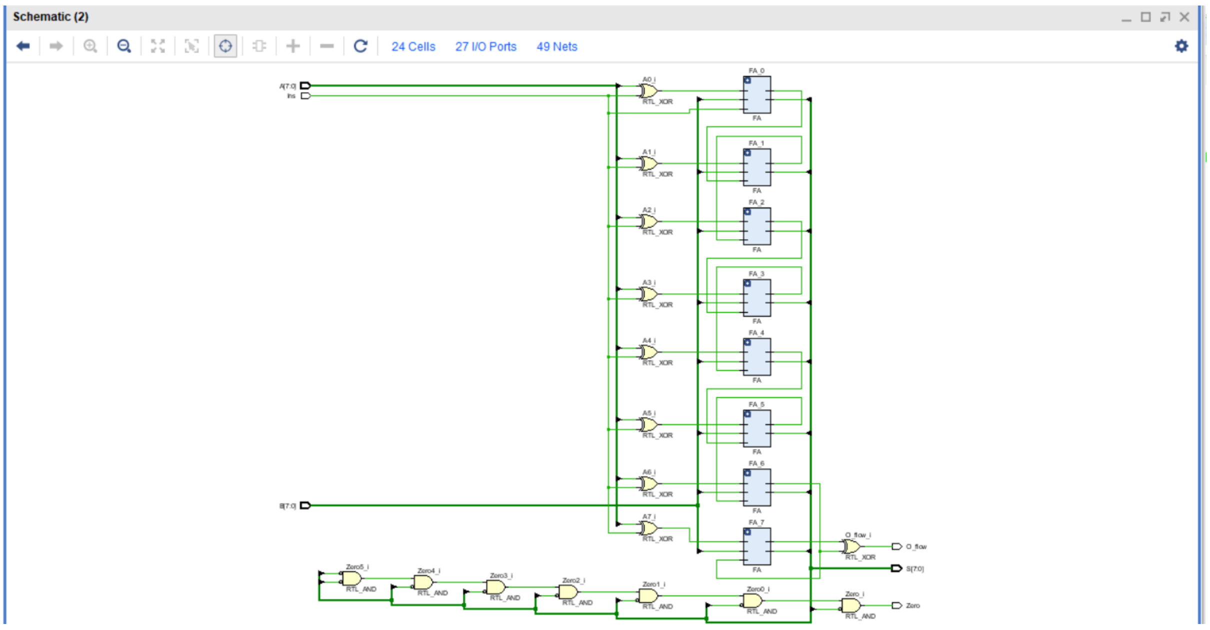


3. Add Sub 8bit

- Design Source VHDL Code of improved ADD_SUB_8bit

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

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

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

entity ADD_SUB_8bit is
    Port ( Ins : in STD_LOGIC; -- instruction (ADD =0, SUB = 1) (also works as C_in)
    A : in STD_LOGIC_VECTOR (7 DOWNTO 0);
    B : in STD_LOGIC_VECTOR (7 DOWNTO 0);
    S : out STD_LOGIC_VECTOR (7 DOWNTO 0);
    O_flow : out STD_LOGIC;
    Zero: out STD_LOGIC);
end ADD_SUB_8bit;

architecture Behavioral of ADD_SUB_8bit is

component FA is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C_in : in STD_LOGIC;
           S : out STD_LOGIC;
           C_out : out STD_LOGIC);
end component;

signal FA0_C, FA1_C, FA2_C, FA3_C, FA4_C, FA5_C, FA6_C, FA7_C : std_logic;
Signal A0, A1, A2, A3, A4, A5, A6, A7: std_logic;
Signal S0, S1, S2, S3, S4, S5, S6, S7 : std_logic;

begin

-- When Subtracting we will do B-A

    A0 <= A(0) XOR Ins;
    A1 <= A(1) XOR Ins;
    A2 <= A(2) XOR Ins;
    A3 <= A(3) XOR Ins;
    A4 <= A(4) XOR Ins;
    A5 <= A(5) XOR Ins;
    A6 <= A(6) XOR Ins;
    A7 <= A(7) XOR Ins;

    FA_0 : FA
        port map(
        A=>A0,
        B=>B(0),
        C_in=> Ins,
        S=> S0,
        C_out => FA0_C);

    FA_1 : FA
        port map(
        A=>A1,
        B=>B(1),
        C_in=> FA0_C,
        S=> S1,
        C_out => FA1_C);
    FA_2 : FA
        port map(
        A=>A2,
        B=>B(2),
        C_in=> FA1_C,
        S=> S2,
        C_out => FA2_C);

    FA_3 : FA
        port map(
        A=>A3,
        B=>B(3),
        C_in=> FA2_C,
        S=> S3,
        C_out => FA3_C);

    FA_4 : FA
        port map(
        A=>A4,
        B=>B(4),
        C_in=> FA3_C,
        S=> S4,
        C_out => FA4_C);

    FA_5 : FA
        port map(
        A=>A5,
        B=>B(5),
        C_in=> FA4_C,
        S=> S5,
        C_out => FA5_C);

    FA_6 : FA
        port map(
        A=>A6,
        B=>B(6),
        C_in=> FA5_C,
        S=> S6,
        C_out => FA6_C);

    FA_7 : FA
        port map(
        A=>A7,
        B=>B(7),
        C_in=> FA6_C,
        S=> S7,
        C_out => FA7_C);

S <= S7 & S6 & S5 & S4 & S3 & S2 & S1 & S0;
Zero <= ( (NOT S0) AND (NOT S1) AND (NOT S2) AND (NOT S3)AND (NOT S4) AND (NOT S5) AND (NOT S6) AND
(NOT S7) );
O_Flow <= FA7_C XOR FA6_C;

end Behavioral;
```

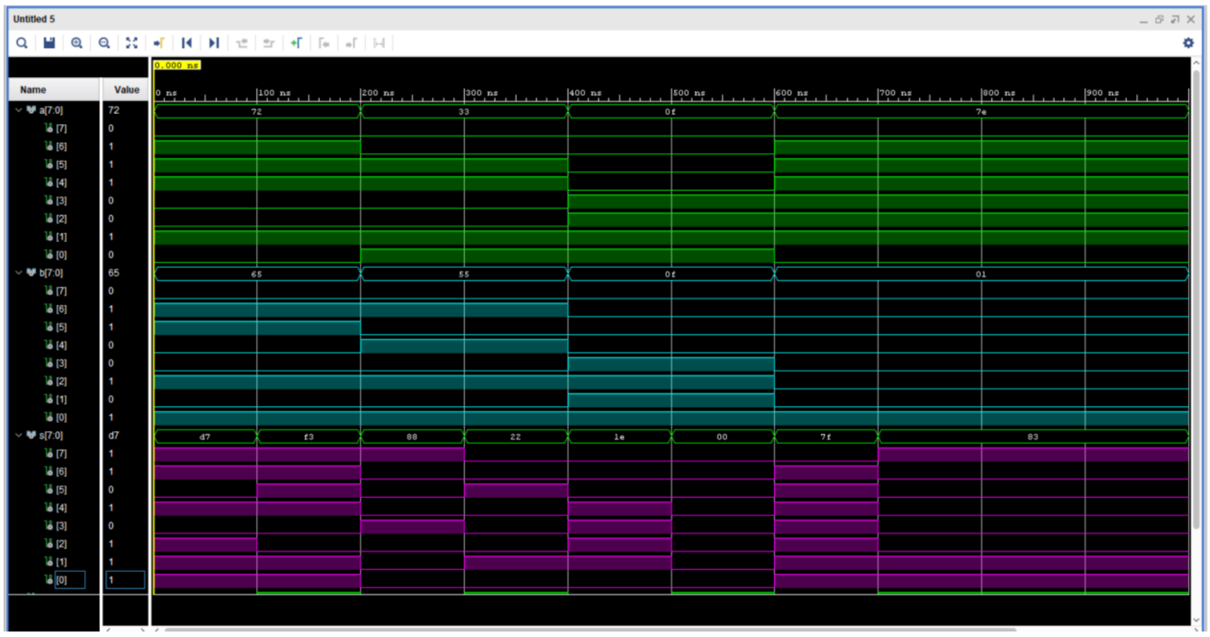
- RTL Schematic Diagram of the improved ADD_SUB_8bit



- Behavioral Simulation Code for improved ADD_SUB_8bit

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

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

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

entity TB_ADD_SUB_8bit is
--  Port ( );
end TB_ADD_SUB_8bit;

architecture Behavioral of TB_ADD_SUB_8bit is

component ADD_SUB_8bit is
    Port ( Ins : in STD_LOGIC; -- instruction (ADD =0, SUB = 1) (also works as C_in)
        A : in STD_LOGIC_VECTOR (7 DOWNTO 0);
        B : in STD_LOGIC_VECTOR (7 DOWNTO 0);
        S : out STD_LOGIC_VECTOR (7 DOWNTO 0);
        O_flow : out STD_LOGIC;
        Zero: out STD_LOGIC);
end component;

signal  a,b,s : std_logic_vector (7 downto 0);
signal ins, O_f, z :std_logic;

begin

    UTT: ADD_SUB_8bit
        port map(
        Ins => ins,
        A => a,
        B => b,
        S => s,
        O_flow => O_f,
        Zero => z );

    process
    begin
        a <= "01110010";
        b <= "01100101";
        ins <= '0';
        wait for 100ns;

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

        a <= "00110011";
        b <= "01010101";
        ins <= '0';
        wait for 100ns;

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

        a <= "00001111";
        b <= "00001111";
        ins <= '0';
        wait for 100ns;

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

        a<="01111110";
        b<="00000001";
        ins <= '0';
        wait for 100ns;

        ins <='1';

        wait;

    end process;
end Behavioral;
```

- <u>Timing Diagram for improved ADD_SUB_8bit</u>

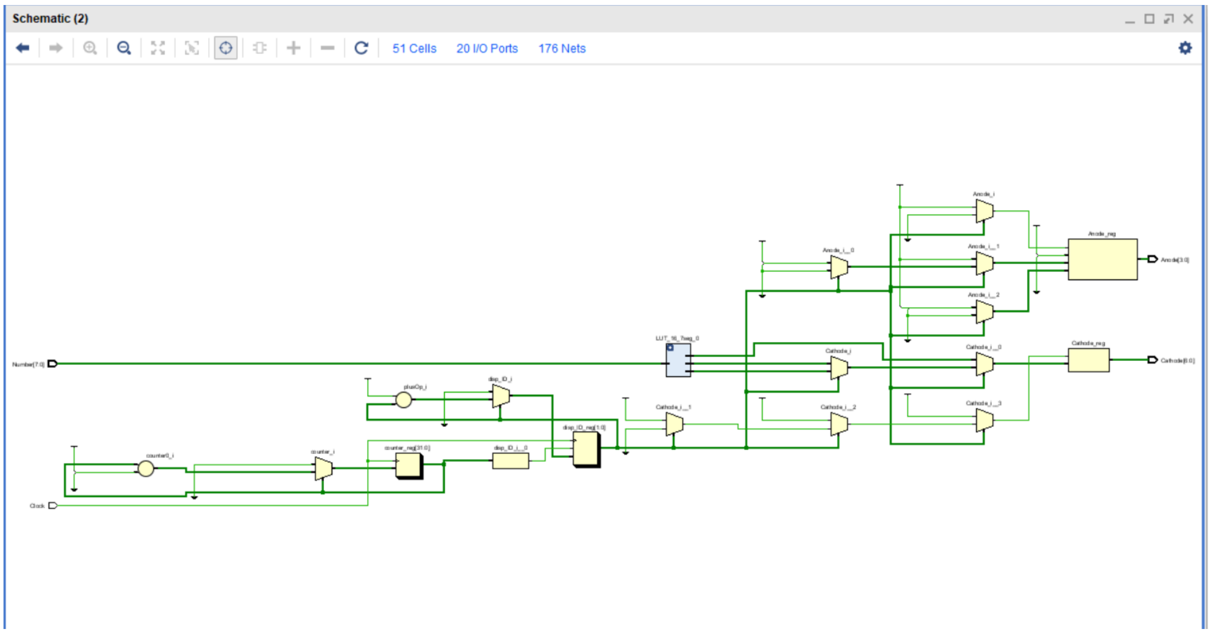


4. Display 7 segment

- Design Source VHDL Code of improved Display_7seg_out

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


entity Display_7seg_out is
    Port ( Number : in STD_LOGIC_VECTOR (7 downto 0);
           Clock : in STD_LOGIC;
           Cathode : out STD_LOGIC_VECTOR (6 downto 0);
           Anode : out STD_LOGIC_VECTOR (3 downto 0));
end Display_7seg_out;

architecture Behavioral of Display_7seg_out is

component LUT_16_7seg is
    Port ( value : in STD_LOGIC_VECTOR (7 downto 0);
           data_dis_0 : out STD_LOGIC_VECTOR (6 downto 0);
           data_dis_1 : out STD_LOGIC_VECTOR (6 downto 0);
           data_dis_2 : out STD_LOGIC_VECTOR (6 downto 0));
end component;


signal counter : integer:= 0;
signal disp_ID : unsigned(1 downto 0):= "00";
signal data_dis_0, data_dis_1, data_dis_2 : STD_LOGIC_VECTOR (6 downto 0);

begin

    process(Clock)
    begin
        if rising_edge(Clock) then
            if counter = 4 then  -- ~333 µs at 100 MHz
                counter <= 0;

                -- Cycle digit index 0 -> 1 -> 2 -> 0 ...
                if disp_ID = "10" then
                    disp_ID <= "00";
                else
                    disp_ID <= disp_ID + 1;
                end if;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;


    LUT_16_7seg_0 : LUT_16_7seg
    port map(
        value   => Number,
        data_dis_0  => data_dis_0,
        data_dis_1  => data_dis_1,
        data_dis_2  => data_dis_2);

    process (data_dis_0, data_dis_1, data_dis_2, disp_ID)
    begin
        if disp_ID ="00" then
            Anode <= "1110";
            Cathode <= data_dis_0;

        elsif disp_ID ="01" then
            Anode <= "1101";
            Cathode <= data_dis_1;

        elsif disp_ID ="10" then
            Anode <= "1011";
            Cathode <= data_dis_2;
        end if;
    end process;

end Behavioral;
```

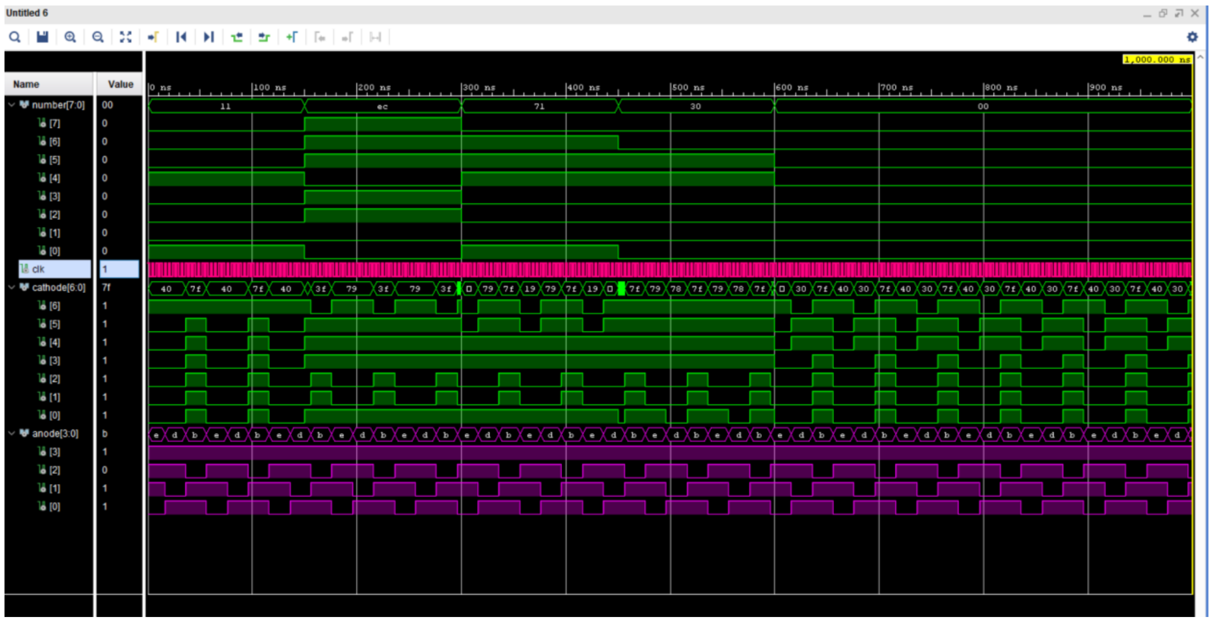
- <u>RTL Schematic Diagram of the improved Display_7seg_out</u>



- <u>Behavioral Simulation Code for improved Display_7seg_out</u>

```
entity TB_Display_7seg_out is
--  Port ( );
end TB_Display_7seg_out;

architecture Behavioral of TB_Display_7seg_out is

component Display_7seg_out is
    Port ( Number : in STD_LOGIC_VECTOR (7 downto 0);
           Clock : in STD_LOGIC;
           Cathode : out STD_LOGIC_VECTOR (6 downto 0);
           Anode : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal number : STD_LOGIC_VECTOR (7 downto 0);
signal clk : STD_LOGIC := '0';
signal cathode: STD_LOGIC_VECTOR (6 downto 0);
signal anode : STD_LOGIC_VECTOR (3 downto 0);

begin

    process
    begin
        clk <= not clk;
        wait for 2 ns;
    end process;

    UUT:Display_7seg_out
    port map(
        Number  => number,
        Clock   => clk,
        Cathode => cathode,
        Anode    => anode);

    process
    begin
        number <= "00010001";
        wait for 150ns;

        number <= "11101100";
        wait for 150ns;

        number <= "01110001";
        wait for 150ns;

        number <= "00110000";
        wait for 150ns;

        number <= "00000000";
--        wait for 150ns;

--        number <= "10011001";
--        wait for 150ns;

--        number <= "11111111";
        wait;
    end process;

end Behavioral;
```

- <u>Timing Diagram for improved Display_7seg_out</u>



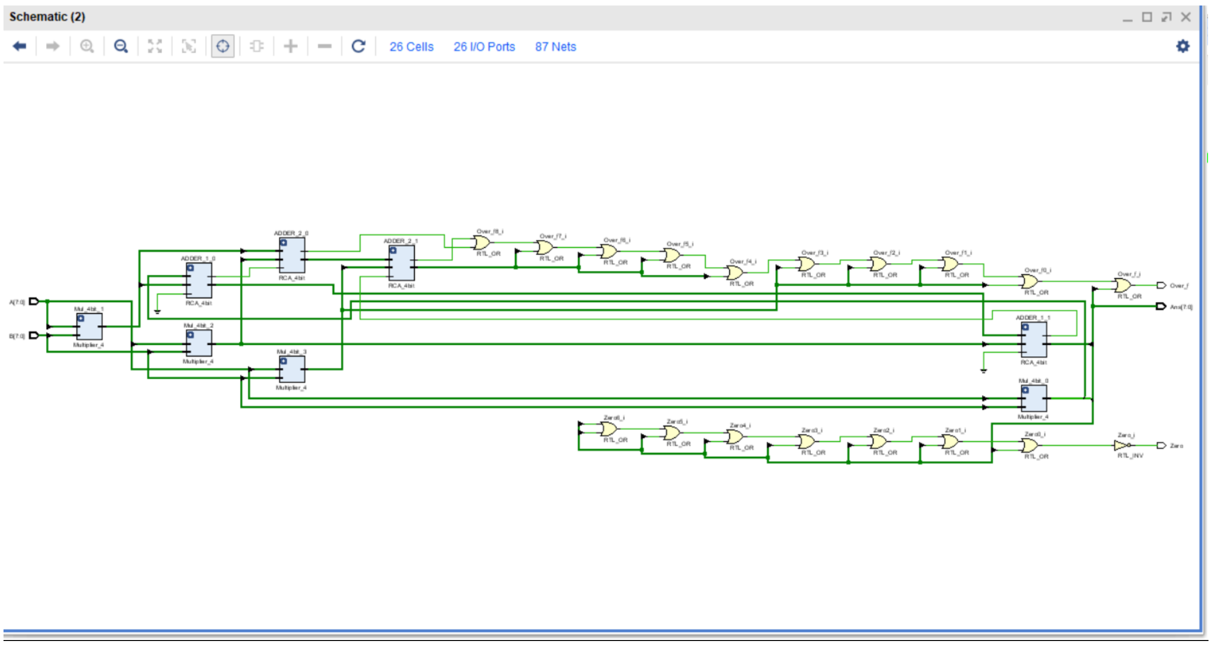
5. Multiplier_8bit

- Design Source VHDL Code of improved Multiplier_8bit

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

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

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

entity Multiplier_8bit is
    Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           Ans : out STD_LOGIC_VECTOR (7 downto 0);
           Over_f : out STD_LOGIC;
           Zero : out STD_LOGIC);
end Multiplier_8bit;

architecture Behavioral of Multiplier_8bit is

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));
end component;

component RCA_4bit is
    Port (
    A : in STD_LOGIC_VECTOR (3 DOWNTO 0);
    B : in STD_LOGIC_VECTOR (3 DOWNTO 0);
    S : out STD_LOGIC_VECTOR (3 DOWNTO 0);
    C_in :in STD_LOGIC;
    C_out: out STD_LOGIC);
end component;

signal Mul_0_out,Mul_1_out,Mul_2_out,Mul_3_out : std_logic_vector(7 downto 0);
signal add_1_0_out, add_1_1_out, add_2_0_out, add_2_1_out :std_logic_vector(3 downto 0);
signal add_1_0_C, add_1_1_C, add_2_0_C, add_2_1_C : std_logic;
signal Answer :std_logic_vector(7 downto 0);

begin

    Mul_4bit_0:Multiplier_4
    port map(
        A   => A(3 downto 0),
        B   => B(3 downto 0),
        Y   => Mul_0_out);    --- mul_0_out (3 downto zero is Ans( 3 downto 0)

    Mul_4bit_1:Multiplier_4
    port map(
        A   => A(7 downto 4),
        B   => B(3 downto 0),
        Y   => Mul_1_out);

    Mul_4bit_2:Multiplier_4
    port map(
        A   => A(3 downto 0),
        B   => B(7 downto 4),
        Y   => Mul_2_out);

    Mul_4bit_3:Multiplier_4
        port map(
        A   => A(7 downto 4),
        B   => B(7 downto 4),
        Y   => Mul_3_out);
```

```
--- Combining answers

    ADDER_1_0:RCA_4bit
    port map(
        A    => Mul_0_out(7 downto 4),
        B    => Mul_1_out(3 downto 0),
        S    => add_1_0_out,
        C_in => '0',
        C_out => add_1_0_C );

    ADDER_1_1:RCA_4bit
    port map(
        A    => add_1_0_out,
        B    => Mul_2_out(3 downto 0),
        S    => add_1_1_out,         --add_1_1_out is Ans( 7 downto 4)
        C_in => '0',
        C_out => add_1_1_C );

    ADDER_2_0:RCA_4bit
    port map(
        A    => Mul_1_out(7 downto 4),
        B    => Mul_2_out(7 downto 4),
        S    => add_2_0_out,
        C_in => add_1_0_C,
        C_out => add_2_0_C );

    ADDER_2_1:RCA_4bit
    port map(
        A    => add_2_0_out,
        B    => Mul_3_out(3 downto 0),
        S    => add_2_1_out,
        C_in => add_1_1_C,
        C_out => add_2_1_C );

    Answer <= add_1_1_out & Mul_0_out (3 downto 0);


    --outputs

    Ans <= Answer;

    Over_f <= add_2_1_C or add_2_0_C or add_2_1_out(0) or add_2_1_out(1) or  add_2_1_out(2) or
                add_2_1_out(3) or Mul_3_out(7) or Mul_3_out(6) or Mul_3_out(5) or Mul_3_out(4) or
add_1_1_out(3);

    Zero <= not( Answer(0) or Answer(1) or Answer(2) or Answer(3) or Answer(4) or Answer(5) or
Answer(6) or Answer(7));

end Behavioral;
```

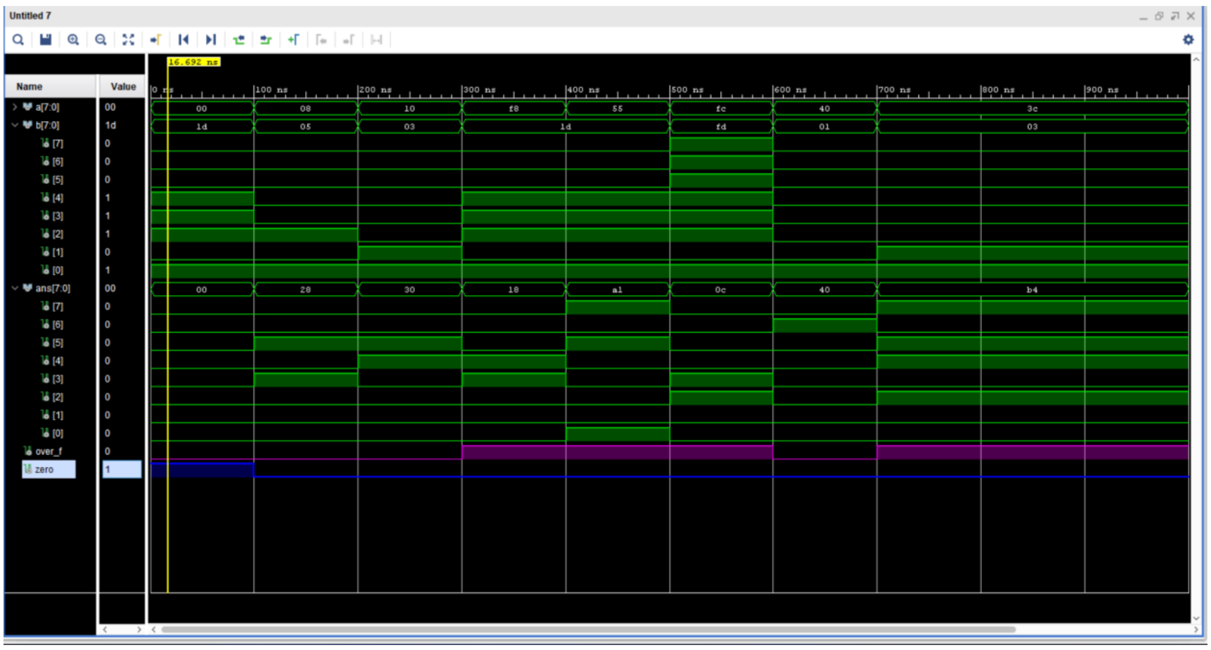
- <u>RTL Schematic Diagram of the improved Multiplier_8bit</u>



- <u>Behavioral Simulation Code for improved Multiplier_8bit</u>

```
entity TB_Multiplier_8bit is
--  Port ( );
end TB_Multiplier_8bit;

architecture Behavioral of TB_Multiplier_8bit is

component Multiplier_8bit is
    Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           Ans : out STD_LOGIC_VECTOR (7 downto 0);
           Over_f : out STD_LOGIC;
           Zero : out STD_LOGIC);
end component;

signal a, b, ans : STD_LOGIC_VECTOR (7 downto 0);
signal over_f ,zero : STD_LOGIC;

begin

    UUT :Multiplier_8bit
    port map(
        A   => a,
        B   => b,
        Ans => ans,
        Over_f  => over_f,
        Zero    => zero);

    process
    begin

    a <= "00000000";
    b <= "00011101";
    wait for 100ns;

    a <= "00001000";
    b <= "00000101";
    wait for 100ns;

    a <= "00010000";
    b <= "00000011";
    wait for 100ns;

    a <= "11111000";
    b <= "00011101";
    wait for 100ns;

    a <= "01010101";
    b <= "00011101";
    wait for 100ns;

    a <= "11111100";
    b <= "11111101";
    wait for 100ns;

    a <= "01000000";
    b <= "00000001";
    wait for 100ns;

    a <= "00111100";
    b <= "00000011";
    wait;

    end process;

end Behavioral;
```

- <u>Timing Diagram for improved Multiplier_8bit</u>



6. Program_ROM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;


entity Program_ROM is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (16 downto 0));
end Program_ROM;

architecture Behavioral of Program_ROM is

type rom_type is array (0 to 15) of std_logic_vector(16 downto 0);

    signal Program_ROM : rom_type := (
        "01011100000001010", -- MOVI R7,10
        "11101011100000000", -- COPY R2,R7
        "01000100000001100", -- MOVI R1,12
        "10011100100000000", -- MUL R7,R1
        "00111100000000000", -- NEG R7
        "00011101000000000", -- ADD R7,R2
        "01111100000001000", -- JZR R7,8
        "10100000000000101", -- JUMP 5
        "01001100000000001", -- MOVI R3,1
        "00101100000000000", -- NEG R3
        "01010000000001111", -- MOVI R4,15
        "00011101000000000", -- ADD R7,R2
        "00010001100000000", -- ADD R4,R3
        "11010000000001011", -- JNZ R4,11
        "01011100000100011", -- MOVI R7,35
        "01100000000001110" -- JZR R0,14
        );

begin
    data <= Program_ROM(to_integer(unsigned(address)));


end Behavioral;
```

7. LUT_16_7seg

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

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

entity LUT_16_7seg is
    Port ( value : in STD_LOGIC_VECTOR (7 downto 0);
           data_dis_0 : out STD_LOGIC_VECTOR (6 downto 0);
```

```vhdl
           data_dis_1 : out STD_LOGIC_VECTOR (6 downto 0);
           data_dis_2 : out STD_LOGIC_VECTOR (6 downto 0));
end LUT_16_7seg;

architecture Behavioral of LUT_16_7seg is

type rom_type is array (0 to 15) of std_logic_vector(6 downto 0);

    signal sevenSegment_ROM : rom_type := (
        "1000000", -- 0
        "1111001", --1
        "0100100",
        "0110000",
        "0011001", --4
        "0010010",
        "0000010",--6
        "1111000",
        "0000000",
        "0010000",--9
        "0001000", -- a
        "0000011",
        "1000110",
        "0100001",
        "0000110",
        "0001110" -- f
        );

signal abs_value : unsigned(7 downto 0);

begin

    process (value)
    begin
        abs_value <= unsigned(abs(signed(value)));

        data_dis_0 <= sevenSegment_ROM(to_integer(abs_value(3 downto 0)));
        data_dis_1 <= sevenSegment_ROM(to_integer(abs_value(7 downto 4)));

        if value(7)='0' then
            data_dis_2 <= "1111111";
        else
            data_dis_2 <= "0111111";
        end if;
    end process;

end Behavioral;
```

8. Constrain File

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



## LEDs
set_property PACKAGE_PIN U16 [get_ports {R7_LED[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[0]}]
set_property PACKAGE_PIN E19 [get_ports {R7_LED[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[1]}]
set_property PACKAGE_PIN U19 [get_ports {R7_LED[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[2]}]
set_property PACKAGE_PIN V19 [get_ports {R7_LED[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[3]}]
set_property PACKAGE_PIN W18 [get_ports {R7_LED[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[4]}]
set_property PACKAGE_PIN U15 [get_ports {R7_LED[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[5]}]
set_property PACKAGE_PIN U14 [get_ports {R7_LED[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[6]}]
set_property PACKAGE_PIN V14 [get_ports {R7_LED[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_LED[7]}]
set_property PACKAGE_PIN P1 [get_ports {Zero_LED}]
        set_property IOSTANDARD LVCMOS33 [get_ports {Zero_LED}]
set_property PACKAGE_PIN L1 [get_ports {Over_F_LED}]
        set_property IOSTANDARD LVCMOS33 [get_ports {Over_F_LED}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {R7_7seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {R7_7seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {R7_7seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {R7_7seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {R7_7seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {R7_7seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {R7_7seg[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {R7_7seg[6]}]


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


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

♦ Contributions

Member	Individual Contribution	Worked Time
Jayasinghe J.D.D.S	• Instruction Decoder • Final Assembly Nano Processor & Simulation • Multiplier & Simulation, Instruction Decoder & Simulation, Final assembly & simulation, 7 segment Display in Improved version.	4 hours 4 hours 5 hours
Rafi M.A.A	• ADD SUB Unit & Program ROM in basic version • Improved ADD SUB Unit • Improved Register Bank • 3 bit Adder	3 hours 4 hour 2 hours
Kirindage S.S	• MUX 2 Way 4 bit • MUX 2 Way 3 bit • MUX 8 Way 4 bit • Simulation for Instruction Decoder in Basic Version • Finalizing Lab Report	1 hour 1 hour 2 hours 2 hours 2 hours
Kariyapperuma K.M.N.S	• Simulation for Program Counter in Basic Version • Simulation Register Bank • Constraint files Basys3.xdc • Finalizing Lab Report • Simulation for Instruction Decoder in improved Version	2 hours 2 hours 1 hour 1 hour 2 hours

♦ Conclusion

The nano processor design project was a culmination of everything we learned through our module and lab sessions. It taught us how to design individual components for specific functions and integrate them into a complete system to achieve a common goal. As a group project, it was a valuable experience in collaboration—sharing responsibilities, working together in harmony, and tackling challenges as a team.

We faced many obstacles, especially when combining components developed by different team members. Integrating these parts into a unified design required careful coordination and problem-solving. Throughout the journey, we embraced diverse ideas and made thoughtful decisions to refine our approach. Ultimately, we embedded our machine code into ROM, bringing our nano processor to life—a meaningful outcome shaped by teamwork, creativity, and perseverance.