Nanoprocessor Design Competition - Lab Report

CS1050 - Computer Organization and Digital Design

Team members - Group 26

1. Mahanama K.J.C 230387V
2. Udayanga D.M.S 230651C
3. Nawarathna T.P 230427V
4. Wickramaarachchi W.A.A.U 230702K

Introduction

This lab exercise was to design a 4-bit nanoprocessor to execute four instructions (**MOVI, ADD, NEG, JZR**).

We designed the following components like a 4-bit add/subtract unit, 3-bit program counter, multiplexers, register bank, and instruction decoder using VHDL. The nanoprocessor was utilized on the BASYS 3 board to run an assembly program.

Lab Task

The task was to build the 4-bit nanoprocessor to execute four instructions. Components included a 4-bit add/subtract unit, 3-bit program counter, multiplexers, register bank, program rom and instruction decoder.

An assembly program to sum integers 1 to 3 was converted to machine code and hardcoded into ROM.

The design was simulated, tested on the BASYS 3 board with R7 output on LEDs and a 7-segment display, and demonstrated.

Table of Content

Work Division Among Team

- **Mahanama K.J.C** -Assembled and interconnected components, ensuring seamless communication. Designed and implemented two versions of the nanoprocessor, an optimized model, and an extended version featuring advanced functionality. (16 hours)
- **Nawarathna T.P**- Designed VHDL source code for components including the Add/Sub Unit, RCA, decoders, and multiplexers, ensuring adherence to performance and functionality requirements.- (10 Hours)
- **Wikramaarachchi W.A.A.U** - Developed VHDL source code for essential components such as the Register Bank, ROM, Instruction Decoder, and Program Counter, ensuring seamless integration into the nanoprocessor. (12 Hours)
- **Udayanga D.M.S**- Developed comprehensive test bench code for individual components, conducted testing of subcomponents to verify correctness and performance.- (15 Hours)

Testing Methodology

Unit Testing: Each component was tested individually by applying inputs and verifying outputs to ensure correct functionality.

Integration Testing: After assembling the components into a full system, integration tests were performed to verify proper communication and overall system behavior.

Assembly Program Testing: The assembly code was tested extensively to ensure it exercised all processor features and produced correct outputs for various inputs.

Simulation Testing: Simulations were run to detect and fix issues before deploying to hardware, validating hardware logic and program behavior.

BASYS 3 Board Testing: The complete system was deployed to the BASYS 3 board for real world testing, confirming final functionality and performance.

Challenges Faced During The Project

Unpredictable Behavior During Testing on Basys3 Board

While testing our design on the Basys3 board, we observed inconsistent behavior—sometimes the output remained at 0, and other times it jumped directly to the final result without intermediate states. To resolve this, we configured a slower clock to better observe transitions and carefully mapped the input/output constraints to match the board specifications, which stabilized the behavior.

Synchronizing the Project Across Team Members

As different team members were responsible for developing various components, inconsistencies in variable names and input/output definitions led to difficulties during integration. To address this, we used GitHub for version control and collaboration, which helped us manage changes efficiently, track progress, and synchronize our work across the team.

Clock Configuration for LEDs

We attempted to use all four LEDs with a clock signal, but this resulted in unpredictable outcomes. To resolve the issue, we conducted research and referred to the Basys3 board manual. Based on the insights gained, we adjusted the clock configuration, which led to the correct and stable functioning of the LEDs.

Unexpected Behavior with Division Instruction

While implementing division, we initially used a single divide instruction to obtain both the quotient and remainder. However, this led to unpredictable results, as the instruction also affected other operations. To fix this, we modified the implementation to explicitly produce the quotient and remainder separately, which ensured accurate and consistent outputs.

Basic Nanoprocessor Design

The 4-bit nanoprocessor was designed following the high-level diagram provided, integrating the following components:

1. **4-bit add/subtract unit** - capable of adding and subtracting numbers represented in 2's complement.
2. **3-bit adder** - capable of incrementing the program counter
3. **3-bit program counter** - capable of storing the memory location of the next instruction to be executed
4. **2-way 3-bit Multiplexer** - take in 2 inputs each with 3 bits and gives output as 3 bits
5. **2-way 4-bit Multiplexer** - take in 2 inputs each with 3 bits and gives output as 4 bits
6. **8-way 4-bit Multiplexer** - take in 8 inputs each with 3 bits and gives output as 4 bits
7. **Register Bank** - provide a storage mechanism for temporary data. Consists 8, 4-bit Registers.
8. **ProgramROM** -Stores the assembly program.
9. **Instruction Decoder**- capable of activating necessary components based on the instructions we are planning to execute

Reset Mechanism: The nanoprocessor is reset by pressing and holding the btnC button, initializing the system.

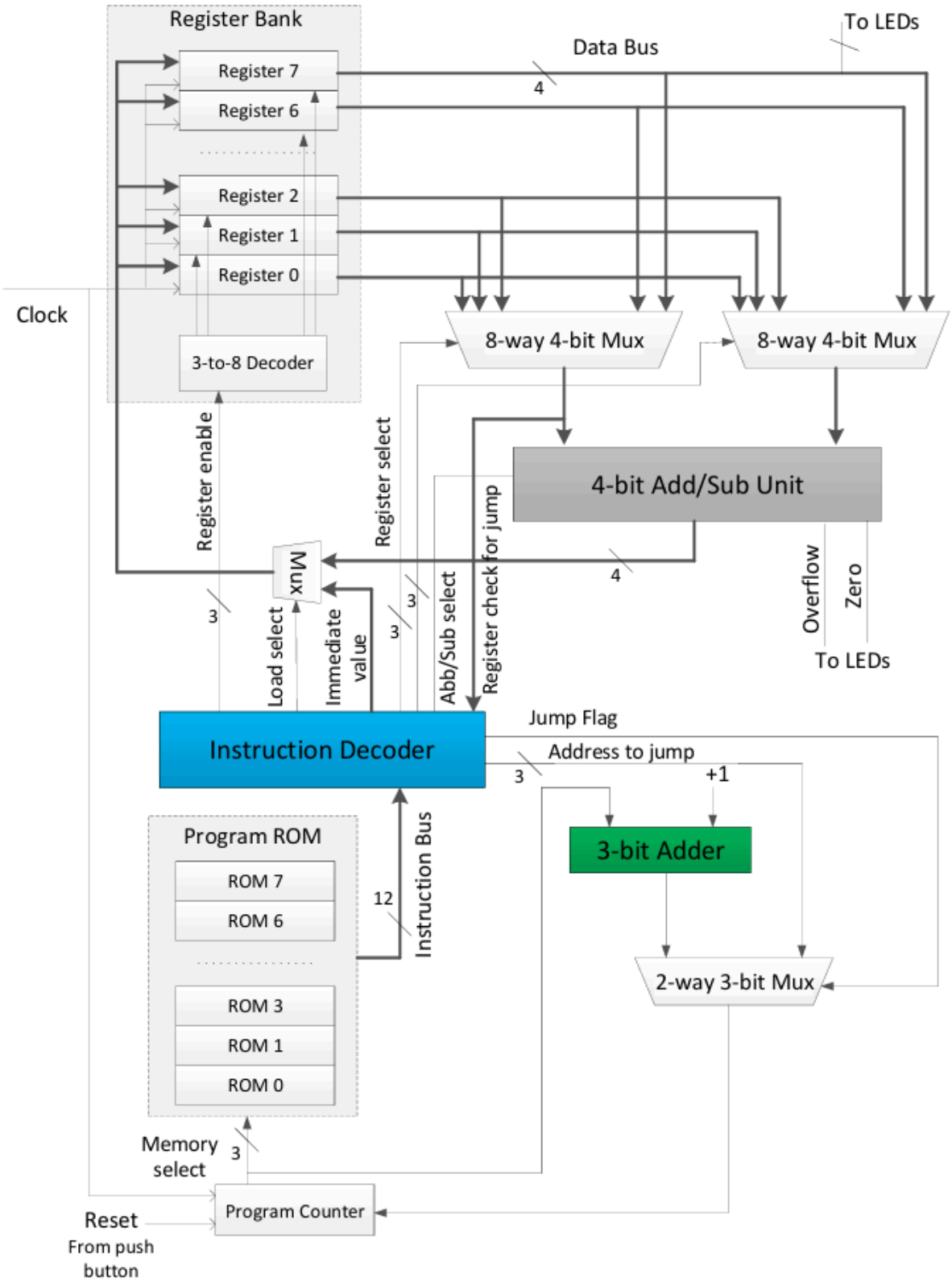
Clock Speed: The internal clock was reduced from 100 MHz to 1 MHz using a slow clock, making the calculation process visible to the naked eye.

LED Outputs:

LED0–LED3: Display the 4-bit signed output of the R7 register in two's complement.
LED14: Overflow Flag
LED15: Zero Flag

7-Segment Display: The rightmost segment shows the magnitude of the R7 register output.

High Level Diagram



Basic Nanoprocessor Implemented Design Utilization Report

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (106)	BUFGCTRL (32)
MainProgram	34	53	22	34	8	19	1
Nanoprocessor_0 (Na...	23	19	9	23	5	0	0
Program_Counter_...	10	3	6	10	2	0	0
Register_Bank_0 (...	13	16	7	13	1	0	0
Slow_Clk_0 (Slow_Clk)	11	34	15	11	3	0	0

Instruction Format

Instruction	Description	Format (12-bit instruction)
MOVI R, *d*	Move immediate value *d* to register R, i.e., $R \leftarrow d$ $R \in [0, 7]$, $d \in [0, 15]$	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 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 Program and Machine Code

10 001 000 0001 -- MOVI R1, 1 -- Move immediate value 1 into register R1
10 010 000 0010 -- MOVI R2, 2 -- Move immediate value 2 into register R2
10 011 000 0011 -- MOVI R3, 3 -- Move immediate value 3 into register R3
10 111 000 0000 -- MOVI R7, 0 -- Move immediate value 0 into register R7
00 111 001 0000 -- ADD R7, R1 -- Add the values in R1,R7 and store in R7
00 111 010 0000 -- ADD R7, R2 -- Add the values in R2,R7 and store in R7
00 111 011 0000 -- ADD R7, R3 -- Add the values in R3,R7 and store in R7
11 000 0000 111 -- JZR, R0 -- Jump to instruction 7 if R0 is zero

VHDL Codes (Design Source Files)

- ## Slow Clock (Slow_Clk.vhd)

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

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

architecture Behavioral of Slow_Clk is

SIGNAL count : integer :=1;
SIGNAL Clk_status : STD_LOGIC :='0';

begin

    process (Clk_in) begin
    if (rising_edge(Clk_in)) then
    count <= count +1 ;
        if (count = 50000000) then
             Clk_status <= NOT (Clk_status);
             Clk_out <= Clk_status ;
             count <= 1;
        end if;
    end if;
    end process;

end Behavior;
```

- ## D Flip-Flop(D_FF.vhd)

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

entity D_FF is
    Port ( D : in STD_LOGIC;
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC;
           Qbar : out STD_LOGIC);
end D_FF;

architecture Behavioral of D_FF is
```

```
begin
    process (Clk) begin
        if (rising_edge(Clk)) then
            if Res = '1' then
                Q <= '0';
                Qbar <= '1';
            else
                Q <= D;
                Qbar <= not D;
            end if;
        end if;
    end process;
end Behavioral;
```

- **Program Counter (Program_Counter.vhd)**

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

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

architecture Behavioral of Program_Counter is

component D_FF
Port     ( D : in STD_LOGIC;
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC;
           Qbar : out STD_LOGIC);
end component;

begin
   D_FF_0 : D_FF
       port map (
           D => D(0),
           Res => Res,
           Clk => Clk,
           Q => Q(0));

   D_FF_1 : D_FF
       port map (
           D => D(1),
```

```
            Res => Res,
            Clk => Clk,
            Q => Q(1));

    D_FF_2 : D_FF
        port map (
            D => D(2),
            Res => Res,
            Clk => Clk,
            Q => Q(2));

end Behavioral;
```

- **Program ROM (Program_ROM.vhd)**

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


entity Program_ROM is
    Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
           instructions : 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 ROM : rom_type := (
                "100010000001", -- MOVI R1, 1
                "100100000010", -- MOVI R2, 2
                "100110000011", -- MOVI R3, 3
                "101110000000", -- MOVI R7, 0
                "001110010000", -- ADD R7, R1
                "001110100000", -- ADD R7, R2
                "001110110000", -- ADD R7, R3
                "110000000111" -- JZR, R0

    );

begin

instructions <= ROM(to_integer(unsigned(address)));

end Behavioral;
```

- **Instruction Decoder (Instruction_Decoder.vhd)**

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

entity Instruction_Decoder is
    Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
           Reg_Check_Jump : in STD_LOGIC_VECTOR (3 downto 0);
           Add_Sub_Sel : out STD_LOGIC;
           RegA: out STD_LOGIC_VECTOR (2 downto 0);
           RegB : out STD_LOGIC_VECTOR (2 downto 0);
           Immediate_Value : out STD_LOGIC_VECTOR (3 downto 0);
           Load_Sel : out STD_LOGIC;
           Reg_EN : out STD_LOGIC_VECTOR (2 downto 0);
           Jump_Flag : out STD_LOGIC;
           Jump_Address : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is

signal Operator : std_logic_vector (1 downto 0);

begin
    Operator <= Instruction(11 downto 10);

    process (Operator, Instruction, Reg_Check_Jump) begin

        Add_Sub_Sel <= '0';
        Jump_Flag <= '0';
        Load_Sel <= '0';
        RegA <= "000";
        RegB <= "000";
        Immediate_Value <= "0000";
        Reg_EN <= "000";
        Jump_Address <= "000";

        if Operator = "00" then --ADD
            RegA <= Instruction(9 downto 7);
            RegB <= Instruction(6 downto 4);
            Reg_EN <= Instruction(9 downto 7);

        elsif Operator = "01" then --NEG
            Reg_EN <= Instruction(9 downto 7);
            RegB <= Instruction(9 downto 7);
            Add_Sub_Sel <= '1';

        elsif Operator = "10" then --MOVI
            Immediate_Value <= Instruction(3 downto 0);
            Reg_EN <= Instruction(9 downto 7);
            Load_Sel <= '1';

        elsif Operator = "11" then --JZR
```

```
            Jump_Address <= Instruction(2 downto 0);
            RegA <= Instruction(9 downto 7);
            if Reg_Check_Jump = "0000" then
                Jump_Flag <= '1';
            end if;
        end if;

    end process;

end Behavioral;
```

- **2-way 4-bit Multiplexer (Mux_2way_4bit.vhd)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Mux_2way_4bit is
    Port ( I0, I1 : in STD_LOGIC_VECTOR(3 downto 0);
           Sel : in STD_LOGIC;
           RegOut : out STD_LOGIC_VECTOR(3 downto 0));
end Mux_2way_4bit;

architecture Behavioral of Mux_2way_4bit is

begin
    process(I0, I1, Sel)
    begin
        if Sel = '0' then
            RegOut <= I0;
        else
            RegOut <= I1;
        end if;
    end process;


end Behavioral;
```

- **2-way 3-bit Multiplexer (Mux_2way_3bit.vhd)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Mux_2way_3bit is
    Port ( I0, I1 : in STD_LOGIC_VECTOR(2 downto 0);
           Sel : in STD_LOGIC;
           RegOut : out STD_LOGIC_VECTOR(2 downto 0));
end Mux_2way_3bit;

architecture Behavioral of Mux_2way_3bit is

begin
```

```
    process(I0, I1, Sel)
    begin
        if Sel = '0' then
            RegOut <= I0;
        else
            RegOut <= I1;
        end if;
    end process;
end Behavioral;
```

- **8-way 4-bit Multiplexer (Mux_8way_4bit.vhd)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Mux_8way_4bit is
    Port ( I0, I1, I2, I3, I4, I5, I6, I7 : in STD_LOGIC_VECTOR(3 downto 0);
           Sel : in STD_LOGIC_VECTOR(2 downto 0);
           RegOut : out STD_LOGIC_VECTOR(3 downto 0));
end Mux_8way_4bit;

architecture Behavioral of Mux_8way_4bit is

begin
    with Sel select
    RegOut <= I0 when "000",
              I1 when "001",
              I2 when "010",
              I3 when "011",
              I4 when "100",
              I5 when "101",
              I6 when "110",
              I7 when "111",
              (others => '0') when others;

end Behavioral;
```

- **Half Adder(HA.vhd)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity HA is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           S : out STD_LOGIC;
           C : out STD_LOGIC);
end HA;

architecture Behavioral of HA is

begin
S <= A XOR B;
C <= A AND B;
end Behavioral;
```

- **Full Adder(FA.vhd)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
SIM.VComponents.all;

entity 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 FA;

architecture Behavioral of FA is
component HA
     port (
     A: in std_logic;
     B: in std_logic;
     S: out std_logic;
     C: out std_logic);
end component;

SIGNAL HA0_S, HA0_C, HA1_S, HA1_C : std_logic;

begin

HA_0 : HA
     port map (
     A => A,
     B => B,
     S => HA0_S,
     C => HA0_C);

HA_1 : HA
     port map (
     A => HA0_S,
     B => C_in,
     S => HA1_S,
     C => HA1_C);

S <= HA1_S;
C_out <= HA0_C OR HA1_C;

end Behavioral;
```

- **3-Bit Adder (Adder_3bit.vhd)**

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

```
entity Adder_3bit is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           S : out STD_LOGIC_VECTOR (2 downto 0);
           C_out : out STD_LOGIC);
end Adder_3bit;

architecture Behavioral of Adder_3bit is

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

signal C1,C2,C3 : std_logic;
signal Sum : std_logic_vector (2 downto 0);
signal Inc : std_logic := '1';

begin
    FA_0 : FA
        port map(
            A => A(0),
            B => Inc,
            C_in => '0',
            S => Sum(0),
            C_out => C1
        );

    FA_1 : FA
        port map(
            A => A(1),
            B => '0',
            C_in => C1,
            S => Sum(1),
            C_out => C2
        );

    FA_2 : FA
        port map(
            A => A(2),
            B => '0',
            C_in => C2,
            S => Sum(2),
            C_out => C3
        );

 S <= Sum;
 C_out <= C3;


end Behavioral;
```

- **4-Bit Ripple Carry Adder(RCA_4.vhd)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity RCA_4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           C_in : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           C_out : out STD_LOGIC);
end RCA_4;

architecture Behavioral of RCA_4 is

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

SIGNAL FA0_S, FA0_C, FA1_S, FA1_C, FA2_S, FA2_C, FA3_S, FA3_C  : std_logic;

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

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

    FA_2 : FA
        port map (
             A => A(2),
             B => B(2),
             C_in => FA1_C,
             S => S(2),
             C_out => FA2_C);

    FA_3 : FA
        port map (
             A => A(3),
```

```
            B => B(3),
            C_in => FA2_C,
            S => S(3),
            C_out => FA3_C);

    C_out <= FA3_C;

end Behavioral;
```

- **4-Bit Add-Subtract Arithmetic Unit(Add_Sub_Unit.vhd)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Add_Sub_Unit is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Add_Sub_Sel : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Carry : out STD_LOGIC;
           Zero : out STD_LOGIC);
end Add_Sub_Unit;

architecture Behavioral of Add_Sub_Unit is

component RCA_4
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           C_in : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           C_out : out STD_LOGIC);
end component;

signal B_new, Sum : std_logic_vector(3 downto 0);
signal Cout: std_logic;

begin

B_new(0) <= Add_Sub_Sel XOR B(0);
B_new(1) <= Add_Sub_Sel XOR B(1);
B_new(2) <= Add_Sub_Sel XOR B(2);
B_new(3) <= Add_Sub_Sel XOR B(3);

RCA_4_0 : RCA_4
    Port map (
           A => A,
           B => B_new,
           C_in => Add_Sub_Sel,  --Two's complement +1
           S => Sum,
           C_out => Cout);

S <= Sum;
Carry <= Cout;
```

```
process(Sum)
begin
    if Sum="0000" and Cout='0'  then
        Zero <= '1';
    else
        Zero <= '0';
    end if;
end process;

end Behavioral;
```

- **4-Bit Register(Register_4bit.vhd)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Register_4bit is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end Register_4bit;

architecture Behavioral of Register_4bit is

begin
    process (Clk) begin
        if (rising_edge(Clk)) then
            if Reset = '0' then
                 if EN = '1' then
                    Q <= D;
                 end if;
            else
                Q <= "0000";
            end if;
         end if;
    end process;

end Behavioral;
```

- **2-to-4 decoder (Decoder_2_to_4.vhd)**

```
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
```

```
begin
    Y(0) <= EN AND (NOT I(1)) AND (NOT I(0));
    Y(1) <= EN AND (NOT I(1)) AND I(0);
    Y(2) <= EN AND I(1) AND (NOT I(0));
    Y(3) <= EN AND I(1) AND I(0);
end Behavioral;
```

- **3-to-8 decoder (Decoder_3_to_8.vhd)**

```
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 : STD_LOGIC_VECTOR (1 downto 0);
signal Y0,Y1 : STD_LOGIC_VECTOR (3 downto 0);
signal en0,en1, I2 : STD_LOGIC;

begin

    en0 <= NOT(I(2)) AND EN;
    en1 <= I(2) AND EN;
    I0 <= I(1 downto 0);

    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 => I0,
        EN => en1,
        Y => Y1 );

    Y(3 downto 0) <= Y0;
    Y(7 downto 4) <= Y1;

end Behavioral;
```

- **Register Bank(Register_Bank.vhd)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Register_Bank is
    Port ( Clk : in STD_LOGIC;
           RegEN : in STD_LOGIC_VECTOR (2 downto 0);
           Data : in STD_LOGIC_VECTOR (3 downto 0);
           Reset : in STD_LOGIC;
           Reg0 : out STD_LOGIC_VECTOR (3 downto 0);
           Reg1 : out STD_LOGIC_VECTOR (3 downto 0);
           Reg2 : out STD_LOGIC_VECTOR (3 downto 0);
           Reg3 : out STD_LOGIC_VECTOR (3 downto 0);
           Reg4 : out STD_LOGIC_VECTOR (3 downto 0);
           Reg5 : out STD_LOGIC_VECTOR (3 downto 0);
           Reg6 : out STD_LOGIC_VECTOR (3 downto 0);
           Reg7 : out STD_LOGIC_VECTOR (3 downto 0));
end Register_Bank;

architecture Behavioral of Register_Bank is

component Register_4Bit
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
       EN : in STD_LOGIC;
       Clk : in STD_LOGIC;
       Reset : 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 Select_Reg : STD_LOGIC_VECTOR (7 downto 0);

begin
    Decoder_3_to_8_0 : Decoder_3_to_8
              Port map( I=>RegEN,
                        EN=>'1',
                        Y=>Select_Reg );

    Reg_0 : Register_4bit
                Port map ( D=>"0000",
                           EN=>Select_Reg(0),
                           Clk=>Clk,
                           Reset=>Reset,
                           Q=> Reg0 ); --Hardcoded to zero
    Reg_1 : Register_4bit
                Port map ( D=>Data,
                           EN=>Select_Reg(1),
```

```
                        Clk=>Clk,
                        Reset=>Reset,
                        Q=> Reg1 );
    Reg_2 : Register_4bit
                Port map ( D=>Data,
                        EN=>Select_Reg(2),
                        Clk=>Clk,
                        Reset=>Reset,
                        Q=> Reg2 );
    Reg_3 : Register_4bit
                Port map ( D=>Data,
                        EN=>Select_Reg(3),
                        Clk=>Clk,
                        Reset=>Reset,
                        Q=> Reg3 );
    Reg_4 : Register_4bit
                Port map ( D=>Data,
                        EN=>Select_Reg(4),
                        Clk=>Clk,
                        Reset=>Reset,
                        Q=> Reg4 );
    Reg_5 : Register_4bit
                Port map ( D=>Data,
                        EN=>Select_Reg(5),
                        Clk=>Clk,
                        Reset=>Reset,
                        Q=> Reg5 );
    Reg_6 : Register_4bit
                Port map ( D=>Data,
                        EN=>Select_Reg(6),
                        Clk=>Clk,
                        Reset=>Reset,
                        Q=> Reg6 );
    Reg_7 : Register_4bit
                Port map ( D=>Data,
                        EN=>Select_Reg(7),
                        Clk=>Clk,
                        Reset=>Reset,
                        Q=> Reg7 );

end Behavioral;
```

- **Nanoprocessor(Nanoprocessor.vhd)**

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

entity Nanoprocessor is
    Port ( Clk : in STD_LOGIC;
       Reset : in STD_LOGIC;
       Overflow : out STD_LOGIC;
       Zero : out STD_LOGIC;
       Reg_7_Out : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end Nanoprocessor;

architecture Behavioral of Nanoprocessor is

component Register_Bank
    Port ( Clk : in STD_LOGIC;
       RegEN : in STD_LOGIC_VECTOR (2 downto 0);
       Data : in STD_LOGIC_VECTOR (3 downto 0);
       Reset : in STD_LOGIC;
       Reg0 : out STD_LOGIC_VECTOR (3 downto 0);
       Reg1 : out STD_LOGIC_VECTOR (3 downto 0);
       Reg2 : out STD_LOGIC_VECTOR (3 downto 0);
       Reg3 : out STD_LOGIC_VECTOR (3 downto 0);
       Reg4 : out STD_LOGIC_VECTOR (3 downto 0);
       Reg5 : out STD_LOGIC_VECTOR (3 downto 0);
       Reg6 : out STD_LOGIC_VECTOR (3 downto 0);
       Reg7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Program_Counter
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
       Res : in STD_LOGIC;
       Clk : in STD_LOGIC;
       Q : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component Program_ROM
    Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
       instructions : out STD_LOGIC_VECTOR (11 downto 0));
end component;

component Instruction_Decoder
    Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
       Reg_Check_Jump : in STD_LOGIC_VECTOR (3 downto 0);
       Add_Sub_Sel : out STD_LOGIC;
       RegA: out STD_LOGIC_VECTOR (2 downto 0);
       RegB : out STD_LOGIC_VECTOR (2 downto 0);
       Immediate_Value : out STD_LOGIC_VECTOR (3 downto 0);
       Load_Sel : out STD_LOGIC;
       Reg_EN : out STD_LOGIC_VECTOR (2 downto 0);
       Jump_Flag : out STD_LOGIC;
       Jump_Address : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component Add_Sub_Unit
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
       B : in STD_LOGIC_VECTOR (3 downto 0);
       Add_Sub_Sel : in STD_LOGIC;
       S : out STD_LOGIC_VECTOR (3 downto 0);
       Carry : out STD_LOGIC;
       Zero : out STD_LOGIC);
end component;

component Adder_3bit
```

```
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
       S : out STD_LOGIC_VECTOR (2 downto 0);
       C_out : out STD_LOGIC);
end component;

component Mux_8way_4bit
    Port ( I0, I1, I2, I3, I4, I5, I6, I7 : in STD_LOGIC_VECTOR(3 downto 0);
       Sel : in STD_LOGIC_VECTOR(2 downto 0);
       RegOut : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component Mux_2way_3bit
    Port ( I0, I1 : in STD_LOGIC_VECTOR(2 downto 0);
       Sel : in STD_LOGIC;
       RegOut : out STD_LOGIC_VECTOR(2 downto 0));
end component;

component Mux_2way_4bit
    Port ( I0, I1 : in STD_LOGIC_VECTOR(3 downto 0);
       Sel : in STD_LOGIC;
       RegOut : out STD_LOGIC_VECTOR(3 downto 0));
end component;

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

signal load_sel, sub, jflag : STD_LOGIC;
signal Pointer, PCaddress, reg_en, jump_add, adderOut, mux_1, mux_2
 :STD_LOGIC_VECTOR (2 downto 0);
signal Instruction : STD_LOGIC_VECTOR (11 downto 0);
signal IVal, result, reg_data, r0,r1,r2,r3,r4,r5,r6,r7, data1, data2
 :STD_LOGIC_VECTOR (3 downto 0);

begin

Program_Counter_0 : Program_Counter
port map(
   D => Pointer,
   Res => Reset,
   Clk => Clk,
   Q => PCaddress);

Program_Rom_0 : Program_Rom
port map (
   address => PCaddress,
   instructions => Instruction);

Instruction_Decoder_0 : Instruction_Decoder
port map (
  Instruction => Instruction,
  Reg_Check_Jump => data1,
  Reg_EN => reg_en,
```

```
  Load_Sel => load_sel,
  Immediate_Value => IVal,
  RegA => mux_1,
  RegB => mux_2,
  Add_Sub_Sel => sub,
  Jump_Flag => jflag,
  Jump_Address => jump_add );

Register_Bank_0 : Register_Bank
port map (
  Clk    => Clk,
  RegEN  => reg_en,
  Data   => reg_data,
  Reset  => Reset,
  Reg0   => r0,
  Reg1   => r1,
  Reg2   => r2,
  Reg3   => r3,
  Reg4   => r4,
  Reg5   => r5,
  Reg6   => r6,
  Reg7   => r7);

Mux_8way_4bit_0 : Mux_8way_4bit
port map (
I0 => r0, I1 => r1, I2 => r2, I3 => r3, I4 => r4, I5 => r5, I6 => r6, I7=> r7,
  Sel => mux_1,
  RegOut => data1);

Mux_8way_4bit_1 : Mux_8way_4bit
port map (
  I0=> r0, I1 => r1, I2 => r2, I3 => r3, I4 => r4, I5=> r5, I6=> r6, I7=> r7,
  Sel => mux_2,
  RegOut => data2);

Add_Sub_Unit_0 : Add_Sub_Unit
port map (
  A => data1,
  B => data2,
  Add_Sub_Sel => sub,
  S => result,
  Carry => Overflow,
  Zero => Zero);

Mux_2way_4bit_0 : Mux_2way_4bit
port map (
  I0 => result,
  I1 => IVal,
  Sel => load_sel,
  RegOut => reg_data);

Adder_3bit_0 : Adder_3bit
port map (
  A => PCaddress,
```

```
  S => adderOut);

Mux_2way_3bit_0 : Mux_2way_3bit
port map (
  I0 => adderOut,
  I1 => jump_add,
  Sel => jflag,
  RegOut => Pointer);

Reg_7_Out <= r7;
end Behavioral;
```

- **16-to-7 Bit Lookup Table (LUT_16_7.vhd)**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
entity LUT_16_7 is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end LUT_16_7;

architecture Behavioral of LUT_16_7 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", --2
                              "0110000", --3
                              "0011001", --4
                              "0010010", --5
                              "0000010", --6
                              "1111000", --7
                              "0000000", --8
                              "0010000", --9
                              "0001000", --a
                              "0000011", --b
                              "1000110", --c
                              "0100001", --d
                              "0000110", --e
                              "0001110"  --f
    );

begin

data <= sevenSegment_ROM(to_integer(unsigned(address)));

end Behavioral;
```

- **Main Program (MainProgram.vhd)**

```
library IEEE;
```

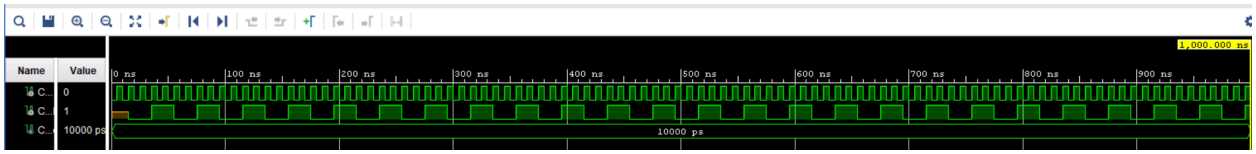
```
use IEEE.STD_LOGIC_1164.ALL;

entity MainProgram is
    Port ( Clk : in STD_LOGIC;
       Reset : in STD_LOGIC;
       Seg_7_Out : out STD_LOGIC_VECTOR (6 downto 0);
       Reg_7_DOut : out STD_LOGIC_VECTOR (3 downto 0);
       Overflow : out STD_LOGIC;
       Zero : out STD_LOGIC;
       Anode : out STD_LOGIC_VECTOR (3 downto 0));
end MainProgram;

architecture Behavioral of MainProgram is

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

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

component Nanoprocessor
    Port ( Clk : in STD_LOGIC;
       Reset : in STD_LOGIC;
       Overflow : out STD_LOGIC;
       Zero : out STD_LOGIC;
       Reg_7_Out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal Display_out : STD_LOGIC_VECTOR (3 downto 0);
signal SlowClk : STD_LOGIC;

begin

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

    Nanoprocessor_0 : Nanoprocessor
    port map (
         Clk => SlowClk,
         Reset => Reset,
         Overflow => Overflow,
         Zero => Zero,
         Reg_7_Out => Display_out);
    Reg_7_Out <= Display_out;
    LUT_16_7_0 : LUT_16_7
        port map (
          address => Display_out,
          data    => Seg_7_Out);
```

```
    Reg_7_DOut <= Display_out;
    Anode <= "1110";

end Behavioral;
```

VHDL Codes (Test Bench Files)

- ## Slow Clock (TB_Slow_Clk.vhd)

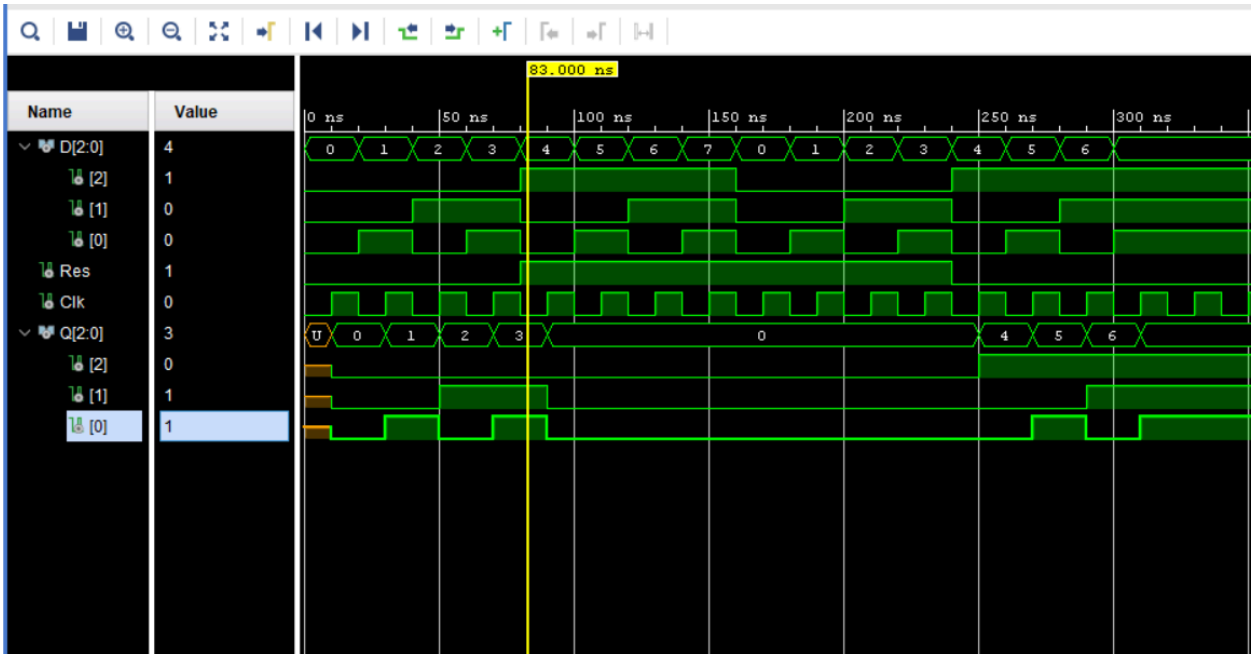
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

architecture Behavioral of TB_Slow_Clk is
    component Slow_Clk
        Port ( Clk_in : in STD_LOGIC;
               Clk_out : out STD_LOGIC);
    end component;

   signal Clk_in_tb : STD_LOGIC := '0';
   signal Clk_out_tb : STD_LOGIC;
   constant Clk_Period : time := 10 ns;

begin
    uut: Slow_Clk
    port map (
        Clk_in => Clk_in_tb,
        Clk_out => Clk_out_tb
    );
    process
        begin
            while now < 1 ms loop  -- Simulate for 1 ms
                Clk_in_tb <= '0';
                wait for Clk_Period/2;
                Clk_in_tb <= '1';
                wait for Clk_Period/2;
        end loop;
        wait;
    end process;

end Behavioral;
```



- **Program Counter(TB_Program_Counter.vhd)**

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

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

architecture Behavioral of TB_Program_Counter is
    component Program_Counter
        Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
               Res : in STD_LOGIC;
               Clk : in STD_LOGIC;
               Q : out STD_LOGIC_VECTOR (2 downto 0));
    end component;

    signal D     : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal Res   : STD_LOGIC ;
    signal Clk   : STD_LOGIC := '0';
    signal Q     : STD_LOGIC_VECTOR(2 downto 0);
begin
    uut: Program_Counter Port Map (
        D => D,
        Res => Res,
        Clk => Clk,
        Q => Q
    );
    process
        begin
        while true loop
            Clk <= '0';
            wait for 10 ns;
            Clk <= '1';
            wait for 10 ns;
        end loop;
    end process;

    process
        begin
            Res <= '0';
            D <= "000";
            wait for 20 ns;

            D <= "001";
            wait for 20 ns;
```

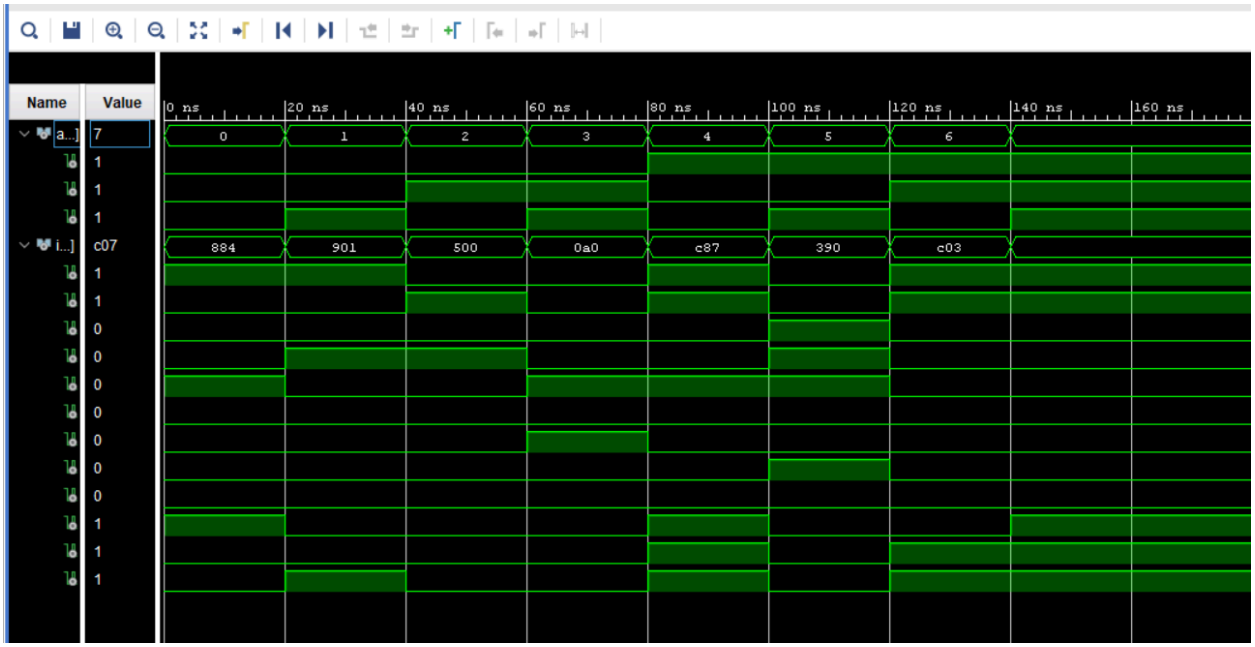
```
            D <= "010";
            wait for 20 ns;

            D <= "011";
            wait for 20 ns;
            Res <= '1';

            D <= "100";
            wait for 20 ns;

            D <= "101";
            wait for 20 ns;

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

            D <= "111";
            wait for 20 ns;

            Res <= '1';
            D <= "000";
            wait for 20 ns;

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

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

            D <= "011";
            wait for 20 ns;
            Res <= '0';

            D <= "100";
            wait for 20 ns;

            D <= "101";
            wait for 20 ns;

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

            D <= "111";
            wait for 20 ns;
            wait;
    end process;

end Behavioral;
```



- **Program ROM (TB_Program_ROM.vhd)**

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

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

architecture Behavioral of TB_Program_ROM is
    component Program_ROM
        Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
               instructions : out STD_LOGIC_VECTOR (11 downto 0));
    end component;

    signal address : STD_LOGIC_VECTOR (2 downto 0) := "000";
    signal instructions : STD_LOGIC_VECTOR (11 downto 0);

begin

    uut: Program_ROM
        Port Map (
            address => address,
            instructions => instructions);

    process
        begin
            address <= "000";
            wait for 20 ns;
            address <= "001";
            wait for 20 ns;
            address <= "010";
```

```
            wait for 20 ns;
            address <= "011";
            wait for 20 ns;
            address <= "100";
            wait for 20 ns;
            address <= "101";
            wait for 20 ns;
            address <= "110";
            wait for 20 ns;
            address <= "111";
            wait for 20 ns;
            wait;
    end process;


end Behavioral;
```



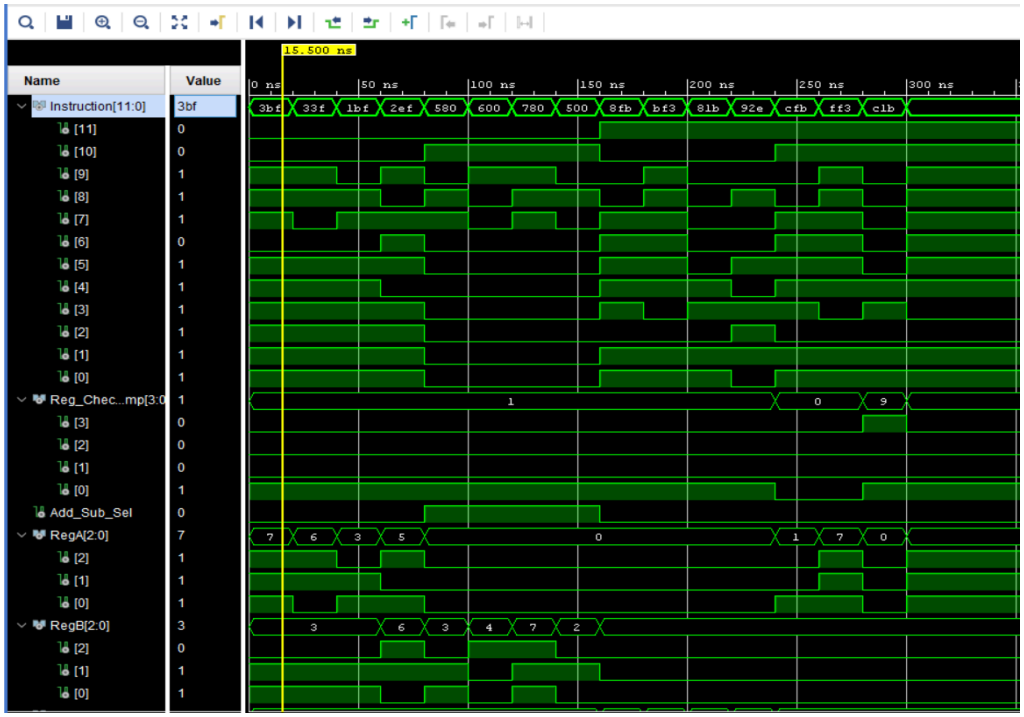
- **Instruction Decoder (TB_Instruction_Decoder.vhd)**

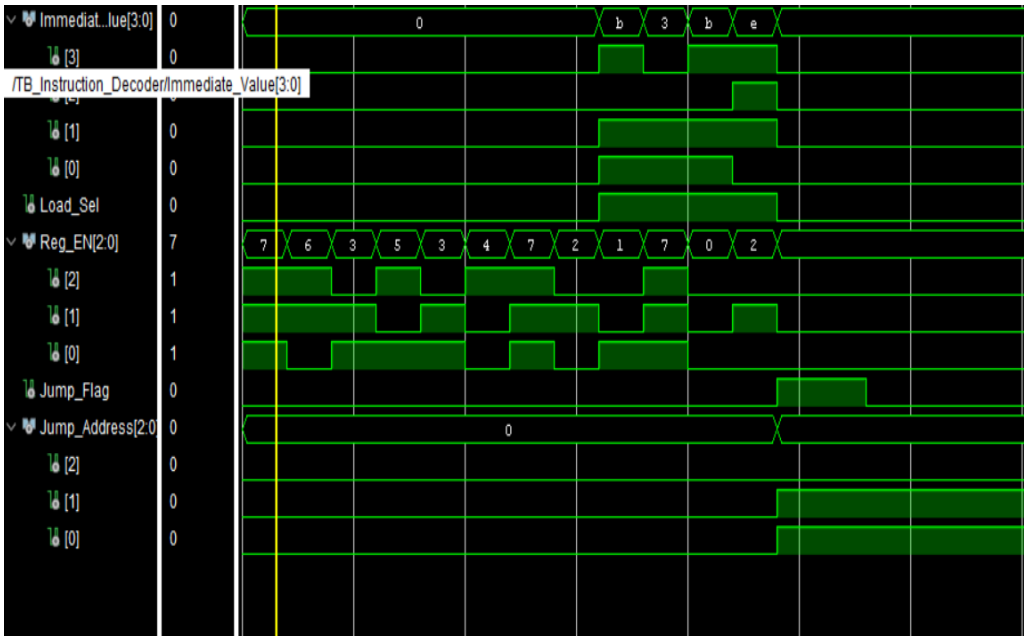
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

architecture Behavioral of TB_Instruction_Decoder is
    component Instruction_Decoder
        Port (
            Instruction : in STD_LOGIC_VECTOR (11 downto 0);
            Reg_Check_Jump : in STD_LOGIC_VECTOR (3 downto 0);
```

```
            Add_Sub_Sel : out STD_LOGIC;
            RegA : out STD_LOGIC_VECTOR (2 downto 0);
            RegB : out STD_LOGIC_VECTOR (2 downto 0);
            Immediate_Value : out STD_LOGIC_VECTOR (3 downto 0);
            Load_Sel : out STD_LOGIC;
            Reg_EN : out STD_LOGIC_VECTOR (2 downto 0);
            Jump_Flag : out STD_LOGIC;
            Jump_Address : out STD_LOGIC_VECTOR (2 downto 0)
        );
    end component;

    signal Instruction : STD_LOGIC_VECTOR (11 downto 0);
    signal Reg_Check_Jump : STD_LOGIC_VECTOR (3 downto 0);
    signal Add_Sub_Sel : STD_LOGIC;
    signal RegA : STD_LOGIC_VECTOR (2 downto 0);
    signal RegB : STD_LOGIC_VECTOR (2 downto 0);
    signal Immediate_Value : STD_LOGIC_VECTOR (3 downto 0);
    signal Load_Sel : STD_LOGIC;
    signal Reg_EN : STD_LOGIC_VECTOR (2 downto 0);
    signal Jump_Flag : STD_LOGIC;
    signal Jump_Address : STD_LOGIC_VECTOR (2 downto 0);

begin

    uut: Instruction_Decoder
        port map (
            Instruction => Instruction,
            Reg_Check_Jump => Reg_Check_Jump,
            Add_Sub_Sel => Add_Sub_Sel,
            RegA => RegA,
            RegB => RegB,
            Immediate_Value => Immediate_Value,
            Load_Sel => Load_Sel,
            Reg_EN => Reg_EN,
            Jump_Flag => Jump_Flag,
            Jump_Address => Jump_Address
        );

    process
    begin
        --230651=11 1000010011 111 011
        --230387=11 1000001111 110 011
        --230427=11 1000010000 011 011
        --230702=11 1000010100 101 110

        -- Test ADD instruction
        --230651=11 0011 111 011
        Instruction <= "001110111111"; -- ADD R1=111, R2=011
        Reg_Check_Jump <= "0001"; -- dummy value
        wait for 20 ns;
        --230387=11 1000001111 110 011
        Instruction <= "001100111111"; -- ADD R1=110, R2=011
        Reg_Check_Jump <= "0001";
        wait for 20 ns;
```

```
--230427=11 1000010000 011 011
Instruction <= "000110111111"; -- ADD R1=011, R2=011
Reg_Check_Jump <= "0001";
wait for 20 ns;
--230702=11 1000010100 101 110
Instruction <= "001011101111"; -- ADD R1=101, R2=110
Reg_Check_Jump <= "0001";
wait for 20 ns;


-- Test NEG instruction:
Instruction <= "010110000000"; -- NEG R2
Reg_Check_Jump <= "0001";
wait for 20 ns;
Instruction <= "011000000000"; -- NEG R2
Reg_Check_Jump <= "0001";
wait for 20 ns;
Instruction <= "011110000000"; -- NEG R2
Reg_Check_Jump <= "0001";
wait for 20 ns;
Instruction <= "010100000000"; -- NEG R2
Reg_Check_Jump <= "0001";
wait for 20 ns;


-- Test MOVI instruction:
--230651=>0011111011
Instruction <= "100011111011";
Reg_Check_Jump <= "0001";
wait for 20 ns;
--230387=>01111110011
Instruction <= "101111110011";
Reg_Check_Jump <= "0001";
wait for 20 ns;
--230427=>0000011011
Instruction <= "100000011011";
Reg_Check_Jump <= "0001";
wait for 20 ns;
--230702=>0100101110
Instruction <= "100100101110";
Reg_Check_Jump <= "0001";
wait for 20 ns;



-- Test JZR instruction with zero reg:
--230651=>0011111011
Instruction <= "110011111011";
Reg_Check_Jump <= "0000"; -- Zero ? Jump expected
wait for 20 ns;
--230387=>01111110011
Instruction <= "111111110011";
Reg_Check_Jump <= "0000";
wait for 20 ns;
```

```
        -- Test JZR instruction with non-zero reg:
        --230427=>0000011011
        Instruction <= "110000011011";
        Reg_Check_Jump <= "1001"; -- Non-zero ? No jump
        wait for 20 ns;
        --230387=>01111110011
        Instruction <= "111111110011";
        Reg_Check_Jump <= "0001";
        wait for 20 ns;
        wait; -- Wait forever
    end process;

end Behavioral;
```





- **2-way 4-bit Multiplexer (TB_Mux_2way_4bit.vhd)**

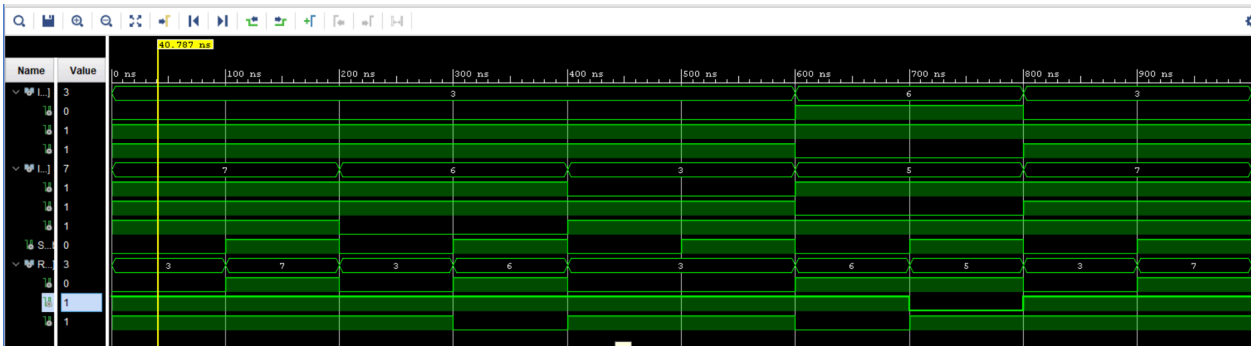
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TB_Mux_2way_4bit is
--  Port ( );
end TB_Mux_2way_4bit;

architecture Behavioral of TB_Mux_2way_4bit is
component Mux_2way_4bit
        Port (
            I0 : in STD_LOGIC_VECTOR(3 downto 0);
            I1 : in STD_LOGIC_VECTOR(3 downto 0);
            Sel : in STD_LOGIC;
            RegOut : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;
signal I0_tb : STD_LOGIC_VECTOR(3 downto 0) ;
signal I1_tb : STD_LOGIC_VECTOR(3 downto 0) ;
signal Sel_tb : STD_LOGIC ;
signal RegOut_tb : STD_LOGIC_VECTOR(3 downto 0);
begin
uut: Mux_2way_4bit
        Port map (
            I0 => I0_tb,
            I1 => I1_tb,
            Sel => Sel_tb,
            RegOut => RegOut_tb);
process
    begin
    --230651=11 1000 0100 1111 1011
    --230387=11 1000 0011 1111 0011
    --230427=11 1000 0100 0001 1011
    --230702=11 1000 0101 0010 1110
    --get each last 8 digits for I0_tb and I1_tb
    I0_tb <= "1011";
    I1_tb <= "1111";
    Sel_tb <= '0';
    wait for 100 ns;
    Sel_tb <= '1';
    wait for 100 ns;

    I0_tb <= "0011";
    I1_tb <= "1111";
    Sel_tb <= '0';
    wait for 100 ns;
    Sel_tb <= '1';
    wait for 100 ns;

    I0_tb <= "1011";
    I1_tb <= "0001";
```

```
    Sel_tb <= '0';
    wait for 100 ns;
    Sel_tb <= '1';
    wait for 100 ns;

    I0_tb <= "1110";
    I1_tb <= "0010";
    Sel_tb <= '0';
    wait for 100 ns;
    Sel_tb <= '1';
    wait for 100 ns;

    end process;

end Behavioral;
```

- ## 2-way 3-bit Multiplexer (TB_Mux_2way_3bit.vhd)

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

entity TB_Mux_2way_3bit is
--  Port ( );
end TB_Mux_2way_3bit;

architecture Behavioral of TB_Mux_2way_3bit is
component Mux_2way_3bit
        Port (
            I0 : in STD_LOGIC_VECTOR(2 downto 0);
            I1 : in STD_LOGIC_VECTOR(2 downto 0);
            Sel : in STD_LOGIC;
            RegOut : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;
signal I0_tb : STD_LOGIC_VECTOR(2 downto 0) ;
signal I1_tb : STD_LOGIC_VECTOR(2 downto 0) ;
```

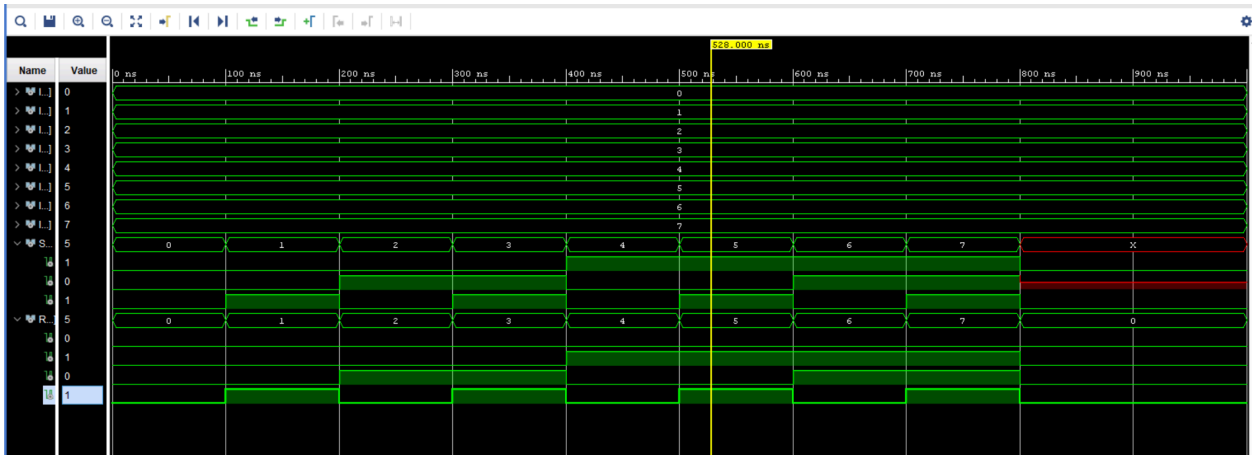
```
signal Sel_tb : STD_LOGIC := '0';
signal RegOut_tb : STD_LOGIC_VECTOR(2 downto 0);

begin
uut: Mux_2way_3bit
        port map (
            I0 => I0_tb,
            I1 => I1_tb,
            Sel => Sel_tb,
            RegOut => RegOut_tb
        );
process
    begin
    --230651=11 1000010011 111 011
    --230387=11 1000001111 110 011
    --230427=11 1000010000 011 011
    --230702=11 1000010100 101 110
    --get each last 6 digits for I0_tb and I1_tb
    I0_tb <= "011";
    I1_tb <= "111";
    Sel_tb <= '0';
    wait for 100 ns;
    Sel_tb <= '1';
    wait for 100 ns;

    I0_tb <= "011";
    I1_tb <= "110";
    Sel_tb <= '0';
    wait for 100 ns;
    Sel_tb <= '1';
    wait for 100 ns;

    I0_tb <= "011";
    I1_tb <= "011";
    Sel_tb <= '0';
    wait for 100 ns;
    Sel_tb <= '1';
    wait for 100 ns;

    I0_tb <= "110";
    I1_tb <= "101";
    Sel_tb <= '0';
    wait for 100 ns;
    Sel_tb <= '1';
    wait for 100 ns;

    end process;
end Behavioral;
```



- **8-way 4-bit Multiplexer (TB_Mux_8way_4bit.vhd)**

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

entity TB_Mux_8way_4bit is
--  Port ( );
end TB_Mux_8way_4bit;

architecture Behavioral of TB_Mux_8way_4bit is

    component Mux_8way_4bit
        Port (
            I0, I1, I2, I3, I4, I5, I6, I7 : in STD_LOGIC_VECTOR(3 downto 0);
            Sel : in STD_LOGIC_VECTOR(2 downto 0);
            RegOut : out STD_LOGIC_VECTOR(3 downto 0));
    end component;

    signal I0_tb, I1_tb, I2_tb, I3_tb : STD_LOGIC_VECTOR(3 downto 0);
    signal I4_tb, I5_tb, I6_tb, I7_tb : STD_LOGIC_VECTOR(3 downto 0);
    signal Sel_tb : STD_LOGIC_VECTOR(2 downto 0);
    signal RegOut_tb : STD_LOGIC_VECTOR(3 downto 0);

begin
    uut: Mux_8way_4bit
    port map (
        I0 => I0_tb, I1 => I1_tb, I2 => I2_tb, I3 => I3_tb,
        I4 => I4_tb, I5 => I5_tb, I6 => I6_tb, I7 => I7_tb,
        Sel => Sel_tb,
        RegOut => RegOut_tb );

process
    begin
    I0_tb <= "0000";
    I1_tb <= "0001";
    I2_tb <= "0010";
    I3_tb <= "0011";
    I4_tb <= "0100";
    I5_tb <= "0101";
    I6_tb <= "0110";
```

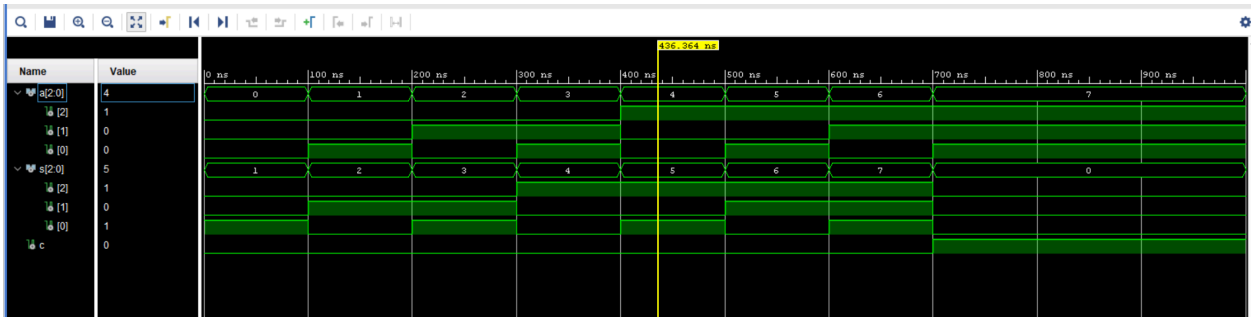
```
    I7_tb <= "0111";

    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;
    Sel_tb <= "0X0"; wait for 100 ns;--Test invalid Sel

            wait;
        end process;
end Behavioral;
```



- **3-Bit Adder (TB_Adder_3bit.vhd)**

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

entity TB_Adder_3bit is
--  Port ( );
end TB_Adder_3bit;

architecture Behavioral of TB_Adder_3bit is

component Adder_3bit
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           S : out STD_LOGIC_VECTOR (2 downto 0);
           C_out : out STD_LOGIC);
end component;

signal a,s : std_logic_vector (2 downto 0);
signal c : std_logic;

begin
```

```
UUT :Adder_3bit port map(
        A => a,
        S => s,
        C_out => c
    );

process begin
   a <= "000";
   wait for 100 ns;

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

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

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

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

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

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

   a <= "111";
   wait;

end process;

end Behavioral;
```



- **4-Bit Add-Subtract Arithmetic Unit(TB_Add_Sub_Unit.vhd)**

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

entity TB_Add_Sub_Unit is
```

```
--  Port ( );
end TB_Add_Sub_Unit;

architecture Behavioral of TB_Add_Sub_Unit is

component Add_Sub_Unit
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
       B : in STD_LOGIC_VECTOR (3 downto 0);
       Add_Sub_Sel : in STD_LOGIC;
       S : out STD_LOGIC_VECTOR (3 downto 0);
       Carry : out STD_LOGIC;
       Zero : out STD_LOGIC);
end component;

signal a,b,s : std_logic_vector (3 downto 0);
signal sub, c, z : std_logic;

begin
    UUT: Add_Sub_Unit port map(
    A => a,
    B => b,
    Add_Sub_Sel => sub,
    S => s,
    Carry => c,
    Zero => z
    );

process begin

-- 1 + 2 --
a <= "0001";
b <= "0010";
sub <= '0';
wait for 100 ns;

-- 7 + 7 --
a <= "0111";
b <= "0111";
sub <= '0';
wait for 100 ns;

-- 7 - 5 --
a <= "0111";
b <= "0101";
sub <= '1';
wait for 100 ns;

-- 4 - 4 --
a <= "0100";
b <= "0100";
sub <= '1';
wait for 100 ns;

-- 1 - 2 --
```

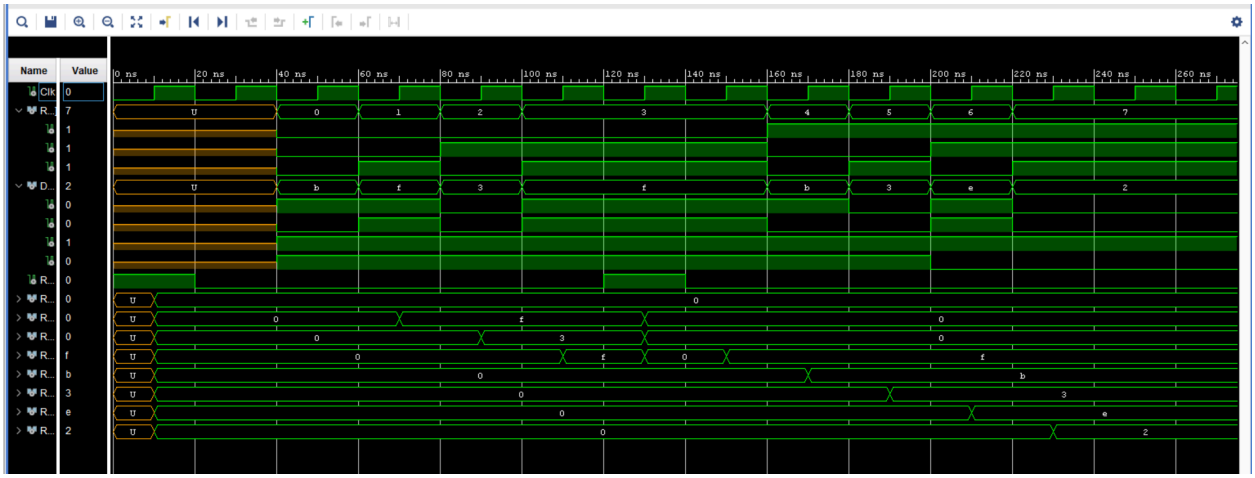
```
a <= "0001";
b <= "0010";
sub <= '1';
wait for 100 ns;

-- (-1) + (-1) --
a <= "1111";
b <= "1111";
sub <= '0';
wait for 100 ns;

-- (-3) - (-7) --
a <= "1101";
b <= "1001";
sub <= '1';
wait ;

end process;

end Behavioral;
```

- **Register Bank(Register_Bank.vhd)**

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

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

architecture Behavioral of TB_Register_Bank is
    component Register_Bank
        Port ( Clk : in STD_LOGIC;
               RegEN : in STD_LOGIC_VECTOR (2 downto 0);
```

```
            Data : in STD_LOGIC_VECTOR (3 downto 0);
            Reset : in STD_LOGIC;
            Reg0 : out STD_LOGIC_VECTOR (3 downto 0);
            Reg1 : out STD_LOGIC_VECTOR (3 downto 0);
            Reg2 : out STD_LOGIC_VECTOR (3 downto 0);
            Reg3 : out STD_LOGIC_VECTOR (3 downto 0);
            Reg4 : out STD_LOGIC_VECTOR (3 downto 0);
            Reg5 : out STD_LOGIC_VECTOR (3 downto 0);
            Reg6 : out STD_LOGIC_VECTOR (3 downto 0);
            Reg7 : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    signal Clk         : STD_LOGIC := '0';
    signal RegEN       : STD_LOGIC_VECTOR (2 downto 0);
    signal Data        : STD_LOGIC_VECTOR (3 downto 0);
    signal Reset       : STD_LOGIC := '0';
    signal Reg0, Reg1, Reg2, Reg3, Reg4, Reg5, Reg6, Reg7 : STD_LOGIC_VECTOR (3
downto 0);

    begin
        uut: Register_Bank
            Port map (
                Clk => Clk,
                RegEN => RegEN,
                Data => Data,
                Reset => Reset,
                Reg0 => Reg0,
                Reg1 => Reg1,
                Reg2 => Reg2,
                Reg3 => Reg3,
                Reg4 => Reg4,
                Reg5 => Reg5,
                Reg6 => Reg6,
                Reg7 => Reg7
            );

    -- Clock process
process
    begin
        Clk <= '0';
        wait for 10 ns;
        Clk <= '1';
        wait for 10 ns;
end process;


process
    begin
        -- Reset system
        Reset <= '1';
        wait for 20 ns;
        Reset <= '0';
        wait for 20 ns;

        --230651=11 1000 0100 1111 1011
```

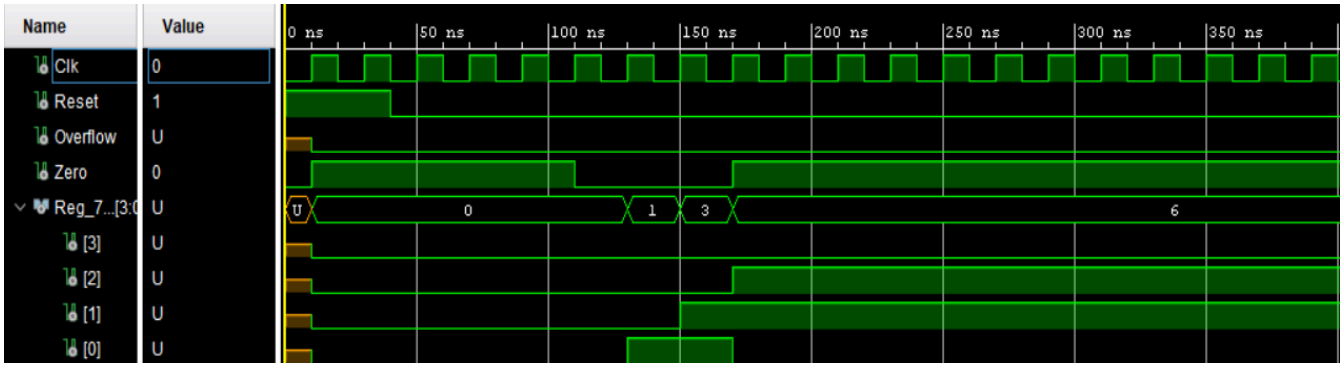
```
        -- Write 1011 to Reg0
        RegEN <= "000";
        Data <= "1011";
        wait for 20 ns;
        -- Write 1111 to Reg1
        RegEN <= "001";
        Data <= "1111";
        wait for 20 ns;

        --230387=11 1000 0011 1111 0011
        -- Write 0011 to Reg2
        RegEN <= "010";
        Data <= "0011";
        wait for 20 ns;
        -- Write 1111 to Reg3
        RegEN <= "011";
        Data <= "1111";
        wait for 20 ns;
        Reset <= '1';
        wait for 20 ns;
        Reset <= '0';
        wait for 20 ns;
        --230427=11 1000 0100 0001 1011
        -- Write 1011 to Reg4
        RegEN <= "100";
        Data <= "1011";
        wait for 20 ns;
        -- Write 0001 to Reg5
        RegEN <= "101";
        Data <= "0011";
        wait for 20 ns;

        --230702=11 1000 0101 0010 1110
        -- Write 1110 to Reg6
        RegEN <= "110";
        Data <= "1110";
        wait for 20 ns;
        -- Write 0010 to Reg7
        RegEN <= "111";
        Data <= "0010";
        wait for 20 ns;
        wait;
end process;

end Behavioral;
```



- **Nanoprocessor(TB_Nanoprocessor.vhd)**

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

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

architecture Behavioral of TB_Nanoprocessor is

    component Nanoprocessor
        Port (
            Clk         : in STD_LOGIC;
            Reset       : in STD_LOGIC;
            Overflow    : out STD_LOGIC;
            Zero        : out STD_LOGIC;
            Reg_7_Out   : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

signal Clk         : STD_LOGIC := '0';
signal Reset       : STD_LOGIC := '1';
signal Overflow    : STD_LOGIC;
signal Zero        : STD_LOGIC;
signal Reg_7_Out   : STD_LOGIC_VECTOR(3 downto 0);

begin
    uut: Nanoprocessor
        port map (
            Clk       => Clk,
            Reset     => Reset,
            Overflow  => Overflow,
            Zero      => Zero,
            Reg_7_Out => Reg_7_Out
        );
```

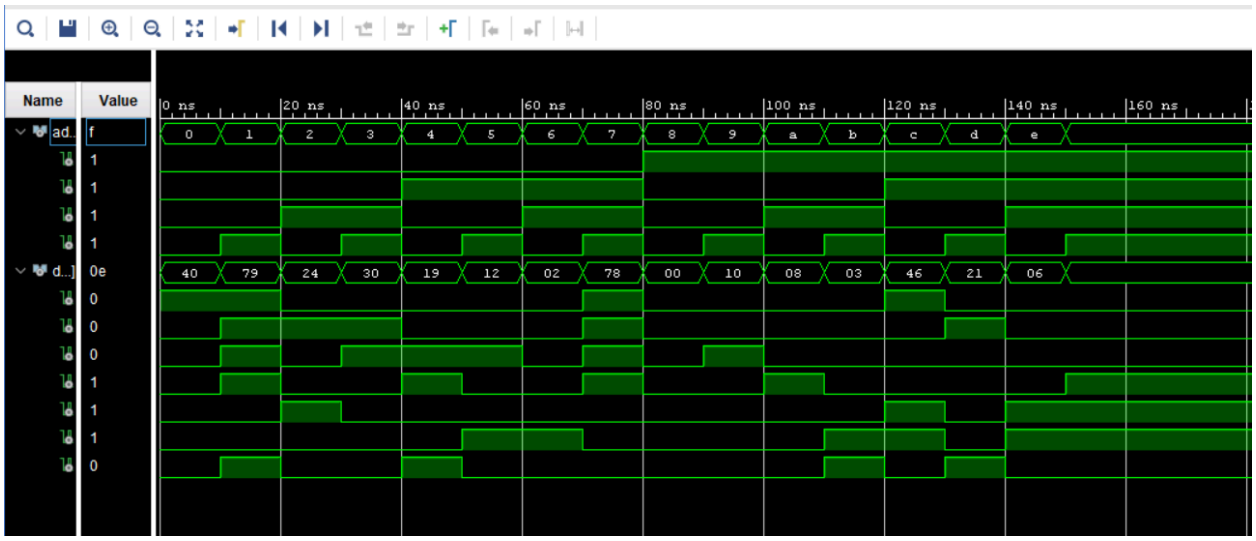
```
Clk_Process : process
begin
    while now < 500 ns loop
        Clk <= '0';
        wait for 10 ns;
        Clk <= '1';
        wait for 10 ns;
    end loop;
    wait;
end process;

process
    begin
        -- Initial Reset
        Reset <= '1';
        wait for 40 ns;
        Reset <= '0';

        wait for 400 ns;
        wait;
end process;

end Behavioral;
```



- **16-to-7 Bit Lookup Table (TB_LUT_16_7.vhd)**

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

entity TB_LUT_16_7 is
--  Port ( );
end TB_LUT_16_7;

architecture Behavioral of TB_LUT_16_7 is
    component LUT_16_7
        Port (
            address : in STD_LOGIC_VECTOR (3 downto 0);
            data : out STD_LOGIC_VECTOR (6 downto 0)
        );
```

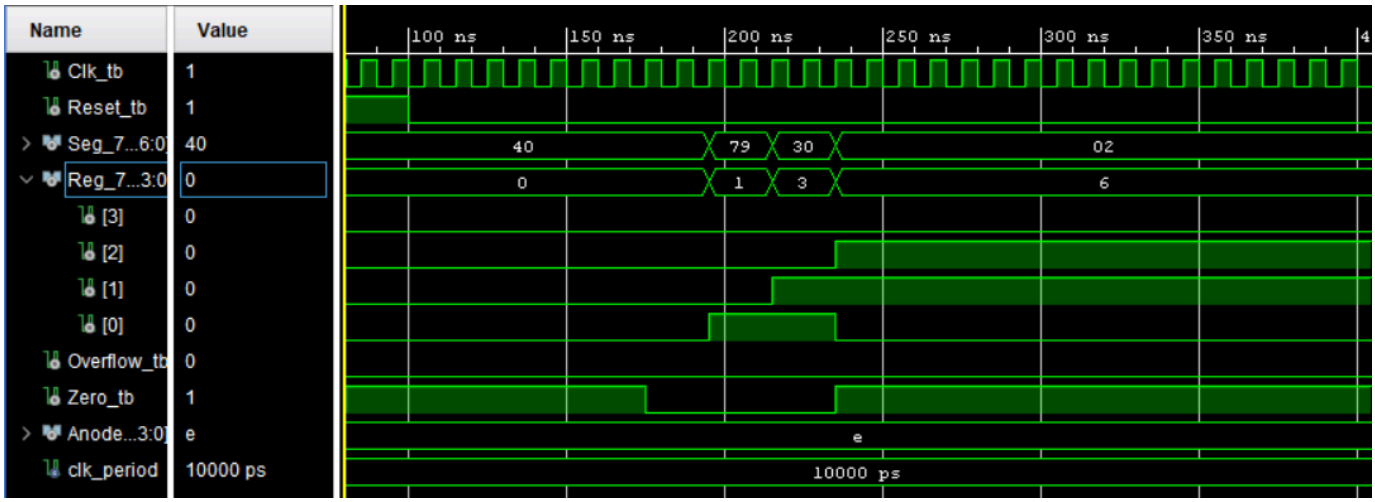
```
    end component;

    signal address_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal data_tb : STD_LOGIC_VECTOR (6 downto 0);
begin
    uut: LUT_16_7
        Port map (
            address => address_tb,
            data => data_tb
        );

process
    begin
        address_tb <= "0000"; wait for 10 ns;
        address_tb <= "0001"; wait for 10 ns;
        address_tb <= "0010"; wait for 10 ns;
        address_tb <= "0011"; wait for 10 ns;
        address_tb <= "0100"; wait for 10 ns;
        address_tb <= "0101"; wait for 10 ns;
        address_tb <= "0110"; wait for 10 ns;
        address_tb <= "0111"; wait for 10 ns;
        address_tb <= "1000"; wait for 10 ns;
        address_tb <= "1001"; wait for 10 ns;
        address_tb <= "1010"; wait for 10 ns;
        address_tb <= "1011"; wait for 10 ns;
        address_tb <= "1100"; wait for 10 ns;
        address_tb <= "1101"; wait for 10 ns;
        address_tb <= "1110"; wait for 10 ns;
        address_tb <= "1111"; wait for 10 ns;

        wait;
    end process;
end Behavioral;
```



- **Main Program (TB_MainProgram.vhd)**

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

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

architecture Behavioral of TB_MainProgram is
component MainProgram
        Port ( Clk : in STD_LOGIC;
               Reset : in STD_LOGIC;
               Seg_7_Out : out STD_LOGIC_VECTOR (6 downto 0);
               Reg_7_DOut : out STD_LOGIC_VECTOR (3 downto 0);
               Overflow : out STD_LOGIC;
               Zero : out STD_LOGIC;
               Anode : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    signal Clk_tb        : STD_LOGIC := '0';
    signal Reset_tb      : STD_LOGIC := '1';
    signal Seg_7_Out_tb  : STD_LOGIC_VECTOR (6 downto 0);
    signal Reg_7_Out_tb  : STD_LOGIC_VECTOR (3 downto 0);
    signal Overflow_tb   : STD_LOGIC;
    signal Zero_tb       : STD_LOGIC;
    signal Anode_tb      : STD_LOGIC_VECTOR (3 downto 0);

    constant clk_period : time := 10 ns;

begin
    uut: MainProgram
        port map (
            Clk         => Clk_tb,
            Reset       => Reset_tb,
            Seg_7_Out   => Seg_7_Out_tb,
            Reg_7_DOut   => Reg_7_Out_tb,
            Overflow    => Overflow_tb,
            Zero        => Zero_tb,
            Anode       => Anode_tb
        );

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

    -- Stimulus process
process
```

```
    begin
        Reset_tb <= '1';
        wait for 30 ns;
        Reset_tb <= '0'; -- Release reset

        wait for 2000 ns;

        wait;
    end process;
end Behavioral;
```

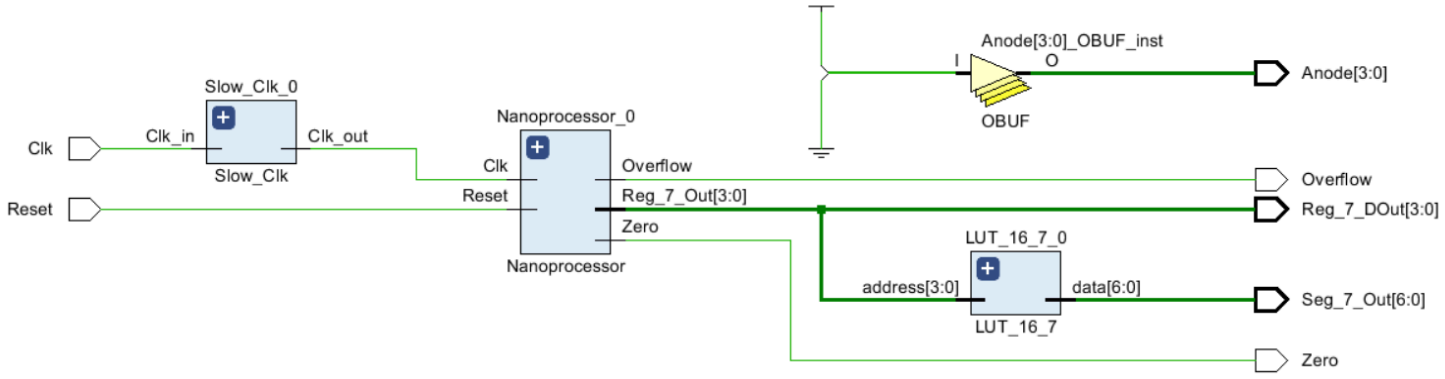


Constraint File

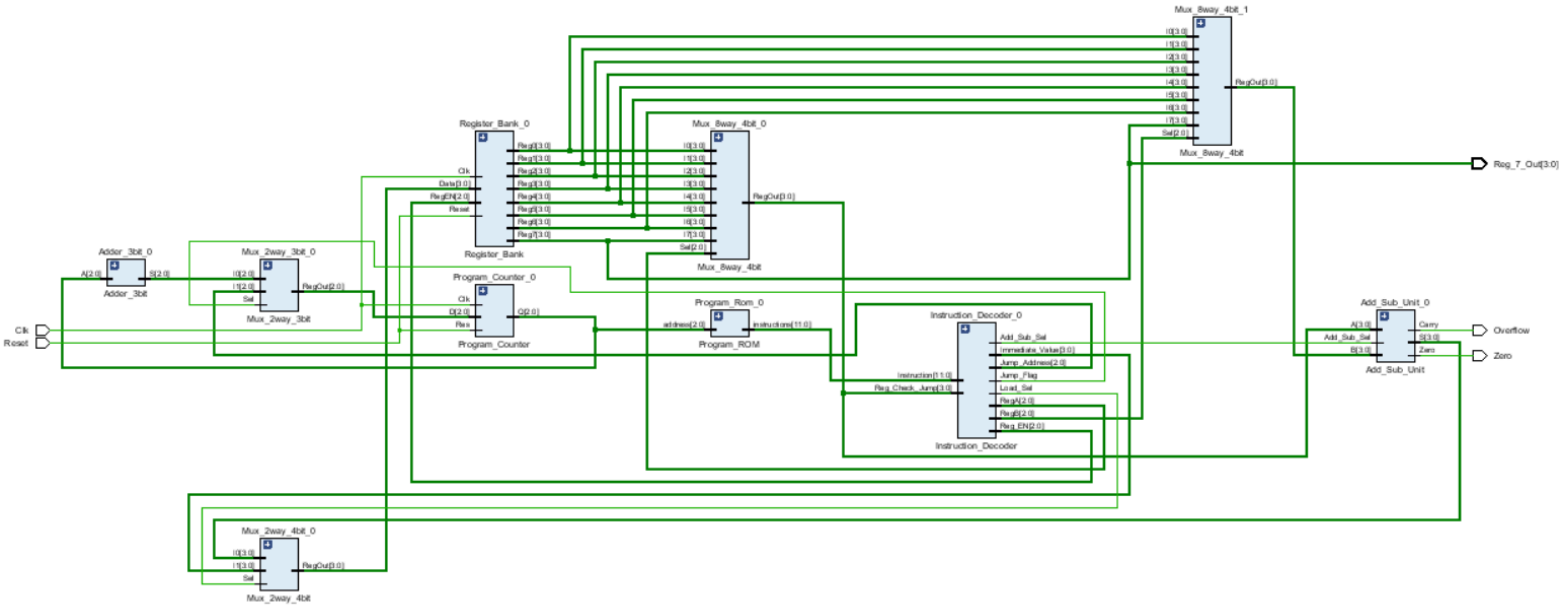
```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports Clk]
      set_property IOSTANDARD LVCMOS33 [get_ports Clk]
      create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
 [get_ports Clk]

# LEDs
set_property PACKAGE_PIN U16 [get_ports {Reg_7_DOut[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Reg_7_DOut[0]}]
set_property PACKAGE_PIN E19 [get_ports {Reg_7_DOut[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Reg_7_DOut[1]}]
set_property PACKAGE_PIN U19 [get_ports {Reg_7_DOut[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Reg_7_DOut[2]}]
set_property PACKAGE_PIN V19 [get_ports {Reg_7_DOut[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Reg_7_DOut[3]}]

set_property PACKAGE_PIN P1 [get_ports {Overflow}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Overflow}]
set_property PACKAGE_PIN L1 [get_ports {Zero}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Zero}]


#7 segment display
set_property PACKAGE_PIN W7 [get_ports {Seg_7_Out[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seg_7_Out[0]}]
set_property PACKAGE_PIN W6 [get_ports {Seg_7_Out[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seg_7_Out[1]}]
set_property PACKAGE_PIN U8 [get_ports {Seg_7_Out[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seg_7_Out[2]}]
set_property PACKAGE_PIN V8 [get_ports {Seg_7_Out[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seg_7_Out[3]}]
set_property PACKAGE_PIN U5 [get_ports {Seg_7_Out[4]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seg_7_Out[4]}]
set_property PACKAGE_PIN V5 [get_ports {Seg_7_Out[5]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seg_7_Out[5]}]
set_property PACKAGE_PIN U7 [get_ports {Seg_7_Out[6]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seg_7_Out[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]
```

Elaborated Schematic Designs

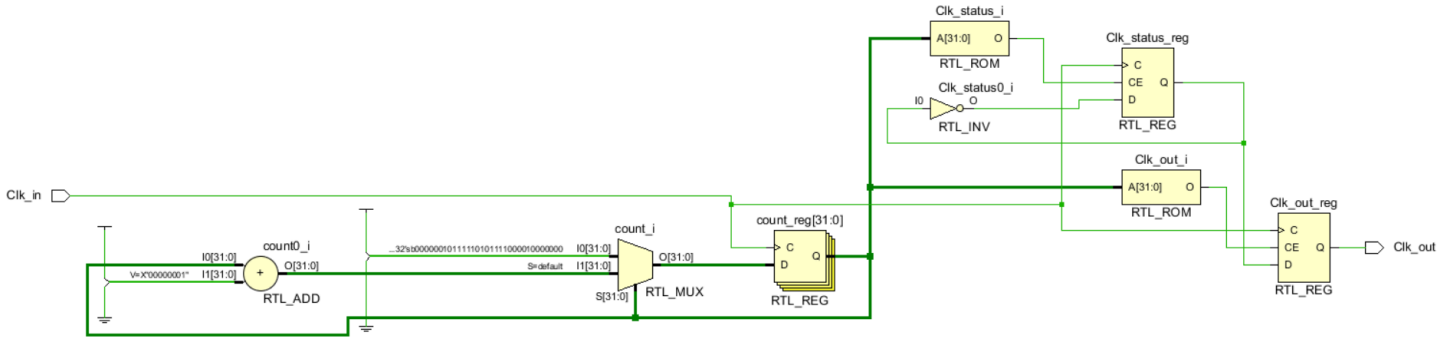
Main Program



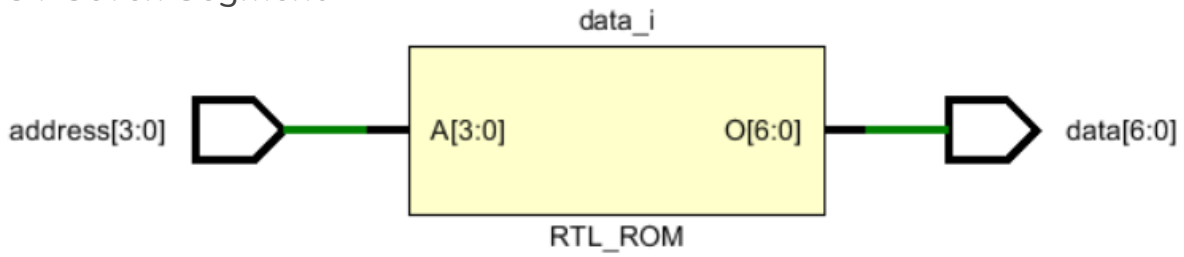
Nanoprocessor



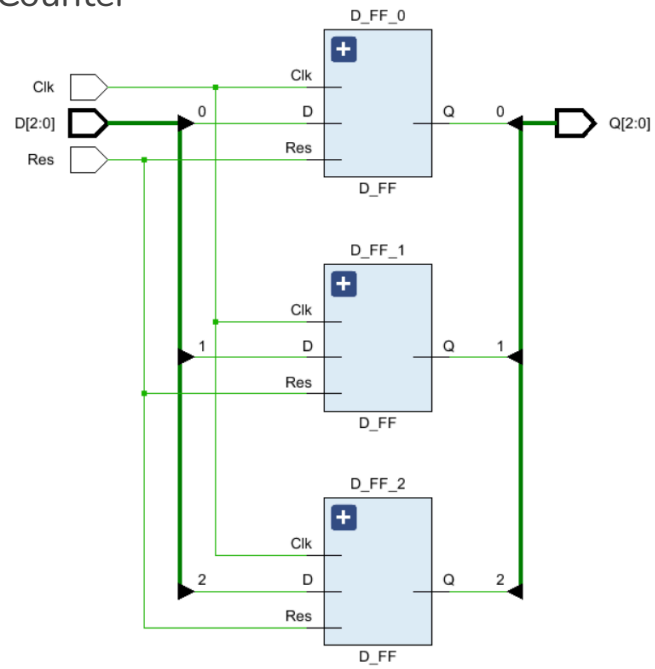
Slow Clock



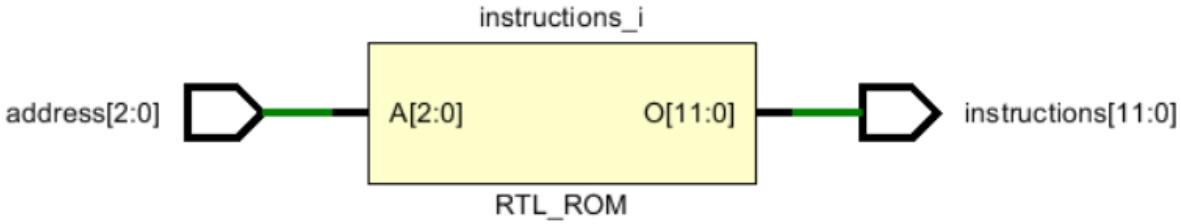
LUT Seven Segment



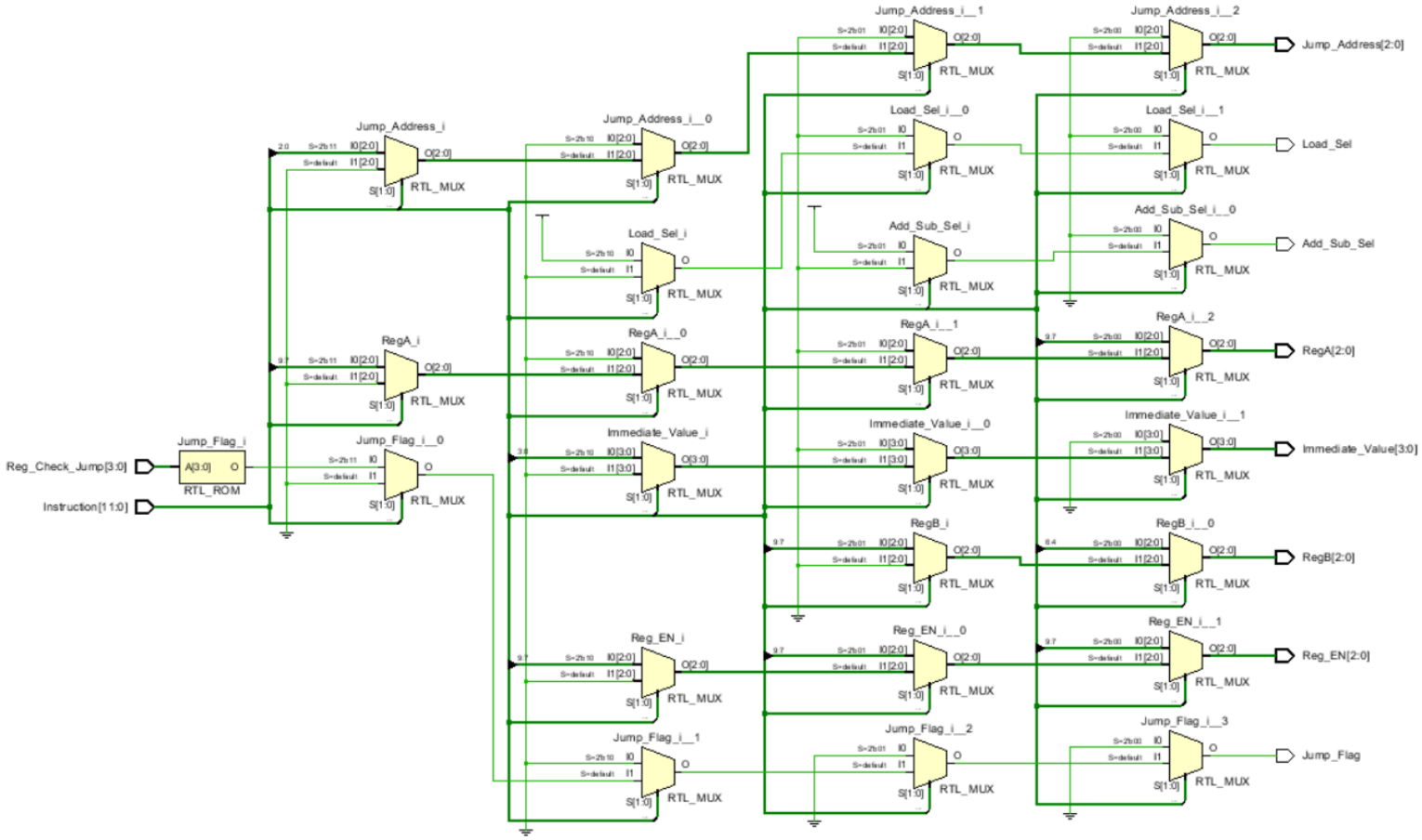
Program Counter



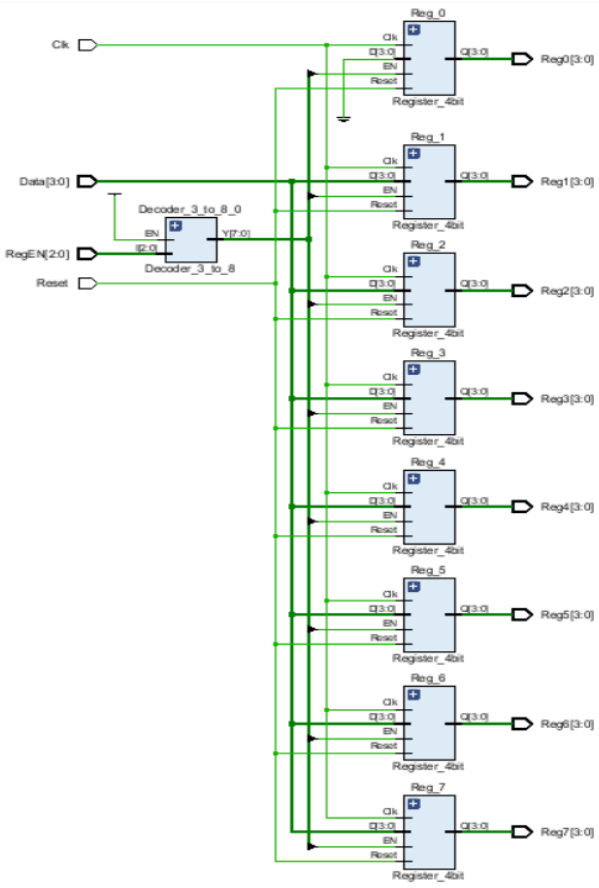
Program ROM



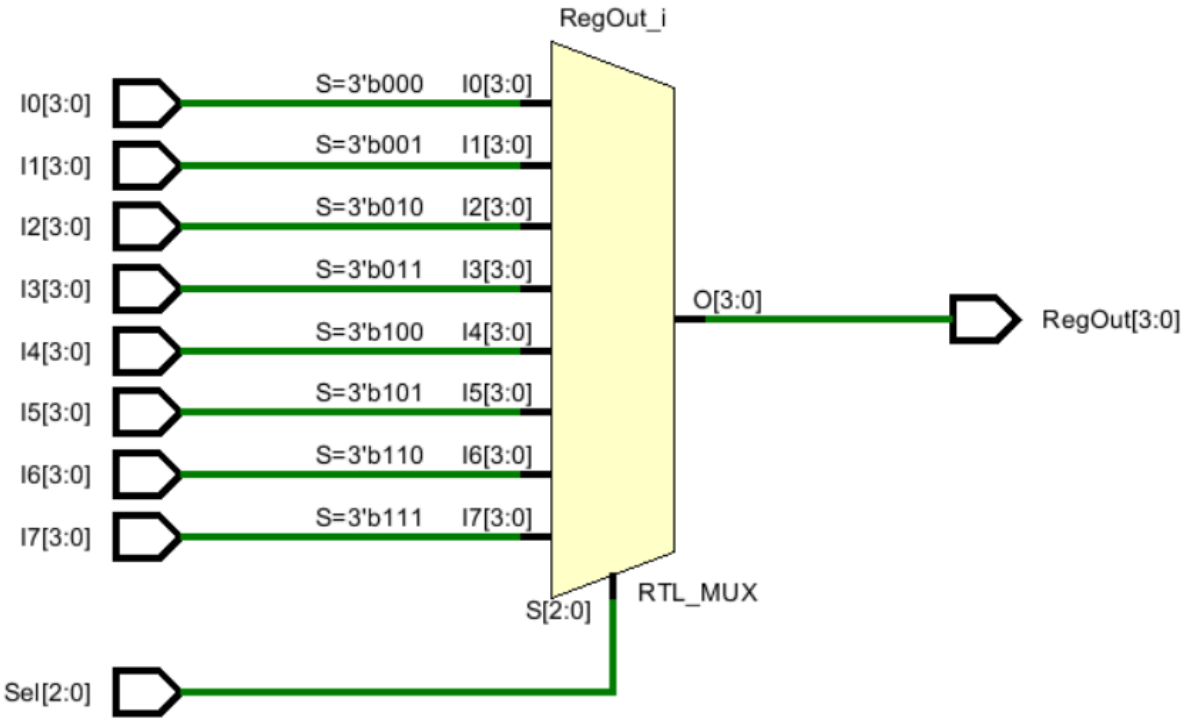
Instruction Decoder



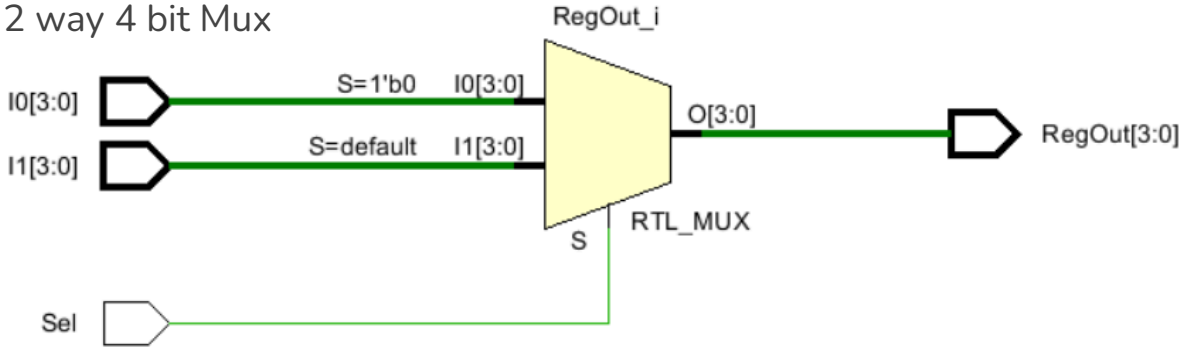
Register Bank



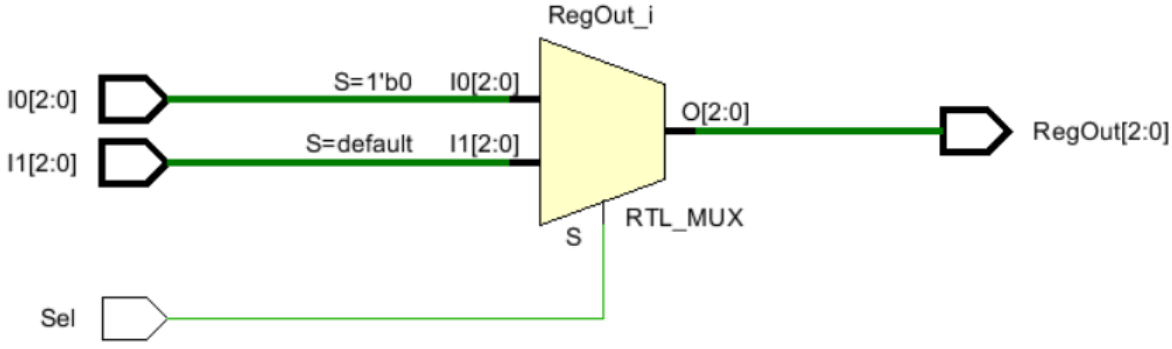
8 way 4 bit Mux



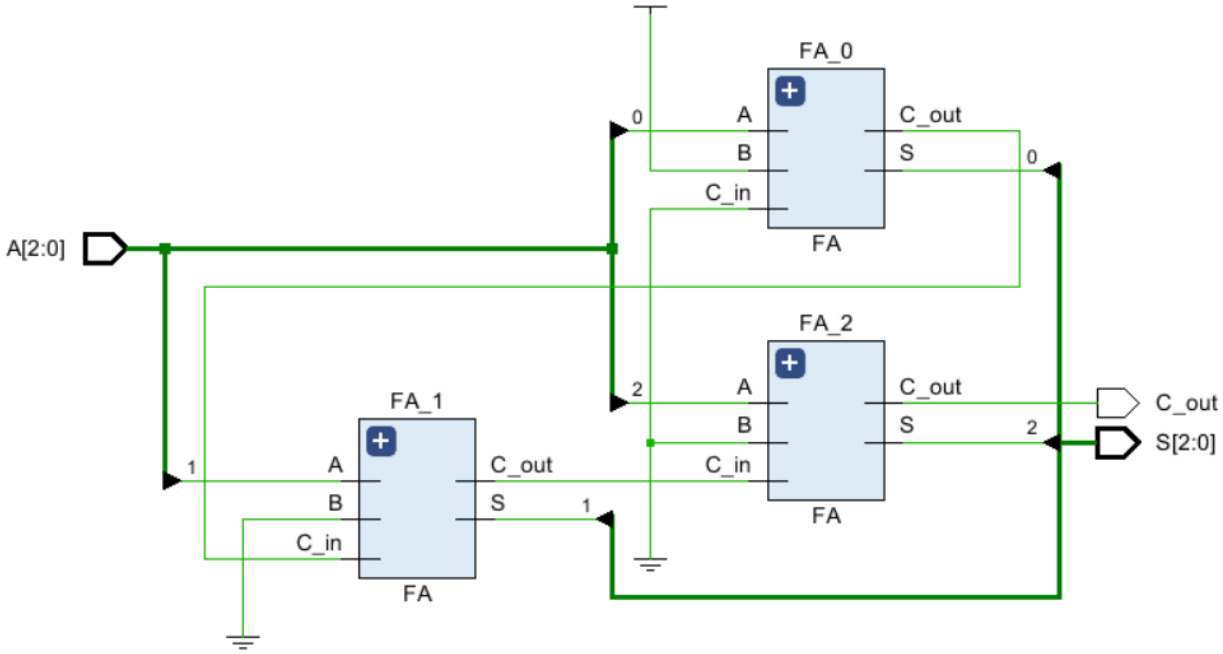
2 way 4 bit Mux



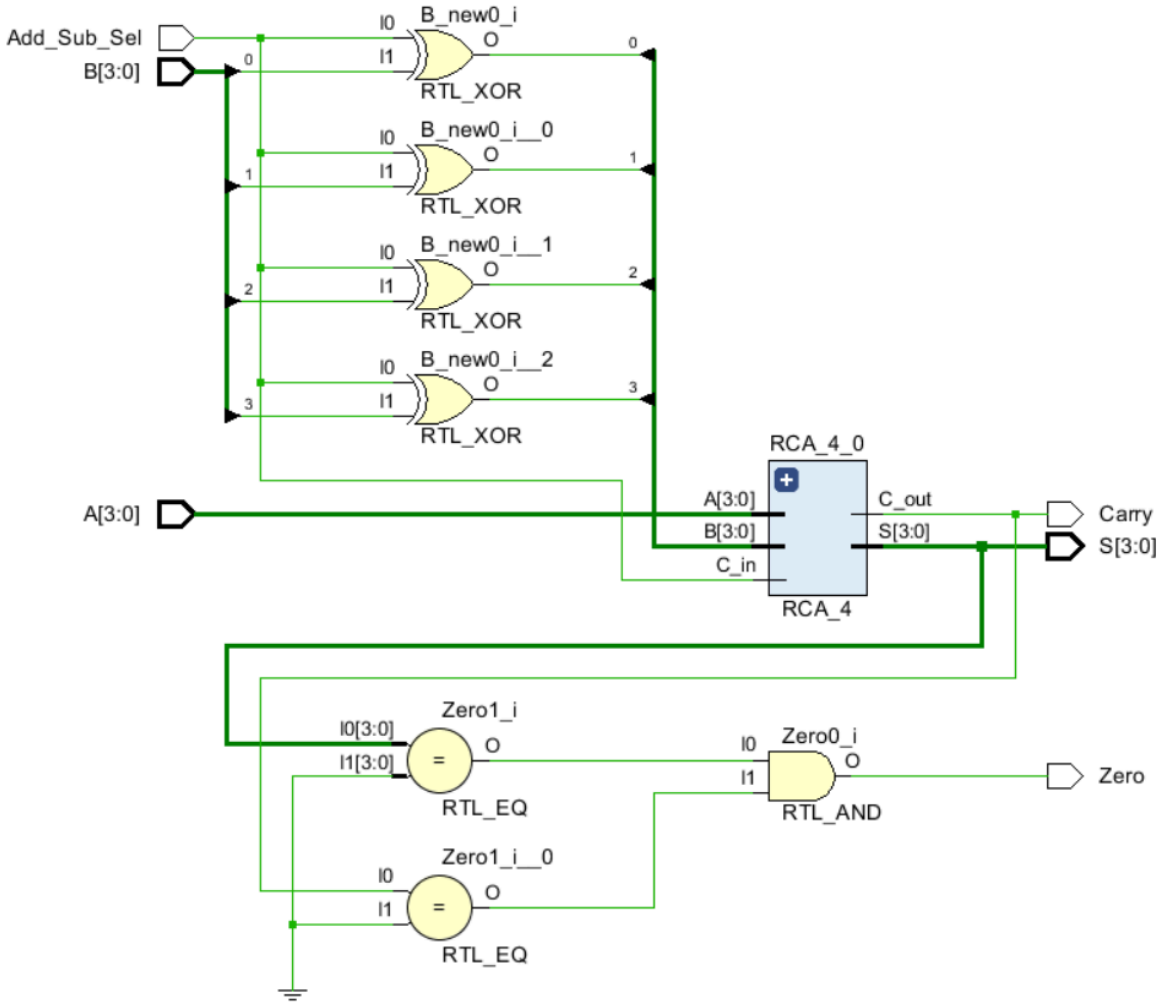
2 way 3 bit Mux



3 bit Adder



Add Sub Unit

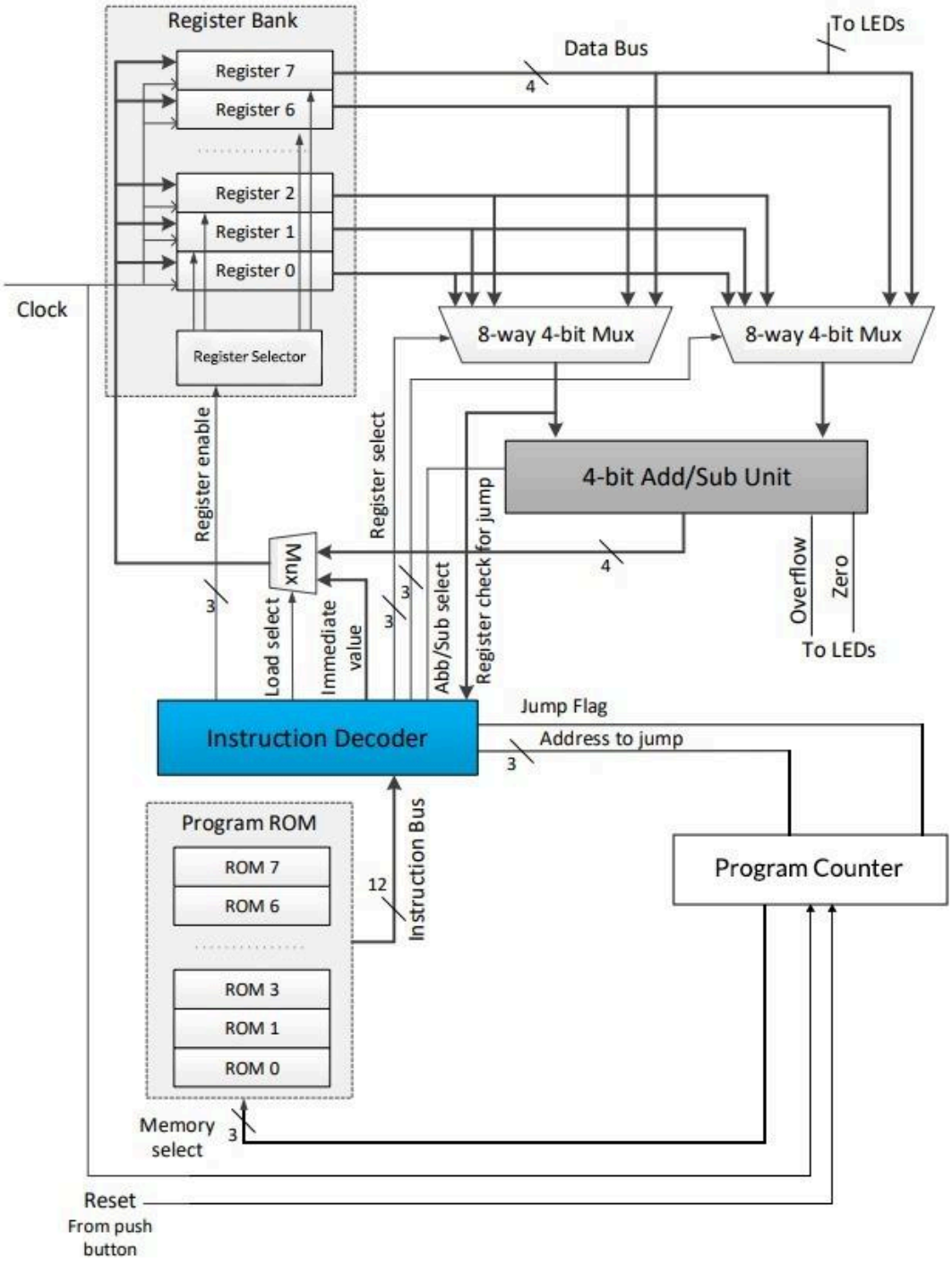


Optimized Nano Processor

1. **Modular Design** - Components were designed in a modular fashion, promoting clean separation of functionality. This allows for efficient resource allocation, easier debugging, and reusability across different parts of the nanoprocessor system.
2. **Reduced LUT count** - The overall LUT count was minimized through component reduction and reuse. The Multiplexers were replaced by if-else and case statements to simplify logic, and removed the unnecessary or duplicate logic blocks.
3. **Use of IP catalog** - The Add/Sub Unit was implemented using IP Catalog, ensuring optimized performance and area efficiency.
   The Program Counter was implemented using a binary counter from the IP Catalog as well, eliminating the need for a separate 3-bit adder and D flip flops.
4. **Efficient Clock Management** - Instead of implementing the Slow Clock by counting rising edges manually, we used a binary counter from the IP Catalog, significantly reducing LUT usage.
5. **Register Bank Implementation** - Created a compact register file using an array based structure. A single synchronous process manages write logic, reset, and clock, replacing eight separate 4-bit registers.

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	DSPs (90)	Bonded IOB (106)
MainProgram	16	12	7	16	2	3	19
Nanoprocessor_0 (Na...	16	12	7	16	2	2	0
Add_Sub_Unit_0 (A...	1	0	1	1	0	1	0
Program_Counter_...	11	0	3	11	0	1	0
Register_Bank_0 (...	4	12	7	4	0	0	0
Slow_Clk_0 (Slow_Clk)	0	0	0	0	0	1	0

High Level Diagram



Nano Processor with Additional Features

1. **Extended Register Size**:
   The nanoprocessor's original 4 bit register has been upgraded to a 12 bit register. This enhancement significantly increases the amount of data that can be stored and processed, enabling the system to handle larger numbers and more complex operations.

2. **Additional Instructions**:
   The instruction set has been expanded to include four new operations: multiplication, division, mod, comparison, and even/odd checking. These additions provide the processor with a wider range of computational capabilities, making it more versatile and suitable for advanced tasks beyond basic arithmetic and logic.

3. **Output of All 8 Registers**:
   The processor now displays the contents of all 8 internal registers. This is achieved by using 3 input switches to select the register number enabling users to view one register at a time on the output display, allowing full visibility into its internal state for debugging and analysis.

4. **Full LED Segment Utilization**:
   All four 7 segment LED displays are now actively used in the system. The leftmost segment is dedicated to showing the current value of the program counter, offering real-time visibility into the execution flow of instructions. This feature is particularly useful for debugging and monitoring the processor's state.

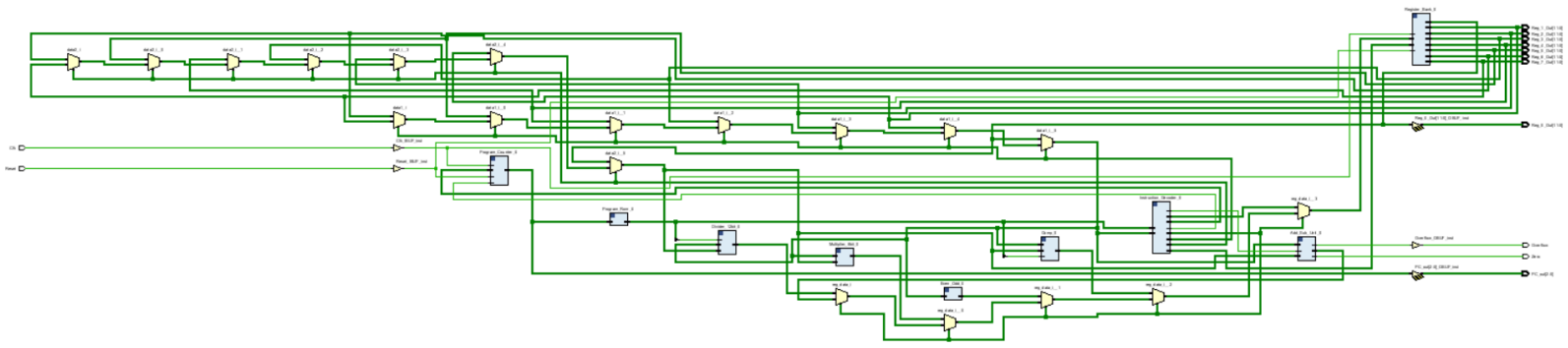
5. **Decimal Representation Display**:
   The processor can now display values in decimal format, improving the user experience by making numerical outputs more intuitive and easier to understand without manual conversion from binary or hexadecimal.

6. **Two's Complement Representation**:
   To support signed numbers, the nanoprocessor has been updated to represent values using two's complement format. This allows it to correctly handle negative numbers in arithmetic operations and display them accurately on the output.

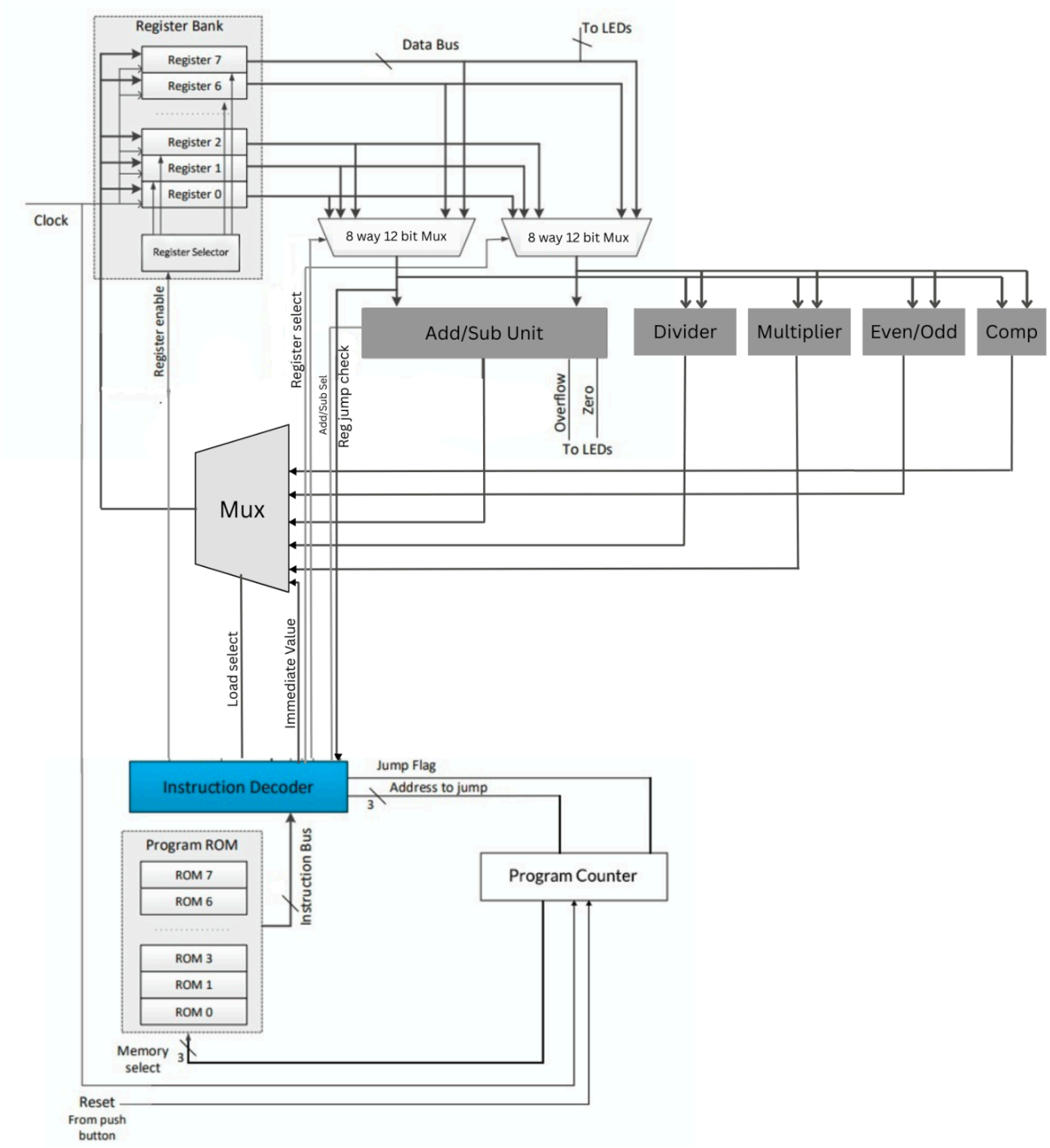
7. **Manual Clock Control**:
   A manual control button has been introduced to override the automatic clock. This feature enables the user to halt the processor's clock and step through instructions one at a time, offering precise control for debugging and educational demonstrations of how the processor executes each instruction.

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	DSPs (90)	Bonded IOB (106)	BUFGCTRL (32)
MainProgram	1974	84	11	616	1974	4	4	35	2
Nanoprocessor_0 (Na...	1605	48	11	523	1605	0	3	0	0
Add_Sub_Unit_0 (A...	3	0	0	1	3	0	1	0	0
Comp_0 (Comp)	0	0	0	2	0	0	0	0	0
Instruction_Decode...	0	0	0	1	0	0	0	0	0
Multiplier_6bit_0 (M...	0	0	0	0	0	0	1	0	0
Program_Counter_...	381	0	0	133	381	0	1	0	0
Register_Bank_0 (...	1227	48	11	436	1227	0	0	0	0
Slow_Clk_0 (Slow_Clk)	1	0	0	1	1	0	1	0	0

Additional Feature Nanoprocessor Elaborated Design



High Level Diagram



Instructions Format

ADD - 000 AAA BBB 000000000000
NEG - 001 RRR 000 000000000000
MOVI- 010 RRR 000 ddddddddddd
JZR - 011 RRR ddddddddddd aaa

COMP- 100 AAA BBB 000000000001 1-high, 0-low
EVEN/ODD - 101 RRR 000 000000000000
MUL - 110 AAA BBB 000000000000
DIV - 111 AAA BBB 000000000000 - Integer Division
111 AAA BBB 000000000001 - Mod (Remainder)

Program ROM Design Source

```
entity Program_ROM is
    Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
           instructions : out STD_LOGIC_VECTOR (20 downto 0));
end Program_ROM;

architecture Behavioral of Program_ROM is

type rom_type is array (0 to 7) of std_logic_vector(20 downto 0);
    signal ROM : rom_type := (
        "01000100000111110100O", --MOVI R1, 3E8  1000
        "01001000000000001111", --MOVI R2, F      15
        "01001100000011111010", --MOVI R3, FA    250
        "01010000000000111101", --MOVI R4, 3d     61
        "01010100010111111111",-- MOVI R5, AFF   2815  /  -1281
        "11001010000000000000",-- MUL R2,R4      Hex=393
        "11101100000000000000",-- DIV R1,R4      Q Hex=10/ R Hex=18
        "01110000000111101111" -- JUMP 111 if R4 is 3d
    );

begin

instructions <= ROM(to_integer(unsigned(address)));
```

Instruction Decoder Design Source

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

entity Instruction_Decoder is
    Port ( Instruction : in STD_LOGIC_VECTOR (20 downto 0);
           Reg_Check_Jump : in STD_LOGIC_VECTOR (11 downto 0);
           Add_Sub_Sel : out STD_LOGIC;
           RegA: out STD_LOGIC_VECTOR (2 downto 0);
           RegB : out STD_LOGIC_VECTOR (2 downto 0);
           Immediate_Value : out STD_LOGIC_VECTOR (11 downto 0);
           Load_Sel : out STD_LOGIC_VECTOR (2 downto 0);
           Reg_EN : out STD_LOGIC_VECTOR (2 downto 0);
           Jump_Flag : out STD_LOGIC;
           Jump_Address : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is
    signal Operator : std_logic_vector(2 downto 0);
begin

    Operator <= Instruction(20 downto 18);  -- OPCODE bits

    process (Operator, Instruction, Reg_Check_Jump)
    begin
        -- Default values
        Add_Sub_Sel      <= '1';
        Jump_Flag        <= '0';
        Load_Sel         <= "000";
        RegA             <= "000";
        RegB             <= "000";
        Immediate_Value  <= (others => '0');
        Reg_EN           <= "000";
        Jump_Address     <= "000";

        case Operator is

            when "000" =>  -- ADD
                RegA   <= Instruction(17 downto 15);
                RegB   <= Instruction(14 downto 12);
                Reg_EN <= Instruction(17 downto 15);

            when "001" =>  -- NEG
                RegB         <= Instruction(17 downto 15);
                Reg_EN       <= Instruction(17 downto 15);
                Add_Sub_Sel  <= '0';

            when "010" =>  -- MOVI
                Reg_EN          <= Instruction(17 downto 15);
                Immediate_Value <= Instruction(11 downto 0);
                Load_Sel        <= "001";
```

```
            when "011" =>  -- JZR
                RegA         <= Instruction(17 downto 15);
                Jump_Address <= Instruction(2 downto 0);
                if Reg_Check_Jump = Instruction(14 downto 3) then
                    Jump_Flag <= '1';
                end if;

            when "100" =>  -- COMP
                RegA   <= Instruction(17 downto 15);
                RegB   <= Instruction(14 downto 12);
                Reg_EN <= Instruction(17 downto 15);
                Load_Sel <= "010";

            when "101" =>  -- EVEN/ODD
                RegA   <= Instruction(17 downto 15);
                Reg_EN <= Instruction(17 downto 15);
                Load_Sel <= "011";

            when "110" =>  -- MUL
                RegA   <= Instruction(17 downto 15);
                RegB   <= Instruction(14 downto 12);
                Reg_EN <= Instruction(17 downto 15);
                Load_Sel <= "100";

            when "111" =>  -- DIV
                RegA   <= Instruction(17 downto 15);
                RegB   <= Instruction(14 downto 12);
                Reg_EN <= Instruction(17 downto 15);
                Load_Sel <= "101";

            when others =>
                null;
        end case;
    end process;

end Behavioral;
```

Multiplier 6 bit Design Source

```
entity Multiplier_6bit is
    Port (
        A        : in  STD_LOGIC_VECTOR(11 downto 0);
        B        : in  STD_LOGIC_VECTOR(11 downto 0);
        Product  : out STD_LOGIC_VECTOR(11 downto 0)
    );

    attribute use_dsp : string;
    attribute use_dsp of Multiplier_6bit : entity is "yes";
end Multiplier_6bit;

architecture Behavioral of Multiplier_6bit is
begin

    process (A, B)
        variable A_6bit : unsigned(5 downto 0);
        variable B_6bit : unsigned(5 downto 0);
        variable Result : unsigned(11 downto 0);
    begin
        A_6bit := unsigned(A(5 downto 0));  -- extract 2-bit value
        B_6bit := unsigned(B(5 downto 0));  -- extract 2-bit value
        Result := A_6bit * B_6bit;
        Product <= std_logic_vector(Result);  -- assign to output
    end process;

end Behavioral;
```

Even Odd Checker Design Source

```
entity Even_Odd is
    Port (
        Data_In         : in  STD_LOGIC_VECTOR(11 downto 0);
        Is_Even         : out STD_LOGIC_VECTOR(11 downto 0)
    );
end Even_Odd;

architecture Behavioral of Even_Odd is
begin
    process(Data_In)
    begin
        if Data_In(0) = '0' then
            Is_Even <= "000000000001";  -- Even number
        else
            Is_Even <= "000000000000";  -- Odd number
        end if;
    end process;
end Behavioral;
```

Divider 12 bit Design Source

```
entity Divider_12bit is
    Port (
        D_or_R : in std_logic;
        Dividend  : in  STD_LOGIC_VECTOR(11 downto 0);
        Divisor   : in  STD_LOGIC_VECTOR(11 downto 0);
        Quotient_Remainder  : out STD_LOGIC_VECTOR(11 downto 0)
    );

    attribute use_dsp : string;
    attribute use_dsp of Divider_12bit : entity is "yes";
end Divider_12bit;

architecture Behavioral of Divider_12bit is
begin
    process (Dividend, Divisor)
        variable A : unsigned(11 downto 0);
        variable B : unsigned(11 downto 0);
        variable Q : unsigned(11 downto 0);
        variable R : unsigned(11 downto 0);
    begin
        A := unsigned(Dividend);
        B := unsigned(Divisor);

        if B /= 0 then
            Q := A / B;
            R := A mod B;
        else
            Q := (others => '0');
            R := (others => '0');
        end if;

        if D_or_R = '1' then
            Quotient_Remainder <= std_logic_vector(R);
        else
            Quotient_Remainder <= std_logic_vector(Q);
        end if;
    end process;
end Behavioral;
```

Comparator Design Source

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

entity Comp is
    Port (
        H_or_L   : in  STD_LOGIC;
        A        : in  STD_LOGIC_VECTOR(11 downto 0);
        B        : in  STD_LOGIC_VECTOR(11 downto 0);
        Max_Out  : out STD_LOGIC_VECTOR(11 downto 0)
    );
end Comp;

architecture Behavioral of Comp is
begin
    process (A, B, H_or_L)
    begin
        if H_or_L = '1' then
            -- Output the higher value
            if unsigned(A) >= unsigned(B) then
                Max_Out <= A;
            else
                Max_Out <= B;
            end if;
        else
            -- Output the lower value
            if unsigned(A) <= unsigned(B) then
                Max_Out <= A;
            else
                Max_Out <= B;
            end if;
        end if;
    end process;
end Behavioral;
```

Nanoprocessor Design Source

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

entity Nanoprocessor is
    Port (
        Clk        : in  STD_LOGIC;
        Reset      : in  STD_LOGIC;
        Overflow   : out STD_LOGIC;
        Zero       : out STD_LOGIC;
        Reg_7_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_0_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_1_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_2_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_3_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_4_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_5_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_6_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        PC_out     : out STD_LOGIC_VECTOR (2 downto 0)
    );
end Nanoprocessor;

architecture Behavioral of Nanoprocessor is

    -- Components updated for 12-bit
    component Register_Bank
        Port (
            Clk    : in  STD_LOGIC;
            RegEN  : in  STD_LOGIC_VECTOR (2 downto 0);
            Data   : in  STD_LOGIC_VECTOR (11 downto 0);
            Reset  : in  STD_LOGIC;
            Reg0   : out STD_LOGIC_VECTOR (11 downto 0);
            Reg1   : out STD_LOGIC_VECTOR (11 downto 0);
            Reg2   : out STD_LOGIC_VECTOR (11 downto 0);
            Reg3   : out STD_LOGIC_VECTOR (11 downto 0);
            Reg4   : out STD_LOGIC_VECTOR (11 downto 0);
            Reg5   : out STD_LOGIC_VECTOR (11 downto 0);
            Reg6   : out STD_LOGIC_VECTOR (11 downto 0);
            Reg7   : out STD_LOGIC_VECTOR (11 downto 0)
        );
    end component;

    component Program_Counter
        Port (
            D   : in  STD_LOGIC_VECTOR(2 downto 0);
            Res : in  STD_LOGIC;
            Clk : in  STD_LOGIC;
            Sel : in  STD_LOGIC;
            Q   : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;
```

```
    component Program_ROM
        Port (
            address      : in  STD_LOGIC_VECTOR(2 downto 0);
            instruction  : out STD_LOGIC_VECTOR(20 downto 0)  -- Keep as-is
unless instruction width increases
        );
    end component;

    component Instruction_Decoder
        Port (
            Instruction     : in  STD_LOGIC_VECTOR(20 downto 0);
            Reg_Check_Jump  : in  STD_LOGIC_VECTOR(11 downto 0);
            Add_Sub_Sel     : out STD_LOGIC;
            RegA            : out STD_LOGIC_VECTOR(2 downto 0);
            RegB            : out STD_LOGIC_VECTOR(2 downto 0);
            Immediate_Value : out STD_LOGIC_VECTOR(11 downto 0);
            Load_Sel        : out STD_LOGIC_VECTOR(2 downto 0);
            Reg_EN          : out STD_LOGIC_VECTOR(2 downto 0);
            Jump_Flag       : out STD_LOGIC;
            Jump_Address    : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;

    component Add_Sub_Unit
        Port (
            A           : in  STD_LOGIC_VECTOR(11 downto 0);
            B           : in  STD_LOGIC_VECTOR(11 downto 0);
            Add_Sub_Sel : in  STD_LOGIC;
            S           : out STD_LOGIC_VECTOR(11 downto 0);
            Carry       : out STD_LOGIC;
            Zero        : out STD_LOGIC
        );
    end component;

    component Even_Odd
        Port (
            Data_In : in  STD_LOGIC_VECTOR(11 downto 0);
            Is_Even : out STD_LOGIC_VECTOR(11 downto 0)
        );
    end component;

    component Comp
        Port (
            H_or_L   : in  STD_LOGIC;
            A       : in  STD_LOGIC_VECTOR(11 downto 0);
            B       : in  STD_LOGIC_VECTOR(11 downto 0);
            Max_Out : out STD_LOGIC_VECTOR(11 downto 0)
        );
    end component;

    component Multiplier_6bit
        Port (
            A       : in  STD_LOGIC_VECTOR(11 downto 0);
            B       : in  STD_LOGIC_VECTOR(11 downto 0);
```

```
            Product : out STD_LOGIC_VECTOR(11 downto 0)
        );
    end component;

    component Divider_12bit
        Port (
            D_or_R             : in  STD_LOGIC;
            Dividend           : in  STD_LOGIC_VECTOR(11 downto 0);
            Divisor            : in  STD_LOGIC_VECTOR(11 downto 0);
            Quotient_Remainder : out STD_LOGIC_VECTOR(11 downto 0)
        );
    end component;

    -- Internal Signals
    signal sub, jflag : STD_LOGIC;
    signal load_sel   : STD_LOGIC_VECTOR(2 downto 0);
    signal reg_en     : STD_LOGIC_VECTOR(2 downto 0);
    signal PCaddress, jump_add : STD_LOGIC_VECTOR(2 downto 0);
    signal mux_1, mux_2 : STD_LOGIC_VECTOR(2 downto 0);
    signal Instruction : STD_LOGIC_VECTOR(20 downto 0);
    signal IVal, result, reg_data, EOVal, MaxVal, product, quotient_remainder :
STD_LOGIC_VECTOR(11 downto 0);
    signal r0, r1, r2, r3, r4, r5, r6, r7 : STD_LOGIC_VECTOR(11 downto 0);
    signal data1, data2 : STD_LOGIC_VECTOR(11 downto 0);

begin

        -- Program Counter
    Program_Counter_0 : Program_Counter
        port map (
            D    => jump_add,
            Res  => Reset,
            Clk  => Clk,
            Sel  => jflag,
            Q    => PCaddress
        );

    Program_Rom_0 : Program_ROM
        port map (
            address      => PCaddress,
            instructions => Instruction
        );

    Instruction_Decoder_0 : Instruction_Decoder
        port map (
            Instruction     => Instruction,
            Reg_Check_Jump  => data1,
            Add_Sub_Sel     => sub,
            RegA            => mux_1,
            RegB            => mux_2,
            Immediate_Value => IVal,
            Load_Sel        => load_sel,
            Reg_EN          => reg_en,
            Jump_Flag       => jflag,
            Jump_Address    => jump_add
```

```
        );

    Register_Bank_0 : Register_Bank
        port map (
            Clk    => Clk,
            RegEN  => reg_en,
            Data   => reg_data,
            Reset  => Reset,
            Reg0   => r0, Reg1 => r1, Reg2 => r2, Reg3 => r3,
            Reg4   => r4, Reg5 => r5, Reg6 => r6, Reg7 => r7
        );

    -- MUX logic
    data1 <= r0 when mux_1 = "000" else r1 when mux_1 = "001" else
             r2 when mux_1 = "010" else r3 when mux_1 = "011" else
             r4 when mux_1 = "100" else r5 when mux_1 = "101" else
             r6 when mux_1 = "110" else r7;

    data2 <= r0 when mux_2 = "000" else r1 when mux_2 = "001" else
             r2 when mux_2 = "010" else r3 when mux_2 = "011" else
             r4 when mux_2 = "100" else r5 when mux_2 = "101" else
             r6 when mux_2 = "110" else r7;

    -- ALU Operations
    Add_Sub_Unit_0 : Add_Sub_Unit
        port map (
            A           => data1,
            B           => data2,
            Add_Sub_Sel => sub,
            S           => result,
            Carry       => Overflow,
            Zero        => Zero
        );

    Even_Odd_0 : Even_Odd
        port map (Data_In => data1, Is_Even => EOVal);

    Comp_0 : Comp
MaxVal);port map (H_or_L => Instruction(0), A => data1, B => data2, Max_Out =>

    Multiplier_6bit_0 : Multiplier_6bit
        port map (A => data1, B => data2, Product => product);

    Divider_12bit_0 : Divider_12bit
        port map (
            D_or_R => Instruction(0),
            Dividend => data1,
            Divisor  => data2,
            Quotient_Remainder => quotient_remainder
        );

    -- Result Selector
    reg_data <= IVal     when load_sel = "001" else
                MaxVal   when load_sel = "010" else
```

```
                EOVal    when load_sel = "011" else
                product  when load_sel = "100" else
                quotient_remainder when load_sel = "101" else
                result;

    -- Outputs
    Reg_0_Out <= r0; Reg_1_Out <= r1; Reg_2_Out <= r2; Reg_3_Out <= r3;
    Reg_4_Out <= r4; Reg_5_Out <= r5; Reg_6_Out <= r6; Reg_7_Out <= r7;
    PC_out    <= PCaddress;

end Behavioral;
```

Main Program Design Source

```
entity MainProgram is
    Port (  Clk : in STD_LOGIC;
            Reset : in STD_LOGIC;
            SW        : in STD_LOGIC_VECTOR(2 downto 0);
            Step_Mode_Switch : in STD_LOGIC;
            Step_Button : in STD_LOGIC;
            Mode_Switch : in STD_LOGIC;
            Twos_Complement_Switch : in STD_LOGIC;
            Seg_7_Out : out STD_LOGIC_VECTOR (7 downto 0);
            Reg_7_Out : out STD_LOGIC_VECTOR (11 downto 0);
            Overflow : out STD_LOGIC;
            Zero : out STD_LOGIC;
            Anode : out STD_LOGIC_VECTOR (3 downto 0));
end MainProgram;

architecture Behavioral of MainProgram is

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

    component Nanoprocessor
        Port (
        Clk        : in  STD_LOGIC;
        Reset      : in  STD_LOGIC;
        Overflow   : out STD_LOGIC;
        Zero       : out STD_LOGIC;
        Reg_7_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_0_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_1_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_2_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_3_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_4_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_5_Out  : out STD_LOGIC_VECTOR (11 downto 0);
        Reg_6_Out  : out STD_LOGIC_VECTOR (11 downto 0);
```

```
        PC_out     : out STD_LOGIC_VECTOR (2 downto 0)
    );
    end component;

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

    signal Display_out : STD_LOGIC_VECTOR (11 downto 0);
    signal Display_out0 : STD_LOGIC_VECTOR (11 downto 0);
    signal Display_out1 : STD_LOGIC_VECTOR (11 downto 0);
    signal Display_out2 : STD_LOGIC_VECTOR (11 downto 0);
    signal Display_out3 : STD_LOGIC_VECTOR (11 downto 0);
    signal Display_out4 : STD_LOGIC_VECTOR (11 downto 0);
    signal Display_out5 : STD_LOGIC_VECTOR (11 downto 0);
    signal Display_out6 : STD_LOGIC_VECTOR (11 downto 0);

    signal SlowClk : std_logic;

    signal count : integer := 1;
    signal LED_Counter : std_logic_vector(1 downto 0) := "00";

    signal pc_out : std_logic_vector(2 downto 0);
    signal seg_pc_in : STD_LOGIC_VECTOR (3 downto 0);
    signal selected_reg_out : STD_LOGIC_VECTOR(11 downto 0);

    signal address_in : std_logic_vector(3 downto 0);
    signal data_out : std_logic_vector(6 downto 0);

    signal EffectiveClk : STD_LOGIC := '0';
    signal Step_Button_prev : STD_LOGIC := '0';

    signal bcd_digits : STD_LOGIC_VECTOR(15 downto 0);

begin

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

    Nanoprocessor_0 : Nanoprocessor
        port map (
             Clk => EffectiveClk,
             Reset => Reset,
             Overflow => Overflow,
             Zero => Zero,
             Reg_7_Out => Display_out,
             Reg_0_Out => Display_out0,
             Reg_1_Out => Display_out1,
             Reg_2_Out => Display_out2,
             Reg_3_Out => Display_out3,
```

```
            Reg_4_Out => Display_out4,
            Reg_5_Out => Display_out5,
            Reg_6_Out => Display_out6,
            PC_out => pc_out);

     LUT_16_7_0 : LUT_16_7
        port map (
            address => address_in,
            data => data_out
            );

    seg_pc_in <= '0' & pc_out;

    -- Clock selection and step button edge detection process
    process(Clk)
    begin
        if rising_edge(Clk) then
            Step_Button_prev <= Step_Button;

            if Step_Mode_Switch = '0' then
                EffectiveClk <= SlowClk;
            else
                if (Step_Button = '1' and Step_Button_prev = '0') then
                    EffectiveClk <= '1';
                else
                    EffectiveClk <= '0';
                end if;
            end if;
        end if;
    end process;

     PROCESS(Clk)
          BEGIN
              IF rising_edge(Clk) THEN
                  count <= count + 1;
                  IF count = 100000 THEN
1);                   LED_Counter <= std_logic_vector(unsigned(LED_Counter) +
                      count <= 1;
                  END IF;
              END IF;
     END PROCESS;

Display_out4, Display_out5, Display_out6, Display_out7);
    process(SW, Display_out0, Display_out1, Display_out2, Display_out3,
       begin
           case SW is
               when "000" => selected_reg_out <= Display_out0;
               when "001" => selected_reg_out <= Display_out1;
               when "010" => selected_reg_out <= Display_out2;
               when "011" => selected_reg_out <= Display_out3;
               when "100" => selected_reg_out <= Display_out4;
               when "101" => selected_reg_out <= Display_out5;
               when "110" => selected_reg_out <= Display_out6;
               when "111" => selected_reg_out <= Display_out;
```

```
            when others => selected_reg_out <= "000000000000";
        end case;
    end process;

    Reg_7_Out <= selected_reg_out;

    process(selected_reg_out, Twos_Complement_Switch)
        variable binary_val : integer;
        variable signed_val : integer;
        variable temp_bcd : std_logic_vector(15 downto 0);
    begin
        if Twos_Complement_Switch = '1' then
            signed_val := to_integer(signed(selected_reg_out));
            if signed_val < 0 then
                binary_val := abs(signed_val);
            else
                binary_val := signed_val;
            end if;
        else

            binary_val := to_integer(unsigned(selected_reg_out));
        end if;


        temp_bcd := (others => '0');
10, 4)); temp_bcd(3 downto 0)   := std_logic_vector(to_unsigned(binary_val mod
10) mod 10, 4));temp_bcd(7 downto 4)   := std_logic_vector(to_unsigned((binary_val /
100) mod 10, 4));temp_bcd(11 downto 8)  := std_logic_vector(to_unsigned((binary_val /
1000, 4));temp_bcd(15 downto 12) := std_logic_vector(to_unsigned(binary_val /

        bcd_digits <= temp_bcd;
    end process;



    process(LED_Counter, selected_reg_out, seg_pc_in, Mode_Switch)
    begin
        if Mode_Switch = '1' then
            case LED_Counter is
                when "00" =>
                    Anode <= "1110";
                    address_in <= bcd_digits(3 downto 0);
                    Seg_7_Out <= '1' & data_out;

                when "01" =>
                    Anode <= "1101";
                    address_in <= bcd_digits(7 downto 4);
                    Seg_7_Out <= '1' & data_out;

                when "10" =>
                    Anode <= "1011";
                    address_in <= bcd_digits(11 downto 8);
                    Seg_7_Out <= '1' & data_out;
```

```
                when "11" =>
                    Anode <= "0111";
                    address_in <= bcd_digits(15 downto 12);
                    Seg_7_Out <= '1' & data_out;

                when others =>
                    Anode <= "1111";
                    Seg_7_Out <= "11111111";
            end case;
        else
            case LED_Counter is
                when "00" =>
                    Anode <= "1110";
                    address_in <= selected_reg_out(3 downto 0);
                    Seg_7_Out <= '1' & data_out;

                when "01" =>
                    Anode <= "1101";
                    address_in <= selected_reg_out(7 downto 4);
                    Seg_7_Out <= '1' & data_out;

                when "10" =>
                    Anode <= "1011";
                    address_in <= selected_reg_out(11 downto 8);
                    Seg_7_Out <= '1' & data_out;

                when "11" =>
                    Anode <= "0111";
                    address_in <= seg_pc_in;
                    Seg_7_Out <= '1' & data_out;

                when others =>
                    Anode <= "1111";
                    Seg_7_Out <= "11111111";
            end case;
        end if;
    end process;

end Behavioral;
```

Conclusion

This lab successfully demonstrated the design and implementation of a 4 bit nanoprocessor capable of executing a limited instruction set (MOVI, ADD, NEG, JZR) using VHDL. By integrating essential components such as the add/subtract unit, program counter, multiplexers, register bank, ROM, and instruction decoder, the nanoprocessor was able to run a simple assembly program on the BASYS 3 board. The use of LEDs and a 7 segment display provided real-time feedback for monitoring the processor's output and status flags. Through simulation, hardware testing, and thoughtful design choices like clock speed reduction and reset control, the project achieved both functional correctness and visibility into internal operations, fulfilling the core objectives of the exercise.