University of Moratuwa

Faculty of Engineering
Department of Computer Science and Engineering

Nanoprocessor Design Competition Lab Report

CS1050 – Computer Organization and Digital Design

Submitted by:

Ruththiragayan S.	230554G
Ahamed H.Y.	230021V
Asma A.R.	230060M
Rathnayake R.M.R.S.	230547M

Submission Date:

May 26, 2025

Contents

01

Introduction

In this lab, we built a 4-bit Nano Processor that can run four basic instructions efficiently. To do this, we used and improved several components we had already made before. These include a 4-bit Add/Subtract unit, a 3-bit adder, a 3-bit Program Counter (PC), different types of multiplexers, a Register Bank, Program ROM, an Instruction Decoder, a 7-Segment Display, and a slow clock. We used 3-bit, 4-bit, and 12-bit buses to connect different parts. This made our design simpler and reduced the number of wires needed.

Since the Nano Processor only understands machine language (binary), we wrote the instructions in binary form and stored them directly in the ROM (hardcoded). We used a slow clock to run the processor at a lower speed so we could clearly see each step of the program while it was running. We tested each part of our design using known inputs and outputs through simulation to make sure they worked correctly.

This was a team project with four members, and we shared the work equally. At first, we designed all the parts, and later we updated them as needed while combining everything into the full Nano Processor.

After finishing the lab, we successfully designed and built: a 4-bit unit that can add and negate numbers, multiplexers that can choose between multiple inputs, a Program ROM to store instructions, and a decoder to control the processor.

Finally, We tested everything through simulation and also by running it on the Basys3 board.

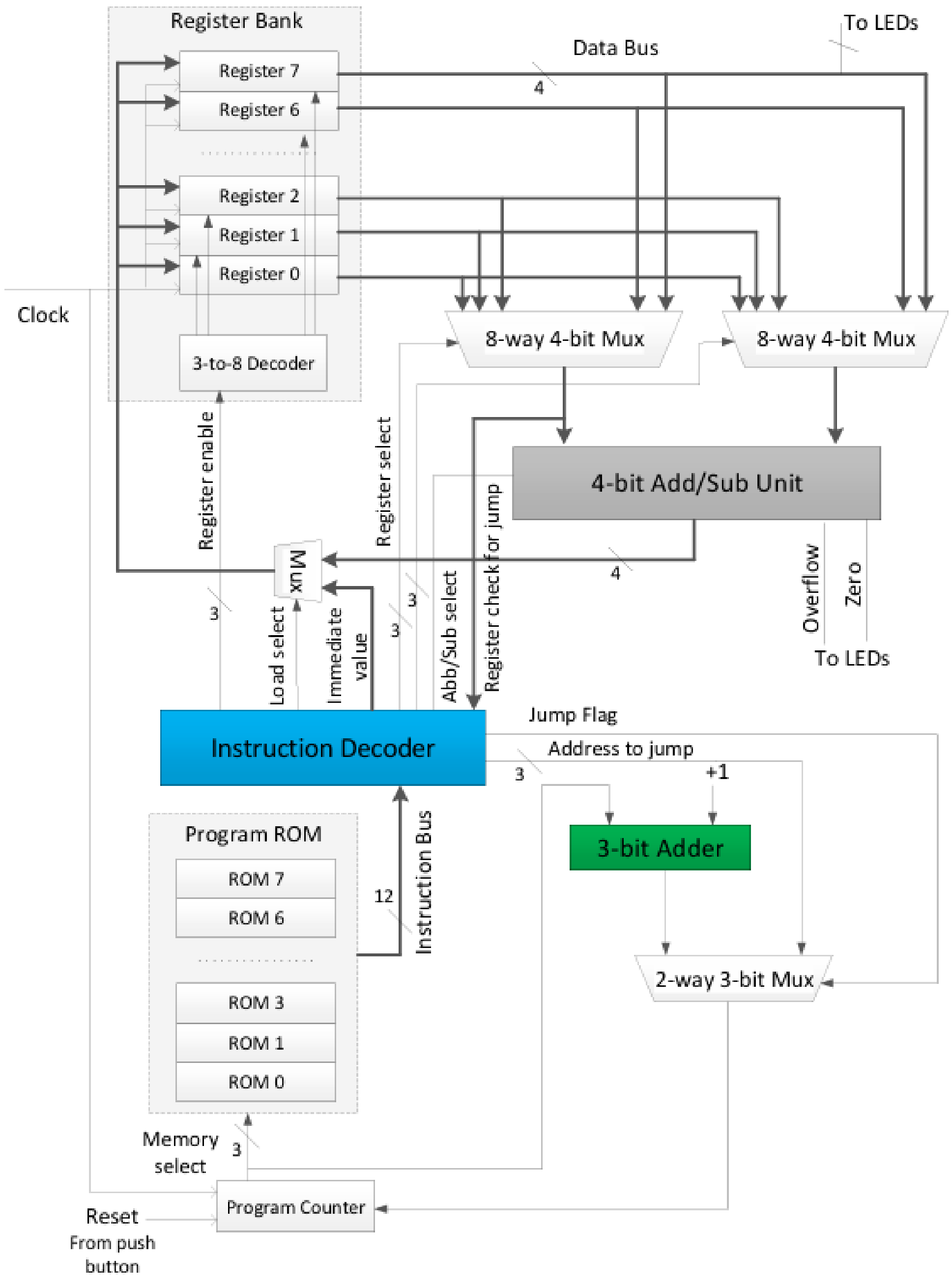


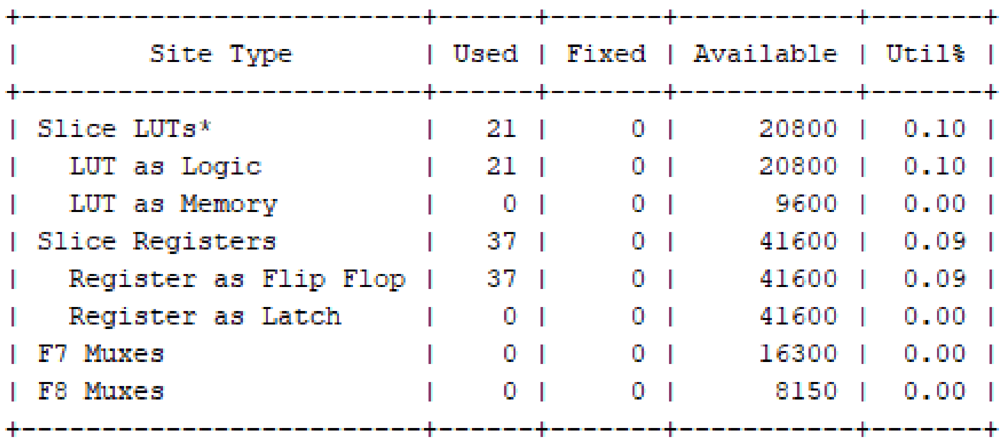
Figure 1.1: High-level diagram of the nanoprocessor

02

Optimization and Enhancement

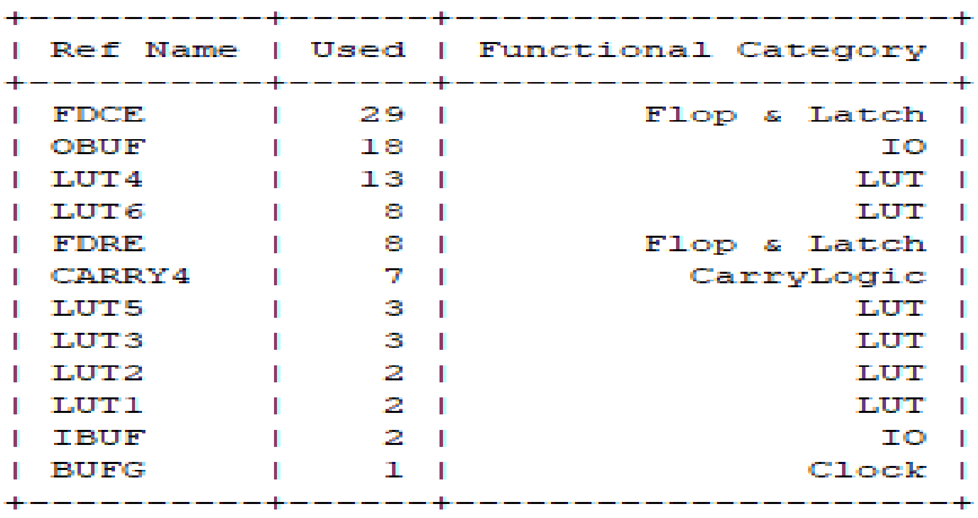
2.1 Optimized Design Primitives and Slice Logic

2.1.1 Slice Logic



```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   21 |     0 |     20800 |  0.10 |
|   LUT as Logic          |   21 |     0 |     20800 |  0.10 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         |   37 |     0 |     41600 |  0.09 |
|   Register as Flip Flop |   37 |     0 |     41600 |  0.09 |
|   Register as Latch     |    0 |     0 |     41600 |  0.00 |
| F7 Muxes                |    0 |     0 |     16300 |  0.00 |
| F8 Muxes                |    0 |     0 |      8150 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

2.1.2 Primitives



```
+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| FDCE     |   29 |        Flop & Latch |
| OBUF     |   18 |                  IO |
| LUT4     |   13 |                 LUT |
| LUT6     |    8 |                 LUT |
| FDRE     |    8 |        Flop & Latch |
| CARRY4   |    7 |          CarryLogic |
| LUT5     |    3 |                 LUT |
| LUT3     |    3 |                 LUT |
| LUT2     |    2 |                 LUT |
| LUT1     |    2 |                 LUT |
| IBUF     |    2 |                  IO |
| BUFG     |    1 |               Clock |
+----------+------+---------------------+
```

Advanced Features

2.2 Programmable Rom

The modified Nano-processor introduces several key improvements over the previous hard-coded version, significantly enhancing its flexibility, educational utility, and debugging capabilities.

2.2.1 Programmable ROM Architecture

The transition from hardcoded to programmable ROM provides critical advantages:

- **Dynamic Instruction Loading:**
  - Enables real-time programming via BASYS3 switches (13-bit `new_instruction` bus)
  - Eliminates the need for resynthesis when modifying programs
- **Programming Workflow:**
  The programming process begins by toggling the `Program_mode` button (bottom button) to enter Program Mode. If needed, the `ROM_reset` button (top button) can be pressed to clear all contents in the `Program_ROM`. The user then sets a 13-bit instruction using the 13 rightmost DIP switches. Once the instruction is configured, it is loaded into the current ROM address by pressing the `load_instruction` button (left button). The `NextROM` button (right button) is used to increment the ROM address, allowing the user to program the next instruction. After all desired instructions have been entered, pressing the `Program_mode` button again exits Program Mode and returns the system to normal execution.

2.3 Instruction Set Expansion

To enhance the capabilities of the Nano Processor, the instruction width is expanded from 12 bits to 13 bits. This change enabled the addition of new instructions such as `MOV` (register-to-register copy), `AND`, `OR`, and `XOR`, which were not possible within the original 12-bit encoding due to limited opcode space. The extra bit was added as the most significant bit (MSB) of the instruction. All legacy instructions from the original set are now prefixed with an MSB of `0`, preserving full backward compatibility, while the newly added instructions begin with an MSB of `1`.

These new instructions provide improved program efficiency and clarity. For example, operations like register transfer and bitwise logic, which previously required multiple instructions or indirect methods, can now be executed directly. Additionally, the overall logic design becomes more straightforward, and runtime performance improves due to fewer clock cycles per task.

13-bit Instruction Format:

```
12  11  10    9  8  7    6  5  4    3  2  1  0
   Opcode       Reg A      Reg B    Immediate
```

Additional Instructions:

Opcode (4-bit)	Instruction	Format	Description
1000	MOV	1 000 RaRaRa RbRbRb 0000	Moves value from Rb to Ra
1001	AND	1 001 RaRaRa RbRbRb 0000	Bitwise AND (Ra = Ra AND Rb)
1010	OR	1 010 RaRaRa RbRbRb 0000	Bitwise OR (Ra = Ra OR Rb)
1011	XOR	1 011 RaRaRa RbRbRb 0000	Bitwise XOR (Ra = Ra XOR Rb)

2.4 Register Monitoring

Real-Time Inspection: The 7-segment display allows users to view the current value of any register (R0 to R7) in real-time. The register to display is selected using the first 3 switches labeled `Reg_to_show`.

Debugging Aid: This live display acts as an on-board debugging tool, making it easy to monitor internal processor states without requiring external interfaces or software.

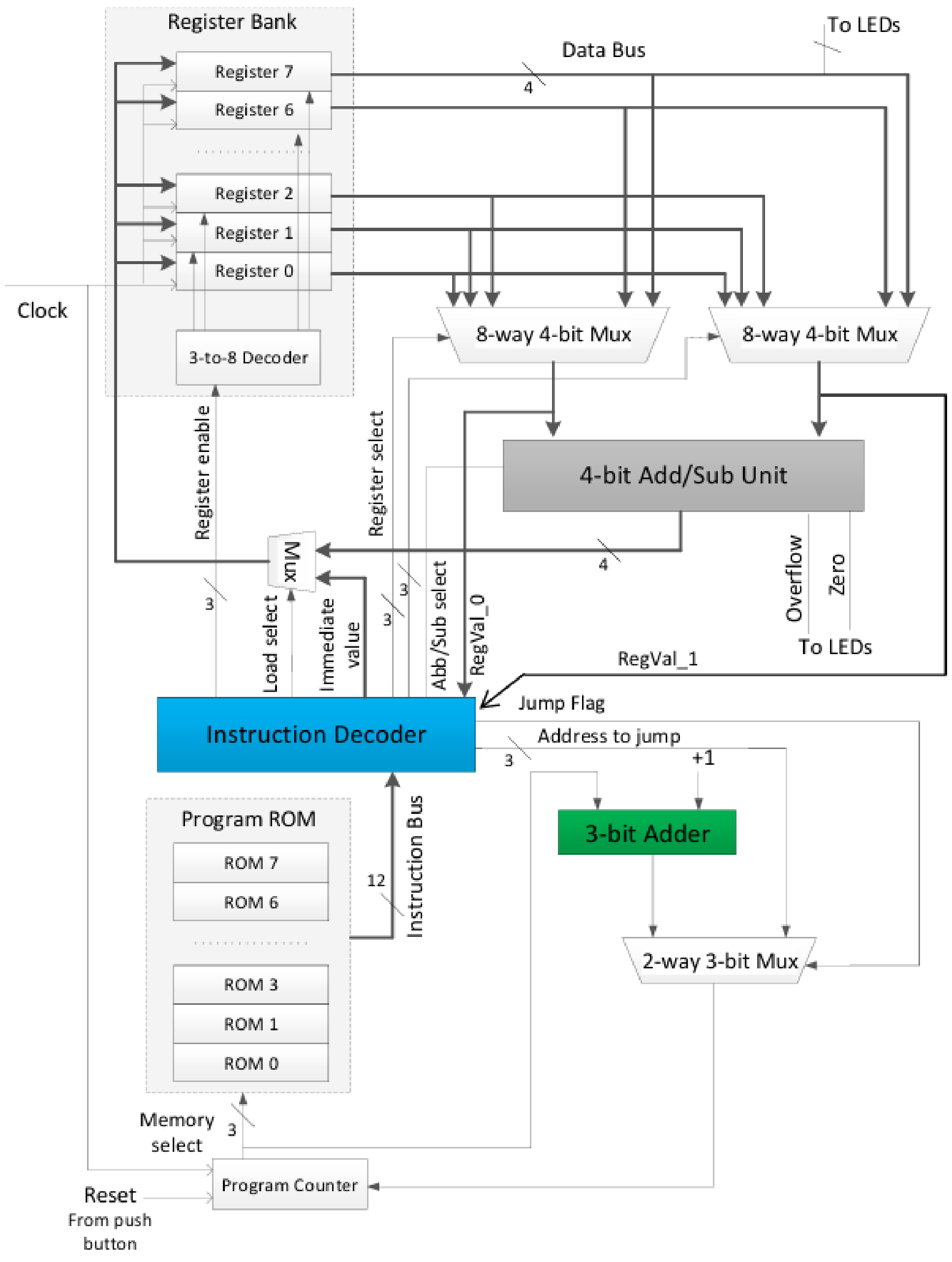
User Interface Enhancements

- Enables real-time visibility of register values.
- Simplifies debugging and step-by-step execution monitoring.
- Makes the system more interactive and suitable for educational use.

Feature	Previous Version	Modified Version
Instruction Width	12-bit	13-bit
ROM Programmability	Hardcoded	Dynamic
Register Inspection	Not Available	7-segment Display
Instruction Set	Limited	Added MOV, AND, OR, XOR

Table 2.1: Enhanced Features Comparison

2.5 High-level diagram of the enhanced nanoprocessor



.

03

Integration

3.1 Hardware Implementation on BASYS3

3.1.1 Board Interface Overview

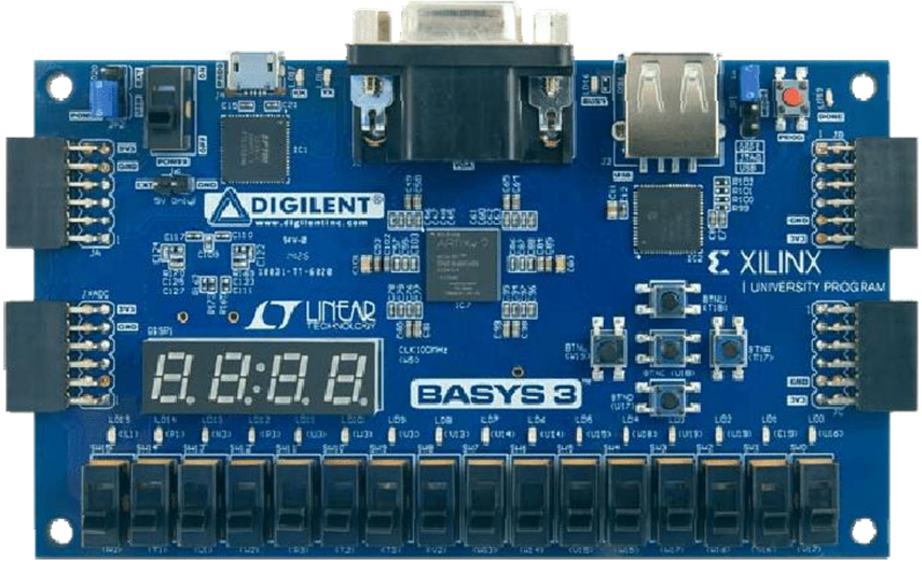


Figure 3.1: Physical interface layout on BASYS3 board

3.1.2 Original Processor Connections

Minimal I/O Configuration:

- **Clock Input:**
  - 100MHz onboard clock (Pin `W5`) $\rightarrow$ `Processor clk`
- **Control Button:**
  - Center pushbutton (Btn0, Pin `U18`) $\rightarrow$ Active-high `Reset`

- **Output Indicators:**
  - LEDs 0–3 (Pins `U16, E19, U19, V19`) → `R7` register output (`Reg_led[3:0]`)
  - LED 13 (Pin `L1`) → `Overflow flag` (`overflow_led`)
  - LED 14 (Pin `P1`) → `Zero flag` (`zero_led`)
  - LED 15 (Pin `L1`) → `Carry flag` (`carry_led`)
- **7-Segment Display:**
  - Rightmost digit only (Anode `U2` enabled)
  - Displays 3-bit program counter value (0–7)
  - Segments connected to pins `W7 (a)` through `U7 (g)`

3.1.3 Enhanced Processor Connections

New Features Added:

A. Programming Interface

- **Mode Control Buttons:**
  - Top pushbutton (BtnU, Pin `U17`) → `Program_mode` toggle
  - Left pushbutton (Btn1, Pin `T18`) → `ROM_reset`
  - Right pushbutton (Btn2, Pin `W19`) → `load_instruction`
  - Bottom pushbutton (Btn3, Pin `T17`) → `NextROM` address increment
- **Instruction Input:**
  - Switches 0–12 (Pins `V17` through `W2`) → 13-bit `new_instruction`

B. Monitoring System

- **Register Selection:**
  - Switches 13–15 (Pins `U1, T1, R2`) → 3-bit `Reg_to_show`
- **Enhanced 7-Segment Display:**
  - Digit 1 (Anode `U2`): Active register value (hexadecimal)
  - Digit 3 (Anode `U4`): Displays 'P' in program mode
  - Digit 4 (Anode `W4`): Current ROM address during programming
  - Updated at 60Hz refresh rate

C. Status Indicators

- **Additional LEDs:**
  - LED 4 (Pin `V14`) → `reset_led`
  - LED 5 (Pin `V13`) → `load_instruction_led`
  - LED 6 (Pin `U3`) → `error_led`
  - LED 7 (Pin `N3`) → `overflow_led`

04

Components

4.1 Nano Processor

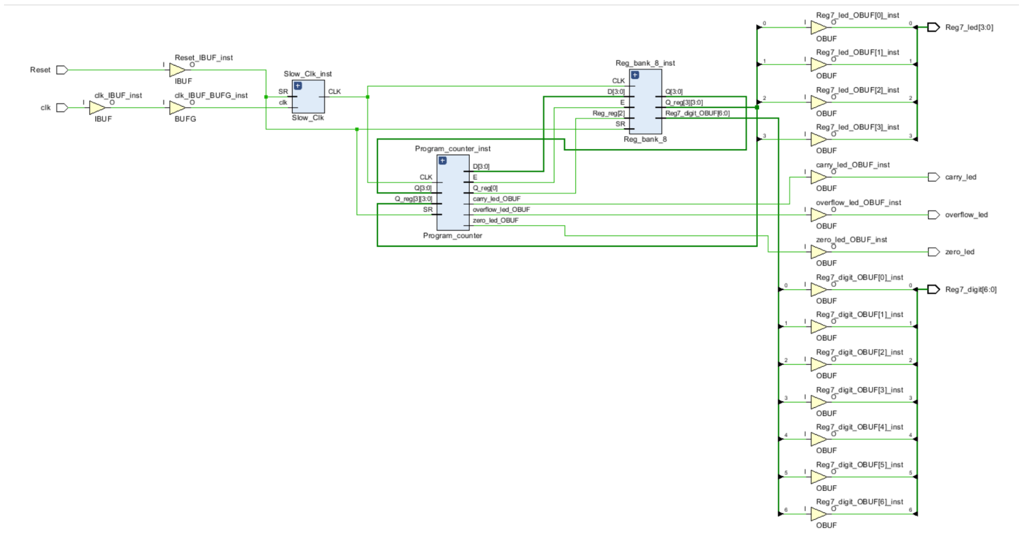
4.1.1 VHDL Design Source Code

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

entity Nano_processor is
    Port (
           Reset : in STD_LOGIC;
           clk : in STD_LOGIC;
           overflow_led : out STD_LOGIC;
           zero_led : out STD_LOGIC;
           carry_led : out STD_LOGIC;
           Reg7_led : out std_logic_vector(3 downto 0);
           Reg7_digit : out std_logic_vector(6 downto 0);
           Anode : out std_logic_vector(3 downto 0)
         );
end Nano_processor;

architecture Behavioral of Nano_processor is

component Program_counter
    Port (
           Mux_out : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           Res : in STD_LOGIC;
           Q : 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;
component Program_Rom
    Port (
           address : in STD_LOGIC_VECTOR (2 downto 0);
           instruction : out STD_LOGIC_VECTOR (11 downto 0)
         );
end component;
component Instruction_decoder
```

```
    Port (
            instruction : in STD_LOGIC_VECTOR (11 downto 0);
            jump_check : in std_logic_vector (3 downto 0);
            reg_en : out STD_LOGIC_VECTOR (2 downto 0);
            load_sel : out STD_LOGIC;
            value : out STD_LOGIC_VECTOR (3 downto 0);
            reg_a : out STD_LOGIC_VECTOR (2 downto 0);
            reg_b : out STD_LOGIC_VECTOR (2 downto 0);
            addORsub : out STD_LOGIC;
            jump_flag : out STD_LOGIC;
            jump_address : out STD_LOGIC_VECTOR (2 downto 0)
          );
end component;
component RCA_3
    Port (
            A      : in  STD_LOGIC_VECTOR(2 downto 0); -- 3-bit input A
            B      : in  STD_LOGIC_VECTOR(2 downto 0); -- 3-bit input B
            S      : out  STD_LOGIC_VECTOR(2 downto 0) -- 3-bit input C
          );
end component;
component MUX_2way_3bit
    Port (
            JumpFlag : in STD_LOGIC;
            JumpAdd : in STD_LOGIC_VECTOR (2 downto 0);
            SeqAdd : in STD_LOGIC_VECTOR (2 downto 0);
            NextPC : out STD_LOGIC_VECTOR (2 downto 0)
          );
end component;
component MUX_2way_4bit
    Port (
            loadSel : in STD_LOGIC;
            ImmedVal : in STD_LOGIC_VECTOR (3 downto 0);
            ALUVal : in STD_LOGIC_VECTOR (3 downto 0);
            OutputVal : out STD_LOGIC_VECTOR (3 downto 0)
          );
end component;
component RCA_4
    Port (
            A      : in  STD_LOGIC_VECTOR(3 downto 0); -- 4-bit input A
            B      : in  STD_LOGIC_VECTOR(3 downto 0); -- 4-bit input B
            mode : in STD_LOGIC;           -- Add/Subtract mode (0 for add,
     1 for subtract)
            S      : out STD_LOGIC_VECTOR(3 downto 0); -- 4-bit sum output
            c_out : out STD_LOGIC;           -- Carry-out
            overflow : out STD_LOGIC         -- Overflow flag
          );
end component;
component Reg_bank_8
    Port (
            RegSel : in STD_LOGIC_VECTOR (2 downto 0);
            Clk : in STD_LOGIC;                   -- Clock signal
            Reset : in STD_LOGIC;                 -- Reset signal
            Input: in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit input data
            Out_0, Out_1, Out_2, Out_3, Out_4, Out_5, Out_6, Out_7 : out
   STD_LOGIC_VECTOR (3 downto 0) -- 4-bit output data
          );
end component;
component MUX_8way_4bit
    Port (
            R0 : in STD_LOGIC_VECTOR (3 downto 0);
            R1 : in STD_LOGIC_VECTOR (3 downto 0);
```

```
           R2 : in STD_LOGIC_VECTOR (3 downto 0);
           R3 : in STD_LOGIC_VECTOR (3 downto 0);
           R4 : in STD_LOGIC_VECTOR (3 downto 0);
           R5 : in STD_LOGIC_VECTOR (3 downto 0);
           R6 : in STD_LOGIC_VECTOR (3 downto 0);
           R7 : in STD_LOGIC_VECTOR (3 downto 0);
           RegSel : in STD_LOGIC_VECTOR (2 downto 0);
           RegVal : out STD_LOGIC_VECTOR (3 downto 0)
         );
end component;
component Slow_Clk
    Port (
           Clk_in : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Clk_out : out STD_LOGIC
         );
end component;
-- For Clock
signal slow_clock : STD_LOGIC;

--For Program Counter
signal RomEn : STD_LOGIC_VECTOR (2 downto 0);
signal Mux_out : STD_LOGIC_VECTOR (2 downto 0);

--For Instruction Decoder
signal instruction : STD_LOGIC_VECTOR (11 downto 0);
signal reg_en : STD_LOGIC_VECTOR (2 downto 0);
signal load_sel : STD_LOGIC;
signal value : STD_LOGIC_VECTOR (3 downto 0);
signal reg_a : STD_LOGIC_VECTOR (2 downto 0);
signal reg_b : STD_LOGIC_VECTOR (2 downto 0);
signal addORsub : STD_LOGIC;
signal jump_flag : STD_LOGIC;
signal jump_address : STD_LOGIC_VECTOR (2 downto 0);

-- For RCA_3
signal Nxt_pc : std_logic_vector (2 downto 0);

--For RCA_4
signal ALUVal : STD_LOGIC_VECTOR (3 downto 0);
signal RegVal_0, RegVal_1 : STD_LOGIC_VECTOR (3 downto 0);

--For Reg Bank
signal OutputVal : STD_LOGIC_VECTOR (3 downto 0);
signal Out_0, Out_1, Out_2, Out_3, Out_4, Out_5, Out_6, Out_7 :
    STD_LOGIC_VECTOR (3 downto 0);


begin

    -- Instantiate the components
    Slow_Clk_inst : Slow_Clk
        port map (
            Clk_in => clk,
            Reset => Reset,
            Clk_out => slow_clock
        );

    Program_counter_inst : Program_counter
        port map (
            Mux_out => Mux_out,
            Clk => slow_clock,
            Res => Reset,
            Q => RomEn
        );

    LUT_16_7_inst : LUT_16_7
        port map (
            address => Out_7,
            data => Reg7_digit
        );

    Program_Rom_inst : Program_Rom
        port map (
            address => RomEn,
            instruction => instruction
        );

    Instruction_decoder_inst : Instruction_decoder
        port map (
            instruction => instruction,
            jump_check => RegVal_0,
            reg_en => reg_en,
            load_sel => load_sel,
            value => value,
            reg_a => reg_a,
            reg_b => reg_b,
            addORsub => addORsub,
            jump_flag => jump_flag,
            jump_address => jump_address
        );

    RCA_3_inst : RCA_3
        port map (
            A => RomEn,
            B => "001",
            S => Nxt_pc
        );

    MUX_2way_3bit_inst : MUX_2way_3bit
        port map (
           JumpFlag => jump_flag,
           JumpAdd => jump_address,
           SeqAdd => Nxt_pc,
           NextPC => Mux_out
         );

    MUX_2way_4bit_inst : MUX_2way_4bit
        port map (
           loadSel => load_sel,
           ImmedVal => value,
           ALUVal => ALUVal,
           OutputVal => OutputVal
         );

    RCA_4_inst : RCA_4
        port map (
              A => RegVal_0,
              B => RegVal_1,
              mode => addORsub,
              S => ALUVal,
              c_out => carry_led,
              overflow => overflow_led
         );

    Reg_bank_8_inst : Reg_bank_8
        port map (
           RegSel => reg_en,
           Clk => slow_clock,
           Reset => Reset,
           Input => OutputVal,
           Out_0 => Out_0,
           Out_1 => Out_1,
           Out_2 => Out_2,
           Out_3 => Out_3,
           Out_4 => Out_4,
           Out_5 => Out_5,
           Out_6 => Out_6,
           Out_7 => Out_7
         );
    MUX_8way_4bit_0 : MUX_8way_4bit
        port map (
              R0 => Out_0,
              R1 => Out_1,
              R2 => Out_2,
              R3 => Out_3,
              R4 => Out_4,
              R5 => Out_5,
              R6 => Out_6,
              R7 => Out_7,
              RegSel => reg_a,
              RegVal => RegVal_0
         );
    MUX_8way_4bit_1 : MUX_8way_4bit
        port map (
           R0 => Out_0,
           R1 => Out_1,
           R2 => Out_2,
           R3 => Out_3,
           R4 => Out_4,
           R5 => Out_5,
           R6 => Out_6,
           R7 => Out_7,
           RegSel => reg_b,
           RegVal => RegVal_1
         );


-- For LED display
    zero_led <= '1' when (ALUVal = "0000") else '0';
    Reg7_led <= Out_1;

--For 7_seg
    Anode <= "1110";

end Behavioral;
```

4.1.2 Schematic Diagram

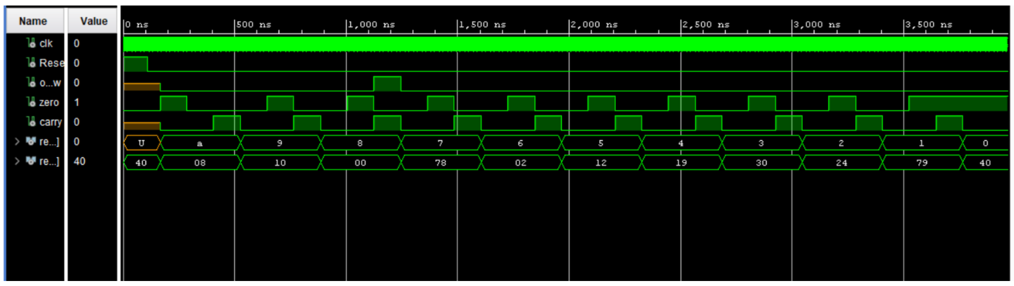


4.1.3 VHDL Simulation Source Code

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

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

architecture Behavioral of TB_nanoprocessor is
component Nano_processor
    Port (
           Reset : in STD_LOGIC;
           clk : in STD_LOGIC;
           overflow_led : out STD_LOGIC;
           zero_led : out STD_LOGIC;
           carry_led : out STD_LOGIC;
           Reg7_led : out std_logic_vector(3 downto 0);
           Reg7_digit : out std_logic_vector(6 downto 0);
           Anode : out std_logic_vector(3 downto 0)
         );
end component;

signal clk,Reset,overflow,zero,carry:std_logic;
signal reg7_led:std_logic_vector(3 downto 0);
signal reg7_dig:std_logic_vector(6 downto 0);
signal anode:std_logic_vector(3 downto 0):="1110";
```

```
begin

UUT : Nano_processor port map(
Reset=>Reset,
clk=>clk,
overflow_led=>overflow,
zero_led=>zero,
carry_led=>carry,
Reg7_led=>reg7_led,
Anode=>anode,
Reg7_digit=>reg7_dig);

process begin
clk<='0';
wait for 5ns;
clk<='1';
wait for 5ns;
end process;

process begin
Reset<='1';
wait for 110ns;
Reset<='0';
wait;
end process;

end Behavioral;
```

4.1.4 Timing Diagram

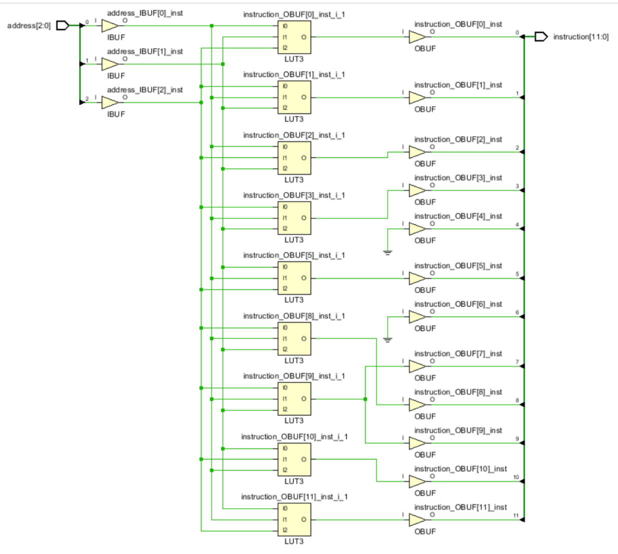


4.2 Program Rom

4.2.1 VHDL Design Source Code

```vhdl
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);
           instruction : out STD_LOGIC_VECTOR (11 downto 0));
end Program_Rom;

architecture Behavioral of Program_Rom is
type rom_type is array (0 to 7) of std_logic_vector(11 downto 0);
signal program_ROM : rom_type :=(

    "101110000001",-- MOVI R7, 1 --0
    "100100000010",-- MOVI R2, 2 --1
    "001110100000",-- ADD R7, R2 --2
    "100100000011",-- MOVI R2, 3 --3
    "001110100000",-- ADD R7, R2 --4
    "000000000000",--             --5
    "000000000000",--             --6
    "000000000000"--              --7
    );

begin
    instruction <= program_ROM(to_integer(unsigned(address)));

end Behavioral;
```

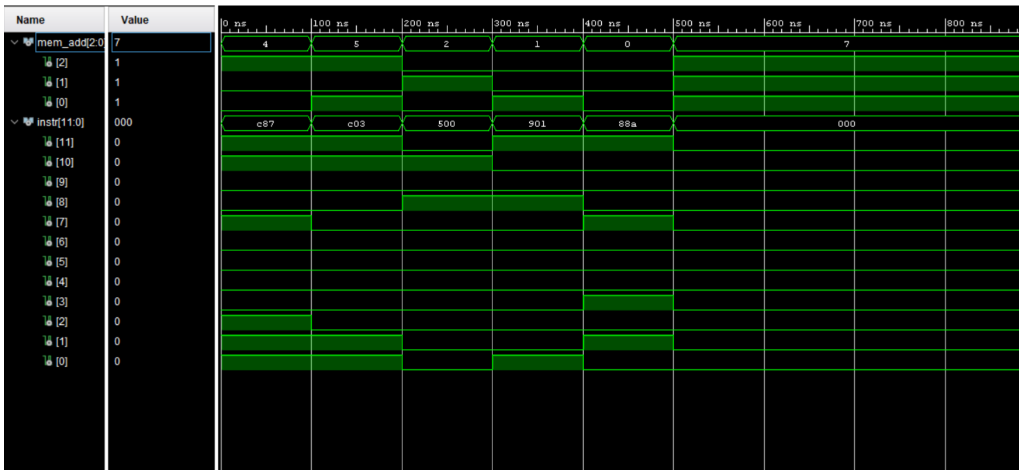
4.2.2 Schematic Diagram



4.2.3 VHDL Simulation Source Code

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

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

architecture Behavioral of TB_ROM is

component Program_Rom
    Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
           instruction : out STD_LOGIC_VECTOR (11 downto 0));
end component;

signal mem_add:std_logic_vector(2 downto 0);
signal instr:std_logic_vector(11 downto 0);

begin

UUT:Program_Rom port map(
address=>mem_add,
instruction=>instr);

process begin
-- 230060M -> 111 000 001 010 101 100
mem_add<="100";
wait for 100ns;
mem_add<="101";
wait for 100ns;
mem_add<="010";
wait for 100ns;
mem_add<="001";
wait for 100ns;
mem_add<="000";
wait for 100ns;
mem_add<="111";

wait;

end process;

end Behavioral;
```

4.2.4 Timing Diagram

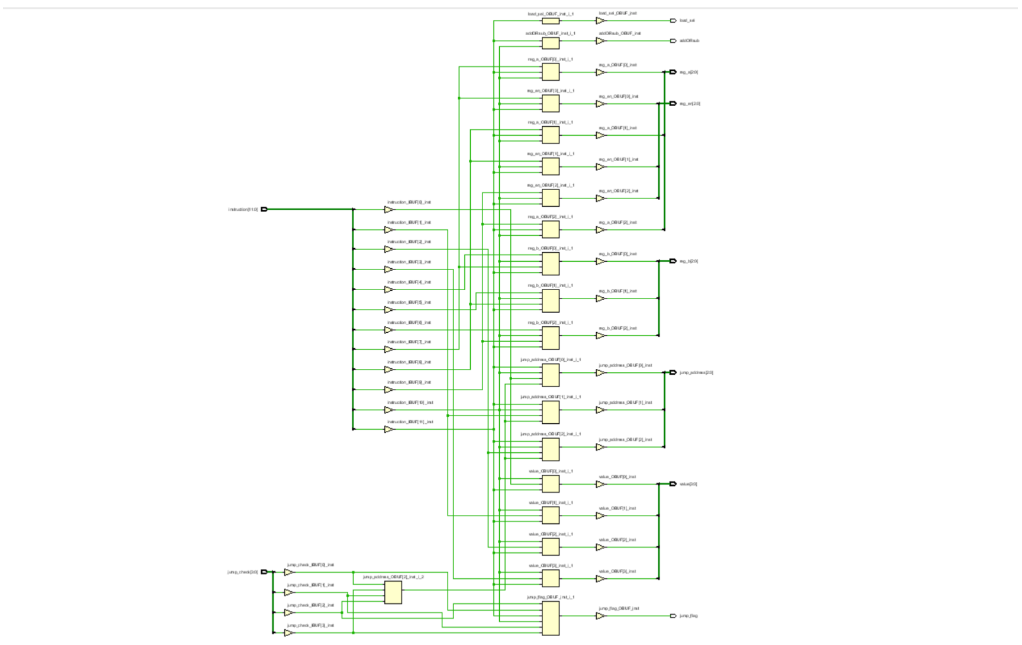


4.3 Instruction Decoder

4.3.1 VHDL Design Source Code

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


entity Instruction_decoder is
    Port ( instruction : in STD_LOGIC_VECTOR (11 downto 0);
           jump_check : in std_logic_vector (3 downto 0);
           reg_en : out STD_LOGIC_VECTOR (2 downto 0);
           load_sel : out STD_LOGIC;
           value : out STD_LOGIC_VECTOR (3 downto 0);
           reg_a : out STD_LOGIC_VECTOR (2 downto 0);
           reg_b : out STD_LOGIC_VECTOR (2 downto 0);
           addORsub : out STD_LOGIC;
           jump_flag : out STD_LOGIC;
           jump_address : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_decoder;

architecture Behavioral of Instruction_decoder is

begin

    process (instruction, jump_check)
    begin
        -- Default values for outputs
        reg_a <= (others => '0');
        reg_b <= (others => '0');
        reg_en <= (others => '0');
        load_sel <= '0';
        value <= (others => '0');
        addORsub <= '0';
        jump_flag <= '0';
        jump_address <= (others => '0');

        -- Decode the instruction
        if (instruction(11 downto 10) = "00") then
            -- Addition instruction
            reg_a <= instruction(9 downto 7);
            reg_b <= instruction(6 downto 4);
            load_sel <= '1';
            reg_en <= instruction(9 downto 7);
        elsif (instruction(11 downto 10) = "01") then
            -- Negation instruction
            reg_a <= "000"; -- In Reg6 must be contained '0000' otherwise
    the result will be wrong
            reg_b <= instruction(9 downto 7);
            load_sel <= '1';
            addORsub <= '1'; -- Changing mode to subtract to negate the
    value
            reg_en<=instruction(9 downto 7);
        elsif (instruction(11 downto 10) = "10") then
            -- Load instruction
            reg_en <= instruction(9 downto 7);
            load_sel <= '0';
            value <= instruction(3 downto 0);
        elsif (instruction(11 downto 10) = "11") then
            -- Jump instruction
            reg_a <= instruction(9 downto 7);
            if jump_check = "0000" then
                jump_flag <= '1';
                jump_address <= instruction(2 downto 0);
            else
                jump_flag <= '0';
            end if;
        end if;
    end process;

end Behavioral;
```

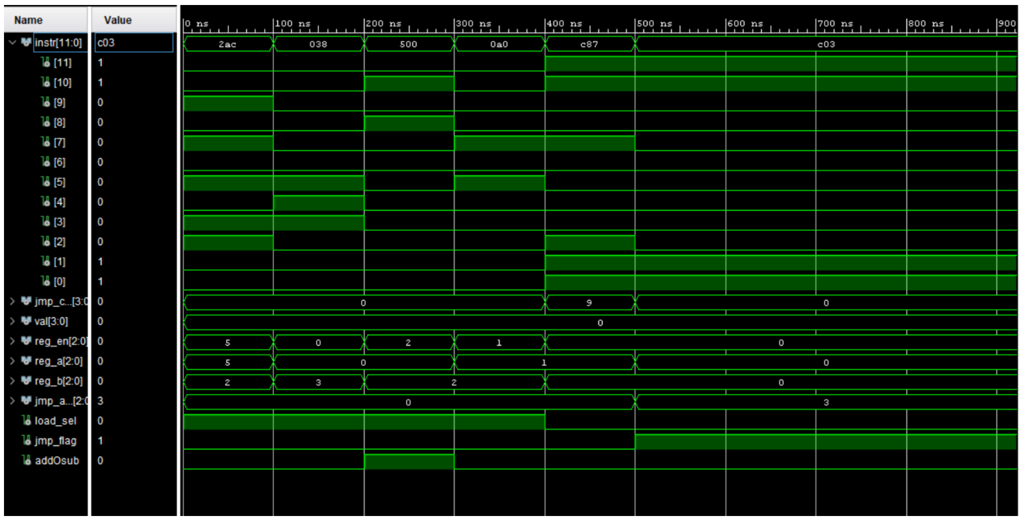
4.3.2 Schematic Diagram



4.3.3 VHDL Simulation Source Code

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

entity TB_Instruction_decoder is
end TB_Instruction_decoder;

architecture Behavioral of TB_Instruction_decoder is

    component Instruction_decoder
        Port (
            instruction   : in  STD_LOGIC_VECTOR(11 downto 0);
            jump_check    : in  STD_LOGIC_VECTOR(3 downto 0);
            reg_en        : out STD_LOGIC_VECTOR(2 downto 0);
            load_sel      : out STD_LOGIC;
            value         : out STD_LOGIC_VECTOR(3 downto 0);
            reg_a         : out STD_LOGIC_VECTOR(2 downto 0);
            reg_b         : out STD_LOGIC_VECTOR(2 downto 0);
            addORsub      : out STD_LOGIC;
            jump_flag     : out STD_LOGIC;
            jump_address  : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;

    signal instr      : STD_LOGIC_VECTOR(11 downto 0);
    signal jmp_check  : STD_LOGIC_VECTOR(3 downto 0);
    signal val        : STD_LOGIC_VECTOR(3 downto 0);
    signal reg_en     : STD_LOGIC_VECTOR(2 downto 0);
    signal reg_a      : STD_LOGIC_VECTOR(2 downto 0);
    signal reg_b      : STD_LOGIC_VECTOR(2 downto 0);
    signal jmp_add    : STD_LOGIC_VECTOR(2 downto 0);
    signal load_sel   : STD_LOGIC;
    signal jmp_flag   : STD_LOGIC;
    signal addOsub    : STD_LOGIC;

begin

    UUT: Instruction_decoder
        port map(
            instruction   => instr,
            jump_check    => jmp_check,
            reg_en        => reg_en,
            load_sel      => load_sel,
            value         => val,
            reg_a         => reg_a,
            reg_b         => reg_b,
            addORsub      => addOsub,
            jump_flag     => jmp_flag,
            jump_address  => jmp_add
        );

    process
    begin
        jmp_check <= "0000";

        -- Test Instructions
        instr <= "001010101100";
        wait for 100 ns;
        instr <= "000000111000";
        wait for 100 ns;

        instr <= "010100000000";
        wait for 100 ns;

        instr <= "000010100000";
        wait for 100 ns;

        instr <= "110010000111";
        jmp_check <= "1001";
        wait for 100 ns;

        instr <= "110000000011";
        jmp_check <= "0000";
        wait;

    end process;

end Behavioral;
```

4.3.4 Timing Diagram

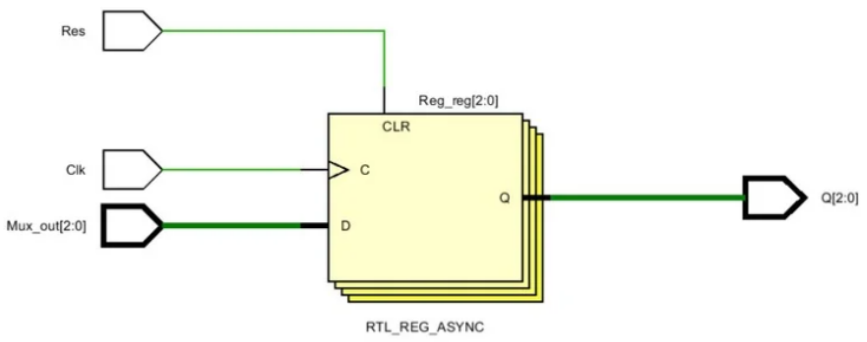


4.4 Program Counter

4.4.1 VHDL Design Source Code

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

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

architecture Behavioral of Program_counter is

    signal Reg : STD_LOGIC_VECTOR (2 downto 0) := "000";

begin

    -- Process to implement the program counter logic
    process (Clk, Res)
    begin
        if (Res = '1') then
            -- Reset the program counter to "000"
            Reg <= "000";
        elsif rising_edge(Clk) then
            -- Update the program counter with the input value
            Reg <= Mux_out;
        end if;
    end process;

    -- Assign the internal register value to the output port
    Q <= Reg;

end Behavioral;
```

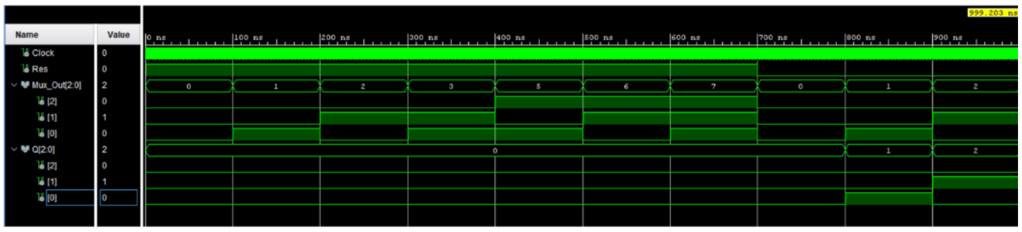
4.4.2 Schematic Diagram



4.4.3 VHDL Simulation Source Code

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

entity Program_counter_Sim is
end Program_counter_Sim;

architecture Behavioral of Program_counter_Sim is

    component Program_counter
        port (
            Mux_out  : in  STD_LOGIC_VECTOR(2 downto 0) := "000";
            Clk      : in  STD_LOGIC;
            Res      : in  STD_LOGIC;
            Q        : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;

    signal Clock     : STD_LOGIC := '0';
    signal Res       : STD_LOGIC;
    signal Mux_Out   : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal Q         : STD_LOGIC_VECTOR(2 downto 0);

begin

    UUT: Program_counter
        port map (
            Clk      => Clock,
            Res      => Res,
            Q        => Q,
            Mux_out  => Mux_Out
        );

    -- Clock generation process (assumes 50ns period)
    process
    begin
        Clock <= not Clock;
        wait for 50 ns;
    end process;

    -- Stimulus process
    process
    begin
        Res <= '1';
        Mux_Out <= "000";
        wait for 100 ns;

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

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

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

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

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

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

        Res <= '0';
        Mux_Out <= "000";
        wait for 100 ns;

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

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

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

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

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

        Mux_Out <= "111";
        wait;

    end process;

end Behavioral;
```

4.4.4 Timing Diagram

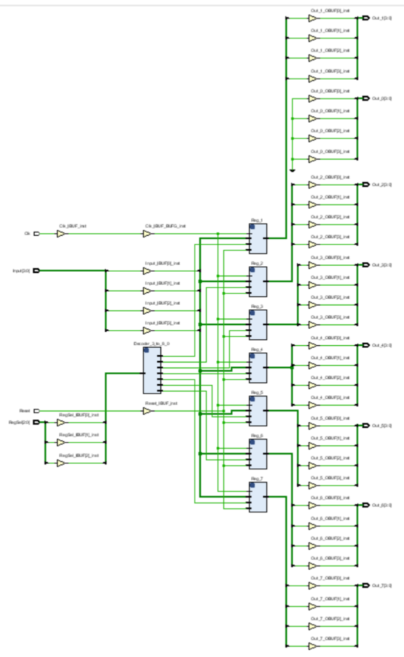


4.5 Register Bank

4.5.1 VHDL Design Source Code

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

entity Reg_bank_8 is
    Port (
            RegSel : in STD_LOGIC_VECTOR (2 downto 0);
            Clk : in STD_LOGIC;                     -- Clock signal
            Reset : in STD_LOGIC;                   -- Reset signal

            Input: in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit input data
            Out_0, Out_1, Out_2, Out_3, Out_4, Out_5, Out_6, Out_7 : out
    STD_LOGIC_VECTOR (3 downto 0) -- 4-bit output data
          );
end Reg_bank_8;

architecture Behavioral of Reg_bank_8 is

component Reg
    Port (
            D : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit input data
            En : in STD_LOGIC;                      -- Enable signal
            Clk : in STD_LOGIC;                     -- Clock signal
            Reset : in STD_LOGIC;                   -- Reset signal
            Q : out STD_LOGIC_VECTOR (3 downto 0) -- 4-bit output data
          );
end component;
component Decoder_3_to_8
    Port (
            I : in STD_LOGIC_VECTOR (2 downto 0);
            Y : out STD_LOGIC_VECTOR (7 downto 0)
          );
end component;
-- Internal signals
signal RegEn : STD_LOGIC_VECTOR (7 downto 0); -- Enable signals for
    registers

begin
--Instantiate the 3-to-8 decoder
Decoder_3_to_8_0 : Decoder_3_to_8
    port map(
        I => RegSel,
        Y => RegEn
    );
-- Instantiate the registers
Reg_0 : Reg
    port map(
        D => "0000",
        En => RegEn(0),
        Clk => Clk,
        Reset => Reset,
        Q => Out_0
    );
Reg_1 : Reg
    port map(
        D => Input,
        En => RegEn(1),
        Clk => Clk,
        Reset => Reset,
        Q => Out_1
    );
Reg_2 : Reg
    port map(
        D => Input,
        En => RegEn(2),
        Clk => Clk,
        Reset => Reset,
        Q => Out_2
    );
Reg_3 : Reg
    port map(
        D => Input,
        En => RegEn(3),
        Clk => Clk,
        Reset => Reset,
        Q => Out_3
    );
Reg_4 : Reg
    port map(
        D => Input,
        En => RegEn(4),
        Clk => Clk,
        Reset => Reset,
        Q => Out_4
    );
Reg_5 : Reg
    port map(
        D => Input,
        En => RegEn(5),
        Clk => Clk,
        Reset => Reset,
        Q => Out_5
    );
Reg_6 : Reg
    port map(
        D => Input,
        En => RegEn(6),
        Clk => Clk,
        Reset => Reset,
        Q => Out_6
    );
Reg_7 : Reg
    port map(
        D => Input,
        En => RegEn(7),
        Clk => Clk,
        Reset => Reset,
        Q => Out_7
    );

end Behavioral;
```

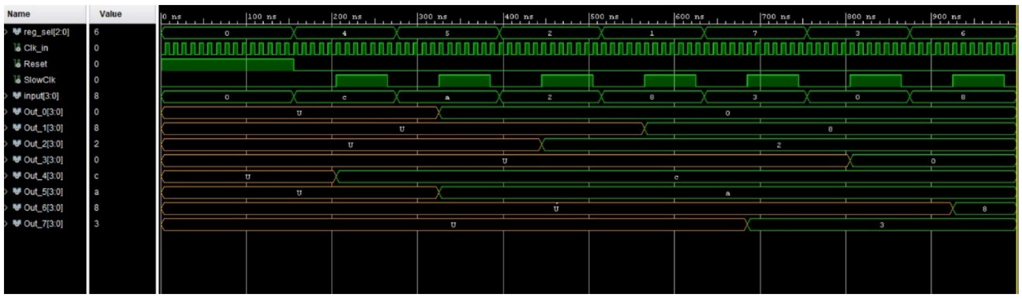
4.5.2 Schematic Diagram



4.5.3 VHDL Simulation Source Code

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

entity TB_RegBank is
end TB_RegBank;

architecture Behavioral of TB_RegBank is

    component Reg_bank_8
        Port (
            RegSel   : in  STD_LOGIC_VECTOR(2 downto 0);
            Clk      : in  STD_LOGIC;
            Reset    : in  STD_LOGIC;
            Input    : in  STD_LOGIC_VECTOR(3 downto 0);
            Out_0, Out_1, Out_2, Out_3,
            Out_4, Out_5, Out_6, Out_7 : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    component Slow_Clk
        Port (
            Clk_in  : in  STD_LOGIC;
            Reset   : in  STD_LOGIC;
            Clk_out : out STD_LOGIC
        );
    end component;
    signal reg_sel : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal Clk_in  : STD_LOGIC := '0';
    signal Reset   : STD_LOGIC := '0';
    signal SlowClk : STD_LOGIC;
    signal input   : STD_LOGIC_VECTOR(3 downto 0) := "0000";

    signal Out_0, Out_1, Out_2, Out_3,
           Out_4, Out_5, Out_6, Out_7 : STD_LOGIC_VECTOR(3 downto 0);

begin

    -- Instantiate Slow Clock Generator
    UUT: Slow_Clk
        port map (
            Clk_in  => Clk_in,
            Reset   => Reset,
            Clk_out => SlowClk
        );

    -- Instantiate Register Bank
    UUT_RegBank: Reg_bank_8
        port map (
            RegSel => reg_sel,
            Clk    => SlowClk,
            Reset  => Reset,
            Input  => input,
            Out_0  => Out_0,
            Out_1  => Out_1,
            Out_2  => Out_2,
            Out_3  => Out_3,
            Out_4  => Out_4,
            Out_5  => Out_5,
            Out_6  => Out_6,
            Out_7  => Out_7
        );

    -- Clock generation process (10 ns period)
    Clk_in_process: process
    begin
        Clk_in <= '0';
        wait for 5 ns;
        Clk_in <= '1';
        wait for 5 ns;
    end process;

    -- Stimulus process
    stim_process: process
    begin
        -- Initial reset
        Reset   <= '1';
        reg_sel <= "000";
        input   <= "0000";
        wait for 155 ns;

        Reset  <= '0';

        -- Load values into registers
        reg_sel <= "100"; input <= "1100"; wait for 120 ns;
        reg_sel <= "101"; input <= "1010"; wait for 120 ns;
        reg_sel <= "010"; input <= "0010"; wait for 120 ns;
        reg_sel <= "001"; input <= "1000"; wait for 120 ns;
        reg_sel <= "111"; input <= "0011"; wait for 120 ns;
        reg_sel <= "011"; input <= "0000"; wait for 120 ns;
        reg_sel <= "110"; input <= "1000"; wait;

    end process;

end Behavioral;
```

4.5.4 Timing Diagram

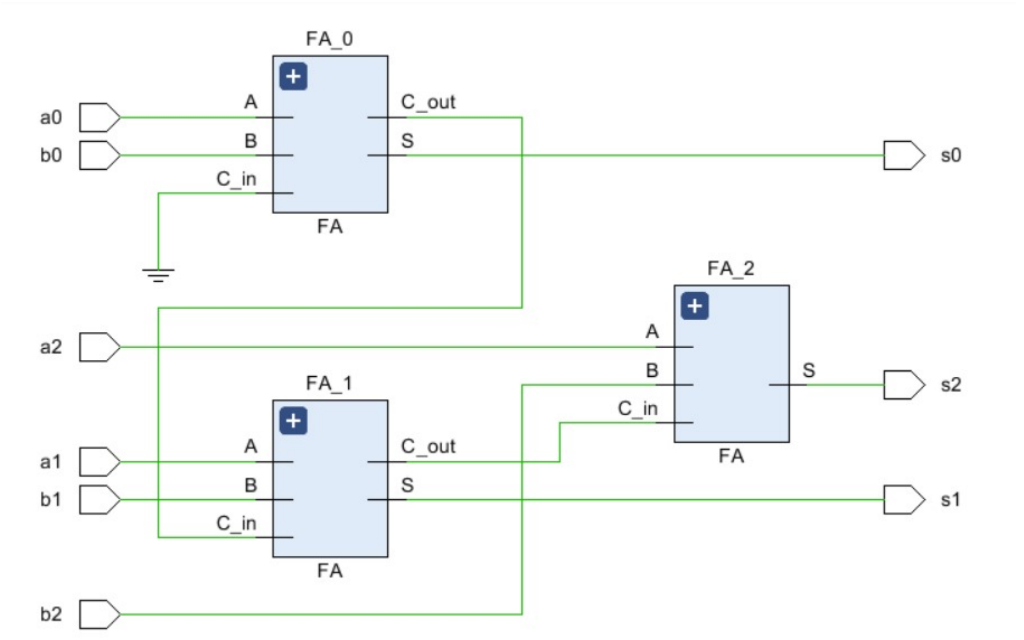


4.6 4 bit Adder/Substractor

4.6.1 VHDL Design Source Code

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

entity RCA_4 is
    Port (
           A : in STD_LOGIC_VECTOR(3 downto 0); -- 4-bit input A
           B : in STD_LOGIC_VECTOR(3 downto 0); -- 4-bit input B
           mode : in STD_LOGIC;            -- Add/Subtract mode (0 for add,
     1 for subtract)
           S : out STD_LOGIC_VECTOR(3 downto 0); -- 4-bit sum output
           c_out : out STD_LOGIC;            -- Carry-out
           overflow : out STD_LOGIC          -- Overflow flag
          );
end RCA_4;

architecture Behavioral of RCA_4 is

    -- Component declaration for the Full Adder
    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;

    -- Internal signals for carry propagation
    SIGNAL FA0_C, FA1_C, FA2_C : STD_LOGIC;
    signal b0_m, b1_m, b2_m, b3_m : std_logic;
    signal c_out_internal : std_logic;



begin

    -- FA_0: Full adder for the least significant bit
    FA_0 : FA
        port map (
            A => A(0),
            B => b0_m, -- XOR with mode for subtraction
            C_in => mode, -- Carry-in is mode for subtraction
            S => S(0),
            C_Out => FA0_C
        );

    -- FA_1: Full adder for the second bit
    FA_1 : FA
        port map (
            A => A(1),
            B => b1_m, -- XOR with mode for subtraction
            C_in => FA0_C,
            S => S(1),
            C_Out => FA1_C
        );
    -- FA_2: Full adder for the third bit
    FA_2 : FA
        port map (
            A => A(2),
            B => b2_m, -- XOR with mode for subtraction
            C_in => FA1_C,
            S => S(2),
            C_Out => FA2_C
        );

    -- FA_3: Full adder for the most significant bit
    FA_3 : FA
        port map (
            A => A(3),
            B => b3_m, -- XOR with mode for subtraction
            C_in => FA2_C,
            S => S(3),
            C_Out => c_out_internal
        );

    -- Overflow detection: Check if the carry-in and carry-out of the MSB
   are different
    c_out <= c_out_internal;
    overflow <= FA2_C xor c_out_internal;


    -- For mode selection
    b0_m <= b(0) xor mode;
    b1_m <= b(1) xor mode;
    b2_m <= b(2) xor mode;
    b3_m <= b(3) xor mode;

end Behavioral;
```

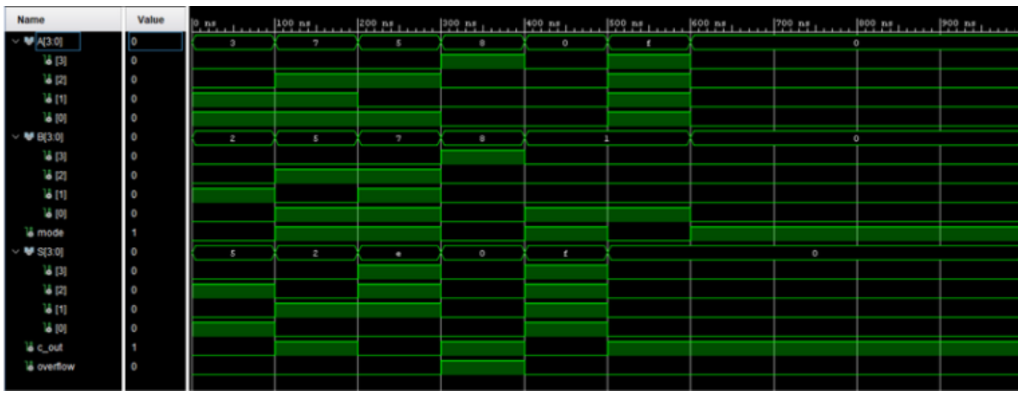
4.6.2 Schematic Diagram



4.6.3 VHDL Simulation Source Code

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

entity RCA_4_tb is
end RCA_4_tb;

architecture Behavioral of RCA_4_tb is

    -- Component Declaration for the Unit Under Test (UUT)
    component RCA_4
        Port (
            A        : in  STD_LOGIC_VECTOR(3 downto 0);
            B        : in  STD_LOGIC_VECTOR(3 downto 0);
            mode     : in  STD_LOGIC;
            S        : out STD_LOGIC_VECTOR(3 downto 0);
            c_out    : out STD_LOGIC;
            overflow : out STD_LOGIC
        );
    end component;

    -- Signals to connect to the UUT
    signal A        : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
    signal B        : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
    signal mode     : STD_LOGIC := '0';
    signal S        : STD_LOGIC_VECTOR(3 downto 0);
    signal c_out    : STD_LOGIC;
    signal overflow : STD_LOGIC;

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: RCA_4
        Port map (
            A        => A,
            B        => B,
            mode     => mode,
            S        => S,
            c_out    => c_out,
            overflow => overflow
        );

    -- Stimulus process
    stim_proc: process
    begin
        -- Test Case 1: 3 + 2 = 5
        A    <= "0011";  -- 3
        B    <= "0010";  -- 2
        mode <= '0';     -- Addition
        wait for 100 ns;

        -- Test Case 2: 7 - 5 = 2
        A    <= "0111";  -- 7
        B    <= "0101";  -- 5
        mode <= '1';     -- Subtraction
        wait for 100 ns;

        -- Test Case 3: 5 - 7 = -2 (Two's complement: 1110)
        A    <= "0101";  -- 5
        B    <= "0111";  -- 7
        mode <= '1';     -- Subtraction
        wait for 100 ns;

        -- Test Case 4: 8 + 8 = 16 (Overflow expected)
        A    <= "1000";  -- 8
        B    <= "1000";  -- 8
        mode <= '0';     -- Addition
        wait for 100 ns;

        -- Test Case 5: 0 - 1 = -1 (Two's complement: 1111)
        A    <= "0000";  -- 0
        B    <= "0001";  -- 1
        mode <= '1';     -- Subtraction
        wait for 100 ns;

        -- Test Case 6: 15 + 1 = 0 (Overflow expected)
        A    <= "1111";  -- 15
        B    <= "0001";  -- 1
        mode <= '0';     -- Addition
        wait for 100 ns;

        -- Test Case 7: 0 - 0 = 0
        A    <= "0000";  -- 0
        B    <= "0000";  -- 0
        mode <= '1';     -- Subtraction
        wait for 100 ns;

        -- End of simulation
        wait;
    end process;

end Behavioral;
```

4.6.4 Timing Diagram

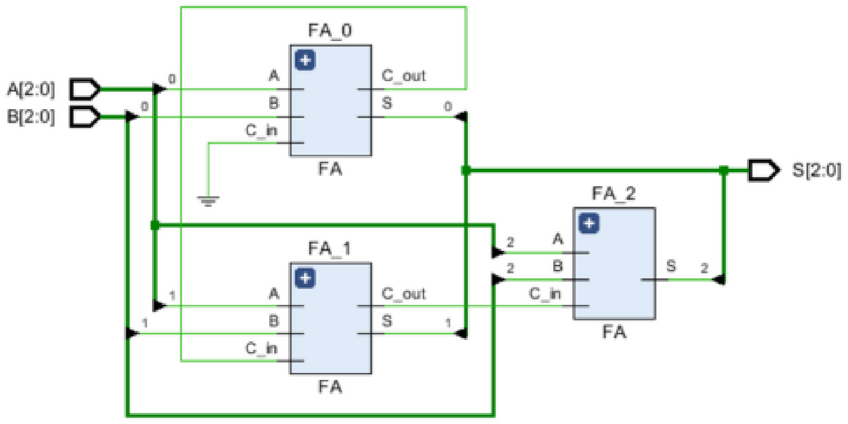


4.7 3 bit Adder

4.7.1 VHDL Design Source Code

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

entity RCA_3 is
    Port (
           A : in STD_LOGIC_VECTOR (2 downto 0); -- 3-bit input A
           B : in STD_LOGIC_VECTOR (2 downto 0);-- 3-bit input B
           S : out STD_LOGIC_VECTOR (2 downto 0) -- 3-bit sum output
         );
end RCA_3;

architecture Behavioral of RCA_3 is

    -- Component declaration for the Full Adder
    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;

    -- Internal signals for carry propagation
    SIGNAL FA0_C, FA1_C, c_out : STD_LOGIC;

begin

    -- FA_0: Full adder for the least significant bit
    FA_0 : FA
        port map (
            A => A(0),
            B => B(0),
            C_in => '0',
            S => S(0),
            C_Out => FA0_C
        );

    -- FA_1: Full adder for the second bit
    FA_1 : FA
        port map (
            A => A(1),
            B => B(1),
            C_in => FA0_C,
            S => S(1),
            C_Out => FA1_C
        );

    -- FA_2: Full adder for the third bit
    FA_2 : FA
        port map (
            A => A(2),
            B => B(2),
            C_in => FA1_C,
            S => S(2),
            C_Out => c_out
        );


end Behavioral;
```

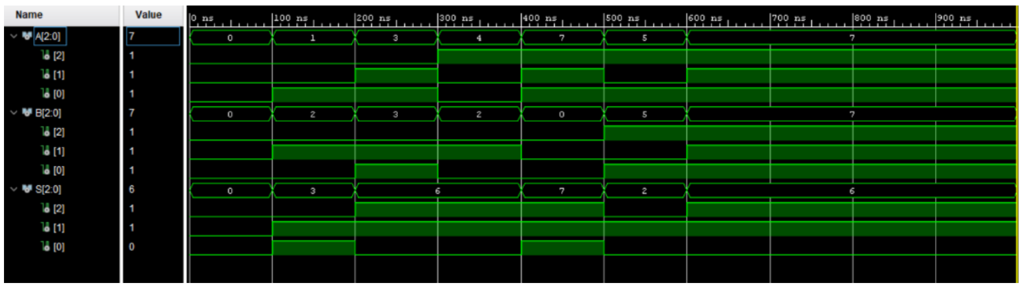
4.7.2 Schematic Diagram



4.7.3 VHDL Simulation Source Code

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

entity RCA_3_tb is
end RCA_3_tb;

architecture Behavioral of RCA_3_tb is

    -- Component declaration for RCA_3
    component RCA_3
        Port (
            A : in  STD_LOGIC_VECTOR(2 downto 0);
            B : in  STD_LOGIC_VECTOR(2 downto 0);
            S : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;

    -- Signals to connect to RCA_3
    signal A, B : STD_LOGIC_VECTOR(2 downto 0) := (others => '0');
    signal S    : STD_LOGIC_VECTOR(2 downto 0);

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: RCA_3
        port map (
            A => A,
            B => B,
            S => S
        );

    -- Stimulus process
    stim_proc: process
    begin
        -- Test 1: 0 + 0 = 0
        A <= "000"; B <= "000";
        wait for 100 ns;

        -- Test 2: 1 + 2 = 3
        A <= "001"; B <= "010";
        wait for 100 ns;

        -- Test 3: 3 + 3 = 6
        A <= "011"; B <= "011";
        wait for 100 ns;

        -- Test 4: 4 + 2 = 6
        A <= "100"; B <= "010";
        wait for 100 ns;

        -- Test 5: 7 + 0 = 7
        A <= "111"; B <= "000";
        wait for 100 ns;

        -- Test 6: 5 + 5 = 10 (wraps around in 3-bit, result = 2)
        A <= "101"; B <= "101";
        wait for 100 ns;

        -- Test 7: 7 + 7 = 14 (wraps to 6)
        A <= "111"; B <= "111";
        wait for 100 ns;

        -- End simulation
        wait;
    end process;

end Behavioral;
```

4.7.4 Timing Diagram

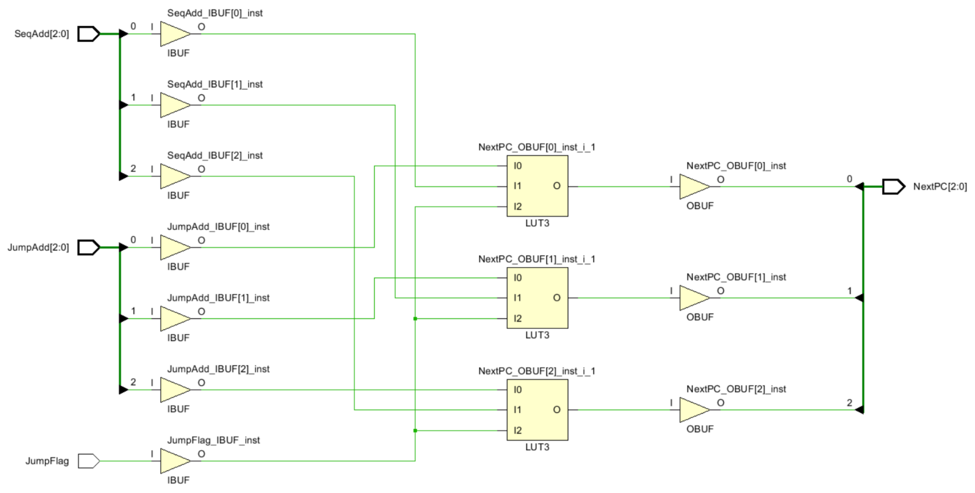


4.8 2 way 3 bit Multiplexer

4.8.1 VHDL Design Source Code

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

entity MUX_2way_3bit is
    Port ( JumpFlag : in STD_LOGIC;
           JumpAdd : in STD_LOGIC_VECTOR (2 downto 0);
           SeqAdd : in STD_LOGIC_VECTOR (2 downto 0);
           NextPC : out STD_LOGIC_VECTOR (2 downto 0));
end MUX_2way_3bit;

architecture Behavioral of MUX_2way_3bit is

begin

NextPC <= JumpAdd when JumpFlag = '1' else SeqAdd;

end Behavioral;
```

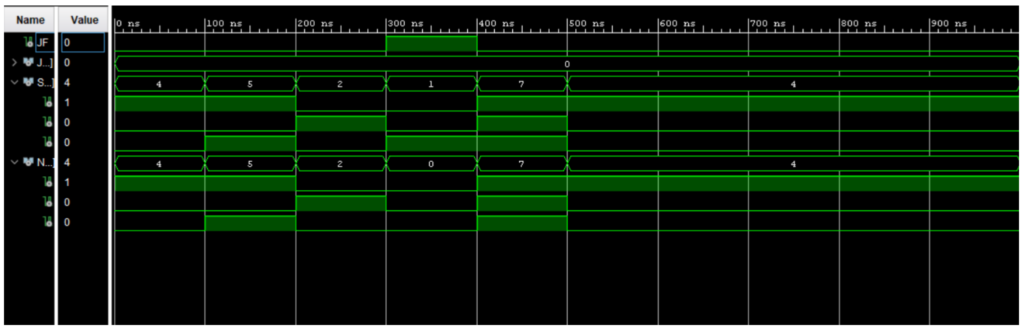
4.8.2 Schematic Diagram



4.8.3 VHDL Simulation Source Code

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

entity MUX_2way_3bit_tb is
end MUX_2way_3bit_tb;

architecture Behavioral of MUX_2way_3bit_tb is

    -- Component declaration for the Unit Under Test (UUT)
    component MUX_2way_3bit
        Port (
            JumpFlag : in STD_LOGIC;
            JumpAdd  : in STD_LOGIC_VECTOR(2 downto 0);
            SeqAdd   : in STD_LOGIC_VECTOR(2 downto 0);
            NextPC    : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;

    -- Signals to connect to the UUT
    signal JumpFlag : STD_LOGIC := '0';
    signal JumpAdd  : STD_LOGIC_VECTOR(2 downto 0) := (others => '0');
    signal SeqAdd   : STD_LOGIC_VECTOR(2 downto 0) := (others => '0');
    signal NextPC   : STD_LOGIC_VECTOR(2 downto 0);

begin

    -- Instantiate the UUT
    uut: MUX_2way_3bit
        port map (
            JumpFlag => JumpFlag,
            JumpAdd  => JumpAdd,
            SeqAdd   => SeqAdd,
            NextPC   => NextPC
        );

    -- Stimulus process
    stim_proc: process
    begin
        -- Test 1: JumpFlag = 0 ? Output = SeqAdd
        SeqAdd   <= "001";
        JumpAdd  <= "101";
        JumpFlag <= '0';
        wait for 100 ns;

        -- Test 2: JumpFlag = 1 ? Output = JumpAdd
        JumpFlag <= '1';
        wait for 100 ns;

        -- Test 3: Different values
        SeqAdd   <= "010";
        JumpAdd  <= "111";
        JumpFlag <= '0';
        wait for 100 ns;

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

        -- Test 4: JumpFlag toggle while inputs are same
        SeqAdd   <= "011";
        JumpAdd  <= "011";
        JumpFlag <= '0';
        wait for 100 ns;

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

        -- End of simulation
        wait;
    end process;

end Behavioral;
```

4.8.4 Timing Diagram

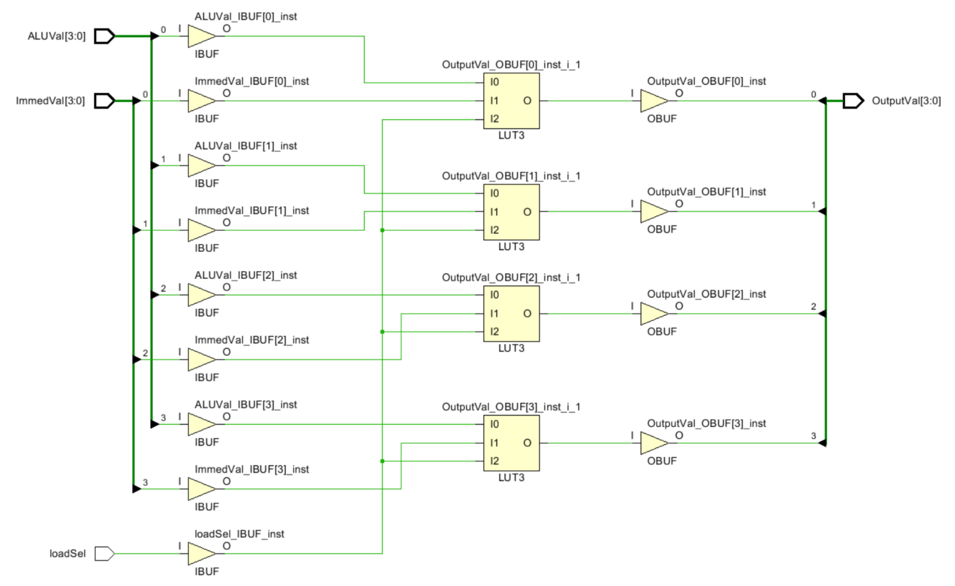


4.9 2 way 4 bit Multiplexer

4.9.1 VHDL Design Source Code

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


entity MUX_2way_4bit is
    Port ( loadSel : in STD_LOGIC;
           ImmedVal : in STD_LOGIC_VECTOR (3 downto 0);
           ALUVal : in STD_LOGIC_VECTOR (3 downto 0);
           OutputVal : out STD_LOGIC_VECTOR (3 downto 0));
end MUX_2way_4bit;

architecture Behavioral of MUX_2way_4bit is

begin

OutputVal <= ImmedVal when loadSel = '0' else ALUVal;

end Behavioral;
```

4.9.2 Schematic Diagram

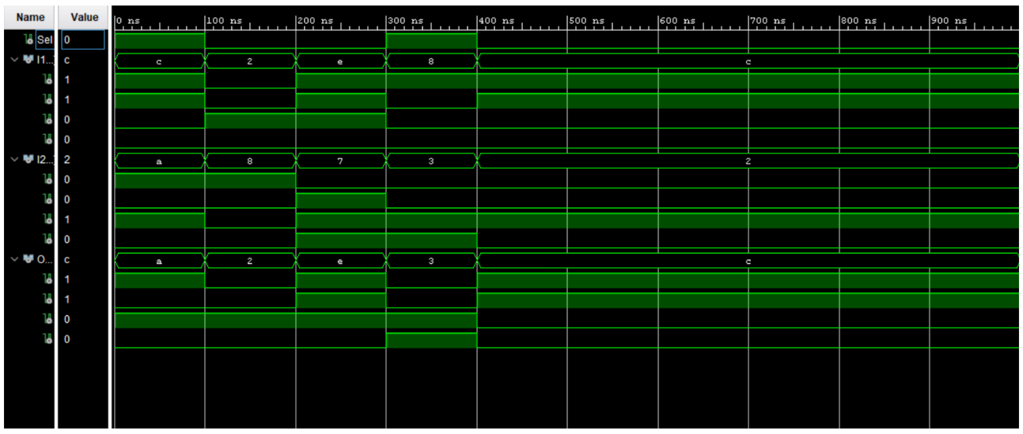


4.9.3 VHDL Simulation Source Code

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

entity MUX_2way_4bit_tb is
end MUX_2way_4bit_tb;

architecture Behavioral of MUX_2way_4bit_tb is

    -- Component declaration for Unit Under Test (UUT)
    component MUX_2way_4bit
```

```
        Port (
            loadSel   : in  STD_LOGIC;
            ImmedVal  : in  STD_LOGIC_VECTOR(3 downto 0);
            ALUVal    : in  STD_LOGIC_VECTOR(3 downto 0);
            OutputVal : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    -- Signals to connect to UUT
    signal loadSel   : STD_LOGIC := '0';
    signal ImmedVal  : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
    signal ALUVal    : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
    signal OutputVal : STD_LOGIC_VECTOR(3 downto 0);

begin

    -- Instantiate the Unit Under Test
    uut: MUX_2way_4bit
        port map (
            loadSel   => loadSel,
            ImmedVal  => ImmedVal,
            ALUVal    => ALUVal,
            OutputVal => OutputVal
        );

    -- Stimulus process
    stim_proc: process
    begin
        -- Test 1: loadSel = 0 ? OutputVal = ImmedVal
        ImmedVal  <= "0001";
        ALUVal    <= "1111";
        loadSel   <= '0';
        wait for 100 ns;

        -- Test 2: loadSel = 1 ? OutputVal = ALUVal
        loadSel <= '1';
        wait for 100 ns;

        -- Test 3: Different values
        ImmedVal  <= "1010";
        ALUVal    <= "0101";
        loadSel   <= '0';
        wait for 100 ns;

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

        -- Test 4: Edge values
        ImmedVal  <= "0000";
        ALUVal    <= "1111";
        loadSel   <= '0';
        wait for 100 ns;

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

        -- End of simulation
        wait;
    end process;

end Behavioral;
```

4.9.4 Timing Diagram

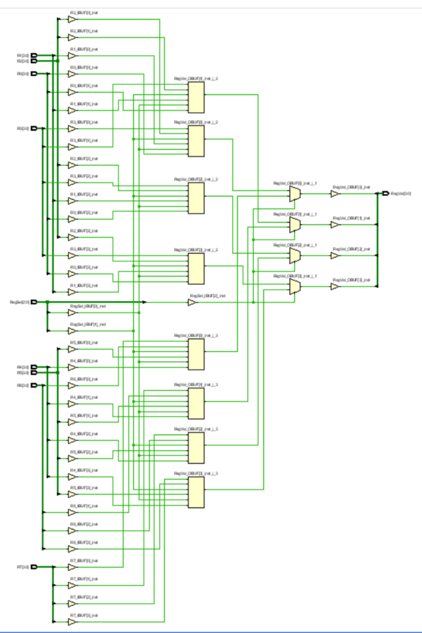


4.10 8 way 4 bit Multiplexer

4.10.1 VHDL Design Source Code

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

entity MUX_8way_4bit is
    Port ( R0 : in STD_LOGIC_VECTOR (3 downto 0);
           R1 : in STD_LOGIC_VECTOR (3 downto 0);
           R2 : in STD_LOGIC_VECTOR (3 downto 0);
           R3 : in STD_LOGIC_VECTOR (3 downto 0);
           R4 : in STD_LOGIC_VECTOR (3 downto 0);
           R5 : in STD_LOGIC_VECTOR (3 downto 0);
           R6 : in STD_LOGIC_VECTOR (3 downto 0);
           R7 : in STD_LOGIC_VECTOR (3 downto 0);
           RegSel : in STD_LOGIC_VECTOR (2 downto 0);
           RegVal : out STD_LOGIC_VECTOR (3 downto 0));
end MUX_8way_4bit;

architecture Behavioral of MUX_8way_4bit is
begin

    with RegSel select
        RegVal <= R0 when "000",
                  R1 when "001",
                  R2 when "010",
                  R3 when "011",
                  R4 when "100",
                  R5 when "101",
                  R6 when "110",
                  R7 when "111";

end Behavioral;
```

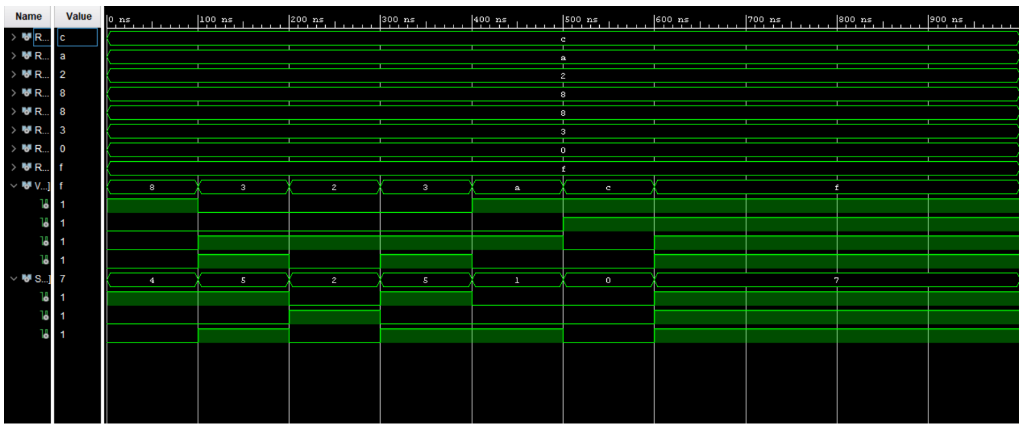
4.10.2 Schematic Diagram



4.10.3 VHDL Simulation Source Code

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

entity MUX_8way_4bit_tb is
end MUX_8way_4bit_tb;

architecture Behavioral of MUX_8way_4bit_tb is

    -- Component Declaration
    component MUX_8way_4bit
        Port (
            R0 : in STD_LOGIC_VECTOR(3 downto 0);
            R1 : in STD_LOGIC_VECTOR(3 downto 0);
            R2 : in STD_LOGIC_VECTOR(3 downto 0);
            R3 : in STD_LOGIC_VECTOR(3 downto 0);
            R4 : in STD_LOGIC_VECTOR(3 downto 0);
            R5 : in STD_LOGIC_VECTOR(3 downto 0);
            R6 : in STD_LOGIC_VECTOR(3 downto 0);
            R7 : in STD_LOGIC_VECTOR(3 downto 0);
            RegSel : in STD_LOGIC_VECTOR(2 downto 0);
            RegVal : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    -- Test Signals
    signal R0, R1, R2, R3, R4, R5, R6, R7 : STD_LOGIC_VECTOR(3 downto 0);
    signal RegSel : STD_LOGIC_VECTOR(2 downto 0);
    signal RegVal : STD_LOGIC_VECTOR(3 downto 0);

begin

    -- Instantiate the UUT
    uut: MUX_8way_4bit
        port map (
            R0 => R0,
            R1 => R1,
            R2 => R2,
            R3 => R3,
            R4 => R4,
            R5 => R5,
            R6 => R6,
            R7 => R7,
            RegSel => RegSel,
            RegVal => RegVal
        );

    -- Test Process
    stim_proc: process
    begin
        -- Assign known values to all inputs
        R0 <= "0000";
        R1 <= "0001";
        R2 <= "0010";
        R3 <= "0011";
        R4 <= "0100";
        R5 <= "0101";
        R6 <= "0110";
        R7 <= "0111";
        -- Test each RegSel
        RegSel <= "000"; wait for 100 ns;
        RegSel <= "001"; wait for 100 ns;
        RegSel <= "010"; wait for 100 ns;
        RegSel <= "011"; wait for 100 ns;
        RegSel <= "100"; wait for 100 ns;
        RegSel <= "101"; wait for 1000 ns;
        RegSel <= "110"; wait for 100 ns;
        RegSel <= "111"; wait for 100 ns;


        wait;
    end process;

end Behavioral;
```

4.10.4 Timing Diagram

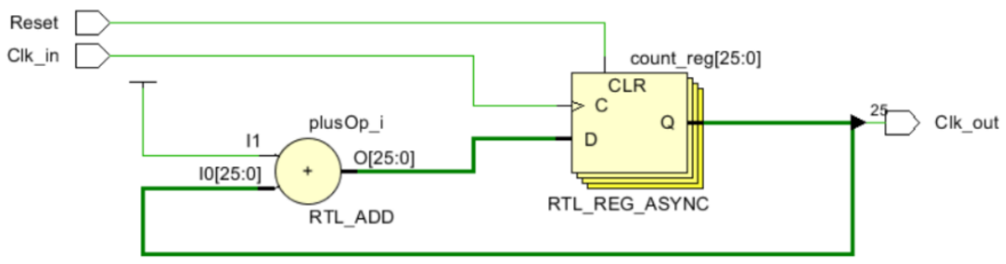


4.11 Slow Clock

4.11.1 VHDL Design Source Code

```
--
   ----------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date: 03/28/2025 12:27:22 PM
-- Design Name:
-- Module Name: Slow_Clk - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--   This module generates a slower clock signal by dividing the input
   clock.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--
   ----------------------------------------------------------------------------------


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

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

architecture Behavioral of Slow_Clk is
    signal count : unsigned(25 downto 0) := (others => '0');
    --modified to reduce lUT usage
begin

    process (Clk_in, Reset)
    begin
        if (Reset = '1') then
            count <= (others => '0');
        elsif rising_edge(Clk_in) then
            count <= count + 1;
        end if;
    end process;

    Clk_out <= count(25); -- Use the MSB as the slow clock

end Behavioral;
```

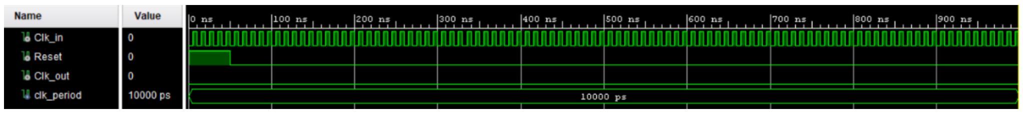
4.11.2 Schematic Diagram



4.11.3 VHDL Simulation Source Code

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

entity Slow_Clk_tb is
end Slow_Clk_tb;

architecture Behavioral of Slow_Clk_tb is

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

    -- Signals for connecting to the DUT
    signal Clk_in  : STD_LOGIC := '0';
    signal Reset   : STD_LOGIC := '0';
    signal Clk_out : STD_LOGIC;

    constant clk_period : time := 10 ns;

begin

    -- Instantiate the Slow_Clk module
    uut: Slow_Clk
        port map (
            Clk_in    => Clk_in,
            Reset     => Reset,
            Clk_out   => Clk_out
        );

    -- Clock generation process
    clk_process: process
    begin
        while now < 2 ms loop   -- simulate for 2 milliseconds
            Clk_in <= '0';
            wait for clk_period / 2;
            Clk_in <= '1';
            wait for clk_period / 2;
        end loop;
        wait;
    end process;
    -- Stimulus process
    stim_proc: process
    begin
        -- Hold reset for 50 ns
        Reset <= '1';
        wait for 50 ns;
        Reset <= '0';

        -- Observe output for a while
        wait for 2 ms;
        wait;
    end process;

end Behavioral;
```

4.11.4 Timing Diagram



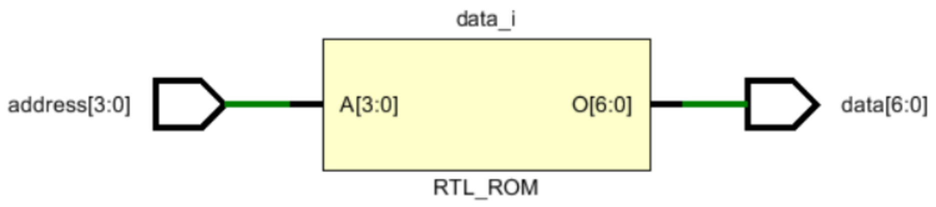
```
Name        Value
Clk_in      0
Reset       0
Clk_out     0
clk_period  10000 ps
```

4.12 7-Segment Displays

4.12.1 VHDL Design Source Code

```
--
    ----------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date: 04/25/2025 06:03:44 PM
-- Design Name:
-- Module Name: LUT_16_7 - Behavioral
-- Project Name: Lab 8
-- Target Devices:
-- Tool Versions:
-- Description:
--   This module implements a 16-to-7 lookup table (LUT) for driving a
    seven-segment display.
--   It maps a 4-bit input address to a 7-bit output based on a predefined
     ROM.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--   The ROM maps hexadecimal digits (0 to F) to their corresponding seven
    -segment display patterns.
--
--
    ----------------------------------------------------------------------------------


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); -- 4-bit input
    address
            data : out STD_LOGIC_VECTOR (6 downto 0)    -- 7-bit output for
     seven-segment display
        );
end LUT_16_7;

architecture Behavioral of LUT_16_7 is
    -- Define the ROM type and initialize it with seven-segment patterns
    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
    -- Map the input address to the corresponding seven-segment pattern
    data <= sevenSegment_ROM(to_integer(unsigned(address)));
end Behavioral;
```

4.12.2 Schematic Diagram

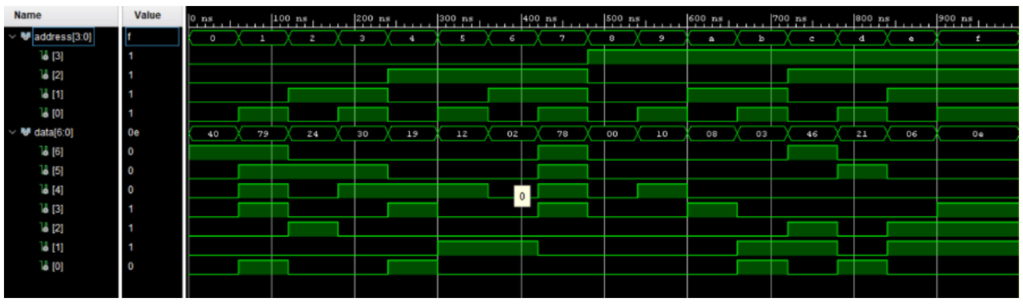


4.12.3 VHDL Simulation Source Code

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

entity LUT_16_7_tb is
end LUT_16_7_tb;

architecture Behavioral of LUT_16_7_tb is

    -- Component declaration of the Unit Under Test (UUT)
    component LUT_16_7
        Port (
            address : in STD_LOGIC_VECTOR (3 downto 0);
            data    : out STD_LOGIC_VECTOR (6 downto 0)
        );
    end component;

    -- Internal signals to connect to the UUT
    signal address : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
    signal data    : STD_LOGIC_VECTOR (6 downto 0);
```

```
begin

    -- Instantiate the UUT
    uut: LUT_16_7
        port map (
            address => address,
            data    => data
        );

    -- Stimulus process
    stim_proc: process
    begin
        for i in 0 to 15 loop
            address <= std_logic_vector(to_unsigned(i, 4));
            wait for 60 ns;
        end loop;

        wait;
    end process;

end Behavioral;
```

4.12.4 Timing Diagram



4.13 Enhanced Nanoprocessor VHDL Source Codes

4.13.1 Nanoprocessor

VHDL Design Source Code

```
--
    ----------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date: 27.02.2025 05:01:10
-- Design Name:
-- Module Name: Nano_processor - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--   This module implements a 4-bit Nano processor with a program counter,
--   instruction decoder, ALU, register bank, and LED display outputs.
--
-- Dependencies:
--   Requires components: Slow_Clk, Program_counter, LUT_16_7, Program_Rom
    ,
--   Instruction_decoder, RCA_3, RCA_4, Reg_bank_8, MUX_8way_4bit,
    MUX_2way_3bit, MUX_2way_4bit.
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--
    ----------------------------------------------------------------------------------


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

entity Nano_processor is
    Port (
            Reset : in STD_LOGIC;
            Program_mode : in STD_LOGIC; -- enabling program mode for
    program ROM
            new_instruction : in STD_LOGIC_VECTOR (12 downto 0);-- new
    instruction input
            load_instruction : in STD_LOGIC; -- load new instruction into
    ROM
            NextROM : in STD_LOGIC; -- next ROM trigger
            ROM_reset : in STD_LOGIC; -- reset signal for program ROM
            Reg_to_show : in STD_LOGIC_VECTOR (2 downto 0); -- register to
    show on 7-segment display
            clk : in STD_LOGIC;
            overflow_led : out STD_LOGIC;
            zero_led : out STD_LOGIC;
            carry_led : out STD_LOGIC;
            reset_led : out STD_LOGIC;
            load_instruction_led : out STD_LOGIC;
            error_led : out STD_LOGIC;
```

```
           Reg_led : out std_logic_vector(3 downto 0);
           Seg7_digit : out std_logic_vector(6 downto 0);
           Anode : out std_logic_vector(3 downto 0)
         );
end Nano_processor;

architecture Behavioral of Nano_processor is

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

-- Uncomment the following component declaration if using LUT_16_7
component LUT_16_7
    Port (
           address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0)
         );
end component;
component Program_Rom
    Port (
           address : in STD_LOGIC_VECTOR (2 downto 0);
           reset : in STD_LOGIC;
           input_instruction : in STD_LOGIC_VECTOR (12 downto 0);
           program_mode : in STD_LOGIC; -- 1 for program mode, 0 for data
    mode
           load : in STD_LOGIC; -- Load new instruction into ROM
           instruction : out STD_LOGIC_VECTOR (12 downto 0);
           clk : in STD_LOGIC; -- Clock signal
           p_signal : out STD_LOGIC; -- Program signal
           load_sig : out STD_LOGIC -- Load signal for external use
         );
end component;
component Instruction_decoder
    Port (
           instruction : in STD_LOGIC_VECTOR (12 downto 0);
           val_MUX_0 : in std_logic_vector (3 downto 0);
           val_MUX_1 : in std_logic_vector (3 downto 0);
           reg_en : out STD_LOGIC_VECTOR (2 downto 0);
           load_sel : out STD_LOGIC;
           value : out STD_LOGIC_VECTOR (3 downto 0);
           reg_a : out STD_LOGIC_VECTOR (2 downto 0);
           reg_b : out STD_LOGIC_VECTOR (2 downto 0);
           addORsub : out STD_LOGIC;
           jump_flag : out STD_LOGIC;
           error : out STD_LOGIC;
           jump_address : out STD_LOGIC_VECTOR (2 downto 0)
         );
end component;
component RCA_3
    Port (
           a0, a1, a2 : in STD_LOGIC; -- 3-bit input A
           b0, b1, b2 : in STD_LOGIC; -- 3-bit input B
           s0, s1, s2 : out STD_LOGIC -- 3-bit sum output
         );
end component;
```

```
component MUX_2way_3bit
    Port (
           JumpFlag : in STD_LOGIC;
           JumpAdd : in STD_LOGIC_VECTOR (2 downto 0);
           SeqAdd : in STD_LOGIC_VECTOR (2 downto 0);
           NextPC : out STD_LOGIC_VECTOR (2 downto 0)
         );
end component;
component MUX_2way_4bit
    Port (
           loadSel : in STD_LOGIC;
           ImmedVal : in STD_LOGIC_VECTOR (3 downto 0);
           ALUVal : in STD_LOGIC_VECTOR (3 downto 0);
           OutputVal : out STD_LOGIC_VECTOR (3 downto 0)
         );
end component;
component RCA_4
    Port (
           a0, a1, a2, a3 : in STD_LOGIC; -- 4-bit input A
           b0, b1, b2, b3 : in STD_LOGIC; -- 4-bit input B
           mode : in STD_LOGIC;           -- Add/Subtract mode (0 for add,
     1 for subtract)
           s0, s1, s2, s3 : out STD_LOGIC; -- 4-bit sum output
           c_out : out STD_LOGIC;          -- Carry-out
           overflow : out STD_LOGIC        -- Overflow flag
         );
end component;
component Reg_bank_8
    Port (
           RegSel : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;                   -- Clock signal
           Reset : in STD_LOGIC;                 -- Reset signal
           Input: in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit input data
           reset_sig : out STD_LOGIC;              -- Reset signal for
    external use
           Out_0, Out_1, Out_2, Out_3, Out_4, Out_5, Out_6, Out_7 : out
    STD_LOGIC_VECTOR (3 downto 0) -- 4-bit output data
         );
end component;
component MUX_8way_4bit
    Port (
           R0 : in STD_LOGIC_VECTOR (3 downto 0);
           R1 : in STD_LOGIC_VECTOR (3 downto 0);
           R2 : in STD_LOGIC_VECTOR (3 downto 0);
           R3 : in STD_LOGIC_VECTOR (3 downto 0);
           R4 : in STD_LOGIC_VECTOR (3 downto 0);
           R5 : in STD_LOGIC_VECTOR (3 downto 0);
           R6 : in STD_LOGIC_VECTOR (3 downto 0);
           R7 : in STD_LOGIC_VECTOR (3 downto 0);
           RegSel : in STD_LOGIC_VECTOR (2 downto 0);
           RegVal : out STD_LOGIC_VECTOR (3 downto 0)
         );
end component;
component Slow_Clk
    Port (
           Clk_in : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Stop : in STD_LOGIC;
           Clk_out : out STD_LOGIC
         );
end component;
```

```
component Program_mode_reg
    Port (
        En     : in  STD_LOGIC;
        Clk    : in  STD_LOGIC;
        Q      : out STD_LOGIC
    );
end component;


-- For Clock
signal slow_clock : STD_LOGIC;
--To stop the clock when in program mode
signal Program_mode_sync : std_logic := '0';
signal Program_mode_sync2 : std_logic := '0';
signal Program_trig : std_logic := '0';

--For Program Counter
signal RomEn : STD_LOGIC_VECTOR (2 downto 0);
signal Mux_out : STD_LOGIC_VECTOR (2 downto 0);
signal PC_clock : STD_LOGIC;

-- Signal to store the previous state of NextROM
signal NextROM_prev : std_logic := '0';
signal NextROM_pulse : std_logic := '0';

--For Instruction Decoder
signal instruction : STD_LOGIC_VECTOR (12 downto 0);
signal reg_en : STD_LOGIC_VECTOR (2 downto 0);
signal load_sel : STD_LOGIC;
signal value : STD_LOGIC_VECTOR (3 downto 0);
signal reg_a : STD_LOGIC_VECTOR (2 downto 0);
signal reg_b : STD_LOGIC_VECTOR (2 downto 0);
signal addORsub : STD_LOGIC;
signal jump_flag : STD_LOGIC;
signal jump_address : STD_LOGIC_VECTOR (2 downto 0);

-- For RCA_3
signal Nxt_pc : std_logic_vector (2 downto 0);

--For RCA_4
signal ALUVal : STD_LOGIC_VECTOR (3 downto 0);
signal RegVal_0, RegVal_1 : STD_LOGIC_VECTOR (3 downto 0);

--For Reg Bank
signal OutputVal : STD_LOGIC_VECTOR (3 downto 0);
signal reset_led_sig : STD_LOGIC:='0';
signal Out_0, Out_1, Out_2, Out_3, Out_4, Out_5, Out_6, Out_7 :
    STD_LOGIC_VECTOR (3 downto 0);
-- signal reset_led_sig : STD_LOGIC;

--For 7_seg
signal refresh_counter : std_logic_vector(15 downto 0) := (others => '0');
signal digit_select : std_logic_vector(1 downto 0) := "00"; -- cycles 0 to
     3
signal Reg_showed : std_logic_vector(3 downto 0) ; -- register to show on
    7-segment display
signal P_signal_in : std_logic; -- program signal from program ROM

-- signal anode : std_logic_vector(3 downto 0);
-- signal digits : std_logic_vector(15 downto 0) := (others => '0'); -- 4
    digits = 4x4 bits
```

```
signal address_7 : std_logic_vector(3 downto 0);
signal data_7 : std_logic_vector(6 downto 0);
signal seg_out : std_logic_vector(6 downto 0);

--For Rom
signal load_instruction_sig : std_logic:='0';

begin

--
    -------------------------------------------------------------------------------------

--slow_clk_out <= slow_clock;  -- Add this line to expose slow_clock
    externally
--
    -------------------------------------------------------------------------------------


    -- Instantiate the components
    Slow_Clk_inst : Slow_Clk
        port map (
            Clk_in => clk,
            Stop => Program_mode_sync2, -- stop the clock when in program
    mode
            Reset => Reset,
            Clk_out => slow_clock
        );

    Program_counter_inst : Program_counter
        port map (
            Mux_out => Mux_out,
            Clk => PC_clock, --If NextROM is high, the clock will be used
    to update the program counter
            Res => Reset,
            Q => RomEn
        );


-- 7 segmentation function added
    LUT_16_7_inst : LUT_16_7
        port map (
            address => address_7,
            data => data_7
        );

    Program_Rom_inst : Program_Rom
        port map (
            clk => clk, -- connect the clock
            address => RomEn,
            reset => ROM_reset,
            input_instruction => new_instruction,
            program_mode => Program_mode_sync2,
            load => load_instruction,
            instruction => instruction,
            p_signal => P_signal_in,
            load_sig => load_instruction_sig
        );

    Instruction_decoder_inst : Instruction_decoder
        port map (
            instruction => instruction,
            val_MUX_0 => RegVal_0,
            val_MUX_1 => RegVal_1,
            reg_en => reg_en,
            load_sel => load_sel,
            value => value,
            reg_a => reg_a,
            reg_b => reg_b,
            addORsub => addORsub,
            jump_flag => jump_flag,
            error => error_led,
            jump_address => jump_address
        );

    RCA_3_inst : RCA_3
        port map (
            a0 => RomEn(0),
            a1 => RomEn(1),
            a2 => RomEn(2),
            b0 => '1',
            b1 => '0',
            b2 => '0',
            s0 => Nxt_pc(0),
            s1 => Nxt_pc(1),
            s2 => Nxt_pc(2)
        );

    MUX_2way_3bit_inst : MUX_2way_3bit
        port map (
           JumpFlag => jump_flag,
           JumpAdd => jump_address,
           SeqAdd => Nxt_pc,
           NextPC => Mux_out
         );

    MUX_2way_4bit_inst : MUX_2way_4bit
        port map (
           loadSel => load_sel,
           ImmedVal => value,
           ALUVal => ALUVal,
           OutputVal => OutputVal
         );

    RCA_4_inst : RCA_4
        port map (
              a0 => RegVal_0(0),
              a1 => RegVal_0(1),
              a2 => RegVal_0(2),
              a3 => RegVal_0(3),
              b0 => RegVal_1(0),
              b1 => RegVal_1(1),
              b2 => RegVal_1(2),
              b3 => RegVal_1(3),
              mode => addORsub,
              s0 => ALUVal(0),
              s1 => ALUVal(1),
              s2 => ALUVal(2),
              s3 => ALUVal(3),
              c_out => carry_led,
              overflow => overflow_led
         );
    Reg_bank_8_inst : Reg_bank_8
        port map (
           RegSel => reg_en,
           Clk => slow_clock,
           Reset => Reset,
           Input => OutputVal,
           Out_0 => Out_0,
           Out_1 => Out_1,
           Out_2 => Out_2,
           Out_3 => Out_3,
           Out_4 => Out_4,
           Out_5 => Out_5,
           Out_6 => Out_6,
           Out_7 => Out_7,
           reset_sig => reset_led_sig
          );
    MUX_8way_4bit_0 : MUX_8way_4bit
        port map (
              R0 => Out_0,
              R1 => Out_1,
              R2 => Out_2,
              R3 => Out_3,
              R4 => Out_4,
              R5 => Out_5,
              R6 => Out_6,
              R7 => Out_7,
              RegSel => reg_a,
              RegVal => RegVal_0
          );
    MUX_8way_4bit_1 : MUX_8way_4bit
        port map (
           R0 => Out_0,
           R1 => Out_1,
           R2 => Out_2,
           R3 => Out_3,
           R4 => Out_4,
           R5 => Out_5,
           R6 => Out_6,
           R7 => Out_7,
           RegSel => reg_b,
           RegVal => RegVal_1
          );

    Program_mode_reg_inst : Program_mode_reg
        port map (
            En => Program_mode,
            Clk => clk,
            Q => Program_trig
        );
--


-- Reset led
    reset_led <= reset_led_sig;

-- Process to synchronize the Program_mode signal
    process(clk)
    begin
        if rising_edge(clk) then
            Program_mode_sync <= Program_trig;
            Program_mode_sync2 <= Program_mode_sync;
        end if;
    end process;

-- Process to generate a clock pulse on the rising edge of NextROM
    process(clk)
    begin
        if rising_edge(clk) then
            -- Detect rising edge of NextROM
            if NextROM = '1' and NextROM_prev = '0' then
                NextROM_pulse <= '1'; -- Generate a pulse
            else
                NextROM_pulse <= '0'; -- Default state
            end if;

            -- Update the previous state of NextROM
            NextROM_prev <= NextROM;
        end if;
    end process;

--PC clock fix
    PC_clock <= slow_clock or NextROM_pulse;

-- For LED display
    zero_led <= '1' when (ALUVal = "0000") else '0';
    Reg_led <= Reg_showed;


--For Load instruction led
load_instruction_led <= load_instruction_sig;

--For 7_seg

    process(clk)
    begin
        if rising_edge(clk) then
            refresh_counter <= std_logic_vector(unsigned(refresh_counter)
    + 1);

            if refresh_counter = X"FFFF" then
                digit_select <= std_logic_vector(unsigned(digit_select) +
   1);
            end if;
        end if;
    end process;

    -- Display the selected register on the 7-segment display
    process(Reg_to_show)
    begin
        case Reg_to_show is
            when "000" => Reg_showed <= Out_0; -- Display Reg0
            when "001" => Reg_showed <= Out_1; -- Display Reg1
            when "010" => Reg_showed <= Out_2; -- Display Reg2
            when "011" => Reg_showed <= Out_3; -- Display Reg3
            when "100" => Reg_showed <= Out_4; -- Display Reg4
            when "101" => Reg_showed <= Out_5; -- Display Reg5
            when "110" => Reg_showed <= Out_6; -- Display Reg6
            when others => Reg_showed <= Out_7; -- Display Reg7
        end case;
    end process;

    process(digit_select, P_signal_in, Reg_showed, data_7, RomEn)
    begin
        case digit_select is
            when "00" =>
                address_7 <= Reg_showed;
                seg_out <= data_7;
                Anode <= "1110";
            when "10" =>
                if P_signal_in = '1' then
                    seg_out <= "0001100"; -- "P"
                    Anode <= "1011";
                else
                    Anode <= "1111";
                end if;
            when "11" =>
                address_7 <= ('0' & RomEn);
                seg_out <= data_7;
                Anode <= "0111";
            when others =>
                seg_out <= (others => '1');
                Anode <= "1111";
        end case;
    end process;

    -- Output ports
    Seg7_digit <= seg_out;




end Behavioral;
```

VHDL Simulation Source Code

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

entity Nano_processor_tb is
end Nano_processor_tb;

architecture Behavioral of Nano_processor_tb is

    -- Component Declaration
    component Nano_processor
        Port (
            Reset : in STD_LOGIC;
            Program_mode : in STD_LOGIC;
            new_instruction : in STD_LOGIC_VECTOR (11 downto 0);
            load_instruction : in STD_LOGIC;
            NextROM : in STD_LOGIC;
            clk : in STD_LOGIC;
            overflow_led : out STD_LOGIC;
            zero_led : out STD_LOGIC;
            carry_led : out STD_LOGIC;
            Reg7_led : out STD_LOGIC_VECTOR(3 downto 0);
            Seg7_digit : out STD_LOGIC_VECTOR(6 downto 0);
            Anode : out STD_LOGIC_VECTOR(3 downto 0)
--              slow_clk_out : out std_logic   -- <---- Add this line
        );
    end component;
    -- Testbench Signals
    signal clk_tb              : STD_LOGIC := '0';
    signal Reset_tb            : STD_LOGIC := '0';
    signal Program_mode_tb     : STD_LOGIC := '0';
    signal new_instruction_tb : STD_LOGIC_VECTOR(11 downto 0) := (others
   => '0');
    signal load_instruction_tb: STD_LOGIC := '0';
    signal NextROM_tb          : STD_LOGIC := '0';
    signal overflow_led_tb     : STD_LOGIC;
    signal zero_led_tb         : STD_LOGIC;
    signal carry_led_tb        : STD_LOGIC;
    signal Reg7_led_tb         : STD_LOGIC_VECTOR(3 downto 0);
    signal Seg7_digit_tb       : STD_LOGIC_VECTOR(6 downto 0);
    signal Anode_tb            : STD_LOGIC_VECTOR(3 downto 0);

--     signal slow_clk_tb : std_logic;

    constant clk_period : time := 10 ns;

begin

    -- Instantiate the Nano Processor
    UUT: Nano_processor
        port map (
            Reset => Reset_tb,
            Program_mode => Program_mode_tb,
            new_instruction => new_instruction_tb,
            load_instruction => load_instruction_tb,
            NextROM => NextROM_tb,
            clk => clk_tb,
            overflow_led => overflow_led_tb,
            zero_led => zero_led_tb,
            carry_led => carry_led_tb,
            Reg7_led => Reg7_led_tb,
            Seg7_digit => Seg7_digit_tb,
            Anode => Anode_tb

--              slow_clk_out => slow_clk_tb  -- connect slow clock here
        );



process begin
clk_tb<='0';
wait for 5ns;
clk_tb<='1';
wait for 5ns;
end process;

process begin
Reset_tb<='1';
wait for 110ns;
Reset_tb<='0';
wait;
end process;


end Behavioral;
```

4.13.2 Program Rom

VHDL Design Source Code

```
--
    ----------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date: 09.05.2025 02:31:33
-- Design Name:
-- Module Name: Program_Rom - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--
    ----------------------------------------------------------------------------------



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

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

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

entity Program_Rom is
    Port ( address : in STD_LOGIC_VECTOR (2 downto 0);
           clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           input_instruction : in STD_LOGIC_VECTOR (12 downto 0);
           program_mode : in STD_LOGIC; -- 1 for program mode, 0 for data
    mode
           load : in STD_LOGIC; -- Load new instruction into ROM
           load_sig : out STD_LOGIC; -- Load signal for external use
           p_signal : out STD_LOGIC; -- Program signal
           instruction : out STD_LOGIC_VECTOR (12 downto 0));


end Program_Rom;

architecture Behavioral of Program_Rom is
type rom_type is array (0 to 7) of std_logic_vector(12 downto 0);
signal program_ROM : rom_type :=(
--      "0101110000001",-- MOVI R7, 1 --0
--      "0100100000010",-- MOVI R2, 2 --1
--      "0100010000011",-- MOVI R1, 3 --2
--      "0001110100000",-- ADD R7, R2 --3
--      "0001110010000",-- ADD R7, R1 --4
--      "0000000000000",--            --5
--      "0000000000000",--            --6
--      "0000000000000"--             --7

    "0100010001010",-- MOVI R1, 10 --0
    "0100100000001",-- MOVI R2, 1  --1
    "0010100000000",-- NEG R2      --2
    "0000010100000",-- ADD R1, R2  --3
    "0110010000111",-- JZR R1, 7   --4
    "0110000000011",-- JZR R0, 3   --5
    "0000000000000",--             --6
    "0000000000000"--              --7
);
begin

    --default values
    -- load_sig <= '0';
    process(clk, reset, load, program_mode, address, input_instruction)
    begin
        if rising_edge(clk) then
            if program_mode = '1' then
                p_signal <= '1'; -- Indicate that the program is in
    programming mode
                load_sig <= '0'; -- Clear load signal
                if reset = '1' then
                    program_ROM <= (others => (others => '0')); -- Clear
    all ROM contents
                    instruction <= (others => '0'); -- Reset instruction
    output to zero
                elsif load = '1' then
                    program_ROM(to_integer(unsigned(address))) <=
    input_instruction; -- Load new instruction
                    load_sig <= '1'; -- Indicate that a new instruction is
     being loaded

                end if;
            elsif program_mode = '0' then
                load_sig <= '0'; -- Clear load signal
                p_signal <= '0'; -- Indicate that the program is in data
    mode
                instruction <= program_ROM(to_integer(unsigned(address)));
     -- Read instruction from ROM
            end if;
        end if;
    end process;

end Behavioral;
```

4.13.3 Instruction Decoder

VHDL Design Source Code

```
--
    --------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date: 09.05.2025 05:44:11
-- Design Name:
-- Module Name: Instruction_decoder - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--
    --------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Instruction Set
-- 000: ADD   -- 000 RaRaRa RbRbRb 0000    -- ADD Ra and Rb and store in
    Ra
-- 001: NEG   -- 001 RaRaRa 000 0000       -- Negate Ra
-- 010: MOVI  -- 010 RaRaRa 000 dddd       -- Move immediate value dddd to
     Ra
-- 011: JZR   -- 011 RaRaRa 000 dddd       -- Jump to address dddd if Ra
    is zero
-- 100: MOV   -- 100 RaRaRa RbRbRb 0000    -- Move value from Ra to Rb
-- 101: AND   -- 101 RaRaRa RbRbRb 0000    -- AND Ra and Rb and store in
    Ra
-- 110: OR    -- 110 RaRaRa RbRbRb 0000    -- OR Ra and Rb and store in Ra
-- 111: XOR   -- 111 RaRaRa RbRbRb 0000    -- XOR Ra and Rb and store in
    Ra

entity Instruction_decoder is
    Port ( instruction : in STD_LOGIC_VECTOR (12 downto 0);
           val_MUX_0 : in std_logic_vector (3 downto 0);
           val_MUX_1 : in STD_LOGIC_VECTOR (3 downto 0);
           reg_en : out STD_LOGIC_VECTOR (2 downto 0);
           load_sel : out STD_LOGIC;
           value : out STD_LOGIC_VECTOR (3 downto 0);
           reg_a : out STD_LOGIC_VECTOR (2 downto 0);
           reg_b : out STD_LOGIC_VECTOR (2 downto 0);
           addORsub : out STD_LOGIC;
           jump_flag : out STD_LOGIC;
           error : out STD_LOGIC;
           jump_address : out STD_LOGIC_VECTOR (2 downto 0));
```

```vhdl
end Instruction_decoder;

architecture Behavioral of Instruction_decoder is

signal processed_value : std_logic_vector (3 downto 0);

begin

    process (instruction, val_MUX_0)
    begin
        -- Default values for outputs
        reg_a <= (others => '0');
        reg_b <= (others => '0');
        reg_en <= (others => '0');
        load_sel <= '0';
        value <= (others => '0');
        addORsub <= '0';
        jump_flag <= '0';
        jump_address <= (others => '0');

        -- Decode the instruction
        if (instruction(12 downto 10) = "000") then
            -- Addition instruction
            reg_a <= instruction(9 downto 7);
            reg_b <= instruction(6 downto 4);
            load_sel <= '1';
            reg_en <= instruction(9 downto 7);
        elsif (instruction(12 downto 10) = "001") then
            -- Negation instruction
            reg_a <= "000"; -- In Reg6 must be contained '0000' otherwise
    the result will be wrong
            reg_b <= instruction(9 downto 7);
            load_sel <= '1';
            addORsub <= '1'; -- Changing mode to subtract to negate the
    value
            reg_en<=instruction(9 downto 7);
        elsif (instruction(12 downto 10) = "010") then
            -- Load instruction
            reg_en <= instruction(9 downto 7);
            load_sel <= '0';
            value <= instruction(3 downto 0);
        elsif (instruction(12 downto 10) = "011") then
            -- Jump instruction
            reg_a <= instruction(9 downto 7);
            if val_MUX_0 = "0000" then
                jump_flag <= '1';
                jump_address <= instruction(2 downto 0);
            else
                jump_flag <= '0';
            end if;
        elsif (instruction(12 downto 10) = "100") then
            -- Move instruction
            reg_a <= instruction(9 downto 7);
            reg_en <= instruction(6 downto 4);
            value <= val_MUX_0;

        elsif (instruction(12 downto 10) = "101") then
            -- AND instruction
            reg_a <= instruction(9 downto 7);
            reg_b <= instruction(6 downto 4);
            reg_en <= instruction(9 downto 7);
            processed_value <= val_MUX_0 and val_MUX_1;
            value <= processed_value;

        elsif (instruction(12 downto 10) = "110") then
            -- OR instruction
            reg_a <= instruction(9 downto 7);
            reg_b <= instruction(6 downto 4);
            reg_en <= instruction(9 downto 7);

            processed_value <= val_MUX_0 or val_MUX_1;
            value <= processed_value;
        elsif (instruction(12 downto 10) = "111") then
            -- XOR instruction
            reg_a <= instruction(9 downto 7);
            reg_b <= instruction(6 downto 4);
            reg_en <= instruction(9 downto 7);

            processed_value <= val_MUX_0 xor val_MUX_1;
            value <= processed_value;
        else
            -- Invalid instruction
            reg_a <= (others => '0');
            reg_b <= (others => '0');
            error <= '1'; -- Set error flag
        end if;

    end process;

end Behavioral;
```

05

Contribution

Individual Contributions of Each Group Member

Name	Individual Contributions	Working Hours
Ruththiragayan S.	▪ Designed instruction decoder, program ROM, register bank, 3-bit and 4-bit adders. ▪ Extended nanoprocessor with features such as programmable ROM. ▪ Debugged component integration issues and verified overall functionality.	40
Ahamed H.Y.	▪ Designed and simulated decoder, counter, and LUT (7-segment display). ▪ Collected and reviewed all timing and schematic diagrams. ▪ Added missing simulation files and organized materials for the final report and edited it.	20
Asma A.R.	▪ Designed and simulated multiplexers. ▪ Developed simulation files for nanoprocessor, instruction decoder, and program ROM. ▪ Prepared the lab report.	30
Rathnayake R.M.R.S.	▪ Designed and simulated 3-bit adder, 4-bit adder/subtractor, and register bank. ▪ Prepared the presentation.	3

06

Challenges and Solutions

During the development of the Nano Processor, one of the significant challenges encountered was the time-consuming process of regenerating the bitstream after each modification to the instruction set in the ROM. This became particularly cumbersome during testing and debugging phases, where frequent updates were necessary to verify various instruction sequences. Additionally, displaying register contents on the 7-segment display required altering the VHDL code and resynthesizing the design, limiting flexibility and slowing down the overall development process.

To address this, a programmable ROM module was implemented, enabling dynamic instruction loading without the need for bitstream regeneration. This enhancement allowed for quicker iterations and greater ease during simulation and hardware validation. Furthermore, the control logic was modified to allow manual selection of register outputs via switches on the Basys3 board. This enabled real-time monitoring of internal data paths on the 7-segment display without modifying the design code, significantly improving debugging efficiency and user interaction.

These improvements not only accelerated the development cycle but also enhanced the processor's flexibility and interactivity, making it more suitable for educational use and further experimentation.

07

Conclusions

In this lab, we successfully designed and implemented a 4-bit Nano Processor capable of executing a variety of instructions with both hardcoded and programmable ROM configurations. Starting from basic arithmetic operations, we progressively enhanced the system to support logical operations, register-to-register data transfer, and conditional branching.

One of the key achievements was the development of a programmable ROM, which allowed dynamic instruction loading through the Basys3 board interface. This eliminated the need for resynthesis when changing programs and significantly improved the system's usability and flexibility.

Another major enhancement was the addition of a real-time register monitoring system using a 7-segment display. This feature, along with dedicated control buttons and status LEDs, made the processor more interactive and suitable for educational and debugging purposes.

Throughout the project, we followed a modular design and integration approach, thoroughly testing each component using simulations before hardware implementation. The complete system was deployed on the Basys3 FPGA board and demonstrated reliable and expected behavior.

This lab gave us practical experience in digital design, VHDL development, hardware interfacing, and system-level integration. The final result is a fully functional and extensible Nano Processor that demonstrates fundamental computer architecture concepts in a clear and hands-on way.

Appendix A

Instruction Set Summary - Basic Nanoprocessor

Opcode (2-bit)	**Instruction**	**Format (12-bit)**	**Description**
00	ADD	00 RaRaRa RbRbRb 0000	Adds Ra and Rb, stores result in Ra
01	NEG	01 RRR 0000000	Negates R (2's complement)
10	MOVI	10 RRR 000 dddd	Loads immediate value dddd into R
11	JZR	11 RRR 000 ddd	Jumps to address ddd if R == 0

Appendix B

Instruction Set Summary - Enhanced Nanoprocessor

Opcode (3-bit)	**Instruction**	**Format**	**Description**
000	ADD	000 RaRaRa RbRbRb 0000	Adds Ra and Rb, stores result in Ra
001	NEG	001 RaRaRa 000 0000	Negates Ra (2's complement)
010	MOVI	010 RaRaRa 000 dddd	Loads immediate value dddd into Ra
011	JZR	011 RaRaRa 000 ddd	Jumps to ddd if Ra = 0
100	MOV	100 RaRaRa RbRbRb 0000	Moves value from Rb to Ra
101	AND	101 RaRaRa RbRbRb 0000	Bitwise AND (Ra = Ra AND Rb)
110	OR	110 RaRaRa RbRbRb 0000	Bitwise OR (Ra = Ra OR Rb)
111	XOR	111 RaRaRa RbRbRb 0000	Bitwise XOR (Ra = Ra XOR Rb)