# LAB 9-10 :
# NANOPROCESSOR DESIGN
# LAB REPORT

## Computer Organization and Digital Design



**GROUP 44**

- **ATHTHANAYAKE A.M.R.N. 230062V**
- **IFAZ M.I.M. 230253H**
- **KUMARASINGHE M.P. 230356C**
- **LAWANYA K.K.H.G. 230373B**

## Assigned Lab Task

The assigned lab task is to design and develop a 4-bit nanoprocessor capable of executing 4 instructions. This includes several components that need to be developed or extended from the previous labs:

- **4-bit Add/Subtract unit**: Capable of adding and subtracting numbers represented using 2's complement.
- **3-bit Adder:** Used to increment the Program Counter.
- **3-bit Program Counter (PC):** Built using D Flip Flops with a clear/reset input.
- **k-way b-bit multiplexers:** These multiplexers offer versatile data routing capabilities within the processor. They can select one of multiple input sources based on control signals, making them invaluable for data manipulation and routing tasks.
- **Register Bank:** Contains 8, 4-bit registers (R0 to R7), with R0 hardcoded to all 0s.
- **Program ROM:** The Program ROM serves as the storage unit for our assembly program. It houses the instructions that dictate the sequence of operations to be executed by the processor.
- **Buses:** Buses are used to transfer data and instructions between different components of the processor. They facilitate communication between the Program ROM, Instruction Decoder, and other units.
- **Instruction Decoder:** Acting as the brain of the processor, the Instruction Decoder interprets instructions fetched from ROM. It deciphers their meaning and provides the necessary control signals to guide the execution of tasks within the processor.

Apart from the above components we decided to improve the nano processor by allowing it to support more instructions using the component below.

## Bitwise Operator

The **Bitwise Operator** unit performs fundamental bitwise operations on two 4-bit binary inputs. Controlled by a 2-bit selection signal, it applies one of four bitwise operations **NOT**, **AND**, **OR**, or **XOR** to the input values. This module is essential for logic-level manipulation within a processor or digital system, enabling low-level computation and decision-making based on individual bits.

- `"00"` → **NOT B** (bitwise complement of B)
- `"01"` → **A AND B** (logical AND between corresponding bits)
- `"10"` → **A OR B** (logical OR between corresponding bits)
- `"11"` → **A XOR B** (logical exclusive OR between corresponding bits)

The result is output as a 4-bit vector, `Bit Out`, allowing downstream components to act based on logical computation results. This module supports modular ALU designs and logic instruction handling in custom processors.

## Assembly code Instructions for the Program ROM

| Instruction | Description | Format (12-bit instruction) |
|---|---|---|
| MOVI R, *d* | Move immediate value *d* to register R, i.e., $R \leftarrow d$<br>$R \in [0, 7]$, $d \in [0, 15]$ | 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<br>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<br>R ∈ [0, 7] | 0 1 R R R 0 0 0 0 0 0 0 |
| JZR R, d | Jump if value in register R is 0, i.e.,<br>If R == 0<br>PC ← d;<br>Else<br>PC ← PC + 1;<br>R ∈ [0, 7], *d* ∈ [0, 7] | 1 1 R R R 0 0 0 0 d d d |

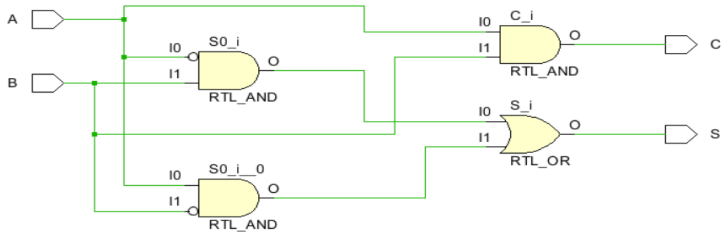
| Opcode | Instruction | Description | Format<br>(13-bit instruction) |
|---|---|---|---|
| 010 | MOVI R,d | Move immediate value d to register R<br><br>R ← d | 010 RRR 0000 dddd |
| 000 | ADD Ra, Rb | Add values in Ra and Rb, store result in Ra<br><br>Ra ← Ra + Rb | 000 RaRaRa RRR 0000 |
| 001 | NEG R | Two's complement of R<br><br>R ← -R | 001 RRR 0000 0000 |
| 011 | JZR R,d | Jump to d if R == 0,<br>else PC ← PC + 1 | 011 RRR 0000 aaa |
| 100 | NOT R<br>(Bitwise NOT) | R ← NOT R (Bitwise complement) | 100 RRR 0000 0000 |
| 101 | AND Ra, Rb | Ra ← Ra AND Rb | 101 RRR RRR 0000 |
| 110 | OR Ra, Rb | Ra ← Ra OR Rb | 110 RRR RRR 0000 |
| 111 | XOR Ra, Rb | Ra ← Ra XOR Rb | 111 RRR RRR 0000 |

# VHDL Design Codes, Design Schematics, Test Bench Codes and Timing Diagrams of the Components

## Half Adder-Design

```
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

signal not_A : std_logic;
signal not_B : std_logic;

begin

not_A <= NOT A;
not_B <= NOT B;

S <= (not_A AND B) OR (A AND not_B);
C <= A AND B;

end Behavioral;
```

## Half Adder - Design Schematic



## Half Adder - Test Bench

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

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

architecture Behavioral of HA_TB is

    component HA
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           S : out STD_LOGIC;
           C : out STD_LOGIC);
    end component;

    signal A_tb : STD_LOGIC := '0';
    signal B_tb : STD_LOGIC := '0';
    signal S_tb : STD_LOGIC;
    signal C_tb : STD_LOGIC;

begin

    uut: HA Port map (
    A => A_tb,
    B => B_tb,
    S => S_tb,
    C => C_tb
    );

    stim_proc: process
    begin
    --230373 => 111000001111100101
    -- Test case 0 + 0
    A_tb <= '0';
    B_tb <= '0';
    wait for 10 ns;

    -- Test case 0 + 1
    A_tb <= '0';
    B_tb <= '1';
    wait for 10 ns;

    -- Test case 1 + 0
    A_tb <= '1';
    B_tb <= '0';
    wait for 10 ns;

    -- Test case 1 + 1
```

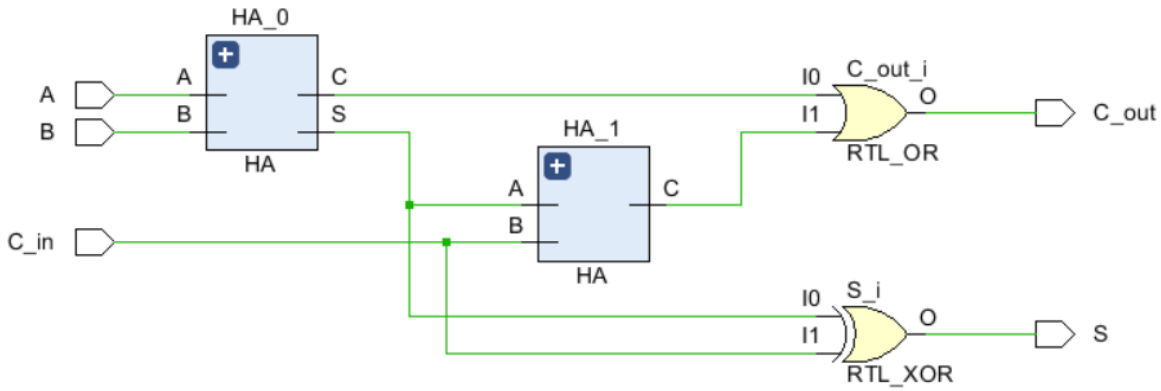
```
        A_tb <= '1';
        B_tb <= '1';
        wait for 10 ns;

        wait;
        end process;


end Behavioral;
```

## Half Adder - Timing Diagram

## Full Adder - Design

```
library IEEE;
use IEEE.STD_LOGIC_1164.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);
C_out<=HA0_C OR HA1_C;
S <= HA0_S XOR C_in;

end Behavioral;
```

## Full Adder - Design Schematic

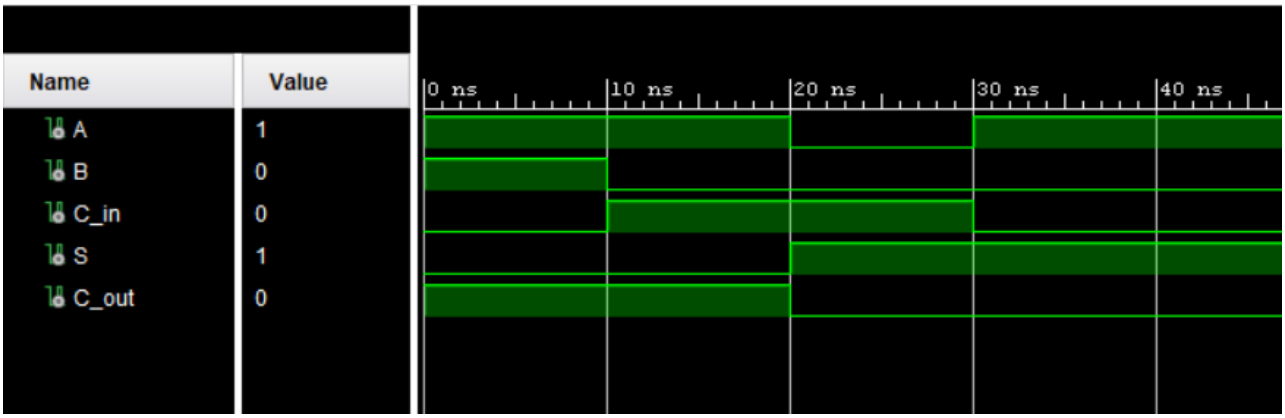



## Full adder - Test Bench

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

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

 architecture Behavioral of FA_TB 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 A   : STD_LOGIC := '0';
      signal B   : STD_LOGIC := '0';
      signal C_in  : STD_LOGIC := '0';
      signal S   : STD_LOGIC;
      signal C_out : STD_LOGIC;

 begin

      uut: FA
      Port map (
            A     => A,
            B     => B,
            C_in  => C_in,
```

```
        S     => S,
        C_out => C_out
    );

    stim_proc: process
    begin
    --230062 => 11 1000 0010 1010 1110
    -- First input: 1110 ? A=1, B=1, C_in=0
    A     <= '1';
    B     <= '1';
    C_in  <= '0';
    wait for 10 ns;

    -- Second input: 1010 ? A=1, B=0, C_in=1
    A     <= '1';
    B     <= '0';
    C_in  <= '1';
       wait for 10 ns;

    -- Third input: 0010 ? A=0, B=0, C_in=1
    A     <= '0';
    B     <= '0';
    C_in  <= '1';
    wait for 10 ns;

    -- Fourth input: 1000 ? A=1, B=0, C_in=0
    A     <= '1';
    B     <= '0';
    C_in  <= '0';
    wait for 10 ns;

    wait;
    end process;

end Behavioral;
```

## Full Adder - Timing Diagram

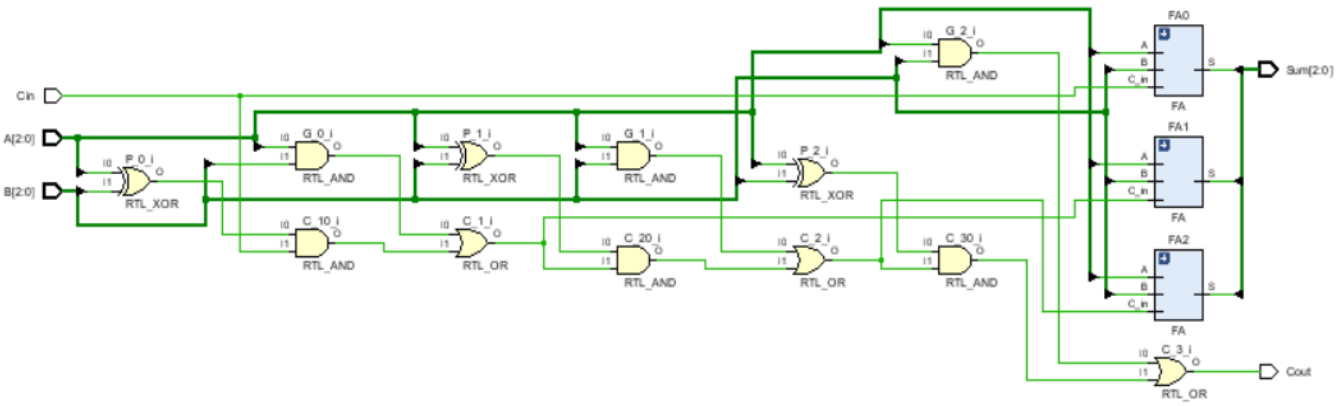


## 3 bit Adder- Design

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

entity CLA_3bit is
    Port (
        A     : in  STD_LOGIC_VECTOR(2 downto 0);
        B     : in  STD_LOGIC_VECTOR(2 downto 0);
        Cin   : in  STD_LOGIC;
        Sum   : out STD_LOGIC_VECTOR(2 downto 0);
        Cout  : out STD_LOGIC
    );
end CLA_3bit;

architecture Behavioral of CLA_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 P, G : STD_LOGIC_VECTOR(2 downto 0); -- propagate and
generate
    signal C    : STD_LOGIC_VECTOR(3 downto 0); -- carries, C(0) = Cin

begin

    C(0) <= Cin;

    -- Propagate and generate signals
    P(0) <= A(0) xor B(0);
    P(1) <= A(1) xor B(1);
    P(2) <= A(2) xor B(2);

    G(0) <= A(0) and B(0);
    G(1) <= A(1) and B(1);
    G(2) <= A(2) and B(2);

    -- Carry Lookahead logic
    C(1) <= G(0) or (P(0) and C(0));
    C(2) <= G(1) or (P(1) and C(1));
    C(3) <= G(2) or (P(2) and C(2));

    -- Full Adders with explicit named port mapping
    FA0: FA port map (
```

```
        A      => A(0),
        B      => B(0),
        C_in   => C(0),
        S      => Sum(0),
        C_out => open
    );

    FA1: FA port map (
        A      => A(1),
        B      => B(1),
        C_in   => C(1),
        S      => Sum(1),
        C_out => open
    );

    FA2: FA port map (
        A      => A(2),
        B      => B(2),
        C_in   => C(2),
        S      => Sum(2),
        C_out => open
    );

    Cout <= C(3);

end Behavioral;
```

## 3 bit Adder - Design Schematic

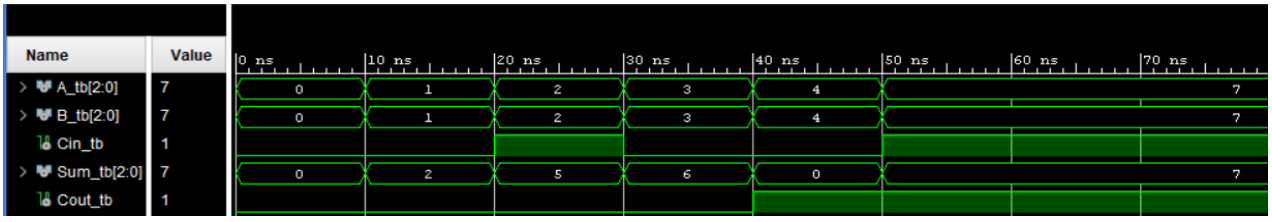


## 3 bit Adder - Test Bench

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

entity CLA_3bit_tb is
end CLA_3bit_tb;

architecture Behavioral of CLA_3bit_tb is

    -- Component Declaration of the Unit Under Test (UUT)
    component CLA_3bit
        Port (
            A     : in  STD_LOGIC_VECTOR(2 downto 0);
            B     : in  STD_LOGIC_VECTOR(2 downto 0);
            Cin   : in  STD_LOGIC;
            Sum   : out STD_LOGIC_VECTOR(2 downto 0);
            Cout  : out STD_LOGIC
        );
    end component;

    -- Testbench signals
    signal A_tb    : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal B_tb    : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal Cin_tb  : STD_LOGIC := '0';
    signal Sum_tb  : STD_LOGIC_VECTOR(2 downto 0);
    signal Cout_tb : STD_LOGIC;

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: CLA_3bit
        port map (
            A     => A_tb,
            B     => B_tb,
            Cin   => Cin_tb,
            Sum   => Sum_tb,
            Cout  => Cout_tb
        );

    -- Stimulus process
    stim_proc: process
    begin
        -- Test vector 1: 000 + 000 + 0 = 000
        A_tb <= "000"; B_tb <= "000"; Cin_tb <= '0';
        wait for 10 ns;

        -- Test vector 2: 001 + 001 + 0 = 010
        A_tb <= "001"; B_tb <= "001"; Cin_tb <= '0';
        wait for 10 ns;
```

```
        -- Test vector 3: 010 + 010 + 1 = 101
        A_tb <= "010"; B_tb <= "010"; Cin_tb <= '1';
        wait for 10 ns;

        -- Test vector 4: 011 + 011 + 0 = 110
        A_tb <= "011"; B_tb <= "011"; Cin_tb <= '0';
        wait for 10 ns;

        -- Test vector 5: 100 + 100 + 0 = 000 (overflow)
        A_tb <= "100"; B_tb <= "100"; Cin_tb <= '0';
        wait for 10 ns;

        -- Test vector 6: 111 + 111 + 1 = 111 (Sum), Cout = 1
        A_tb <= "111"; B_tb <= "111"; Cin_tb <= '1';
        wait for 10 ns;

        -- End simulation
        wait;
    end process;

end Behavioral;
```

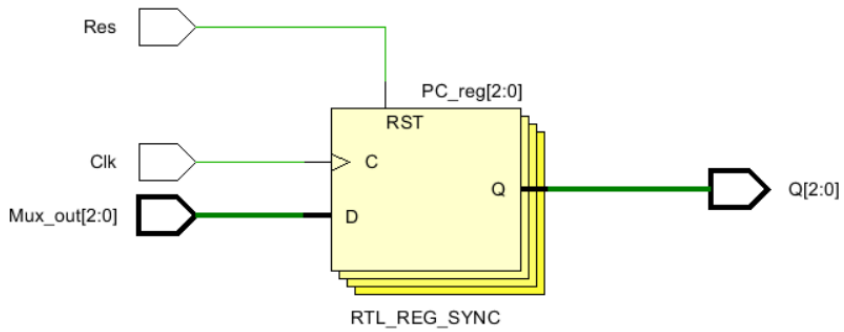
## 3 bit Adder - Timing Diagram



## Program Counter - Design

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

entity Program_Counter_3bit is
    Port (
        Mux_out : in  STD_LOGIC_VECTOR(2 downto 0);  -- Next PC value
        Clk     : in  STD_LOGIC;
        Res     : in  STD_LOGIC;
        Q       : out STD_LOGIC_VECTOR(2 downto 0)  -- Current PC
value
       -- Qbar    : out STD_LOGIC_VECTOR(2 downto 0)   -- Inverted PC
value
    );
end Program_Counter_3bit;

architecture Behavioral of Program_Counter_3bit is
    signal PC : STD_LOGIC_VECTOR(2 downto 0) := "000";
begin

    process(Clk, Res)
    begin
        if rising_edge(Clk) then
              if Res = '1' then
                 PC <= "000";
              else
                 PC <= Mux_out;
              end if;
        end if;
    end process;

    Q    <= PC;
   -- Qbar <= not PC;

end Behavioral;
```

## Program Counter - Design Schematic

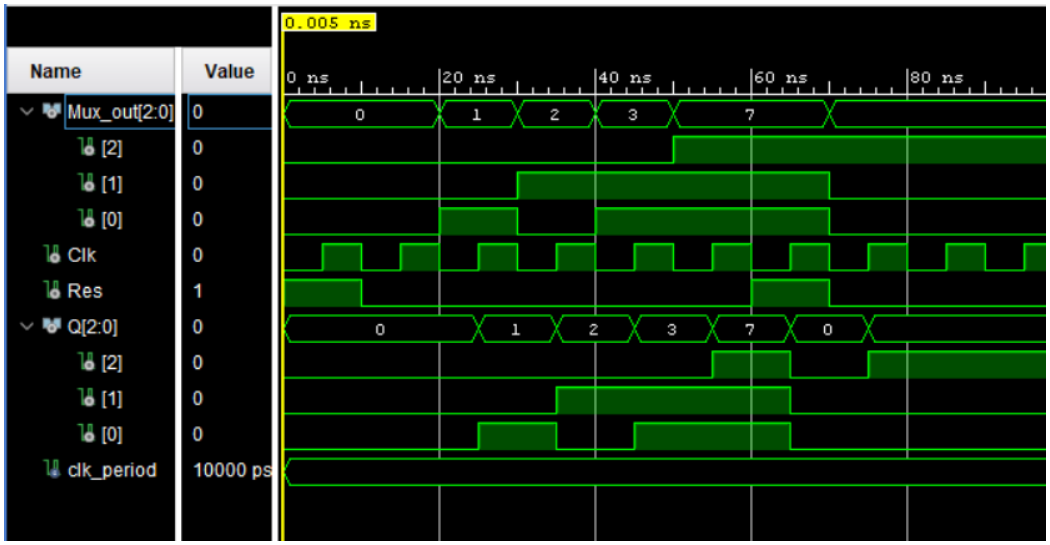


## Program Counter - Test Bench

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

entity Program_Counter_3bit_tb is
end Program_Counter_3bit_tb;

architecture behavior of Program_Counter_3bit_tb is

    -- Component Declaration
    component Program_Counter_3bit 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 component;

    -- Testbench signals
    signal Mux_out : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal Clk     : STD_LOGIC := '0';
    signal Res     : STD_LOGIC := '0';
    signal Q       : STD_LOGIC_VECTOR(2 downto 0);

    constant clk_period : time := 10 ns;

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: Program_Counter_3bit
        port map (
            Mux_out => Mux_out,
            Clk     => Clk,
            Res     => Res,
            Q       => Q
        );

    -- Clock generation process
    clk_process :process
    begin
        while true loop
            Clk <= '0';
            wait for clk_period/2;
            Clk <= '1';
            wait for clk_period/2;
        end loop;
    end process;

    -- Stimulus process
```

```
    stim_proc: process
    begin
        -- Reset the PC
        Res <= '1';
        wait for clk_period;
        Res <= '0';

        -- Apply test values
        wait for clk_period;
        Mux_out <= "001"; -- PC = 001
        wait for clk_period;
        Mux_out <= "010"; -- PC = 010
        wait for clk_period;
        Mux_out <= "011"; -- PC = 011
        wait for clk_period;
        Mux_out <= "111"; -- PC = 111
        wait for clk_period;

        -- Trigger reset again
        Res <= '1';
        wait for clk_period;
        Res <= '0';

        -- Apply another value
        Mux_out <= "100";
        wait for clk_period;

        wait; -- Wait forever
    end process;

end behavior;
```

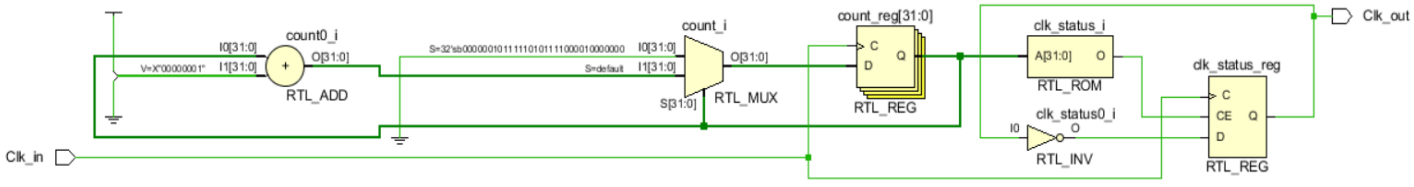
## Program Counter - Timing Diagram



## Slow clock - Design

```
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 := 0;
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;
                  count <= 0;
            end if;
      end if;
      end process;
 Clk_out <= clk_status;

end Behavioral;
```

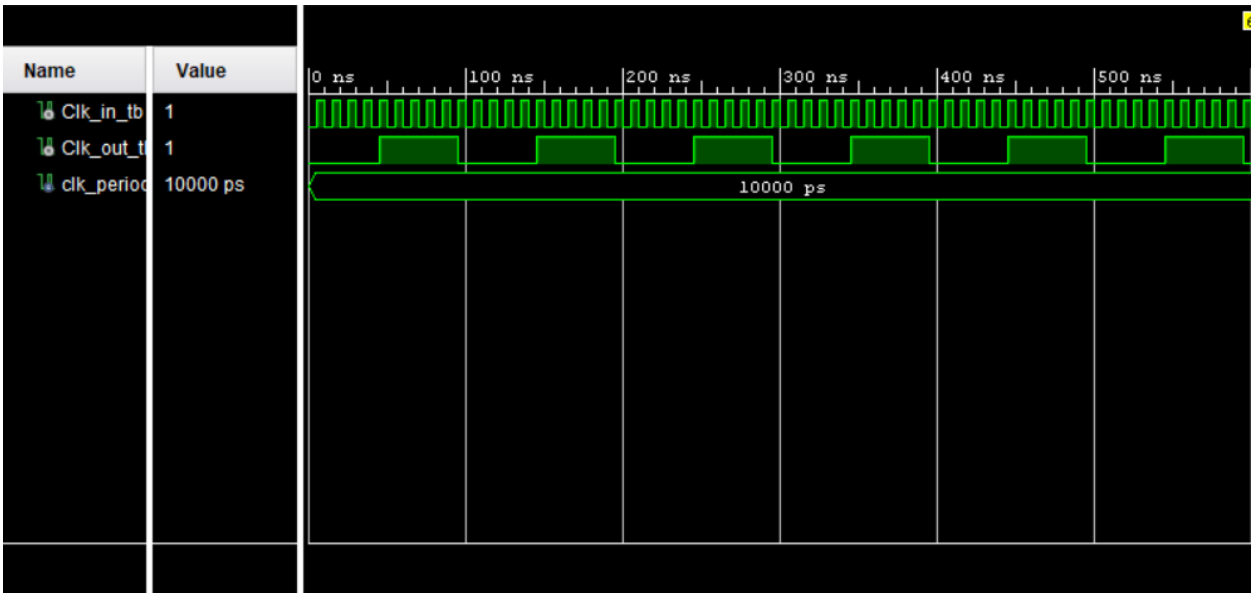
## Slow clock - Design Schematic



## Slow clock - Test Bench

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

 entity Slow_Clock_TB is
 end Slow_Clock_TB;

 architecture Behavioral of Slow_Clock_TB 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;  -- 100 MHz clock

 begin

      uut: Slow_Clk port map (
      Clk_in  => Clk_in_tb,
      Clk_out => Clk_out_tb
     );

      clk_process: process
      begin
      while true loop
            Clk_in_tb <= '0';
            wait for clk_period / 2;
            Clk_in_tb <= '1';
            wait for clk_period / 2;
      end loop;
      end process;

      stim_proc: process
      begin
      wait for 2 ms;
      wait;
      end process;
end Behavioral;
```

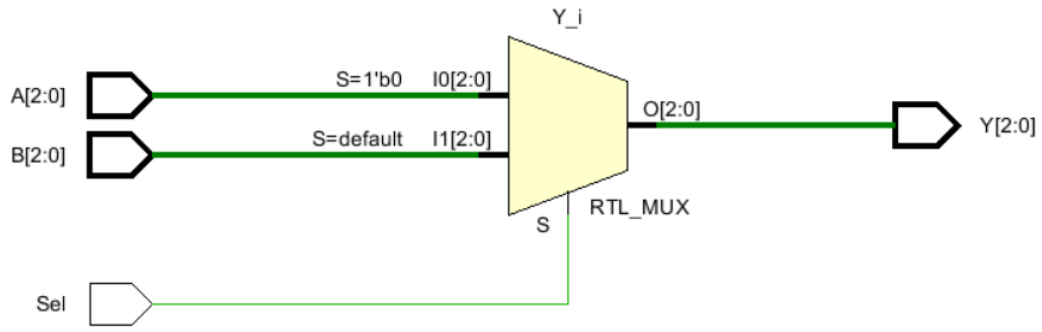
## Slow down Clock - Timing Diagram



## 2-way 3-bit Multiplexer - Design

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

entity Mux2_3bit is
    Port (
        A : in STD_LOGIC_VECTOR(2 downto 0);  -- Input 0
        B : in STD_LOGIC_VECTOR(2 downto 0);  -- Input 1
        Sel : in STD_LOGIC;                   -- Select line
        Y : out STD_LOGIC_VECTOR(2 downto 0)  -- Output
    );
end Mux2_3bit;

architecture Behavioral of Mux2_3bit is
begin
    process (A, B, Sel)
    begin
        if Sel = '0' then
            Y <= A;
        else
            Y <= B;
        end if;
    end process;
end Behavioral;
```

## 2-way 3-bit Multiplexer - Design Schematic

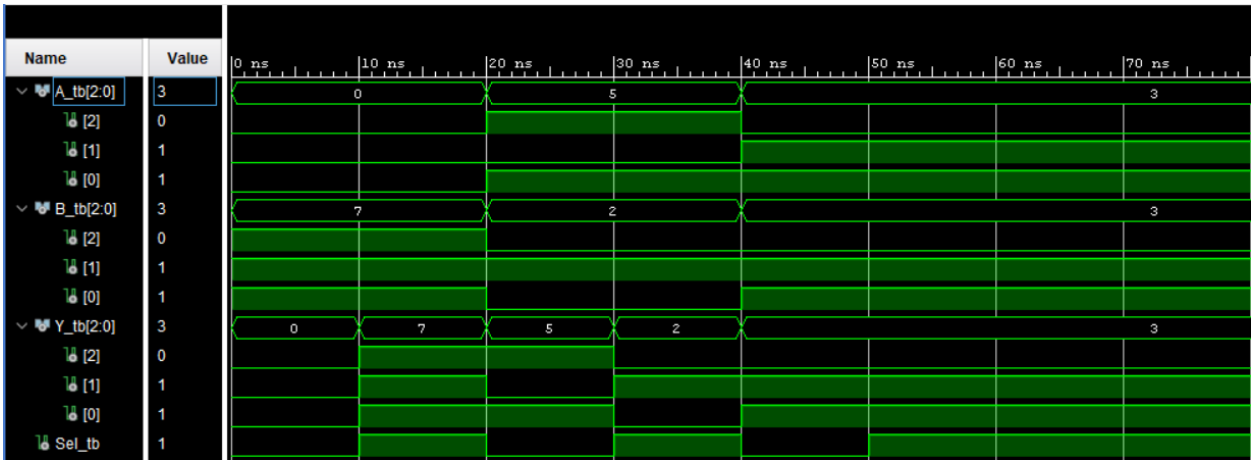


## 2-way 3-bit Multiplexer - Test Bench

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

entity Mux2_3bit_tb is
end Mux2_3bit_tb;

architecture Behavioral of Mux2_3bit_tb is

    -- Component declaration
    component Mux2_3bit
        Port (
            A   : in  STD_LOGIC_VECTOR(2 downto 0);
            B   : in  STD_LOGIC_VECTOR(2 downto 0);
            Sel : in  STD_LOGIC;
            Y   : out STD_LOGIC_VECTOR(2 downto 0)
        );
    end component;

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

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: Mux2_3bit
        port map (
            A   => A_tb,
            B   => B_tb,
            Sel => Sel_tb,
            Y   => Y_tb
        );

    -- Stimulus process
    stim_proc: process
    begin
        -- Test 1: Select A
        A_tb <= "000"; B_tb <= "111"; Sel_tb <= '0';
        wait for 10 ns;

        -- Test 2: Select B
        Sel_tb <= '1';
        wait for 10 ns;

        -- Test 3: Change inputs
        A_tb <= "101"; B_tb <= "010"; Sel_tb <= '0';
        wait for 10 ns;

        Sel_tb <= '1';
```

```
        wait for 10 ns;

        -- Test 4: Same values on A and B
        A_tb <= "011"; B_tb <= "011"; Sel_tb <= '0';
        wait for 10 ns;

        Sel_tb <= '1';
        wait for 10 ns;

        -- End simulation
        wait;
    end process;

end Behavioral;
```

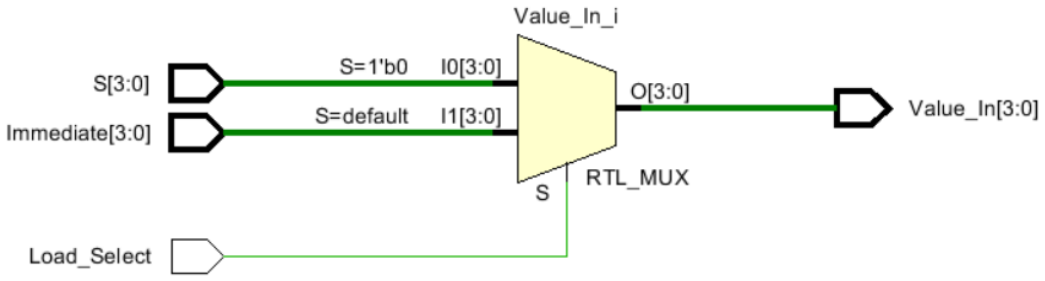
## 2-way 3-bit Multiplexer Timing Diagram



## 2-way 4-bit Multiplexer - Design

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity w2_4_MUX is
      Port ( Load_Select : in STD_LOGIC;
            Immediate : in STD_LOGIC_VECTOR (3 downto 0);
            S : in STD_LOGIC_VECTOR (3 downto 0);
            Value_In : out STD_LOGIC_VECTOR (3 downto 0));
end w2_4_MUX;

architecture Behavioral of w2_4_MUX is
begin
process(Load_Select, Immediate, S)
begin
      if (Load_Select = '0') then
      Value_In <= S;
      else
      Value_In <= Immediate;
      end if;
end process;
end Behavioral;
```

## 2-way 4-bit Multiplexer - Design Schematic

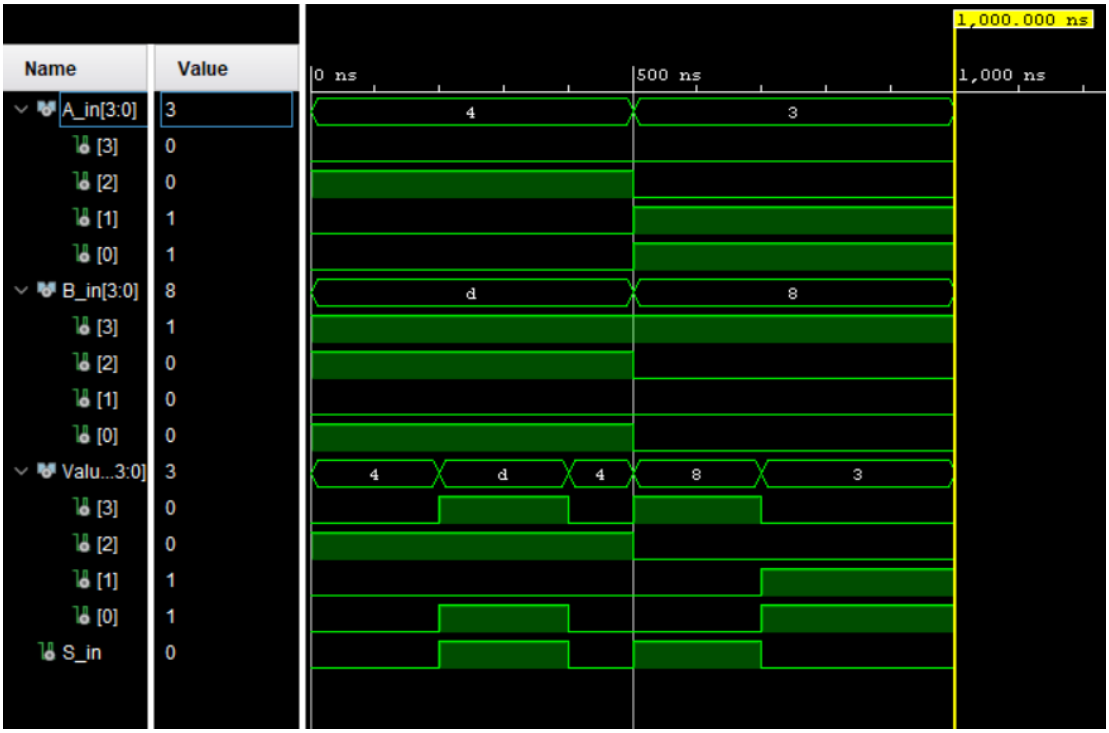


## 2-way 4-bit Multiplexer - Test Bench

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

 entity w2_4_MUX_TB is
 --  Port ( );
 end w2_4_MUX_TB;

 architecture Behavioral of w2_4_MUX_TB is

      component w2_4_MUX
      Port ( Load_Select : in STD_LOGIC;
                 Immediate : in STD_LOGIC_VECTOR (3 downto 0);
                 S : in STD_LOGIC_VECTOR (3 downto 0);
                 Value_In : out STD_LOGIC_VECTOR (3 downto 0));
      end component;

      signal A_in, B_in, Value_Out : STD_LOGIC_VECTOR(3 downto 0);
      signal S_in : STD_LOGIC;

 begin

      uut: w2_4_MUX port map (
      Load_Select => S_in,
      Immediate   => B_in,
      S           => A_in,
      Value_In    => Value_Out
      );

      stim_proc: process
      begin
      --230356 => 11 1000 0011 1101 0100
      -- First case: 0100 and 1101
      A_in <= "0100";
      B_in <= "1101";
      S_in <= '0';
      wait for 200 ns;

      S_in <= '1';
      wait for 200 ns;

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

      -- Second case: 0011 and 1000
      A_in <= "0011";
       B_in <= "1000";
      S_in <= '1';
      wait for 200 ns;
```

```
        S_in <= '0';
        wait;

        end process;
end Behavioral;
```

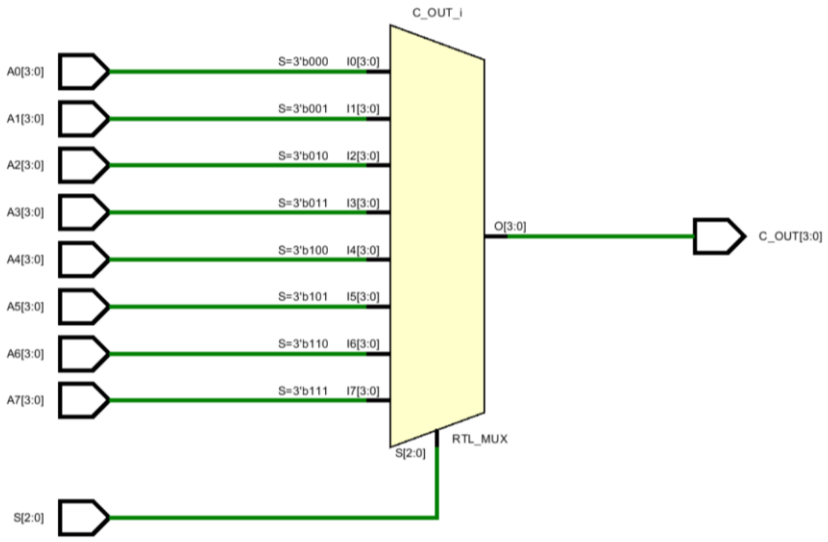
## 2-way 4-bit Multiplexer - Timing Diagram



## 8-way 4-bit Multiplexer - Design

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

entity Mux_8_W_4_B is
    Port (
        A0      : in  STD_LOGIC_VECTOR (3 downto 0);
        A1      : in  STD_LOGIC_VECTOR (3 downto 0);
        A2      : in  STD_LOGIC_VECTOR (3 downto 0);
        A3      : in  STD_LOGIC_VECTOR (3 downto 0);
        A4      : in  STD_LOGIC_VECTOR (3 downto 0);
        A5      : in  STD_LOGIC_VECTOR (3 downto 0);
        A6      : in  STD_LOGIC_VECTOR (3 downto 0);
        A7      : in  STD_LOGIC_VECTOR (3 downto 0);
        C_OUT   : out STD_LOGIC_VECTOR (3 downto 0);
        S       : in  STD_LOGIC_VECTOR (2 downto 0)
    );
end Mux_8_W_4_B;

architecture Behavioral of Mux_8_W_4_B is

begin

process (S, A0, A1, A2, A3, A4, A5, A6, A7)
begin
 case S is
   when "000" => C_OUT <= A0;
   when "001" => C_OUT <= A1;
   when "010" => C_OUT <= A2;
   when "011" => C_OUT <= A3;
   when "100" => C_OUT <= A4;
   when "101" => C_OUT <= A5;
   when "110" => C_OUT <= A6;
   when "111" => C_OUT <= A7;
  when others => C_OUT <= (others => '0');
 end case;
end process;

end Behavioral;
```

## 8-way 4-bit Multiplexer Design Schematic

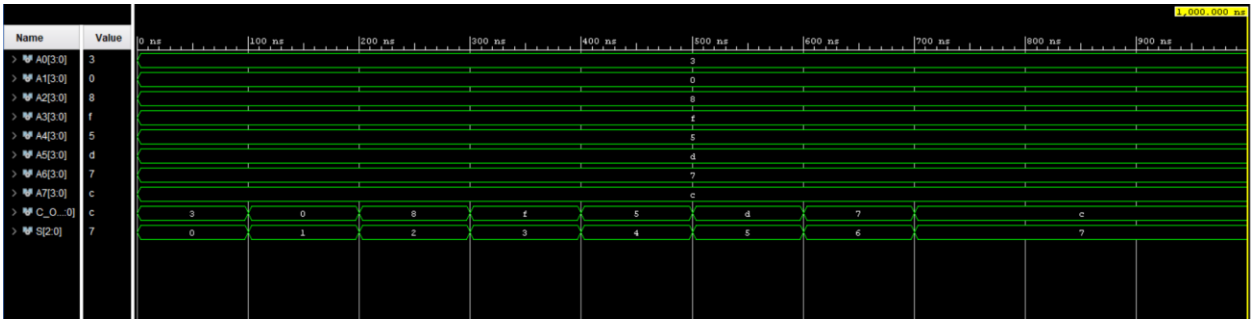



## 8-way 4-bit - Test Bench

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

entity TB_Mux_8_W_4_B is
end TB_Mux_8_W_4_B;

architecture Behavioral of TB_Mux_8_W_4_B is

    component Mux_8_W_4_B
        Port (
            A0    : in  STD_LOGIC_VECTOR (3 downto 0);
            A1    : in  STD_LOGIC_VECTOR (3 downto 0);
            A2    : in  STD_LOGIC_VECTOR (3 downto 0);
            A3    : in  STD_LOGIC_VECTOR (3 downto 0);
            A4    : in  STD_LOGIC_VECTOR (3 downto 0);
            A5    : in  STD_LOGIC_VECTOR (3 downto 0);
            A6    : in  STD_LOGIC_VECTOR (3 downto 0);
            A7    : in  STD_LOGIC_VECTOR (3 downto 0);
            C_OUT : out STD_LOGIC_VECTOR (3 downto 0);
            S     : in  STD_LOGIC_VECTOR (2 downto 0)
        );
    end component;

    -- Signals to connect to the UUT
    signal A0, A1, A2, A3, A4, A5, A6, A7 : STD_LOGIC_VECTOR(3 downto
0);
    signal C_OUT                          : STD_LOGIC_VECTOR(3 downto
0);
```

```
    signal S                                : STD_LOGIC_VECTOR(2 downto
0);

begin

    -- Instantiate the Unit Under Test (UUT)
    UUT: Mux_8_W_4_B
        port map (
            A0    => A0,
            A1    => A1,
            A2    => A2,
            A3    => A3,
            A4    => A4,
            A5    => A5,
            A6    => A6,
            A7    => A7,
            C_OUT => C_OUT,
            S     => S
        );

    -- Stimulus process to drive the inputs
    process
    begin
        -- Set all input values
        A0 <= "0011";  -- 3
        A1 <= "0000";  -- 0
        A2 <= "1000";  -- 8
        A3 <= "1111";  -- F
        A4 <= "0101";  -- 5
        A5 <= "1101";  -- D
        A6 <= "0111";  -- 7
        A7 <= "1100";  -- C

        -- Test all selector inputs
        S <= "000"; wait for 100 ns;  -- Expect C_OUT = A0 = 0011
        S <= "001"; wait for 100 ns;  -- Expect C_OUT = A1 = 0000
        S <= "010"; wait for 100 ns;  -- Expect C_OUT = A2 = 1000
        S <= "011"; wait for 100 ns;  -- Expect C_OUT = A3 = 1111
        S <= "100"; wait for 100 ns;  -- Expect C_OUT = A4 = 0101
        S <= "101"; wait for 100 ns;  -- Expect C_OUT = A5 = 1101
        S <= "110"; wait for 100 ns;  -- Expect C_OUT = A6 = 0111
        S <= "111"; wait for 100 ns;  -- Expect C_OUT = A7 = 1100

        wait;  -- Stop simulation
    end process;

end Behavioral;
```

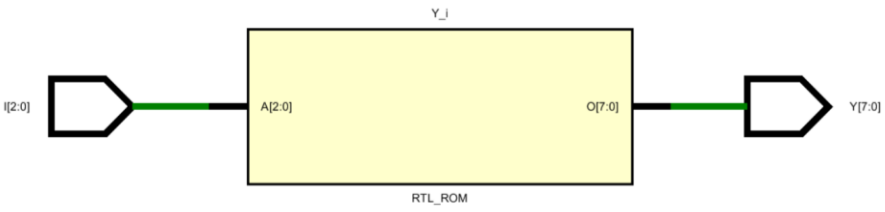
## 8-way 4-bit Timing Diagram



## 3-to-8 Decoder - Design

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

entity Decoder_3_TO_8 is
    Port (
        I  : in  STD_LOGIC_VECTOR(2 downto 0);
        Y  : out STD_LOGIC_VECTOR(7 downto 0)
    );
end Decoder_3_TO_8;

architecture Behavioral of Decoder_3_TO_8 is

begin
process(I)
begin
    case I is
       when "000" => Y <= "00000001";
       when "001" => Y <= "00000010";
       when "010" => Y <= "00000100";
       when "011" => Y <= "00001000";
       when "100" => Y <= "00010000";
       when "101" => Y <= "00100000";
       when "110" => Y <= "01000000";
       when "111" => Y <= "10000000";
       when others => Y <= "00000000";
    end case;
end process;


end Behavioral;
```

## 3-to-8 Decoder Design Schematic

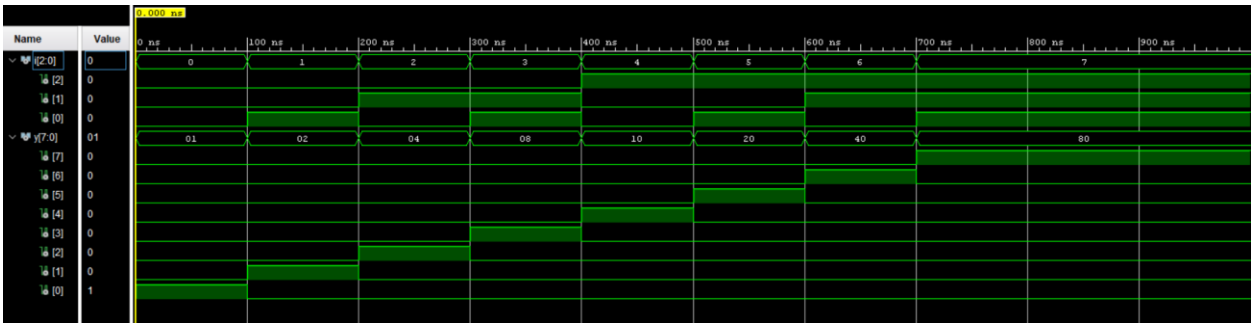


## 3-to-8 Decoder - Test Bench

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

entity TB_Decode_3_To_8 is
end TB_Decode_3_To_8;

architecture Behavioral of TB_Decode_3_To_8 is

    -- Component declaration matches the decoder without EN
    component Decoder_3_TO_8
        Port (
            I  : in  STD_LOGIC_VECTOR(2 downto 0);
            Y  : out STD_LOGIC_VECTOR(7 downto 0)
        );
    end component;

    -- Signals for testing
    signal i : STD_LOGIC_VECTOR(2 downto 0);
    signal y : STD_LOGIC_VECTOR(7 downto 0);

begin

    -- Instantiate the Unit Under Test (UUT)
    UUT: Decoder_3_TO_8
        port map (
            I => i,
            Y => y
        );

    -- Stimulus process
    process
    begin
        i <= "000";
        wait for 100 ns;

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

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

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

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

        i <= "101";
        wait for 100 ns;
```

```
        i <= "110";
        wait for 100 ns;

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

        wait;  -- Stop simulation
    end process;

end Behavioral;
```

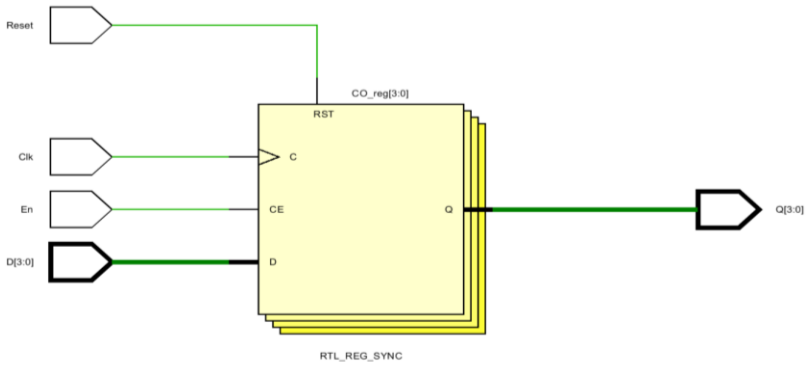
## 3-to-8 Decoder Timing Diagram



## 4-Bit Register - Design

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

entity Reg_4_B 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 Reg_4_B;

architecture Behavioral of Reg_4_B is
signal CO : STD_LOGIC_VECTOR(3 downto 0) := (others=>'0');

begin

process (Clk, En)
begin
    if rising_edge(Clk) then
        if Reset = '1' then
            CO <= "0000";  -- Synchronous reset
        else
          if En = '1' then
              CO <= D;
          end if;
        end if;
    end if;
end process;
  Q <= CO;

end Behavioral;
```

## 4-Bit Register Design Schematic

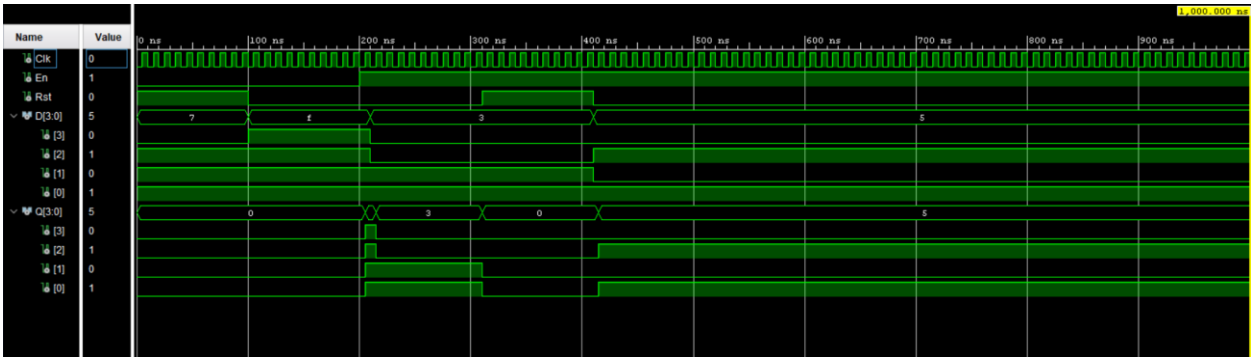


## 4-Bit Register - Test Bench

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

entity TB_Reg_4_B is
    -- No ports for a testbench
end TB_Reg_4_B;

architecture Behavioral of TB_Reg_4_B is

    -- Component declaration for the Unit Under Test (UUT)
    component Reg_4_B 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 component;

    -- Testbench signals
    signal Clk, En, Rst : STD_LOGIC;
    signal D            : STD_LOGIC_VECTOR(3 downto 0);
    signal Q            : STD_LOGIC_VECTOR(3 downto 0);

begin

    -- Instantiate the Unit Under Test (UUT)
    UUT: Reg_4_B
        port map (
            Clk   => Clk,
            Reset => Rst,
            En    => En,
            D     => D,
            Q     => Q
        );

    -- Clock generation process (10ns period)
    clock_process: process
    begin
        while true loop
            Clk <= '0';
            wait for 5 ns;
            Clk <= '1';
            wait for 5 ns;
        end loop;
    end process;

    -- Stimulus process to test behavior
```

```
    reg_process: process
    begin
        -- Apply asynchronous reset
        Rst <= '1';
        En  <= '0';
        D   <= "0111";
        wait for 100 ns;

        -- Release reset, don't write yet
        Rst <= '0';
        D   <= "1111";
        wait for 100 ns;

        -- Enable write and allow storing of "1111"
        En  <= '1';
        wait for 10 ns;

        -- Change data to "0011", should be stored on next clock
        D   <= "0011";
        wait for 100 ns;

        -- Reset the register again
        Rst <= '1';
        wait for 100 ns;

        -- Release reset and load new data
        Rst <= '0';
        D   <= "0101";
        wait for 100 ns;

        -- End of simulation
        wait;
    end process;

end Behavioral;
```

## 4-Bit Register Timing Diagram



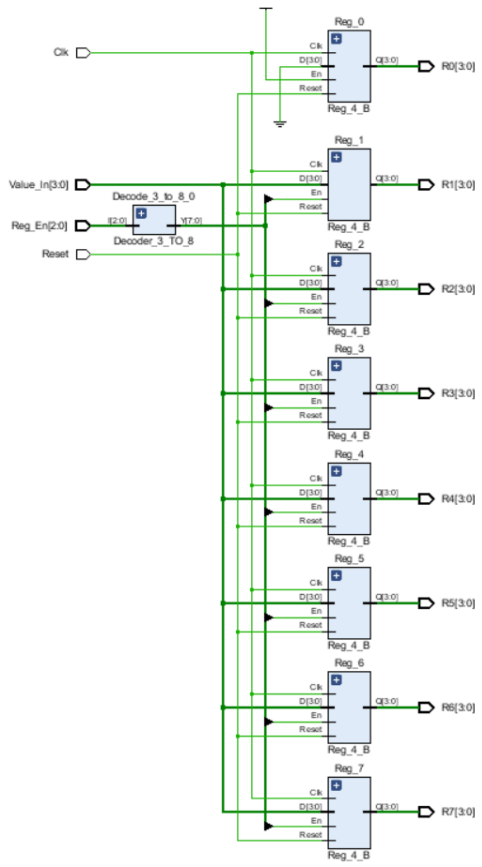
## Register Bank - Design

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

entity Register_Bank is
    Port (
        Value_In : in STD_LOGIC_VECTOR(3 downto 0);
        Clk      : in STD_LOGIC;
        Reg_En   : in STD_LOGIC_VECTOR(2 downto 0);
        Reset    : in STD_LOGIC;
        R0       : out STD_LOGIC_VECTOR(3 downto 0);
        R1       : out STD_LOGIC_VECTOR(3 downto 0);
        R2       : out STD_LOGIC_VECTOR(3 downto 0);
        R3       : out STD_LOGIC_VECTOR(3 downto 0);
        R4       : out STD_LOGIC_VECTOR(3 downto 0);
        R5       : out STD_LOGIC_VECTOR(3 downto 0);
        R6       : out STD_LOGIC_VECTOR(3 downto 0);
        R7       : out STD_LOGIC_VECTOR(3 downto 0)
    );
end Register_Bank;

architecture Behavioral of Register_Bank is

    component Decoder_3_to_8
        Port (
            I  : in STD_LOGIC_VECTOR;
            Y  : out STD_LOGIC_VECTOR
        );
    end component;

    component Reg_4_B
        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) := "0000"
        );
    end component;

    signal reg_en_out : STD_LOGIC_VECTOR(7 downto 0);

begin

    -- Decoder to enable only one register based on Reg_En
    Decode_3_to_8_0 : Decoder_3_to_8
        Port map (
            I  => Reg_En,
            Y  => reg_en_out
        );
```

```
-- R0 is hardwired to 0000 (read-only)
Reg_0 : Reg_4_B
    Port map (
        D     => "0000",
        En    => '1',
        Clk   => Clk,
        Reset => Reset,
        Q     => R0
    );

Reg_1 : Reg_4_B
    Port map (
        D     => Value_In,
        En    => reg_en_out(1),
        Clk   => Clk,
        Reset => Reset,
        Q     => R1
    );

Reg_2 : Reg_4_B
    Port map (
        D     => Value_In,
        En    => reg_en_out(2),
        Clk   => Clk,
        Reset => Reset,
        Q     => R2
    );

Reg_3 : Reg_4_B
    Port map (
        D     => Value_In,
        En    => reg_en_out(3),
        Clk   => Clk,
        Reset => Reset,
        Q     => R3
    );

Reg_4 : Reg_4_B
    Port map (
        D     => Value_In,
        En    => reg_en_out(4),
        Clk   => Clk,
        Reset => Reset,
        Q     => R4
    );

Reg_5 : Reg_4_B
    Port map (
        D     => Value_In,
        En    => reg_en_out(5),
        Clk   => Clk,
```

```
            Reset => Reset,
            Q     => R5
        );

    Reg_6 : Reg_4_B
        Port map (
            D     => Value_In,
            En    => reg_en_out(6),
            Clk   => Clk,
            Reset => Reset,
            Q     => R6
        );

    Reg_7 : Reg_4_B
        Port map (
            D     => Value_In,
            En    => reg_en_out(7),
            Clk   => Clk,
            Reset => Reset,
            Q     => R7
        );

end Behavioral;
```

## Register Bank Design Schematic



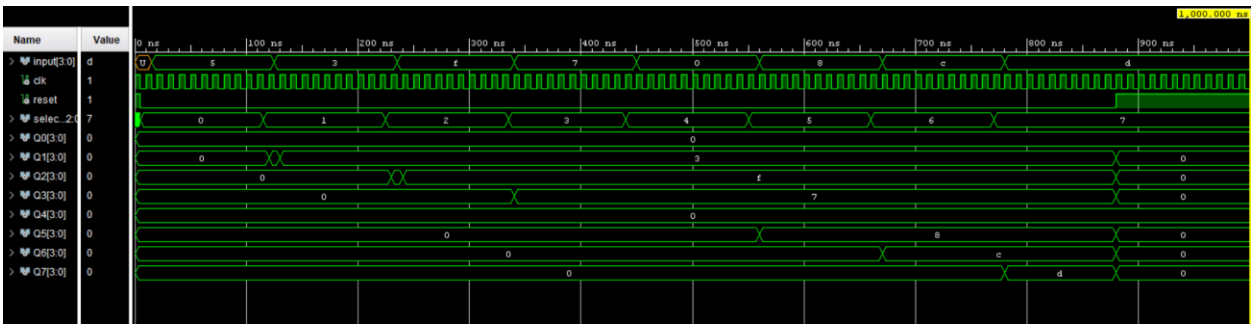
## Register Bank - Test Bench

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

entity TB_Register_Bank is
-- No ports for testbench
end TB_Register_Bank;

architecture Behavioral of TB_Register_Bank is

    component Register_Bank
        Port (
            Value_In : in  STD_LOGIC_VECTOR(3 downto 0);
            Clk      : in  STD_LOGIC;
            Reg_En   : in  STD_LOGIC_VECTOR(2 downto 0);
            Reset    : in  STD_LOGIC;
            R0       : out STD_LOGIC_VECTOR(3 downto 0);
            R1       : out STD_LOGIC_VECTOR(3 downto 0);
            R2       : out STD_LOGIC_VECTOR(3 downto 0);
            R3       : out STD_LOGIC_VECTOR(3 downto 0);
            R4       : out STD_LOGIC_VECTOR(3 downto 0);
            R5       : out STD_LOGIC_VECTOR(3 downto 0);
            R6       : out STD_LOGIC_VECTOR(3 downto 0);
            R7       : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    signal input    : STD_LOGIC_VECTOR(3 downto 0);
    signal clk      : STD_LOGIC := '0';
    signal reset    : STD_LOGIC := '0';
    signal selecter : STD_LOGIC_VECTOR(2 downto 0);
    signal Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7 : STD_LOGIC_VECTOR(3 downto
0);

begin

    -- Instantiate the Unit Under Test (UUT)
    UUT: Register_Bank
        port map (
            Value_In => input,
            Clk      => clk,
            Reg_En   => selecter,
            Reset    => reset,
            R0       => Q0,
            R1       => Q1,
            R2       => Q2,
            R3       => Q3,
            R4       => Q4,
            R5       => Q5,
            R6       => Q6,
```

```
        R7       => Q7
    );

-- Clock generation process (10 ns period)
process
begin
    Clk <= NOT Clk;
    wait for 5 ns;
end process;

-- Stimulus process
process
begin
    reset <= '1';
    wait for 5 ns;
    reset <= '0';

    selecter <= "000";
    wait for 10 ns;
    input <= "0101"; -- 5
    wait for 100 ns;

    selecter <= "001";
    wait for 10 ns;
    input <= "0011"; -- 3
    wait for 100 ns;

    selecter <= "010";
    wait for 10 ns;
    input <= "1111"; -- F
    wait for 100 ns;

    selecter <= "011";
    wait for 5 ns;
    input <= "0111"; -- 7
    wait for 100 ns;

    selecter <= "100";
    wait for 10 ns;
    input <= "0000"; -- 0
    wait for 100 ns;

    selecter <= "101";
    wait for 10 ns;
    input <= "1000"; -- 8
    wait for 100 ns;

    selecter <= "110";
    wait for 10 ns;
    input <= "1100"; -- C
    wait for 100 ns;
```

```
        selecter <= "111";
        wait for 10 ns;
        input <= "1101"; -- D
        wait for 100 ns;

        reset <= '1';
        wait;
    end process;

end Behavioral;
```

## Register Bank Timing Diagram

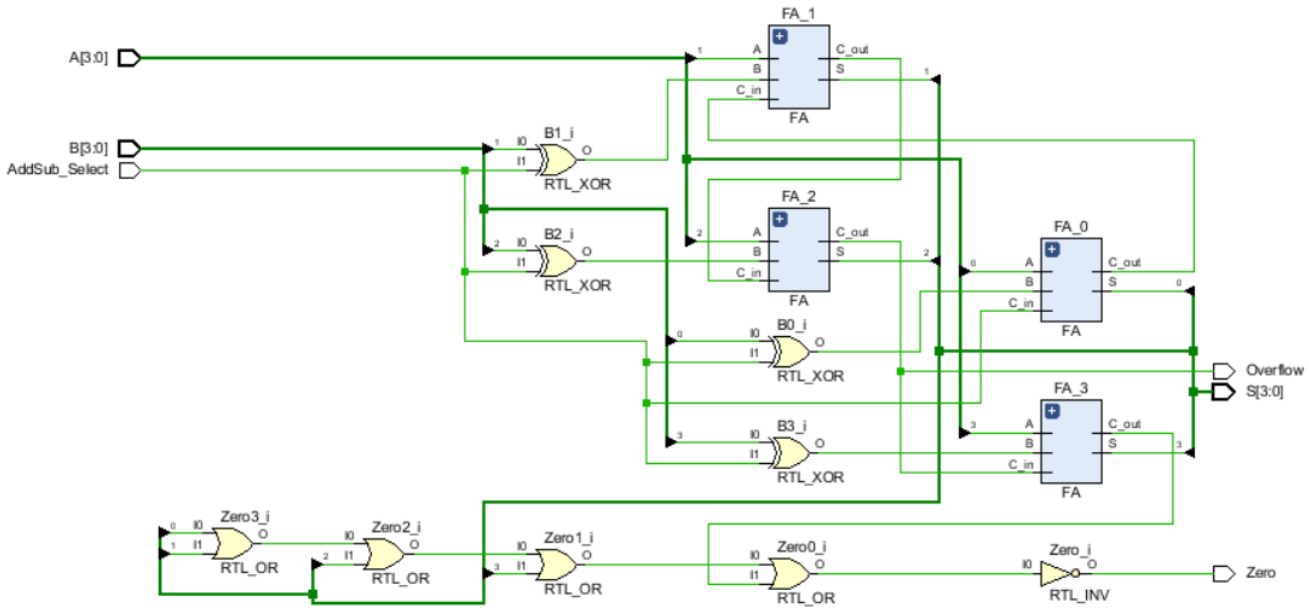


## 4_bit_add_sub_unit - Design

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

entity ADD_SUB is
      Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
             B : in STD_LOGIC_VECTOR (3 downto 0);
             AddSub_Select : in STD_LOGIC;
             Overflow : out STD_LOGIC;
             Zero : out STD_LOGIC;
             S : out STD_LOGIC_VECTOR (3 downto 0));
end ADD_SUB;

architecture Behavioral of ADD_SUB 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 , B0,
B1, B2, B3 : std_logic;
SIGNAL S_out : STD_LOGIC_VECTOR(3 downto 0);
begin

B0 <= B(0) XOR AddSub_Select;
B1 <= B(1) XOR AddSub_Select;
B2 <= B(2) XOR AddSub_Select;
B3 <= B(3) XOR AddSub_Select;

FA_0 : FA
      port map (
      A => A(0),
      B => B0,
      C_in => AddSub_Select ,
      S => S_out(0),
      C_Out => FA0_C);

FA_1 : FA
      port map (
      A => A(1),
      B => B1,
      C_in => FA0_C,
      S => S_out(1),
      C_Out => FA1_C);
FA_2 : FA
      port map (
```

```
      A => A(2),
      B => B2,
      C_in => FA1_C,
      S => S_out(2),
      C_Out => FA2_C);

FA_3 : FA
      port map (
      A => A(3),
      B => B3,
      C_in => FA2_C,
      S => S_out(3),
      C_Out => FA3_C);
 Overflow <= FA2_C;
 S <= S_out;
 Zero <= not (S_out(0) or S_out(1) or S_out(2) or S_out(3) or FA3_C);

end Behavioral;
```

## 4_bit_add_sub_unit Design Schematic



## 4_bit_add_sub_unit - TB

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

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

 architecture Behavioral of ADD_SUB_TB is

      component ADD_SUB
      Port (
            A : in STD_LOGIC_VECTOR (3 downto 0);
            B : in STD_LOGIC_VECTOR (3 downto 0);
            AddSub_Select : in STD_LOGIC;
            Overflow : out STD_LOGIC;
            Zero : out STD_LOGIC;
            S : out STD_LOGIC_VECTOR (3 downto 0)
      );
      end component;

      signal A_tb : STD_LOGIC_VECTOR (3 downto 0);
      signal B_tb : STD_LOGIC_VECTOR (3 downto 0);
      signal AddSub_Select_tb : STD_LOGIC;
      signal Overflow_tb : STD_LOGIC;
      signal Zero_tb : STD_LOGIC;
      signal S_tb : STD_LOGIC_VECTOR (3 downto 0);

 begin

      uut: ADD_SUB port map (
      A => A_tb,
      B => B_tb,
      AddSub_Select => AddSub_Select_tb,
      Overflow => Overflow_tb,
      Zero => Zero_tb,
      S => S_tb
      );

      stim_proc: process
      begin
      --230253 => 11 1000 0011 0110 1101
      -- Test case 1: 1101 + 0110 (13 + 6 = 19)
      A_tb <= "1101";
      B_tb <= "0110";
      AddSub_Select_tb <= '0';  -- 0 = Addition
      wait for 10 ns;

      --Test case 2: subtract 15 from 8
      AddSub_Select_tb <= '1';
```

```
        A_tb <= "1000";
        B_tb <= "1111";
        AddSub_Select_tb <= '1';
        wait for 10 ns;


        -- Test case 3: 0011 + 1000 (3 + 8 = 11)
        A_tb <= "0011";
        B_tb <= "1000";
        AddSub_Select_tb <= '0';  -- 0 = Addition
          wait for 10 ns;


        --Test case 4: subtract 7 from 15
        AddSub_Select_tb <= '1';
        A_tb <= "1111";
        B_tb <= "0111";

        wait;
        end process;

end Behavioral;
```

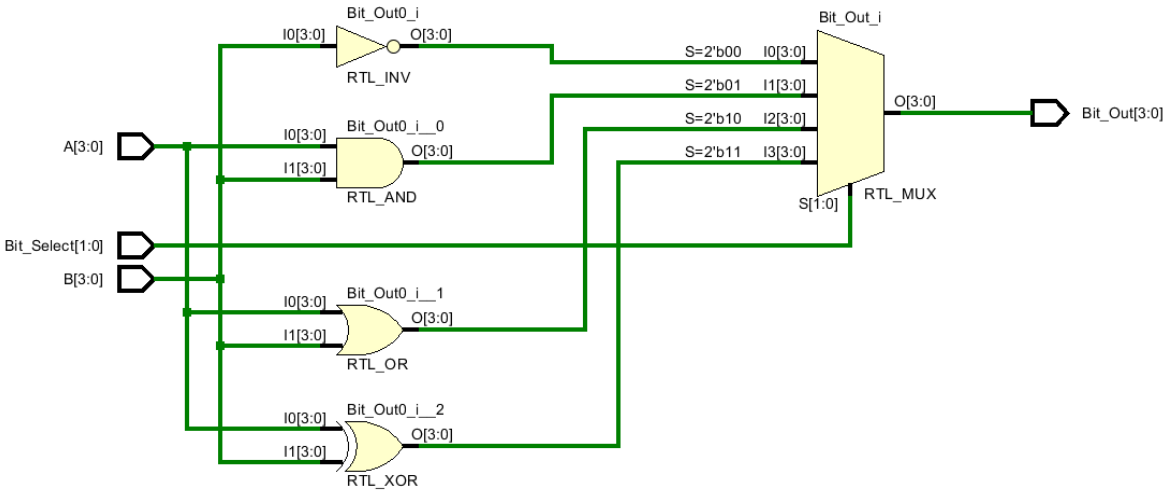
## 4_bit_add_sub_unit Timing Diagram


Name
Value
A_tb[3:0]
f
[3]
1
[2]
1
[1]
1
[0]
1
B_tb[3:0]
7
[3]
0
[2]
1
[1]
1
[0]
1
AddS...t_tb
1
Overf...w_t
1
Zero_tb
0
S_tb[3:0]
8
[3]
1
[2]
0
[1]
0
[0]
0
0 ns
10 ns
20 ns
30 ns
40 ns
d
8
3
f
6
f
8
7
3
9
b
8

## Bitwise Operator - Design

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

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

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

entity Bitwise_operator is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Bit_Select : in STD_LOGIC_VECTOR (1 downto 0);
           Bit_Out : out STD_LOGIC_VECTOR (3 downto 0));
end Bitwise_operator;


architecture Behavioral of Bitwise_operator is

begin
process(A,B,Bit_Select)
begin
    case Bit_Select  is
       when "00" =>   --compliment
             Bit_Out <= Not B ;

       when "01" =>  --and
            Bit_Out <= A and B;

       when "10" =>  --or
            Bit_Out <= A or B;

       when "11" =>  --xor
            Bit_Out <= A xor B;

       when others =>
            Bit_Out <= (others => '0');  -- or assign to A, or keep
last value if desired


    end case;
end process;


end Behavioral;
```

## Bitwise Operator Design Schematic

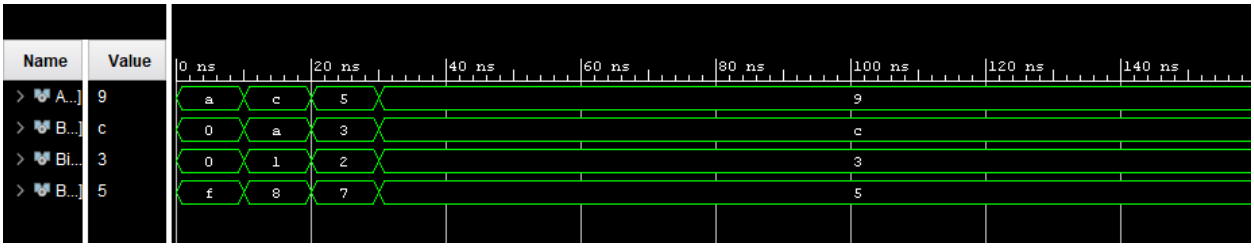



## Bitwise Operator- Test Bench

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

entity tb_Bitwise_operator is
end tb_Bitwise_operator;

architecture Behavioral of tb_Bitwise_operator is

    -- Component Declaration
    component Bitwise_operator
        Port (
            A          : in  STD_LOGIC_VECTOR(3 downto 0);
            B          : in  STD_LOGIC_VECTOR(3 downto 0);
            Bit_Select : in  STD_LOGIC_VECTOR(1 downto 0);
            Bit_Out    : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    -- Signals
    signal A          : STD_LOGIC_VECTOR(3 downto 0) := (others =>
'0');
    signal B          : STD_LOGIC_VECTOR(3 downto 0) := (others =>
'0');
    signal Bit_Select : STD_LOGIC_VECTOR(1 downto 0) := "00";
    signal Bit_Out    : STD_LOGIC_VECTOR(3 downto 0);

begin

    -- Instantiate the Unit Under Test (UUT)
```

```
    uut: Bitwise_operator
        port map (
            A => A,
            B => B,
            Bit_Select => Bit_Select,
            Bit_Out => Bit_Out
        );

    -- Test Process
    stim_proc: process
    begin
        -- Test NOT A
        A <= "1010";
        B <= "0000";  -- unused
        Bit_Select <= "00";
        wait for 10 ns;

        -- Test A AND B
        A <= "1100";
        B <= "1010";
        Bit_Select <= "01";
        wait for 10 ns;

        -- Test A OR B
        A <= "0101";
        B <= "0011";
        Bit_Select <= "10";
        wait for 10 ns;

        -- Test A XOR B
        A <= "1001";
        B <= "1100";
        Bit_Select <= "11";
        wait for 10 ns;

        wait;
    end process;

end Behavioral;
```

## Bitwise Operator Timing Diagram

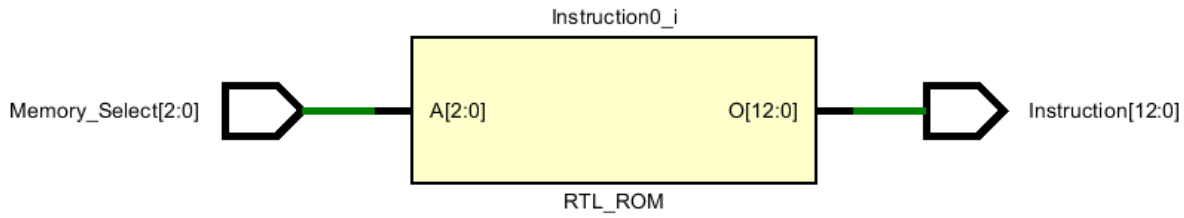


## Program Rom - Design

```
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 ( Memory_Select : in STD_LOGIC_VECTOR (2 downto 0);
           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 P_ROM : rom_type := (
--            "0101110000011",     -- MOVI R7, 3
--            "0100010000001",     -- MOVI R1, 1
--            "0010010000000",     -- NEG  R1
--            "0100100000011",     -- MOVI R2, 3
--            "0000100010000",     -- ADD  R2, R1
--            "0001110100000",     -- ADD  R7, R2
--            "0110100000110",     -- JZR  R2, 6
--            "0110000000100"      -- JZR  R0, 4

            "0101110000011",     -- 0 MOVI R7, 3
            "0100010000001",     -- 1 MOVI R1, 1
            "1111110010000",     -- 2 bit xor  R1 R7
            "0110000000011",     -- 3 JZR  R2, 6
            "0000100010000",     -- 4 ADD  R2, R1
            "0001110100000",     -- 5 ADD  R7, R2
            "0110100000110",     -- 6 JZR  R2, 6
            "0110000000100"      -- 7 JZR  R0, 4
    );


begin
        process(Memory_Select)
        begin
            if to_integer(unsigned(Memory_Select)) < P_ROM'length then
                Instruction <=
P_ROM(to_integer(unsigned(Memory_Select)));
            else
                Instruction <= (others => '0');
```

```
            end if;
        end process;

    end Behavioral;
```

## Program Rom Design Schematic

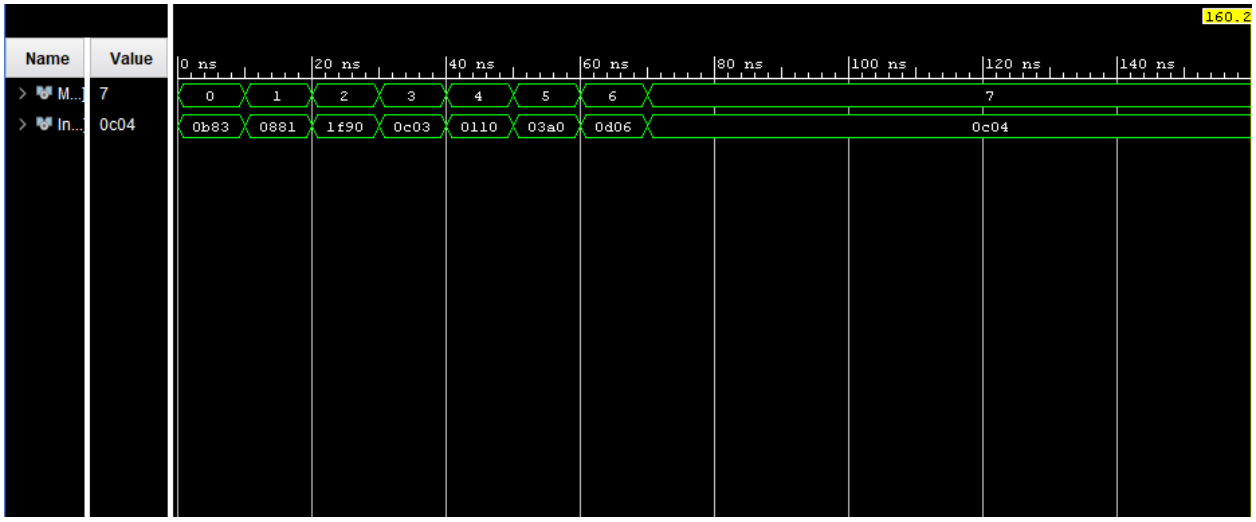



## Program Rom - Test Bench

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

entity tb_Program_ROM is
end tb_Program_ROM;

architecture Behavioral of tb_Program_ROM is

    -- Component Declaration
    component Program_ROM
        Port (
            Memory_Select : in STD_LOGIC_VECTOR (2 downto 0);
            Instruction   : out STD_LOGIC_VECTOR (12 downto 0)
        );
    end component;

    -- Testbench Signals
    signal Memory_Select : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal Instruction   : STD_LOGIC_VECTOR(12 downto 0);

begin

    -- Instantiate the Program ROM
    uut: Program_ROM
        port map (
            Memory_Select => Memory_Select,
            Instruction   => Instruction
        );
```

```
    -- Stimulus process
    stim_proc: process
    begin
        -- Go through all ROM locations from 0 to 7
        for i in 0 to 7 loop
            Memory_Select <= std_logic_vector(to_unsigned(i, 3));
            wait for 10 ns;
        end loop;

        wait;
    end process;

end Behavioral;
```

## Program Rom Timing Diagram

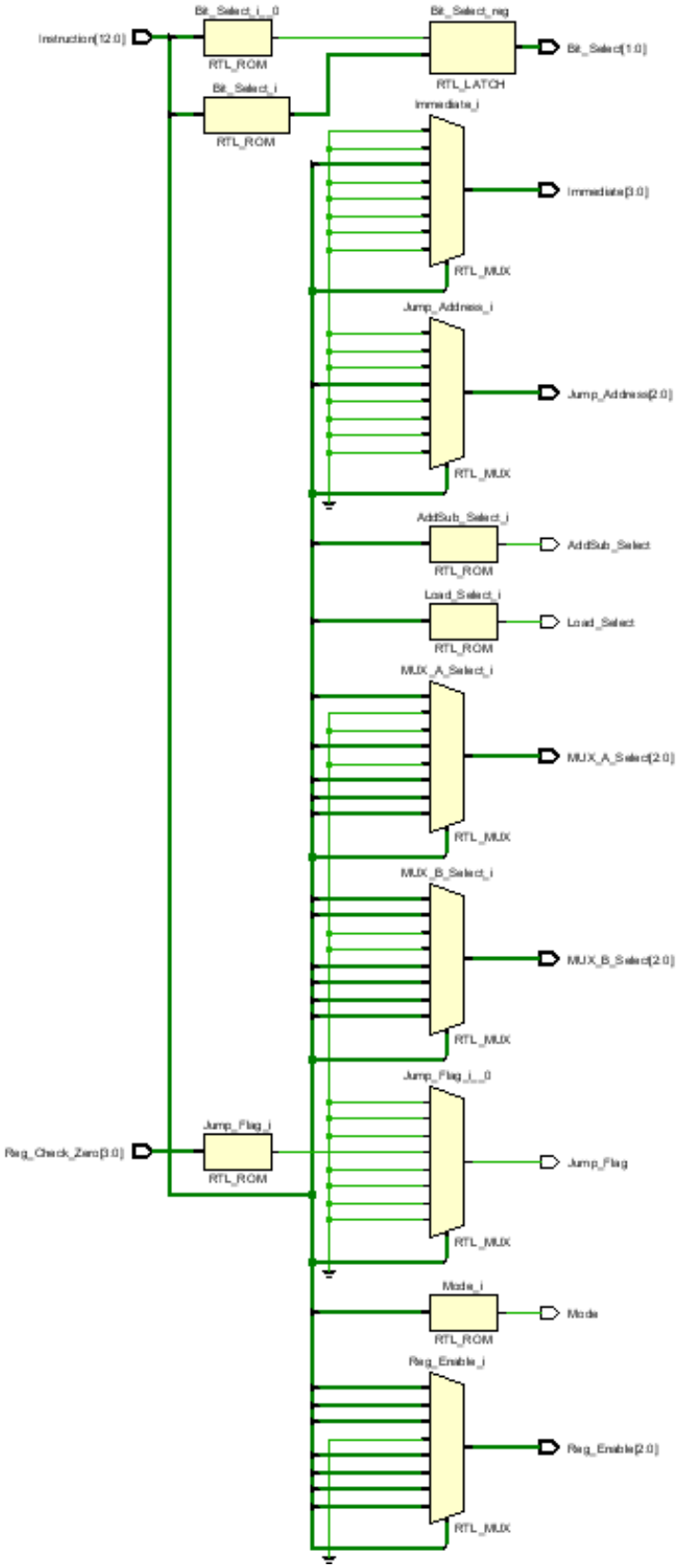


## Instruction Decoder- Design

```
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 Instruction_Decoder is
    Port (
        Instruction     : in  STD_LOGIC_VECTOR(12 downto 0);  -- 13-
bit instruction
        Reg_Check_Zero  : in STD_LOGIC_VECTOR(3 downto 0);
        Reg_Enable      : out STD_LOGIC_VECTOR(2 downto 0);
        Load_Select     : out STD_LOGIC;
        Immediate       : out STD_LOGIC_VECTOR(3 downto 0);
        MUX_A_Select    : out STD_LOGIC_VECTOR(2 downto 0);
        MUX_B_Select    : out STD_LOGIC_VECTOR(2 downto 0);
        AddSub_Select   : out STD_LOGIC;
        Jump_Flag       : out STD_LOGIC;
        Jump_Address    : out STD_LOGIC_VECTOR(2 downto 0);
        Bit_Select    : out STD_LOGIC_VECTOR(1 downto 0);
        Mode       : out STD_LOGIC
    );
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is
   signal opcode : STD_LOGIC_VECTOR(2 downto 0);
begin
 opcode <= Instruction(12 downto 10);
 process(opcode, Instruction, Reg_Check_Zero)
 begin
     -- Default outputs
     Reg_Enable       <= (others => '0');
     MUX_A_Select       <= (others => '0');
     MUX_B_Select       <= (others => '0');
     Load_Select     <= '0';
     Immediate      <= (others => '0');
     AddSub_Select          <= '0';
     Mode         <= '0';
     Jump_Flag    <= '0';
     Jump_Address <= (others => '0');
     case opcode is
```

```
        when "000" =>  -- ADD
            Reg_Enable <= Instruction(9 downto 7);
            MUX_A_Select <= Instruction(9 downto 7);
            MUX_B_Select <= Instruction(6 downto 4);
        when "001" =>  -- NEG
            Reg_Enable <= Instruction(9 downto 7);
            MUX_B_Select <= Instruction(9 downto 7);
            AddSub_Select   <= '1';
        when "010" =>  -- MOVI
            Reg_Enable   <= Instruction(9 downto 7);
            Load_Select <= '1';
            Immediate  <= Instruction(3 downto 0);
        when "011" =>  -- JZR
            MUX_A_Select       <= Instruction(9 downto 7);
            Jump_Address <= Instruction(2 downto 0);
            if Reg_Check_Zero = "0000" then
                Jump_Flag <= '1';
            end if;




        when "100" =>  -- Bit complement
             Reg_Enable   <= Instruction(9 downto 7);
             MUX_B_Select <= Instruction(9 downto 7);
             Bit_Select <="00";
             Mode <='1';
        when "101" =>  -- Bit And
             Reg_Enable   <= Instruction(9 downto 7);
             MUX_A_Select <= Instruction(9 downto 7);
             MUX_B_Select <= Instruction(6 downto 4);
             Bit_Select <="01";
             Mode <='1';
        when "110" =>  -- Bit Or
             Reg_Enable   <= Instruction(9 downto 7);
             MUX_A_Select <= Instruction(9 downto 7);
             MUX_B_Select <= Instruction(6 downto 4);
             Bit_Select <="10";
             Mode <='1';
        when "111" =>  -- Bit Xor
             Reg_Enable   <= Instruction(9 downto 7);
             MUX_A_Select <= Instruction(9 downto 7);
             MUX_B_Select <= Instruction(6 downto 4);
             Bit_Select <="11";
             Mode <='1';

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

## Instruction Decoder Design Schematic



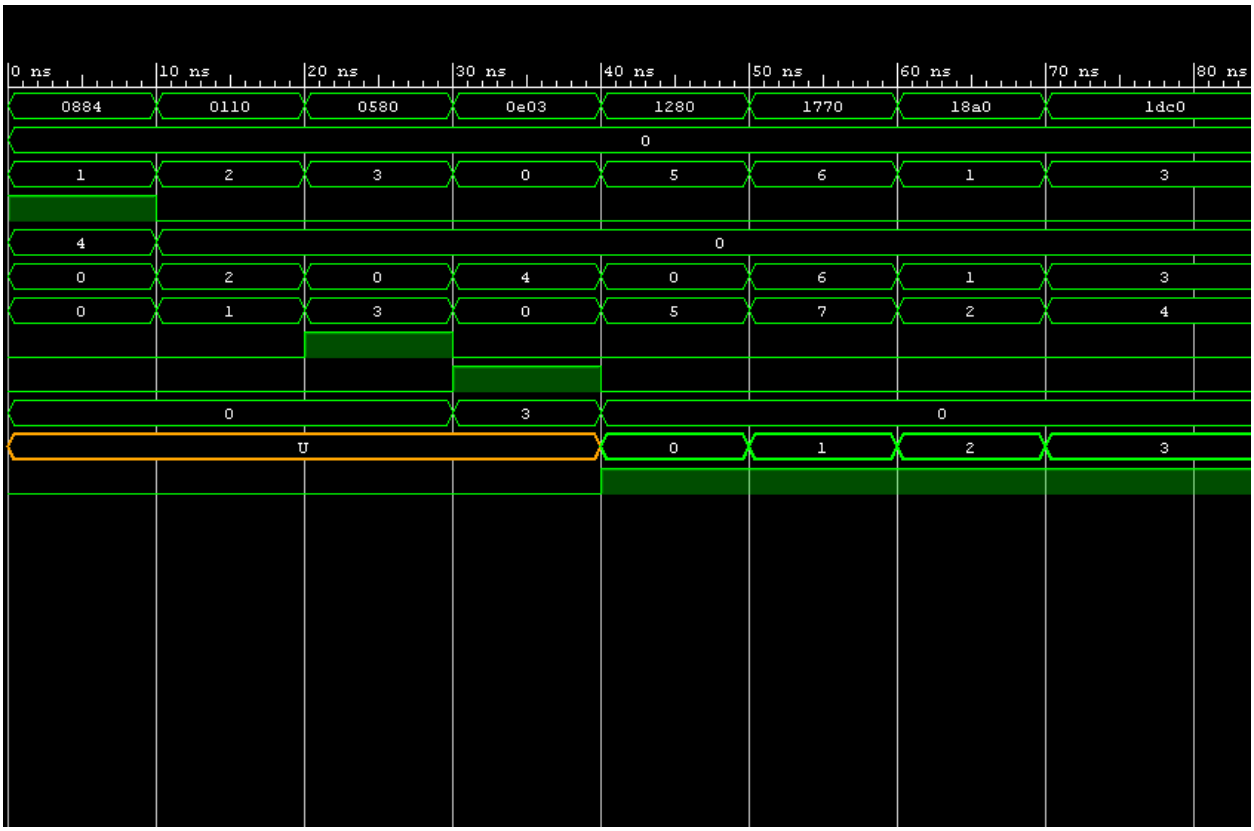
## Instruction Decoder - Test Bench

```
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 declaration
    component Instruction_Decoder
        Port (
            Instruction     : in  STD_LOGIC_VECTOR(12 downto 0);
            Reg_Check_Zero  : in  STD_LOGIC_VECTOR(3 downto 0);
            Reg_Enable      : out STD_LOGIC_VECTOR(2 downto 0);
            Load_Select     : out STD_LOGIC;
            Immediate       : out STD_LOGIC_VECTOR(3 downto 0);
            MUX_A_Select    : out STD_LOGIC_VECTOR(2 downto 0);
            MUX_B_Select    : out STD_LOGIC_VECTOR(2 downto 0);
            AddSub_Select   : out STD_LOGIC;
            Jump_Flag       : out STD_LOGIC;
            Jump_Address    : out STD_LOGIC_VECTOR(2 downto 0);
            Bit_Select      : out STD_LOGIC_VECTOR(1 downto 0);
            Mode            : out STD_LOGIC
        );
    end component;

    -- Signals for inputs
    signal Instruction     : STD_LOGIC_VECTOR(12 downto 0) := (others
=> '0');
    signal Reg_Check_Zero  : STD_LOGIC_VECTOR(3 downto 0) := "0000";

    -- Signals for outputs
    signal Reg_Enable      : STD_LOGIC_VECTOR(2 downto 0);
    signal Load_Select     : STD_LOGIC;
    signal Immediate       : STD_LOGIC_VECTOR(3 downto 0);
    signal MUX_A_Select    : STD_LOGIC_VECTOR(2 downto 0);
    signal MUX_B_Select    : STD_LOGIC_VECTOR(2 downto 0);
    signal AddSub_Select   : STD_LOGIC;
    signal Jump_Flag       : STD_LOGIC;
    signal Jump_Address    : STD_LOGIC_VECTOR(2 downto 0);
    signal Bit_Select      : STD_LOGIC_VECTOR(1 downto 0);
    signal Mode            : STD_LOGIC;

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: Instruction_Decoder
        Port Map (
            Instruction     => Instruction,
```

```
        Reg_Check_Zero  => Reg_Check_Zero,
        Reg_Enable      => Reg_Enable,
        Load_Select     => Load_Select,
        Immediate       => Immediate,
        MUX_A_Select    => MUX_A_Select,
        MUX_B_Select    => MUX_B_Select,
        AddSub_Select   => AddSub_Select,
        Jump_Flag       => Jump_Flag,
        Jump_Address    => Jump_Address,
        Bit_Select      => Bit_Select,
        Mode            => Mode
    );

-- Stimulus process
stim_proc: process
begin
    -- MOVI R1, 4 (opcode=010, Reg=001, imm=0100)
    Instruction <= "0100010000100";
    wait for 10 ns;

    -- ADD R2, R1 (opcode=000, Reg=010, Rb=001)
    Instruction <= "0000100010000";
    wait for 10 ns;

    -- NEG R3 (opcode=001, Reg=011)
    Instruction <= "0010110000000";
    wait for 10 ns;

    -- JZR R4, address 3 (opcode=011, Reg=100, addr=011)
    Instruction <= "0111000000011";
    Reg_Check_Zero <= "0000";  -- should trigger jump
    wait for 10 ns;

    -- Bitwise NOT R5 (opcode=100, Reg=101)
    Instruction <= "1001010000000";
    wait for 10 ns;

    -- Bitwise AND R6, R7 (opcode=101, Ra=110, Rb=111)
    Instruction <= "1011101110000";
    wait for 10 ns;

    -- Bitwise OR R1, R2 (opcode=110, Ra=001, Rb=010)
    Instruction <= "1100010100000";
    wait for 10 ns;

    -- Bitwise XOR R3, R4 (opcode=111, Ra=011, Rb=100)
    Instruction <= "1110111000000";
    wait for 10 ns;

    wait;
end process;
```

```
end Behavioral;
```

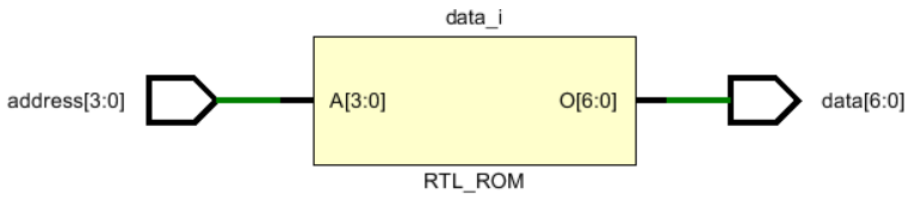
## Instruction Decoder Timing Diagram



## 7- Segment Design

```
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;
```

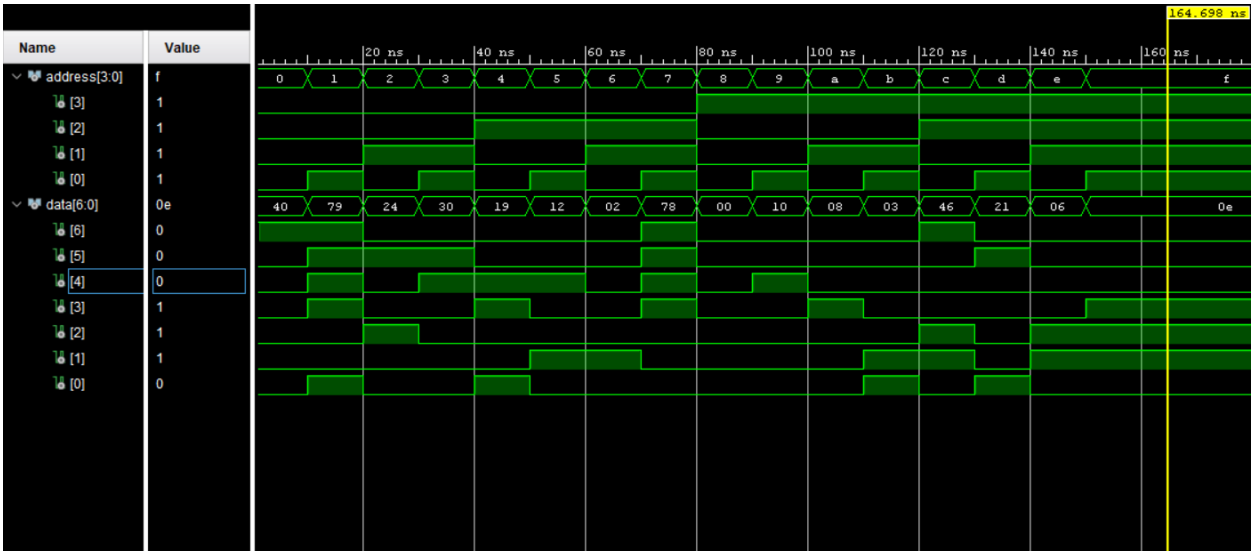
## 7 - Segment Design Schematic



## 7 - Segment Test Bench

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

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

 architecture Behavioral of Seven_Segment_TB 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 : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
      signal data      : STD_LOGIC_VECTOR(6 downto 0);

 begin

      uut: LUT_16_7
      Port map (
            address => address,
            data => data
      );

      stim_proc: process
      begin
      for i in 0 to 15 loop
            address <= std_logic_vector(to_unsigned(i, 4));
            wait for 10 ns;
      end loop;

      wait;
      end process;

 end Behavioral;
```

## 7 - Segment Timing Diagram



## Nano processor - Design

```
----------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Nano_Processor is
Port ( Reset : in STD_LOGIC; --(Done)
       Clk_in : in STD_LOGIC;
         Zero : out STD_LOGIC;
     Overflow : out STD_LOGIC;
         RLED : out STD_LOGIC_VECTOR (3 downto 0);
    Seven_Seg_Out : out STD_LOGIC_VECTOR (6 downto 0);
        Anode : out STD_LOGIC_VECTOR (3 downto 0);
       Clk_Step : in STD_LOGIC;
       Clk_Mode : in STD_LOGIC );
end Nano_Processor;
architecture Behavioral of Nano_Processor is


------------------------ Program ROM ------------------------ (done)


COMPONENT Program_ROM
PORT(
    Memory_Select : in STD_LOGIC_VECTOR (2 downto 0);
    Instruction : out STD_LOGIC_VECTOR (12 downto 0));
end COMPONENT;


-------------------- Instruction Decoder --------------------(done)

COMPONENT Instruction_Decoder
PORT(
        Instruction     : in  STD_LOGIC_VECTOR(12 downto 0);  -- 12-
bit instruction
        Reg_Check_Zero  : in STD_LOGIC_VECTOR(3 downto 0);
        Reg_Enable      : out STD_LOGIC_VECTOR(2 downto 0);
        Load_Select     : out STD_LOGIC;
        Immediate       : out STD_LOGIC_VECTOR(3 downto 0);
        MUX_A_Select    : out STD_LOGIC_VECTOR(2 downto 0);
        MUX_B_Select    : out STD_LOGIC_VECTOR(2 downto 0);
        AddSub_Select   : out STD_LOGIC;
        Jump_Flag       : out STD_LOGIC;
        Jump_Address    : out STD_LOGIC_VECTOR(2 downto 0);
        Bit_Select    : out STD_LOGIC_VECTOR(1 downto 0);
        Mode       : out STD_LOGIC);
end COMPONENT;
```

```
---------------------- Program Counter ---------------------- (Done)
COMPONENT Program_Counter_3bit
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;


----------------------- 3-bit Adder ----------------------- (Done)
COMPONENT CLA_3bit
PORT(       A : in STD_LOGIC_VECTOR (2 downto 0);
        B     : in  STD_LOGIC_VECTOR(2 downto 0);
        Cin   : in  STD_LOGIC;
        Sum   : out STD_LOGIC_VECTOR(2 downto 0);
        Cout  : out STD_LOGIC );
end COMPONENT;


---------------------- Register Bank ---------------------- (done)
COMPONENT Register_Bank
Port (
        Value_In : in STD_LOGIC_VECTOR(3 downto 0);
        Clk      : in STD_LOGIC;
        Reg_En   : in STD_LOGIC_VECTOR(2 downto 0);
        Reset    : in STD_LOGIC;
        R0       : out STD_LOGIC_VECTOR(3 downto 0);
        R1       : out STD_LOGIC_VECTOR(3 downto 0);
        R2       : out STD_LOGIC_VECTOR(3 downto 0);
        R3       : out STD_LOGIC_VECTOR(3 downto 0);
        R4       : out STD_LOGIC_VECTOR(3 downto 0);
        R5       : out STD_LOGIC_VECTOR(3 downto 0);
        R6       : out STD_LOGIC_VECTOR(3 downto 0);
        R7       : out STD_LOGIC_VECTOR(3 downto 0)
    );
end COMPONENT;


--done------------------ 4-bit Add/Sub Unit --------------------
COMPONENT ALU
PORT(      A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Bit_Select : in STD_LOGIC_VECTOR (1 downto 0);
           AddSub_Select : in STD_LOGIC;
           Mode : in STD_LOGIC;
           F_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC);
end COMPONENT;
```

```
--done-------------------- 8-way 4-bit Mux ----------------------
COMPONENT MUX_8_W_4_B
PORT( A0 : in STD_LOGIC_VECTOR (3 downto 0);
A1 : in STD_LOGIC_VECTOR (3 downto 0);
A2 : in STD_LOGIC_VECTOR (3 downto 0);
A3 : in STD_LOGIC_VECTOR (3 downto 0);
A4 : in STD_LOGIC_VECTOR (3 downto 0);
A5 : in STD_LOGIC_VECTOR (3 downto 0);
A6 : in STD_LOGIC_VECTOR (3 downto 0);
A7 : in STD_LOGIC_VECTOR (3 downto 0);
S : in STD_LOGIC_VECTOR (2 downto 0);
C_OUT : out STD_LOGIC_VECTOR (3 downto 0));
end COMPONENT;



---------------------- 2-way 3-bit Mux ---------------------- (Done)
COMPONENT Mux2_3bit
PORT( A : in STD_LOGIC_VECTOR(2 downto 0);  -- Input 0
        B : in STD_LOGIC_VECTOR(2 downto 0);  -- Input 1
        Sel : in STD_LOGIC;                   -- Select line
        Y : out STD_LOGIC_VECTOR(2 downto 0) );
end COMPONENT;


--done-------------------- 2-way 4-bit Mux ----------------------
COMPONENT w2_4_MUX
PORT( Load_Select : in STD_LOGIC;
Immediate : in STD_LOGIC_VECTOR (3 downto 0);
S : in STD_LOGIC_VECTOR (3 downto 0);
Value_In : out STD_LOGIC_VECTOR (3 downto 0));
end COMPONENT;


--done---------------------- Slow Clock ------------------------
COMPONENT Slow_Clk
PORT( Clk_in : in STD_LOGIC;
Clk_out : out STD_LOGIC);
end COMPONENT;


--done---------------------- Seven Segment ----------------------
COMPONENT LUT_16_7
PORT( address : in STD_LOGIC_VECTOR (3 downto 0);
data    : out STD_LOGIC_VECTOR (6 downto 0));
end COMPONENT;


------------------------- SIGNALS -------------------------
```

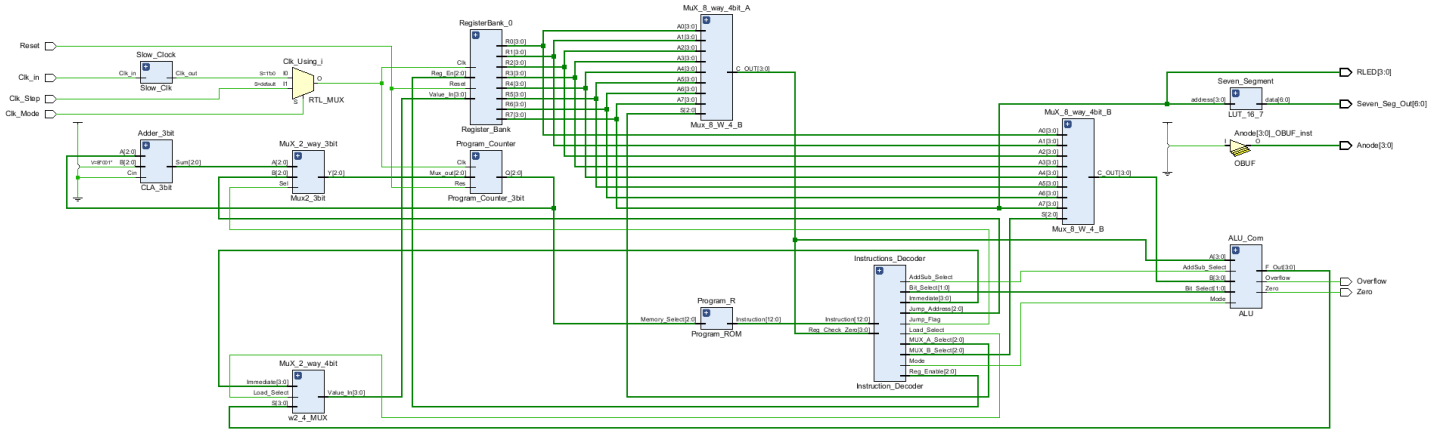
```
SIGNAL Slw_Clk : STD_LOGIC;
SIGNAL Clk_Using : STD_LOGIC;
SIGNAL dummy_cout  : STD_LOGIC;
SIGNAL Res  : STD_LOGIC;

SIGNAL P_Counter_Out    : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL ADDER_3_Out      : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL MUX_2_1_3bit_Out : STD_LOGIC_VECTOR (2 downto 0);

SIGNAL I                : STD_LOGIC_VECTOR (12 downto 0);

SIGNAL Reg_Sel_MuxA  : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL Reg_Sel_MuxB  : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL Decoder_Val   : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL Address_JMP   : STD_LOGIC_VECTOR (2 downto 0);
SIGNAL Bit_Select_S   : STD_LOGIC_VECTOR (1 downto 0);

SIGNAL Im_Value      : STD_LOGIC_VECTOR (3 downto 0);
SIGNAL Load_Sel : STD_LOGIC;
SIGNAL Add_or_Sub : STD_LOGIC;
SIGNAL Jmp_Flag : STD_LOGIC;
SIGNAL Mode_S  : STD_LOGIC;

SIGNAL D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7, M_A, M_B, M_0, Result :
STD_LOGIC_VECTOR (3 downto 0);


-------------------------- Mapping --------------------------
begin

Clock_type : Process(Clk_Mode, Slw_Clk, Clk_Step)
begin
case Clk_Mode is
  when '1' => Clk_Using <= Clk_Step;
  when '0' => Clk_Using <= Slw_Clk;
  when others => Clk_Using <= Slw_Clk;
end case;
end process;

Res <= Reset;

Slow_Clock : Slow_Clk --(Done)
PORT MAP ( Clk_in => Clk_in,
          Clk_out => Slw_Clk );


Program_Counter : Program_Counter_3bit --(Done)
PORT MAP (
    Mux_out => MUX_2_1_3bit_Out,
    Clk     => Clk_Using,
    Res     => Res,
    Q       => P_Counter_Out);
```

```
--done
ALU_Com : ALU
PORT MAP (
               A => M_A,
               B => M_B,
   AddSub_Select => Add_or_Sub,
        Overflow => Overflow,
            Zero => Zero,
            Mode=> Mode_S,
            F_Out=>Result,
               Bit_Select => Bit_Select_S);


Adder_3bit : CLA_3bit --(Done)
PORT MAP (
        A => P_Counter_Out,
        B =>"001",
      Cin =>'0',
     Cout => dummy_cout,

      Sum => ADDER_3_Out);


MuX_2_way_3bit : Mux2_3bit --(Done)
PORT MAP ( Sel => Jmp_Flag,
           A  => ADDER_3_Out,
           B  => Address_JMP,
           Y  => MUX_2_1_3bit_Out);


Program_R : Program_ROM --(done)
PORT MAP ( Memory_Select   => P_Counter_Out,
           Instruction => I);



Instructions_Decoder : Instruction_Decoder --(doone)
PORT MAP (
           Instruction => I,
           Reg_Check_Zero => M_A,
           Reg_Enable     => Decoder_Val,
           Load_Select    => Load_Sel,
           Immediate      => Im_Value,
           MUX_A_Select    => Reg_Sel_MuxA,
           MUX_B_Select    => Reg_Sel_MuxB,
           AddSub_Select  =>  Add_or_Sub,
           Jump_Flag       => Jmp_Flag,
           Mode       => Mode_S,
           Bit_Select       => Bit_Select_S,
           Jump_Address    => Address_JMP);
```

```
--DONE
RegisterBank_0 : Register_Bank
    port map(
   Value_In => M_0,
        Clk => Clk_Using,
     Reg_En => Decoder_Val,
      Reset => Res,

   R0 => D_0,
   R1 => D_1,
   R2 => D_2,
   R3 => D_3,
   R4 => D_4,
   R5 => D_5,
   R6 => D_6,
   R7 => D_7
);


--DONE
MuX_8_way_4bit_A : MUX_8_W_4_B
PORT MAP ( A0 => D_0,
           A1 => D_1,
           A2 => D_2,
           A3 => D_3,
           A4 => D_4,
           A5 => D_5,
           A6 => D_6,
           A7 => D_7,
           S => Reg_Sel_MuxA,
           C_OUT => M_A);



MuX_8_way_4bit_B : MUX_8_W_4_B
PORT MAP ( A0 => D_0,
A1 => D_1,
A2 => D_2,
A3 => D_3,
A4 => D_4,
A5 => D_5,
A6 => D_6,
A7 => D_7,
S => Reg_Sel_MuxB,
C_OUT => M_B);

--DONE
MuX_2_way_4bit : w2_4_MUX
PORT MAP ( Load_Select => Load_Sel,
Immediate => Im_Value,
S => Result,
```

```
Value_In=> M_0);

--done
Seven_Segment : LUT_16_7
PORT MAP ( address => D_7,
data    => Seven_Seg_Out);

RLED  <= D_7;
Anode <= "1110";

end Behavioral;
```

## Nano processor Design Schematic



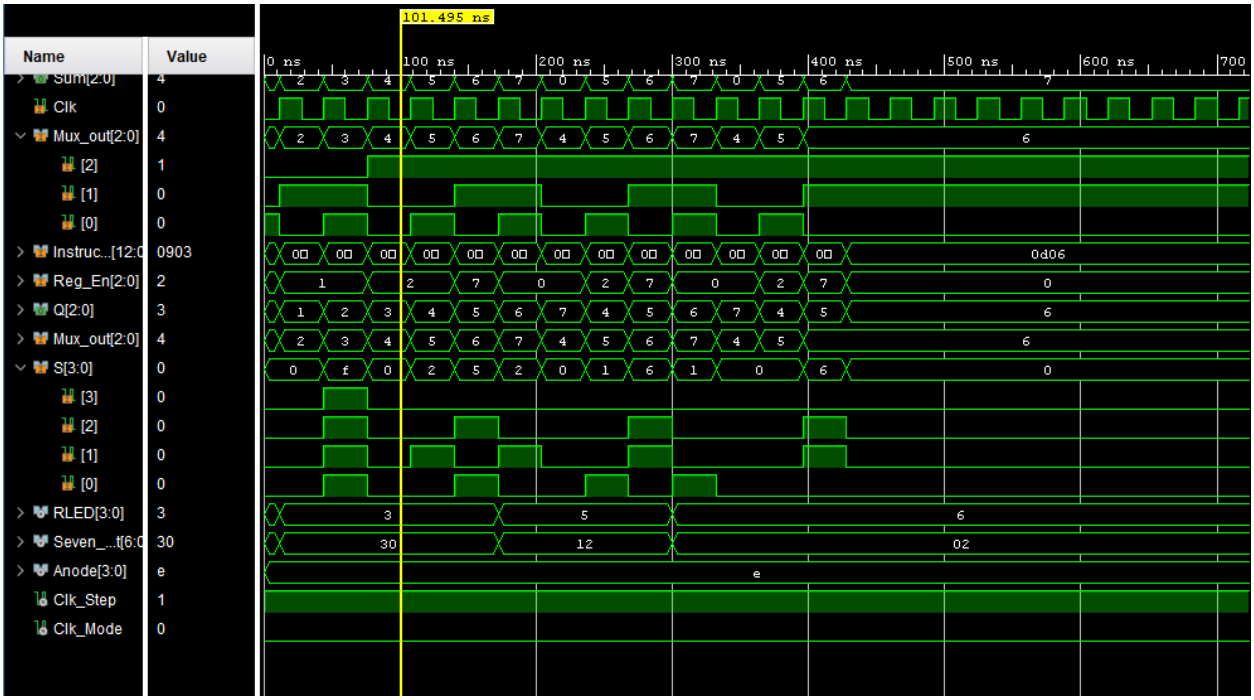

## Nano processor - Test Bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.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;
            Clk_in         : in  STD_LOGIC;
            Zero           : out STD_LOGIC;
            Overflow       : out STD_LOGIC;
            RLED           : out STD_LOGIC_VECTOR(3 downto 0);
```

```
            Seven_Seg_Out : out STD_LOGIC_VECTOR(6 downto 0);
            Anode         : out STD_LOGIC_VECTOR(3 downto 0);
            Clk_Step      : STD_LOGIC;
            Clk_Mode      : STD_LOGIC
        );
    END COMPONENT;

    -- Signals for stimulus and output observation
    signal Reset         : STD_LOGIC := '1';
    signal Clk_in        : STD_LOGIC := '0';
    signal Zero          : STD_LOGIC;
    signal Overflow      : STD_LOGIC;
    signal RLED          : STD_LOGIC_VECTOR(3 downto 0);
    signal Seven_Seg_Out : STD_LOGIC_VECTOR(6 downto 0);
    signal Anode         : STD_LOGIC_VECTOR(3 downto 0);

begin

    -- Instantiate the Nano Processor
    uut: Nano_Processor
        PORT MAP (
            Reset         => Reset,
            Clk_in        => Clk_in,
            Zero          => Zero,
            Overflow      => Overflow,
            RLED          => RLED,
            Seven_Seg_Out => Seven_Seg_Out,
            Anode         => Anode,
            Clk_Step      => '0',
            Clk_Mode      => '0'
        );

    -- Clock generation: 5 ns period (200 MHz)
    clk_process : process
    begin
        while true loop
            Clk_in <= '0';
            wait for 4 ns;
            Clk_in <= '1';
            wait for 4 ns;
        end loop;
    end process;

    -- Reset and simulation logic
    stim_proc: process
    begin
        -- Apply reset pulse
--        Reset <= '1';
--        wait for 42 ns;
        Reset <= '0';

--        wait for 100 ns;
```

```
--         Reset <= '1';
--         wait for 42 ns;
--         Reset <= '0';

        -- Let it run for some cycles (adjust as needed)
        wait for 1000 ns;

        -- End simulation
        wait;
    end process;

end Behavioral;
```

## Nano processor Timing Diagram



## Constraint File

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

## Switches

set_property PACKAGE_PIN R2 [get_ports {Clk_Mode}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Clk_Mode}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {Zero}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Zero}]
set_property PACKAGE_PIN E19 [get_ports {Overflow}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Overflow}]
set_property PACKAGE_PIN P3 [get_ports {RLED[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {RLED[0]}]
set_property PACKAGE_PIN N3 [get_ports {RLED[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {RLED[1]}]
set_property PACKAGE_PIN P1 [get_ports {RLED[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {RLED[2]}]
set_property PACKAGE_PIN L1 [get_ports {RLED[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {RLED[3]}]

##7 segment display
set_property PACKAGE_PIN W7 [get_ports {Seven_Seg_Out[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_Out[0]}]
set_property PACKAGE_PIN W6 [get_ports {Seven_Seg_Out[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_Out[1]}]
set_property PACKAGE_PIN U8 [get_ports {Seven_Seg_Out[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_Out[2]}]
set_property PACKAGE_PIN V8 [get_ports {Seven_Seg_Out[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_Out[3]}]
set_property PACKAGE_PIN U5 [get_ports {Seven_Seg_Out[4]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_Out[4]}]
set_property PACKAGE_PIN V5 [get_ports {Seven_Seg_Out[5]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_Out[5]}]
set_property PACKAGE_PIN U7 [get_ports {Seven_Seg_Out[6]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Seven_Seg_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}]
set_property PACKAGE_PIN T18 [get_ports Clk_Step]
      set_property IOSTANDARD LVCMOS33 [get_ports Clk_Step]
```

## Contribution from Members

| Member | Contribution |
|---|---|
| ATHTHANAYAKE A.M.R.N. 230062V | • 2-way 3-bit Multiplexer<br>• Program Counter<br>• 3 Bit Adder<br>• Test Benches for All Components<br>• Instruction Decoder |
| IFAZ M.I.M. 230253H | • 8-way 4-bit Multiplexer<br>• Register<br>• Register Bank<br>• 3-8 Decoder<br>• Final Lab Report |
| KUMARASINGHE M.P. 230356C | • Program ROM<br>• Program Counter<br>• Instruction Decoder<br>• Bitwise Operator<br>• Nano Processor |
| LAWANYA K.K.H.G. 230373B | • Half Adder<br>• Full Adder<br>• Slow Clock<br>• 2-Way 4-bit Multiplexer<br>• 4-bit Adder Subtractor Unit<br>• Seven Segment Display |