CS2052 Computer Architecture

Department of Computer Science and Engineering, University of Moratuwa

Lab 9– Nano processor Design Competition

Prepared By : Ayesh Vininda - 190649F
Nipun Pramuditha - 190653L
Dasun Nimantha - 190415K

Table of Contents

Introduction

Project

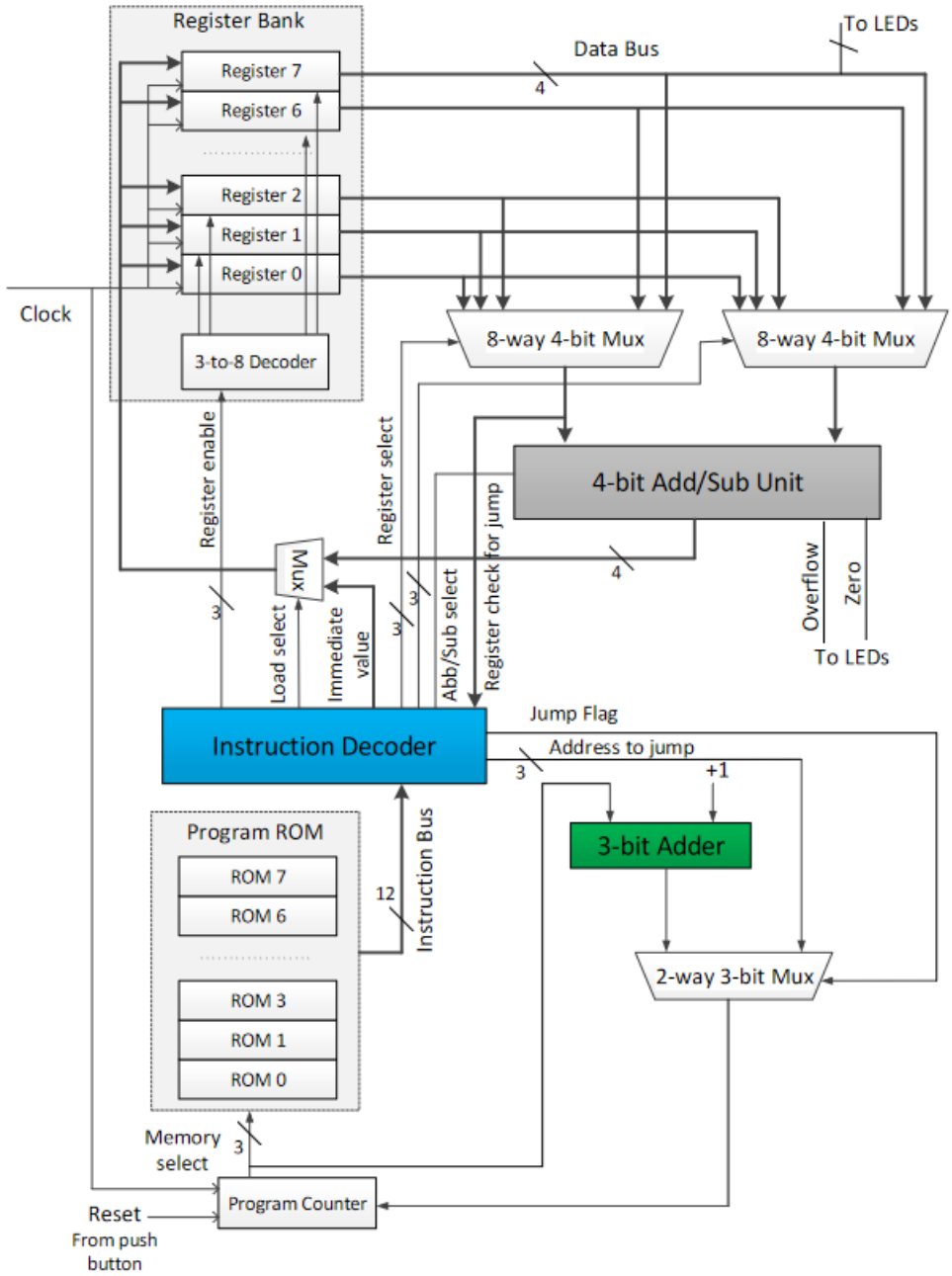
Project Name: Lab 9

Product Family: Artix-7

Project part: Basys 3(xc7a35tcpg236-1)

Target Language: VHDL

Design View

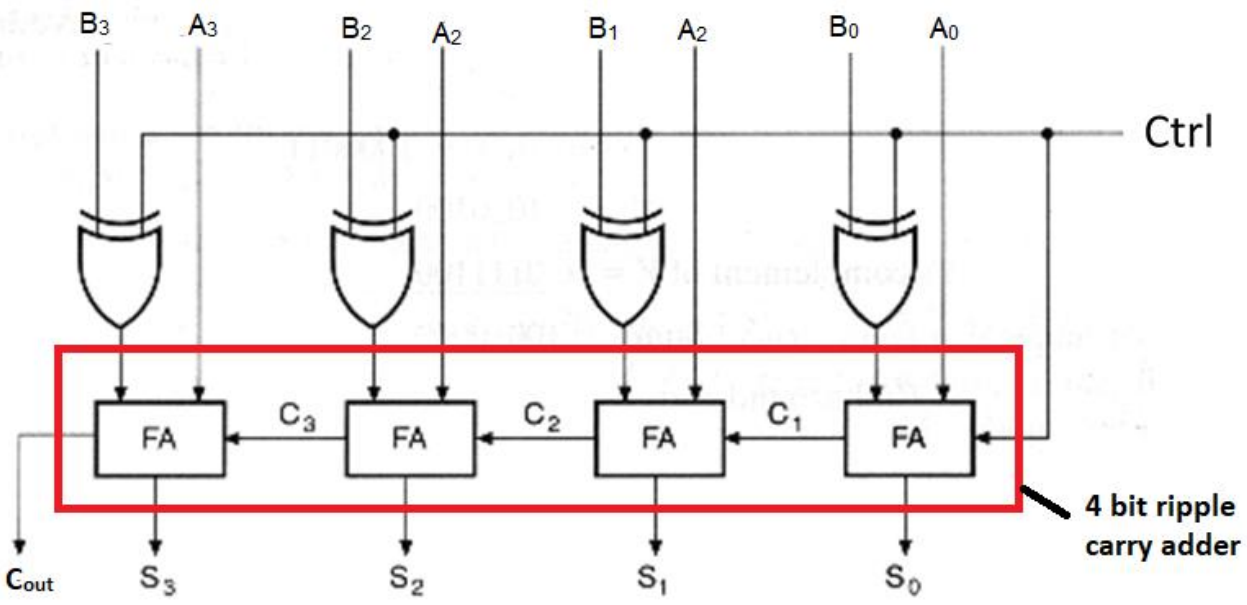


Description

- In this lab, we will design a 4-bit processor capable of executing 4 instructions (MOVI, ADD, NEG, and JZR) we need to design and build the following components, In order to build the 4-bit nano processor,
  - 4 – bit add/subtract unit.
  - 3 – bit Adder
  - 3 – bit Program Counter (PC).
  - k – way b – bit Multiplexer.
  - Register Bank.
  - Program ROM.
  - Instruction Decoder.
- The steps taken in this assignment are as follows,
  - First, we implemented components using the VHDL language.
  - At each step we verified their functionality via simulation, obtained timing diagrams for each component.
  - By writing simulation codes for each component, we verified their functionalities and obtained timing diagrams for each.
  - We used XDC VHDL Code to map the inputs and outputs to the BASYS3 Board.
  - We used seven segment unit and 4 LEDs to show the value of register 7
  - Used 2 LEDs for displaying Zero and Overflow flags

4-bit Add Sub Unit

- From previous labs, we can use the 4-bit ripple carry adder for adding part of this unit.
- We can use the 4-bit ,2's complement Adder Subtractor for this unit.



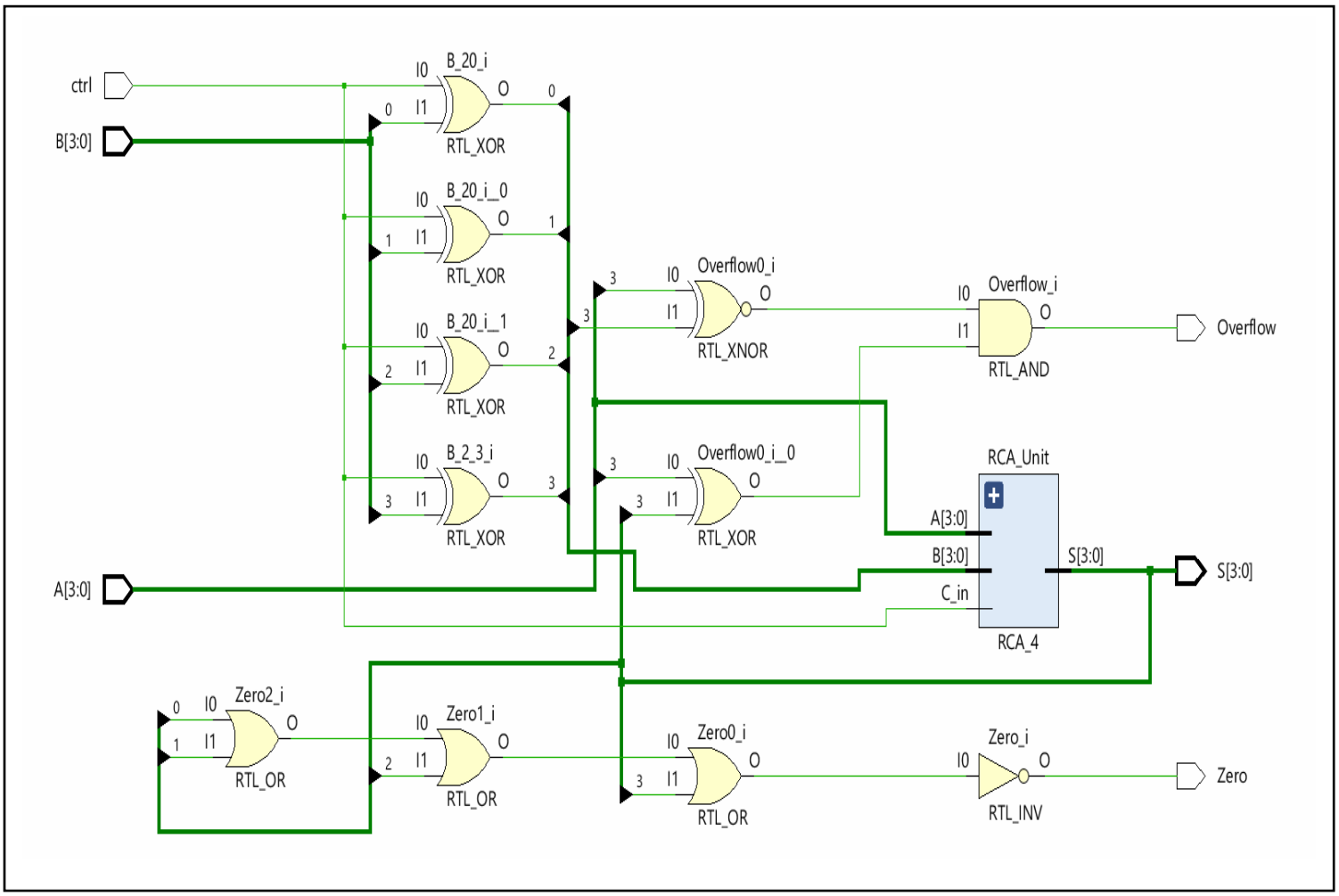
VHDL code for Add Sub unit

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



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

architecture Behavioral of Add_Sub_Unit is

```

```
--import RCA_4
component RCA_4
    port(
        A: in std_logic_vector(3 downto 0);
        B: in std_logic_vector(3 downto 0);
        C_in: in std_logic;
        S: out std_logic_vector(3 downto 0);
        C_out: out std_logic
    );
end component ;

  signal RCA_C :std_logic;
  signal B_2:std_logic_vector(3 downto 0);
  signal S_0:std_logic_vector(3 downto 0);

begin
    --port map for the RCA
    RCA_Unit:RCA_4
        Port map(
           A =>A ,
           B =>B_2,
           C_in =>ctrl,
           S =>S_0,
           C_out =>RCA_C);

    -- create the adder subtractor part
    B_2(0) <= ctrl XOR B(0);
    B_2(1) <= ctrl XOR B(1);
    B_2(2) <= ctrl XOR B(2);
    B_2(3) <= ctrl XOR B(3);
```

```
    -- result logics

    S<= S_0;
    Overflow <= ((A(3) xnor B_2(3)) and (A(3) xor S_0(3)));
    Zero<= NOT(S_0(0) OR S_0(1) OR S_0(2) OR S_0(3));

end Behavioral;
```

Elaborated design view for Add Sub Unit

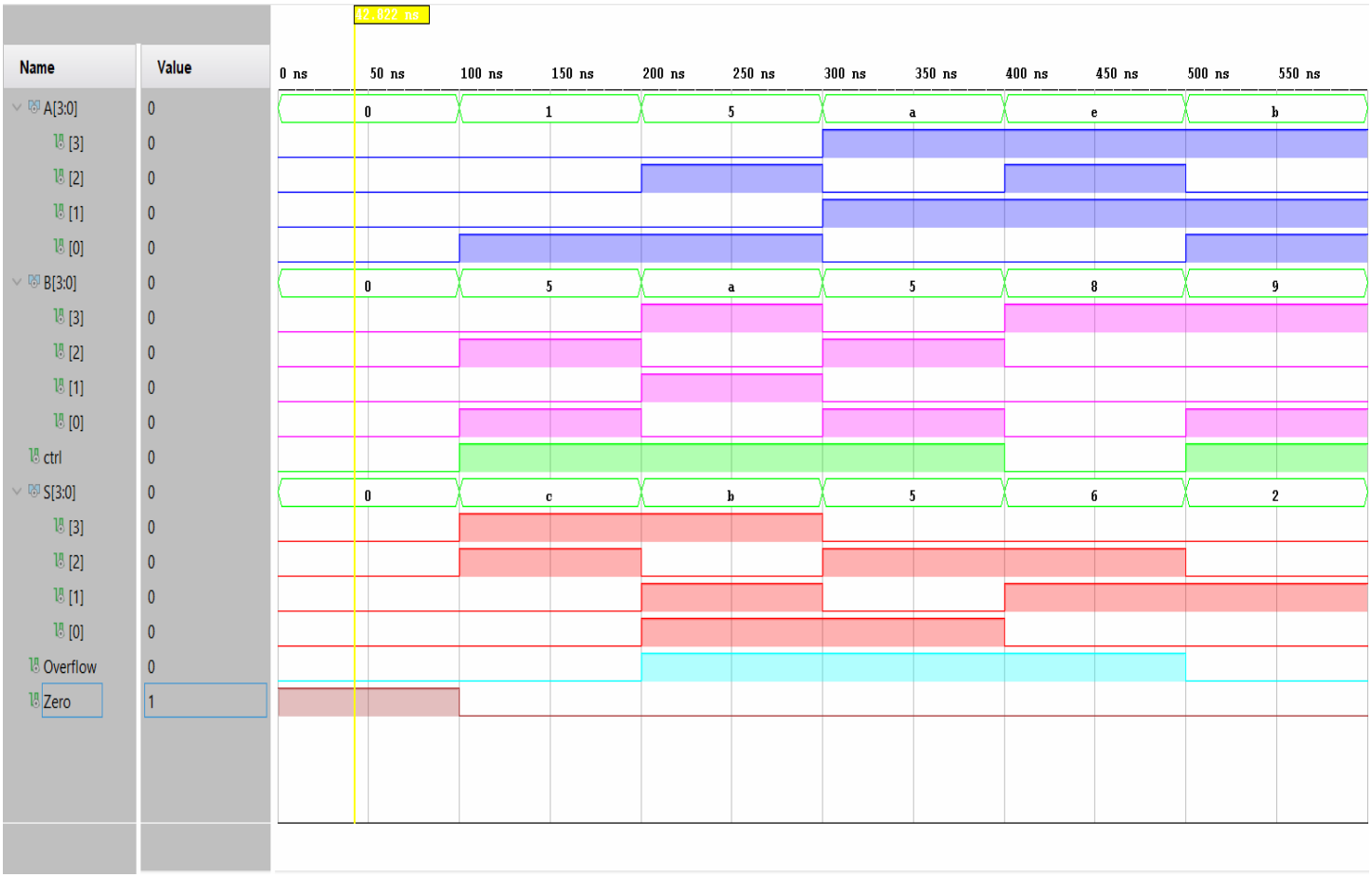


Simulation code (Test bench) for Add Sub Unit

```
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_Unit
    Port (
           A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           ctrl : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC
    );
end component;

signal A,B :std_logic_vector(3 downto 0);
signal ctrl:std_logic;
signal S: std_logic_vector(3 downto 0);
signal Overflow,Zero :std_logic;

begin
--port map for the Add_Sub_Unit
UUT:Add_Sub_Unit
    Port map (
           A =>A,
           B =>B,
           ctrl=>ctrl,
           S =>S,
           Overflow =>Overflow,
           Zero =>Zero);
process
```

```
    begin
        A   <= "0000";
        B   <= "0000";
        ctrl <= '0';
        wait for 100ns;

        A <= "0001";
        B <= "0101";
        ctrl <= '1';
        wait for 100ns;

        A <= "0101";--  5
        B <= "1010";-- -6         1
        ctrl <= '1';
        wait for 100ns;

        A <= "1010"; -- -6
        B <= "0101"; -- 5
        ctrl <= '1';
        wait for 100ns;

        --190649F --> 10_1110_1000_1011_1001
        A <= "1110";
        B <= "1000";
        ctrl <= '0';
        wait for 100ns;

        A<= "1011";
        B<= "1001";
        ctrl<='1';

        wait;
end process;
```

Simulation diagram for Add Sub Unit



3-bit Adder

- To implement 3-bit adder we need ripple carry adder.
- For that we design 3-bit ripple carry adder using previous 4-bit ripple carry adder

VHDL code for 3-bit Ripple Carry Adder

- 3-bit adder use for to get address of next instruction. Which we store inside the program counter
- 3-bit adder increment the previous address by one.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity RCA_3 is
    port(
        C_in: in std_logic;
```

```
        A: in std_logic_vector(2 downto 0);
        B: in std_logic_vector(2 downto 0);
        S: out std_logic_vector(2 downto 0);
        C_out: out std_logic);
end RCA_3;
architecture Behavioral of RCA_3 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_C, FA1_C : std_logic;
begin
    FA_0 : FA
        port map (
            A => A(0),
            B => B(0),
            C_in =>C_in,--set ground
            S => S(0),
            C_Out => FA0_C);
    FA_1 : FA
        port map (
            A => A(1),
            B => B(1),
            C_in => FA0_C,
            S => S(1),
            C_Out => FA1_C);
```

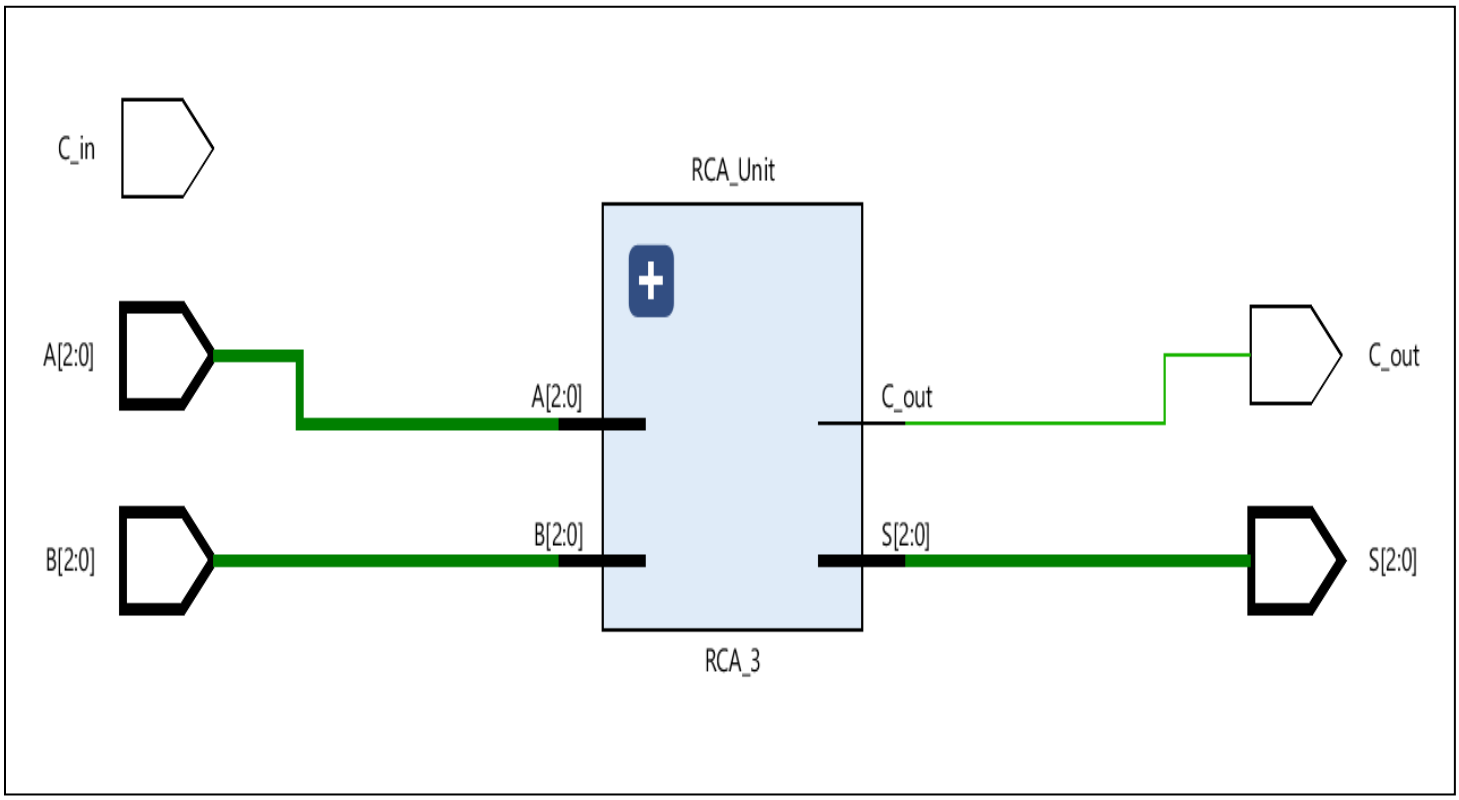
```
    FA_2 : FA
        port map (
            A => A(2),
            B => B(2),
            C_in => FA1_C,
            S => S(2),
            C_Out => C_out);
end Behavioral;
```

VHDL code for 3-bit Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Bit_3_Adder is
    Port (
            A : in STD_LOGIC_VECTOR(2 downto 0); --get inputs for A as
vector array (first number)
            B : in STD_LOGIC_VECTOR(2 downto 0); --get inputs for B as
vector array (seconed number)
            C_in : in STD_LOGIC;                 --initial carry bit for
first adding
            S : out STD_LOGIC_VECTOR(2 downto 0);--sum output
            C_out : out STD_LOGIC                --carry output
);
end Bit_3_Adder;
architecture Behavioral of Bit_3_Adder is
-- import RCA_4
component RCA_3
    port(
        C_in: in std_logic;
        A: in std_logic_vector(2 downto 0);
        B: in std_logic_vector(2 downto 0);
```

```
        S: out std_logic_vector(2 downto 0);
        C_out: out std_logic);
end component;
    signal RCA_out:std_logic;
begin
    RCA_Unit:RCA_3
        port map(
            C_in => C_in,
            A=>A,
            B=>B,
            S=>S,
            C_out=>C_out);
end Behavioral;
```

Elaboration design view (RTL) for 3-bit adder

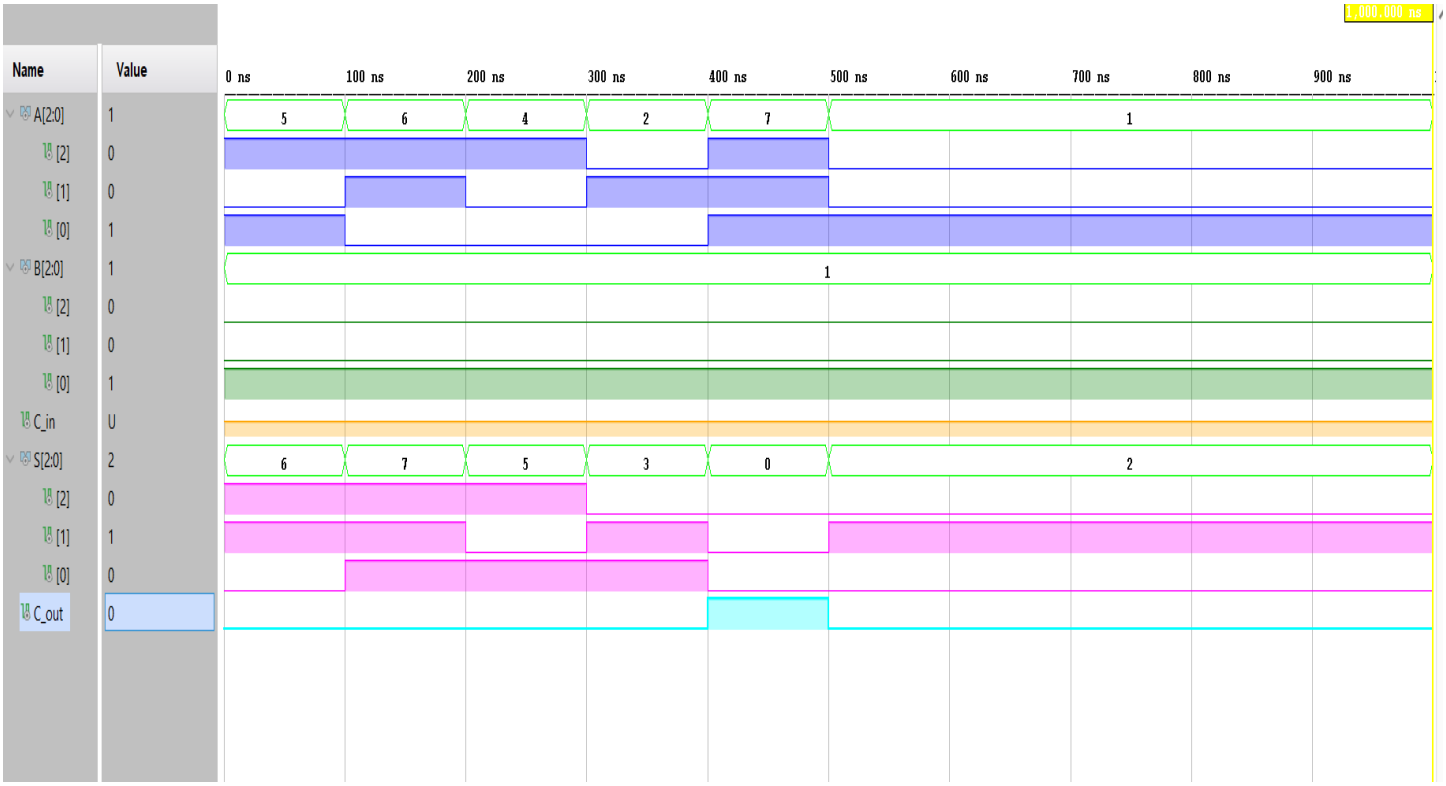


Simulation code (Test bench) for 3-bit adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Bit_3_Adder_TB is
--  Port ( );
end Bit_3_Adder_TB;
architecture Behavioral of Bit_3_Adder_TB is
component Bit_3_Adder
    port ( A : in STD_LOGIC_VECTOR(2 downto 0); --get inputs for A as
vector array (first number)
           B : in STD_LOGIC_VECTOR(2 downto 0); --get inputs for B as
vector array (seconed number)
           C_in : in STD_LOGIC;                 --initial carry bit for
first adding
           S : out STD_LOGIC_VECTOR(2 downto 0);--sum output
           C_out : out STD_LOGIC                --carry output);
end component;
signal A,B:std_logic_vector(2 downto 0);
signal C_in:std_logic;
signal S :std_logic_vector(2 downto 0);
signal C_out:std_logic;
begin
UUT:Bit_3_Adder
    port map(
           A =>A,
           B =>B,
           C_in =>C_in,
           S =>S,
           C_out =>C_out);
process --190649F 101_110_100_010_111_001
begin
```

```
    A <= "101";
    B <= "001";
    wait for 100ns;
    A <= "110";
    B <= "001";
    wait for 100ns;
    A <= "100";
    B <= "001";
    wait for 100ns;
    A <= "010";
    B <= "001";
    wait for 100ns;

    A <= "111";
    B <= "001";
    wait for 100ns;
    A <= "001";
    B <= "001";
    wait for 100ns;
    wait;
end process;
end Behavioral;
```

Timing diagram for 3-bit Adder



Instruction Decoder

- Instruction decoder is important part of the nano processor. It decodes the instruction and allocates tasks to the specific sections.
- This unit consists with 2 to 4 decoder to decode opcode of given instruction

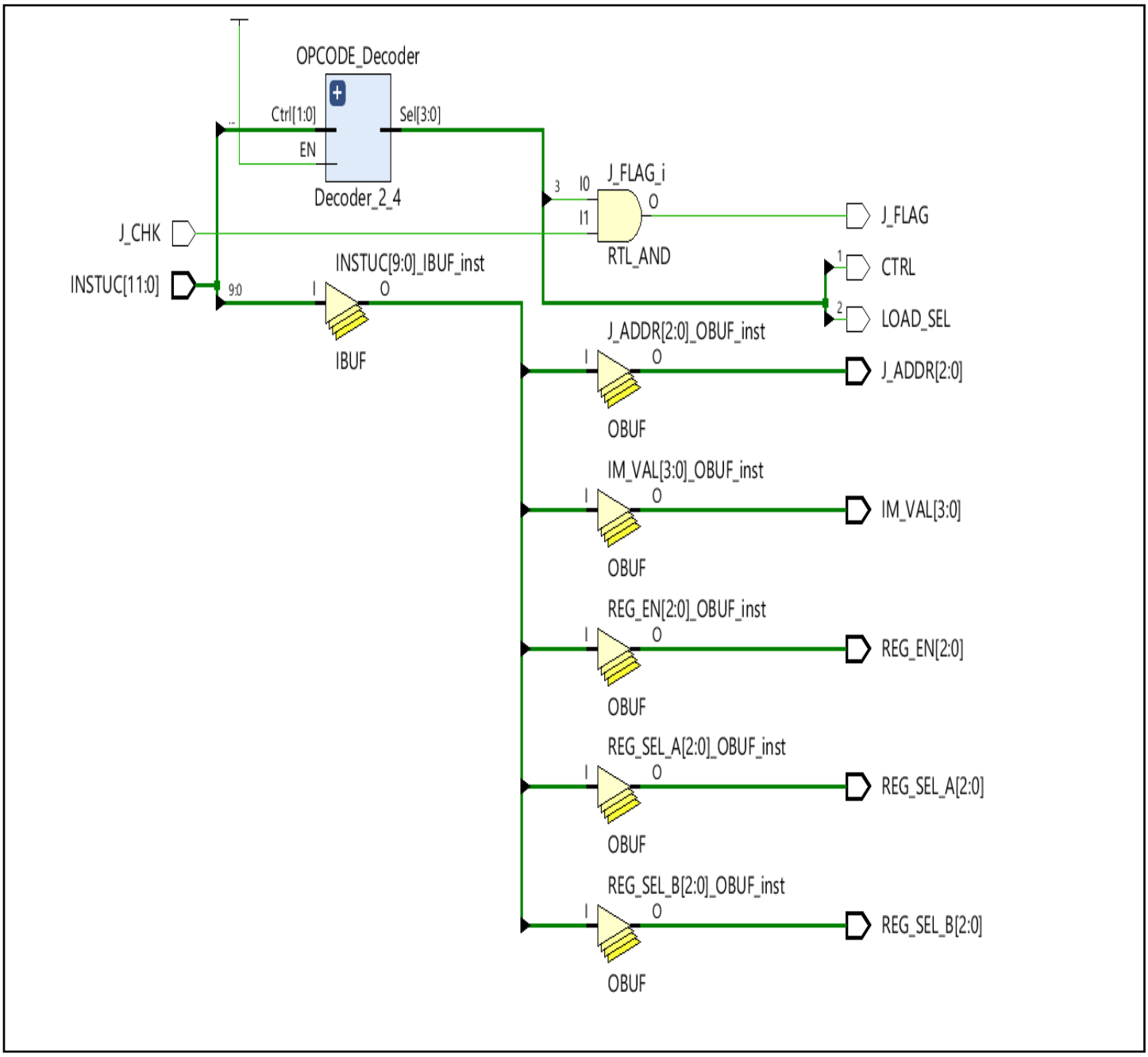
VHDL code for Instruction Decoder

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

entity Instruction_Decoder is

    Port (
        INSTUC : in STD_LOGIC_VECTOR (11 downto 0);      --Get
instruction from programme rom
        J_CHK : in STD_LOGIC;                            --Jump check

        REG_EN : out STD_LOGIC_VECTOR (2 downto 0);      --Register
enable in register bank
```

```
        REG_SEL_A : out STD_LOGIC_VECTOR (2 downto 0);  --Register
selection for multiplexer A
        REG_SEL_B : out STD_LOGIC_VECTOR (2 downto 0);  --Register
selection for multiplexer B


        LOAD_SEL : out STD_LOGIC;                       --Load selction
for multiplexer 0
        IM_VAL : out STD_LOGIC_VECTOR (3 downto 0);     --Immediate
value for multiplexer 0


        CTRL : out STD_LOGIC;                           --Add Subract
selection for Add_Sub_Unit


        J_FLAG : out STD_LOGIC;                         --Jump Flag for
multiplexer 1
        J_ADDR : out STD_LOGIC_VECTOR (2 downto 0)      --Jump address
for multiplexer 1
        );
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is
component Decoder_2_4
   Port (
        Ctrl : in STD_LOGIC_VECTOR (1 downto 0);
        EN : in std_logic;
        Sel : out STD_LOGIC_VECTOR (3 downto 0)
    );
end component;
signal ADD,NEG,MOV,JZR :std_logic;
begin
    --decode opcode into four instructions
    OPCODE_Decoder:Decoder_2_4
        Port map (
            Ctrl =>INSTUC(11 downto 10),
            EN   =>'1',
            Sel(0)=>ADD,
            Sel(1)=>NEG,
            Sel(2)=>MOV,
            Sel(3)=>JZR);
    CTRL <= NEG; --select add or sub
```

```
    LOAD_SEL <= MOV; --Select the load comes form instruction decoder or
add sub unit

    IM_VAL <= INSTUC(3 downto 0);--1 0 R R R 0 0 0 [d d d d]

    REG_EN  <=INSTUC(9 downto 7);--Reg enable  1 0 [R R R] 0 0 0 d d d d

    --commad for the  muxA and muxB to register selection
    REG_SEL_A <=INSTUC(9 downto 7);
    REG_SEL_B <=INSTUC(6 downto 4);

    --Register check for jump-----------------------------
    J_FLAG <= JZR and J_CHK; --check and set flag true

    J_ADDR<=INSTUC(2 downto 0);--set jump address --

end Behavioral;
```

Elaborated Design View of Instruction Decoder

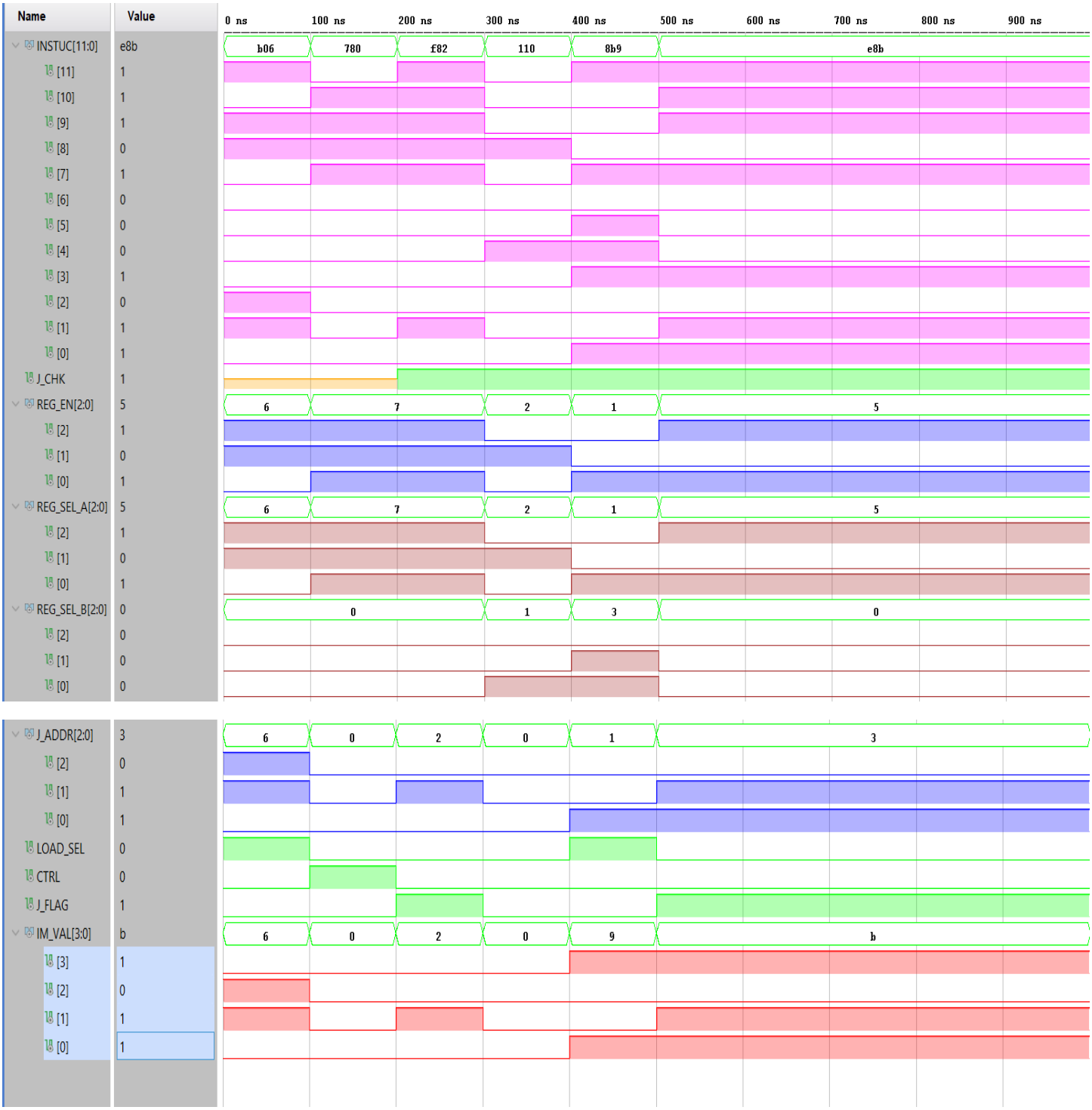


Simulation code (Test bench) for Instruction Decoder

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

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

--import Intruction Decoder
architecture Behavioral of Instruction_Decoder_TB is
    component Instruction_Decoder
        Port (
        INSTUC : in STD_LOGIC_VECTOR (11 downto 0);     --Get
instruction from programme rom
        J_CHK : in STD_LOGIC;                           --Jump check

        REG_EN : out STD_LOGIC_VECTOR (2 downto 0);     --Register
enable in register bank

        REG_SEL_A : out STD_LOGIC_VECTOR (2 downto 0);  --Register
selection for multiplexer A
        REG_SEL_B : out STD_LOGIC_VECTOR (2 downto 0);  --Register
selection for multiplexer B

        LOAD_SEL : out STD_LOGIC;                       --Load selction
for multiplexer 0
        IM_VAL : out STD_LOGIC_VECTOR (3 downto 0);     --Immediate
value for multiplexer 0

        CTRL : out STD_LOGIC;                           --Add Subract
selection for Add_Sub_Unit

        J_FLAG : out STD_LOGIC;                         --Jump Flag for
multiplexer 1
        J_ADDR : out STD_LOGIC_VECTOR (2 downto 0)      --Jump address
for multiplexer 1
        );

end component;
```

```
begin
--port map for Instuction_Deocder
UUT:Instruction_Decoder
        port map (
        INSTUC =>INSTUC,
        J_CHK =>J_CHK,
        REG_EN =>REG_EN,
        REG_SEL_A =>REG_SEL_A,
        REG_SEL_B =>REG_SEL_B,
        LOAD_SEL =>LOAD_SEL,
        IM_VAL =>IM_VAL,
        CTRL =>CTRL,
        J_FLAG =>J_FLAG,
        J_ADDR =>J_ADDR
    );
process
   begin
    INSTUC <= "101100000110"; --MOVE
    wait for 100ns ;

    INSTUC <= "011110000000";  --NEG
    wait for 100ns;

    J_CHK  <= '1';
    INSTUC <= "111110000010"; --JUMP
    wait for 100ns;

    INSTUC <= "000100010000"; --ADD
    wait for 100ns;

    --190649F 10_1110_1000_1011_1001
    INSTUC <= "100010111001";
    wait for 100ns;

    INSTUC <= "111010001011";
    wait;
end process;
end Behavioral;
```

Simulation Diagram for Instruction Decoder



Program Rom

- Program Rom is specific memory that keeps instructions need to execute.
- Use this instruction set to create instructions.

Table 1 – Instruction Set.

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

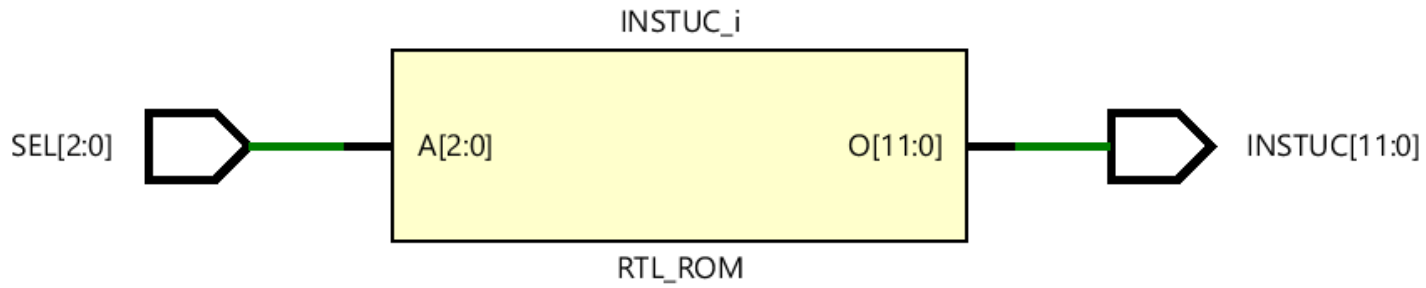
- These instructions used in Instruction Decoder and instructions stored in Program Rom.

VHDL code for Program Rom

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
--get numeric library
entity PROGRM_ROM is
    Port ( SEL : in STD_LOGIC_VECTOR (2 downto 0);
           INSTUC : out STD_LOGIC_VECTOR (11 downto 0));
end PROGRM_ROM;
architecture Behavioral of PROGRM_ROM is

--                  heigth of rom(8)                width of rom(12)
(instruction width)
type rom_type is array (0 to 7) of std_logic_vector(11 downto 0);
    signal PR_ROM:rom_type:=(

            "100010000011", -- MOVI R1, 3
            "100100000001", -- MOVI R2, 1
```

```
            "010100000000", -- NEG R2
            "001110010000", -- ADD R7, R1
            "000010100000", -- ADD R1, R2
            "110010000111", -- JZR R1,7
            "110000000011", -- JZR R0,3
            "110000000111"  -- JZR R0,7 );
begin
    --select instruction for given value
    --use numeric library to convert binary to integer value
    INSTUC <= PR_ROM(to_integer(unsigned(SEL)));
end Behavioral;
```

Elaboration Design View (RTL) for Program Rom

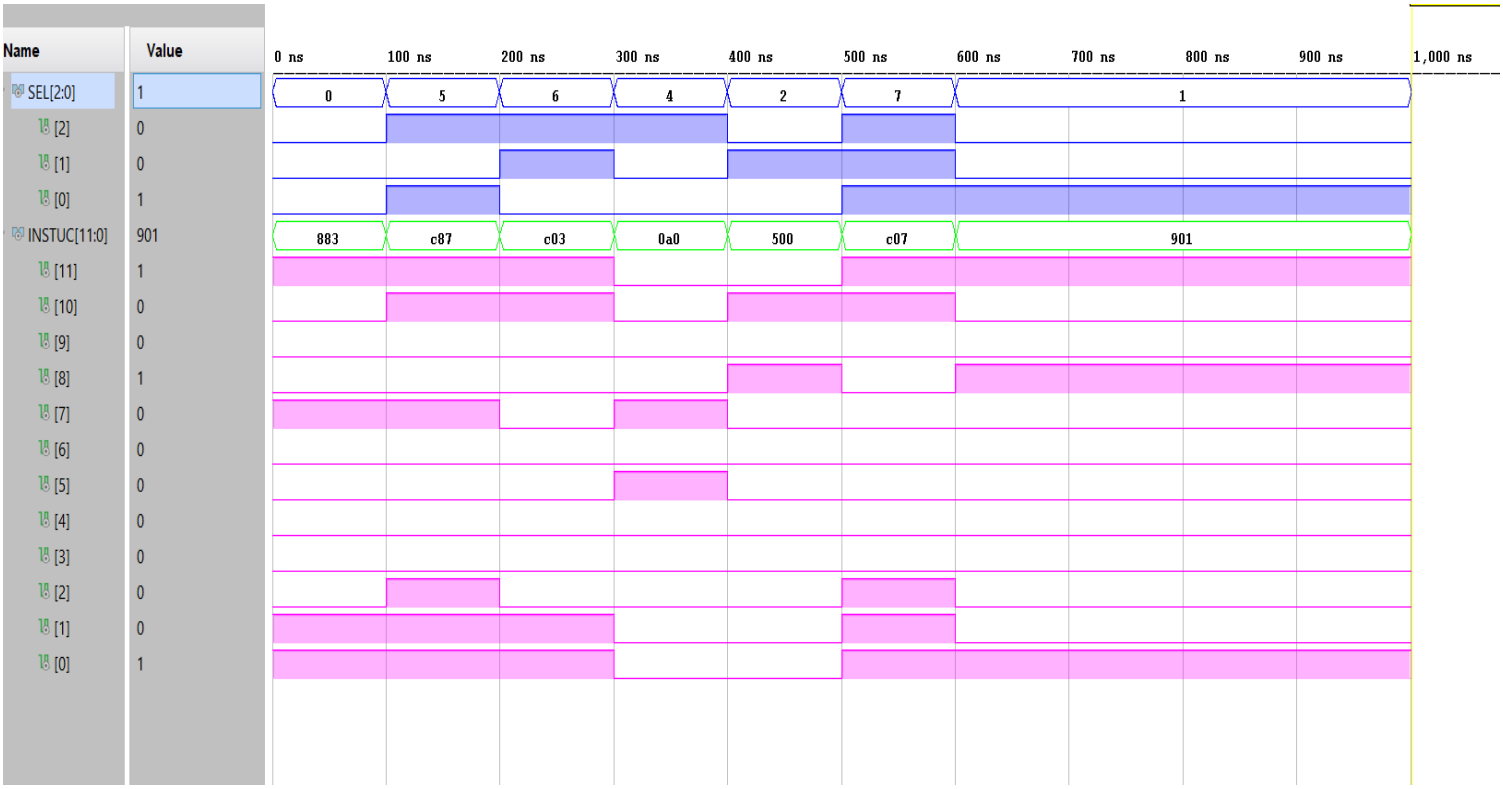


Simulation code (Test bench) for Program Rom

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

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

architecture Behavioral of Program_Rom_TB is
component PROGRM_ROM
    Port (
           SEL : in STD_LOGIC_VECTOR (2 downto 0);
           INSTUC : out STD_LOGIC_VECTOR (11 downto 0)
    );
end component;

signal SEL:std_logic_vector(2 downto 0);
signal INSTUC:std_logic_vector(11 downto 0);

begin
UUT:PROGRM_ROM
    port map(
           SEL =>SEL,
           INSTUC =>INSTUC
    );
--190649F 101_110_100_010_111_001
--190649F 101_110_100_010_111_001
process
    begin
        SEL<="000";
        wait for 100ns;

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

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

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

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

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

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

        wait;
end process;
end Behavioral;
```

Simulation Diagram for Program Rom



3 – bit Program Counter

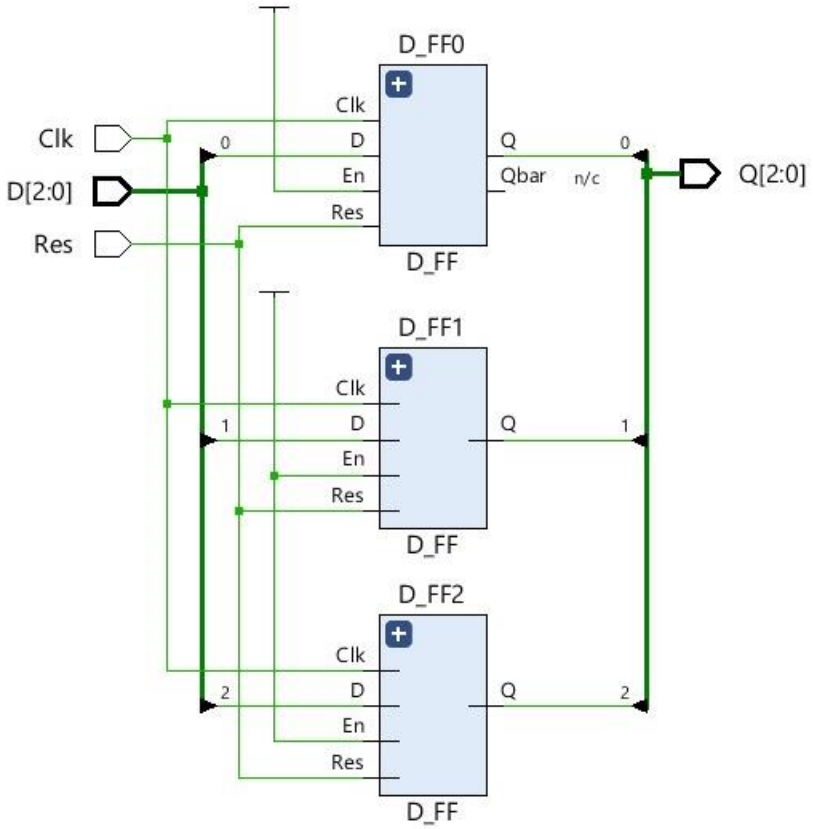
- 3-bit Program Counter is a special purpose register.
- It is built using three D – Flip Flops
- It is used to store address of the next instruction to be executed.

Source Code for 3 – bit Program Counter

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

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

architecture Behavioral of P_Counter is
 component D_FF
 port (
 D : in STD_LOGIC;
 Res: in STD_LOGIC;
 En : in STD_LOGIC;
 Clk : in STD_LOGIC;
 Q : out STD_LOGIC;
 Qbar : out STD_LOGIC);
 end component;
begin

 D_FF0 : D_FF
 port map (
 D => D(0),
 Res => Res,
 En => '1',
 Clk => Clk,
 Q => Q(0));
```

```
D_FF1 : D_FF
 port map (
 D => D(1),
 Res => Res,
 En => '1',
 Clk => Clk,
 Q => Q(1));

 D_FF2 : D_FF
 port map (
 D => D(2),
 Res => Res,
 En => '1',
 Clk => Clk,
 Q => Q(2));
end Behavioral;
```

Elaborated design view (RTL schematic diagram) for Program Counter

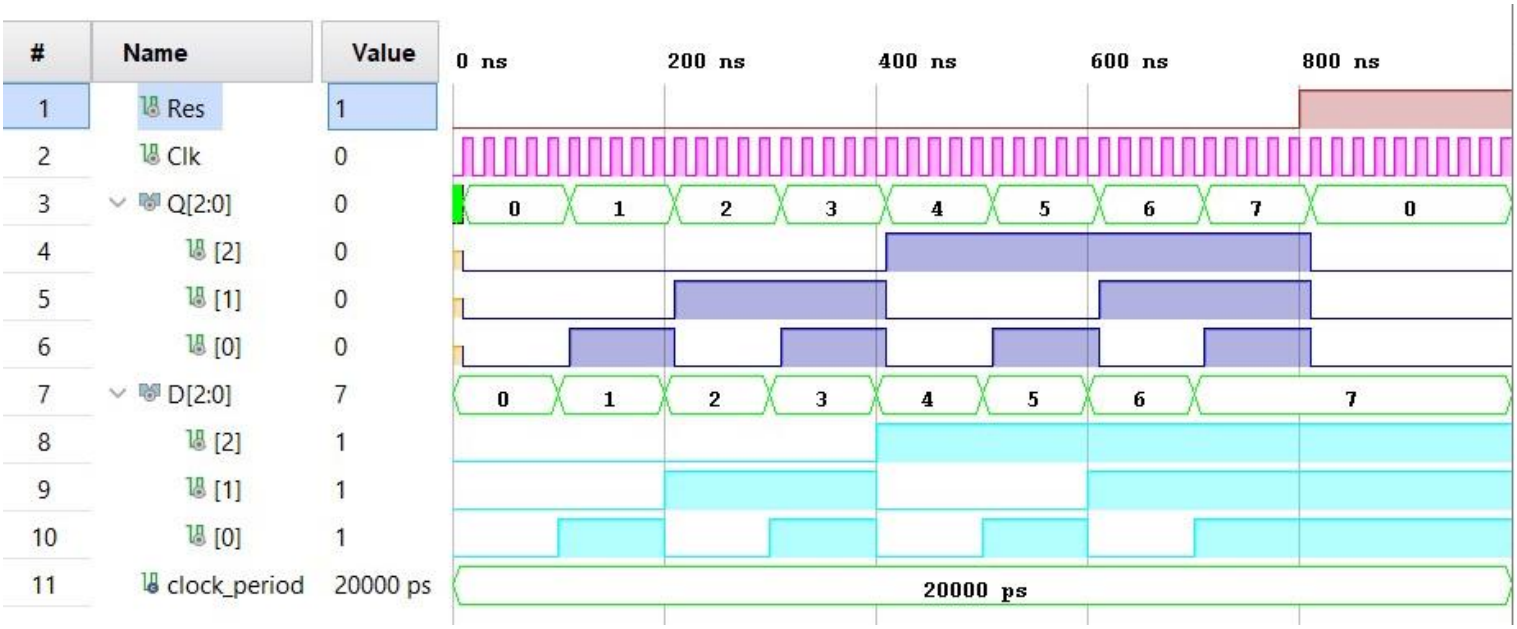


Simulation Code for Program Counter

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

entity P_Counter_TB is
end P_Counter_TB;

architecture Behavioral of P_Counter_TB is
component P_Counter
    port (
        D : in std_logic_vector;
        Res, Clk : in std_logic;
        Q : out std_logic_vector (2 downto 0));
end component;

signal Res, Clk : std_logic;
signal Q, D : std_logic_vector (2 downto 0);
constant clock_period : time := 20ns;

begin

UUT: P_Counter port map(
    D => D,
    Res => Res,
    Clk => Clk,
    Q => Q
);
clock_process: process
  begin
    Clk <= '0';
  wait for clock_period/2;
    Clk <= '1';
  wait for clock_period/2;
end process;
sim: process
```

```
  begin
    D <= "000";
    Res <= '0';
    wait for 100 ns;

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

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

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

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

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

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

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

    Res <= '1';
    wait;

end process;
end Behavioral;
```

Timing Diagram for Program Counter



Register Bank

- Register bank is used to store values.
- It has eight 4 – bit registers and a 3 – to – 8 decoder.
- Decoder is used to enable the correct register.
- We used D – Flip Flops with a enable input to create a register.

Source Code for D – Flip Flop

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

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

architecture Behavioral of D_FF is

begin
    process (Clk) begin
        if (rising_edge(Clk)) then

                if Res = '1' then
                    Q<='0';
                    Qbar<='1';
                else if En = '1' then
                    Q<=D;
                    Qbar <= not D;
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

Source code for Register

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

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

architecture Behavioral of Reg is

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

begin

 D_FF0 : D_FF
 port map (
 D => D(0),
 Res => Res,
 En => En,
 Clk => Clk,
 Q => Q(0));
D_FF1 : D_FF
 port map (
```

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

 D_FF2 : D_FF
 port map (
 D => D(2),
 Res => Res,
 En => En,
 Clk => Clk,
 Q => Q(2));

 D_FF3 : D_FF
 port map (
 D => D(3),
 Res => Res,
 En => En,
 Clk => Clk,
 Q => Q(3));

end Behavioral;
```

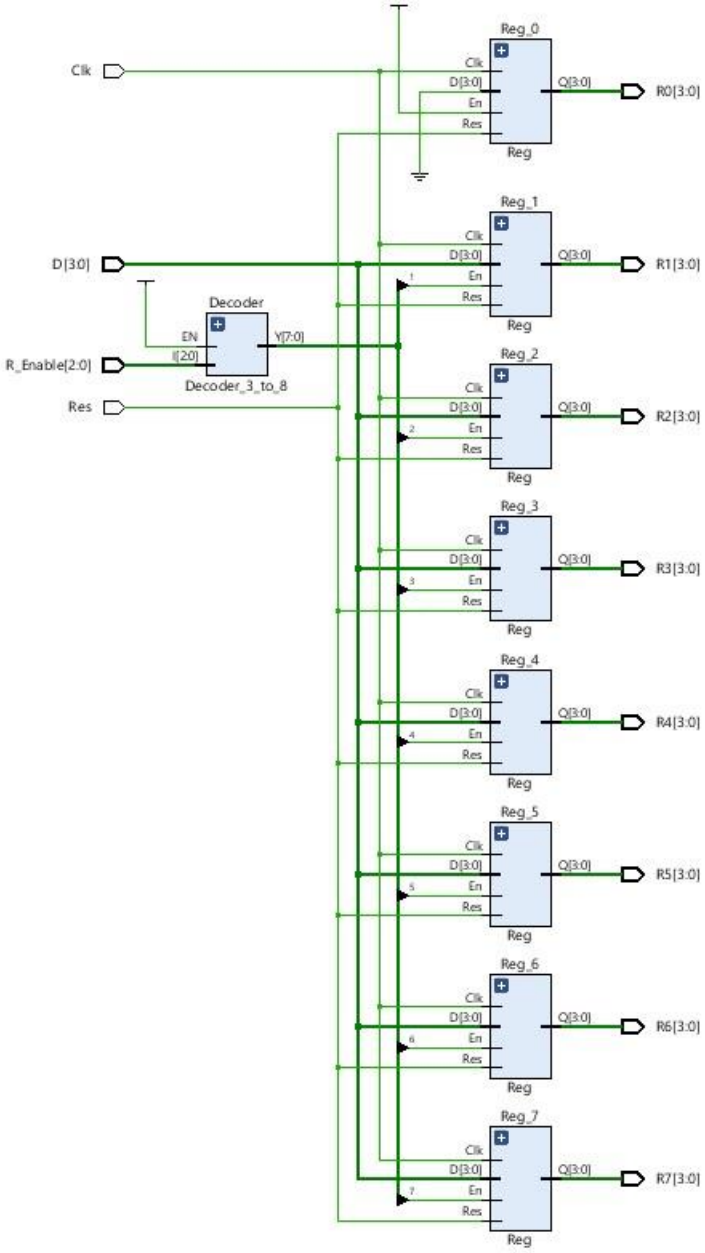
Source Code for Register Bank

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

entity Register_Bank is
    Port ( Clk : in STD_LOGIC;
           R_Enable : in STD_LOGIC_VECTOR (2 downto 0);
           Res : in STD_LOGIC;
           D : in STD_LOGIC_VECTOR (3 downto 0);
           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 Reg
port(
    D : in STD_LOGIC_VECTOR (3 downto 0);
    En : in STD_LOGIC;
    Res : in STD_LOGIC;
    Clk : in STD_LOGIC;
    Q : out STD_LOGIC_VECTOR (3 downto 0));
end component;
component Decoder_3_to_8
Port (
    I : in STD_LOGIC_VECTOR (2 downto 0);
    EN : in STD_LOGIC;
    Y : out STD_LOGIC_VECTOR (7 downto 0));
end component;

signal En_Reg : std_logic_vector (7 downto 0);
```

```
begin
Decoder : Decoder_3_to_8
port map (
    I => R_Enable,
    En => '1',
    Y => En_Reg);

Reg_0 : Reg
port map (
    D => "0000",
    En => '1',
    Res => Res,
    Clk => Clk,
    Q => R0);

Reg_1 : Reg
port map (
    D => D,
    En => En_Reg(1),
    Res => Res,
    Clk => Clk,
    Q => R1);
Reg_2 : Reg
port map (
    D => D,
    En => En_Reg(2),
    Res => Res,
    Clk => Clk,
    Q => R2);

Reg_3 : Reg
port map (
    D => D,
    En => En_Reg(3),
    Res => Res,
```

```
    Clk => Clk,
    Q => R3);

Reg_4 : Reg
port map (
    D => D,
    En => En_Reg(4),
    Res => Res,
    Clk => Clk,
    Q => R4);

Reg_5 : Reg
port map (
    D => D,
    En => En_Reg(5),
    Res => Res,
    Clk => Clk,
    Q => R5);

Reg_6 : Reg
port map (
    D => D,
    En => En_Reg(6),
    Res => Res,
    Clk => Clk,
    Q => R6);

Reg_7 : Reg
port map (
    D => D,
    En => En_Reg(7),
    Res => Res,
    Clk => Clk,
    Q => R7);

end Behavioral;
```

Elaborated design view (RTL schematic diagram) for Register Bank



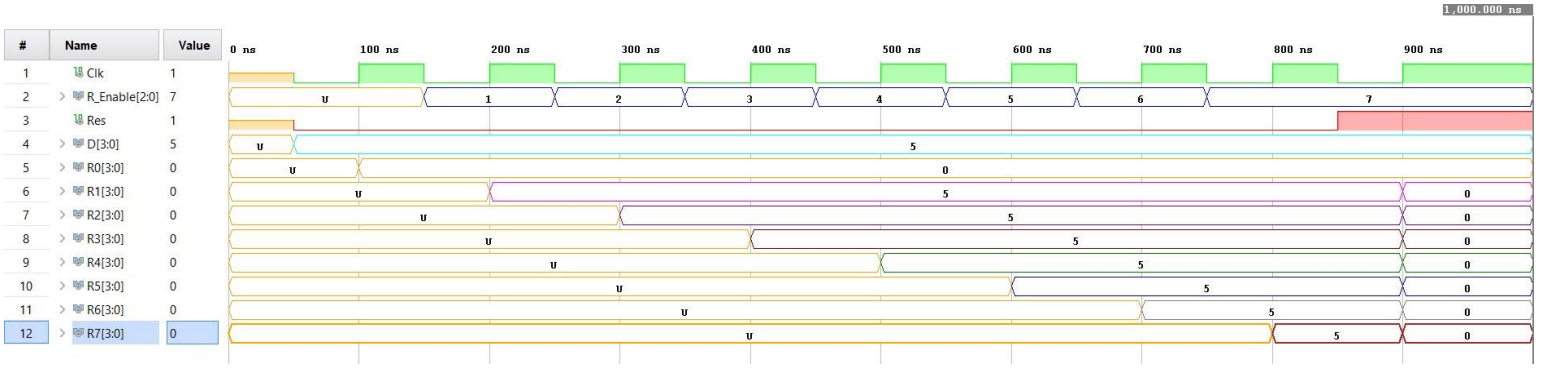
Simulation Code for Register Bank

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

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

architecture Behavioral of Register_Bank_TB is

component Register_Bank
Port (
       Clk : in STD_LOGIC;
       R_Enable : in STD_LOGIC_VECTOR (2 downto 0);
       Res : in STD_LOGIC;
       D : in STD_LOGIC_VECTOR (3 downto 0);
       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 Clk : STD_LOGIC;
signal R_Enable : STD_LOGIC_VECTOR (2 downto 0);
signal Res : STD_LOGIC;
signal D : STD_LOGIC_VECTOR (3 downto 0);
signal R0, R1, R2, R3, R4, R5, R6, R7 : STD_LOGIC_VECTOR (3 downto 0);
begin

UUT : Register_Bank
port map (
```

```
    Clk => Clk,
    R_Enable => R_Enable,
    Res => Res,
    D => D,
    R0 => R0,
    R1 => R1,
    R2 => R2,
    R3 => R3,
    R4 => R4,
    R5 => R5,
    R6 => R6,
    R7 => R7 );

process
begin
    wait for 50 ns;
    D <= "0101";
    Res <= '0';
    Clk <= '0';
    wait for 50 ns;
    Clk <= '1';
    wait for 50 ns;
    R_Enable <= "001";
    Clk <= '0';
    wait for 50 ns;
    Clk <= '1';
    wait for 50 ns;
    R_Enable <= "010";
    Clk <= '0';
    wait for 50 ns;
    Clk <= '1';
    wait for 50 ns;
    R_Enable <= "011";
    Clk <= '0';
    wait for 50 ns;
    Clk <= '1';
```

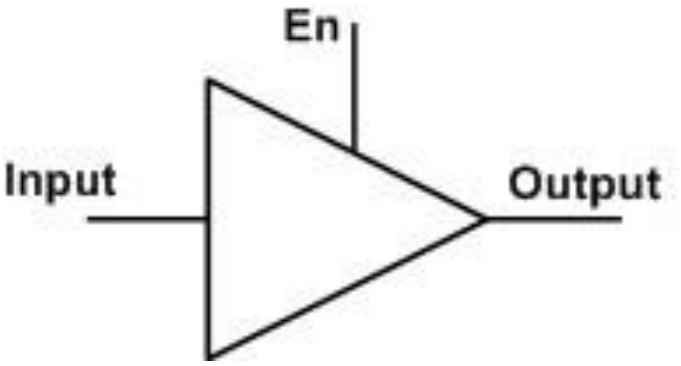
```
    wait for 50 ns;
    R_Enable <= "100";
    Clk <= '0';
    wait for 50 ns;
    Clk <= '1';
    wait for 50 ns;
    R_Enable <= "101";
    Clk <= '0';wait for 50 ns;
    Clk <= '1';
    wait for 50 ns;
    R_Enable <= "110";
    Clk <= '0';wait for 50 ns;
    Clk <= '1';
    wait for 50 ns;
    R_Enable <= "111";
    Clk <= '0';
    wait for 50 ns;
    Clk <= '1';
    wait for 50 ns;
    Res <= '1';
    Clk <= '0';
    wait for 50 ns;
    Clk <= '1';
    wait;

end process;
end Behavioral;
```

Timing Diagram for Register Bank



3-bit Tri-state-buffer

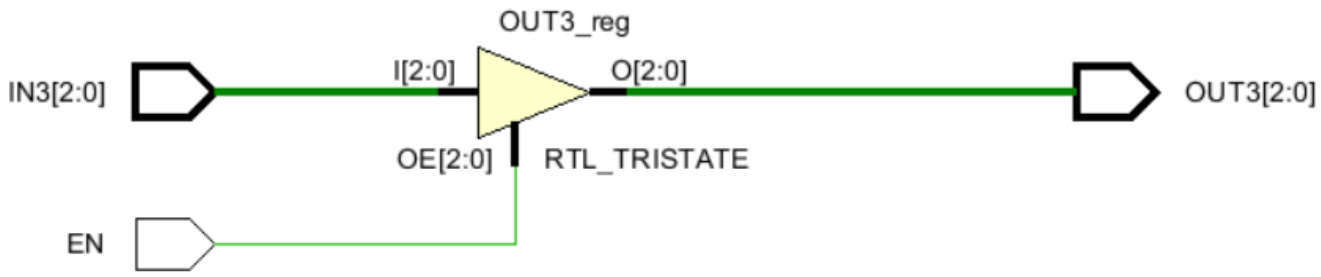
- We can use tri state buffers to build the multiplexers.



VHDL Code for 3-bit Tri-state-buffer

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

entity tri_state_buffer_3bit is

Port (      IN3 : in STD_LOGIC_VECTOR (2 downto 0);
            OUT3 : out STD_LOGIC_VECTOR (2 downto 0);
            EN : in STD_LOGIC);
end tri_state_buffer_3bit;

architecture Behavioral of tri_state_buffer_3bit is

begin
OUT3<=IN3 WHEN (EN='1') else "ZZZ";

end Behavioral;
```

Elaborated design view for 3-bit Tri-state-buffer

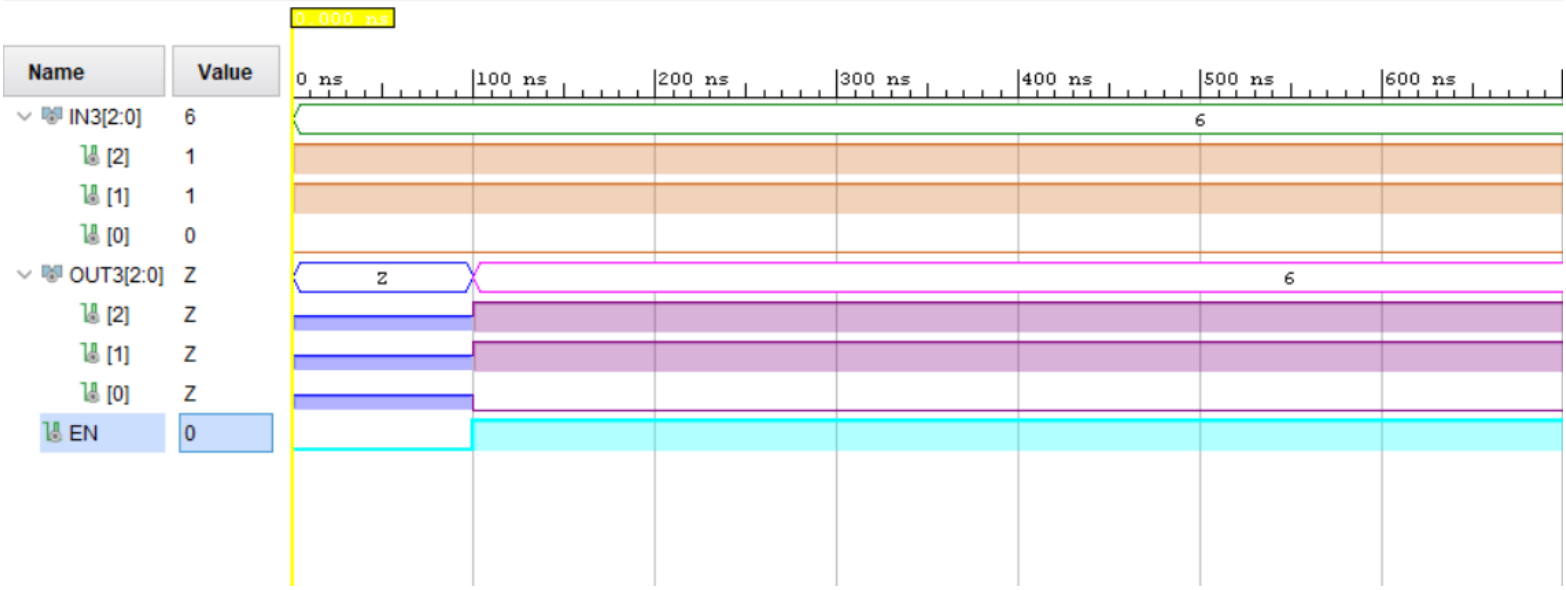


Simulation code for 3-bit Tri-state-buffer

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

entity tri_state_buffer_3bit_TB is
--  Port ( );
end tri_state_buffer_3bit_TB;

architecture Behavioral of tri_state_buffer_3bit_TB is
component tri_state_buffer_3bit
port(   IN3 : in STD_LOGIC_VECTOR (2 downto 0);
        OUT3 : out STD_LOGIC_VECTOR (2 downto 0);
        EN : in STD_LOGIC);
end component;
signal IN3 : std_logic_vector(2 downto 0);
signal OUT3 : std_logic_vector(2 downto 0);
signal EN : std_logic;
begin
UUT: tri_state_buffer_3bit
port map( IN3 => IN3,
            OUT3=> OUT3,
            EN=>EN);
```

```
process
begin

IN3<= "110";
EN<='0';

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

end process;
end Behavioral;
```

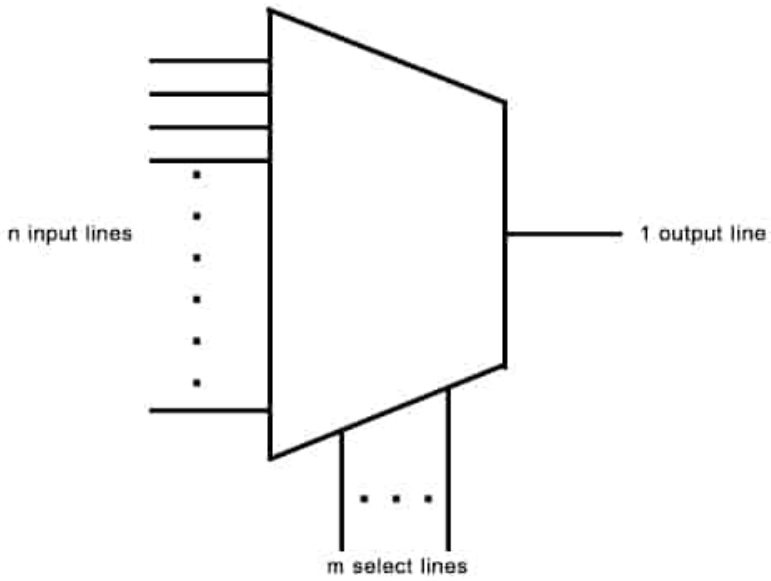
Timing Diagram for 3-bit Tri-state-buffer



- **Similarly, we created a 4-bit tri-state-buffer with the difference of it only being 4-bits.**

Multiplexers

- We can use multiplexers to select only a single input from a given set of inputs by giving specific selection instructions.



2-way 3-bit Multiplexer

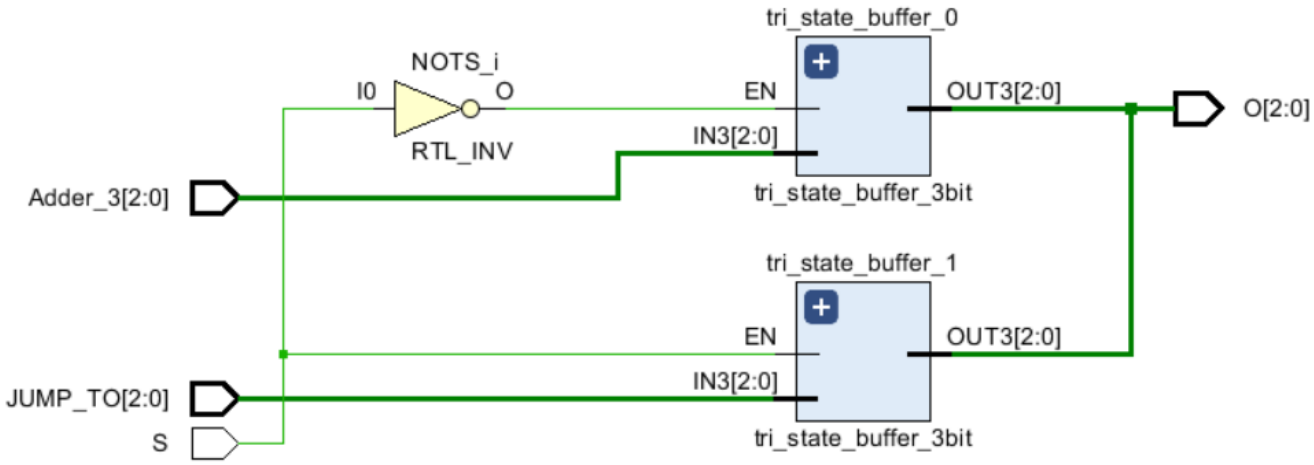
- Used to select one input from the adder and jump address with the use of jump flag.

VHDL Code for 2-way 3-bit multiplexer

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

entity Mux_2way_3bit is
    Port ( Adder_3 : in STD_LOGIC_VECTOR (2 downto 0);
           JUMP_TO : in STD_LOGIC_VECTOR (2 downto 0);
           S : in STD_LOGIC;
           O : out std_logic_vector(2 downto 0));
end Mux_2way_3bit;

architecture Behavioral of Mux_2way_3bit is
component tri_state_buffer_3bit
    port ( IN3 : in STD_LOGIC_VECTOR (2 downto 0);
           OUT3 : out STD_LOGIC_VECTOR (2 downto 0);
```

```
           EN : in STD_LOGIC);
end component;

signal NOTS: std_logic;
begin
tri_state_buffer_0 :tri_state_buffer_3bit
port map(   IN3 => Adder_3,
            OUT3 => O,
            EN => NOTS);
tri_state_buffer_1: tri_state_buffer_3bit
port map(   IN3 => JUMP_TO,
            OUT3 =>O,
            EN =>S);

NOTS <= NOT S;

end Behavioral;
```

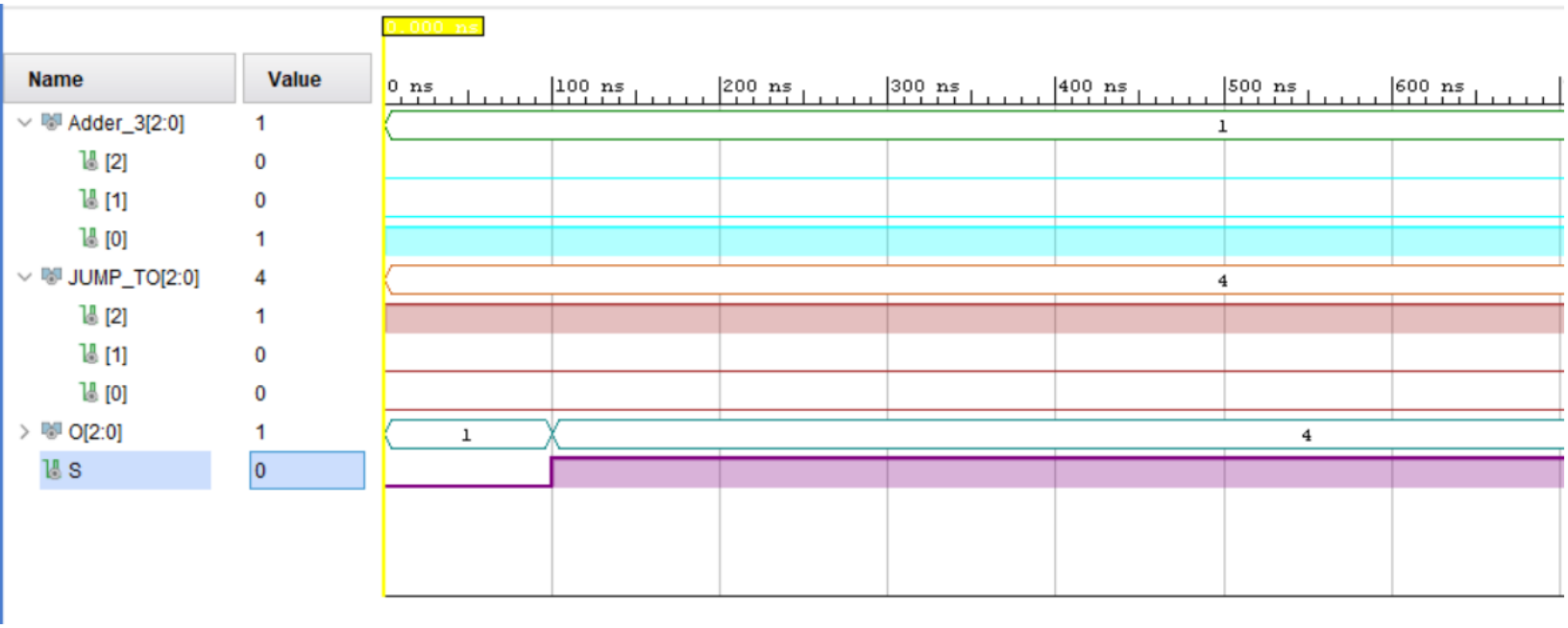
Elaborated design view for 2-way 3-bit multiplexer



Simulation code for 2-way 3-bit multiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Mux_2way_3bit_TB is
--  Port ( );
end Mux_2way_3bit_TB;
architecture Behavioral of Mux_2way_3bit_TB is
component Mux_2way_3bit
    port(   Adder_3 : in STD_LOGIC_VECTOR (2 downto 0);
            JUMP_TO : in STD_LOGIC_VECTOR (2 downto 0);
            O : out STD_LOGIC_VECTOR (2 downto 0);
            S : in STD_LOGIC);
end component;
signal Adder_3,JUMP_TO : std_logic_vector(2 downto 0);
signal O : std_logic_vector(2 downto 0);
signal S : std_logic;
begin
UUT: Mux_2way_3bit
port map(
            Adder_3(2 downto 0)=>Adder_3(2 downto 0),
            JUMP_TO(2 downto 0)=>JUMP_TO(2 downto 0),
            O(2 downto 0)=> O(2 downto 0),
            S=> S);
process
begin
Adder_3<="001";
JUMP_TO<="100";
S<='0';

WAIT FOR 100ns;
S<='1';
WAIT;

end process;
end Behavioral;
```

Timing Diagram for 2-way 3-bit multiplexer



2-way 4-bit Multiplexer

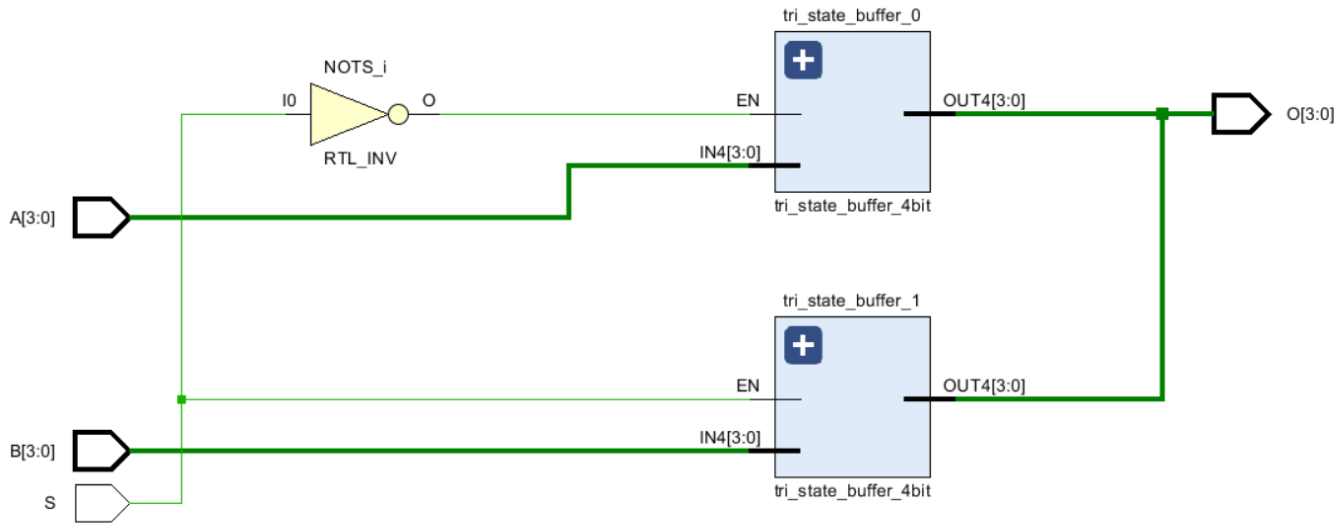
- Used to select one from the output of Add/Sub unit and immediate value with the use of load select

VHDL Code 2-way 4-bit Multiplexer

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

entity Mux_2way_4bit is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           O : out STD_LOGIC_VECTOR (3 downto 0);
           S : in STD_LOGIC);
end Mux_2way_4bit;

architecture Behavioral of Mux_2way_4bit is
component tri_state_buffer_4bit
    port (  IN4 : in STD_LOGIC_VECTOR (3 downto 0);
           OUT4 : out STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC);
```

```
end component;
signal NOTS: std_logic;
begin
tri_state_buffer_0:tri_state_buffer_4bit
port map(   IN4=> A,
            OUT4 => O,
            EN=> NOTS);
tri_state_buffer_1: tri_state_buffer_4bit
port map(   IN4=> B,
            OUT4 =>O,
            EN=>S);

NOTS <= NOT S;

end Behavioral;
```

Elaborated design view 2-way 4-bit Multiplexer



Simulation code 2-way 4-bit Multiplexer

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

entity Mux_2way_4bit_TB is
--  Port ( );
end Mux_2way_4bit_TB;

architecture Behavioral of Mux_2way_4bit_TB is
component Mux_2way_4bit
    port(   A : in STD_LOGIC_VECTOR (3 downto 0);
            B : in STD_LOGIC_VECTOR (3 downto 0);
            O : out STD_LOGIC_VECTOR (3 downto 0);
            S : in STD_LOGIC);
end component;
signal A,B : std_logic_vector(3 downto 0);
signal O : std_logic_vector(3 downto 0);
signal S : std_logic;
begin
UUT: Mux_2way_4bit
port map(
            A(3 downto 0)=>A(3 downto 0),
            B(3 downto 0)=>B(3 downto 0),
            O(3 downto 0)=> O(3 downto 0),
            S=> S);
process
begin
A<="0011";
B<="1000";
S<='0';
WAIT FOR 100ns;
S<='1';
WAIT;
end process;
end Behavioral;
```

Timing Diagram 2-way 4-bit Multiplexer

8-way 4-bit Multiplexer

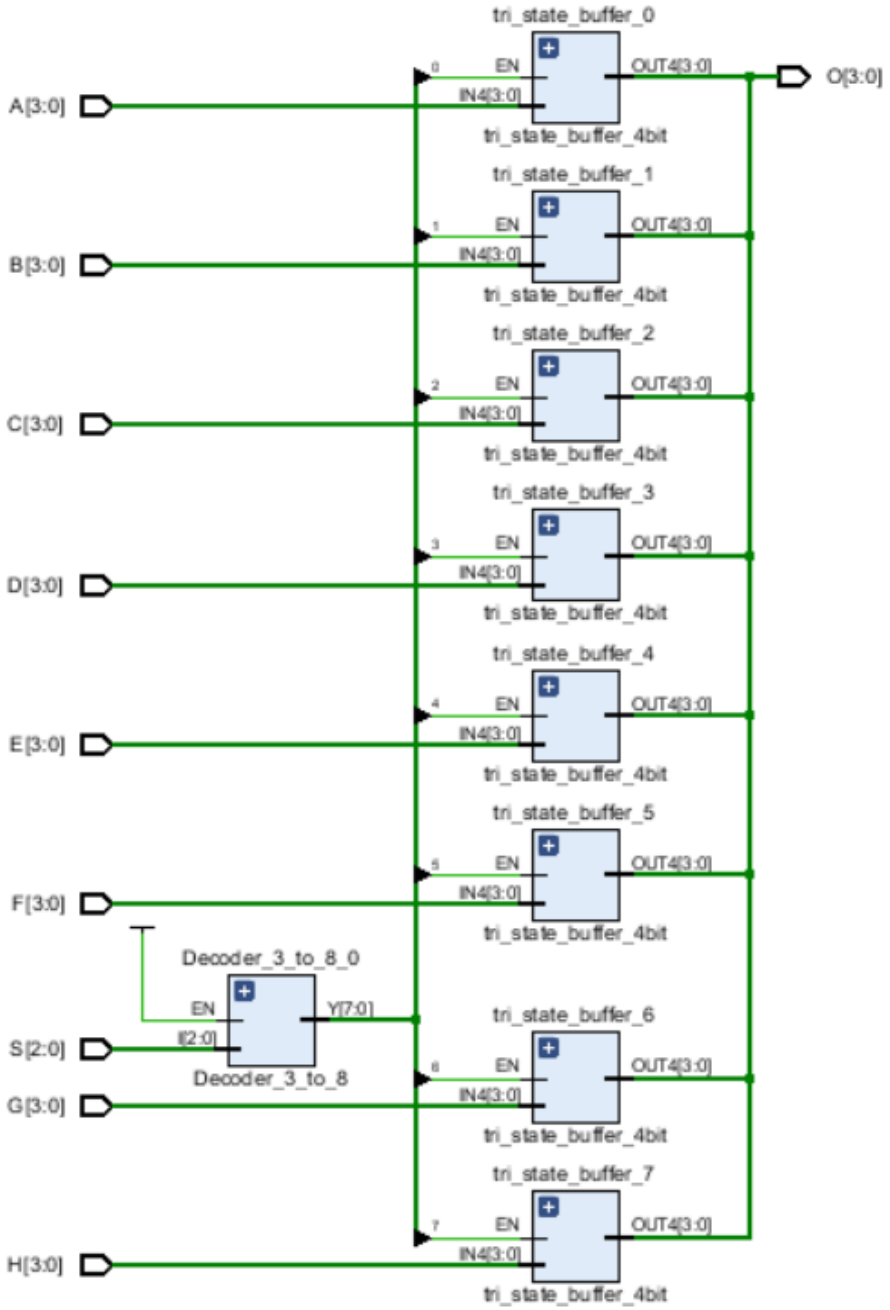
- Used to select one register from the 8 registers with use of register select line

VHDL Code for 8-way 4-bit Multiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Mux_8way_4bit is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC_VECTOR (3 downto 0);
           D : in STD_LOGIC_VECTOR (3 downto 0);
           E : in STD_LOGIC_VECTOR (3 downto 0);
           F : in STD_LOGIC_VECTOR (3 downto 0);
           G : in STD_LOGIC_VECTOR (3 downto 0);
           H : in STD_LOGIC_VECTOR (3 downto 0);
           O : out STD_LOGIC_VECTOR (3 downto 0);
           S : in STD_LOGIC_VECTOR (2 downto 0));
end Mux_8way_4bit;

architecture Behavioral of Mux_8way_4bit is
Component Decoder_3_to_8
    port(   I : in STD_LOGIC_VECTOR (2 downto 0);
            EN : in STD_LOGIC;
            Y : out STD_LOGIC_VECTOR (7 downto 0));
 end component;

Signal Y0 : std_logic_vector(7 downto 0);

component tri_state_buffer_4bit
    port(  IN4 : in STD_LOGIC_VECTOR (3 downto 0);
           OUT4 : out STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC);
end component;
begin
Decoder_3_to_8_0 : Decoder_3_to_8
    port map (  I => S,
```

```
            EN => '1',
            Y => Y0);
tri_state_buffer_0 : tri_state_buffer_4bit
    port map(   IN4 => A,
                EN=> Y0(0),
                OUT4 => O);
tri_state_buffer_1 : tri_state_buffer_4bit
    port map(   IN4 => B(3 downto 0),
                EN=> Y0(1),
                OUT4 => O(3 downto 0));
tri_state_buffer_2 : tri_state_buffer_4bit
    port map(   IN4 => C(3 downto 0),
                EN=> Y0(2),
                OUT4 => O(3 downto 0));
tri_state_buffer_3 : tri_state_buffer_4bit
    port map(   IN4 => D(3 downto 0),
                EN=> Y0(3),
                OUT4 => O(3 downto 0));
tri_state_buffer_4 : tri_state_buffer_4bit
    port map(   IN4 => E(3 downto 0),
                EN=> Y0(4),
                OUT4 => O(3 downto 0));
tri_state_buffer_5 : tri_state_buffer_4bit
    port map(   IN4 => F(3 downto 0),
                EN=> Y0(5),
                OUT4 => O(3 downto 0));
tri_state_buffer_6 : tri_state_buffer_4bit
    port map(   IN4 => G(3 downto 0),
                EN=> Y0(6),
                OUT4 => O(3 downto 0));
tri_state_buffer_7 : tri_state_buffer_4bit
    port map(   IN4 => H(3 downto 0),
                EN=> Y0(7),
                OUT4 => O(3 downto 0));
end Behavioral;
```

Elaborated design view for 8-way 4-bit Multiplexer

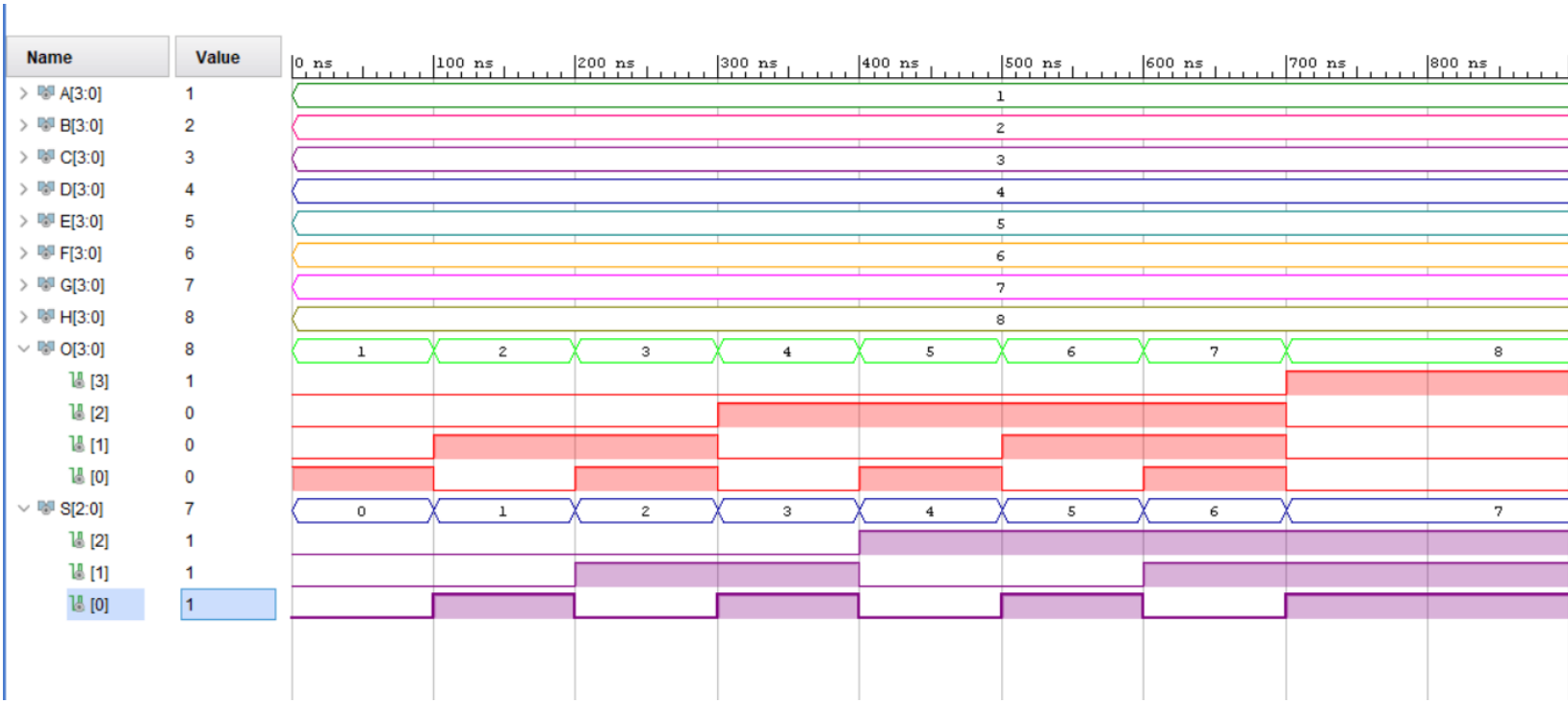


Simulation code for 8-way 4-bit Multiplexer

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

entity Mux_8way_4bit_TB is
--  Port ( );
end Mux_8way_4bit_TB;

architecture Behavioral of Mux_8way_4bit_TB is
component Mux_8way_4bit
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           C : in STD_LOGIC_VECTOR (3 downto 0);
           D : in STD_LOGIC_VECTOR (3 downto 0);
           E : in STD_LOGIC_VECTOR (3 downto 0);
           F : in STD_LOGIC_VECTOR (3 downto 0);
           G : in STD_LOGIC_VECTOR (3 downto 0);
           H : in STD_LOGIC_VECTOR (3 downto 0);
           O : out STD_LOGIC_VECTOR (3 downto 0);
           S : in STD_LOGIC_VECTOR (2 downto 0));
end component;
signal A,B,C,D,E,F,G,H : std_logic_vector(3 downto 0);
signal O: std_logic_vector(3 downto 0);
signal S : std_logic_vector(2 downto 0);

begin
UUT : Mux_8way_4bit
port map(   A(3 downto 0)=> A(3 downto 0),
            B(3 downto 0)=> B(3 downto 0),
            C(3 downto 0)=> C(3 downto 0),
            D(3 downto 0)=> D(3 downto 0),
            E(3 downto 0)=> E(3 downto 0),
            F(3 downto 0)=> F(3 downto 0),
            G(3 downto 0)=> G(3 downto 0),
            H(3 downto 0)=> H(3 downto 0),
```

```
            O(3 downto 0)=> O(3 downto 0),
            S(2 downto 0)=> S(2 downto 0));

process
begin

A<="0001";
B<="0010";
C<="0011";
D<="0100";
E<="0101";
F<="0110";
G<="0111";
H<="1000";
S<="000";

WAIT FOR 100ns;
S<="001";
WAIT FOR 100ns;
S<="010";
WAIT FOR 100ns;
S<="011";
WAIT FOR 100ns;
S<="100";
WAIT FOR 100ns;
S<="101";
WAIT FOR 100ns;
S<="110";
WAIT FOR 100ns;
S<="111";
WAIT;

end process;
end Behavioral;
```

Timing Diagram for 8-way 4-bit Multiplexer



Look Up Table

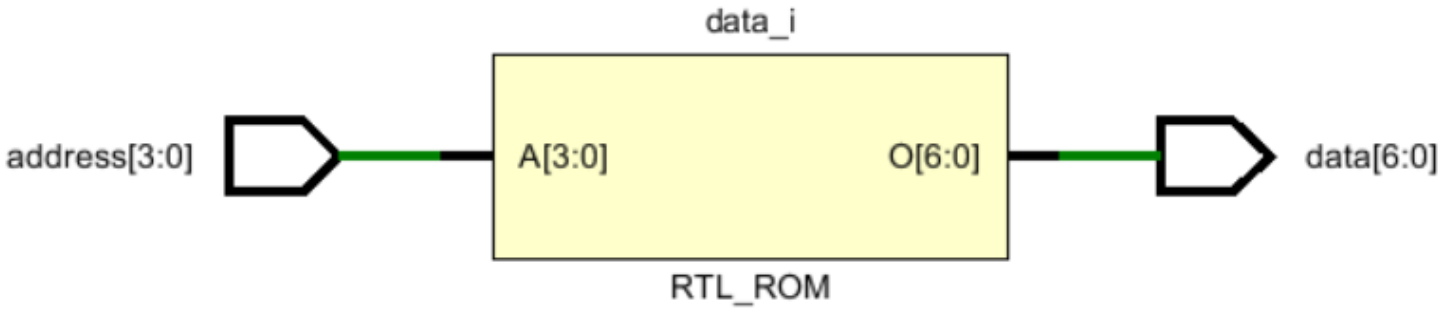
- Used to create the binary values corresponding to the 7-segment display.

Vhdl Code for Look Up Table

```
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;
```

Elaborated design view for Look Up Table

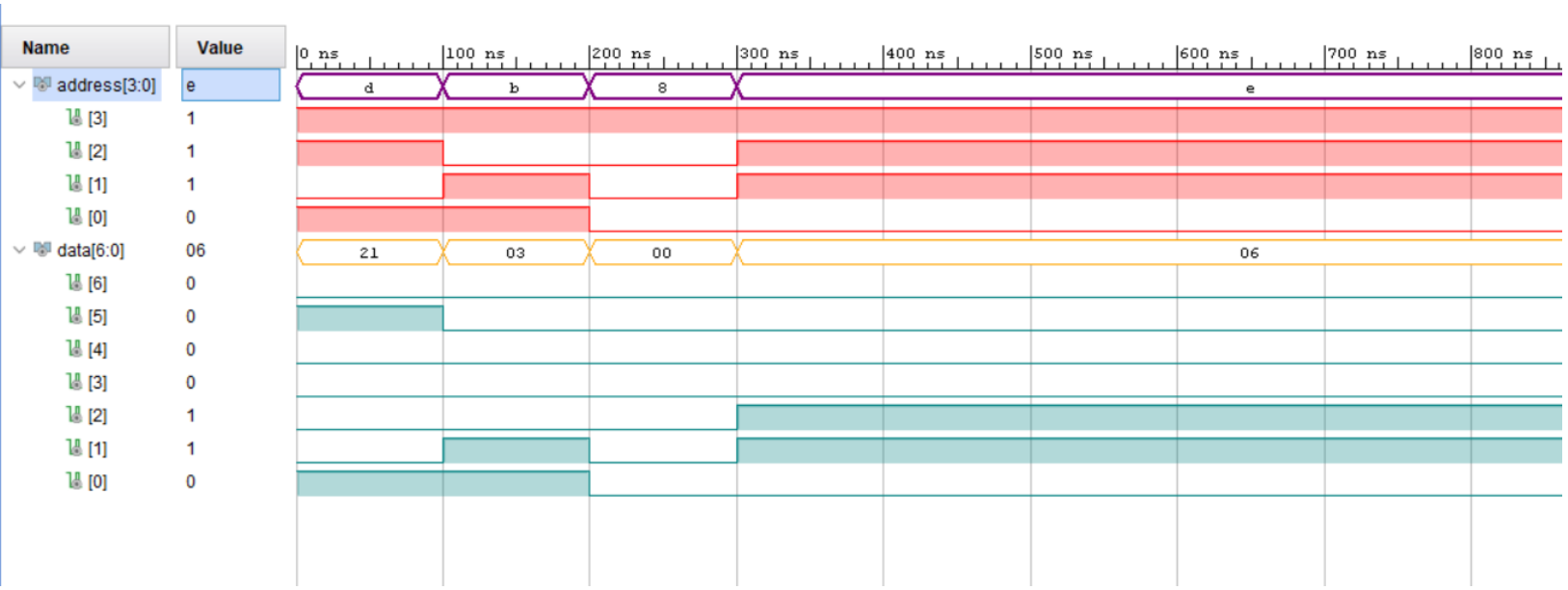


Simulation code for Look Up Table

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

entity LUT_sim is
end LUT_sim;

architecture Behavioral of LUT_sim 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);
signal data : STD_LOGIC_VECTOR (6 downto 0);

begin
UUT: LUT_16_7 PORT MAP(
address=> address,
```

```
data=>data
);

process
begin

-- index 190653
-- 1110 1000 1011 1101
address <= "1101";
wait for 100ns;

address <= "1011";
wait for 100ns;

address <= "1000";
wait for 100ns;

address <= "1110";
wait;

end process;



end Behavioral;
```

Timing Diagram for Look Up Table



Nano Processor

- All above mention components are used in Nano processor in hierarchical order.
- Inputs – Reset button, Clock.
- Outputs – Zero flag and Overflow flag LEDs, 7-Segment Display, 4 LEDs for register 7th value.

Source Code for Nano Processor

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

entity NanoProcessor is
    Port (
           Reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC;
           Reg_7_LED : out STD_LOGIC_VECTOR (3 downto 0);
           S_7seg : out STD_LOGIC_VECTOR (6 downto 0) );

end NanoProcessor;

architecture Behavioral of NanoProcessor is

component Instruction_Decoder

    Port (
        INSTUC : in STD_LOGIC_VECTOR (11 downto 0);
        J_CHK : in STD_LOGIC;
        REG_EN : out STD_LOGIC_VECTOR (2 downto 0);
        REG_SEL_A : out STD_LOGIC_VECTOR (2 downto 0);
        REG_SEL_B : out STD_LOGIC_VECTOR (2 downto 0);
        LOAD_SEL : out STD_LOGIC;
        IM_VAL : out STD_LOGIC_VECTOR (3 downto 0);
        CTRL : out STD_LOGIC;
        J_FLAG : out STD_LOGIC;
        J_ADDR : out STD_LOGIC_VECTOR (2 downto 0));
```

```
end component;

component Add_Sub_Unit
    port (
           A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           ctrl : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC);
end component;

component Progrm_Rom
    Port (
           SEL : in STD_LOGIC_VECTOR (2 downto 0);
           INSTUC : out STD_LOGIC_VECTOR (11 downto 0));
end component;


component Bit_3_Adder
     Port (
            A : in STD_LOGIC_VECTOR(2 downto 0);
            B : in STD_LOGIC_VECTOR(2 downto 0);
            C_in : in STD_LOGIC;
            S : out STD_LOGIC_VECTOR(2 downto 0);
            C_out : out STD_LOGIC
);
end component;

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

```
component Register_Bank
    Port (    Clk : in STD_LOGIC;
              R_Enable : in STD_LOGIC_VECTOR (2 downto 0);
              Res : in STD_LOGIC;
              D : in STD_LOGIC_VECTOR (3 downto 0);
              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;

component Mux_2way_4bit
    Port (  A : in STD_LOGIC_VECTOR (3 downto 0);
            B : in STD_LOGIC_VECTOR (3 downto 0);
            O : out STD_LOGIC_VECTOR (3 downto 0);
            S : in STD_LOGIC);
end component;

component Mux_2way_3bit
    Port (
            Adder_3 : in STD_LOGIC_VECTOR (2 downto 0);
            JUMP_TO : in STD_LOGIC_VECTOR (2 downto 0);
            S : in STD_LOGIC;
            O : out std_logic_vector(2 downto 0));
end component;

component Mux_8way_4bit
    Port (  A : in STD_LOGIC_VECTOR (3 downto 0);
            B : in STD_LOGIC_VECTOR (3 downto 0);
            C : in STD_LOGIC_VECTOR (3 downto 0);
            D : in STD_LOGIC_VECTOR (3 downto 0);
            E : in STD_LOGIC_VECTOR (3 downto 0);
```

```
            F : in STD_LOGIC_VECTOR (3 downto 0);
            G : in STD_LOGIC_VECTOR (3 downto 0);
            H : in STD_LOGIC_VECTOR (3 downto 0);
            O : out STD_LOGIC_VECTOR (3 downto 0);
            S : in STD_LOGIC_VECTOR (2 downto 0));
end component;

--import values for seven segement display
component LUT_16_7
    Port (
        address : in STD_LOGIC_VECTOR (3 downto 0);
        values : out STD_LOGIC_VECTOR (6 downto 0));
end component;

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

signal Slw_Clk : std_logic;
signal R0_out,R1_out,R2_out,R3_out,R4_out,R5_out,R6_out,R7_out :
std_logic_vector (3 downto 0); --from register bank to mux
signal CTRL:std_logic; --from instruction decoder to Add sub unit
signal Reg_En : std_logic_vector (2 downto 0); --form instruction
decoder to register bank
signal Instruction :std_logic_vector(11 downto 0);--from program rom to
instruction decoder
signal Mem_Sel :std_logic_vector (2 downto 0); --from p_counter to ROM
and adder

signal C_out : std_logic; --for carry out of 3-bit adder
signal next_Address : std_logic_vector (2 downto 0); --Output of 3 - bit
adder
signal load_select : std_logic; --from instruction decoder to 2way 4bit
mux
signal data : std_logic_vector (3 downto 0); --from 2 way 4 bit mux to
register bank
```

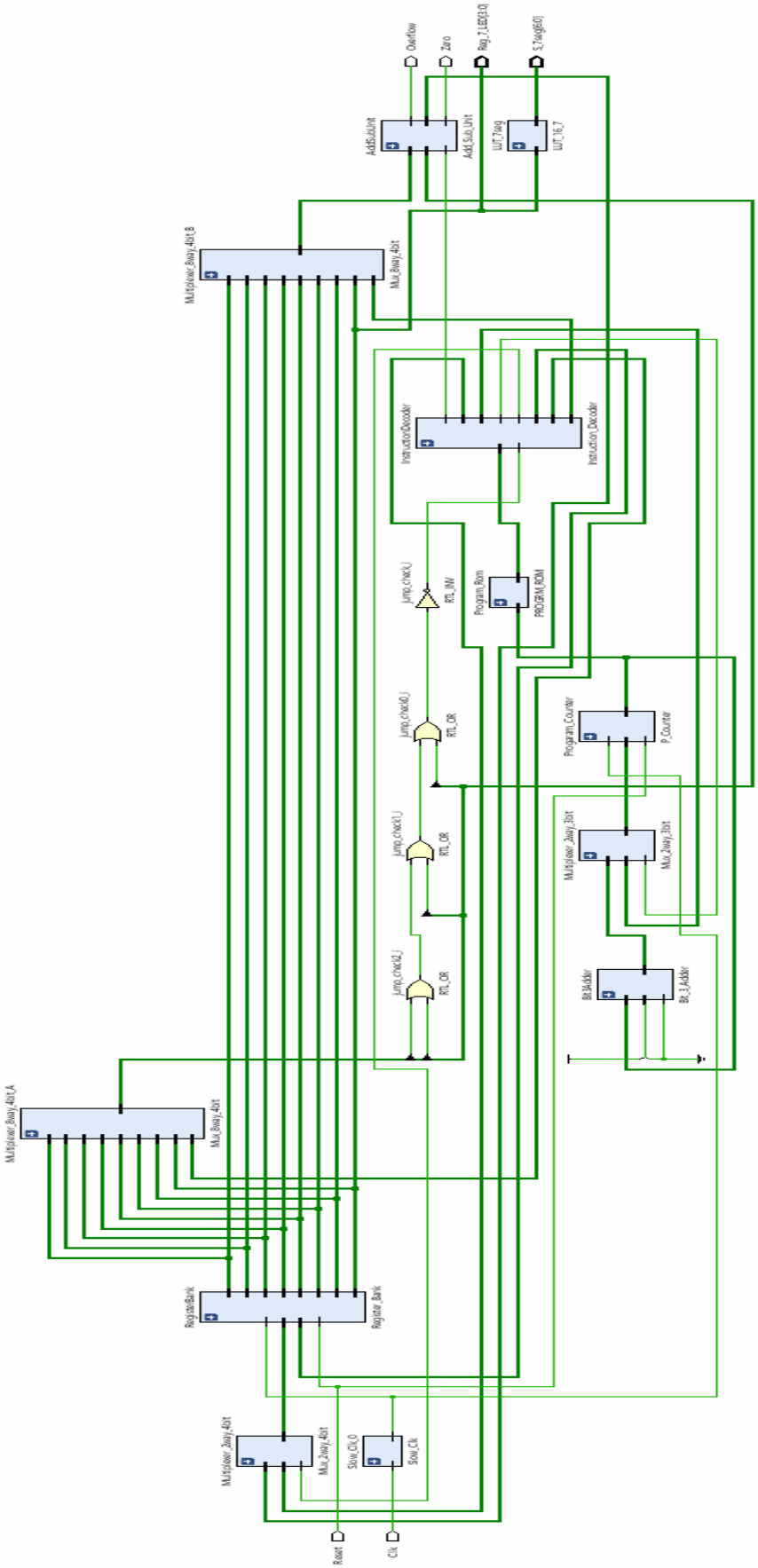
```
signal Cal_value : std_logic_vector (3 downto 0); --from addSub unit to
mux
signal im_value : std_logic_vector (3 downto 0); --from instruction
decoder to mux
signal selected_val_A : std_logic_vector (3 downto 0); --from muxA to
add sub unit
signal selected_val_B : std_logic_vector (3 downto 0); --from muxB to
add sub unit
signal reg_sel_A : std_logic_vector (2 downto 0); --from instruction
decoder to muxA
signal reg_sel_B : std_logic_vector (2 downto 0); --from instruction
decoder to muxB
signal Address_To_Jump:std_logic_vector(2 downto 0); --from instruction
decoder to 2 way 3 bit mux
signal Jump:std_logic;--form instruction decoder to 2 way 3 bit mux
signal jump_check : std_logic; --
signal pc_in:std_logic_vector(2 downto 0);--from 2 way 3 bit mux to
programme counter



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

    InstructionDecoder:Instruction_Decoder
        port map (
            INSTUC => Instruction,
            J_CHK => jump_check,  --: in STD_LOGIC;
            REG_EN => Reg_En,
            REG_SEL_A => reg_sel_A,--: out STD_LOGIC_VECTOR (2 downto
0);
            REG_SEL_B => reg_sel_B,--: out STD_LOGIC_VECTOR (2 downto
0);
            LOAD_SEL => load_select, --: out STD_LOGIC;
            IM_VAL => im_value, --: out STD_LOGIC_VECTOR (3 downto 0);
            CTRL => CTRL, --: out STD_LOGIC;
```

```
            J_FLAG=>Jump, --: out STD_LOGIC;
            J_ADDR=>Address_To_Jump --: out STD_LOGIC_VECTOR (2 downto
0));

    Multiplexer_2way_4bit: Mux_2way_4bit
        port map (
            A => Cal_value,
            B => im_value,
            O => data,
            S => load_select);

    Program_Rom:Progrm_Rom
        Port map(
            SEL =>Mem_Sel, --: in STD_LOGIC_VECTOR (2 downto 0);
            INSTUC =>Instruction);

    --bit 3 adder can be change but for get the adder implemnentation
use
    Bit3Adder : Bit_3_Adder
        port map (
              A =>Mem_Sel,
              B =>"001",
              C_in => '0', --no carry inputs for the Bit3Adder
              S => next_Address,
              C_out => C_out);

    Multiplexer_2way_3bit:Mux_2way_3bit
        port map(
            Adder_3 =>next_Address,
            JUMP_TO =>Address_To_Jump,
            S =>Jump,
            O =>pc_in);

    Progaram_Counter : P_Counter
        port map (
              D => pc_in,
              Res => Reset,
```

```
            Clk => Slw_Clk,
            Q => Mem_Sel);

    RegisterBank: Register_Bank
        port map(
            Clk => Slw_Clk,
            R_Enable => Reg_En,
            Res => Reset,
            D => data,
            R0 => R0_out,
            R1 => R1_out,
            R2 => R2_out,
            R3 => R3_out,
            R4 => R4_out,
            R5 => R5_out,
            R6 => R6_out,
            R7 => R7_out);

    AddSubUnit: Add_Sub_Unit
        port map(
           A => selected_val_B,
           B => selected_val_A,
           ctrl => CTRL,
           S => Cal_value,
           Overflow =>Overflow,
           Zero =>Zero);

    Multiplexer_8way_4bit_A : Mux_8way_4bit
        port map(
            A => R0_out,
            B => R1_out,
            C => R2_out,
            D => R3_out,
            E => R4_out,
            F => R5_out,
            G => R6_out,
```

```
            H => R7_out,
            O => selected_val_A,
            S => reg_sel_A);

    Multiplexer_8way_4bit_B : Mux_8way_4bit
        port map(
            A => R0_out,
            B => R1_out,
            C => R2_out,
            D => R3_out,
            E => R4_out,
            F => R5_out,
            G => R6_out,
            H => R7_out,
            O => selected_val_B,
            S => reg_sel_B);



    --sevent segment display
    LUT_7seg : LUT_16_7
        port map (
            address => R7_out,
            values =>  S_7seg);

    Reg_7_LED <= R7_out;
    jump_check <= not(selected_val_A(0) or selected_val_A(1) or
selected_val_A(2) or selected_val_A(3));



end Behavioral;
```

Elaborated design view (RTL schematic diagram) for Nano Processor

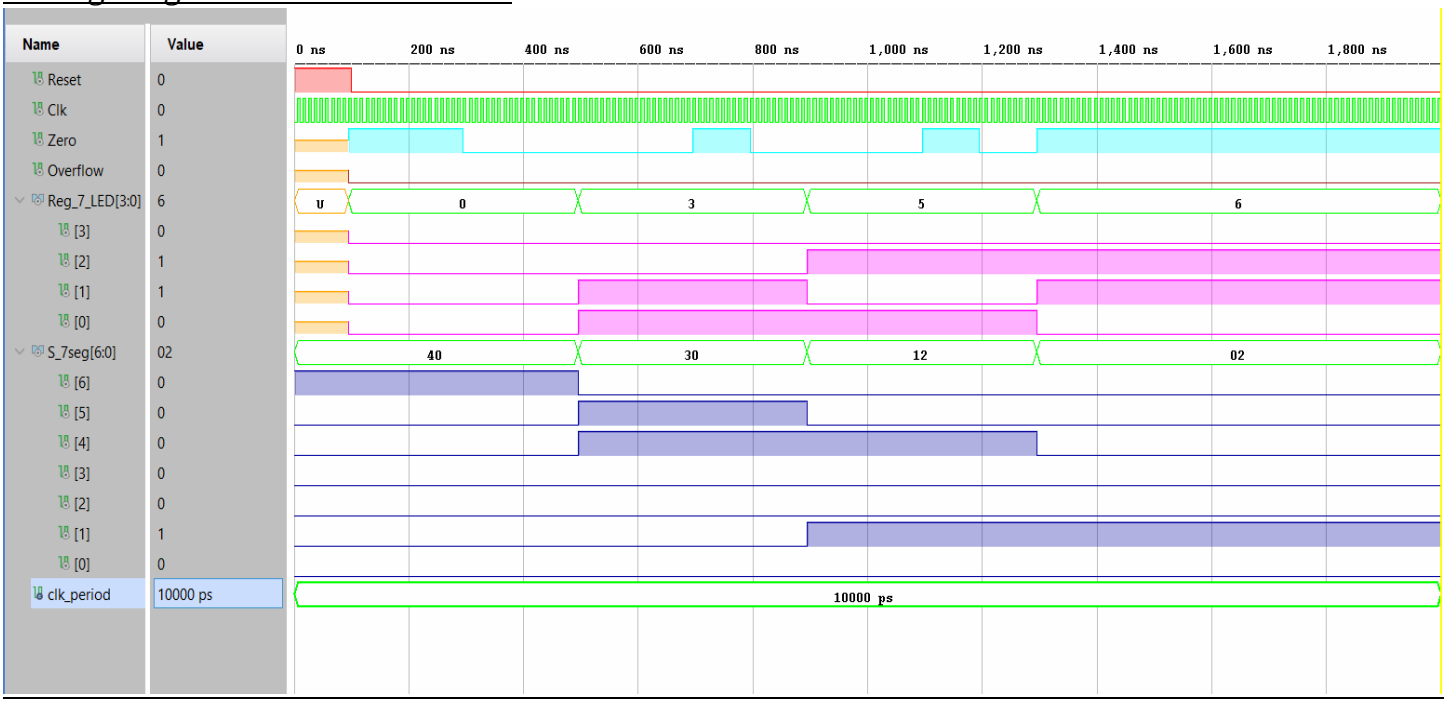


Simulation Code for Nano Processor

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

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

architecture Behavioral of Nanoprocessor_TB is

component NanoProcessor
    Port (
                Reset : in STD_LOGIC;
                Clk : in STD_LOGIC;
                Zero : out STD_LOGIC;
                Overflow : out STD_LOGIC;
                Reg_7_LED : out STD_LOGIC_VECTOR (3 downto 0);
                S_7seg : out STD_LOGIC_VECTOR (6 downto 0)
        );
end component;
constant clk_period:time:=10ns;
signal Reset,Clk,Zero,Overflow:std_logic;
signal Reg_7_LED:std_logic_vector(3 downto 0);
signal S_7seg:std_logic_vector(6 downto 0);

begin

UUT:NanoProcessor
    Port map(
                Reset =>Reset,
                Clk =>Clk,
                Zero =>Zero,
                Overflow =>Overflow,
                Reg_7_LED =>Reg_7_LED,
                S_7seg  =>S_7seg
```

```
        );

clk_sim:process
            begin
                Clk <= '0';
                wait for clk_period/2;

                Clk <= '1';
                wait for clk_period/2;
        end process;

sim:process
        begin
            Reset <='1';
            wait for 100ns ;
            Reset <='0';
            wait;

end process;
end Behavioral;
```

Timing Diagram for Nano Processor



Slow Clock

Source code for Slow Clock

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

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

architecture Behavioral of Slow_Clk is

SIGNAL count : integer := 1;
SIGNAL clk_status : std_logic := '0';
begin
process(Clk_in)
begin
if(rising_edge(Clk_in)) then
    count <= count+1;
if(count = 5) then
    clk_status <= not clk_status;
    Clk_out <= clk_status;
    count <= 1;
end if;
end if;
end process;

end Behavioral;
```

Xilinx Design Constraints(XDC) File

- Used LEDs for zero flag, overflow flag and 4 more LEDs to output register 7 value.
- Used a button for Reset.
- Used the Basys3 Master XDC file to take the clock of the Basys3 board.

```
## R7 values
set_property PACKAGE_PIN U16 [get_ports {Reg_7_LED[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Reg_7_LED[0]}]
set_property PACKAGE_PIN E19 [get_ports {Reg_7_LED[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Reg_7_LED[1]}]
set_property PACKAGE_PIN U19 [get_ports {Reg_7_LED[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Reg_7_LED[2]}]
set_property PACKAGE_PIN V19 [get_ports {Reg_7_LED[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Reg_7_LED[3]}]


## Zero flag
set_property PACKAGE_PIN P1 [get_ports {Zero}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Zero}]


## Overflow
set_property PACKAGE_PIN L1 [get_ports {Overflow}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Overflow}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {S_7seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {S_7seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {S_7seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {S_7seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {S_7seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {S_7seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {S_7seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {S_7seg[3]}]
```

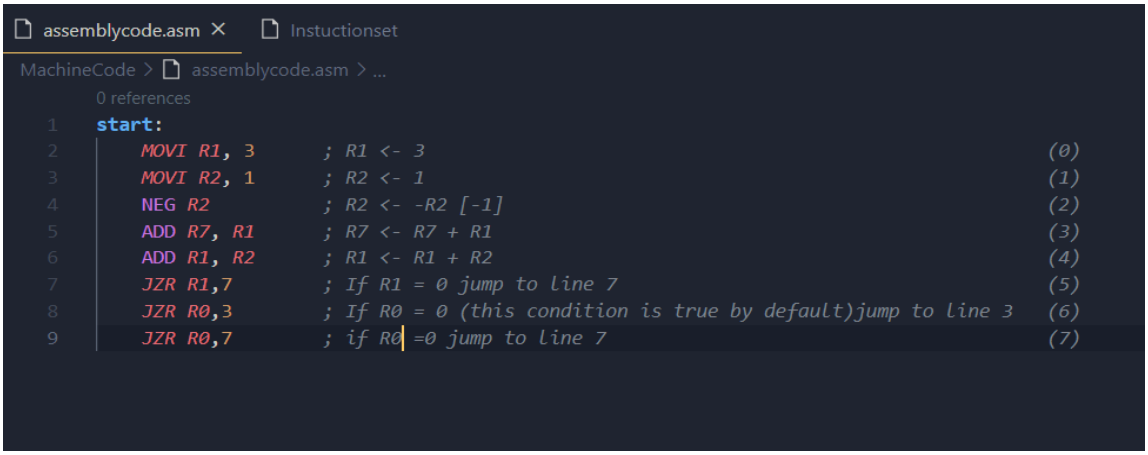
```
set_property PACKAGE_PIN U5 [get_ports {S_7seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {S_7seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {S_7seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {S_7seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {S_7seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {S_7seg[6]}]

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

## Reset button
set_property PACKAGE_PIN U18 [get_ports Reset]
    set_property IOSTANDARD LVCMOS33 [get_ports Reset]
```

Instruction set

```
Assembly program was written to calculate the total of all integers between 1
and 3 (inclusive).
Sum always stored in Register 7.
Loop is working until from 3 down to 1 and in each iteration current number was
decremented and added to the sum.

If R0 is zero jump keep not going to 0 instruction again

(0) MOVI R1, 3        ; R1 <- 3
(1) MOVI R2, 1        ; R2 <- 1
(2) NEG R2          ; R2 <- -R2 [-1]
(3) ADD R7, R1        ; R7 <- R7 + R1
(4) ADD R1, R2        ; R1 <- R1 + R2
(5) JZR R1,7          ; If R1 = 0 jump to line 7
(6) JZR R0,3          ; If R0 = 0 (this condition is true by default)jump to
line 3
(7) JZR R0,7          ; if R0 =0 jump to line 7

Instructions in machine code
```

```
0)    100010000011
1)    100100000001
2)    010100000000
3)    001110010000
4)    000010100000
5)    110010000111
6)    110000000011
7)    110000000111
```

Assembly code (.asm code)



Conclusion

- Components designed in previous labs were useful when creating the nano processor (LUT, REGISTER, DECODERS, RCA, FA,HA etc.)
- Using tri state buffers to design multiplexers were much easier than using basic logic gates since it reduces redundant connections.
- 4-bit Add/Sub unit can be easily modified to do subtraction by having a SUB instruction.
- Only -8 to +7 values can be used for Add/Sub unit considering 2's complement method
- The logic for Overflow flag has been design considering 2's complement method.
- Designing the nano processor was easier since our team had 3 members to distribute the workload.
- Using online platforms such as Discord, Zoom, WhatsApp made it easier to communicate between team members.

Contribution

Name	Index Number	Contributions
Dasun Nimantha	190415K	program counter register bank. slow clock Nano processor
Nipun Pramuditha	190653L	multiplexer tri state buffer LUT XDC file Nano processor
Ayesh Vininda	190649F	instruction decoder ROM Add/sub unit. 3-bit adder Nano processor