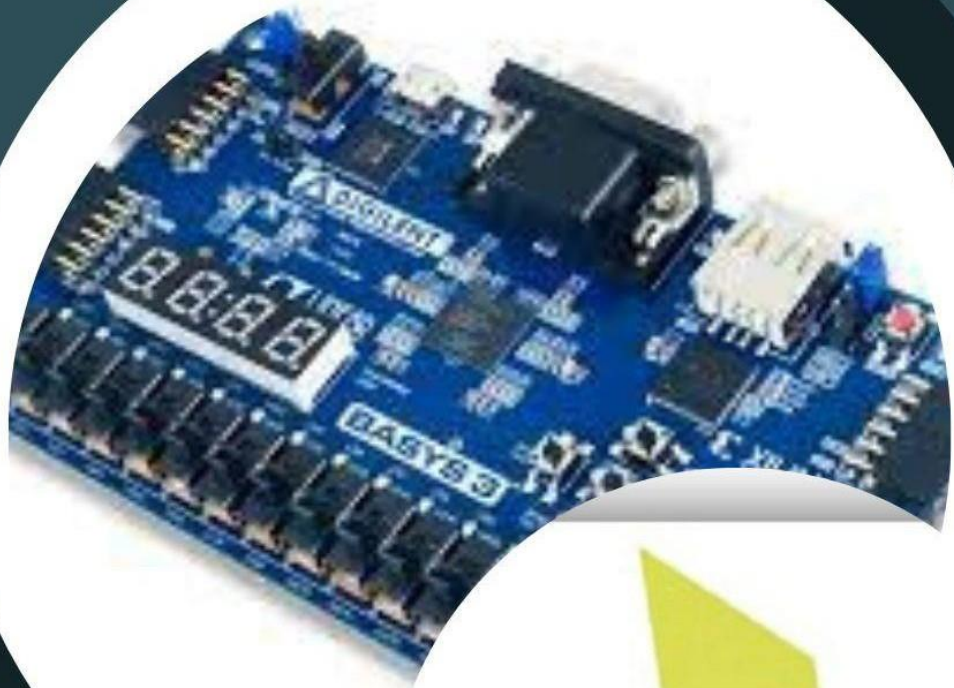


NANOPROCESSOR



GROUP 47

210329E LAKSARA K.Y

210523T RANAWEERA H.K

Assigned Lab Task

In this lab, our task was to build a microprocessor that is capable of executing four simple instructions. First, we built certain components, and Some of them we have already designed in previous labs.

In this lab we designed

1. And develop a 4-bit arithmetic unit that can add and subtract integers. (With help of lab 3)
2. 3-bit adder which is used to increment program counter.
3. 3-bit program counter using D flip flops.
4. A 2-way 4-bit multiplexer (Lab 4)
5. Four 8-way 4-bit mux which can take 8 inputs with 4 bits and gives a 4-bit output
6. Register bank with seven 4-bit registers.
7. To store our assembly codes, we build a program ROM using 12ROM16x1s.
8. Instruction decoder: activate necessary components based on the instructions we wish to execute.

9. Used 3, 4, and 12-bit buses to connect components.

10. After creating all these components, we created a Nano processor by connecting these components properly.

And then verified functionality using simulation and on the Basys3 board.

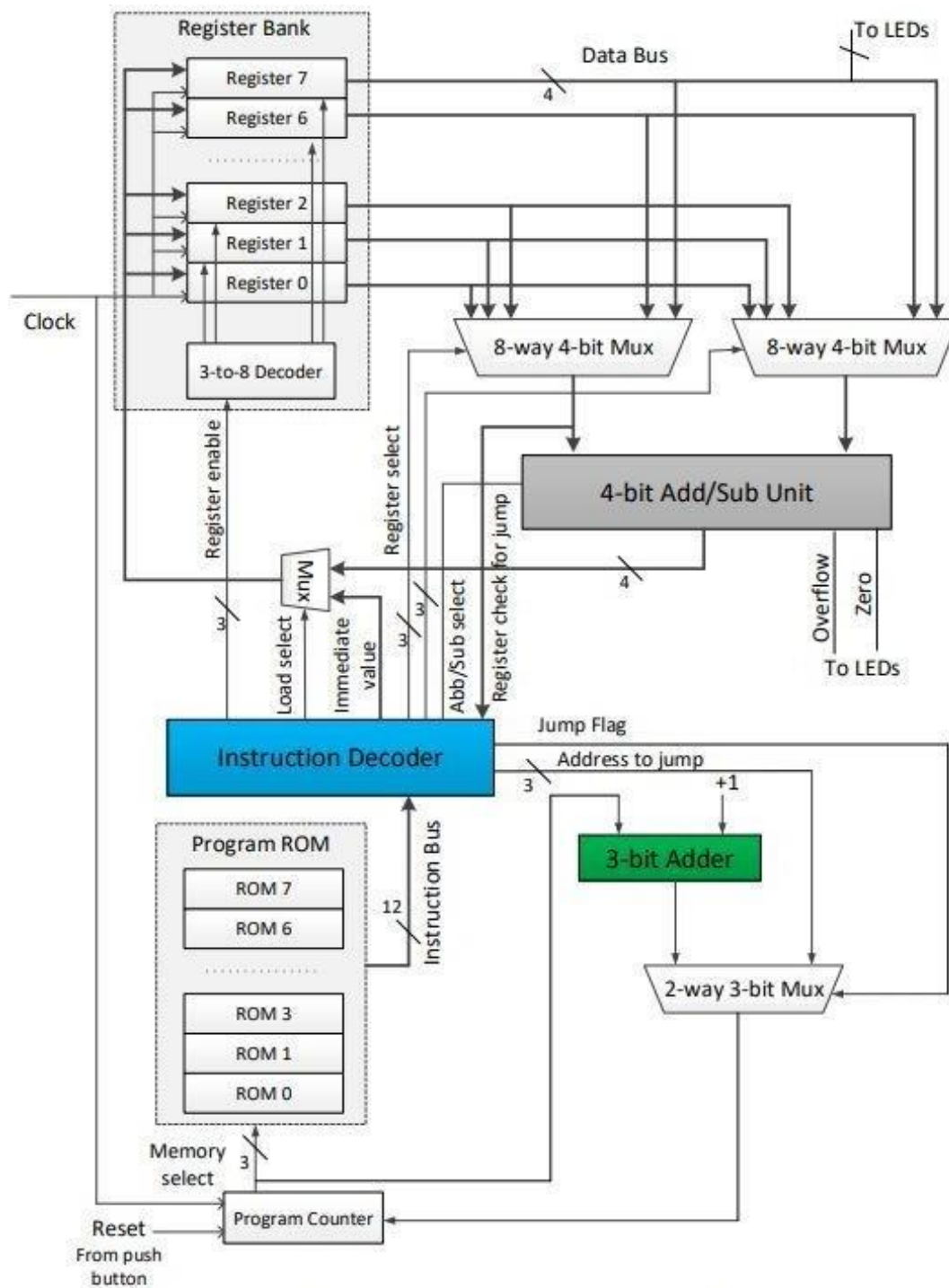


Figure 1 – High-level diagram of the nanoprocessor.

1. Adder/ Subtractor unit

Vhdl code

```
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

  -- Importing the RCA_4 component
  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 ;

  -- Signals declaration
  signal RCA_C : std_logic;
  signal B_2 : std_logic_vector(3 downto 0);
```

```

    signal S_0 : std_logic_vector(3 downto 0);

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

    -- Creating 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);

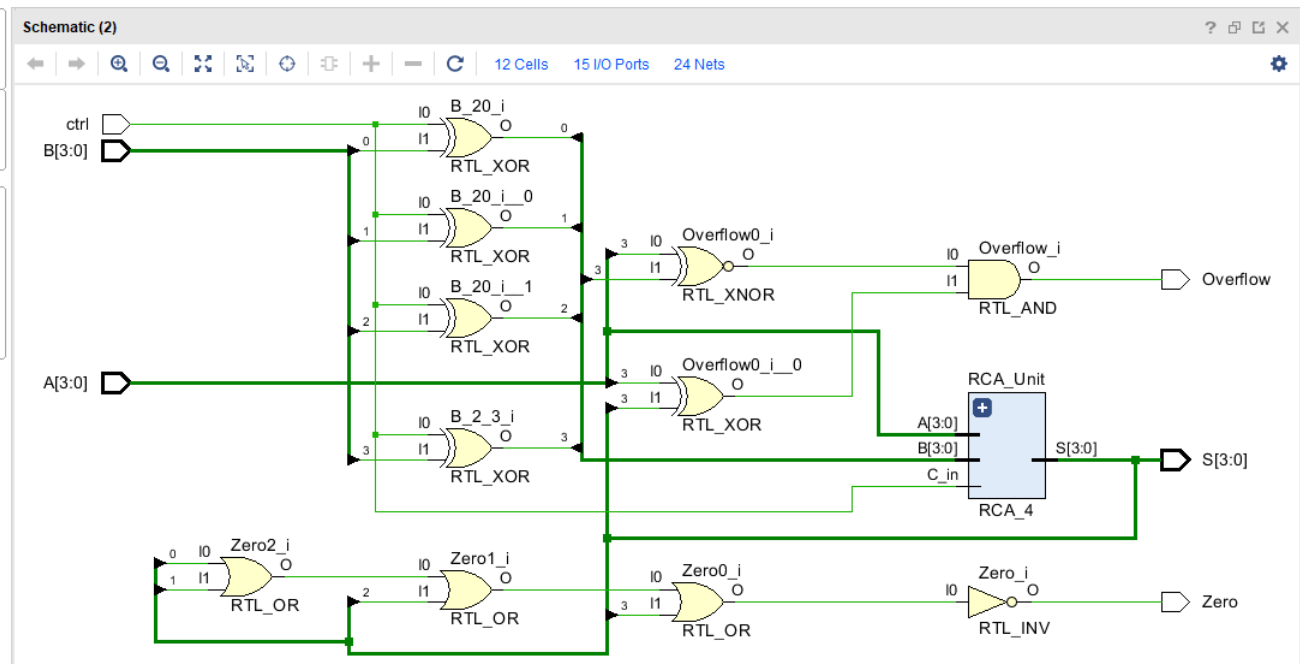
    -- Result logics

    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;

```

RTL Diagram



Test Bench file

```
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
    -- Importing the Add_Sub_Unit 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;
```

```
-- Signal declarations
```

```
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 component
```

```
UUT: Add_Sub_Unit
```

```
  Port map (
```

```
    A => A,
```

```
    B => B,
```

```
    ctrl => ctrl,
```

```
    S => S,
```



```
Overflow => Overflow,  
Zero => Zero  
);
```

```
process
```

```
begin
```

```
-- Testbench stimulus
```

```
-- 210523 == 0011 0011 0110 0101 1011
```

```
-- 210329 == 0011 0011 0101 1001 1001
```

```
-- Test case 1
```

```
A <= "0011"; --3
```

```
B <= "0011"; --3
```

```
ctrl <= '0';
```

```
wait for 100ns;
```

```
-- Test case 2
```

```
A <= "0011"; --3
```

```
B <= "0011"; --3
```

```
ctrl <= '1';
```

```
wait for 100ns;
```

```
-- Test case 3
```

```
A <= "0110"; --6
```

```
B <= "0101"; --5
```

```
ctrl <= '0';
```

```
wait for 100ns;
```

```
-- Test case 4
```

```
A <= "0101"; --5
```

```
B <= "1001"; --9
```

```
ctrl <= '1';
```

```
wait for 100ns;
```

```
-- Test case 5
```

```
A <= "1011"; --11
```

```
B <= "1001"; --9
```

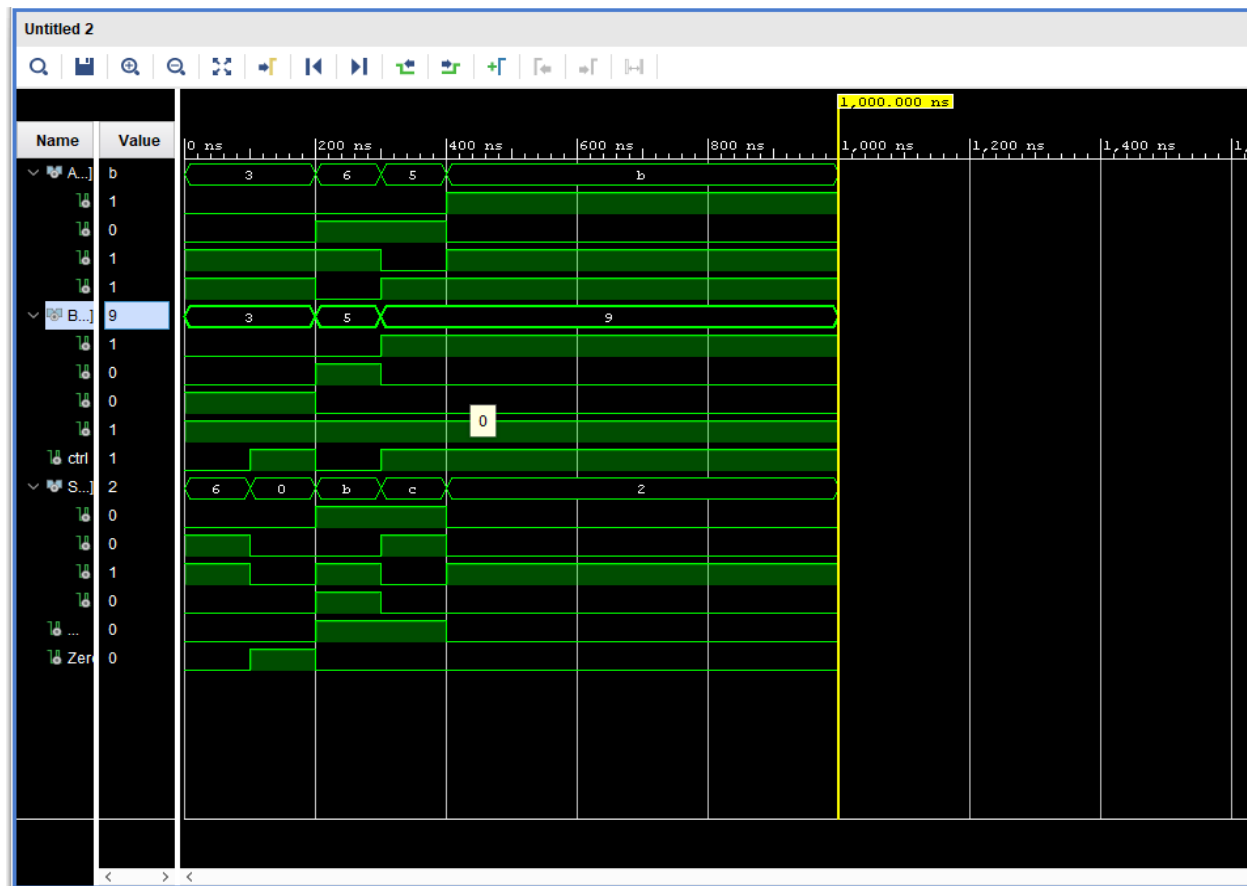
```
ctrl <= '1';
```

```
wait;
```

```
end process;
```

```
end Behavioral;
```

Timing Diagram



2. Program Rom

VHDL code

```
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

    --height 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:=(
        "100010000001", -- MOVI R1,1
        "100100000010", -- MOVI R2,2
        "100110000011", -- MOVI R3,3
        "101110000000", -- MOVI R7,0
        "001110010000", -- ADD R7,R1
        "001110100000", -- ADD R7,R2
```

```

"001110110000", -- ADD R7,R3
"110000000110" -- JZR R7,6

);

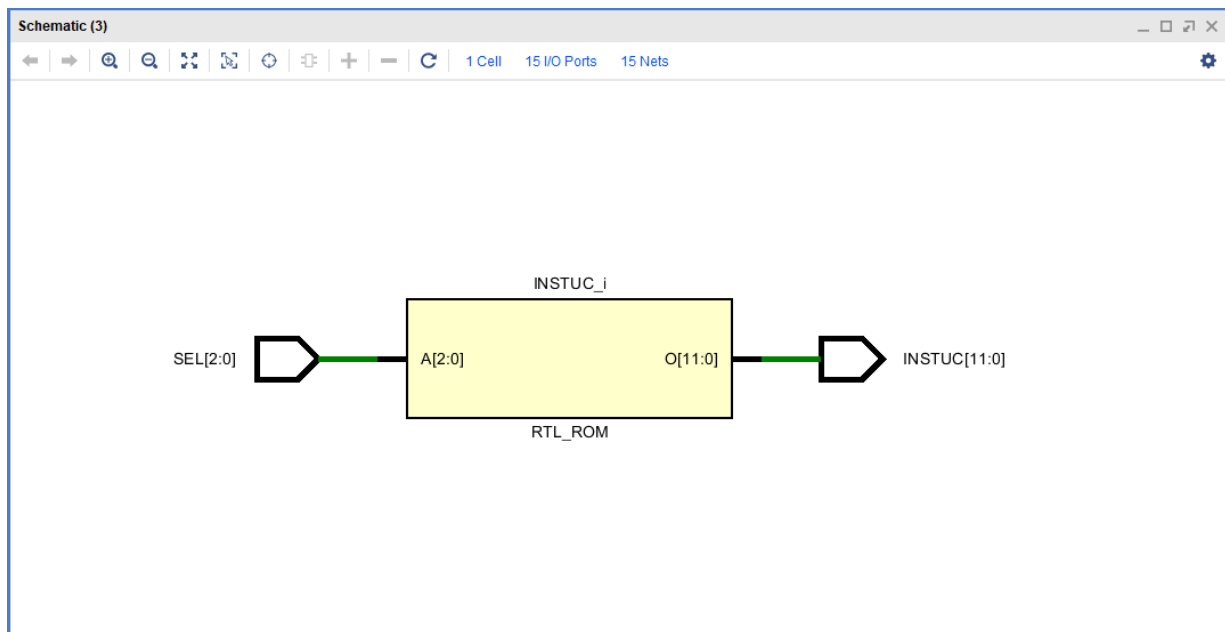
begin

--select instruction for given value
--use numeric library to convert binary to integer value
INSTUC <= PR_ROM(to_integer(unsigned(SEL)));

end Behavioral;

```

RTL Diagram



Test Bench File

```
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
```

```
);
```

```
-- Simulation process to test the behavior of the component
```

```
process
```

```
begin
```

```
-- Set SEL to "000"
```

```
SEL<="000";
```

```
wait for 100ns;
```

```
-- Set SEL to "101"
```

```
SEL<="101";
```

```
wait for 100ns;
```

```
-- Set SEL to "110"
```

```
SEL<="110";
```

```
wait for 100ns;
```

```
-- Set SEL to "100"
```

```
SEL<="100";
```

```
wait for 100ns;
```

```
-- Set SEL to "010"
```

```
SEL<="010";
```

```
wait for 100ns;
```

```
-- Set SEL to "111"
```

```
SEL<="111";
```

```
wait for 100ns;
```

```
-- Set SEL to "001"
```

```
SEL<="001";
```

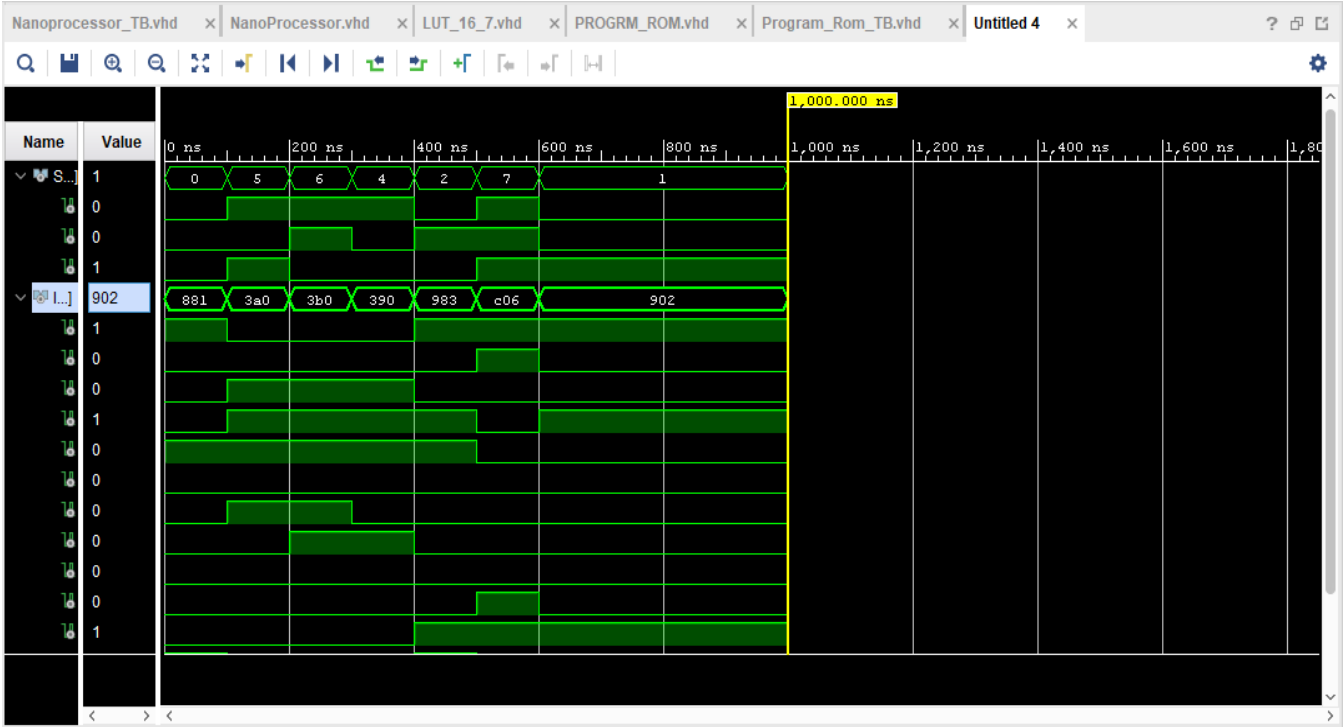
```
wait for 100ns;
```

```
wait;
```

```
end process;
```

```
end Behavioral;
```


Timing Diagram



3. Register Bank

VHDL Code

```
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 maps the R_Enable signals to individual enable signals for each register
```

```
Decoder : Decoder_3_to_8
```

```
port map (
```

```
I => R_Enable,
```

```
En => '1',
```

```
Y => En_Reg);
```

```
-- Register 0 (R0)
```

```
Reg_0 : Reg
```

```
port map (
```

```
    D => "0000",
```

```
    En => '1',
```

```
    Res => Res,
```

```
    Clk => Clk,
```

```
    Q => R0);
```

```
-- Register 1 (R1)
```

```
Reg_1 : Reg
```

```
port map (
```

```
    D => D,
```

```
    En => En_Reg(1),
```

```
    Res => Res,
```

```
    Clk => Clk,
```

```
    Q => R1);
```

```
-- Register 2 (R2)
```

```
Reg_2 : Reg
```

```
port map (  
    D => D,  
    En => En_Reg(2),  
    Res => Res,  
    Clk => Clk,  
    Q => R2);
```

-- Register 3 (R3)

```
Reg_3 : Reg  
port map (  
    D => D,  
    En => En_Reg(3),  
    Res => Res,  
    Clk => Clk,  
    Q => R3);
```

-- Register 4 (R4)

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

-- Register 5 (R5)

Reg_5 : Reg

port map (

D => D,

En => En_Reg(5),

Res => Res,

Clk => Clk,

Q => R5);

-- Register 6 (R6)

Reg_6 : Reg

port map (

D => D,

En => En_Reg(6),

Res => Res,

Clk => Clk,

Q => R6);

-- Register 7 (R7)

Reg_7 : Reg

port map (

D => D,

En => En_Reg(7),

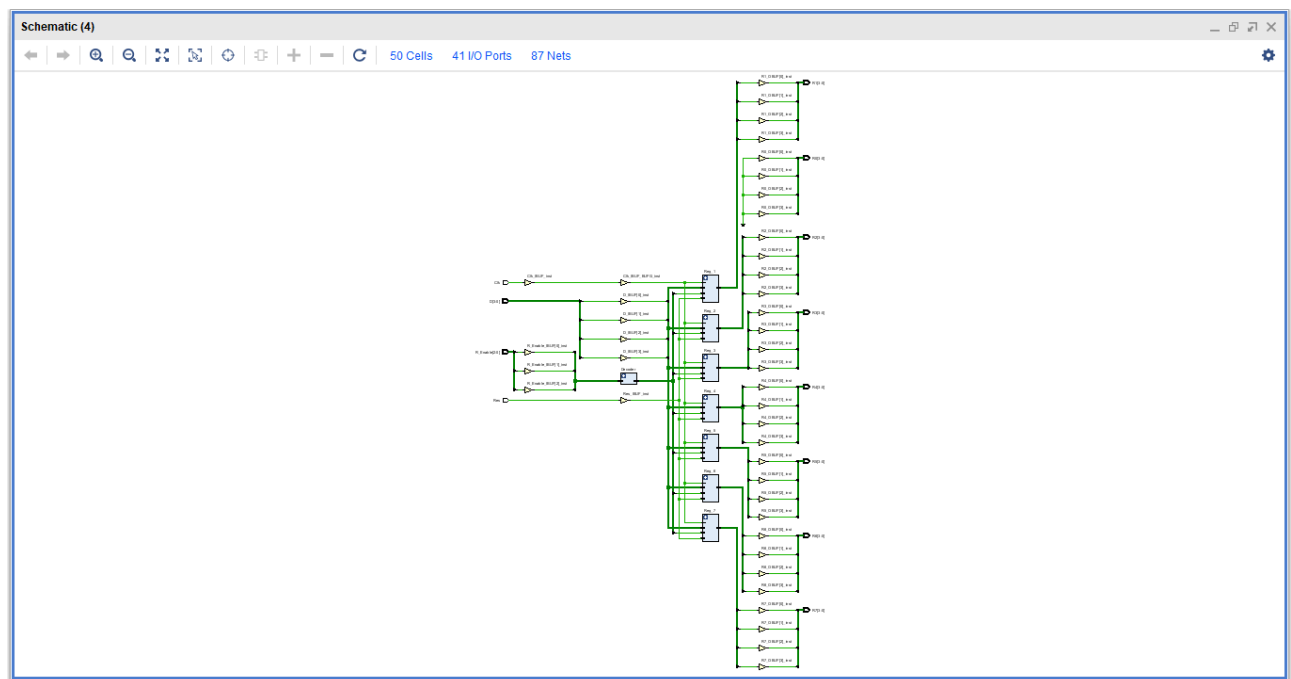
Res => Res,

Clk => Clk,

Q => R7);

end Behavioral;

Schematic



Test Bench file

```
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
```

```
    Res<='1';
```

```
    Clk <='0';
```

```
    wait for 50 ns;
```

```
    Clk <='1';
```

```
    wait for 50ns;
```

```
    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;
```

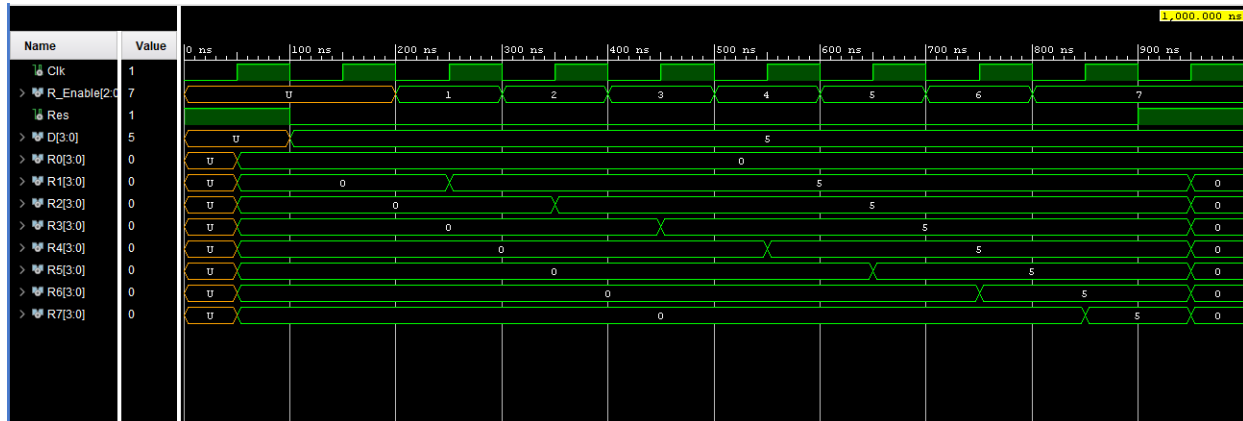
```
Clk <= '1';
```

```
wait;
```

```
end process;
```

```
end Behavioral;
```

Timing Diagram



4. 3-bit Adder

VHDL Code

```
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
        B : in STD_LOGIC_VECTOR(2 downto 0); --get inputs for B as vector
        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 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 in the nano processor
```

```
      S => S(0),
```

```
      C_Out => FA0_C);
```

```
  FA_1 : FA
```

```
    port map (
```

```
      A => A(1),
```

```
      B => B(1),
```

```
      C_in => FA0_C,
```

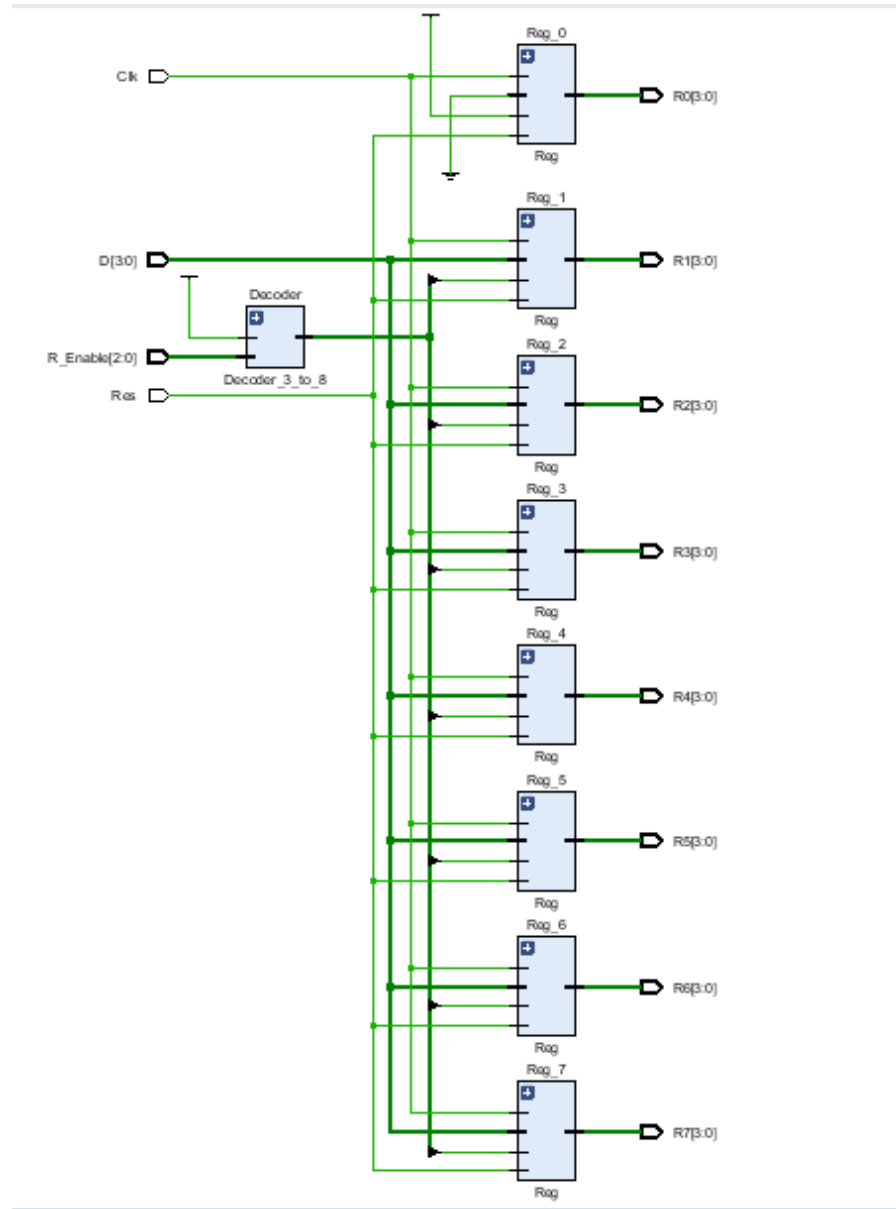
```
      S => S(1),
```

```
      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;
```

RTL Diagram



5. Instruction Decoder

VHDL Code

```
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 Subtract 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_to_4
    Port (
        I : in STD_LOGIC_VECTOR (1 downto 0);
        EN : in std_logic;
        Y : 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_to_4
```

```
Port map (
```

```
    I =>INSTUC(11 downto 10),
```

```
    EN =>'1',
```

```
    Y(0)=>ADD,
```

```
    Y(1)=>NEG,
```

```
    Y(2)=>MOV,
```

```
    Y(3)=>JZR
```

```
);
```

```
CTRL <= NEG; --select add or sub
```

```
--Select the load comes form instruction decoder or add sub unit
```

```
LOAD_SEL <= MOV;
```

```
IM_VAL <= INSTUC(3 downto 0) when (MOV = '1') else "0000";--1 0 R R R 0  
0 0 [d d d d]
```

```
REG_EN <=INSTUC(9 downto 7) when (JZR = '0' ) else "000";--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) when (NEG ='0' ) else "000";
```

```
REG_SEL_B <=INSTUC(6 downto 4) when (NEG = '0' else INSTUC ( 9
downto 7);
```

```
--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;
```

Test Bench file

```
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 Subtract 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;
```

```
signal INSTUC: std_logic_vector(11 downto 0);  
signal J_CHK: std_logic;  
signal REG_EN,REG_SEL_A,REG_SEL_B,J_ADDR :std_logic_vector(2 downto  
0);  
signal LOAD_SEL,CTRL,J_FLAG:std_logic;  
signal IM_VAL:std_logic_vector(3 downto 0);
```

```
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;
end process;

end Behavioral;
```


Name	Value	0 ns	100 ns	200 ns	300 ns	400 ns	500 ns	600 ns	700 ns	800 ns	900 ns	1,000.000 ns
> INSTRUCT[11:0]	110	b06	780	f82				110				
J_CHK	1											
> REG_EN[2:0]	2	6	7					2				
> REG_S_[2:0]	2	6	7					2				
> REG_S_[2:0]	1		0					1				
> J_ADDR[2:0]	0	6	0	2				0				
LOAD_SEL	0											
CTRL	0											
J_FLAG	0											
> IM_VAL[3:0]	0	6	0	2				0				

6.Slow Clock

VHDL Code

```
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

    -- Define two signals used to generate the output clock signal
    SIGNAL count : integer := 1; -- Keeps track of the number of rising edges on
    Clk_in
    SIGNAL clk_status : std_logic := '0'; -- The current state of the output clock
    signal

begin
    process(Clk_in)
```

```
begin
```

```
-- Check if there is a rising edge on Clk_in
```

```
if(rising_edge(Clk_in)) then
```

```
    count <= count+1;
```

```
-- If count reaches a value of 5, it's time to toggle the output clock signal
```

```
if(count = 5) then
```

```
    clk_status <= not clk_status;
```

```
    Clk_out <= clk_status;
```

```
    count <= 1;
```

```
end if;
```

```
end if;
```

```
end process;
```

```
end Behavioral;
```

7.Look Up Table

VHDL Code

```
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);
          anode : out STD_LOGIC_VECTOR (3 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
```

```
);
```

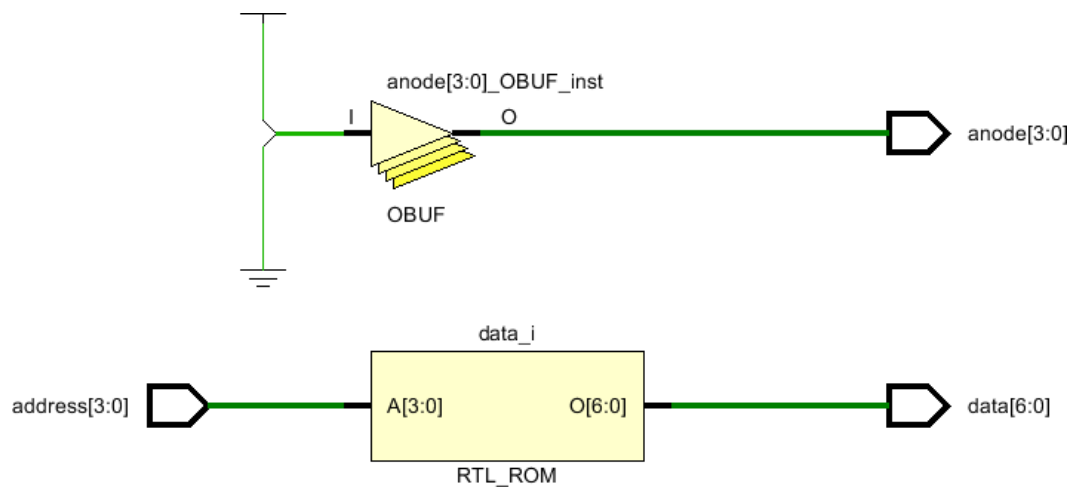
```
begin
```

```
    anode <= "1110";
```

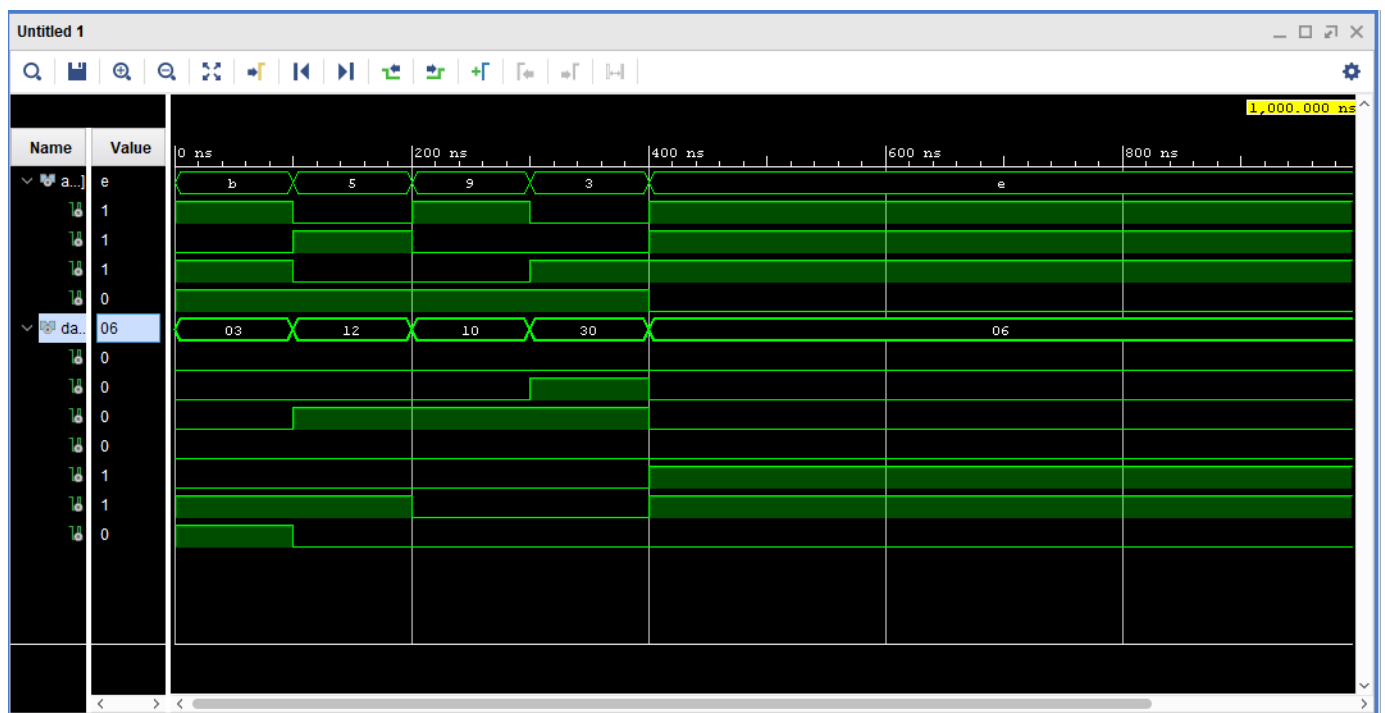
```
    data <= sevenSegment_ROM(to_integer(unsigned(address)));
```

```
end Behavioral;
```

RTL Diagram



Timing Diagram



Test Bench file

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

entity LUT_TB is
end LUT_TB;

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

begin

    UUT: LUT_16_7 PORT MAP (
        address => address,
        data    => data
    );
```

```
process
```

```
begin
```

```
-- 210523 == 0010 0001 0001 0101 0011
```

```
-- 210329 == 0010 0001 0001 0011 1001
```

```
address <= "1011";
```

```
wait for 100ns;
```

```
address <= "0101";
```

```
wait for 100ns;
```

```
address <= "1001";
```

```
wait for 100ns;
```

```
address <= "0011";
```

```
wait for 100ns;
```

```
address <= "1110";
```

```
wait;
```

```
end process;
```

```
end Behavioral;
```


8.4-bit Tri-state-buffer

VHDL Code

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

entity tri_state_buffer_4bit is

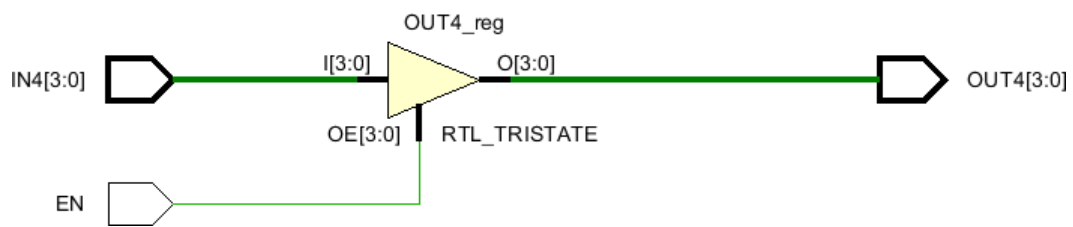
Port (   IN4 : in STD_LOGIC_VECTOR (3 downto 0);
        OUT4 : out STD_LOGIC_VECTOR (3 downto 0);
        EN : in STD_LOGIC);
end tri_state_buffer_4bit;

architecture Behavioral of tri_state_buffer_4bit is

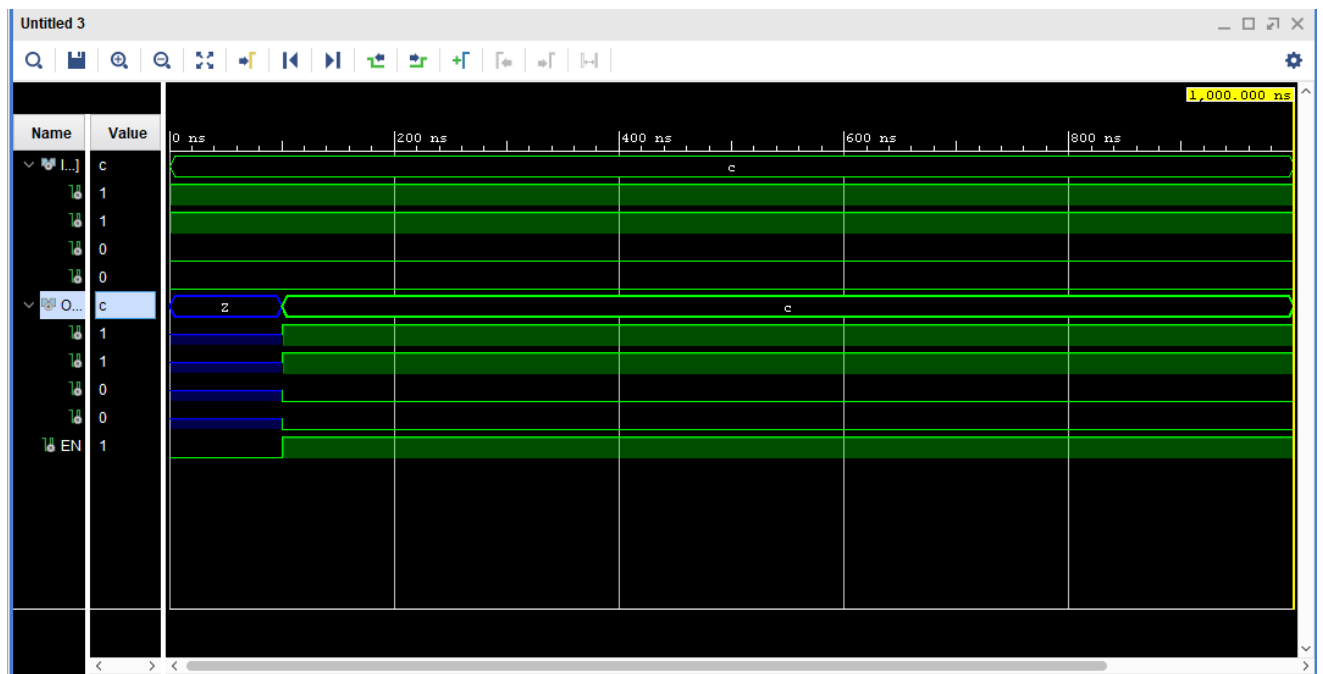
begin
OUT4<=IN4 WHEN (EN='1') else "ZZZZ";

end Behavioral;
```

RTL Diagram



Timing Diagram



Test Bench file

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

entity tri_state_buffer_4bit_TB is
-- Port ( );
end tri_state_buffer_4bit_TB;

architecture Behavioral of tri_state_buffer_4bit_TB 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 IN4 : std_logic_vector(3 downto 0);
signal OUT4 : std_logic_vector(3 downto 0);
signal EN : std_logic;
```

```
begin
```

```
UUT: tri_state_buffer_4bit
```

```
port map( IN4 => IN4,
```

```
          OUT4=> OUT4,
```

```
          EN=>EN);
```

```
process
```

```
begin
```

```
IN4<= "1100";
```

```
EN<='0';
```

```
wait for 100ns;
```

```
EN<='1';
```

```
wait;
```

```
end process;
```

```
end Behavioral;
```

9. 3-bit Tri-state-buffer

VHDL Code

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

entity tri_state_buffer_3bit is

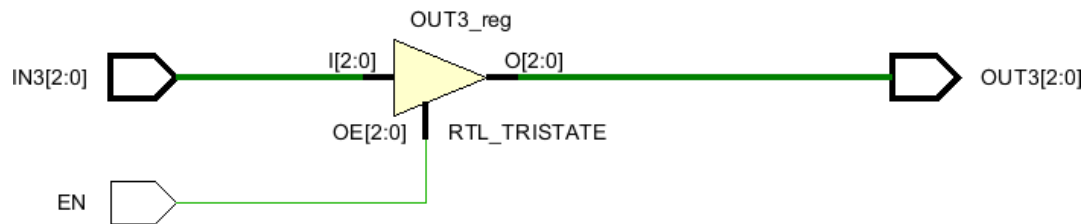
Port (   IN3 : in STD_LOGIC_VECTOR (2 downto 0);
        OUT3 : out STD_LOGIC_VECTOR (2 downto 0);
        EN : in STD_LOGIC);
end tri_state_buffer_3bit;

architecture Behavioral of tri_state_buffer_3bit is

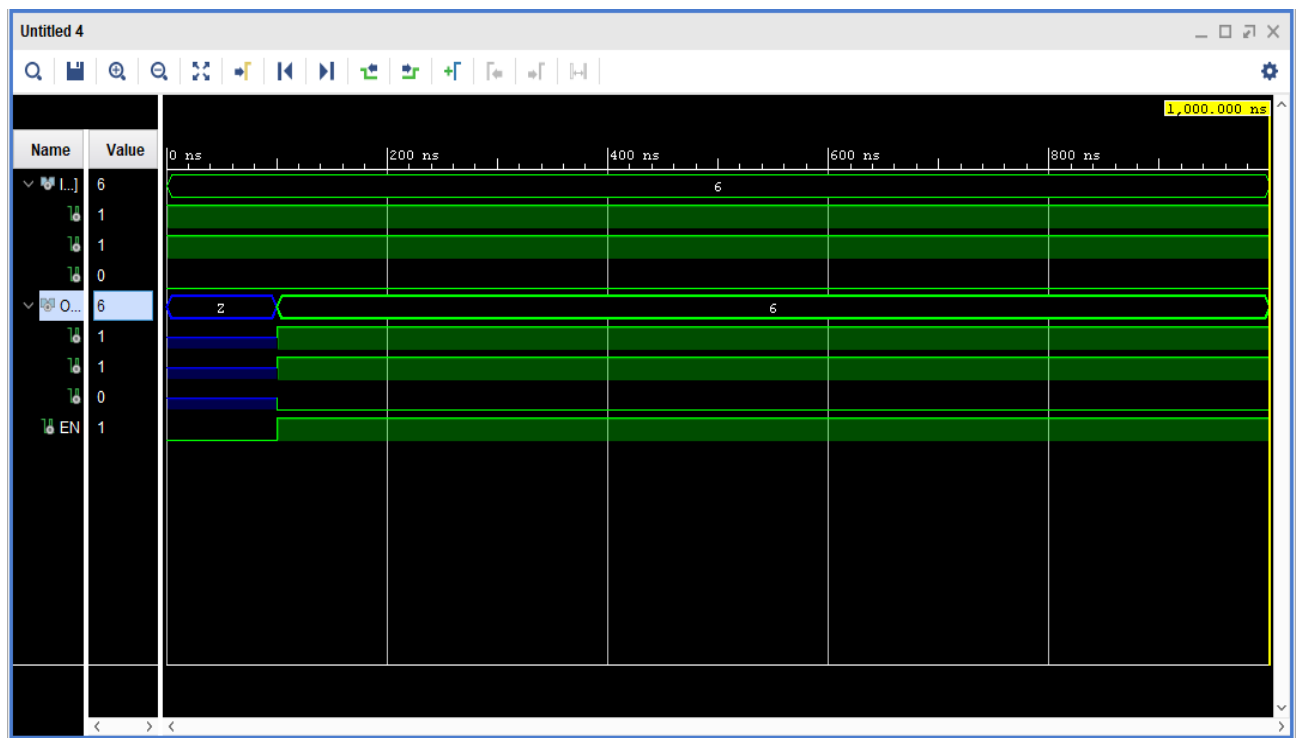
begin
OUT3<=IN3 WHEN (EN='1') else "ZZZ";

end Behavioral;
```

RTL Diagram



Timing Diagram



Test Bench file

```
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;
```


10. 2-way 3-bit Multiplexer

VHDL Code

```
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; --to store the inverse of the S input.

begin
```

```
-- Instantiate tri_state_buffer_0 component and map its input and output ports.
```

```
-- Adder_3 input is mapped to O output when NOTS is enabled.
```

```
tri_state_buffer_0 :tri_state_buffer_3bit
```

```
port map( IN3 => Adder_3,
```

```
          OUT3 => O,
```

```
          EN => NOTS);
```

```
-- Instantiate tri_state_buffer_1 component and map its input and output ports.
```

```
-- JUMP_TO input is mapped to O output when S is enabled.
```

```
tri_state_buffer_1: tri_state_buffer_3bit
```

```
port map( IN3 => JUMP_TO,
```

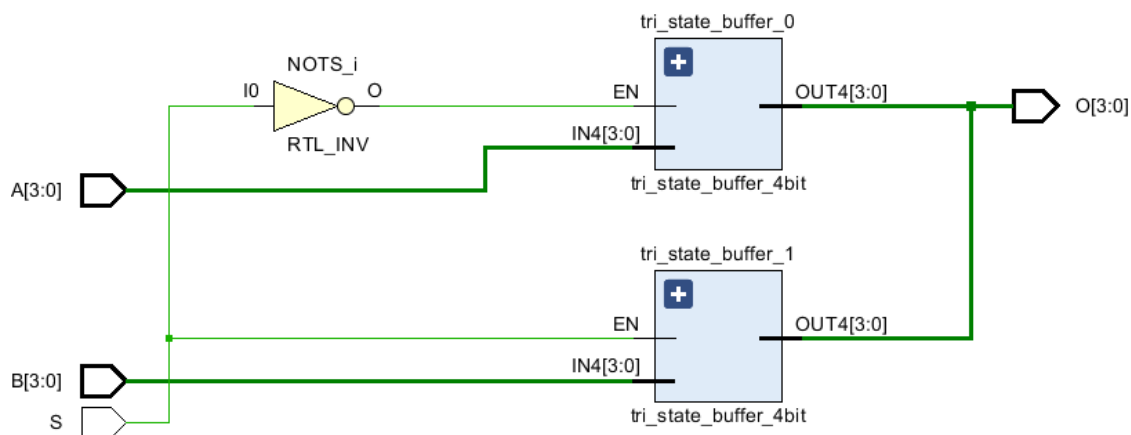
```
          OUT3 =>O,
```

```
          EN =>S);
```

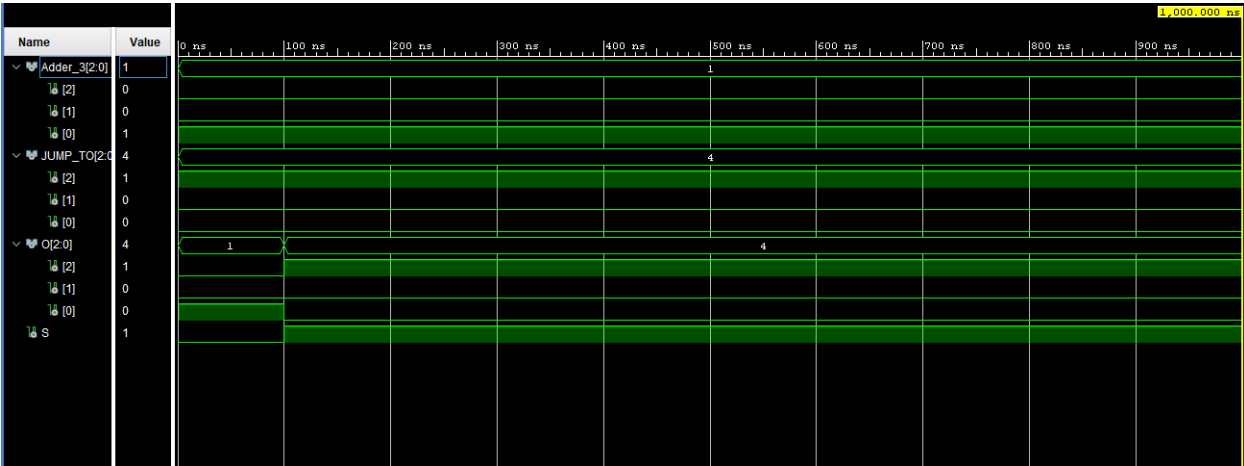
```
NOTS <= NOT S; -- Store the inverse of S input to NOTS signal.
```

```
end Behavioral;
```

RTL Diagram



Timing Diagram



Test Bench file

```
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;
```

11. 2-way 4-bit Multiplexer

VHDL Code

```
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

    -- Declare a component called tri_state_buffer_4bit with three input/output
    ports.
    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; --to store the inverse of the S input.
```

```
begin
```

```
-- Instantiate tri_state_buffer_0 component and map its input and output ports.
```

```
-- The input A is mapped to the IN4 port, output O is mapped to OUT4 port when NOTS is enabled.
```

```
tri_state_buffer_0:tri_state_buffer_4bit
```

```
port map( IN4=> A,
```

```
          OUT4 => O,
```

```
          EN=> NOTS);
```

```
--Instantiate tri_state_buffer_1 component and map its input and output ports.
```

```
-- The input B is mapped to the IN4 port, output O is mapped to OUT4 port when S is enabled.
```

```
tri_state_buffer_1: tri_state_buffer_4bit
```

```
port map( IN4=> B,
```

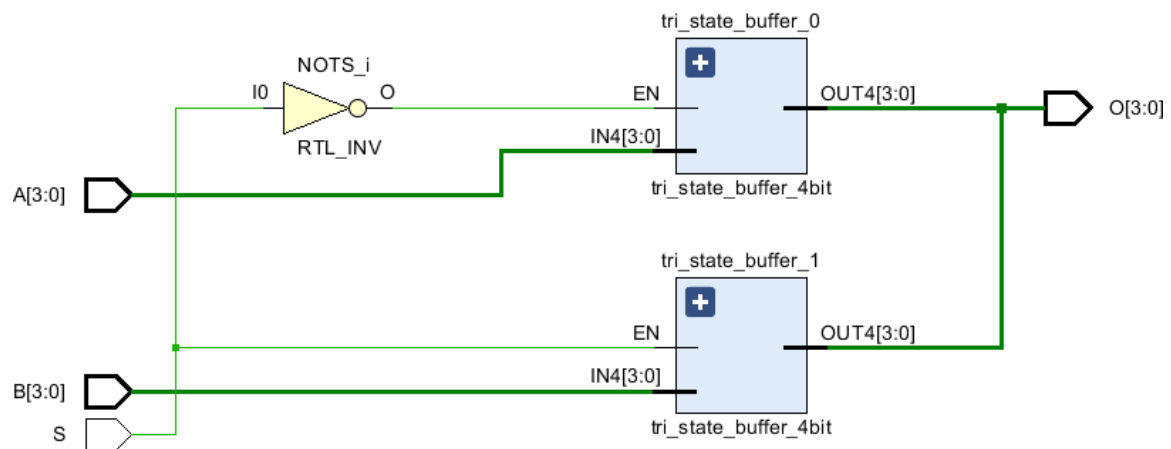
```
          OUT4 =>O,
```

```
          EN=>S);
```

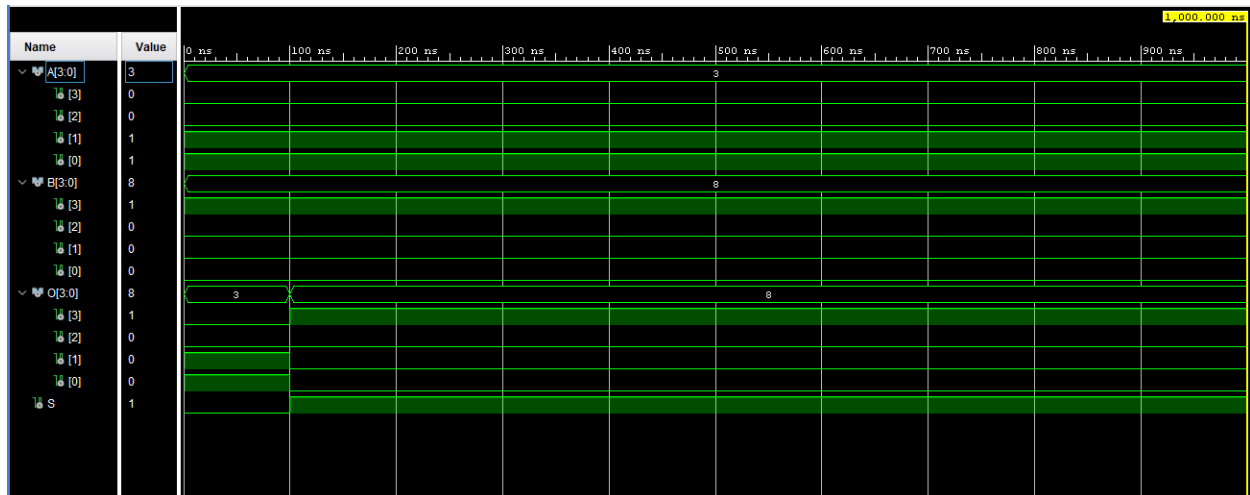
```
NOTS <= NOT S; -- Store the inverse of S input to NOTS signal.
```

```
end Behavioral;
```

RTL Diagram



Timing Diagram



Test Bench file

```
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;
```

12. 8-way 4-bit Multiplexer

VHDL Code

```
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

-- Repeat instantiation for inputs B-H.
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
```

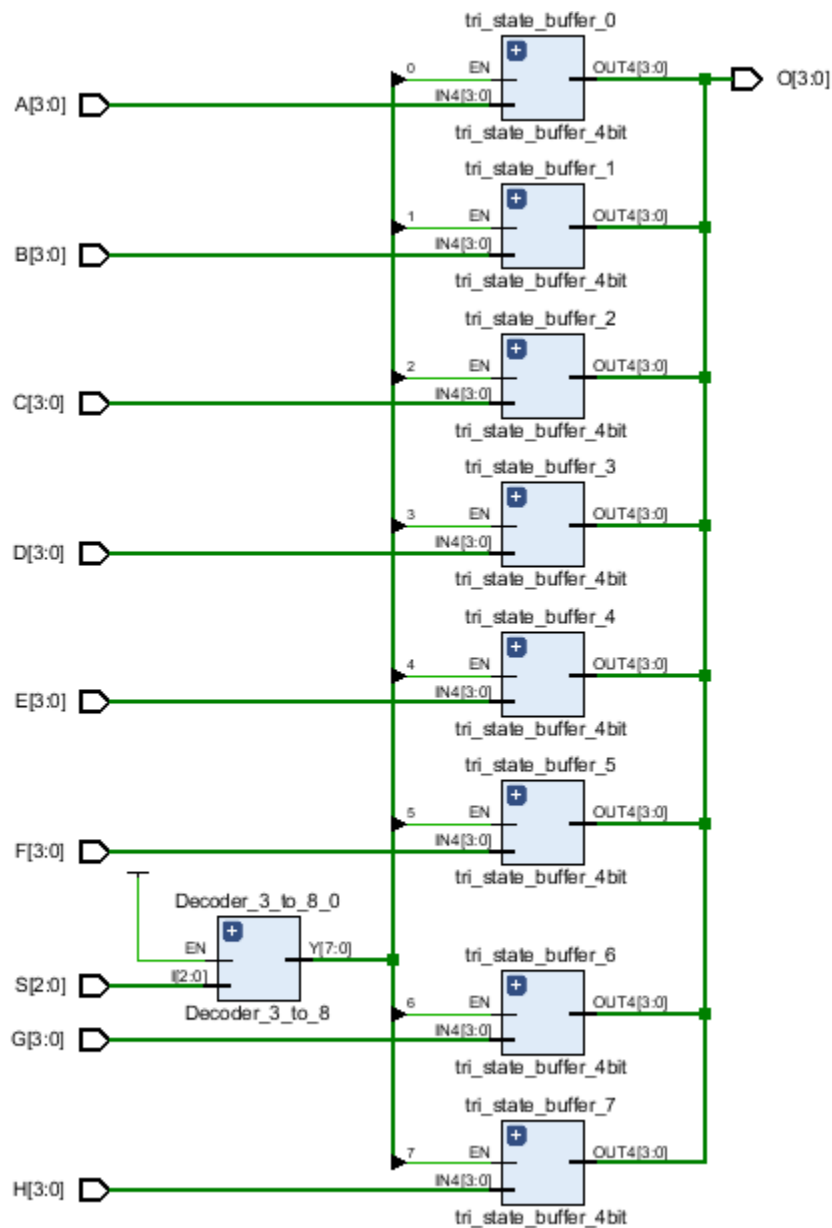
```
  port map( IN4 => H(3 downto 0),
```

```
           EN=> Y0(7),
```

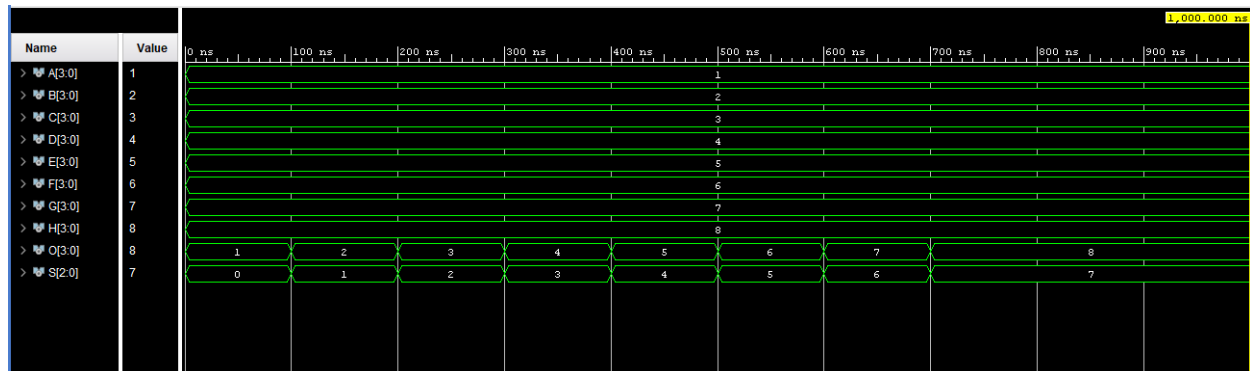
```
           OUT4 => O(3 downto 0));
```

```
end Behavioral;
```

RTL Diagram



Timing Diagram



Test Bench file

```
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;
```

13. NanoProcessor

VHDL Code

```
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);
        anode : out STD_LOGIC_VECTOR (3 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 Progm_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 ( I_In : in STD_LOGIC_VECTOR(2 downto 0);
        I_Out : out STD_LOGIC_VECTOR(2 downto 0));
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 segment display
```

```
component LUT_16_7
```

```
Port (
```

```
      address : in STD_LOGIC_VECTOR (3 downto 0);
```

```
      data : out STD_LOGIC_VECTOR (6 downto 0);
```

```
      anode : out STD_LOGIC_VECTOR(3 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_vector(2 downto 0); --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 implemmentation use

Bit3Adder : Bit_3_Adder

port map (

I_In =>Mem_Sel,

I_Out =>next_Address

);

Multiplexer_2way_3bit:Mux_2way_3bit

port map(

Adder_3 =>next_Address,

JUMP_TO =>Address_To_Jump,

S =>Jump,

O =>pc_in

);

Proqram_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_A,  
    B => selected_val_B,  
    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,
        data => S_7seg,
        anode => anode
    );

```

```

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));

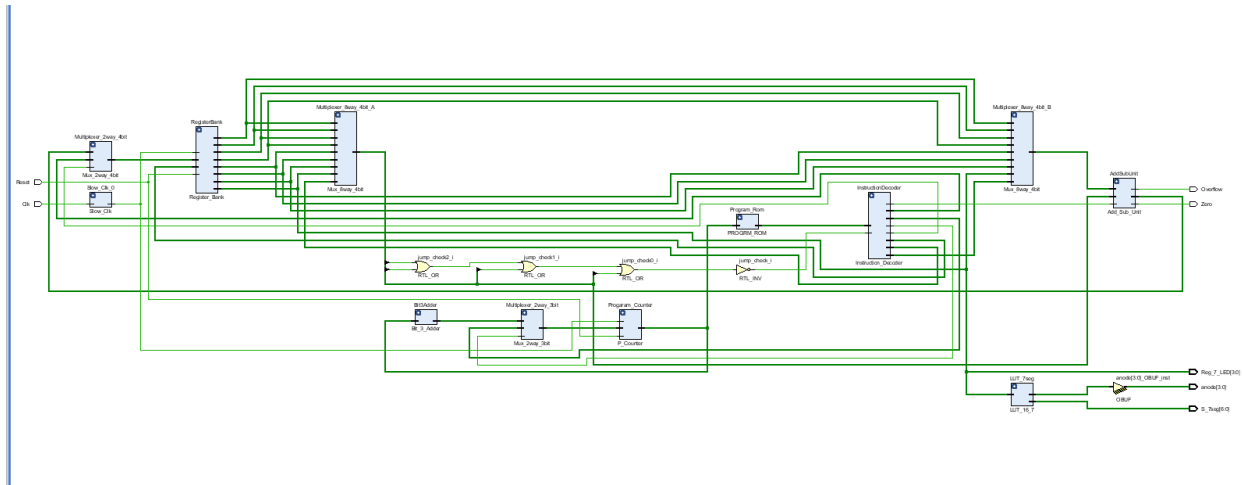
```

```

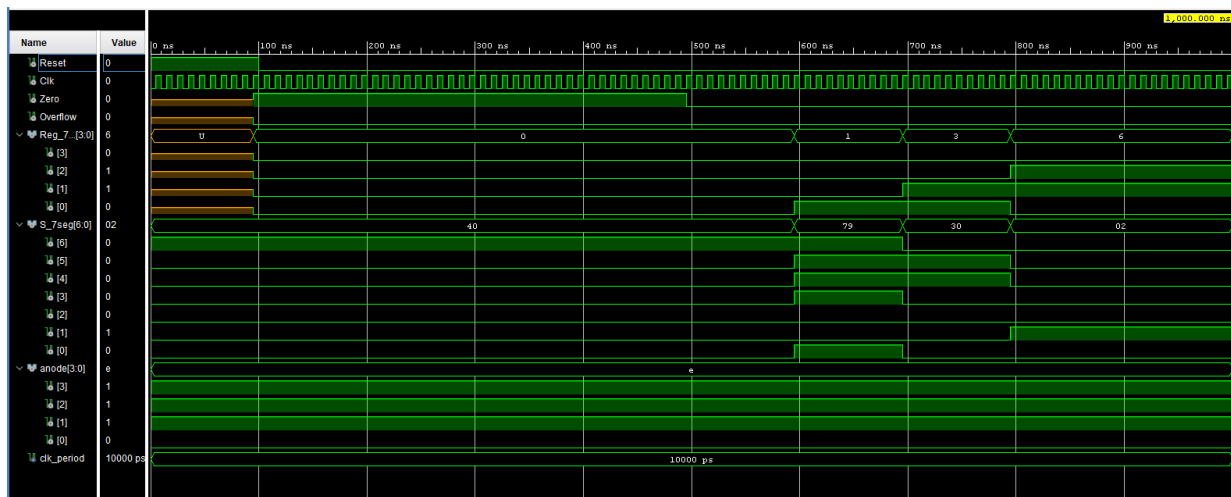
end Behavioral;

```


RTL Diagram



Timing Diagram



Test Bench file

```
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) ;
        anode : out STD_LOGIC_VECTOR(3 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);
```

```
signal anode : std_logic_vector(3 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,
```

```
    anode => anode
```

```
  );
```

```
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;`
```

FA.vhd

```
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 FA is
    Port ( A : in STD_LOGIC; --input A
          B : in STD_LOGIC; --input B
          C_in : in STD_LOGIC; --Carry in
          S : out STD_LOGIC; -- Sum
          C_out : out STD_LOGIC); --Carry out
end FA;

architecture Behavioral of FA is

    component HA
        port (
            A: in std_logic;
            B: in std_logic;
            S: out std_logic;
            C: out std_logic);
    end component;

    signal HA0_S,HA0_C,HA1_S,HA1_C : std_logic;
```

```
begin
```

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

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

```
S <= HA1_S;  
C_out <= HA0_C or HA1_C;
```

```
end Behavioral;
```

HA.vhd

```
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 HA is
  Port ( A : in STD_LOGIC; --input A
        B : in STD_LOGIC; --input B
        S : out STD_LOGIC; --output Sum
        C : out STD_LOGIC); --output Carry
end HA;

```

architecture Behavioral of HA is

```

begin
  S <= A xor B;
  C <= A and B;

end Behavioral;

```

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 1 up to 3 and in each iteration current number was decremented and added to the sum.

```

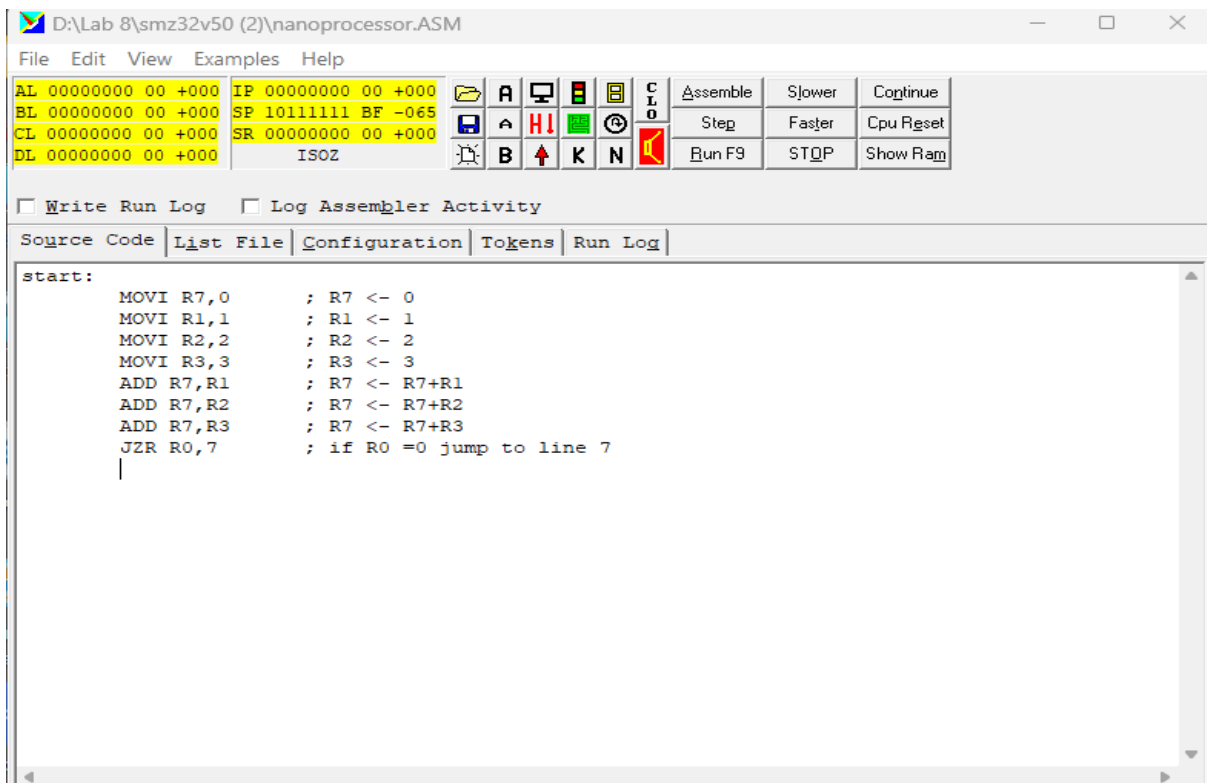
MOVI R7,0      ; R7 <- 0
MOVI R1,1      ; R1 <- 1
MOVI R2,2      ; R2 <- 2
MOVI R3,3      ; R3 <- 3
ADD R7,R1      ; R7 <- R7+R1
ADD R7,R2      ; R7 <- R7+R2
ADD R7,R3      ; R7 <- R7+R3
JZR R0,7       ; if R0 =0 jump to line 7

```

Instructions in Machine code

"101110000000", -- mov R7,0
"100010000001", -- mov R1,1
"100100000010", -- mov R2,2
"100110000011", -- mov R3,3
"001110010000", -- add R7,R1
"001110100000", -- add R7,R2
"001110110000", -- add R7,R3
"110000010111" -- jzr R0,7

Assembly code



The screenshot shows the nanoAssembler software interface. The title bar indicates the file path: D:\Lab 8\smz32v50 (2)\nanoprocessor.ASM. The menu bar includes File, Edit, View, Examples, and Help. Below the menu is a toolbar with icons for file operations and assembly functions. A status bar at the top shows registers AL, BL, CL, and DL with their current values and addresses. The main window is divided into two panes: Source Code and Tokens. The Source Code pane displays the assembly code for the program, starting with a 'start:' label. The Tokens pane shows the corresponding machine code in hexadecimal and binary.

```
start:
    MOVI R7,0      ; R7 <- 0
    MOVI R1,1      ; R1 <- 1
    MOVI R2,2      ; R2 <- 2
    MOVI R3,3      ; R3 <- 3
    ADD R7,R1      ; R7 <- R7+R1
    ADD R7,R2      ; R7 <- R7+R2
    ADD R7,R3      ; R7 <- R7+R3
    JZR R0,7       ; if R0 =0 jump to line 7
```


Conclusions

- Usage of buses simplified the design rather than using so many wires.
- We can calculate only the total of integers between 1 and 3 as our microprocessor is only 4 bit wide .
- -8 to 7 values can be used for ADD_SUB unit considering 2's complement.
- We need to hardcode assembly instructions as binary values, as microprocessors only understand machine language. We have achieved this via the help of ROM.
- Results are generated by the clock input.
- The Reset input can reset the nano processor.
- We could import required code files from previous labs and use them in building the processor.
- Proper performance of subcomponents was ensured before completing the microprocessor by simulating them using xsim simulator.

- The basic knowledge about how a microprocessor is internally structured and how those components work inside was received through the project .
- Team working skills were highlighted through this project, as we could focus on sub parts separately and finally combine them to form a microprocessor.
- Teamwork is essential in the computer science field.

CONTRIBUTION

NAME	CONTRIBUTION	TIME TAKEN
LAKSARA K.Y	<ul style="list-style-type: none"> • Register bank • 3 bit adder • Instruction Decoder • Program Counter • Add- sub unit • Program rom • 8 - way 4 bit Mux • 2 - way 3 bit Mux • 2 - way 4 bit mux • Tri state buffers • Multiplexers • LUT • RCA 	6 days

RANAWEERA H.K	<ul style="list-style-type: none"> • Register bank • 3 bit adder • Instruction Decoder • Program Counter • Add- sub unit • Program rom • 8 - way 4 bit Mux • 2 - way 3 bit Mux • 2 - way 4 bit mux • Tri state buffers • Multiplexers • LUT • RCA 	6 days
----------------------	--	---------------

THE END.