LAB 9-10 : NANOPROCESSOR DESIGN LAB REPORT

CS1050 Computer Organization and Digital Design

Group 21

- **230202D - GINIGE D.N.**
- **230104E - CHATHURANGA R.M.R.**
- **230102V - CHARINDITH A.J.R.J.**
- **230185B - FERNANDO S.D.**

INTRODUCTION

In this project, we designed and implemented a 4-bit nano processor that can execute a basic instruction set. The development process included building several key components, thereby interconnecting them together with synchronized clock. Each component playing a vital role in ensuring the processor's functionality and efficiency.

Components

1. **Program ROM**: This module stores the assembly code for the processor. It contains the instruction set that the processor will perform. In this project, program rom is hardcoded with the required program to execute in Basys 3 board.
2. **Instruction Decoder**: Functioning as the processor's control center, the Instruction Decoder analyzes instructions retrieved from the ROM and generates the corresponding control signals to coordinate internal components.
3. **Program Counter (PC)**: The Program Counter keeps track of the address of the next instruction to be executed. It increments after each operation to maintain a sequential instruction flow. If a jump instruction is received, then it points to the relevant address.
4. **Register Bank**: Acting as short-term memory, the Register Bank consists of multiple 4-bit registers that temporarily hold data for fast access and processing during execution.
5. **k-way b-bit Multiplexers**: These components allow dynamic selection of input data based on control signals. They play a key role in routing and managing data flow within the processor.
6. **3-bit Adder**: This component updates the Program Counter by incrementing its value, ensuring the orderly retrieval of the next instruction from memory during program execution. As we store only up to 8 ROM instructions, we require a 3bit adder.
7. **4-bit Add/Subtract Unit**: This unit performs arithmetic addition and subtraction on 4-bit values. For subtraction, one operand is handled by negating it in 2's complement representation.
8. **Buses**: While serving for communication among all components, Buses enable data and instruction transfer between key components such as the Program ROM, Instruction Decoder, and others.

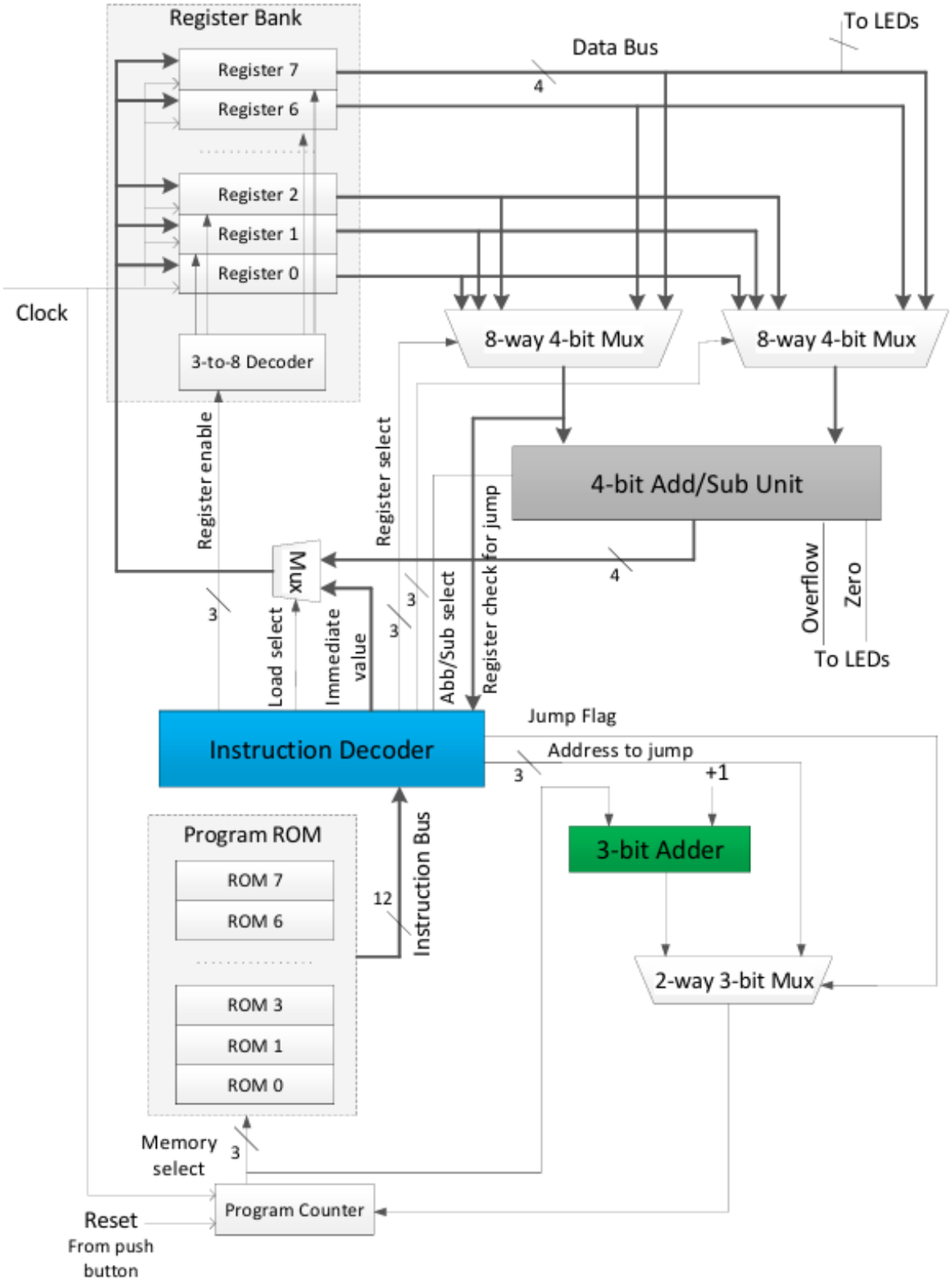
Execution Procedure

1. **Clocked Synchronization:** The processor operates using clocked synchronization, ensuring that all components perform their tasks in a coordinated manner based on a shared clock signal.
2. **Program Initialization**: The execution process starts by loading the assembly program into the Program ROM, where each instruction is stored at a designated memory address.
3. **Fetching Instructions**: The Program Counter (PC) holds the address of the next instruction to be executed. It sends this address to the Program ROM, which retrieves the corresponding instruction and forwards it to the Instruction Decoder.
4. **Decoding Instructions**: Upon receiving the instruction, the Instruction Decoder interprets its opcode and operands. It then generates the necessary control signals to guide the processor's actions.
5. **Data Handling**: If the instruction requires data manipulation—such as arithmetic operations or data movement—the Instruction Decoder coordinates with the Register Bank and multiplexers to access and modify the relevant register data.
6. **Performing Arithmetic**: For operations like addition or subtraction, the Add/Subtract Unit is engaged. It receives input from selected registers through the multiplexers, performs the required computation, and writes the result back into the target register.
7. **Updating the Program Counter**: Once an instruction is executed, the Program Counter is updated to point to the address of the next instruction. This usually involves incrementing the PC by one or applying an offset, depending on the instruction's nature and control flow.

Assembly Code Instruction Set

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

High-level diagram of the Nano processor



Nano Processor

Design Source File – Nano Processor

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

--##################################################################################

entity nanoprocessor is
    Port ( NanoClock : in STD_LOGIC;
           Reset : in STD_LOGIC;
           FourLED : out STD_LOGIC_VECTOR (3 downto 0);
           SevenSegment : out STD_LOGIC_VECTOR (6 downto 0);
           OverflowLED : out STD_LOGIC;
           ZeroLED : out STD_LOGIC ;
           Anode : out STD_LOGIC_VECTOR (3 downto 0));
end nanoprocessor;
architecture Behavioral of nanoprocessor is

--
##################################################################################
Component Slow_Clk
PORT( Clk_in  : in STD_LOGIC;
      Clk_out : out STD_LOGIC);
end Component;

--
##################################################################################
component RegisterBank is
    Port ( BankIn : in STD_LOGIC_VECTOR (3 downto 0);
           BankClock : in STD_LOGIC;
           BankReset : in STD_LOGIC;
           BankRegEn : in STD_LOGIC_VECTOR (2 downto 0);
           BankOut0 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut1 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut2 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut3 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut4 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut5 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut6 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

--##################################################################################
```

```
component EightWay4Bit_MUX is
    Port (
        In0, In1, In2, In3, In4, In5, In6, In7 : in STD_LOGIC_VECTOR(3 downto 0);
        Reg_sel : in STD_LOGIC_VECTOR (2 downto 0);
        Y       : out STD_LOGIC_VECTOR(3 downto 0)
    );
end component;

--#################################################################################

component ASU is
    Port ( AAddSub : in STD_LOGIC_VECTOR (3 downto 0);
           BAddSub : in STD_LOGIC_VECTOR (3 downto 0);
           AddSubSelect : in STD_LOGIC;
           OAddSub : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC);
end component;

--#################################################################################

component TwoWay4Bit_Mux is
    Port ( In0 : in STD_LOGIC_VECTOR (3 downto 0);
           In1 : in STD_LOGIC_VECTOR (3 downto 0);
           S0 : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

--#################################################################################

component InstructionDecoder is
    Port ( InstructionBus : in STD_LOGIC_VECTOR (11 downto 0);
           RegValueForCheck : in STD_LOGIC_VECTOR (3 downto 0);
           JumpFlag : out STD_LOGIC;
           JumpAddress : out STD_LOGIC_VECTOR (2 downto 0);
           AddSubSelect : out STD_LOGIC;
           RegSelect1 : out STD_LOGIC_VECTOR (2 downto 0);
           RegSelect2 : out STD_LOGIC_VECTOR (2 downto 0);
           ImmediateValue : out STD_LOGIC_VECTOR (3 downto 0);
           LoadSelect : out STD_LOGIC;
           RegEN : out STD_LOGIC_VECTOR (2 downto 0));
end component;

--#################################################################################
```

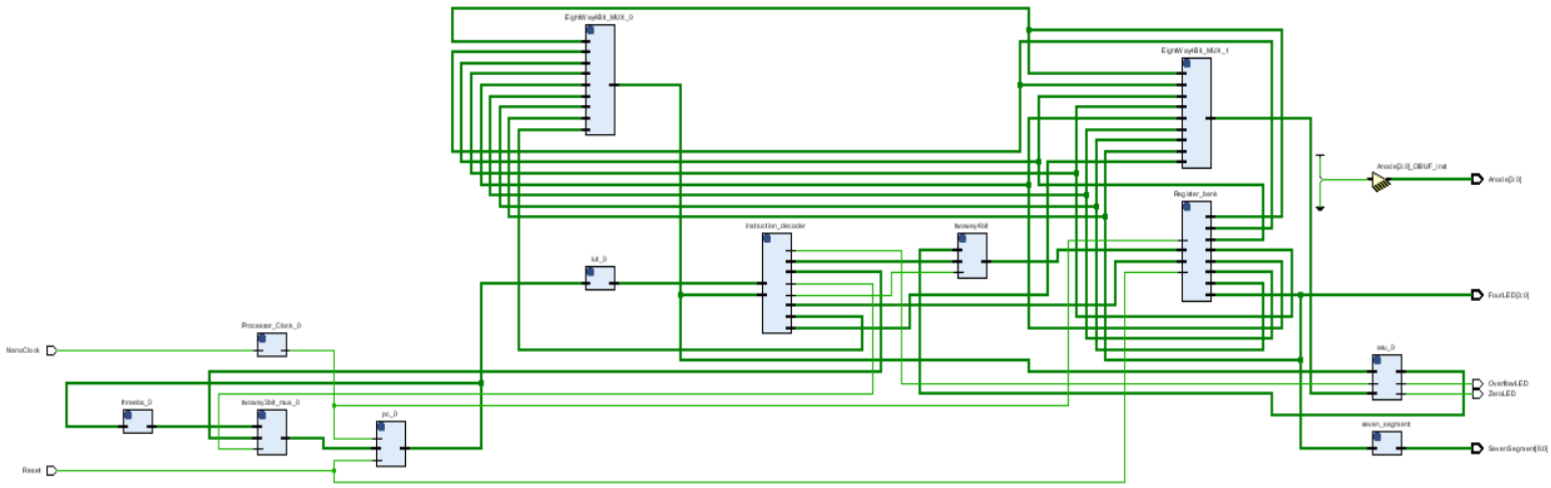
```
component LUT_8_12 is --this is programROM
    Port ( MemorySelect : in STD_LOGIC_VECTOR (2 downto 0);
           InstructionBus : out STD_LOGIC_VECTOR (11 downto 0));
end component;

--#################################################################################

component PC is
    Port ( PCIn : in STD_LOGIC_VECTOR (2 downto 0);
           PCOut : out STD_LOGIC_VECTOR (2 downto 0);
           Reset : in STD_LOGIC;
           PCClock : in STD_LOGIC);
end component;

--#################################################################################

component ThreeBA is
    Port ( AThreeBA : in STD_LOGIC_VECTOR (2 downto 0);
           OThreeBA : out STD_LOGIC_VECTOR (2 downto 0));
end component;

--#################################################################################

component TwoWay3Bit_MUX is
    Port ( In0 : in STD_LOGIC_VECTOR (2 downto 0);
           In1 : in STD_LOGIC_VECTOR (2 downto 0);
           S0 : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (2 downto 0));
end component;

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

signal clkout_to_components : std_logic;

SIGNAL D_0,D_1,D_2,D_3,D_4,D_5,D_6,D_7 : STD_LOGIC_VECTOR (3 downto 0);  --Data Bus
signal bank_data_in : std_logic_vector(3 downto 0);
```

```
signal eightway4bit_mux_out_0,eightway4bit_mux_out_1 : std_logic_vector(3 downto
0); --Bus inputs for 4bit add/sub unit
signal regselect1, regselect2 : std_logic_vector(2 downto 0);

signal bank_reg_en_from_instruction_decoder : std_logic_vector(2 downto 0);

signal twoway4bit_mux_in1, twoway4bit_mux_in2 : std_logic_vector(3 downto 0);
signal addsubselector : std_logic ;
signal overflow, zero : std_logic ;

signal loadselector : std_logic ;
signal instructions : std_logic_vector(11 downto 0); -- 12bit instruction Bus

signal jumpingflag : std_logic;
signal jumpingaddress : std_logic_vector(2 downto 0);

signal three_bit_adder_out : std_logic_vector(2 downto 0);
signal twoway3bit_mux_out : std_logic_vector(2 downto 0);

signal memoryselector : std_logic_vector(2 downto 0);


begin

Processor_Clock_0 : Slow_Clk
port map (
   Clk_in  => NanoClock , -- put nanoclock in constraints file
   Clk_out  => clkout_to_components);


Register_bank : RegisterBank
    Port map ( BankIn => bank_data_in ,
           BankClock => clkout_to_components,
           BankReset => Reset,  -- mapped to RESET button of whole processor
           BankRegEn => bank_reg_en_from_instruction_decoder,
           BankOut0 => D_0,
           BankOut1 => D_1,
           BankOut2 => D_2,
           BankOut3 => D_3,
           BankOut4 => D_4,
           BankOut5 => D_5,
           BankOut6 => D_6,
           BankOut7 => D_7);
```

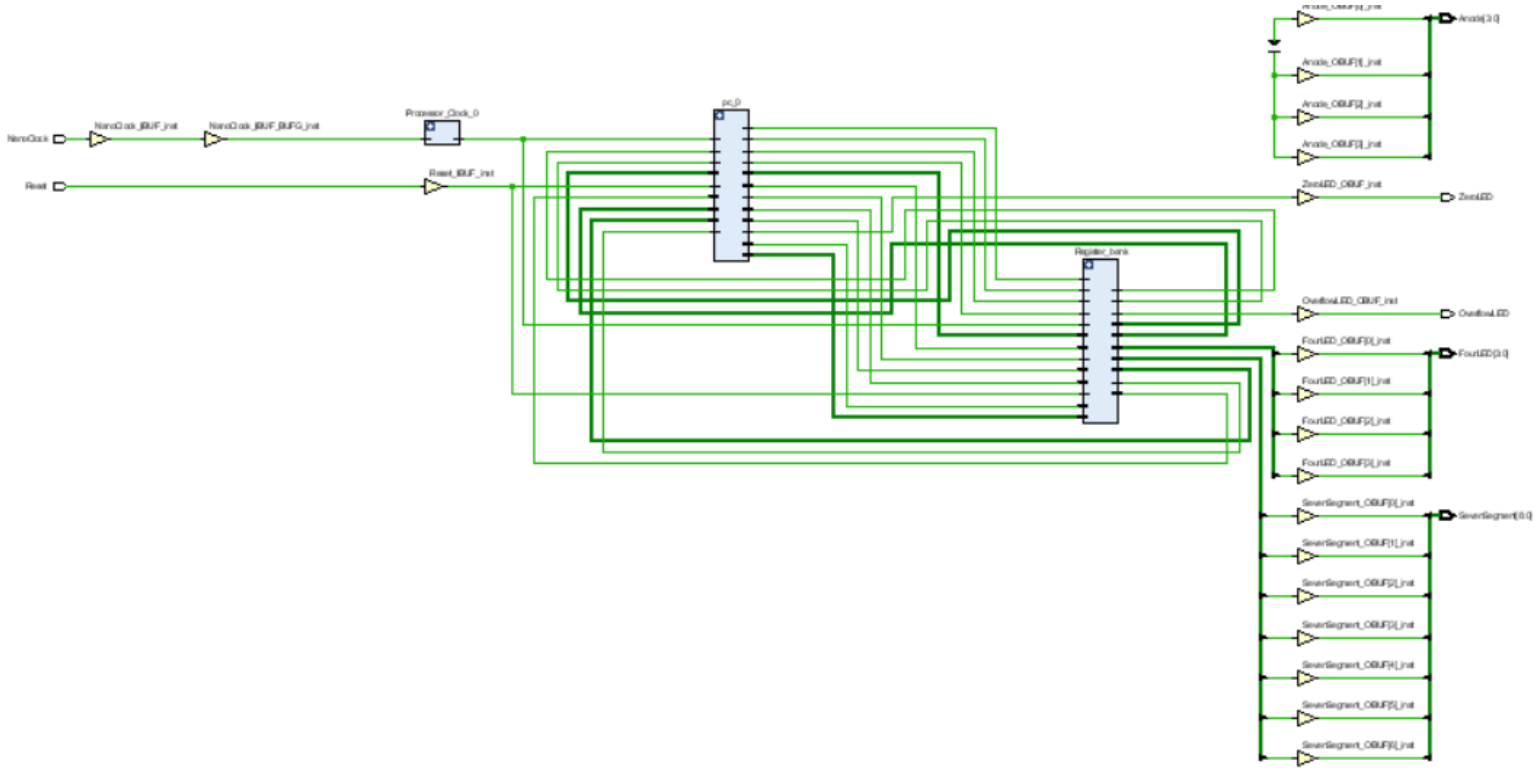
```
EightWay4Bit_MUX_0 : EightWay4Bit_MUX
    Port map(
       In0=> D_0,
       In1=> D_1,
       In2=> D_2,
       In3 => D_3,
       In4=> D_4,
       In5=> D_5,
       In6=> D_6,
       In7=> D_7,
       Reg_sel=> RegSelect1,
       Y => eightway4bit_mux_out_0 );

EightWay4Bit_MUX_1 : EightWay4Bit_MUX
           Port map (
              In0=> D_0,
              In1=> D_1,
              In2=> D_2,
              In3 => D_3,
              In4=> D_4,
              In5=> D_5,
              In6=> D_6,
              In7=> D_7,
              Reg_sel => RegSelect2,
              Y => eightway4bit_mux_out_1);

twoway4bit : TwoWay4Bit_Mux
     Port map( In0 =>twoway4bit_mux_in1,
                In1 => twoway4bit_mux_in2 ,
                S0  => loadselector ,
                Y   => bank_data_in );

 asu_0 :  ASU
    Port map ( AAddSub => eightway4bit_mux_out_0,
              BAddSub => eightway4bit_mux_out_1,
              AddSubSelect => addsubselector,
              OAddSub => twoway4bit_mux_in1,
              Overflow => overflow ,
              Zero => zero );

instruction_decoder : InstructionDecoder
    Port map( InstructionBus => instructions ,
           RegValueForCheck  => eightway4bit_mux_out_0 ,
           JumpFlag          => jumpingflag ,
           JumpAddress       => jumpingaddress,
```

```
           AddSubSelect      => addsubselector,
           RegSelect1        => regselect1,
           RegSelect2        => regselect2,
           ImmediateValue    => twoway4bit_mux_in2,
           LoadSelect        => loadselector,
           RegEN             => bank_reg_en_from_instruction_decoder );

 lut_0 : LUT_8_12
       Port map( MemorySelect => memoryselector,
              InstructionBus  => instructions);

pc_0 : PC
      Port map ( PCIn   => twoway3bit_mux_out,
                 PCOut  => memoryselector ,
                 Reset  => Reset,
                 PCClock => clkout_to_components );

threeba_0 : ThreeBA
     Port map ( AThreeBA => memoryselector,
                OThreeBA => three_bit_adder_out);

twoway3bit_mux_0 : TwoWay3Bit_MUX
    Port map ( In0 => three_bit_adder_out,
               In1 => jumpingaddress,
               S0 => jumpingflag,
               Y =>  twoway3bit_mux_out);


 seven_segment : LUT_16_7
        Port map( address => D_7 ,
                  data    => SevenSegment );


FourLED <= D_7 ;
OverflowLED <= overflow;
ZeroLED <=zero;
Anode <= "1110";

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC - NANO PROCESSOR



IMPLEMENTED DESIGN SCHEMATIC - NANO PROCESSOR

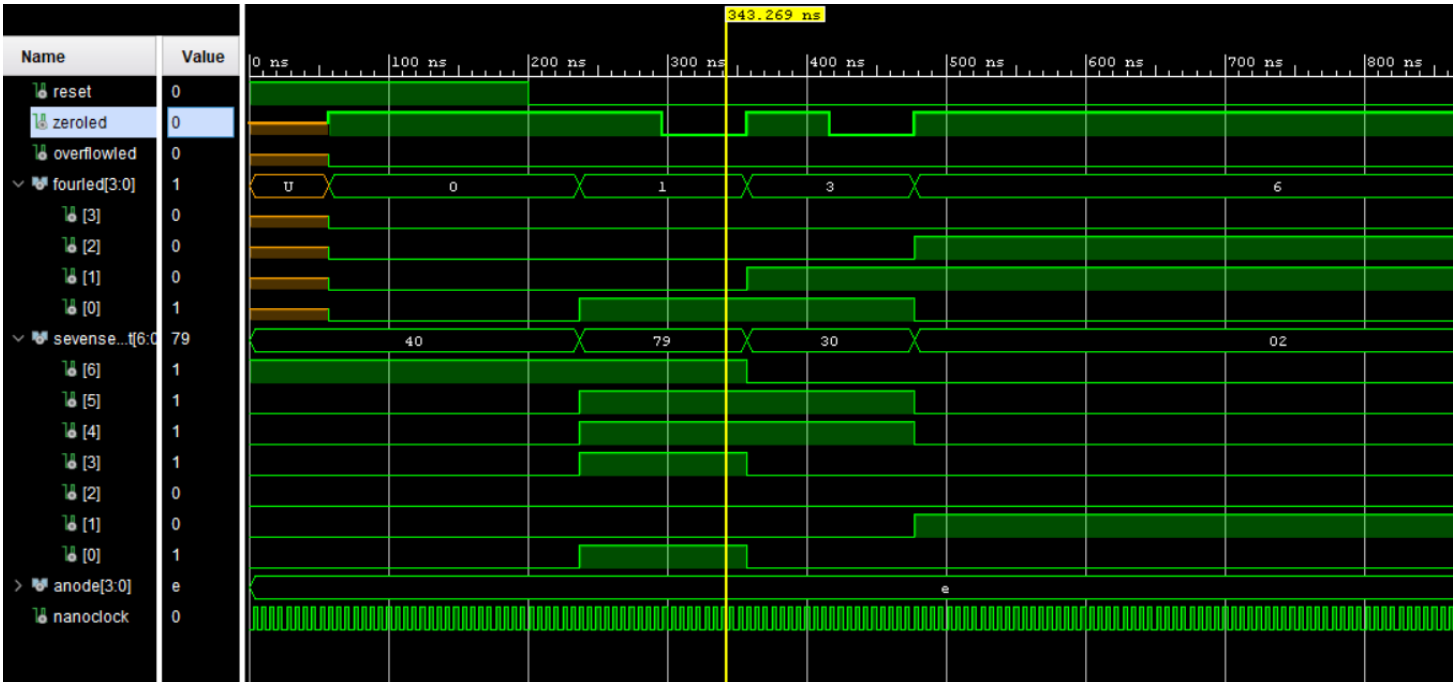


SIMULATION SOURCE FILE - NANO PROCESSOR

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

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

architecture Behavioral of NanoprocessorSim is

component nanoprocessor is
Port (NanoClock : in STD_LOGIC;
       Reset : in STD_LOGIC;
       FourLED : out STD_LOGIC_VECTOR (3 downto 0);
       SevenSegment : out STD_LOGIC_VECTOR (6 downto 0);
       OverflowLED : out STD_LOGIC;
       ZeroLED : out STD_LOGIC ;
       Anode : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal reset , zeroled , overflowled : STD_LOGIC;
signal fourled :STD_LOGIC_VECTOR (3 downto 0);
signal sevensegment :  STD_LOGIC_VECTOR (6 downto 0);
signal anode : STD_LOGIC_VECTOR (3 downto 0);
signal nanoclock :std_logic:='1';

begin

uut : nanoprocessor port map (
   NanoClock => nanoclock ,
   Reset => reset ,
   FourLED => fourled,
   SevenSegment => sevensegment ,
   OverflowLED => overflowled,
   ZeroLED => zeroled ,
   Anode => anode
    );


process begin
nanoclock <= NOT nanoclock;
wait for 3ns;
end process;
```

```
process
begin
    Reset <='1';
    wait for 200ns;
    Reset <='0';
    wait;
end process;

end Behavioral;
```

TIMING DIAGRAM - NANO PROCESSOR



PROGRAM ROM

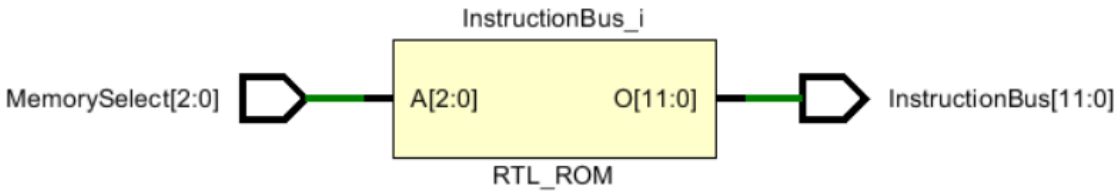
DESIGN SOURCE FILE – PROGRAM ROM

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

```
entity LUT_8_12 is
    Port ( MemorySelect : in STD_LOGIC_VECTOR (2 downto 0);
           InstructionBus : out STD_LOGIC_VECTOR (11 downto 0));
end LUT_8_12;

architecture Behavioral of LUT_8_12 is

type rom_type is array (0 to 7) of std_logic_vector(11 downto 0);
signal program_ROM : rom_type := (
"101110000001", -- MOVI R7, 1
"100100000010", -- MOVI R2, 2
"001110100000", -- ADD R7, R2
"100110000011", -- MOVI R3, 3
"001110110000", -- ADD R7, R3
"110000000111", -- JZR R0, 7
"110000000111", -- JZR R0, 7
"110000000111"  -- JZR R0, 7
);

begin
InstructionBus <= program_ROM(to_integer(unsigned(MemorySelect)));
end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – PROGRAM ROM

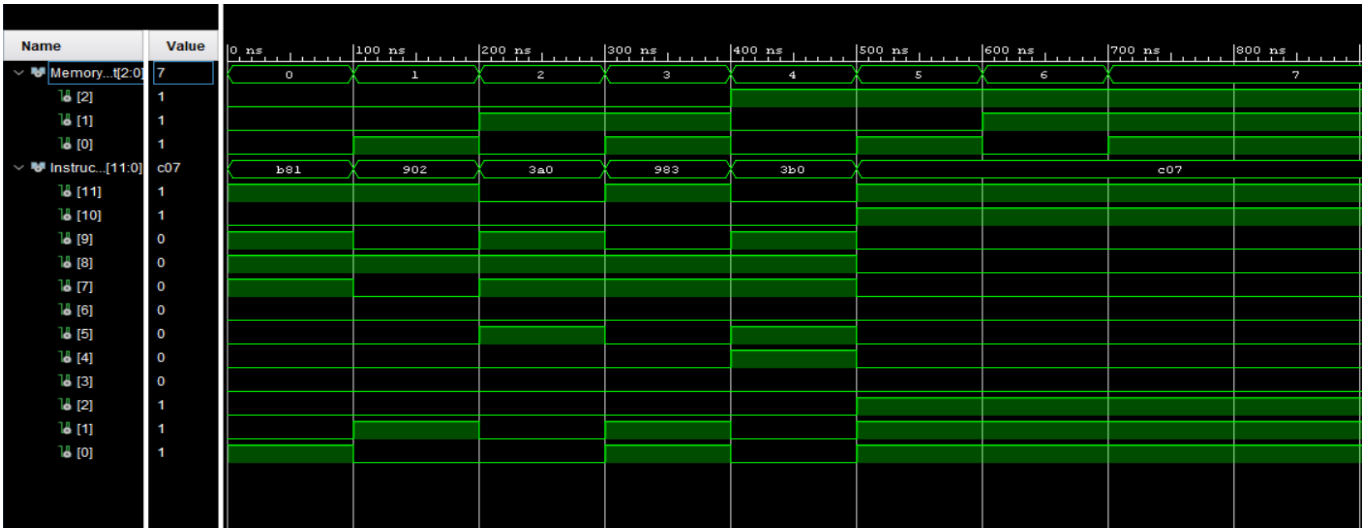
SIMULATION SOURCE FILE – PROGRAM ROM




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

entity LUT_8_12Sim is
    --  Port ( );
end LUT_8_12Sim;

architecture Behavioral of LUT_8_12Sim is
```

```
component LUT_8_12
port ( MemorySelect : in STD_LOGIC_VECTOR (2 downto 0);
       InstructionBus : out STD_LOGIC_VECTOR (11 downto 0));
end component;

signal Memory_Select : STD_LOGIC_VECTOR (2 downto 0);
signal Instruction_Bus : STD_LOGIC_VECTOR (11 downto 0);

begin
UUT: LUT_8_12
port map( MemorySelect => Memory_Select,
          InstructionBus => Instruction_Bus);
process
begin
Memory_Select <= "000";
wait for 100ns;
Memory_Select <= "001";
wait for 100ns;
Memory_Select <= "010";
wait for 100ns;
Memory_Select <= "011";
wait for 100ns;
Memory_Select <= "100";
wait for 100ns;
Memory_Select <= "101";
wait for 100ns;
Memory_Select <= "110";
wait for 100ns;
Memory_Select <= "111";
wait;
end process;
end Behavioral;
```

TIMING DIAGRAM – PROGRAM ROM

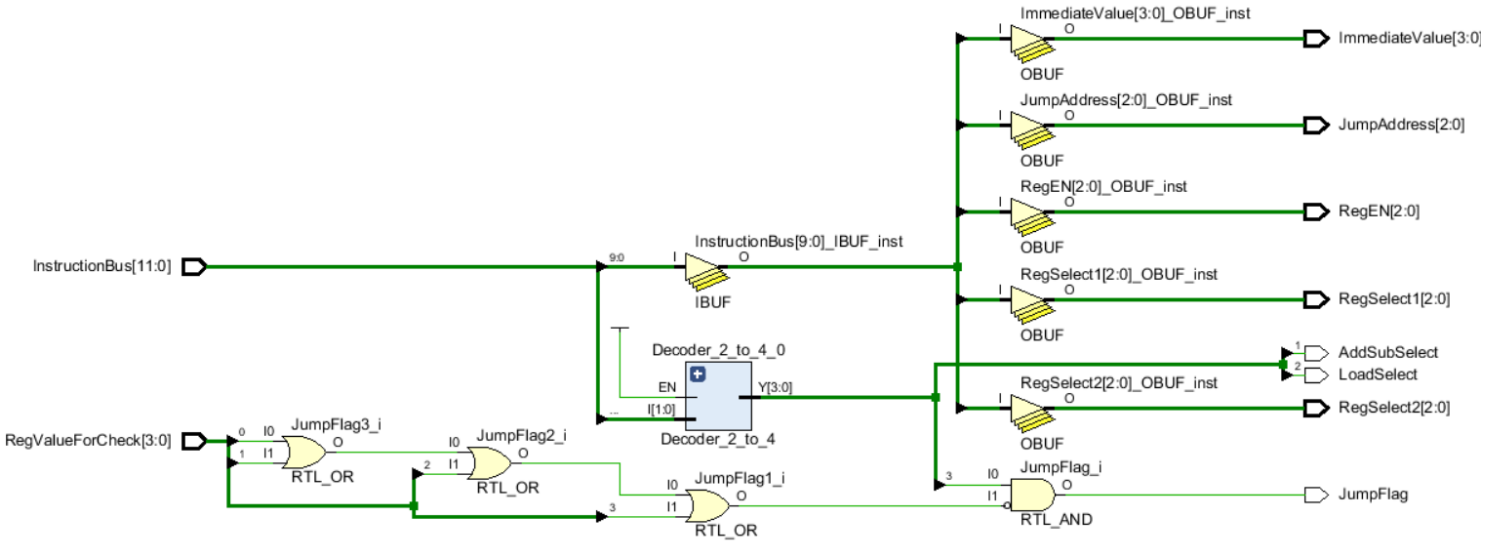


INSTRUCTION DECODER

DESIGN SOURCE FILE - INSTRUCTION DECODER

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

entity InstructionDecoder is
    Port ( InstructionBus : in STD_LOGIC_VECTOR (11 downto 0);
           RegValueForCheck : in STD_LOGIC_VECTOR (3 downto 0);
           JumpFlag : out STD_LOGIC;
           JumpAddress : out STD_LOGIC_VECTOR (2 downto 0);
           AddSubSelect : out STD_LOGIC;
           RegSelect1 : out STD_LOGIC_VECTOR (2 downto 0);
           RegSelect2 : out STD_LOGIC_VECTOR (2 downto 0);
           ImmediateValue : out STD_LOGIC_VECTOR (3 downto 0);
           LoadSelect : out STD_LOGIC;
           RegEN : out STD_LOGIC_VECTOR (2 downto 0));
end InstructionDecoder;

architecture Behavioral of InstructionDecoder is

component Decoder_2_to_4 is
    Port ( I : in STD_LOGIC_VECTOR (1 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal add, neg , movi, jzr : std_logic;

begin

Decoder_2_to_4_0 : Decoder_2_to_4
Port map ( I(0) =>InstructionBus(10),
I(1) => InstructionBus(11),
EN   => '1',
Y(0) => ADD,
Y(1) => NEG,
Y(2) => MOVI,
Y(3) => JZR );
RegSelect1   <= InstructionBus(9 downto 7);
RegSelect2   <= InstructionBus(6 downto 4);
RegEN        <= InstructionBus(9 downto 7);
AddSubSelect <= NEG;
JumpFlag      <= JZR AND ( NOT(RegValueForCheck(0) OR RegValueForCheck(1) OR
RegValueForCheck(2) OR RegValueForCheck(3)));
JumpAddress   <= InstructionBus(2 downto 0);
LoadSelect    <= MOVI;
ImmediateValue <= InstructionBus(3 downto 0);

end behavioral;
```

ELABORATED DESIGN SCHEMATIC - INSTRUCTION DECODER

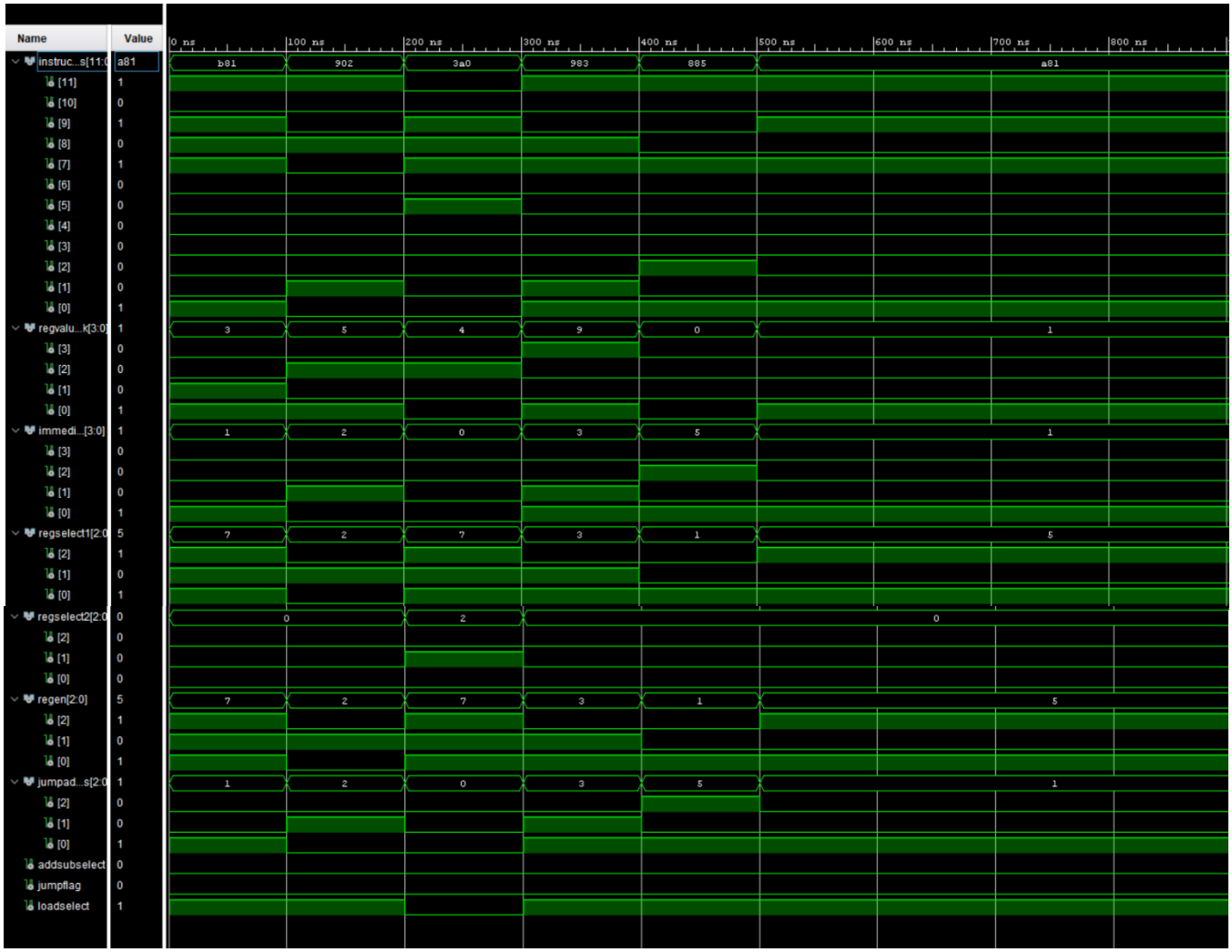


SIMULATION SOURCE FILE - INSTRUCTION DECODER

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


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

architecture Behavioral of InstructionDecoderSim is

component InstructionDecoder is
    Port ( InstructionBus : in STD_LOGIC_VECTOR (11 downto 0);
           RegValueForCheck : in STD_LOGIC_VECTOR (3 downto 0);
           JumpFlag : out STD_LOGIC;
           JumpAddress : out STD_LOGIC_VECTOR (2 downto 0);
           AddSubSelect : out STD_LOGIC;
           RegSelect1 : out STD_LOGIC_VECTOR (2 downto 0);
           RegSelect2 : out STD_LOGIC_VECTOR (2 downto 0);
           ImmediateValue : out STD_LOGIC_VECTOR (3 downto 0);
           LoadSelect : out STD_LOGIC;
           RegEN : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal instructionbus : STD_LOGIC_VECTOR (11 downto 0);
signal regvalueforcheck, immediatevalue : STD_LOGIC_VECTOR (3 downto 0);
signal regselect1, regselect2, regen, jumpaddress : STD_LOGIC_VECTOR
(2 downto 0);
signal addsubselect, jumpflag,loadselect : STD_LOGIC;
```

```
begin

UUT : InstructionDecoder port map(
     InstructionBus => instructionbus,
      RegValueForCheck => regvalueforcheck ,
      JumpFlag => jumpflag,
      JumpAddress=> jumpaddress,
      AddSubSelect => addsubselect,
      RegSelect1 => regselect1,
      RegSelect2=> regselect2,
      ImmediateValue => immediatevalue,
      LoadSelect =>  loadselect,
      RegEN => regen
);



process
begin

regvalueforcheck<="0011";
instructionbus<="101110000001";
wait for 100ns ;

regvalueforcheck <="0101";
instructionbus<="100100000010";
wait for 100ns;

regvalueforcheck<="0100";
instructionbus<="001110100000";
wait for 100ns;

regvalueforcheck<="1001";
instructionbus<="100110000011";
wait for 100ns;

regvalueforcheck<="0000";
instructionbus<="100010000101";
wait for 100ns;

regvalueforcheck<="0001";
instructionbus<="101010000001";
wait;
end process;

end Behavioral;
```

TIMING DIAGRAM - INSTRUCTION DECODER



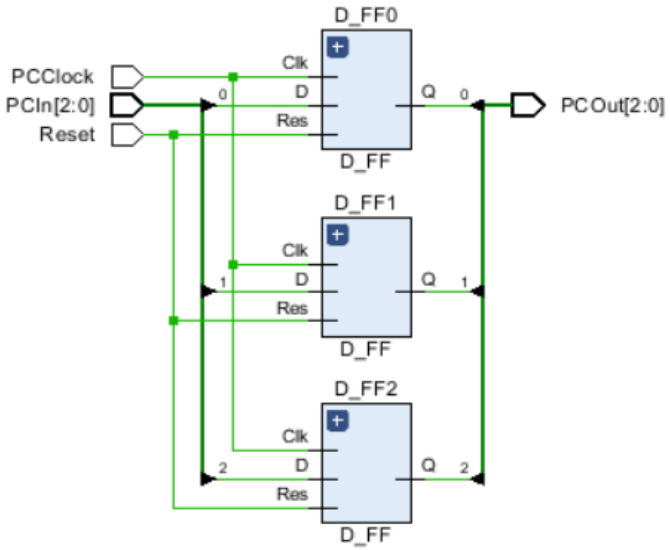
PROGRAM COUNTER

DESIGN SOURCE FILE – PROGRAM COUNTER

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


entity PC is
    Port ( PCIn : in STD_LOGIC_VECTOR (2 downto 0);
           PCOut : out STD_LOGIC_VECTOR (2 downto 0);
           Reset : in STD_LOGIC;
           PCClock : in STD_LOGIC);
end PC;
```

```
architecture Behavioral of PC is

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

end component;

begin

D_FF0 : D_FF
    port map ( D => PCIn(0),
               Res => Reset,
               Clk => PCClock,
               Q => PCOut(0));
    D_FF1 : D_FF
    port map ( D => PCIn(1),
               Res => Reset,
               Clk => PCClock,
               Q => PCOut(1));
    D_FF2 : D_FF
    port map ( D => PCIn(2),
               Res => Reset,
               Clk => PCClock,
               Q => PCOut(2));

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – PROGRAM COUNTER

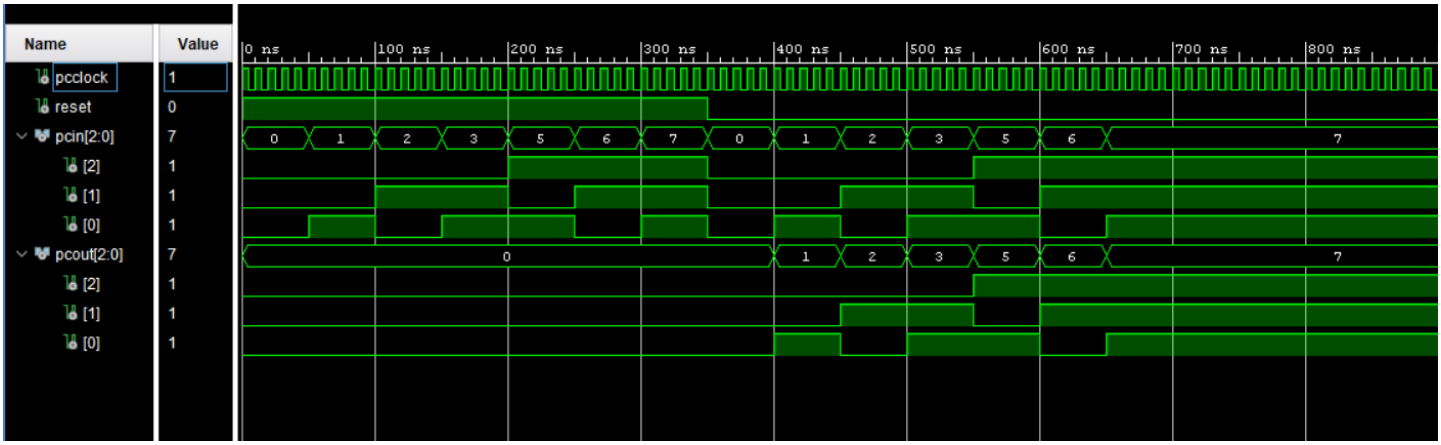


SIMULATION SOURCE FILE – PROGRAM COUNTER

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

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

architecture Behavioral of PCSim is

component PC
port ( PCIn : in STD_LOGIC_VECTOR (2 downto 0) ;
       PCClock : in STD_LOGIC;
       Reset : in STD_LOGIC;
       PCOut : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal pcclock : std_logic:='0';
signal reset : std_logic;
signal pcin : std_logic_vector(2 downto 0) := "000";
signal pcout : std_logic_vector(2 downto 0);


begin

UUT:PC
port map(
PCClock=>pcclock,
Reset=>reset,
PCOut=>pcout,
PCIn=>pcin
);

process begin
pcclock <= not(pcclock);
wait for 5ns;
end process;

process begin
reset<='1';
pcin<="000";
wait for 50ns;
pcin<="001";
wait for 50ns;
```

```
pcin<="010";
wait for 50ns;
pcin<="011";
wait for 50ns;
pcin<="101";
wait for 50ns;
pcin<="110";
wait for 50ns;
pcin<="111";
wait for 50ns;

reset<='0';
pcin<="000";
wait for 50ns;
pcin<="001";
wait for 50ns;
pcin<="010";
wait for 50ns;
pcin<="011";
wait for 50ns;
pcin<="101";
wait for 50ns;
pcin<="110";
wait for 50ns;
pcin<="111";
wait;
end process;

end Behavioral;
```

TIMING DIAGRAM – PROGRAM COUNTER

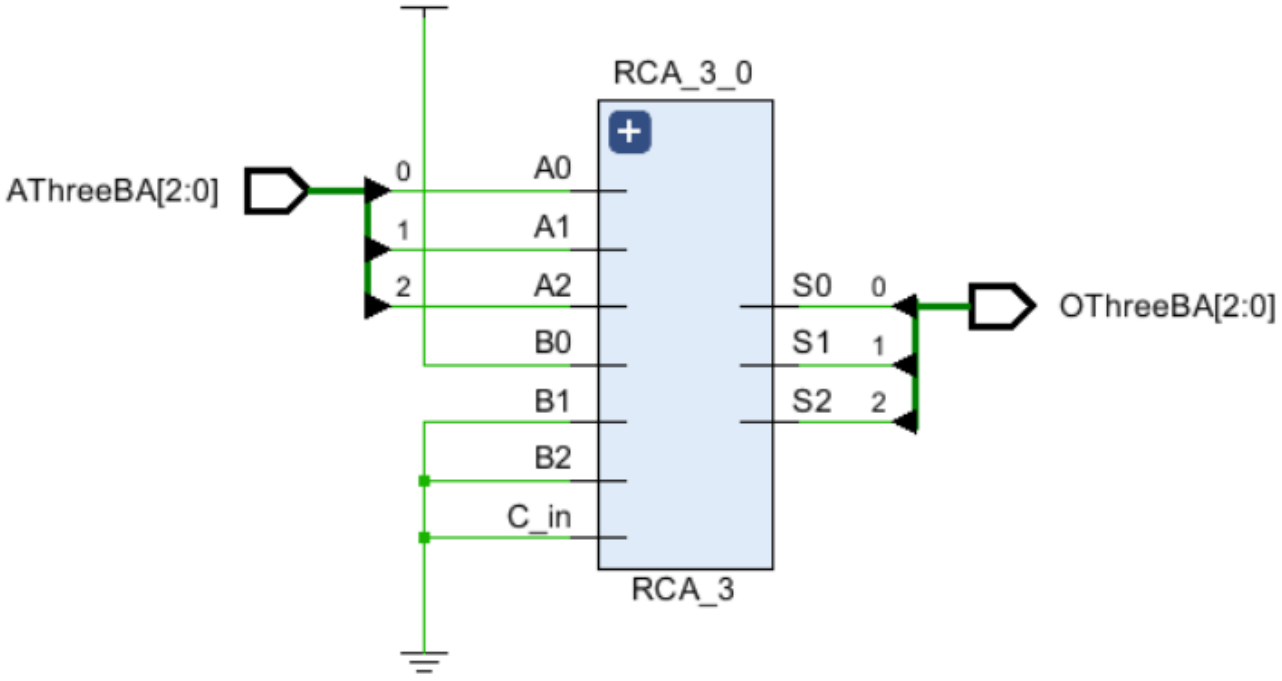


3 BIT ADDER

DESIGN SOURCE FILE – 3 BIT ADDER

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

entity ThreeBA is
    Port ( AThreeBA : in STD_LOGIC_VECTOR (2 downto 0);
           OThreeBA : out STD_LOGIC_VECTOR (2 downto 0));
end ThreeBA;

architecture Behavioral of ThreeBA is
component RCA_3 is
    Port ( A0 : in STD_LOGIC;
           A1 : in STD_LOGIC;
           A2 : in STD_LOGIC;
           B0 : in STD_LOGIC;
           B1 : in STD_LOGIC;
           B2 : in STD_LOGIC;
           C_in : in STD_LOGIC;
           S0 : out STD_LOGIC;
           S1 : out STD_LOGIC;
           S2 : out STD_LOGIC;
           C_out : out STD_LOGIC);
end component;

signal cout:std_logic;
begin

 RCA_3_0 : RCA_3
 port map( A0 => AThreeBA(0),
           A1 =>AThreeBA(1),
           A2 =>AThreeBA(2),

           B0 =>'1',
           B1 =>'0',
           B2 =>'0',
           C_in =>'0',
           S0 =>OThreeBA(0),
           S1 =>OThreeBA(1),
           S2 =>OThreeBA(2),
           C_out => cout );
end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – 3 BIT ADDER

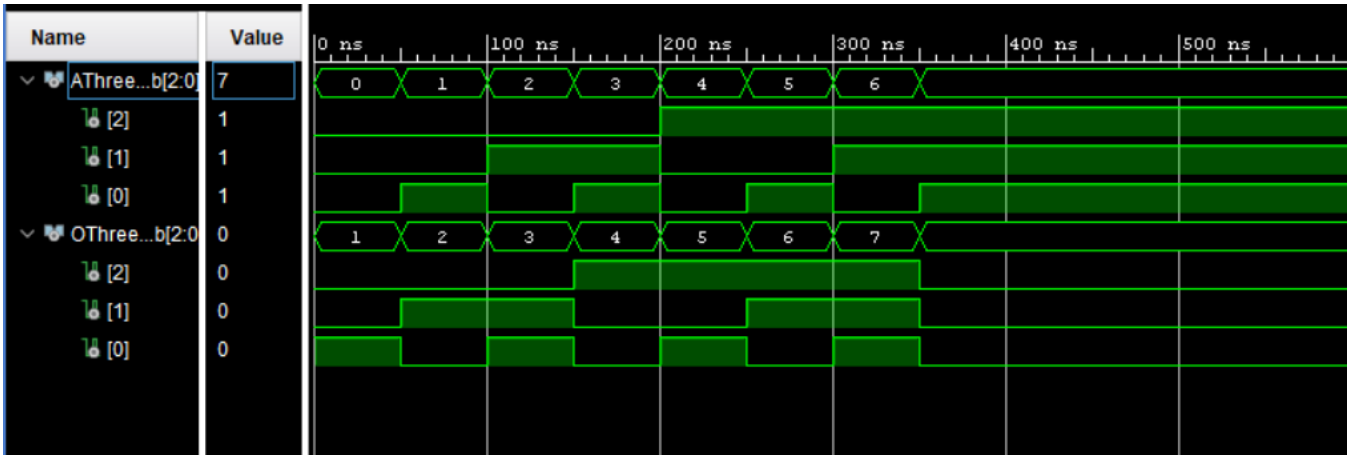


SIMULATION SOURCE FILE – 3 BIT ADDER

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

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

architecture Behavioral of ThreeBASim is
component ThreeBA is
        Port ( AThreeBA : in STD_LOGIC_VECTOR (2 downto 0);
               OThreeBA : out STD_LOGIC_VECTOR (2 downto 0));
    end component;

    -- Signals for Testbench
    signal AThreeBA_tb : STD_LOGIC_VECTOR (2 downto 0);
    signal OThreeBA_tb : STD_LOGIC_VECTOR (2 downto 0);

begin
    -- Instantiate the Unit Under Test (UUT)
    UUT: ThreeBA
        port map (AThreeBA => AThreeBA_tb,
                  OThreeBA => OThreeBA_tb);
```

```
    -- Stimulus Process
    process
    begin
        -- Test Case: Add 001 to 000
        AThreeBA_tb <= "000"; -- Input 000
        wait for 50 ns;

        -- Test Case: Add 001 to 001
        AThreeBA_tb <= "001"; -- Input 001
        wait for 50 ns;

        -- Test Case: Add 001 to 010
        AThreeBA_tb <= "010"; -- Input 010
        wait for 50 ns;

        -- Test Case: Add 001 to 011
        AThreeBA_tb <= "011"; -- Input 011
        wait for 50 ns;

        -- Test Case: Add 001 to 100
        AThreeBA_tb <= "100"; -- Input 100
        wait for 50 ns;

        -- Test Case: Add 001 to 101
        AThreeBA_tb <= "101"; -- Input 101
        wait for 50 ns;

        -- Test Case: Add 001 to 110
        AThreeBA_tb <= "110"; -- Input 110
        wait for 50 ns;

        -- Test Case: Add 001 to 111
        AThreeBA_tb <= "111"; -- Input 111
        wait for 50 ns;

        -- End Simulation
        wait;
    end process;

end Behavioral;
```

TIMING DIAGRAM – 3 BIT ADDER



REGISTER BANK

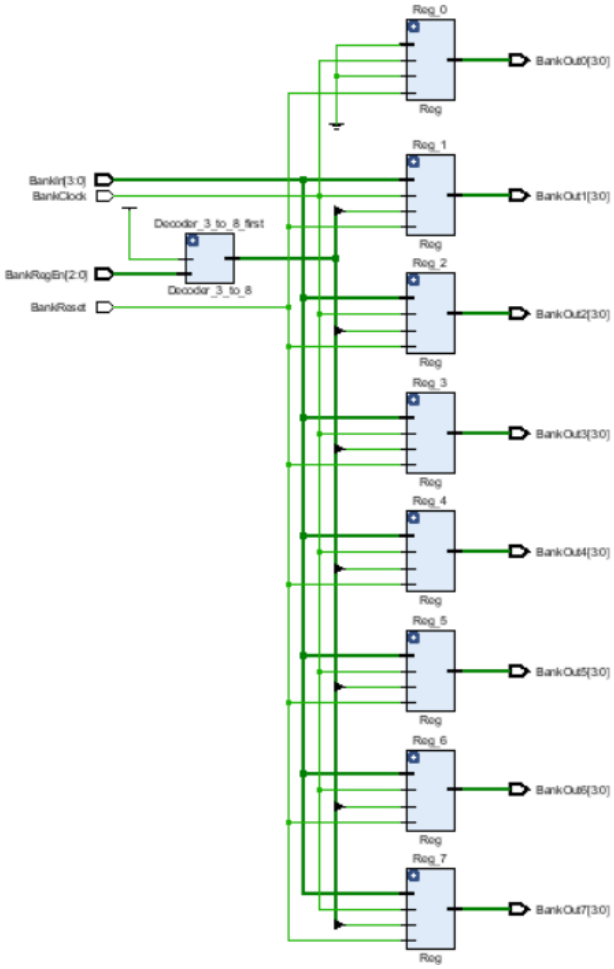
DESIGN SOURCE FILE – REGISTER BANK

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

entity RegisterBank is
    Port ( BankIn : in STD_LOGIC_VECTOR (3 downto 0);
           BankClock : in STD_LOGIC;
           BankReset : in STD_LOGIC;
           BankRegEn : in STD_LOGIC_VECTOR (2 downto 0);
           BankOut0 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut1 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut2 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut3 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut4 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut5 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut6 : out STD_LOGIC_VECTOR (3 downto 0);
           BankOut7 : out STD_LOGIC_VECTOR (3 downto 0));
end RegisterBank;

architecture Behavioral of RegisterBank is
component Reg is
     Port ( I : in STD_LOGIC_VECTOR (3 downto 0);
          RegClock : in STD_LOGIC;
```

```
          RegEn : in STD_LOGIC;
          RegReset : in STD_LOGIC;
          Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Decoder_3_to_8 is
    Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end component;

signal en : std_logic_vector(7 downto 0);
begin

Decoder_3_to_8_first : Decoder_3_to_8
port map (
           I => BankRegEn,
           EN => '1',
           Y => en
);


Reg_0 : Reg
port map(
     I =>"0000" ,
     RegClock=> BankClock ,
     RegEn => '0',
     RegReset => BankReset ,
     Y => BankOut0
);

Reg_1 : Reg
port map(
     I => BankIn,
     RegClock=>BankClock,
     RegEn => en(1),
     RegReset =>BankReset ,
     Y => BankOut1);

Reg_2 : Reg
port map(
     I =>BankIn,
```

```
     RegClock=> BankClock,
     RegEn => en(2),
     RegReset =>BankReset ,
     Y => BankOut2);

Reg_3 : Reg
port map(
     I => BankIn,
     RegClock=>BankClock ,
     RegEn => en(3),
     RegReset => BankReset,
     Y => BankOut3);
Reg_4 : Reg
port map(
     I => BankIn,
     RegClock=>BankClock ,
     RegEn => en(4),
     RegReset =>BankReset ,
     Y => BankOut4);
Reg_5 : Reg
port map(
     I => BankIn,
     RegClock=>BankClock ,
     RegEn => en(5),
     RegReset =>BankReset ,
     Y => BankOut5);
Reg_6 : Reg
port map(
     I => BankIn,
     RegClock=> BankClock,
     RegEn => en(6),
     RegReset =>BankReset ,
     Y => BankOut6);
Reg_7 : Reg
port map(
     I => BankIn ,
     RegClock=>BankClock ,
     RegEn => en(7),
     RegReset => BankReset,
     Y => BankOut7);

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – REGISTER BANK



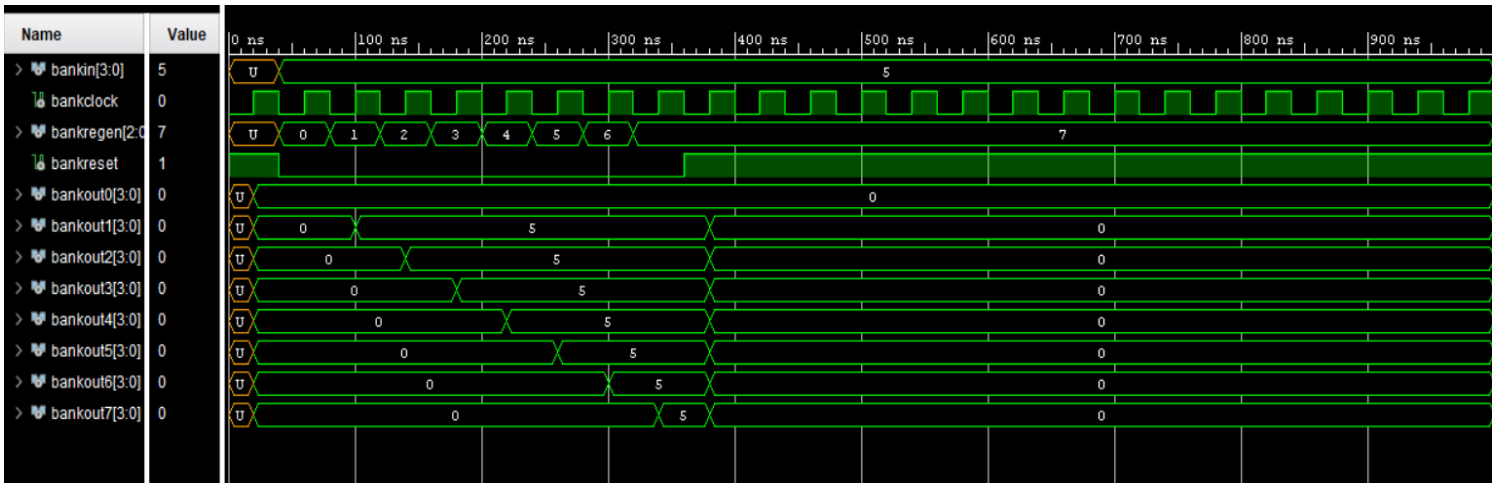
SIMULATION SOURCE FILE – REGISTER BANK

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

-- Uncomment the following library declaration if using
entity RegisterBankSim is
--  Port ( );
end RegisterBankSim;

architecture Behavioral of RegisterBankSim is

component RegisterBank is
   port ( BankIn : in STD_LOGIC_VECTOR (3 downto 0);
          BankClock : in STD_LOGIC;
          BankReset : in STD_LOGIC;
          BankRegEn : in STD_LOGIC_VECTOR (2 downto 0);
          BankOut0 : out STD_LOGIC_VECTOR (3 downto 0);
          BankOut1 : out STD_LOGIC_VECTOR (3 downto 0);
```

```
            BankOut2 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut3 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut4 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut5 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut6 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal bankin : std_logic_vector(3 downto 0);
signal bankclock : std_logic := '0';  -- Clock signal
signal bankregen : std_logic_vector(2 downto 0);-- Memory Selector
signal bankreset : std_logic; -- Reset signal
signal  bankout0, bankout1,bankout2,bankout3,bankout4,bankout5,bankout6,bankout7 :
std_logic_vector(3 downto 0);


begin

UUT  : RegisterBank port map(
  BankIn => bankin,
  BankClock => bankclock,
  BankReset => bankreset,
  BankRegEN => bankregen,
  BankOut0 => bankout0,
  BankOut1 => bankout1,
  BankOut2 => bankout2,
  BankOut3 => bankout3,
  BankOut4 => bankout4,
  BankOut5 => bankout5,
  BankOut6 => bankout6,
  BankOut7 => bankout7);

process begin

Wait for 20ns;
bankclock<=NOT(bankclock);
end process;

process begin -- Reset activated
bankreset <= '1';
wait for 40 ns;

bankreset <= '0'; -- Select different registers
```

```
bankin <= "0101";

bankregen <= "000"; -- Select R0
wait for 40 ns;

bankregen <= "001"; -- Select R1
wait for 40 ns;

bankregen <= "010"; -- Select R2
wait for 40 ns;

bankregen<= "011"; -- Select R3
wait for 40 ns;

bankregen <= "100"; -- Select R4
wait for 40 ns;

bankregen <= "101"; -- Select R5
wait for 40 ns;

bankregen <= "110"; -- Select R6
wait for 40 ns;

bankregen <= "111"; -- Select R7
wait for 40ns;

bankreset <= '1';
wait ;

end process;
end Behavioral;
```

TIMING DIAGRAM – REGISTER BANK



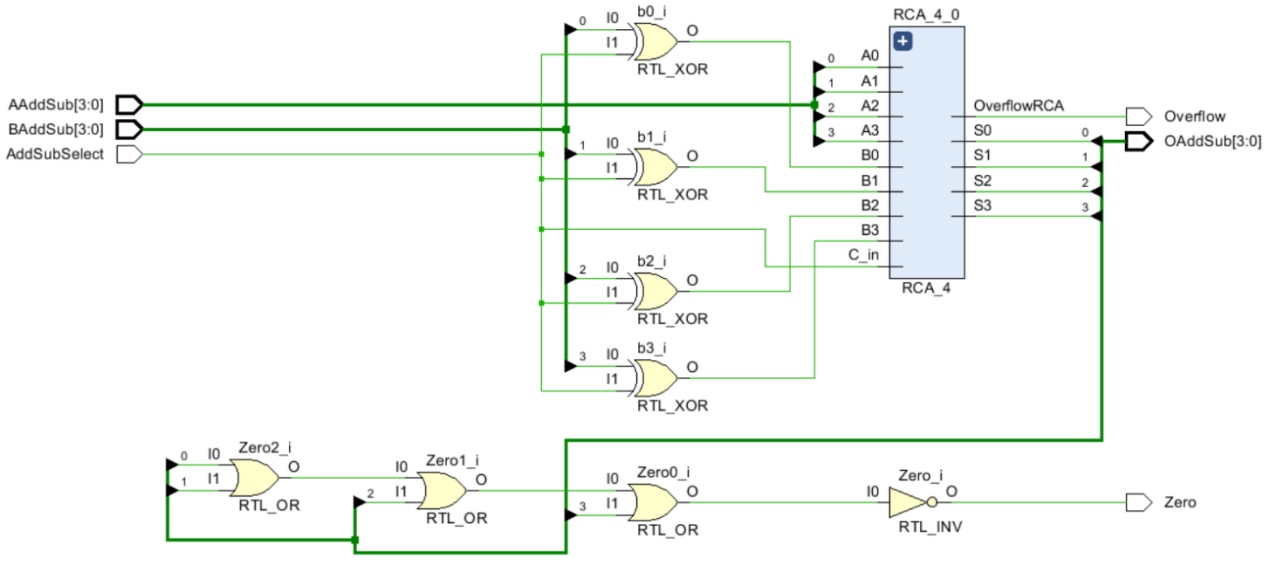
4 BIT ADD/SUB UNIT

DESIGN SOURCE FILE - 4 BIT ADD/SUB UNIT

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

entity ASU is
    Port ( AAddSub : in STD_LOGIC_VECTOR (3 downto 0);
           BAddSub : in STD_LOGIC_VECTOR (3 downto 0);
           AddSubSelect : in STD_LOGIC;
           OAddSub : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC);
end ASU;

architecture Behavioral of ASU is

component RCA_4 is
    Port ( A0 : in STD_LOGIC;
           A1 : in STD_LOGIC;
           A2 : in STD_LOGIC;
           A3 : in STD_LOGIC;
           B0 : in STD_LOGIC;
           B1 : in STD_LOGIC;
           B2 : in STD_LOGIC;
           B3 : in STD_LOGIC;
           C_in : in STD_LOGIC;
           S0 : out STD_LOGIC;
           S1 : out STD_LOGIC;
           S2 : out STD_LOGIC;
           S3 : out STD_LOGIC;
           C_out : out STD_LOGIC;
           OverflowRCA : out STD_LOGIC);
end component;
signal b0, b1, b2, b3 :std_logic;
signal s0, s1, s2, s3 :std_logic;
signal cout, overflowRCA : std_logic;

begin
b0<= BAddSub(0) XOR AddSubSelect;
b1<= BAddSub(1) XOR AddSubSelect;
b2<= BAddSub(2) XOR AddSubSelect;
b3<= BAddSub(3) XOR AddSubSelect;
```

```
RCA_4_0 : RCA_4
port map( A0 => AAddSub(0),
          A1 => AAddSub(1),
          A2 => AAddSub(2),
          A3 => AAddSub(3),
          B0 => b0,
          B1 => b1,
          B2 => b2,
          B3 => b3,
          C_in => AddSubSelect,
          S0  => s0,
          S1  => s1,
          S2  => s2,
          S3  => s3,
          C_out => cout,
          OverflowRCA => overflowrca);

-- Zero Flag  ---check whether normal meyhod have less gates
Zero <= not( s0 or s1 or s2 or  s3);

-- Overflow Flag
Overflow <= overflowrca;

OAddSub(0)<= s0;
OAddSub(1)<= s1;
OAddSub(2)<= s2;
OAddSub(3)<= s3;

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – 4 BIT ADD/SUB

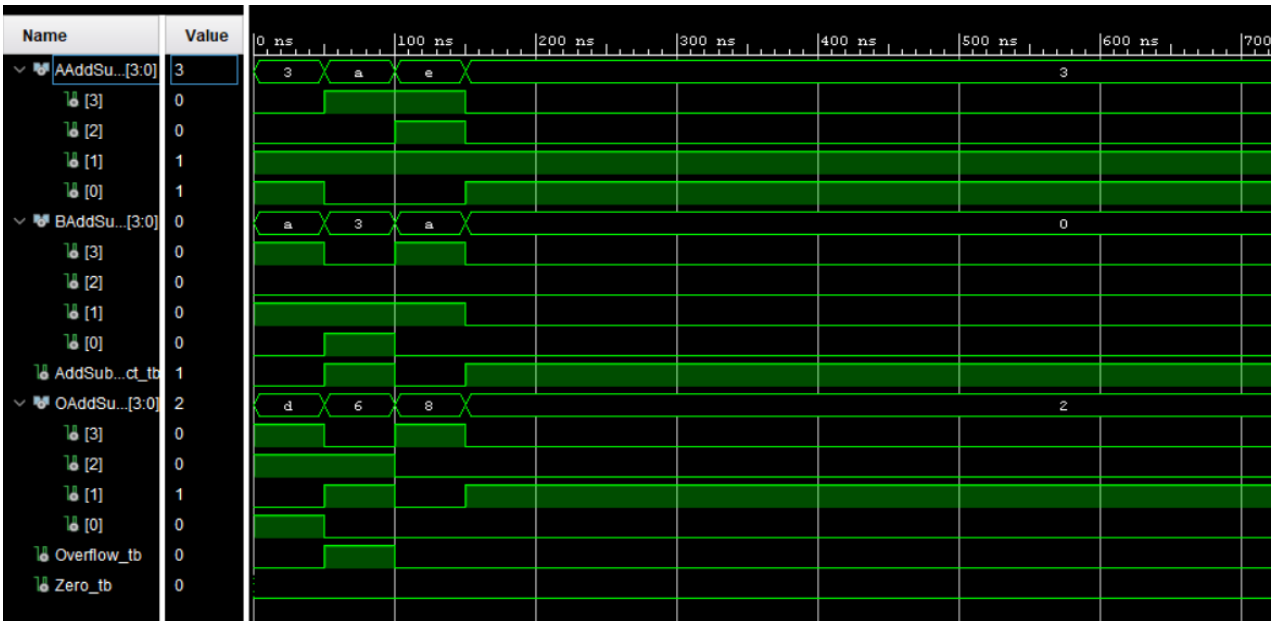


SIMULATION SOURCE FILE – 4 BIT ADD/SUB

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

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

architecture Behavioral of ASU_Testbench is

    -- Component Declaration
    component ASU is
        Port ( AAddSub : in STD_LOGIC_VECTOR (3 downto 0);
               BAddSub : in STD_LOGIC_VECTOR (3 downto 0);
               AddSubSelect : in STD_LOGIC;
               OAddSub : out STD_LOGIC_VECTOR (3 downto 0);
               Overflow : out STD_LOGIC;
               Zero : out STD_LOGIC);
    end component;

    -- Signals for Inputs and Outputs
    signal AAddSub_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal BAddSub_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal AddSubSelect_tb : STD_LOGIC;
    signal OAddSub_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal Overflow_tb : STD_LOGIC;
    signal Zero_tb : STD_LOGIC;

begin

    -- Instantiate the Unit Under Test (UUT)
    UUT : ASU
        port map (
            AAddSub => AAddSub_tb,
            BAddSub => BAddSub_tb,
            AddSubSelect => AddSubSelect_tb,
            OAddSub => OAddSub_tb,
            Overflow => Overflow_tb,
            Zero => Zero_tb);

    -- Stimulus Process
    Stimulus_Process : process
```

```
    Begin
        -- Test 1: Add 0011 and 1010
        AAddSub_tb <= "0011"; -- 3
        BAddSub_tb <= "1010"; -- -6
        AddSubSelect_tb <= '0'; -- Addition mode
        wait for 50 ns;

        -- Test 2: Subtract 0011 from 1010
        AAddSub_tb <= "1010"; -- -6
        BAddSub_tb <= "0011"; -- 3
        AddSubSelect_tb <= '1'; -- Subtraction mode
        wait for 50 ns;

        -- Test 3: Input a random vector 1110 0000 1100 1110 1010
        -- To test, feed smaller 4-bit segments sequentially
        AAddSub_tb <= "1110"; -- -2
        BAddSub_tb <= "1010"; -- -6
        AddSubSelect_tb <= '0'; -- Addition mode
        wait for 50 ns;

        -- Test 4: negate 0011
                AAddSub_tb <= "0011"; -- -6
                BAddSub_tb <= "0000"; -- 0 from Register 0
                AddSubSelect_tb <= '1'; -- Subtraction mode
                wait for 50 ns;

        -- End simulation
        wait;
    end process;

end Behavioral;
```

TIMING DIAGRAM - 4 BIT ADD/SUB UNIT

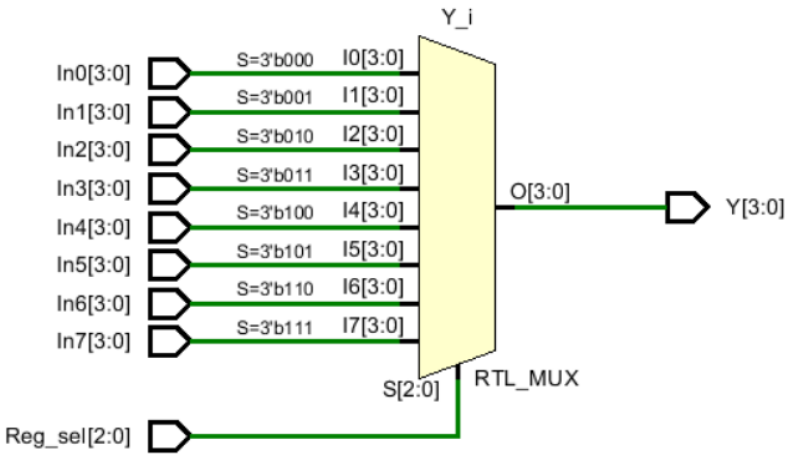


8-WAY 4-BIT MULTIPLEXER

DESIGN SOURCE FILE - 8-WAY 4-BIT MULTIPLEXER

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

entity Eight_way_4_bit_MUX is
    Port (
        Reg_00     : in  STD_LOGIC_VECTOR(3 downto 0);
        Reg_01     : in  STD_LOGIC_VECTOR(3 downto 0);
        Reg_02     : in  STD_LOGIC_VECTOR(3 downto 0);
        Reg_03     : in  STD_LOGIC_VECTOR(3 downto 0);
        Reg_04     : in  STD_LOGIC_VECTOR(3 downto 0);
        Reg_05     : in  STD_LOGIC_VECTOR(3 downto 0);
        Reg_06     : in  STD_LOGIC_VECTOR(3 downto 0);
        Reg_07     : in  STD_LOGIC_VECTOR(3 downto 0);
        Reg_sel    : in  STD_LOGIC_VECTOR(2 downto 0);
        Selected_reg : out STD_LOGIC_VECTOR(3 downto 0)
    );
end Eight_way_4_bit_MUX;

architecture Behavioral of Eight_way_4_bit_MUX is
begin
    process (Reg_sel, Reg_00, Reg_01, Reg_02, Reg_03, Reg_04, Reg_05, Reg_06,
Reg_07)
    begin
        case Reg_sel is
            when "000" => Selected_reg <= Reg_00;
            when "001" => Selected_reg <= Reg_01;
            when "010" => Selected_reg <= Reg_02;
            when "011" => Selected_reg <= Reg_03;
            when "100" => Selected_reg <= Reg_04;
            when "101" => Selected_reg <= Reg_05;
            when "110" => Selected_reg <= Reg_06;
            when "111" => Selected_reg <= Reg_07;
            when others => Selected_reg <= (others => '0'); -- Default case
        end case;
    end process;
end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – 8-WAY 4-BIT MULTIPLEXER

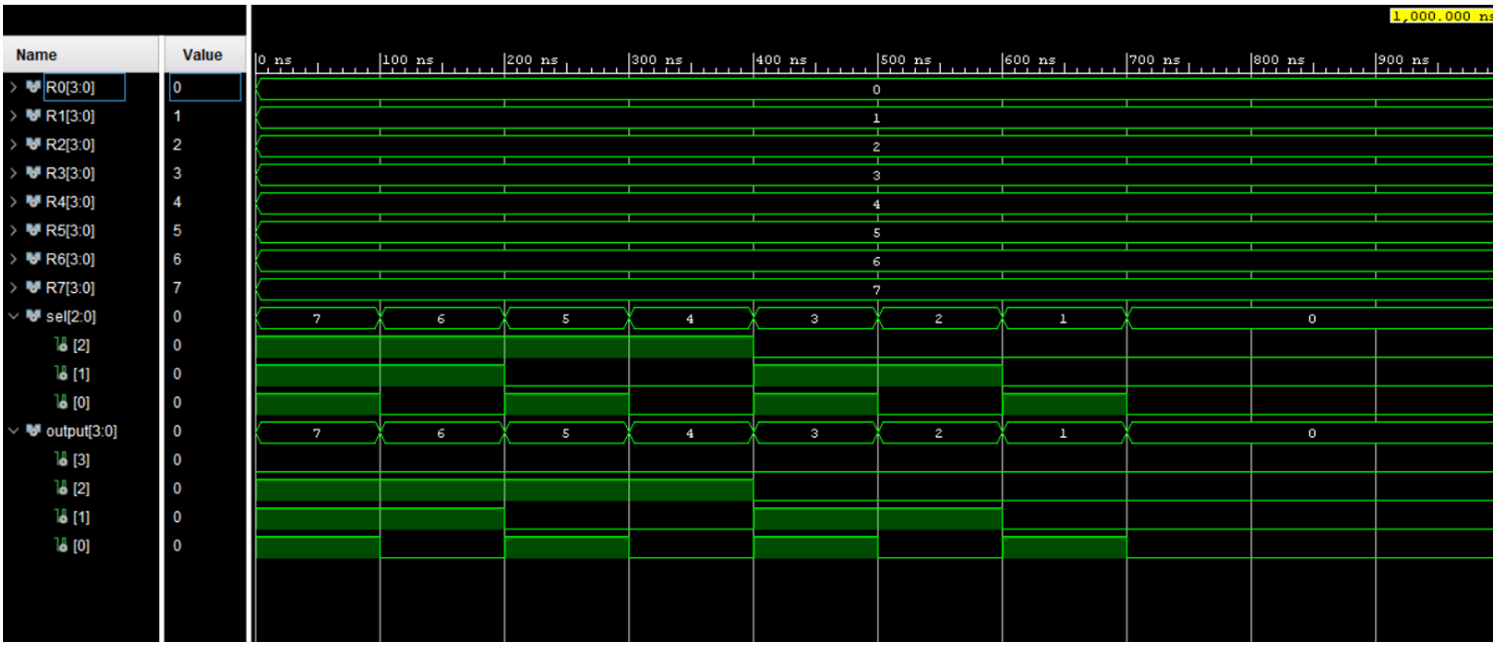


SIMULATION SOURCE FILE – 8-WAY 4-BIT MULTIPLEXER

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

entity Eight_way_4_bit_MUX_TB is
end Eight_way_4_bit_MUX_TB;

architecture Behavioral of Eight_way_4_bit_MUX_TB is

    component Eight_way_4_bit_MUX
        Port (
            Reg_00       : in  STD_LOGIC_VECTOR(3 downto 0);
            Reg_01       : in  STD_LOGIC_VECTOR(3 downto 0);
            Reg_02       : in  STD_LOGIC_VECTOR(3 downto 0);
            Reg_03       : in  STD_LOGIC_VECTOR(3 downto 0);
            Reg_04       : in  STD_LOGIC_VECTOR(3 downto 0);
            Reg_05       : in  STD_LOGIC_VECTOR(3 downto 0);
            Reg_06       : in  STD_LOGIC_VECTOR(3 downto 0);
            Reg_07       : in  STD_LOGIC_VECTOR(3 downto 0);
            Reg_sel      : in  STD_LOGIC_VECTOR(2 downto 0);
            Selected_reg : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    signal R0, R1, R2, R3, R4, R5, R6, R7 : STD_LOGIC_VECTOR(3 downto 0);
    signal sel                            : STD_LOGIC_VECTOR(2 downto 0);
    signal output                         : STD_LOGIC_VECTOR(3 downto 0);
```

```
begin

    UUT: Eight_way_4_bit_MUX
        port map (
            Reg_00       => R0,
            Reg_01       => R1,
            Reg_02       => R2,
            Reg_03       => R3,
            Reg_04       => R4,
            Reg_05       => R5,
            Reg_06       => R6,
            Reg_07       => R7,
            Reg_sel      => sel,
            Selected_reg => output
        );

    stimulus_process: process
    begin
        -- Assign unique values to each input
        R0 <= "0000";
        R1 <= "0001";
        R2 <= "0010";
        R3 <= "0011";
        R4 <= "0100";
        R5 <= "0101";
        R6 <= "0110";
        R7 <= "0111";

        -- Test selection of each input
        sel <= "111"; wait for 100 ns;  -- Select R7
        sel <= "110"; wait for 100 ns;  -- Select R6
        sel <= "101"; wait for 100 ns;  -- Select R5
        sel <= "100"; wait for 100 ns;  -- Select R4
        sel <= "011"; wait for 100 ns;  -- Select R3
        sel <= "010"; wait for 100 ns;  -- Select R2
        sel <= "001"; wait for 100 ns;  -- Select R1
        sel <= "000"; wait for 100 ns;  -- Select R0

        wait;
    end process;

end Behavioral;
```

TIMING DIAGRAM - 8-WAY 4-BIT MULTIPLEXER

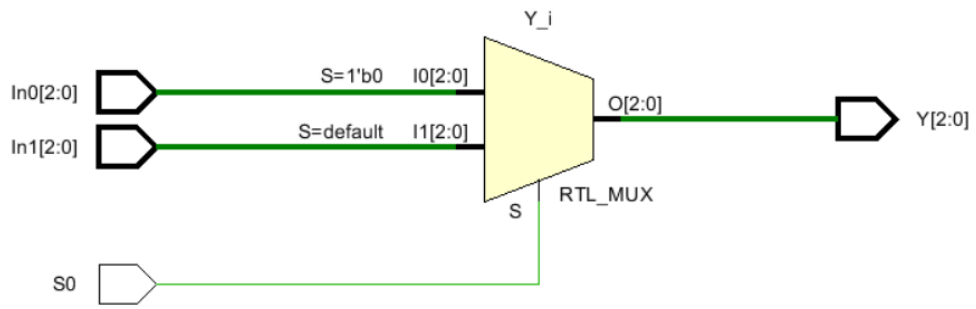


2-WAY 3-BIT MULTIPLEXER

DESIGN SOURCE FILE - 2-WAY 3-BIT MULTIPLEXER

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

entity TwoWay3Bit_MUX is
    Port ( In0 : in STD_LOGIC_VECTOR (2 downto 0); --take this(In0) as input to MUX
from 3 bit addder
           In1 : in STD_LOGIC_VECTOR (2 downto 0); -- take this as the JUMP ADDRESS
input to MUX
           S0 : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (2 downto 0));
end TwoWay3Bit_MUX;

architecture Behavioral of TwoWay3Bit_MUX is

begin

 Y <= In0 when S0 = '0' else In1;

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – 2-WAY 3-BIT MULTIPLEXER

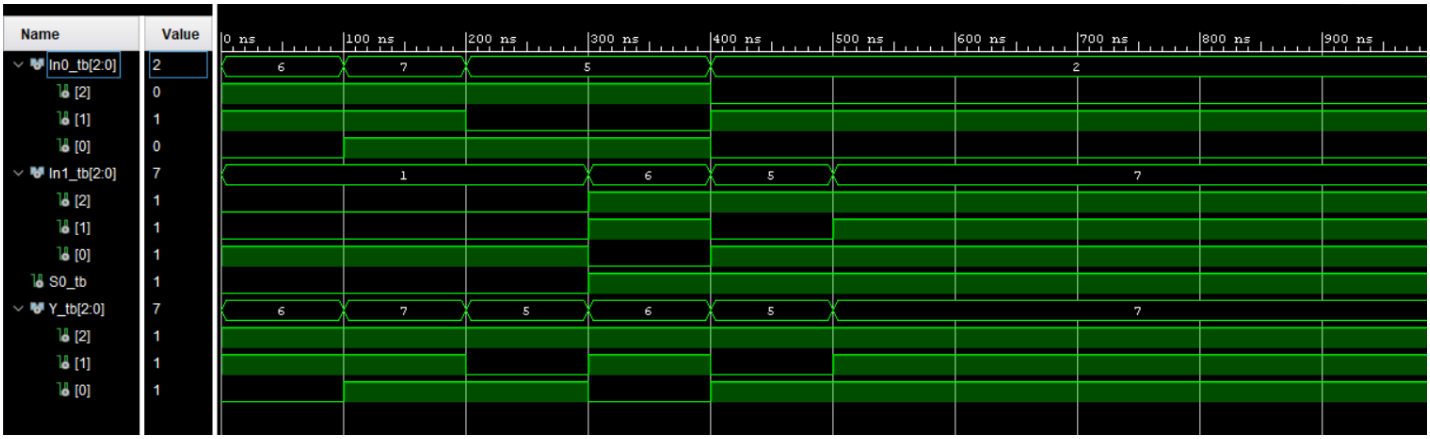


SIMULATION SOURCE FILE – 2-WAY 3-BIT MULTIPLEXER

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

entity TwoWay3Bit_MUXSim is
-- Testbench has no ports
end TwoWay3Bit_MUXSim;

architecture Behavioral of TwoWay3Bit_MUXSim is

    -- Component declaration for the Unit Under Test (UUT)
    component TwoWay3Bit_MUX
        Port (
            In0 : in STD_LOGIC_VECTOR (2 downto 0);
            In1 : in STD_LOGIC_VECTOR (2 downto 0);
            S0 : in STD_LOGIC;
            Y : out STD_LOGIC_VECTOR (2 downto 0)
        );
    end component;

    -- Signals for the testbench
    signal In0_tb : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal In1_tb : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal S0_tb : STD_LOGIC := '0';
    signal Y_tb : STD_LOGIC_VECTOR(2 downto 0);

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: TwoWay3Bit_MUX
        Port map (
```

```
            In0 => In0_tb,
            In1 => In1_tb,
            S0 => S0_tb,
            Y => Y_tb
        );

    -- Test process
    stim_proc: process
    begin
        -- Test case 1: S0 = '0', expect Y = In0
        In0_tb <= "110";  -- Example value
        In1_tb <= "001";  -- Example value
        S0_tb <= '0';
        wait for 100 ns;

        In0_tb <= "111";
        In1_tb <= "001";
        S0_tb <= '0';
        wait for 100 ns;

        In0_tb <= "101";
        In1_tb <= "001";
        S0_tb <= '0';
        wait for 100 ns;


        In0_tb <= "101";
        In1_tb <= "110";
        S0_tb <= '1';
        wait for 100 ns;

        In0_tb <= "010";
        In1_tb <= "101";
        S0_tb <= '1';
        wait for 100 ns;

        In0_tb <= "010";
        In1_tb <= "111";
        S0_tb <= '1';
        wait for 100 ns;

        wait;
    end process;
end Behavioral;
```

TIMING DIAGRAM - 2-WAY 3-BIT MULTIPLEXER

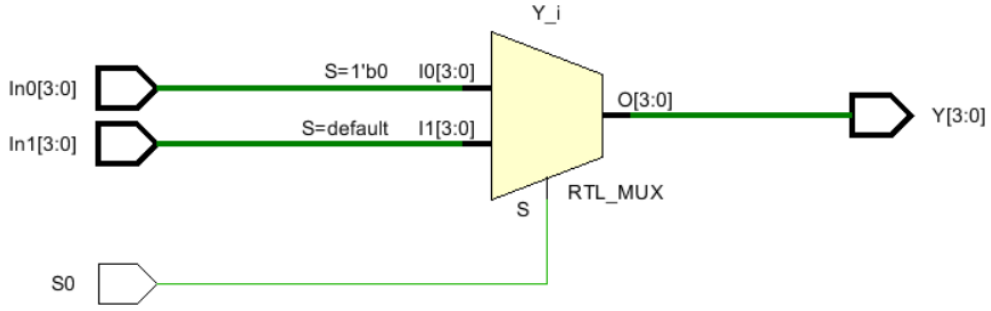


2-WAY 4-BIT MULTIPLEXER

DESIGN SOURCE FILE - 2-WAY 4-BIT MULTIPLEXER

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

entity TwoWay4Bit_Mux is
    Port ( In0 : in STD_LOGIC_VECTOR (3 downto 0); -- Output from ADD SUB Unit
           In1 : in STD_LOGIC_VECTOR (3 downto 0); -- Immediate value
           S0 : in STD_LOGIC;      -- load select
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end TwoWay4Bit_Mux;

architecture Behavioral of TwoWay4Bit_Mux is

begin

 Y <= In0 when S0 = '0' else In1;

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – 2-WAY 4-BIT MULTIPLEXER

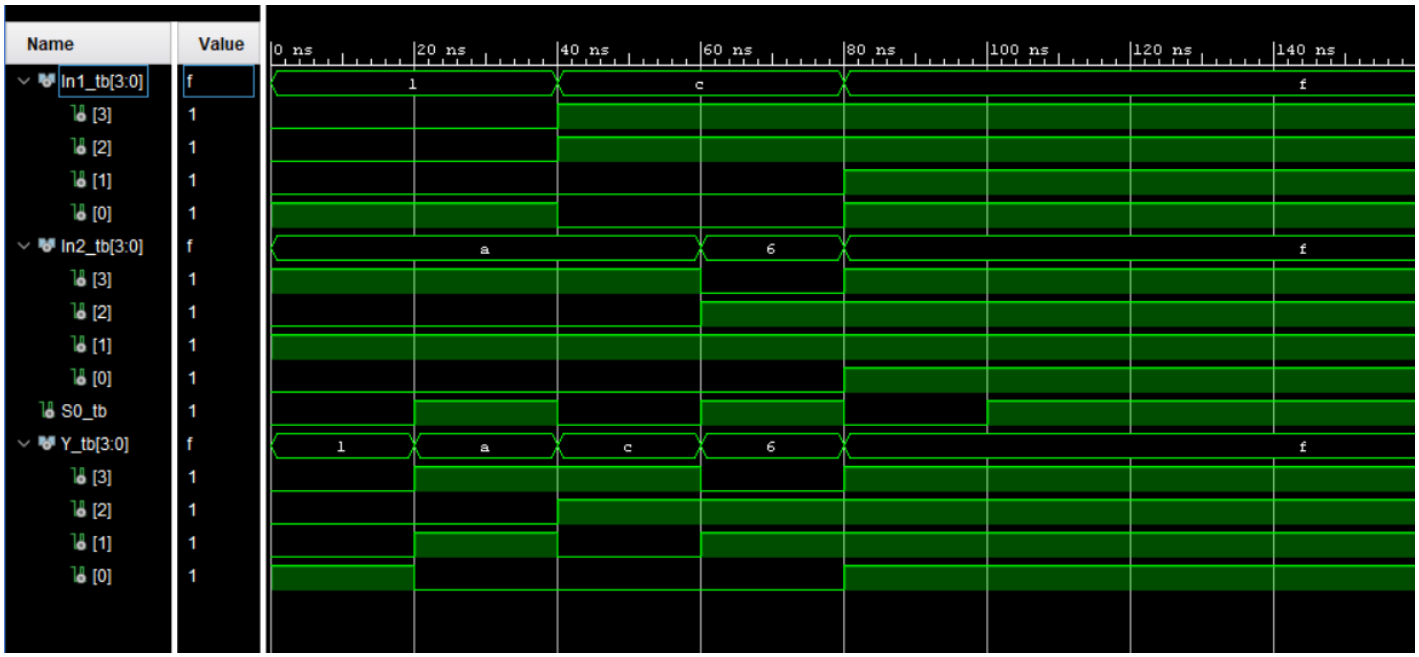


SIMULATION SOURCE FILE – 2-WAY 4-BIT MULTIPLEXER

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

entity TwoWay4Bit_Mux_Testbench is
end TwoWay4Bit_Mux_Testbench;

architecture Behavioral of TwoWay4Bit_Mux_Testbench is
    -- Component Declaration for the Unit Under Test (UUT)
    component TwoWay4Bit_Mux is
        Port (
            In0 : in STD_LOGIC_VECTOR (3 downto 0);
            In1 : in STD_LOGIC_VECTOR (3 downto 0);
            S0  : in STD_LOGIC;
            Y   : out STD_LOGIC_VECTOR (3 downto 0)
        );
    end component;

    -- Signals for the testbench
    signal In1_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal In2_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal S0_tb  : STD_LOGIC;
    signal Y_tb   : STD_LOGIC_VECTOR (3 downto 0);

begin
    -- Instantiate the Unit Under Test (UUT)
    UUT: TwoWay4Bit_Mux
        port map (
            In0 => In1_tb,
            In1 => In2_tb,
            S0  => S0_tb,
            Y   => Y_tb
        );

    -- Stimulus Process
    process
    begin
        -- Test Case 1: S0 = '0', Y should equal In1
        In1_tb <= "0001";
        In2_tb <= "1010";
        S0_tb <= '0';
        wait for 20 ns;

        -- Test Case 2: S0 = '1', Y should equal In2
```

```
        S0_tb <= '1';
        wait for 20 ns;

        -- Test Case 3: Change In1 and verify S0 = '0'
        In1_tb <= "1100";
        S0_tb <= '0';
        wait for 20 ns;

        -- Test Case 4: Change In2 and verify S0 = '1'
        In2_tb <= "0110";
        S0_tb <= '1';
        wait for 20 ns;

        -- Test Case 5: Both inputs the same, verify behavior
        In1_tb <= "1111";
        In2_tb <= "1111";
        S0_tb <= '0';
        wait for 20 ns;
        S0_tb <= '1';
        wait for 20 ns;

        -- End Simulation
        wait;
    end process;

end Behavioral;
```

TIMING DIAGRAM - 2-WAY 4-BIT MULTIPLEXER

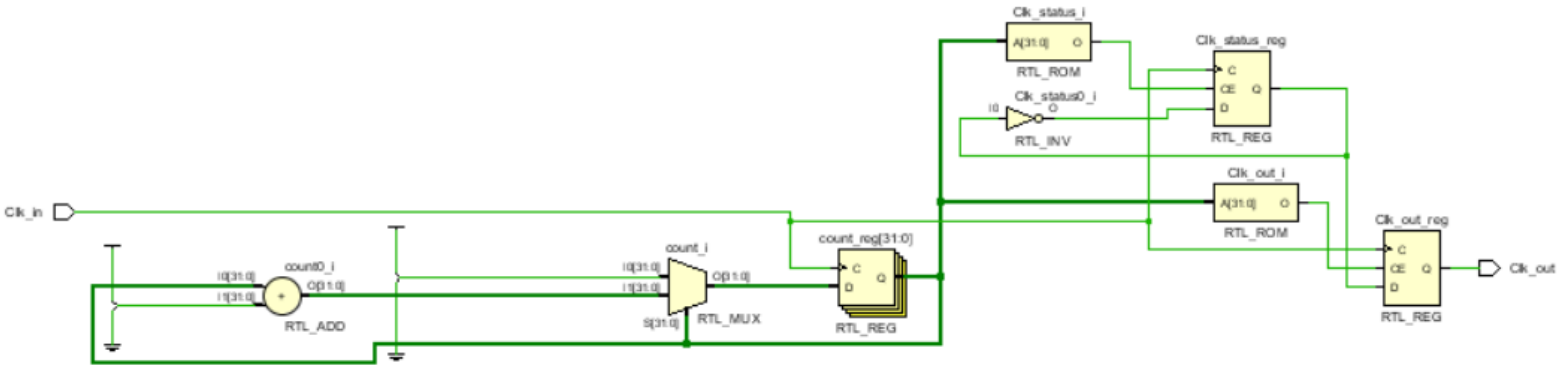


SLOW CLOCK

DESIGN SOURCE FILE – 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 = 50000000) then
         Clk_status <= NOT (Clk_status);
         Clk_out <= Clk_status ;
         count <= 1;
    end if;
 end if;
 end process;

 end Behavioral;
```

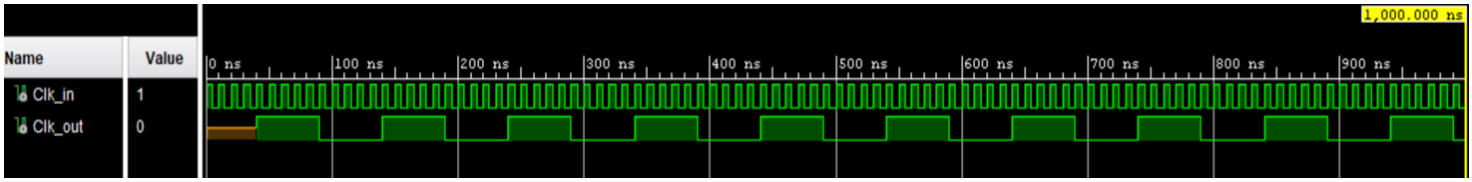
ELABORATED DESIGN SCHEMATIC – SLOW CLOCK



SIMULATION SOURCE FILE – SLOW CLOCK

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

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

architecture Behavioral of Slow_Clk_Sim is

COMPONENT Slow_Clk

PORT (Clk_in :IN STD_LOGIC := '0';
      Clk_out :OUT STD_LOGIC);

end COMPONENT;

signal Clk_in : STD_LOGIC := '0';
signal Clk_out : STD_LOGIC;

begin

clock_process: process
begin
      Clk_in <= NOT(Clk_in);
      wait for 5ns;
end process;

UUT: Slow_Clk PORT MAP(
      Clk_in => Clk_in,
      Clk_out => Clk_out );

end Behavioral;
```

TIMING DIAGRAM - SLOW CLOCK

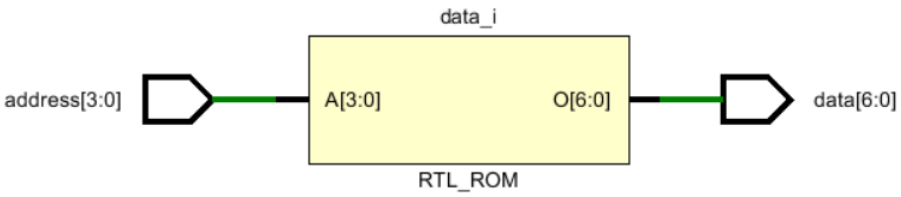


7 SEGMENT DISPLAY

DESIGN SOURCE FILE – 7 SEGMENT DISPLAY

```
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",
"0100100",
"0110000",
"0011001",
"0010010",
"0000010",
"1111000",
"0000000",
"0010000",
"0001000", -- a
"0000011",
"1000110",
"0100001",
"0000110",
"0001110"  -- f
);

begin

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

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – 7 SEGMENT DISPLAY

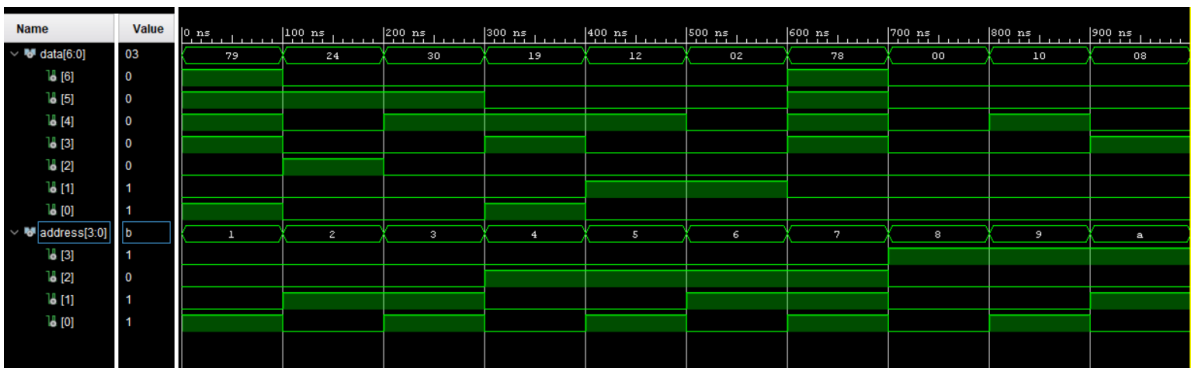


SIMULATION SOURCE FILE – 7 SEGMENT DISPLAY

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

entity LUT_Sim is
--  Port ( );
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 data : STD_LOGIC_VECTOR (6 downto 0);
signal address : STD_LOGIC_VECTOR (3 downto 0);

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

process
begin
 --1
 address<="0001";
 wait for 100ns;
 --2
 address<="0010";
 wait for 100ns;
 --3
 address<="0011";
 wait for 100ns;
  --04
```

```
 address<="0100";
 wait for 100ns;
 --5
 address<="0101";
 wait for 100ns;
 --6
 address<="0110";
 wait for 100ns;
 --7
 address<="0111";
 wait for 100ns;
 --8
 address<="1000";
 wait for 100ns;
 --9
 address<="1001";
 wait for 100ns;
 --A
 address<="1010";
 wait for 100ns;
 --B
 address<="1011";
 wait for 100ns;
 --C
address<="1100";
wait for 100ns;
--D
address<="1101";
wait for 100ns;
--E
address<="1110";
wait for 100ns;
--F
address<="1111";
wait;
end process;
end Behavioral;
```

TIMING DIAGRAM – 7 SEGMENT DISPLAY



CONSTRAINT FILE

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


## LEDs
set_property PACKAGE_PIN U16 [get_ports {FourLED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {FourLED[0]}]
set_property PACKAGE_PIN E19 [get_ports {FourLED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {FourLED[1]}]
set_property PACKAGE_PIN U19 [get_ports {FourLED[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {FourLED[2]}]
set_property PACKAGE_PIN V19 [get_ports {FourLED[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {FourLED[3]}]
set_property PACKAGE_PIN P1 [get_ports {ZeroLED}]
set_property IOSTANDARD LVCMOS33 [get_ports {ZeroLED}]
set_property PACKAGE_PIN L1 [get_ports {OverflowLED}]
set_property IOSTANDARD LVCMOS33 [get_ports {OverflowLED}]


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



set_property PACKAGE_PIN U2 [get_ports Anode[0]]
set_property IOSTANDARD LVCMOS33 [get_ports Anode[0]]
set_property PACKAGE_PIN U4 [get_ports Anode[1]]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports Anode[1]]
set_property PACKAGE_PIN V4 [get_ports Anode[2]]
set_property IOSTANDARD LVCMOS33 [get_ports Anode[2]]
set_property PACKAGE_PIN W4 [get_ports Anode[3]]
set_property IOSTANDARD LVCMOS33 [get_ports Anode[3]]

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

CONCLUSION

- The Instruction Decoder was implemented using IF-ELSE conditional structures to enable straightforward opcode interpretation and allow easy updates. This approach made managing multiple control signals for various instructions more efficient. An alternative method, using different logic, is also included in the Vivado files as a commented reference.
- For the Register Bank, IF-ELSE conditions triggered on the rising clock edge were employed. While flip-flop-based modules were an option, this method provided improved readability and modularity, supporting future instruction extensions.
- The 8-to-1, 4-bit Multiplexers were also constructed using IF-ELSE logic. Although cascading 2-to-1 multiplexers was possible, the chosen method simplified selection logic and enhanced ease of debugging.
- The 4-bit Add/Subtract Unit was designed with basic gates and Full Adders to support two's complement operations. Alternatively, it could be refactored into a Ripple Carry Adder (RCA) for better scalability and consistent delay handling.
- Program Counter logic combines a 3-bit Adder and a 2-to-1, 3-bit Multiplexer to support both sequential and conditional jumps, enabling operations like JZR that alter program flow based on register values at runtime.
- A slow clock module was introduced to synchronize timing across components during both simulation and hardware implementation. For accurate behavior in Vivado simulations, the slow clock counter should be set to 5 to ensure proper instruction timing and signal transitions.
- In summary, the nanoprocessor's modular structure, reusable components, and simulation-friendly design provide a strong base for future upgrades such as stack handling, branching, and multi-cycle execution.

IMPROVED NANO PROCESSOR DESIGN

INTRODUCTION

The enhanced version of the nano processor incorporates several newly added components, each designed with specific functionalities. Therefore, the instruction set supported by the nano processor is increased from 4 to 12. The program rom also can house more instructions facilitating an execution of more complex program compared to the above basic nano processor.

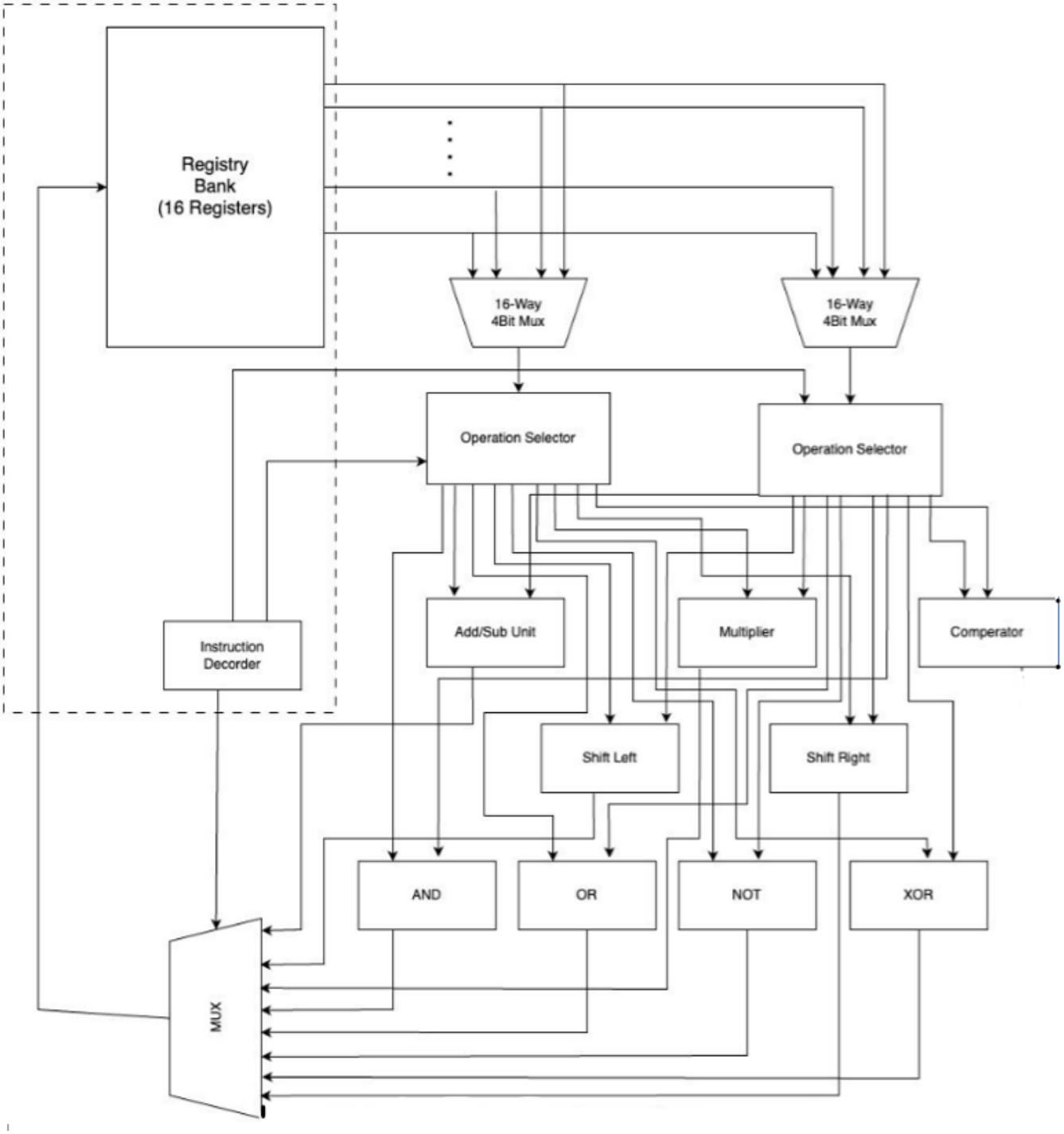
Newly Added Components

- **4-bit Multiplier:**
  This unit performs multiplication between values stored in the registers. The result is calculated and then stored back into the given register for further calculations.
- **Bit Comparator:**
  The comparator compares two values held in registers and indicates the result—whether one is greater than, less than, or equal to the other—using LEDs as indicators.
- **AND Operator:**
  This module executes a logical AND operation between two binary values stored in separate registers, then result is stored into a register.
- **Left Bit Shifter:**
  This unit enables shifting of 1 bit to the left of a binary value and then stored within a selected register .
- **Operation Selector:**
  Serving as a control interface, this unit manages communication between components. It ensures the correct register values are routed to the respective processing units for operation.
- **Right Bit Shifter:**
  Similar to the left shifter, this component shifts bits to the right and stores within a register.
- **NOT Operator:**
  This unit performs a bitwise NOT operation on the value of a selected register, inverting each bit and storing the result back in the register.
- **OR Operator:**
  Executes a logical OR operation between two register values and stores the resulting binary output into the appropriate register.
- **XOR Operator:**
  This module performs a bitwise exclusive OR (XOR) operation on two register inputs, with the outcome stored for subsequent use.

Improved Assembly Code Instruction Set

Instruction	Description	Format (16-bit instruction)
MOVI R,d	Move immediate value d to register R. i.e., R ← d R ∈ [0,15], d ∈ [0,15]	0010 RRRR 0000 dddd
ADD Ra, Rb	Add values in registers Ra and Rb and store the result in Ra. i.e., Ra ← Ra + Rb Ra, Rb ∈ [0,15]	0000 RaRaRaRa RbRbRb 0000
NEG R	2's complement of register R. i.e., R ← -R R ∈ [0,15]	0001 RRRR 0000 0000
JZR R,d	Jump if value in register R is 0. i.e., if R == 0 then PC ← d else PC ← PC + 1 R ∈ [0,15], d ∈ [0,15]	0011 RRRR 0000 dddd
MUL Ra, Rb	Multiply values in registers Ra and Rb and store the result in Ra. i.e., Ra ← Ra × Rb Ra, Rb ∈ [0,15]	0100 RaRaRa Ra RbRbRb 0000
COMP Ra, Rb	Compare values in registers Ra and Rb and output LEDs. Ra, Rb ∈ [0,15]	0101 RaRaRa Ra RbRbRb 0000
Bitwise AND Ra, Rb	AND operation for registers Ra and Rb and store the result in Ra. i.e., Ra ← Ra & Rb Ra, Rb ∈ [0,15]	0110 RaRaRa Ra RbRbRb 0000
Bitwise OR Ra, Rb	OR operation for values in registers Ra and Rb and store the result in Ra. i.e., Ra ← Ra | Rb Ra, Rb ∈ [0,15]	0111 RaRaRa Ra RbRbRb 0000
Bitwise NOT Ra	NOT operation for value in register Ra and store the result in Ra. i.e., Ra ← ~Ra Ra ∈ [0,15]	1000 RaRaRa Ra 0000 0000
Bitwise XOR Ra, Rb	XOR operation. Ra ← Ra ⊕ Rb Ra, Rb ∈ [0,15]	1001 RaRaRa Ra RbRbRb 0000
Left Shift R	Bitwise Left Shift by 1 bit R ← R << 1	1010 RRRR 0000 0000
Right Shift R	Bitwise Right Shift by 1 bit R ← R >> 1	1011 RRRR 0000 0000

High-level diagram of the Improved Nano processor



NANO PROCESSOR (IMPROVED VERSION)

DESIGN SOURCE FILE - NANO PROCESSOR (IMPROVED VERSION)

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values

entity ImprovedNanoprocessor is
    Port ( NanoClock : in STD_LOGIC;
           Reset : in STD_LOGIC;
           FourLED : out STD_LOGIC_VECTOR (3 downto 0);
           SevenSegment : out STD_LOGIC_VECTOR (6 downto 0);
           OverflowLED_Add : out STD_LOGIC;
           ZeroLED : out STD_LOGIC ;
           Anode : out STD_LOGIC_VECTOR (3 downto 0);

           OverflowLED_Mul : out STD_LOGIC;
           AgreaterB : out STD_LOGIC;
           AequalB : out STD_LOGIC;
           AlessB : out STD_LOGIC
           );

end ImprovedNanoprocessor;

architecture Behavioral of ImprovedNanoprocessor is

--#################################################################################
Component Slow_Clk
PORT( Clk_in  : in STD_LOGIC;
      Clk_out : out STD_LOGIC);
end Component;

--#################################################################################

component ImprovedRegisterBank is
    Port (
        BankIn : in STD_LOGIC_VECTOR (3 downto 0);
        BankClock : in STD_LOGIC;
        BankReset : in STD_LOGIC;
        BankRegEn : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit register enable
        BankOut0 : out STD_LOGIC_VECTOR (3 downto 0);
```

```
        BankOut1 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut2 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut3 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut4 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut5 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut6 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut7 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut8 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut9 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut10 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut11 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut12 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut13 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut14 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut15 : out STD_LOGIC_VECTOR (3 downto 0)
    );
end component;

--#################################################################################

component SixteenWay4BitMUX is
    Port (
        In0, In1, In2, In3, In4, In5, In6, In7,In8, In9, In10, In11, In12, In13,
In14, In15 : in STD_LOGIC_VECTOR(3 downto 0);
        Reg_sel : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit selector
        Y       : out STD_LOGIC_VECTOR(3 downto 0)
    );
end component;

--#################################################################################

component ASU is
    Port ( AAddSub : in STD_LOGIC_VECTOR (3 downto 0);
           BAddSub : in STD_LOGIC_VECTOR (3 downto 0);
           AddSubSelect : in STD_LOGIC;
           OAddSub : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC);
end component;

--#################################################################################



component TwoWay4Bit_Mux is
```

```
    Port ( In0 : in STD_LOGIC_VECTOR (3 downto 0);
           In1 : in STD_LOGIC_VECTOR (3 downto 0);
           S0 : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

--#################################################################################

component ImprovedInstructionDecoder is
    Port ( InstructionBus : in STD_LOGIC_VECTOR (15 downto 0);
           RegValueForCheck : in STD_LOGIC_VECTOR (3 downto 0);
           JumpFlag : out STD_LOGIC;
           JumpAddress : out STD_LOGIC_VECTOR (3 downto 0);
           AddSubSelect : out STD_LOGIC;
           RegSelect1 : out STD_LOGIC_VECTOR (3 downto 0);
           RegSelect2 : out STD_LOGIC_VECTOR (3 downto 0);
           ImmediateValue : out STD_LOGIC_VECTOR (3 downto 0);
           LoadSelect : out STD_LOGIC_VECTOR(3 downto 0);
           RegEN : out STD_LOGIC_VECTOR (3 downto 0);
           OperationSelectorKey : out STD_LOGIC_VECTOR(3 downto 0));
end component;

--#################################################################################
component ImprovedProgramROM is
    Port ( MemorySelect : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit memory
selector
           InstructionBus : out STD_LOGIC_VECTOR (15 downto 0) -- 14-bit
instruction bus
    );
end component;

--#################################################################################
component ImprovedPC is
    Port ( PCIn    : in  STD_LOGIC_VECTOR(3 downto 0);
           PCOut   : out STD_LOGIC_VECTOR(3 downto 0);
           Reset   : in  STD_LOGIC;
           PCClock : in  STD_LOGIC);
end component;
--#################################################################################



component Improved4BitAdder is
    Port (
        A4BA   : in  STD_LOGIC_VECTOR(3 downto 0);
```

```
        O4BA   : out STD_LOGIC_VECTOR(3 downto 0));
end component;

--##############################################################################
component NineWay4Bit_MUX is
    Port (
        In0, In1, In2, In3, In4, In5, In6, In7, In8 : in  STD_LOGIC_VECTOR(3 downto
0);
        Sel   : in  STD_LOGIC_VECTOR(3 downto 0);  -- can represent 0 to 8
        Y     : out STD_LOGIC_VECTOR(3 downto 0)
    );
end component;

--##############################################################################

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

--##############################################################################
component Multiplier_4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow_Mul : out std_logic
           );
end component;
--##############################################################################
component BitShifterLeft is
    Port (
        A : in STD_LOGIC_VECTOR (3 downto 0);  -- Input 4-bit vector
        Y : out STD_LOGIC_VECTOR (3 downto 0) -- Output 4-bit vector
    );
end component;
--##############################################################################
component BitShifterRight is
    Port (
        A : in STD_LOGIC_VECTOR (3 downto 0);  -- Input 4-bit vector
        Y : out STD_LOGIC_VECTOR (3 downto 0) -- Output 4-bit vector
    );
end component;
--##############################################################################

component BitwiseAND is
```

```
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           X : out STD_LOGIC_VECTOR (3 downto 0));
end component;

--#################################################################################

component BitwiseNOT is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           X : out STD_LOGIC_VECTOR (3 downto 0));
end component;

--#################################################################################

component BitwiseOR is
Port (     A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           X : out STD_LOGIC_VECTOR (3 downto 0));
end component;

--#################################################################################

component BitwiseXOR is
Port (     A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           X : out STD_LOGIC_VECTOR (3 downto 0));
end component;

--#################################################################################

component Comparator_4bit is
port(
        A : in STD_LOGIC_vector(3 downto 0);         -- Input bit A
        B : in STD_LOGIC_vector(3 downto 0 );         -- Input bit B
        A_equals_B : out STD_LOGIC; -- Output for A = B
        A_greater_B : out STD_LOGIC; -- Output for A > B
        A_less_B : out STD_LOGIC   -- Output for A < B);
end Component;


--#################################################################################

component OperationSelector is
    Port (
      OutputMux    : in  STD_LOGIC_VECTOR(3 downto 0);
```

```
      SelectOpr    : in  STD_LOGIC_VECTOR(3 downto 0);
      ToAddSub     : out STD_LOGIC_VECTOR(3 downto 0);
      ToMul        : out STD_LOGIC_VECTOR(3 downto 0);
      ToComp       : out STD_LOGIC_VECTOR(3 downto 0);
      ToAND        : out STD_LOGIC_VECTOR(3 downto 0);
      ToOR         : out STD_LOGIC_VECTOR(3 downto 0);
      ToNOT        : out STD_LOGIC_VECTOR(3 downto 0);
      ToXOR        : out STD_LOGIC_VECTOR(3 downto 0);
      ToBitShifterLeft : out STD_LOGIC_VECTOR(3 downto 0);
      ToBitShifterRight : out STD_LOGIC_VECTOR(3 downto 0));
end component;

--#############################################################################

-------------------------------------------------------------
---------------------- Defining Signals-----------------------

signal clkout_to_components : std_logic;

SIGNAL D_0,D_1,D_2,D_3,D_4,D_5,D_6,D_7,D_8,D_9,D_10,D_11,D_12,D_13,D_14,D_15 :
STD_LOGIC_VECTOR (3 downto 0);  --Data Busses
signal bank_data_in : std_logic_vector(3 downto 0);
signal bank_reg_en_from_instruction_decoder : std_logic_vector(3 downto 0);

signal sixteenway4bit_mux_out_0,sixteenway4bit_mux_out_1 : std_logic_vector(3
downto 0); --Bus inputs for 4bit add/sub unit
signal regselect1, regselect2 : std_logic_vector(3 downto 0);

signal loadselector : std_logic_vector(3 downto 0) ;

signal addsubselector : std_logic ;
signal overflowledadd, overflowledmul : std_logic ;

signal opr_selection_key_0 : std_logic_vector(3 downto 0);

signal instructions : std_logic_vector(15 downto 0);
signal memoryselector : std_logic_vector(3 downto 0);

signal twoway4bit_mux_out : std_logic_vector(3 downto 0);
signal four_bit_adder_out : std_logic_vector(3 downto 0);
signal jumpingaddress : std_logic_vector(3 downto 0);
signal jumpingflag : std_logic;

signal toaddsubA,tomulA,tocompA,toandA,toorA,tonotA,toxorA,toshiftlA,toshiftrA :
std_logic_vector(3 downto 0);
```

```
signal toaddsubB,tomulB,tocompB,toandB,toorB,toxorB,toshiftlB,toshiftrB,tonotB :
std_logic_vector(3 downto 0);

signal addsubOUT,mulOUT,compOUT,andOUT,orOUT,notOUT,xorOUT,shiftlOUT,shiftrOUT :
std_logic_vector(3 downto 0);

signal nineway4bit_mux_in8 : std_logic_vector(3 downto 0);


begin

Processor_Clock_0 : Slow_Clk
port map (
   Clk_in  => NanoClock , -- put nanoclock in constraints file
   Clk_out  => clkout_to_components);

Improved_register_bank : ImprovedRegisterBank
    Port map ( BankIn => bank_data_in ,
           BankClock => clkout_to_components,
           BankReset => Reset,  -- mapped to RESET button of whole processor
           BankRegEn => bank_reg_en_from_instruction_decoder,
           BankOut0 => D_0,
           BankOut1 => D_1,
           BankOut2 => D_2,
           BankOut3 => D_3,
           BankOut4 => D_4,
           BankOut5 => D_5,
           BankOut6 => D_6,
           BankOut7 => D_7,
           BankOut8 => D_8,
          BankOut9 => D_9,
          BankOut10 => D_10,
          BankOut11 => D_11,
          BankOut12 => D_12,
          BankOut13 => D_13,
          BankOut14 => D_14,
          BankOut15 => D_15);


SixteenWay4Bit_MUX_0 : SixteenWay4BitMUX
    Port map(
       In0=> D_0,
       In1=> D_1,
       In2=> D_2,
       In3 => D_3,
```

```
        In4=> D_4,
        In5=> D_5,
        In6=> D_6,
        In7=> D_7,
        In8=> D_8,
        In9=> D_9,
        In10=> D_10,
        In11 => D_11,
        In12=> D_12,
        In13=> D_13,
        In14=> D_14,
        In15=> D_15,
        Reg_sel=> regselect1,
        Y => sixteenway4bit_mux_out_0 );

SixteenWay4Bit_MUX_1 : SixteenWay4BitMUX
    Port map(
        In0=> D_0,
        In1=> D_1,
        In2=> D_2,
        In3 => D_3,
        In4=> D_4,
        In5=> D_5,
        In6=> D_6,
        In7=> D_7,
        In8=> D_8,
        In9=> D_9,
        In10=> D_10,
        In11 => D_11,
        In12=> D_12,
        In13=> D_13,
        In14=> D_14,
        In15=> D_15,
        Reg_sel=> regselect2,
        Y => sixteenway4bit_mux_out_1 );


nineway_mux : NineWay4Bit_MUX
            Port map(
                In0 => addsubOUT,
                In1=> mulOUT,
                In2=> andOUT,
                In3=> orOUT,
                In4=> notOUT,
                In5=> xorOUT,
```

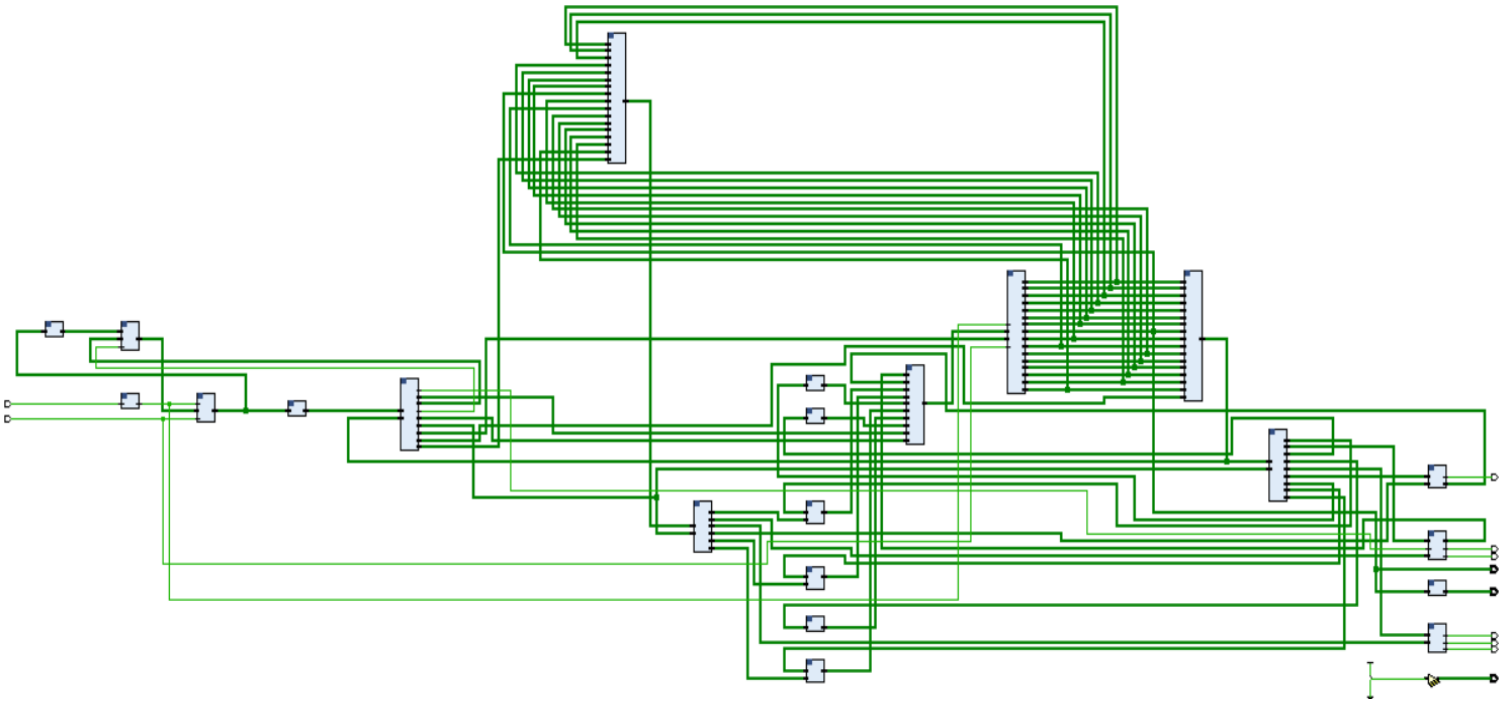
```
                In6=> shiftrOUT,
                In7=> shiftlOUT,
                In8=> nineway4bit_mux_in8,
                Sel=> loadselector,
                Y  => bank_data_in );

asu_0 :  ASU
            Port map ( AAddSub => toaddsubA,
                       BAddSub => toaddsubB,
                       AddSubSelect => addsubselector,
                       OAddSub => addsubOUT,
                       Overflow => overflowledadd ,
                       Zero => zeroLED );

improved_instruction_decoder : ImprovedInstructionDecoder
             Port map( InstructionBus => instructions ,
                    RegValueForCheck  => sixteenway4bit_mux_out_0 ,
                    JumpFlag          => jumpingflag ,
                    JumpAddress       => jumpingaddress,
                    AddSubSelect      => addsubselector,
                    RegSelect1        => regselect1,
                    RegSelect2        => regselect2,
                    ImmediateValue    => nineway4bit_mux_in8,
                    LoadSelect        => loadselector,
                    RegEN             => bank_reg_en_from_instruction_decoder,
                    OperationSelectorKey => opr_selection_key_0 );


improved_rom : ImprovedProgramROM
    Port map( MemorySelect => memoryselector,
              InstructionBus  => instructions);


imporved_pc_0 : ImprovedPC
      Port map ( PCIn   => twoway4bit_mux_out,
                 PCOut  => memoryselector ,
                 Reset  => Reset,
                 PCClock => clkout_to_components );


improved_4ba_0 : Improved4BitAdder
     Port map ( A4BA => memoryselector,
                O4BA => four_bit_adder_out);
```

```
next_address_select : TwoWay4Bit_Mux
port map( In0 => four_bit_adder_out,
          In1 => jumpingaddress,
          S0 =>jumpingflag,
          Y => twoway4bit_mux_out );


 seven_segment : LUT_16_7
        Port map( address => D_7 ,
                  data    => SevenSegment );


 multiplier :Multiplier_4
    Port map ( A => tomulA ,
           B =>tomulB ,
           Y => mulOUT,
           Overflow_Mul => overflowledmul);


bit_left : BitShifterLeft
    Port map (
        A => toshiftlA,
        Y => shiftlOUT
    );

bit_right : BitShifterRight
        Port map (
            A => toshiftrA,
            Y => shiftrOUT
        );

bit_and : BitwiseAND
        Port map (
            A => toandA,


 B => toandB,
 X => andOUT );

bit_not : BitwiseNOT
        Port map (
            A => tonotA,
            X => notOUT);
```

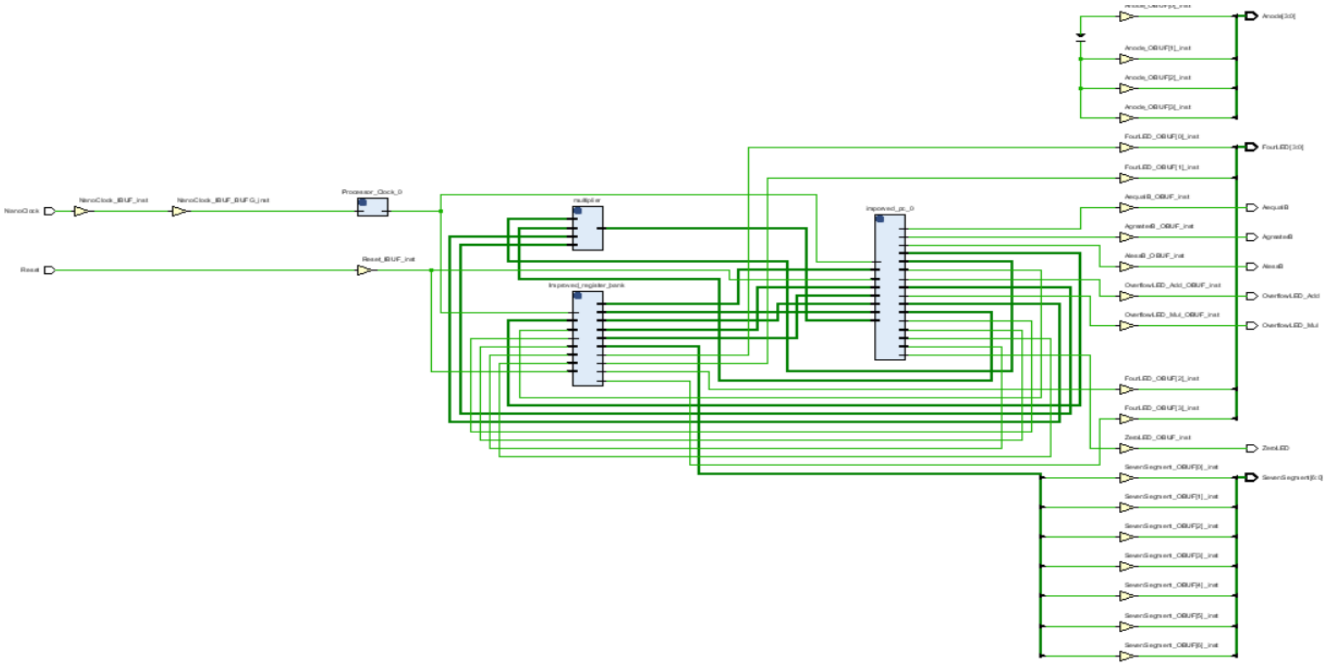
```
bit_or : BitwiseOR
        Port map (
        A => toorA,
        B => toorB,
        X => orOUT);

bit_xor : BitwiseXOR
        Port map (
            A => toxorA,
            B => toxorB,
            X => xorOUT);

comparator : Comparator_4bit
    port map(
        A => tocompA,
        B =>tocompB,
        A_equals_B => AequalB,
        A_greater_B =>AgreaterB,
        A_less_B => AlessB);

opr_selector_0 : OperationSelector
        Port map (
          OutputMux      => sixteenway4bit_mux_out_0,
          SelectOpr      => opr_selection_key_0,
          ToAddSub       => toaddsubA,
          ToMul          => tomulA,
          ToComp         => tocompA,
          ToAND          => toandA,
          ToOR           => toorA,
          ToNOT          => tonotA,
          ToXOR          => toxorA,
          ToBitShifterLeft  => toshiftlA,
          ToBitShifterRight => toshiftrA);


opr_selector_1 : OperationSelector
        Port map (
          OutputMux      => sixteenway4bit_mux_out_1,
          SelectOpr      =>opr_selection_key_0,
          ToAddSub       => toaddsubB,
          ToMul          =>tomulB,
          ToComp         =>tocompB,
          ToAND          =>toandB,
          ToOR           =>toorB,
          ToNOT          =>tonotB,
```

```
            ToXOR           =>toxorB,
            ToBitShifterLeft  => toshiftlB,
            ToBitShifterRight => toshiftrB);

FourLED <= D_7 ;
OverflowLED_Add <= overflowledadd ;
OverflowLED_Mul <= overflowledmul ;
Anode <= "1110";

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC - NANO PROCESSOR (IMPROVED VERSION)



IMPLEMENTED DESIGN SCHEMATIC - NANO PROCESSOR (IMPROVED VERSION)

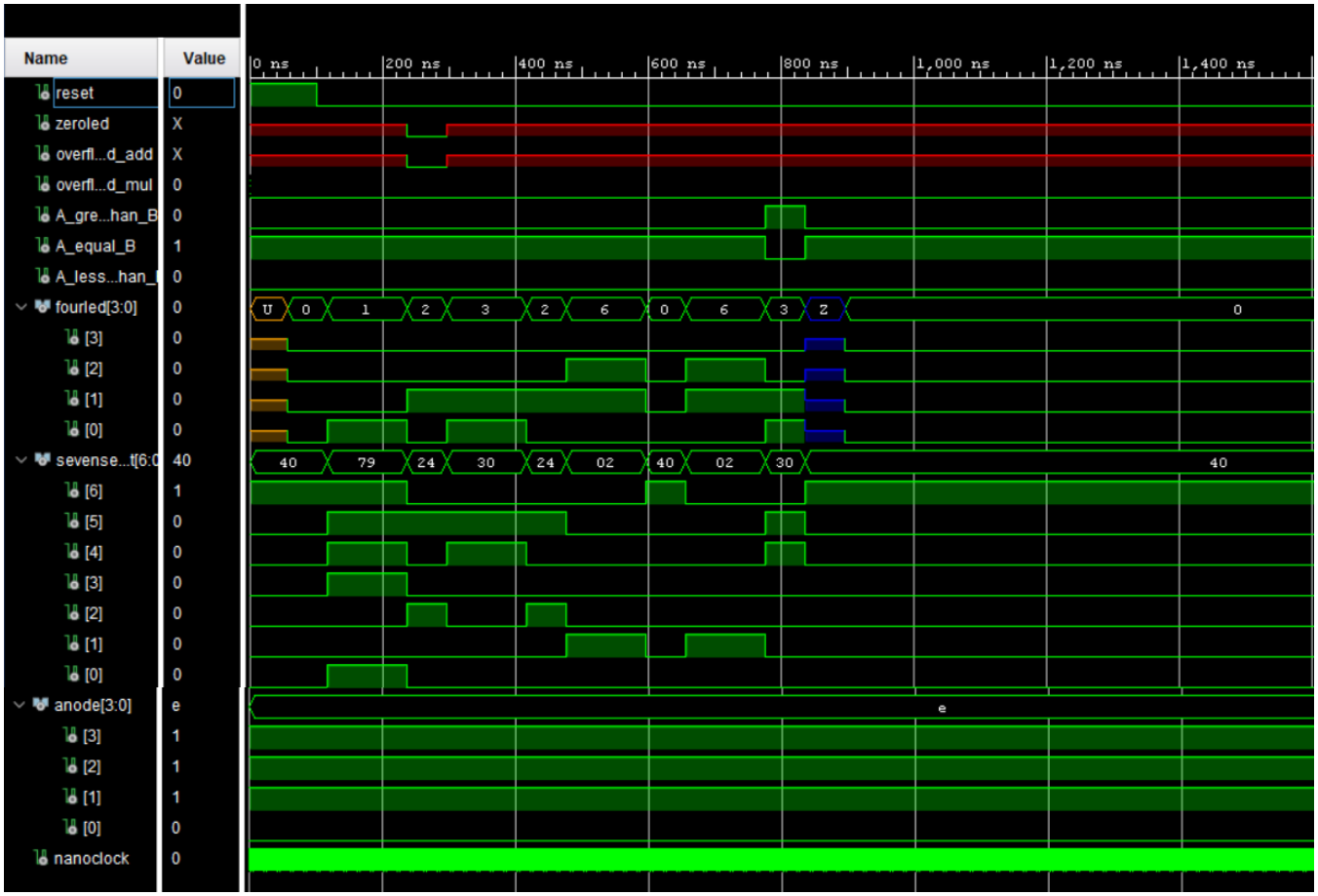


SIMULATION SOURCE FILE - NANO PROCESSOR (IMPROVED VERSION)

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

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

architecture Behavioral of ImprovedNanoprocessorSim is

component ImprovedNanoprocessor is
Port (NanoClock : in STD_LOGIC;
       Reset : in STD_LOGIC;
       FourLED : out STD_LOGIC_VECTOR (3 downto 0);
       SevenSegment : out STD_LOGIC_VECTOR (6 downto 0);
       OverflowLED_Add : out STD_LOGIC;
       ZeroLED : out STD_LOGIC ;
       Anode : out STD_LOGIC_VECTOR (3 downto 0);

      OverflowLED_Mul : out STD_LOGIC;
      AgreaterB : out STD_LOGIC;
      AequalB : out STD_LOGIC;
      AlessB : out STD_LOGIC );
```

```
end component;

signal reset , zeroled , overflowled_add, overflowled_mul : STD_LOGIC;
signal A_greater_than_B,A_equal_B,A_less_than_B  : STD_LOGIC;

signal fourled :STD_LOGIC_VECTOR (3 downto 0);
signal sevensegment :  STD_LOGIC_VECTOR (6 downto 0);
signal anode : STD_LOGIC_VECTOR (3 downto 0);
signal nanoclock :std_logic:='1';

begin

uut : ImprovedNanoprocessor port map (
   NanoClock => nanoclock ,
   Reset => reset ,
   FourLED => fourled,
   SevenSegment => sevensegment ,
   OverflowLED_Add => overflowled_add,
   ZeroLED => zeroled ,
   Anode => anode,
  OverflowLED_Mul => overflowled_mul,
  AgreaterB => A_greater_than_B ,
  AequalB  => A_equal_B,
  AlessB   => A_less_than_B  );


process begin
nanoclock <= NOT nanoclock;
wait for 3ns;
end process;


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

end Behavioral;
```

TIMING DIAGRAM - NANO PROCESSOR (IMPROVED VERSION)



INSTRUCTION DECODER (IMPROVED VERSION)

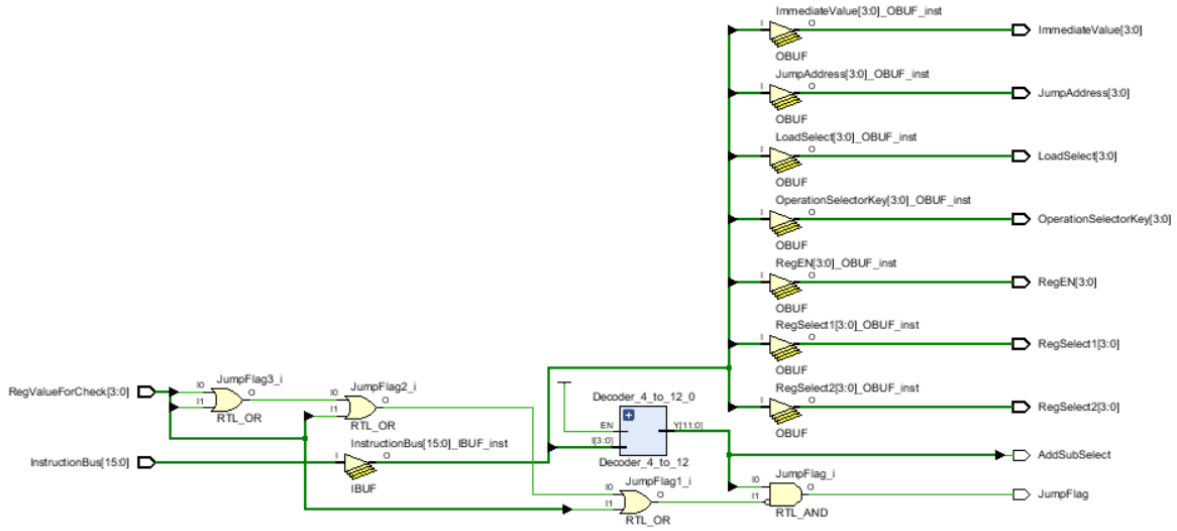
DESIGN SOURCE FILE - INSTRUCTION DECODER (IMPROVED VERSION)

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

entity ImprovedInstructionDecoder is
    Port ( InstructionBus : in STD_LOGIC_VECTOR (15 downto 0);
           RegValueForCheck : in STD_LOGIC_VECTOR (3 downto 0);
           JumpFlag : out STD_LOGIC;
           JumpAddress : out STD_LOGIC_VECTOR (3 downto 0);
           AddSubSelect : out STD_LOGIC;
           RegSelect1 : out STD_LOGIC_VECTOR (3 downto 0);
           RegSelect2 : out STD_LOGIC_VECTOR (3 downto 0);
```

```
            ImmediateValue : out STD_LOGIC_VECTOR (3 downto 0);
            LoadSelect : out STD_LOGIC_Vector(3 downto 0);
            RegEN : out STD_LOGIC_VECTOR (3 downto 0);
            OperationSelectorKey : out STD_LOGIC_VECTOR(3 downto 0));
end ImprovedInstructionDecoder;

architecture Behavioral of ImprovedInstructionDecoder is

component Decoder_4_to_12 is
Port ( I : in STD_LOGIC_VECTOR (3 downto 0);
EN : in STD_LOGIC;
Y : out STD_LOGIC_VECTOR (11 downto 0));
end component;


signal ADD, NEG, MOVI, JZR, MUL, COMP, BitwiseAND, BitwiseOR, BitwiseNOT,
BitwiseXOR, BitShiftLeft, BitShiftRight:STD_LOGIC;

begin

Decoder_4_to_12_0 : Decoder_4_to_12
Port map ( I(0) => InstructionBus(12),
           I(1) => InstructionBus(13),
           I(2) => InstructionBus(14),
           I(3) => InstructionBus(15),
           EN   => '1',
           Y(0) => ADD,
           Y(1) => NEG,
           Y(2) => MOVI,
           Y(3) => JZR,
           Y(4) => MUL,
           Y(5) => COMP,
           Y(6) => BitwiseAND,
           Y(7) => BitwiseOR,
           Y(8) => BitwiseNOT,
           Y(9) => BitwiseXOR,
           Y(10) => BitShiftLeft,
           Y(11) => BitShiftRight);


RegSelect1   <= InstructionBus(11 downto 8);
RegSelect2   <= InstructionBus(7 downto 4);
RegEN        <= InstructionBus(11 downto 8);
AddSubSelect <= NEG;
```

```
JumpFlag      <= JZR AND ( NOT(RegValueForCheck(0) OR RegValueForCheck(1) OR
RegValueForCheck(2) OR RegValueForCheck(3)));
JumpAddress   <= InstructionBus(3 downto 0);
LoadSelect    <= InstructionBus(15 downto 12) ;
ImmediateValue <= InstructionBus(3 downto 0);
OperationSelectorKey <= InstructionBus(15 downto 12);

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC - INSTRUCTION DECODER (IMPROVED VERSION)



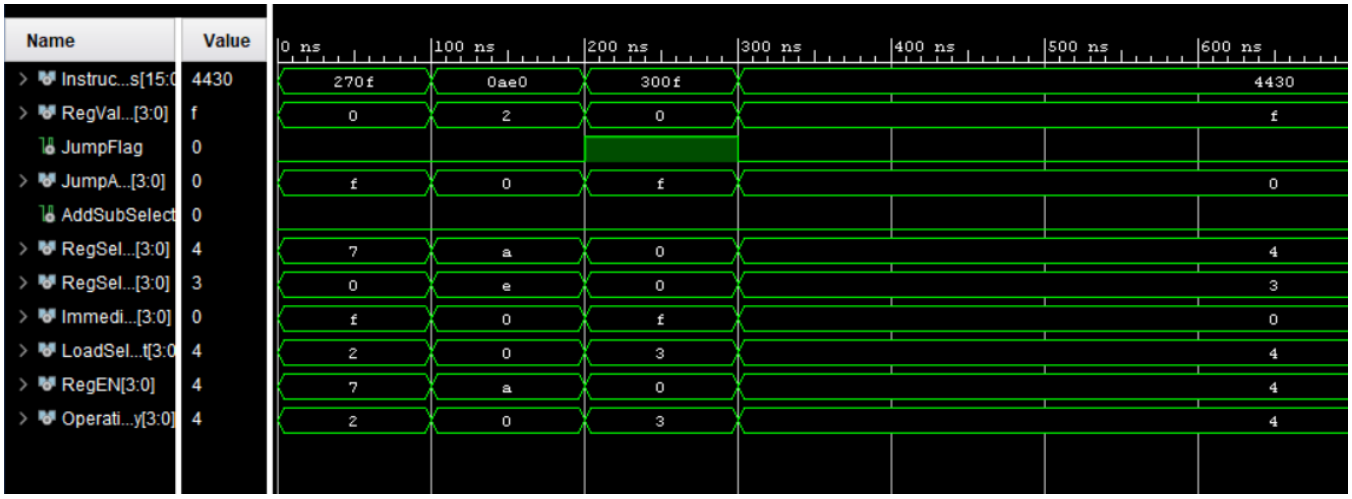
SIMULATION SOURCE FILE - INSTRUCTION DECODER (IMPROVED VERSION)

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

entity ImprovedInstructionDecoder_tb is
-- Testbench does not have any ports
end ImprovedInstructionDecoder_tb;

architecture Behavioral of ImprovedInstructionDecoder_tb is

    -- Component Declaration for the Unit Under Test (UUT)
    component ImprovedInstructionDecoder
        Port ( InstructionBus : in STD_LOGIC_VECTOR (15 downto 0);
               RegValueForCheck : in STD_LOGIC_VECTOR (3 downto 0);
               JumpFlag : out STD_LOGIC;
               JumpAddress : out STD_LOGIC_VECTOR (3 downto 0);
```

```
            AddSubSelect : out STD_LOGIC;
            RegSelect1 : out STD_LOGIC_VECTOR (3 downto 0);
            RegSelect2 : out STD_LOGIC_VECTOR (3 downto 0);
            ImmediateValue : out STD_LOGIC_VECTOR (3 downto 0);
            LoadSelect : out STD_LOGIC_VECTOR (3 downto 0);
            RegEN : out STD_LOGIC_VECTOR (3 downto 0);
            OperationSelectorKey : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    -- Signals to connect to the UUT
    signal InstructionBus : STD_LOGIC_VECTOR (15 downto 0);
    signal RegValueForCheck : STD_LOGIC_VECTOR (3 downto 0);
    signal JumpFlag : STD_LOGIC;
    signal JumpAddress : STD_LOGIC_VECTOR (3 downto 0);
    signal AddSubSelect : STD_LOGIC;
    signal RegSelect1 : STD_LOGIC_VECTOR (3 downto 0);
    signal RegSelect2 : STD_LOGIC_VECTOR (3 downto 0);
    signal ImmediateValue : STD_LOGIC_VECTOR (3 downto 0);
    signal LoadSelect : STD_LOGIC_VECTOR (3 downto 0);
    signal RegEN : STD_LOGIC_VECTOR (3 downto 0);
    signal OperationSelectorKey : STD_LOGIC_VECTOR (3 downto 0);

begin

    -- Instantiate the Unit Under Test (UUT)
    UUT: ImprovedInstructionDecoder
        Port map (
            InstructionBus => InstructionBus,
            RegValueForCheck => RegValueForCheck,
            JumpFlag => JumpFlag,
            JumpAddress => JumpAddress,
            AddSubSelect => AddSubSelect,
            RegSelect1 => RegSelect1,
            RegSelect2 => RegSelect2,
            ImmediateValue => ImmediateValue,
            LoadSelect => LoadSelect,
            RegEN => RegEN,
            OperationSelectorKey => OperationSelectorKey
        );

    -- Testbench Process
    process
    begin
        -- Test case 1: Basic instruction decode
```

```
        InstructionBus <= "0010011100001111"; -- Example MOVI operation

        RegValueForCheck <= "0000";
        wait for 100 ns;

        -- Test case 2: NEG operation with valid registers
        InstructionBus <= "0000101011100000"; -- Example ADD operation
        RegValueForCheck <= "0010";
        wait for 100 ns;

        -- Test case 3: JZR operation with zero register
        InstructionBus <= "0011000000001111"; -- Example JZR operation
        RegValueForCheck <= "0000"; -- Simulate zero register
        wait for 100 ns;

        -- Test case 4: Immediate operation
        InstructionBus <= "0100010000110000"; -- MUL operation
        RegValueForCheck <= "1111";
        wait for 100 ns;

        -- Add additional test cases as needed

        -- End simulation
        wait;
    end process;

end Behavioral;
```

TIMING DIAGRAM - INSTRUCTION DECODER (IMPROVED VERSION)



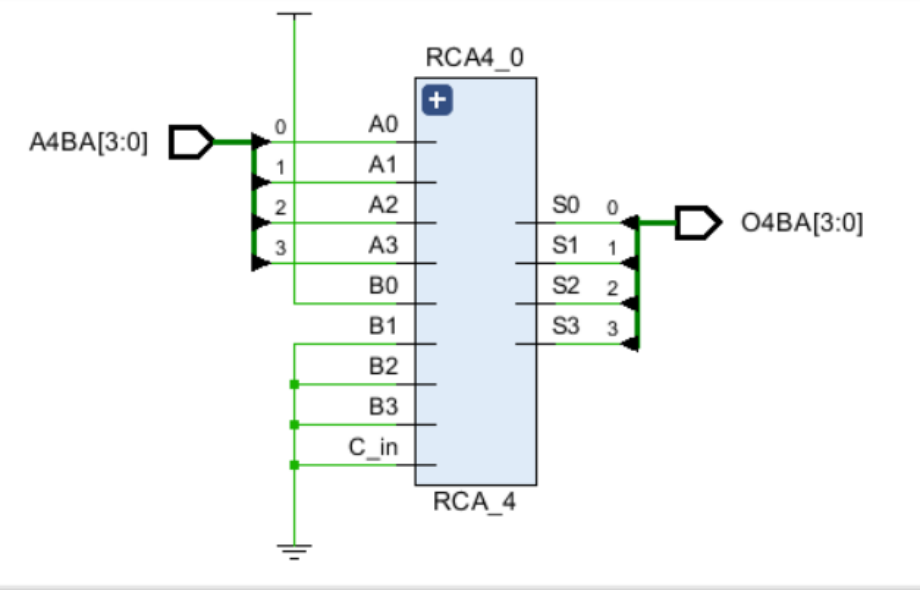
4 BIT ADDER

DESIGN SOURCE FILE – 4 BIT ADDER

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


entity Improved4BitAdder is
    Port (
        A4BA   : in  STD_LOGIC_VECTOR(3 downto 0);
        O4BA   : out STD_LOGIC_VECTOR(3 downto 0));
end Improved4BitAdder;

architecture Behavioral of Improved4BitAdder is
    component RCA_4 is
        Port (
            A0    : in  STD_LOGIC;
            A1    : in  STD_LOGIC;
            A2    : in  STD_LOGIC;
            A3    : in  STD_LOGIC;
            B0    : in  STD_LOGIC;
            B1    : in  STD_LOGIC;
            B2    : in  STD_LOGIC;
            B3    : in  STD_LOGIC;
            C_in  : in  STD_LOGIC;
            S0    : out STD_LOGIC;
            S1    : out STD_LOGIC;
            S2    : out STD_LOGIC;
            S3    : out STD_LOGIC;
            C_out : out STD_LOGIC
        );
    end component;

    signal carryout : STD_LOGIC;

begin

    RCA4_0 : RCA_4
        port map (
            A0    => A4BA(0),
            A1    => A4BA(1),
            A2    => A4BA(2),
            A3    => A4BA(3),

            B0    => '1',
```

```
            B1    => '0',
            B2    => '0',
            B3    => '0',
            C_in  => '0',
            S0    => O4BA(0),
            S1    => O4BA(1),
            S2    => O4BA(2),
            S3    => O4BA(3),
            C_out => carryout );

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – 4 BIT ADDER



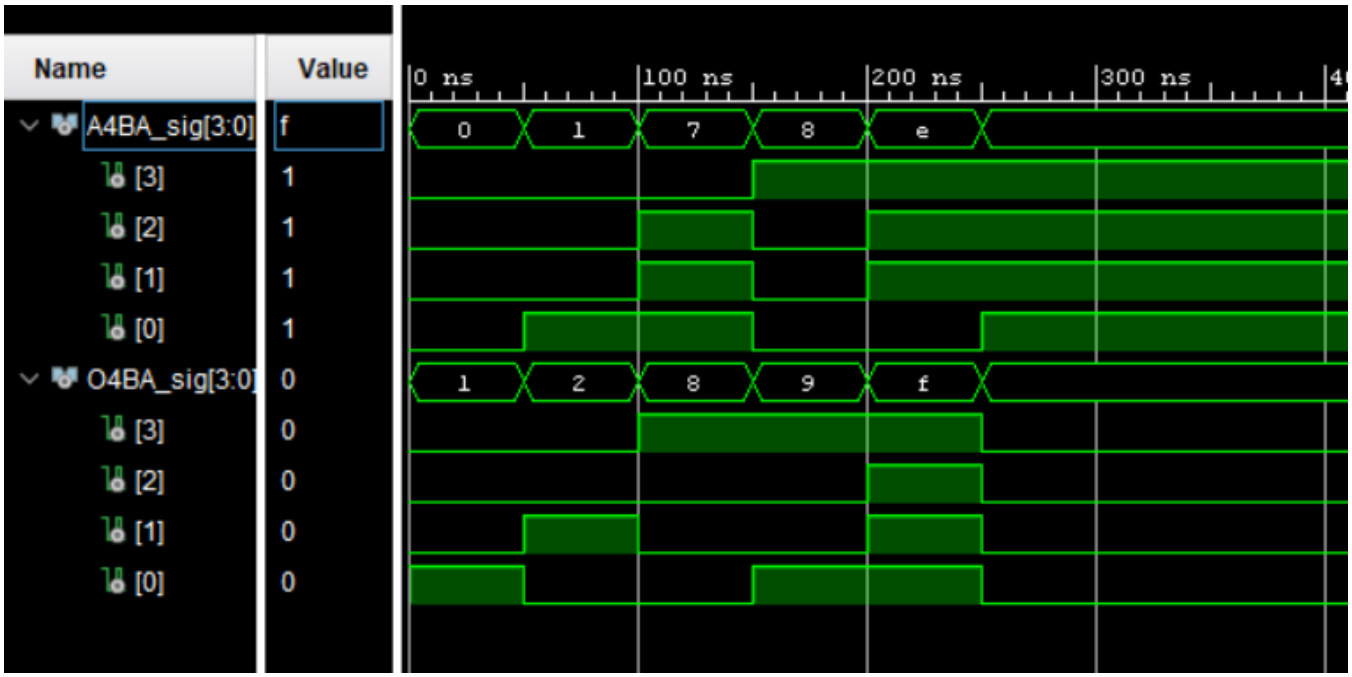
SIMULATION SOURCE FILE – 4 BIT ADDER

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

-- 4?bit Adder Testbench: Improved4BitAdderSim
entity Improved4BitAdderSim is
-- Testbench has no ports
end Improved4BitAdderSim;

architecture Behavioral of Improved4BitAdderSim is

    -- Unit Under Test declaration
    component Improved4BitAdder
        Port (
```

```
            A4BA : in  STD_LOGIC_VECTOR(3 downto 0);
            O4BA : out STD_LOGIC_VECTOR(3 downto 0)

        );
    end component;

    -- Signals to connect to the UUT
    signal A4BA_sig   : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
    signal O4BA_sig   : STD_LOGIC_VECTOR(3 downto 0);


begin

    -- Instantiate the Improved4BitAdder
    UUT: Improved4BitAdder
        port map (
            A4BA => A4BA_sig,
            O4BA => O4BA_sig);

    -- Stimulus process
    stim_proc: process
    begin
        -- Test increment of 0 => 1
        A4BA_sig <= "0000";
        wait for 50 ns;

        -- Test increment of 1 => 2
        A4BA_sig <= "0001";
        wait for 50 ns;

        -- Test increment of 7 => 8
        A4BA_sig <= "0111";
        wait for 50 ns;

        -- Test increment of 8 => 9
        A4BA_sig <= "1000";
        wait for 50 ns;

        -- Test increment of 14 => 15
        A4BA_sig <= "1110";
        wait for 50 ns;

        -- Test increment of 15 => 0
        A4BA_sig <= "1111";
```

```
        wait for 50 ns;

        wait;
    end process;

end Behavioral;
```

TIMING DIAGRAM – 4 BIT ADDER



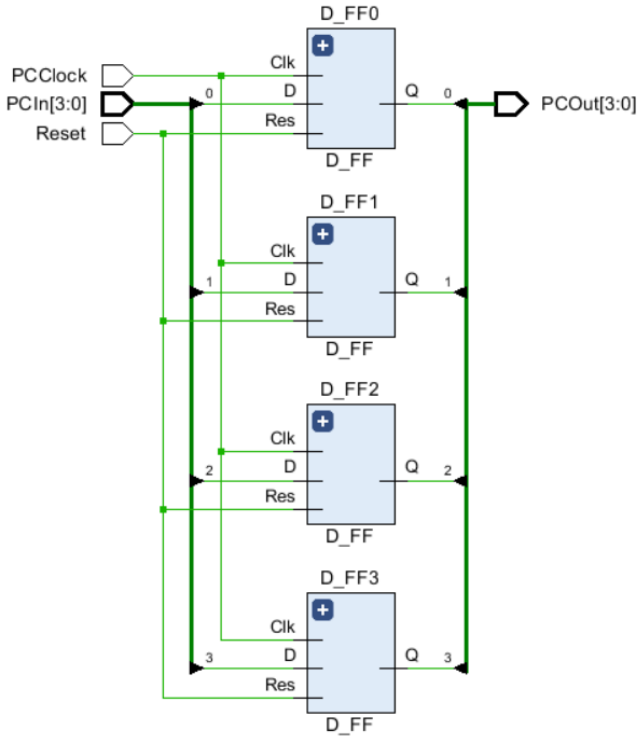
PROGRAM COUNTER (IMPROVED VERSION)

DESIGN SOURCE FILE – PROGRAM COUNTER (IMPROVED VERSION)

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

-- 4-bit Program Counter with Reset and Clock
entity ImprovedPC is
    Port ( PCIn    : in  STD_LOGIC_VECTOR(3 downto 0);
           PCOut   : out STD_LOGIC_VECTOR(3 downto 0);
           Reset   : in  STD_LOGIC;
           PCClock : in  STD_LOGIC);
end ImprovedPC;
```

```
architecture Behavioral of ImprovedPC is

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

begin

    -- Instantiate four D flip-flops for each bit of the PC
    D_FF0 : D_FF
        port map ( D   => PCIn(0),
                   Res => Reset,
                   Clk => PCClock,
                   Q   => PCOut(0) );

    D_FF1 : D_FF
        port map ( D   => PCIn(1),
                   Res => Reset,
                   Clk => PCClock,
                   Q   => PCOut(1) );

    D_FF2 : D_FF
        port map ( D   => PCIn(2),
                   Res => Reset,
                   Clk => PCClock,
                   Q   => PCOut(2)
                    );

    D_FF3 : D_FF
        port map ( D   => PCIn(3),
                   Res => Reset,
                   Clk => PCClock,
                   Q   => PCOut(3)
                   );

end Behavioral;
```

ELOBARATED DESIGN SCHEMATIC – PROGRAM COUNTER (IMPROVED VERSION)



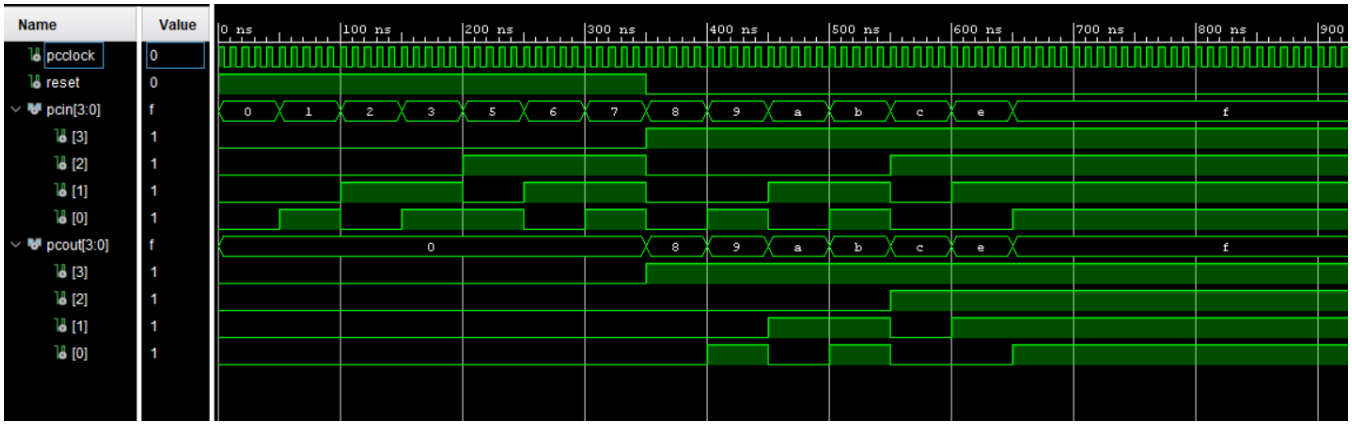
SIMULATION SOURCE FILE – PROGRAM COUNTER (IMPROVED VERSION)

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

entity ImprovedPCSim is
end ImprovedPCSim;

architecture Behavioral of ImprovedPCSim is

component ImprovedPC
    port (
        PCIn : in STD_LOGIC_VECTOR (3 downto 0);
        PCClock : in STD_LOGIC;
        Reset : in STD_LOGIC;
        PCOut : out STD_LOGIC_VECTOR (3 downto 0)
    );
end component;

signal pcclock : std_logic := '0';
signal reset : std_logic;
signal pcin : std_logic_vector(3 downto 0) := "0000";
```

```
signal pcout : std_logic_vector(3 downto 0);
begin

UUT: ImprovedPC
    port map (
        PCClock => pcclock,
        Reset => reset,
        PCOut => pcout,
        PCIn => pcin
    );

-- Clock generation
process
begin
    pcclock <= not pcclock;
    wait for 5 ns;
end process;

-- Stimulus process
process
begin
    reset <= '1';
    pcin <= "0000";
    wait for 50 ns;
    pcin <= "0001";
    wait for 50 ns;
    pcin <= "0010";
    wait for 50 ns;
    pcin <= "0011";
    wait for 50 ns;
    pcin <= "0101";
    wait for 50 ns;
    pcin <= "0110";
    wait for 50 ns;
    pcin <= "0111";
    wait for 50 ns;

    reset <= '0';
    pcin <= "1000";
    wait for 50 ns;
    pcin <= "1001";
    wait for 50 ns;
    pcin <= "1010";
    wait for 50 ns;
```

```
    pcin <= "1011";
    wait for 50 ns;
    pcin <= "1100";
    wait for 50 ns;
    pcin <= "1110";
    wait for 50 ns;
    pcin <= "1111";
    wait;
end process;

end Behavioral;
```

TIMING DIAGRAM – PROGRAM COUNTER (IMPROVED VERSION)



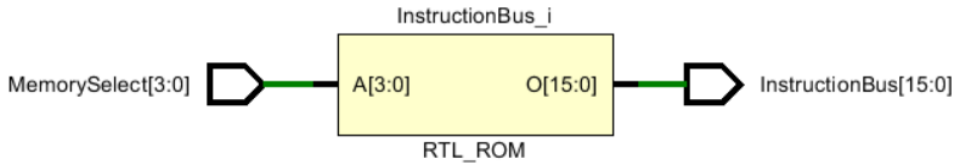
PROGRAM ROM (IMPROVED VERSION)

DESIGN SOURCE FILE – PROGRAM ROM (IMPROVED VERSION)

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

entity ImprovedProgramROM is
    Port ( MemorySelect : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit memory
selector
           InstructionBus : out STD_LOGIC_VECTOR (15 downto 0) -- 14-bit
instruction bus
    );
end ImprovedProgramROM;
```

```
architecture Behavioral of ImprovedProgramROM is


    -- ROM array to store 16 14-bit instructions
    type rom_type is array (0 to 15) of std_logic_vector(15 downto 0);
    signal program_ROM : rom_type := (
        "0010011100000001", -- MOVI R7, 1
        "0010000100000001", -- MOVI R1, 1
        "0010011100000010", -- MOVI R7, 2
        "0000011100010000", -- ADD R7, R1  (ANSWER 3)
        "0010001000000011", -- MOVI R2, 3
        "0010011100000010", -- MOVI R7, 2
        "0100011100100000", -- MUL R7, R2 (ANSWER 6)
        "0010001100000110", -- MOVI R3, 6
        "0010011100000000", -- MOVI R7, 0000
        "1001011100110000", -- xor R7, R3(ANSWER IS 6 again)
        "0010010100000000", -- MOVI R5,6
        "0010011100000011", -- MOVI R7,3
        "0101011101010000", -- COMP R7, R5 (COMPARISON BETWEEN 3 AND 6 LIGHTS UP
less THAN BULB)
        "0010011100000000", -- MOVI R7,0
        "0011000000001111", -- jump R0,15
        "0011000000001110"  -- JUMP R0,14);
begin

    -- Map the InstructionBus to the selected ROM entry
    InstructionBus <= program_ROM(to_integer(unsigned(MemorySelect)));

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – PROGRAM ROM (IMPROVED VERSION)



REGISTER BANK (IMPROVED VERSION)

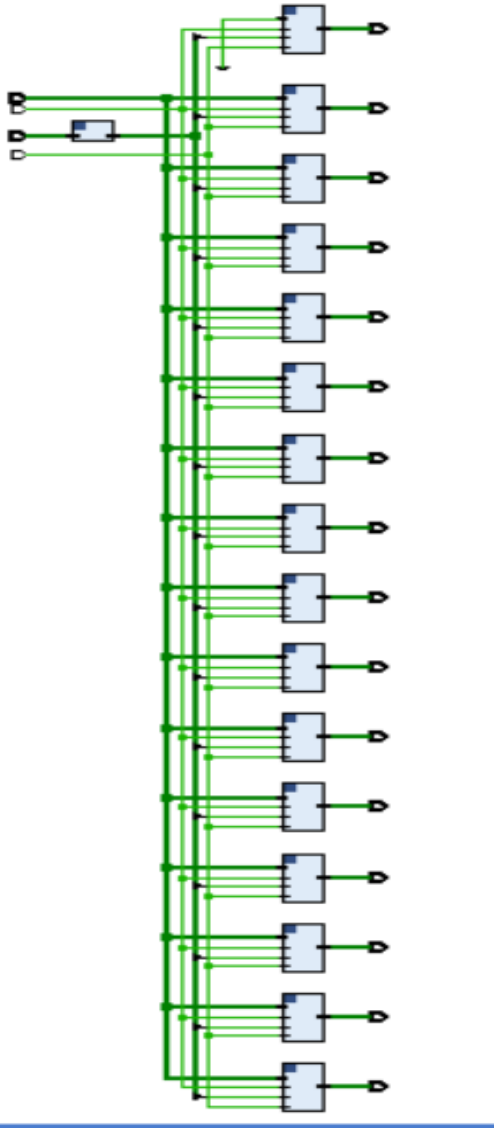
DESIGN SOURCE FILE – REGISTER BANK (IMPROVED VERSION)

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


entity ImprovedRegisterBank is
    Port (
        BankIn : in STD_LOGIC_VECTOR (3 downto 0);
        BankClock : in STD_LOGIC;
        BankReset : in STD_LOGIC;
        BankRegEn : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit register enable
        BankOut0 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut1 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut2 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut3 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut4 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut5 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut6 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut7 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut8 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut9 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut10 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut11 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut12 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut13 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut14 : out STD_LOGIC_VECTOR (3 downto 0);
        BankOut15 : out STD_LOGIC_VECTOR (3 downto 0)
    );
end ImprovedRegisterBank;

architecture Behavioral of ImprovedRegisterBank is

component Reg is
    Port (
        I : in STD_LOGIC_VECTOR (3 downto 0);
        RegClock : in STD_LOGIC;
        RegEn : in STD_LOGIC;
        RegReset : in STD_LOGIC;
        Y : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end component;

    component Decoder_4_to_16 is
        Port (
            I : in STD_LOGIC_VECTOR (3 downto 0);
            Y : out STD_LOGIC_VECTOR (15 downto 0)
        );
    end component;

    signal en : STD_LOGIC_VECTOR(15 downto 0);

begin

    -- Instantiate the 4-to-16 decoder
    Decoder_4_to_16_inst : Decoder_4_to_16
        port map (
            I => BankRegEn,
            Y => en
        );

    -- Instantiate 16 registers
    Reg_0 : Reg port map (I => "0000", RegClock => BankClock, RegEn => en(0),
RegReset => BankReset, Y => BankOut0);
    Reg_1 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(1),
RegReset => BankReset, Y => BankOut1);
    Reg_2 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(2),
RegReset => BankReset, Y => BankOut2);
    Reg_3 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(3),
RegReset => BankReset, Y => BankOut3);
    Reg_4 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(4),
RegReset => BankReset, Y => BankOut4);
    Reg_5 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(5),
RegReset => BankReset, Y => BankOut5);
    Reg_6 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(6),
RegReset => BankReset, Y => BankOut6);
    Reg_7 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(7),
RegReset => BankReset, Y => BankOut7);
    Reg_8 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(8),
RegReset => BankReset, Y => BankOut8);
    Reg_9 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(9),
RegReset => BankReset, Y => BankOut9);
```

```
    Reg_10 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(10),
RegReset => BankReset, Y => BankOut10);
    Reg_11 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(11),
RegReset => BankReset, Y => BankOut11);
    Reg_12 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(12),
RegReset => BankReset, Y => BankOut12);
    Reg_13 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(13),
RegReset => BankReset, Y => BankOut13);
    Reg_14 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(14),
RegReset => BankReset, Y => BankOut14);
    Reg_15 : Reg port map (I => BankIn, RegClock => BankClock, RegEn => en(15),
RegReset => BankReset, Y => BankOut15);

end Behavioral;
```

ELABOARTED DESIGN SCHEMATIC – REGISTER BANK (IMPROVED VERSION)



SIMULATION SOURCE FILE – REGISTER BANK (IMPROVED VERSION)

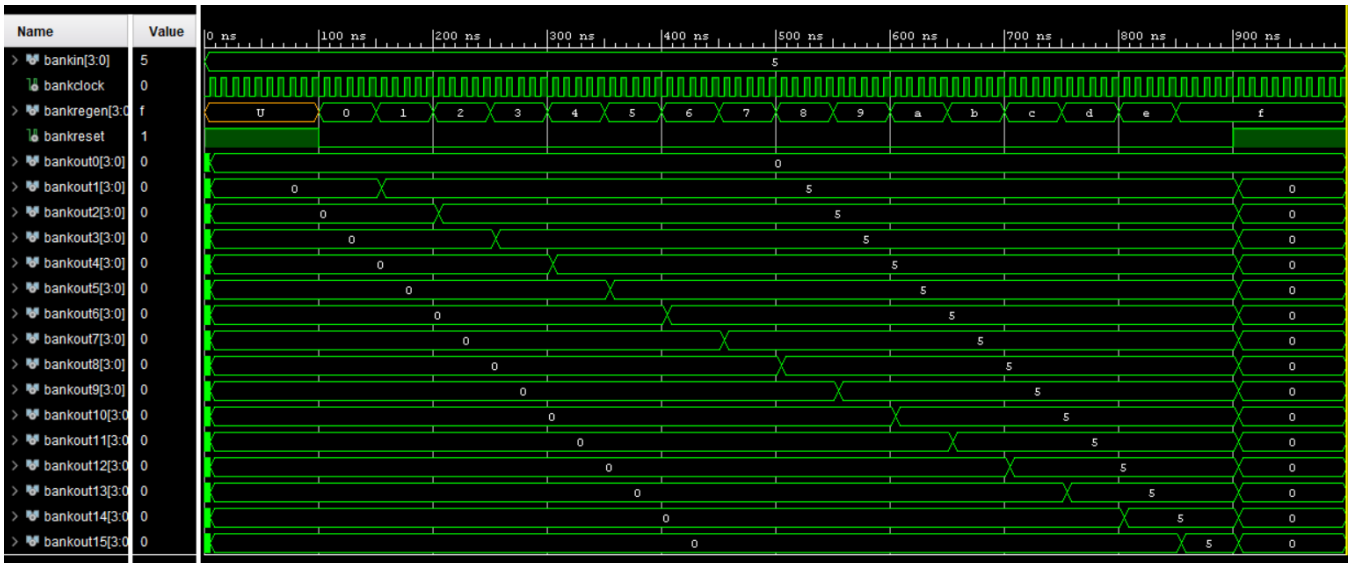
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ImproverRegisterBankSim is
end ImproverRegisterBankSim;

architecture Behavioral of ImproverRegisterBankSim is

    component ImprovedRegisterBank is
        Port (
            BankIn : in STD_LOGIC_VECTOR (3 downto 0);
            BankClock : in STD_LOGIC;
            BankReset : in STD_LOGIC;
            BankRegEn : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit enable
            BankOut0 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut1 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut2 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut3 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut4 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut5 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut6 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut7 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut8 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut9 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut10 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut11 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut12 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut13 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut14 : out STD_LOGIC_VECTOR (3 downto 0);
            BankOut15 : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    signal bankin : STD_LOGIC_VECTOR(3 downto 0);
    signal bankclock : STD_LOGIC := '0';  -- Clock signal
    signal bankregen : STD_LOGIC_VECTOR(3 downto 0); -- Memory Selector
    signal bankreset : STD_LOGIC; -- Reset signal
    signal bankout0, bankout1, bankout2, bankout3 : STD_LOGIC_VECTOR(3 downto 0);
    signal bankout4, bankout5, bankout6, bankout7 : STD_LOGIC_VECTOR(3 downto 0);
    signal bankout8, bankout9, bankout10, bankout11 : STD_LOGIC_VECTOR(3 downto 0);
    signal bankout12, bankout13, bankout14, bankout15 : STD_LOGIC_VECTOR(3 downto
0);
```

```
begin

    -- Instantiate the RegisterBank
    UUT : ImprovedRegisterBank
        port map (
            BankIn => bankin,
            BankClock => bankclock,
            BankReset => bankreset,
            BankRegEn => bankregen,
            BankOut0 => bankout0,
            BankOut1 => bankout1,
            BankOut2 => bankout2,
            BankOut3 => bankout3,
            BankOut4 => bankout4,
            BankOut5 => bankout5,
            BankOut6 => bankout6,
            BankOut7 => bankout7,
            BankOut8 => bankout8,
            BankOut9 => bankout9,
            BankOut10 => bankout10,
            BankOut11 => bankout11,
            BankOut12 => bankout12,
            BankOut13 => bankout13,
            BankOut14 => bankout14,
            BankOut15 => bankout15
        );

    -- Clock process
    ClockProcess : process
    begin
        wait for 5 ns;
        bankclock <= NOT(bankclock);
    end process;

    -- Simulation process
    SimulationProcess : process
    begin
        -- Initial Reset
        bankin <= "0101";
        bankreset <= '1';
        wait for 100 ns;
```

```
bankreset <= '0';

bankregen <= "0000";
wait for 50 ns;

bankregen <= "0001";
wait for 50 ns;

bankregen <= "0010";
wait for 50 ns;

bankregen <= "0011";
wait for 50 ns;

bankregen <= "0100";
wait for 50 ns;
bankregen <= "0101";
wait for 50 ns;

bankregen <= "0110";
wait for 50 ns;
bankregen <= "0111";
wait for 50 ns;

bankregen <= "1000";
wait for 50 ns;

bankregen <= "1001";
wait for 50 ns;

bankregen <= "1010";
wait for 50 ns;

bankregen <= "1011";
wait for 50 ns;

bankregen <= "1100";
wait for 50 ns;
bankregen <= "1101";
wait for 50 ns;

bankregen <= "1110";
```

```
        wait for 50 ns;
        bankregen <= "1111";
        wait for 50 ns;
        -- Final Reset
        bankreset <= '1';
        wait;

    end process;

end Behavioral;
```

TIMING DIAGRAM – REGISTER BANK (IMPROVED VERSION)



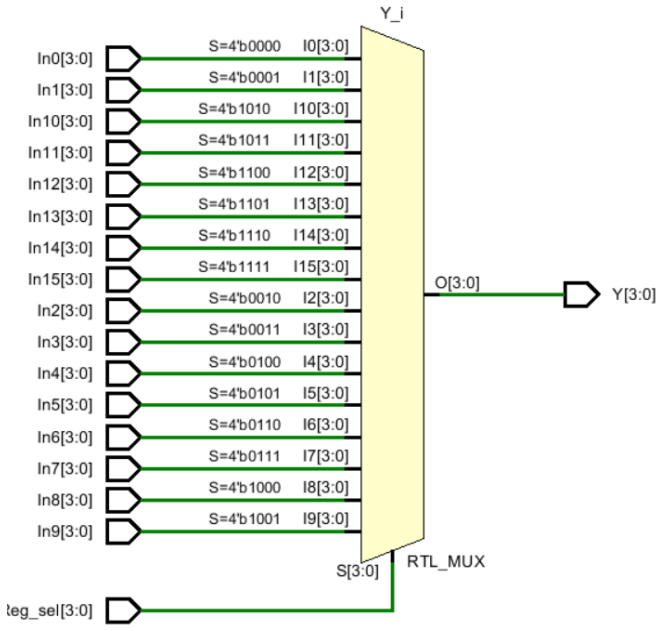
16 WAY 4 BIT MUX

DESIGN SOURCE FILE – 16 WAY 4 BIT MUX

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

entity SixteenWay4BitMUX is
    Port (
        In0, In1, In2, In3, In4, In5, In6, In7,In8, In9, In10, In11, In12, In13,
In14, In15 : in STD_LOGIC_VECTOR(3 downto 0);
        Reg_sel : in STD_LOGIC_VECTOR (3 downto 0); -- 4-bit selector
        Y       : out STD_LOGIC_VECTOR(3 downto 0));
```

```
end SixteenWay4BitMUX;

architecture Behavioral of SixteenWay4BitMUX is
begin
    process(Reg_sel, In0, In1, In2, In3, In4, In5, In6, In7,
            In8, In9, In10, In11, In12, In13, In14, In15)
    begin
        case Reg_sel is
            when "0000" => Y <= In0;
            when "0001" => Y <= In1;
            when "0010" => Y <= In2;
            when "0011" => Y <= In3;
            when "0100" => Y <= In4;
            when "0101" => Y <= In5;
            when "0110" => Y <= In6;
            when "0111" => Y <= In7;
            when "1000" => Y <= In8;
            when "1001" => Y <= In9;
            when "1010" => Y <= In10;
            when "1011" => Y <= In11;
            when "1100" => Y <= In12;
            when "1101" => Y <= In13;
            when "1110" => Y <= In14;
            when "1111" => Y <= In15;
            when others => Y <= (others => '0'); -- Safe default
        end case;
    end process;
end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – 16 WAY 4 BIT MUX

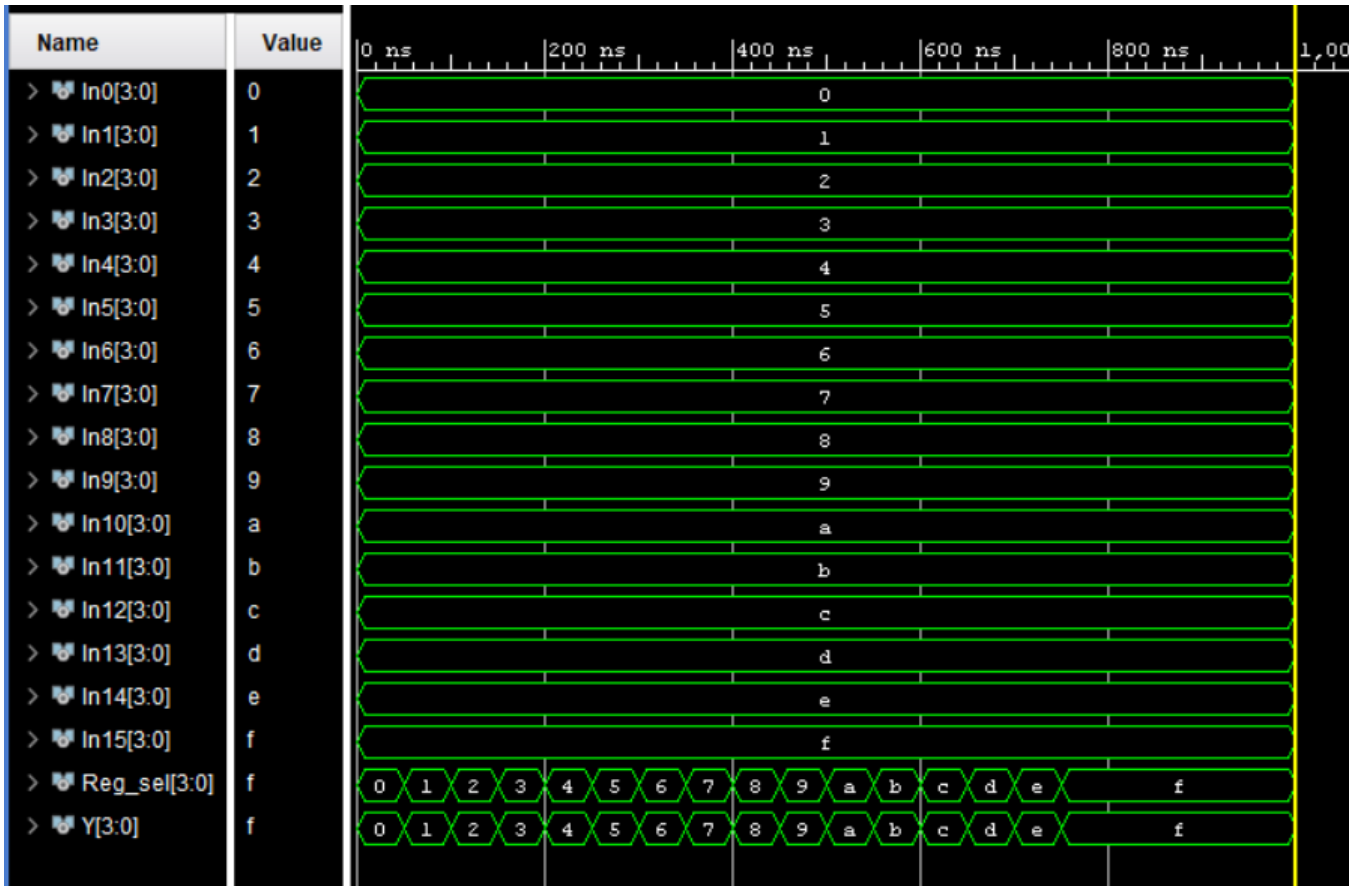


SIMULATION SOURCE FILE – 16 WAY 4 BIT MUX

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

entity SixteenWay4BitMUXSim is
end SixteenWay4BitMUXSim;

architecture Behavioral of SixteenWay4BitMUXSim is
    component SixteenWay4BitMUX
        Port (
            In0, In1, In2, In3, In4, In5, In6, In7,
            In8, In9, In10, In11, In12, In13, In14, In15 : in STD_LOGIC_VECTOR(3
downto 0);
            Reg_sel : in STD_LOGIC_VECTOR (3 downto 0);
            Y       : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    signal In0, In1, In2, In3, In4, In5, In6, In7 : STD_LOGIC_VECTOR(3 downto 0);
    signal In8, In9, In10, In11, In12, In13, In14, In15 : STD_LOGIC_VECTOR(3 downto
0);
    signal Reg_sel : STD_LOGIC_VECTOR(3 downto 0);
    signal Y      : STD_LOGIC_VECTOR(3 downto 0);

begin
    DUT: SixteenWay4BitMUX
        Port map (
            In0 => In0, In1 => In1, In2 => In2, In3 => In3,
            In4 => In4, In5 => In5, In6 => In6, In7 => In7,
            In8 => In8, In9 => In9, In10 => In10, In11 => In11,
            In12 => In12, In13 => In13, In14 => In14, In15 => In15,
            Reg_sel => Reg_sel,
            Y => Y
        );

    process
    begin
        In0  <= "0000"; In1  <= "0001"; In2  <= "0010"; In3  <= "0011";
        In4  <= "0100"; In5  <= "0101"; In6  <= "0110"; In7  <= "0111";
        In8  <= "1000"; In9  <= "1001"; In10 <= "1010"; In11 <= "1011";
        In12 <= "1100"; In13 <= "1101"; In14 <= "1110"; In15 <= "1111";

        Reg_sel <= "0000"; wait for 50 ns;
```

```
        Reg_sel <= "0001"; wait for 50 ns;
        Reg_sel <= "0010"; wait for 50 ns;
        Reg_sel <= "0011"; wait for 50 ns;
        Reg_sel <= "0100"; wait for 50 ns;
        Reg_sel <= "0101"; wait for 50 ns;
        Reg_sel <= "0110"; wait for 50 ns;
        Reg_sel <= "0111"; wait for 50 ns;
        Reg_sel <= "1000"; wait for 50 ns;
        Reg_sel <= "1001"; wait for 50 ns;
        Reg_sel <= "1010"; wait for 50 ns;
        Reg_sel <= "1011"; wait for 50 ns;
        Reg_sel <= "1100"; wait for 50 ns;
        Reg_sel <= "1101"; wait for 50 ns;
        Reg_sel <= "1110"; wait for 50 ns;
        Reg_sel <= "1111"; wait for 50 ns;

        wait;
    end process;
end Behavioral;
```

TIMING DIAGRAM – 16 WAY 4 BIT MUX



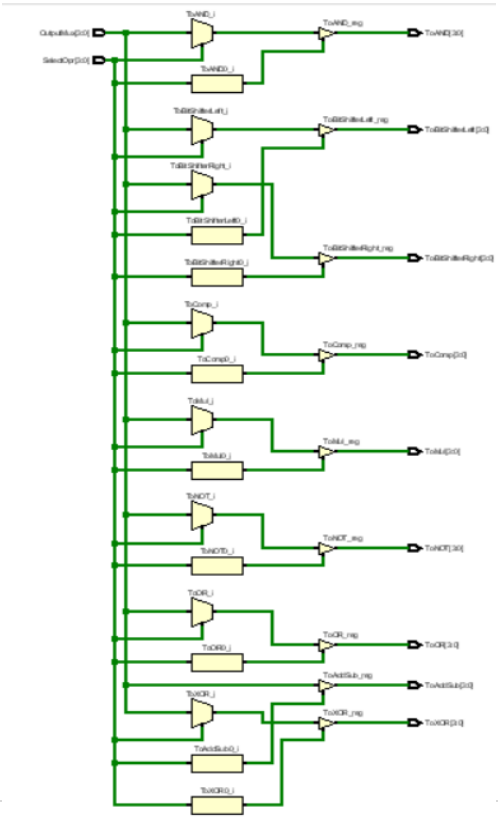
OPERATION SELECTOR

DESIGN SOURCE FILE – OPERATION SELECTOR

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

entity OperationSelector is
    Port (
      OutputMux    : in  STD_LOGIC_VECTOR(3 downto 0);
      SelectOpr    : in  STD_LOGIC_VECTOR(3 downto 0);
      ToAddSub     : out STD_LOGIC_VECTOR(3 downto 0);
      ToMul        : out STD_LOGIC_VECTOR(3 downto 0);
      ToComp       : out STD_LOGIC_VECTOR(3 downto 0);
      ToAND        : out STD_LOGIC_VECTOR(3 downto 0);
      ToOR         : out STD_LOGIC_VECTOR(3 downto 0);
      ToNOT        : out STD_LOGIC_VECTOR(3 downto 0);
      ToXOR        : out STD_LOGIC_VECTOR(3 downto 0);
      ToBitShifterLeft : out STD_LOGIC_VECTOR(3 downto 0);
      ToBitShifterRight : out STD_LOGIC_VECTOR(3 downto 0)


    );
end OperationSelector;

architecture Behavioral of OperationSelector is
begin

  process(OutputMux, SelectOpr)
  begin
    -- 1) Default everything to 'Z' (disconnected)
    ToAddSub     <= (others => 'Z');
    ToMul        <= (others => 'Z');
    ToComp       <= (others => 'Z');
    ToAND        <= (others => 'Z');
    ToOR         <= (others => 'Z');
    ToNOT        <= (others => 'Z');
    ToXOR        <= (others => 'Z');
    ToBitShifterLeft <= (others => 'Z');
    ToBitShifterRight <= (others => 'Z');

    -- 2) Drive exactly one port when SelectOpr matches
    case SelectOpr is
      when "0000" | "0001" =>
```

```
        ToAddSub <= OutputMux;
      when "0100" =>
        ToMul <= OutputMux;
      when "0101" =>
        ToComp <= OutputMux;
      when "0110" =>
        ToAND <= OutputMux;
      when "0111" =>
        ToOR <= OutputMux;
      when "1000" =>
        ToNOT <= OutputMux;
      when "1001" =>
        ToXOR <= OutputMux;
      when "1010"  =>
        ToBitShifterLeft <= OutputMux;
      when "1011"  =>
        ToBitShifterRight <= OutputMux;

      when others =>
        -- leave all outputs at 'Z'
        null;
    end case;
  end process;

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – OPERATION SELECTOR

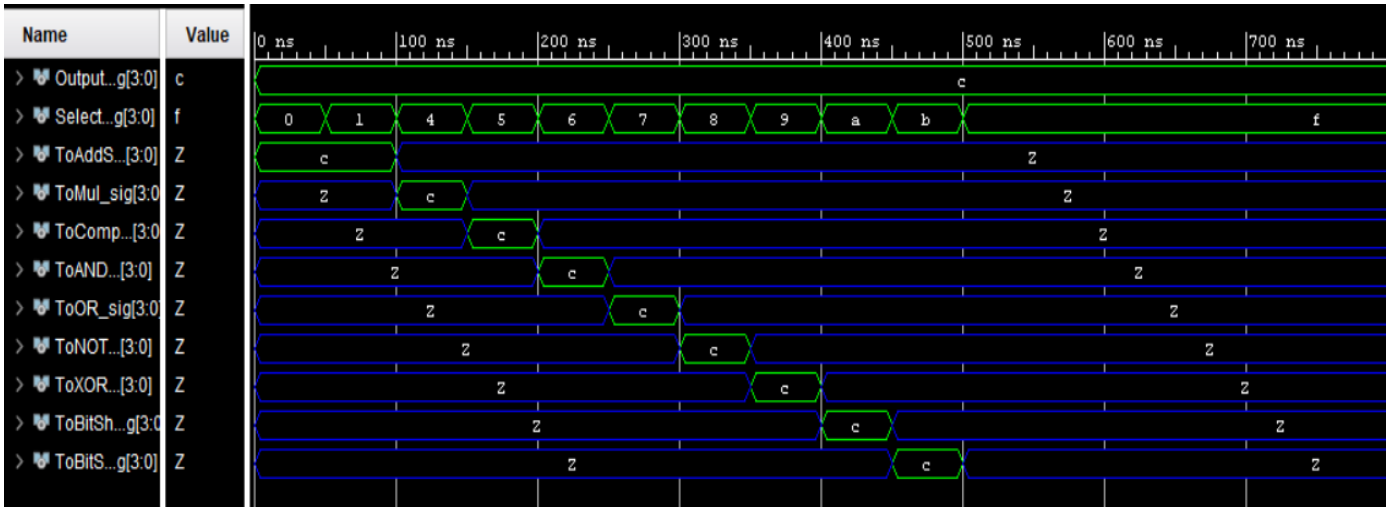


SIMULATION SOURCE FILE – OPERATION SELECTOR

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

-- Testbench for OperationSelector
entity OperationSelectorSim is
end OperationSelectorSim;

architecture Behavioral of OperationSelectorSim is

    -- Component declaration for the Unit Under Test (UUT)
    component OperationSelector
        Port (
            OutputMux          : in  STD_LOGIC_VECTOR(3 downto 0);
            SelectOpr          : in  STD_LOGIC_VECTOR(3 downto 0);
            ToAddSub           : out STD_LOGIC_VECTOR(3 downto 0);
            ToMul              : out STD_LOGIC_VECTOR(3 downto 0);
            ToComp             : out STD_LOGIC_VECTOR(3 downto 0);
            ToAND              : out STD_LOGIC_VECTOR(3 downto 0);
            ToOR               : out STD_LOGIC_VECTOR(3 downto 0);
            ToNOT              : out STD_LOGIC_VECTOR(3 downto 0);
            ToXOR              : out STD_LOGIC_VECTOR(3 downto 0);
            ToBitShifterLeft   : out STD_LOGIC_VECTOR(3 downto 0);
            ToBitShifterRight  : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    -- Signals
    signal OutputMux_sig        : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
    signal SelectOpr_sig        : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
    signal ToAddSub_sig         : STD_LOGIC_VECTOR(3 downto 0);
    signal ToMul_sig            : STD_LOGIC_VECTOR(3 downto 0);
    signal ToComp_sig           : STD_LOGIC_VECTOR(3 downto 0);
    signal ToAND_sig            : STD_LOGIC_VECTOR(3 downto 0);
    signal ToOR_sig             : STD_LOGIC_VECTOR(3 downto 0);
    signal ToNOT_sig            : STD_LOGIC_VECTOR(3 downto 0);
    signal ToXOR_sig            : STD_LOGIC_VECTOR(3 downto 0);
    signal ToBitShifterLeft_sig : STD_LOGIC_VECTOR(3 downto 0);
    signal ToBitShifterRight_sig: STD_LOGIC_VECTOR(3 downto 0);

begin
```

```
    UUT: OperationSelector
        port map (
            OutputMux          => OutputMux_sig,
            SelectOpr          => SelectOpr_sig,
            ToAddSub           => ToAddSub_sig,
            ToMul              => ToMul_sig,
            ToComp             => ToComp_sig,
            ToAND              => ToAND_sig,
            ToOR               => ToOR_sig,
            ToNOT              => ToNOT_sig,
            ToXOR              => ToXOR_sig,
            ToBitShifterLeft   => ToBitShifterLeft_sig,
            ToBitShifterRight  => ToBitShifterRight_sig
        );

    -- Stimulus process
    stim_proc: process
    begin
        OutputMux_sig <= "1100";  -- Sample input

        SelectOpr_sig <= "0000"; wait for 50 ns;
        SelectOpr_sig <= "0001"; wait for 50 ns;
        SelectOpr_sig <= "0100"; wait for 50 ns;
        SelectOpr_sig <= "0101"; wait for 50 ns;
        SelectOpr_sig <= "0110"; wait for 50 ns;
        SelectOpr_sig <= "0111"; wait for 50 ns;
        SelectOpr_sig <= "1000"; wait for 50 ns;
        SelectOpr_sig <= "1001"; wait for 50 ns;
        SelectOpr_sig <= "1010"; wait for 50 ns;
        SelectOpr_sig <= "1011"; wait for 50 ns;
        SelectOpr_sig <= "1111"; wait for 50 ns;  -- Invalid input

        wait;
    end process;

end Behavioral;
```

TIMING DIAGRAM – OPERATION SELECTOR



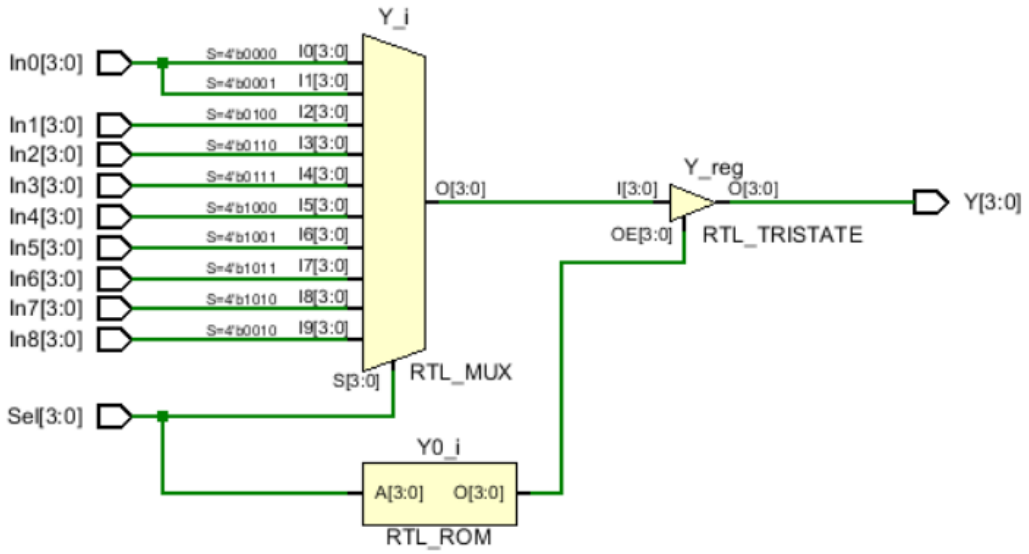
9 WAY 4BIT MUX

DESIGN SOURCE FILE – 9 WAY 4 BIT MUX

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

-- 9-way 4-bit multiplexer
entity NineWay4Bit_MUX is
    Port (
        In0, In1, In2, In3, In4, In5, In6, In7, In8 : in  STD_LOGIC_VECTOR(3 downto
0);
        Sel   : in  STD_LOGIC_VECTOR(3 downto 0);  -- can represent 0 to 8
        Y     : out STD_LOGIC_VECTOR(3 downto 0)
    );
end NineWay4Bit_MUX;

architecture Behavioral of NineWay4Bit_MUX is
begin
    process(Sel, In0, In1, In2, In3, In4, In5, In6, In7, In8)
    begin
        case Sel is
            when "0000" => Y <= In0; --add
            when "0001" => Y <= In0; --neg
            when "0100" => Y <= In1; --mul
            when "0110" => Y <= In2; --and
            when "0111" => Y <= In3; --or
            when "1000" => Y <= In4; --not
```

```
            when "1001" => Y <= In5; --xor
            when "1011" => Y <= In6; --shift right
            when "1010" => Y <= In7; --shift left
            when "0010" => Y <= In8; -- Immediate value
            when others => Y <= (others => 'Z');  -- safe default
        end case;
    end process;
end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – 9 WAY 4 BIT MUX



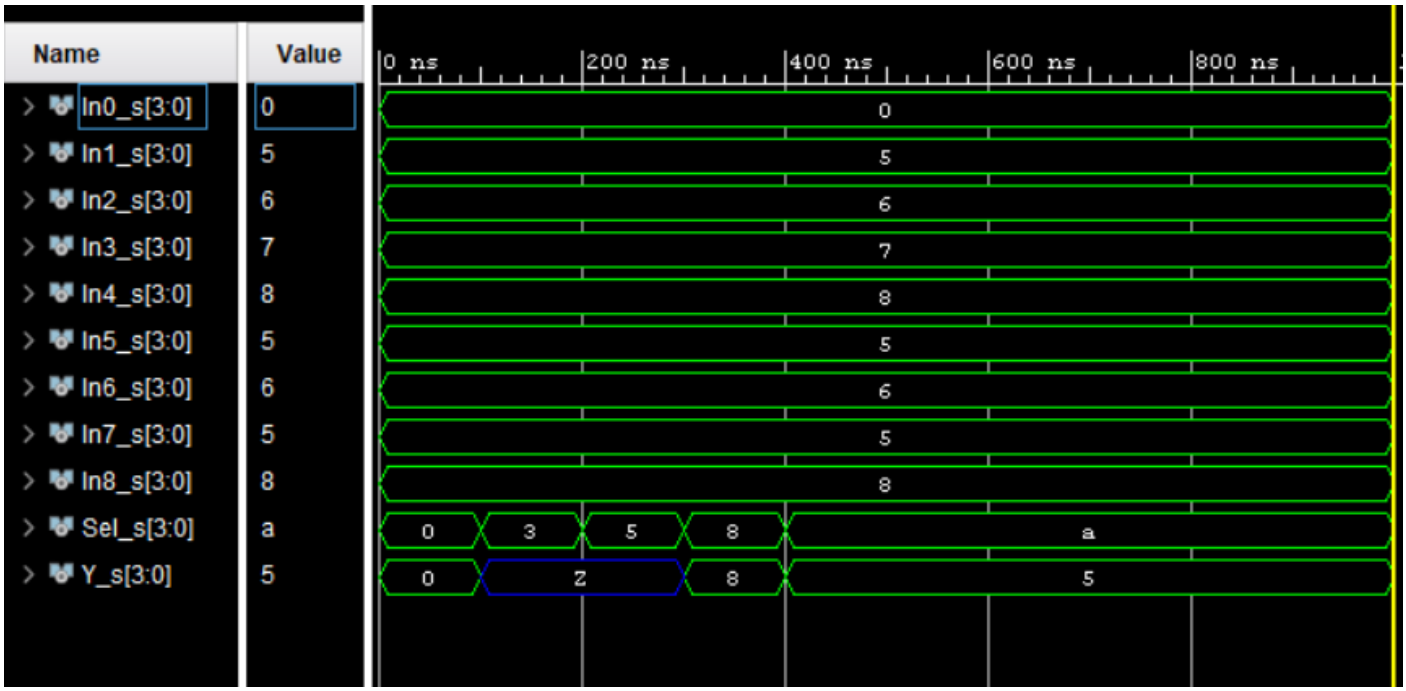
SIMULATION SOURCE FILE – 9 WAY 4 BIT MUX

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

-- Simple testbench for NineWay4Bit_MUX using UUT instantiation
entity NineWay4Bit_MUX_tb is
--  No ports
end NineWay4Bit_MUX_tb;

architecture Behavioral of NineWay4Bit_MUX_tb is

    -- Signals for inputs and output
    signal In0_s, In1_s, In2_s, In3_s : STD_LOGIC_VECTOR(3 downto 0);
    signal In4_s, In5_s, In6_s, In7_s : STD_LOGIC_VECTOR(3 downto 0);
    signal In8_s                      : STD_LOGIC_VECTOR(3 downto 0);
    signal Sel_s                      : STD_LOGIC_VECTOR(3 downto 0);
```

```
    signal Y_s                      : STD_LOGIC_VECTOR(3 downto 0);

begin

    ---------------------------------------------------------------
    -- Unit Under Test (UUT) instantiation
    ---------------------------------------------------------------
    UUT: entity work.NineWay4Bit_MUX
        port map (
            In0 => In0_s,
            In1 => In1_s,
            In2 => In2_s,
            In3 => In3_s,
            In4 => In4_s,
            In5 => In5_s,
            In6 => In6_s,
            In7 => In7_s,
            In8 => In8_s,
            Sel => Sel_s,
            Y   => Y_s
        );

    ---------------------------------------------------------------
    -- Simple stimulus process
    ---------------------------------------------------------------
    stim_proc: process
    begin
        -- Initialize inputs
            -- Initialize inputs to distinct patterns 0..8
        In0_s <= "0000";  -- 0
        In1_s <= "0101";  -- 1
        In2_s <= "0110";  -- 2
        In3_s <= "0111";  -- 3
        In4_s <= "1000";  -- 4
        In5_s <= "0101";  -- 5
        In6_s <= "0110";  -- 6
        In7_s <= "0101";  -- 7
        In8_s <= "1000";  -- 8


        -- Cycle through a few select values
        Sel_s <= "0000"; wait for 100 ns;  -- expect Y_s = "0000"
        Sel_s <= "0011"; wait for 100 ns;  -- expect Y_s = "0011"
        Sel_s <= "0101"; wait for 100 ns;  -- expect Y_s = "0101"
```

```
        Sel_s <= "1000"; wait for 100 ns;  -- expect Y_s = "1000"
        Sel_s <= "1010"; wait for 100 ns;  -- out-of-range, expect "0000"
        wait;
    end process;

end Behavioral;
```

TIMING DIAGRAM– 9 WAY 4 BIT MUX



MULTIPLIER

DESIGN SOURCE FILE – MULTIPLIER

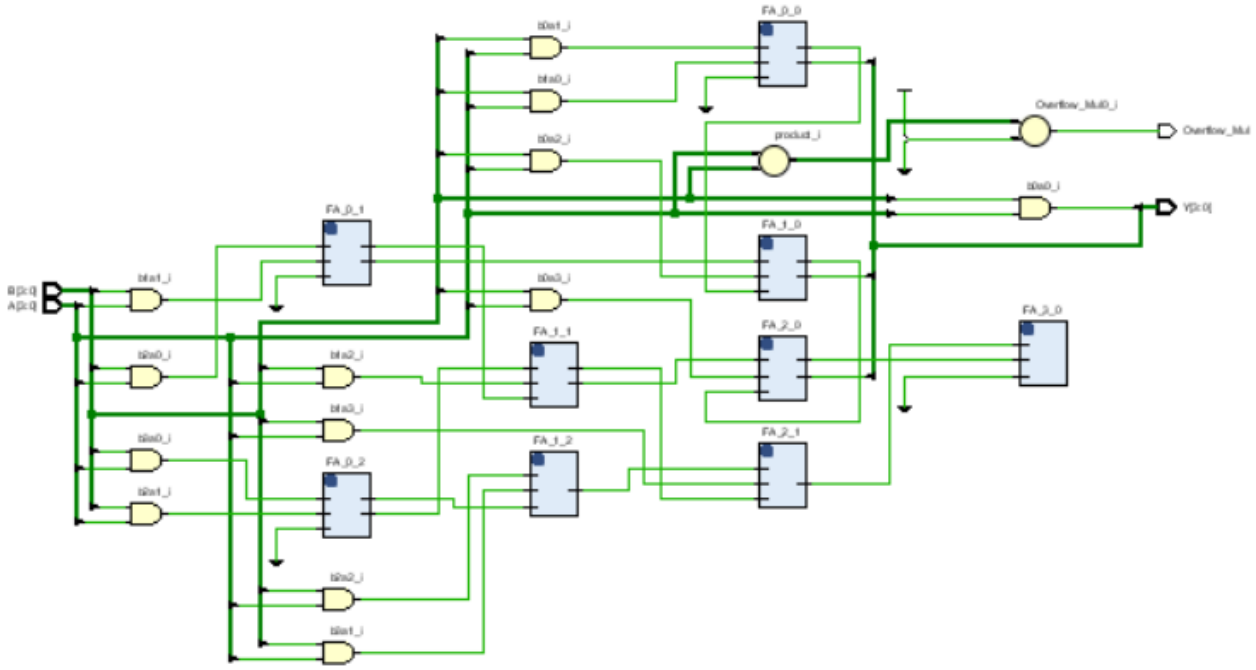
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;  -- Add this for unsigned operations

entity Multiplier_4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow_Mul : out std_logic
           );
end Multiplier_4;
```

```
architecture Behavioral of Multiplier_4 is

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

 SIGNAL b0a0, b0a1, b0a2, b0a3, b1a0, b1a1, b1a2, b1a3, b2a0, b2a1, b2a2, b3a0,
b3a1 :std_logic;
 SIGNAL s_0_1, s_0_2 , s_0_3, s_0_4 , s_1_0,s_1_1, s_1_2, s_1_3,s_1_4, c_0_0 ,
c_0_1, c_0_2 , c_0_3, c_1_0, c_1_1, c_1_2,c_1_3,c_1_4:std_logic;

SIGNAL product : std_logic_vector(7 downto 0);

begin
    FA_0_0 : FA port map(
        A=>b0a1,
        B=>b1a0,
        C_in=>'0',
        S=>s_0_1,
        C_out=>c_0_0
    );

    FA_0_1: FA port map(
        A=>b2a0,
        B=>b1a1,
        C_in=>'0',
        S=>s_1_0,
        C_out=>c_1_0
    );

    FA_1_0: FA port map(
        A=>s_1_0,
        B=>b0a2,
        C_in=>c_0_0,
        S=>s_0_2,
        C_out=>c_0_1 );
```

```
FA_0_2: FA port map(
    A=>b3a0,
    B=>b2a1,
    C_in=>'0',
    S=>s_1_1,
    C_out=>c_1_1
);

FA_1_1: FA port map(
    A=>s_1_1,
    B=>b1a2,
    C_in=>c_1_0,
    S=>s_1_2,
    C_out=>c_1_2
);



FA_2_0: FA port map(
    A=>s_1_2,
    B=>b0a3,
    C_in=>c_0_1,
    S=>s_0_3,
    C_out=>c_0_2
);

FA_1_2: FA port map(
    A=>b2a2,
    B=>b3a1,
    C_in=>c_1_1,
    S=>s_1_3,
    C_out=>c_1_3
);

FA_2_1: FA port map(
    A=>s_1_3,
    B=>b1a3,
    C_in=>c_1_2,
    S=>s_1_4,
    C_out=>c_1_4
);
```

```
FA_3_0: FA port map(
        A=>s_1_4,
        B=>c_0_2,
        C_in=>'0',
        S=>s_0_4,
        C_out=>c_0_3
    );



    b0a0<= B(0) AND A(0);
    b0a1<= B(0) AND A(1);
    b0a2<= B(0) AND A(2);
    b0a3<= B(0) AND A(3);
    b1a0<= B(1) AND A(0);
    b1a1<= B(1) AND A(1);
    b1a2<= B(1) AND A(2);


b1a3<= B(1) AND A(3);
b2a0<= B(2) AND A(0);
b2a1<= B(2) AND A(1);
b2a2<= B(2) AND A(2);
b3a0<= B(3) AND A(0);
b3a1<= B(3) AND A(1);


Y(0)<=b0a0;
Y(1)<=s_0_1;
Y(2)<=s_0_2;
Y(3)<=s_0_3;

 -- Overflow_Mul logic
 --  Overflow_Mul <= '1' when (A > "0011" and B > "0011") else '0';

-- Calculate the full 8-bit product
    product <= std_logic_vector(unsigned(A) * unsigned(B));

    -- Overflow_Mul logic - check if product exceeds 15 (4-bit max)
    Overflow_Mul <= '1' when unsigned(product) > 15 else '0';

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – MULTIPLIER

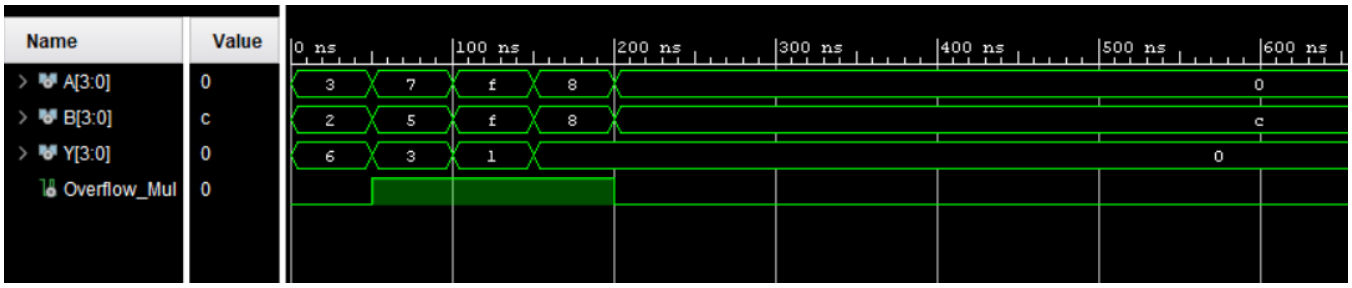


SIMULATION SOURCE FILE – MULTIPLIER

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

entity Multiplier_4Sim is
-- No ports for the testbench
end Multiplier_4Sim;

architecture Behavioral of Multiplier_4Sim is
    -- Component declaration for the DUT (Device Under Test)
    component Multiplier_4 is
        Port (
            A : in STD_LOGIC_VECTOR (3 downto 0);
            B : in STD_LOGIC_VECTOR (3 downto 0);
            Y : out STD_LOGIC_VECTOR (3 downto 0);
            Overflow_Mul : out STD_LOGIC
        );
    end component;

    -- Signals to connect to the DUT
    signal A, B : STD_LOGIC_VECTOR(3 downto 0);
    signal Y : STD_LOGIC_VECTOR(3 downto 0);
```

```
    signal Overflow_Mul : STD_LOGIC;

begin
    -- Instantiate the DUT
    DUT: Multiplier_4
        Port map (
            A => A,
            B => B,
            Y => Y,
            Overflow_Mul => Overflow_Mul
        );

    -- Stimulus process
    process
    begin
        -- Test Case 1: 3 * 2 = 6
        A <= "0011";  -- 3
        B <= "0010";  -- 2
        wait for 50 ns;

        -- Test Case 2: 7 * 5 = 35
        A <= "0111";  -- 7
        B <= "0101";  -- 5
        wait for 50 ns;

        -- Test Case 3: 15 * 15 = 225 (Overflow expected)
        A <= "1111";  -- 15
        B <= "1111";  -- 15
        wait for 50 ns;

        -- Test Case 4: 8 * 8 = 64
        A <= "1000";  -- 8
        B <= "1000";  -- 8
        wait for 50 ns;

        -- Test Case 5: 0 * 12 = 0
        A <= "0000";  -- 0
        B <= "1100";  -- 12
        wait for 50 ns;

        -- End simulation
        wait;
    end process;
end Behavioral;
```

TIMING DIAGRAM – MULTIPLIER



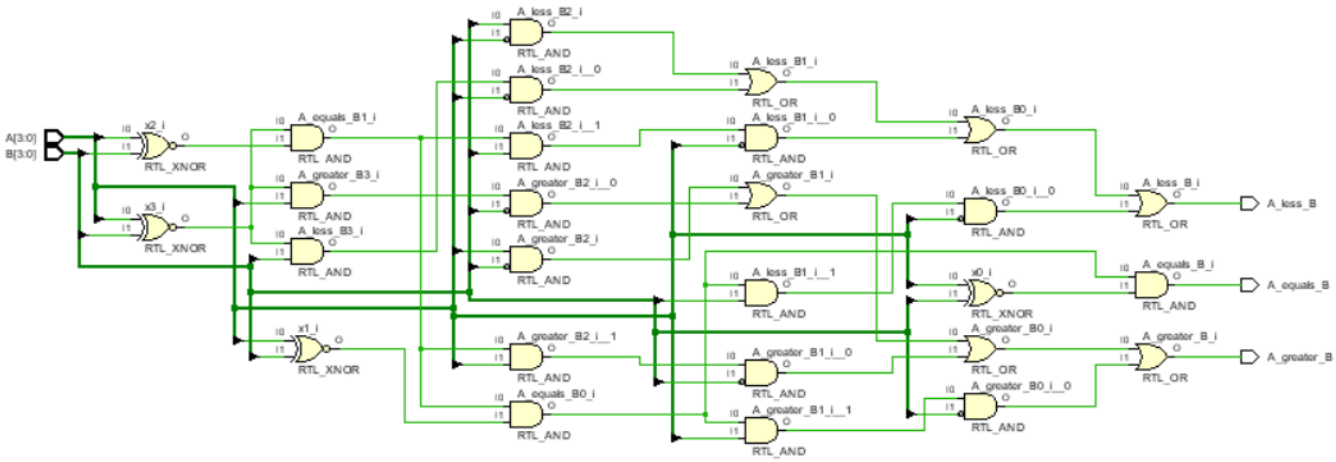
COMPARATOR

DESIGN SOURCE FILE – COMPARATOR

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

entity Comparator_4bit is
port(
        A : in STD_LOGIC_vector(3 downto 0);         -- Input bit A
        B : in STD_LOGIC_vector(3 downto 0 );         -- Input bit B
        A_equals_B : out STD_LOGIC; -- Output for A = B
        A_greater_B : out STD_LOGIC; -- Output for A > B
        A_less_B : out STD_LOGIC   -- Output for A < B
    );
end Comparator_4bit;
architecture Behavioral of Comparator_4bit is

signal x0, x1, x2, x3 : std_logic;

begin
x0 <=A(0) xnor B(0);
x1 <=A(1) xnor B(1);
x2 <=A(2) xnor B(2);
x3 <=A(3) xnor B(3);

-- Equality (A = B)
    A_equals_B <= x3 and x2 and x1 and x0;

    -- Greater than (A > B)
    A_greater_B <= (A(3) and (not B(3))) or (x3 and A(2) and (not B(2))) or(x3 and
x2 and A(1) and (not B(1))) or (x3 and x2 and x1 and A(0) and (not B(0))) ;
```

```
    -- Less than (A < B)
    A_less_B <= (B(3) and (not A(3))) or (x3 and B(2) and (not A(2))) or(x3 and x2
and B(1) and (not A(1))) or (x3 and x2 and x1 and B(0) and (not A(0))) ;

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – COMPARATOR



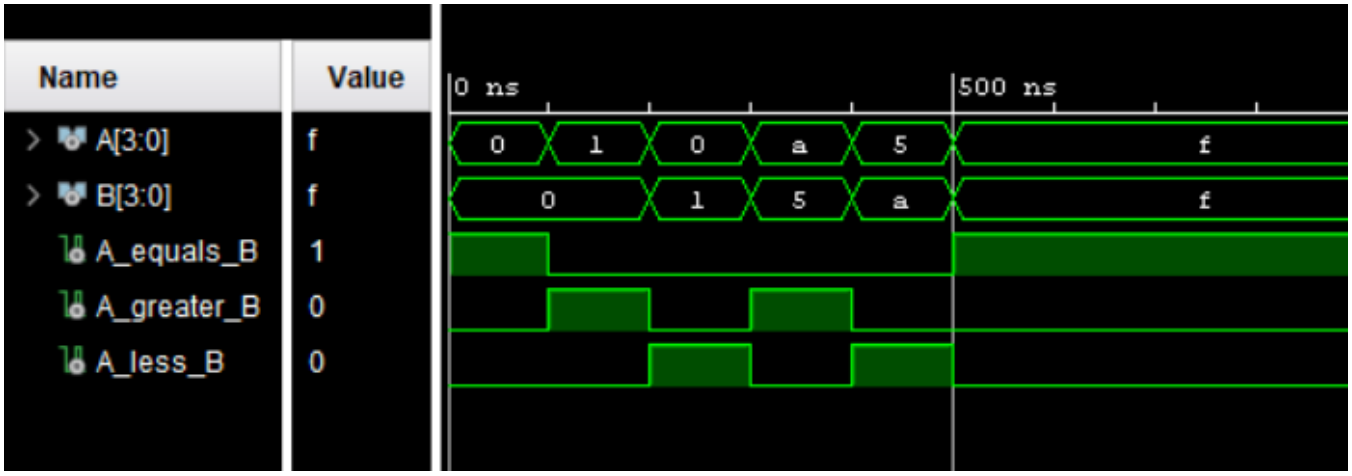
SIMULATION SOURCE FILE – COMPARATOR

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

entity ComparatorSim is
-- Testbench has no ports
end ComparatorSim;

architecture behavioral of ComparatorSim is

    component Comparator_4bit
    port(
        A : in STD_LOGIC_VECTOR(3 downto 0);
        B : in STD_LOGIC_VECTOR(3 downto 0);
        A_equals_B : out STD_LOGIC;
        A_greater_B : out STD_LOGIC;
        A_less_B : out STD_LOGIC
    );
    end component;
```

```
    signal A : STD_LOGIC_VECTOR(3 downto 0);
    signal B : STD_LOGIC_VECTOR(3 downto 0);
    signal A_equals_B : STD_LOGIC;
    signal A_greater_B : STD_LOGIC;
    signal A_less_B : STD_LOGIC;

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: Comparator_4bit
        port map (
            A => A,
            B => B,
            A_equals_B => A_equals_B,
            A_greater_B => A_greater_B,
            A_less_B => A_less_B
        );

    -- Test Process
  process
    begin
        -- Test case 1: A = 0000, B = 0000 (Equal)
        A <= "0000"; B <= "0000";
        wait for 100 ns;

        -- Test case 2: A = 0001, B = 0000 (A > B)
        A <= "0001"; B <= "0000";
        wait for 100 ns;

        -- Test case 3: A = 0000, B = 0001 (A < B)
        A <= "0000"; B <= "0001";
        wait for 100 ns;

        -- Test case 4: A = 1010, B = 0101 (A > B)
        A <= "1010"; B <= "0101";
        wait for 100 ns;

        -- Test case 5: A = 0101, B = 1010 (A < B)
        A <= "0101"; B <= "1010";
        wait for 100 ns;

        -- Test case 6: A = 1111, B = 1111 (Equal)
```

```
        A <= "1111"; B <= "1111";
        wait for 100 ns;

        -- Test complete, stop simulation
        wait;
    end process;

end behavioral;
```

TIMING DIAGRAM – COMPARATOR

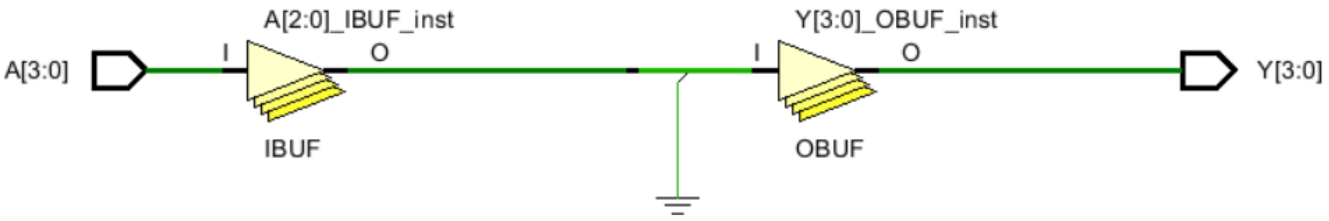


BIT SHIFTER (LEFT)

DESIGN SOURCE FILE – BITSHIFTER(LEFT)

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

entity BitShifterLeft is
    Port (
        A : in STD_LOGIC_VECTOR (3 downto 0);  -- Input 4-bit vector
        Y : out STD_LOGIC_VECTOR (3 downto 0) -- Output 4-bit vector
    );
end BitShifterLeft;

architecture Behavioral of BitShifterLeft is
begin
    -- Shift left by 1 bit
    Y <= A(2 downto 0) & '0';  -- Concatenate the lower 3 bits of A with '0'
end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – BITSHIFTER(LEFT)

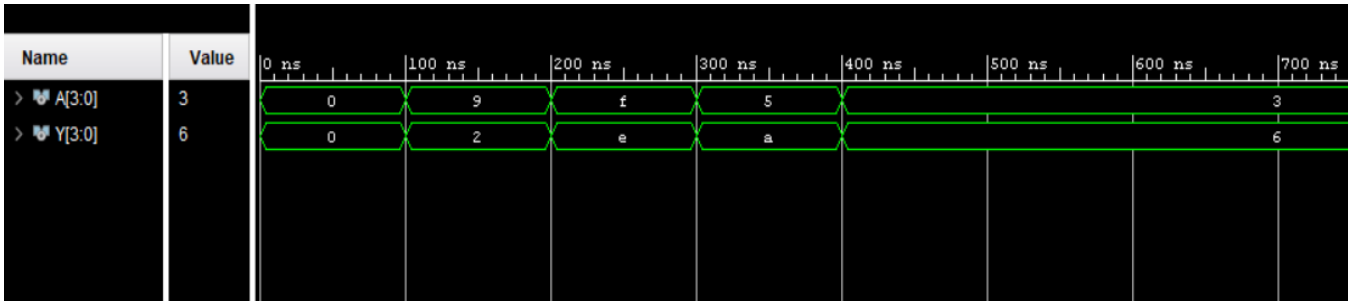


SIMULATION SOURCE FILE – BITSHIFTER(LEFT)

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

entity BitShifterLeftSim is
    -- No ports for the testbench entity
end BitShifterLeftSim;

architecture Behavioral of BitShifterLeftSim is

    -- Component Declaration for the Unit Under Test (UUT)
    component BitShifterLeft
        Port (
            A : in STD_LOGIC_VECTOR (3 downto 0);
            Y : out STD_LOGIC_VECTOR (3 downto 0)
        );
    end component;

    -- Signals for connecting to the UUT
    signal A : STD_LOGIC_VECTOR (3 downto 0);
    signal Y : STD_LOGIC_VECTOR (3 downto 0);


begin

    -- Instantiate the Unit Under Test (UUT)
    UUT: BitShifterLeft
        Port map (
            A => A,
            Y => Y);
```

```
    -- Testbench Process
    stim_proc: process
    begin
        -- Test case 1: A = "0000"
        A <= "0000";
        wait for 100 ns;


       -- Test case 2: A = "1001"
        A <= "1001";
        wait for 100 ns;

        -- Test case 3: A = "1111"
        A <= "1111";
        wait for 100 ns;

        -- Test case 4: A = "0101"
        A <= "0101";
        wait for 100 ns;

        -- Test case 5: A = "0011"
        A <= "0011";
        wait for 100 ns;

        -- Stop the simulation
        wait;
    end process;

end Behavioral;
```

TIMING DIAGRAM – BITSHIFTER(LEFT)

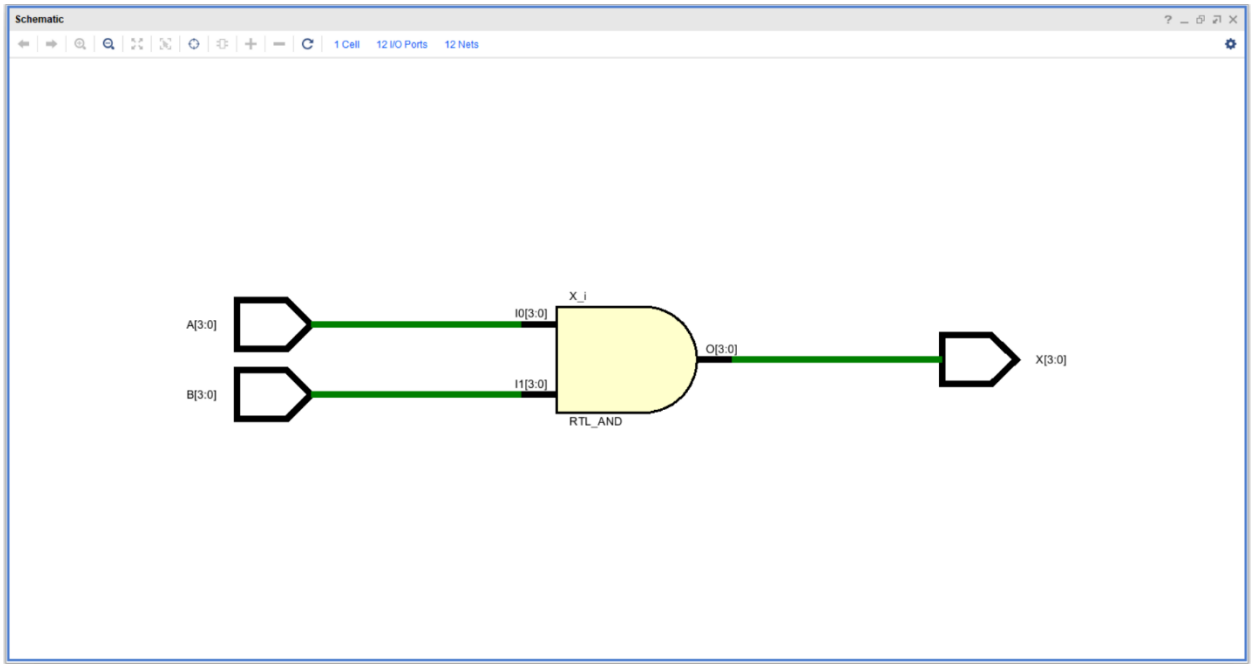


BITWISE – AND

DESIGN SOURCE FILE – BITWISE AND

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

entity BitwiseAND is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           X : out STD_LOGIC_VECTOR (3 downto 0));
end BitwiseAND;

architecture Behavioral of BitwiseAND is

begin
X<=A AND B;


end Behavioral;
```

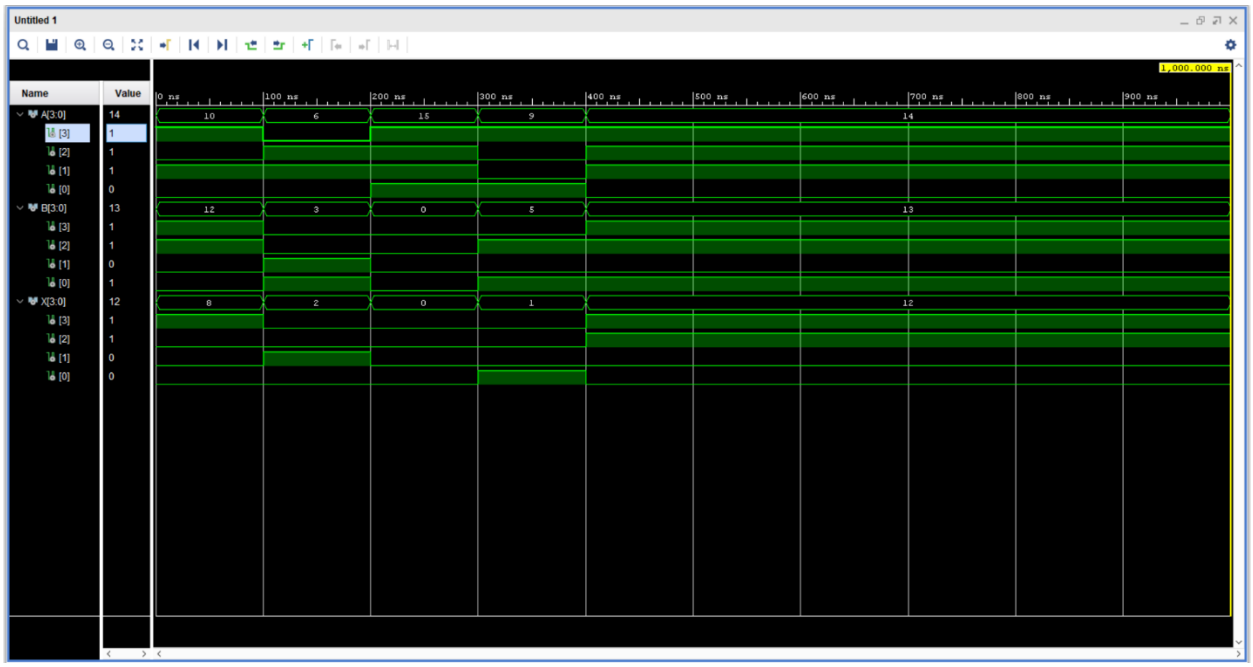
ELABORATED DESIGN SCHEMATIC – BITWISE AND



SIMULATION SOURCE FILE – BITWISE AND

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

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

architecture Behavioral of BitwiseANDSim is

Component BitwiseAND is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           X : out STD_LOGIC_VECTOR (3 downto 0));
end Component;
signal A,B,X: STD_LOGIC_VECTOR(3 downto 0);
begin
 UUT :BitwiseAND Port Map(
             A=>A,
             B=>B,
             X=>X);
 process
 begin
 A<= "1010";
 B<="1100";
 Wait for 100ns;
 A<= "0110";
 B<="0011";
 Wait for 100ns;
 A<= "1111";
 B<="0000";
 Wait for 100ns;
 A<= "1001";
 B<="0101";
 Wait for 100ns;
 A<= "1110";
 B<="1101";
 Wait ;
 end process;

end Behavioral;
```

TIMING DIAGRAM – BITWISE AND

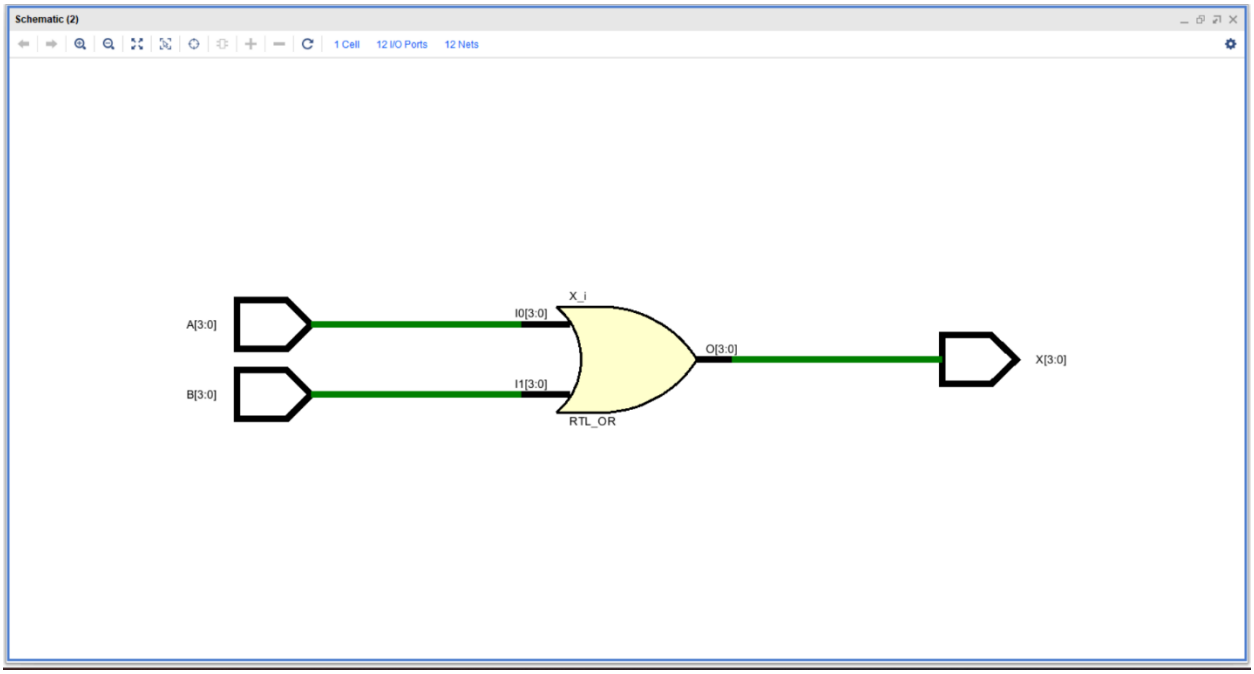


BITWISE OR

DESIGN SOURCE FILE – BITWISE OR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity BitwiseOR is
Port (      A : in STD_LOGIC_VECTOR (3 downto 0);
            B : in STD_LOGIC_VECTOR (3 downto 0);
            X : out STD_LOGIC_VECTOR (3 downto 0));
end BitwiseOR;
architecture Behavioral of BitwiseOR is
begin
X<=A OR B;
end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – BITWISE OR

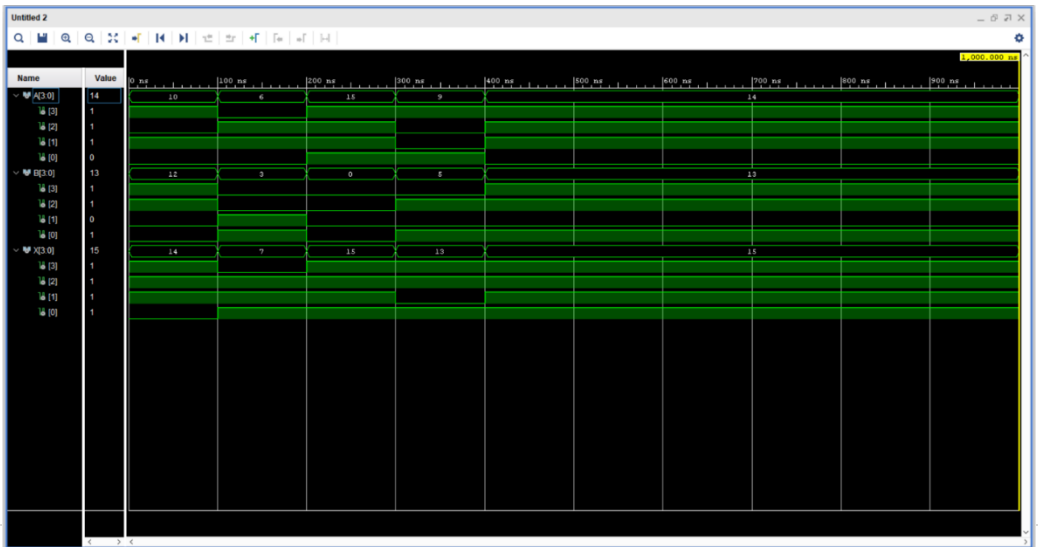


SIMULATION SOURCE FILE – BITWISE OR

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

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

architecture Behavioral of BitwiseORSim is
Component BitwiseOR is
Port (      A : in STD_LOGIC_VECTOR (3 downto 0);
            B : in STD_LOGIC_VECTOR (3 downto 0);
            X : out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal A,B,X: STD_LOGIC_VECTOR(3 downto 0);
begin
```

```
 UUT :BitwiseOR Port Map(
              A=>A,
              B=>B,
              X=>X);
process
begin
 A<= "1010";
 B<="1100";
 Wait for 100ns;
 A<= "0110";
 B<="0011";
 Wait for 100ns;
 A<= "1111";
 B<="0000";
 Wait for 100ns;
 A<= "1001";
 B<="0101";
 Wait for 100ns;
 A<= "1110";
 B<="1101";
 Wait ;
 end process;
end Behavioral;
```

TIMING DIAGRAM – BITWISE OR

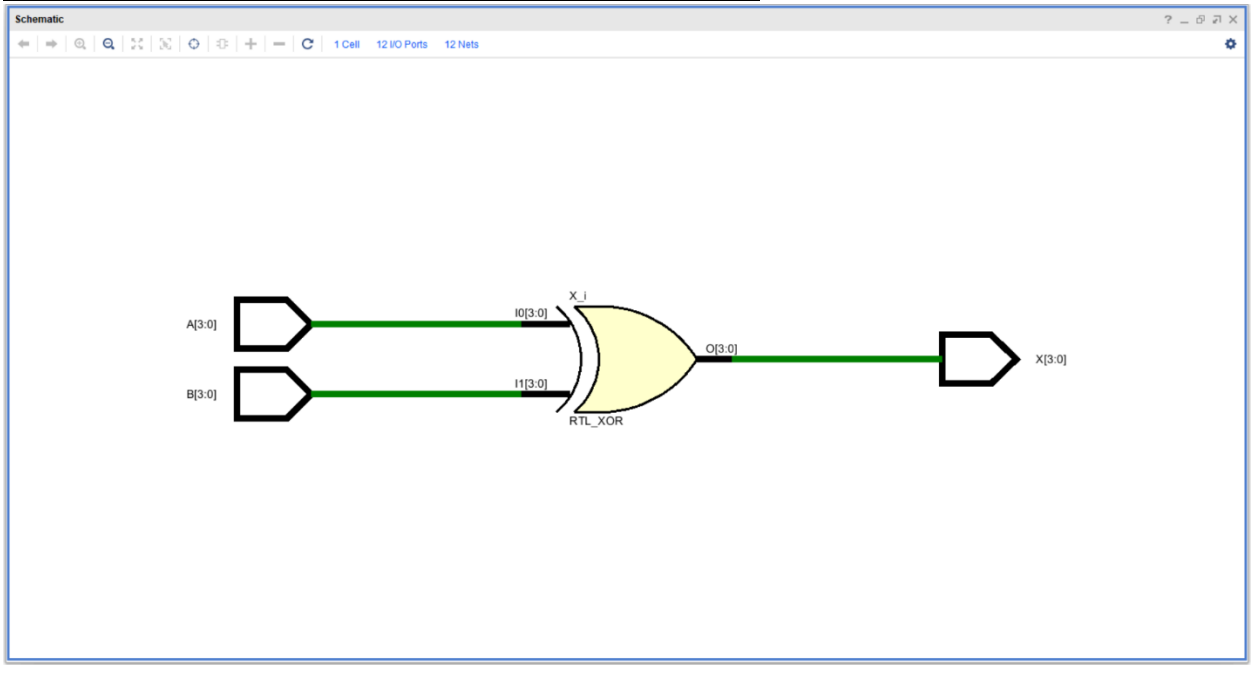


BITWISE XOR

DESIGN SOURCE FILE – BITWISE XOR

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

entity BitwiseXOR is
Port (      A : in STD_LOGIC_VECTOR (3 downto 0);
            B : in STD_LOGIC_VECTOR (3 downto 0);
            X : out STD_LOGIC_VECTOR (3 downto 0));
end BitwiseXOR;
architecture Behavioral of BitwiseXOR is
begin
X<=A XOR B;
end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – BITWISE XOR

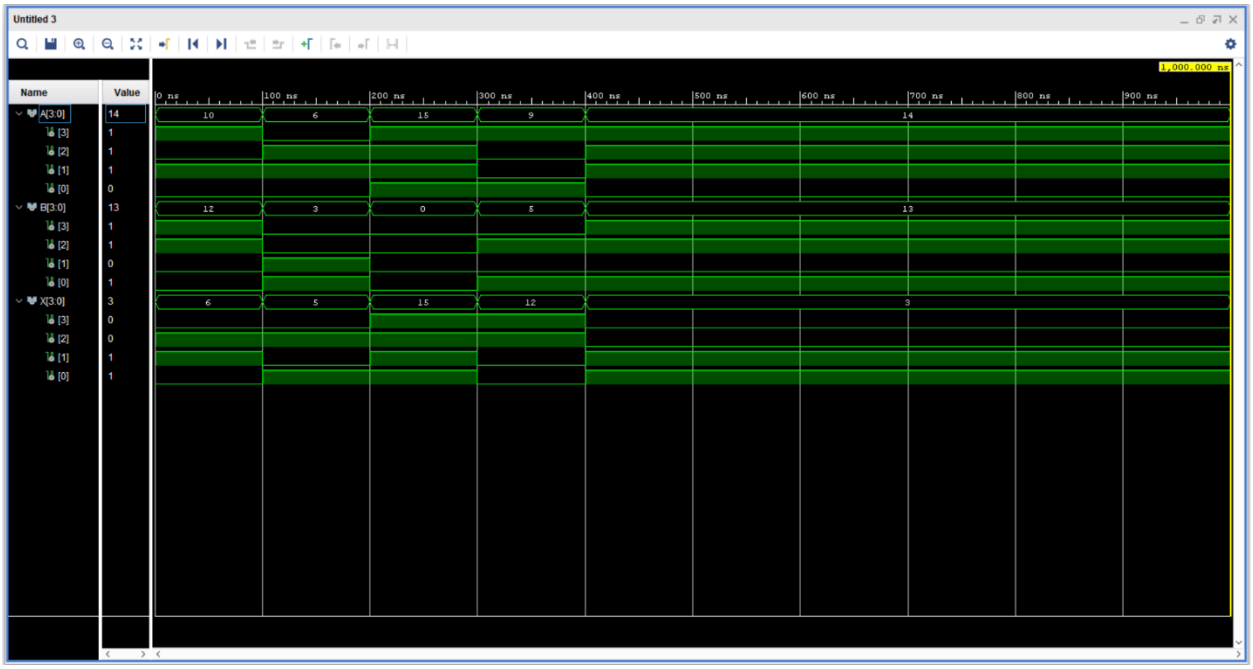


SIMULATION SOURCE FILE – BITWISE XOR

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

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

architecture Behavioral of BitwiseXORSim is
Component BitwiseXOR is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           X : out STD_LOGIC_VECTOR (3 downto 0));
end Component;
signal A,B,X: STD_LOGIC_VECTOR(3 downto 0);
begin
UUT :BitwiseXOR Port Map(
             A=>A,
             B=>B,
             X=>X);
 process
 begin
 A<= "1010";
 B<="1100";
 Wait for 100ns;
 A<= "0110";
 B<="0011";
 Wait for 100ns;
 A<= "1111";
 B<="0000";
 Wait for 100ns;
```

```
 A<= "1001";
 B<="0101";
 Wait for 100ns;
 A<= "1110";
 B<="1101";
 Wait ;
 end process;

end Behavioral;
```

TIMING DIAGRAM – BITWISE XOR



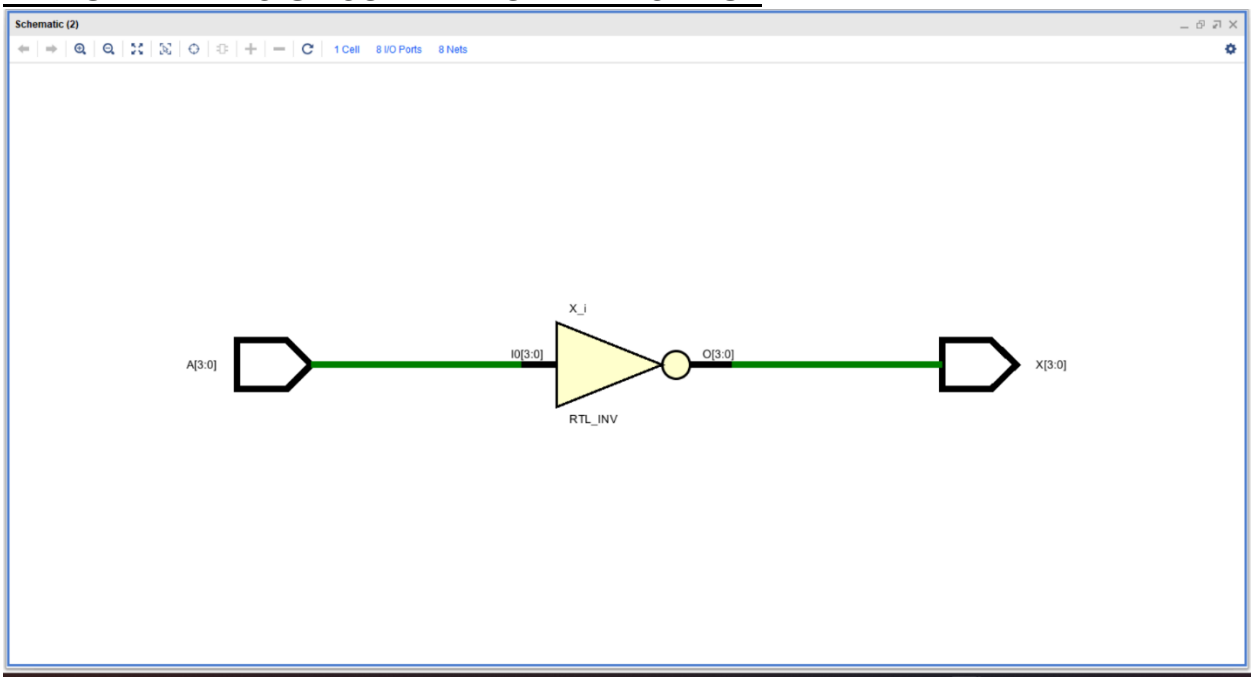
BITWISE NOT

DESIGN SOURCE FILE – BITWISE NOT

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

```
entity BitwiseNOT is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           X : out STD_LOGIC_VECTOR (3 downto 0));
end BitwiseNOT;

architecture Behavioral of BitwiseNOT is

begin
X<= NOT A;

end Behavioral;
```

ELABORATED DESIGN SCHEMATIC – BITWISE NOT

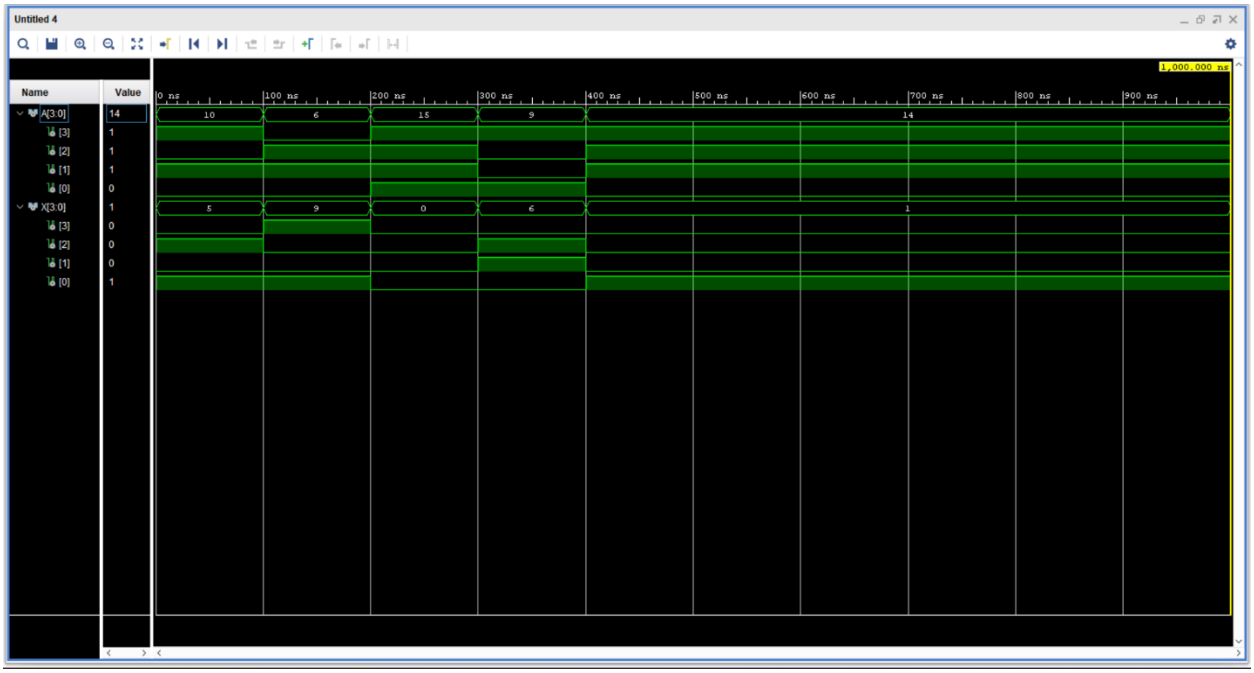


SIMULATION SOURCE FILE – BITWISE NOT

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


entity BitwiseNOTSim is
--  Port ( );
end BitwiseNOTSim;
```

```
architecture Behavioral of BitwiseNOTSim is
component  BitwiseNOT is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           X : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal A,X: STD_LOGIC_VECTOR(3 downto 0);
begin
UUT :BitwiseNOT Port Map(
             A=>A,
             X=>X);
 process
 begin
 A<= "1010";

 Wait for 100ns;
 A<= "0110";

 Wait for 100ns;
 A<= "1111";

 Wait for 100ns;
 A<= "1001";
 Wait for 100ns;
 A<= "1110";

 Wait ;
 end process;
end Behavioral;
```

TIMING DIAGRAM – BITWISE NOT



CONSTRAINT FILE

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


## LEDs
set_property PACKAGE_PIN U16 [get_ports {FourLED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {FourLED[0]}]
set_property PACKAGE_PIN E19 [get_ports {FourLED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {FourLED[1]}]
set_property PACKAGE_PIN U19 [get_ports {FourLED[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {FourLED[2]}]
set_property PACKAGE_PIN V19 [get_ports {FourLED[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {FourLED[3]}]
set_property PACKAGE_PIN P1 [get_ports {ZeroLED}]
set_property IOSTANDARD LVCMOS33 [get_ports {ZeroLED}]
```

```
set_property PACKAGE_PIN W3 [get_ports {AlessB}]
set_property IOSTANDARD LVCMOS33 [get_ports {AlessB}]
set_property PACKAGE_PIN U3 [get_ports {AequalB}]
set_property IOSTANDARD LVCMOS33 [get_ports {AequalB}]
set_property PACKAGE_PIN P3 [get_ports {AgreaterB}]
set_property IOSTANDARD LVCMOS33 [get_ports {AgreaterB}]
set_property PACKAGE_PIN N3 [get_ports {OverflowLED_Mul}]
set_property IOSTANDARD LVCMOS33 [get_ports {OverflowLED_Mul}]
set_property PACKAGE_PIN L1 [get_ports {OverflowLED_Add}]
set_property IOSTANDARD LVCMOS33 [get_ports {OverflowLED_Add}]



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



set_property PACKAGE_PIN U2 [get_ports Anode[0]]
set_property IOSTANDARD LVCMOS33 [get_ports Anode[0]]
set_property PACKAGE_PIN U4 [get_ports Anode[1]]
set_property IOSTANDARD LVCMOS33 [get_ports Anode[1]]
set_property PACKAGE_PIN V4 [get_ports Anode[2]]
set_property IOSTANDARD LVCMOS33 [get_ports Anode[2]]
set_property PACKAGE_PIN W4 [get_ports Anode[3]]
set_property IOSTANDARD LVCMOS33 [get_ports Anode[3]]

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

CONCLUSIONS – IMPROVED VERSION

- **Increased Register Count**: Expanded from 8 to 16 registers for more data storage and flexibility.
- **Wider Instruction & Program Support**: Instruction width increased from 12-bit to 16-bit; Program ROM expanded from 8 to 16 instructions.
- **Enhanced Data Routing**: MUXs upgraded to support 16 inputs.
- **Additional ALU Features**: Added comparison logic (greater than, equal, less than),Bitwise operations, Bitwise shifts and hardware multiplier for advanced arithmetic.
- **Richer Output Flags**: More status flags (e.g., overflow, comparison results) for improved decision-making and branching.
- **Improved Control Logic**: Improved instruction decoder provides sophisticated control with more signals.
- Above upgrades make the improved processor more powerful, flexible, and suitable for complex operations.

SIMULATION AND TESTING METHODOLOGY

- Functionalities of each sub component is tested individually using simulation by providing input stimuli and observing the output behavior. This ensures that each component operates correctly according to its design specifications.
- The simulation and testing of the overall nano processor were carried out using the Vivado Xilinx simulator for functional verification, and the Basys 3 FPGA board was used in the laboratory for real-time hardware implementation and validation.
- The output of register R7, where the result of computations is stored, is connected to a set of LEDs (LD0 – LD3) and a 7-segment display for visual representation.

RESOURCE CONSUMPTION OPTIMIZATION

- Efficient resource utilization was a key focus in the development of our nano processor, aimed at optimizing performance while making full use of the available hardware. This section outlines the strategies implemented to effectively manage and minimize resource consumption, contributing to a streamlined and power-efficient design.
- **Look-Up Tables (LUTs):**LUTs played a central role in the design, serving as compact memory elements that store output values corresponding to specific input combinations. These were employed for various purposes, such as storing instruction sets and executing complex logic functions. Their use helped achieve a highly efficient design by maximizing logical and memory resource utilization.

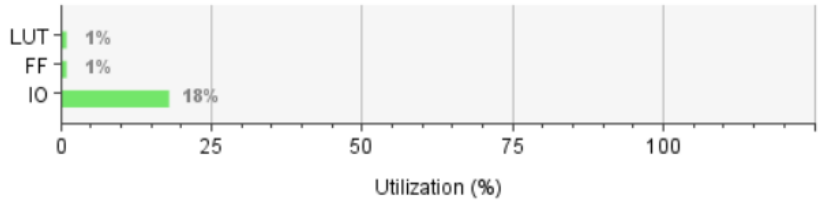
- **Logic Gates:** Fundamental logic gates—AND, OR, NOT, NAND, NOR, and XOR—formed the basis of our digital circuit. These gates enabled basic binary logic operations and were combined in various ways to construct more sophisticated logic structures necessary for the processor's functionality, all while ensuring efficient use of logic resources.
- **Clock Signal:**
  Clock signals were carefully managed to coordinate internal operations and maintain synchronization across components. By optimizing the clock distribution and reducing unnecessary toggling, we were able to minimize power consumption and improve overall energy efficiency.

RESOURCE UTILIZATION

BASIC VERSION

Summary

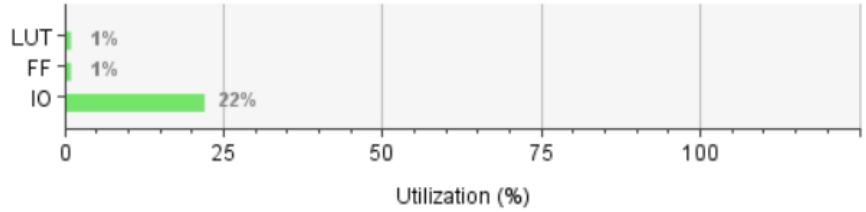
Resource	Utilization	Available	Utilization %
LUT	37	20800	0.18
FF	49	41600	0.12
IO	19	106	17.92



IMPROVED VERSION

Summary

Resource	Utilization	Available	Utilization %
LUT	114	20800	0.55
FF	58	41600	0.14
IO	23	106	21.70



CONTRIBUTION OF THE TEAM MEMBERS

Name	*Work done*	*Time taken*
Ginige D.N.	Nanoprocessor	4 hours
	Nanoprocessor - improved	3 hours
	Instruction decoder - improved	2 hours
	16 way 4bit mux	1 hour 30 minutes
	Multiplier	30 minutes
	Operation Selector	1 hour
	9way 4bit MUX	1 hour 15 minutes
	Program ROM - improved	45 minutes
	Final Report	2 hours
Chathuranga R.M.R	3bit adder	1hour and 30 minutes
	8way 4bit mux	1 hour
	LUT 7segment	1 hour
	Comparator	1 hour 30 minutes
	Bit shifter right	45 minutes
	Bit shifter left	25 minutes
	3bit adder - improved	30 minutes
	Final Report	2 hours
Fernando S.D.	Instruction decoder	2 hours
	Program counter	1 hour
	Program counter - improved	45 minutes
	4bit add sub unit	2 hours
	Bitwise AND operator	45 minutes
	Bitwise OR operator	45 minutes
	Bitwise XOR operator	45 minutes
	Bitwise NOT operator	30 minutes
	Final report	3 hours
Charindith A.J.R.J	Register bank	3 hours
	Register Bank - Improved	1 hour 30 minutes
	Program ROM	1 hour
	2 way 3bit mux	45 minutes
	2way 4bit mux	45 minutes
	Slow clock	1 hour
	Constraint files	30 minutes
	Final report	2 hours