CS1050 Computer Organization and Digital Design

Lab 09 & 10- Nano processor Design Competition

- **Group 36**
  - S.Thanikan 200635E
  - R.Jathushan 200240M
  - K.Thivaharan 200655N
  - T.Sangaran 200575T
  - N.Thenujan 200647R

Lab Task

This lab is about designing a 4-bit Nano Processor capable of executing four instructions efficiently. For designing this circuit, we developed and extended several components that we have already created. They are 4-bit Add/Subtract unit, 3-bit adder, 3-bit Program Counter (PC), several k-way b-bit multiplexers, Register Bank, Program ROM, Instruction Decoder, 7 – Segment Display and slow-clock. Here we used 3, 4, and 12-bit buses for connecting various components as they simplify our design rather than running many wires around.

Since the nano processor only understands machine language, we gave the instructions as binary values. Then we hard coded our program to ROM. Moreover, we used the slow clock to drive our nano processor so that we were able to see the changes as our program executes reducing the clock rate. Furthermore, we verified the functionality of each component with known inputs and outputs through simulation.

We have also additionally added the functionality to indicate when there is a negative result through both LEDs and 7-segment display. We used an additional ROM to show this functionality since the original ROM doesn't give any negative output. Since this was a team project, workload was divided between all five members. During the entire process we designed all the components initially and modified them several times later according to our new thoughts and requirements while combining to design the Nano Processor.

After completing this lab, we were able to design and develop a 4-bit Arithmetic Unit that can add and subtract signed integers, k-way b-bit Multiplexers, Program ROM and to decode instructions to activate necessary components on the Nano Processor and verify their functionality via simulation and by implementing it to our Basys3 board.

Optimized Design Primitives and Slice Logic

❖ Slice Logic

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	37	0	20800	0.18
LUT as Logic	37	0	20800	0.18
LUT as Memory	0	0	9600	0.00
Slice Registers	51	0	41600	0.12
Register as Flip Flop	51	0	41600	0.12
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

* Warning! The Final LUT count, after physical optimizations and full implementation, is typically lower. Run opt_design after synthesis, if not already completed, for a

❖ Primitives

7. Primitives

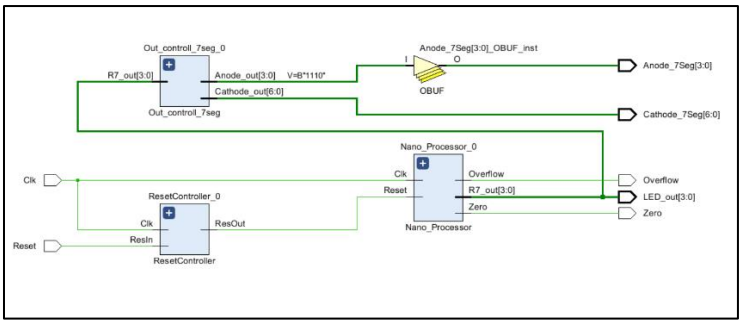
Ref Name	Used	Functional Category
FDRE	35	Flop & Latch
LUT4	19	LUT
OBUF	17	IO
FDCE	15	Flop & Latch
LUT5	13	LUT
LUT3	11	LUT
CARRY4	8	CarryLogic
LUT6	7	LUT
LUT2	4	LUT
IBUF	2	IO
LUT1	1	LUT
FDSE	1	Flop & Latch
BUFG	1	Clock

- You can verify this from the Project file **“Group36_Optimized”** file
- **We reduced our counts by implementing our design using logic gate level circuits instead of using LUT, Flip-flops and Latches as before when Demonstrating at Lab.**

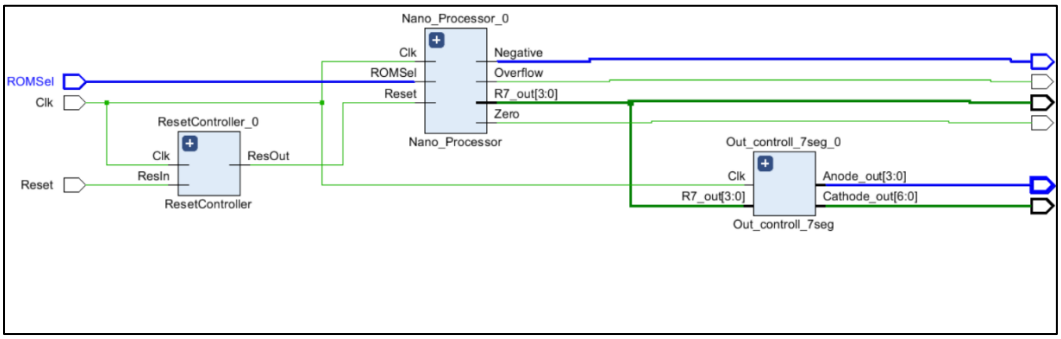
Additional features and optimizations to our design

How high-level entities look like

- **Optimized Design**



- **Additional Features added Design**



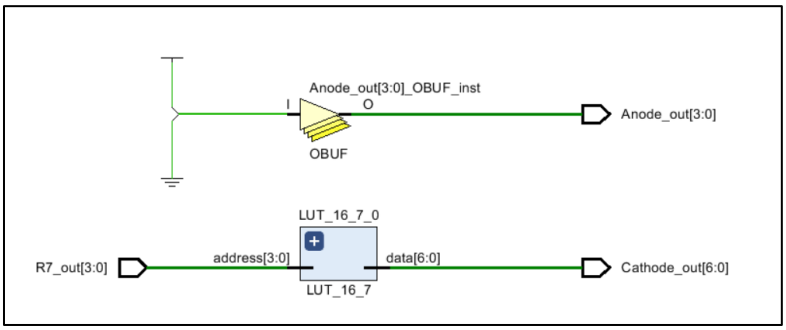
(Blue lines show the final reflection of improvements in the Design)

Part of Reset Controller for the design

- It has a predefined Reset (‘1’) inside the circuit when installing it on Basys3 board. It will prevent the circuit from unexpected behavior because of garbage values stored in Basys3 board.
- It will hold the Program Counter to not count from a garbage value number and make the Registers in the Register Bank to be reset when the circuit installed. Since the circuit output the value of the Register_7, it will be better when we reset the values stored in Registers at early.
- Predefined reset will set to ‘0’ when a manual reset given to circuit. Then for every manual circuit again and again will reset.

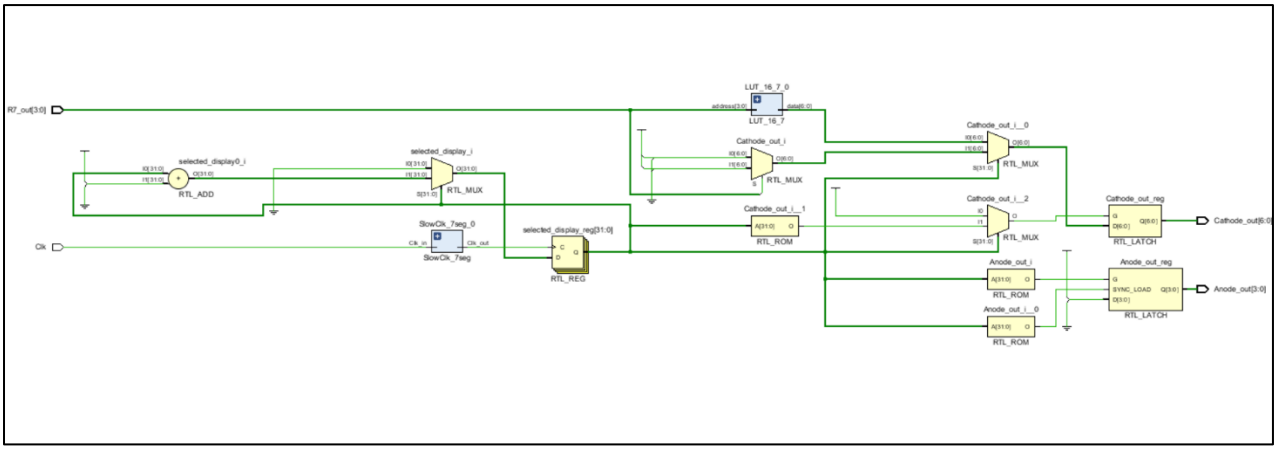
Improved Out control 7seg

- **Previous design**



  - It was a simple design and only the last segment of 7-Segments will light up for the output from Register_7 in the register bank and others will never light up (Anode_out is hard coded to "1110"). Thus, it can only show the magnitude of 4-bit numbers stored in Register_7.
  -

- **Improved design**



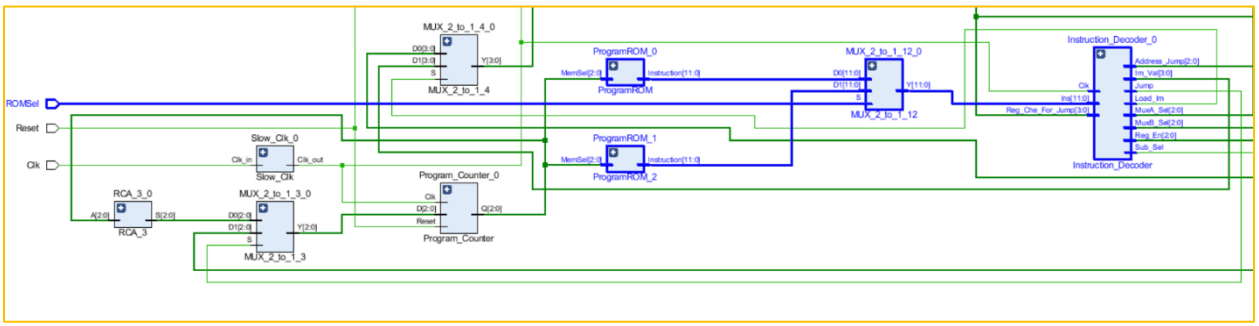
  - Here last two 7-Segments will light up for the output from Register_7 in the register bank and others will never light up.
  - In last two segments right segment will show the magnitude of output and other will show the sign of the output

    If **MSB of digit is '1'** => **minus** will display

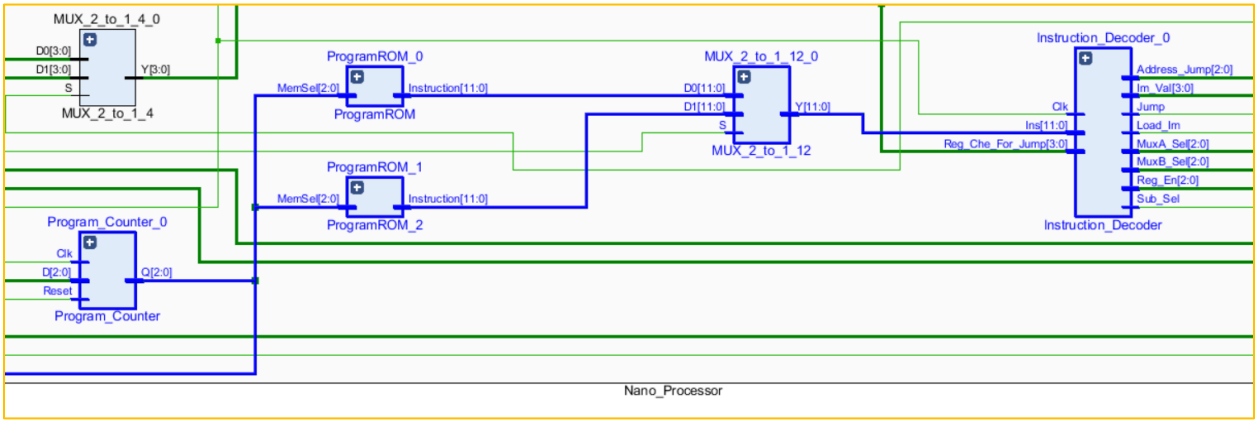
    Else If **MSB of digit is '0'** => **zero** will display

Implementation of Dual ROM by extending the previous design

- For this our design needs an extra input **ROMSel** which directly taken into **NanoProcessor.** If ROMSel = '0' it will select the ROM in the previous design (ProgramROM_0) and else if ROMSel = '1' it will select another new ROM (ProgramROM_1).
- We used a **2-way 12-bit multiplexer** to select Program ROM with the input **ROMSel.**
- The following diagram shows how our design connected with **ROMSel** till **Instruction_Decoder** to get out desired function of the Nano Processor.

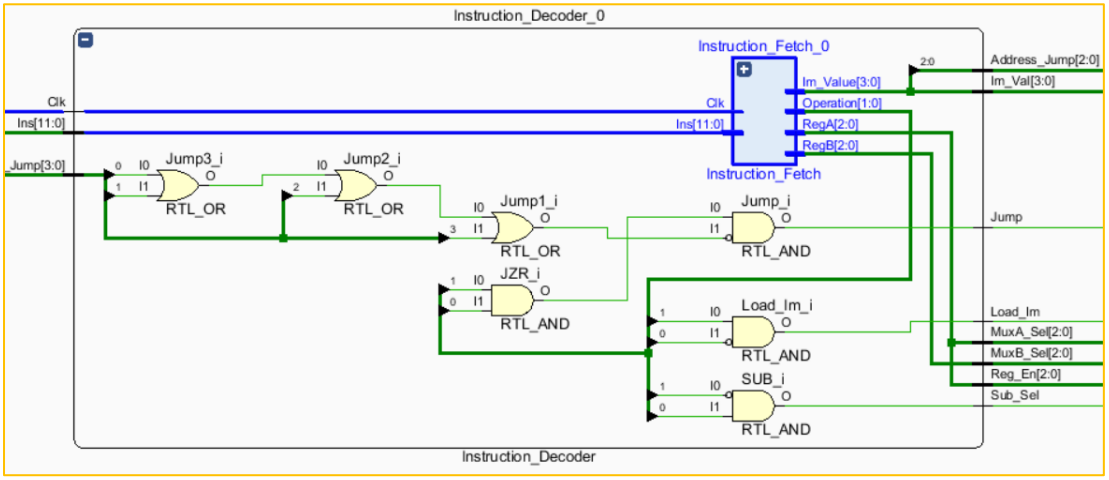


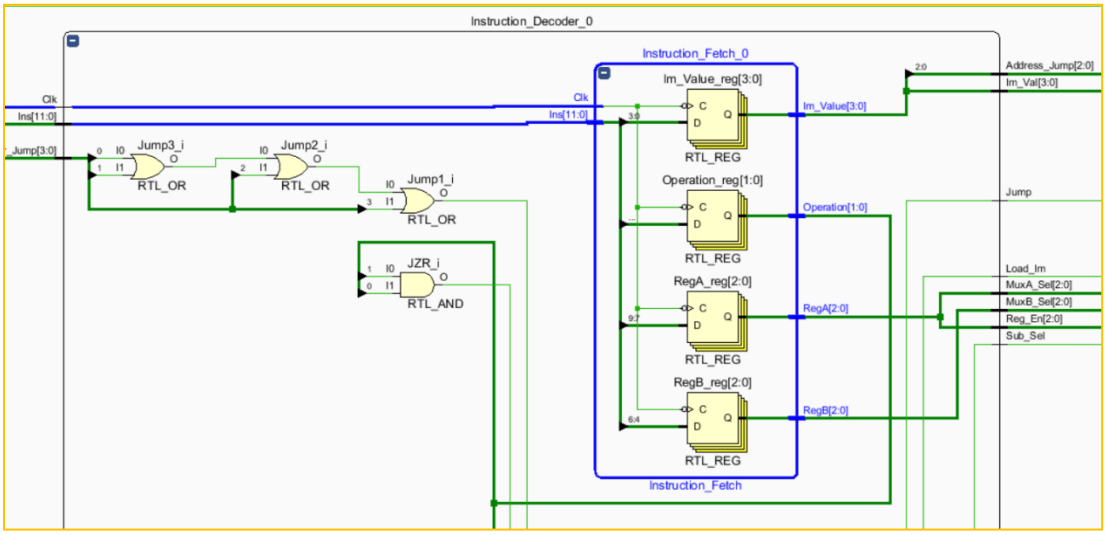
- **Challenges faced when implementing Dual Rom Design and how we managed them**



  - When implementing Dual ROM because of usage of Multiplexer in between the path of Program ROM and Instruction Decoder, the circuit showed some delay in getting instructions from Program ROM and output from Instruction Decoder synchronous to Program Counter.
  - Since Instruction Decoder is the main part of Nano Processor as it gives most of the commands to the components, the delay in the output of Instruction Decoder can affect the output of the whole entity. Therefore, our design showed different output than the expected one.

- Thus, to manage this issue we used a new entity called Instruction Fetch inside the Instruction Decoder as follows,

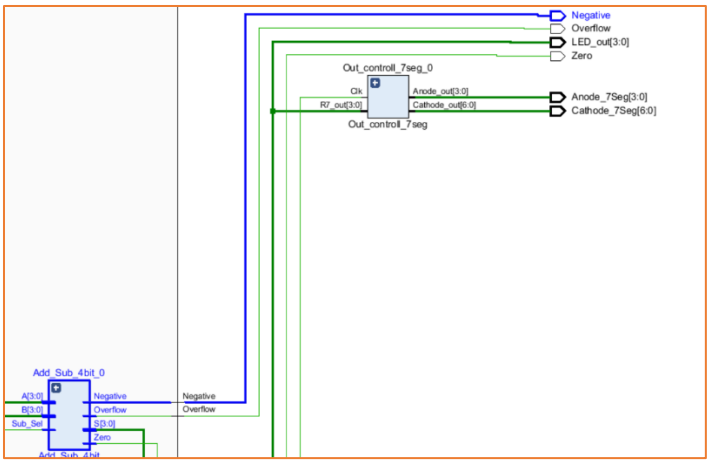




The above entity is work as a clock synchronous circuit which change the outputs when the falling edges of clock ticks. Since the program counter is work as a clock synchronous circuit which changes the outputs when the rising edges of clock ticks, it is sufficient for the Instruction fetcher to fetch instructions to Instruction Decoder at falling edges.

Register Bank also work as a clock synchronous circuit which change the outputs when the falling edges of clock ticks. Making Instruction Decoder as working with falling edges of clock ticks did not make any unexpected outputs.

Additional Negative flag as Output from 4-bit Adder-Subtracter Unit



Slice Logic & Primitives of Additional Features Added Design

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	81	0	20800	0.39
LUT as Logic	81	0	20800	0.39
LUT as Memory	0	0	9600	0.00
Slice Registers	153	0	41600	0.37
Register as Flip Flop	144	0	41600	0.35
Register as Latch	9	0	41600	0.02
F7 Muxes	4	0	16300	0.02
F8 Muxes	0	0	8150	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	112	Flop & Latch
LUT4	35	LUT
FDCE	31	Flop & Latch
LUT5	25	LUT
CARRY4	24	CarryLogic
LUT6	23	LUT
OBUF	18	IO
LUT3	11	LUT
LDCE	8	Flop & Latch
MUXF7	4	MuxFx
LUT2	4	LUT
LUT1	3	LUT
IBUF	3	IO
BUFG	3	Clock
LDPE	1	Flop & Latch
FDSE	1	Flop & Latch

- You can verify this from the Project file "**Group36_withFeatures**"

Specific Instructions for operating the machine.

- **Allocations of inputs and outputs on the BASYS3 Board.**

- **LED0 – LED3** represents the output of Register_7 in the register bank
  - Our group has implemented the Registers consisting of 4-bit two's complement numbers from **-8** to **+7**
  - Most significant digit of register is responsible for the sign bit and its output is connected to **LED3**

- **7-Segment Display**
  - **Optimized Design**
    - In this design the last segment of 7-Segment display will display the output from Register_7 in the register bank and the other three will not light up.
    - It will display the magnitude of 4-bit two' complement number stored in Register_7

❖ **Design with added additional features**

- In this design, the last **two** 7-Segments will display the output from Register_7 in the register bank and others will not light up.

- In last two segments, right segment will display the magnitude of output and while the left segment will display the **sign** of the output.

  If **LED3 is on** => **minus sign** will be displayed
  Else If **LED3 is off** => **zero** will be displayed on this left segment.

- **Center Button** --> for **Reset** operation
  - We reduced the clock speed of the internal clock of board from 100MHz to 0.5Hz. Therefore, **hold Reset for at least 2 to 3 seconds** to reset the whole entity.

- **LED13-LED15** --> **Flags of 4-bit Adder-Subtracter units**
  - LED15 --> Zero Flag
  - LED14 -->Overflow Flag
  - LED13 -->Negative Flag **(Additional Features added Design)**

  Above flags will light up to show the results of Addition and Subtraction operations.

- **Switch15** --> **Only applicable for Additional Features added Design**
  - To switch between two program ROMs included in the design
    If **Switch15=>OFF,** then ROM_0
    Else **Switch15=>ON,** then ROM_1 is activated
  - **If you switch between ROMs,** you must press reset for proper execution of the programs in ROMs.

➢ **Expected Outputs of the designs**

- Each design needs a manual reset to start executing the program in the ROM. Because Program Counter and Registers should be initialized to **Zero** at the start, and Program Counter will wait for a reset signal to change its count to start executing the program in the selected ROM

- **Program inside ROM in the Optimized Design,** is a calculation of total of the numbers from 1 to 3
    - First initialize Register_7 with 0
    - Then add 3,2,1 to Register_7 in the order mentioned and show **0,3,5,6** in seven segment display respectively. (Step 2)
    - Finally, after addition of all numbers from 1 to 3, program will stay in a waiting condition **(shows 6)** until another reset given manually.
    - We can also reset the program to run from the start during both the execution of step 2 and after waiting condition.

- **In Features added Design,** it has two ROMs.

- ROM_0 has the same program as the one in the Optimized design
- ROM_1 has a simple counter from -8 to 6 with incrementation of +2 to show the additional functionality of displaying negative numbers
    - First initializes Register_7 with 0
    - Then put immediate value of -8 into Register_7
    - Then increment the value in Register_7 by +2 in each iteration and updates it
    - The output sequence for this program will be as: -8, -6, -4, -2, 00, 02, 04, 06, -8, -6, ……

Conclusion

From this lab, we were able to design and develop a 4-bit arithmetic unit that can add and subtract signed integers. We designed various components and integrated them together to achieve this. Since each component was designed separately by group members, it was very difficult to combine them to design nano processor as design idea of each differs from others. As a remedy we had many group discussions and meetings and understood the design of each component.

Then we designed the nano processor that does the basic task and later implemented new features and increased the efficiency. We also faced many issues in the process of designing the processor. First, we did not get the desired output on the board. We solved this implementing a Reset controller which prevents garbage values getting outputted on the 7segment before giving the initial reset. We also faced issues on showing the negative signal on the 7segment display, which we corrected after many group discussions.

At the beginning we designed all the components according to the given instructions and the instruction set. But later we modified them in various ways to reduce the lines of code, number of bits used and wires to enhance the performance of the Nano Processor. We also simulated each component individually before adding it to our processor.

As it is very difficult to deal with many inputs and outputs, we have used buses here in designing components to implement both the inputs and outputs. By the Vivado software each wire in the bus can easily be separately accessed based on a zero-based index. We also used hierarchical design techniques and simple components wherever possible to design complex components.

Furthermore, through this group project, we also developed our team-working skills such as communication, coordination, and sharing responsibilities.

Individual contributions of each group member

Member	Individual Contribution	Worked Time
S.Thanikan 200635E	• Nano_Proccessor_with_7SegOut, Nano processor, Instruction Decoder, Instruction fetch, Reset Controller • TB_Instruction_Decoder • Optimizing Designs and Added some additional Features by getting essential components from members. • Described additional features of the design in Report	45 Hours
R.Jathushan 200240M	• 8-way 1-bit, 8-way 4-bit, 2-way 3-bit, 2-way 4-bit and 2-way 12-bit Multiplexers, Nano processor • Test Bench files for each multiplexer • Report Assembling by getting each components Source files, TB files, Timing diagrams and schematics from other members.	40 Hours
K.Thivaharan 200655N	• ProgramROM_0, ProgramROM_1, Nano processor, RCA_3 • TB_Add_Sub, TB_Program_Counter, TB_Program_ROM_0, TB_Program_ROM_1, TB_RCA_3 • Lab activity and Conclusion of Report	35 Hours
T.Sangaran 200575T	• Slow_Clk, SlowClk_7seg, Registers & Register Bank, Nano processor, 3-8 Decoder • Test bench files for each slow clocks, TB_Registers, TB_Register_Bank • Described Operation of our designs in Report	35 Hours
N.Thenujan 200647R	• Nano processor, Out_Controll_7seg, LUT_16_7_0, Add_Sub_4bit, Program_Counter • TB_Instruction_Fetch, TB_Nano_Proccessor_with_7SegOut, TB_Nano processor, TB_Out_Controll_7seg, TB_LUT_16_7_0, TB_Program_Counter • Constraint file Basys3Labs.xdc	40 Hours

Optimized design components

Nano processor with 7 segment displays

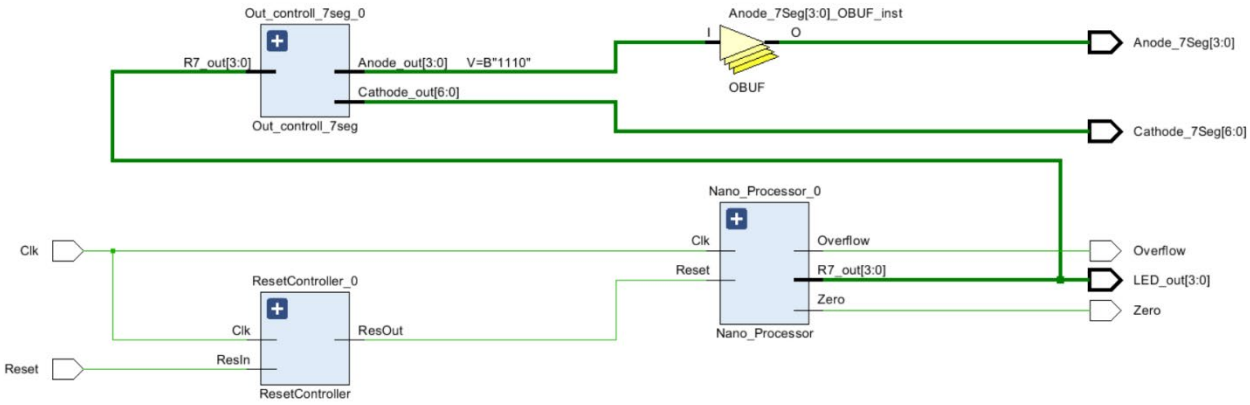
---

01. Design source VHDL code of nano processor with 7 seg out.

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

entity Nano_Processor_with_7SegOut is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC;
           LED_out : out STD_LOGIC_VECTOR (3 downto 0);
           Anode_7Seg : out STD_LOGIC_VECTOR (3 downto 0);
           Cathode_7Seg : out STD_LOGIC_VECTOR (6 downto 0));
end Nano_Processor_with_7SegOut;

architecture Behavioral of Nano_Processor_with_7SegOut is
    component Nano_Processor
        Port ( Clk : in STD_LOGIC;
               Reset : in STD_LOGIC;
               Overflow : out STD_LOGIC;
               Zero : out STD_LOGIC;
               R7_out : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    component Out_controll_7seg
        Port (
               R7_out : in STD_LOGIC_VECTOR (3 downto 0);
               Anode_out : out STD_LOGIC_VECTOR (3 downto 0);
               Cathode_out : out STD_LOGIC_VECTOR (6 downto 0));
    end component;

    component ResetController
        Port ( ResIn : in STD_LOGIC;
               Clk : in STD_Logic;
               ResOut : out STD_LOGIC);
    end component;

    signal R7_output:std_logic_vector(3 downto 0);
    signal ResINPUT:std_logic;

begin
ResetController_0:ResetController
    port map(
        ResIn=>Reset,
        Clk=>Clk,
        ResOut=>ResINPUT    );

Nano_Processor_0:Nano_processor
    port map(
           Clk=>Clk,
           Reset=>ResINPUT,
           Overflow=>Overflow,
```

```
            Zero=>Zero,
            R7_out=>R7_output     );

LED_out<=R7_output;

Out_controll_7seg_0:Out_controll_7seg
    Port map (
            R7_out=>R7_output,
            Anode_out=>Anode_7Seg,
            Cathode_out=>Cathode_7Seg);

end Behavioral;
```

02. RTL Schematic diagram of the nano processor with 7 seg out.

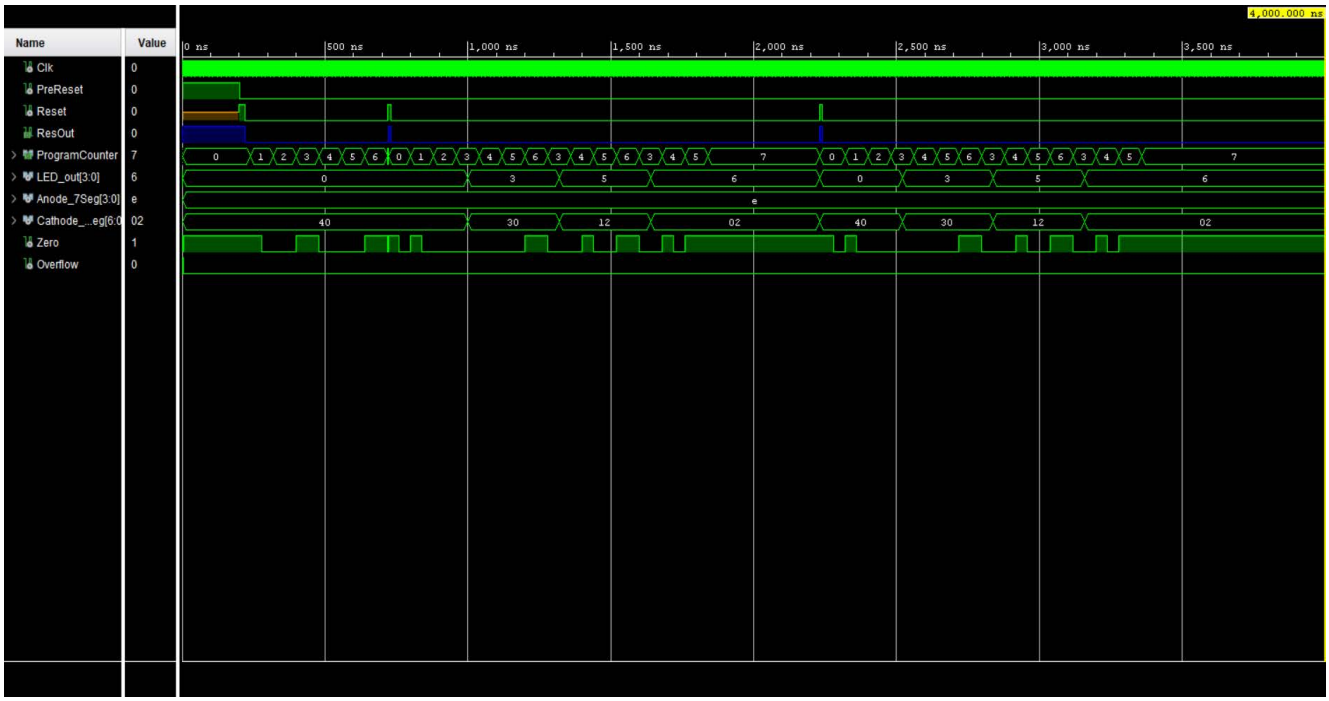


03. Behavioral simulation Code for nano processor with 7 seg out.

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

entity TB_Nano_Processor_with_7SegOut_1 is
--  Port ( );
end TB_Nano_Processor_with_7SegOut_1;

architecture Behavioral of TB_Nano_Processor_with_7SegOut_1 is
component
    Nano_Processor_with_7SegOut
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC;
           LED_out : out STD_LOGIC_VECTOR (3 downto 0);
           Anode_7Seg : out STD_LOGIC_VECTOR (3 downto 0);
           Cathode_7Seg : out STD_LOGIC_VECTOR (6 downto 0));
end component;
signal Clk : std_logic := '0';
signal Reset : std_logic;
signal overflow,Zero,Negative : std_logic;
```

```
signal LED_out,Anode_7Seg : STD_LOGIC_VECTOR (3 downto 0);
signal Cathode_7Seg :STD_LOGIC_VECTOR (6 downto 0);
begin
UUT:Nano_Processor_with_7SegOut
    port map(
        Clk=>Clk,
        Reset=>Reset,
        OverFlow=>Overflow,
        Zero=>Zero,
        LED_out => LED_out,
        Anode_7Seg => Anode_7Seg,
        Cathode_7Seg => Cathode_7Seg);
process
begin
    wait for 1ns;
    Clk<=not(Clk);
end process;

process
begin
    wait for 200ns;
    Reset<='1';
    wait for 20ns;
    Reset<='0';
    wait for 500ns;
    Reset<='1';
    wait for 10ns;
    Reset<='0';
    wait for 1500ns;
    Reset<='1';
    wait for 10ns;
    Reset<='0';
    wait;
end process;
end Behavioral;
```

04. Timing Diagram for nano processor with 7 seg out.



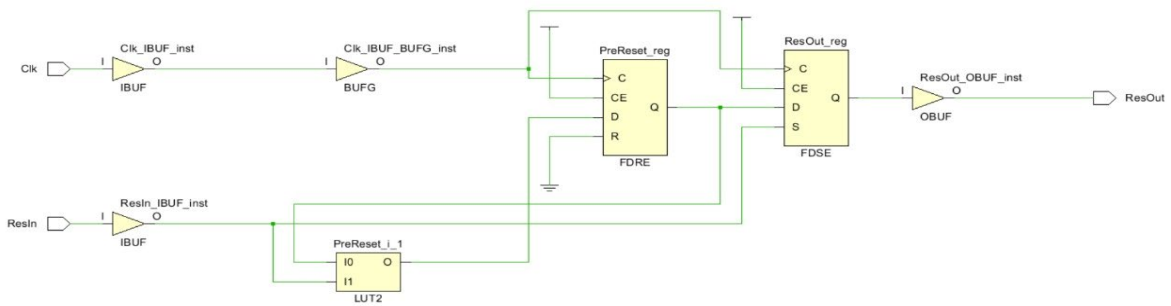
Reset Controller

---

01. Design source VHDL code of reset controller.

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

entity ResetController is
    Port ( ResIn : in STD_LOGIC;
           Clk   : in STD_LOGIC;
           ResOut : out STD_LOGIC);
end ResetController;

architecture Behavioral of ResetController is
signal PreReset:std_logic:='1';

begin

    process (Clk)
    begin
        if (rising_edge(Clk)) then
        if ResIn='1' then
                ResOut<='1';
                PreReset<='0';
            elsif PreReset='1' then
                ResOut<='1';
            else
                ResOut<='0';
            end if;
        end if;
    end process;

end Behavioral;
```

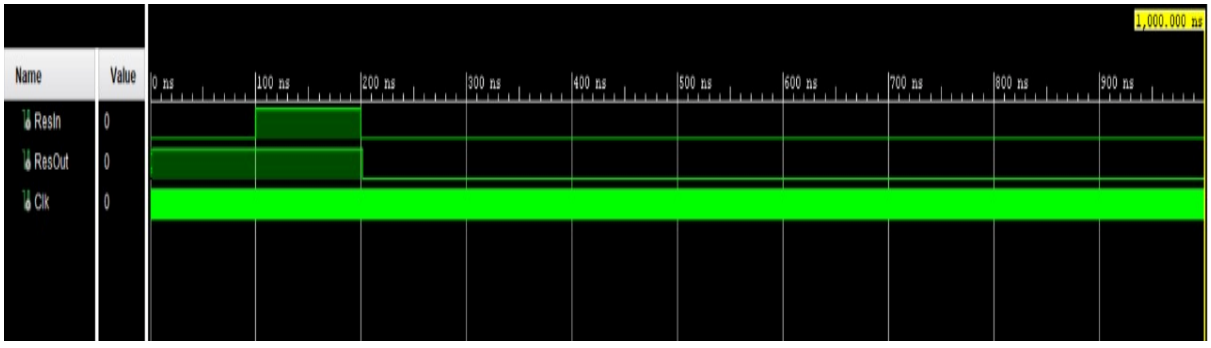
02. RTL Schematic diagram of reset controller.



03. Behavioral simulation source Code for the reset controller.

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

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

architecture Behavioral of Sim_Reset_Controller is
component ResetController
Port ( ResIn : in STD_LOGIC;
           Clk   : in STD_LOGIC;
           ResOut : out STD_LOGIC);
end component;

signal ResIn,ResOut:std_logic;
signal Clk:std_logic := '0';
begin
UUT:ResetController
    port map(
        Clk=>Clk,
        ResIn=>ResIn,
        ResOut => ResOut);
process
begin
    wait for 1ns;
    Clk<=not(Clk);
end process;


process
begin
    ResIn <= '0';
    wait for 100ns;
    ResIn <= '1';
    wait for 100ns;
    ResIn <= '0';
    wait;
end process;
end Behavioral;
```

04. Timing Diagram for the reset controller.



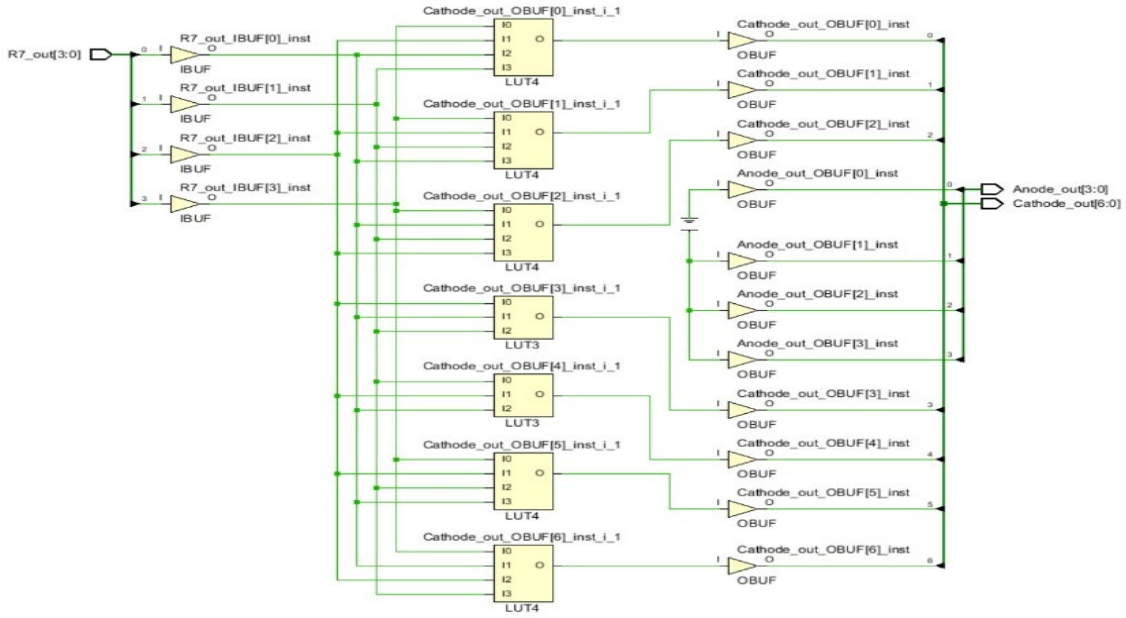
Out_controll_7seg

---

01. Design source VHDL code of Out_controll_7seg.

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


entity Out_controll_7seg is
    Port ( R7_out : in STD_LOGIC_VECTOR (3 downto 0);
           Anode_out : out STD_LOGIC_VECTOR (3 downto 0);
           Cathode_out : out STD_LOGIC_VECTOR (6 downto 0));
end Out_controll_7seg;

architecture Behavioral of Out_controll_7seg 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 S_7seg:std_logic_vector(6 downto 0);

begin
    LUT_16_7_0:LUT_16_7
            port map(
                    address=>R7_out,
                    data=>S_7Seg    );
                Anode_out<="1110";
                Cathode_out<=S_7Seg;

end Behavioral;
```

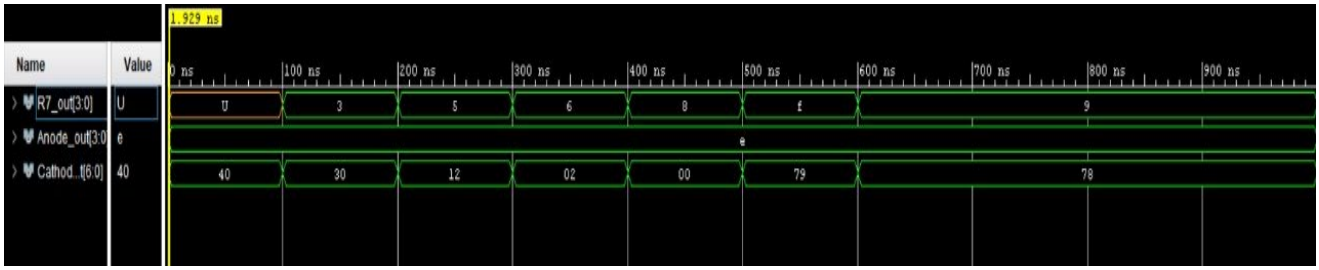
02. RTL Schematic diagram of Out_controll_7seg.



03. Behavioral simulation source Code for the Out_controll_7seg.

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

entity Sim_Out_Control_7Seg is
--  Port ( );
end Sim_Out_Control_7Seg;

architecture Behavioral of Sim_Out_Control_7Seg is
component Out_controll_7seg
    Port ( R7_out : in STD_LOGIC_VECTOR (3 downto 0);
           Anode_out : out STD_LOGIC_VECTOR (3 downto 0);
           Cathode_out : out STD_LOGIC_VECTOR (6 downto 0));
end component ;

signal R7_out,Anode_out :STD_LOGIC_VECTOR (3 downto 0);
signal Cathode_out :STD_LOGIC_VECTOR (6 downto 0);

begin
UUT:Out_controll_7seg
    port map(
        R7_out=>R7_out,
        Anode_out => Anode_out,
        Cathode_out=>Cathode_out);

process
begin
    wait for 100ns;
    R7_out <= "0011";
    wait for 100ns;
    R7_out <= "0101";
    wait for 100ns;
    R7_out <= "0110";
    wait for 100ns;
    R7_out <= "1000";
    wait for 100ns;
    R7_out <= "1111";
    wait for 100ns;
    R7_out <= "1001";
    wait;
end process;

end Behavioral;
```

04. Timing Diagram for the Out_controll_7seg.



Nano Processor

---

01. Design source VHDL code of nano processor.

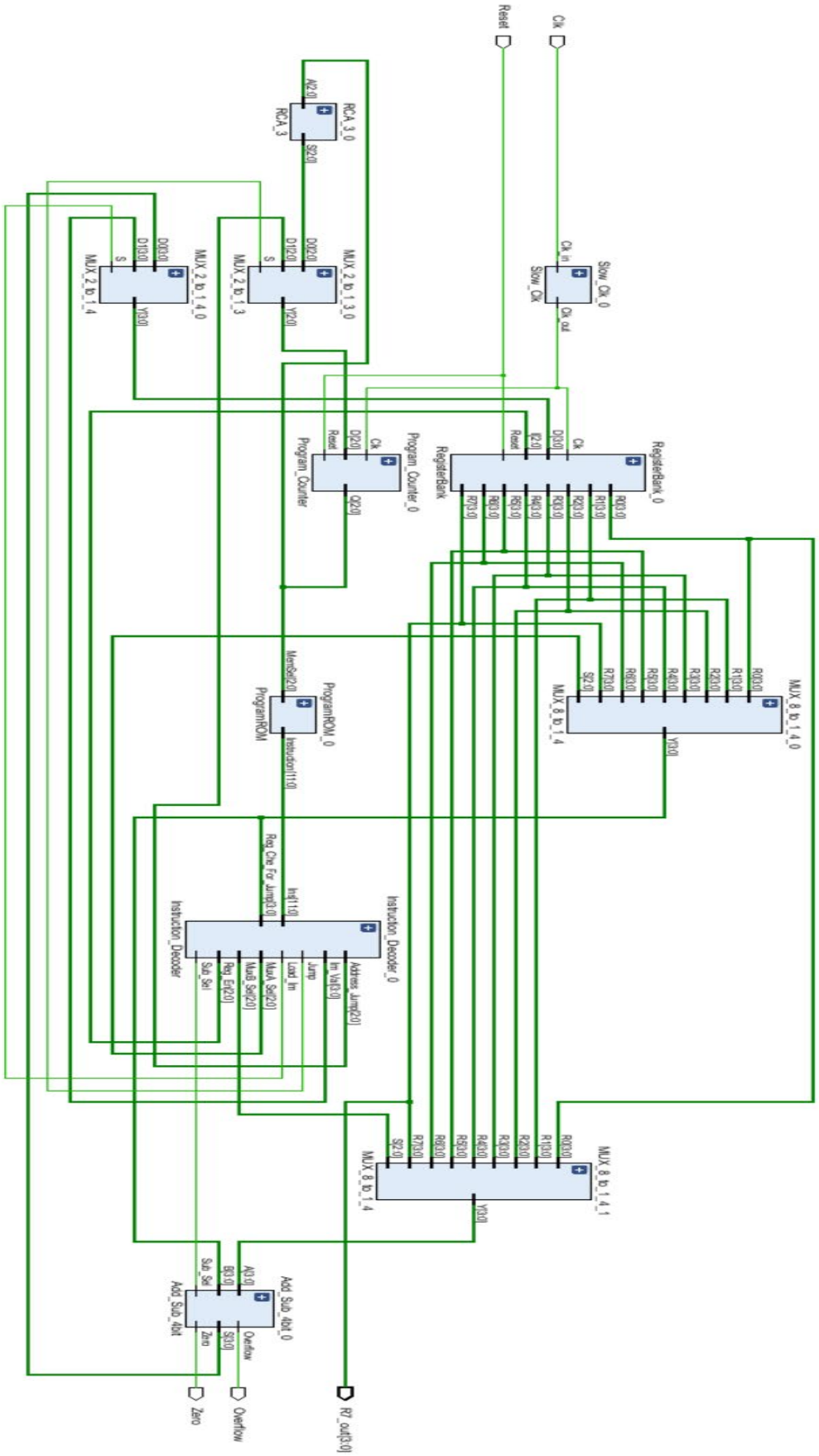
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity Nano_Processor is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC;
           --Negative:out STD_Logic;
           R7_out : out STD_LOGIC_VECTOR (3 downto 0));
end Nano_Processor;

architecture Behavioral of Nano_Processor is
component Slow_Clk
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end component;
signal SlowClk:std_logic;
component MUX_8_to_1_4
    Port ( R0 : in STD_LOGIC_VECTOR (3 downto 0);
           R1 : in STD_LOGIC_VECTOR (3 downto 0);
           R2 : in STD_LOGIC_VECTOR (3 downto 0);
           R3 : in STD_LOGIC_VECTOR (3 downto 0);
           R4 : in STD_LOGIC_VECTOR (3 downto 0);
           R5 : in STD_LOGIC_VECTOR (3 downto 0);
           R6 : in STD_LOGIC_VECTOR (3 downto 0);
           R7 : in STD_LOGIC_VECTOR (3 downto 0);
           S : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

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

component Add_Sub_4bit
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
```

```
            S : out STD_LOGIC_VECTOR (3 downto 0);
            Overflow : out STD_LOGIC;
            Sub_Sel : in STD_LOGIC;
            Zero : out STD_LOGIC
            );
end component;

component RCA_3
    Port (  A : in STD_LOGIC_VECTOR (2 downto 0);
            S: out STD_LOGIC_VECTOR (2 downto 0);
            C_out : out STD_LOGIC);
end component;

component Instruction_Decoder
    Port (
            Ins : in STD_LOGIC_VECTOR (11 downto 0);
            Reg_Che_For_Jump : in STD_LOGIC_VECTOR (3 downto 0);
            Reg_En : out STD_LOGIC_VECTOR(2 downto 0);
            Load_Im : out STD_LOGIC;
            Im_Val : out STD_LOGIC_VECTOR(3 downto 0);
            MuxA_Sel : out STD_LOGIC_VECTOR (2 downto 0);
            MuxB_Sel : out STD_LOGIC_VECTOR (2 downto 0);
            Sub_Sel : out STD_LOGIC;
            Jump : out STD_LOGIC;
            Address_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component MUX_2_to_1_3
    Port ( D0 : in STD_LOGIC_VECTOR (2 downto 0);
            D1 : in STD_LOGIC_VECTOR (2 downto 0);
            S : in STD_LOGIC;
            Y : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component MUX_2_to_1_4
    Port ( D0 : in STD_LOGIC_VECTOR(3 downto 0);
            D1 : in STD_LOGIC_VECTOR(3 downto 0);
            S : in STD_LOGIC;
            Y : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component ProgramROM
    Port (  MemSel : in STD_LOGIC_VECTOR (2 downto 0);
            Instruction : out STD_LOGIC_VECTOR (11 downto 0));
end component;

component Program_Counter
    Port ( Clk : in STD_LOGIC;
            Reset : in STD_LOGIC;
            D : in STD_LOGIC_VECTOR (2 downto 0);
            Q : out STD_LOGIC_VECTOR (2 downto 0));
end component;
signal Ins0:std_logic_vector(11 downto 0);
signal PC_out,J_A,A_3,M0_S,M1_S,R_E:std_logic_vector(2 downto 0);
signal I_V,M3,M1,M0,A_4:std_logic_vector(3 downto 0);
signal R0,R1,R2,R3,R4,R5,R6,R7:std_logic_vector(3 downto 0);
signal J,L_S,Sub:std_logic;
```

```
signal M2:std_logic_vector(2 downto 0);

begin
Slow_Clk_0:Slow_Clk
port map(
Clk_in=>Clk,
Clk_out=>SlowClk);
Program_Counter_0:Program_Counter
    port map(
            Clk=>SlowClk,
            Reset=>Reset,
            D=>M2,
            Q=>PC_out);

ProgramROM_0:ProgramROM
    port map(
        MemSel=>PC_out,
        Instruction=>Ins0);

MUX_2_to_1_4_0:MUX_2_to_1_4
    port map(
        D0=>A_4,
        D1=>I_V,
        S=>L_S,
        Y=>M3);

MUX_2_to_1_3_0:MUX_2_to_1_3
            port map(
                D0=>A_3,
                D1=>J_A,
                S=>J,
                Y=>M2);

MUX_8_to_1_4_0:MUX_8_to_1_4
    port map(
        R0=>R0,
        R1=>R1,
        R2=>R2,
        R3=>R3,
        R4=>R4,
        R5=>R5,
        R6=>R6,
        R7=>R7,
        Y=>M0,
        S=>M0_S);

MUX_8_to_1_4_1:MUX_8_to_1_4
    port map(
        R0=>R0,
        R1=>R1,
        R2=>R2,
        R3=>R3,
        R4=>R4,
        R5=>R5,
        R6=>R6,
        R7=>R7,
        Y=>M1,
```

```
        S=>M1_S);

RegisterBank_0:RegisterBank
    port map(
        Clk=>SlowClk,
        Reset=>Reset,
        D=>M3,
        R0=>R0,
        R1=>R1,
        R2=>R2,
        R3=>R3,
        R4=>R4,
        R5=>R5,
        R6=>R6,
        R7=>R7,
        I=>R_E);

Add_Sub_4bit_0:Add_Sub_4bit
    port map(
        A=>M1,
        B=>M0,
        Sub_Sel=>Sub,
        S=>A_4,
        Overflow=>Overflow,
        Zero=>Zero

        );
RCA_3_0:RCA_3
    port map(
        A=>PC_out,
        S=>A_3);
Instruction_Decoder_0:Instruction_Decoder
Port map (
           Ins =>Ins0,
           Reg_Che_For_Jump =>M0,
           Reg_En =>R_E,
           Load_Im =>L_S,
           Im_Val =>I_V,
           MuxA_Sel =>M0_S,
           MuxB_Sel =>M1_S,
           Sub_Sel =>Sub,
           Jump =>J,
           Address_Jump =>J_A);

R7_out<=R7;
end Behavioral;
```

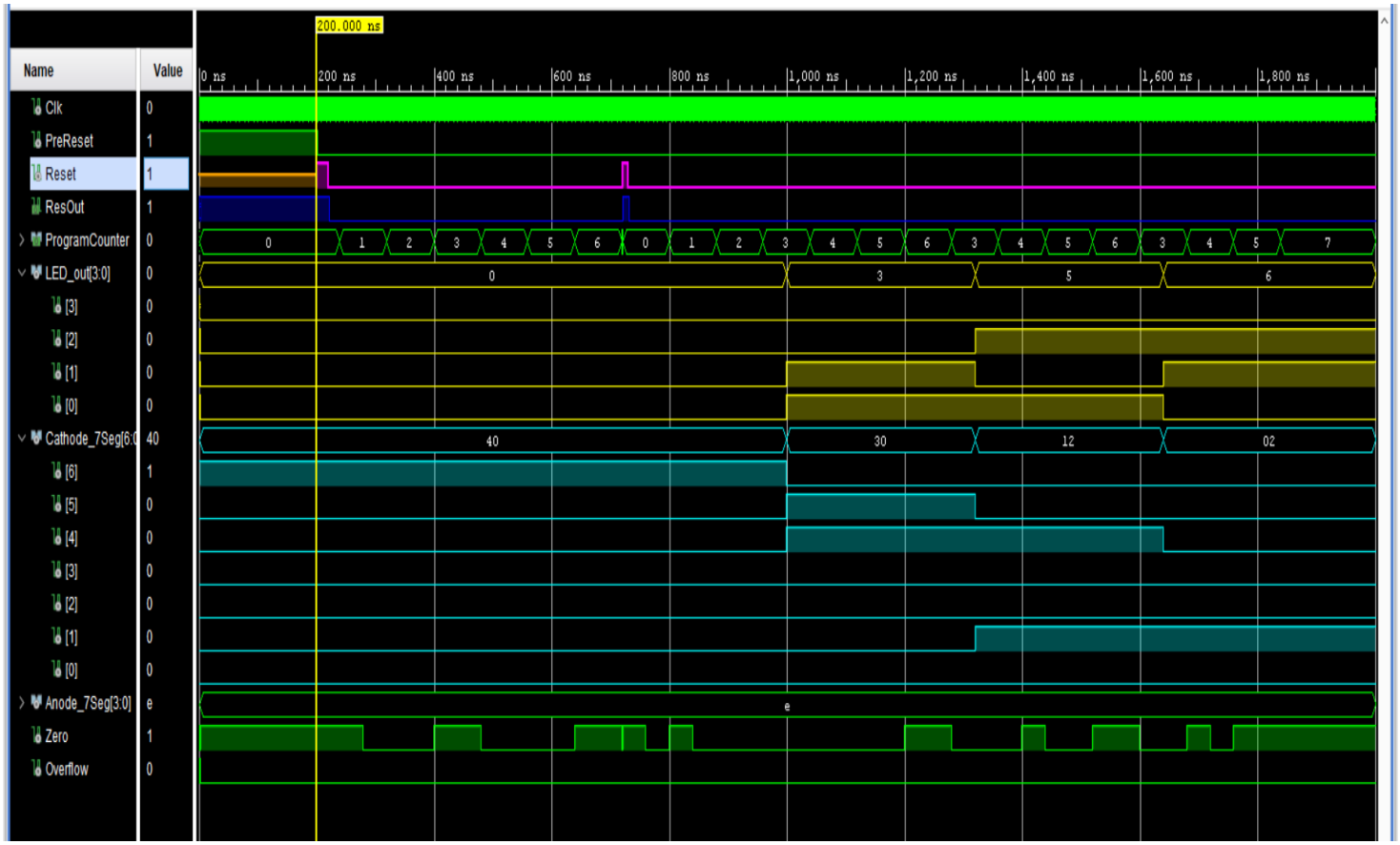
02. RTL Schematic diagram of the nano processor.



03. Behavioral simulation Code for nano processor.

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

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

architecture Behavioral of sim_Nano_Processor is
component Nano_Processor
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC;
           R7_out : out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal Overflow,Zero:std_logic;
signal Clk,Reset:std_logic:='1';
signal R7_out :STD_LOGIC_VECTOR (3 downto 0);
begin
UUT:Nano_processor
    port map(
        Clk=>Clk,
        Reset=>Reset,
        OverFlow=>Overflow,
        Zero=>Zero,
        R7_out=>R7_out);
process
begin
    wait for 1ns;
    Clk<=not(Clk);
end process;

process
begin
    wait for 200ns;
    Reset<='1';
    wait for 20ns;
    Reset<='0';
    wait for 500ns;
    Reset<='1';
    wait for 20ns;
    Reset<='0';
    wait for 1500ns;
    Reset<='1';
    wait for 20ns;
    Reset<='0';
    wait;
end process;
end Behavioral;
```

04. Timing Diagram for nano processor.



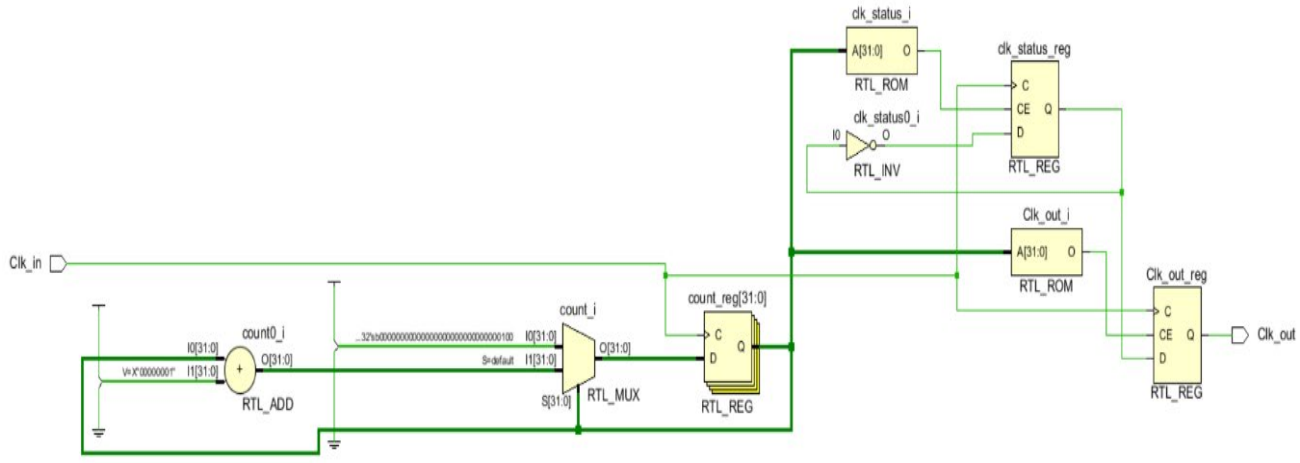
Slow Clock

---

01. Design source VHDL code of 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

 --for 100 MHz input clock this generates 1 Hz clock

process(Clk_in) begin
    if (rising_edge(Clk_in)) then
        count<=count+1; --Increment counter
        if (count=4) then --count 50M pluses(1/2 of period)
            clk_status<=NOT (clk_status); --invert clock status
            Clk_out<=clk_status;
            count<=1; --Reset counter
        end if;
    end if;
end process;

end Behavioral;
```

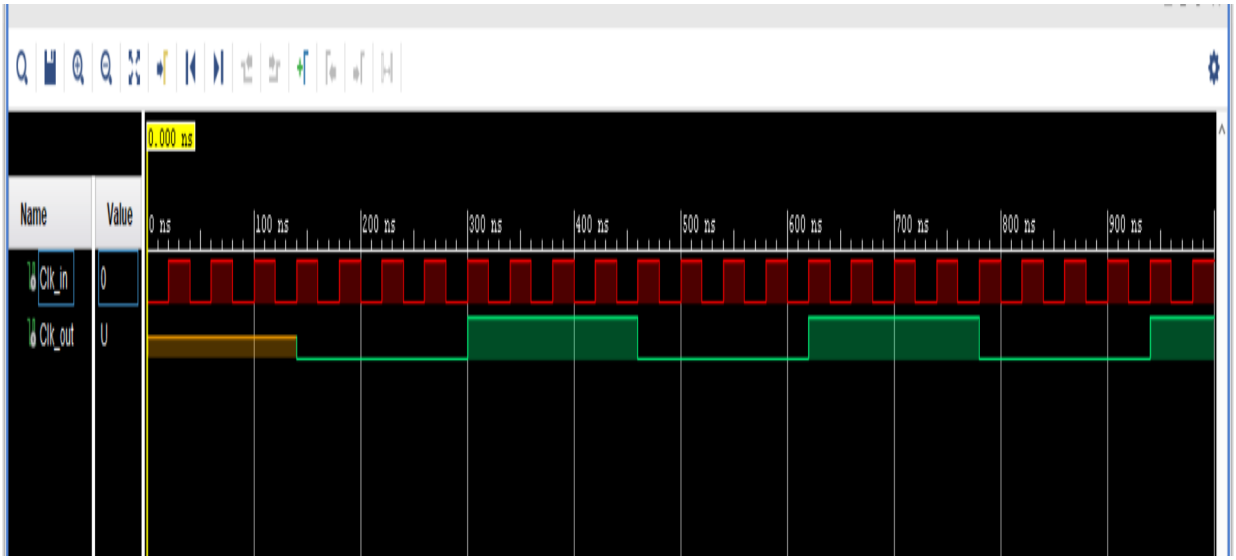
02. RTL Schematic diagram of Slow Clock.



03. Behavioral simulation source code for Slow Clock.

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

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

architecture Behavioral of TB_Slow_Clk is
component Slow_Clk
    port (Clk_in : in STD_LOGIC;
        Clk_out : out STD_LOGIC);
end component;
signal Clk_in : STD_LOGIC:='0';
signal Clk_out : STD_LOGIC;
begin

 UUT: Slow_Clk
 port map(
    Clk_in=>Clk_in,
    Clk_out=>Clk_out );

 process
 begin
    wait for 20ns;
    Clk_in <= NOT(Clk_in);

 end process;
end Behavioral;
```

04. Timing Diagram for Slow Clock.



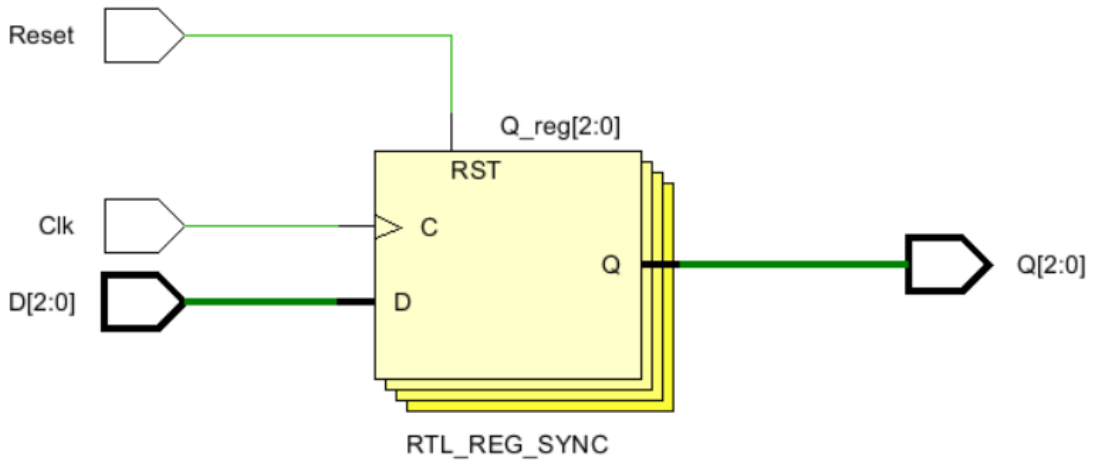
3-bit program counter

---

01. Design source VHDL code of 3-bit program counter.

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

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

architecture Behavioral of Program_Counter is

begin
process (Clk)
    begin
        if (rising_edge(Clk)) then
            if Reset = '1' then
                Q <= "000";
            else
                Q <= D;
                --Qbar <= not D;
            end if;
        end if;
    end process;

end Behavioral;
```

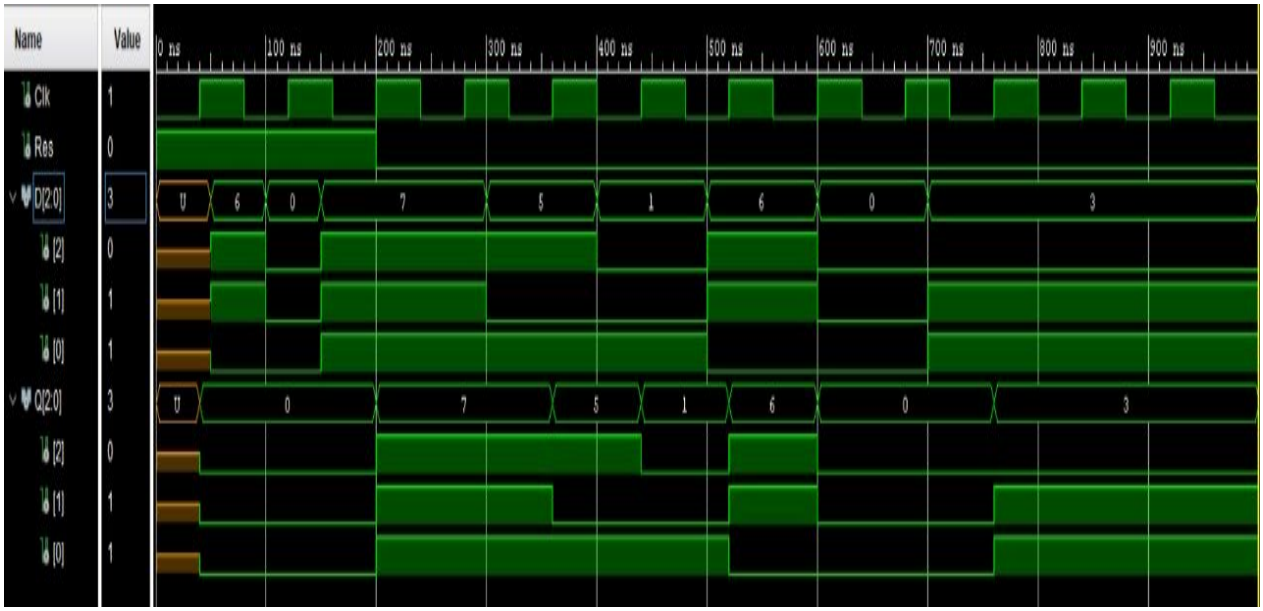
02. RTL Schematic diagram of the 3-bit program counter.



03. Behavioral simulation source code for 3-bit program counter.

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

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

architecture Behavioral of TB_Program_Counter is
component Program_Counter
Port (      Clk : in STD_LOGIC;
            Reset : in STD_LOGIC;
            D : in STD_LOGIC_VECTOR (2 downto 0);
            Q : out STD_LOGIC_VECTOR (2 downto 0));
end component;
signal Q,D :STD_LOGIC_VECTOR (2 downto 0);
signal Res,Clk:std_logic:='0';
begin
UUT:Program_Counter
    port map(
        Clk=>Clk,
        Reset=>Res,
        D=>D,
        Q=>Q);
process
begin
   wait for 40ns;
   Clk<=not(Clk);
end process;

process
begin
Res<='1';
wait for 50ns;
D<="110";
wait for 50ns;
D<="000";
wait for 50ns;
D<="111";
wait for 50ns;
Res<='0';
wait for 100ns;
D<="101";
wait for 100ns;
D<="001";
wait for 100ns;
D<="110";
wait for 100ns;
D<="000";
wait for 100ns;
D<="011";
wait;

end process;
end Behavioral;
```

04. Timing Diagram for 3-bit program counter.



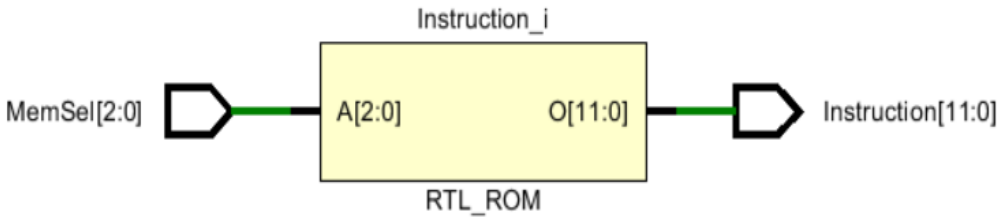
Program ROM

---

01. Design source VHDL code of program ROM.

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

entity ProgramROM is
Port ( MemSel : in STD_LOGIC_VECTOR (2 downto 0);
Instruction : out STD_LOGIC_VECTOR (11 downto 0));
end ProgramROM;

architecture Behavioral of ProgramROM is
type rom_type is array (0 to 7) of std_logic_vector(11 downto 0);
signal ProgramROM_0 : rom_type :=(
   "100010000011",  -- MOVI R1,3    --line->0
   "100100000001",  -- MOVI R2,1
   "010100000000",  -- NEG R2
   "001110010000",  -- ADD R7,R1    --line->3
   "000010100000",  -- ADD R1,R2
   "110010000111",  -- JZR R1,7
   "110000000011",  -- JZR R0,3
   "110000000111"   -- JZR R0,7     --line->7

);

begin
Instruction <= ProgramROM_0(to_integer(unsigned(MemSel)));
end Behavioral;
```

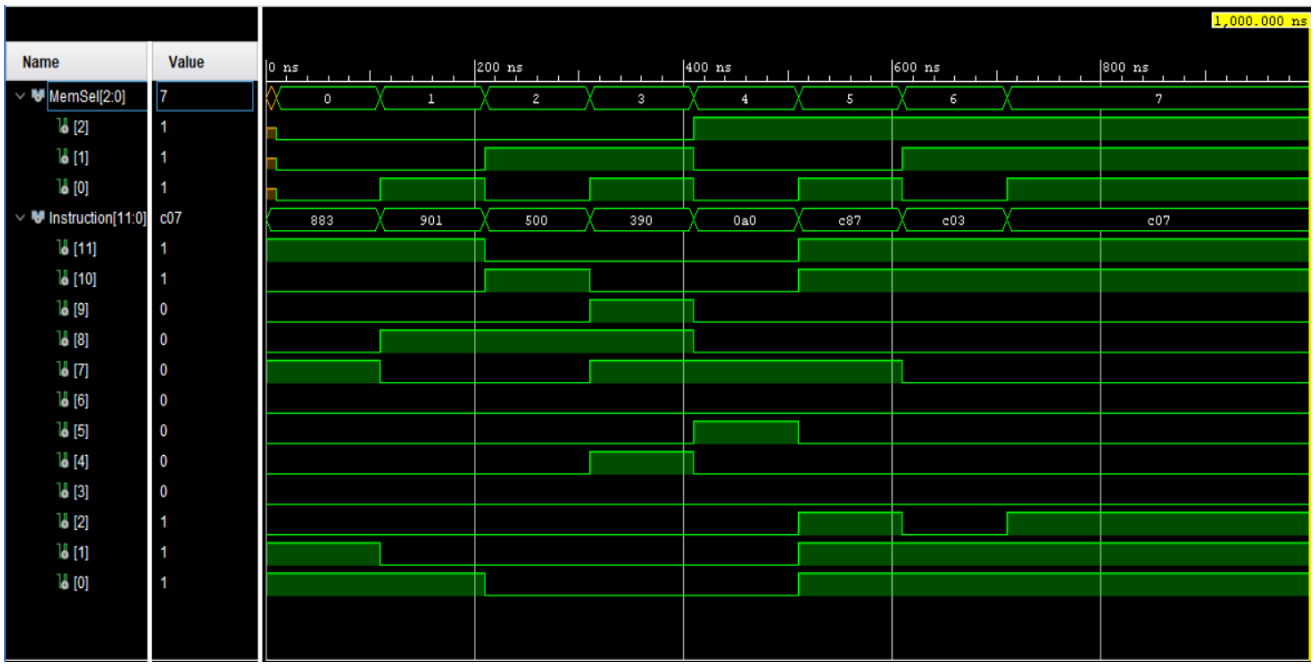
02. RTL Schematic diagram of program ROM.



03. Behavioral simulation source code for program ROM.

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

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

architecture Behavioral of RomSim is
component ProgramROM

  port(
MemSel : in STD_LOGIC_VECTOR (2 downto 0);
Instruction : out STD_LOGIC_VECTOR (11 downto 0)
);
end component;

Signal MemSel : STD_LOGIC_VECTOR(2 downto 0);
Signal Instruction : STD_LOGIC_VECTOR(11 downto 0);


begin
UUT : ProgramROM port map(
MemSel => MemSel,
Instruction => Instruction
);
process
begin

wait for 10ns;
MemSel <= "000";
wait for 100ns;
MemSel <= "001";
wait for 100ns;
MemSel <= "010";
wait for 100ns;
MemSel <= "011";
wait for 100ns;
MemSel <= "100";
wait for 100ns;
MemSel<="101";
Wait for 100ns;
MemSel<="110";
Wait for 100ns;
MemSel<="111";
wait;
end process;
end Behavioral;
```

04. Timing Diagram for program ROM.



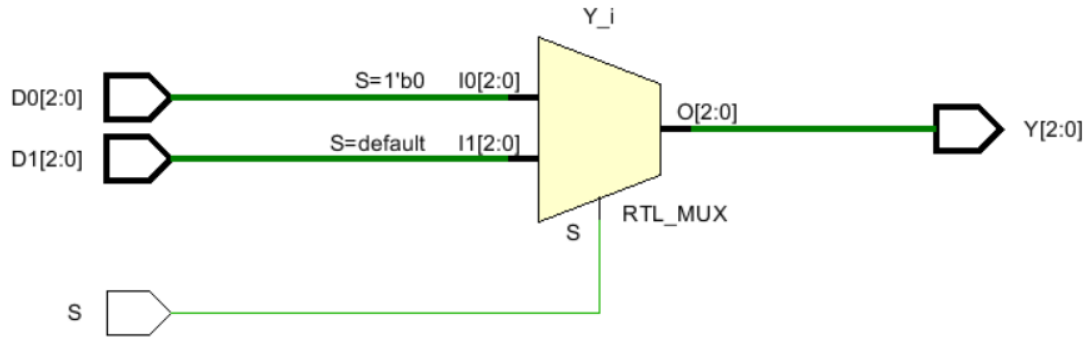
2-way 3-bit multiplexer

---

01. Design source VHDL code of 2-way 3-bit multiplexer.

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

entity MUX_2_to_1_3 is
    Port ( D0 : in STD_LOGIC_VECTOR (2 downto 0);
           D1 : in STD_LOGIC_VECTOR (2 downto 0);
           S : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (2 downto 0));
end MUX_2_to_1_3;

architecture Behavioral of MUX_2_to_1_3 is

begin

    Y <= D0 when (S = '0') else D1;

end Behavioral;
```

02. RTL Schematic diagram of 2-way 3-bit multiplexer.



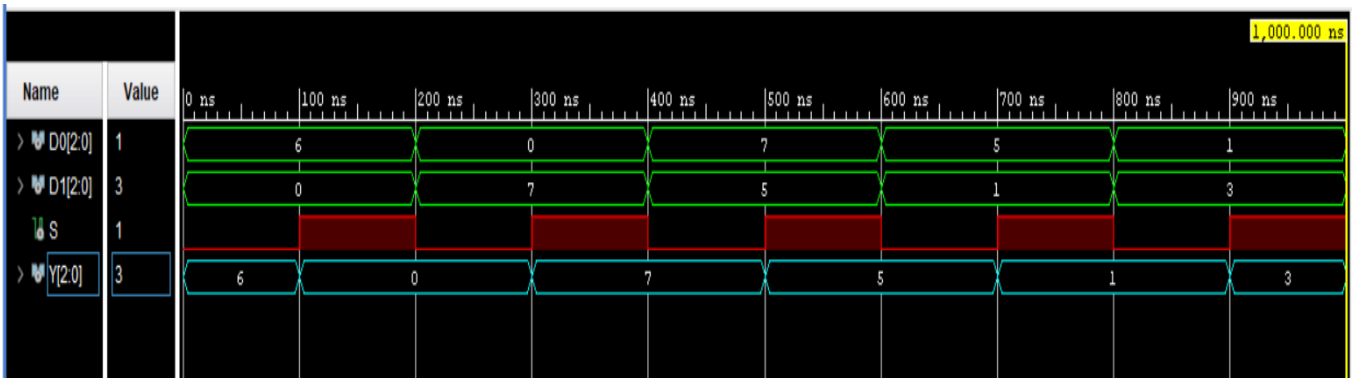
03. Behavioral simulation source code for the 2-way 3-bit multiplexer.

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

entity TB_MUX_2_to_1_3 is
--  Port ( );
end TB_MUX_2_to_1_3;

architecture Behavioral of TB_MUX_2_to_1_3 is


component MUX_2_to_1_3
    Port ( D0 : in STD_LOGIC_VECTOR(2 downto 0);
```

```
            D1 : in STD_LOGIC_VECTOR(2 downto 0);
            S : in STD_LOGIC;
            Y : out STD_LOGIC_VECTOR(2 downto 0));
end component;

signal D0,D1,Y : std_logic_vector(2 downto 0);
signal S : std_logic;

begin

UUT: MUX_2_to_1_3
    port map(
        D0 => D0,
        D1 => D1,
        S => S,
        Y => Y
    );


process
begin
    --200240 - 110 000 111 000 110 000
    --200575 - 110 000 111 101 111 111
    --200635 - 110 000 111 110 111 011
    --200647 - 110 000 111 111 000 111
    --200655 - 110 000 111 111 001 111

    D0 <= "110";
    D1 <= "000";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "000";
    D1 <= "111";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "111";
    D1 <= "101";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "101";
    D1 <= "001";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "001";
    D1 <= "011";
```

```
        S <= '0';
        wait for 100 ns;
        S <= '1';
        wait;

    end process;

    end Behavioral;
```

04. Timing diagram for 2-way 3-bit multiplexer.



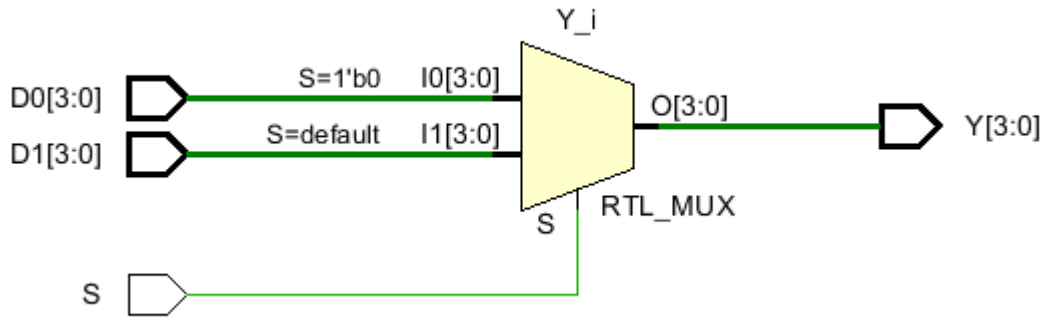
2-way 4-bit multiplexer

---

01. Design source VHDL code of 2-way 4-bit multiplexer.

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

entity MUX_2_to_1_4 is
    Port ( D0 : in STD_LOGIC_VECTOR(3 downto 0);
           D1 : in STD_LOGIC_VECTOR(3 downto 0);
           S : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR(3 downto 0));
end MUX_2_to_1_4;

architecture Behavioral of MUX_2_to_1_4 is

begin
    Y <= D0 when (S = '0') else D1;
end Behavioral;
```

02. RTL Schematic diagram of 2-way 4-bit multiplexer.

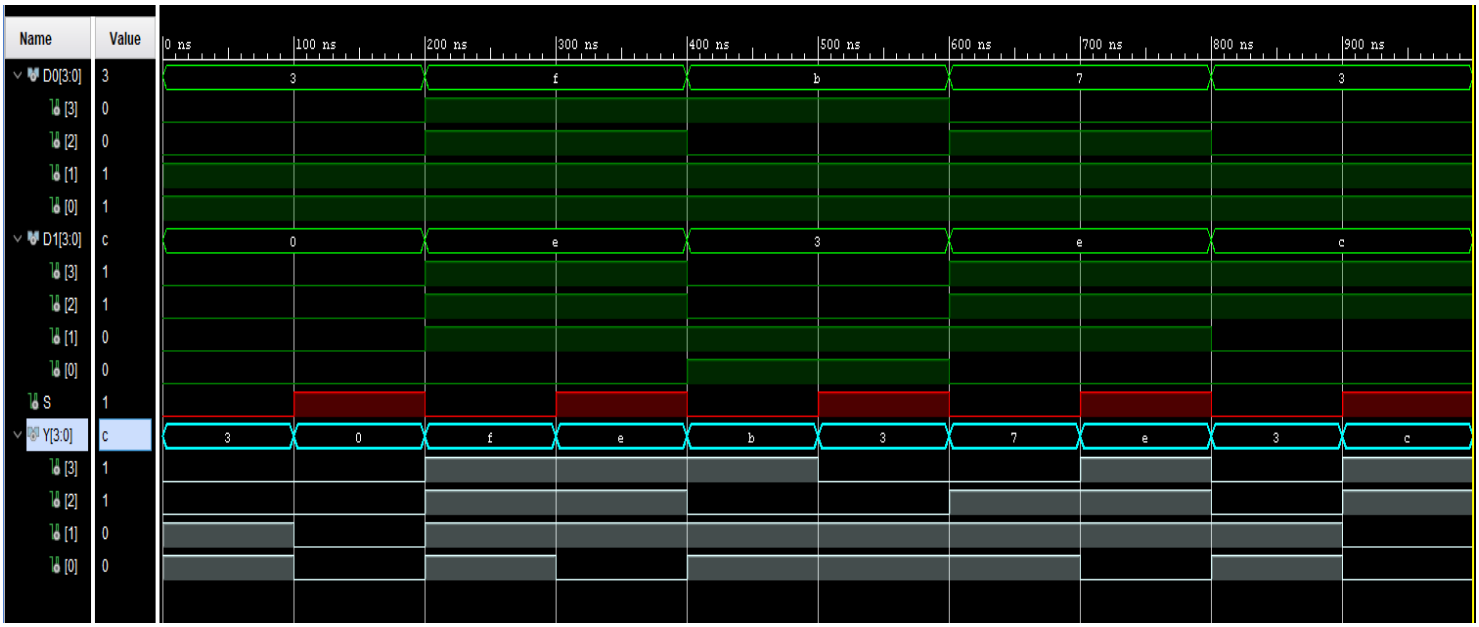


03. Behavioral simulation source code for 2-way 4-bit multiplexer.

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

entity TB_MUX_2_to_1_4 is
--  Port ( );
end TB_MUX_2_to_1_4;

architecture Behavioral of TB_MUX_2_to_1_4 is

component MUX_2_to_1_4
    Port ( D0 : in STD_LOGIC_VECTOR(3 downto 0);
           D1 : in STD_LOGIC_VECTOR(3 downto 0);
           S : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR(3 downto 0));
end component;
```

```
signal D0,D1,Y : std_logic_vector(3 downto 0);
signal S : std_logic;

begin

UUT: MUX_2_to_1_4
    port map(
        D0 => D0,
        D1 => D1,
        S => S,
        Y => Y
    );

process
begin
    --200635->11 0000 1111 1011 1011
    --200240->11 0000 1110 0011 0000
    --200575->11 0000 1111 0111 1111
    --200647->11 0000 1111 1100 0111
    --200655->11 0000 1111 1100 1111
    D0 <= "0011";
    D1 <= "0000";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "1111";
    D1 <= "1110";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "1011";
    D1 <= "0011";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "0111";
    D1 <= "1110";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "0011";
    D1 <= "1100";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait;
end process;

end Behavioral;
```

04. Timing diagram for 2-way 4-bit multiplexer.



8-way 4-bit multiplexer

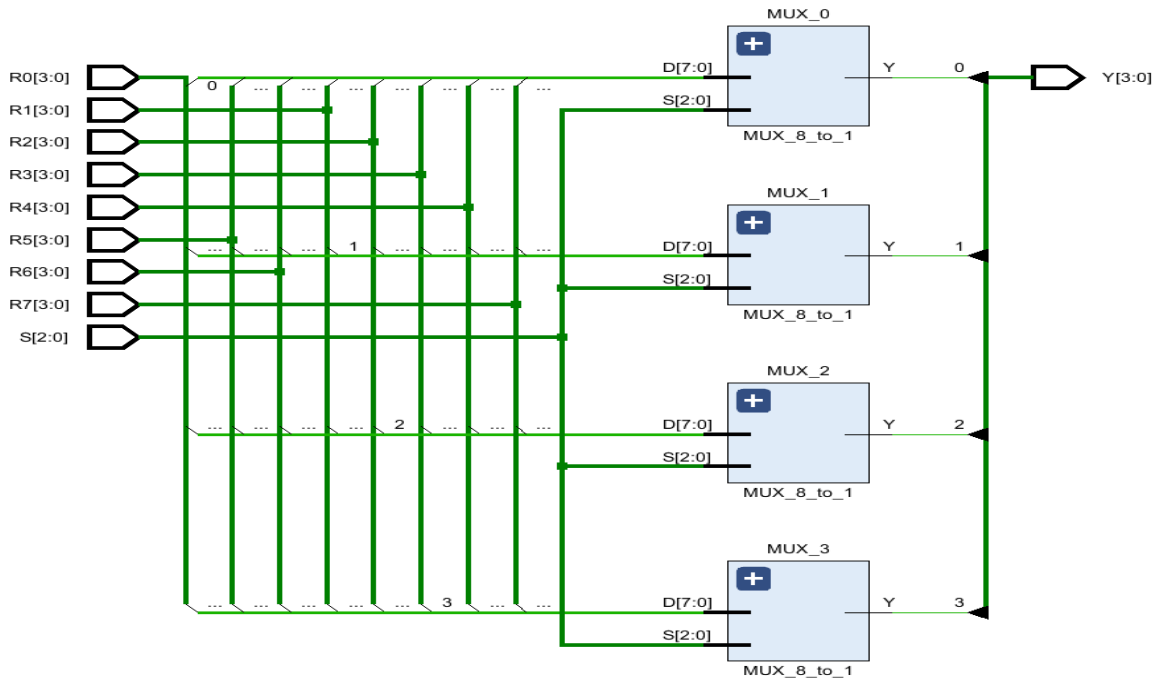
---

01. Design source VHDL code of 8-way 4-bit multiplexer.

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

entity MUX_8_to_1_4 is
    Port ( R0 : in STD_LOGIC_VECTOR (3 downto 0);
           R1 : in STD_LOGIC_VECTOR (3 downto 0);
           R2 : in STD_LOGIC_VECTOR (3 downto 0);
           R3 : in STD_LOGIC_VECTOR (3 downto 0);
           R4 : in STD_LOGIC_VECTOR (3 downto 0);
           R5 : in STD_LOGIC_VECTOR (3 downto 0);
           R6 : in STD_LOGIC_VECTOR (3 downto 0);
           R7 : in STD_LOGIC_VECTOR (3 downto 0);
           S : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end MUX_8_to_1_4;

architecture Behavioral of MUX_8_to_1_4 is

component MUX_8_to_1
    Port ( D : in STD_LOGIC_VECTOR (7 downto 0);
           S : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC);
end component;

begin

MUX_0 : MUX_8_to_1
    Port map(
        D(0) => R0(0),
        D(1) => R1(0),
        D(2) => R2(0),
        D(3) => R3(0),
        D(4) => R4(0),
        D(5) => R5(0),
        D(6) => R6(0),
        D(7) => R7(0),
        S => S,
        Y => Y(0)
    );

MUX_1 : MUX_8_to_1
    Port map(
        D(0) => R0(1),
        D(1) => R1(1),
        D(2) => R2(1),
        D(3) => R3(1),
        D(4) => R4(1),
        D(5) => R5(1),
        D(6) => R6(1),
        D(7) => R7(1),
```

```
        S => S,
        Y => Y(1)
    );
MUX_2 : MUX_8_to_1
    Port map(
        D(0) => R0(2),
        D(1) => R1(2),
        D(2) => R2(2),
        D(3) => R3(2),
        D(4) => R4(2),
        D(5) => R5(2),
        D(6) => R6(2),
        D(7) => R7(2),
        S => S,
        Y => Y(2)
    );
MUX_3 : MUX_8_to_1
    Port map(
        D(0) => R0(3),
        D(1) => R1(3),
        D(2) => R2(3),
        D(3) => R3(3),
        D(4) => R4(3),
        D(5) => R5(3),
        D(6) => R6(3),
        D(7) => R7(3),
        S => S,
        Y => Y(3)
    );

end Behavioral;
```

02. RTL Schematic diagram of 8-way 4-bit multiplexer.

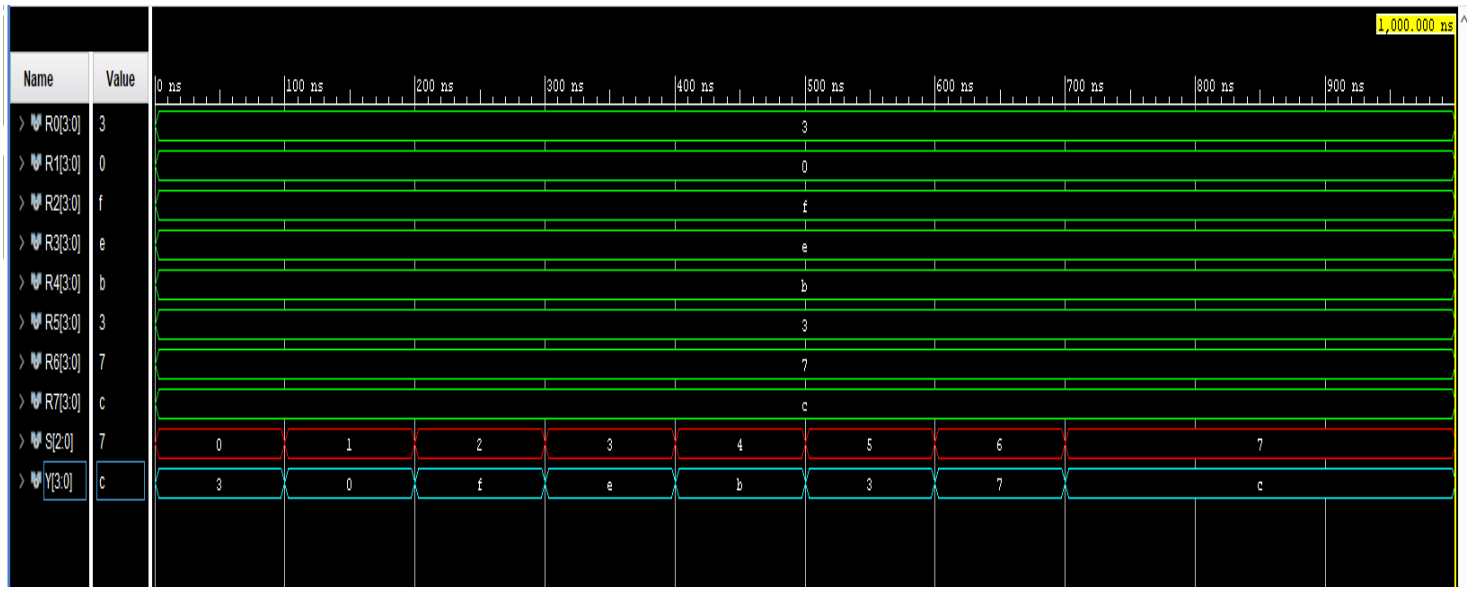


03. Behavioral simulation code for 8-way 4-bit multiplexer.

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

entity TB_MUX_8_to_1_4 is
--  Port ( );
end TB_MUX_8_to_1_4;

architecture Behavioral of TB_MUX_8_to_1_4 is
component MUX_8_to_1_4
    Port (      R0 : in STD_LOGIC_VECTOR (3 downto 0);
                R1 : in STD_LOGIC_VECTOR (3 downto 0);
                R2 : in STD_LOGIC_VECTOR (3 downto 0);
                R3 : in STD_LOGIC_VECTOR (3 downto 0);
                R4 : in STD_LOGIC_VECTOR (3 downto 0);
                R5 : in STD_LOGIC_VECTOR (3 downto 0);
                R6 : in STD_LOGIC_VECTOR (3 downto 0);
                R7 : in STD_LOGIC_VECTOR (3 downto 0);
                S : in STD_LOGIC_VECTOR (2 downto 0);
                Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal R0,R1,R2,R3,R4,R5,R6,R7,Y : std_logic_vector(3 downto 0);
signal S : std_logic_vector(2 downto 0);

begin

UUT: MUX_8_to_1_4
    port map (
        R0 => R0,
        R1 => R1,
        R2 => R2,
        R3 => R3,
        R4 => R4,
        R5 => R5,
        R6 => R6,
        R7 => R7,
        S => S,
        Y => Y
    );

process
begin
    --200635->11 0000 1111 1011 1011
    --200240->11 0000 1110 0011 0000
    --200575->11 0000 1111 0111 1111
    --200647->11 0000 1111 1100 0111
    --200655->11 0000 1111 1100 1111

    R0 <= "0011";
    R1 <= "0000";
    R2 <= "1111";
    R3 <= "1110";
    R4 <= "1011";
    R5 <= "0011";
    R6 <= "0111";
```

```
        R7 <= "1100";

        S <= "000";
        wait for 100 ns;
        S <= "001";
        wait for 100 ns;
        S <= "010";
        wait for 100 ns;
        S <= "011";
        wait for 100 ns;
        S <= "100";
        wait for 100 ns;
        S <= "101";
        wait for 100 ns;
        S <= "110";
        wait for 100 ns;
        S <= "111";
        wait;
    end process;

    end Behavioral;
```

04. Timing diagram for the 8-way 4-bit multiplexer.



Register Bank

---

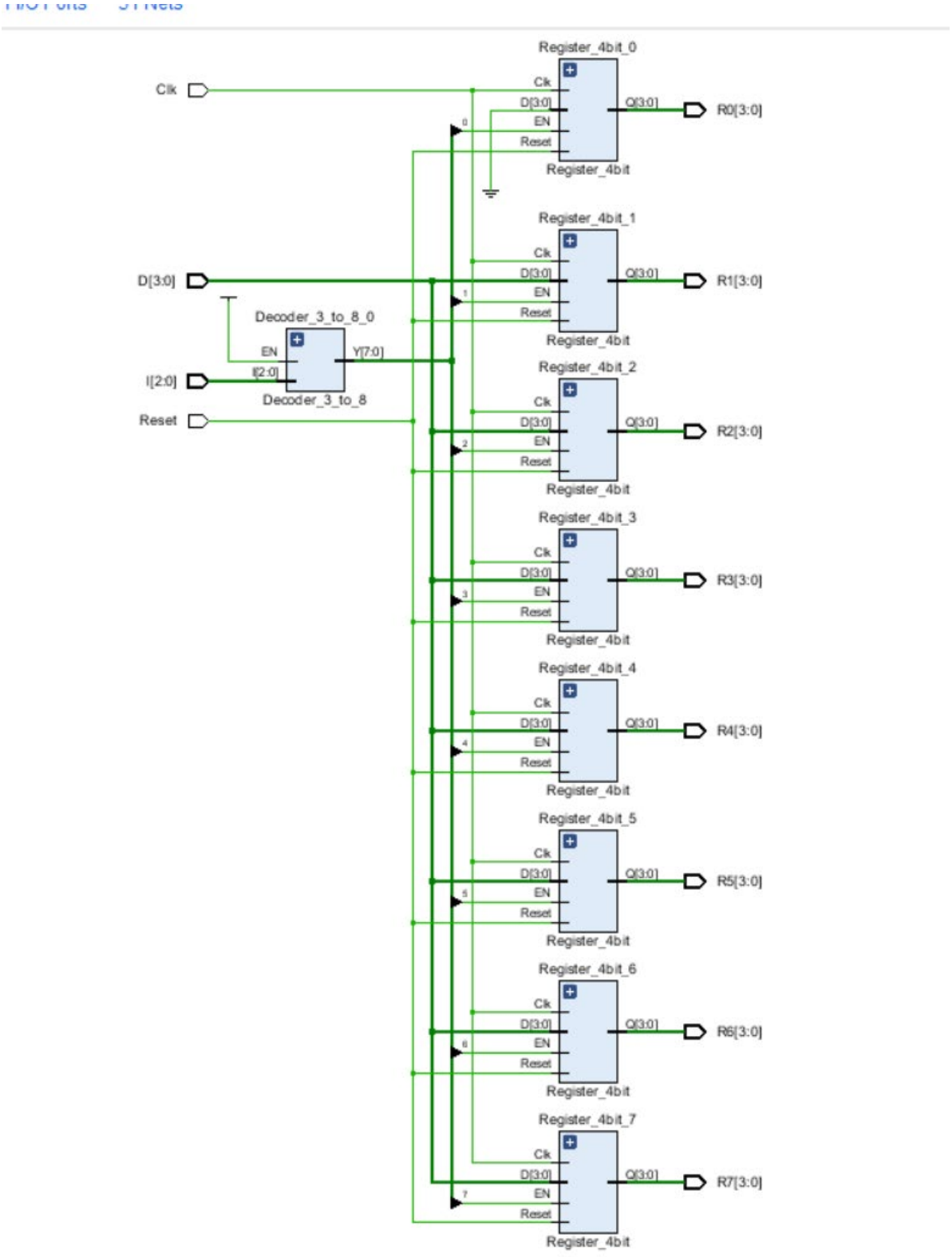
01. Design source VHDL code of Register Bank.

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

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

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

component Register_4bit
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal EN : STD_LOGIC:='1';
signal Y : STD_LOGIC_VECTOR (7 downto 0);
---signal Reset : STD_LOGIC;
begin
Decoder_3_to_8_0:Decoder_3_to_8
    port map(
        I => I,
        EN => EN,
        Y => Y
    );
Register_4bit_0:Register_4bit
    port map(
        D => "0000",
        EN => Y(0),
        Clk => Clk,
        Reset => Reset,
        Q => R0
```

```
    );

Register_4bit_1:Register_4bit
        port map(
            D => D,
            EN => Y(1),
            Clk => Clk,
            Reset => Reset,
            Q => R1
        );
Register_4bit_2:Register_4bit
        port map(
            D => D,
            EN => Y(2),
            Clk => Clk,
            Reset => Reset,
            Q => R2
        );
Register_4bit_3:Register_4bit
        port map(
            D => D,
            EN => Y(3),
            Clk => Clk,
            Reset => Reset,
            Q => R3
        );

Register_4bit_4:Register_4bit
        port map(
            D => D,
            EN => Y(4),
            Clk => Clk,
            Reset => Reset,
            Q => R4
        );
Register_4bit_5:Register_4bit
        port map(
            D => D,
            EN => Y(5),
            Clk => Clk,
            Reset => Reset,
            Q => R5
        );
Register_4bit_6:Register_4bit
        port map(
            D => D,
            EN => Y(6),
            Clk => Clk,
            Reset => Reset,
            Q => R6
        );
Register_4bit_7:Register_4bit
        port map(
            D => D,
            EN => Y(7),
            Clk => Clk,
            Reset => Reset,
```

```
            Q => R7
        );
end Behavioral;
```

02. RTL Schematic diagram of Register Bank.

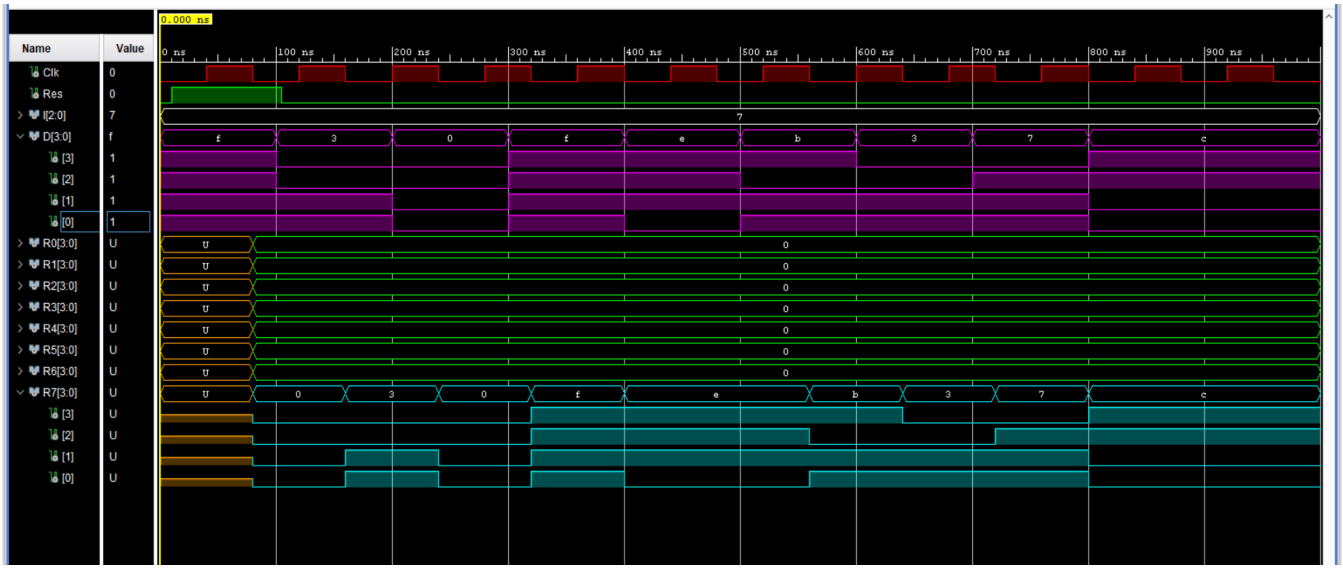


03. Behavioral simulation source code for Register Bank.

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

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

architecture Behavioral of TB_RegisterBank is
component RegisterBank
    Port ( Clk : in STD_LOGIC;
       Reset : in STD_LOGIC;
       D : in STD_LOGIC_VECTOR (3 downto 0);
       R0 : out STD_LOGIC_VECTOR (3 downto 0);
       R1 : out STD_LOGIC_VECTOR (3 downto 0);
       R2 : out STD_LOGIC_VECTOR (3 downto 0);
       R3 : out STD_LOGIC_VECTOR (3 downto 0);
       R4 : out STD_LOGIC_VECTOR (3 downto 0);
       R5 : out STD_LOGIC_VECTOR (3 downto 0);
       R6 : out STD_LOGIC_VECTOR (3 downto 0);
       R7 : out STD_LOGIC_VECTOR (3 downto 0);
       I : in STD_LOGIC_VECTOR (2 downto 0));
end component;
Signal I:STD_LOGIC_VECTOR (2 downto 0):="111";
Signal Clk, Res: std_logic:='0';
Signal D: STD_LOGIC_VECTOR (3 downto 0):="1111";
Signal R0, R1, R2, R3, R4, R5, R6, R7 : STD_LOGIC_VECTOR (3 downto 0);

begin
UUT: RegisterBank
    port map(
    Clk => Clk,
    Reset => Res,
    D => D,
    R0 => R0,
    R1 => R1,
    R2 => R2,
    R3 => R3,
    R4 => R4,
    R5 => R5,
    R6 => R6,
    R7 => R7,
    I => I
    );

process begin
    wait for 40ns;
    Clk <= Not(Clk);
end process;

process begin
    wait for 10ns;
    Res <= Not(Res);
    wait for 95ns;
    Res <= Not(Res);
    wait for 5ns;
```

```
    I <= "111";
    wait;
end process;
--Group Members Index numbers' binary form

--200635->11 0000 1111 1011 1011
--200240->11 0000 1110 0011 0000
--200575->11 0000 1111 0111 1111
--200647->11 0000 1111 1100 0111
--200655->11 0000 1111 1100 1111

--unique 4bit numbers from index
numbers=>0011,0000,1111,1110,1011,0011,0111,1100
process begin
    wait for 100ns;
    D <= "0011";
    wait for 100ns;
    D <= "0000";
    wait for 100ns;
    D <= "1111";
    wait for 100ns;
    D <= "1110";
    wait for 100ns;
    D <= "1011";
    wait for 100ns;
    D <= "0011";
    wait for 100ns;
    D <= "0111";
    wait for 100ns;
    D <= "1100";
    wait;
end process;


end Behavioral;
```

04. Timing Diagram for Register Bank.



4-bit Add/Subtract unit

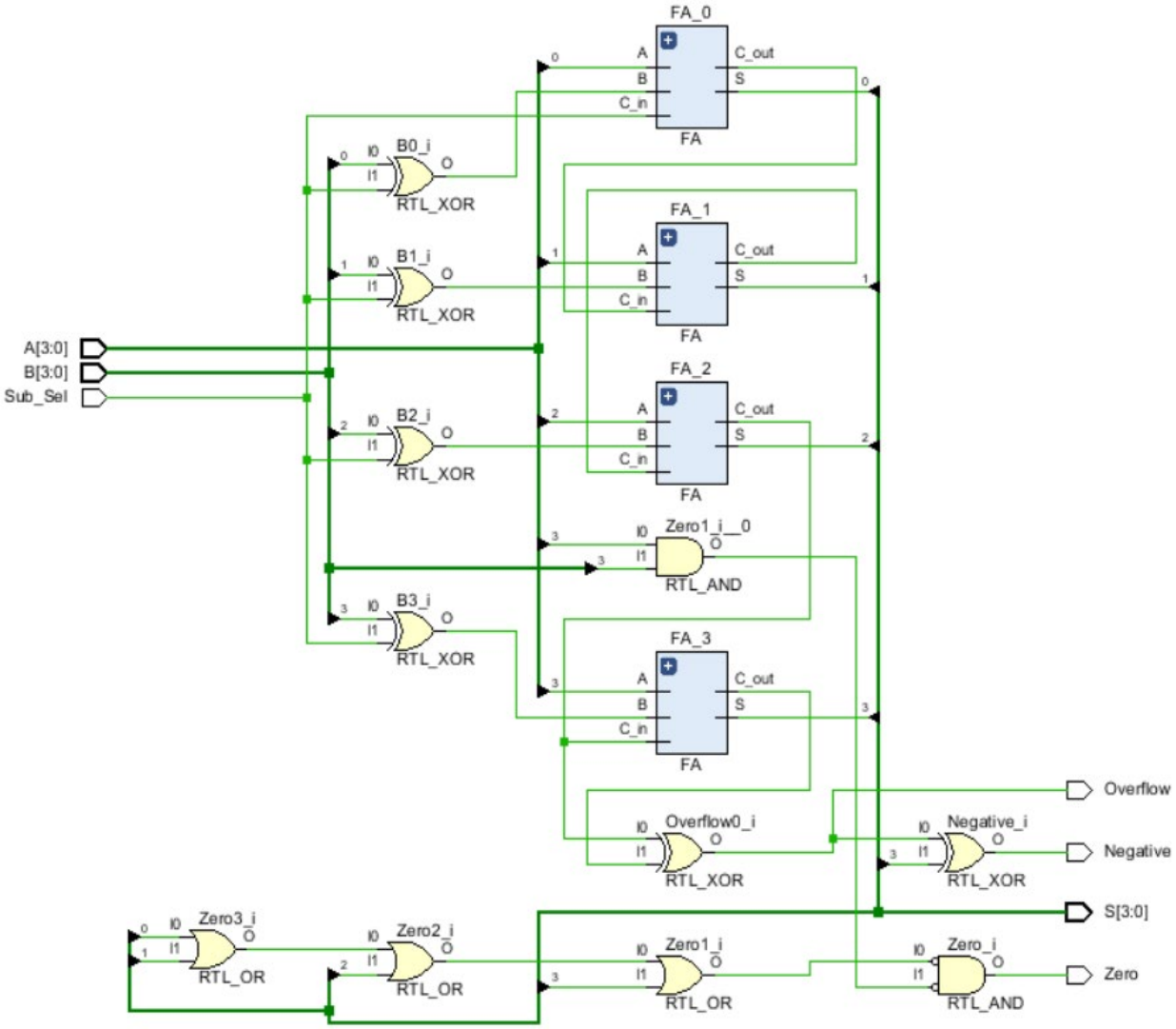
---

01. Design source VHDL code of 4-bit Add/Subtract unit.

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

entity Add_Sub_4bit is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Sub_Sel : in STD_LOGIC;
           Zero:out std_logic;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow : out STD_LOGIC;
           Negative:out std_logic);
end Add_Sub_4bit;

architecture Behavioral of Add_Sub_4bit is
component FA port(
        A:in std_logic;
        B:in std_logic;
        C_in:in std_logic;
        S:out std_logic;
        C_out:out std_logic );
       end component;
       signal  FA0_S, FA0_C, FA1_S, FA1_C, FA2_S, FA2_C, FA3_S,
FA3_C:std_logic;
       signal B0,B1,B2,B3:std_logic;
       signal S1 : std_LOGIC_VECTOR (3 downto 0);
       signal Overflow0,Zero0:std_logic;
begin
    B0<=B(0) xor Sub_Sel;
    B1<=B(1) xor Sub_Sel;
    B2<=B(2) xor Sub_Sel;
    B3<=B(3) xor Sub_Sel;
    FA_0 : FA
        port map (
         A => A(0),
         B => B0,
         C_in => Sub_Sel,
         S=>S1(0),
         C_Out => FA0_C);
    FA_1 : FA
          port map (
          A => A(1),
          B => B1,
          C_in => FA0_C,
          S => S1(1),
          C_Out => FA1_C);
    FA_2 : FA
           port map (
           A => A(2),
           B => B2,
           C_in => FA1_C,
```

```
            S => S1(2),
            C_Out => FA2_C);
    FA_3 : FA
             port map (
             A => A(3),
             B => B3,
             C_in => FA2_C,
             S => S1(3),
             C_Out => FA3_C);

    Overflow0<=FA2_C xor FA3_C;
    Negative<=Overflow0 xor S1(3);
    Overflow<=Overflow0;
    Zero<=not(S1(0) or S1(1) or S1(2) or S1(3) or Overflow0) ;
    S<=S1;


end Behavioral;
```

02. RTL Schematic diagram of 4-bit Add/Subtract unit.



03. Behavioral simulation source code for 4-bit Add/Subtract unit.

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

entity sim_Add_Sub_4bit is
--  Port ( );
end sim_Add_Sub_4bit;

architecture Behavioral of sim_Add_Sub_4bit is
component Add_Sub_4bit
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Sub_Sel : in STD_LOGIC;
           Zero:out std_logic;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Overflow : out STD_LOGIC;
           Negative: out std_logic);
end component;
signal A,B,S_out: std_logic_vector (3 downto 0);
signal Overflow,Sub,Zero,Negative:std_logic;
begin
UUT: Add_Sub_4bit
    PORT MAP(
        A=>A(3 downto 0),
        B=>B(3 downto 0),
        Sub_Sel=>Sub,
        zero=>Zero,
        S=>S_out(3 downto 0),
        Overflow=>Overflow,
        Negative=>Negative );
process
            begin
            --200635->11 0000 1111 1011 1011
            --200240->11 0000 1110 0011 0000
            --200575->11 0000 1111 0111 1111
            --200647->11 0000 1111 1100 0111
            --200655->11 0000 1111 1100 1111

            Sub<='0';     --add
            A<="0111"; --7
            B<="1100"; --(-4)
            wait for 100ns;

            A<="1000"; --(-8)
            B<="0010"; --2
            wait for 100ns;

            A<="0111"; --7
            B<="1001"; --(-7)
            wait for 100ns;

            A<="0111"; --7
            B<="0111"; --7
```

```
        wait for 100ns;

        A<="1000"; --(-8)
        B<="1000"; --(-8)
        wait for 100ns;

        Sub<='1';    --Sub
        A<="1111"; --(-1)
        B<="0001"; --1
        wait for 100ns;

        A<="0000"; --0
        B<="1100"; --(-4)
        wait for 100ns;

        A<="0010"; --(2)
        B<="0010"; --(2)
        wait for 100ns;

        A<="0011"; --3
        B<="1001"; --(-7)
        wait for 100ns;

        A<="1111"; --7
        B<="0001"; --1
        wait for 100ns;
    end process;

end Behavioral;
```

04. Timing Diagram for 4-bit Add/Subtract unit.

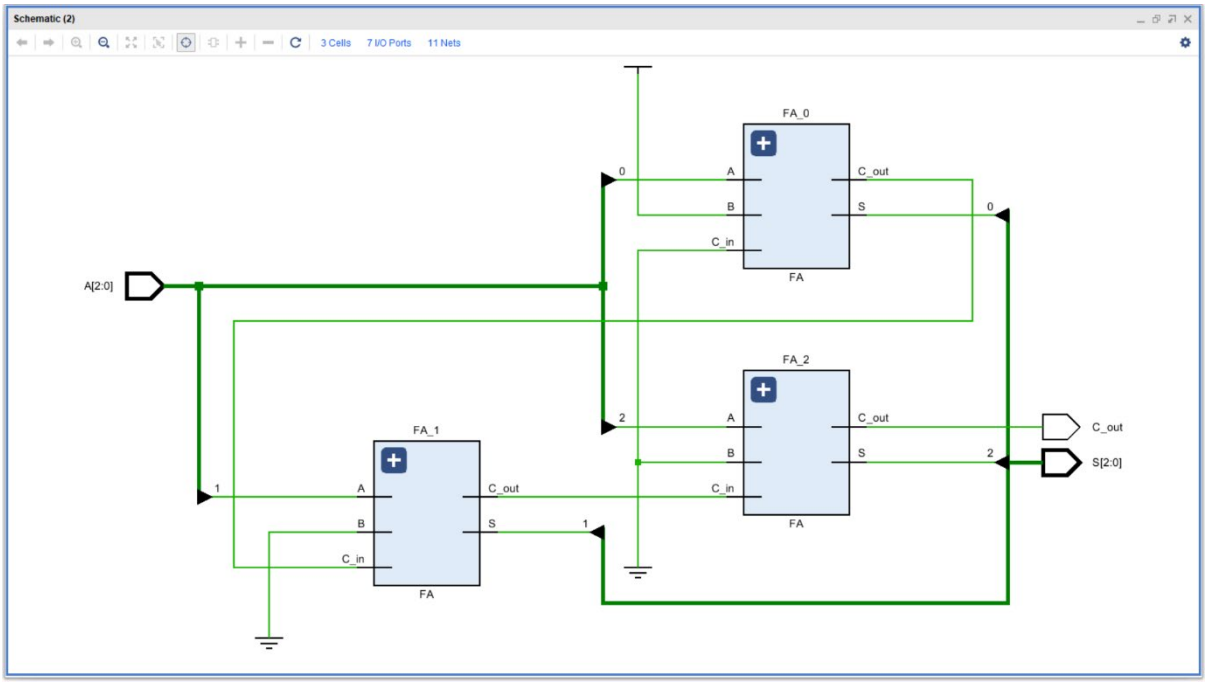
3-bit Ripple Carry Adder

---

01. Design source VHDL code of RCA.

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

entity RCA_3 is
    Port (  A : in STD_LOGIC_VECTOR (2 downto 0);
            --B: in STD_LOGIC_VECTOR (2 downto 0);
            S: out STD_LOGIC_VECTOR (2 downto 0);
            --C_in : in STD_LOGIC;
            C_out : out STD_LOGIC);
end RCA_3;

architecture Behavioral of RCA_3 is

 component FA
 port (
     A: in std_logic;
     B: in std_logic;
     C_in: in std_logic;
     S: out std_logic;
     C_out: out std_logic);
 end component;
 signal FA0_S, FA0_C, FA1_S, FA1_C, FA2_S, FA2_C: std_logic;
 --signal B : std_LOGIC_VECTOR :="001";
begin
 FA_0 : FA
 port map (
     A => A(0),
     B => '1',
     C_in => '0', -- Set to ground
     S => S(0),
     C_out => FA0_C);
 FA_1 : FA
 port map (
     A => A(1),
     B => '0',
     C_in => FA0_C,
     S => S(1),
     C_out => FA1_C);
 FA_2 : FA
 port map (
     A => A(2),
     B => '0',
     C_in => FA1_C,
     S => S(2),
     C_out => FA2_C);

 C_out <= FA2_C;

end Behavioral;
```

02. RTL Schematic diagram of RCA.



03. Behavioral simulation source code for RCA.

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

entity TB_RCA_3 is
--  Port ( );
end TB_RCA_3;

architecture Behavioral of TB_RCA_3 is
component RCA_3
Port (  A : in STD_LOGIC_VECTOR (2 downto 0);
            --B: in STD_LOGIC_VECTOR (2 downto 0);
            S: out STD_LOGIC_VECTOR (2 downto 0);
            --C_in : in STD_LOGIC;
            C_out : out STD_LOGIC);
end component;
signal C_out:std_logic;
signal A,S :STD_LOGIC_VECTOR (2 downto 0);

begin
UUT:RCA_3
    port map(
    A=>A,
    S=>S,
    C_out=>C_out
    );

--200635->110 000 111 110 111 011
--200240->110 000 111 000 110 000
--200575->110 000 111 101 111 111
```

```
--200647->110 000 111 111 000 111
--200655->110 000 111 111 001 111

--unique 3bit numbers from index numbers=>110,000,111,101,001,011

process
begin
A<="000";
wait for 100ns;

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

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

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

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

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

end process;

end Behavioral;
```

04. Timing Diagram for RCA.

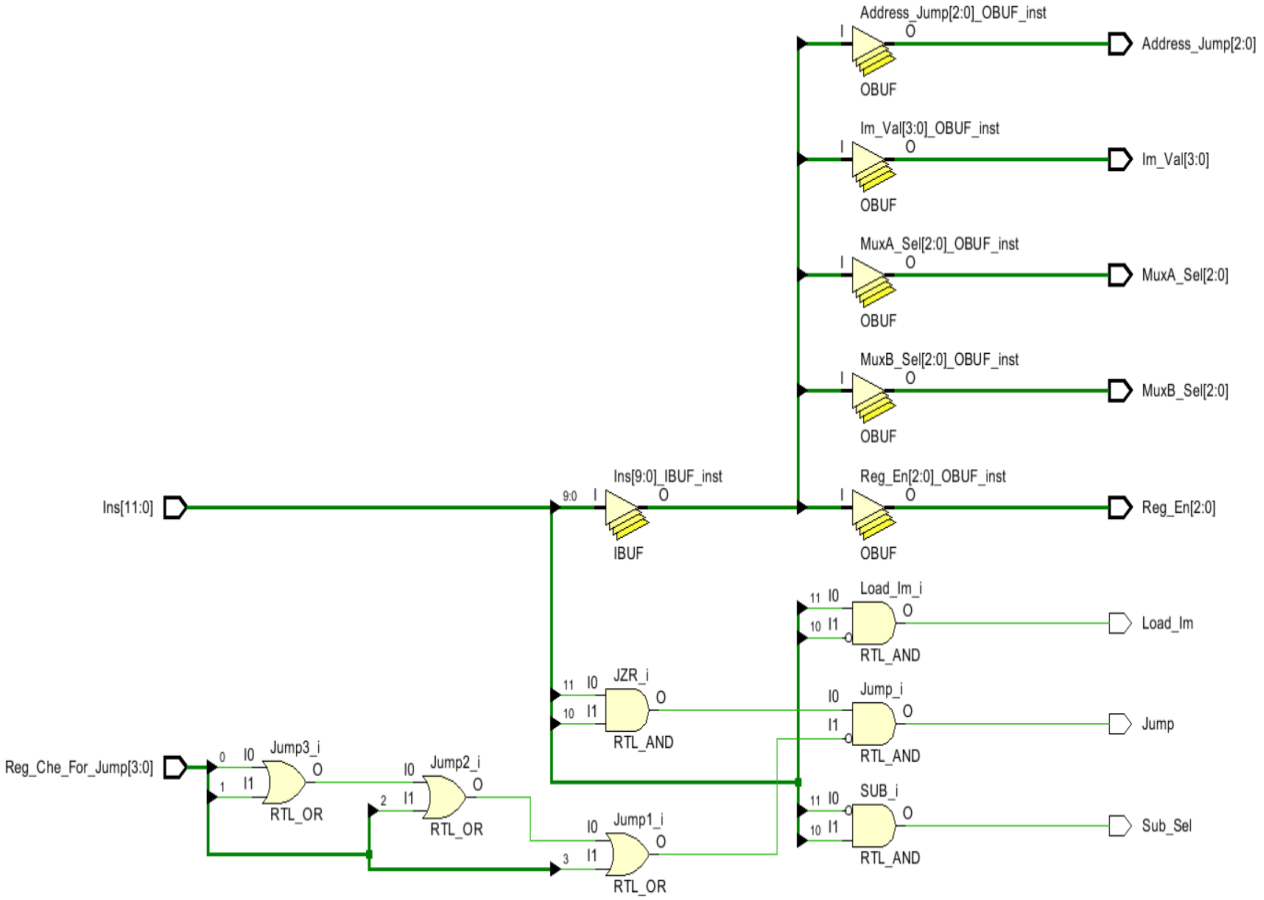
Instruction decoder

---

01. Design source VHDL code of instruction decoder.

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

entity Instruction_Decoder is
    Port (
            Ins : in STD_LOGIC_VECTOR (11 downto 0);
            Reg_Che_For_Jump : in STD_LOGIC_VECTOR (3 downto 0);
            Reg_En : out STD_LOGIC_VECTOR(2 downto 0);
            Load_Im : out STD_LOGIC;
            Im_Val : out STD_LOGIC_VECTOR(3 downto 0);
            MuxA_Sel : out STD_LOGIC_VECTOR (2 downto 0);
            MuxB_Sel : out STD_LOGIC_VECTOR (2 downto 0);
            Sub_Sel : out STD_LOGIC;
            Jump : out STD_LOGIC;
            Address_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is

signal SUB,JZR:std_logic;
signal Op:std_logic_vector(1 downto 0);
signal Reg_A,Reg_B:std_logic_vector(2 downto 0);
signal Im_Val_Temp,R:std_logic_vector(3 downto 0);

begin

Im_Val<=Ins(3 downto 0);
Reg_En<=Ins(9 downto 7);
Address_Jump<=Ins(2 downto 0);
R<=Reg_Che_For_Jump;

Load_Im<=Ins(11) and not(Ins(10));
SUB<=not(Ins(11)) and Ins(10);
JZR<=Ins(11) and Ins(10);

MuxA_Sel<=Ins(9 downto 7);
MuxB_Sel<=Ins(6 downto 4);


Sub_Sel<=SUB;
Jump<=JZR and (not(R(0) or R(1) or R(2) or R(3)));

end Behavioral;
```

02. RTL Schematic diagram of the instruction decoder.

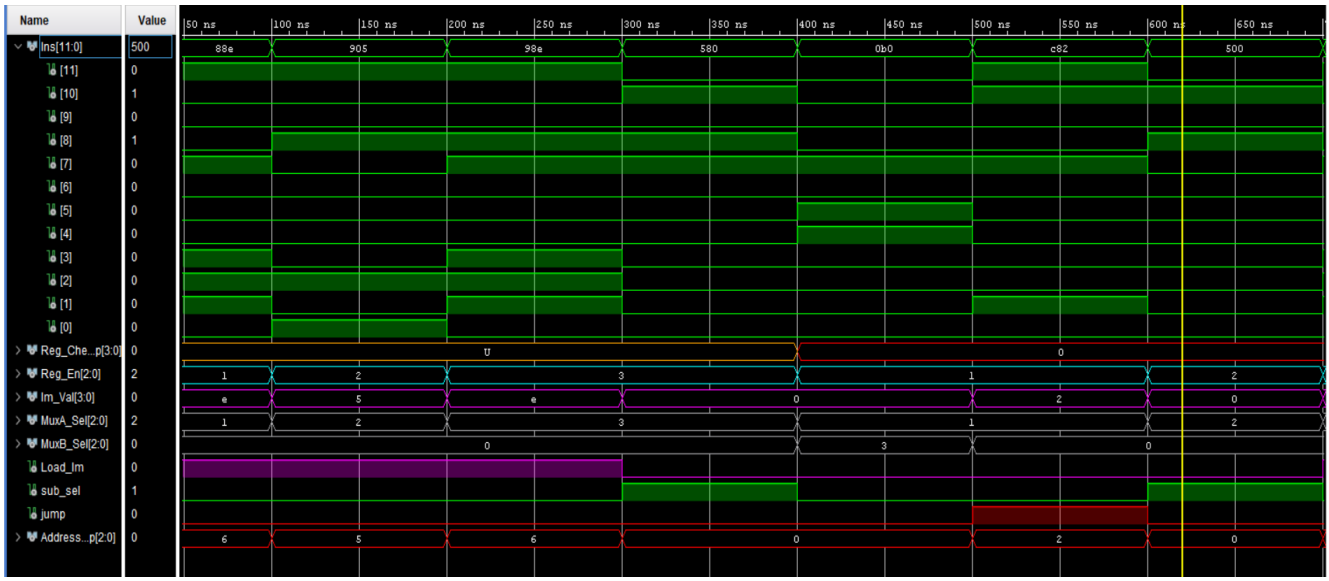


03. Behavioral simulation Code for instruction decoder.

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


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

architecture Behavioral of TB_Instruction_Decoder is
component Instruction_Decoder
    Port ( --Clk : in STD_LOGIC;
           Ins : in STD_LOGIC_VECTOR (11 downto 0);
           Reg_Che_For_Jump : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_En : out STD_LOGIC_VECTOR(2 downto 0);
           Load_Im : out STD_LOGIC;
           Im_Val : out STD_LOGIC_VECTOR(3 downto 0);
           MuxA_Sel : out STD_LOGIC_VECTOR (2 downto 0);
           MuxB_Sel : out STD_LOGIC_VECTOR (2 downto 0);
           Sub_Sel : out STD_LOGIC;
           Jump : out STD_LOGIC;
           Address_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end component;
```

```
signal Load_Im,sub_sel,jump : std_logic;
signal Reg_Che_For_Jump,Im_Val : STD_LOGIC_VECTOR(3 downto 0);
signal Reg_En,MuxA_Sel,MuxB_Sel,Address_Jump : STD_LOGIC_VECTOR (2 downto
0);
signal Ins : STD_LOGIC_VECTOR (11 downto 0);
begin

UUT : Instruction_Decoder
        port map (
                  Ins => Ins,
                  Reg_Che_For_Jump => Reg_Che_For_Jump,
                  Reg_En => Reg_En,
                  Load_Im => Load_Im,
                  Im_Val => Im_Val,
                  MuxA_Sel => MuxA_Sel,
                  MuxB_Sel => MuxB_Sel,
                  Sub_Sel => Sub_Sel,
                  Jump => Jump,
                  Address_Jump => Address_Jump
        );


process
begin
 Ins <= "100010001110";
 wait for 100ns;

 Ins <= "100100000101";
 wait for 100ns;

 Ins <= "100110001110";
 wait for 100ns;

 Ins <= "010110000000";
 wait for 100ns;

 Reg_Che_For_Jump<="0000";
 Ins <= "000010110000";
 wait for 100ns;

 Ins <= "110010000010";
 wait for 100ns;

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


end process;
end Behavioral;
```

04. Timing Diagram for instruction decoder.



Additional features added design improved components

Instruction decoder

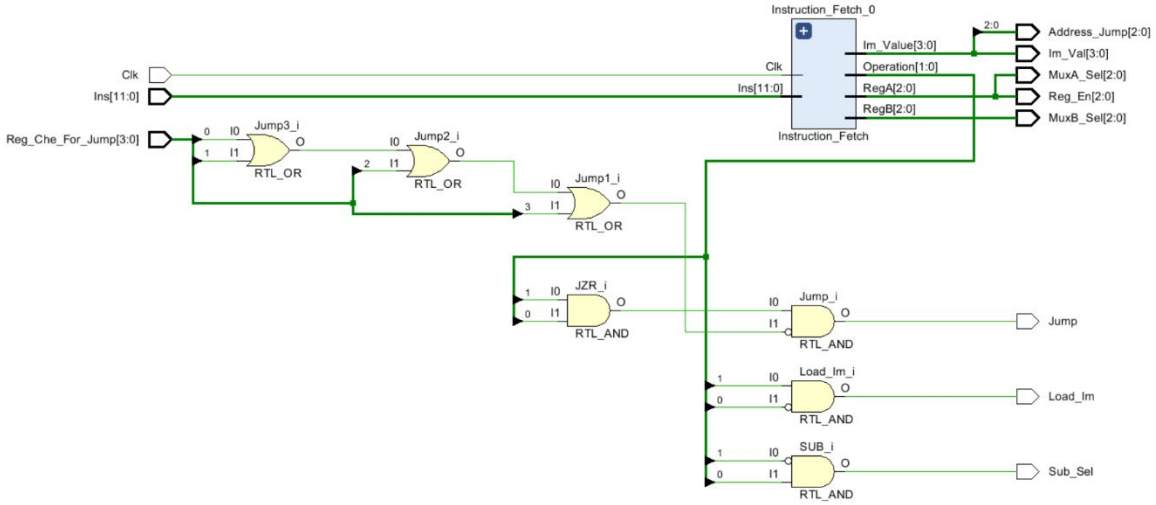
---

01. Design source VHDL code of instruction decoder.

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

entity Instruction_Decoder is
    Port ( Clk : in STD_LOGIC;
           Ins : in STD_LOGIC_VECTOR (11 downto 0);
           Reg_Che_For_Jump : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_En : out STD_LOGIC_VECTOR(2 downto 0);
           Load_Im : out STD_LOGIC;
           Im_Val : out STD_LOGIC_VECTOR(3 downto 0);
           MuxA_Sel : out STD_LOGIC_VECTOR (2 downto 0);
           MuxB_Sel : out STD_LOGIC_VECTOR (2 downto 0);
           Sub_Sel : out STD_LOGIC;
           Jump : out STD_LOGIC;
           Address_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is
component Instruction_Fetch
    Port ( Clk : in STD_LOGIC;
           Ins : in STD_LOGIC_VECTOR (11 downto 0);
           Operation : out STD_LOGIC_VECTOR (1 downto 0);
           RegA : out STD_LOGIC_VECTOR (2 downto 0);
           RegB : out STD_LOGIC_VECTOR (2 downto 0);
           Im_Value : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal SUB,JZR:std_logic;
signal Op:std_logic_vector(1 downto 0);
signal Reg_A,Reg_B:std_logic_vector(2 downto 0);
signal Im_Val_Temp,R:std_logic_vector(3 downto 0);

begin
Instruction_Fetch_0:Instruction_Fetch
    port map(
        Clk=>Clk,
        Ins=>Ins,
        Operation=>Op,
        RegA=>Reg_A,
        RegB=>Reg_B,
        Im_Value=>Im_Val_Temp);

Im_Val<=Im_Val_Temp;
Reg_En<=Reg_A;
Address_Jump<=Im_Val_Temp(2 downto 0);
R<=Reg_Che_For_Jump;
```

```
Load_Im<=Op(1) and not(Op(0));
SUB<=not(Op(1)) and Op(0);
JZR<=Op(1) and Op(0);
MuxA_Sel<=Reg_A;
MuxB_Sel<=Reg_B;

Sub_Sel<=SUB;
Jump<=JZR and (not(R(0) or R(1) or R(2) or R(3)));

end Behavioral;
```

02. RTL Schematic diagram of instruction decoder.



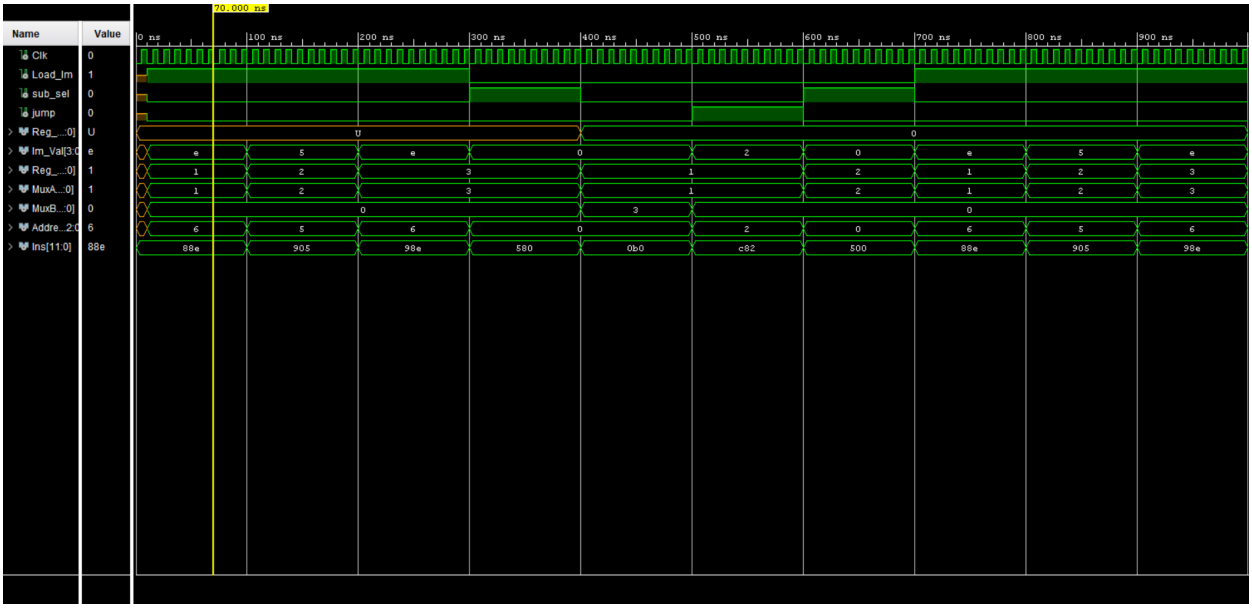
03. Behavioral simulation source Code for the instruction decoder.

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

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

architecture Behavioral of TB_Instruction_Decoder is
component Instruction_Decoder
    Port ( Clk : in STD_LOGIC;
           Ins : in STD_LOGIC_VECTOR (11 downto 0);
           Reg_Che_For_Jump : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_En : out STD_LOGIC_VECTOR(2 downto 0);
           Load_Im : out STD_LOGIC;
           Im_Val : out STD_LOGIC_VECTOR(3 downto 0);
           MuxA_Sel : out STD_LOGIC_VECTOR (2 downto 0);
```

```
            MuxB_Sel : out STD_LOGIC_VECTOR (2 downto 0);
            Sub_Sel : out STD_LOGIC;
            Jump : out STD_LOGIC;
            Address_Jump : out STD_LOGIC_VECTOR (2 downto 0));
end component;
signal Clk : std_logic:= '0';
signal Load_Im,sub_sel,jump : std_logic;
signal Reg_Che_For_Jump,Im_Val : STD_LOGIC_VECTOR(3 downto 0);
signal Reg_En,MuxA_Sel,MuxB_Sel,Address_Jump : STD_LOGIC_VECTOR (2 downto 0);
signal Ins : STD_LOGIC_VECTOR (11 downto 0);
begin

UUT : Instruction_Decoder
        port map (
                  Clk => Clk,
                  Ins => Ins,
                  Reg_Che_For_Jump => Reg_Che_For_Jump,
                  Reg_En => Reg_En,
                  Load_Im => Load_Im,
                  Im_Val => Im_Val,
                  MuxA_Sel => MuxA_Sel,
                  MuxB_Sel => MuxB_Sel,
                  Sub_Sel => Sub_Sel,
                  Jump => Jump,
                  Address_Jump => Address_Jump
        );

process
 begin
    wait for 5ns;
    Clk <= not Clk;
end process;

process
begin
 Ins <= "100010001110";
 wait for 100ns;

 Ins <= "100100000101";
 wait for 100ns;

 Ins <= "100110001110";
 wait for 100ns;

 Ins <= "010110000000";
 wait for 100ns;

 Reg_Che_For_Jump<="0000";
 Ins <= "000010110000";
 wait for 100ns;

 Ins <= "110010000010";
 wait for 100ns;
```

```
 Ins <= "010100000000";
 wait for 100ns;


end process;
end Behavioral;
```

04. Timing Diagram for the instruction decoder.



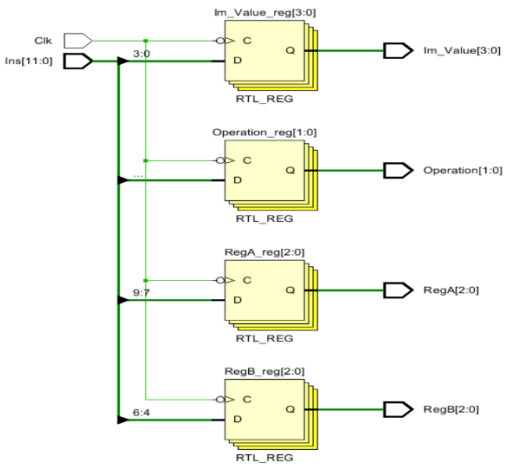
Instruction fetch

---

01. Design source VHDL code of instruction fetch.

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

entity Instruction_Fetch is
    Port ( Clk : in STD_LOGIC;
           Ins : in STD_LOGIC_VECTOR (11 downto 0);
           Operation : out STD_LOGIC_VECTOR (1 downto 0);
           RegA : out STD_LOGIC_VECTOR (2 downto 0);
           RegB : out STD_LOGIC_VECTOR (2 downto 0);
           Im_Value : out STD_LOGIC_VECTOR (3 downto 0));
end Instruction_Fetch;

architecture Behavioral of Instruction_Fetch is

begin
process (Clk)
    begin
        if (falling_edge(Clk)) then
           Operation<=Ins(11 downto 10);
           RegA<=Ins(9 downto 7);
           RegB<=Ins(6 downto 4);
           Im_Value<=Ins(3 downto 0);

        end if;
end process;

end Behavioral;
```

02. RTL Schematic diagram of instruction fetch.

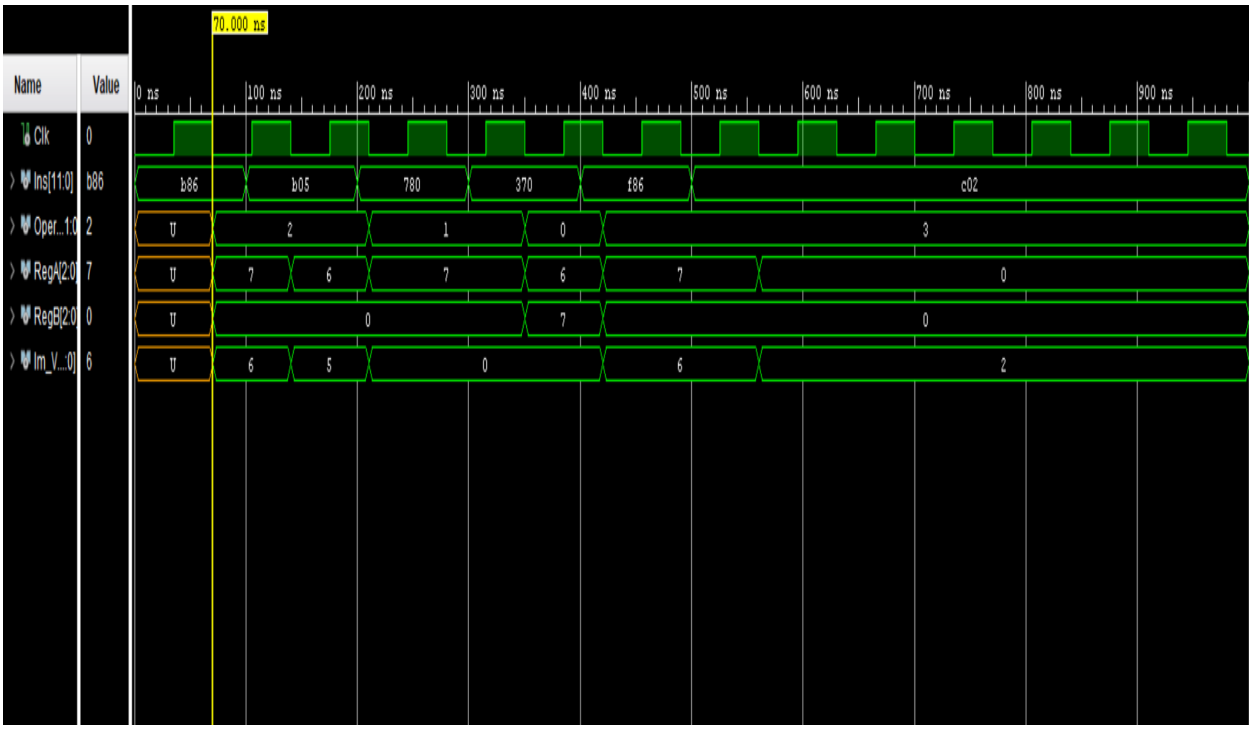


03. Behavioral simulation source Code for the instruction fetch.

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

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

architecture Behavioral of TB_Ins_Fetch is
component Instruction_Fetch
    Port ( Clk : in STD_LOGIC;
           Ins : in STD_LOGIC_VECTOR (11 downto 0);
           Operation : out STD_LOGIC_VECTOR (1 downto 0);
           RegA : out STD_LOGIC_VECTOR (2 downto 0);
           RegB : out STD_LOGIC_VECTOR (2 downto 0);
           Im_Value : out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal Clk:std_logic:='0';
signal Ins:std_logic_vector(11 downto 0);
signal Operation:std_logic_vector(1 downto 0);
signal RegA,RegB:std_logic_vector(2 downto 0);
signal Im_Value:std_logic_vector(3 downto 0);

begin
UUT:Instruction_Fetch
    port map(
    Clk=>Clk,
    Ins=>Ins,
    Operation=>Operation,
    RegA=>RegA,
    RegB=>RegB,
    Im_Value=>Im_Value  );

process
begin
    wait for 35ns;
    Clk <= not Clk;
end process;

process
begin
    --200635->11 0000 1111 1011 1011
    --200240->11 0000 1110 0011 0000
    --200575->11 0000 1111 0111 1111
    --200647->11 0000 1111 1100 0111
    --200655->11 0000 1111 1100 1111

    --unique 4bit numbers from index
numbers=>0011,0000,1111,1110,1011,0011,0111,1100

    Ins <= "101110000110"; --MOVI R7,6;
```

```
    wait for 100ns;          --R7=6

    Ins <= "101100000101"; --MOVI R6,5;
    wait for 100ns;          --R6=5

    Ins <= "011110000000"; --NEG R7;
    wait for 100ns;          --R7=-6

    Ins <= "001101110000"; --ADD R6,R7;
    wait for 100ns;

    Ins <= "111110000110";     --JZR R7,6;
    wait for 100ns;

    Ins <= "110000000010";     --JZR R0,2;
    wait;

end process;

end Behavioral;
```

04. Timing Diagram for the instruction fetch.



Nano processor with 7 seg output

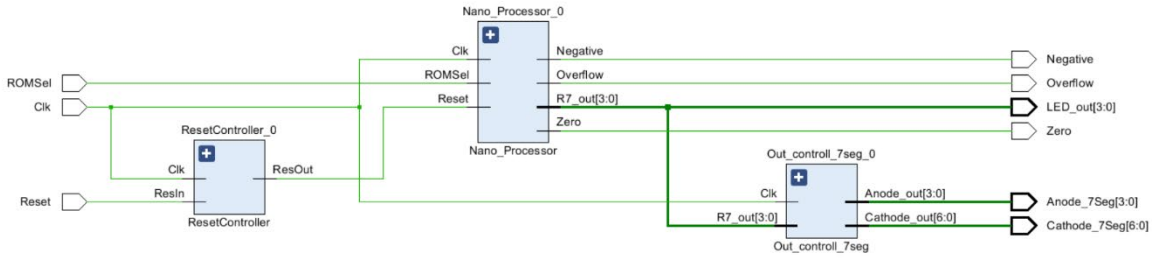
---

01. Design source VHDL code of nano processor with 7 seg out.

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

entity Nano_Processor_with_7SegOut is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           ROMSel: in STD_LOGIC;
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC;
           Negative : out STD_LOGIC;
           LED_out : out STD_LOGIC_VECTOR (3 downto 0);
           Anode_7Seg : out STD_LOGIC_VECTOR (3 downto 0);
           Cathode_7Seg : out STD_LOGIC_VECTOR (6 downto 0));
end Nano_Processor_with_7SegOut;

architecture Behavioral of Nano_Processor_with_7SegOut is
    component Nano_Processor
        Port ( Clk : in STD_LOGIC;
               Reset : in STD_LOGIC;
               ROMSel: in STD_LOGIC;
               Overflow : out STD_LOGIC;
               Zero : out STD_LOGIC;
               Negative:out STD_Logic;
               R7_out : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    component Out_controll_7seg
        Port ( Clk : in STD_LOGIC;
               R7_out : in STD_LOGIC_VECTOR (3 downto 0);
               Anode_out : out STD_LOGIC_VECTOR (3 downto 0);
               Cathode_out : out STD_LOGIC_VECTOR (6 downto 0));
    end component;

    component ResetController
        Port ( ResIn : in STD_LOGIC;
               Clk : in STD_Logic;
               ResOut : out STD_LOGIC);
    end component;

    signal R7_output:std_logic_vector(3 downto 0);
    signal ResINPUT:std_logic;

begin
ResetController_0:ResetController
    port map(
        ResIn=>Reset,
```

```
        Clk=>Clk,
        ResOut=>ResINPUT    );

Nano_Processor_0:Nano_processor
    port map(
            Clk=>Clk,
            Reset=>ResINPUT,
            ROMSel=>ROMSel,
            Overflow=>Overflow,
            Zero=>Zero,
            Negative=>Negative,
            R7_out=>R7_output    );

LED_out<=R7_output;

Out_controll_7seg_0:Out_controll_7seg
    Port map (
            Clk =>Clk,
            R7_out=>R7_output,
            Anode_out=>Anode_7Seg,
            Cathode_out=>Cathode_7Seg);

end Behavioral;
```

02. RTL Schematic diagram of nano processor with 7 seg out.



03. Behavioral simulation source Code for the nano processor with 7 seg out.

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

entity TB_Nano_Processor_with_7SegOut_1 is
--  Port ( );
end TB_Nano_Processor_with_7SegOut_1;

architecture Behavioral of TB_Nano_Processor_with_7SegOut_1 is
component
    Nano_Processor_with_7SegOut
    Port ( Clk : in STD_LOGIC;
```

```
            Reset : in STD_LOGIC;
            ROMSel: in STD_LOGIC;
            Overflow : out STD_LOGIC;
            Zero : out STD_LOGIC;
            Negative : out STD_LOGIC;
            LED_out : out STD_LOGIC_VECTOR (3 downto 0);
            Anode_7Seg : out STD_LOGIC_VECTOR (3 downto 0);
            Cathode_7Seg : out STD_LOGIC_VECTOR (6 downto 0));
end component;
signal Clk : std_logic := '0';
signal Reset : std_logic;
signal ROMSel : std_logic:='0';
signal overflow,Zero,Negative : std_logic;
signal LED_out,Anode_7Seg : STD_LOGIC_VECTOR (3 downto 0);
signal Cathode_7Seg :STD_LOGIC_VECTOR (6 downto 0);
begin
UUT:Nano_Processor_with_7SegOut
    port map(
        Clk=>Clk,
        Reset=>Reset,
        ROMSel=>ROMSel,
        OverFlow=>Overflow,
        Zero=>Zero,
        Negative=>Negative,
        LED_out => LED_out,
        Anode_7Seg => Anode_7Seg,
        Cathode_7Seg => Cathode_7Seg);
process
begin
    wait for 1ns;
    Clk<=not(Clk);
end process;

process
begin
wait for 200ns;
Reset<='1';
wait for 20ns;
Reset<='0';
wait for 1500ns;
ROMSel<='1';
wait for 200ns;
Reset<='1';
wait for 10ns;
Reset<='0';
wait;
end process;
end Behavioral;
```

04. Timing Diagram for the nano processor with 7 seg out.

Out control 7 seg

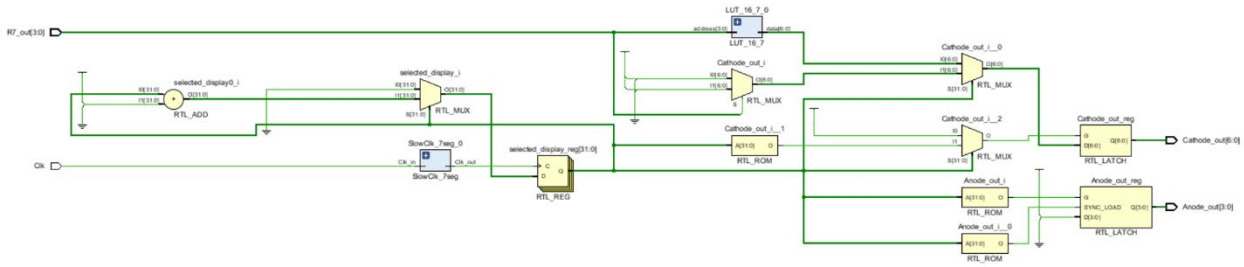
---

01. Design source VHDL code of out control 7 seg.

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


entity Out_controll_7seg is
    Port ( Clk : in STD_LOGIC;
           R7_out : in STD_LOGIC_VECTOR (3 downto 0);
           Anode_out : out STD_LOGIC_VECTOR (3 downto 0);
           Cathode_out : out STD_LOGIC_VECTOR (6 downto 0));
end Out_controll_7seg;

architecture Behavioral of Out_controll_7seg is
    component SlowClk_7seg
        Generic (count_max:integer:= 100000000); --100 million
        Port ( Clk_in : in STD_LOGIC;
               Clk_out : out STD_LOGIC);
    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;

    signal Clk_out_7seg,MSB_R7out:std_logic;
    signal selected_display:integer:=0;
    signal S_7seg:std_logic_vector(6 downto 0);

begin
    SlowClk_7seg_0:SlowClk_7seg
        Generic map(count_max => 2) --for simulation
        port map(
            Clk_in=>Clk,
            Clk_out=>Clk_out_7seg   );
    LUT_16_7_0:LUT_16_7
            port map(
                    address=>R7_out,
                    data=>S_7Seg    );

    process(Clk_out_7seg)
    begin
        if (rising_edge(Clk_out_7seg)) then
            selected_display<=selected_display + 1;
            if (selected_display=2)then
                selected_display<=0;
            end if;
        end if;
```

```
    end process;

    MSB_R7out<=R7_out(3);
    process(selected_display,MSB_R7out,S_7Seg)
        begin
        if (selected_display = 0) then
            Anode_out<="1110";
            Cathode_out<=S_7Seg;
        elsif(selected_display = 1)then
            Anode_out<="1101";
            if (MSB_R7out='1') then
                Cathode_out<="0111111"; -- to show minus(-)
            elsif(MSB_R7out='0')then
                Cathode_out<="1000000"; -- show zero
            end if;

        end if;
    end process;

end Behavioral;
```

02. RTL Schematic diagram of out control 7 seg.

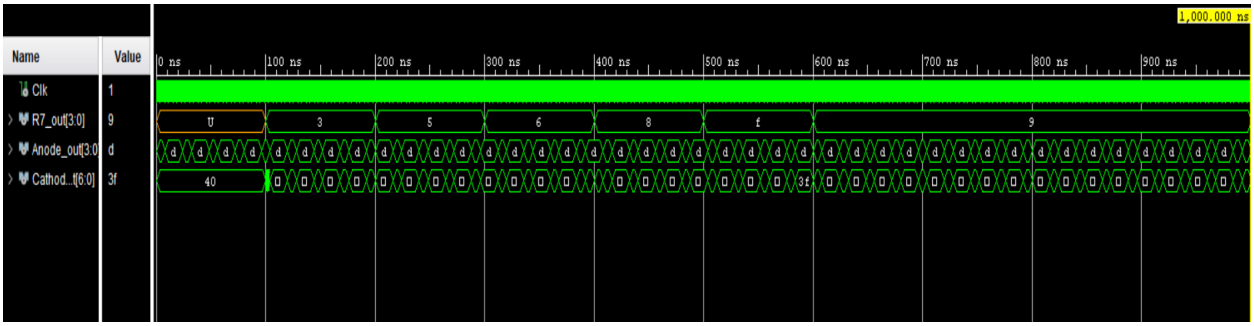


03. Behavioral simulation source Code for the out control 7 seg.

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

entity Sim_Out_Control_7Seg is
--  Port ( );
end Sim_Out_Control_7Seg;

architecture Behavioral of Sim_Out_Control_7Seg is
component Out_controll_7seg
    Port ( Clk : in STD_LOGIC;
           R7_out : in STD_LOGIC_VECTOR (3 downto 0);
           Anode_out : out STD_LOGIC_VECTOR (3 downto 0);
           Cathode_out : out STD_LOGIC_VECTOR (6 downto 0));
end component ;

signal Clk: std_logic := '1';
```

```
signal R7_out,Anode_out :STD_LOGIC_VECTOR (3 downto 0);
signal Cathode_out :STD_LOGIC_VECTOR (6 downto 0);

begin
UUT:Out_controll_7seg
    port map(
        Clk=>Clk,
        R7_out=>R7_out,
        Anode_out => Anode_out,
        Cathode_out=>Cathode_out);

process
begin
    wait for 1ns;
    Clk<=not(Clk);
end process;

process
begin
    wait for 100ns;
    R7_out <= "0011";
    wait for 100ns;
    R7_out <= "0101";
    wait for 100ns;
    R7_out <= "0110";
    wait for 100ns;
    R7_out <= "1000";
    wait for 100ns;
    R7_out <= "1111";
    wait for 100ns;
    R7_out <= "1001";
    wait;
end process;

end Behavioral;
```

04. Timing Diagram for the out control 7 seg.



Slow clock for 7 segment displays

---

01. Design source VHDL code of slow clock 7 seg.

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

entity SlowClk_7seg is
    Generic (count_max:integer:= 100000000);
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end SlowClk_7seg;

architecture Behavioral of SlowClk_7seg 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=count_max) then
            clk_status<= not(clk_status);
            clk_out<= clk_status;
            count<=1;
        end if;
     end if;
end process;

end Behavioral;
```

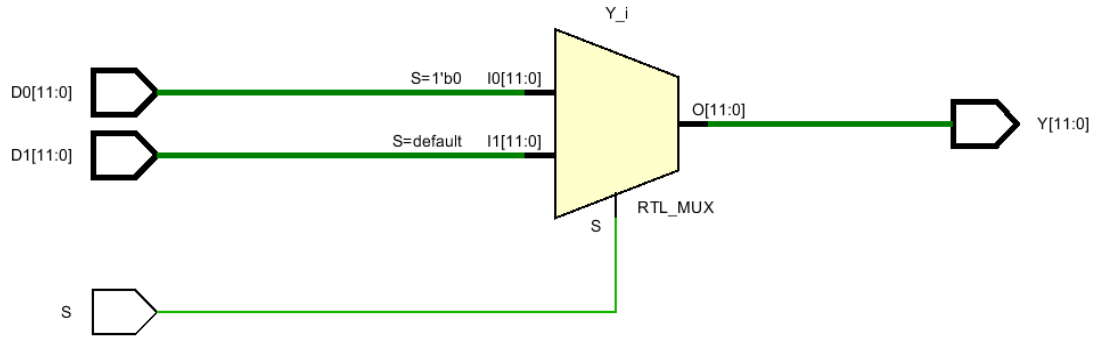
2-way 12-bit multiplexer

---

01. Design source VHDL code of 2-way 12-bit multiplexer.

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

entity MUX_2_to_1_12 is
    Port ( D0 : in STD_LOGIC_VECTOR(11 downto 0);
           D1 : in STD_LOGIC_VECTOR(11 downto 0);
           S : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR(11 downto 0));
end MUX_2_to_1_12;

architecture Behavioral of MUX_2_to_1_12 is

begin
    Y <= D0 when (S = '0') else D1;
end Behavioral;
```

02. RTL Schematic diagram of 2 way 12 bit multiplexer..



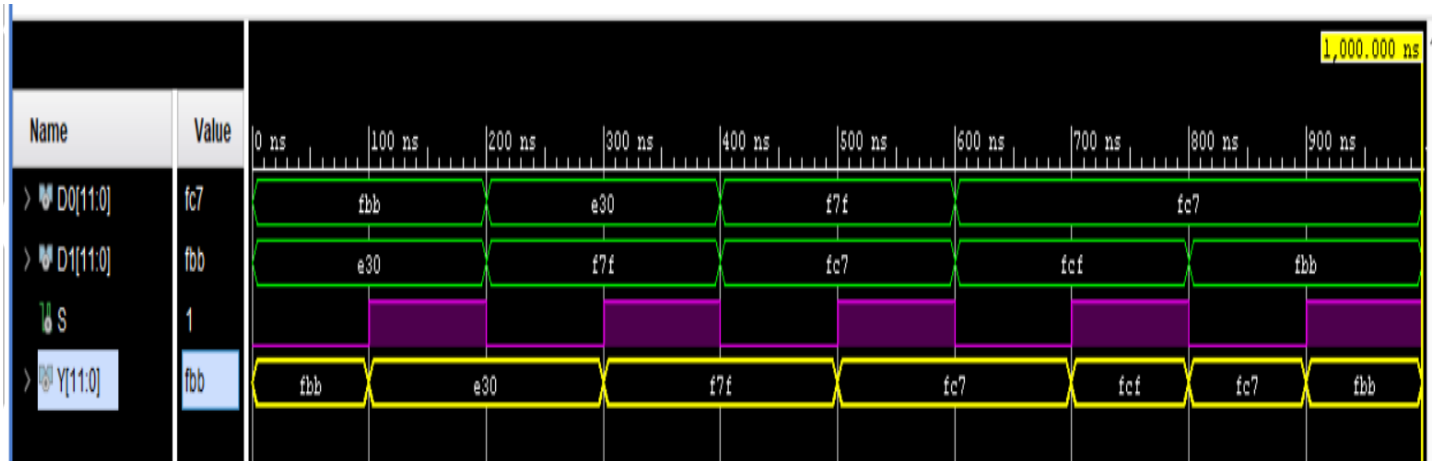
03. Behavioral simulation source Code for the 2-way 12-bit multiplexer.

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

entity TB_MUX_2_to_1_12 is
--  Port ( );
end TB_MUX_2_to_1_12;

architecture Behavioral of TB_MUX_2_to_1_12 is

component MUX_2_to_1_12
    Port ( D0 : in STD_LOGIC_VECTOR(11 downto 0);
```

```
            D1 : in STD_LOGIC_VECTOR(11 downto 0);
            S : in STD_LOGIC;
            Y : out STD_LOGIC_VECTOR(11 downto 0));
end component;

signal D0,D1,Y : std_logic_vector(11 downto 0);
signal S : std_logic;

begin

UUT: MUX_2_to_1_12
    port map(
        D0 => D0,
        D1 => D1,
        S => S,
        Y => Y
    );


process
begin
    --200635->11 0000 1111 1011 1011
    --200240->11 0000 1110 0011 0000
    --200575->11 0000 1111 0111 1111
    --200647->11 0000 1111 1100 0111
    --200655->11 0000 1111 1100 1111

    D0 <= "111110111011";
    D1 <= "111000110000";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "111000110000";
    D1 <= "111101111111";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "111101111111";
    D1 <= "111111000111";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "111111000111";
    D1 <= "111111001111";
    S <= '0';
    wait for 100 ns;
    S <= '1';
    wait for 100 ns;

    D0 <= "111111000111";
    D1 <= "111110111011";
```

```
        S <= '0';
        wait for 100 ns;
        S <= '1';
        wait;

    end process;

    end Behavioral;
```

04. Timing Diagram for the 2-way 12-bit multiplexer.



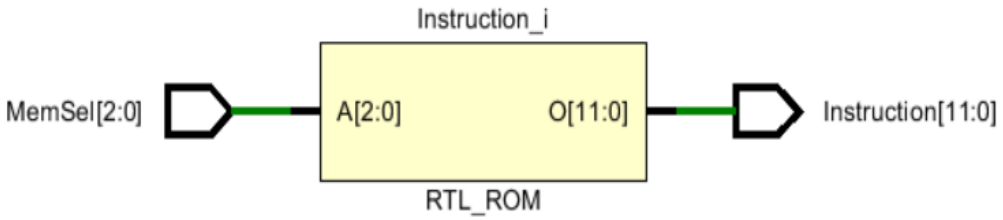
Program ROM

---

01. Design source VHDL code of program ROM.

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

entity ProgramROM is
Port ( MemSel : in STD_LOGIC_VECTOR (2 downto 0);
Instruction : out STD_LOGIC_VECTOR (11 downto 0));
end ProgramROM;

architecture Behavioral of ProgramROM is
type rom_type is array (0 to 7) of std_logic_vector(11 downto 0);
signal ProgramROM_0 : rom_type :=(
   "100010000011",  -- MOVI R1,3    --line->0
   "100100000001",  -- MOVI R2,1
   "010100000000",  -- NEG R2
   "001110010000",  -- ADD R7,R1    --line->3
   "000010100000",  -- ADD R1,R2
   "110010000111",  -- JZR R1,7
   "110000000011",  -- JZR R0,3
   "110000000111"   -- JZR R0,7     --line->7

);

begin
Instruction <= ProgramROM_0(to_integer(unsigned(MemSel)));
end Behavioral;
```

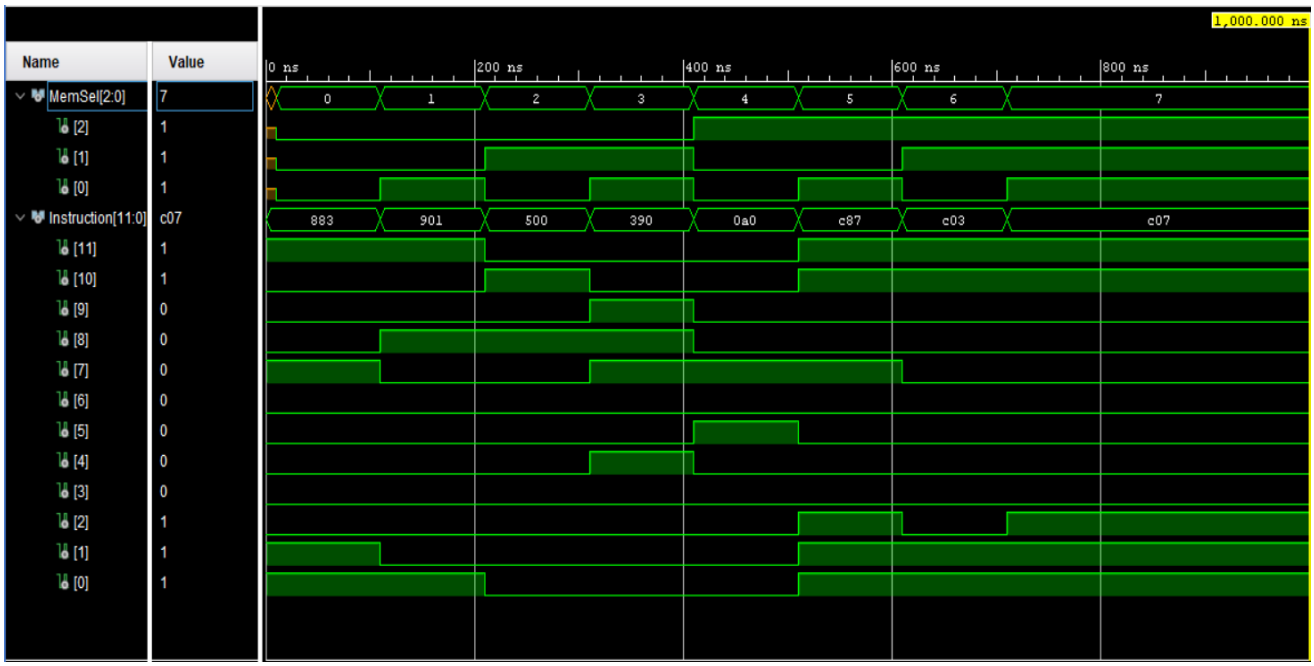
02. RTL Schematic diagram of program ROM.



03. Behavioral simulation source code for program ROM.

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

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

architecture Behavioral of RomSim is
component ProgramROM

  port(
MemSel : in STD_LOGIC_VECTOR (2 downto 0);
Instruction : out STD_LOGIC_VECTOR (11 downto 0)
);
end component;

Signal MemSel : STD_LOGIC_VECTOR(2 downto 0);
Signal Instruction : STD_LOGIC_VECTOR(11 downto 0);


begin
UUT : ProgramROM port map(
MemSel => MemSel,
Instruction => Instruction
);
process
begin

wait for 10ns;
MemSel <= "000";
wait for 100ns;
MemSel <= "001";
wait for 100ns;
MemSel <= "010";
wait for 100ns;
MemSel <= "011";
wait for 100ns;
MemSel <= "100";
wait for 100ns;
MemSel<="101";
Wait for 100ns;
MemSel<="110";
Wait for 100ns;
MemSel<="111";
wait;
end process;
end Behavioral;
```

04. Timing Diagram for program ROM.



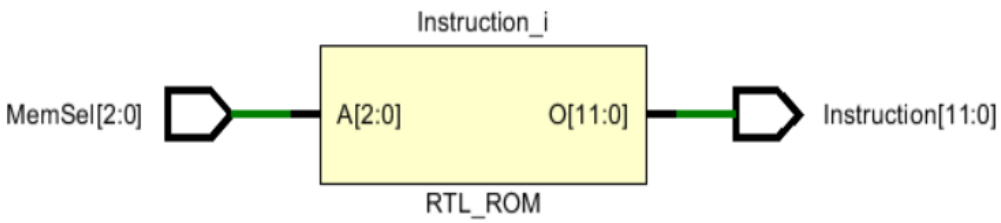
program ROM 2

---

01. Design source VHDL code of program ROM 2.

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

entity ProgramROM_2 is
Port ( MemSel : in STD_LOGIC_VECTOR (2 downto 0);
Instruction : out STD_LOGIC_VECTOR (11 downto 0));
end ProgramROM_2;

architecture Behavioral of ProgramROM_2 is
type rom_type is array (0 to 7) of std_logic_vector(11 downto 0);
signal ProgramROM_0 : rom_type :=(

   --Another program to count from (-8) to 6 with increment 2
     "101110001000",  -- MOVI R7,(-8)  --line->0
     "100100000010",  -- MOVI R2,2
     "001110100000",  -- ADD R7,R2
     "110000000010",  -- JZR R0,2    --line->3 , after this nothing will
execute and always jump to line 2
     "000010100000",  -- ADD R1,R2
     "110010000111",  -- JZR R1,7
     "110000000011",  -- JZR R0,3
     "110000000111"   -- JZR R0,7     --line->7
);

begin
Instruction <= ProgramROM_0(to_integer(unsigned(MemSel)));
end Behavioral;
```

02. RTL Schematic diagram of program ROM 2.



03. Behavioral simulation source Code for the program ROM 2.

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

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

architecture Behavioral of RomSim is
component ProgramROM_2

  port(
MemSel : in STD_LOGIC_VECTOR (2 downto 0);
Instruction : out STD_LOGIC_VECTOR (11 downto 0)
);
end component;

Signal MemSel : STD_LOGIC_VECTOR(2 downto 0);
Signal Instruction : STD_LOGIC_VECTOR(11 downto 0);


begin
UUT : ProgramROM_2 port map(
MemSel => MemSel,
Instruction => Instruction
);
process begin
wait for 10ns;
MemSel <= "000";
wait for 100ns;
MemSel <= "001";
wait for 100ns;
MemSel <= "010";
wait for 100ns;
MemSel <= "011";
wait for 100ns;
MemSel <= "100";
wait;
end process;
end Behavioral;
```

04. Timing Diagram for the program ROM 2.